

УДК 004.2(075.32)
ББК 32.973-02я723
Н50

Рецензенты:

доктор технических наук, профессор кафедры «Информатика и программное обеспечение вычислительных систем» *О.И. Лисов* (Московский государственный институт электронной техники (технический университет));

кандидат технических наук, доцент кафедры «Информационные технологии» *А.А. Петров* (Институт искусств и информационных технологий)

Немцова Т.И.

Н50 Программирование на языке высокого уровня. Программирование на языке C++ : учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — Москва : ИД «ФОРУМ» : ИНФРА-М, 2023. — 512 с. + Доп. материалы [Электронный ресурс]. — (Среднее профессиональное образование).

ISBN 978-5-8199-0699-6 (ИД «ФОРУМ»)


ISBN 978-5-16-013214-3 (ИНФРА-М, print)

ISBN 978-5-16-102802-5 (ИНФРА-М, online)

В пособии рассматриваются работа в среде программирования Microsoft Visual Studio 2010, основы программирования и объектно-ориентированное программирование на языке C++. Представленный теоретический материал сопровождается подробно разобранными примерами программ со схемами алгоритмов. Для закрепления материала предлагаются контрольные вопросы, тесты и задания для самостоятельного решения.

Предназначено для школьников, студентов средних специальных заведений и вузов (технических, экономических и других специальностей), изучающих дисциплину «Программирование», может быть рекомендовано преподавателям, слушателям курсов повышения квалификации, а также может быть использовано как самоучитель.

УДК 004.2(075.32)
ББК 32.973-02я723

Материалы, отмеченные знаком ,
доступны в электронно-библиотечной системе Znanium

ISBN 978-5-8199-0699-6 (ИД «ФОРУМ»)
ISBN 978-5-16-013214-3 (ИНФРА-М, print)
ISBN 978-5-16-102802-5 (ИНФРА-М, online)

© Немцова Т.И., Голова С.Ю.,
Терентьев А.И., 2016
© ИД «ФОРУМ», 2016

Предисловие

Учебное пособие представляет собой курс по изучению языка C++. В настоящее время язык C++ является одним из самых распространенных языков программирования, поскольку идеально подходит для разработки прикладного программного обеспечения.

Учебное пособие предназначено для широкого круга читателей: как для начинающих программистов, так и для тех, кто уже знаком с основами программирования и в будущем собирается стать профессиональным программистом. Материал учебного пособия служит базисом для перехода к разработке приложений под Windows в среде Microsoft Visual Studio с применением стандартных библиотек MFC, STL и др.

Учебное пособие включает как информацию по основам программирования на языке C++, так и информацию по объектно-ориентированному программированию.

В главах 1—14 рассматриваются основы программирования на языке C++: работа в среде программирования Microsoft Visual Studio 2010, простые программы с линейной, разветвленной, циклической структурами, программирование задач с использованием одномерных массивов, работа с функциями. Также рассматривается материал по использованию препроцессорных средств и указателей. Представленный материал иллюстрируется примерами.

Главы 15—20 посвящены изучению объектно-ориентированного программирования. В них рассматривается реализация классов, использование статических полей, функций класса, констант. Рассматриваются вопросы применения дружественных функций и классов, наследования, потокового ввода/вывода, работы с файлами и обработки исключений.

Учебное пособие разбито на главы. Каждая глава содержит теоретический материал с подробно разобранными примерами программ. Примеры программ сопровождаются схемами алгоритмов, что особенно важно для тех, кто только начинает изучать программирование.

Для закрепления материала в конце каждой главы предлагаются контрольные вопросы, в конце большей части глав даны тесты и задачи для самостоятельного решения.

В конце учебного пособия в приложениях 1—3 приведены, соответственно, таблица символов ASCII, математические функции библиотеки `math.h`, классы и функции потокового ввода/вывода и ответы на тесты. Приложения 1—3 удобно использовать в качестве справочного материала при программировании, а ответы на тесты помогут проверить степень усвоения материала.

Исходные тексты разобранных примеров программ доступны в электронно-библиотечной системе Znanium.com.

Учебное пособие предназначено для школьников, студентов средних специальных заведений и вузов (технических, экономических и других специальностей), изучающих дисциплину «Программирование», может быть рекомендовано преподавателям, слушателям курсов повышения квалификации. Учебное пособие также может быть использовано как самоучитель.

Данное учебное пособие является результатом многолетней методической и преподавательской работы в Центре Компьютерного Обучения (www.sko-miet.ru) и на кафедре «Информатика и программное обеспечение вычислительных систем» Московского института электронной техники (технического университета).

Глава 1

ОСНОВНЫЕ СВЕДЕНИЯ О ПРОГРАММИРОВАНИИ

Понятие «задача» применительно к программированию имеет более широкое значение, чем аналогичное понятие в математике, физике или химии. В программировании под *задачей* понимается получение требуемого результата с использованием средств программирования. *Результатом решения* задачи в программировании может быть окно на экране, графическая картинка (с движением и без), вывод результатов вычислений на экран или в отдельный файл, построение на экране таблиц и графиков и т. д. Решение задач на ПК — это совместная деятельность программиста и компьютера, при этом человек выполняет творческую часть работы (постановка задачи, составление последовательности шагов решения, создание программы), а компьютер обрабатывает информацию в соответствии с разработанной программой.

1.1. Основные этапы решения задач

Для решения любой задачи с помощью компьютера необходимо выполнить семь этапов:

- постановка задачи;
- математическое моделирование;
- алгоритмизация задачи;
- программирование;
- ввод программы и исходных данных в компьютер;
- тестирование и отладка программы;
- исполнение отлаженной задачи и анализ результатов.

Рассмотрим этапы решения на примере: пусть требуется вычислить сумму двух целых чисел.

Первый этап — постановка задачи, т. е. формулирование условий задачи на естественном (русском) языке.

Пример постановки задачи: Даны A , B . Найти их сумму.

Второй этап — математическое моделирование, т. е. определение математических формул, необходимых для решения задачи (в нашем случае $S = A + B$).

Третий этап — алгоритмизация задачи. В общем случае программа решения задачи предназначена для обработки входных данных и получения выходных данных. Отсюда — в любой программе три основных компонента: входные данные, выходные данные, алгоритм обработки данных. Схематично процесс решения задачи представлен на рис. 1.1.

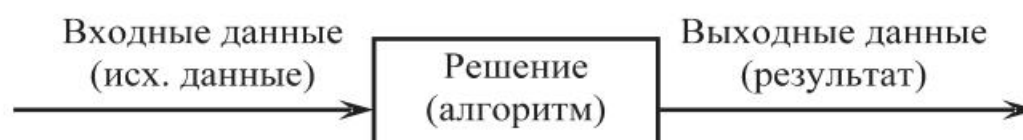


Рис. 1.1. Процесс решения задачи

В нашем примере входными данными являются целые числа A и B (их значения вводятся с клавиатуры), S — результат (выходные данные), который вычисляется в программе.

Этот этап является одним из важных этапов в решении задач, и он будет подробно рассмотрен в следующем параграфе.

Четвертый этап — программирование. Составление программы обеспечивает возможность выполнения алгоритма исполнителем-компьютером (соответственно решается поставленная задача). Для того чтобы компьютер выполнил решение какой-либо задачи, ему необходимо получить от человека инструкции, как ее решать. Набор таких инструкций для компьютера, направленный на решение конкретной задачи, называется *компьютерной программой*. Для написания программы и предназначены языки программирования (фиксированная система обозначений и правил для описания алгоритмов и структур данных). Например, языки программирования высокого уровня Бейсик, Pascal, C++ и т. д.

Пятый этап — ввод программы и исходных данных в компьютер.

Шестой этап — тестирование и отладка программы. На этом этапе исправляются ошибки и анализируется (тестируется) правильность работы программы (алгоритма).

Седьмой этап — исполнение отлаженной программы и анализ результатов.

1.2. Схемы алгоритмов

Алгоритм — это последовательность действий, которая определяет процесс получения выходных данных из входных, т. е. приводит к решению задачи.

Алгоритмы подразделяются на:

- линейные (действия выполняются последовательно одно за другим);
- ветвящиеся (есть условие и есть хотя бы два пути выполнения алгоритма);
- циклические (многократное повторение некоторой группы шагов).

Один из способов описания алгоритма — графический (схема алгоритма).

Обозначения в схемах алгоритмов.

1. Начало



2. Конец



3. Вычислительные действия

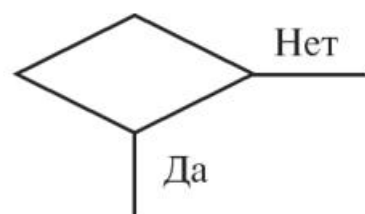


Для записи математических выражений используются только математические символы, а не операторы, без привязки к конкретному языку программирования (например, знак равенства, а не операция присваивания).

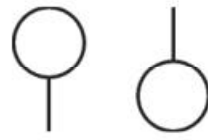
4. Ввод, вывод



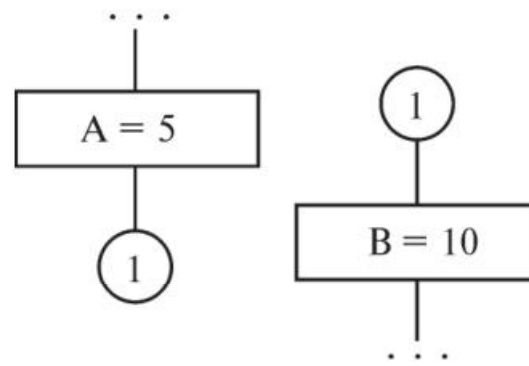
5. Проверка условия



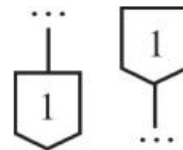
6. Соединитель (для внутрестраничного переноса)



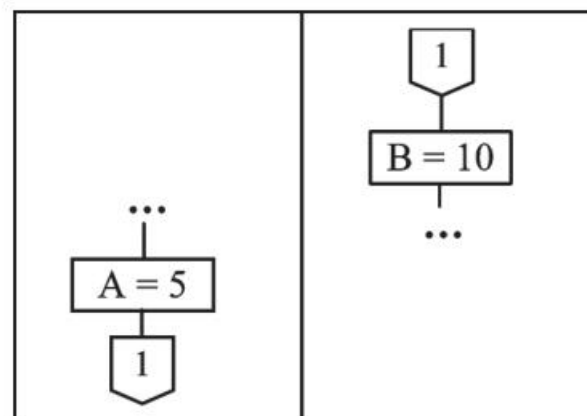
Например:



7. Межстраничный перенос



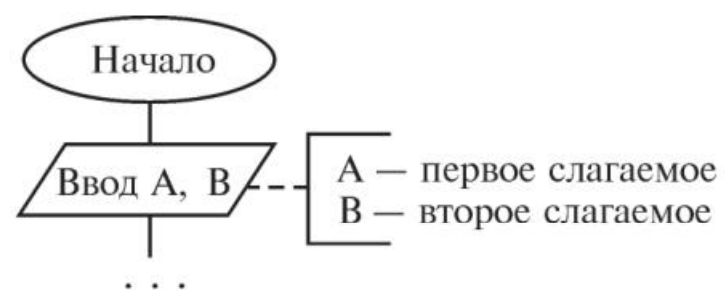
Например:



8. Комментарии




Например:



9. Линии и стрелки. Порядок выполнения программы в схемах показывается линиями. Линии подходят к блокам сверху или слева, а выходят из блоков вниз или вправо.

Направления линий сверху вниз и слева направо принимают за основные, и, если нет изломов, стрелками их можно не обозначать. В остальных случаях направление линий обязательно обозначается

стрелкой, т. е. стрелки ставятся в направлениях  (справа налево и снизу вверх) и когда есть излом.

10. Нумерация блоков. В сложных схемах блоку может быть присвоен номер, например, для ссылки в других частях документации.

Блоки нумеруются сверху вниз, слева направо. Номер ставится в левом верхнем углу над блоком.

Пример схемы алгоритма (линейный алгоритм). Даны A, B . Найти $S = A + B$ (рис. 1.2).

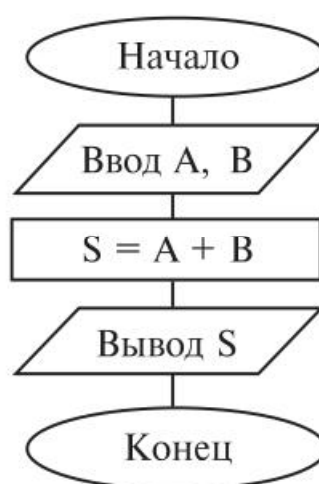


Рис. 1.2. Пример схемы линейного алгоритма

Пояснение. Схема алгоритма для решения этой задачи достаточно простая (линейная), поэтому нумерация блоков не требуется. Сначала с клавиатуры вводятся значения слагаемых A и B (блок «Ввод A, B »). Далее (блок « $S = A + B$ ») вычисляется значение S . После этого вычисленное значение S выводится на экран (блок «Вывод S »).

Пример схемы алгоритма (ветвящийся алгоритм). Найти $Q = \min\{A, B\}$ (Q равно наименьшему значению из A и B) (рис. 1.3).

Пояснение. Блок 1 — начало алгоритма (решения). В блоке 2 с клавиатуры вводятся значения неизвестных A и B . В блоке 3 проверяется условие « $A < B$ ». Если это условие выполняется (A меньше B , т. е. A является наименьшим из двух чисел), то переменной Q присваивается значение переменной A (блок 4). Если условие « $A < B$ » не выполня-

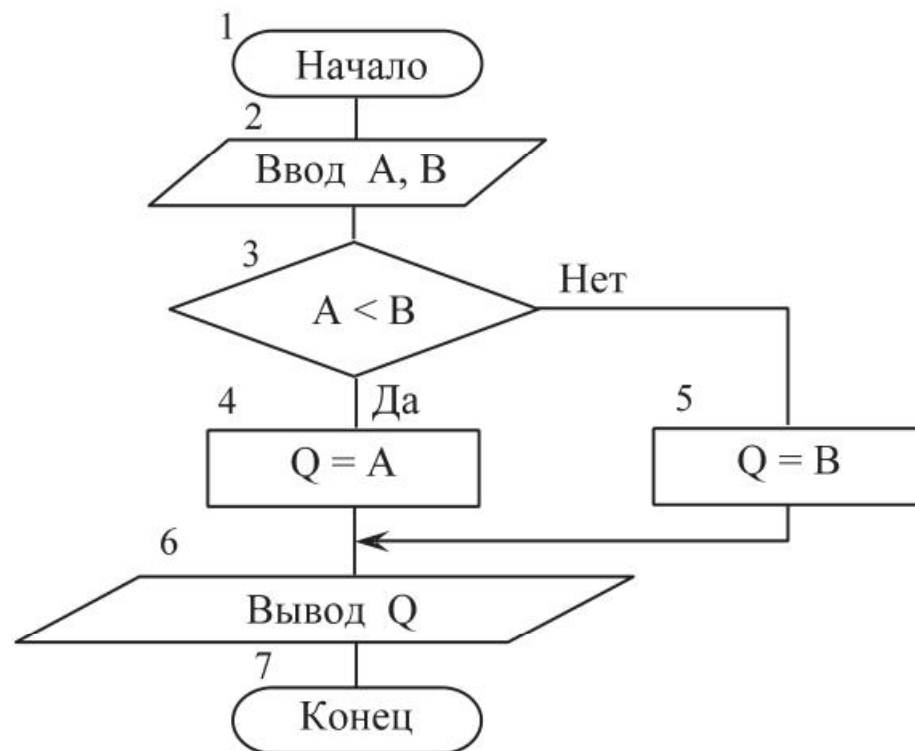


Рис. 1.3. Пример схемы ветвящегося алгоритма

ется (A больше или равно B , т. е. B является наименьшим из двух чисел), то переменной Q присваивается значение переменной B (блок 5). После этого в блоке 6 вывода выводится вычисленное значение Q . Блок 7 — конец алгоритма (решения).

1.3. Рекомендации по стилю программирования

1. При разработке алгоритма и написании программы необходимо использовать технологию *программирования сверху вниз*: задача разбивается на более простые подзадачи до тех пор, пока не станут ясны все детали решения.

2. При написании программы лучше использовать один и тот же способ записи текста программы, чтобы со временем легче было ее модифицировать.

3. Каждый оператор нужно начинать с новой строки. Следует избегать строк, длина которых превышает ширину экрана.

4. Для внесения определенности во вложенность управляющих структур следует записывать операторы программы следующим образом: ключевые слова, начинающие и заканчивающие некоторый оператор, записываются с одинаковым отступом от левой границы окна редактора, а все вложенные операторы при записи сдвигаются относительно этих слов вправо на 2—3 позиции (см. главы 3—20).

5. При написании программы желательно использовать комментарии к разделам описаний и в отдельных блоках программы.

6. Идентификаторы (имена переменных, меток, констант, типов, подпрограмм) в программе должны соответствовать своему назначению по условию задачи.

7. В разделах описаний следует группировать описания по назначению и типам (см. главы 6—17).

Задания

Составьте для решения каждой задачи схему алгоритма с комментариями.

1. Даны A, B . Найти $P = A * B$.

2. Вычислить $S = V \cdot T$.

3. Вычислить $E = M \cdot C^2/2$.

4. Сколько времени в минутах школьник затратит на дорогу от дома до школы, если известно расстояние S и средняя скорость движения школьника V км/ч?

5. Ввести количество минут, прошедших с 0 часов. Определить время в часах и минутах ($[...]$ — обозначение в математике целой части от числа, остаток от целочисленного деления написать словами в вычислительном блоке).

6. Вычислить значения Y по формуле $Y = \frac{7x - 4}{5x + 3}$.

7. Вычислить объем и площадь поверхности параллелепипеда.

8. Вы положили деньги в банк на счет из расчета 60 % годовых от исходной суммы (накопления процентов ежемесячно не происходит). Составьте схему алгоритма, которая будет вычислять причитающийся вам доход, когда вы будете задавать ей, сколько месяцев деньги лежат в банке.

Контрольные вопросы

1. Как следует трактовать понятие «задача» в программировании? Приведите примеры.
2. Что понимается под «результатом решения задачи» в программировании? Поясните на примерах.
3. Что представляет собой процесс решения задач на компьютере?

4. Перечислите этапы решения задач.
5. В чем заключается каждый этап решения задачи на компьютере? Поясните на примере.
6. Что такое алгоритм?
7. Какие виды алгоритмов вы знаете? Приведите примеры.
8. Каким образом можно записать алгоритм?
9. Что такое схема алгоритма?
10. Какие графические символы используются при составлении схем алгоритмов? Назовите назначение каждого символа.
11. В каких случаях на схеме алгоритма следует использовать стрелки?
12. Как на схеме алгоритма показать начало и конец алгоритма (решения)?
13. Каким символом на схеме алгоритма обозначается ввод данных и вывод результатов? Приведите примеры.
14. Как на схеме алгоритма можно показать вычисление по формуле или несколько подряд идущих вычислений по формулам? Приведите примеры.
15. Каким образом рекомендуется записывать программы?

Глава 2

ОСНОВЫ РАБОТЫ В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКИ MICROSOFT VISUAL STUDIO 2010

В настоящее время для разработки программного обеспечения используются интегрированные среды разработчика — ИСР (англ. **IDE — Integrated development environment**). Интегрированная среда разработчика позволяет повысить производительность и эффективность работы программиста. Одной из интегрированных сред разработки и является **Microsoft Visual Studio 2010**, рассматриваемая в данной главе.

2.1. Назначение и состав среды программирования Microsoft Visual Studio 2010

Создание программы (приложения) в среде программирования **Microsoft Visual Studio 2010** начинается с создания проекта (**project**). Проект содержит текст программы (написанный программистом), а также всю служебную информацию, необходимую для создания и выполнения программы.

Microsoft Visual Studio 2010 представляет собой интегрированную среду разработки программного обеспечения, которая содержит:

- 1) *редактор текста* — для создания и редактирования текста программы на языке высокого уровня (на языке **C++**);
- 2) *препроцессор* — для замены комментариев пустыми строками, включения файлов с заголовками функций, обработки макроподстановок, условной компиляции и т. д.;
- 3) *компилятор* — для перевода текста программы с языка высокого уровня в машинные (объектные) коды;

4) *компоновщик* (редактор связей или линкер, англ. link editor, linker) — для формирования связей между программой и библиотеками стандартных средств языка (внешних, ранее созданных), т. е. формирования загрузочного файла (файла с расширением *.exe*);

5) *загрузчик* — для выполнения загрузочного файла программы.

2.2. Процесс создания и выполнения программы в среде программирования Microsoft Visual Studio 2010

Процесс создания и выполнения программы состоит в следующем: до выполнения программы необходимо подготовить ее текст в файле с расширением **.cpp*. Передать этот файл на компиляцию и устранить синтаксические ошибки, выявленные компилятором; безошибочно откомпилировать (получится объектный файл с расширением **.obj*); дополнить объектный файл нужными библиотечными

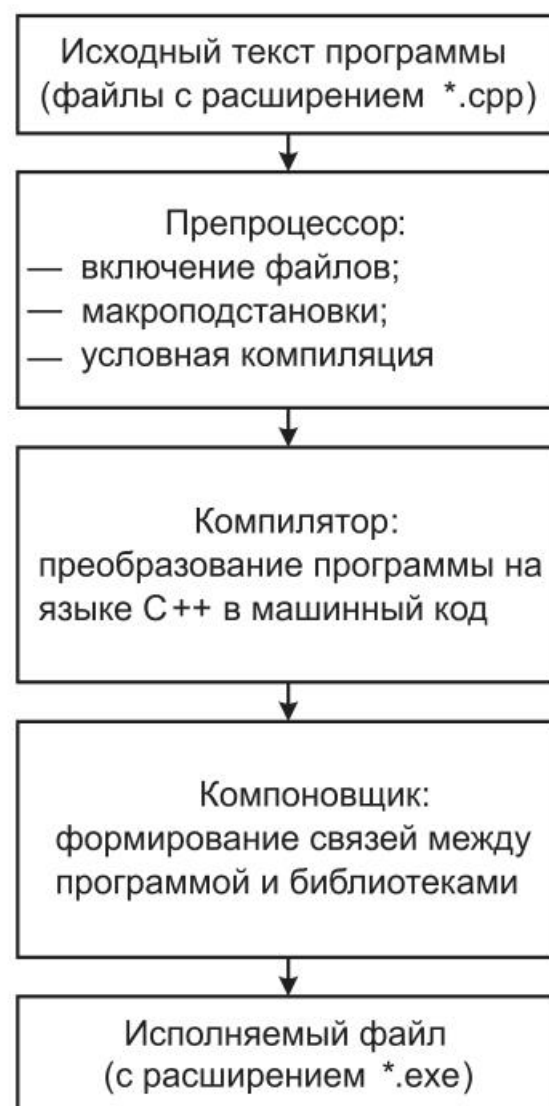


Рис. 2.1. Схема процесса подготовки исполняемой программы

функциями (компоновка) и получить исполняемый файл программы с расширением *.exe. Схема данного процесса представлена на рис. 2.1.

2.3. Запуск интегрированной среды разработки Microsoft Visual Studio 2010

Запуск интегрированной среды разработки Microsoft Visual Studio 2010 можно осуществить несколькими способами.

1-й способ: кнопка **Пуск** → **Программы** → **Microsoft Visual Studio 2010** → **Microsoft Visual Studio 2010**;

2-й способ: найти и запустить загрузочный файл *devenv.exe*;

3-й способ: запустить с помощью ярлыка **Microsoft Visual Studio 2010** на Рабочем столе.

После запуска появится главное окно среды **Microsoft Visual Studio 2010**, представленное на рис. 2.2.

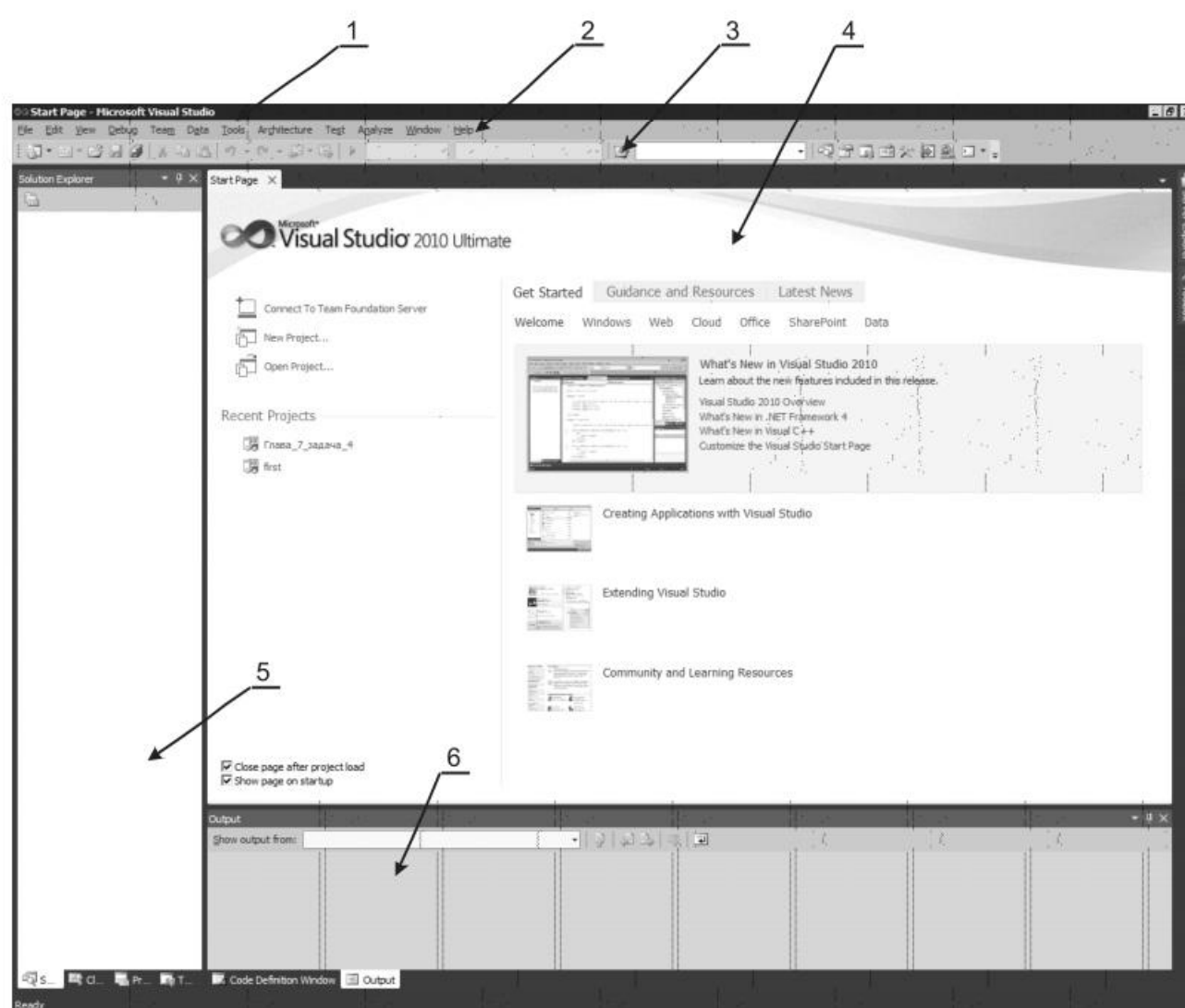


Рис. 2.2. Окно среды Microsoft Visual Studio 2010

Окно среды **Microsoft Visual Studio 2010** имеет следующие элементы:

1 — заголовок окна программы;

2 — меню **Microsoft Visual Studio 2010**:

File	— операции с файлами;
Edit	— редактирование текста программы;
View	— просмотр файлов проекта;
Project	— операции с проектами;
Build	— компиляция проекта;
Debug	— выполнение и отладка программы;
Team	— командная разработка;
Data	— работа с данными;
Tools	— выбор режимов и настроек среды разработки;
Architecture	— работа с диаграммой проекта;
Test	— тестирование проекта;
Analyze	— анализ кода программы;
Window	— операции с окнами документов;
Help	— справка;

3 — панель инструментов;

4 — окно **Start Page**, которое позволяет начать создание нового проекта; открыть ранее созданный проект или продолжить работу с недавно открывавшимися проектами;

5, 6 — окна **Solution Explorer** и **Output**, которые будут использоваться в процессе работы с проектом.

2.4. Работа с проектами в среде Microsoft Visual Studio 2010

Каждую задачу, решаемую с использованием среды **Microsoft Visual Studio 2010**, рекомендуется оформлять в виде самостоятельного проекта. Рассмотрим подробнее основные этапы работы с проектами.

Создание проекта. Выполнить команду **File** → **New** → **Project** или нажать кнопку **New Project...** в окне **Start Page**.

В результате этих действий появляется окно создания нового проекта, показанное на рис. 2.3.

1. В левой части окна создания нового проекта в предложенном дереве выбрать раздел **Visual C++**.

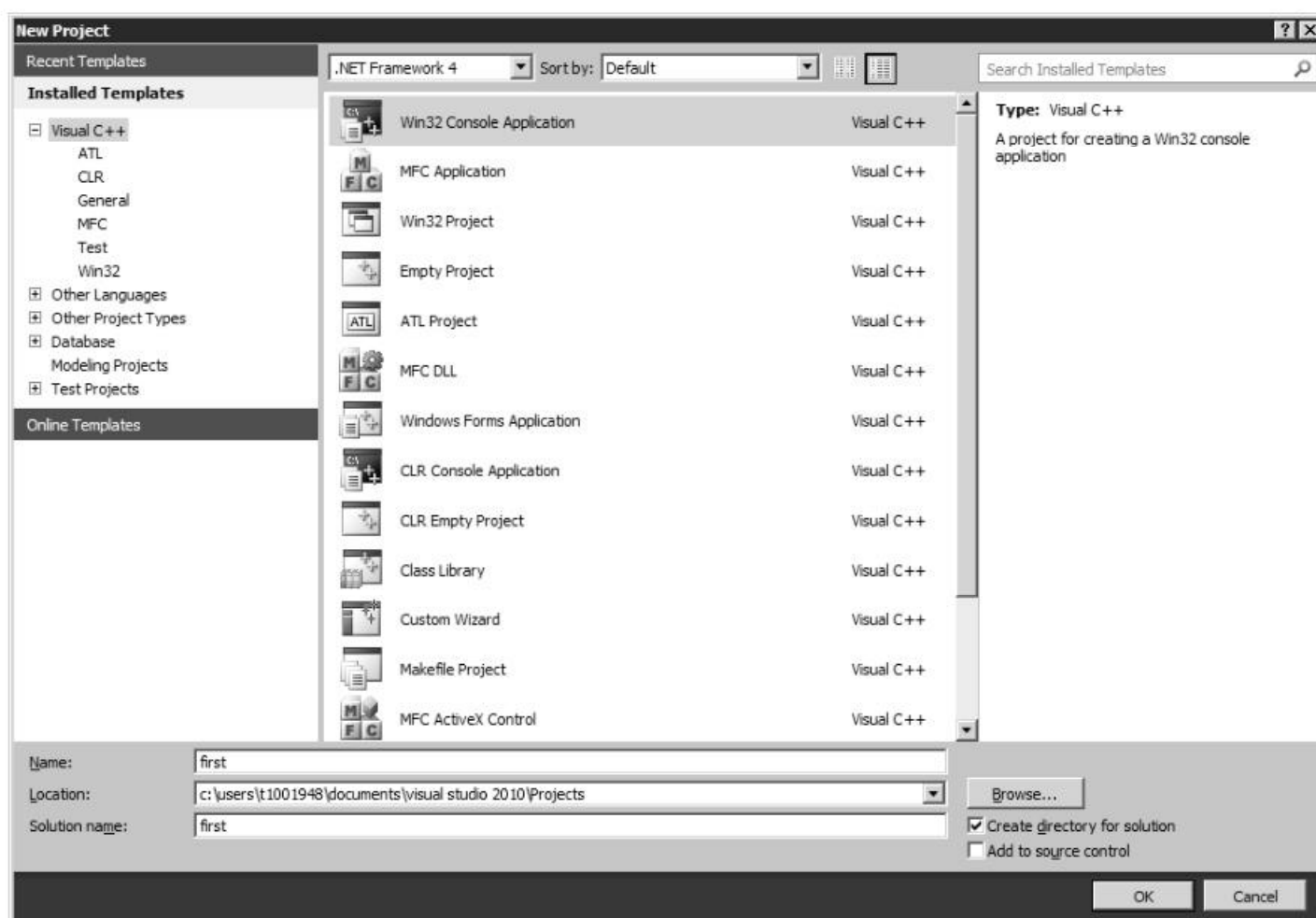


Рис. 2.3. Окно создания нового проекта

2. В средней части окна создания проекта выбрать строку **Win32 Console Application** (рассматриваем пример создания консольного приложения для **Win32**).

3. В поле **Location** с помощью кнопки **Browse...** выбрать папку, в которую будет сохранен проект.

4. В поле **Name** ввести имя проекта, например *test*, и нажать кнопку **OK**. В результате появится окно мастера создания проектов — **Win32 Application Wizard**. На первом шаге мастер выдает сводную характеристику о создаваемом проекте (рис. 2.4).

5. Нажать кнопку **Next**, перейти к выбору доступных опций проекта (рис. 2.5).

6. В окне мастера создания проектов — пункт **Application Type** выбрать тип проекта **Console Application**.

7. В окне мастера создания проектов — пункт **Additional options** выбрать пункт **Empty project**.

8. После нажатия кнопки **Finish** мастер создания проектов завершает свою работу, а в окне **Solution Explorer** появится дерево папок, в которых будут располагаться файлы проекта (рис. 2.6).

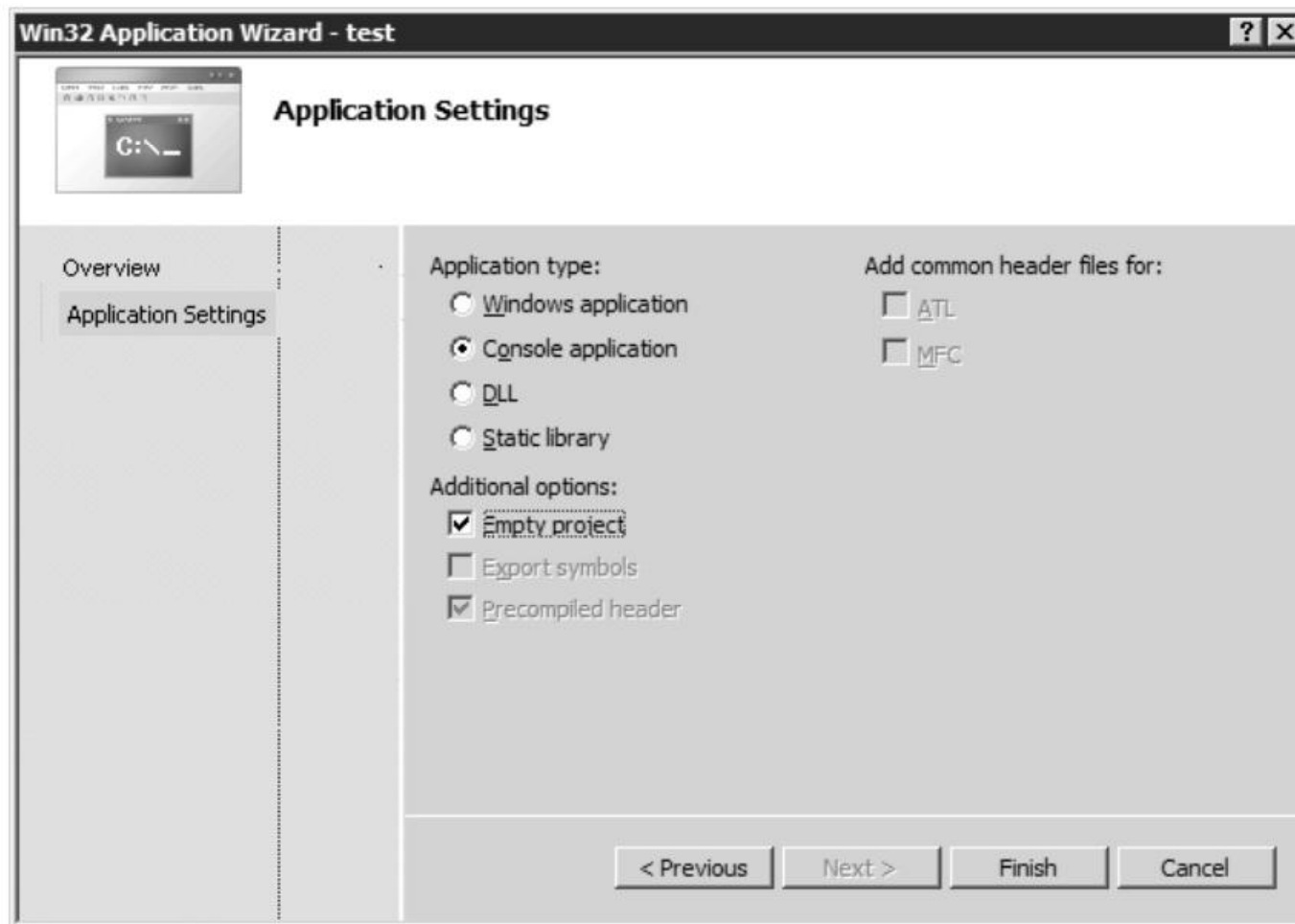


Рис. 2.4. Окно мастера создания проектов — Шаг 1

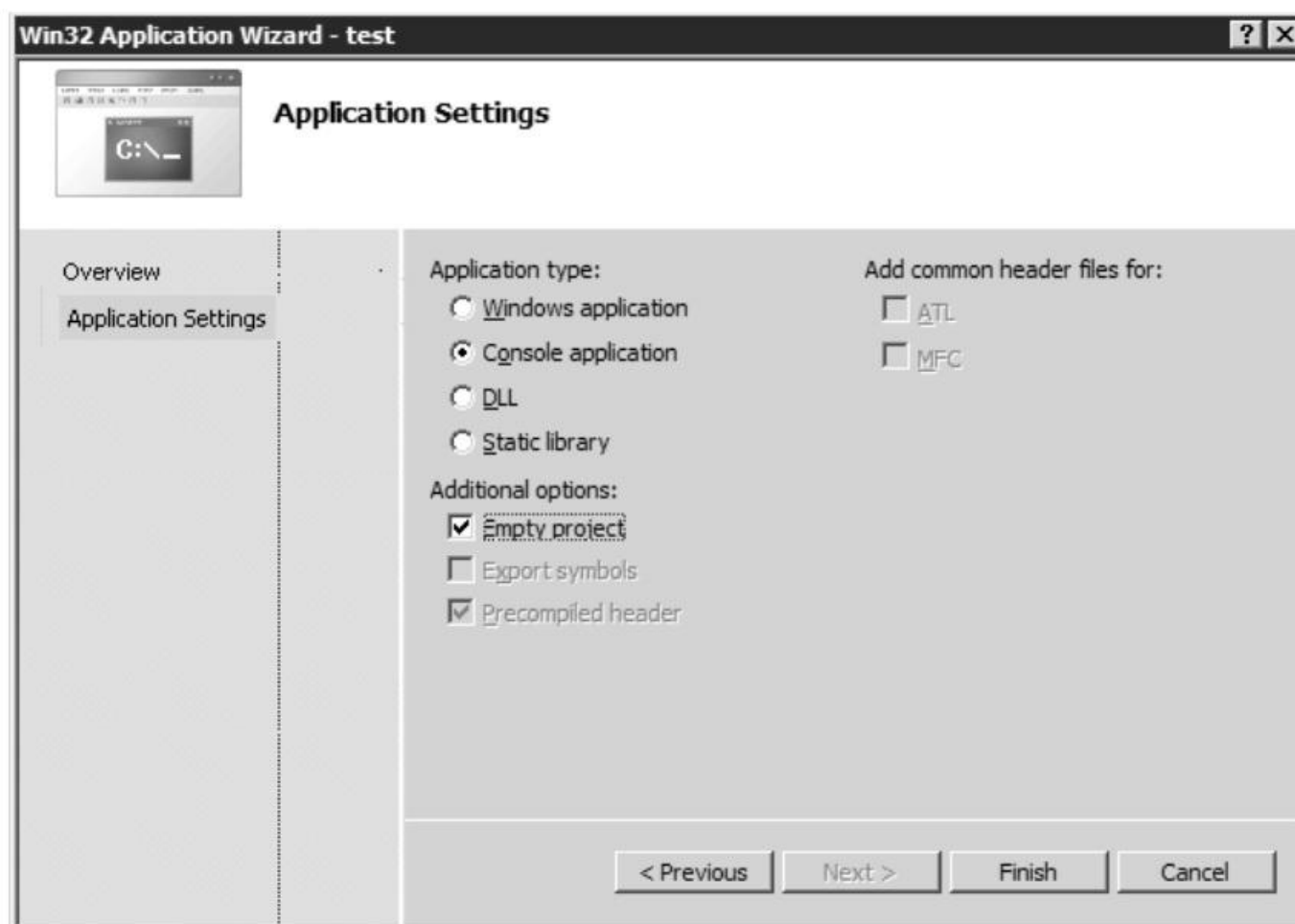


Рис. 2.5. Задание настроек проекта в окне мастера создания проектов

По умолчанию в состав проекта входят:

- папка **Header Files** — содержит заголовочные файлы (*.h);
- папка **Resource File** — содержит файлы с ресурсами проекта (изображения, иконки и т. д.);
- папка **Source Files** — содержит файлы с исходным кодом программы (*.cpp).

9. Следующим этапом необходимо добавить файлы в созданный проект. Для этого в окне **Solution Explorer** нужно щелкнуть правой кнопкой мыши по папке **Source Files**, в появившемся меню выбрать пункт **Add**, далее пункт **New Item** (рис. 2.6).

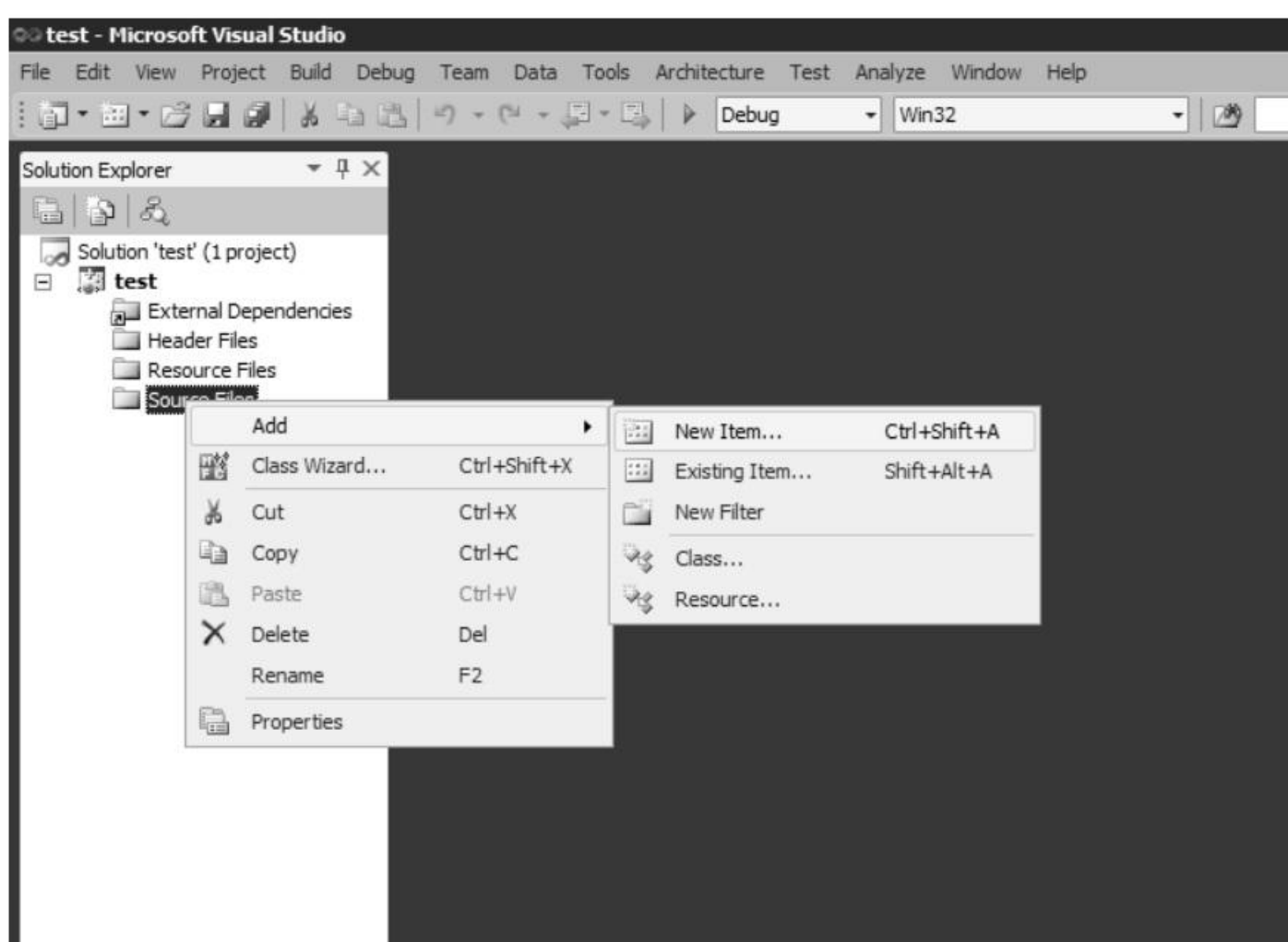


Рис. 2.6. Окно создаваемого проекта с контекстным меню **Source Files** и подменю **Add**

В результате появится окно добавляемых в проект элементов (рис. 2.7).

10. В левой части окна добавляемых в проект элементов в предложенном дереве выбрать **Visual C++**.

11. В средней части окна добавляемых в проект элементов выбрать элемент **C++ File (*.cpp)**.

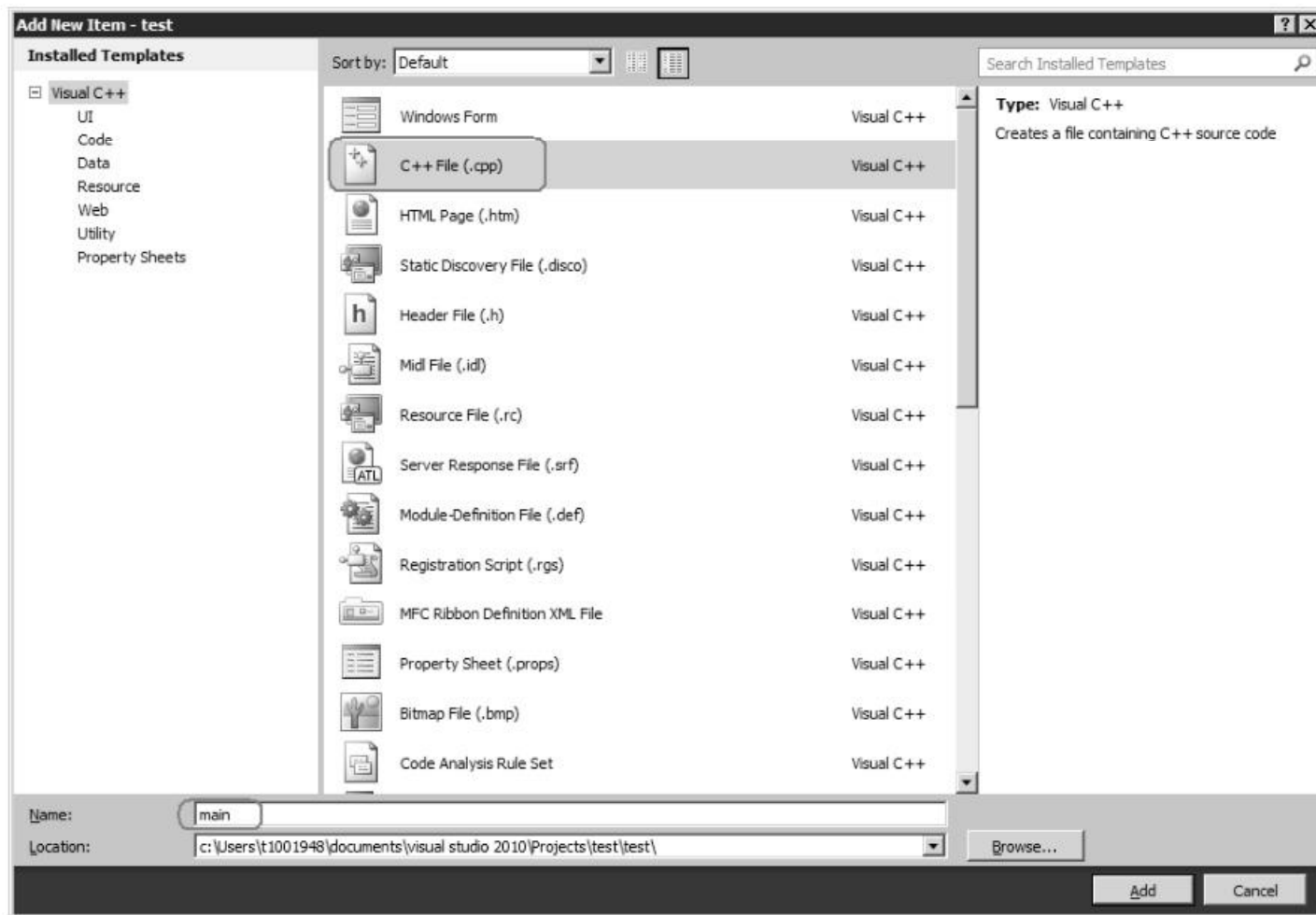


Рис. 2.7. Окно добавляемых в проект элементов

12. Поле **Location** должно содержать имя папки, в которую был сохранен проект, при желании можно изменить указанную директорию.

13. В поле **Name** ввести имя файла. Для основного файла проекта рекомендуется использовать имя — *main* и нажать кнопку **Add**. В результате появится окно текстового редактора исходного кода программы (рис. 2.8).

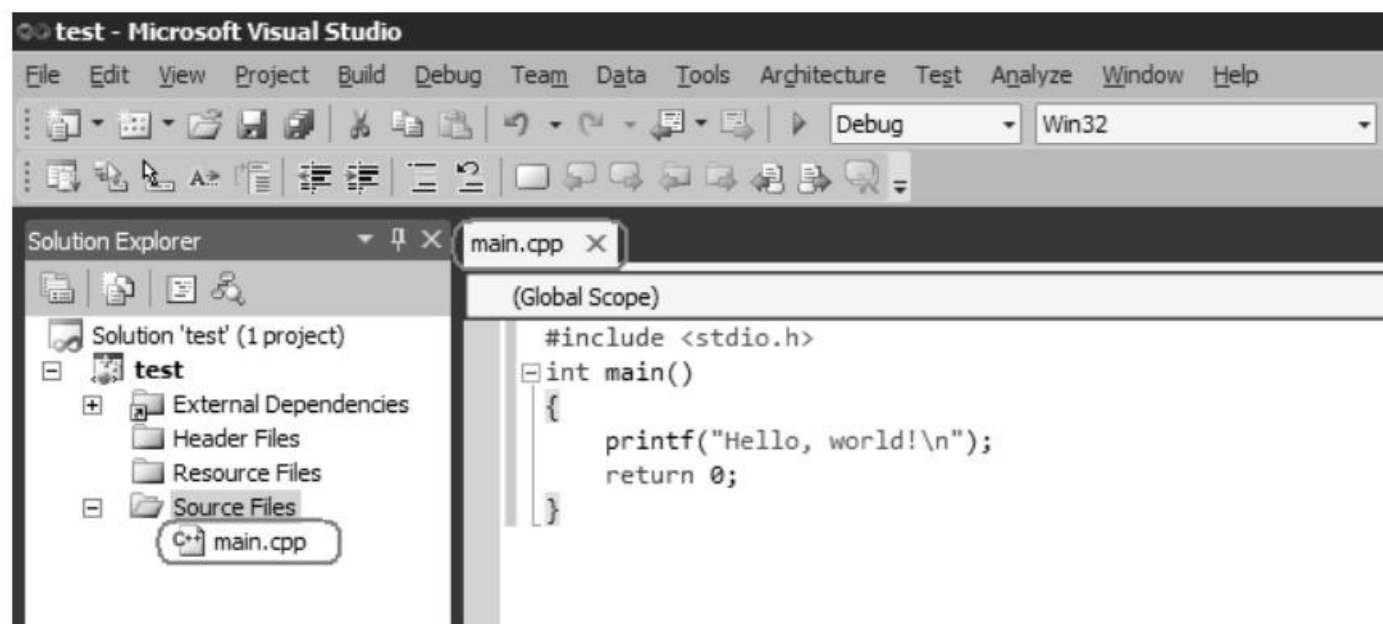


Рис. 2.8. Окно с текстовым редактором кода программы

14. После набора текста программы необходимо сохранить проект.

Сохранение проекта. После внесения любых изменений в проект (добавление файлов, редактирование файлов и т. д.) необходимо выполнить сохранение проекта. Для этого можно воспользоваться следующими способами.

1-й способ: выполнить команду **File** → **Save** «имя проекта» (если проект состоит из одного файла) или **File** → **Save** «имя файла» (если проект состоит из нескольких файлов).

Данная команда сохраняет изменения только в том файле, который открыт в окне текстового редактора. То есть если были изменены несколько файлов, необходимо поочередно открыть их и дать команду на сохранение изменений.

2-й способ: выполнить команду **File** → **Save All**. При выборе этой команды сохраняются все измененные файлы проекта.

Закрытие проекта. После окончания работы с проектом необходимо закрыть его. Это действие можно выполнить несколькими способами.

1-й способ: закончить работу со средой **Microsoft Visual Studio 2010**, выполнив команду **File** → **Exit**.

2-й способ: выполнить команду **File** → **Close solution**.

Отметим, что команда **File** → **Close** закрывает только файл, открытый в данный момент в окне редактора. При этом проект остается открытым.

Открытие проекта. Если созданный проект был закрыт, то его можно открыть для продолжения работы с ним. Это действие можно выполнить несколькими способами.

1-й способ: после запуска **Microsoft Visual Studio 2010** в окне **Start Page** выбрать пункт **Open Project...** (см. рис. 2.2).

2-й способ: если проект недавно уже открывался на компьютере, то ссылка на него будет доступна в окне **Start Page** в разделе **Recent Projects**. Достаточно щелкнуть левой кнопкой мыши по имени требуемого проекта, и он будет загружен в среду **Microsoft Visual Studio 2010**.

3-й способ: выполнить команду **File** → **Open** → **Project/Solution...** В появившемся окне **Open Project** необходимо указать файл проекта, имеющий расширение **.sln* (проекты, созданные в более ранних версиях среды **Microsoft Visual Studio**, могут иметь расширение **.dsw*), и нажать кнопку **Открыть**.

4-й способ: непосредственно в директории, где хранится проект, найти файл проекта (*.sln) и, выполнив двойной клик мышки, загрузить этот проект.

Следует отметить, что открытие файла с исходным текстом программы (*.cpp) не равноценно открытию проекта. Поэтому рекомендуется открывать именно проекты (*.sln), а не файлы с исходным кодом программы (*.cpp).

2.5. Компиляция, компоновка и выполнение проекта

Во время компиляции осуществляется:

- проверка текста программы (проекта) на наличие синтаксических ошибок;
- если ошибки отсутствуют, то происходит перевод исходного текста программы (проекта) с языка C++ на язык машинных кодов, т. е. создание объектного кода программы (проекта).

Во время компоновки осуществляется:

- подключение к объектному коду программы (проекта) объектных кодов внешних ранее созданных средств языка C++ (библиотек);
- создание загрузочного файла программы (проекта) (имя проекта с расширением .exe).

После этого загрузчик запускает программу (проект) на выполнение.

Для запуска процесса компиляции необходимо выполнить команду **Build** → **Build Solution** (клавиша **F7**) (рис. 2.9).

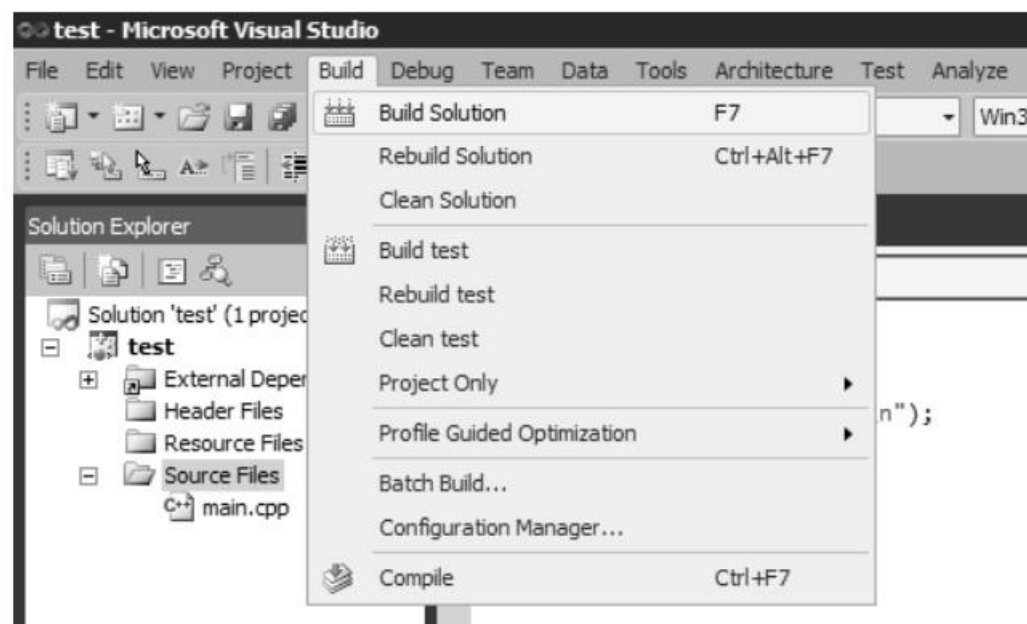


Рис. 2.9. Окно проекта с меню **Build**

Результат компиляции отображается в окне **Output**. Если в программе (проекте) есть ошибки, то компиляция не будет завершена успешно и в окне **Output** появятся сообщения об ошибках (рис. 2.10).

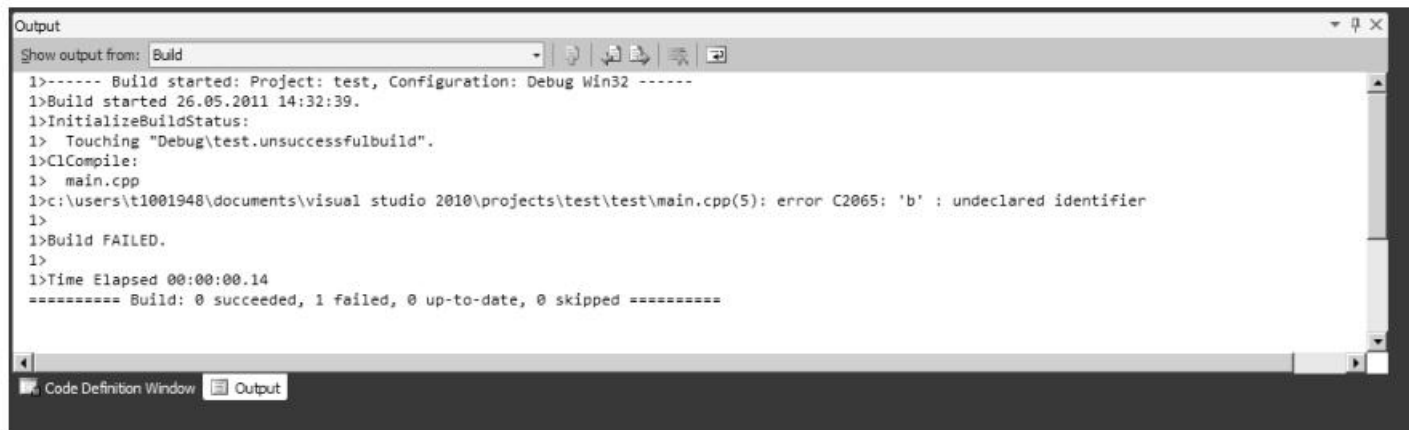


Рис. 2.10. Окно с результатами компиляции

Найти ошибку в программе можно следующим образом. В окне **Output** дважды щелкнуть мышью по строке, содержащей слово **error**. После этого курсор автоматически переключается на строку в исходном тексте программы, содержащей выбранную ошибку (рис. 2.11).

После исправления всех ошибок необходимо повторить команду **Build** → **Build Solution** и добиться успешной компиляции проекта (рис. 2.12).

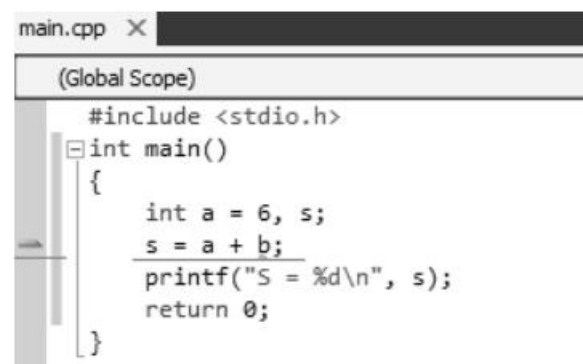


Рис. 2.11. Строка исходного кода, содержащая ошибку

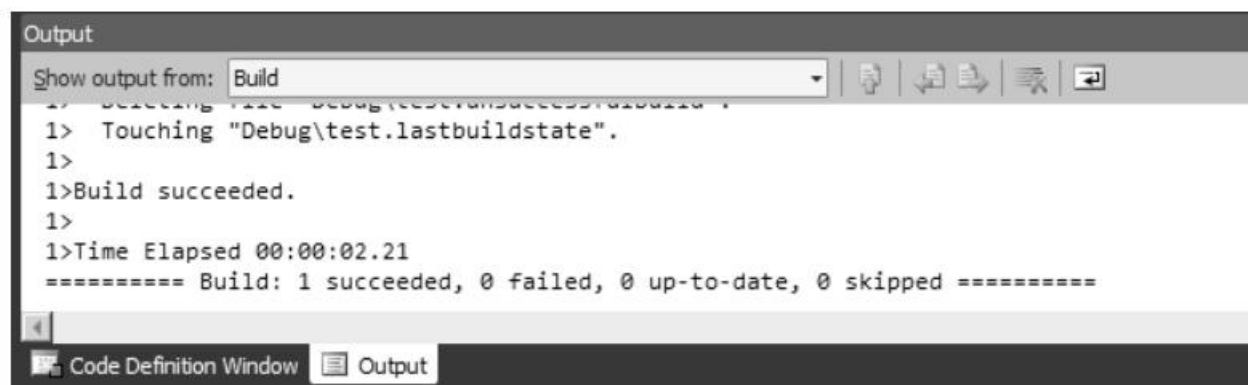


Рис. 2.12. Окно **Output** после успешной компиляции проекта

После успешной компиляции можно выполнить программу: **Debug → Start Without Debugging** (сочетание клавиш **Ctrl + F5**) (рис. 2.13). Эта команда осуществляет запуск программы без режима отладки. Подробнее режим отладки будет рассмотрен ниже по тексту.

Результат работы программы представлен на рис. 2.14.

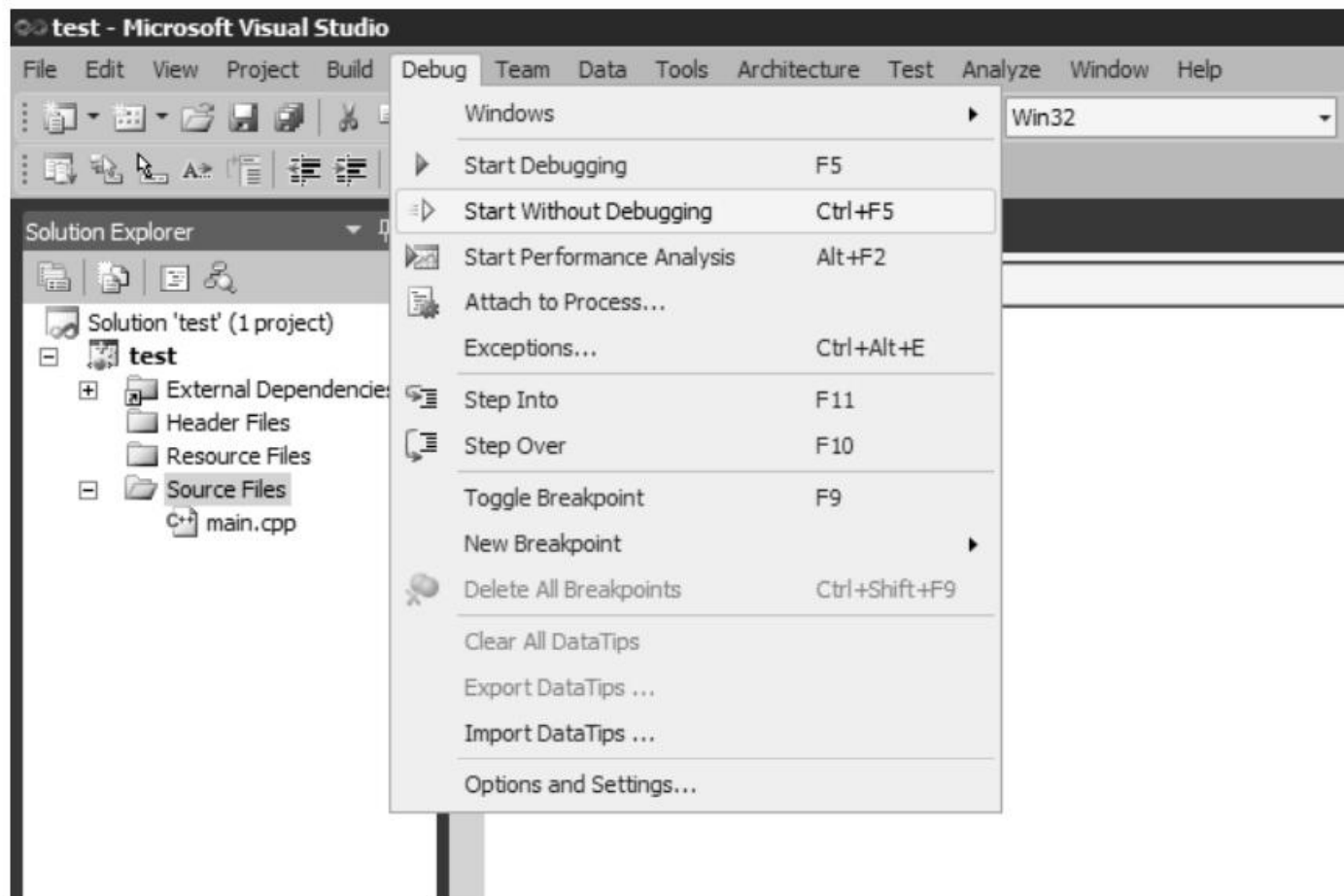


Рис. 2.13. Окно проекта с меню **Debug**

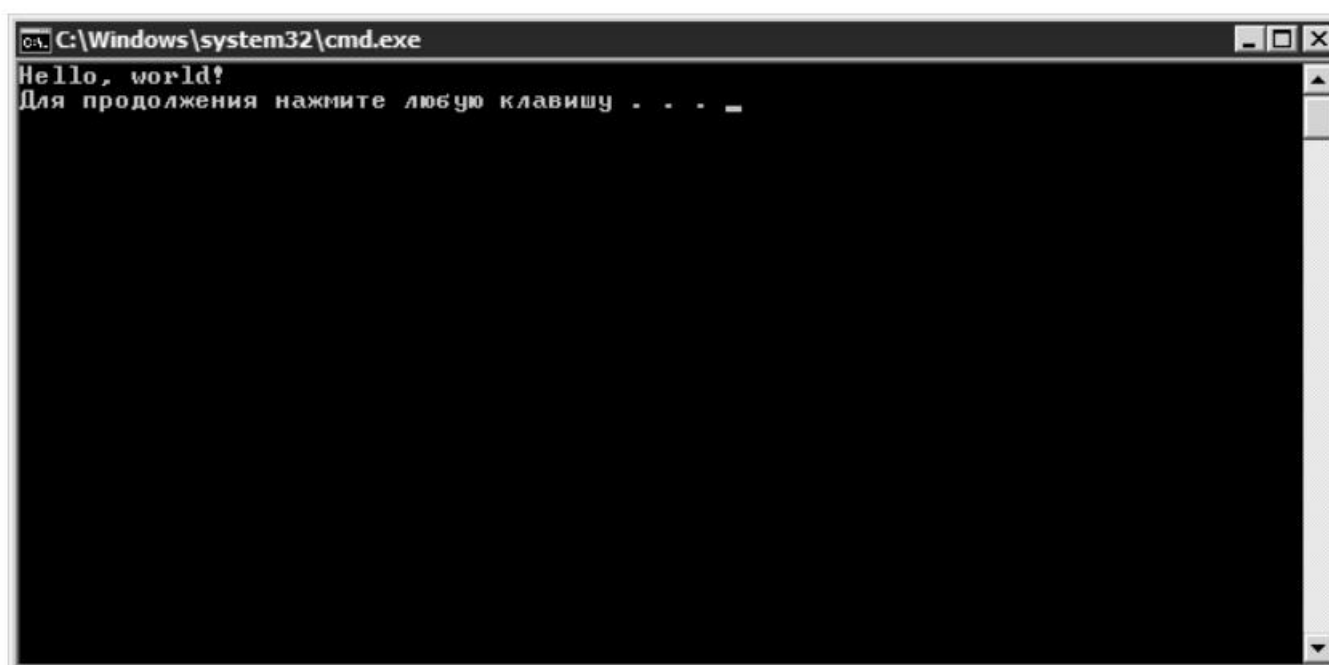


Рис. 2.14. Окно выполнения программы

После завершения работы с проектом его нужно закрыть, выполнив команду **File** → **Close Solution**. Данное действие нужно выполнять перед началом работы с новым проектом.

2.6. Отладка проекта

Любой серьезный проект требует отладки. Программисту необходимо проверить правильность выполнения написанных им алгоритмов, а в случае некорректной работы программы найти и устранить возникшие ошибки.

Режим отладки проекта дает программисту возможность:

- указать точки останова в программе (точки, где выполнение программы будет остановлено);
- запустить программу до ближайшей точки останова;
- выполнить программу по шагам;
- просмотреть значения переменных в процессе выполнения программы.

Рассмотрим действия, которые необходимо выполнить для отладки проекта.

Установка точки останова. В файле с исходным кодом программы нужно установить курсор на строчку, где требуется остановка. После этого выполнить одно из действий:

- а) **Debug** → **Toggle Breakpoint**;
- б) нажать горячую клавишу **F9**;
- в) щелкнуть мышкой слева от текста программы напротив требуемой строки (рис. 2.15).

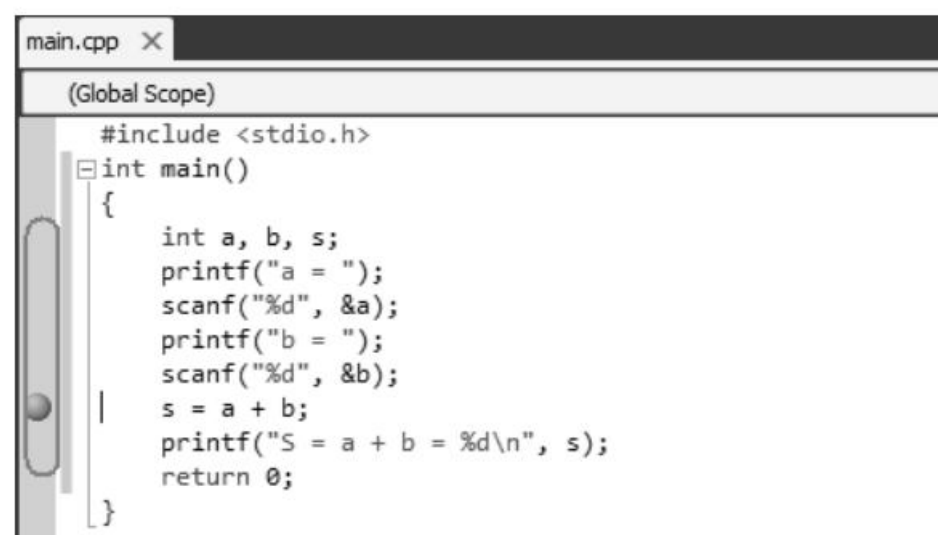


Рис. 2.15. Окно программы с полем для задания точек останова программы

Запуск программы до ближайшей точки останова. После указания точки останова можно запустить программу: команда **Debug** → **Start Debugging** или нажать горячую клавишу **F5**.

Программа начнет выполняться до тех пор, пока не будет достигнута точка останова. О достижении этой точки будет свидетельствовать стрелочка, отображаемая поверх точки останова (рис. 2.16). В этой точке выполнение программы будет остановлено до указания дальнейших действий.

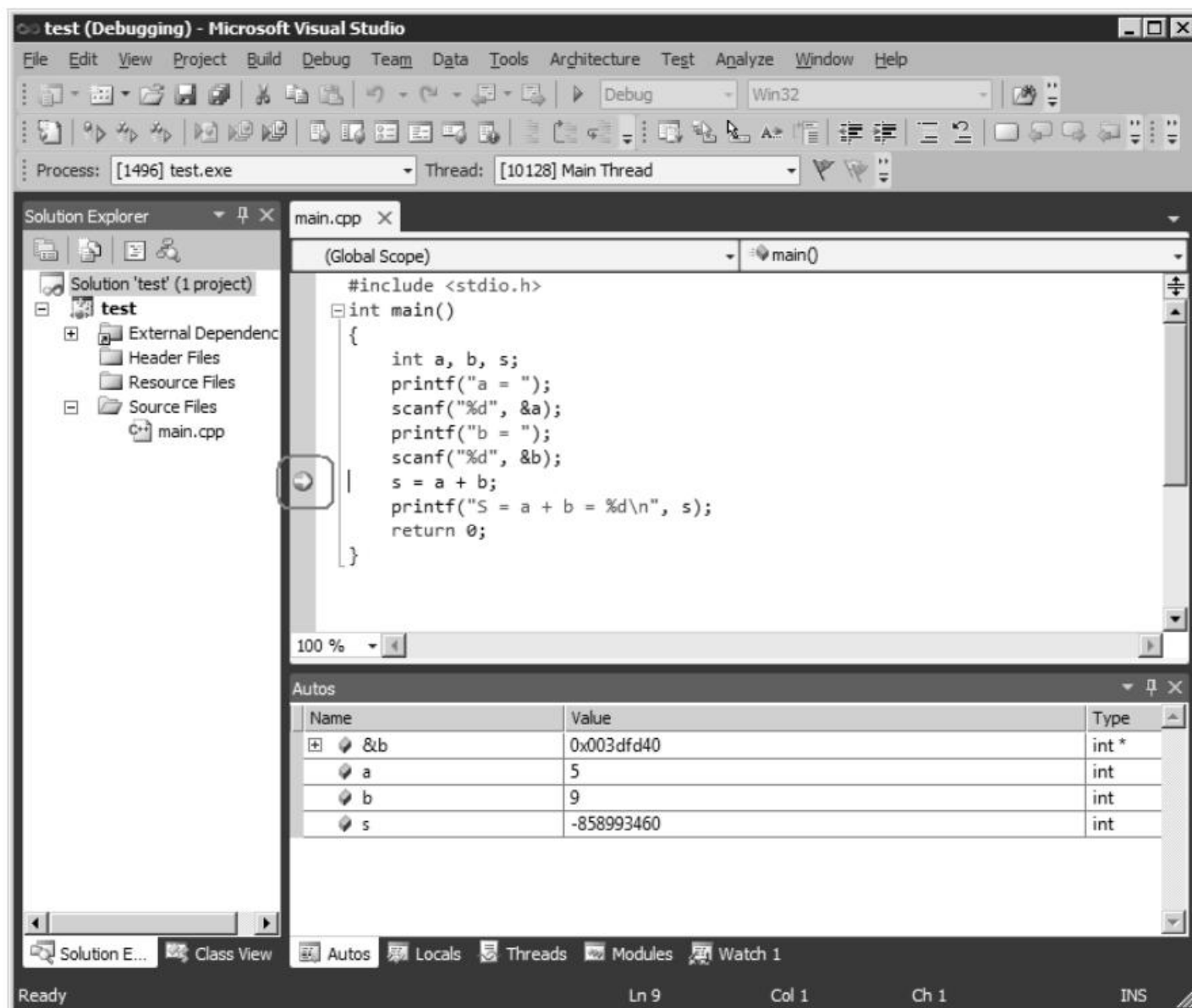


Рис. 2.16. Достижение точки останова в программе

Выполнение программы по шагам. Выполнение программы по шагам можно осуществить следующим образом. Команда **Debug** → **Step Over** выполняет одну строчку программы.

Если очередным шагом программы является вызов функции, то программист может проанализировать работу этой функции. Для этого в режиме отладки используется команда **Debug** → **Step Into** или го-

рячая клавиша **F11**. После этой команды отладчик заходит «внутрь» функции и программист может продолжить пошаговое выполнение команд. Если отладка функции закончена и необходимо вернуться в вызывающую программу, то можно использовать команду **Debug** → **Step Out** или сочетание клавиш **Shift+F11**.

Просмотр значения переменных в процессе выполнения программы. В любой момент отладки программист может просмотреть значения переменных, используемых в программе. Для этого используется нижняя часть окна среды **Microsoft Visual Studio 2010** (рис. 2.16). В этом подокне доступны несколько вкладок:

Autos — список переменных, создаваемый отладчиком автоматически;

Local — список локальных переменных;

Threads — список потоков;

Modules — список модулей, используемых для работы программы;

Watch 1 — список переменных, указываемых программистом самостоятельно.

Для наблюдения значений переменных используются вкладки **Autos** и **Watch 1**.

Для добавления переменной на вкладку **Watch 1** необходимо сделать ее активной и ввести с клавиатуры имя переменной. Если указанная переменная имеется в программе, то ее значение будет автоматически отражено в поле **Value**.

Таким образом, используя описанные приемы, программист может тщательно проанализировать выполнение написанной программы, проследить изменения значений используемых переменных, устранить некорректную работу реализованных алгоритмов и добиться правильного выполнения программы.

Контрольные вопросы

1. Что такое ИСП (IDE)?
2. Какие составные части включает в себя среда Microsoft Visual Studio 2010?
3. Опишите процесс создания исполняемого файла программы (*.exe).
4. Какими способами можно создать проект в среде Microsoft Visual Studio 2010?
5. Какие команды служат для сохранения проекта? Объясните различие между доступными командами.

6. Как закрыть проект после окончания работы с ним?
7. Что включает в себя компиляция проекта?
8. Как найти ошибку в тексте программы, если она была обнаружена компилятором?
9. Какие возможности отладки проекта предоставляет среда Microsoft Visual Studio 2010?
10. Каким образом можно просмотреть значения переменных при отладке проекта?

Глава 3

ВВЕДЕНИЕ В ЯЗЫК C++

Язык C++ используется практически во всех областях науки, техники и производства. Он позволяет удобно, эффективно и быстро разрабатывать программы, тестировать и изменять их. Язык C++ используется для решения множества технических, научных и инженерных задач. Его можно использовать для создания программ широкого пользования, в частности в локальных и глобальных сетях, численных расчетах, интерактивной работе с пользователями, при создании и использовании баз данных и т. д.

В данной главе рассматриваются некоторые основные элементы языка C++.

3.1. Алфавит, лексемы, разделители

Алфавит. Алфавит любого языка программирования — это набор допустимых символов этого языка. В языке C++ используются следующие символы:

- прописные и строчные латинские буквы:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z;
a b c d e f g h i j k l m n o p q r s t u v w x y z;
- десятичные цифры: 0 1 2 3 4 5 6 7 8 9;
- знак подчеркивания;
- буквы русского алфавита (для комментариев и вывода сообщений на экран);
- специальные символы:

Лексемы языка C++. Лексема — это логически выделенная единица языка, воспринимаемая как единое целое компилятором и программистом.

Лексемы бывают следующих видов:

- идентификаторы;
- ключевые слова;

Таблица 3.1. Специальные символы

Символ, пояснение	Символ, пояснение	Символ, пояснение
" кавычки	– минус	~ тильда
() круглые скобки	/ дробная черта (слэш)	^ циркумфлекс
, запятая	\ обратный слэш	& амперсанд
знак «ИЛИ»	> больше	: двоеточие
[] квадратные скобки	= равно	; точка с запятой
{ } фигурные скобки	< меньше	' апостроф
. точка	! восклицательный знак	# номер
+ плюс	? вопросительный знак	* звездочка

- константы;
- знаки операций;
- разделители.

Идентификаторы (имена) служат для обозначения объектов программы (переменных, констант, меток, функций и т. д.).

Ключевые (служебные) слова — это такие лексемы, которые используются системой программирования для своих специальных целей (имена операторов, директив препроцессора и т. д.). *Препроцессор* — специальная служебная программа, работающая перед компилятором и используемая для вставки библиотечных файлов, определения констант и т. д. *Директива* — это команда препроцессору выполнить то или иное действие.

Константа — это лексема, представляющая изображение фиксированного числового, строкового или символьного значения, например: 100, 3.14159, «Здравствуй, мир!», 'Y'.

Знаки операций — это лексемы, используемые при вычислении выражений, которые определяют порядок и правила вычисления значения и могут содержать имена переменных, константы, знаки операций, имена функций и скобки для определения порядка вычислений.

Разделители используются для отделения друг от друга (разделения) лексем языка. Например, *запятая* ',' разделяет элементы списка.

При анализе выражения или оператора компилятор воспринимает их как последовательность лексем (табл. 3.2).

Разделители. *Разделители* используются для отделения друг от друга (разделения) лексем языка. В языке C++ используются следующие разделители:

[] () { } , ; : ... и т. д.

Таблица 3.2. Лексический разбор

Лексический разбор выражения 5*sin(x+Pi/2)	Лексический разбор оператора scanf("%d",&x);
5 — константа	scanf — идентификатор
* — знак операции	(— разделитель
(— разделитель	" — разделитель
x — идентификатор	% — знак операции
+ — знак операции	d — идентификатор
Pi — идентификатор	" — разделитель
/ — знак операции	, — разделитель
2 — константа	& — знак операции
) — разделитель	x — идентификатор
) — разделитель
	; — разделитель

Квадратные скобки '[']' ограничивают индексы одно- и многомерных массивов и индексированных элементов.

Круглые скобки '(')'

- выделяют условные выражения в операторах *if* (условный оператор), *while*, *do-while*, *for* (операторы цикла);
- вводятся как обязательные элементы в определение, описание и вызов любой функции;
- группируют выражения, изменяя последовательность выполнения операций, например $sr = (a + b)/2$ и т. д.

Фигурные скобки '{ }'

- обозначают начало и конец блока операторов;
- используются в определении структур и классов;
- используются при инициализации массивов и структур.

Запятая ',' разделяет элементы списка.

Точка с запятой ';' завершает каждый оператор, каждое определение (кроме определения функции) и каждое описание. Любое допустимое выражение, за которым следует ';' , воспринимается как оператор.

Двоеточие ':' служит для отделения метки от помеченного ею оператора.

Многоточие '...' используется для обозначения переменного числа параметров в функции.

3.2. Ключевые слова. Идентификаторы

Ключевые (служебные) слова. Ключевые (служебные) слова — это такие лексемы, которые используются системой программирования для своих специальных целей.

Ключевыми словами могут быть:

- директивы препроцессора:
 - ***include*** — включить файл в данное место программы;
 - ***define*** — определить константу;
 - ***typedef*** — задать тип пользователя;
- операторы языка:
 - ***if*** — условный оператор;
 - ***switch*** — оператор выбора;
 - ***break*** — оператор прерывания;
 - ***continue*** — оператор продолжения;
 - ***do, while, for*** — операторы цикла;
 - ***return*** — оператор возврата;
 - ***goto*** — оператор перехода;
- служебные слова — ***else, case, default, extern, friend***;
- операции — ***new, delete, sizeof***;
- имена типов данных — ***char, int, float, double***;
- спецификаторы типов — ***struct, class, const, register, auto, enum, short, signed, unsigned, static, long***.

Идентификаторы (имена). Идентификаторы (имена) служат для обозначения объектов программы (переменных, констант, меток, функций и т. д.). Примеры идентификаторов: *S, P, K, primer1*.

Правила написания идентификаторов:

- идентификатор может состоять из букв латинского алфавита, цифр и знака подчеркивания (нельзя использовать пробелы, специальные символы, буквы русского алфавита);
- идентификатор начинается только с буквы или знака подчеркивания;
- длина идентификатора ничем не ограничена;
- в идентификаторах прописные и строчные латинские буквы воспринимаются компилятором как разные, например: *MAX, Max, max* — это три разных идентификатора;
- идентификатор не может совпадать ни с одним из служебных слов (например, *do, while*).

Например, *D24* — правильно!

15с, 7_s — неправильно! (ошибка — начинаются с цифры).

Kol_vo — правильно!

F(x) — неправильно! (ошибка — содержит скобки).

_7A — правильно, начинается с символа подчеркивания.

3.3. Константы и переменные

Константа. *Константа* — это лексема, представляющая изображение фиксированного числового, строкового или символьного значения, например: 100, 3.14159, «Здравствуй, мир!», 'Y'. *Константа* служит для обозначения какого-то определенного значения и в процессе работы программы не меняется. Например, 5.25, 100.

Константы делятся на группы:

- целые;
- вещественные;
- символьные;
- строковые.

Десятичная целая константа определена как последовательность десятичных цифр, начинающаяся не с нуля, если это не число нуль. Допустимый диапазон положительных целых значений от 0 до 4 294 967 295 (от 0 до $2^{32} - 1$ — беззнаковое целое). Константы, превышающие указанное значение, вызывают ошибку на этапе компиляции. Абсолютные значения отрицательных констант не должны превышать значения 2 147 483 648 (-2^{31} — минимальное значение знакового целого).

Вещественные константы — это либо числа с десятичной точкой, отделяющей целую часть от дробной, либо очень большие числа, например 0.57, 6E+24, -1.67e-23. В записи вещественных констант может опускаться или целая часть, или дробная, например 56. , .89 . При записи в программе больших или маленьких чисел сначала записывается мантисса числа (коэффициент перед степенью 10) , число 10 заменяется латинской буквой E, вслед за которой ставится показатель степени 10. Например, для числа $1,67 \cdot 10^{23}$ коэффициент перед 10 — 1.67 является мантиссой данного числа, степень 10 — 23 является порядком числа, и в языке C++ данное число записывается в виде 1.67E+23 (или 1.67e+23).

Диапазон допустимых значений для вещественных констант от 3.4E-4932 до 3.4E+4932 по абсолютной величине.

Символьная константа — это один символ, заключенный в апострофы, например 'Z', 'v', 'П'.

Чтобы задать в качестве символьной константы апостроф ' ', или обратный слэш ' \ ', необходимо в записи символьной константы перед этими символами поставить обратный слэш — '\', '\\'.

Для задания управляющих символов используются записи:

'\a' — звуковой сигнал;

'\b' — удаление предыдущего символа (backspace);

'\f' — переход на новую страницу;

'\n' — переход на новую строку;

'\r' — переход в начало строки;

'\t' — горизонтальная табуляция;

'\v' — вертикальная табуляция.

Строковая константа — это последовательность символов, заключенная в кавычки, например: «Здравствуй, мир!»; «Программа выполнена», «Строка номер 3».

В строковых константах можно использовать управляющие символы, например:

"Программа выполнена.\nДо свидания."

Переменные. Переменная служит для хранения какого-либо значения и *всегда* имеет собственное имя. В процессе работы программы значение переменной может меняться.

3.4. Понятие типа данных

Тип данных определяет допустимый диапазон изменения переменных и констант, а также допустимые операции над данными этого типа. Каждая переменная и константа программы должны принадлежать к определенному типу данных. Это необходимо для того, чтобы отвести необходимое место в оперативной памяти и для проверки правильности записи выражений. Например, над числовыми данными можно производить арифметические операции, а символьные данные используются для анализа текста.

3.5. Целые типы данных

Целые типы данных используются для представления целых чисел. Целые типы, допустимый диапазон значений и требуемая память в байтах приведены в табл. 3.3.

Таблица 3.3. Целые типы данных

Тип	Назначение типа	Размер в байтах	Значение
unsigned char	Символьный	1	Целые числа или символы с кодами от 0 до 255
char	Символьный	1	Целые числа или символы с кодами от -128 до 127
short int	Короткий целый	2	Целые числа от -32 768 до 32 767
int	Целый	4	Целые числа от -2 147 483 648 до 2 147 483 647
long int	Длинный целый		
unsigned int	Беззнаковый целый	4	Целые числа от 0 до 4 294 967 295

Целый тип **char** определяется двумя способами:

- это либо целые числа от -128 до 127;
- либо множество символов кодовой таблицы компьютера. Каждому символу ставится в соответствие целое число от 0 до $2^8 - 1$. Для кодировки используется код ASCII (*American Standard Code for Information Interchange* — американский стандартный код для обмена информацией). Фактически допустимыми значениями для символьного типа являются все символы клавиатуры компьютера, в том числе и управляющие символы (Esc, Tab и т. д.).

Значения символьного типа можно вводить с клавиатуры, выводить на экран и сравнивать между собой, при этом бóльшим считается символ с бóльшим ASCII-кодом. Для символов типа **char** определены операции отношения =, <>, <, >, >=, <= (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

Если данные типа **char** рассматриваются как целые числа, то для них определены арифметические операции +, -, *, /, % (результат выполнения этих операций также имеет тип **char**), операции отношения =, <>, <, >, >=, <= (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

Целый тип **short int** (короткое целое) задает целые данные с диапазоном изменения от -32 768 до 32 767 и занимает в памяти два байта. Для данных этого типа определены арифметические операции +, -, *, /, % (результат выполнения этих операций — целого типа **short int**), операции отношения =, <>, <, >, >=, <= (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

Наиболее часто используемый целый тип **int** задает целые данные с диапазоном изменения от $-2\,147\,483\,648$ до $2\,147\,483\,647$ и занимает в памяти четыре байта. Для данных типа **int** определены арифметические операции $+$, $-$, $*$, $/$, $\%$ (результат выполнения этих операций — целого типа), операции отношения $=$, $<>$, $<$, $>$, $>=$, $<=$ (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

*Целый тип **long int** (длинное целое)* задает целые данные с диапазоном изменения от $-2\,147\,483\,648$ до $2\,147\,483\,647$ и занимает в памяти четыре байта. Для данных этого типа определены арифметические операции $+$, $-$, $*$, $/$, $\%$ (результат выполнения этих операций — целого типа **long int**), операции отношения $=$, $<>$, $<$, $>$, $>=$, $<=$ (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

*Тип беззнаковое целое **unsigned int*** задает целые числа с диапазоном изменения от 0 до $4\,294\,967\,295$ и занимает четыре байта. Для данных этого типа определены арифметические операции $+$, $-$, $*$, $/$, $\%$ (результат выполнения этих операций — беззнакового целого типа **unsigned int**), операции отношения $=$, $<>$, $<$, $>$, $>=$, $<=$ (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

3.6. Вещественные типы данных

Вещественные типы используются для задания чисел с ненулевой дробной частью (например, 0.7, -18.567 , 3.14) или чисел, меньших $-2\,147\,483\,648$, или чисел, больших $2\,147\,483\,647$ ($4\,294\,967\,295$).

Вещественные типы, допустимый диапазон значений и требуемая память в байтах приведены в табл. 3.4.

*Вещественный тип **float*** задает вещественные числа с диапазоном изменения от $-3,4 \cdot 10^{38}$ до $3,4 \cdot 10^{38}$ для больших чисел и от $-3,4 \cdot 10^{-38}$ до $3,4 \cdot 10^{-38}$ и занимает в памяти четыре байта. Для данных этого типа определены арифметические операции $+$, $-$, $*$, $/$ (результат выполнения этих операций — вещественного типа **float**), операции отношения $=$, $<>$, $<$, $>$, $>=$, $<=$ (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

Наиболее часто используемый вещественный тип **double** задает вещественные числа в диапазоне от $-1,7 \cdot 10^{308}$ до $1,7 \cdot 10^{308}$ для боль-

Таблица 3.4. Вещественные типы данных

Тип	Назначение типа	Размер в байтах	Значение
float	Вещественный	4	Числа от $3.4E-38$ до $3.4E+38$ Точность представления — семь значащих цифр
double	Вещественный с двойной точностью	8	Числа от $1.7E-308$ до $1.7E+308$. Точность представления — 15 значащих цифр
long double	Вещественный с повышенной точностью	10	Числа от $3.4E-4932$ до $3.4E+4932$. Точность представления — 19 значащих цифр

ших чисел и от $-1,7 \cdot 10^{-308}$ до $1,7 \cdot 10^{-308}$ для маленьких чисел. Переменные типа **double** занимают в памяти восемь байт. Для данных типа **double** определены арифметические операции $+$, $-$, $*$, $/$ (результат выполнения этих операций — вещественного типа **double**), операции отношения $=$, $<>$, $<$, $>$, $>=$, $<=$ (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

Вещественный тип long double задает вещественные числа с диапазоном изменения от $-3,4 \cdot 10^{4932}$ до $3,4 \cdot 10^{4932}$ для больших чисел и от $-3,4 \cdot 10^{-4832}$ до $3,4 \cdot 10^{-4932}$ и занимает в памяти 10 байт. Для данных этого типа определены арифметические операции $+$, $-$, $*$, $/$ (результат выполнения этих операций — вещественного типа **long double**), операции отношения $=$, $<>$, $<$, $>$, $>=$, $<=$ (результат — логического типа) и стандартные функции, которые приведены в *приложении 1*.

3.7. Логический тип данных

Булевский (логический) тип данных — BOOL. Переменные булевского типа данных представляют собой логические значения, такие как **TRUE** (истина) и **FALSE** (ложь). При вычислении значений в арифметических выражениях значение, равное нулю, принимается за ложь, а любое другое значение, не равное нулю, будет являться истиной.

3.8. Операторы описания и определения переменных

Чтобы переменную можно было использовать в программе, она должна быть предварительно объявлена (описана).

Оператор объявления (описания) переменных имеет следующий синтаксис:

```
<тип> <имя1>, <имя2>, ...;
```

Здесь <тип> — тип переменных, имена которых перечислены в операторе описания;

<имя1>, <имя 2>, ... — имена переменных, необходимых в программе.

Например:

```
int   x,y,z;           //оператор объявления переменных x,y,z типа int
double a,b;           //оператор объявления переменных a,b типа double
float t;               //оператор объявления переменной t типа float
```

При описании переменных под каждую из них резервируется память. Размер отведенной памяти зависит от типа переменных. При этом никаких значений в зарезервированную память не заносится.

Оператор определения переменных имеет следующий синтаксис:

```
<тип> <имя1>=<нач.значение1>, <имя2>=<нач.значение2>, ...;
```

Здесь <тип> — тип переменных, имена которых перечислены в операторе описания;

<имя1>, <имя 2>, ... — имена переменных, необходимых в программе;

<нач.значение1>, <нач.значение2> — начальные значения, заносимые компилятором до выполнения программы.

При описании переменных под каждую из них резервируется память. Размер отведенной памяти зависит от типа переменных. После этого компилятор заносит начальное значение в область памяти соответствующей переменной. Например:

```
double s=0;           //переменной s задано начальное значение равное 0
```

В языке C++ объявлять и определять переменные можно в любом месте программы, однако область их действия начинается с момента объявления и заканчивается ближайшей за этой переменной закры-

вающейся фигурной скобкой. То есть область определения и действия переменной ограничена блоком, внутри которого она описана.

Именованные константы используются как переменные, значение которых не может быть изменено после инициализации. Объявляется с помощью ключевого слова `const`, за которым следует указание типа константы, идентификатор и значение:

```
const double pi=3.14159; //объявление именованной константы pi
```

3.9. Преобразование типов

В некоторых случаях необходимо преобразование типов, например, чтобы присвоить *целое* значение выражения *вещественной* переменной или для согласования типов при использовании аргументов функций.

Рассмотрим простые выражения — арифметические действия. Эти выражения состоят из двух операндов, над которыми осуществляются действия, и знака операции ('+', '-', '*', '/'). Например:

Операнд 1 + Операнд 2 — сложение.

Операнд 1 — Операнд 2 — вычитание.

*Операнд 1 * Операнд 2 — умножение.*

Операнд 1 / Операнд 2 — деление.

Существует три способа преобразования типов:

- неявное преобразование типов;
- явное преобразование типов;
- функциональное преобразование типов.

Неявное преобразование типов. Приведение результата к типу одного из операндов называется неявным преобразованием типов:

- если оба операнда одного типа, то и результат будет того же типа, например, в выражении $2/3$ оба операнда целого типа, значит, и результат будет целого типа, в этом случае он будет равен целому частному от деления 2 на 3, т. е. 0;
- если операнды имеют разный тип, то тип результата будет совпадать с более широким типом операнда, например, в выражении $2/3.0$ первый операнд — целого типа, второй — вещественного типа, значит, и результат будет вещественного типа, т. е. 0.6667.

Явное преобразование типов и функциональное преобразование типов. В языке C++ существует также операция явного преобразования типов

<тип> <выражение>

и операция функционального преобразования типов

(<тип>) <выражение>

Например, предыдущее выражение можно записать как `2/double 3` или `2/(double)3`.

Если применить ко всему выражению операцию явного преобразования типов `double (2/3)` или операцию функционального преобразования типов `(double)(2/3)`, то сначала будет вычислено выражение в скобках $(2/3)$, равное 0, а затем результат будет преобразован к типу `double` и станет равным 0.0.

3.10. Знаки операций

Знаки операций. Знаки операций — это лексемы, используемые при вычислении выражений. По числу операндов операции бывают:

- унарными (с одним операндом);
- бинарными (с двумя операндами).

Примером унарной операции может служить операция изменения знака `—A` (значение переменной `A` меняет знак на противоположный), операция определения адреса переменной `&A` — адрес переменной `A`. Операции сложения, вычитания, умножения, деления и т. д. являются бинарными.

В языке C++ используются следующие виды операций:

- арифметические операции;
- операции инкремента и декремента;
- операции присваивания;
- операции отношения;
- логические операции.

Арифметические операции. Знаки арифметических операций используются в арифметических или алгебраических выражениях:

- + сложение;
- вычитание;
- * умножение;
- / деление;
- % — нахождение остатка от деления нацело.

Таблица 3.5. Примеры операции нахождения остатка целочисленного деления

Выражение	Результат
11 % 5	1
10 % 3	1
2 % 3	2

Операции инкремента и декремента. *Инкремент* — это увеличение значения на 1, *декремент* — уменьшение значения на 1.

++ знак операции инкремента — увеличения операнда на единицу;

-- знак операции декремента — уменьшения операнда на единицу.

Операции *инкремента и декремента* бывают *постфиксными* и *префиксными*.

Постфиксные операции инкремента и декремента записываются *п о с л е* соответствующего операнда, например $x++$; $z--$;

При использовании постфиксных операций инкремента и декремента в операторе присваивания (или каком-либо другом операторе) используется старое значение операнда, и лишь после выполнения соответствующего оператора значение операнда изменяется.

Например, при выполнении операторов

```
X=1;
Y=X++;      (Y=1,      затем X=2)
```

переменная Y примет значение, равное старому значению переменной X (1), и лишь после выполнения операции присваивания значение X увеличится на 1 и станет равно 2.

При выполнении операторов

```
Z=5;
T=Z--;      (T=5,      затем Z=4)
```

переменная T примет значение, равное старому значению переменной Z (5), и лишь после выполнения операции присваивания значение Z уменьшится на 1 и станет равно 4.

Префиксные операции инкремента и декремента записываются *п е р е д* соответствующим операндом, например: $++x$; $--z$;

При использовании префиксных операций инкремента и декремента в каком-либо операторе сначала изменяется значение операнда, и это новое значение операнда используется в соответствующем операторе.

Например, при выполнении операторов

```
X=1;  
Y=++X;      (X=2,      затем Y=2)
```

сначала значение переменной *X* увеличится на 1 и станет равным 2, а затем это новое значение (2) будет присвоено переменной *Y*.

При выполнении операторов

```
Z=5;  
T=--Z;      (Z=4,      затем T=4)
```

сначала значение переменной *Z* уменьшится на 1 и станет равным 4, а затем это новое значение (4) будет присвоено переменной *T*.

Операции инкремента и декремента можно использовать и в отдельных операторах, например,

```
++a;      c++;  
--b;      d--;
```

здесь значения переменных *a*, *c* будут увеличены на единицу, а значения переменных *b*, *d* будут уменьшены на единицу, при этом ни старые, ни новые значения этих переменных ни в каких выражениях или операторах участвовать не будут.

Знаки операций присваивания. Существуют следующие знаки операций присваивания:

= — операция присваивания (значение, стоящее справа от знака операции, присваивается переменной, стоящей слева от знака присваивания, например *x=6*);

+= — присваивание со сложением, запись *x+=5*; идентична записи *x = x + 5*;

-= — присваивание с вычитанием, запись *x-=3*, идентична записи *x = x - 3*;

*= — присваивание с умножением, запись *x*=7*, идентична записи *x = x * 7*;

/= — присваивание с делением, запись *x/=4*, идентична записи *x = x / 4*;

%= — присваивание с нахождением остатка, запись *x%=2*, идентична записи *x = x % 2*;

Операции отношения. Знаки операций отношения используются в условных выражениях:

= = равно;

!= не равно;

> больше;
< меньше;
>= больше или равно;
<= меньше или равно;

Логические операции. Знаки логических операций используются в логических выражениях:

&& — логическое И (истинно тогда и только тогда, когда оба операнда истинны);

|| — логическое ИЛИ (ложно тогда и только тогда, когда оба операнда ложны);

! — логическое отрицание.

3.11. Оператор присваивания. Арифметические выражения. Приоритет операций

Оператор присваивания. Оператор присваивания используется для присваивания переменной значения какого-то выражения. Оператор присваивания имеет вид:

<имя переменной>=<выражение>;

где <выражение> — это выражение, записанное с использованием арифметических и/или логических операций и скобок, значение которого можно вычислить;

<имя переменной> — это переменная, в которую будет записано значение <выражения>.

Правила записи арифметических выражений:

- для получения правильного результата в нужных случаях необходимо применять операции явного или неявного преобразования типов;
- если в числителе выражения есть хотя бы один знак сложения или вычитания, то числитель берется в скобки;
- если знаменатель выражения содержит два и более члена, то, вне зависимости от знаков операций, знаменатель берется в скобки;
- аргументы функций нужно брать в скобки;
- число открывающихся скобок должно быть равно числу закрывающихся скобок;
- операции умножения в выражении записывать обязательно (математическая запись $2x$ в языке C++ заменяется на $2*x$).

Оператор присваивания выполняется следующим образом:

- вычисляется значение выражения, стоящего в правой части оператора присваивания;
- при необходимости осуществляется преобразование типа значения выражения к типу переменной, стоящей слева от знака присваивания;
- вычисленное и преобразованное значение присваивается переменной, имя которой стоит в левой части оператора присваивания, т. е. записывается в то место оперативной памяти, которое выделено для данной переменной.

Например:

```
x=5;                //переменной x присваивается значение, равное 5
y=x*x;              //переменной y присваивается значение x2
```

Приоритет операций. Приоритет операций — это порядок выполнения операций в выражении. Выполнение каждой операции происходит с учетом ее приоритета.

Приоритет операций в выражении без скобок:

- 1) *, /, %;
- 2) +, -;
- 3) ! — логическое отрицание;
- 4) && — логическое И;
- 5) || — логическое ИЛИ;
- 6) =, <>, <, >, >=, <=.

Операции, перечисленные под одной цифрой, имеют одинаковый приоритет, т. е. операции *, /, % имеют одинаковый приоритет первого наивысшего уровня, операции +, - имеют одинаковый приоритет второго уровня, операции =, <>, <, >, >=, <= имеют одинаковый приоритет третьего уровня.

В сложном выражении сначала выполняются вычисления в скобках, затем остальные операции в порядке убывания их приоритета (операции с равным приоритетом выполняются слева направо). Например, при вычислении значения выражения

$$12+14/(8+2*3/6-2)-4$$

сначала вычисляется значение выражения, стоящего в скобках, — $(8+2*3/6-2)$:

1) операции умножения и деления имеют наивысший приоритет; операция умножения стоит слева, поэтому она будет выполняться первой — выражение в скобках преобразуется к виду $(8+6/6-2)$;

2) в полученном выражении операция деления имеет наивысший приоритет и будет выполняться следующей — выражение в скобках преобразуется к виду $(8+1-2)$;

3) операции сложения и вычитания имеют одинаковый приоритет; операция сложения стоит слева, поэтому она будет выполняться следующей — выражение в скобках преобразуется к виду $(9-2)$;

4) выполняется операция вычитания — выражение в скобках преобразуется к виду (7) ;

5) после вычисления значения выражения, стоящего в скобках, общее выражение преобразуется к виду $12+14/7-4$. В полученном выражении операция деления имеет наивысший приоритет и будет выполняться следующей — общее выражение преобразуется к виду $12+2-4$;

6) операции сложения и вычитания имеют одинаковый приоритет; операция сложения стоит слева, поэтому она будет выполняться следующей — общее выражение преобразуется к виду $14-4$;

7) выполняется операция вычитания — общее выражение преобразуется к виду (10) , т. е. значение выражения $12+14/(8+2*3/6-2)-4=10$.

3.12. Структура программы на языке C++

Структура программы на языке C++. Под *структурой программы* понимается набор разделов программы и порядок их следования. В общем виде структура программы на языке C++ выглядит следующим образом:

- набор директив препроцессора, которые включают необходимые заголовочные файлы;
- описание глобальных констант;
- описания глобальных переменных;
- описание именованных констант;
- определения и описания функций пользователя;
- описание функции *main()*, главной функции программы;
- определение тех функций пользователя, описания которых находятся перед функцией *main()*.

В программе могут отсутствовать один или несколько разделов, однако набор директив препроцессора и функция *main()* являются обязательными.

Пример простой программы. Составить программу вычисления площади круга.

```
#include <stdio.h>      /*директива препроцессора для подключения
                        стандартной библиотеки ввода-вывода*/
const double Pi=3.14    /*определ. вещественной константы Pi
                        //равной 3.14
int main()              //заголовок функции main()
{
    double R,S;         //оператор описания вещественных переменных R и S
    printf("Введите радиус круга  ");    /*вывод приглашения
                                           к вводу значения радиуса*/
    scanf("%lf",&R);    //ввод с клавиатуры значения
                        //вещественной переменной R

    S=Pi*R*R;
        //оператор присваивания S значения площади круга радиуса R
    printf("S=%lf\n",S);    //вывод на экран значения S
    getchar();              //остановка работы программы до нажатия
                        //любого символа

    return 0;
}
```

Пояснения. Программа начинается с директивы препроцессора `#include <stdio.h>`, которая подключает к программе заголовочный файл стандартной библиотеки ввода-вывода. Затем с помощью оператора `const double pi=3.14;` объявляется вещественная константа `pi`, равная 3.14.

Далее идет заголовок главной функции программы `main`. Эта функция должна присутствовать в каждой программе. Сразу же после запуска программы на выполнение операционная система компьютера передает управление функции `main` программы, после завершения работы функция `main` возвращает управление операционной системе компьютера.

Заголовок `int main()` указывает на то, что функция `main` не получает никакой информации от операционной системы компьютера (пустые круглые скобки после имени функции `main`). Тип данных `int` перед именем функции `main` означает, что операционной системе будет возвращено целое значение (оператором `return 0;`).

Открывающаяся фигурная скобка открывает тело функции `main` — блок, соответствующий функции `main`. Затем с помощью оператора `double R,S;` описываются вещественные переменные программы. На экран выводится приглашение к вводу значения ра-

диуса, с клавиатуры вводится значение радиуса. В операторе присваивания вычисляется площадь круга, которая затем выводится на экран.

3.13. Форматированный ввод и вывод данных

В ядре языка C++ отсутствуют средства (функции) осуществления ввода и вывода данных. Функции ввода-вывода данных содержатся в нескольких библиотеках. Рассмотрим одну из них:

stdio — *STandarD Input Output library* — стандартная библиотека ввода-вывода, содержит функции ввода с клавиатуры *scanf()* и вывода на экран дисплея *printf()*. Для того чтобы в программе можно было пользоваться этими функциями, нужно вначале подключить эту библиотеку. Директива препроцессора для включения в текст программы заголовочного файла библиотеки ***stdio*** имеет следующий вид:

```
#include <stdio.h>
```

Функция ввода *scanf()*: функция ввода *scanf()* имеет следующий синтаксис:

```
scanf(<форматное выражение>, <список ввода>);
```

scanf — ключевое (зарезервированное) слово;

<форматное выражение> содержит форматные спецификации, которые показывают, каким образом интерпретировать (преобразовывать) данные, вводимые с клавиатуры.

Форматные спецификации начинаются с символа '%' (процент), зависят от типа вводимых данных и имеют следующий вид:

Форматная спецификация	Назначение
%d	Форматная спецификация для ввода чисел типа <i>int</i> и <i>char</i>
%i	Форматная спецификация для ввода чисел типа <i>int</i> и <i>char</i>
%ld	Форматная спецификация для ввода чисел типа <i>long int</i>
%f	Форматная спецификация для ввода чисел типа <i>float</i> в форме с фиксированной точкой
%lf	Форматная спецификация для ввода чисел типа <i>double</i> в форме с фиксированной точкой

Окончание форматных спецификаций

Форматная спецификация	Назначение
<code>%e</code>	Форматная спецификация для ввода чисел типа <i>float</i> в экспоненциальной форме
<code>%c</code>	Форматная спецификация для ввода символов
<code>%s</code>	Форматная спецификация для ввода строк

<СПИСОК ВВОДА> содержит адреса переменных, значения которых вводятся с клавиатуры. Например:

```
int x;
double a;
.....
scanf("%d%lf",&x,&a);
```

Оператор `scanf("%d%lf",&x,&a)` служит для ввода с клавиатуры значений переменных `x`, `a`. Его действие можно расшифровать следующим образом:

- ввести с клавиатуры первое значение по форматной спецификации `%d` (интерпретировать его как число типа `int` или `char`);
- записать введенное значение по адресу переменной `x`;
- затем ввести с клавиатуры второе значение по форматной спецификации `%lf` (интерпретировать его как число типа `double` в форме с фиксированной точкой);
- записать введенное значение по адресу переменной `a`.

При вводе с помощью таких операторов ввода сами вводимые числовые значения можно отделять одним или несколькими пробелами или размещать на разных строках, например, ввод целых переменных `x` и `y` можно осуществить следующими способами:

1-й способ:

```
4    5.7 <Enter>
```

2-й способ:

```
4 <Enter>
5.7 <Enter>
```

Функция вывода `printf()`. Функция вывода `printf()` имеет следующий синтаксис:

```
printf(<форматное выражение>,<список вывода>);
```

<форматное выражение> содержит форматные спецификации, аналогичные форматным спецификациям оператора `scanf()`, а также строковые константы.

<СПИСОК ВЫВОДА> — это список идентификаторов переменных или констант, значения которых нужно вывести на экран дисплея. Например:

```
printf("значение переменной x =%d\n",x);
```

На экране появится текст:

```
значение переменной x =4
```

Форматы вывода. Формат вывода предназначен для корректного вывода результатов работы программы и используется *только при выводе значений числовых переменных*. Формат вывода указывается в операторе сразу после символа «%» и перед соответствующей форматной спецификацией.

1. Для *вещественных* чисел используется формат с фиксированной точкой (*использовать обязательно*, иначе по умолчанию вывод будет в формате с плавающей точкой.

Например, 1.230000000000000000000000E+00 вместо 1.23). Для вещественных переменных формат вывода имеет вид:

```
%<число1>.<число2>lf
```

или

```
%<число1>.<число2>f;;
```

где <число1> указывает *общее количество позиций* для вывода всего числа вместе с десятичной точкой и знаком;

<число2> указывает количество позиций для вывода дробной части числа;

lf — форматная спецификация для вывода данных типа `double`;

f — форматная спецификация для вывода данных типа `float`;

% — признак форматной спецификации.

Таблица 3.6. Примеры форматов вывода вещественной переменной

Значение переменной	Формат вывода	Результат (на экране)
511.04	<code>printf("%8.4lf",R);</code>	511.0400
-46.78	<code>printf("%6.2f",R);</code>	-46.78
-46.78	<code>printf("%9.4lf",R);</code>	<пробел>-46.7800

Если для вывода числа отводится позиций больше, чем нужно, то перед числом ставится столько пробелов, сколько необходимо для указанного формата. Например, один пробел перед числом в последнем примере.

2. Для *целых* чисел (использовать необязательно — по умолчанию выделяется столько позиций, сколько необходимо; в основном используется для вывода табличных результатов). Для целочисленных переменных формат вывода имеет вид:

```
%<число1>d;
```

где <число1> указывает *общее количество позиций* для вывода всего числа вместе со знаком;

d — форматная спецификация для вывода целых чисел;

% — признак форматной спецификации.

Таблица 3.7. Примеры форматов вывода целочисленной переменной

Значение переменной	Формат вывода	Результат (на экране)
85	printf("%3d",A);	<пробел>85
85	printf("%2d",A);	85

3.14. Особенности ввода и вывода символов и строк

Символы (переменные и константы типа `char`) в компьютере хранятся в виде кодов, т. е. последовательностей нулей и единиц, которые интерпретируются как целые числа в диапазоне от -128 до 127 (в соответствии с таблицей *ASCII*). В программе переменные типа `char` можно вводить, выводить и использовать либо как символы, либо как код (целое число) в зависимости от желаний программиста.

Ввод одного символа с клавиатуры можно осуществить с помощью функции `getchar`, которая считывает очередной символ с клавиатуры (из стандартного входного потока `stdin`, связанного с клавиатурой), при этом вводимый символ на экране не отображается. Чтобы ввести два или более символов, необходимо несколько раз вызвать функцию `getchar()`, в этом случае символы вводятся один за другим без каких-либо разделителей. Например,

```
char smv1, smv2;
.....
smv1=getchar();
smv2=getchar();
```


При вводе с помощью этих операторов значения вводимых символов не разделяются никакими разделителями, например, при вводе следующей последовательности символов

```
ab<Enter>
```

переменная `smv1` получит значение `a`, `smv2` — значение `b`.

Один или несколько символов также можно ввести с помощью функции `scanf()` с форматной спецификацией `%c`:

```
char c1,c2,c3;
.....
scanf("%c%c%c",&c1,&c2,&c3);
```

При вводе с помощью этого оператора значения вводимых символов тоже не разделяются никакими разделителями, например, при вводе следующей последовательности символов

```
qwe<Enter>
```

переменная `c1` получит значение `q`, `c2` — значение `w`, `c3` — значение `e`.

Если при вводе символы нужно разделить пробелом или запятой, то эти разделители ставятся между форматными спецификациями ввода:

```
scanf("%c %c %c",&c1,&c2,&c3); ( q w e<Enter> - строка ввода)
scanf("%c,%c,%c",&c1,&c2,&c3); (q,w,e<Enter> - строка ввода)
```

При вводе числа и символа с помощью одного оператора `scanf()` в форматном выражении обязательно нужно указывать разделитель между форматными спецификациями:

```
scanf("%c %lf",&c1,&b); (q -1.89<Enter> - строка ввода)
scanf("%lf %c",&b,&c1); (-1.89 q<Enter> - строка ввода)
```

Если нужно сначала с помощью оператора ввода ввести число, а затем с помощью другого оператора ввода ввести символ, то перед вводом символа из стандартного потока нужно сначала убрать коды клавиши *Enter* с помощью оператора:

```
c=getchar(); или scanf("\n");
```

При выводе символов можно вывести сам символ по форматной спецификации `%c` и (или) его код по форматной спецификации `%d`:

```
printf("Код символа %c равен %d\n",c1,c1);
```

Здесь сначала значение переменной `c1` будет выведено в виде символа (по форматной спецификации `%c`), а затем значение переменной `c1` будет выведено по форматной спецификации `%d` как целое число — код символа, являющегося значением переменной `c1`.

Пример. Ввести вещественное число и символ. Найти среднее арифметическое кода символа и числа.

```
#include <stdio.h>           //подключение стандартной библиотеки
                               //ввода-вывода
int  main()                  //заголовок функции main()
{
double x,sr;                 //описание вещественных переменных x и sr
char smb;                   //описание символьной переменной smb
printf("Введите число и символ\n");    /*вывод на экран
                                       приглашения к вводу исходных данных*/
scanf("%lf  %c",&x,&smb)    //ввод с клавиатуры через пробел числа
                               //и символа
sr=(x+smb)/2;               //вычисление среднего арифметического кода символа
                               //и числа
printf("sr=%lf\n",sr);      /*вывод на экран значения переменной sr
                               и перевод курсора на начало следующей строки экрана*/
getchar(); //остановка работы программы до нажатия любого символа
return 0;
}
```

Тесты

1. Выберите пример правильного идентификатора в языке C++:

- a) Fr_5;
- б) 10Sd;
- в) scanf.

2. К какому типу данных относится число –543.12?

- a) double;
- б) int;
- в) char.

3. К какому типу данных относится число –1000?

- a) double;
- б) int;
- в) char.

4. Правильна ли структура следующей программы?

```
#include <stdio.h>
{int k;
  double m;
  k=5;
  m=-2;
  int main()
  return 0;
}
```

- a) да;
- б) нет.

5. $A = 12 + 14 / (8 + 2 * 3 / 6 - 2) * 4$; Чему равно A?

- a) 10/7;
- б) 14;
- в) 20.

6. Каково значение выражения: $31 / 6$?

- a) 5.01;
- б) 1;
- в) 5.

7. Каково значение выражения: $31 \% 6$?

- a) 5.01;
- б) 1;
- в) 5.

8. Что будет на экране после выполнения следующих операторов?

```
printf("Пусть \n"); printf("всегда \n");
printf("будет \n"); printf("солнце.\n");
```

- a) Пусть всегда будет солнце;
- б) Пусть
всегда
будет
солнце;

Для каждой задачи составить схему алгоритма и написать программу.

7. Ввести целые числа A и B , результаты их суммирования, вычитания и произведения вывести в таблице, используя форматы вывода. Например, введены числа «6» и «2», тогда таблица будет иметь вид:

A	B	A+B	A-B	A*B
6	2	8	4	12

10. Ввести с клавиатуры значения A , B , C , D , организовать вычисление и вывод его по действиям, нумеруя каждое из них: $F = A - B + C - (A + A + B - (C + A)) + D$.

12. Составить программу исследования положительного вещественного числа A , в которой определялись бы значения следующих величин: целая часть, дробная часть, значение арифметического квадратного корня, остаток от деления целой части на 5.

14. Поменять местами значения переменных X и Y .

Контрольные вопросы

1. Что такое алфавит языка программирования?
2. Какие символы входят в алфавит языка C++?
3. Что такое зарезервированные (ключевые слова)? Приведите примеры ключевых слов.
4. Что такое идентификатор? Для чего используются идентификаторы в программах?
5. Может ли идентификатор содержать русские буквы?
6. Может ли идентификатор состоять только из латинских букв и символов подчеркивания?
7. С каких символов может начинаться идентификатор?
8. Может ли зарезервированное слово использоваться как идентификатор в программе? Почему?
9. Можно ли использовать в программе несколько одинаковых идентификаторов? Почему?
10. Что такое константа? Для чего можно использовать константы в программе?
11. Что такое переменная? Для чего в программе используются переменные?
12. Верно ли, что в любой программе на C++ обязательно должны использоваться переменные и константы? Почему?
13. Назовите основные отличия константы от переменной?
14. Можно ли в программе использовать две переменные с одинаковыми именами? Почему?
15. Могут ли в одной и той же программе переменная и константа иметь одинаковые имена? Почему?
16. Может ли изменяться значение переменной при выполнении программы? Почему?
17. Может ли в процессе работы программы изменяться значение константы? Почему?
18. Что такое операция? Приведите примеры.
19. Какие типы операций существуют в языке C++?
20. Назовите арифметические операции, используемые в языке C++.
21. Чем отличаются операции деления «3/2» и 3.0/2.0?
22. Чем отличаются операции «/» и «%»? Приведите примеры.
23. Какие операции отношения можно использовать в программе на языке C++?

24. Что такое выражение? Приведите примеры.
25. Что такое приоритет операций? Объясните на примере.
26. Какие операции выполняются в первую очередь, если в выражении есть скобки?
27. Какие операции имеют самый высокий приоритет, если в выражении нет скобок?
28. Какие операции имеют самый низкий приоритет?
29. Какие из операций отношения имеют более высокий приоритет?
30. Какая из операций имеет более высокий приоритет: умножение или деление?
31. Какая из операций имеет более высокий приоритет: сложение или вычитание?
32. Какая из операций имеет более высокий приоритет: умножение или вычитание?
33. Какие операции имеют более высокий приоритет: арифметические или операции отношения?
34. Что такое тип данных?
35. Какие типы данных в языке C++ вы знаете?
36. Что такое стандартный тип данных? Приведите примеры.
37. Для чего используются типы данных?
38. Приведите примеры данных типа `int`. Какие операции применимы к данным этого типа?
39. Приведите примеры данных типа `double`. Какие операции применимы к данным этого типа?
40. Какие данные относятся к символьному типу? Приведите примеры.
41. Для чего можно использовать данные строкового типа в программе?
42. Какие операции применимы в C++ к символьным и к строковым данным?
43. Из каких разделов состоит программа на языке C++?
44. Какие разделы и ключевые слова должны присутствовать в самой простой программе на языке C++?
45. Какие разделы не являются обязательными для программы на языке C++?
46. Что такое оператор в языке C++?
47. Для чего предназначены функции вывода? Назовите их.

Глава 4

ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЕННЫХ АЛГОРИТМОВ

В программах с разветвленной структурой используются условные операторы, которые предназначены для выбора к исполнению одного из нескольких возможных действий (операторов) в зависимости от некоторого условия (при этом одно из действий может отсутствовать — пустой оператор).

4.1. Условный оператор

Условный оператор. *Условный оператор* используется, если необходимо выполнять действия в зависимости от выполнения или невыполнения какого-то условия.

Краткая форма оператора *if*. Краткая форма условного оператора записывается следующим образом:

```
if <условие>  
  <оператор>;
```

где <условие> — это выражение логического типа, про которое можно сказать, что оно «истина» или «ложь» (фактически, это вопрос, предполагающий только 2 варианта ответа — «Да» или «Нет»). В зависимости от <условия> (от ответа на вопрос) происходит дальнейшее выполнение программы.

Описание работы краткой формы оператора *if*. Оператор **if** в краткой форме работает следующим образом:

- вычисляется выражение, записанное в условии (формулируется ответ на вопрос, записанный в условии);
- если получили результат — «истина» (не равно 0 — ответ «Да»), то выполняется <оператор>;

- если «ложь» (равно 0 — ответ «Нет»), то выполняется следующая за условным оператором строка программы.

На схеме алгоритма краткая форма **if** представлена следующим образом (рис. 4.1).

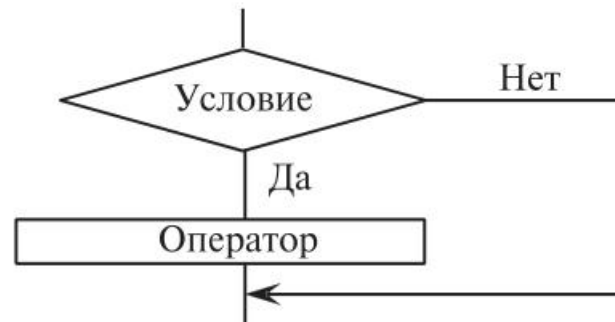
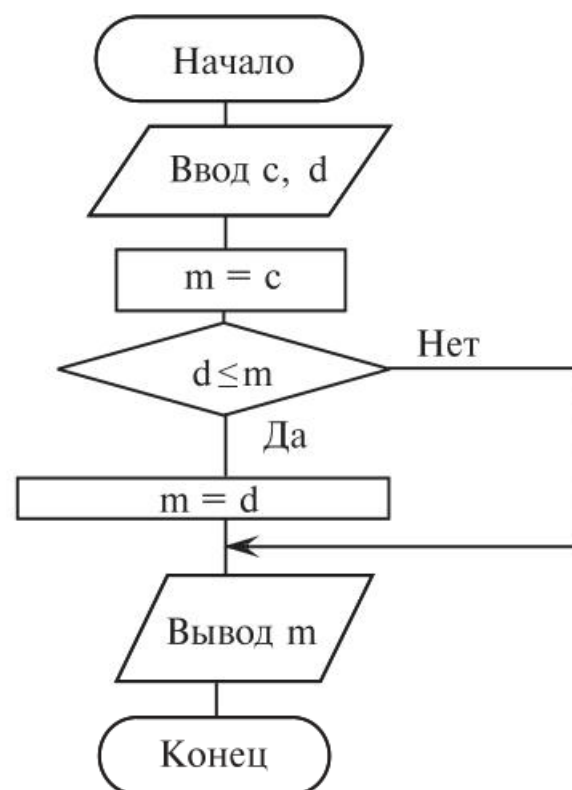


Рис. 4.1. Схема алгоритма краткой формы условного оператора

Пример программы с краткой формой оператора if. Вычислить $m = \min(c, d)$.

1. *Схема алгоритма*



Пояснение. Входными данными являются переменные c и d . После ввода входных данных (переменных c и d) переменной m (предполагаемый минимум) присваивается значение переменной c . Затем проверяется условие $d < m$. Если результат «истина» (не равно 0 — ответ «Да»), то переменной m присваивается значение d . Если результат «ложь» (ответ «Нет»), то никаких действий не выполняется и в обоих случаях переменная m становится минимумом из c и d . После этого на экран выводится результат (значение m).

2. Программа

```
#include <stdio.h>           //подключ. стандартной библиотеки
                               //ввода-вывода
int main() {                  //заголовок функции main()
int c, d, m;                  //описание переменных c, d, m типа int
printf("Введите через пробел c и d\n "); /*вывод на экран
                                           приглашения к вводу c и d и перевод курсора
                                           на начало следующей строки экрана*/
scanf("%d%d", &c, &d);       //ввод значений переменных c и d
m=c;                          //оператор присваивания переменной m значения переменной c
if (d<m)                       //если d<m
m=d;                          //то присваиваем переменной m значение переменной d
printf("m=%d\n", m);          //вывод m и перевод курсора на начало следующей строки экрана
getchar();                   //остановка работы программы до нажатия любой клавиши
return 0;
}
```

Полная форма оператора if. Полная форма условного оператора записывается следующим образом:

```
if (<условие>)
    <оператор1>;
else <оператор2>;
if и else — ключевые слова;
```

<условие> — это выражение логического или арифметического типа, про которое можно сказать, что оно «истина» (арифметическое выражение не равно нулю) или «ложь» (арифметическое выражение равно нулю) — фактически, это вопрос, предполагающий только 2 варианта ответа — «Да» или «Нет». В зависимости от <условия> происходит дальнейшее выполнение программы.

<оператор1> — оператор, который выполняется, если условие истинно (не равно нулю);

<оператор 2> — оператор, который выполняется, если условие ложно (равно нулю).

Условный оператор выполняется следующим образом:

- сначала проверяется <условие>;
- если <условие> истинно (значение выражения, представляющего собой условие, не равно 0), то выполняется <оператор1>;
- если условие ложно (значение выражения, представляющего собой условие, равно 0), то выполняется <оператор2>.

На схеме алгоритма полная форма **if** представлена следующим образом (рис. 4.2).

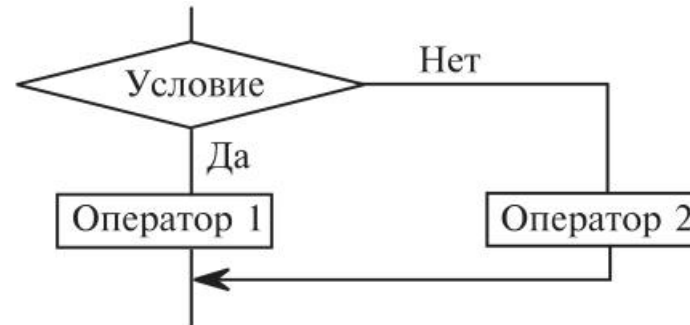
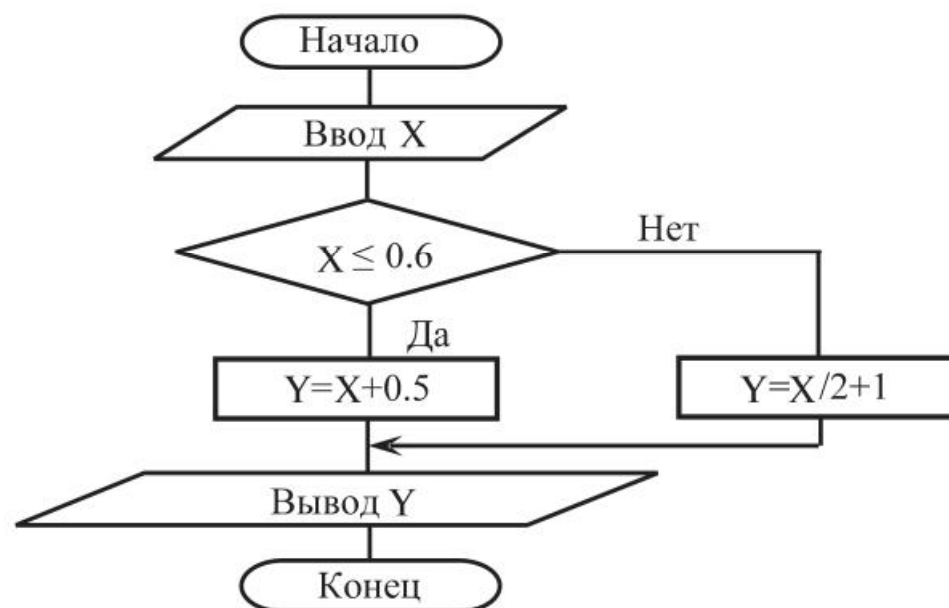


Рис. 4.2. Схема алгоритма полной формы условного оператора

Задача 2. Дано x . Вычислить $y = \begin{cases} x + 0,5, & \text{если } x \leq 0,6; \\ \frac{x}{2} + 1, & \text{если } x > 0,6. \end{cases}$

1. Схема алгоритма



Пояснение. Входным данным является переменная X . Перед вычислением Y проверяется условие « $X \leq 0,6$ ». Если результат «истина» (ответ «Да»), то происходит вычисление значения Y по формуле $Y = X + 0,5$. Если результат «ложь» (ответ «Нет»), то значение Y вычисляется по формуле $Y = X/2 + 1$. После вычисления Y (по одной из формул), на экран выводится результат (значение Y).

2. Программа

```

#include <stdio.h>                                //подключ. стандарт. библиотеки
                                                    //ввода-вывода

int main() {                                       //заголовок функции main()
double x, y;                                     //описание переменных x, y типа double

```

```

printf("Vvedite  x  ");
//вывод на экран приглашения к вводу переменной x
scanf("%lf",&x); //ввод значения переменной x
if ( x <= 0.6) //если x<=0.6
y=x+0.5; //то вычисляем y=x+0.5
else y= x/2+1; //иначе вычисляем y=x/2+1
printf("y=%lf\n",y);
//вывод y и перевод курсора на начало следующей строки экрана
getchar();
//остановка работы программы до нажатия любой клавиши
return 0;
}

```

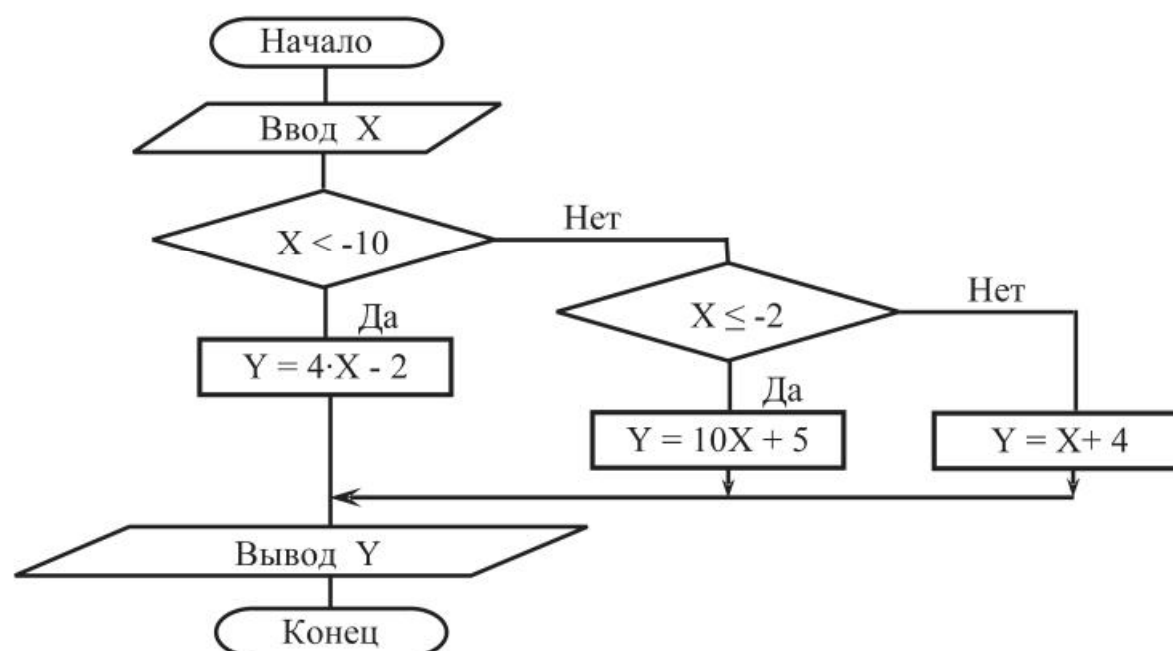
Пояснение. Так как y вычисляется по одной из двух формул в зависимости от значения x , то в программе используется условный оператор `if`. После вычисления y по одной из формул значение y выводится на экран по форматной спецификации `%lf`, так как y имеет тип `double`.

В качестве одного из операторов, «встроенных» в условный оператор, можно также использовать новый условный оператор.

Задача 3. Вычислить значение y :

$$y = \begin{cases} 4x - 2, & \text{если } x < -10; \\ 10x + 5, & \text{если } -10 \leq x \leq -2; \\ x + 4, & \text{если } x > -2. \end{cases}$$

1. Схема алгоритма



Пояснение. Входным данным является X . Перед вычислением y проверяется условие « $X < -10$ ». Если результат «истина» (ответ «Да»),

то происходит вычисление значения Y по формуле $Y = 4 \cdot X - 2$ ($X < -10$). Если результат «ложь» (ответ «Нет»), то проверяется условие « $X \leq -2$ ». Если это условие выполняется (ответ «Да»), то Y вычисляется по формуле $Y = 10X + 5$ ($X \leq -2$). Если это условие не выполняется (ответ «Нет»), то Y вычисляется по формуле $Y = X + 4$ ($X > -2$). После вычисления Y (по одной из формул) на экран выводится результат (значение Y).

2. Программа

```
#include <stdio.h>                                //подключ. стандарт. библиотеки
                                                    //ввода-вывода

int main() {                                       //заголовок функции main()
double x, y;                                     //описание переменных x, y типа double
printf("Введите x ");
                                                    //вывод на экран приглашения к вводу переменной x
scanf("%lf", &x);                                //ввод значения переменной x
if ( x < -10)                                    //если x < -10
y=4*x+2;                                         //то вычисляем y=4*x+2
else if (x <= -2)                                //иначе если x <= -2
    y=10*x+5;                                   //то вычисляем y=10*x+5
    y=x+4;                                       //иначе вычисляем y = x+4
printf("y=%lf\n", y);
                                                    //вывод y и перевод курсора на начало следующей строки экрана
getchar();                                       //остановка работы программы до нажатия любой
клавии
return 0;
}
```

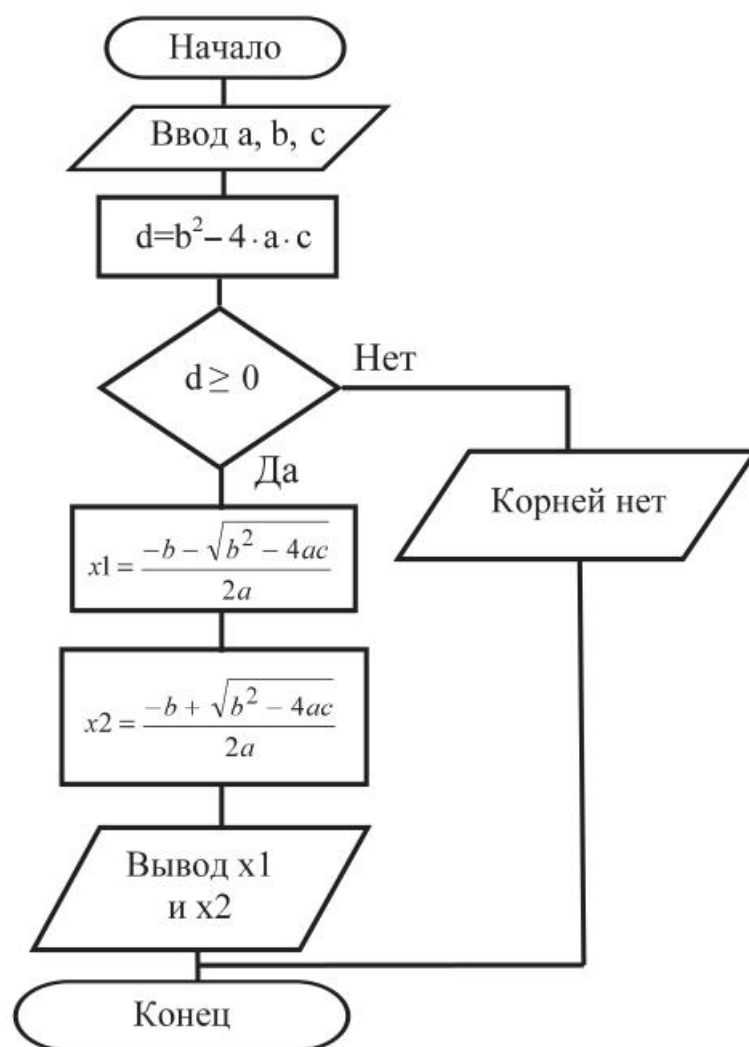
Пояснение. Так как y вычисляется по одной из трех формул в зависимости от значения x , то в программе используются два условных оператора **if** (с вложением). После ввода значения x с клавиатуры выполняется проверка условия « $x < -10$ » (1-й оператор **if**). Если оно истинно, то выполняется $y = 4 \cdot x + 2$. Если ложно, то проверяется следующее условие « $x \leq -2$ » (2-й оператор **if** расположен внутри ветви **else** 1-го оператора **if**). Если условие « $x \leq -2$ » истинно, то выполняется $y = 10 \cdot x + 5$. Если условие « $x \leq -2$ » ложно, то выполняется $y = x + 4$. После вычисления y по одной из формул его значение выводится на экран по форматной спецификации `%lf`, так как y имеет тип `double`.

Несколько операторов (блоки операторов) в операторе if. Если по одной или обоим ветвям условного оператора необходимо использовать несколько операторов, то они объединяются в блок — перед пер-

вым ставится открывающаяся фигурная скобка, после последнего — закрывающаяся фигурная скобка

Пример 1. Даны a, b, c — коэффициенты квадратного уравнения $ax^2 + bx + c = 0$. Найти корни этого уравнения или вывести сообщение «корней нет», считая, что $a \neq 0$.

1. Схема алгоритма



Пояснение. Входными данными являются коэффициенты квадратного уравнения — переменные a, b, c . Сначала вычисляется дискриминант квадратного уравнения $d = b^2 - 4 \cdot a \cdot c$. Если дискриминант больше или равен нулю ($d \geq 0$), то вычисляются два корня квадратного уравнения по следующим формулам:

$$x1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}; \quad x2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

Затем на экран выводятся результаты (значения переменных $x1$ и $x2$).

Если дискриминант меньше нуля ($d < 0$), на экран выводится сообщение «Корней нет».

2. Программа

```
#include <stdio.h>           //подключение стандартной библиотеки ввода-вывода
#include <math.h>             //подключение библиотеки математических
                              //функций
int main() {                 //заголовок функции main()
double a,b,c,x1,x2,d;        //описание переменных
printf("Введите через пробел a,b,c\n");
                             //вывод приглашения к вводу значений переменных a,b,c
scanf("%lf%lf%lf",&a,&b,&c); //ввод значений переменных a,b,c
d=b*b-4*a*c;                 //вычисление дискриминанта (d)
if (d>=0)                     //если d>=0
{
x1=(-b-sqrt(d))/(2*a);        //то вычисляем корни квадратного уравнения x1 и x2
x2=(-b+sqrt(d))/(2*a);
printf("x1=%lf x2=%lf\n",x1,x2); //и выводим их на экран
}
else                           //если d<0
printf("Корней нет\n");        //выводим текст «Корней нет»
getchar();
return 0;
}
```

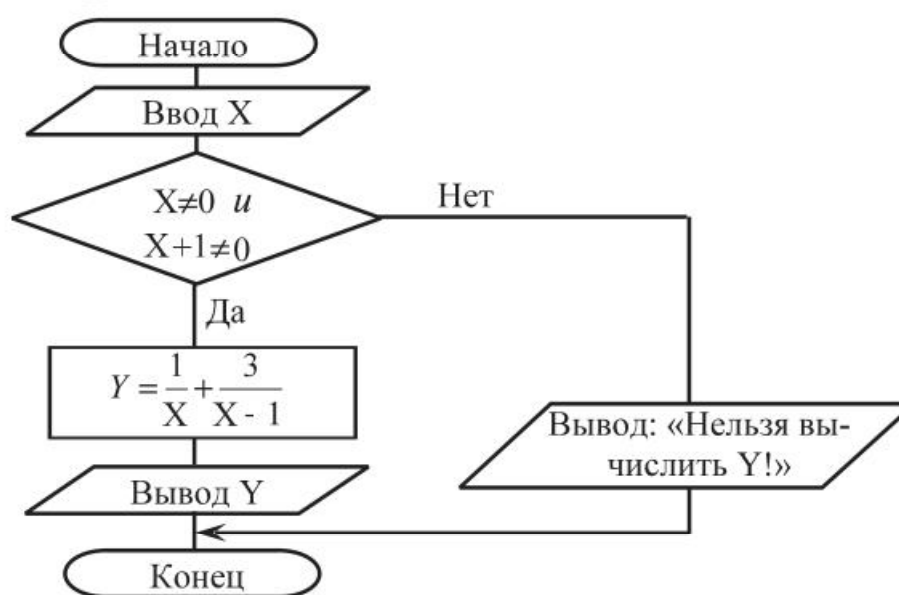
Пример 2. Вычислить значение $y = \frac{1}{x} + \frac{3}{x+1}$. Если y вычислить

нельзя, то вывести на экран сообщение об этом. Значение x вводится с клавиатуры.

Данную задачу можно решить двумя способами: а) используя логическое **И** (&&); б) используя логическое **ИЛИ** (||).

1-й способ (использование логического **И**).

1. Схема алгоритма



Пояснение. Входным данным является X . На нуль делить нельзя, поэтому знаменатели дробей не должны быть равны нулю (в условии задачи даны две дроби: $1/X$ и $3/(X+1)$). Таким образом, для вычисления Y надо проверить, что знаменатели обеих дробей одновременно не равны нулю. Для этого проверяется сложное условие « $X \neq 0$ и $X+1 \neq 0$ ». Если оно истинно (ответ «Да» на оба вопроса), то происходит вычисление значения Y и вывод его на экран. Если условие ложно (ответ «Нет» хотя бы на один из вопросов), то выводится сообщение «Нельзя вычислить $Y!$ ».

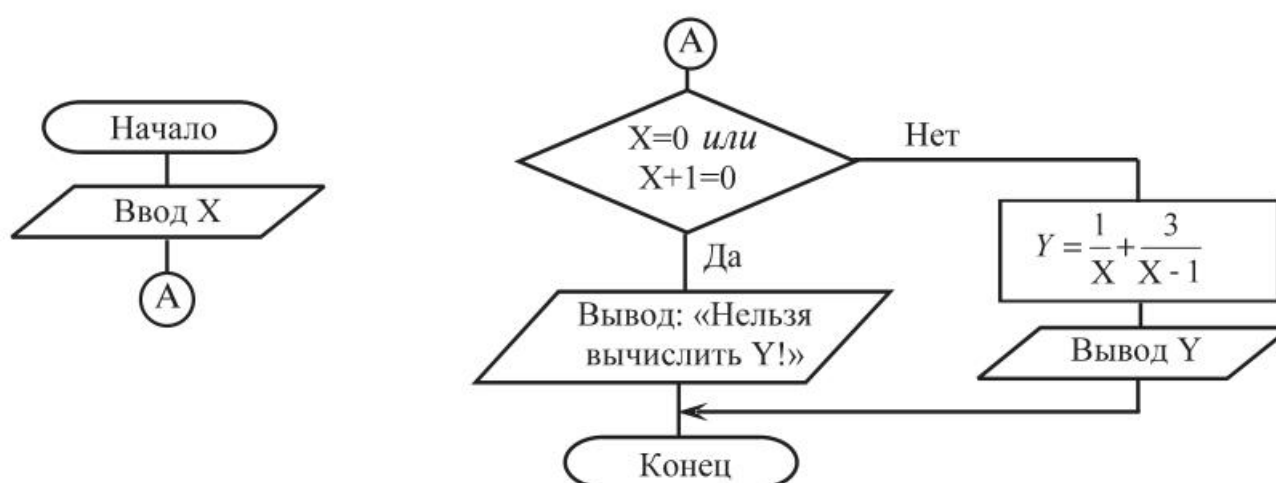
2. Программа

```
#include <stdio.h>
int main() {
    double x,y;
    printf("Введите x ");
    if ((x!=0)&&(x+1!=0)) {           //если одновременно x≠0 и x+1≠0
        y=1/x+3/(x+1);               //вычисляем y
        printf("y=5lf\n",y);         //и выводим y на экран
    }
    else printf("Нельзя вычислить y!\n");
    getchar();
    return 0;
}
```

Пояснение. Для вычисления y необходимо, чтобы одновременно знаменатели обеих дробей были не равны нулю. Для этого используется операция **&&** (логическое И). В операторе **if** проверяется сложное условие « $x \neq 0$ и $x+1 \neq 0$ » (в C++ оно записывается в виде $((x!=0) \&\& ((x+1) != 0))$, причем каждое простое условие заключается в отдельные круглые скобки).

2-й способ (использование ИЛИ).

1. Схема алгоритма



Пояснение. При решении задачи с помощью «логического ИЛИ» проверяется невозможность вычисления Y . Для этого надо проверить, что хотя бы один из знаменателей дробей равен нулю. Для этого проверяется сложное условие « $X = 0$ или $X + 1 = 0$ ». Если оно истинно (ответ «Да» хотя бы на один из вопросов), то выводится сообщение «Нельзя вычислить Y !». Если условие ложно (ответ «Нет» на оба вопроса), то происходит вычисление значения Y и вывод его на экран.

2. Программа

```
#include <stdio.h>
int main() {
    double x,y;
    printf("Введите x ");
    scanf("%lf",&x);
    if ((x==0) || (x+1==0))
        printf("Нельзя вычислить y!\n");
    else {y=1/x+3/(x+1);
        printf("y=%lf\n",y);
    }
    getchar();
    return 0;
}
```

Пояснение. В программе используется операция `||` (логическое ИЛИ). В операторе `if` проверяется сложное условие « $x=0$ или $x+1=0$ » (в C++ оно записывается в виде `((x==0) || ((x+1)==0))`).

4.2. Условная операция

Условная операция записывается следующим образом:

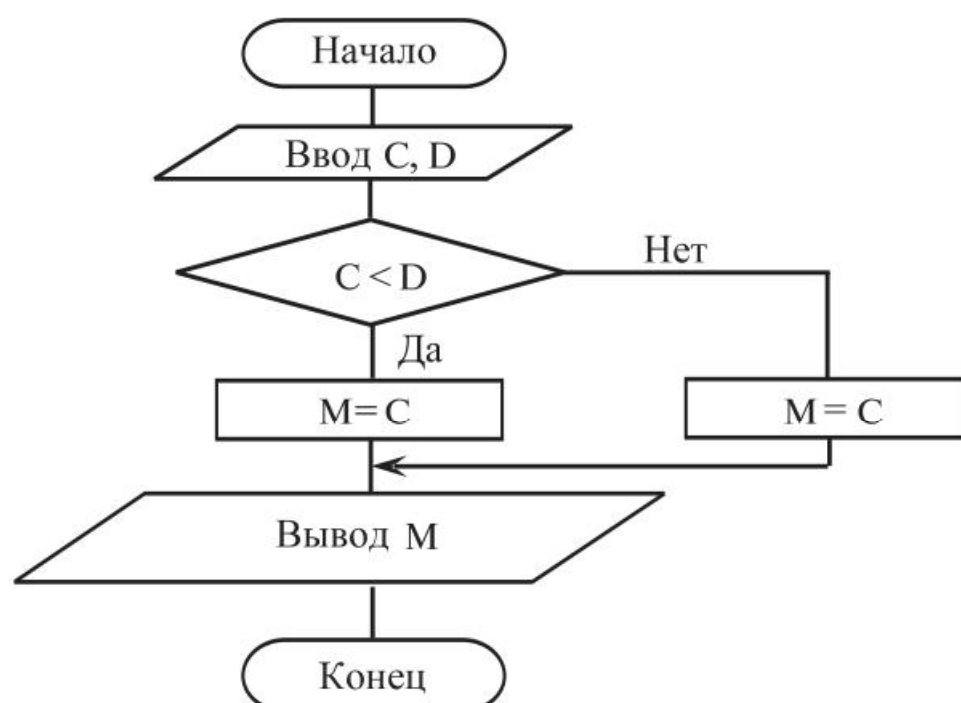
`<выражение1>?<выражение2>:<выражение3>`

Значение условной операции определяется следующим образом:

- сначала вычисляется `<выражение1>`;
- если `<выражение1>` истинно (не равно 0), то вычисляется `<выражение2>`, значение которого становится результатом условной операции;
- если `<выражение1>` ложно (равно 0), то в качестве результата берется значение `<выражение3>`.

Пример 1. С помощью условной операции вычислить $m = \min(c, d)$.

1. Схема алгоритма



Пояснение. Входными данными являются числа C и D . Результат M равен наименьшему (минимальному) из этих чисел. Для этого C сравнивается с D . Если условие « $C < D$ » выполняется (не равно 0), то C — наименьшее и $M = C$. Если условие « $C < D$ » не выполняется (равно 0), то D — наименьшее и $M = D$.

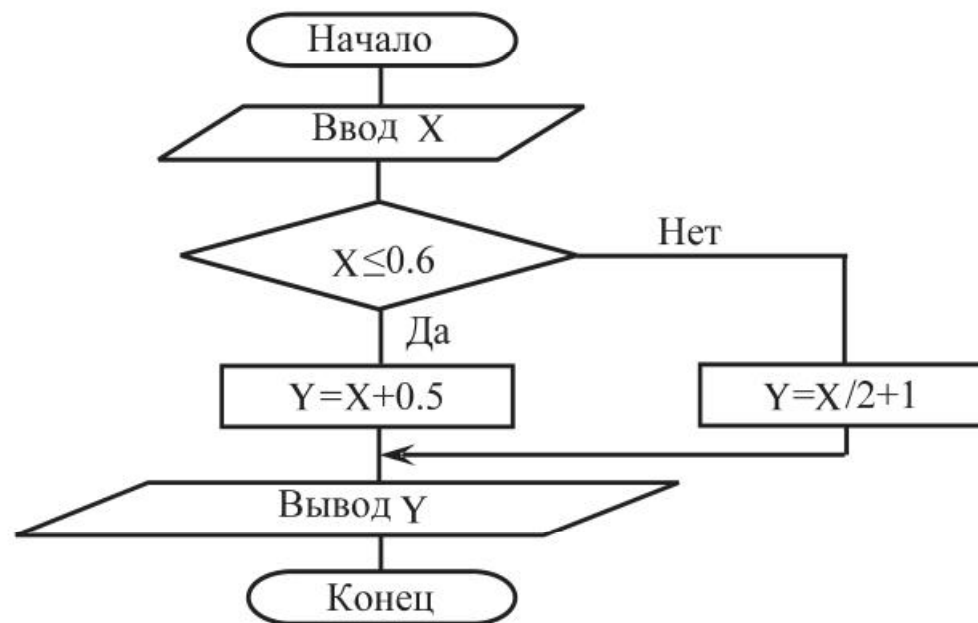
2. Программа

```

#include <stdio.h>
int main() {
    double c,d,m;
    printf("введите c и d\n");
    scanf("%lf%lf",&c,&d);
    m=c<d ? c:d;
    //вычисление значения переменной m с помощью условной операции
    printf("m=%lf",m);
    getchar();
    return 0;
}
  
```

Пример 2. Дано x . Вычислить $y = \begin{cases} x + 0,5, & \text{если } x \leq 0,6; \\ \frac{x}{2} + 1, & \text{если } x > 0,6. \end{cases}$

1. Схема алгоритма



Пояснение. Входным данным является X . Перед вычислением Y проверяется условие « $X \leq 0,6$ ». Если результат «истина» (ответ «Да»), то происходит вычисление значения Y по формуле $Y = X + 0,5$ ($X \leq 0,6$). Если результат «ложь» (ответ «Нет»), то значение Y вычисляется по формуле $Y = X/2 + 1$ ($X > 0,6$). После вычисления Y (по одной из формул), на экран выводится результат (значение Y).

2. Программа

```

#include <stdio.h>
int main() {
    double x, y;
    printf("Введите x ");
    scanf("%lf",&x);
    y= x<=0.6 ? x+0.5 : x/2+1;
    //вычисление значения переменной y с помощью условной операции
    printf("y=%lf\n",y);
    getchar();
    return 0;
}
  
```

4.3. Оператор выбора

Оператор выбора (оператор переключатель) служит для выбора действий из нескольких возможных альтернативных вариантов.

Оператор выбора имеет вид:

```
switch (<переключатель>)  
{ case <константа1>: <операторы1>;  
  case <константа2>: <операторы2>;  
  .....  
  case <константаN>: <операторыN>;  
  default: <операторы>;  
}
```

Здесь `switch` — ключевое слово (название) оператора выбора;
`<переключатель>` (селектор) — выражение (переменная) любого целого типа — `char`, `short int`, `int`, `unsigned int`, `long int`. Выражения (переменные) любого вещественного типа — `float`, `double`, `long double` — в качестве переключателя использовать запрещается;

`case` — ключевое слово;

`<константа>` — константа выбора;

`default` — ключевое слово, которое используется в случае, если ни одна из альтернатив не выбрана.

Оператор выбора выполняется следующим образом:

- сначала вычисляется значение выражения — `<переключатель>`;
- затем это значение по очереди сравнивается с константами выбора;
- если значение `<переключатель>` совпадает с какой-то константой выбора, то выполняются операторы, соответствующие этой константе выбора;
- если выполненные операторы не предусматривают какого-либо перехода, то далее выполняются операторы всех следующих вариантов, пока не появится оператор перехода или не закончится оператор выбора;
- если значение `<переключатель>` не совпало ни с одной из констант выбора, то выполняются `<операторы>`, соответствующие ключевому слову `default`.

Пример 1. Ввести номер дня недели и вывести его название.

Рассмотрим несколько вариантов программы решения этого примера. Сначала неправильный вариант, который при правильном вводе будет выводить на экран некорректную информацию.

1. Программа (неправильный вариант).

```
#include <stdio.h>
int main() {
    int n;                                //описание целой переменной n
    printf("Введите номер дня недели ");
    scanf("%d",&n);
    switch (n)
    {   case 1: printf("понедельник\n");
        case 2: printf("вторник\n");
        case 3: printf("среда\n");
        case 4: printf("четверг\n");
        case 5: printf("пятница\n");
        case 6: printf("суббота\n");
        case 7: printf("воскресенье\n");
        default: printf("неверный номер дня недели\n");
    }
    getchar();
    return 0;
}
```

Программа будет выполняться следующим образом; если будет введено значение n меньше 1 (0 или какое-нибудь отрицательное число) или больше 7 (8 и далее), то это значение не совпадет ни с одной константой выбора, тогда выполняется оператор, соответствующий ключевому слову `default`, и на экране появится текст:

неверный номер дня недели

Если же будет введено значение n , равное, например, 3 , то сначала выполнится оператор, соответствующий константе выбора 3 , а затем все последующие операторы из остальных ветвей оператора выбора. Информация на экране:

среда
четверг
пятница
суббота
воскресенье
неверный номер дня недели

То есть программа работает неверно.

Чтобы избежать этого, используют оператор `break` — прерывания действия (оператор выхода из оператора выбора (или из оператора цикла)).

2. *Программа (верный вариант) с использованием оператора `break`.*

```
#include <stdio.h>
int main() {
    int n;
    printf("Введите номер дня недели ");
    scanf("%d",&n);
    switch (n)
    { case 1: printf("понедельник\n"); break;
      case 2: printf("вторник\n"); break;
      case 3: printf("среда\n"); break;
      case 4: printf("четверг\n"); break;
      case 5: printf("пятница\n"); break;
      case 6: printf("суббота\n"); break;
      case 7: printf("воскресенье\n"); break;
      default: printf("неверный номер дня недели\n");
    }
    getchar();
    return 0;
}
```

Этот вариант будет выполняться следующим образом: если будет введено значение `n` меньше *1* (*0* или какое-нибудь отрицательное число) или больше *7* (*8* и далее), то это значение не совпадет ни с одной константой выбора, тогда выполняется оператор, соответствующий ключевому слову `default`, и на экране появится текст:

```
неверный номер дня недели
```

Если же будет введено значение `n`, равное *3*, то начнут выполняться операторы, соответствующие константе выбора *3*: сначала оператор вывода `printf("среда\n");`, затем выполнится оператор `break`, и выполнение оператора выбора будет закончено. На экране появится текст:

```
среда
```

Пример 2. Ввести два числа типа `double` и знак арифметической операции. Вычислить значение полученного выражения.


```
#include <stdio.h>
int main()
{double a,b,c;
char z;
printf("введите два числа и знак через пробел\n");
scanf("%lf %lf %c",&a,&b,&z);
switch (z)
{ case '+': c=a+b; break;
  case '-': c=a-b; break;
  case '*': c=a*b; break;
  case '/': c=a/b; break;
default: printf("неверный знак операции\n");
}
getchar();
return 0;
}
```

В языке C++ в операторе выбора запрещается использовать диапазоны значений типа 3..5, вместо этого нужно каждый раз писать ключевое слово case, например, case 3:, case 4:, case 5:.

Пример 3. По номеру месяца вывести время года.

```
#include <stdio.h>
int main() {
int n;
printf("введите номер месяца ");
scanf("%d",&n);
switch (n)
{ case 12:
  case 1:
  case 2: printf("зима\n"); break;
  case 3:
  case 4:
  case 5: printf("весна \n"); break;
  case 6:
  case 7:
  case 8: printf("лето\n"); break;
  case 9:
  case 10:
  case 11: printf("осень\n"); break;
default: printf("неверный номер месяца\n");
}
getchar();
return 0;
}
```

Тесты

1. Переменная `Month` содержит номер месяца. Укажите неверный вариант условия того, что месяц — летний.

- а) `(Month==6) || (Month==7) || (Month==8);`
- б) `(Month >5) && (Month <9);`
- в) `(Month >5) || (Month <9);`

2. Определите значение логического выражения: `(X<1) || (X=2)` при `X=5`.

- а) истина;
- б) ложь;
- в) нет правильного ответа.

3. Укажите неправильный вариант использования условного оператора `if`:

- а)

```
if (A==23)
    printf ("A== 23\n")
else printf ("A!= 23\n");
```
- б)

```
if (A>23)
    printf("A > 23\n");
else printf ("A <= 23\n");
```
- в)

```
if (A<23)
    printf("A < 23\n");
else printf ("A > = 23\n");
```

4. Чему будет равно значение переменой `Y` после выполнения следующего фрагмента программы?

```
...
X=17;
if (X<3)
    Y=10;
else Y=(5*X+3)/(4*(X+8));
...
```

- а) 10;
- б) 0.88;
- в) фрагмент содержит ошибку и не будет работать.

5. Выберите правильный вариант использования условного оператора `if` для нахождения `MAX{C*D, E+F}`:

- а)

```
if (C*D>E+F)
    MAX=C*D;
else MAX:=E+F;
```

```
б) X=C*D;  
   Y=E+F;  
   if (X>Y)  
       MAX=X;  
   else MAX=Y;
```

в) оба варианта правильные.

6. Какого типа должна быть переменная S, чтобы следующий фрагмент программы работал правильно?

```
...  
printf("Подпишите одну букву и нажмите Enter");  
printf('Ma');  
scanf("%c",&S);  
switch (S)  
{case 'n': printf("Это мужчина\n"); break;  
  case 'p': printf("Это карта\n"); break;  
  case 'y': printf("Это весенний месяц\n"); break;  
  default: printf("Неизвестное слово\n");  
}  
...  
а) int;  
б) double;  
в) char.
```

Задания

Для каждой задачи составить схему алгоритма и написать программу.

1. Дано x . Вычислить $y = \begin{cases} 2x + 3, & \text{если } x < 1; \\ 4x + 5, & \text{если } x \geq 1. \end{cases}$

2. Витязь на распутье:

Куда поедешь? 1- НАПРАВО 2- НАЛЕВО Введи значение:

В зависимости от введенного значения вывести:

Если НАПРАВО поедешь, то себя спасешь, коня потеряешь. Если НАЛЕВО поедешь, то коня спасешь, себя потеряешь. Иначе женатому быть (дополнительно).

3. Витамины.

Ввод:

«Сколько яблок ты можешь съесть?»

(Используя переменную Kol1, ввести с клавиатуры количество яблок).

«Сколько яблок может съесть твой друг?»

(Используя переменную Kol2, ввести с клавиатуры количество яблок).

Вывод:

«Я могу съесть ... яблок»

«Мой друг может съесть ... яблок»

«Мы можем съесть ... яблок»

«Слишком много» (если общее количество яблок > 10) или«Мало витаминов получаете» (если общее количество яблок ≤ 10).

Дополнительно: > 10 — «Много»
 ≥ 5 или ≥ 10 — «Хорошо»
 < 5 — «Плохо»

4. Даны C, M, X, B . Найти $Z = \min(C + M, 2X - B)$.

5. Дано x . Вычислить $y = \begin{cases} \frac{3}{x+2}, & \text{если } x \neq -2; \\ 4, & \text{если } x = -2. \end{cases}$

6. Дано x . Вычислить $y = \begin{cases} x - 10, & \text{если } x > 20; \\ 3x, & \text{если } 0 \leq x \leq 20; \\ \frac{x}{5}, & \text{если } x < 0. \end{cases}$

7. Даны A, B, C, D . Найти $M = \max\{A * B, C * D\}$.8. Даны A, B, C, D . Найти $P = \min\{A + B, C - D\}$.

9. Программа, которая проверяет: положительно ли введенное число, и выводит ответ на экран.

10. Вводятся два числа, после чего проверяется: больше ли сумма первого числа с самим собой, чем второе.

11. Вводятся два числа, после чего из первого вычитается второе, и если результат отрицательный, выводится, что он отрицателен. Иначе ответ выводится на экран.

12. Диалог с ПК.

Ввод:

- «Сколько времени в течение недели ты сидишь за компьютером (в часах)?»

Вывод:

- «Я сижу за компьютером ... часов»
- «Ты можешь испортить зрение» (если больше 10 ч) или «Подумай, прежде чем садиться за компьютер: сделал ли ты уроки?» (если меньше 10 ч).

13. Дано x (вводится с клавиатуры). Вычислить значение функции $s = \frac{1}{x} - \frac{1}{1+x}$. Если вычислить s невозможно, вывести на экран сообщение «В точке $X=(\text{значение } X)$ s не существует» (для учеников 6—8 классов).

Решить данную задачу для функции $s(x) = \sqrt{\frac{1}{\sqrt{x}-1}} - 2$ (для учеников 9—11 классов).

14. Однажды встретились три толстяка и начали спорить, кто же из них самый толстый. После долгого спора обратились к мудрецу. Мудрец думал, думал, да и сказал: «Тот из вас самый толстый, у кого больше веса». Так кто же из них?

15. Проверка действительности пропуска. Пропуск действителен, если код пропуска ≤ 100 . В зависимости от введенного кода пропуска выдать сообщение «Турникет открыт» или звуковой сигнал.

16. Проверить, имеет ли пользователь право доступа или нет.

17. Проверить, есть ли у пользователя водительские права. Если есть, то ввести по запросу его водительскую категорию и вывести на экран сообщение «Вы имеете водительские права категории ...».

18. Даны A, B, C . Найти $\text{MIN}(A, B, C)$.

19. Определить, успевает ли пассажир на старт космического корабля «Земля—Марс»?

20. Определить наибольшее и наименьшее из трех чисел, вводимых с клавиатуры.

21. Вводятся три числа. Найти сумму двух больших из них.

22. По коэффициентам a, b, c , введенным с клавиатуры, определить, сколько корней имеет квадратное уравнение $a \cdot x^2 + b \cdot x + c = 0$. Вывести корни уравнения, если они существуют.

23. Составить программу, определяющую, пройдет ли график функции $y(x) = 5x^2 - 7x + 2$ через заданную точку с координатами (a, b) .

24. К финалу конкурса лучшего по профессии «Специалист электронного офиса» были допущены трое: Иванов, Петров, Сидоров. Соревнования проходили в три тура. Иванов в 1-м туре набрал $m1$ баллов, во 2-м — $n1$, в 3-м — $p1$. Петров — соответственно $m2$, $n2$ и $p2$; Сидоров — $m3$, $n3$ и $p3$. Составить программу, определяющую, сколько баллов набрал победитель, и вывести его фамилию.

25. Составить программу, реализующую эпизод применения компьютера в книжном магазине. Компьютер запрашивает стоимость книг, сумму денег, внесенную покупателем; если сдачи не требуется, выводит на экран «Спасибо»; если денег внесено больше, то печатает «Возьмите сдачу» и указывает сумму сдачи; если денег недостаточно, то выводит об этом сообщение и указывает размер недостающей суммы.

26. Даны три числа: a , b , c . Определить, могут ли они быть сторонами треугольника, и если могут, то определить его тип: равносторонний, равнобедренный, разносторонний. (Условие существования треугольника: сумма длин любых двух сторон треугольника превышает длину 3-й стороны. Следует также учесть случай, когда длина одной из сторон равна нулю или имеет отрицательное значение.)

27. По паролю, вводимому с клавиатуры, определить степень допуска сотрудника к секретной информации в базе данных. Доступ к базе имеют только 6 человек, разбитых на три группы по степени допуска. Они имеют следующие пароли: 9583, 1747 — доступны модули A , B , C базы данных; 3331, 7922 — доступны модули B , C ; 9455, 8997 — доступен только модуль C . Придумайте сообщения, которые будут выводиться на экран в случае удачного и неудачного ввода паролей.

28. Запросите текущее время (в часах). В зависимости от времени суток выведите приветствие на экран: «Доброе утро!», или «Добрый день!», или «Доброй ночи!»

29. Написать программу, которая анализирует человека по возрасту и относит его к одной из четырех групп: дошкольник (младше 6 лет), ученик (от 7 до 22 лет), работник (от 23 до 60 лет), пенсионер (старше 60 лет).

30. Вычислить значение y для заданного x .

$$y = \begin{cases} 0, & x < -5; \\ 3 \cdot x - 5, & -5 \leq x \leq -2; \\ x, & -2 \leq x < 2; \\ x^2, & 2 \leq x < 5; \\ 3, & x \geq 5. \end{cases}$$

Решить задачу двумя способами: с помощью оператора **if** и с помощью оператора **Case**.

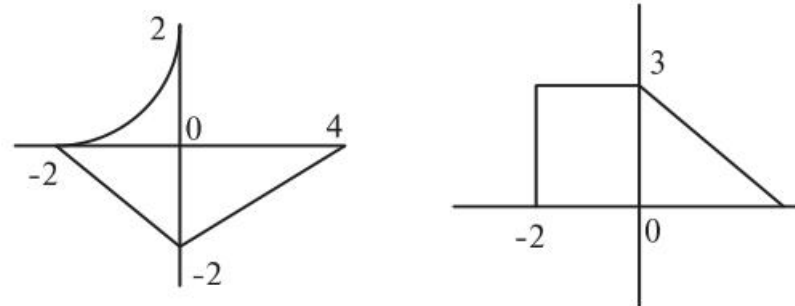
31. Дано x (вводится с клавиатуры). Вычислить значение функции $s = \frac{5-x}{x+2} - \frac{x^2+1}{1+x}$. Если вычислить s невозможно, вывести на экран сообщение «В точке $X=(\text{значение } X)$ s не существует». Вывод результатов оформить в виде меню:

1. Вывод X .
2. Вывод s .

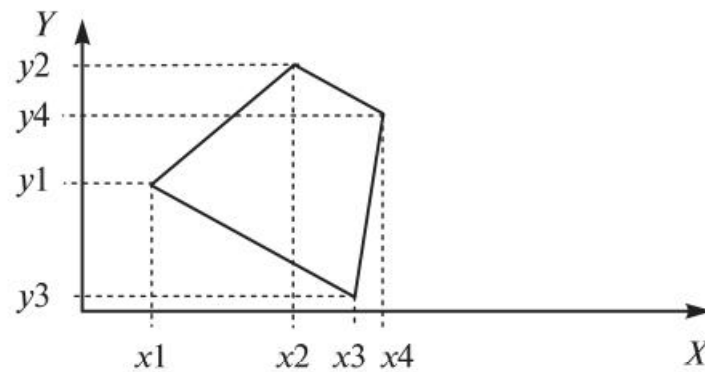
Если пункт меню выбран неверно, то вывести сообщение об ошибке.

32. Ввести два числа и символ — знак арифметической операции. В зависимости от введенного знака операции вычислить значение арифметического выражения.

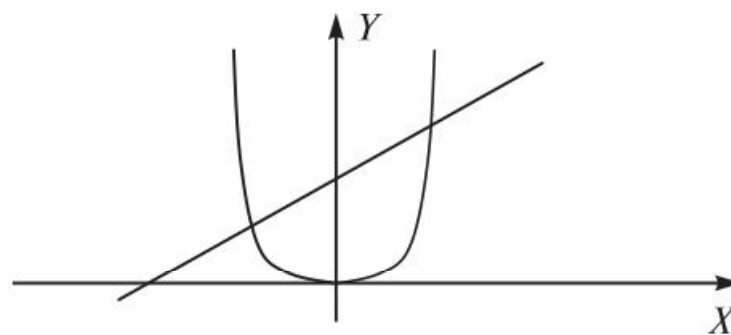
33. Определить, принадлежит ли точка $A(x, y)$ заданной фигуре (левый рисунок). Определить, принадлежит ли точка $C(x, y)$ заданной фигуре (правый рисунок).



34. Определить, принадлежит ли точка $A(x, y)$ заданной фигуре.



35. Определить, принадлежит ли точка $N(a, b)$ заданной фигуре, ограниченной графиками функций $y = x^2$ (парабола) и $y = kx + c$ (прямая, k и c вводятся с клавиатуры).



36. Написать программу, которая по выбору пользователя вычисляет площадь прямоугольника либо значение функции $y(x)$. Организовать на экране меню:

```
*****
Выберите один из пунктов меню, нажав нужную цифру.
1: Вычисление площади прямоугольника.
2: Вычисление функции  $Y(x) = (2+x) / (x*x-1)$ .
3: Выход из программы.
*****
```

При вычислении площади прямоугольника длины сторон вводятся с клавиатуры. При вычислении y с клавиатуры вводится значение x (учесть, что значение y может не существовать).

Контрольные вопросы

1. Для чего используется условный оператор?
2. Какие формы оператора `if` вы знаете?
3. Что такое <условие> в условном операторе? Какие требования к нему предъявляются? Приведите примеры.
4. Приведите примеры простого условия в условном операторе.
5. Что такое сложное условие в условном операторе? Приведите примеры.
6. Какие логические операции в языке C++ вы знаете?
7. Чем операция «логического И» отличается от операции «логического ИЛИ»? Приведите примеры использования для каждой операции.
8. Объясните на примере принцип работы краткой формы условного оператора.
9. Объясните на примере принцип работы полной формы условного оператора.
10. Как показать условный оператор на схеме алгоритма? Приведите примеры.
11. Можно ли в условном операторе в ветви `else` использовать два и более операторов? Приведите пример.
12. Можно ли использовать условный оператор внутри другого условного оператора (например, в ветви `else`)? Приведите пример.
13. Для чего используется условная операция? Приведите примеры.
14. Как записывается условная операция?
15. Каким образом вычисляется значение условной операции?

16. Объясните назначение оператора выбора.
17. Объясните принцип работы оператора выбора. Приведите примеры.
18. Что такое <выражение (селектор)> в операторе выбора? Какие требования к нему предъявляются?
19. Объясните, для чего в операторе switch используются константы выбора. Приведите пример.
20. Можно ли в одной ветви оператора switch указать несколько различных констант выбора? Как это сделать? Приведите пример.

Глава 5

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Алгоритмы, отдельные действия в которых повторяются несколько раз, называются *алгоритмами циклической структуры*.

При решении задач часто приходится повторять одни и те же действия. Например, при построении таблиц значений функции при изменении аргумента в некотором диапазоне. Для выполнения таких повторяющихся вычислений используются *операторы цикла*.

5.1. Оператор цикла с предусловием *while*

Оператор цикла с предусловием **while** применяется для выполнения части программы с неизвестным заранее числом повторений. В зависимости от условия может возникнуть ситуация, когда часть программы не будет выполнена ни разу.

```
while (<условие продолжения цикла (S)>)  
    <оператор>;
```

где <условие продолжения цикла (S)> — логическое выражение, условие продолжения цикла обязательно заключается в круглые скобки;

<оператор> — может быть или единичным оператором, или блоком операторов.

Описание работы оператора цикла с предусловием **while**:

- в начале каждого прохода по циклу проверяется *истинность* <условия продолжения цикла (S)>;
- если <условие продолжения цикла (S)> является истинным, то выполняется <оператор> (тело цикла) и снова прове-

ряется истинность <условия продолжения цикла (S)> и т. д.;

- выход из цикла происходит, как только <условие продолжения цикла (S)> становится ложным; в этом случае выполняется следующий за оператором цикла оператор программы.

Один проход по циклу называется *итерация*. Для оператора **while** одна итерация (проход по циклу) включает в себя проверку истинности <условия продолжения цикла> и выполнения <оператора> (тела цикла).

Так как истинность <условия_продолжения_цикла (S)> проверяется в начале каждой итерации, то <оператор> *может не выполняться ни разу*.

На схеме алгоритма оператор цикла с предусловием **while** представлен следующим образом (рис. 5.1).

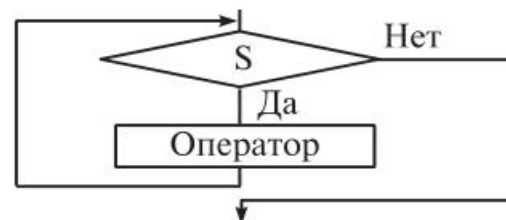
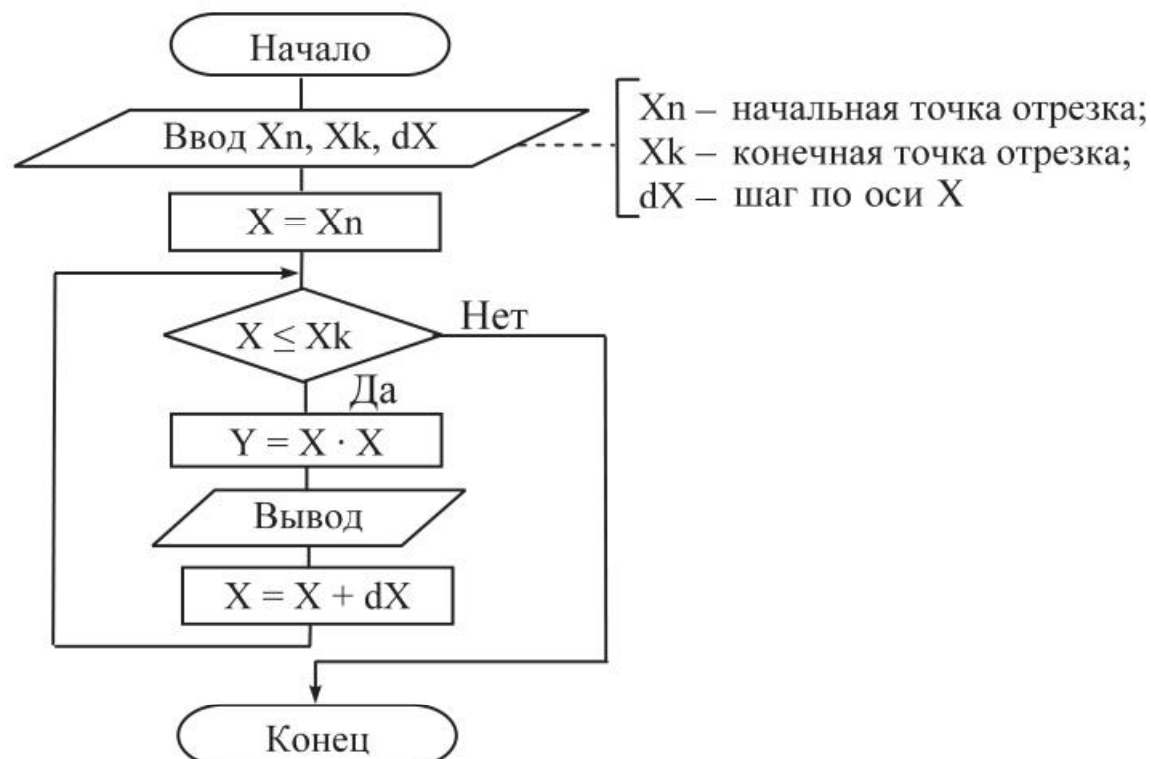


Рис. 5.1. Схема алгоритма оператора *while*

Пример программы с оператором цикла *while*. Вычислить значения функции $y = x^2$ на отрезке $0 \leq x \leq 1$ с шагом $\Delta x = 0,1$. Использовать оператор **while**.

1. *Схема алгоритма*



Пояснение. Нужно вычислить значения функции $Y = X^2$ для $X = 0, 0.1, 0.2, \dots, 0.9, 1$ (X изменяется от $X_n = 0$ до $X_k = 1$ с шагом $dX = 0.1$). В начале программы X присваивается значение начальной точки отрезка ($X = X_n$). Для того чтобы вычислить значения функции $Y = X^2$ в указанных точках, нужно для каждой точки отрезка проверить истинность условия « $X \leq X_k$ ». Если это условие истинно, то выполняются следующие действия:

- вычисляется значение $Y = X^2$ для очередного значения X ;
- выводится на экран пара значений (X, Y);
- вычисляется следующая точка отрезка ($X = X + dX$) для следующей итерации.

Эти действия должны выполняться до тех пор, пока значения $X \leq X_k$. Как только это условие станет ложным (т. е. $X > X_k$), указанная последовательность действий перестанет выполняться. То есть это алгоритм с циклической структурой.

2. Программа

```
#include <stdio.h>
int main() {
    double Xn, Xk, dX, X, Y;
    printf("Введите через пробел Xn, Xk, dX\n");
    scanf("%lf%lf%lf", &Xn, &Xk, &dX);
    X=Xn;                                //присвоить переменной X начальное значение Xn
    while (X<=Xk)                        //в цикле пока X <= Xk
    {
        Y=X*X;                          //вычисление значения Y
        printf("X=%lf Y=%lf\n", X, Y);  //вывод на экран значений X и Y
        X=X+dX;                         //получение нового значения X может X+=dX;
    }
    getchar();
    return 0;
}
```

Пояснение. В программе используются 5 переменных: X_n — начальная точка отрезка вычислений, X_k — конечная точка отрезка вычислений, dX — шаг вычислений, X — аргумент вычисляемой функции, Y — функция. Все переменные объявлены как переменные типа **double**.

В данной задаче используется оператор цикла **while** с предусловием « $X \leq X_k$ ». Так как в теле цикла выполняются три действия (вычисления Y , вывод значений X и Y на экран, вычисление следующего значения X), то используется блок операторов (выполняемые действия заключены в фигурные скобки $\{ \dots \}$).

5.2. Оператор цикла с постусловием **do-while**

Оператор цикла с постусловием **do-while** применяется для выполнения части программы с неизвестным заранее числом повторений. В зависимости от условия может возникнуть ситуация, когда часть программы выполнится всего один раз.

```
do  
<оператор>  
while ( <условие_продолжения_цикла (S) > );
```

где <условие_продолжения_цикла (S) > — *логическое выражение*;
<оператор> может быть или единичным оператором, или блоком операторов.

Описание работы оператора цикла с постусловием **do-while**:

- в каждом проходе по циклу сначала выполняется <оператор>;
- затем проверяется истинность <условия_продолжения_цикла (S) >;
- если <условие_продолжения_цикла (S) > — истинно, то выполняется следующий проход;
- выход из цикла происходит, если <условие_продолжения_цикла (S) > становится ложным; в этом случае выполняется следующий за **do-while** оператор программы.

Для оператора **do-while** итерация состоит из выполнения <оператора> (тела цикла) и проверки <условия_продолжения_цикла (S) >. Так как истинность <условия_продолжения_цикла (S) > проверяется в конце каждой итерации, то цикл с постусловием всегда выполняется хотя бы один раз.

На схеме алгоритма цикл с постусловием **do-while** представлен следующим образом (рис. 5.2).

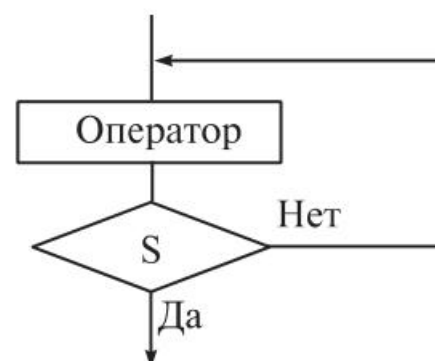
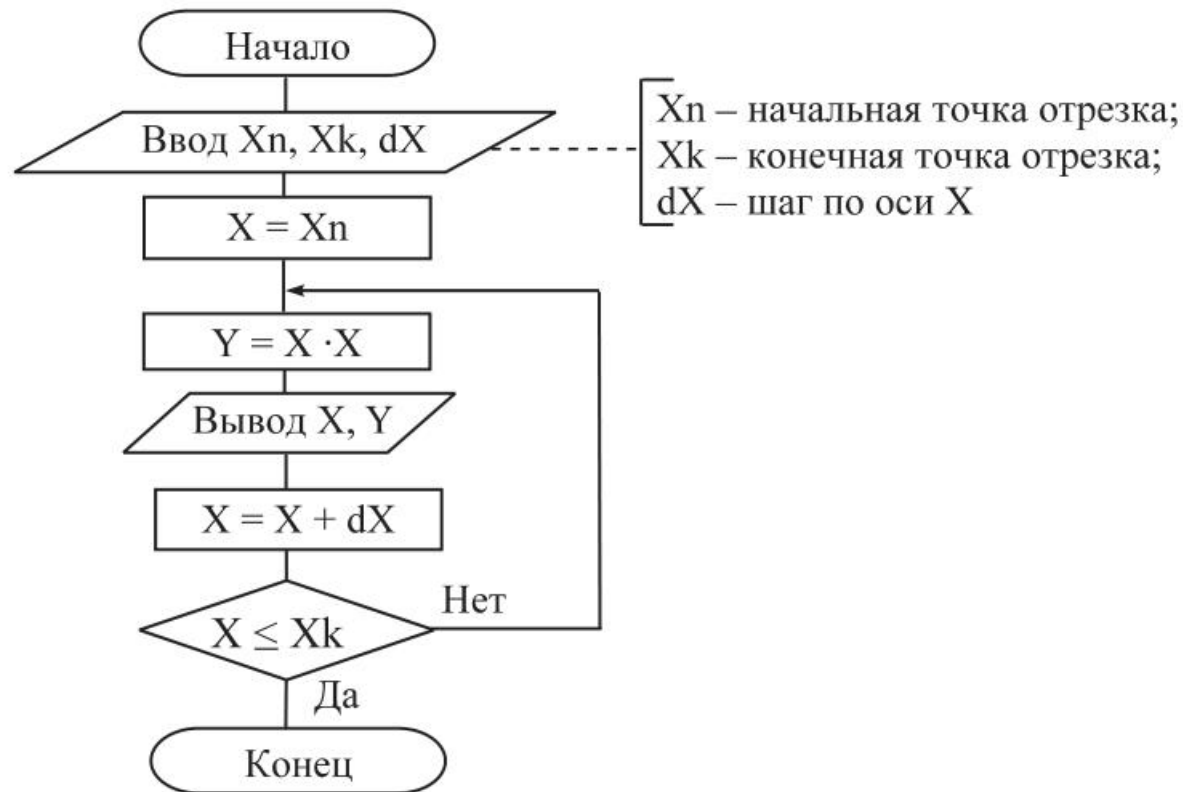


Рис. 5.2. Схема алгоритма оператора *Repeat*

Пример программы с оператором цикла *do-while*. Вычислить значения $y = x^2$ на отрезке $0 \leq x \leq 1$ с шагом $\Delta x = 0,1$. Использовать оператор **do-while**.

1. *Схема алгоритма*



Пояснение. Алгоритм решения задачи с использованием цикла с постусловием аналогичен алгоритму решения задачи с использованием цикла с предусловием (см. п. 4.2.1), за исключением того, что условие продолжения цикла ($X \leq X_k$) проверяется в конце каждой итерации.

2. *Программа*

```

#include <stdio.h>
int main() {
    double Xn, Xk, dX, X, Y;
    printf("Введите через пробел Xn, Xk, dX\n");
    scanf("%lf%lf%lf", &Xn, &Xk, &dX);
    X=Xn;                                //присвоить переменной X начальное значение Xn
    do {                                  //в цикле с постусловием делать
        Y=X*X;                            //вычисление значения Y
        printf("X=%lf Y=%lf\n", X, Y);    //вывод на экран значений X и Y
        X+=dX;                            //получение нового значения X (X=X+dX)
    } while (X<=Xk);                      //повторяем цикл пока X <= Xk
    getchar();
    return 0;
}
  
```

Пояснение. Для реализации вычислений в данной задаче используется оператор цикла **do-while** с постусловием (условие продолжения цикла — « $X \leq X_k$ »). В цикле используется блок операторов, заключенных в фигурные скобки.

5.3. Оператор цикла *for*

Оператор цикла **for** записывается следующим образом:

```
for (<выражение1>; <выражение2>; <выражение3>)  
    <оператор>;
```

где <выражение1> содержит операторы, в которых некоторым переменным присваиваются начальные значения;

<выражение2> — условие продолжения цикла (S);

<выражение3> содержит операторы, в которых у некоторых переменных изменяются значения;

<оператор> — может быть или единичным оператором, или блоком операторов.

Описание работы оператора цикла **for**:

- вначале (один раз) выполняются операторы <выражения1>;
- затем, в начале каждого прохода по циклу проверяется истинность <выражения2>;
- если <выражение 2> является истинным, то выполняется <оператор>;
- затем после <оператора> выполняются операторы <выражения3> и снова проверяется истинность <выражения2> и т. д.;
- выход из цикла происходит, как только <выражение2> становится ложным; в этом случае выполняется следующий за оператором цикла оператор программы.

Один проход по циклу называется *итерация*. Для оператора **for** одна итерация (проход по циклу) включает в себя проверку истинности <выражения2> и выполнения <оператора> и операторов <выражения3>.

Так как истинность <выражения2> проверяется в начале каждой итерации, то <оператор> *может не выполняться ни разу*.

Оператор цикла **for** рекомендуется использовать в случаях, если заранее известно количество раз повторения определенных операторов.

ров. На схеме алгоритма оператор цикла **for** представлен следующим образом (рис. 5.3).

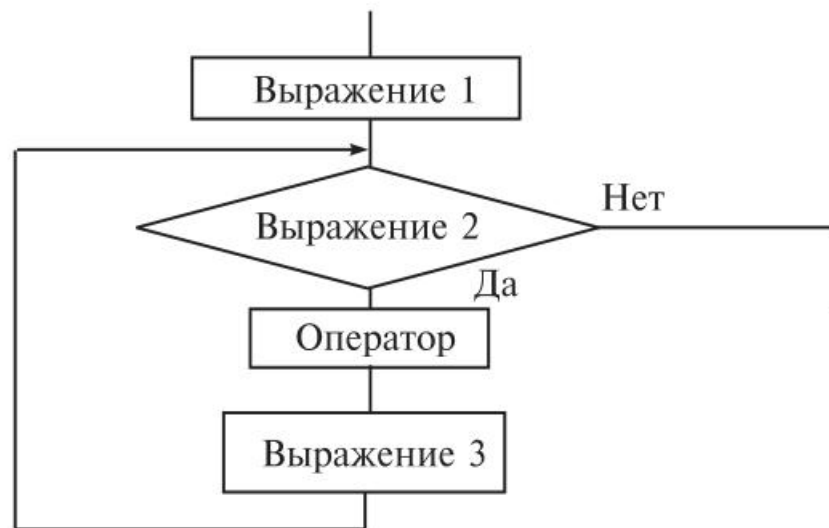
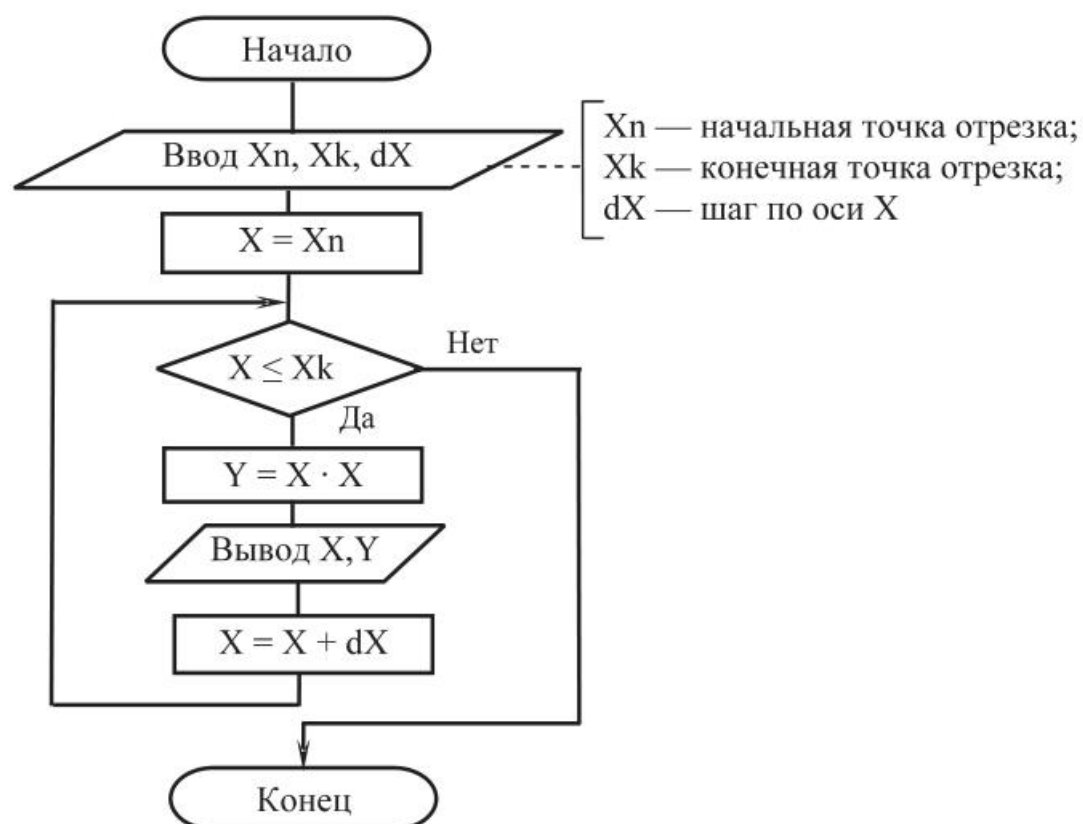


Рис. 5.3. Схема алгоритма оператора *for*

Пример 1 программы с оператором цикла *for*. Вычислить значения функции $y = x^2$ на отрезке $0 \leq x \leq 1$ с шагом $\Delta x = 0,1$. Использовать оператор **for**.

1. Схема алгоритма



Пояснение. Так как используется оператор цикла **for**, то необходимо определить <выражение1>, <выражение2> и <выражение3>.

<выражение1> содержит операторы, в которых некоторым переменным присваивается начальное значение, в нашем случае это оператор $X = X_n$;

<выражение2> — это условие продолжения цикла $X \leq X_k$;

<выражение3> содержит операторы, в которых у некоторых переменных изменяется значение, в нашем случае — это оператор $X = X + dX$.

В начале программы выполняется оператор <выражения1> ($X = X_n$). Затем проверяется <выражение2>, если <выражение2> истинно, то выполняются следующие действия:

- вычисляется значение $Y = X^2$ для очередного значения X ;
- выводится на экран пара значений (X, Y) ;

Затем выполняется оператор <выражения3> — $X = X + dX$. Эти действия и оператор <выражения3> должны выполняться до тех пор, пока <выражение2> истинно. Как только это выражение станет ложным, указанная последовательность действий и оператор <выражения3> перестанут выполняться. То есть это алгоритм с циклической структурой.

2. Программа

```
#include <stdio.h>
int main() {
    double Xn, Xk, dX, X, Y;
    printf("Введите через пробел Xn, Xk, dX\n");
    scanf("%lf%lf%lf", &Xn, &Xk, &dX);
    for (X=Xn; X<=Xk; X=X+dX){                                //в цикле
        Y=X*X;                                                  //вычисление значения Y
        printf("X=%lf Y=%lf\n", X, Y);                          //вывод на экран значений X и Y
    }
    getchar();
    return 0;
}
```

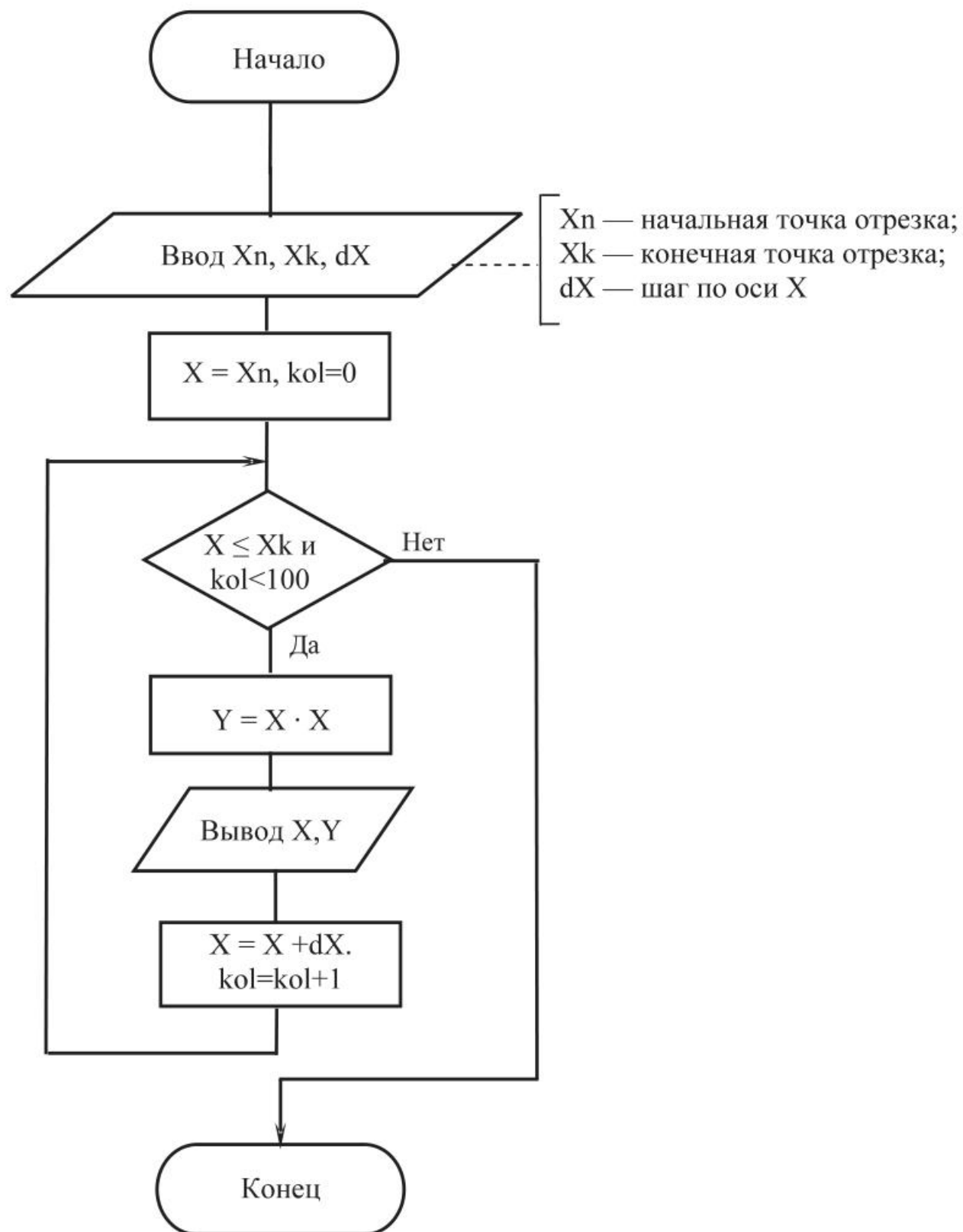
Пояснение. В программе используются 5 переменных: X_n — начальная точка отрезка вычислений, X_k — конечная точка отрезка вычислений, dX — шаг вычислений, X — аргумент вычисляемой функции, Y — функция. Все переменные объявлены как переменные типа **double**.

В данной задаче используется оператор цикла **for**. Так как в теле цикла выполняются 2 действия (вычисления Y , вывод значений X и Y на экран), то используется блок операторов (выполняемые действия заключены в фигурные скобки $\{ \dots \}$).

Пример 2 программы с использованием оператора *for*. Вычислить значения функции $y = x^2$ на отрезке $0 \leq x \leq 1$ с шагом $\Delta x = 0,1$. Опреде-

лить число повторений тела цикла, причем число повторений тела цикла должно быть меньше 100.

1. *Схема алгоритма*



Пояснение. Так как используется оператор цикла **for**, то необходимо определить <выражение1>, <выражение2> и <выражение3>.

<выражение1> содержит операторы, в которых некоторым переменным присваивается начальное значение, в нашем случае это операторы $X = X_n$; и $kol=0$;

<выражение2> — это условие продолжения цикла ($X \leq X_k$) и ($kol < 100$);

<выражение3> содержит операторы, в которых у некоторых переменных изменяется значение, в нашем случае — это операторы $X = X + dX$; и $kol++$.

В начале программы выполняются операторы <выражения1> ($X = X_n$ и $kol = 0$). Затем проверяется <выражение2>, если <выражение2> истинно, то выполняются следующие действия:

- вычисляется значение $Y = X^2$ для очередного значения X ;
- выводится на экран пара значений (X, Y);

Затем выполняется оператор <выражения3> — $X = X + dX$ и $kol++$. Эти действия и операторы <выражения3> должны выполняться до тех пор, пока <выражение2> истинно. Как только это выражение станет ложным, указанная последовательность действий и операторы <выражения3> перестанут выполняться. То есть это алгоритм с циклической структурой.

2. Программа

```
#include <stdio.h>
int main() {
    double Xn, Xk, dX, X, Y;
    int kol;
    printf("Введите через пробел Xn, Xk, dX\n");
    scanf("%lf%lf%lf", &Xn, &Xk, &dX);
    for (X=Xn, kol=0; (X<=Xk)&&(kol<100); X=X+dX, kol++){
        //в цикле
        Y=X*X; //вычисление значения Y
        printf("X=%lf Y=%lf\n", X, Y); //вывод на экран значений X и Y
    }
    printf("количество повторений = %d\n", kol);
    getchar();
    return 0;
}
```

Пояснение. В программе используются 6 переменных: X_n — начальная точка отрезка вычислений, X_k — конечная точка отрезка вычислений, dX — шаг вычислений, X — аргумент вычисляемой функции, Y — функция, kol — число повторений.

В данной задаче используется оператор цикла **for**. В качестве <выражения 1> используются два оператора, разделенных запятой ($X=X_n, kol=0$). В качестве <выражения 2> используется сложное ус-

ловие продолжения цикла $((X \leq X_k) \&\& (kol < 100))$. В качестве <выражения 3> используются два оператора $(X = X + dX, kol++)$. Так как в теле цикла выполняются 2 действия (вычисления Y , вывод значений X и Y на экран), то используется блок операторов (выполняемые действия заключены в фигурные скобки $\{ \dots \}$).

5.4. Решение задач с использованием операторов цикла

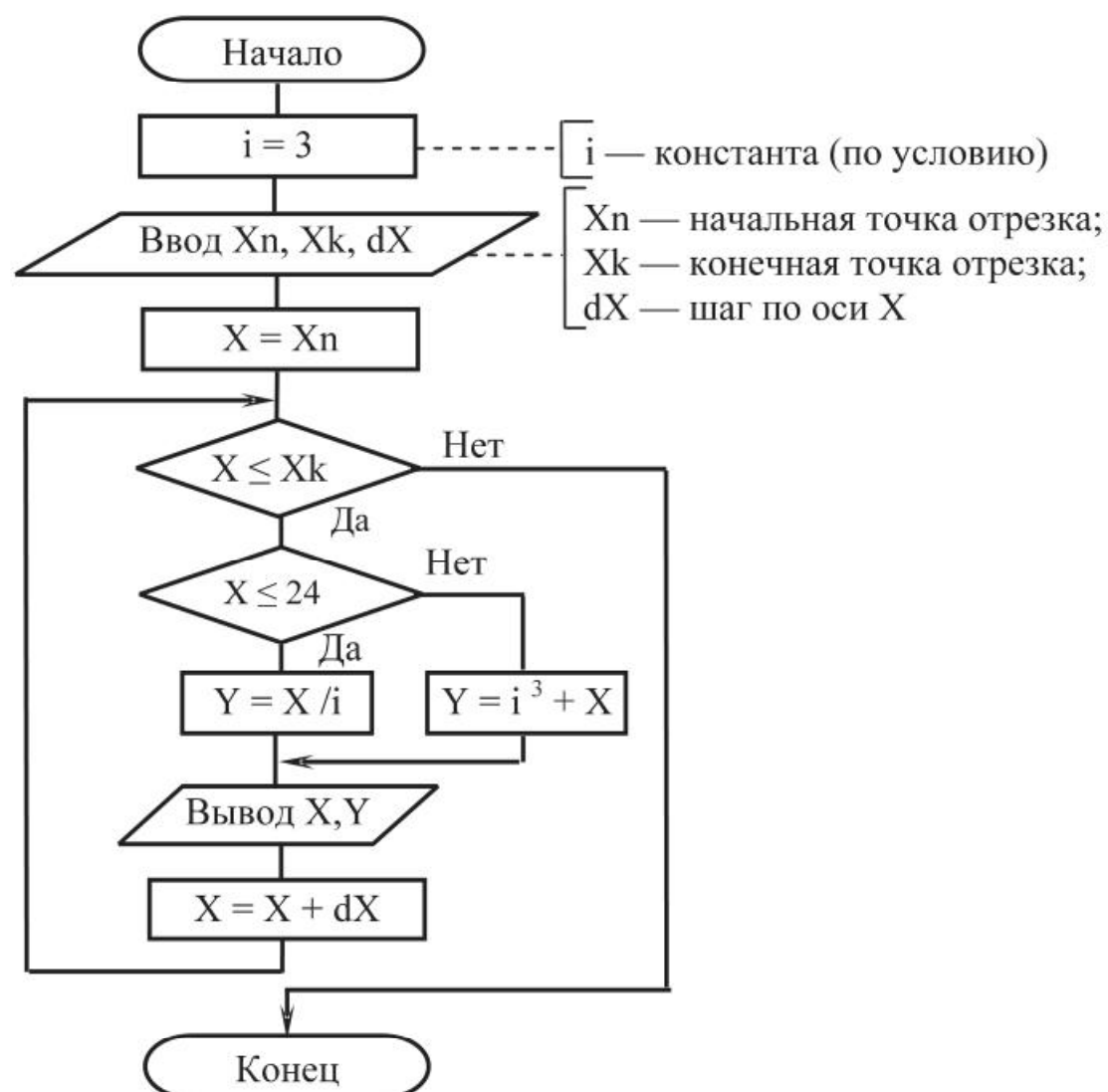
Задача 1. Дано $i = 3$. Вычислить значения Y на отрезке $15 \leq X \leq 26$ с шагом $\Delta X = 1,5$.

$$Y = \begin{cases} X/i, & \text{если } X \leq 24; \\ i^3 + X, & \text{если } X > 24. \end{cases}$$

Решить задачу тремя способами, используя различные операторы цикла.

1-й способ (**While**)

1. Схема алгоритма



Пояснение. В задаче требуется вычислить значения Y для различных значений X . Для каждого значения X (из указанного отрезка) выполняется:

- проверка условия « $X \leq 24$ ». Если оно истинно, то вычисление $Y = X/i$; если ложно, то вычисление $Y = i^3 + X$;
- вывод на экран очередной пары значений (X , Y);
- вычисление следующего значения X .

Для решения задачи используется алгоритм с циклической структурой, в котором сначала проверяется условие входа в тело цикла ($X \leq X_k$), т. е. в начале каждой итерации происходит проверка условия выполнения тела цикла (цикл с предусловием).

2. Программа

```
#include <stdio.h>
const int i=3;           //описание целой константы i равной 3
int main() {
    double Xn, Xk, dX, X, Y;
    printf("Введите через пробел Xn, Xk, dX\n");
    scanf("%lf%lf%lf", &Xn, &Xk, &dX);
    X=Xn;                 //присвоить переменной X начальное значение Xn
    while (X<=Xk)          //в цикле, пока X<=Xk
    { if (X<=24 )           //если X<=24
        Y=X/i;              //вычисление Y по верхней формуле
      else Y=i*i*i+X;        //иначе, вычисление Y по нижней формуле
      printf("X=%lf Y=%lf\n", X, Y); //вывод на экран значений X и Y
      X=X+dX;                //получение нового значения X
    }
    getchar();
    return 0;
}
```

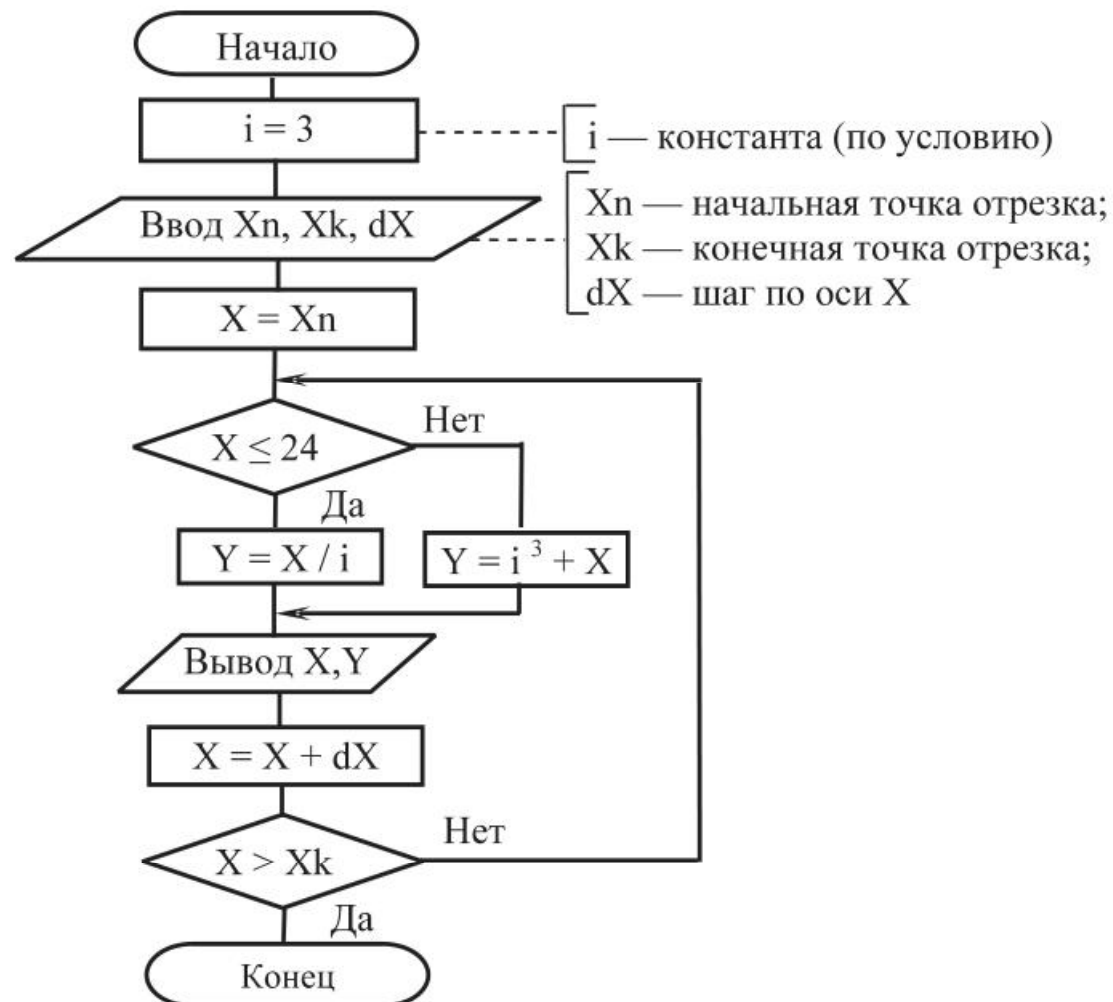
Пояснение. В программе используются 5 переменных (X_n , X_k , dX , X , Y) и 1 константа (i). Константа i — целочисленная (по условию задачи), объявлена перед заголовком функции `main()` с помощью ключевого слова **const**. Все переменные объявлены как переменные типа **double**.

Для вычисления значений Y на заданном отрезке используется оператор цикла **While** с предусловием « $X \leq X_k$ »: пока X не превышает значения X_k , для каждой точки указанного отрезка вычисляется значение Y по одной из формул в зависимости от значения X (оператор **if**)

и на экран выводится пара значений (X, Y) . Вычисление следующего значения X выполняется в конце каждой итерации.

2-й способ (do-while)

1. Схема алгоритма



Пояснение. В этом случае для решения задачи используется алгоритм с циклической структурой, в конце которого проверяется условие входа в цикл ($X \leq Xk$), т. е. в конце каждой итерации происходит проверка условия продолжения выполнения тела цикла (цикл с постусловием).

2. Программа

```

#include <stdio.h>
const int i=3;           //описание целой константы i равной 3
int main() {
    double Xn, Xk, dX, X, Y;
    printf("Введите через пробел Xn, Xk, dX\n");
    scanf("%lf%lf%lf", &Xn, &Xk, &dX);
    X=Xn;                 //присвоить переменной X начальное значение Xn
    do{                   //начало цикла do-while
        if (X<=24 )       //если X<=24
            Y=X/i;         //вычисление Y по верхней формуле
        Y=i³ + X;
        X=X + dX;
    } while (X >= Xk);
}
  
```

```

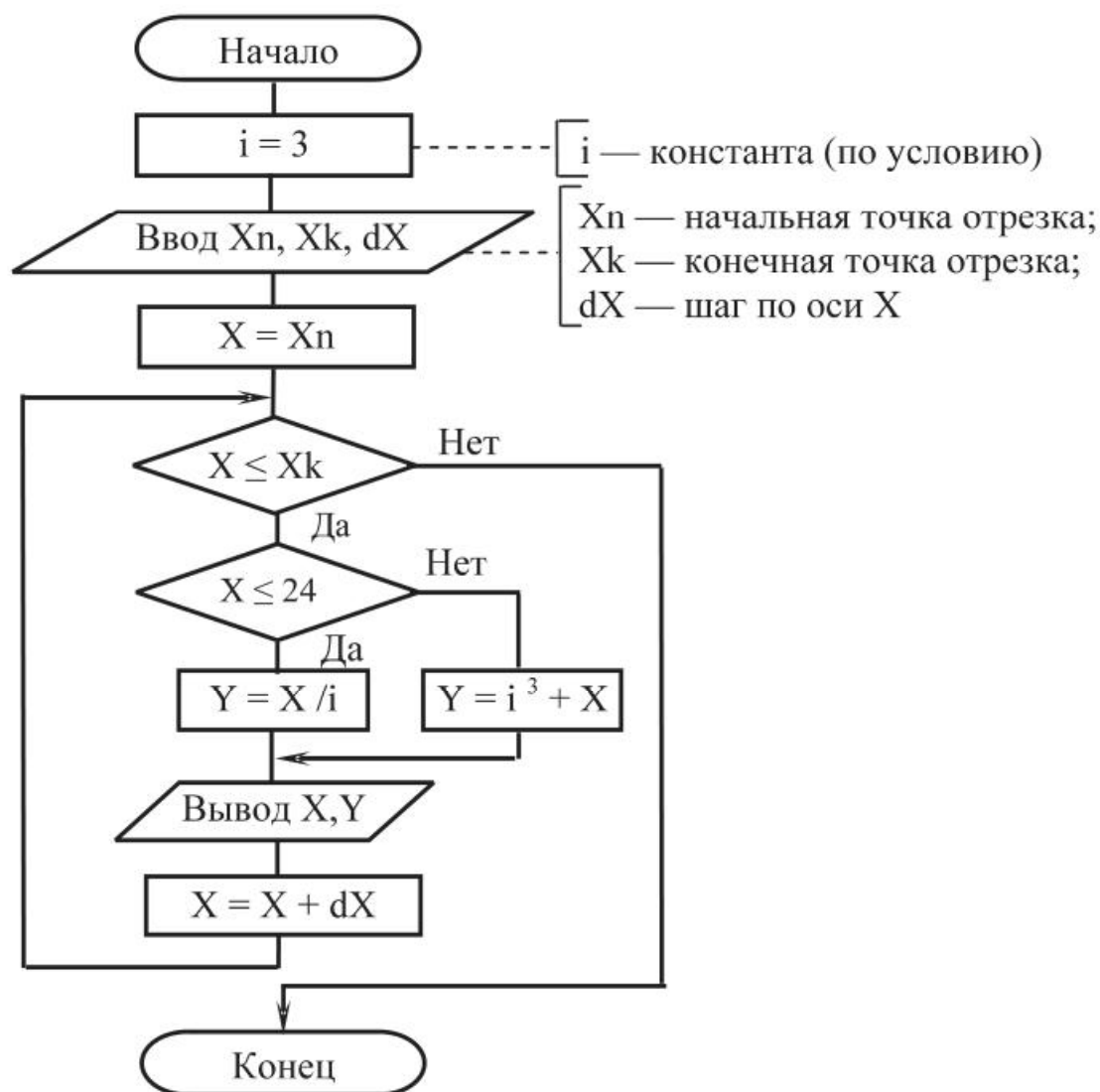
else Y=i*i*i+X;           //иначе, вычисление Y по нижней формуле
printf("X=%lf Y=%lf\n",X,Y); //вывод на экран значений X и Y
X=X+dX;                   //получение нового значения X
} while (X<=Xk);           //цикл, пока X <= Xk
getchar();
return 0;
}

```

Пояснение. Задача решается аналогично 1-му способу (**while**). Но для вычисления значений Y на заданном отрезке используется оператор цикла **do-while** с постусловием « $X \leq X_k$ »: пока условие « $X \leq X_k$ » истинно, для каждой точки указанного отрезка вычисляется значение Y по одной из формул в зависимости от значения X (оператор **if**) и на экран выводится пара значений (X, Y) . Вычисление следующего значения X выполняется в конце каждой итерации.

3-й способ (**for**).

1. Схема алгоритма



Пояснение. Так как используется оператор цикла **for**, то необходимо определить <выражение1>, <выражение2> и <выражение3>.

<выражение1> содержит операторы, в которых некоторым переменным присваивается начальное значение, в нашем случае это оператор $X = X_n$;

<выражение2> — это условие продолжения цикла $X \leq X_k$;

<выражение3> содержит операторы, в которых у некоторых переменных изменяется значение, в нашем случае — это оператор $X = X + dX$.

В начале программы выполняется оператор <выражения1> ($X = X_n$). Затем проверяется <выражение2>, если <выражение2> истинно, то выполняются следующие действия:

- вычисляется значение Y для очередного значения X ;
- выводится на экран пара значений (X, Y) ;

Затем выполняется оператор <выражения3> — $X = X + dX$. Эти действия и оператор <выражения3> должны выполняться до тех пор, пока <выражение2> истинно. Как только это выражение станет ложным, указанная последовательность действий и оператор <выражения3> перестанет выполняться. То есть это алгоритм с циклической структурой.

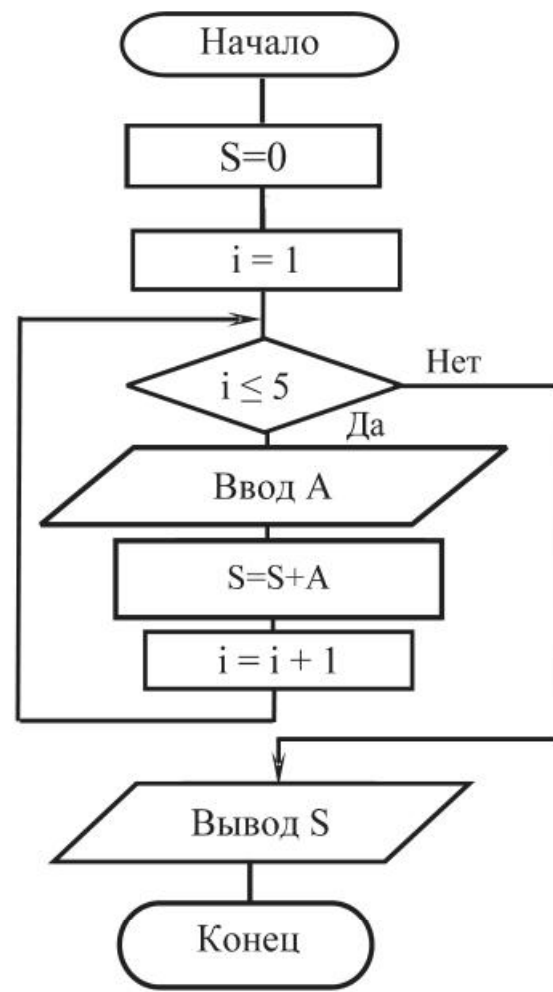
2. Программа

```
#include <stdio.h>
const int i=3;           //описание целой константы i равной 3
int main() {
    double Xn, Xk, dX, X, Y;
    printf("Введите через пробел Xn, Xk, dX\n");
    scanf("%lf%lf%lf", &Xn, &Xk, &dX);
    for (X=Xn; X<=Xk; X=X+dX){           //в цикле
        if (X<=24 )                       //если X<=24
            Y=X/i;                         //вычисление Y по верхней формуле
        else Y=i*i*i+X;                   //иначе вычисление Y по нижней формуле
        printf("X=%lf Y=%lf\n", X, Y);    //вывод на экран значений X и Y
        X=X+dX;                           //получение нового значения X
    }
    getchar();
    return 0;
}
```

Пояснение. В программе используются пять переменных (X_n , X_k , dX , X , Y) и одна константа (i). Константа i — целочисленная (по условию задачи), объявлена перед заголовком функции `main()` с помощью ключевого слова **const**. Все переменные объявлены как переменные типа **double**.

Задача 2. Ввести последовательность из 5 чисел с клавиатуры. Найти сумму всех чисел последовательности.

1. *Схема алгоритма*



Пояснение. В начале программы, пока с клавиатуры еще ничего не вводилось, сумма чисел равна нулю ($S=0$). Это начальное значение суммы. Далее с клавиатуры вводится 1-е число в переменную A . Это число прибавляется к сумме S . Аналогично поступаем со 2-м, 3-м, 4-м и 5-м числами: каждое число нужно ввести с клавиатуры и прибавить к сумме уже введенных чисел. В результате в переменной S будет сформирована сумма пяти чисел.

В решении есть повторяющаяся последовательность действий, поэтому алгоритм решения — циклический. Параметром цикла (i) является счетчик введенных чисел, который принимает значения от 1 до 5 с шагом 1.

2. *Программа*

```

#include <stdio.h>
int main() {
    double A, S;
    int i;
    S=0;                                     //присвоение сумме (S) начального значения

```

```

for (i=1; i<=5; i++){                               //начало цикла
printf("Введите число >");                          //вывод приглашения к вводу очередного числа
scanf("%lf",&A);                                     //ввод очередного числа
S=S+A;                                                //прибавление к старому значению S числа A
}
printf("S=%lf\n",S);
getchar();
return 0;
}

```

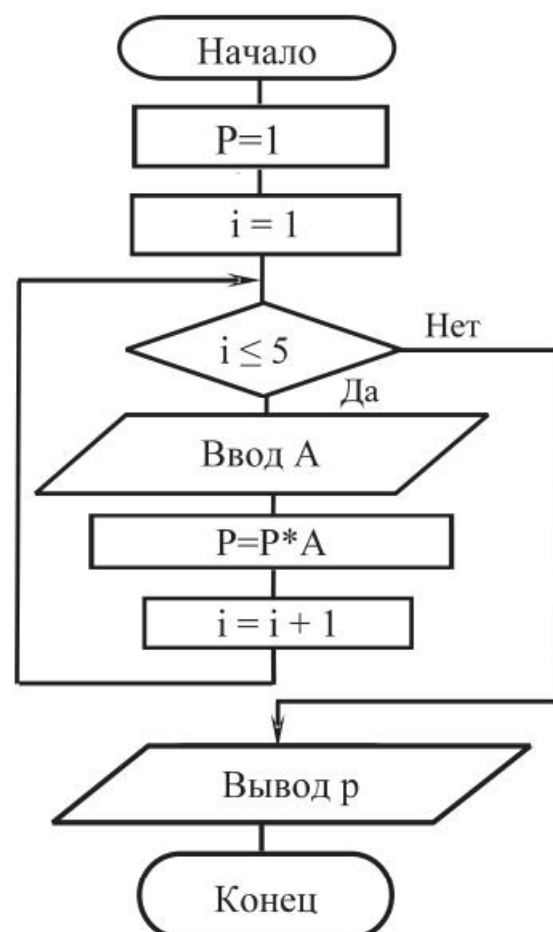
Пояснение. Для решения задачи в программе пять раз подряд нужно выполнить два действия:

- ввести с клавиатуры очередное число (в переменную A);
- прибавить введенное число к имеющейся сумме.

Для этого удобно использовать оператор цикла **for** (тело цикла будет выполняться пять раз). В программе используются три переменные: A — для хранения очередного числа, введенного с клавиатуры (может быть **double** или **int**); S — для подсчета суммы вводимых чисел (того же типа, что A); i — параметр цикла типа **int**.

Задача 3. Ввести последовательность из пяти чисел с клавиатуры. Найти произведение всех чисел последовательности.

1. *Схема алгоритма*



Пояснение. В начале программы, пока с клавиатуры еще ничего не вводилось, произведение равно 1 ($P = 1$). Это начальное значение произведения (при формировании произведения используется умножение, поэтому начальное значение произведения равно 1, а не нулю). Далее с клавиатуры вводится 1-е число в переменную A . Это число умножается на имеющееся произведение P . Аналогично поступаем со 2-м, 3-м, 4-м и 5-м числами: каждое число нужно ввести с клавиатуры и умножить на произведение уже введенных чисел. В результате в переменной P будет сформировано произведение пяти чисел.

В решении есть повторяющаяся последовательность действий, значит, алгоритм решения — циклический. Параметром цикла (i) является счетчик введенных чисел, который принимает значения от 1 до 5 с шагом 1.

2. Программа

```
#include <stdio.h>
int main() {
    double A, P;
    int i;
    P=1; //присвоение произведению (P) начального значения
    for (i=1; i<=5; i++){ //начало цикла
        printf("Введите число >");
        //вывод приглашения к вводу очередного числа
        scanf("%lf",&A); //ввод очередного числа
        P=P*A; //умножение старого значения P на число A
    }
    printf("P=%lf\n",P); //вывод на экран значения P
    getchar();
    return 0;
}
```

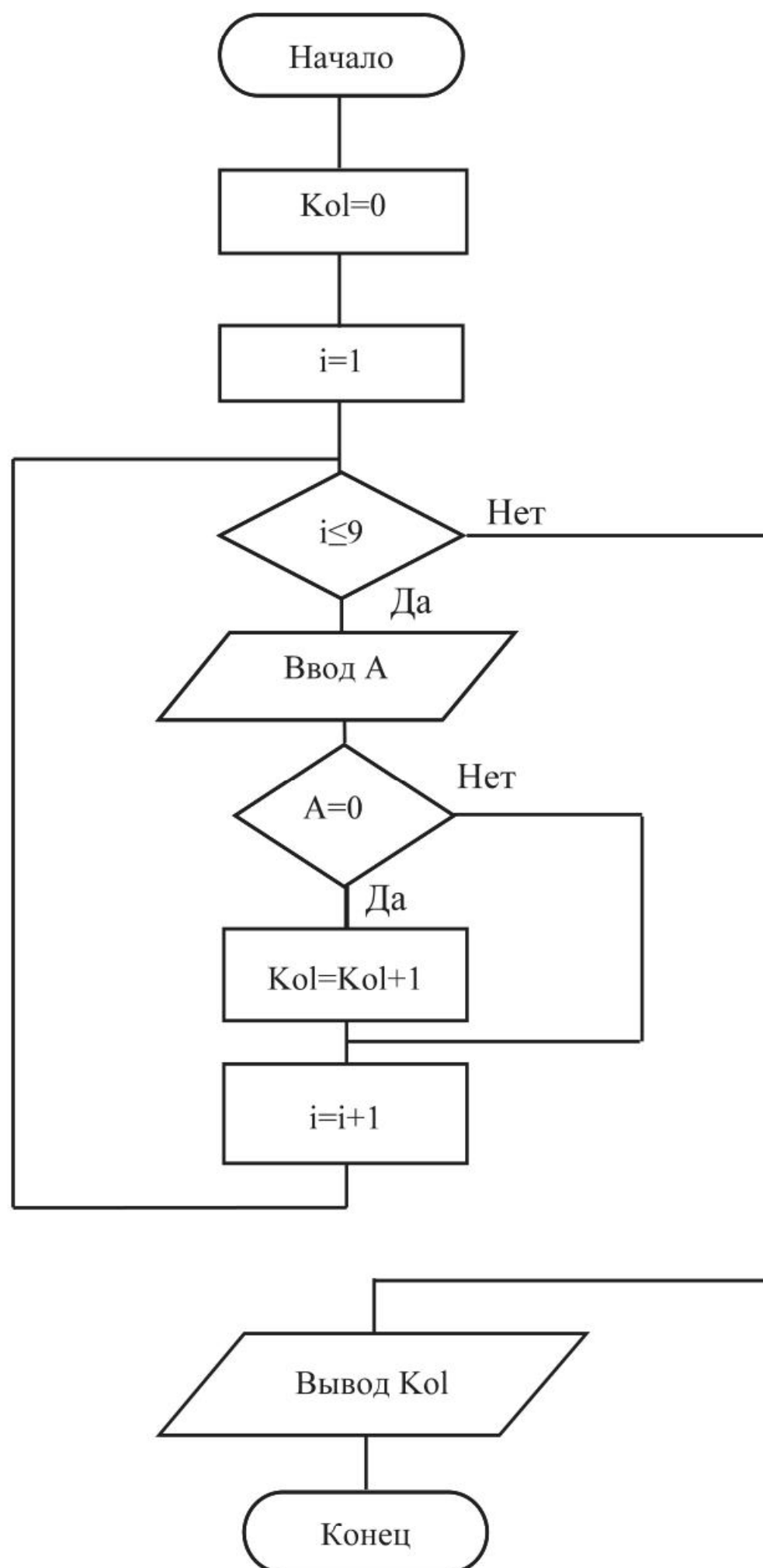
Пояснение. В программе пять раз подряд нужно выполнить два действия:

- ввести с клавиатуры очередное число (в переменную A);
- умножить введенное число на имеющееся произведение.

Для этого удобно использовать оператор цикла **for** (тело цикла будет выполняться пять раз). В программе используются три переменные: A — для хранения очередного числа, введенного с клавиатуры (может быть **double** или **int**); P — для подсчета произведения вводимых чисел (того же типа, что A); i — параметр цикла типа **int**.

Задача 4. Ввести последовательность из девяти чисел с клавиатуры. Найти количество чисел, значения которых равны нулю.

1. Схема алгоритма



Пояснение. В начале программы, пока с клавиатуры ничего еще не вводилось, количество нулевых чисел равно нулю ($Kol=0$). Это начальное значение количества. Далее с клавиатуры вводится 1-е число в переменную A . Если оно равно нулю ($A=0$), то количество нулевых чисел (Kol) увеличивается на 1. Аналогично поступаем с остальными числами. В результате в переменной Kol сформируется количество нулевых чисел последовательности.

В решении есть повторяющаяся последовательность действий, значит, алгоритм решения — циклический. Параметром цикла (i) является счетчик введенных чисел, который принимает значения от 1 до 9 с шагом 1.

2. Программа

```
#include <stdio.h>
int main() {
    double A;
    int Kol, i;
    Kol=0;                //присвоение количеству (Kol) начального значения
    for (i=1; i<=9; i++){ //начало цикла
        printf("Введите число >");
        //вывод приглашения к вводу очередного числа

        scanf("%lf",&A); //ввод очередного числа
        if (A == 0)      //если введенное число равно 0
            Kol=Kol+1;   //прибавление единицы к старому значению Kol
    }
    printf("Kol=%d\n",Kol); //вывод на экран значения Kol
    getchar();
    return 0;
}
```

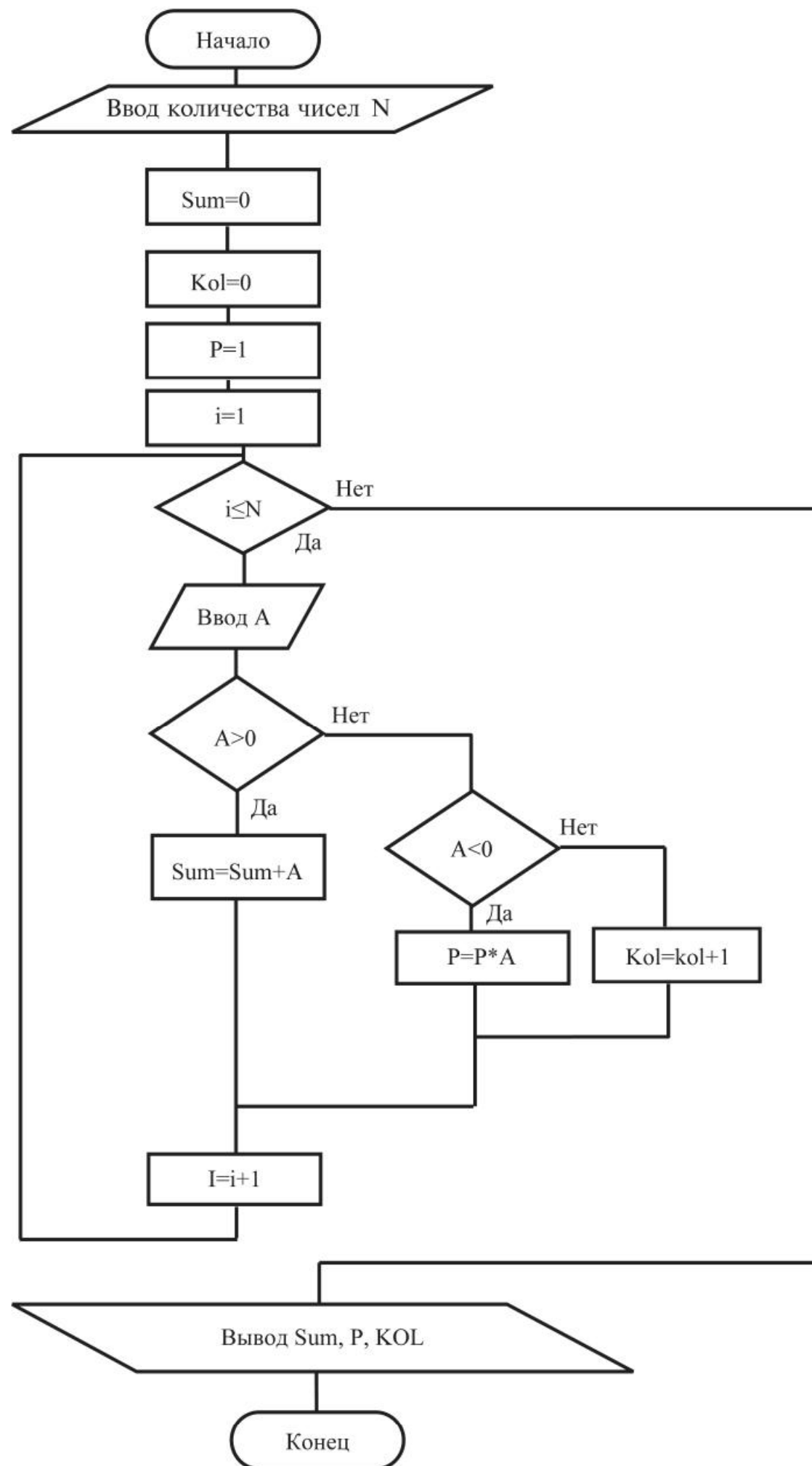
Пояснение. В программе девять раз подряд (последовательность состоит из девяти чисел) нужно выполнить два действия:

- ввести с клавиатуры очередное число в переменную A ;
- если оно равно нулю, то количество (Kol) увеличить на 1, (т. е. найдено еще одно нулевое число и общее количество нулевых чисел увеличивается на 1).

Для этого удобно использовать оператор цикла **for** (тело цикла будет выполняться 9 раз). Переменная Kol (для подсчета количества) должна иметь тип **int**.

Задача 5. Дана числовая последовательность. Найти сумму положительных, количество нулевых и произведение отрицательных элементов последовательности.

1. Схема алгоритма



Пояснение. В условии не сказано, сколько элементов (чисел) в последовательности, поэтому в начале программы с клавиатуры нужно ввести число элементов в последовательности (N). Пока с клавиатуры не вводились элементы последовательности, задаются начальные значения количеству ($Kol = 0$), сумме ($Sum = 0$) и произведению ($P = 1$). Затем с клавиатуры вводится 1-е число в переменную A . Если оно положительное ($A > 0$), то прибавляется к уже имеющейся сумме. Если оно отрицательное ($A < 0$), то умножается на уже имеющееся произведение. Если оно равно нулю ($A = 0$), то количество нулевых чисел увеличивается на 1. Аналогично с остальными числами. В результате в переменной Sum будет сформирована сумма положительных чисел, в переменной P — произведение отрицательных чисел, в переменной Kol — количество нулевых чисел.

Алгоритм решения — циклический, так как есть повторяющаяся последовательность действий. Параметр цикла (i) изменяется от 1 до N с шагом 1, так как одновременно является счетчиком введенных чисел.

2. Программа

```
#include <stdio.h>
int main() {
    double A, Sum, P;
    int i, N, Kol;
    printf("Сколько чисел будете вводить? ");
    scanf("%d", &N);
    Sum=0; //присвоение сумме (Sum) начального значения
    P=1; //присвоение произведению (P) начального значения
    Kol=0; //присвоение количеству (Kol) начального значения
    for (i=1; i<=N; i++){ //начало цикла
        printf("Введите число >");
        //вывод приглашения к вводу очередного числа
        scanf("%lf", &A); //ввод очередного числа
        if (A>0)
            S=S+A; //если A>0, вычисление Sum
        else if (A<0)
            P=P*A; //если A<0, вычисление P
        else Kol=Kol+1; //если A=0, вычисление Kol
    }
    printf("Sum=%lf P=%lf Kol=%d\n", Sum, P, Kol);
    //вывод на экран значения Sum, P, Kol
    getchar();
    return 0;
}
```

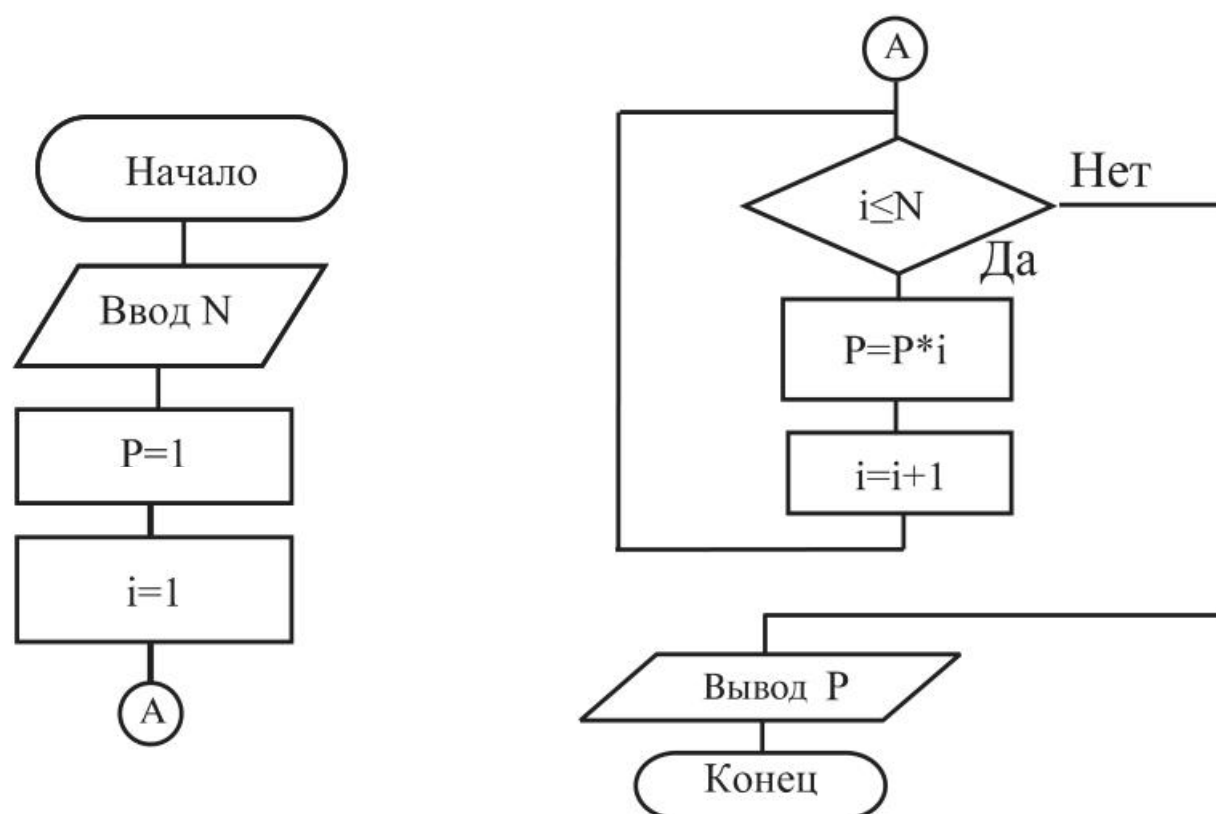
Пояснение. В программе N раз подряд (последовательность состоит из N чисел, переменная N типа **int**) выполняются следующие действия:

- ввод с клавиатуры в переменную A очередного элемента (числа) последовательности;
- если введенное число — положительное, то прибавить его к уже имеющейся сумме положительных чисел (`if (A>0) Sum=Sum+A`);
- если условие « $A>0$ » не выполняется (т. е. $A<0$ или $A=0$), то проверяется условие « $A<0$ » (`else if (A<0) ...`);
- если введенное число — отрицательное, то умножить его на уже имеющееся произведение отрицательных (`...if (A<0) P=P*A`);
- если введенное число равно нулю, то увеличить количество таких чисел на 1 (`else Kol=Kol+1`).

Для выполнения вышеописанных действий используется оператор **for**. Для вычисления требуемых суммы, произведения и количества в операторе **for** используется 2 условных оператора (один вложен в другой).

Задача 6. Вывести на экран значение $n!$, n — целое число, вводимое с клавиатуры.

1. *Схема алгоритма*



Пояснение. В математике есть функция $N!$ (читается «эн факториал»), где N — целое неотрицательное число. Функция $N!$ вычисляется по формуле: $N! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot N$, т. е. $N!$ — это произведение всех це-

лых положительных (натуральных) чисел от 1 до N (исключение $0! = 1$). Например: $3! = 1 \cdot 2 \cdot 3 = 6$, $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$.

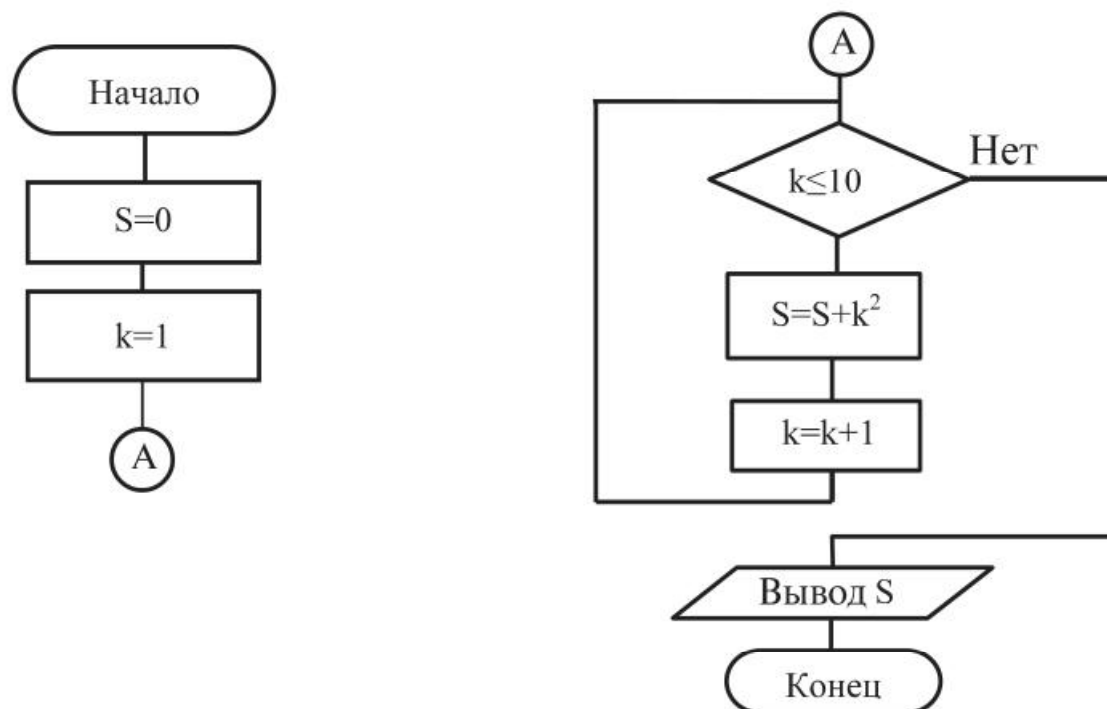
2. Программа

```
#include <stdio.h>
int main() {
    double P;
    int i, N;
    printf("Введите N   ");
    scanf("%d",&N);
    P=1;                               //присвоение произведению (P) начального значения
    for (i=1; i<=N; i++)                //в цикле
        P=P*i;                         //умножение старого значения P на значение i
    printf("P=%lf\n",P);                //вывод на экран значения P
    getchar();
    return 0;
}
```

Пояснение. Так как надо вычислить произведение N подряд идущих целых чисел (N вводится с клавиатуры), то удобно использовать оператор **for**: параметр цикла i будет одновременно очередным числом, на которое надо умножить уже имеющееся произведение P . Произведение P объявлено как переменная типа **double**, так как этот тип данных позволяет работать с большим диапазоном чисел, чем тип **int** (см. главу 3).

Задача 7. Вычислить $S = \sum_{k=1}^{10} k^2$.

1. Схема алгоритма



Пояснение. Запись $S = \sum_{k=1}^{10} k^2$ означает, что k принимает значения

1, 2, 3, ..., 10, а S — это сумма квадратов всех значений k , т. е. $S = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 + 8^2 + 9^2 + 10^2$. Вычисление S производится по стандартному алгоритму нахождения суммы, т. е. это алгоритм с циклической структурой (восходящий цикл).

2. Программа

```
#include <stdio.h>
int main() {
    double S;
    int k;
    S=0;                                     //присвоение сумме (S) начального значения
    for (k=1; k<=10; k++)                   //в цикле
        S=S+k*k;                           //увеличение старого значения S на значение k*k
    printf("S=%lf\n", S);                   //вывод на экран значения S
    getchar();
    return 0;
}
```

Пояснение. Так как k изменяется от 1 до 10 с шагом 1 и требуется для каждого значения k вычислить его квадрат и прибавить к сумме S , то удобно использовать оператор цикла **for** с параметром k (типа **int**).

Задача 8. С клавиатуры вводится последовательность чисел. Признак конца ввода — нуль. Найти среднее арифметическое отрицательных чисел последовательности. Использовать оператор **do-while**.

Примечание. Для того чтобы найти среднее арифметическое отрицательных элементов последовательности, надо вычислить отдельно сумму и количество отрицательных элементов последовательности, а затем поделить сумму на количество.

Программа

```
#include <stdio.h>
int main() {
    double X, S, Sr;
    int K;
    S=0;                                     //присвоение сумме (S) начального значения
    K=0;                                     //присвоение количеству (K) начального значения
    do {                                     //начало цикла
        printf("Введите число >");
        //вывод приглашения к вводу очередного числа
```

```

scanf ("%lf", &X);           //ввод очередного числа
if (X<0)
{S=S+X;                      //если X>0, вычисление S
K=K+1;                       //и вычисление K
}
}while (X!=0);               //цикл, пока введенное число не равно 0
if (K>0)
{Sr=S/K;                     //вычисление среднего
printf("среднее = %lf\n", Sr);
}
else printf("Отрицательных чисел не было\n");
getchar();
return 0;
}

```

Пояснение. В программе используются четыре переменные: X (типа **double**) — для хранения очередного элемента последовательности; S (того же типа, что и переменная X) — сумма отрицательных элементов; K (только типа **int**) — количество отрицательных элементов; Sr (типа **double**) — среднее арифметическое отрицательных элементов.

Ввод элементов последовательности осуществляется в операторе цикла **do-while** с постусловием « $X \neq 0$ » (ввод элементов прекращается, если с клавиатуры введено значение «нуль»). Для каждого элемента последовательности в теле цикла выполняются 2 действия:

- ввод значение элемента с клавиатуры в переменную X ;
- проверка условия « $X < 0$ »: если X — отрицательное, то прибавить его значение к уже имеющейся сумме отрицательных элементов (S) и увеличить на 1 количество таких элементов (K).

По окончании ввода элементов, если в последовательности были отрицательные элементы ($K > 0$), выполняется вычисление среднего арифметического отрицательных элементов ($Sr := S/K$) и вывод его на экран.

Задача 9. С точностью $\text{eps} = 10^{-6}$ вычислить сумму бесконечного ряда $S = \sum_{k=1}^{\infty} \frac{1}{k^2}$. Вычисления прекращаются, когда очередной член ряда $\frac{1}{k^2}$ станет меньше eps . Вывести на экран вычисленную сумму и количество просуммированных членов ряда. Использовать оператор цикла **while**.

Примечание. В математике под понятием «ряд» («сумма ряда») понимается бесконечная сумма, т. е. сумма, содержащая бесконечное количество членов:

$$S = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \dots$$

Ни в математике, ни в программировании невозможно определить численное значение суммы, содержащей бесконечное количество членов. Однако значение члена ряда $\frac{1}{k^2}$ уменьшается при увеличении k и при некотором значении k станет меньше ϵ . При добавлении последующих членов ряда значение суммы будет изменяться на очень маленькую (меньшую ϵ) величину, и, следовательно, подсчет данной суммы можно прекратить.

Программа

```
#include <stdio.h>
const double eps=1.e-6;
int main() {
double S, k;
S=0; //задание начального значения S
k=1; //задание начального значения k
while (1/(k*k)>eps) { //пока 1/(k*k) > eps
    S=S+1/(k*k); //увеличение S на 1/(k*k)
    k=k+1; //увеличение k на 1
}
printf("S=%10.6lf k=%1f\n",S,k-1);
getchar();
return 0;
}
```

Пояснение. В программе используются две переменные (S , k) и одна константа (ϵ). Константа ϵ равна 10^{-6} ($1.e-6$) **double** объявлена перед заголовком функции **main()** с помощью ключевого слова **const**. Все переменные объявлены как переменные типа **double**.

Начальное значение K равно «1» (1-е слагаемое готово для суммирования). Так как вычисление суммы S продолжается, *пока* значение очередного слагаемого больше заданной точности, то используется оператор цикла **while** с предусловием « $\frac{1}{k^2} > \epsilon$ ». Как только это условие становится ложным, происходит выход из цикла и вывод на экран значений S и $K-1$.

Задача 10. Пользователь вводит коэффициенты квадратного уравнения, программа вводит на экран корни уравнения. После этого спрашивает «выйти (y/n)?». В зависимости от ответа осуществляется выход из программы или вновь начинается ввод коэффициентов уравнения.

Программа

```
#include <stdio.h>
#include <math.h>
int main() {
double a,b,c,d,x,x1,x2;
char ch;
do{
printf("Введите коэффициенты квадратного уравнения
a,b,c\n");
scanf("%lf%lf%lf",&a,&b,&c);
if (a==0){
if (b==0){
if (c==0) printf("x - любое\n");
else printf("корней нет\n");
}
else { x=-c/b;
printf("x=%lf\n",x);
}
}
else {
d=b*b-4*a*c;
if (d>=0){
x1=(-b-sqrt(d))/(2*a);
x2=(-b+sqrt(d))/(2*a);
printf("x1=%lf x2=%lf\n",x1,x2);
}
else printf("корней нет \n");
}
printf("продолжить (y/n)? ");
ch=getchar();
}while ((ch=='y')||(ch=='Y'));
return 0;
}
```

Тесты

1. Какое значение будет храниться в переменной X после выполнения следующего цикла?

```
...  
X=0;  
for (i=3; i>=1; i--)  
    X=X+I;  
...
```

- а) 6;
- б) 3;
- в) 0.

2. Что нужно изменить в следующем фрагменте программы, чтобы в переменной S сформировалась сумма введенных значений?

```
...  
1. for (i=1; i<=10; i++)  
2.     {  
3.         scanf("%lf",&A);  
4.         S=S+A;  
5.     }  
...
```

- а) перед 1-й строкой вставить строку: S=1;
- б) перед 1-й строкой вставить строку: S=0;
- в) перед 3-й строкой вставить строку: S=0.

3. Что нужно изменить в следующем фрагменте программы, чтобы в переменной P сформировалось произведение введенных значений?

```
...  
1. P=0;  
2. for (i=1; i<=10; i++)  
3.     {  
4.         scanf("%lf",&A);  
5.         P=P*A;  
6.     }  
...
```

- а) в 1-й строке изменить начальное значение P на 1;
- б) между 3-й и 4-й строками вставить строку: P=1;
- в) ничего изменять не надо, произведение и так сформируется.

4. Какие значения примут переменные S, N и P после выполнения следующего фрагмента программы, если в переменную Y поочередно ввести следующие значения: -2; 0; -5; 6; 0?

```
...  
S=0; N=0; P=1;  
for (i=1; i<=5; i++)  
{  
scanf("%d",&Y);  
if (Y>0)  
    S=S+Y;  
else if (Y=0)  
    N=N+1;  
else    P=P*Y;  
}  
...
```

- а) S= -7, N=1, P=0;
- б) S= 6, N=2, P=10;
- в) S= -1, N=4, P=10.

5. Чему будет равно значение переменной N после выполнения фрагмента программы?

```
...  
N=1;  
While (N<4.5)  
    N=N+0.2;  
...
```

- а) 4.5;
- б) 4.6;
- в) 1.

6. Что будет в результате выполнения следующего фрагмента программы?

```
...  
N=1; S=0;  
While N<5  
    S=S+N*N;  
...
```

- а) S=0;
- б) в S будет посчитана сумма квадратов чисел от 1 до 5:
 $1^2 + 2^2 + 3^2 + 4^2 + 5^2$;
- в) в S будет посчитана сумма квадратов чисел от 1 до 4: $1^2 + 2^2 + 3^2 + 4^2$;
- г) произойдет закливание и переполнение переменной S.

7. Какие значения будут выведены на экран в результате выполнения следующего фрагмента программы?

```
.....
x=0;
while (x<=10)
{
    if (x<=5)
        y=2*x
    else y=3*x;
    printf("%4d ",y);
    x=x+3;
}
...
```

- a) 0 3 6 9 12;
- б) 0 6 18 27;
- в) 2 4 6 8 10.

8. Чему будет равно значение переменной N после выполнения фрагмента программы?

```
...
N:=1;
do
    N:=N+0.1;
while (N<10.75);
...
```

- a) 1.1;
- б) 10.8;
- в) 1.

9. Какие значения будут выведены на экран в результате выполнения следующего фрагмента программы?

```
...
x=0;
do{
    if (x<5)
        y=2*x;
    else y=3*x;
    printf("%4d ",y);
    x=x+3;
}while (x<10);
...
```

- а) 0 2 4 9 12 15;
- б) 0 6 18 27;
- в) 3 6 9.

Задания

Для каждой задачи составить схему алгоритма и написать программу.

1. Построить таблицу $Y = 2X$ при $X = 1, 2, \dots, 9, 10$.

X	Y
1	2
2	4
...	...
9	18
10	20

2. Даны восемь чисел. Найти сумму положительных чисел.
3. Даны 10 чисел. Найти произведение отрицательных чисел.
4. Даны 12 чисел. Найти произведение ненулевых чисел и количество нулей.
5. Даны восемь чисел. Найти сумму положительных и произведение отрицательных чисел.
6. Пять человек садятся в лодку, грузоподъемность которой 300 кг. Посадка в лодку производится постепенно — один пассажир за другим (вес пассажиров ввести с клавиатуры). Подсчитать количество человек с весом больше 60 кг и вывести на экран сообщение о результатах посадки: «Посадка прошла успешно!» или «Лодка потонула».
7. Ввести с клавиатуры последовательность из N элементов. Найти количество ненулевых элементов и сумму элементов, значения которых меньше двух. Вывести найденные сумму и количество (если они не были сформированы, вывести соответствующее сообщение).
8. Ввести с клавиатуры последовательность из N элементов. Найти произведение отрицательных элементов и сумму элементов, значения которых больше пяти. Вывести найденные сумму и произведение (если они не были сформированы, вывести соответствующее сообщение).
9. Дана числовая последовательность из N элементов. Вывести номера всех нулевых элементов.

10. Дана числовая последовательность из N элементов. Вывести номер последнего нуля.

11. Дана числовая последовательность из N элементов. Вывести номер 1-го нуля или вывести сообщение: «Нули отсутствуют».

12. Составить программу, которая проверяет, является заданное число совершенным. Совершенным называется натуральное число, равное сумме всех своих делителей (за исключением самого себя). Например: $28 = 1 + 2 + 4 + 7 + 14$.

13. Дана последовательность из N чисел. Подсчитать количество чисел, кратных пяти, и сумму чисел, имеющих четные номера в последовательности.

14. В компьютер вводятся по очереди данные о росте N учащихся класса. Определить средний рост учащихся класса.

15. В 1202 г. итальянский математик Леонард Фибоначчи подсчитывал, на сколько увеличивается число кроликов в хозяйстве каждый год. При этом он получил последовательность такого вида: 1, 1, 2, 3, 5, 8, 13, 21, 34 Написать программу, которая для заданного числа N выводит N членов последовательности Фибоначчи.

16. Каждая бактерия делится на две в течение одной минуты. В начальный момент времени имеется одна бактерия. Составить программу для расчета количества бактерий через заданное целое количество минут.

17. Вычислить произведение N членов последовательности вида 1, $1 + 2$, $1 + 2 + 3$, $1 + 2 + 3 + 4$,

18. Найти сумму членов ряда, предварительно определив формулу общего члена: $S = 3/2 + 4/5 + 5/8 + 6/11 + \dots + 22/59$.

19. Найти сумму, предварительно определив формулу общего члена: $S = 0 + 1/5 + 2/7 + 3/9 + \dots + 16/35$.

20. Вычислить сумму нечетных членов ряда $a_i = 1/i^2$, $1 \leq i \leq N$. Число N ввести с клавиатуры. Вывести вычисленную сумму и количество просуммированных членов ряда.

21. Вычислить значения $y = x^2$ на отрезке $0 \leq x \leq 1$ с шагом $\Delta x = 0,1$. Оформить вывод значений функции в виде вертикальной и горизонтальной таблиц.

22. Вычислить $y = x/5$, $0,5 \leq x \leq 5$, $\Delta x = 0,5$. Вывести все значения x и y в виде вертикальной таблицы. Использовать оператор while.

23. Вычислить значения $y = \begin{cases} x^2, & \text{если } x < 15; \\ x^3, & \text{если } x \geq 15 \end{cases}$ на отрезке $0 \leq x \leq 30$

с шагом $\Delta x = 1$. Вывести все значения x и y в виде вертикальной и горизонтальной таблиц. Использовать оператор while.

24. С помощью `while` написать программу определения идеального веса для взрослых людей по формуле: $\text{Ид. вес} = \text{рост} - 100$. Выход из цикла: значение роста = 250 см.

25. С помощью `while` написать программу определения суммы всех нечетных чисел в диапазоне от 1 до 99 включительно.

26. Вычислить $y = x/5$, $0,5 \leq x \leq 5$, $\Delta x = 0,5$. Вывести все значения x и y в виде горизонтальной таблицы. Использовать оператор `repeat`.

27. С помощью `repeat` написать программу, которая требует ввести пароль, например 111, и если пароль правильный, то заполняет все строками экрана сообщением «Молодец!!!». Если после пятой попытки пароль все равно неверен, выйти из программы.

28. Вычислить значения y на отрезке $0 \leq x \leq 2$ с шагом $\Delta x = 0,1$. Вывести все значения x и y в виде вертикальной и горизонтальной таблиц. Решить задачу двумя способами (с помощью оператора `while` и с помощью оператора `repeat`).

$$y = \begin{cases} 3x - 4, & \text{если } x^2 < 0,5; \\ 10x, & \text{если } x^2 \geq 0,5. \end{cases}$$

29. Дано: $i = 138$. Вычислить значения x на отрезке $-2 \leq m \leq 2$ с шагом $\Delta m = 0,2$.

$$x = \begin{cases} m^2 + i, & \text{если } m \geq 0,89; \\ 100m - 1, & \text{если } m < 0,89. \end{cases}$$

Вывести все значения m и x . Решить задачу тремя способами, используя три различных оператора цикла.

30. Вычислить значения функции $y(x)$ на отрезке $-3 \leq x \leq 8$ с шагом $\Delta x = 1$.

$$y(x) = \begin{cases} 18x, & \text{если } x > 5; \\ 9x + 10, & \text{если } 2 \leq x \leq 5; \\ -12x, & \text{если } x < 2. \end{cases}$$

Вывести все значения x и y в виде горизонтальной таблицы. Решить задачу тремя способами, используя три различных оператора цикла.

31. Решить задачу тремя способами, используя различные операторы цикла. (Написать три разные программы или объединить все три оператора цикла в одну программу, используя меню.) Дано $n = 100$, $b = 3,8$. Вычислить значения T на отрезке $90 \leq x \leq 120$ с шагом 10.

$$T = \begin{cases} nb^2, & \text{если } x \leq 100; \\ n/b, & \text{если } x > 100. \end{cases}$$

32. Дано $i = 138$. Вычислить значения x на отрезке $-2 \leq m \leq 2$ с шагом $\Delta m = 0,2$.

$$x = \begin{cases} m^2 + i, & \text{если } m \geq 0,89; \\ 100m - 1, & \text{если } m < 0,89. \end{cases}$$

(Дополнительно.) Вывести все значения m и x . Для решения задачи организовать меню:

- 1) цикл с предусловием;
- 2) цикл с постусловием;
- 3) цикл с параметром.

33. Вывести на экран значение одной из строк формулы (N вводится с клавиатуры):

$$S = \begin{cases} 2 * 4 * \dots * N, & \text{если } N \text{ — четное;} \\ 1 * 3 * \dots * N, & \text{если } N \text{ — нечетное.} \end{cases}$$

34. Найти среднее арифметическое чисел, введенных с клавиатуры. Признак конца ввода — нуль. Определить количество введенных чисел.

35. Дана арифметическая прогрессия из N членов. Число элементов N , 1-й элемент a_1 и разность d задаются с клавиатуры. Вывести все члены прогрессии на экран и вычислить сумму элементов прогрессии, значение которых меньше пяти. Решить задачу тремя способами.

36. Дана геометрическая прогрессия из N членов. Число элементов N , 1-й элемент b_1 и знаменатель q задаются с клавиатуры. Вывести все члены прогрессии на экран и вычислить количество элементов прогрессии, меньших 1. Решить задачу тремя способами.

37. Дана арифметическая прогрессия из N элементов, 1-й элемент прогрессии a_1 и разность d задаются случайным образом. Вывести все члены прогрессии на экран и найти произведение элементов прогрессии, значение которых меньше 30. Использовать оператор While.

38. Дана арифметическая прогрессия из N элементов, 1-й элемент прогрессии a_1 и разность d задаются случайным образом. Вывести все члены прогрессии на экран и найти произведение элементов прогрессии, значение которых меньше 30. Решить задачу тремя способами.

39. Дана арифметическая прогрессия из N членов, 1-й элемент a_1 и разность d задаются случайным образом. Вывести все члены прогрессии на экран и вычислить произведение элементов прогрессии, значение которых больше 5. Решить задачу тремя способами.

40. Дана геометрическая прогрессия из N членов, 1-й элемент b_1 и знаменатель q задаются случайным образом. Вывести все члены прогрессии на экран и вычислить сумму элементов прогрессии, больших -4 . Решить задачу тремя способами.

41. Предприниматель, начав дело, взял кредит размером k рублей под p процентов годовых и вложил его в свое дело. По прогнозам, его дело должно давать прибыль r рублей в год. Сможет ли он накопить сумму, достаточную для погашения кредита, и если да, то через сколько лет?

42. Корень некоторого уравнения находится последовательными приближениями по формуле $x_{n+1} = \frac{2 - x_n^3}{5}$. Написать программу для нахождения такого приближения корня, при котором разность по модулю между двумя соседними приближениями не превосходит 10^{-5} , а начальное приближение $x_0 = 1$.

43. С помощью Repeat написать программу-фильтр, которая вводит любые символы, но комментирует только буквы русского алфавита. Завершение работы программы — по нажатию буквы «Я».

44. Составить программу вычисления степени числа A с натуральным показателем n . Записать варианты программы с разными видами циклов while, repeat, for.

45. Вычислить сумму 50 членов ряда $a_i = 1/i$, $i = 1, 2, \dots$. Если в процессе вычислений очередной член ряда окажется меньше epsilon, то дальнейшие вычисления прекратить. Число epsilon ввести с клавиатуры. Распечатать вычисленную сумму и количество просуммированных членов ряда.

46. Ввести два целых числа. Вывести в порядке убывания все числа, лежащие между ними, и количество этих чисел. Каждое третье число не печатать и не учитывать.

47. Ввести с клавиатуры 20 чисел и найти среднее арифметическое всех четных чисел и количество таких чисел.

48. Ввести с клавиатуры N чисел в диапазоне от 0 до 10. Число N ввести с клавиатуры. Найти количество и произведение всех чисел, меньших 5.

49. Ввести с клавиатуры 15 чисел в диапазоне от 0 до 10. Найти количество и сумму всех чисел, меньших 5, но больших 3.

50. Найти все положительные четырехзначные числа, для которых выполняется условие: $AB - CD = A + B + C + D$.

51. В соревнованиях по прыжкам в длину принимают участие 20 спортсменов. Определить, сколько из них выполнило норму, если она составляет k метров.

52. Вычислить и вывести значение $y = \cos(x)$ для x от 0, $\Delta x = 0,05$. Вычисления прекратить при $y \geq z$ (z ввести с клавиатуры, $-1 < z < 1$). Вывести количество вычисленных значений.

53. Вычислить значение $y = \sin(x)$ для x от 0 до 1, $\Delta x = 0.1$. Вывод результата производить для каждой второй точки.

54. Найти максимальное число среди чисел, введенных с клавиатуры. Признак конца ввода — 0. Вывести количество введенных чисел.

55. Ввести 25 целых чисел, найти минимальное число среди нечетных чисел и количество таких чисел.

56. Вычислить значение $y = \cos(x)$ для x от 0 до 2, $\Delta x = 0.1$. Вывод результата производить для каждой третьей точки.

57. Ввести 16 целых чисел и найти максимальное число среди нечетных чисел и количество таких чисел.

58. Вычислить и вывести значение $y = \sin(x)$ для x от 0, $\Delta x = 0,05$. Вычисления прекратить при $y \geq 0,5$. Вывести количество вычисленных значений.

59. Ввести два целых числа. Вывести в порядке возрастания все числа, лежащие между ними и количество этих чисел. Каждое четвертое число не печатать и не учитывать.

60. Вычислить сумму N членов ряда $a_i = 1/i$, $1 \leq i \leq N$. Число N ввести с клавиатуры. Если в процессе вычислений очередной член ряда окажется меньше 10^{-3} , то дальнейшие вычисления прекратить. Вывести вычисленную сумму и количество просуммированных членов ряда.

61. При помощи датчика случайных чисел ввести N чисел в диапазоне от 0 до 10. Число N ввести с клавиатуры. Найти количество и произведение всех чисел, меньших 3.

62. В 1985 г. урожай ячменя составил 20 ц с гектара. В среднем каждые 2 года за счет применения передовых агротехнических приемов урожай увеличивается на 5 %. Определить, через сколько лет урожайность достигнет 25 ц с гектара.

63. Найдите наименьшее трехзначное число, сумма кубов цифр которого равна 730.

64. Найдите три натуральных числа x , y , z , удовлетворяющих условию $15x + 20y + 30z = 270$.

65. Составить программу получения в порядке убывания всех делителей заданного числа N .

66. Ввести 23 целых числа, найти минимальное число среди четных чисел и количество таких чисел.

67. Ввести 12 чисел и найти среднее арифметическое всех нечетных чисел и количество таких чисел.

68. При помощи датчика случайных чисел ввести 100 чисел в диапазоне от 0 до 10. Найти среднее арифметическое и количество всех чисел, меньших 1 или больших 9.

69. При помощи датчика случайных чисел ввести 100 целых чисел и найти максимальное число среди четных чисел и количество таких чисел.

70. Вывести на экран все четные числа от 1 до 100 включительно.

71. Составить программу вычисления суммы квадратов чисел от 1 до введенного целого числа n .

72. Вычислить сумму квадратов N четных натуральных чисел.

73. Составить программу вычисления суммы всех двузначных чисел.

74. Найти наибольший общий делитель двух натуральных чисел.

75. Найти наименьшее общее кратное двух натуральных чисел.

76. Подсчитать количество цифр вводимого целого неотрицательного числа. (Можно использовать операцию целочисленного деления для последовательного уменьшения числа на один разряд.)

77. Вычислить сумму N нечетных натуральных чисел.

78. Вычислить $x = 2^{2^n}$, $n = \{0...10\}$.

79. Вычислить $x = (1 + 2) * (1 + 2 + 3) * ... * (1 + 2 + ... + 10)$.

80. Для заданного n вычислить сумму $S = \sum_{i=1}^N (-1)^{i+1} / (2 \cdot i - 1)$, где $i = 1, 2, 3, ..., N$.

2, 3, ..., N .

81. Вывести на экран все простые числа, не превосходящие заданного N . Простым числом называется натуральное число больше единицы, имеющее только 2 делителя: единицу и само себя.

82. С клавиатуры последовательно вводятся координаты N точек. Определить, сколько из них попадает в круг радиусом R с центром в точке (a, b) .

83. Вычислить $S(x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^{2k+1}}{(2k+1)!}$, $|x| < 1$.

84. Вычислить $S(x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^{2k}}{(2k)!}$.

85. Методом деления отрезка пополам и методом итераций найти приближенное значение корня уравнения $x^5 - x - 0,2 = 0$ на интервале $[0,9, 1,1]$. Абсолютная погрешность не превышает 0,0001. Сравнить методы вычисления.

86. С заданной точностью ϵ вычислить $S = \sum_{k=1}^{\infty} \frac{k+4}{k^3+7}$.

Контрольные вопросы

1. Что такое «алгоритм циклической структуры»? Чем он отличается от линейного и разветвленного алгоритмов? Приведите примеры.
2. Запишите в общем виде оператор цикла с предусловием. Какие ключевые слова при этом используются?
3. Что такое <условие продолжения цикла> операторе while? Какие требования к нему предъявляются?
4. Может ли в качестве <условия продолжения цикла> использоваться сложное условие? Приведите примеры.
5. Может ли <оператор> в цикле while быть блоком? Приведите пример.
6. Объясните механизм работы оператора while на примерах.
7. Из чего состоит итерация для оператора while? Объясните на примере.
8. Напишите пример <условия продолжения цикла> для оператора while, при котором тело цикла не выполнится ни разу. Позволит ли компилятор запустить такую программу на выполнение?
9. Напишите пример <условия продолжения цикла> для оператора while, при котором тело цикла будет выполняться бесконечно (произойдет закливание программы). Позволит ли компилятор запустить такую программу на выполнение?
10. Запишите в общем виде оператор цикла с постусловием. Какие ключевые слова при этом используются?
11. Может ли <оператор> в цикле do-while быть блоком? Приведите пример.
12. Объясните механизм работы оператора do-while на примерах.
13. Из чего состоит итерация для оператора do-while? Объясните на примере.
14. Запишите в общем виде оператор цикла for. Какие ключевые слова при этом используются?
15. Может ли <оператор> в цикле for быть блоком? Приведите пример.
16. Объясните механизм работы оператора for на примерах.
17. Из чего состоит итерация для оператора for? Объясните на примере.
18. Что такое тело цикла? Объясните на примерах для каждого оператора цикла.
19. Сформулируйте в общем виде определение «цикла с предусловием».
20. Сформулируйте в общем виде определение «цикла с постусловием».
21. Чем цикл с предусловием отличается от цикла с постусловием?

Глава 6

ПРЕПРОЦЕССОРНЫЕ СРЕДСТВА

Препроцессор — это программа, которая обрабатывает исходный код, написанный программистом, подготавливая его для последующей компиляции. Препроцессор выполняет различные действия, например заменяет комментарии пустыми строками, включает содержимое файлов в текст программы, обрабатывает макроопределения, обеспечивает условную компиляцию и т. д. Команды препроцессора называются директивами, начинаются с символа «#» и могут размещаться в любом месте программы.

При каждом запуске компилятора сначала запускается препроцессор, который ищет команды препроцессора. При выполнении любой из этих команд в текст исходного кода вносятся соответствующие изменения и создается новый временный файл исходного кода. Директивы препроцессора могут появиться в любом месте исходного кода, но они применимы только к его оставшейся части (расположенной после директивы).

6.1. Директива `#include`

Директива

```
#include <имя_файла>
```

вставляет содержимое указанного файла в ту точку исходного файла, где она записана. Если имя файла указано в угловых скобках, то поиск указанного файла осуществляется в стандартных каталогах для включаемых файлов, если вместо угловых скобок используются кавычки (" "), то в этом случае поиск файла ведется в каталоге, содержащем исходный файл, а затем уже в стандартных каталогах. Заголовочные файлы обычно имеют расширение `*.h` и содержат объявления и

определения типов, констант, встроенных функций, шаблонов, комментариев и директивы препроцессора.

Например, директива препроцессора

```
#include <stdio.h>
```

вставляет содержимое заголовочного файла *stdio.h* стандартной библиотеки ввода-вывода в то место программы, где эта директива находилась до работы препроцессора (обычно в начале программы).

Директива препроцессора

```
#include "myfile.h"
```

вставляет содержимое созданного пользователем и находящегося в том же каталоге, где и программа, файла *myfile.h* в то место программы, где эта директива находилась до работы препроцессора.

6.2. Директива #define

Директива *#define* определяет подстановку в тексте программы

```
#define <текстовое_имя> <текст_подстановки>
```

В тексте исходного файла, следующем за директивой *#define*, все вхождения <текстового_имени> заменяются на <текст_подстановки>.

Например, в результате выполнения директивы

```
#define PI 3.1415926
```

текстовое имя *PI* в нижеследующем тексте программы будет заменено подстановкой *3.1415926*.

```
#include <stdio.h>
#define PI 3.1415926
int main()
{
    double S, R;
    printf("Введите радиус круга ");
    scanf("%lf",&R);
    S=PI*R*R;
    printf("Площадь круга = %lf\n",S);
    getchar();
    return 0;
}
```


Тогда в дальнейшей программе можно использовать подстановку COUNT. А в случае изменения числа школьников достаточно будет в одной строчке программы изменить старое значение «21» на новое, например 25, не редактируя остальной код.

6.3. Директивы условной компиляции

Директивы условной компиляции применяются для того, чтобы исключить компиляцию отдельных частей программы. То есть данные директивы указывают препроцессору, какой текст программы необходимо включить в дальнейшую компиляцию. Это бывает полезно, например, при отладке, при поддержке нескольких версий программы для различных платформ или адаптации программы к текущей ситуации (например, к определенному типу процессора). Полный вариант использования директив условной компиляции имеет следующий формат:

```
#if <константное_выражение_1>
    <текст_программы_1>
#elif <константное_выражение_2>
    <текст_программы_2>
#elif <константное_выражение_3>
    <текст_программы_3>
...
#else
    <текст_программы_4>
#endif
```

На первом шаге препроцессор проверяет <константное_выражение_1> в директиве #if. Если выражение ложно, то проверяется <константное_выражение_2>, расположенное в директиве #elif, и т. д. Если все выражения окажутся ложными, то в компилируемый текст будет включен только <текст_программы_4>, расположенный в ветке #else.

Если же хотя бы одно из выражений будет истинно, то в компилируемый текст будет включен код, расположенный непосредственно за этим выражением. В этом случае все нижеследующие директивы не будут рассматриваться препроцессором.

Например:

```
#define N 1                                //задание постановки N равной 1
.....
#if N==1                                    //если подстановка N равна 1
    printf("N=1\n");                        //вывести на экран текст N=1
    //вывести на экран сообщение, что должно работать меню File
    printf("Должно выполняться меню File\n");

<команды меню File>                        //отлаживаемые команды меню File
#elif N==2                                  //если подстановка N равна 2
    printf("N=2\n");                        //вывести на экран текст N=2
    //вывести на экран сообщение, что должно работать меню Edit
    printf("Должно выполняться меню Edit\n");
<команды меню Edit>                        //отлаживаемые команды меню Edit
#elif N==3                                  //если подстановка N равна 3
    printf("N=3\n");                        //вывести на экран текст N=3
    //вывести на экран сообщение, что должно работать меню View
    printf("Должно выполняться меню View\n");
<команды меню View>                        //отлаживаемые команды меню View
#else                                       //если подстановка N не равно ни 1, ни 2, ни 3
    //вывести на экран сообщение, что это не команды меню
    printf("Это не команды меню\n");
<команды завершения программы>
    //отлаживаемые команды завершения программы

#endif
```

Директива `#endif` сообщает препроцессору об окончании блока `#if`. То есть каждой директиве `#if` в исходном файле должна соответствовать закрывающая директива `#endif`. Между директивами `#if` и `#endif` могут использоваться директивы `#elif` и не более одной директивы `#else`. Директива `#else`, если она есть, должна быть последней директивой перед директивой `#endif`.

В случае если все константные выражения ложны или отсутствуют директивы `#elif`, то в компилируемый текст препроцессор выбирает текст программы, следующий после директивы `#else`. Если директива `#else` отсутствует, в компилируемый текст не будет включено никакого кода программы.

В качестве константного выражения может выступать любое выражение, результат вычисления которого должен быть известен до начала компиляции программы. То есть препроцессор должен определить, будет ли значение выражения являться «истиной» или «ложью».

Например, следующая директива всегда включит <операторы> в компилируемую программу, так как выражение $2 > 1$ является истинным:

```
#if 2>1
    <операторы варианта 2>1;>
#endif
```

Аналогичным образом будет работать код:

```
#if 1
    <операторы варианта 1;>
#endif
```

Следующий пример демонстрирует, как при необходимости исключить компиляцию части кода программы:

```
#if 0                                //выражение в этой ветке всегда ложно
    int i, j;                        //а так как отсутствует ветка #else
    printf("i = ");
                                     //то весь код от директивы #if до #endif
    scanf("%d", &i)                  //не будет включен в компилируемый текст
    printf("j = ");
    scanf("%d", &j);
#endif
```

В этом примере весь текст, находящийся между директивами `#if` и `#endif`, не будет включен в компилируемый текст.

Иногда удобно использовать подстановки, введенные директивой `#define`. Рассмотрим пример:

```
#define header 1
#if header
    printf("Программа поиска максимума двух чисел.\n");
    printf("Автор программы Progger.\n");
#endif
...
```

В директиве `#if` препроцессор проверяет истинность выражения `header` и в случае его истинности код программы, выводящий на экран название и автора программы, будет включен в компилируемый текст. А так как ранее с помощью директивы `#define` текст `header` заменяется на подстановку `1`, то выражение, находящееся в ветке `#if`,

будет истинно, а следовательно, при работе программы на экран будет выводиться название и автор программы. Если потребуется убрать вывод этой информации на экран, достаточно будет изменить первую строчку примера на:

```
#define header 0
```

Еще раз отметим тот факт, что рассмотренные директивы не управляют компиляцией, а указывают препроцессору, какой код необходимо включить в компилируемую программу. Таким образом, компилятор будет обрабатывать не весь текст программы, а только его часть, выделенную программистом с помощью директив условной компиляции.

Дополнительные возможности условной компиляции предоставляют директивы `#ifdef` и `#ifndef`, позволяющие управлять компиляцией в зависимости от того, определена ли с помощью директивы `#define` указанная в них подстановка (хотя бы как пустая строка). Эти директивы имеют следующий формат.

<code>#ifdef подстановка</code>	<code>#ifndef подстановка</code>
<code><текст_программы;></code>	<code><текст_программы;></code>
<code>#endif</code>	<code>#endif</code>

Разница между ними заключается в том, что директива `#ifdef` проверяет наличие подстановки (дословно можно прочесть так: если подстановка определена, то включить следующий текст программы в компиляцию), а директива `#ifndef` проверяет отсутствие подстановки (дословно можно прочесть так: если подстановка не определена, то включить следующий текст программы в компиляцию).

Наиболее часто директива `#ifndef` используется для предотвращения многократного включения заголовочных файлов. В сложных проектах может возникнуть ситуация, когда содержимое одного заголовочного файла включается в другие заголовочные файлы, которые, в свою очередь, включаются в третьи файлы. В результате образовавшейся цепочки содержимое заголовочных файлов может быть включено в компилируемый текст несколько раз. Наличие в коде этих файлов объявления переменных приведет к возникновению ошибки компилятора, указывающей на «переопределение» переменных. Избежать возникновения этой ошибки можно с использованием директив условной компиляции `#ifndef`. Приведем пример кода, обеспе-

чивающего лишь однократное включение заголовочного файла "myfunc.h" в компилируемый текст.

```
#ifndef _myfunc_                //если _myfunc не определено
#define _myfunc_                //то включить этот код
    текст_программы;           //в компилируемый текст
#endif                          //окончание ifndef
```

При первой обработке препроцессором этого кода подстановка `_myfunc_` еще не определена, а следовательно, код программы, находящийся между директивами `#ifndef` и `#endif`, включается в компилируемый текст. Далее выполняется директива `#define` и объявляется подстановка `_myfunc_`. Таким образом, при последующих обработках препроцессором этого кода условие в директиве `#ifndef` не будет выполнено и код программы не будет включаться в компилируемый текст. То есть код программы, находящийся между директивами `#ifndef` и `#endif`, будет включен в компилируемый текст только один раз.

В заголовочные файлы обычно помещают объявления различных констант, переменных, функций, пользовательских типов данных, которые будут использоваться во всем проекте. Добавление заголовочного файла осуществляется аналогично добавлению файла с исходным кодом (см. главу 2, параграф 2.4), только размещать эти файлы нужно в папке Header Files (см. рис. 2.6) и при выборе типа файла необходимо указать Header File (.h) (см. рис. 2.7).

Рассмотрим простой пример использования заголовочного файла в проекте.

```
//заголовочный файл myheader.h
#ifndef _MYHEADER_              //проверяем
#define _MYHEADER_              //если еще не определяли,
                                //то определяем

    const int M = 15;
    int count = 0;
#endif                          //конец ifndef

//основной файл программы main.cpp
#include <stdio.h>
#include "myheader.h"
    //включаем содержимое заголовочного файла в текст программы
int main()
{
    printf("Значение константы M = %d\n", M);
```



```
    printf("Значение счетчика count  = %d\n", count);  
    return 0;  
}
```

В этом примере проект состоит из двух файлов `myheader.h` и `main.cpp`. Заголовочный файл содержит защиту, обеспечивающую только однократную компиляцию его содержимого. В основной программе директивой `#include` содержимое `*.h` файла включается в код программы. После этого в программе становятся доступны константа `M` и переменная `count`, объявленные в заголовочном файле.

Рассмотрим пример, который выводит на экран номер демоверсии программы.

```
#define DEMO                                //определение подстановки DEMO  
#define NUMBER 1.0                         //определение номера версии программы  
#include <stdio.h>  
int main()  
{  
#ifdef DEMO                                //если программа работает как демоверсия  
                                           //то сообщаем об этом  
    puts("Программа работает в демоверсии\n");  
#else  
    //иначе сообщаем о полноценном режиме функционирования программы  
    puts("Программа работает в полноценном режиме\n");  
#endif  
#ifndef NUMBER  
    //если определен номер версии программы, то выводим номер на экран  
    printf("Номер версии %3.1f\n", NUMBER);  
#endif  
    return 0;  
}
```

С помощью директив `#define` вводятся две подстановки `DEMO` (указывает на то, что программа функционирует в демонстрационном режиме) и `NUMBER` (указывает номер версии программы). В директиве `#ifndef` происходит проверка определения подстановки `DEMO`, и в случае ее наличия соответствующее сообщение выводится на экран. Если такая подстановка не определена, то в компилируемый текст будет включена строка кода, находящаяся после директивы `#else`. В случае если задан номер версии программы, то в компилируемый текст будет включена функция `printf`, выводящая этот номер на экран.

Иногда в программе требуется отменить ранее введенные подстановки. Для этого можно воспользоваться директивой `#undef`:

```
#undef текстовое_имя
```

Директива отменяет ранее введенную подстановку с именем «текстовое_имя». То есть все последующие вхождения «текстового_имени» будут игнорироваться препроцессором. Рассмотрим использование этой директивы на примере:

```
#define N 5                                //вводим подстановку с именем N
#include <stdio.h>
int main()
{
    #ifdef N                                //если подстановка с именем N определена
        printf("N = %d\n", N);             //то выводим значение N на экран
    #else
        printf("Подстановка N не определена!\n");
                                           //сообщаем, что N — не определено
    #endif
    #undef N                                //отменяем подстановку с именем N
    #ifdef N                                //проверяем, есть ли такая подстановка
        printf("N = %d\n", N);             //если есть, то выводим значение N
    #else
        printf("Подстановка N не определена!\n");
                                           //сообщаем, что N — не определено
    #endif
    return 0;
}
```

В начале программы вводится подстановка с именем `N`. Далее при первой проверке наличия этой подстановки условие в директиве `N` будет истинным и на экран будет выведено значение `N`. После этого с использованием директивы `#undef` отменяется подстановка с именем `N`. При следующей проверке условие директивы `#ifdef` будет ложным и на экран будет выведено сообщение о том, что подстановка с именем `N` не определена.

Результат работы программы будет выглядеть следующим образом:

```
N = 5
Подстановка N не определена!
```

Директиву `#undef` удобно использовать для изменения значения ранее введенной подстановки. Сделать это можно следующим образом.

```
#define N 5                                //подстановка N имеет значение 5
#include <stdio.h>
int main()
{
    int mas[N];                            //объявляем массив из 5 элементов
    ...                                    //выполняем необходимые действия с этим массивом
    ...                                    //после окончания работы с массивом mas
#undef N                                    //отменяем подстановку N
#define N 10
    //задаем новую подстановку с таким же именем, но с другим значением
    int newmas[N];
    ...                                    //объявляем новый массив с другим количеством элементов
    ...                                    //работаем с этим массивом
    ...
    return 0;
}
```

6.4. Макроподстановки с параметрами

Макроопределения (макросы) с параметрами похожи на функции. Макрос — это лексема, созданная с помощью директивы `#define`:

```
#define текстовое_имя(<список_параметров>) <текст_под-
  становки>
```

<Список_параметров> содержит один или более формальных параметров, разделенных запятыми. Не допускаются пробелы между <текстовым_именем> и открывающей скобкой <списка_параметров>. Первый пробел завершает имя макроопределения, и все, что следует за этим пробелом, считается <текстом_подстановки>. Макрос принимает параметры, переданные ему, и обрабатывает их. Особенность макроопределений заключается в том, что в списке не указываются типы параметров. Поэтому рекомендуется каждый аргумент в <тексте_подстановки> и сам <текст_подстановки> заключать в круглые скобки. Это будет гарантией того, что элементы будут сгруппированы надлежащим образом при вызове макроса.

Приведем пример макроса, осуществляющего умножение произвольного значения на два.

```
#define TWICE(x) ((x)*2)
```

В этом примере с помощью директивы `#define` вводится подстановка, заменяющая текст `TWICE(x)` на выражение `«(x)*2»`. Наличие круглых скобок `«(x)»` после имени говорит о том, что эта подстановка имеет один параметр, обозначенный как `«x»`. Препроцессор во всем тексте программы, находящемся после этой директивы, заменит текст `TWICE(x)` на `(x)*2`, где вместо `«x»` может находиться произвольное значение. Например, текст `TWICE(5)` будет заменен на `«(5)*2»`, а текст `TWICE(1,45)` — на `«(1,45)*2»`.

Отметим, что круглые скобки в выражении `«(x)*2»` необходимы для задания правильной логики работы макроса. Например, макрос `TWICE(2+3)` будет заменен на выражение `«(2+3)*2»`, значение которого будет равно 10. Если же использовать макроподстановку, в которой параметр используется без круглых скобок, то в результате вычисления аналогичного макроса `TWICE(2+3)` будет получен результат `«2+3*2»`, что в конечном итоге будет равно 8.

Задача. Написать макроподстановку, вычисляющую минимум из двух чисел.

```
#include <stdio.h>
#define MIN(x,y) ((x)<(y) ? (x) : (y))
    //определяем макроподстановку, вычисляющую минимум из двух чисел
int main()
{
    int a, b, min;
    printf("a = ");
    scanf("%d", &a);                //вводим с клавиатуры значение a
    printf("b = ");
    scanf("%d", &b);                //вводим с клавиатуры значение b
    min = MIN(a, b);                //вычисление минимума
    printf("Минимум из %d и %d = %d\n", a, b, min);
    return 0;
}
```

Пояснения. Макроподстановка `MIN(x, y)` имеет два параметра и заменяется на выражение `(x)<(y) ? (x) : (y)`, в котором используется условная операция для вычисления минимума из двух чисел.

6.5. Прагмы

Рассмотренные директивы препроцессора всегда выполняются одинаковым образом вне зависимости от используемого компилятора. В отличие от них выполнение прагм зависит от версии компилятора. То есть прагмы — это команды компилятору, которые определяются реализацией. Синтаксис использования прагм имеет следующий вид:

```
#pragma лексема
```

Приведем некоторые из доступных прагм.

```
#pragma once
```

Данная команда обеспечивает однократную компиляцию ниже следующего текста. По своей сути это аналог директив условной компиляции `#ifndef`, используемых для защиты заголовочных файлов от многократного включения.

```
#pragma warning( спецификатор : номер )
```

Эта команда позволяет управлять выводом возникающих предупреждений в процессе компиляции программы. В качестве спецификатора могут использоваться:

- **disable** блокирует вывод предупреждения;
- **once** выводит предупреждение только один раз;
- **error** выводит предупреждение как ошибку.

Номер — номер любого предупреждения, выдаваемого компилятором.

Примеры использования могут выглядеть следующим образом:

```
#pragma warning( disable : 4034 )  
//предупреждение с номером 4034 выводиться не будет  
#pragma warning( once : 4385 )  
//предупреждение 4385 будет выведено всего один раз  
#pragma warning( error : 4164 )  
/*предупреждение 4164 будет интерпретировано как  
ошибка, и программа не будет скомпилирована*/
```

Рассмотрим простую программу, в которой объявляется вещественная переменная `x`.

```
#pragma warning( once : 4305 )
int main()
{
    float x = 1.6;
    return 0;
}
```

В строке `float x = 1.6` возникает предупреждение «warning C4305: 'initializing' : truncation from 'double' to 'float'» – «предупреждение C4305: 'инициализация': попытка преобразовать значение типа `double` (1.6) к типу `float` (`float x`)". Так как в начале программы используется прагма `warning` со спецификатором `once`, то это предупреждение будет выведено на экран всего один раз. То есть добавление в программу строки `float y = 2.568;` не вызовет возникновения новых предупреждений.

Чтобы убрать вывод этого предупреждения на экране, нужно заменить спецификатор `once` на `disable`. Если же нужно отследить возникновение всех подобных предупреждений и устранить их, то можно использовать спецификатор `error`. В этом случае предупреждения с указанным номером будут рассматриваться компилятором как ошибки, и программа не будет скомпилирована до их устранения.

Полный список доступных прагм, а также номера возникающих ошибок и предупреждений необходимо смотреть в документации на используемый компилятор.

Тесты

1. Препроцессор — это:

- а) программа, обрабатывающая исходный код программы для его последующей компиляции;
- б) инструмент предварительной обработки команд процессора;
- в) процесс обработки команд программиста.

2. Директива `#include`:

- а) вставляет содержимое указанного файла, начиная со следующей строки за самой директивой;
- б) вставляет содержимое указанного файла в начало программы;
- в) вставляет содержимое указанного файла в конец программы.

3. Что делает следующая строчка кода: #define M

- а) такая строчка недопустима, после буквы M должно идти число;
- б) определяет значение переменной M по умолчанию равным 0;
- в) удаляет со всего текста программы символ M.

4. Какой директивой должен заканчиваться блок программы, начинающийся с директивы #if ?

- а) #else;
- б) #endif;
- в) #end.

5. Укажите правильный вариант защиты заголовочного файла от многократного включения:

- а) #ifndef _file_
#define _file_
...
#endif
- б) #pragma once;
- в) оба варианта правильные.

6. Что будет выведено на экран в результате выполнения следующего кода?

```
#define DEMO
#ifndef DEMO
    printf("Демоверсия!\n");
#else
    printf("Полная версия!\n");
#endif
```

- а) Демоверсия!
- б) Полная версия!
- в) На экран ничего выведено не будет.

7. Какой директивой можно отменить ранее введенную подстановку?

- а) #end;
- б) #enddef;
- в) #undef.

8. Что будет выведено на экран после выполнения следующего фрагмента программы?

```
...
#define MACROS(x) 5*x+x
    printf("%d\n", MACROS(3+2));
...
```


- a) 15;
- б) 22;
- в) 30.

9. Укажите правильный вариант блокировки вывода предупреждения компиляции с номером 4305 на экран:

- a) `#pragma warning (once : 4305);`
- б) `#pragma warning (disable : 4305);`
- в) `#pragma warning (error : 4305).`

10. В чем отличие прагм от остальных директив препроцессора?

- a) между ними отличий нет;
- б) прагмы выполняются после обработки всех директив;
- в) результат работы прагм зависит от их реализации.

Задачи

1. Напишите программу вычисления длины окружности и площади круга. Число `pi` задайте директивой `#define`.

2. С помощью директивы `#define` задайте подстановку `VVOD "Введите значение переменной: "`. Напишите программу, которая просит пользователя ввести значения вещественной и целой переменных. Для вывода приглашения используйте подстановку.

3. Добавьте в проект заголовочный файл, содержащий объявление целочисленной константы и символьной переменной. В основной программе выведите их значения на экран.

4. С помощью директивы `#define` обозначьте номер версии программы. Выведите этот номер на экран.

5. Напишите макрос вычисления минимума из двух чисел. Продемонстрируйте работу макроса с переменными вещественного и целого типа.

6. Напишите макрос вычисления максимума из двух чисел. Продемонстрируйте работу макроса с переменными вещественного и целого типа.

7. Напишите макрос возведения числа в квадрат. Продемонстрируйте применение макроса.

Контрольные вопросы

1. Что такое препроцессор? С какого символа начинаются команды препроцессора?

2. Что делает директива `#include`? В чем разница между использованием угловых скобок (`<...>`) и кавычек (`"..."`) при указании имени файла?
3. Какие действия выполняет директива `#define`? Приведите примеры.
4. Приведите синтаксис использования директивы условной компиляции `#if`. Поясните на примерах.
5. Каким образом можно исключить компиляцию части кода программы?
6. Что такое заголовочные файлы? Для чего они используются? Как его добавить в проект?
7. Как организовать однократное включение заголовочного файла в компилируемый текст?
8. С помощью какой директивы отменить ранее введенную подстановку?
9. Что такое макроподстановки с параметрами?
10. Приведите пример макроподстановки, вычисляющий максимум из двух чисел.
11. Что такое прагмы? В чем их особенность?
12. Приведите примеры управления выдачей предупреждений компилятором.

Глава 7

ПАМЯТЬ. АДРЕСА. УКАЗАТЕЛИ

Практически в любой программе используются переменные, которые располагаются в оперативной памяти. Упрощенно память можно представить в виде массива байтов, каждый из которых имеет адрес, начиная с нуля. В зависимости от типа переменные занимают различный объем памяти.

7.1. Организация памяти. Хранение переменных в памяти

Условно можно разделить всю память на два вида: *статическую* и *динамическую*. С физической точки зрения разницы между этими типами памяти нет. Поэтому, когда будем говорить о видах памяти, будем иметь в виду способы организации работы с ней.

Статическая память выделяется до начала работы программы, а статические переменные создаются и инициализируются до входа в функцию `main`. Например, при объявлении переменной `int a[10];` автоматически выделяется 10 ячеек памяти, каждая из которых предназначена для хранения целого значения.

Динамическую память часто называют «*кучей*». Программа может захватывать участки динамической памяти требуемого размера. Необходимо помнить, что после использования захваченную память следует освободить.

7.2. Указатели. Объявление. Инициализация

Указатели. Когда компилятор обрабатывает оператор определения статической переменной, например `int j=10;`, он выделяет память в соответствии с типом (`int`) и инициализирует ее указанным значением (10). Все обращения в программе к переменной по ее имени (`j`) заменяются компилятором на ее адрес области памяти, в ко-

торой хранится значение переменной. Адресом участка оперативной памяти является номер байта, с которого начинается этот участок. Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются указателями. Указатель — это переменная, которая содержит адрес другой переменной или функции. Описание указателя:

```
тип_ данных *имя_указателя;
```

Звездочка относится непосредственно к имени_указателя, поэтому, чтобы объявить несколько указателей, требуется ставить ее перед каждым именем.

Примеры объявления указателей.

```
int *pi, a, b, *pk; /*объявляются две целые переменные a и b и два
                    указателя pi и pk на неименованные области памяти типа int*/

double *pa, b; /*объявление переменной b типа double и указателя pa
                на тип double*/

char *ptext; //объявление указателя ptext на переменную типа char
```

Инициализация указателей. При объявлении указателя желательно выполнить его инициализацию. Если значение указателя заранее неизвестно, то можно его инициализировать нулевым адресом (NULL).

Рассмотрим некоторые способы инициализации указателей.

1. Инициализация указателя адресом существующего объекта:

- с помощью операции получения адреса:

```
int a=5;                                //объявление целой переменной
int *p=&a;                               //в указатель записывается адрес a
```

- с помощью значения другого инициализированного указателя:

```
int a=5; int *p=&a;
int *r=p;
```

2. Инициализация пустым значением:

```
float *r=NULL; или float *r=0;
```

7.3. Операции взятия адреса и разыменования

Символ звездочка используется не только для объявления указателя, но и как оператор разыменования для организации доступа к содержимому области памяти, на которую указывает указатель. Ком-

пилятор по контексту определяет, как именно должен использоваться символ звездочка.

Примеры разыменования указателей.

```
int i=0; //объявление целой переменной i
int *pi=&i; //объявление указателя pi на тип int
*pi=5; //присваивает значение 5 переменной по адресу в указателе pi

char text='A'; //объявление символьной переменной text
char *ptext=&text; //объявление указателя ptext на тип char
*ptext='h'
//присваивает значение 'h' переменной по адресу в указателе ptext
```

Описание указателя без инициализации выделяет память для него, но пока ему не присвоено никакое значение, он ни на что не указывает, т. е. не содержит ни адреса области памяти, ни значения NULL (0). Для получения адреса какого-либо объекта используется операция взятия адреса &.

Пример использования операции взятия адреса.

```
char cval, *pc; //описание символьной переменной cval и указателя pc
int ival, *pi; //описание целой переменной ival и указателя pi
pc=&cval; //указатель pc содержит адрес переменной cval
pi=&ival; //указатель pi содержит адрес переменной ival
```

7.4. Арифметические операции с указателями

С указателями могут быть использованы операции инкремента (++), декремента (--), сложения (+), вычитания (-); операции сравнения, присваивания. Если к указателю применяются операции ++ или --, то указатель увеличивается или уменьшается на размер объекта, на который он указывает, что условно можно представить так:

```
тип *ptr;
... //присваивание значения указателю ptr
...
ptr++ = содержимое_ptr+sizeof(тип)
ptr-- = содержимое_ptr-sizeof(тип)
```

где типом может быть любой основной тип языка C++ или тип, введенный пользователем.

Операции сложения или вычитания можно применять к указателю, если вторым операндом будет целое число. Целое число складывается с указателем или вычитается из него следующим образом:

```
тип *ptr;
int n;
...                               //присваивание значения указателю ptr
...
ptr+n=содержимое_ptr+n*sizeof(тип)
ptr-n =содержимое_ptr-n*sizeof(тип)
```

Один указатель можно вычитать из другого, если они оба указывают на объекты одного и того же типа. Если, например, два указателя ссылаются на разные элементы массива, вычитание одного указателя из другого позволяет получить количество элементов массива, находящихся между двумя заданными.

К указателям можно применять только описанные выше операции. Следующие операции недопустимы с указателями:

- сложение двух указателей;
- вычитание двух указателей на различные объекты;
- сложение указателей с числом с плавающей точкой;
- вычитание из указателей числа с плавающей точкой;
- умножение и деление указателей.

Присваивание указателей. Можно присваивать один указатель другому. Однако эта операция имеет смысл только в том случае, если оба указателя являются указателями на объекты одного и того же типа.

Пример присваивания указателей.

```
#include <stdio.h>
int main()
{
    int j =12525, *p1, *p2;           /* объявление целой переменной j
                                      и описание указателей p1 и p2 */
    p1=&j;                           //присваивание указателю p1 адреса переменной j
    p2=p1;                           //присваивание указателей
    printf("%d\n", *p1);             //оба оператора напечатают одно и то же
    printf("%d\n", *p2);
    return 0;
}
```

Сравнение указателей. Можно сравнивать два указателя, если они указывают на объекты одного и того же типа. Указатели равны (сов-

падают), если указывают на один и тот же объект, указатели не равны (не совпадают), если указывают на разные объекты.

Пример сравнения на неравенство указателей.

```
#include <stdio.h>
int main()
{
    int a,b;
    int *p1, *p2;
    p1=&a;
    p2=&b           //указатели p1 и p2 не равны, так как они указывают
                    //на разные объекты
    if (p1 != p2) printf ("Указатели p1 и p2 не равны \n");
    else printf("Эта строка не напечатается \n");
    return 0;
}
```

7.5. Выделение динамической памяти. Операторы *new* и *delete*

Условно можно разделить всю память на два вида: автоматически распределяемую при работе программы (например, при объявлении переменной `int a[10]`; автоматически выделяется 10 ячеек памяти, каждая из которых предназначена для хранения целого значения) и свободную или динамически распределяемую. Доступ к ячейкам свободной памяти осуществляется посредством указателя, хранящего адрес нужной ячейки. Для выделения памяти в области динамического распределения используется операция *new*. Формат оператора для выделения фрагмента памяти в области динамического распределения памяти

```
new имя_ типа;
```

new — операция выделения фрагмента памяти;

имя_типа — имя типа, который будет использован при доступе к выделенному фрагменту памяти.

Например, `new int;` выделит четыре байта памяти, а `new double;` — восемь байт. В качестве результата оператор *new* возвращает адрес выделенного фрагмента памяти. Этот адрес должен присваиваться указателю. Сама же выделенная область (фрагмент) ника-

кого имени не получает, к ней можно обращаться только через указатель.

Примеры.

```
unsigned int *ps;  
ps = new unsigned int;
```

Теперь `ps` указывает на ячейку памяти, достаточную для хранения значения типа `unsigned int`, и туда можно занести какое-нибудь значение: `*ps = 72;`

```
double *pa;  
pa = new double;
```

Теперь `pa` указывает на ячейку памяти, достаточную для хранения значения типа `double`, и туда можно занести какое-нибудь значение: `*pa = 3.14;`

Операция *delete*. Когда память, выделенная под переменную, больше не нужна, ее следует освободить. Делается это с помощью оператора `delete`:

```
delete имя_указателя;
```

Примеры.

```
delete ps;  
delete pa;
```

Повторное применение оператора `delete` к тому же самому указателю приведет к зависанию программы. Рекомендуется при освобождении памяти присваивать связанному с ней указателю нулевое значение. Повторный вызов оператора `delete` для нулевого указателя пройдет безболезненно для программы, например,

```
char *pc;  
pc = new char;  
*pc = 'a';  
delete pc;  
pc = NULL;  
delete pc;
```

При освобождении памяти с самим указателем ничего не происходит и его можно снова использовать, присвоив ему новый адрес.

При невнимательной работе с указателями может возникнуть эффект утечки памяти. Это происходит, если указателю присваивается новое значение, а память, на которую он ссылался, не освобождается и возможно зависание программы из-за неправильного использования динамически распределяемой памяти. Каждый раз, когда в программе используется оператор *new*, за ним должен следовать оператор *delete*.

Пример утечки памяти

```
int *pi = new int //выделение памяти для хранения переменной типа
int
*pi=72;           //в выделенную память записывается значение 72
pi = new int;     //выделение новой области памяти
*pi =84;          //запись в новую область значения 84
```

После выполнения этих операторов память, содержащая значение 72, оказывается недоступной, поскольку указателю на эту область было присвоено новое значение. В результате невозможно ни использовать, ни освободить зарезервированную память до завершения программы (когда вся выделенная память освобождается автоматически).

Пример 1 выделения, использования и освобождения динамической памяти:

```
#include <stdio.h>
int main()
{
    int a=5;
    int * pa = &a;
    int *pheap = new int;
                        //выделение памяти под неименованную целую переменную
    *pheap = 7;
    printf("a=%d, *pa =%d, *pheap=%d \n", a, *pa, *pheap);
    delete pheap;      //освобождение памяти, выделенной оператором new
    pheap =pa;          //присвоение указателю нового значения
    printf(" *pheap=%d \n", *pheap);
    return 0;
}
```

Пример 2 выделения, использования и освобождения динамической памяти:

```
#include <stdio.h>
int main()
```



```
{
    int *pheap = new int;
        //выделение памяти под неименованную целую переменную
    *pheap = 7;
    printf(" *pheap=%d \n", *pheap);
    delete pheap;    //освобождение памяти, выделенной оператором new
    pheap=NULL;      //присвоение указателю нового пустого значения
    pheap =new int;
        //выделение памяти под новую неименованную целую переменную
    *pheap=-32768;
    printf(" *pheap=%d \n", *pheap);
    return 0;
}
```

7.6. Ссылки

В C++ имеется несколько видоизмененная форма указателя, называемая ссылкой. Ссылка на некоторую переменную может рассматриваться как указатель, который при употреблении всегда разыменовывается. Однако для ссылки не требуется дополнительного пространства в памяти: она является просто другим именем или псевдонимом уже существующей объявленной или описанной переменной. Поэтому все, что делается со ссылкой, в действительности происходит с переменной, на которую она ссылается. Ссылка определяется в программе следующим образом:

```
тип    &имя_ссылки=имя_переменной;
```

При этом переменная, на которую указывает (ссылается) ссылка, должна быть объявлена или описана до определения ссылки, например `int t, &rt = t;`. Оператор ссылки (&) выглядит так же, как оператор адреса, который используется для возвращения адреса при работе с указателями. Но это не одинаковые операторы. При определении ссылки необходимо ее инициализировать. Если объявить ссылку без инициализации, будет сгенерирована ошибка компиляции. Так как ссылка является псевдонимом своего адресата, то применение оператора взятия адреса к ссылке возвратит адрес ее адресата. Переназначить ссылку нельзя. Она ссылается на свой адресат и только на него. Ссылка не может быть нулевой.

Пример 1. Дана переменная `x` и ссылка на нее — `rx`. Изменить значение ссылки `rx` и после этого вывести на экран значение переменной `x`.

```
#include <stdio.h>
int main()
{
    int x;
    int &rx=x;           //rx — ссылка на x
    x=5;
    printf("x=%d \n", x);      //значение переменной x
    printf("rx=%d \n", rx);    //совпадает со значением ссылки rx
    rx = 7;                  //меняем значение ссылки,
                             //что автоматически меняет значение переменной x
    printf("x=%d \n", x);
    printf("rx=%d \n", rx);
    return 0;
}
```

Пример 2. Дано `a`, `b`. Найти `s = a + b`. Изменить значения переменных с помощью ссылки на них.

```
#include <stdio.h>
int main()
{
    int a,b,s;
    int &ra=a, &rb=b, &rs=s;
    printf("введите a,b\n");
    scanf("%d%d",&a,&b);      //ввод чисел a и b
    s=a+b; //вывод адреса переменных a, b, s и значений переменных a, b, s
    printf("&a=%d &b=%d &s=%d a=%d b=%d s=%d\n", (int)&a,
        (int)&b, (int)&s, a, b, s);
    /* вывод на экран значений адресов ссылок ra, rb, rs и значений самих
       ссылок ra, rb, rs, что полностью эквивалентно предыдущему выводу*/
    printf("&ra=%d &rb=%d &rs=%d ra=%d rb=%d
        rs=%d\n", (int)&ra, (int)&rb, (int)&rs, ra, rb, rs);
    /* изменение значений ссылок ra, rb, rs, что эквивалентно
       изменению значений переменных a, b, s */
    ra=ra*2;
    rb=rb*2;
    rs=ra+rb;
}
```

```
printf("&a=%d &b=%d &s=%d a=%d b=%d s=%d\n", (int)&a,
      (int)&b, (int)&s, a, b, s);
printf("&ra=%d &rb=%d &rs=%d ra=%d rb=%d
rs=%d\n", (int)&ra, (int)&rb, (int)&rs, ra, rb, rs);
}
```

Связь между переменной, ячейкой памяти, указателем и ссылкой на переменную показана на рис. 7.1.

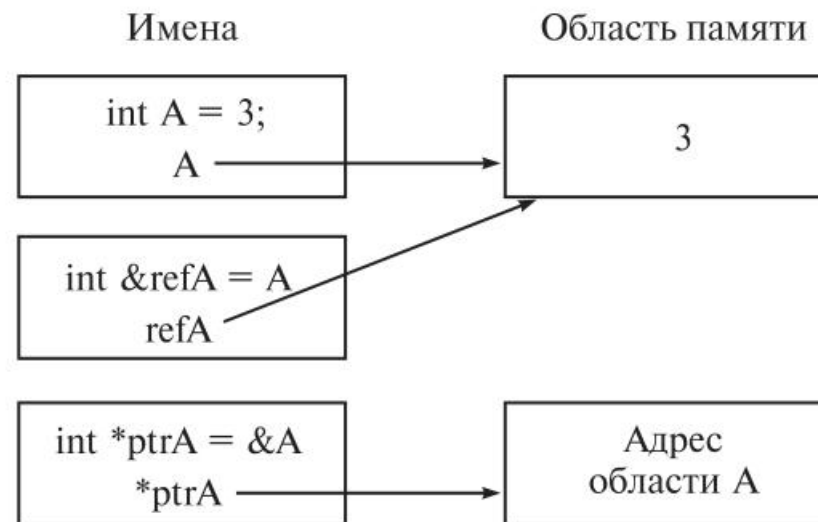


Рис. 7.1. Переменная, ссылка и указатель на переменную

Примеры использования указателей. Пример 1. Объявить и инициализировать переменную типа `float` и указатель на нее. Вывести на экран содержимое области памяти, на которую указывает этот указатель.

```
#include <stdio.h>
int main()
{
    float f=100.25;
    float *pi=&f;
    printf("Значение переменной f= %f \n",f);
    printf("Адрес переменной f: pi=%p\n",&f);
    //форматная спецификация %p используется для печати
    //значения указателя
    printf("Значение переменной pi= %f \n",pi);
    printf("Содержимое ячейки по адресу pi %f",*pi);
    return 0;
}
```

Пример 2. Задать значения двух целых переменных. Используя указатели на них, поменять значения этих переменных.


```
#include <stdio.h>
int main() {
    int *px, *py, x=5,y=10;           //px и py являются указателями
    int temp;
    px = &x; py = &y;
    printf("x=%3d; y=%3d \n", x,y);
    temp=*px;
    //переменной temp присваивается значение, на которое указывает px
    *px=*py;                          //теперь по адресу px находится значение y
    *py=temp;
    printf("x=%3d; y=%3d \n", x,y);
    return 0;
}
```

Тесты

1. Выберите правильный вариант описания указателя на тип `int`:

- a) `int p*;`
- б) `int *p;`
- в) `int &p.`

2. Укажите правильное обозначение операции разыменования:

- a) `* .;`
- б) `&;`
- в) правильный вариант отсутствует.

3. Укажите правильное обозначение операции взятия адреса:

- a) `*;`
- б) `&;`
- в) правильный вариант отсутствует.

4. Чему будет равна переменная `a` в результате работы следующего фрагмента программы?

```
...
int a = 5;
int *p = &a;
*p = a * (*p);
```

- a) 5;
- б) 25;
- в) 125.

5. Укажите операцию, недопустимую при работе с указателем:

```
int a = 4, *p = &a;
```

- а) `p++;`
- б) `p += 4;`
- в) `p = 5.`

6. Укажите эквивалент операции `p = p + 5`:

- а) `p = содержимое_p + 5;`
- б) `p = содержимое_p + 5 * sizeof(тип p); ..;`
- в) указанная операция недопустима при работе с указателем.

7. Укажите правильный вариант выделения динамической памяти для целочисленной переменной:

- а) `int *p = NULL; .p = new int;`
- б) `int *p = NULL; .p = new;`
- в) `int *p = NULL; .p = int new.`

8. Укажите правильный вариант освобождения ранее выделенной динамической памяти для целочисленной переменной:

- а) `p delete;`
- б) `delete int p;`
- в) `delete p.`

9. Укажите правильный вариант определения ссылки на вещественную переменную `x`:

- а) `float x, ℞`
- б) `float x, ℞ rx = x;`
- в) `float x, &rx = x.`

10. Чему будет равна переменная `x` в результате работы следующего фрагмента программы?

```
...  
int x=0;  
int &rx=x;  
x=1;  
rx=2*rx; ...
```

- а) 2;
- б) 1;
- в) 0.

Задания

1. Объявить указатели на типы `int`, `double`, `char`. Вывести на экран размер памяти, занимаемый этими указателями. Объясните полученный результат.
2. Ввести с клавиатуры значения переменных *a* и *b*. Поменять значения переменных местами, используя указатели.
3. С клавиатуры ввести значение целочисленной переменной. Записать адрес этой переменной в указатель и вывести его значение на экран. Увеличить значение указателя на 5. Вывести новое значение на экран. Сравнить его с предыдущим значением.
4. С клавиатуры ввести значения двух вещественных переменных. Посчитать их сумму и разность, вывести результат на экран. Решить задачу, используя динамическую память.
5. Написать программу, демонстрирующую утечку памяти.
6. С клавиатуры ввести значение вещественной переменной. Объявить ссылку на эту переменную. Вывести на экран объем памяти, занимаемый переменной и ссылкой. Объяснить полученные результаты.
7. В программе объявить целочисленную переменную и ссылку на нее. Изменить значение переменной, используя ссылку. Вывести значение ссылки на экран.
8. Ввести с клавиатуры значения переменных *a* и *b*. Поменять значения переменных местами, используя ссылки.

Контрольные вопросы

1. Как упрощенно можно представить память для хранения переменных?
2. Какие виды памяти доступны программисту?
3. Что такое указатель?
4. Как в программе объявить указатель? Приведите примеры.
5. Какой объем памяти занимает указатель? Зависит ли этот объем от типа переменной, адрес которой содержит указатель?
6. Как корректно проводить инициализацию указателей? Приведите примеры.
7. Какие операции используются при работе с указателями?
8. Приведите примеры использования операции разыменования.
9. Приведите примеры использования операции взятия адреса.
10. Какие арифметические операции можно выполнять с указателями?

11. Перечислите операции, которые недопустимы в работе с указателями.
12. С помощью какой операции можно выделить динамическую память для хранения переменных? Приведите примеры использования этой операции.
13. Какая операция служит для освобождения ранее выделенной памяти? Приведите примеры.
14. Что такое утечка памяти?
15. Что такое ссылка?
16. Чем ссылка отличается от указателя?
17. Приведите пример использования ссылки.

Глава 8

ОДНОМЕРНЫЕ МАССИВЫ

При решении многих задач необходимо вводить, выводить, сохранять и обрабатывать большое количество однотипных данных. Для обеспечения эффективной обработки большого количества данных в языке C++ используются массивы.

8.1. Понятие одномерного массива

Что такое одномерный массив? Предположим, что в программе есть много значений одного и того же типа, и они должны сохраняться в течение всего времени выполнения программы. При использовании большого количества переменных процесс программирования будет очень громоздким и трудоемким. Гораздо удобнее сгруппировать эти значения в одном месте оперативной памяти, пронумеровать их, дать этой области памяти одно имя и обращаться к значениям по этому имени и порядковому номеру. Для этого в C++ используются массивы.

Рассмотрим переменную A и массив A (рис. 8.1).

Массив — упорядоченный набор однотипных элементов, снабженных индексами (порядковыми номерами). Например, $A = \{5, 7,$

1) A — переменная (ячейка памяти, в которой находится значение 5)



2) A — массив (ячейки памяти, в которых находятся значения 5, 7, -3, 9, 1, 7)



Рис. 8.1. Представление в памяти переменной и массива

$-3, 9, 1, 7\}$ — массив A из 6 целочисленных элементов, каждый из которых имеет свой номер — индекс. Если в этом массиве поменять местами любые 2 элемента, то это будет тот же самый массив с тем же самым именем, расположенный в том же самом месте оперативной памяти, но имеющий другие значения элементов. Массив $A=\{5, 7, -3, 1, 9, 7\}$ является тем же самым массивом, расположенным в том же самом месте, что и исходный массив $A=\{5, 7, -3, 9, 1, 7\}$, но имеет уже другие значения элементов.

Объявление одномерного массива в программе. Массив в программе объявляется следующим образом:

```
<тип_элементов_массива> <имя_массива> [<количество_элементов_массива>];
```

Количество элементов массива должно быть определено до начала компиляции программы. Возможны несколько вариантов задания количества элементов массива.

1. Непосредственно указать число `float mas[12];`
2. Объявить целочисленную константу и потом использовать ее

```
const int k = 10;
char massiv[k];
```

3. Воспользоваться директивами препроцессора

```
#define N 14
int M[N];
```

Обратите внимание, что в отличие от 2-го варианта, требующего память для хранения константы, в 3-м варианте выделение памяти не происходит.

Атрибуты массива:

- имя массива;
- количество элементов в массиве;
- тип элементов (любой тип данных, применяемый в C++).

Доступ к элементам массива. Доступ к элементам массива осуществляется через имя массива и индекс элемента. Индексация элементов массива начинается с нуля. Например, дан массив A из 5 элементов (рис. 8.2).

A				
3	-5	6	7	8
A[1]	A[2]	A[3]	A[4]	A[5]

Рис. 8.2. Одномерный массив A

Для обнуления 0-го и 1-го элементов массива используются 2 оператора присваивания:

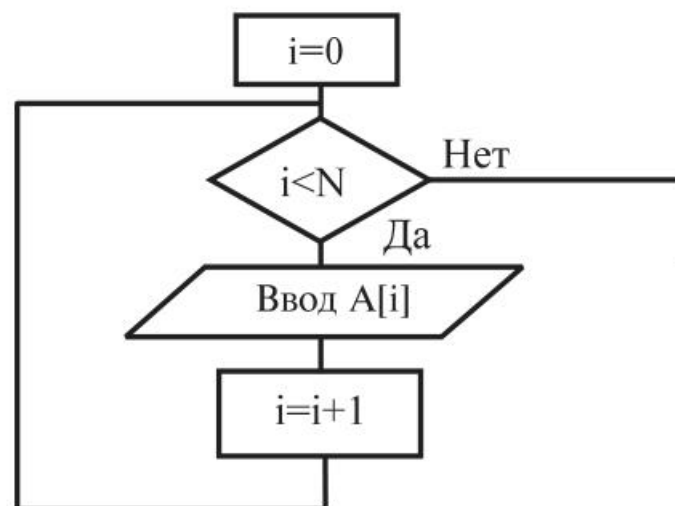
```
A[0]=0;  
A[1]=0;
```

8.2. Работа с одномерными массивами

Массивы можно поэлементно складывать, умножать, вычитать, делить и применять к элементам массивов различные сочетания арифметических операций для получения нового массива, находить минимальный (максимальный) элемент массива, сумму, произведение, количество каких-либо элементов и т. д. Все действия над массивами в C++ производятся поэлементно с использованием одного из операторов цикла.

Ввод массива с клавиатуры одной строкой и по одному элементу в строке.

1. Схема алгоритма



2. Программы

1-й способ. Программа ввода элементов массива одной строкой.

```
#include <stdio.h>  
const int N=10; //объявление константы — числа элементов массива  
int main() { //заголовок функции main()  
    int A[N], i; /*объявление переменных: A — целочисленный массив,  
                 i — параметр цикла, равный индексу элемента массива*/  
    printf("Введите %d элементов массива через  
    пробел\n", N); //запрос ввода массива
```

```

for                                     (i=0; i<N; i++)
scanf("%d",&A[i]);                     //ввод i-го элемента массива
...
getchar();
getchar();                             //фиксирование результата на экране
return 0;
}

```

2-й способ. Программа ввода элементов массива по одному в каждой строке.

```

#include <stdio.h>
const int N=10; //объявление константы — числа элементов массива
int main() {     //заголовок функции main()
int A[N], i;     /*объявление переменных: A — целочисленный массив,
                  i — параметр цикла, равный индексу элемента массива*/
printf("Введите %d элементов массива по одному в
строке\n",N);

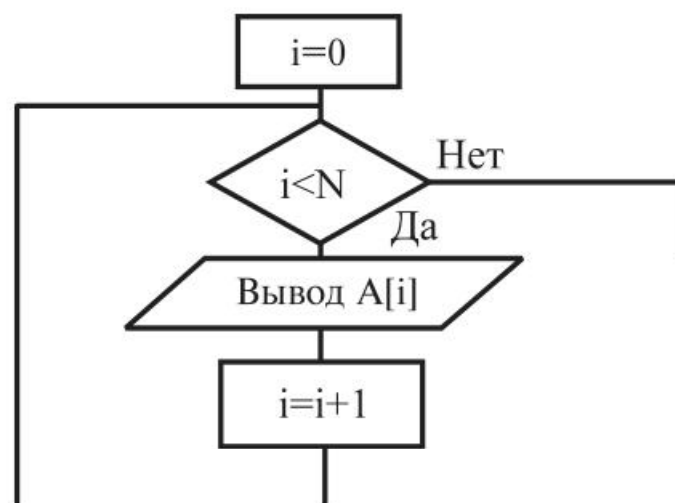
//запрос ввода массива

for (i=0; i<N; i++)
{printf("введите A[%d] ",i);
scanf("%d",&A[i]);      //ввод i-го элемента массива
}
...
getchar();
getchar();              //фиксирование результата на экране
return 0;
}

```

Вывод массива на экран одной строкой и по одному элементу в строке.

1. Схема алгоритма



2. Программы

1-й способ. Программа вывода всех элементов массива одной строкой.

```
#include <stdio.h>
const int N=10;                                //объявление константы —
                                                //числа элементов массива
int main() {                                    //заголовок функции main()
int A[N], i;                                  /*объявление переменных: A — целочисленный массив,
                                                i — параметр цикла, равный индексу элемента массива*/
    ...
    ...
    printf("Массив A\n");
    for                                          (i=0; i<N; i++)
    printf("%d ", A[i]);                        //вывод i-го элемента массива
    printf("\n");
    ...
    getchar();
    getchar();                                //фиксирование результата на экране
    return 0;
}
```

2-й способ. Программа вывода элементов массива по одному в каждой строке.

```
#include <stdio.h>
const int N=10;                                //объявление константы — числа элементов массива
int main() {                                    //заголовок функции main()
int A[N], i;                                  /*объявление переменных: A — целочисленный массив,
                                                i — параметр цикла, равный индексу элемента массива*/
    ...
    ...
    printf("Массив A\n");
    for                                          (i=0; i<N; i++)
    printf("A[%d]=%d\n ", i, A[i]);            //вывод i-го элемента массива
    ...
    getchar();
    getchar();                                //фиксирование результата на экране
    return 0;
}
```

Формирование массива с помощью генератора случайных чисел.

В языке C++ есть встроенный генератор случайных чисел — функция — `rand()`, которая формирует целые псевдослучайные значения в

диапазоне от 0 до `RAND_MAX`. Константа `RAND_MAX` определена в заголовочном файле `stdlib.h` и равняется 32 767.

Формируемые значения называются *псевдослучайными*, так как генератор создает их по строго заданному алгоритму, неизвестному программисту, для которого эти значения кажутся случайными. Для формирования значения необходимо сначала запустить датчик случайных чисел следующим образом:

```
srand(time(NULL));
```

где `srand` — функция инициализации генератора случайных чисел, которая обязательно должна быть вызвана перед использованием генератора случайных чисел `rand`;

`time` — функция работы с таймером компьютера;

`NULL` — нулевой адрес, указывает на то, что от таймера необходимо получить машинное время в миллисекундах.

Затем для получения целого случайного числа необходимо выполнить оператор:

```
<имя переменной> = rand();
```

Функция `rand()` — генерирует значение целого случайного числа в диапазоне от 0 до `RAND_MAX`.

Если необходимо сгенерировать целое случайное число в диапазоне от 0 до `N`, можно поступить следующим образом:

```
<имя переменной> =rand()%(N+1);
```

`<имя переменной>` — переменная целого типа, в которую запишется сформированное псевдослучайное число;

`N` — целое положительное (натуральное) число меньше 32 767.

Сначала функция `rand()` генерирует значение случайного целого числа в диапазоне от 0 до `RAND_MAX`, затем ищется остаток от деления этого числа на `N+1` (`rand()%(N+1)`), и получается целое случайное число в диапазоне от 0 до `N`.

Специальной функции для получения вещественного случайного числа в языке C++ не существует, но получение вещественного случайного числа в диапазоне от 0 до 1 можно осуществить следующим образом: `(double)rand()/RAND_MAX` (значение функции `rand()` поделить на максимальную границу диапазона `RAND_MAX`). Обратите внимание на необходимость преобразования результата возвращаемого функцией `rand` к вещественному типу данных. Если в приве-

денном выражении опустить `(double)`, то результат целочисленного деления `rand() / RAND_MAX` почти всегда будет равен нулю, а иногда 1.

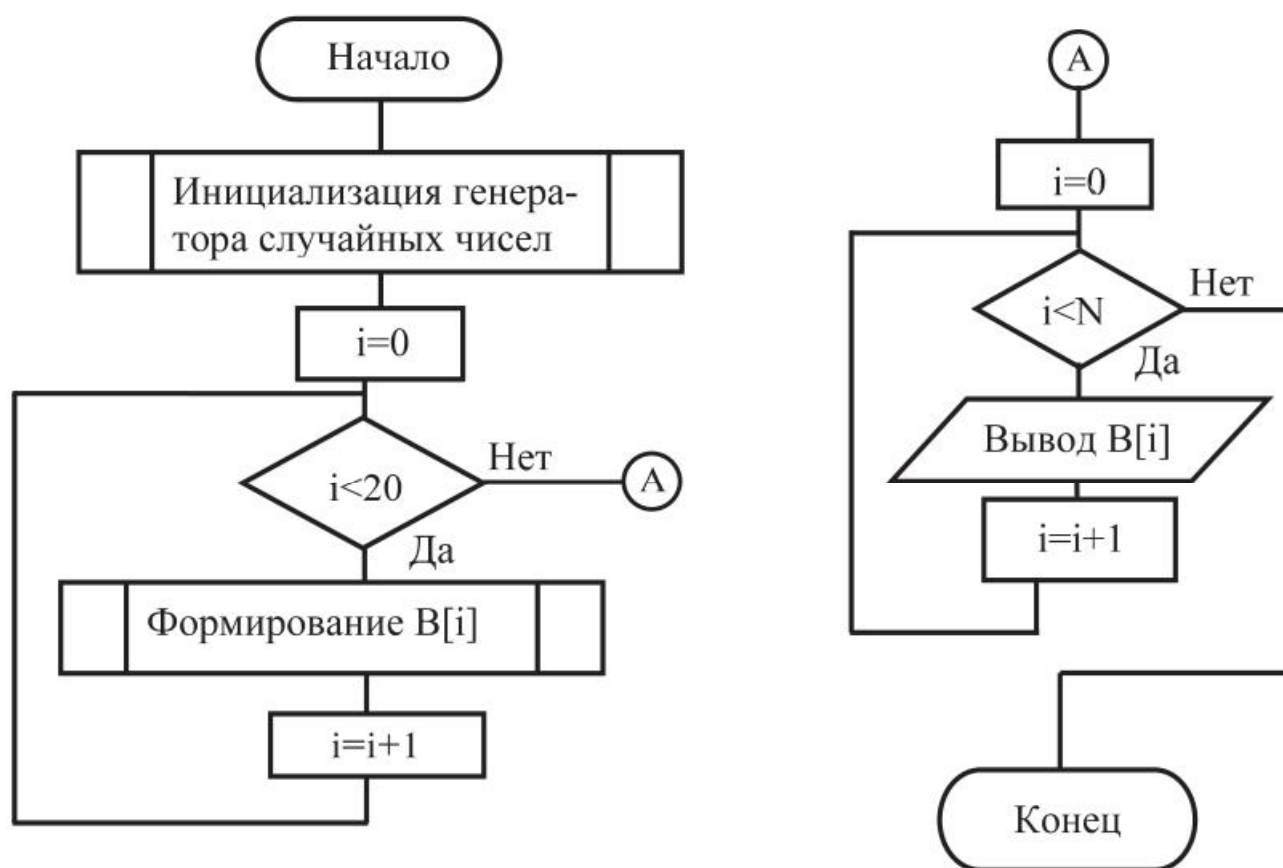
Рассмотрим алгоритм получения вещественного случайного числа, лежащего в диапазоне от A до B , где A, B — вещественные константы типа `double`.

```
double x;
x = ((double) rand() / RAND_MAX) * (B - A) + A;
```

Здесь первая часть выражения, заключенная в круглые скобки, генерирует случайное число в диапазоне от 0 до 1. В результате умножения полученного случайного числа на $(B-A)$ получаем случайную величину, лежащую в диапазоне от 0 до $(B-A)$. В заключение прибавляем значение нижней границы A и получаем требуемый диапазон от A до B .

Пример программы с использованием генератора случайных чисел. Сформировать целочисленный массив B из 20 элементов случайным образом. Вывести массив B на экран.

1. Схема алгоритма



Пояснение. Сначала инициализируется генератор случайных чисел. В 1-м цикле случайным образом формируются значения 20 эле-

ментов массива В. Во 2-м цикле значения элементов массива В выводятся на экран.

2. Программа

```
#include <stdlib.h>
//подключение библиотеки для работы с датчиком случайных чисел
#include <time.h> //подключение библиотеки работы с таймером
#include <stdio.h> //подключение стандартной библиотеки
//ввода-вывода

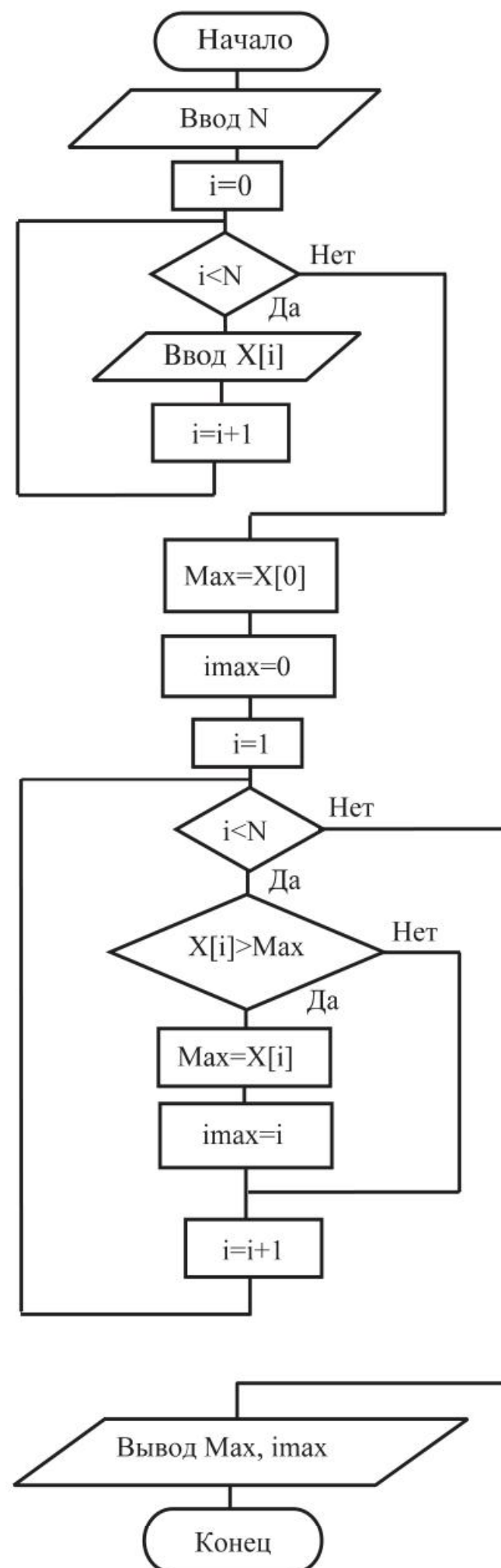
const int N=20; //число элементов массива
int main() {
    int B[N], i; //описание целочисленного массива В и переменной i
    srand(time(NULL)); //запуск датчика случайных чисел
    for (i=0; i<N; i++) //в цикле
        B[i]=rand()%10-5; //формирование i-го элемента массива В
    printf("Массив В\n"); //вывод заголовка с именем массива
    for (i=0; i<N; i++) //в цикле
        printf("B[%d]=%d\n ", i, B[i]); //вывод i-го элемента массива
    getchar();
    return 0;
}
```

Пояснение. В программе используются две переменные: В — целочисленный массив из 20 элементов, i — параметр цикла (в программе используются два оператора цикла **for**) и одновременно индекс элемента массива (обязательно **int**). В 1-м цикле по формуле `rand()%10-5` формируется целое случайное число в диапазоне от (-5) до 4 и полученное значение присваивается i-му элементу массива (`B[i]=rand()%10-5`). Во 2-м цикле значения элементов массива выводятся на экран.

8.3. Поиск максимального (минимального) элемента в массиве и определение его индекса

Пример программы поиска максимального элемента в массиве и определения его порядкового номера. Ввести массив Х. Найти максимальный элемент массива и его индекс.

1. Схема алгоритма



Пояснение. Пусть массив X состоит из семи элементов. Введем рабочие переменные Max (для хранения текущего значения максимального элемента массива), $imax$ (номер текущего максимального элемента). Предположим, что 0-й элемент массива — максимальный. Запомним его значение в переменной Max ($Max=X_0$), в переменной $imax$ запомним значение «0» ($imax=0$). Далее 1-й элемент массива сравниваем со значением max . Если 1-й элемент больше Max , то запишем его значение в переменную Max ($Max=X_1$), а номер 1-го элемента («0») запомним в переменной $imax$ ($imax=1$) и переходим к следующему элементу массива. Если 1-й элемент не больше Max , то сразу переходим к следующему элементу массива, не изменяя значений переменных Max и $imax$. Аналогичные действия повторяем со всеми элементами массива. Таким образом, после прохода по всему массиву в переменной Max будет находиться значение максимального элемента массива, а в $imax$ — порядковый номер максимального элемента массива.

Поясним это на рис. 8.3.

	$X[1]$	$X[2]$	$X[3]$	$X[4]$	$X[5]$	$X[6]$	$X[7]$
i	1	2	3	4	5	6	7
$X[i]$	8	-3	0	12	5	-7	19
$X[i] > max?$	-	-3 < 8? Нет	0 < 8? Нет	12 > 8? Да	5 < 12? Нет	-7 < 12? Нет	19 > 12? Да
Max	8	8	8	12	12	12	19
$imax$	1	1	1	4	4	4	7

Рис. 8.3. Поиск максимального элемента массива

2. Программа

```
#include <stdio.h>
const int M=100;
    //определение константы — максимального числа элементов в массиве
int main() {
    int X[M], N, i, Max, imax;
    printf("Сколько элементов будет в массиве ");
    //запрос ввода длины массива
    scanf("%d",&N);
    for(i=0; i<N; i++)
        scanf("%d",&X[i]);
    //в цикле
    //ввод элемента массива
```

```

Max=X[0];           //пусть нулевой элемент будет максимальным
imax=0;             //индекс максимального элемента будет нулевым
for (i=1; i<N; i++) //в цикле происходит перебор элементов массива
if (X[i]>Max) //сравнение каждого элемента с текущим максимальным
{Max=X[i];         //если X[i]>Max, то переопределяем Max и imax
imax=i;
}
printf("максимальный элемент = %d\n",Max);
printf("индекс максимального элемента %d\n",imax);
printf("Массив X\n");
for (i=0; i<n; i++) //вывод массива X одной строкой
printf("%d ",X[i]);
printf("\n");
getchar();
return 0;
}

```

Пояснение. В программе используются пять переменных: X — целочисленный массив не более чем из 100 элементов; N — число элементов в массиве X (не более 100); i — параметр цикла и одновременно индекс элементов массива в цикле; Max — максимальный элемент массива; $imax$ — индекс (порядковый номер) максимального элемента массива. Переменные N , i , $imax$ могут быть только типа **int**. Тип переменной Max определяется типом элементов массива X (в данном случае **int**).

Примечание. Нахождение минимального элемента массива и его индекса аналогично поиску максимального элемента, но значения переменных Min (текущий минимальный элемент) и $imin$ (индекс текущего минимального элемента) переопределяются, если некоторый элемент массива меньше текущего минимального. Поясним это на рис. 8.4 (для того же массива X из семи элементов).

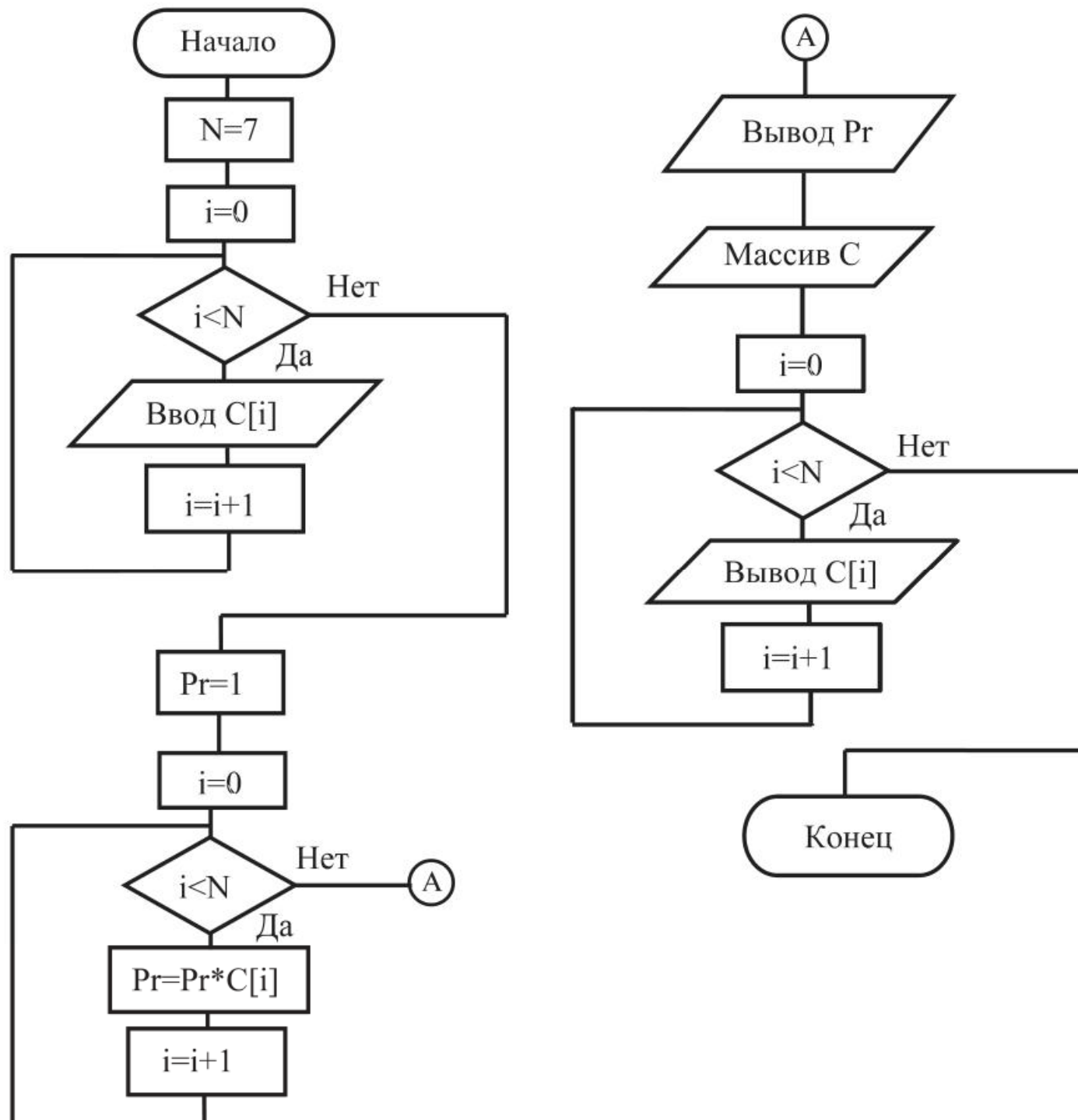
	$X[1]$	$X[2]$	$X[3]$	$X[4]$	$X[5]$	$X[6]$	$X[7]$
i	1	2	3	4	5	6	7
$X[i]$	8	-3	0	12	5	-7	19
$X[i] < min?$	-	-3 < 8? Да	0 > -3? Нет	12 > -3? Нет	5 > -3? Нет	-7 < -3? Да	19 > -7? Нет
Min	8	-3	-3	-3	-3	-7	-7
$imin$	1	2	2	2	2	6	6

Рис. 8.4. Поиск минимального элемента массива

8.4. Решение задач с использованием одномерных массивов

Задача 1. Ввести массив C из семи элементов. Найти произведение его элементов. Вывести произведение и массив C на экран.

1. *Схема алгоритма*



Пояснение. С клавиатуры вводятся значения семи элементов массива C . Произведению присваивается начальное значение «1» ($Pr=1$). Далее в цикле каждый элемент массива умножается на уже имеющееся произведение элементов. В конце программы на экран выводятся полученное произведение и массив C .

2. Программа

```
#include <stdio.h>
const int N=7;    //объявление константы — числа элементов массива
int main() {
    int C[N], i;    /*объявление переменных: A — целочисленный массив,
                    i — параметр цикла, равный индексу элемента массива*/
    double Pr;
    for (i=0; i<N; i++){                                //в цикле
        printf("введите C[%d] ", i); //запрос ввода i-го элемента массива
        scanf("%d", &C[i]); //ввод i-го элемента массива
    }
    Pr=1; //присвоение начального значения произведения (Pr)
    for (i=0; i<N; i++) //в цикле
        Pr=Pr*C[i]; //умножение старого значения Pr на элемент массива
    printf("Pr=%lf\n", Pr); //вывод на экран значения Pr
    printf("Массив C\n");
    for (i=0; i<N; i++) //в цикле
        printf("%d\t", C[i]); //вывод элемента массива
    printf("\n");
    getchar();
    return 0;
}
```

Пояснение. В программе используются три переменные: целочисленный массив *C* из 7 элементов (количество элементов в массиве объявлено перед заголовком функции *main()* как константа *const int N=7*); *i* — параметр цикла и одновременно индекс элементов массива *C*; *Pr* — произведение всех элементов (объявлена как переменная типа **double**, так как тип **double** имеет более широкий диапазон возможных значений).

Для решения задачи необходимо ввести с клавиатуры значения 7 элементов массива *C* (1-й оператор **for**), вычислить произведение всех элементов массива (2-й оператор **for**) и вывести его на экран; а также вывести массив *C* на экран (3-й оператор **for**).

Задача 2. Сформировать массив *B* с помощью генератора случайных чисел. Найти среднее арифметическое ненулевых элементов массива. Вывести элементы массива *B* на экран одной строкой.

Программа

```
#include <stdio.h>
#include <stdlib.h>
//подключение библиотеки работы с датчиком случайных чисел
#include <time.h> //подключение библиотеки работы с таймером
```

```

const int N1=100;
    //определение константы — максимального числа элементов массива
int main() {
int  B[N1],N,I, Sum, Kol;
double Sr;
printf("Сколько элементов в массиве ? ");    //запрос длины
                                              //массива

scanf("%d",&N);
srand(time(NULL));    //инициализация генератора случайных чисел
for (i=0; i<N; i++)    //в цикле случайным образом
B[i]=rand()%10-5;    //формируются элементы массива В
Sum=0;    //начальное значение суммы ненулевых элементов
Kol=0;    //начальное значение количества ненулевых элементов
for (i=1; i<N; i++)    //в цикле
if (B[i]!=0)    //вычисляются
{
    Sum=Sum+B[i];    //сумма ненулевых элементов
    Kol=Kol+1;    //и количество ненулевых элементов
}
printf("Массив В\n");
for (i=0; i<N; i++)    //вывод массива В одной строкой
printf("%d ",B[i]);
printf("\n");
if (Kol>0)    //если количество ненулевых элементов положительное,
{    //то вычисление среднего арифметического
    Sr=Sum/Kol;    //ненулевых элементов массива В
    printf("Среднее арифметическое равно %lf ",Sr);
}
else printf("Все нули!\n");    //иначе вывод «Все нули!»
getchar();
return 0;
}

```

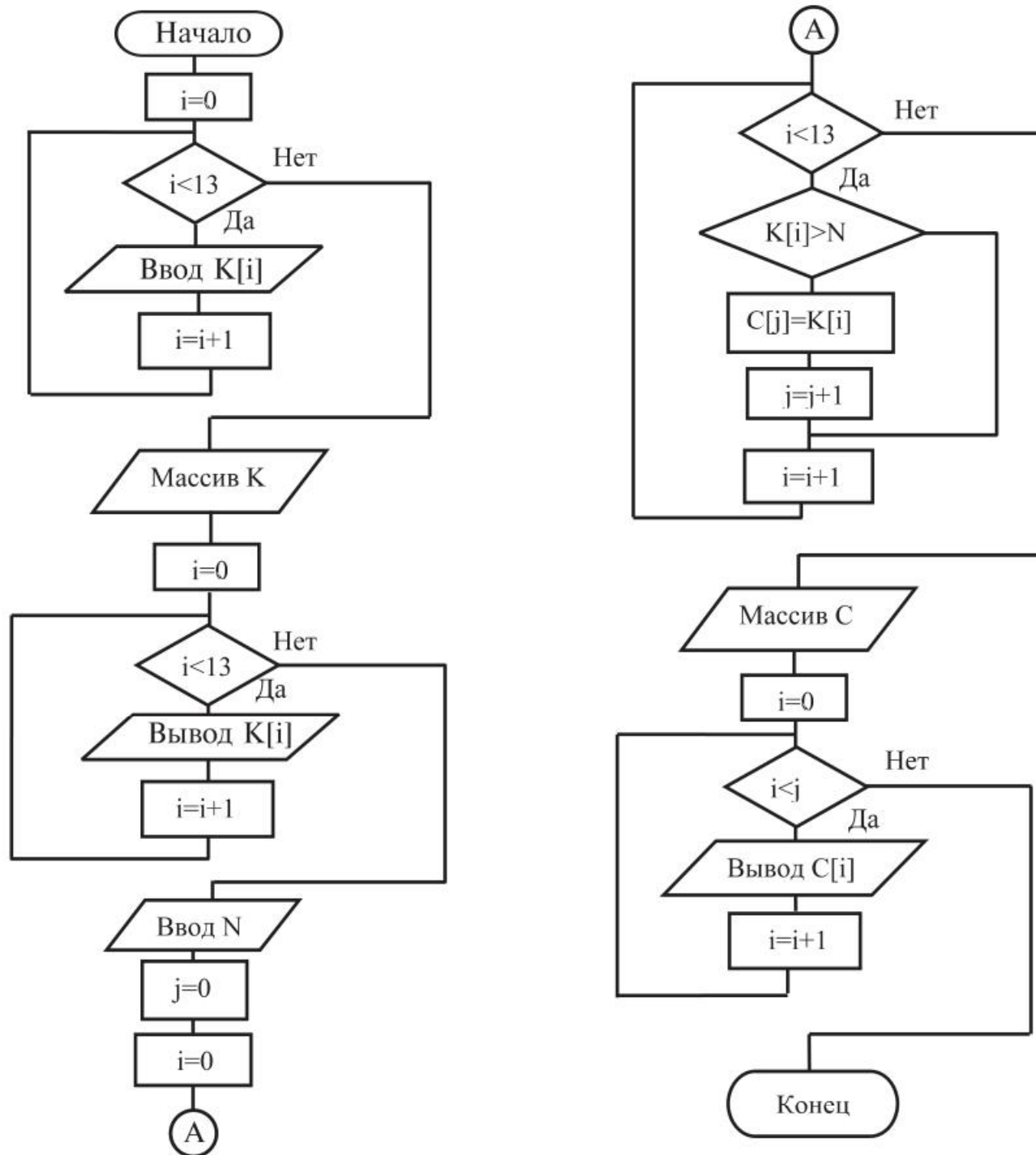
Пояснение. В начале программы с клавиатуры по запросу вводится число элементов (N) в массиве и инициализируется генератор случайных чисел. В цикле (1-й оператор **for**) случайным образом формируются целочисленные значения элементов массива В. В следующем цикле (2-й оператор **for**) вычисляются сумма и количество ненулевых элементов массива. Далее на экран выводится массив В одной строкой (3-й оператор **for**). Если в массиве есть ненулевые элементы (Kol>0), то вычисляется и выводится на экран значение среднего арифметического ненулевых элементов (Sr). Если же ненулевых элементов в массиве нет, то на экран выводится сообщение «Все нули!».

Обратите внимание: количество элементов массива заранее задано в программе $N1$ и максимально может равняться 100. Если пользователь введет значение больше 100, то при работе с массивом возникнут проблемы.

Для корректного формирования и дальнейшей работы с массивами, размер которых не известен, до начала компиляции программы необходимо использовать динамические массивы. Данный материал будет рассмотрен далее в этой главе.

Задача 3. Дан массив K из 13 элементов и целое число N (вводится с клавиатуры). Из элементов массива K , больших числа N , сформировать массив C . Вывести оба массива на экран.

1. Схема алгоритма



Пояснение. С клавиатуры вводятся элементы массива К и число N. В переменной j будет храниться индекс последнего элемента массива С, т. е. j равно количеству элементов в массиве С (в массиве С может быть не более 13 элементов). До формирования массива С переменная j = 0 (в массиве С еще нет элементов). В цикле каждый элемент массива К сравнивается с N. Если $K_i > N$, то индекс j увеличивается на 1 (в массиве С появился новый элемент), и значение элемента K_i записывается в массива С с индексом j. Если же $K_i \leq N$, то происходит переход к следующему элементу массива К (в массив С ничего не записывается). Далее на экран выводится сформированный массив С.

2. Программа

```
#include <stdio.h>
const int N1=13;
int main() {
    int K[N1], C[N1], N, i, j;
    for(i=0; i<N1; i++){                                //в цикле
        printf("Введите K[%d] ", i);
        scanf("%d", &K[i]);
    }
    printf("Массив K\n");
    for (i=0; i<N1; i++)                                  //в цикле
        printf("%d ", K[i]);
    printf("\n");
    printf("vvedite chislo N ");                          //запрос ввода числа N
    scanf("%d", &N);                                       //ввод числа N
    j=0;           //присвоение начального значения номеру элемента в массиве С
    for (i=0; i<N1; i++)                                   //в цикле
        if (K[i]>N)                                         //если элемент массива К больше N
        {C[j]=K[i];                                         //запишем его на j-е место массива С
        j=j+1;                                              //увеличим на 1 число элементов массива С
        }
    printf("Массив C\n");
    for (i=0; i<j; i++)                                    //в цикле
        printf("%d ", C[i]);                               //вывод элемента массива С
    printf("\n");
    getchar();
    return 0;
}
```

Задача 4. С помощью генератора случайных чисел сформировать массив А из 15 элементов. Сформировать массив С из элементов массива А, имеющих нечетные значения. Вывести оба массива на экран.

Найти количество положительных элементов массива А, расположенных перед минимумом, и произведение отрицательных элементов массива С, расположенных после максимума.

Примечание. Поясним условие задачи на рисунке:

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]	A[13]	A[14]	A[15]
A	3	-1	6	15	2	14	0	-13	7	8	-3	10	8	-1	5

Min

Количество положительных элементов массива А, стоящих перед минимальным, равно 5

	C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]
C	3	-1	15	-13	7	-3	-1	5

Max

Произведение отрицательных элементов массива С, стоящих после максимального, равно (-39)

Программа

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N=15; //определение константы — числа элементов массива
int main() {
    int A[N],C[N],i,j,Min, Imin, Max, Imax, Kol;
    Double P;
    srand(time(NULL)); //инициализация генератора случайных чисел
    for (i=0; i<N; i++) //в цикле случайным образом
        A[i]=rand()%10-5; //формируются элементы массива А
    printf("Массив А\n");
    for (i=0; i<N; i++)
        printf("%d ",A[i]);
    printf("\n");
    j=0; //присвоение начального значения номеру элемента в массиве С
    for (i=0; i<N; i++) //в цикле
        if (A[i]%2 != 0) //если A[i] имеет нечетное значение
            {C[j]=A[i]; //запишем его на j-е место массива С
            j=j+1; //увеличим на 1 число элементов массива С
            }
    printf("Массив С\n");
    for (i=0; i<j; i++)
        printf("%d ",C[i]);
    printf("\n");
    Min=A[0]; //присвоим предполагаемому минимуму значение A[0]
```



```

Imin=0; //присвоим Imin значение 0
for (i=1; i<N; i++) //в цикле
if (A[i]<Min) //если A[i] меньше предполагаемого минимума
{Min=A[i]; //меняем предполагаемый минимум
Imin=i; //и его номер
}
printf("Минимальный элемент массива A = %d\n",Min);
printf("Индекс минимального элемента =%d\n", Imin);
Max=C[0]; //присвоим предполагаемому максимуму значение C[0]
Imax=0; //присвоим Imax значение 0
for (i=1; i<j; i++) //в цикле
if (C[i]>Max) //если C[i] больше предполагаемого максимума
{Max=C[i]; //меняем предполагаемый максимум
Imax=i; //и его номер
}
printf("Максимальный элемент массива C = %d\n",Max);
printf("Индекс максимального элемента =%d\n", Imax);
Kol=0; //присвоение начального значения переменной Kol
for (i=0; i<Imin; i++) //в цикле
if (A[i]>0) //если A[i]>0
Kol=Kol+1; //увеличить на 1 значение переменной Kol
printf("Кол-во положит. элементов массива A распол.
перед минимумом =%d\n",Kol);
P=1; //присвоение начального значения переменной P
fp=0; //присвоение начального значения переменной fp
for (i=Imax+1; i<j; i++) //в цикле
if (C[i]<0) //если C[i]<0
{P=P*C[i]; //умножение старого значения P на C[i]
fp=1; //присвоение переменной fp значения 1
}
if (fp == 1)
//если после максимума в массиве C были отрицательные элементы
printf("Произ. отр. элементов массива C, расп. после
максимума=%lf\n",P);
else printf("Отрицательных элементов после максимума
в массиве C не было\n");
getchar();
return 0;
}

```

Пояснение. Для решения данной задачи надо выполнить следующие действия:

- с помощью генератора случайных чисел сформировать массив A из 15 элементов;

- сформировать одномерный массив C из элементов массива A , имеющих нечетные значения (в цикле проверяется остаток от целочисленного деления каждого элемента массива A на 2; если этот остаток не равен нулю, то элемент $A[i]$ — нечетный, поэтому значение $A[i]$ записывается в элемент массива $C[j]$ и j увеличивается на 1);
- вывести массивы A и C на экран любым способом (одной строкой);
- найти минимальный элемент массива A (\min) и обязательно его индекс (i_{\min});
- найти максимальный элемент массива C (\max) и обязательно его индекс (i_{\max});
- вычислить количество положительных элементов массива A , стоящих перед минимальным, т. е. элементов с индексами от 0 до $(i_{\min} - 1)$, и вывести полученное количество на экран, если в массиве A есть положительные элементы, стоящие перед минимальным;
- вычислить произведение отрицательных элементов массива C , расположенных после максимального, т. е. элементов с индексами от $(i_{\max} + 1)$ до $j - 1$, и вывести его на экран, если в массиве C есть отрицательные элементы, стоящие после максимального, т. е. флаг $fp=1$.

8.5. Динамические одномерные массивы

Понятие динамического массива. При использовании статических массивов область памяти под массив резервирует компилятор, затем компилятор записывает адрес начала этой области в имя массива. После этого имя статического массива становится неизменяемым указателем на массив и содержит адрес нулевого элемента массива (адрес массива). Например, при использовании операторов

```
const int n=5;      //задание константы n — числа элементов массива
double mas[n];      //описание массива mas типа double
```

компилятор по константе n определит число элементов массива ($n=5$), зарезервирует область памяти под n ($n=5$) элементов типа `double` и адрес этой области памяти (номер ее первого байта — адрес нулевого элемента массива) запишет в имя массива `mas`. После чего

имя массива `mas` станет равно адресу нулевого элемента массива (адресу массива) и станет неизменяемым указателем на массив.

Если в программе возникает необходимость использовать массив, количество элементов в котором заранее неизвестно, то определить его статическим способом невозможно, так как компилятору неизвестно количество элементов массива и, соответственно, неизвестен размер области, которую необходимо зарезервировать. И следующая последовательность операторов

```
int n;           //описание переменной n — числа элементов массива
n = 5;          //присваивание числу элементов массива значения 5
double mas[n];   //описание массива mas типа double
```

при компиляции выдаст ошибку о том, что при описании массива в качестве количества элементов необходима константа.

Таким образом, при использовании массивов, количество элементов в которых заранее неизвестно, приходится пользоваться динамически распределяемой памятью, а полученные таким образом массивы называются *динамическими*.

Для динамических массивов память выделяется в программе в тот момент, когда это необходимо программисту, и количество элементов можно либо ввести с клавиатуры, либо вычислить по определенному алгоритму. При этом выделяется область памяти, которая не имеет своего имени (неименованная область памяти). Доступ к выделенной области возможен только через указатель — имя динамического массива, который содержит номер первого байта выделенной области памяти и является указателем на массив.

Объявление динамических массивов. Динамические массивы объявляются точно так же, как динамические переменные, тип которых совпадает с типом элементов массива:

```
<тип_элементов_массива> *<имя_массива>;
```

Здесь `<тип_элементов_массива>` может быть любым допустимым в языке C++ типом;

`<имя_массива>` — является указателем, т. е. может содержать адрес некоторой области памяти, в которой может находиться один или несколько элементов заданного типа.

Например, в следующем фрагменте программы

```
int n;           //описание целой переменной — числа элементов массива
int *A;          //описание целочисленного динамического массива
```

описываются целочисленная переменная `n` и динамический массив `A`.

Выделение и освобождение памяти для динамического массива. Для выделения памяти под динамический массив используется следующий оператор:

```
<имя_массива> = new <тип_элемент_массива> [<кол-во_элементов_массива>];
```

Например, фрагмент программы

```
...
printf("Vvedite chislo elementov massiva ");
//вывод приглашения к вводу числа элементов массива
scanf("%d",&n); //ввод числа элементов массива
A=new int[n]; //выделение памяти под n элементов целочисленного массива A
...
```

выделяет память под массив, число элементов которого вводится с клавиатуры.

Когда память, выделенная под динамический массив, больше не нужна, ее следует освободить. Делается это с помощью оператора delete:

```
delete []имя_динамического_массива;
```

Пример:

```
delete []A;
```

Повторное применение оператора delete [] к тому же самому указателю приведет к зависанию программы. Рекомендуется при освобождении памяти присваивать связанному с ней указателю нулевое значение. Повторный вызов оператора delete [] для нулевого указателя пройдет безболезненно для программы, например:

```
int n; //описание целой переменной — числа элементов массива
int *A; //описание целочисленного динамического массива
printf("Vvedite chislo elementov massiva ");
//вывод приглашения к вводу числа элементов массива
scanf("%d",&n); //ввод числа элементов массива
A=new int[n];
//выделение памяти под n элементов целочисленного массива A
...
delete []A;
A = NULL;
delete []A;
```

При освобождении памяти с самим указателем ничего не происходит, и его можно снова использовать, присвоив ему новый адрес.

Адреса элементов массива. После выделения области памяти под массив (при статическом или динамическом распределении памяти) адрес начала этой области (номер ее первого байта) записывается в имя массива. Имя массива становится указателем на массив и содержит адрес нулевого элемента массива (адрес массива).

При использовании статических массивов область памяти под массив резервирует компилятор, затем компилятор записывает адрес начала этой области в имя массива. После этого имя статического массива становится неизменяемым указателем на массив и содержит адрес нулевого элемента массива (адрес массива).

При использовании динамических массивов выделение памяти происходит при использовании операции `new` с указанием числа элементов массива, например:

```
A=new int[n];
```

При этом сначала происходит выделение области памяти в куче (динамической памяти компьютера), затем адрес начала этой области памяти записывается в имя динамического массива, который является указателем на массив.

В соответствии с правилами использования арифметических операций над указателями адрес любого элемента массива можно вычислить как сумму адреса массива (адреса нулевого элемента) и индекса элемента в массиве (второй столбец табл. 8.1). А так как адрес массива содержится в имени массива, то адрес любого элемента массива можно вычислить по формуле, представленной в третьем столбце табл. 8.1.

Таблица 8.1. Вычисление адресов в массиве

Адреса элементов массива	Формулы для вычисления адресов	Формулы для вычисления адресов
Адрес 0-го элемента массива	Адрес массива + 0	Имя_массива + 0
Адрес 1-го элемента массива	Адрес массива + 1	Имя_массива + 1
Адрес 2-го элемента массива	Адрес массива + 2	Имя_массива + 2
Адрес <i>i</i> -го элемента массива	Адрес массива + <i>i</i>	Имя_массива + <i>i</i>

Доступ к элементам массива. Используя операцию разыменования, можно по адресу *i*-го элемента получить его значение —

$*(\text{Имя_массива} + i)$, (второй столбец табл. 8.2). Для массивов существует и другой, более удобный способ записи операции разыменования — $\text{Имя_массива}[i]$ (третий столбец табл. 8.2). Например, для динамического массива A , описанного ранее, значение i -го элемента массива можно получить двумя тождественными способами — $*(A + i)$ и $A[i]$.

Таблица 8.2. Определение значений элементов в массиве

Значения элементов массива	Формулы для определения значений	Формулы для определения значений
Значение 0-го элемента	$*(\text{Имя_массива} + 0)$	$\text{Имя_массива}[0]$
Значение 1-го элемента	$*(\text{Имя_массива} + 1)$	$\text{Имя_массива}[1]$
Значение 2-го элемента	$*(\text{Имя_массива} + 2)$	$\text{Имя_массива}[2]$
Значение i -го элемента	$*(\text{Имя_массива} + i)$	$\text{Имя_массива}[i]$

Решение задач с использованием динамических массивов. Рассмотрим решение задач с использованием динамических массивов.

Пример. Ввести с клавиатуры количество элементов целочисленного массива. Сформировать массив с помощью датчика случайных чисел. Вывести массив на экран. Найти произведение всех элементов массива.

```
#include <stdio.h>           //подключение стандартной библиотеки
                               //ввода-вывода
#include <stdlib.h>
                               //подключение библиотеки работы с датчиком случайных чисел
#include <time.h> //подключение библиотеки для работы с таймером
void main() {
    int *X = NULL;           //описание указателя на массив X
    int n;                   //описание переменной n — числа элементов массива
    int i;                   //описание переменной i — параметра цикла
    srand((unsigned int) time(NULL)); //запуск датчика
                               //случайных чисел
    printf("Введите число элементов массива n = ");
                               //вывод приглашения к вводу числа элементов массива
    scanf("%d", &n);         //ввод числа элементов массива
    X = new int[n];          //резервирование памяти под массив X
    for (i = 0; i < n; i++)   //в цикле
    for (i = 0; i < n; i++)   //в цикле
        X[i] = rand() % 101 - 50;
```



```

    //формирование элемента массива с помощью датчика случайных чисел
printf("Массив X\n");           //вывод заголовка с именем массива
for (i=0; i<n; i++)              //в цикле
printf("%d ",X[i]);              //вывод элемента массива
printf("\n");                    //перевод курсора на начало следующей строки экрана
    Pr=1;                        //присвоение начального значения произведения (Pr)
    for (i=0; i<N; i++)          //в цикле
        Pr=Pr*X[i];             //умножение старого значения Pr на элемент массива
printf("Произведение всех элементов массива Pr =
    %lf\n", Pr);                 //вывод на экран значения Pr
getchar();
getchar();
}

```

8.6. Массивы указателей

Иногда необходимо использовать массивы указателей, т.е. одномерные массивы, элементами которых являются указатели, содержащие адреса областей памяти или пустые значения. Статические массивы указателей описываются следующим образом:

```
<тип>*    <имя_массива>[<количество элементов>];
```

Здесь `тип*` — задает тип указателя, содержащего адрес области памяти заданного типа `<тип>` или пустое значение, например:

```
int*    NUM[10];           //описание массива указателей на тип int
```

Значение указателей — элементов массива при этом неопределенно. Для правильной работы программы желательно в самом начале задать им пустые значения (NULL), например, для массива указателей NUM

```
for (i=0; i<10; i++)
NUM[i]=NULL;
```

При работе с указателями — элементами массива пользуются теми же приемами, что и при работе с «обычными» указателями.

Примечание. Динамические массивы указателей и работа с ними будут рассмотрены в главе 10 в разделе «Динамические матрицы».

Решение задач с использованием массивов указателей. Рассмотрим решение задач с использованием массивов указателей.

Пример 1. Задать фамилии четырех учеников и с использованием массива указателей вывести их на экран.

```
#include <stdio.h>           //подключение стандартной библиотеки
                               //ввода-вывода
void main() {                 //заголовок функции main
    char name1[] = "Иванов";
                               //описание и инициализация символьного массива name1
    char name2[] = "Сидоров";
                               //описание и инициализация символьного массива name2
    char name3[] = "Козлов";
                               //описание и инициализация символьного массива name3
    char name4[] = "Крикетов";
                               //описание и инициализация символьного массива name4
    char* mas[4];              //описание массива указателей на тип char
    mas[0] = name1;
                               //0-й элемент массива указателей содержит указатель (адрес)
                               //на массив name1
    mas[1] = name2;
                               //1-й элемент массива указателей содержит указатель (адрес)
                               //на массив name2
    mas[2] = name3;
                               //2-й элемент массива указателей содержит указатель (адрес)
                               //на массив name3
    mas[3] = name4;
                               //3-й элемент массива указателей содержит указатель (адрес)
                               //на массив name2
    for (int i = 0; i<4; i++ ) //в цикле
        printf("%d - ученик - %s\n", i, mas[i]);
                               /*вывод на экран значения i и символьного массива,
                               адрес которого содержится в указателе mas[i]*/
    getchar();
}
```

Пример 2. С помощью датчика случайных чисел создать массив из N целых чисел в диапазоне от 0 до 9. Сформировать одномерные массивы из номеров элементов исходного массива, значения которых равны числам из диапазона от 0 до 9.

```
#include <stdio.h>           //подключение стандартной
                               //библиотеки ввода-вывода
#include <stdlib.h>
                               //подключение библиотеки для работы с датчиком случайных чисел
#include <time.h>             //подключение библиотеки работы с таймером
```

```

void main() {
    int *X;
    //описание указателя на исходный массив целых чисел
    int N;
    //описание целой переменной N
    int Kol[10];
    //описание массива количеств появления каждого из чисел от 0 до 9
    int K[10];
    /*описание массива с текущими количествами
    элементов в результирующих массивах */
    int i, j;
    //описание целых переменных i, j — параметра цикла
    int* NUM[10];
    //описание массива указателей на тип int
    for (i=0; i<10; i++)
        Kol[i]=0;
    //в цикле
    //задание начальных значений количеств
    //появлений каждого из чисел от 0 до 9*/
    printf("Введите количество элементов в массиве N=");
    //приглашение к вводу переменной N
    scanf("%d",&N);
    //ввод с клавиатуры N
    srand((unsigned int) time(NULL));
    //запуск датчика случайных чисел
    X=new int[N];
    //резервирование памяти под исходный массив
    for (i=0; i<N; i++)
    {
        X[i]=rand()%10;
        //случайным образом формирование элемента массива
        Kol[X[i]]=Kol[X[i]]+1;
        //увеличение на 1 числа появлений значения X[i]
    }
    for (i=0; i<10; i++)
    {
        Num[i]=new int[Kol[i]];
        //резервирование памяти под массивы
        //из номеров элементов, равных i в исходном массиве X*/
        K[i]=0;
    }
    for (i=0; i<N; i++)
    {
        NUM[X[i]][K[X[i]]]=i;
        //записываем номер X[i] в соответствующий результирующий массив
        K[X[i]]=K[X[i]]+1;
        //увеличиваем на 1 количество уже записанных номеров
    }
    printf("Nomera elementov v ishodnim massive\n");
    for (i=0; i<10; i++) {
        printf("Nomera znachenia%d: ",i);
        //вывод значения
        for (j=0; j<Kol[i]; j++)
            printf("%d ",NUM[i][j]);
        //в цикле
        //вывод номеров значения i в исходном массиве
    }
}

```



```
printf("\n"); //перевод курсора на начало следующей строки экрана
getchar();
getchar();
}
```

Тесты

1. Дан массив $A = (-5 \ 6 \ 0 \ 4 \ 0 \ -7)$. Что будет сформировано в переменной X после выполнения следующего фрагмента программы?

```
...
X=0;
for (i=0; i<6; i++)
if (A[i]>0)
    X=X+A[i];
...
```

- а) сумма отрицательных элементов целочисленного массива A ($X=-12$);
- б) сумма положительных элементов целочисленного массива A ($X=10$);
- в) сумма всех элементов массива A ($X = -2$).

2. Дан массив $A = (-5 \ 6 \ 0 \ 4 \ 0 \ -7)$. Что будет сформировано в переменной X после выполнения следующего фрагмента программы?

```
...
X=0;
for (i=0; i<6; i++)
if (A[i]!=0)
    X=X+1;
...
```

- а) количество ненулевых элементов ($X=4$);
- б) количество отрицательных элементов ($X=2$);
- в) количество положительных элементов ($X=2$).

3. Дан массив $A = (-5 \ 6 \ 0 \ 4 \ 0 \ -7)$. Что будет сформировано в переменной X после выполнения следующего фрагмента программы?

```
...
X=1;
for (i=0; i<6; i++)
if (A[i]>0)
    X=X*A[i];
...
```

- а) произведение отрицательных элементов ($X=35$);
- б) произведение положительных элементов ($X=24$);
- в) произведение всех элементов ($X=0$).

4. Каков будет результат выполнения следующего фрагмента программы?

```
...
int A[10], P, Q, I;
...
Q=A[0];
P=A[0];
for(i=0; i<10; i++)
{
    if (A[i]<P)
        P=A[i];
    if (A[i]>Q)
        Q=A[i];
}
```

- а) в переменных Q и P будут содержаться непредсказуемые значения;
- б) в переменной Q будет содержаться значение максимального элемента массива A, а в переменной P — значение минимального элемента;
- в) в переменной Q будет содержаться значение минимального элемента массива A, а в переменной P — значение максимального элемента.

5. Какой фрагмент программы формирует массив B, содержащий только отрицательные элементы массива A?

- а) ...

```
for (i=0; i<10; i++)
    if (A[i]<0)
        B[i]=A[i];
...
```
- б) ...

```
j=0;
for (i=0; i<10; i++)
{
    if (A[i]<0)
        B[j]=A[i];
    j=j+1;
}
...
```
- в) ...

```
j=0;
for (i=0; i<10; i++)
```

```
if (A[i]<0)
{
    B[j]=A[i];
    j=j+1;
}
...
```

6. Что делает следующий фрагмент программы с массивом А, содержащим 10 элементов?

```
...
M=A[0];
for (i=0; i<9; i++)
A[i]=A[i+1];
A[9]:=M;
...
```

- а) присваивает каждому элементу массива значение следующего элемента, а последнему — значение 0-го элемента;
- б) каждый элемент массива увеличивает на 1;
- в) фрагмент содержит ошибки и работать не будет.

7. Что делает следующий фрагмент программы с массивом А, содержащим N элементов?

```
...
for (i=0; i<N/2; i++)
{
    M=A[i];
    A[i]=A[N-1-i];
    A[N-1-i]=M;
}
...
```

- а) присваивает каждому элементу массива значение следующего элемента, а последнему — значение 1-го элемента;
- б) записывает элементы массива в обратном порядке;
- в) фрагмент содержит ошибки и работать не будет.

8. Дан массив А= (-9 0 2 -6 5 4). Что будет сформировано в переменной X после выполнения следующего фрагмента программы?

```
...
X=0;
for (i=0; i<6; i++)
if ((A[i]<0) && (A[i] % 2 != 0))
    X=X+A[i];
...
```


- а) сумма отрицательных нечетных элементов целочисленного массива A ($X = -9$);
- б) произведение положительных четных элементов целочисленного массива A ($X = 8$);
- в) количество ненулевых элементов массива A ($X = 5$).

Задания

Для каждой задачи составить схему алгоритма и написать программу.

1. Ввести восемь элементов массива X . Вычислить элементы массива Y по формуле $y[i] = 2 * x[i]$. Вывести массив Y .
2. Ввести с клавиатуры восемь элементов массива X . Вычислить сумму всех элементов массива X . Вывести на экран сумму и массив X .
3. Ввести шесть элементов массива A . Вычислить произведение всех элементов массива A . Вывести массив A .
4. Сформировать массив $K\{21\}$. Вычислить сумму элементов, значение которых больше (-2) .
6. Сформировать массив K из 23 элементов. Вывести его на экран. Вычислить произведение положительных элементов.
7. Ввести массив P из 15 элементов. Найти количество отрицательных элементов в массиве.
8. Ввести массив K из семи элементов. Вычислить сумму отрицательных элементов.
9. Ввести массив B из девяти элементов, определить количество элементов, значение которых меньше 10.
10. Ввести массив B из девяти элементов. Увеличить на 5 все элементы, значение которых меньше 10, и определить их количество.
11. Ввести массив. Найти отдельно произведение положительных элементов и произведение отрицательных элементов.
12. Сформировать массив из 12 элементов. Вывести его на экран, выделив минимальный и максимальный элементы массива (например, восклицательными знаками).
13. Дан массив A из восьми элементов и целое число M (вводится с клавиатуры или формируется с помощью генератора случайных чисел). Из элементов массива A , не превышающих числа M , сформировать массив B . Вывести оба массива на экран. Для обоих массивов найти сумму ненулевых элементов (отдельно для каждого массива).

14. Дан массив C из 11 элементов. Из положительных элементов массива C сформировать массив A . Вывести оба массива на экран. Для обоих массивов найти произведение ненулевых элементов (отдельно для каждого массива).

15. С помощью генератора случайных чисел сформировать массив D из 18 элементов. Вывести его на экран. Найти сумму положительных элементов массива, произведение отрицательных и количество нулей.

16. Ввести массив A из 15 целочисленных элементов. Сформировать массив B из квадратных корней и массив C из квадратов элементов массива A . Вывести массивы B и C .

17. Ввести массив. Найти произведение положительных элементов массива, стоящих после максимального элемента.

18. Дан массив из восьми элементов. Поменять местами минимальный и максимальный элементы. Вывести на экран исходный массив и массив после перестановки минимального и максимального элементов, выделив их (например, восклицательными знаками).

19. Сформировать массив K . Вывести его. Из элементов массива, абсолютное значение которых не превышает пяти, сформировать массив C . Вывести его на экран.

20. Сформировать массив из 12 элементов случайным образом. Вывести его. Найти заданный элемент (вводится с клавиатуры) и вывести его индекс. Если искомый элемент не найден, то вывести сообщение об этом на экран.

21. Дан массив A из N элементов типа `Char`. Сформировать массив B из N элементов типа `Boolean` следующим образом: если элемент $a[i]$ — буква, то элементу $b[i]$ присвоить значение `True`, если же $a[i]$ — цифра, то $b[i]$ присвоить значение `False`.

22. Сформировать массив K из N элементов. Из ненулевых элементов массива K сформировать массив X . Вывести оба массива на экран. Вычислить количество положительных элементов, расположенных после минимального элемента массива X .

23. Сформировать массив A из N элементов. Вывести массив на экран. Определить, есть ли в массиве A элементы, равные числу M .

24. С помощью генератора случайных чисел сформировать массив E из 20 элементов. Сформировать массив F из положительных элементов массива E . Вывести оба массива на экран. Найти сумму нечетных элементов массива F , расположенных перед минимумом, и количество четных элементов массива E , расположенных после максимума.

25. Таблица содержит 100 номеров выигрышных билетов. Проверить, является ли билет с номером N выигрышным.

26. Ввести массивы $A = \{a_i \mid i = 0, 1, \dots, 7\}$, $B = \{b_j \mid j = 0, 1, \dots, 7\}$.

Вычислить массив $C = \{c_k \mid k = 0, 1, \dots, 7\}$:
$$c_i = \frac{\sum_{j=0}^7 a_j}{a_i + b_i}.$$

27. Ввести массивы $A = \{a_i \mid i = 0, 1, \dots, 7\}$, $B = \{b_j \mid j = 0, 1, \dots, 7\}$.

Вычислить $C = (\max_{0 \leq i \leq 7} \{a_i - b_i\}, \max_{0 \leq i \leq 7} \{b_i\})$.

28. Сформировать массив B из 30 элементов случайным образом. Вывести его. Из массива сформировать массив A таким образом, чтобы в массиве A были только неповторяющиеся элементы массива B .

29. С помощью генератора случайных чисел сформировать массив $A\{20\}$. Вывести его на экран. Сформировать одномерный массив B из положительных элементов массива A . Вывести полученный массив на экран. Определить среднее арифметическое отрицательных элементов массива A и минимальный элемент и его координаты массива B .

30. Сформировать массив K . Вывести его. Из элементов массива, абсолютное значение которых не превышает пяти, сформировать массив C . Вывести его на экран. Отсортировать массив K по неубыванию двумя способами (линейная и быстрая с разделением). Вывести массив K после каждой сортировки. Отсортировать массив C по убыванию двумя способами (пузырьковая и быстрая с разделением). Вывести массив C после каждой сортировки.

31. Сформировать массив из 12 элементов случайным образом. Вывести его. Упорядочить массив по возрастанию любым способом. Вывести упорядоченный массив. Найти заданный элемент (вводится с клавиатуры) и вывести его индекс. Если искомый элемент не найден, то вывести сообщение об этом на экран. Использовать метод бинарного поиска и метод линейного поиска (для каждого метода поиска написать свою программу).

32. Сформировать массив P из N элементов. Из элементов массива, меньших числа K , сформировать массив M . Вывести оба массива на экран. Массив P упорядочить по убыванию (метод «пузырька») и вывести на экран после сортировки. Вычислить произведение элементов, расположенных перед максимальным элементом массива M .

33. Сформировать массив K из N элементов. Из ненулевых элементов массива K сформировать массив X . Вывести оба массива на экран. Массив K упорядочить по возрастанию (быстрая сортировка с разделением) и вывести на экран после сортировки. Вычислить количество положительных элементов, расположенных после минимального элемента массива X .

34. Сформировать массив A из N элементов. Вывести массив на экран. Массив A упорядочить по возрастанию (любым способом) и вывести

на экран после сортировки. Определить, есть ли в массиве A элементы, равные числу M (использовать бинарный поиск).

35. Сформировать массив B из 30 элементов случайным образом. Вывести его. Из массива сформировать массив A таким образом, чтобы в массиве A были только неповторяющиеся элементы массива B . Упорядочить массив A по возрастанию. Вывести массив A . Организовать бинарный поиск заданного элемента C в упорядоченном массиве.

Контрольные вопросы

1. Что такое «одномерный массив»? Приведите примеры.
2. Данные каких типов могут быть элементами массива? Приведите примеры.
3. При каких условиях может считаться одномерным массивом последовательность, состоящая из букв латинского алфавита и цифр от 0 до 9? Объясните.
4. Назовите атрибуты одномерного массива. Поясните на примерах.
5. Как объявить одномерный массив в программе? Приведите примеры.
6. Можно ли в программе объявить пользовательский тип «массив»? Как это сделать? Приведите примеры.
7. Что такое «индекс элемента массива»? Приведите примеры.
8. Как получить доступ к конкретному элементу массива? Приведите примеры.
9. Что такое «доступ» к элементам массива?
10. Какие действия можно осуществлять с одномерными массивами?
11. Как сформировать элементы целочисленного массива, лежащие в заданном диапазоне?
12. Как сформировать элементы вещественного массива, лежащие в заданном диапазоне?
13. Как вывести массив на экран? Приведите примеры.
14. Для чего в программе удобно использовать массив? Приведите пример.
15. Как вычислить сумму всех элементов массива? Приведите пример.
16. Как вычислить произведение всех элементов массива? Приведите пример.
17. Как найти среднее арифметическое всех элементов массива?
18. Как определить максимальный элемент массива? Приведите пример.
19. Как определить индекс максимального элемента массива? Приведите пример.
20. Как найти минимальный элемент массива? Приведите пример.
21. Как определить индекс минимального элемента массива? Приведите пример.
22. Как найти все максимальные (минимальные) элементы массива? Приведите примеры.

Глава 9

РАБОТА СО СТРОКАМИ

Понятие строки. Строка представляет собой массив символов, заканчивающийся нуль-символом — символом с кодом 0 (`'\0'`). По положению нуль-символа определяется фактическая длина строки. Длина строки равна количеству символов, расположенных до нуль-символа. При резервировании памяти под строку необходимо выделить количество байт на 1 большее максимально возможной длины строки, так как один байт занимает символ — признак конца строки.

В программе строки описываются следующим образом:

```
char <имя_строки>[число_элементов];
```

где `<число_элементов>` определяется как максимально необходимое число символов + 1 (для нуль-символа).

Инициализация строк. Инициализация строк осуществляется по тем же правилам, что и инициализация одномерного массива, например:

```
char st1[6]=('H','e','l','l','o','\0');  
           //инициализация строковой переменной массивом символов  
char st2[]=(' ','w','o','r','l','d','!','\0');  
           //инициализация строковой переменной массивом символов
```

Существует и другой, более удобный способ инициализации строк, когда в качестве инициализирующего значения используется строковая константа, например:

```
char st1[6]="Hello";  
           //инициализация строковой переменной строковой константой  
char st2[]=" world!";  
           //инициализация строковой переменной строковой константой  
char *ptext="Указатель инициализирован этой строкой.";
```


При инициализации указателей на строку компилятор добавит к строковой константе нуль-символ, затем подсчитает количество символов с учетом нуль-символа. При выполнении программы в динамической области памяти будет выделено количество байт, равное количеству символов в строковой константе с учетом нуль-символа, адрес этой области будет записан в указатель на строку и затем в выделенную область заносится инициализирующее значение.

Работа со строками. Строки можно вводить с клавиатуры, выводить на экран, сравнивать друг с другом, объединять несколько строк в одну, разбивать на слова, сортировать по алфавиту и т. д.

Приведем простой пример работы со строками:

```
# include <stdio.h>
# include <string.h>

int main()
{
    char str1[] = "Hello, World!";
    char str2[7] = "Hello,";
    char *str3 = " World!";
    printf("%s\n", str1);
    printf("%s", str2);
    printf("%s\n", str3);
    return 0;
}
```

Пояснение. В этом примере при объявлении переменной `str1` количество символов не указано. В этом случае компилятор сам посчитает необходимое число байт для хранения строки и зарезервирует память. Строка `str2` обязательно должна состоять из семи символов, включая символ — признак конца строки. В случае несоответствия будет сгенерирована ошибка компилятора. Для объявления третьей строки используется указатель на массив символов. Все строки выводятся на экран стандартной функций `printf`, с форматной спецификацией «%s».

9.1. Функции работы со строками

Функции ввода строки `gets()` и `scanf()` библиотеки `stdio`. Функция `gets(char *)` читает вводимые с клавиатуры символы до тех пор, пока не встретит символ новой строки (`'\n'`), который создается при нажа-

тии клавиши <ввод>. Функция записывает в строковую переменную все символы до символа новой строки, не включая его, присоединяет к ним нуль-символ ('\0') и возвращает указатель на введенную строку вызывающей программе.

Пример. Ввести строку символов, вывести ее на экран, вывести пятый символ строки. Для доступа к символам строки использовать указатели.

```
#include <stdio.h>
int main()
{
    char text[81], *ptext;
    //описание статической строки text и указателя на строку ptext
    printf("Введите строку символов (<=10)");
    ptext = gets(text); /*ввод строки text с клавиатуры и присваивание
    адреса строки (значения указателя на строку) text указателю ptext*/
    printf("%s \n %s \n", text, ptext);
    //вывод введенной строки двумя способами
    printf("%c \n %c \n", text[5], *(ptext+5));
    //вывод 5-го символа введенной строки двумя способами
    getchar();
    return 0;
}
```

Для функции `scanf()` строка начинается с первого непустого, т. е. отличного от пробела, символа табуляции и символа новой строки, введенного символа. Если строка вводится по формату `%s`, то строка продолжается до следующего пустого символа (пробел, табуляция, новая строка). Функция `scanf()` возвращает целое значение, равное числу считанных по форматной спецификации `%s` строк, если ввод прошел успешно.

Пример использования функции `scanf()` для ввода строки. Необходимо ввести две последовательности символов и вывести их на экран.

```
#include <stdio.h>
int main()
{
    char text1[81], text2[81];
    int number;
    printf("Введите 2 последовательности символов через пробел ");
    number=scanf("%s %s",text1, text2);
}
```

```
printf("Функция scanf() считала %d строки: \" %s\" и  
\"%s\" \n", number, text1, text2);  
return 0;  
}
```

При выполнении этого фрагмента программы с клавиатуры можно будет ввести через пробел две последовательности символов, например,

Изучай программирование

Функция ввода `scanf` сначала считывает с клавиатуры последовательность символов — «Изучай», запишет ее в строковую переменную `text1`, добавив в конец строки нуль-символ. Затем функция `scanf` пропустит пустой символ (пробел), считывает с клавиатуры последовательность символов — «программирование» и запишет ее в строковую переменную `text2`, добавив в конец строки нуль-символ. После этого количество считанных значений (2 — результат функции `scanf`) будет присвоено переменной `number`.

Затем полученные результаты с помощью функции `printf` выводятся на экран.

Функция `scanf()` считала две строки: "Изучай" и "программирование".

Рассмотрим решение этой же задачи с использованием функции `gets`. Необходимо ввести с клавиатуры 2 последовательности символов, вывести их на экран.

```
#include <stdio.h>  
int main()  
{  
    char text[81];  
    printf("Введите 2 последовательности символов через  
    пробел ");  
    gets(text);  
    printf("Функция gets считала строку: \" %s\" \n",  
    text);  
    return 0;  
}
```

При выполнении этого фрагмента программы с клавиатуры можно будет ввести через пробел две последовательности символов, например,

Изучай программирование

Функция ввода `gets` считывает с клавиатуры обе последовательности символов как одну последовательность — Изучай программирование, запишет ее в строковую переменную `text`, добавив в конец строки нуль-символ. Действие функции `gets` на этом закончится. Затем введенная строка с помощью функции `printf` выводится на экран.

Функция `gets` считала строку: "Изучай программирование"

Функции вывода строк `puts()` и `printf()` библиотеки `stdio`. Функция `puts(char *)` имеет только один аргумент — указатель на строку. Функция выводит на экран все символы строки, пока ей не встретится нуль-символ. Если с помощью этой функции попытаться вывести на экран массив символов без завершающего нуль-символа, то функция `puts` станет перебирать все ячейки памяти в поисках нуль-символа, программа зависнет, а на экране будет непредсказуемая информация. Любая строка, выводимая с помощью функции `puts()`, печатается с новой строки, но после окончания работы функция `puts()` сама не переводит курсор в начало следующей строки экрана. В вызывающую программу `puts()` возвращает неотрицательное число.

Пример использования функции `puts()` для вывода строки:

```
#include <stdio.h>
int main()
{
    char text1[]="Массив инициализирован этой строкой.";
    char *ptext2="Указатель инициализирован этой строкой.";
    puts("Любая строка легко печатается с помощью
        puts().");
    puts(text1);      puts(ptext2);
    puts(ptext2+3);
        //печатать строки начнется не с первого символа, а с четвертого
    puts(&text1[4]);   //печатать строки начнется с пятого символа
}
```

Обратите внимание, что аргументом функции `puts` является указатель на строку, т. е. адрес первого символа, который нужно вывести на экран, затем выводится следующий символ и т. д., пока не встретится нуль-символ. При вызове функции `puts(ptext2+3);` аргументом этой функции будет указатель на символ с индексом 3 (адрес символа с индексом 3, т. е. указатель (адрес) на 4-й символ строки, начиная с которого и будет происходить вывод).

При вызове функции `puts(&text1[4])`; аргументом функции будет адрес символа с индексом 4, т. е. адрес 5-го символа строки, начиная с которого и будет происходить вывод.

Функция `printf()` менее удобна, чем `puts()`, но она предоставляет дополнительные возможности вывода строк на экран. Она использует указатель строки в качестве аргумента, но не выводит каждую строку текста с новой строки. Зато с помощью функции `printf()` можно объединить последовательности символов для печати в единую строку. В вызывающую программу `printf()` возвращает число байт выводимой на экран информации. Приведем пример использования функции `printf`.

```
char text[81], *ptext;
    //описание статической строки text и указателя на строку ptext
printf("Введите строку символов: ");
ptext=gets(text); /*ввод строки text с клавиатуры и присваивание
    адреса строки (значения указателя на строку) text указателю ptext*/
printf("%s \n %s \n", text, ptext);
    //вывод введенной строки двумя способами
```

9.2. Работа со строками как с массивом символов

Так как строка представляет собой массив символов, то можно работать со строками как с обычными массивами. При этом необходимо помнить, что строка обязательно должна заканчиваться нуль-символом.

Пример 1. С клавиатуры ввести строку символов, подсчитать длину строки.

```
#include <stdio.h>
int main()
{
    char str[256], ch;
    int count = 0;
    printf("Введите строку >> ");
    gets(str);
    ch = str[0];
    while (ch != '\0')
    {
        count++;
        ch = str[count];
    }
}
```

```

    printf("Длина введенной строки = %d\n", count);
    return 0;
}

```

Пример 2. Ввести строку, состоящую из нескольких предложений. Предложение заканчивается точкой. Определить число предложений в строке.

```

#include <stdio.h>
int main()
{
    char str[256];                //описание строки
    int k;                        //описание целой переменной — длины строки
    int count;                    //описание переменной — количества предложений
    printf("Введите строку >> ");
    gets(str);                    //ввод строки
    k=0;                          //задать начальное значение длине строки (k=0)
    count=0;                      //задать нач. значение количеству предложений
    while (str[k]!='\0')          //пока k-й символ не равен нуль-символу
    {
        if(str[k]=='.')           //если k-й символ равен точке
            count=count+1;        //увеличить на 1 количество предложений
        k=k+1;                    //увеличить на 1 длину строки
    }
    printf("Длина введенной строки = %d\n", k);
    printf("Количество предложений в строке = %d\n", count);
    getchar();
    return 0;
}

```

При работе со строками часто используются указатели. Используя указатель на начало строки, можно получить доступ ко всем ее символам.

Пример 3. Инициализировать строку, объявить два указателя на строку, инициализировать один из них адресом заданной строки, присвоить значение первого указателя второму. Напечатать строку, на которую указывают оба указателя, и их значения. Напечатать нулевой и шестой символы строки.

```

#include <stdio.h>
int main()
{
    char *text="Не делай глупостей", *copy;

```



```
copy=text;           //указатель copy теперь указывает на строку text
printf("%s\n", copy);
    //печатать нулевого символа строки (форматная спецификация %с)
printf("Нулевой символ строки: %с \n", * copy);
    //печатать шестого символа строки, copy+6 — указатель на шестой символ
printf("Шестой символ строки: %с \n", *( copy+6));
}
```

Пояснение. В программе строка `text` инициализируется строковой константой «Не делай глупостей». В программе описан указатель на строку `*copy`. С помощью оператора присваивания `copy=text` указателю `copy` присваивается значение указателя `text`, т. е. адрес строки `text`. Затем с помощью указателя `copy` и функции `printf` производится вывод строки на экран.

Так как строка — это массив символов, `copy` — указатель на нулевой символ массива (строки), используя операцию разыменования — `*copy`, получаем значение 0-го символа, который выводим на экран. Используя правила арифметики указателей и операцию разыменования — `*(copy+6)`, получаем значение 6-го символа строки, который выводим на экран.

9.3. Стандартные функции обработки строк

Библиотеки языка C++ содержат функции обработки строк. Прототипы этих функций находятся в файле `string.h`, поэтому в начале программы необходимо подключить заголовочный файл библиотеки `string`. Рассмотрим наиболее часто используемые функции `strlen()`, `strcpy()`, `strcat()`, `strcmp()`, `strch()`:

`int strlen(char *s)` — определяет длину строки `s`, возвращает количество символов в строке, включая и последний нуль-символ (`'\0'`);

`char *strcpy(char *dest, const char *src)` — копирует строку `src` в `dest`, возвращает указатель `dest`;

`char *strcat(char *s1, char *s2)` — присоединяет строку `s2` в конец строки `s1` (предполагается, что длина строки `s1` позволяет это сделать), возвращает указатель на объединенную строку (`s1`);

`int strcmp(char *s1, char *s2)` — сравнивает две строки `s1` и `s2`, возвращает значение, полученное путем вычитания первых несовпадающих символов из `s1` и `s2`;

`char *strchr(char *s, int c)` — находит в строке первое вхождение символа `c`, возвращает указатель на этот символ;

`int strcspn(char *s1, char *s2)` — находит длину отрезка строки `s1`, содержащего символы, не входящие в множество символов строки `s2`.

Пример 1. Ввести две строки. Сравнить их. Объединить в одну. Найти первое вхождение первого символа первой строки во второй строке. Скопировать вторую строку в первую и вывести на экран первую.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char text1[81], text2[20], *ps, c;
    int j;
    puts("Введите первую строку символов");
    gets(text1);
    puts("Введите вторую строку символов");
    gets(text2);

    //иллюстрация работы функции strcmp()
    j= strcmp(text1, text2);
    if (j==0) puts("Строки полностью совпадают");
    if (j<0) puts("Первая строка лексикографически меньше
                второй");
    if (j>0) puts("Первая строка лексикографически больше
                второй");

    //объединение строк в одну с проверкой необходимой памяти
    if (strlen(text1)+strlen(text2)+1<81) {
        strcat(text1, text2);
        puts("Результат объединения двух строк");
        puts(text1);
        /* можно иначе: ps= strcat(text1, text2); puts(ps);
           поиск первого символа первой строки во второй */
        c= text1[0];
        ps=strchr(text2, c); /* ps — указатель на символ c в строке text2,
                           text2 —указатель на нулевой символ строки text2,
                           (ps-text2+1) — номер позиции символа c в строке text2 */
        if (ps)
            printf("Символ %c в строке %s находится на %d месте
                  \n", c, text2, ps-text2+1);
        else
            printf("Символ %c не найден в строке %s\n", c, text2);
        strcpy(text1, text2);
        puts(text1);
    }
}
```

Пример 2. Для вывода русских символов в консольном приложении созданном в среде Microsoft Visual Studio 2010 можно использовать функцию из библиотеки `ctype`:

```
char * setlocale ( int category, const char * locale );
```

Эта функция позволяет получить или установить параметры, которые зависят от настроек локализации операционной системы. Если указатель `locale` является нулем, функция `setlocale()` возвращает указатель на строку текущей локализации. В противном случае функция `setlocale()` попытается задать строку `locale` для локальных параметров в соответствии с параметром `category`.

В качестве параметра `category` должен быть использован один из следующих макросов (определенных в заголовочном файле `<ctype>`).

Таблица 9.1. Возможные значения параметра `category`

Значение параметра	Описание
LC_ALL	Относится ко всем категориям локализации
LC_COLLATE	Оказывает влияние на выполнение функции <code>strcoll()</code>
LC_CTYPE	Изменяет характер работы символьных функций
LC_MONETARY	Определяет денежный формат
LC_NUMERIC	Изменяет символ десятичной точки для функций форматированного ввода-вывода
LC_TIME	Определяет поведение функции <code>strftime()</code>

Более подробную информацию о строках локализации можно посмотреть в документации на используемый компилятор.

```
#include <stdio.h>
#include <ctype>          //подключаем библиотеку для использования
int main()                //функции setlocale
{
    setlocale(LC_ALL, "Russian");
                        //устанавливаем локализацию для России
    printf("Выводим сообщение на русском языке!\n");
                        //теперь можем выводить сообщения кириллицей
    return 0;
}
```

Пример 3. Ввести строку, состоящую из букв, цифр и пробелов. Определить количество слов в строке. Слова отделены друг от друга любым количеством пробелов. В начале и конце строки также может стоять любое количество пробелов:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[256], *p_str;
    int lengthWord, countWord = 0;
    puts("Enter string:");
    gets(str);
    p_str = str;
        //устанавливаем указатель p_str на начало введенной строки
    do
    {
        if ((*p_str == ' '))
            //если символ, на который указывает ps, равен пробелу
        do
        {
            p_str++;
            /*увеличить на 1 указатель на символ строки (переходим к следующему символу строки)*/
        }
        while (*p_str == ' ');
        //пока символ, на который указывает ps, равен пробелу
        if (*p_str != '\0')
            //если текущий символ не равен нуль-символу (строка не кончилась)
            {countWord++;
            //увеличить на 1 количество слов
            lengthWord = strcspn(p_str, " ");
            /*длина слова равна длине отрезка в остатке строки, не содержащего пробелы, или длине остатка строки, если пробелов в остатке строки нет*/
            if (lengthWord == strlen(p_str))
                //если длина слова равна длине остатка строки
                break;
            //выходим из цикла
            p_str += lengthWord;
            //устанавливаем указатель p_str на первый символ за выделенным словом
        }
        else break;
        //строка закончилась, выходим из цикла
    }
    while(1);
    printf("countWord = %d\n", countWord);
    return 0;
}
```


Пояснение. В программе описывается строковая переменная `str`, указатель на строку `p_str`, который будет использоваться как рабочая переменная, целые переменные `lengthWord` — длина выделяемого слова и `countWord` — количество слов.

После ввода строки `str` оператором `p_str = str`; устанавливаем указатель `p_str` на нулевой символ строки `str`. Затем во внешнем цикле `do-while` проверяем, является ли текущий символ (символ, на который указывает `p_str`) пробелом. Если да, то во внутреннем цикле `do-while` устанавливаем указатель `p_str` на следующий символ строки (остатка строки) `str`, проверяем, является ли новый символ пробелом, если да, то переходим к следующему символу. Выход из внутреннего цикла осуществляется, когда текущий символ станет не равен пробелу. То есть во внутреннем цикле пропускается несколько стоящих рядом пробелов.

Во внешнем цикле в условном операторе `if (p_str != '\0')` проверяем, что текущий символ не является нуль-символом (строка не закончена). Если условие выполняется (строка не закончена), то увеличиваем на 1 количество слов, затем с помощью оператора `lengthWord = strcspn(p_str, " ");` определяем длину слова, которая равна длине отрезка остатка строки, не содержащего пробелов, или длине остатка строки, если в нем нет пробелов.

Если длина выделенного слова равна длине остатка строки, выходим из цикла (все слова выделены). В противном случае оператором `p_str += lengthWord`; устанавливаем указатель `p_str` на первый символ за выделенным словом.

Тесты

1. Каким символом заканчивается строка?

- а) `'\n'`;
- б) `'\s'`;
- в) `'\0'`.

2. Что делает функция `gets()`?

- а) считывает строку с клавиатуры до нажатия клавиши Enter;
- б) считывает строку с клавиатуры до первого пробела;
- в) выводит строку на экран.

3. Что будет выведено на экран после выполнения следующей строчки?

```
puts("Hello, World!"); ...
```

- а) Hello,;
- б) Hello;
- в) Hello, World!

4. Укажите правильный вариант инициализации строки:

- а) `char str[] = "Программист";`
- б) `char str[12] = "Программист";`
- в) оба варианта правильные.

5. Какая строка форматной спецификации используется для вывода строки на экран?

- а) `%c;`
- б) `%s;`
- в) `%ss.`

6. Что будет выведено на экран в результате выполнения следующего кода?

```
char text[] = "нофелет";  
puts(&text[4]);
```

- а) елет;
- б) лет;
- в) ет.

Задания

1. С клавиатуры ввести последовательность символов. Написать функцию, вычисляющую длину этой последовательности.

2. С клавиатуры ввести предложение из нескольких слов, разделенных любым количеством пробелов. Написать функцию, вычисляющую количество слов в предложении.

3. С клавиатуры ввести последовательность символов, не содержащую пробелов. Вывести на экран эту последовательность в обратном порядке («информатика» — «акитамрофни»).

4. С клавиатуры ввести предложение, состоящее из произвольного числа слов. Написать функцию, которая подсчитывает количество повторяющихся слов в предложении.

5. С клавиатуры ввести строку, состоящую из произвольного числа слов. Написать функцию, которая выводит на экран строку, в которой перевернуто каждое слово («мир труд май» — «рим дурт йам»).

Контрольные вопросы

1. Что представляет собой строка в языке C++?
2. Каким символом должна заканчиваться строка?
3. С помощью каких функций можно считать последовательность символов с клавиатуры? В чем особенности применения этих функций?
4. С помощью каких функций можно вывести последовательность символов на экран? В чем особенности применения этих функций?
5. Как получить доступ к произвольному символу строки? Приведите примеры.
6. Какой заголовочный файл необходимо подключить для использования стандартных функций работы со строками?
7. С помощью каких функций можно вычислить длину строки, объединить, сравнить строки? Приведите примеры.
8. Какие действия необходимо выполнить, чтобы в консольном приложении, созданном в среде Microsoft Visual Studio 2010, выводить на экран русские символы?

Глава 10

ДВУМЕРНЫЕ МАССИВЫ (МАТРИЦЫ)

10.1. Понятие матрицы (двумерного массива)

Определение матрицы. Двумерный массив или матрица — это прямоугольная таблица однотипных элементов, снабженных двумя индексами. Матрица имеет строки (горизонтальные ряды значений) и столбцы (вертикальные ряды значений). Индексация (нумерация) строк и столбцов начинается с нуля. В оперативной памяти матрица хранится по строкам, сначала элементы 0-й строки, затем 1-й строки, 2-й и т. д. Например, матрица A из девяти целочисленных элементов имеет вид, приведенный на рис. 10.1.

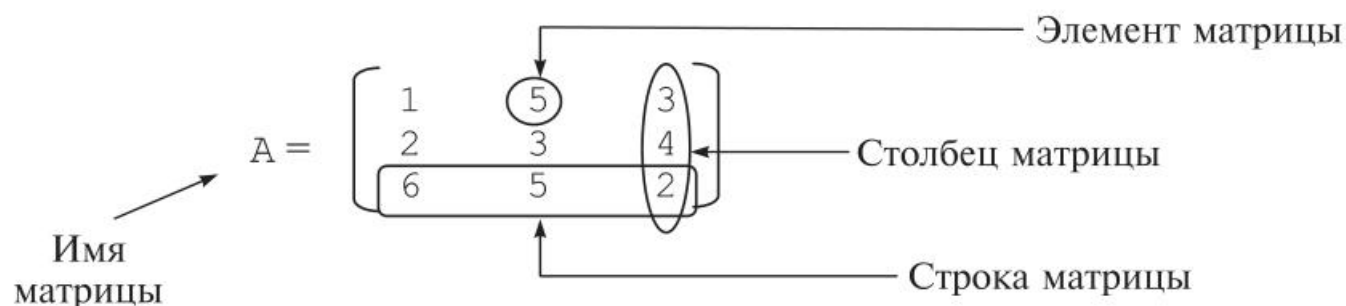


Рис. 10.1. Матрица A

В оперативной памяти эта матрица будет храниться следующим образом (рис. 10.2).

1	5	3	2	3	4	6	5	2
A[1,1]	A[1,2]	A[1,3]	A[2,1]	A[2,2]	A[2,3]	A[3,1]	A[3,2]	A[3,3]

Рис. 10.2. Матрица A в оперативной памяти

Если в этой матрице поменять местами любые два элемента, то это будет та же самая матрица, расположенная в том же месте опера-

тивной памяти, но с измененными значениями соответствующих элементов.

Атрибуты матрицы. Атрибутами матрицы являются:

- имя матрицы;
- количество строк N ;
- количество столбцов M ;
- тип элементов матрицы.

Объявление матрицы. Матрицу в программе можно объявить следующим образом:

```
<тип_элементов> <имя_матрицы> [<число_строк>]  
[<число_столбцов>];
```

где <число_строк>, <число_столбцов> — соответственно количество строк и столбцов в матрице;

<тип_элементов> — любой тип данных, применяемый в C++.

Например, целочисленная матрица B из 5 строк, 7 столбцов может быть объявлена:

```
int B[5][7];
```

Доступ к элементам матрицы. Доступ к элементам матрицы осуществляется через индексы. Каждый элемент матрицы имеет два индекса: индекс (номер) строки i , и индекс (номер) столбца j , на пересечении которых он находится:

```
<элемент_матрицы> = <имя_матрицы> [индекс_строки]  
[ индекс_столбца].
```

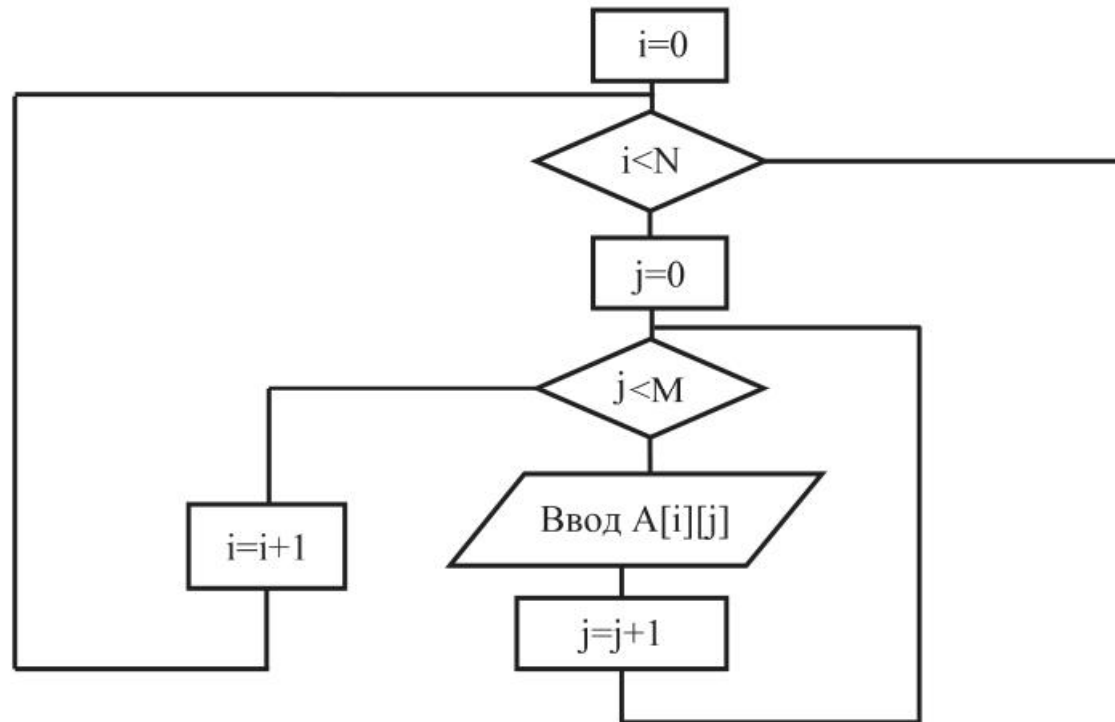
Например, элемент, стоящий на пересечении 2-й строки и 1-го столбца матрицы A , обозначается $A[2][1]$.

10.2. Формирование матриц и вывод их на экран

Элементы матрицы могут быть сформированы с помощью генератора случайных чисел или введены с клавиатуры. Формирование, ввод и вывод матриц производятся поэлементно с использованием двойного цикла.

Ввод матрицы с клавиатуры. Ввод матрицы с клавиатуры осуществляется построчно: сначала вводятся значения элементов 0-й строки, затем 1-й и т. д.

1. Схема алгоритма



Пояснение. Для ввода элементов матрицы по строкам организованы два цикла: внешний цикл по строкам, внутренний по столбцам.

2. Программа (ввод матрицы по строкам)

```

#include <stdio.h>
const int N=3, M=4;           /*объявление констант N — числа строк,
                                М — числа столбцов матрицы*/

int main() {
    double A[N][M];           //описание вещественной матрицы A
    int i, j;
    for (i=0; i<N; i++){      //в цикле по строкам
        printf("Введите через пробел %d элементов %d строки\n", M, i); //вывод приглашения к вводу строки матрицы
        for (j=0; j<M; j++)    //цикл по столбцам
            scanf("%lf", &A[i][j]); //ввод элемента матрицы
    }
    ...
    return 0;
}

```

2. Программа (ввод матрицы по элементам)

```

#include <stdio.h>
const int N=3, M=4;
//объявление констант N — числа строк, М — числа столбцов

```



```

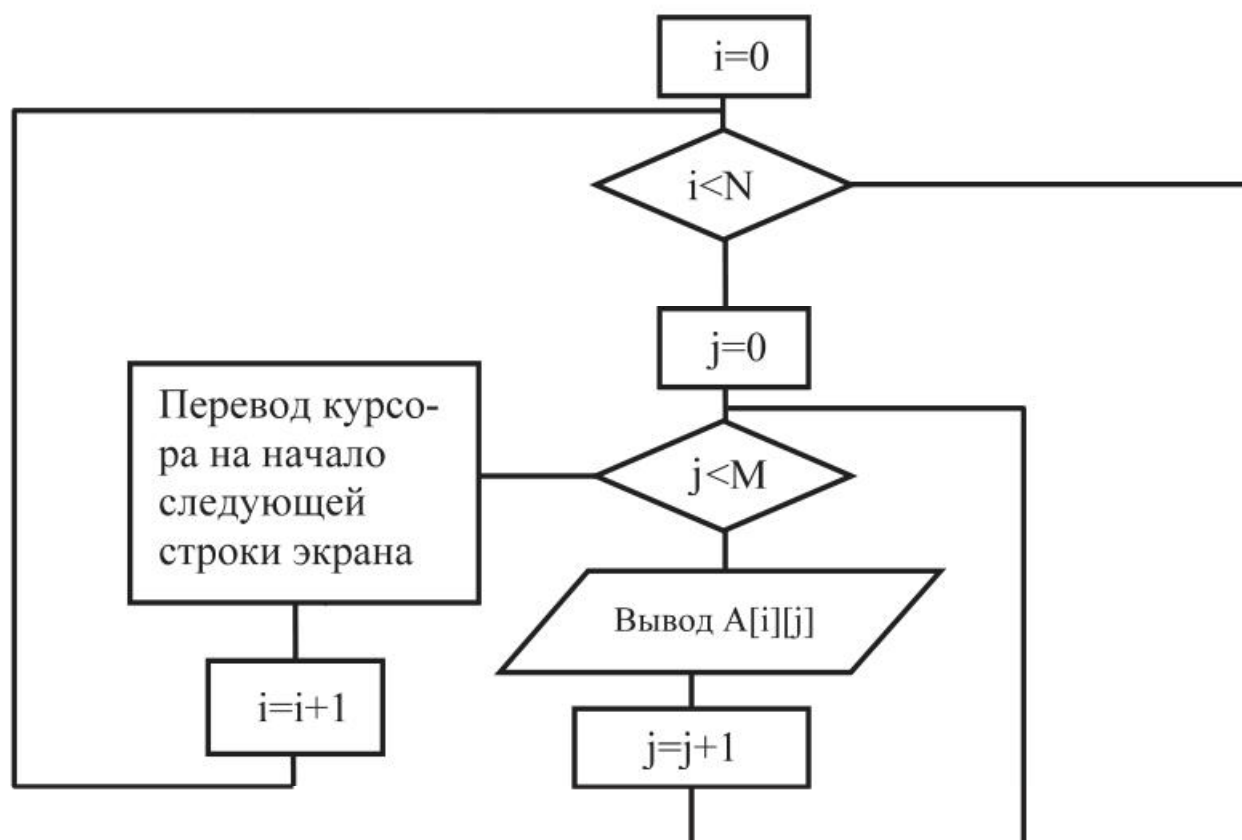
int main()
{double A[N][M];           //описание вещественной матрицы A
int i, j;
for (i=0; i<N; i++)        //цикл по строкам
for (j=0; j<M; j++){       //цикл по столбцам
printf("Введите A[ %d][ %d]  ", i, j);
                           //вывод приглашения к вводу элемента матрицы
scanf("%lf", &A[i][j]);    //ввод элемента матрицы
}
...
return 0;
}

```

Пояснение. Используются 2 оператора цикла `for` с параметром: внешний цикл (оператор `for`) с параметром `i` (начальное значение `i` равно 0, конечное значение `i` равно количеству строк в матрице — 1, шаг равен 1), внутренний цикл (оператор `for`) с параметром `j` (начальное значение `j` равно 0, конечное значение `j` равно количеству столбцов в матрице — 1, шаг равен 1). Переменные `i`, `j` являются индексами элементов матрицы (так как индекс элемента матрицы не может быть дробным, то переменные `i` и `j` должны иметь тип `int`).

Вывод матрицы на экран. Вывод матрицы на экран производится построчно в виде прямоугольной таблицы.

1. Схема алгоритма



Пояснение. Организация вывода матрицы по строкам аналогична организации ввода матрицы по строкам.

2. Программа

```
#include <stdio.h>
const int N=3, M=4;
//объявление констант N — числа строк, M — числа столбцов

int main() {
double A[N][M];           //описание вещественной матрицы A
int i, j;
...
for (i=0; i<N; i++){      //цикл по строкам
for (j=0; j<M; j++){      //цикл по столбцам
printf("%5.3lf ", A[i][j]);
//форматированный вывод элемента матрицы и пробела
printf("\n");             //перевод курсора на начало следующей строки экрана
}
...
return 0;
}
```

Формирование матриц с помощью датчика случайных чисел. Элементы матриц так же, как и элементы одномерных массивов, можно не только вводить с клавиатуры, но формировать с помощью генератора случайных чисел.

1. Схема алгоритма



2. Программа

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N=5, M=8;
//объявление констант N — числа строк,
//M — числа столбцов

int main() {
    int A[N][M]; //описание целочисленной матрицы A
    int i, j;
    srand(time(NULL)); //запуск датчика случайных чисел
    for (i=0; i<N; i++) //цикл по строкам
        for (j=0; j<M; j++) //цикл по столбцам
            A[i][j]=rand()%101 -50; //формирование элемента матрицы
    printf("Матрица сформирована\n");
    ...
}
```

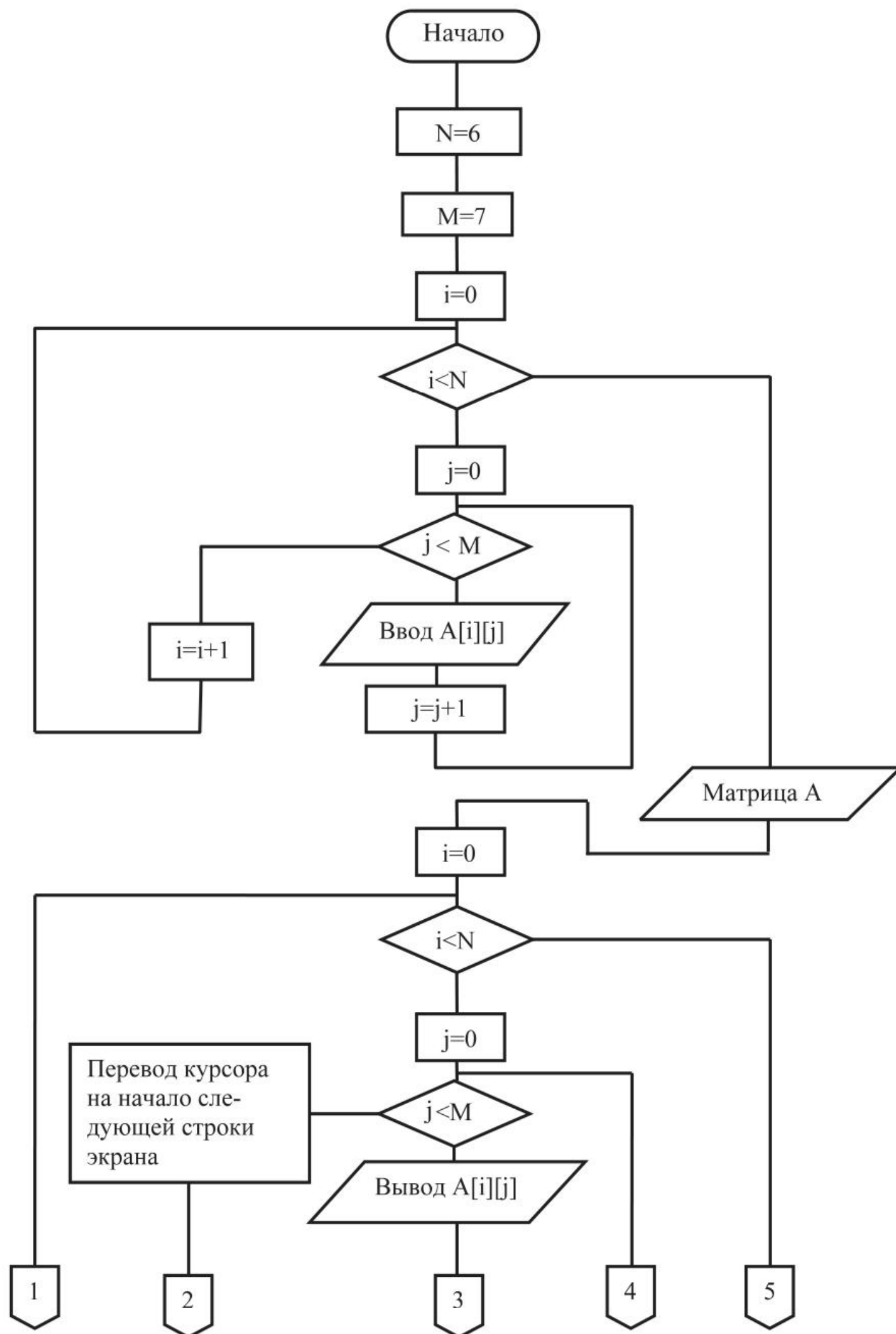
10.3. Работа с матрицами

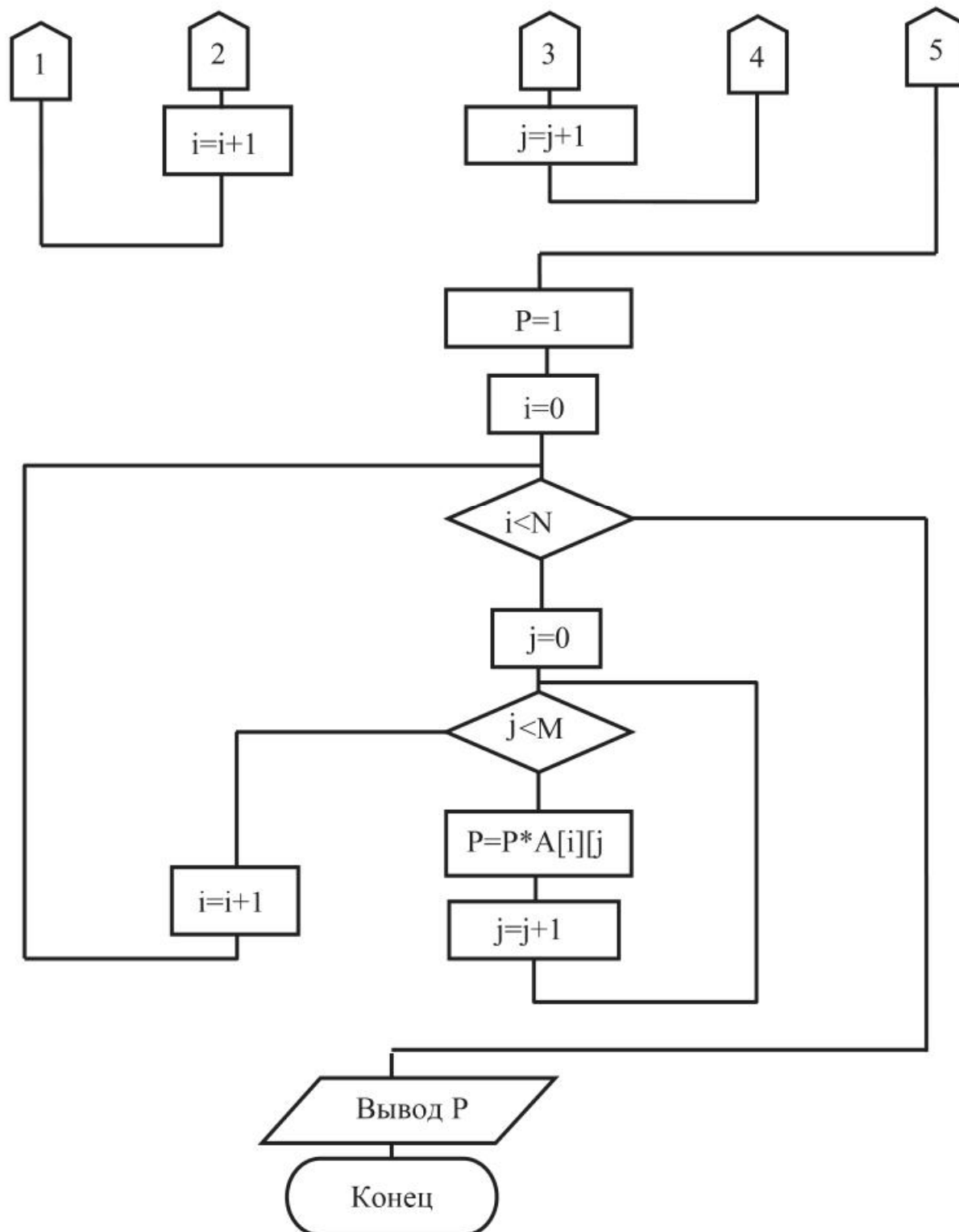
Матрицы можно поэлементно складывать, умножать, вычитать, делить и применять к элементам матрицы различные сочетания арифметических операций для получения новой матрицы или массива из элементов матрицы по различным критериям, находить минимальный (максимальный) элемент, сумму, произведение всех элементов, количество каких-либо элементов и т. д. Все операции с матрицами производятся поэлементно с использованием двойного цикла.

Пример программы определения произведения всех элементов матрицы. Ввести с клавиатуры матрицу $A[6][7]$, вывести ее на экран, найти произведение всех элементов матрицы.

1. *Схема алгоритма*
(см. с. 204—205).

Пояснение. Запись « $A[6][7]$ » означает, что в матрице 6 строк и 7 столбцов. С клавиатуры поэлементно вводятся значения элементов матрицы. Затем матрица выводится на экран. В конце программы на экран выводится полученное произведение.





2. Программа

```

#include <stdio.h>
const int N=6, M=7;
int main() {
double A[N][M];
int i, j;
for (i=0; i<N; i++)
for (j=0; j<M; j++)
{printf("Введите A[ %d][ %d]  ", i, j);
scanf("%lf", &A[i][j]);
}
}

```

//цикл по строкам
//цикл по столбцам

```

printf("Матрица A\n");
for (i=0; i<N; i++){                               //цикл по строкам
    for (j=0; j<M; j++){                             //цикл по столбцам
        printf("%5.3lf ", A[i][j]);
        //форматированный вывод элемента матрицы и пробела
    }
    printf("\n"); //перевод курсора на начало следующей строки экрана
}
P=1; //задание начального значения произведения (P=1)
for (i=0; i<N; i++){                               //цикл по строкам
    for (j=0; j<M; j++){                             //цикл по столбцам
        P=P*A[i][j]; //умножение старого значения P на элемент матрицы
    }
    printf("P=%lf\n", P);
}
getchar();
return 0;
}

```

Пояснение. В программе используются четыре переменные: вещественная матрица *A* из 6 строк и 7 столбцов (размеры матрицы объявлены как константы *N* и *M* для того, чтобы проще было модифицировать программу при изменении размеров матрицы); *i*, *j* — параметры циклов (индексы элементов матрицы) объявлены как переменные типа **int**; *P* — произведение всех элементов (объявлена как переменная типа **double**, так как тип **double** имеет более широкий диапазон возможных значений).

Для решения задачи используются трижды два вложенных оператора **for**: ввод с клавиатуры значения элементов матрицы, вывод матрицы на экран, вычисление произведения всех элементов матрицы. В матрице *A* 6 строк, 7 столбцов, поэтому в каждом двойном цикле внешний цикл (по строкам) выполняется *N*=6 раз, а внутренний цикл (по столбцам) выполняется *M*=7 раз.

Примечание. В рассмотренной задаче можно объединить ввод матрицы с клавиатуры и вычисление произведения всех элементов матрицы в один цикл, используя составной оператор:

```

...
P=1;
for (i=0; i< N; i++)
    for (j=0; j < K; j++)
        {printf("Введите A[%d][%d]=", i, j);
         scanf("%lf",&A[i][j]);
         P=P*A[i][j];
        }
printf("Матрица A\n");
...

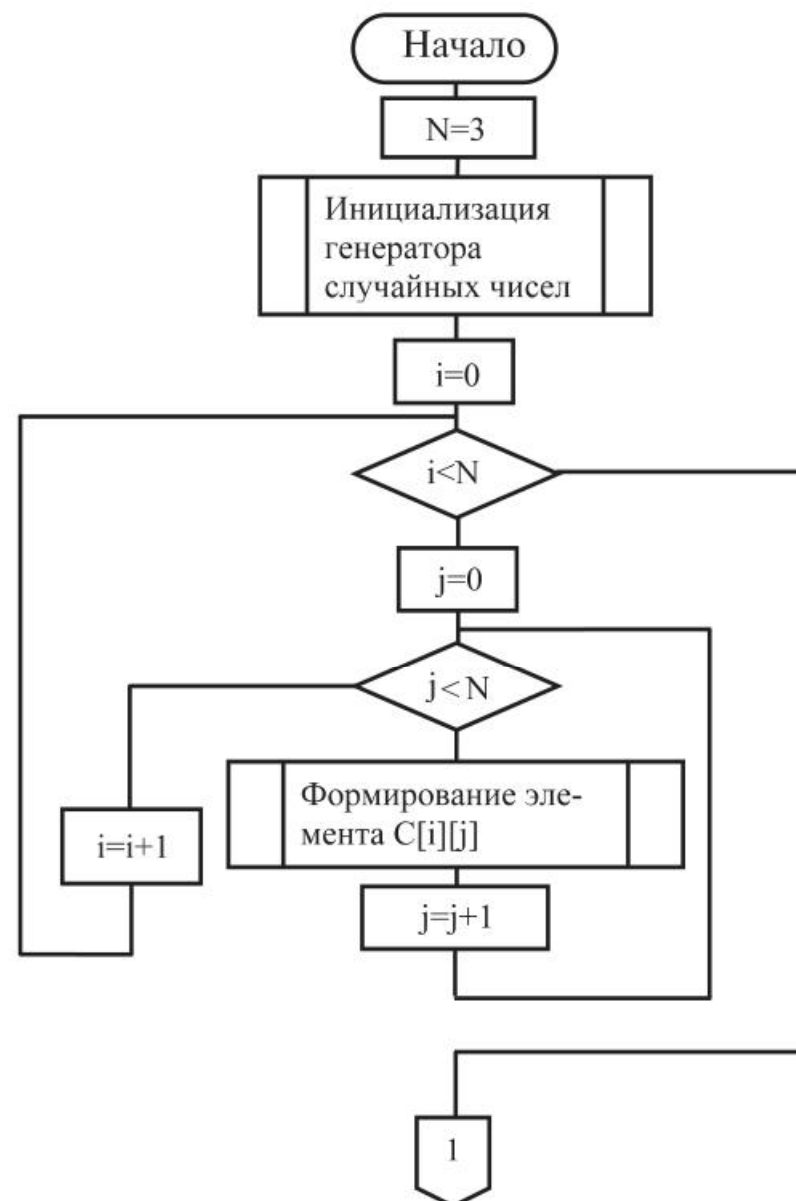
```

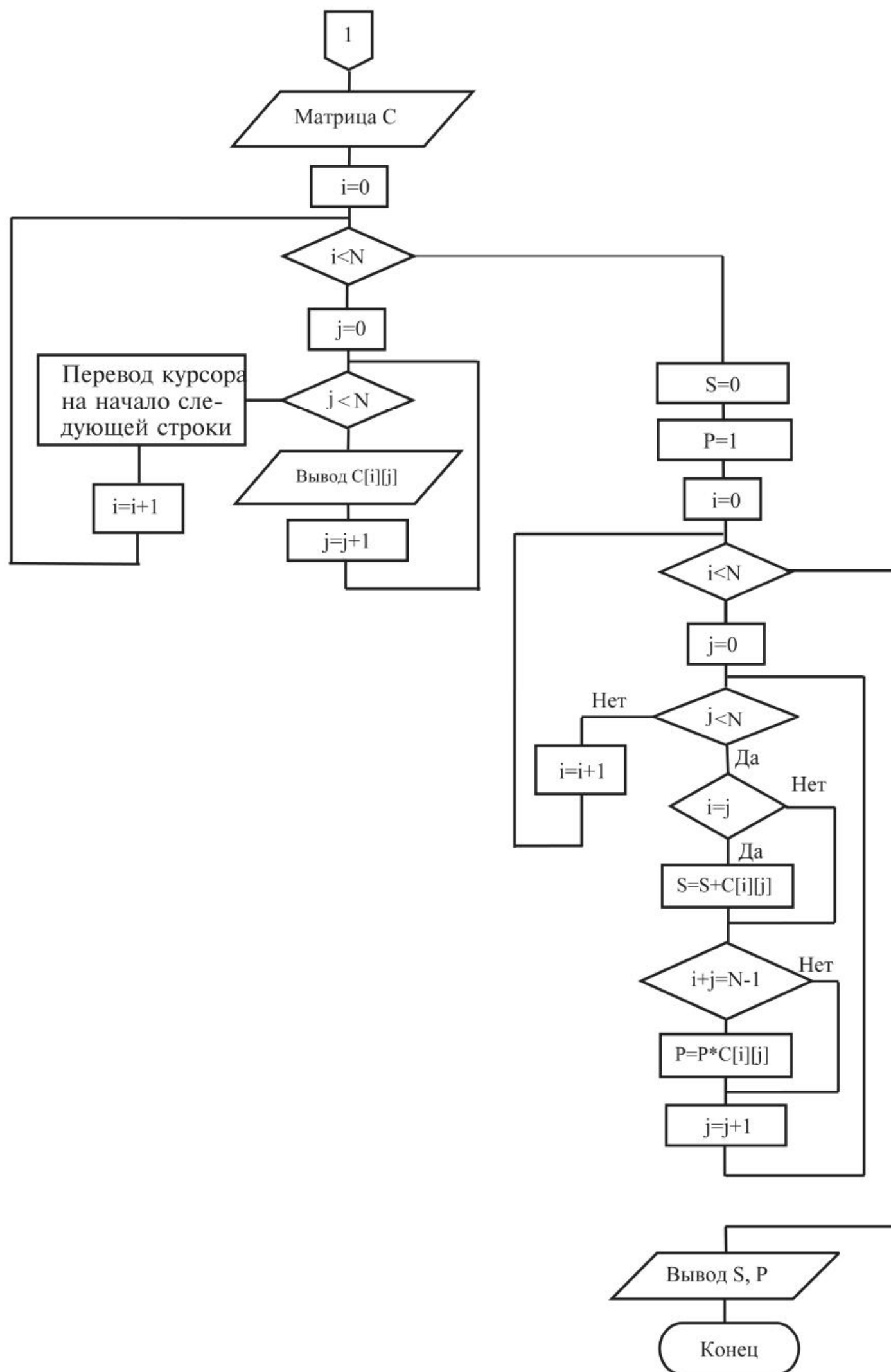

Пример программы определения суммы элементов главной и произведения элементов побочной диагоналей матрицы. Сформировать матрицу $C[3][3]$ случайным образом. Вывести ее на экран. Найти сумму элементов главной диагонали матрицы и произведение элементов побочной диагонали матрицы (рис. 10.3).

Матрица C — квадратная, размерности $N \times N$ ($N=3$ в нашем примере). Элементы $C[0,0]=5$, $C[1,1]=11$, $C[2,2]=8$ расположены на *главной диагонали* матрицы C , для этих элементов *номер строки равен номеру столбца*, т. е. $i=j$. Элементы $C[0,2]=6$, $C[1,1]=11$, $C[2,0]=3$ расположены на *побочной диагонали* матрицы C , для этих элементов (*номер строки + номер столбца*) $= N-1$, т. е. $i+j=2$ в нашем случае.

Рис. 10.3. Матрица и ее диагонали

1. Схема алгоритма





Пояснение. Для решения задачи необходимо сформировать с помощью генератора случайных чисел квадратную матрицу 3-го порядка (матрица C из трех строк, трех столбцов) и вывести ее на экран. Затем задаются начальные значения сумме ($S=0$) и произведению ($P=1$) и перебираются все элементы матрицы (по строкам). Если для элемента $C_{i,j}$ выполняется условие главной диагонали ($i=j$), то он прибавляется к уже имеющейся сумме таких элементов. Если для элемента $C_{i,j}$ выполняется условие побочной диагонали ($i+j=N-1$), то он умножается на уже имеющееся произведение таких элементов.

2. Программа. 1-й способ решения

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N=3;           //объявление константы N — числа строк
                           //и числа столбцов

int main() {
    int C[N][N];
    int i, j;
    double S,P;
    srand(time(NULL));
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            C[i][j]=rand()%101-50;
    printf("Матрица C\n");
    for (i=0; i<N; i++){
        for (j=0; j<N; j++)
            printf("%6d ",C[i][j]);
        printf("\n");
    }
    S=0;                   //задание начального значения суммы (S=0)
    P=1;                   //задание начального значения произведения (P=1)
    for (i=0; i<N; i++)    //цикл по строкам
        for (j=0; j<N; j++) //цикл по столбцам
        { if (i == j)      //если элемент на главной диагонали
            S=S+C[i][j];   //прибавляем его к сумме
            if ((i+j) == N-1) //если элемент на побочной диагонали
                P=P*C[i][j]; //умножаем его на произведение
        }
    printf("S=%lf P=%lf\n",S,P);
    getchar();
    return 0;
}
```


Пояснение. Матрица C — квадратная (число строк равно числу столбцов), поэтому объявлена только одна константа (порядок матрицы). В программе используются пять переменных: C — квадратная целочисленная матрица третьего порядка; i, j — параметры цикла **for** (индексы элементов матрицы); S — сумма элементов главной диагонали типа **double**; P — произведение элементов побочной диагонали типа **double**. В первом двойном цикле происходит формирование элементов матрицы с помощью генератора случайных чисел. Во втором двойном цикле — вывод матрицы на экран. В третьем двойном цикле — вычисление суммы и произведения по диагоналям матрицы.

3. Программа. 2-й способ решения

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N=3;           //объявление константы N — числа строк
                           //и числа столбцов

int main() {
    int C[N][N];
    int i, j;
    double S, P;
    srand(time(NULL));
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            C[i][j]=rand()%101 -50;
    printf("Матрица C\n");
    for (i=0; i<N; i++){
        for (j=0; j<N; j++)
            printf("%6d ", C[i][j]);
        printf("\n");
    }
    S=0;                  //задание начального значения суммы (S=0)
    P=1;                  //задание начального значения произведения (P=1)
    for (i=0; i<N; i++)   //цикл по строкам
    { S=S+C[i][i];        //прибавляем к сумме элемент на главной диагонали
      P=P*C[i][N-1-i];    //умножаем произвед. на элем. на побочн. диагонали
    }
    printf("S=%lf P=%lf\n", S, P);
    getchar();
    return 0;
}
```

Пояснение. Матрица C — квадратная (число строк равно числу столбцов), поэтому объявлена только одна константа (порядок матрицы). В программе используются пять переменных: C — квадратная

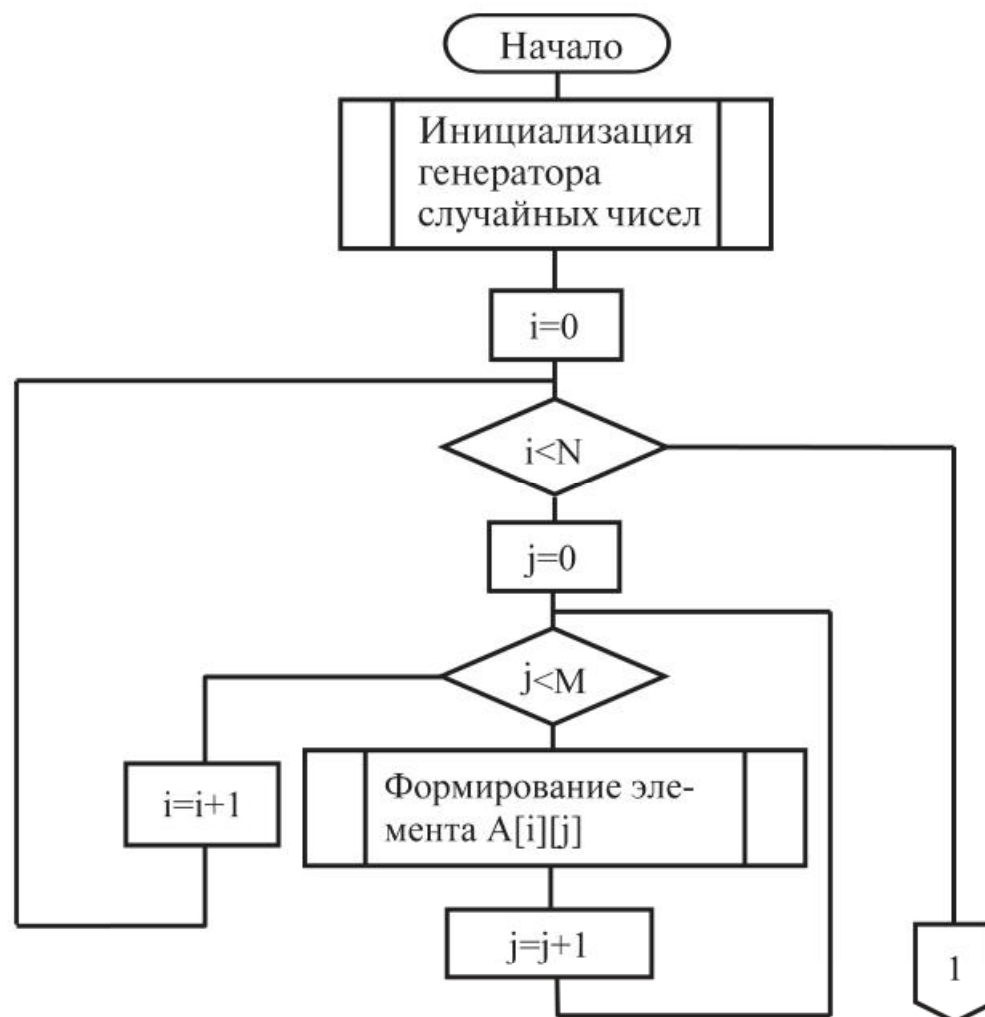
целочисленная матрица третьего порядка; i, j — параметры цикла **for** (индексы элементов матрицы); S — сумма элементов главной диагонали типа **double**; P — произведение элементов побочной диагонали типа **double**. В первом двойном цикле происходит формирование элементов матрицы с помощью генератора случайных чисел. Во втором двойном цикле — вывод матрицы на экран. В третьем одинарном цикле — вычисление суммы и произведения по диагоналям матрицы.

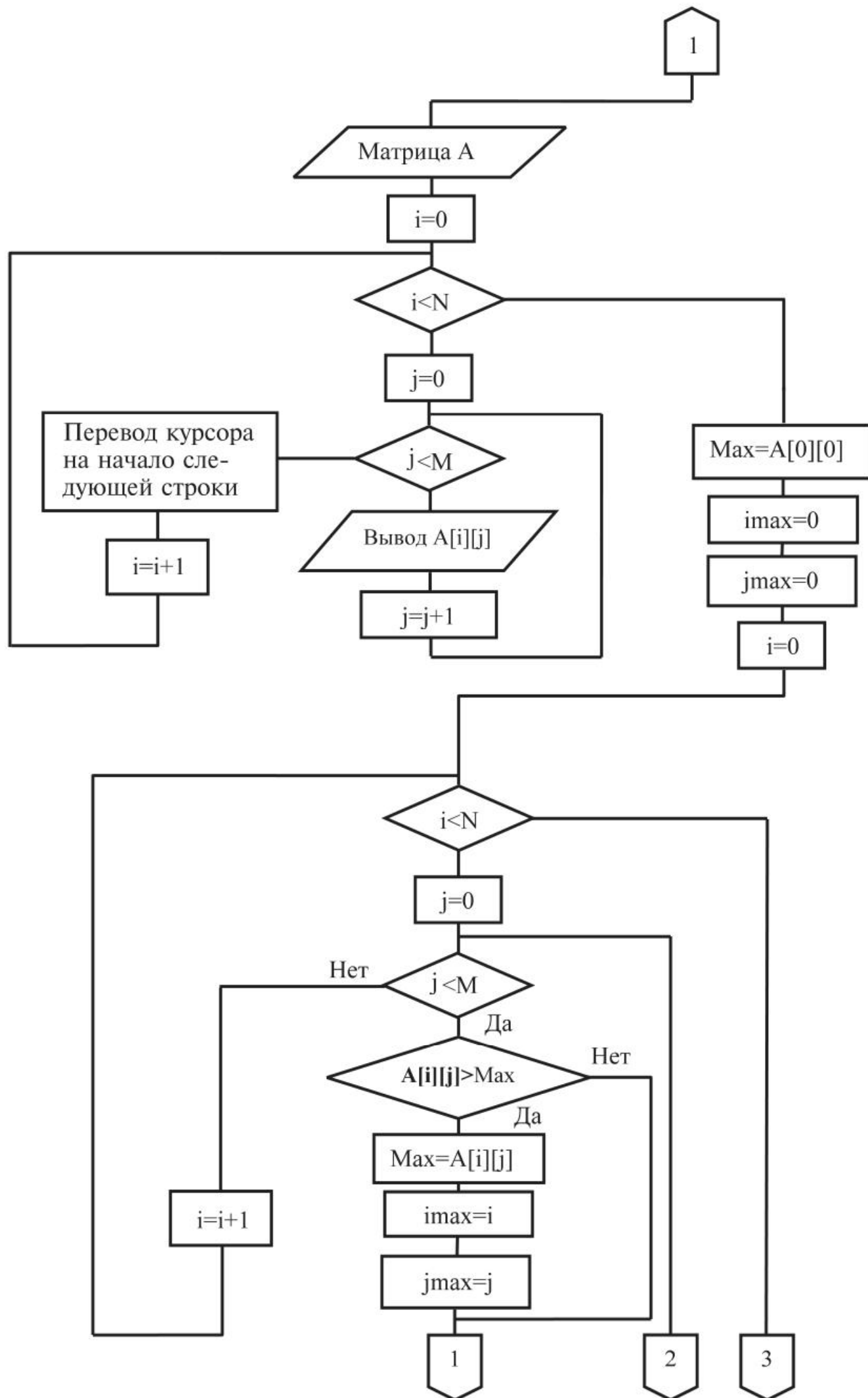
10.4. Поиск максимального (минимального) элемента матрицы и определение его координат (индексов)

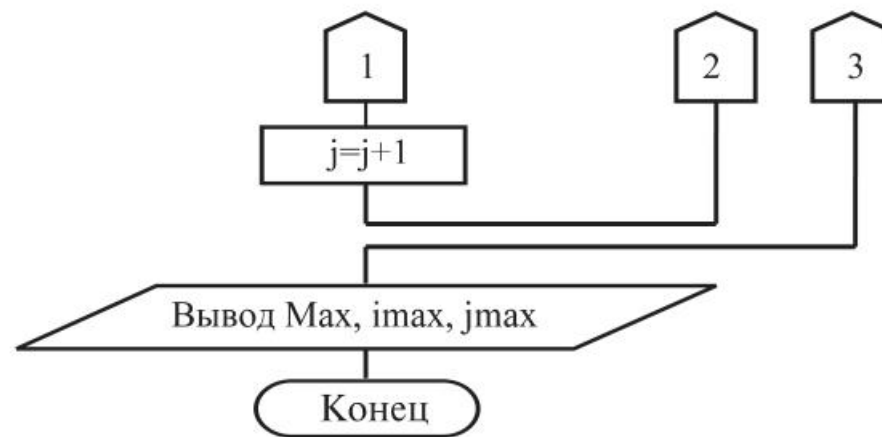
Алгоритм поиска максимального элемента в матрице и определения его координат полностью аналогичен алгоритму поиска минимального элемента в матрице и определения его координат.

Пример программы поиска максимального элемента в матрице и определения его координат. Сформировать матрицу A произвольной размерности. Вывести матрицу. Найти ее максимальный элемент и его координаты.

1. Схема алгоритма







Пояснение. Введем рабочие переменные: *Max* — для хранения текущего значения максимального элемента матрицы; *imax* — номер строки текущего максимального элемента; *jmax* — номер столбца текущего максимального элемента. Предположим, что 0-й элемент 0-й строки матрицы — максимальный. Запомним его значение в переменной *Max*; в переменной *imax* запомним значение «0» — индекс (номер строки) предполагаемого максимального элемента; в переменной *jmax* запомним значение «0» — индекс (номер столбца) предполагаемого максимального элемента. Далее первый элемент 0-й строки матрицы сравниваем со значением *Max*. Если первый элемент больше *Max*, то запишем его значение в переменную *Max*, номер строки 1-го элемента («0») запомним в переменной *imax*, а номер столбца («1») — в переменной *jmax*. Переходим к следующему элементу 0-й строки матрицы. Если первый элемент не больше *Max*, то сразу переходим к следующему элементу 0-й строки матрицы, не изменяя значений переменных *Max*, *imax*, *jmax*. Эту процедуру повторяем со всеми элементами 0-й строки, потом переходим к элементам 1-й строки и т. д. Таким образом, после прохода по всем строкам матрицы в переменной *Max* будет находиться значение максимального элемента матрицы, а в *imax* и *jmax* — соответственно номер строки и номер столбца максимального элемента матрицы.

2. Программа

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N=7, M=10;
int main() {
    int A[N][M];
    int i, j, Max, imax, jmax;
    srand((unsigned int)time(NULL));
    for (i=0; i<N; i++)
        for (j=0; j<M; j++)

```

```

A[i][j]=rand()%101 -50;
printf("Матрица A\n");
for (i=0; i<N; i++)
{
for (j=0; j<M; j++)
printf("%6d ",A[i][j]);
printf("\n");
}
Max=A[0][0];
    //присвоить перем. Max значение элем. матрицы с координатами 0,0
imax=0;      //присваивание 0 номеру строки предполагаемого максимума
jmax=0;      //присваивание 0 номеру столбца предполагаемого максимума
for (i=0; i<N; i++)                                //цикл по строкам
for (j=0; j<M; j++)                                //цикл по столбцам
if (A[i][j]>Max)
    //если текущий элемент матрицы больше предполагаемого максимума
    {Max=A[i][j];                                //меняем предполагаемый максимум
    imax=i;                                       //номер строки предполагаемого максимума
    jmax=j;                                       //номер столбца предполагаемого максимума
    }
printf("Max=%d imax=%d jmax=%d\n",Max,imax,jmax);
getchar();
return 0;
}

```

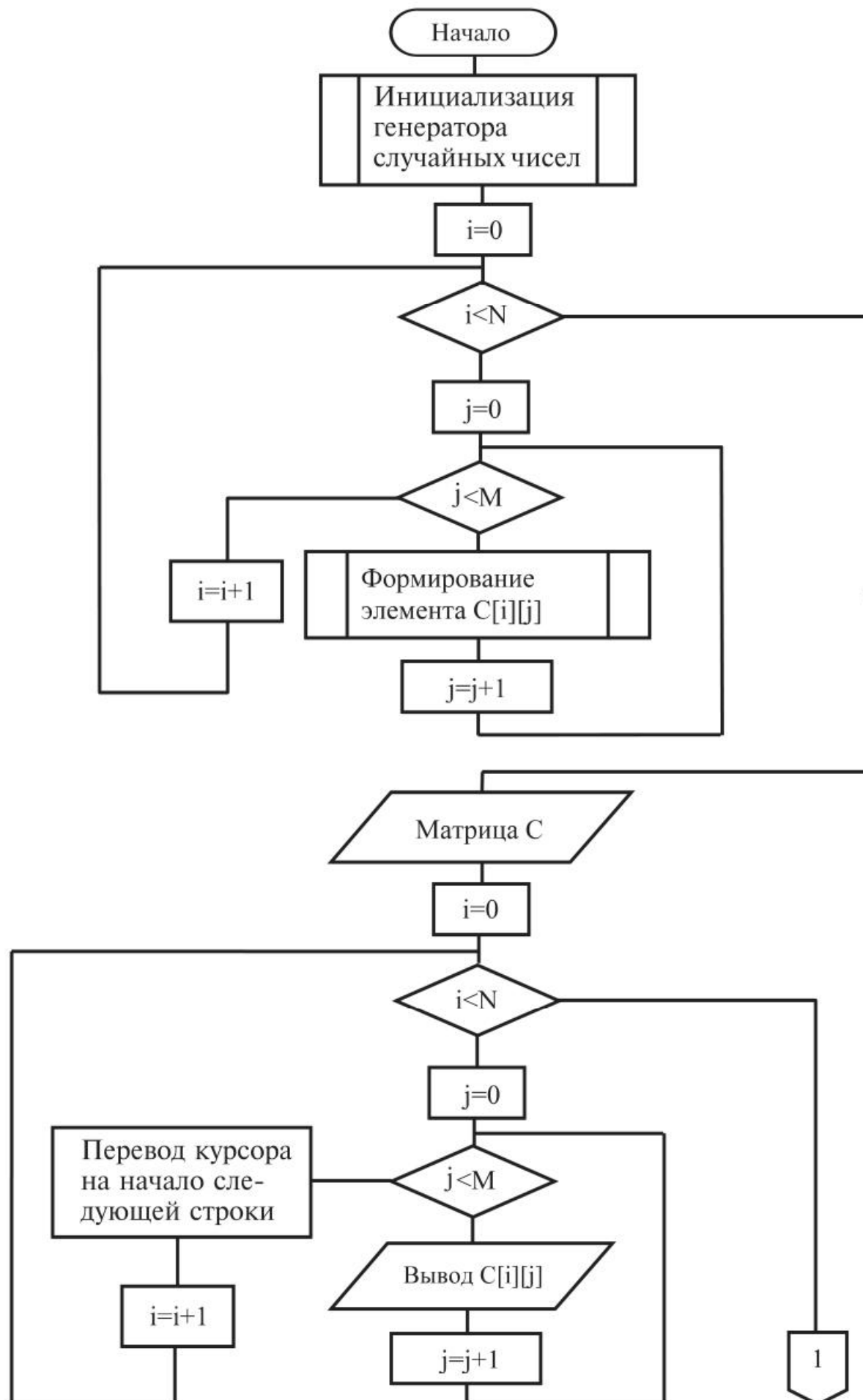
Пояснение. В программе используются шесть переменных: *A* — целочисленная матрица; *i*, *j* — параметры цикла **for**, *Max* — максимальный элемент матрицы; *imax* — номер строки максимального элемента; *jmax* — номер столбца максимального элемента. Переменные *i*, *j*, *imax*, *jmax* имеют тип **int**. Тип переменной *Max* определяется типом элементов матрицы (в данном случае **int**).

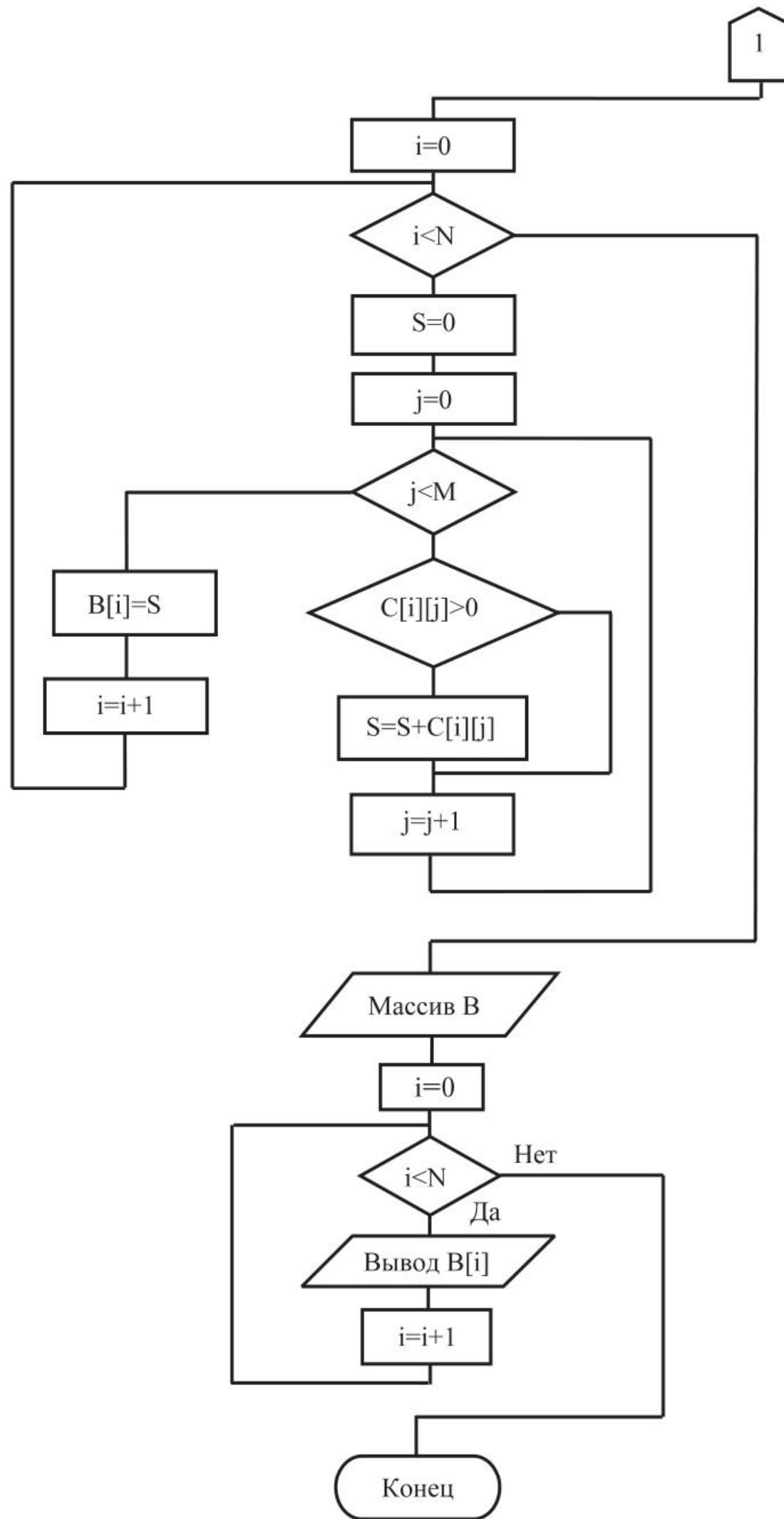
Примечание. Нахождение минимального элемента матрицы и его координат аналогично поиску максимального элемента, но значения переменных *Min* (текущий минимальный элемент), *imin*, *jmin* (индексы текущего минимального элемента) переопределяются, если некоторый элемент массива меньше текущего минимального.

10.5. Формирование одномерных массивов из элементов матриц

Пример программы формирования одномерных массивов из сумм положительных элементов каждой строки матрицы. Сформировать матрицу *C*{7,8}. Вывести ее. Сформировать одномерный массив *B* из

1. Схема алгоритма





Пояснение. Матрица C формируется с помощью генератора случайных чисел и выводится на экран в одном цикле. Затем формируется массив M . Для этого матрица C просматривается по строкам (внешний цикл по строкам — по i). Для каждой строки выполняются 3 действия:

- задается начальное значение суммы положительных элементов i -й строки ($S=0$);
- вычисляется сумма положительных элементов i -й строки (внутренний цикл по j);
- вычисленная сумма для i -й строки записывается в элемент массива B с индексом i (массив B формируется по строкам матрицы, поэтому индексы элементов массива совпадают с индексами строк матрицы: сумма положительных элементов 0-й строки записывается в B_0 , 1-й строки — в B_1 и т. д.)

2. Программа

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N=7, M=8;
int main() {
    int C[N][M];
    int i,j;
    double S, B[N];
    srand(time(NULL));
    for (i=0; i<N; i++)
        for (j=0; j<M; j++)
            C[i][j]=rand()%101 - 50;
    printf("Матрица C\n");
    for (i=0; i<N; i++){
        for (j=0; j<M; j++)
            printf("%4d ",C[i][j]);
        printf("\n");
    }
    for(i=0; i<N; i++){
        S=0; //в цикле по строкам //зад. нач. значения сумме положит. элементов i-й строки (S=0)
        for(j=0; j<M; j++) //в цикле по столбцам
            if (C[i][j]>0) //если элемент матрицы больше 0
                S=S+C[i][j]; //прибавляем элемент матрицы к старому значению S
        B[i]=S; //присвоение i-му элементу массива B значения S
    }
    printf("Массив B\n");
```

```
for (i=0; i<n; i++)
printf("%5.0lf ",B[i]);
printf("\n");
getchar();
return 0;
}
```

Пояснение. В программе используются пять переменных: *S* — целочисленная матрица из семи строк, восьми столбцов; *B* — вещественный массив из семи элементов (длина массива определяется числом строк в матрице, тип элементов определяется типом элементов матрицы); *i*, *j* — параметры цикла и индексы элементов матрицы (типа **int**), *S* — текущее значение суммы положительных элементов в каждой строке матрицы. В первом двойном цикле происходит формирование элементов матрицы *S*. Во втором цикле происходит вывод на экран элементов матрицы *S*. В третьем цикле происходит вычисление суммы положительных элементов в каждой (*i*-й) строке матрицы и запись полученного значения в элементы массива *B* (*B[i]*). В конце программы на экран выводится массив *B*.

Пример программы формирования одномерного массива из минимальных элементов каждого столбца матрицы. Сформировать матрицу $H\{12,5\}$. Вывести ее. Сформировать одномерный массив *F* из минимальных элементов каждого столбца матрицы. Вывести массив на экран.

2. Программа

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int N=12, M=5;
int main() {
int H[N][M];
int i,j;
int min, F[M];
srand(time(NULL));
for (i=0; i<N; i++)
for (j=0; j<M; j++)
H[i][j]=rand()%101 - 50;
printf("Матрица H\n");
for (i=0; i<N; i++){
for (j=0; j<M; j++)
printf("%4d ",H[i][j]);
printf("\n");
}
```



```

for (j=0; j<M; j++){                                     //в цикле по столбцам
min=H[0][j];      //присв. нач. знач. предполагаемому миним. в столбце
for (i=0; i<N; i++)                                     //в цикле по строкам
if (H[i][j]<min)
    //если элемент матрицы меньше предполагаемого минимума
min=H[i][j];      //меняем предполагаемый минимум
F[j]=min;         //присваивание j-му элементу массива F значения min
}
printf("Массив F\n");
for (j=0; j<M; j++)
printf("%d ",F[j]);
printf("\n");
getchar();
return 0;
}

```

Пояснение. В программе используются пять переменных: *H* — целочисленная матрица; *F* — одномерный массив (длина массива *F* и тип его элементов определяются соответственно числом столбцов и типом элементов матрицы *H*); *i*, *j* — параметры цикла и индексы элементов; *min* — текущее значение минимального элемента *j*-го столбца. Формирование элементов матрицы и вывод матрицы на экран происходит в первых двух циклах. При формировании массива *F* матрица рассматривается по столбцам (внешний цикл по *j*, внутренний цикл по *i*), для каждого столбца выполняются три действия:

- 1-й элемент *j*-го столбца принимается за минимальный (*min*=*H*[0, *j*]);
- происходит поиск минимального элемента в одномерном массиве (*j*-й столбец рассматривается как одномерный массив, изменяется только индекс *i*);
- найденный минимальный элемент *j*-го столбца записывается в элемент массива *F* с индексом *j* (*F*[*j*]=*min*).

10.6. Представление двумерного массива (только для И-32,33)

Представление двумерного массива (матрицы). Матрица (двумерный массив) в языке C++ рассматривается как одномерный массив, элементами которого являются одномерные массивы — строки матрицы. Так как имя одномерного массива (строки матрицы) является указателем на массив, то элементами матрицы как одномерного массива являются адреса строк матрицы (рис. 10.4).

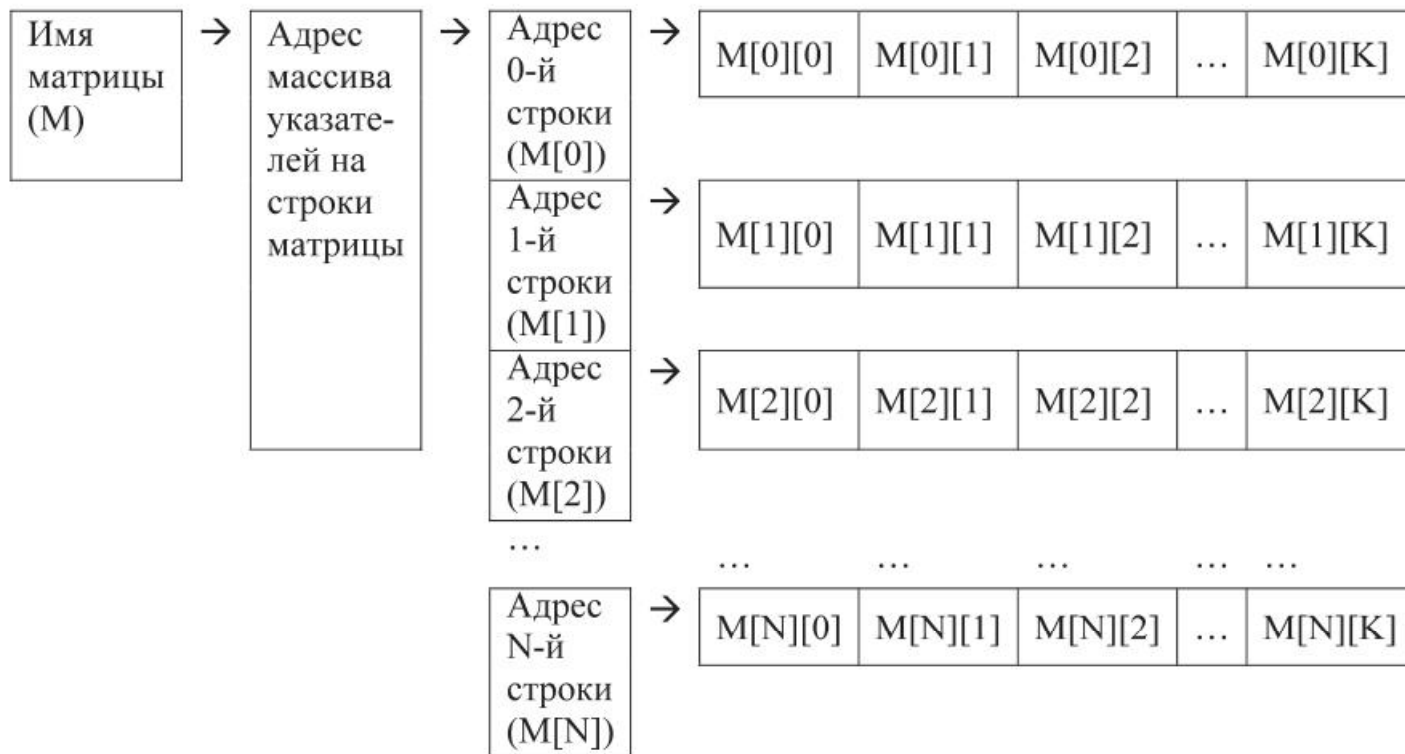


Рис. 10.4. Представление матрицы как одномерного массива

Адреса элементов матрицы. Из рис. 10.4 видно, что имя матрицы содержит адрес массива указателей (адресов) на строки матрицы. Элементами безымянного массива указателей на строки являются адреса соответствующих строк матрицы. По адресам (указателям) строк можно получить доступ к каждой строке матрицы (табл. 10.1). Так как в имени матрицы хранится только адрес массива указателей на строки, то для получения адреса каждой строки необходимо выполнять разыменование указателя — «имя матрицы + номер строки».

Зная адрес каждой строки матрицы, можно вычислить адрес каждого элемента этой строки (табл. 10.2).

Доступ к элементам матрицы. Используя операцию разыменования, можно по адресу элемента, стоящего на пересечении i -й строки и j -го столбца — $*(\text{Имя_матрицы} + i) + j$ — получить его значение — $*(*(\text{Имя_матрицы} + i) + j)$ (второй столбец табл. 10.3). Для матриц существует и другой, более удобный способ записи операции разыменования — $\text{Имя_матрицы}[i][j]$ (третий столбец табл. 10.3). Например, для матрицы M из N строк и K столбцов, представленной на рис. 10.4, значение элемента матрицы, стоящего на пересечении i -й строки и j -го столбца, можно получить двумя тождественными способами — $*(*(M+i)+j)$ и $M[i][j]$. Любую запись доступа к элементу матрицы $A[3][5]$ компилятор автоматически преобразует к виду $*(*(A+3) + 5)$, но так как первый вариант записи более наглядный, то на практике можно пользоваться им.

Таблица 10.1. Вычисление адресов строк в матрице

Адреса строк в матрице	Формулы для вычисления адресов	Формулы для вычисления адресов с использованием операции разыменования	Формулы для вычисления адресов с использованием доступа []
Адрес 0-й строки матрицы	Адрес массива указателей на строки + 0	$*(\text{Имя_матрицы} + 0)$	$\text{Имя_матрицы}[0]$
Адрес 1-й строки матрицы	Адрес массива указателей на строки + 1	$*(\text{Имя_матрицы} + 1)$	$\text{Имя_матрицы}[1]$
Адрес 2-й строки матрицы	Адрес массива указателей на строки + 2	$*(\text{Имя_матрицы} + 2)$	$\text{Имя_матрицы}[2]$
Адрес i-й строки матрицы	Адрес массива указателей на строки + i	$*(\text{Имя_матрицы} + i)$	$\text{Имя_матрицы}[i]$

Таблица 10.2. Вычисление адресов элементов в i-й строке матрицы

Адреса элементов в i-й строке матрицы	Формулы для вычисления адресов	Формулы для вычисления адресов	Формулы для вычисления адресов
Адрес 0-го элемента	Адрес i-й строки + 0	$*(\text{Имя_матрицы} + i) + 0$	$\text{Имя_матрицы}[i] + 0$
Адрес 1-го элемента	Адрес i-й строки + 1	$*(\text{Имя_матрицы} + i) + 1$	$\text{Имя_матрицы}[i] + 1$
Адрес 2-го элемента	Адрес i-й строки + 2	$*(\text{Имя_матрицы} + i) + 2$	$\text{Имя_матрицы}[i] + 2$
Адрес j-го элемента	Адрес i-й строки + j	$*(\text{Имя_матрицы} + i) + j$	$\text{Имя_матрицы}[i] + j$

Таблица 10.3. Определение значений элементов в i-й строке матрицы

Значения элементов матрицы	Формулы для определения значений	Формулы для определения значений
Значение 0-го элемента	$*(\text{Имя_матрицы} + i) + 0$	$\text{Имя_матрицы}[i][0]$
Значение 1-го элемента	$*(\text{Имя_массива} + i) + 1$	$\text{Имя_матрицы}[i][1]$
Значение 2-го элемента	$*(\text{Имя_массива} + i) + 2$	$\text{Имя_матрицы}[i][2]$
Значение j-го элемента	$*(\text{Имя_массива} + i) + j$	$\text{Имя_матрицы}[i][j]$

10.7. Представление статического двумерного массива (только для И-32,33)

Представление двумерного массива (матрицы), формулы вычисления адресов и значений элементов матрицы, изложенные в предыдущем параграфе, полностью справедливы и для статических и для динамических матриц.

В оперативной памяти статическая матрица хранится по строкам, сначала элементы 0-й строки, затем 1-й строки, 2-й и т. д. Например, описанная ранее матрица *A* из 10 целочисленных элементов (см. рис. 10.1) в оперативной памяти будет храниться следующим образом (рис. 10.5).

1	5	3	2	3	4	6	5	2
<i>A</i> [0,0]	<i>A</i> [0,1]	<i>A</i> [0,2]	<i>A</i> [1,0]	<i>A</i> [1,1]	<i>A</i> [1,2]	<i>A</i> [2,0]	<i>A</i> [2,1]	<i>A</i> [2,3]

Рис. 10.5. Матрица *A* в оперативной памяти

Так как статическая матрица хранится по строкам, то можно рассчитать адрес каждой строки в байтах (табл. 10.4). В приведенных формулах `sizeof` — операция определения количества байтов в значении типа, указанного в скобках, например, `sizeof(int)=4`, `sizeof(double)=8`. Отдельного массива указателей на строки статической матрицы не существует, в качестве значений указателей на строки используются значения, вычисленные по формулам таблицы 10.4. Полученные значения указателей, например, можно вывести на экран.

Таблица 10.4. Вычисление адресов строк в статической матрице в байтах

Адреса строк в матрице в байтах	Формулы для вычисления адресов строк матрицы в байтах
Адрес 0-й строки матрицы	Адрес матрицы $0 * \langle \text{число столбцов} \rangle * \text{sizeof}(\langle \text{тип элементов матрицы} \rangle)$
Адрес 1-й строки матрицы	Адрес матрицы $1 * \langle \text{число столбцов} \rangle * \text{sizeof}(\langle \text{тип элементов матрицы} \rangle)$
Адрес 2-й строки матрицы	Адрес матрицы $2 * \langle \text{число столбцов} \rangle * \text{sizeof}(\langle \text{тип элементов матрицы} \rangle)$
Адрес <i>i</i> -й строки матрицы	Адрес матрицы $i * \langle \text{число столбцов} \rangle * \text{sizeof}(\langle \text{тип элементов матрицы} \rangle)$

Адрес каждого элемента матрицы рассчитывается по формулам, представленным в табл. 10.5.

Таблица 10.5. Вычисление адресов элементов i -й строки в статической матрице в байтах

Адреса элементов i -й строки матрицы в байтах	Формулы для вычисления адресов элементов i -й строки матрицы в байтах
Адрес 0-го элемента	Адрес матрицы*+<число столбцов>*sizeof(<тип элементов матрицы>)+0*sizeof(<тип элементов матрицы>)
Адрес 1-го элемента	Адрес матрицы*+<число столбцов>*sizeof(<тип элементов матрицы>)+1*sizeof(<тип элементов матрицы>)
Адрес 2-го элемента	Адрес матрицы*+<число столбцов>*sizeof(<тип элементов матрицы>)+2*sizeof(<тип элементов матрицы>)
Адрес j -го элемента	Адрес матрицы*+<число столбцов>*sizeof(<тип элементов матрицы>)+ j *sizeof(<тип элементов матрицы>)

10.8. Динамические двумерные массивы (только для И-32,33)

Если в программе возникает необходимость использовать матрицу (двумерный массив), количество строк и столбцов в которой заранее неизвестно, то определить ее статическим способом невозможно, так как компилятору неизвестно количество строк и столбцов матрицы и, соответственно, неизвестен размер области, которую необходимо зарезервировать. И следующая последовательность операторов

```
int n;           //описание переменной n — числа строк матрицы
int k;           //описание переменной k — числа столбцов матрицы
n = 3;           //присваивание числу строк матрицы значения 3
k = 3;           //присваивание числу столбцов матрицы значения 3
int A[n][k];     //описание целочисленной матрицы A
```

при компиляции выдаст ошибку о том, что при описании матрицы в качестве количеств строк и столбцов необходимы константы.

Таким образом, при использовании матриц, количество строк и столбцов в которых заранее неизвестно, приходится пользоваться динамически распределяемой памятью, а полученные таким образом матрицы называются *динамическими*.

Для динамических матриц память выделяется в программе в тот момент, когда это необходимо программисту, и количество строк и столбцов можно либо ввести с клавиатуры, либо вычислить по определенному алгоритму. При этом выделяется область памяти, которая не имеет своего имени (неименованная область памяти). Доступ к выделенной области возможен только через указатель — имя динамической матрицы, который содержит номер первого байта выделенной области памяти, и является указателем на матрицу.

Объявление динамических массивов. Динамические матрицы являются одномерными динамическими массивами, элементами которых являются указатели на одномерные динамические массивы, и объявляются следующим образом:

```
<тип_элементов_матрицы> **<имя_матрицы>;
```

Здесь <тип_элементов_массива> может быть любым допустимым в языке C++ типом;

<имя_матрицы> — является указателем, т. е. может содержать адрес некоторой области памяти, в которой может находиться массив указателей (адреса начала каждой строки матрицы) на заданный тип.

Например, в следующем фрагменте программы

```
int n;           //описание целой переменной — числа строк матрицы
int k;           //описание целой переменной — числа столбцов матрицы
int **M;         //указатель на одномерный массив указателей
```

описываются целочисленные переменные n, k и динамическая матрица M.

Выделение и освобождение памяти для динамической матрицы. Для выделения памяти под динамическую матрицу необходимо воспользоваться представлением матрицы как одномерного массива указателей (см. рис. 10.4). То есть сначала надо зарезервировать память под одномерный массив указателей на строки с помощью оператора

```
<имя_матрицы> = new <тип_элементов_массива*>[<кол-во_строк_матрицы>];
```

При этом получится распределение памяти, показанное на рис. 10.6.

Например, фрагмент программы

```
...
printf("Введите число строк матрицы ");
```



```

scanf ("%d", &n);           //вывод приглашения к вводу числа строк матрицы
M=new int*[n];              //ввод числа строк матрицы
//выделение памяти под n указателей на целочисленные строки матрицы M
...
```

выделяет память под указатель на массив указателей на n строк целочисленной матрицы M .

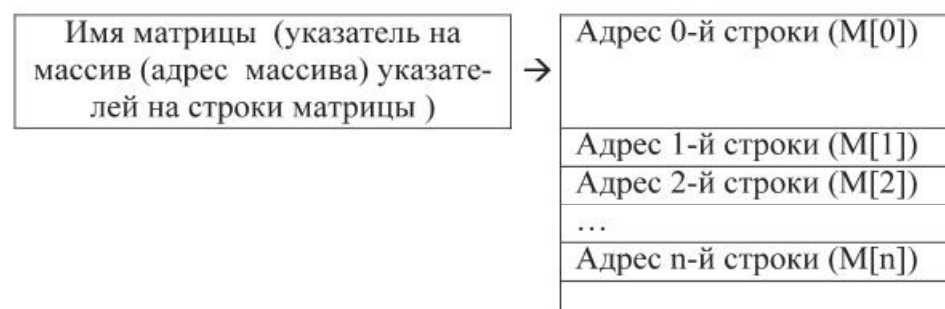


Рис. 10.6. Распределение памяти после выделения памяти под одномерный массив указателей на строки

После резервирования памяти под массив указателей на строки матрицы необходимо выделить память под каждую из строк матрицы с помощью операторов

```

for (i=0; i<число_строк_матрицы; i++)
<имя_матрицы>[i]=new <тип элементов матрицы> [количество_столбцов_матрицы]
```

Например, фрагмент программы:

```

...printf("Введите число столбцов матрицы ");
//вывод приглашения к вводу числа столбцов матрицы
scanf ("%d", &k); //ввод числа столбцов матрицы
for (i=0; i<n; i++) //в цикле
M[i]=new int[k]; //выделение памяти под i-ю строку матрицы M
```

После выделения памяти с динамической матрицей можно работать точно так же, как с обычной матрицей, т. е. можно использовать все алгоритмы и методы, описанные выше.

Когда память, выделенная под динамическую матрицу, больше не нужна, ее следует освободить. Делается это в порядке, обратном процессу выделения памяти под матрицу: сначала освобождается память под строки матрицы

```

for (i=0; i<число_строк_матрицы; i++)
delete []имя_динамической_матрицы[i];
```

Пример.

```
for (i=0; i<n; i++) //в цикле
delete []M[i];
//освобождаем память, выделенную под i-ю строку матрицы M
```

Далее необходимо освободить память, выделенную под массив указателей на строки матрицы

```
delete []имя_динамической_матрицы;
```

Пример.

```
delete []M;
//освобождение памяти, выделенной под массив указателей
//на строки матрицы
```

Решение задач с использованием динамических матриц. Рассмотрим решение задач с использованием динамических матриц.

Пример. Ввести с клавиатуры количество строк и столбцов целочисленной матрицы. Сформировать матрицу с помощью датчика случайных чисел. Вывести матрицу на экран. Найти произведение всех элементов матрицы.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int **X = NULL; //описание и инициализация указателя на матрицу X
    int n;           //описание переменной n — числа строк матрицы
    int k;           //описание переменной k — числа столбцов матрицы
    int i, j;
    double Pr;
    srand(time(NULL));
    printf("Введите число строк матрицы n = ");
    scanf("%d", &n);
    printf("Введите число столбцов матрицы k = ");
    scanf("%d", &k);
    X = new int*[n]; //выделение памяти под указатели на строки матрицы
    for (i=0; i<n; i++) //в цикле по строкам
        X[i] = new int[k]; //выделяем память под i-ю строку матрицы
    for (i=0; i<n; i++) //в цикле по строкам
        for (j=0; j<k; j++) //в цикле по столбцам
            X[i][j] = rand()%101-50; //формирование элемента матрицы
    printf("Матрица X\n");
```

```

for (i=0; i<n; i++){                               //в цикле по строкам
for (j=0; j<k; j++)                                //в цикле по столбцам
printf("%4d ",X[i][j]);                            //вывод элемента матрицы
printf("\n");    //перевод курсора на начало следующей строки экрана
}
Pr=1;        //присвоение начального значения произведения (Pr)
for (i=0; i<n; i++)                                //в цикле по строкам
for (j=0; j<k; j++)                                //в цикле по столбцам
Pr=Pr*X[i][j]; //умножение старого значения Pr на элемент матрицы
printf("Произведение всех элементов массива Pr =
%lf\n", Pr);
getchar();
return 0;
}

```

Тесты

1. Элемент матрицы A[4][2] находится на пересечении:

- а) 2-й строки и 4-го столбца;
- б) 4-й строки и 2-го столбца;
- в) правильного ответа нет.

2. В каком диапазоне будут находиться значения элементов матрицы A?

```

...
for (i=0; i<3; i++)
for (j=0; j<5; j++)
    A[i][j]=rand()%300-200;
...

```

- а) от 0 до 300;
- б) от 0 до 200;
- в) от -200 до 1010.

3. В каком порядке производится заполнение значениями матрицы A размером 10x10 в следующем фрагменте программы?

```

...
for (i=0; i<10; i++)
for (j=10; j>=0; j--)
    A[i][j]=rand()%100;
...

```

- а) по строкам справа налево;
- б) по столбцам снизу вверх;
- в) правильного ответа нет.

4. В каком порядке производится заполнение значениями матрицы A размером 10x10 в следующем фрагменте программы?

```
...  
for (j=0; j<10; j++)  
for (i=10; i>=0; i--)  
    do A[i][j]=rand()%100;  
...
```

- а) по строкам справа налево;
- б) по столбцам снизу вверх;
- в) правильного ответа нет.

5. Что делает следующий фрагмент программы, если дана матрица A размером 3x5?

```
...  
S:=0;  
for (i=0; i<3; i++)  
for (j=0; j<5; j++)  
    if (A[i][j]<0)  
        S=S+A[i][j];  
...
```

- а) присваивает переменной S значение суммы всех элементов матрицы A;
- б) присваивает переменной S значение суммы отрицательных элементов матрицы A;
- в) правильного ответа нет.

6. Что делает следующий фрагмент программы, если дана матрица A размером 3x5?

```
...  
M=A[0][0];  
for (i=0; i<3; i++)  
for (j=0; j<5; j++)  
    if (A[i][j]<M)  
        M=A[i][j];  
...
```

- а) присваивает всем элементам матрицы A значение переменной M;
- б) присваивает переменной M значение наименьшего элемента матрицы A;
- в) правильного ответа нет.

7. Дана матрица

$$H = \begin{pmatrix} 4,1 & -3,14 & 17 \\ -13 & 0 & 9,8 \\ 113 & 53,2 & -37,8 \end{pmatrix}.$$

Какой массив К сформируется в результате выполнения следующего фрагмента программы?

```
...  
for (i=0; i<3; i++)  
{  
    M=H[i][0];  
    for (j=1; j<3; j++)  
        if (H[i][j]>M)  
            Then M:=H[i,j];  
    K[i]=M;  
}  
...
```

- а) $K = (113 \quad 53.2 \quad 17);$
- б) $K = (17 \quad 10.8 \quad 113);$
- в) $K = (-3.14 \quad -13 \quad -37.8).$

8. Что делает следующий фрагмент программы, если дана матрица Н размером 12х5?

```
...  
for (j=0; j<5; j++)  
{  
    P=1;  
    for (i=0; i<12; i++)  
        P=P*H[i,j];  
    K[j]=P;  
}  
...
```

- а) присваивает переменной Р значение произведений всех элементов матрицы Н;
- б) формирует массив К из произведений всех элементов каждого столбца матрицы Н;
- в) формирует массив К из произведений всех элементов каждой строки матрицы Н.

Задания

1. Ввести с клавиатуры матрицу $P\{7,10\}$. Вывести ее на экран. Найти сумму отрицательных элементов матрицы, произведение положительных элементов, количество ненулевых элементов.

2. Ввести с клавиатуры матрицу $H\{5,7\}$ построчно. Вывести ее на экран. Все ненулевые элементы заменить обратными по величине (обратным для числа a является число $1/a$) и противоположными по знаку. Вывести измененную матрицу на экран.

3. Сформировать матрицу $S\{12,12\}$. Вывести матрицу. Заменить нулями все элементы, расположенные на главной диагонали и выше ее. Вывести измененную матрицу.

4. Ввести матрицу $T\{4,4\}$. Вывести ее на экран. Вычислить произведение положительных элементов, лежащих под главной диагональю.

5. Сформировать случайным образом целочисленную матрицу $N \times N$. Сформировать два одномерных массива A и B : в массив A записать элементы матрицы, расположенные на главной диагонали и ниже нее, в массив B — элементы, лежащие на побочной диагонали и ниже нее. Вывести оба массива на экран теми же цветами, что главная и побочная диагональ матрицы соответственно.

6. Ввести с клавиатуры матрицу $C\{4,3\}$. Вывести ее на экран. Создать из матрицы C матрицу P таким образом, что все столбцы матрицы C станут строками матрицы P , а все строки — столбцами матрицы P в том же порядке (1-й столбец в C станет первой строкой в P), т. е. $P = C^T$. Вывести матрицу P .

7. Ввести с клавиатуры матрицу $M\{3,4\}$ и число N . Вывести ее на экран. Определить номер 1-го элемента, большего N , и посчитать сумму элементов, ему предшествующих, если считать от левого верхнего угла матрицы построчно.

8. Латинским квадратом порядка N называется квадратная таблица размером $N \times N$, каждая строка и каждый столбец которой содержат все числа от 1 до N . Для заданного N (вводится с клавиатуры) в матрице $L\{N,N\}$ построить латинский квадрат порядка N .

9. Составить программу упорядочения по возрастанию элементов каждой строки матрицы C (размеры матрицы произвольные, задаются с клавиатуры).

10. Сформировать матрицу $K\{12,15\}$. Поменять местами минимальный и максимальный элементы. Вывести матрицу до замены и после.

11. Сформировать матрицу $A\{6,7\}$. Вывести ее на экран. Найти произведение элементов строки, в которой находится элемент с наибольшим значением.

12. Ввести матрицу $A\{7,8\}$. Сформировать три одномерных массива: B — из максимальных элементов строк матрицы A ; C — из минимальных элементов столбцов матрицы A ; D — из средних арифметических элементов строк матрицы A .

13. Сформировать вещественную матрицу $A\{6,10\}$. Найти среднее арифметическое наибольшего и наименьшего значений ее элементов, расположенных ниже главной диагонали.

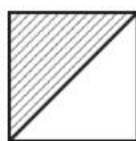
14. Сформировать матрицу $K\{5,7\}$. Вывести матрицу. Найти и вывести значение минимального элемента. Вывести координаты всех минимальных элементов матрицы и их количество.

15. Дана целочисленная матрица 10×10 . Сформировать одномерный массив, состоящий из сумм отрицательных элементов строк, и одномерный массив, состоящий из сумм отрицательных элементов столбцов.

16. Сформировать матрицу $B\{10,10\}$. Все элементы с наибольшим значением в данной матрице заменить нулями, если эти элементы не принадлежат 1-й или последней строкам. Вывести матрицу до и после замены.

17. Используя генератор случайных чисел, сформировать матрицу $H\{10,8\}$ из вещественных чисел. Вывести матрицу H . Получить новую матрицу C из матрицы H путем деления всех элементов матрицы H на ее наибольший по модулю элемент.

18. Дана целочисленная матрица 8×8 . Найти ее минимальный элемент и сумму элементов заштрихованной области.



19. Дана целочисленная матрица 8×8 . Найти ее минимальный элемент и сумму элементов заштрихованной области.



20. Дана целочисленная матрица размерностью $M \times K$. Найти минимальное значение для верхней половины матрицы и максимальное значение для нижней половины.

21. Дана целочисленная матрица 8×8 . Найти ее максимальный элемент и сумму элементов заштрихованной области.



22. Ввести символьную матрицу 5×7 . Сформировать одномерный массив из сумм кодов символов каждой строки и одномерный массив из сумм кодов символов каждого столбца.

23. Дана целочисленная матрица $M \times K$. Сформировать новую матрицу путем прибавления в каждой строке к каждому элементу исходной матрицы максимального значения элемента данной строки.

24. Дана матрица 7×6 . Сформировать одномерный массив, содержащий сумму максимального и минимального элементов каждой строки. Вывести матрицу и массив на экран.

25. Ввести символьную матрицу 5×7 . Сформировать одномерный массив из минимальных кодов символов каждой строки и одномерный массив из максимальных кодов символов каждого столбца.

26. Дана целочисленная матрица $M \times K$. Сформировать новую матрицу путем прибавления в каждом столбце к каждому элементу исходной матрицы минимального значения элемента данного столбца.

27. Дана матрица 7×7 . Сформировать одномерный массив, содержащий элементы главной диагонали исходной матрицы. Вывести матрицу и массив на экран.

28. Сформировать вещественную матрицу $A\{6,6\}$. Найти среднее арифметическое наибольшего и наименьшего значений ее элементов, расположенных ниже главной диагонали.

29. Сформировать матрицу $K\{5,7\}$. Вывести матрицу. Сформировать одномерный массив из минимальных элементов каждой строки матрицы.

30. Ввести с клавиатуры матрицу $Q\{4,5\}$. Вывести ее на экран. Найти среднее арифметическое элементов каждой строки матрицы и вычесть его из элементов этой строки.

31. Сформировать вещественную матрицу $Q\{14,14\}$. Построить одномерный массив P по правилу: если в i -й строке матрицы элемент, принадлежащий главной диагонали, отрицателен, то элемент массива P_i равен сумме положительных элементов i -й строки матрицы, в противном случае P_i равен произведению отрицательных элементов i -й строки матрицы. Вывести массив P на экран.

32. Дана целочисленная матрица $M \times K$. Найти минимальное значение для верхней половины матрицы и максимальное значение для нижней половины.

33. Целочисленную матрицу $K\{8,8\}$ заполнить нулями и единицами, расположив их в шахматном порядке. Дополнительно: сделать то же самое для квадратной матрицы произвольного порядка.

Контрольные вопросы

1. Что такое «матрица»? Приведите примеры.
2. Каким образом элементы матрицы записываются в оперативную память?
3. Данные каких типов могут быть элементами матрицы? Приведите примеры.
4. Назовите атрибуты матрицы. Поясните на примерах.
5. Как объявить двумерный массив в программе? Приведите примеры.
6. Можно ли в программе объявить число строк и число столбцов матрицы как константы? Для чего это делается? Приведите примеры.
7. Можно ли не объявлять размеры матрицы не как константы? Будет ли это считаться ошибкой? Объясните.
8. Можно ли задавать размеры матрицы при каждом запуске программы? Как это сделать? Приведите пример.
9. Что такое «индекс элемента матрицы»? Какие индексы имеют элементы матрицы? Приведите примеры.
10. Как получить доступ к конкретному элементу матрицы? Приведите примеры.
11. Каким образом можно сформировать матрицу? Приведите примеры.
12. Как вывести матрицу на экран? Приведите примеры.
13. Что такое главная диагональ матрицы? Как в программе определить, принадлежит ли элемент матрицы главной диагонали? Приведите примеры.
14. Что такое побочная диагональ матрицы? Как в программе определить, принадлежит ли элемент матрицы побочной диагонали? Приведите примеры.
15. Для каких матриц существуют главная и побочная диагонали? Приведите примеры.
16. Что такое порядок матрицы? Объясните на примере.
17. Для чего в программе можно использовать двумерный массив? Приведите пример.
18. Как вычислить сумму всех элементов матрицы? Приведите пример.
19. Как вычислить произведение всех элементов матрицы? Приведите пример.
20. Как найти среднее арифметическое всех элементов матрицы?
21. Как определить максимальный элемент двумерного массива? Приведите пример.
22. Как определить координаты максимального элемента двумерного массива? Приведите пример.
23. Как найти минимальный элемент двумерного массива? Приведите пример.

24. Как определить координаты минимального элемента двумерного массива? Приведите пример.
25. Как найти все максимальные (минимальные) элементы матрицы? Приведите примеры.
26. Можно ли сформировать одномерный массив из элементов матрицы? Как это сделать? Приведите примеры.
27. Чем формирование одномерного массива по строкам матрицы отличается от формирования одномерного массива по столбцам матрицы? Поясните на примерах.
28. Можно ли сформировать одномерный массив из элементов главной (побочной) диагонали матрицы? Приведите примеры.

Глава 11

ПОНЯТИЕ ФУНКЦИИ

Помимо структурирования при написании программ применяется технология программирования «сверху вниз» (нисходящее программирование). В основе этой технологии лежит идея дробления сложной задачи на более простые подзадачи до тех пор, пока не станут очевидны все детали решения. В этом случае такие подзадачи обычно оформляются в виде подпрограмм. Так как каждая подпрограмма решает некоторую логически законченную элементарную задачу (например, вывод одномерного массива на экран), то она может использоваться в программе несколько раз. В результате это приводит к упрощению написания программы и уменьшению ее объема. В отличие от других языков программирования (например, Паскаля), где программисту доступны процедуры и функции, в языке C++ используются только функции.

11.1. Функции

Понятие функции. Функцией в языке C++ называется часть программы, оформленная особым образом, которая выполняет логически законченные действия или логически законченные вычисления. Например:

- функции ввода-вывода `printf`, `scanf` стандартной библиотеки ввода-вывода;
- функции работы с датчиком случайных чисел `srand`, `rand` библиотеки `stdlib`;
- функция работы с таймером `time` библиотеки `time` и т. д.

Назначение функций. В программах функции используются в двух случаях:

- для упрощения решения сложных задач (программирование «сверху вниз»);
- при использовании одинаковых алгоритмов с различными объектами.

В первом случае сложная задача (программа) разбивается на ряд более простых подзадач. Каждая подзадача, в свою очередь, разбивается на несколько более простых подзадач и т. д. Разбиение завершается, когда полученную подзадачу можно реализовать с помощью функции не более чем из десяти, двадцати операторов.

Во втором случае, гораздо удобнее и надежнее один раз запрограммировать и тщательно проверить несколько небольших функций, а затем использовать их в программах в различных назначениях, например, функции стандартных библиотек C++.

Классификация функций. Функции по числу результатов подразделяют на три группы:

- функции с одним результатом;
- функции с несколькими результатами;
- функции, не имеющие результатов.

Функции с одним результатом возвращают в вызывающую функцию одно-единственное значение определенного типа (например, сумма элементов массива, площадь треугольника и т. д.). Эти функции обычно выполняют вычисления по определенному алгоритму и полученное единственное значение возвращают в вызывающую функцию с помощью специального оператора `return`.

Функции с несколькими результатами передают в вызывающую функцию два или более значений разных типов или массив (например, формирование целочисленного массива, вычисление характеристик геометрической фигуры и т. д.). Эти функции обычно выполняют вычисления нескольких значений, которые передают в вызывающую функцию с помощью параметров, используя ссылки или указатели.

Функции, не имеющие результатов, не возвращают никаких данных в вызывающую функцию. Эти функции обычно выполняют определенные действия, не требующие никаких вычислений, например вывод на экран сообщения, рисование графического объекта, вывод одномерного массива и т. д.

11.2. Определение функции. Объявление (прототип) функции

Возвращаемое значение функции. Возвращаемым значением называется один-единственный результат работы функции, который возвращается в вызывающую функцию с помощью специального оператора `return`.

Функции с одним результатом имеют возвращаемое значение, *функции с несколькими результатами и функции, не имеющие результатов*, не имеют возвращаемого значения, результаты их работы передаются через параметры для функций с несколькими результатами или полностью отсутствуют для функций, не имеющих результатов.

Определение функции. Определение функции имеет следующий вид:

```
<тип> <имя_функции>(<список формальных параметров>)  
{  
  <операторы>  
}
```

где <тип> — тип возвращаемого значения или ключевое слово `void`, если функция не имеет возвращаемого значения;

<имя_функции> задается по правилам задания идентификаторов (имен);

<список формальных параметров> — содержит список параметров, через которые функция обменивается информацией с вызывающей ее функцией.

Элементы списка формальных параметров имеют следующий формат:

```
<тип параметра> <имя параметра>
```

<операторы> — любые операторы, которые необходимы для работы функции.

Объявление (прототип) функции. Объявление функции или *прототип функции* имеет следующий вид:

```
<тип> <имя_функции>(<список типов формальных параметров>);
```

где <тип> — тип возвращаемого значения или ключевое слово `void`, если функция не имеет возвращаемого значения;

<имя_функции> задается по правилам задания идентификаторов (имен);

<список типов формальных параметров> — содержит список типов параметров, через которые функция обменивается информацией с вызывающей ее функцией.

В программе прототип функции может помещаться перед заголовком функции `main`, сразу после директив препроцессора. А опре-

деление функции может размещаться после определения функции `main`. В этом случае в функции `main` известны (из прототипа) имя вызываемой функции, тип возвращаемого значения и список типов формальных параметров.

Если в программе с функциями не используются прототипы, то определения функций должны размещаться перед заголовком функции `main`.

Кроме этого, объявление функций может быть помещено в заголовочный файл (`*.h`), а определение — в отдельный файл с кодом программы (`*.cpp`).

11.3. Формальные и фактические параметры

При запуске программы на выполнение коды операторов функции `main` и необходимые данные записываются в постоянную (статическую) область оперативной памяти, которая возникает в момент начала выполнения программы и исчезает только после окончания выполнения программы. То есть *постоянная память используется программой во время всей ее работы*.

Кроме постоянной памяти, при запуске программы возникает *временная память*. Эта память используется для выполнения функции, вызванной из функции `main`. Вызванная функция вместе со всеми необходимыми данными для ее работы загружается во временную память. Затем происходит выполнение функции, и управление снова передается функции `main` в постоянную память. *Временная память доступна только во время работы функции*.

Во время работы функция обрабатываются данные, переданные из вызывающей функции, а по завершении выполнения функция может возвращать результат вызывающей функции.

Взаимодействие между вызывающей и вызываемой функциями может осуществляться через параметры. Различают *формальные* и *фактические* параметры.

Формальные параметры. Имена формальных параметров задаются в <списке формальных параметров> заголовка функции. Перед именем формального параметра через пробел указывается его тип, формальные параметры в списке разделяются запятой.

Например, в заголовке функции

```
double srednee(double t, double u, double v)
```


используются три формальных параметра:

- 1-й формальный параметр `t` имеет тип `double` (1-е значение из трех, для которых вычисляется среднее);
- 2-й формальный параметр `u` имеет тип `double` (второе значение из трех, для которых вычисляется среднее);
- 3-й формальный параметр `v` имеет тип `double` (третье значение из трех, для которых вычисляется среднее).

Формальные параметры используются внутри функции для указания того, какие действия или вычисления необходимо выполнять над данными, переданными из вызывающей или возвращаемыми в вызываемую функцию. Эти параметры называются формальными, так как не могут иметь никакого конкретного значения, размещаются во временной памяти и существуют только во время работы функции.

Фактические параметры. Имена фактических параметров задаются в <списке фактических параметров> при вызове функции. При этом на место формальных параметров подставляются конкретные значения.

Фактические параметры используются в функции `main` и недоступны вызываемой функции. Они называются фактическими, так как имеют конкретные числовые значения и размещаются в постоянной памяти программы.

При этом каждый формальный параметр заменяется копией фактического параметра (или его адресом) и все действия внутри функции производятся с копией соответствующего фактического параметра (или его адресом).

Соответствие формальных и фактических параметров. Между формальными и фактическими параметрами должно выполняться соответствие:

- по количеству (формальных и фактических параметров должно быть одинаковое количество);
- по порядку следования (порядок следования фактических и формальных параметров должен быть один и тот же);
- по типам (тип каждого фактического параметра должен совпадать с типом соответствующего формального параметра).

Например, в заголовке функции нахождения среднего трех значений типа `double`

```
double srednee(double t, double u, double v);
```


присутствуют три формальных параметра:

- 1-й формальный параметр `t` имеет тип `double` (первое значение из трех, для которых вычисляется среднее);
- 2-й формальный параметр `u` имеет тип `double` (второе значение из трех, для которых вычисляется среднее);
- 3-й формальный параметр `v` имеет тип `double` (третье значение из трех, для которых вычисляется среднее).

При вызове этой функции

```
Z=srednee(a,b,c);
```

также должны присутствовать три фактических параметра:

- 1-й фактический параметр `a` имеет тип `double` (первое значение из трех, для которых вычисляется среднее);
- 2-й фактический параметр `b` имеет тип `double` (второе значение из трех, для которых вычисляется среднее);
- 3-й фактический параметр `c` имеет тип `double` (третье значение из трех, для которых вычисляется среднее).

Имена формальных и фактических параметров могут совпадать, так как формальные и фактические параметры хранятся в разных областях памяти (во временной и в постоянной областях памяти).

Способы использования параметров. Функциям для работы могут потребоваться данные из вызывающей функции. Передача данных из вызывающей функции в вызываемую осуществляется через *параметры*. В языке C++ используются три способа передачи данных через параметры:

- передача по значению;
- передача по ссылке;
- передача по указателю.

11.4. Передача по значению

Передача по значению. При передаче по значению в функцию передается копия значения фактического параметра. При передаче по значению в списке формальных параметров указывается только тип и имя формального параметра, как, например, в заголовке функции нахождения среднего из трех значений типа `double`

```
double srednee(double x, double y, double z)
```

При вызове функции при передаче по значению в качестве фактического параметра может использоваться либо константа, либо переменная, либо выражения того же типа, что и соответствующий формальный параметр, например:

`X=srednee(3.14, a, b*b+4)` //a,b - переменные типа double

При использовании выражения в качестве фактического параметра сначала происходит вычисление значения выражения, затем копия полученного значения передается в вызываемую функцию.

Пример 1. Определить функцию $srednee(x, y, z) = \frac{x + y + z}{3}$ и вычислить $t = srednee(a, b, c) + \frac{srednee(2a, 3b, 4c + 5)}{2}$.

```
#include <stdio.h>
double srednee(double x, double y, double z) /*заголовок
      функции нахождения среднего из трех значений типа double */
{ double sr; /*описание локальной переменной типа double
  sr=(x+y+z)/3; /*вычисление среднего из трех значений
  return sr; /*возврат вычислен. значения sr в вызывающую функцию
}
int main() { /*заголовок функции main
double a,b,c,t;
printf("Введите a,b,c\n");
scanf("%lf%lf%lf",&a,&b,&c);
t=srednee(a,b,c)+srednee(2*a,3*b,4*c+5)/2;
/*вычисление значения t
printf("t=%lf\n",t);
getchar();
return 0;
}
```

Пояснение. В этой программе определена функция `srednee`, которая определяет среднее значение трех параметров типа `double`. Для передачи информации в функцию `srednee` используется способ передачи по значению.

В функции `main` сначала происходит ввод конкретных числовых значений переменных `a`, `b`, `c`. Затем при первом вызове этой функции — `srednee(a,b,c)` в нее передаются копии введенных значений `a`, `b`, `c`. Вызванная функция `srednee` вычисляет среднее от трех

полученных копий числовых значения и полученный результат (число) возвращает в качестве первого слагаемого в выражение для вычисления значения t .

При втором вызове функции `srednee(2*a, 3*b, 4*c+5)` сначала вычисляются значения выражений $2*a$, $3*b$ и $4*c+5$, затем копии полученных числовых значений передаются в функцию `srednee`. Функция вычисляет среднее от трех полученных копий числовых значений и полученный результат (число) возвращает в соответствующее место выражения для вычисления значения t .

Затем производится вычисление значения t по заданной формуле и полученное значение выводится на экран (оператор `printf("t=%lf\n", t);`)

Пример 2. Определить функцию $srednee(x, y, z) = \frac{x + y + z}{3}$ и вычислить $t = srednee(a, b, c) + \frac{srednee(2a, 3b, 4c + 5)}{2}$.

При составлении программы использовать прототипы.

```
#include <stdio.h>
double srednee(double, double, double); /*прототип функции
                                         нахождения среднего из трех значений типа double*/

int main() {                               //заголовок функции main
    double a, b, c, t;
    printf("Введите a, b, c\n");
    scanf("%lf%lf%lf", &a, &b, &c);
    t = srednee(a, b, c) + srednee(2*a, 3*b, 4*c+5) / 2; //вычисление
                                                         //значения t

    printf("t=%lf\n", t);
    getchar();
    return 0;
}

double srednee(double x, double y, double z) /*заголовок
                                              функции нахождения среднего из трех значений типа double*/
{double sr;                                //описание локальной переменной типа double
  sr = (x+y+z) / 3;                        //вычисление среднего из трех значений
  return sr;                               //возврат вычислен. значения sr в вызывающую функцию
}
```

Пояснение. В программе задан прототип (объявление) функции

```
double srednee(double, double, double);
```


который определяет:

- имя функции — `srednee`;
- тип возвращаемого значения (`double` перед именем функции);
- количество параметров (3);
- тип каждого параметра (`double`).

Прототип находится после директив препроцессора перед заголовком функции `main`. Соответственно функция `main` будет «знать» всю необходимую для использования функции `srednee` информацию.

Определения функции `srednee` находится после окончания функции `main` и полностью совпадает с определением этой функции в предыдущей программе. Текст функции `main` и работа программы такие же, как и в примере 1.

Пример 3. Определить функцию $srednee(x, y, z) = \frac{x + y + z}{3}$ и вы-

числить $t = srednee(a, b, c) + \frac{srednee(2a, 3b, 4c + 5)}{2}$.

Разместить функцию в заголовочном файле `my_func`.

Содержимое файла `my_func.h`:

```
#ifndef _my_func_    //если не определено текстовое имя _my_func_
#define _my_func_    //определить текстовое имя _my_func_
double srednee(double, double, double);
                                     //прототип функции srednee
#endif
```

Содержимое файла `my_func.cpp`:

```
#include "my_func.h"
                                     //подключение заголовочного файла _my_func.h
double srednee(double a, double b, double c)
                                     //заголовок функции srednee
{
    return ( a + b + c ) /3;
                                     //возврат вычисленного среднего значения в вызывающую функцию
}
```

Содержимое файла с проектом

```
#include <stdio.h>
#include "my_func.h»
int main()
```

```

{
double a,b,c,t;
printf("Введите a,b,c\n");
scanf("%lf%lf%lf",&a,&b,&c);
t=srednee(a,b,c)+srednee(2*a,3*b,4*c+5)/2;    //вычисление
                                              //значения t

printf("t=%lf\n",t);
getchar();
return 0;
}

```

11.5. Передача по ссылке

Передача по ссылке. При передаче по ссылке в функцию передается ссылка на фактический параметр. То есть по определению ссылки в функцию передается адрес соответствующего фактического параметра, а сам фактический параметр остается в вызывающей функции. При передаче по ссылке в списке формальных параметров указывается тип формального параметра, знак операции задания ссылки (&) и имя формального параметра, как, например, в прототипе функции перемены местами двух значений

```
void swap1(int &x, int &y).
```

При вызове функции при передаче по ссылке в качестве фактического параметра может использоваться только переменная того же типа, что и соответствующий формальный параметр, например:

```

int k = 58,n = 125;    //описание целых переменных
swap1(k,n);            //вызов функции swap1

```

Пример 1. Задать значение целым переменным k, n и с помощью функции swap1 поменять местами эти значения. Программу составлять с использованием прототипов

```

#include <stdio.h>
void swap1(int&, int&);    //прототип функции перемены местами двух значений

int main() {
int k,n;
k=3;    //присвоение переменной k значения 3
n=5;    //присвоение переменной n значения 5

```

```

swap1(k,n);          /*после вызова функции swap1 k стало равно 5,
                        а n стало равно 3*/

printf("k=%d n=%d\n",k,n);
getchar();
return 0;
}

void swap1(int &x, int &y)
    //заголовок функции перемены местами двух значений
{int u;                //описание целой локальной переменной
u=x;                   //присвоение u значения переменной x, u стало равно x
x=y;                   //присвоение x значения переменной y, x стало равно y
y=u;                   //присв. y знач. переменной u (u=x), y равно x (u=x)
return;                /*если функция не возвращает никакого значения, в прототипе
                        указан тип void, то можно воспользоваться оператором return для
                        выхода из функции*/
}

```

Пояснение. В этой программе задан прототип (объявление) функции

```
void swap1(int&, int&;,
```

который определяет:

- имя функции — `swap1`;
- сведение о том, что функция не имеет возвращаемого значения (`void` перед именем функции);
- количество параметров (2);
- тип каждого параметра (ссылка на `int`).

Прототип находится после директив препроцессора перед заголовком функции `main`. Соответственно функция `main` будет «знать» всю необходимую для использования функции `swap1` информацию.

Определение функции `swap1` находится после окончания функции `main`. Функция `swap1` меняет местами два значения типа `int`. Для передачи информации в функцию `swap1` используется способ передачи по ссылке.

В функции `main` сначала происходит задание конкретных числовых значений переменных `k` (`k=3`) и `n` (`n=5`). При вызове функции — `swap1(k,n)`; в нее передаются адреса переменных `k,n`. Вызванная функция `swap1`, используя переданные адреса, находит в области памяти функции `main` соответствующие значения переменных `k,n` и в области памяти функции `main` меняет местами значения этих переменных. Затем новые значения переменных `k,n` выводятся на экран.

11.6. Передача по указателю

Передача по указателю. При передаче по указателю в функцию передается копия значения указателя — фактического параметра. То есть по определению указателя в функцию передается копия адреса области памяти, хранящегося в фактическом параметре-указателе, а сам фактический параметр остается в вызывающей функции. При передаче по указателю в списке формальных параметров указывается тип формального параметра, знак операции задания указателя (*) и имя формального параметра, как, например, в заголовке функции перемены местами двух значений

```
void swap2(int *px, int *py).
```

При вызове функции при передаче по указателю в качестве фактического параметра может использоваться только переменная-указатель того же типа, что и соответствующий формальный параметр, например:

```
int k,n;                                //описание целых переменных
int *pk=&k,*pn=&n;    //описание и инициализация указателей pk, pn
swap2(pk,pn);        //вызов функции swap2
```

Пример 1. Задать значение целым переменным k, n и с помощью функции swap2 поменять местами эти значения.

```
#include <stdio.h>
void swap2(int *px, int *py)
    //заголовок функции перемены местами двух значений
{
    int u;
    u=*px;                                //u стало равно *px
    *px=*py;                              //*px стало равно *py
    *py=u;                                //*py стало равно *px (u=*px)
}
int main() {
    int k,n;                                //описание целых переменных
    int *pk=NULL, *pn=NULL;                //описание указателей pk, pn
                                           //и инициализация их пустыми значениями*/
    k=3;                                   //присвоение переменной k значения 3
    n=5;                                   //присвоение переменной n значения 5
    pk=&k;                                  //запись в указатель pk адреса переменной k (*pk=3)
    pn=&n;                                  //запись в указатель pn адреса переменной n (*pn=5)
```

```

printf("pk=%d pn=%d\n", *pk, *pn);
//вывод на экран старых значений *pk и *pn
swap2(pk, pn);
/*после вызова функции swap2 *pk стало
равно 5, а *pn стало равно 3*/
printf("pk=%d pn=%d\n", *pk, *pn);
//вывод на экран новых значений *pk и *pn
printf("k=%d n=%d\n", k, n);
//вывод на экран новых значений k(5) и n(3)
swap2(&k, &n);
/*после вызова функции swap2 k стало
равно 3, а n стало равно 5*/
printf("k=%d n=%d\n", k, n); //вывод на экран значений k(3) и n(5)
getchar();
return 0;
}

```

Пояснение. В этой программе определена функция `swap2`, которая меняет местами два значения типа `int`. Для передачи информации в функцию `swap2` используется способ передачи по указателю.

В функции `main` сначала происходит задание конкретных числовых значений переменных `k` (`k=3`) и `n` (`n=5`). Затем указателям `pk` и `pn` присваиваются значения адресов переменных `k` и `n`. При вызове функции — `swap2(pk, pn)`; в нее передаются копии значений указателей `pk`, `pn` (копии адресов переменных `k`, `n`). Вызванная функция `swap2`, используя переданные копии адресов и операцию разыменования, находит в области памяти функции `main` соответствующие значения `*pk`, `*pn` и в области памяти функции `main` меняет местами эти значения. Затем новые значения `*pk` и `*pn` выводятся на экран. После этого функция `swap2` вызывается снова с параметрами — адресами переменных `k` и `n` и меняет местами их значения.

11.7. Локальные и глобальные переменные

Функции имеют одинаковую структуру — заголовок функции, описание переменных функции, операторы функции. Поэтому возникает вопрос об области действия переменных, описанных в функции.

Под *областью действия* переменной понимают область программы, в которой переменная доступна для использования. Если переменная объявлена внутри функции или блока программы, то такая переменная называется локальной. Она доступна или видима только в пределах этой функции или блока. Под блоком понимается часть кода, заключенная в фигурные скобки.

Рассмотрим пример использования локальных переменных.

Пример. Ввести с клавиатуры a , b и вычислить

$$Z = \max(a, b) * \max(10 * a - 4, b + 5),$$

где $\max(x, y)$ — функция определения максимума для двух значений типа `double`.

```
#include <stdio.h>
double max(double x, double y)
    //заголовок функции определения максимума из двух значений
{double m;                //описание локальной переменной функции max
  if (x>=y)                //если x>=y
  m=x;                    //присвоить m значение формального параметра x
  else m=y;               //иначе присвоить m значение формального параметра y
  return m;               //возврат вычислен. значения m в вызывающую функцию
}
int main() {
  double a,b,Z;           //описание локальных переменных функции main
  printf("Введите a,b\n");
  scanf("%lf%lf",&a,&b);
  Z=max(a,b)*max(10*a-4,b+5); //вычисление значения переменной Z
  printf("Z=%lf\n",Z);
  getchar();
  return 0;
}
```

Пояснение. В этой программе описаны локальные переменные:

m — локальная переменная функция `max`, которая определена и видима только внутри этой функции, в функции `main` переменная m неопределенна и ее нельзя использовать;

a, b, Z — локальные переменные функции `main`, определены и используются внутри функции `main`, в функции `max` эти переменные неопределенны и в функции `max` их нельзя использовать.

Все переменные, описанные в функции, являются локальными переменными. Локальные переменные доступны только своей функции (для других функций и функции `main` они невидимы) и существуют, только пока выполняется эта функция (память под них выделяется при входе в функцию, а при выходе из функции доступ к ним освобождается).

Переменная, объявленная вне любой из функции, называется глобальной переменной и доступна в области от точки ее объявления до конца программы. Если только имя локальной переменной какой-ли-

бо функции не совпадает с именем глобальной функции, тогда в этой функции доступна только локальная переменная.

Глобальные переменные существуют на протяжении всей работы программы. Поэтому глобальные переменные могут использоваться для обмена данными между функциями. Но при этом надо помнить о том, что любое некорректное действие с глобальной переменной в одной из функций приведет к ошибкам не только в самой функции, но и в других функциях, где используется эта переменная. В связи с этим с точки зрения надежности предпочтительнее обмен данными между функциями через фактические и формальные параметры.

Пример использования глобальных переменных

```
#include <stdio.h>
int var1=11;                                //глобальная переменная var1
void func(void)
{
    printf("Внутри func() значение var1=%d\n",var1);
    //на экран будет выведено значение var1, равное 11
    var1=5*var1;
    //глобальная переменная var1 получает значение, равное 55
}
int main()
{
    int var2=22;
    //локальная переменная var2, доступна только внутри main()
    printf("Внутри main() значение var2=%d\n",var2);
    //на экран будет выведено значение var2, равное 22
    printf("Внутри main() до вызова функции func значение
    var1=%d\n",var1);
    //на экран будет выведено значение var1, равное 11
    func();
    printf("Внутри main() после вызова функции func
    значение var1=%d\n",var1);
    //на экран будет выведено значение var1, равное 55
    return 0;
}
```

В случае совпадения имени глобальной и локальной переменных внутри функции все действия происходят с локальной переменной. В этом случае изменения локальной переменной никак не отразятся на глобальной переменной. Но во избежание ошибок в логике работы программы лучше не использовать одни и те же имена для локальных и глобальных объектов.


```

for(i=0;i<n;i++)                                //в цикле
printf("%d ",x[i]);                             //вывод на экран элемента массива
printf("\n");
}
int main() {
int A[15], B[10];
srand((unsigned int)time(NULL)); //запуск датчика случайных
                                //чисел
create_massiv(A,15);                /*вызов функции create_massiv
                                для создания массива A из 15 элементов*/
printf("Массив A\n");
print_massiv(A,15); /*вызов функции print_massiv для вывода
                                на экран массива A из 15 элементов*/
create_massiv(B,10);                /*вызов функции create_massiv
                                для создания массива B из 10 элементов*/
printf("Массив B\n");
print_massiv(B,10); /*вызов функции print_massiv для вывода
                                на экран массива B из 10 элементов*/

getchar();
return 0;
}

```

Задача 2. С помощью датчика случайных чисел сформировать массивы $A\{3\}$, $B\{9\}$, $C\{5\}$. Вывести их на экран. Вычислить и вывести на экран значение $P = \frac{\max(A) - \max(B) + \max(C)}{\min(A) - \min(C)}$, где $\max(A)$, $\max(B)$,

$\max(C)$ — максимальные элементы массивов A , B , C соответственно; $\min(A)$, $\min(C)$ — минимальные элементы массивов A и C соответственно. Программу составлять с использованием прототипов.

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void create_massiv(int [], int); /*прототип функции создания
                                массива с помощью датчика случайных чисел*/
void print_massiv(int [], int);
                                //прототип функции вывода массива на экран
int max(int [], int);
                                //прототип функции определения максимального элемента массива
int min(int [], int);
                                //прототип функции определения минимального элемента массива
int main() {
int A[3], B[9], C[5];
double P;

```



```

srand((unsigned int)time(NULL));
create_massiv(A,3); /*вызов функции create_massiv для создания
                    массива A из 3 элементов*/
printf("Массив A\n");
print_massiv(A,3); /*вызов функции print_massiv для вывода
                   на экран массива A из 3 элементов*/
create_massiv(B,9); /*вызов функции create_massiv для создания
                    массива B из 9 элементов*/
printf("Массив B\n");
print_massiv(B,9); /*вызов функции print_massiv для вывода
                   на экран массива B из 9 элементов*/
create_massiv(C,5); /*вызов функции create_massiv для создания
                    массива C из 5 элементов*/
printf("Массив C\n");
print_massiv(C,5); /*вызов функции print_massiv для вывода
                   на экран массива C из 5 элементов*/
P=double(max(A,3)-max(B,9)+max(C,5))/(min(A,3)-min(C,5)
);
//вычисление значения P
printf("P=%lf\n",P);
getchar();
return 0;
}
void create_massiv(int x[], int n) /*заголовок функции
создания массива с помощью датчика случайных чисел*/
{int i;
for (i=0; i<n; i++)
x[i]=rand()%101-50;
}
void print_massiv(int x[], int n) /*заголовок функции вывода
массива на экран*/
{ int i;
for(i=0;i<n;i++)
printf("%d ",x[i]);
printf("\n");
}
int max(int x[], int n)
//заголовок функции определения максимального элемента массива
{ int m,i;
m=x[0]; //присвоение предпол. макс. (m) начального значения (x[0])
for (i=0; i<n; i++) //в цикле
if (x[i]>m) //если x[i] больше предполагаемого максимума
m=x[i]; //меняем предполагаемый максимум
return m; //возврат вычисленного значения m в вызывающую функцию
}

```

```

int min(int x[], int n)
    //заголовок функции определения минимального элемента массива
{ int m,i;
  m=x[0];    //присвоение предпол. мин. (m) начального значения(x[0])
  for (i=0; i<n; i++)    //в цикле
    if (x[i]<m)    //если x[i] меньше предполагаемого минимума
      m=x[i];    //меняем предполагаемый минимум
  return m; //возврат вычисленного значения m в вызывающую функцию
}

```

Задача 4. С помощью датчика случайных чисел сформировать динамические массивы A , B , C . Количество элементов каждого массива ввести с клавиатуры. Вывести полученные массивы на экран. Вычислить и вывести на экран значение $P = \frac{\max(A) - \max(B) + \max(C)}{\min(A) - \min(C)}$, где

$\max(A)$, $\max(B)$, $\max(C)$ — максимальные элементы массивов A , B , C соответственно; $\min(A)$, $\min(C)$ — минимальные элементы массивов A и C соответственно.

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void create_massiv(int *X, int &n)
    /*заголовок функции создания динамического массива,
    число элементов которого вводится с клавиатуры*/
{ int i;
  printf("Введите число элементов массива ");
  scanf("%d",&n);
  X=new int[n];
    //выделение памяти под целочисленный массив из n элементов
  for (i=0; i<n; i++)
    X[i]=rand()%101-50;
}
void print_massiv(int *X, int n)
    //заголовок функции вывода на экран динамического массива
{ int i;
  for (i=0; i<n; i++)
    printf("%d ",X[i]);
  printf("\n");
}
int max(int *X, int n)
    //заголовок функции определения максимального элемента массива
{int m,i;
  m=X[0]; //присвоение предпол. максимуму (m) начального значения(X[0])

```

```

    for (i=0; i<n; i++)                                //в цикле
    if (X[i]>m)                                           //если X[i] больше предполагаемого максимума
    m=X[i];                                             //меняем предполагаемый максимум
    return m;    //возврат вычисленного значения m в вызывающую функцию
}
int min(int *X, int n)
    //заголовок функции определения минимального элемента массива
{int m,i;
  m=X[0]; //присвоение предпол. минимуму (m) начального значения (X[0])
  for (i=0; i<n; i++)                                //в цикле
  if (X[i]<m)                                           //если X[i] меньше предполагаемого минимума
  m=X[i];                                             //меняем предполагаемый минимум
  return m;    //возврат вычисленного значения m в вызывающую функцию
}
int main() {
int *A=NULL, *B=NULL, *C=NULL;
int nA=0, nB=0, nC=0;
double P;
srand(time(NULL));
create_massiv(A,nA);    /*вызов функции create_massiv
                        для создания динамического массива A из nA элементов*/
printf("Массив A\n");
print_massiv(A,nA);    /*вызов функции print_massiv для вывода
                        на экран динамического массива A из nA элементов*/
create_massiv(B,nB);    /*вызов функции create_massiv
                        для создания динамического массива B из nB элементов*/
printf("Массив B\n");
print_massiv(B,nB);    /*вызов функции print_massiv для вывода
                        на экран динамического массива B из nB элементов*/
create_massiv(C,nC);    /*вызов функции create_massiv
                        для создания динамического массива C из nC элементов*/
printf("Массив C\n");
print_massiv(C,nC);    /*вызов функции print_massiv для вывода
                        на экран динамического массива C из nC элементов*/
P=double (max (A,nA) -max (B,nB) +max (C,nC) ) /
    (min (A,nA) -min (C,nC) ) ;
printf("P=%lf\n",P);
getchar();
return 0;
}

```

Пояснение. В этой программе для того, чтобы динамически создаваемыми массивами А, В, С могли пользоваться все функции про-

граммы, в функцию `create_massiv` необходимо передать адреса указателей `A`, `B` и `C`, которые инициализированы пустыми значениями и находятся в области памяти функции `main`. В функции `create_massiv` при выделении памяти под массив, в соответствующий указатель, расположенный в функции `main`, записывается значение адреса выделенной области. Затем в цикле с помощью датчика случайных чисел формируются элементы динамического массива, которые становятся доступны и функции `main` и всем остальным функциям программы. Таким образом создаются и заполняются массивы `A`, `B` и `C`, используемые в задаче.

Для функций `print_massiv`, `max` и `min` в эти функции передается копия значения соответствующих указателей — `*A` — для массива `A`, `*B` — для массива `B`, `*C` — для массива `C`, т. е. копии адресов динамических областей памяти, выделенных под массивы `A`, `B` и `C`. Сами указатели остаются в функции `main`, и с ними никаких действий не производится.

Остальная работа функций программы производится так, как это было описано ранее в примере 1.

11.9. Использование двумерных массивов в качестве параметров

При использовании двумерных массивов (матриц) в качестве параметров для любой функции в эту функцию передается указатель на матрицу, т. е. адрес этой матрицы (адрес элемента матрицы, стоящего на пересечении 0-й строки и 0-го столбца). Сама матрица остается в вызывающей функции (например, `main`), и все изменения элементов матрицы происходят в области памяти вызывающей функции.

Чтобы функция получила правильный доступ к элементам матрицы и для согласования типов формальных и фактических параметров при задании статической матрицы в качестве параметра необходимо указывать число столбцов матрицы. *Подробно распределение памяти под матрицу и организация доступа к элементам матрицы были рассмотрены в главе 10 «Двумерные массивы».* Так как в функцию передается только адрес матрицы (одномерного массива из указателей на одномерные массивы), можно при задании параметра матрицы не указывать число строк матрицы, а передавать его в качестве второго

параметра функции, а число столбцов — в качестве третьего параметра, например, заголовок функции создания матрицы может иметь вид:

```
void create_matrix(int x[][10], int n, int m),
```

где указано три формальных параметра:

- 1-й формальный параметр — указатель на матрицу X из 10 столбцов;
- 2-й формальный параметр — число столбцов матрицы n типа `int`;
- 3-й формальный параметр — число строк матрицы m типа `int`.

Заголовок функции создания динамической матрицы может иметь такой прототип:

```
void create_matrix(int **&matrix, int &n, int &m);
```

В этой функции число строк и столбцов вводится с клавиатуры и нет ограничений на размеры обрабатываемой матрицы, что удобно при работе с динамическими матрицами.

Если в задаче используется несколько статических матриц с различным количеством строк и столбцов, то для матрицы-параметра необходимо указать максимальное количество столбцов для всех заданных матриц. А затем в функции `main` для согласования формальных и фактических параметров по типам необходимо объявить все используемые матрицы с одинаковым числом столбцов, равных максимальному.

Задача 1. С помощью датчика случайных чисел сформировать матрицы $A[6][8]$ и $B[5][9]$. Вывести их на экран.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void create_matrix(int x[][9], int n, int m)
    /*заголовок функции создания матрицы
    с 9 столбцами с помощью датч. случайных чисел*/
{
    int i, j;
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            x[i][j]=rand()%101-50;
}
void print_matrix(int x[][9], int n, int m)
    /*заголовок функции вывода матрицы с 9 столбцами на экран*/
```



```

{ int i, j;
  for(i=0; i<n; i++){
    for (j=0; j<m; j++)
      printf("4%d ", x[i][j]);
    printf("\n");
  }
}

int main() {
int A[6][9], B[5][9];
srand((unsigned int)time(NULL));
create_matrix(A, 6, 8);          /*вызов функции create_matrix
                                для создания матрицы A из 6 строк и 8 столбцов*/
printf("Матрица A\n");
print_matrix(A, 6, 8);          /*вызов функции print_matrix для вывода
                                на экран матрицы A из 6 строк и 8 столбцов*/
create_matrix(B, 5, 9);          /*вызов функции create_matrix
                                для создания матрицы B из 5 строк и 9 столбцов*/
printf("Матрица B\n");
print_matrix(B, 5, 9);          /*вызов функции print_matrix для вывода
                                на экран матрицы B из 5 строк и 9 столбцов*/

getchar();
return 0;
}

```

Задача 2. С помощью датчика случайных чисел сформировать матрицы $Q[5][7]$ и $R[5][8]$. Вывести их на экран. Сформировать одномерные массивы из:

- произведений положительных элементов каждой строки матриц;
- произведений отрицательных элементов каждой строки матриц;
- минимальных элементов каждого столбца матриц.

Все полученные массивы вывести на экран. Определить суммы положительных и отрицательных элементов матриц.

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void create_matrix(int x[][8], int n, int m)
                                /*заголовок функции создания матрицы с 8 столбцами
                                с помощью датчика случайных чисел*/

{ int i, j;
  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
      x[i][j]=rand()%101-50;
}

```



```

void print_matrix(int x[][8], int n, int m)      /*заголовок
                                                функции вывода матрицы с 8 столбцами на экран*/
{ int i, j;
  for(i=0; i<n; i++){
    for (j=0; j<m; j++)
      printf("4%d ", x[i][j]);
    printf("\n");
  }
}

void pr_pol(int X[][8], double Y[], int n, int m)
  /*заголовок функции формирования одномерного массива из произведений
    положительных элементов каждой строки матрицы*/
{ int i, j;
  double p;
  for (i=0; i<n; i++){                                //в цикле по строкам
    p=1;          /*задание начального значения произведения положительных
                                                           элементов i-й строки (p=1) */
    for (j=0; j<m; j++)                                //в цикле по столбцам
      if (X[i][j]>0)                                     //если элемент матрицы положителен
        p=p*X[i][j];    //умножаем старое значение p на элемент матрицы
    Y[i]=p;        //присвоим значение произведения i-у элементу массива Y
  }
}

void pr_otr(int X[][8], double Y[], int n, int m)
  /*заголовок функции формирования одномерного массива из произведений
    отрицательных элементов каждой строки матрицы*/
{ int i, j;
  double p;
  for (i=0; i<n; i++){                                //в цикле по строкам
    p=1;          /*задание начального значения произведения отрицательных
                                                           элементов i-й строки (p=1) */
    for (j=0; j<m; j++)                                //в цикле по столбцам
      if (X[i][j]<0)                                     //если элемент матрицы отрицателен
        p=p*X[i][j];    //умножаем старое значение p на элемент матрицы
    Y[i]=p;        //присвоим значение произведения i-у элементу массива Y
  }
}

void print_massiv1(double Y[], int n)      /*заголовок функции
    вывода на экран одномерного массива с элементами типа double*/
{ int i;
  for (i=0; i<n; i++)
    printf("%lf ", Y[i]);
  printf("\n");
}

```

```
void min_column(int X[][8], int Z[], int n, int m)
    /*заголовок функции формирования одномерного массива
    из минимальных элементов каждого столбца матрицы*/
{ int i,j,min;
  for (j=0; j<m; j++){ //в цикле по столбцам
    min=X[0][j]; //присвоение нач. значен. предполагаемому минимуму
    for (i=0; i<n; i++) //в цикле по строкам
      if (X[i][j]<min) //если элемент матрицы меньше предполаг.
        min=X[i][j]; //минимума, меняем предполагаемый минимум
      Z[j]=min; //присвоим значение минимума j-у элементу массива Z
    }
  }
void print_massiv2(int Z[], int m) /*заголовок функции вывода
на экран массива с элементами типа int*/
{ int j;
  for (j=0; j<m; j++)
    printf("%d ",Z[j]);
  printf("\n");
}
double sum_pol(int X[][8], int n, int m) /*заголовок функции
вычисления суммы положительных элементов матрицы*/
{ int i, j;
  double s;
  s=0; //задание начального значения сумме полож. элементов матрицы
  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
      if (X[i][j]>0) //если элемент матрицы положителен
        s=s+X[i][j]; //увеличим старое значение s на элемент матрицы
  return s; //возврат вычисленного значения s в вызывающую функцию
}
double sum_otr(int X[][8], int n, int m) /*заголовок функции
вычисления суммы отрицательных элементов матрицы*/
{ int i, j;
  double s;
  s=0; //задание начального значения сумме отриц.элементов матрицы
  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
      if (X[i][j]<0) //если элемент матрицы отрицателен
        s=s+X[i][j]; //увеличим старое значение s на элемент матрицы
  return s; //возврат вычисленного значения s в вызывающую функцию
}
```

```

int main() {
    int Q[5][8], R[5][8], Q3[7], R3[8];
    double Q1[5], Q2[5], R1[5], R2[5], sum_p_Q, sum_o_Q,
    sum_p_R, sum_o_R;
    srand((unsigned int)time(NULL));
    create_matrix(Q, 5, 7);          /*вызов функции create_matrix
                                   для создания матрицы Q из 5 строк и 7 столбцов*/
    printf("Матрица Q\n");
    print_matrix(Q, 5, 7);          /*вызов функции print_matrix для вывода
                                   на экран матрицы Q из 5 строк и 7 столбцов*/
    pr_pol(Q, Q1, 5, 7);           /*вызов функции pr_pol для формирования
                                   массива Q1 из произведения положительных элементов матрицы Q*/
    printf("Массив Q1\n");
    print_massiv1(Q1, 5);
    //вызов функции print_massiv1 для вывода на экран массива Q1
    pr_otr(Q, Q2, 5, 7);           /*вызов функции pr_otr для формирования
                                   массива Q2 из произведения отрицательных элементов матрицы Q*/
    printf("Массив Q2\n");
    print_massiv1(Q2, 5);
    //вызов функции print_massiv1 для вывода на экран массива Q2
    min_column(Q, Q3, 5, 7);       /*вызов функции min_column
                                   для формирования массива Q3 из минимальных элементов каждого
                                   столбца матрицы Q*/
    printf("Массив Q3\n");
    print_massiv2(Q3, 7);
    //вызов функции print_massiv2 для вывода на экран массива Q3
    sum_p_Q = sum_pol(Q, 5, 7);     /*вызов функции sum_pol
                                   для вычисления суммы положительных элементов матрицы Q*/
    printf("Сумма положительных элементов матрицы Q=%lf\n",
    sum_p_Q);
    sum_o_Q = sum_otr(Q, 5, 7);     /*вызов функции sum_otr
                                   для вычисления суммы отрицательных элементов матрицы Q*/
    printf("Сумма отрицательных элементов матрицы Q=%lf\n",
    sum_o_Q);
    create_matrix(R, 5, 8);         /*вызов функции create_matrix
                                   для создания матрицы R из 5 строк и 8 столбцов*/
    printf("Матрица R\n");
    print_matrix(R, 5, 8);          /*вызов функции print_matrix для вывода
                                   на экран матрицы R из 5 строк и 8 столбцов*/
    pr_pol(R, R1, 5, 8);           /*вызов функции pr_pol для формирования
                                   массива R1 из произведения положительных элементов матрицы R*/
    printf("Массив R1\n");
    print_massiv1(R1, 5);
    //вызов функции print_massiv1 для вывода на экран массива R1

```



```

pr_otr(R,R2,5,8);          /*вызов функции pr_otr для формирования
                             массива R2 из произведения отрицательных элементов матрицы R*/
printf("Массив R2\n");
print_massiv1(R2,5);
    //вызов функции print_massiv1 для вывода на экран массива R2
min_column(R,R3,5,8);      /*вызов функции min_column
                             для формирования массива R3 из минимальных элементов каждого столбца
                             матрицы R*/

printf("Массив R3\n");
print_massiv2(R3,8);
    //вызов функции print_massiv2 для вывода на экран массива R3
sum_p_R=sum_pol(R,5,8);    /*вызов функции sum_pol
                             для вычисления суммы положительных элементов матрицы R*/
printf("Сумма положительных элементов матрицы R=%lf\n",
sum_p_R);
sum_o_R=sum_otr(R,5,8);    /*вызов функции sum_otr
                             для вычисления суммы отрицательных элементов матрицы R*/
printf("Сумма отрицательных элементов матрицы Q=%lf\n",
sum_o_Q);
getchar();
return 0;
}

```

Задача 3. С помощью датчика случайных чисел сформировать динамические целочисленные матрицы A и B , число строк и столбцов которых вводится с клавиатуры. Вывести их на экран.

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void create_matrix(int **&X, int &n, int &m) /*заголовок
        функции создания динамической матрицы, число строк и столбцов которой
        вводится с клавиатуры*/
{
    int i,j;
    printf("Введите через пробел число строк и столбцов
        матрицы\n");
    scanf("%d%d",&n,&m);
    X=new int*[n]; //выделение памяти под массив указателей на строки
    for (i=0; i<n; i++)
        X[i]=new int[m]; //выделение памяти под i-ю строку матрицы
    for (i=0; i<n; i++) //в цикле по строкам
        for (j=0; j<m; j++) //в цикле по столбцам
            X[i][j]=rand()%101-50; //формируем элемент матрицы
}

```

```

void print_matrix(int **X, int n, int m) /*заголовок функции
                                          вывода на экран динамической матрицы*/

{ int i, j;
  for (i=0; i<n; i++){
    for (j=0; j<m; j++)
      printf("%4d ",X[i][j]);
    printf("\n");
  }
}

int main() {
  int **A=NULL, **B=NULL; /*описание и инициализация пустыми
                              значениями указателей на матрицы A и B*/
  int nA=0, mA=0, nB=0, mB=0; /*описание и инициализация числа
                              строк и столбцов матриц*/

  srand((unsigned int)time(NULL));
  create_matrix(A,nA,mA); /*вызов функции create_matrix
                              для создания с помощью датчика случайных чисел
                              матрицы A из nA строк и mA столбцов*/

  printf("Матрица A\n");
  print_matrix(A,nA,mA); /*вызов функции print_matrix для вывода
                              на экран матрицы A из nA строк и mA столбцов*/
  create_matrix(B,nB,mB); /*вызов функции create_matrix
                              для создания с помощью датчика случайных чисел
                              матрицы B из nB строк и mB столбцов*/

  printf("Матрица B\n");
  print_matrix(B,nB,mB); /*вызов функции print_matrix для вывода
                              на экран матрицы B из nB строк и mB столбцов*/

  getchar();
  return 0;
}

```

Пояснение. В этой программе для того, чтобы динамически создаваемыми матрицами A, B могли пользоваться все функции программы, в функцию `create_matrix` необходимо передать адреса указателей A, B, которые инициализированы пустыми значениями и находятся в области памяти функции `main`. В функции `create_matrix` при выделении памяти под массив указателей на строки матрицы в соответствующий указатель, расположенный в функции `main`, записывается значение адреса выделенной области. Затем в цикле выделяется память под каждую из строк матрицы. После этого в двойном цикле по строкам и столбцам с помощью датчика случайных чисел формируются элементы динамической матрицы, которые становятся доступны и функции `main` и всем остальным функциям программы.

Таким образом создаются и заполняются массивы *A*, *B* и *C*, используемые в задаче.

Для функции `print_matrix` в эти функции передается копия значения соответствующих указателей: `**A` — для матрицы *A*, `**B` — для матрицы *B*, т. е. копии адресов динамических областей памяти, выделенных под матрицы *A*, *B*. Сами указатели остаются в функции `main`, и с ними никаких действий не производится.

Остальная работа функций программы производится так, как это было описано в предыдущих примерах.

Тесты

1. В чем разница между формальными и фактическими параметрами?

- а) никакой разницы нет;
- б) формальные параметры используются при описании подпрограммы, а фактические — при вызове подпрограммы;
- в) фактические параметры используются при описании подпрограммы, а формальные — при вызове подпрограммы.

2. В чем разница между глобальными и локальными переменными?

- а) разницы нет;
- б) глобальные переменные не могут использоваться в подпрограммах, для этого служат локальные переменные;
- в) глобальные переменные могут использоваться во всех подпрограммах и в функции `main()`, а локальные переменные только в своей подпрограмме.

3. Выберите определение функции, не содержащее ошибок:

- а)

```
int Area(int A, int B)
{ float S;
  Area=A*B;
}
```
- б)

```
int Area(int A, int B)
{ int S;
  S=A*B;
  return S;
}
```
- в)

```
int Area(int A, int B)
{ int S;
  S=A*B;
  return Area;
}
```


4. Выберите определение функции, не содержащее ошибок:

- a)

```
int Area(int A, int B)
{
    int S;
    S = A * B;
    return S;
}
```
- б)

```
Area(int A, int B)
{
    return A*B;
}
```
- в) оба варианта правильные.

5. Что будет выведено на экран в результате работы следующей программы?

```
int A;
void Prim(int A)
{
    A=5;
    printf(" %d",A);
}
int main()
{
    int A=10;
    Prim(A);
    printf(" %d", A);
    return 0;
}
```

- a) 10 10;
- б) 5 10;
- в) 5 5.

6. Какие переменные в следующем фрагменте программы глобальные, а какие локальные?

```
#include <stdio.h>
int A;
float B;
float Summa(float Y, float C, float D)
{
    int I,J;
    ...
}
```

- a) локальные: Y, C, D, I, J; глобальные: A, B;
- б) локальные: I, J; глобальные: A, B;
- в) локальные: A, B, I, J; глобальные: Y, C, D.

7. Какие переменные являются формальными, а какие фактическими параметрами в следующем фрагменте программы?

```
void Primer (int A, float B)
{...}
int main()
{ ...
Primer (C,D);
}
```

- а) формальные параметры: A, B; фактические: C, D;
- б) формальные параметры: C, D; фактические: A, B;
- в) формальные параметры: A, C; фактические: B, D.

8. Выберите правильные варианты вызова функции `summa` из функции `main()`:

```
...
float summa (float *x, int n)
{float s=0; int i;
  for (i=0; i<n; i++) s=s+x[i];
  return s;
}
int main()
{float a[10], S;
  ...
1.   S=summa(a, 10);
2.   S=summa(&a[2], 3);
3.   S=summa(a[2], 10);
  ...
}
```

- а) 1 и 2;
- б) 2 и 3;
- в) 1 и 3.

9. Выберите правильный вариант описания прототипа функции:

- а) `int Atr(int j, float a[], int n);`
- б) `int Atr(int j, a[20], int n);`
- в) оба варианта правильные.

10. Выберите пример правильного описания прототипа функции:

- а) `int summa(float a, float b, int c);`
- б) `int summa(float, float, int);`
- в) оба варианта правильные.

11 . Как можно передать одномерный массив в качестве параметра функции?

- а) `void print(int Mas[], int k);`
- б) `void print(int *pm, int k);`
- в) оба варианта верные.

12. Как можно передать двумерный массив размера 3 на 5 в качестве параметра функции?

- а) `void print(int Mas[3][5], int m, int n);`
- б) `void print(int **pm, int m, int n);`
- в) оба варианта верные.

13. Может ли функция иметь следующий прототип?

```
...  
info(float, int a, char*); ...
```

- а) да;
- б) нет.

14. Можно ли в программе объявить следующий прототип функции?

```
...  
int function(int*, int **, int ***); ...
```

- а) да;
- б) нет.

15. Должны ли имена параметров, указанных в прототипе, определении и вызове функции, соответствовать друг другу?

- а) да;
- б) нет;
- в) должны соответствовать в прототипе и определении.

Задания

Для каждой задачи составить схему алгоритма и написать программу.

1. Написать программу с функцией-заставкой к курсовой работе в виде:

```
*****  
*          Курсовая работа          *  
*    по программированию            *  
*    Автор: Иванов И. И.            *  
*    Дата: 26.03.2012 г.            *  
*****
```


2. Ввести с клавиатуры три массива разной длины. Вывести одновременно на экран все массивы, каждый массив выводится с соответствующей надписью «Массив ...(имя массива) из ...(кол-во) элементов». При составлении программы использовать процедуры.

3. Сформировать массивы $A\{15\}$ и $B\{7\}$. Вывести их. Используя функцию, найти максимальные элементы массивов.

4. Используя подпрограммы, сформировать и вывести на экран массивы $A\{15\}$ и $B\{14\}$. Используя функции, найти максимальный и минимальный элементы в каждом массиве.

5. Сформировать три одномерных массива из элементов арифметических прогрессий. 1-е элементы прогрессий и количества элементов в каждой прогрессии (и в соответствующем массиве) вводятся с клавиатуры, разность каждой прогрессии формируется случайным образом. Вывести массивы на экран. Вычислить в каждом массиве сумму элементов, значения которых меньше 10.

6. Сформировать одномерные массивы из элементов арифметических прогрессий. 1-е элементы прогрессий и количества элементов в каждой прогрессии (и в соответствующем массиве) вводятся с клавиатуры, разность каждой прогрессии формируется случайным образом. Вывести массивы на экран. Вычислить в каждом массиве произведение всех элементов.

7. Сформировать два одномерных массива из элементов геометрических прогрессий. 1-е элементы прогрессий и количества элементов в каждой прогрессии (и в соответствующем массиве) вводятся с клавиатуры. Знаменатель каждой прогрессии формируется случайным образом. Вывести массивы на экран. Вычислить в каждом массиве количество отрицательных элементов.

8. Сформировать одномерные массивы из элементов геометрических прогрессий. 1-е элементы прогрессий и количества элементов в каждой прогрессии (и в соответствующем массиве) вводятся с клавиатуры. Знаменатель каждой прогрессии формируется случайным образом. Вывести массивы на экран. Определить максимальные и минимальные элементы для каждого массива.

9. Дан массив A из восьми элементов и целое число M (вводится с клавиатуры или формируется с помощью генератора случайных чисел). Из элементов массива A , не превышающих числа M , сформировать массив B . Вывести оба массива на экран. Для обоих массивов найти сумму ненулевых элементов (отдельно для каждого массива). Использовать функции.

10. Дан массив C из 11 элементов. Из положительных элементов массива C сформировать массив A . Вывести оба массива на экран. Для обоих массивов найти произведение ненулевых элементов (отдельно для каждого массива). Использовать функции.

11. Сформировать массивы $K\{4\}$, $M\{7\}$, $C\{12\}$. Вывести их на экран. Вычислить и вывести на экран значение $N = \frac{\max(K) \cdot \max(M)}{\min(K) \cdot \min(C)}$, где

$\max(K)$, $\max(M)$ — максимальные элементы массивов K , M соответственно; $\min(K)$, $\min(C)$ — минимальные элементы массивов K и C соответственно. Использовать подпрограммы.

12. Сформировать массивы $X\{20\}$, $Y\{8\}$, $Z\{14\}$. Вывести их на экран. Вычислить и вывести на экран значение $R = \frac{P(X) \cdot P(Y)}{S(X) - S(Z)}$, где $P(X)$,

$P(Y)$ — произведения всех элементов массивов X и Y соответственно; $S(X)$, $S(Z)$ — суммы положительных элементов массивов X и Z соответственно. Использовать подпрограммы.

13. Ввести массив M из девяти элементов. Вычислить разность между минимальным и максимальным элементами. Использовать подпрограммы.

14. Сформировать массив $A\{20\}$. Вычислить произведение элементов, расположенных между минимальным и максимальным элементами массива. Вывести массив A .

15. Ввести массив K из семи элементов. Вычислить сумму квадратов элементов, стоящих после максимального элемента.

16. Ввести с клавиатуры одномерные массивы $M(8)$, $F(6)$, $W(10)$. Найти и вывести на экран минимальные элементы этих массивов. Использовать функции.

17. С помощью генератора случайных чисел сформировать одномерные массивы $M(8)$, $F(6)$, $W(10)$. Найти и вывести на экран максимальное из произведений элементов этих массивов. Использовать функции.

18. Сформировать массив вещественных чисел $A(30)$. Сформировать массив B таким образом: $b[1] = a[1] + a[30]$, $b[2] = a[2] + a[29]$, $b[3] = a[3] + a[28]$, Сформировать массив C таким образом: $c[1] = a[1] * a[16]$, $c[2] = c[2] * c[17]$, $c[3] = c[3] * c[18]$, Найти максимальный элемент в массиве B и минимальный элемент в массиве C .

19. Вычислить сумму $1! + 2! + 3! + \dots + N!$, используя функцию вычисления факториала числа $k!$.

20. Составить программу поиска большего из четырех чисел с использованием подпрограммы поиска большего из двух.

21. Сформировать массивы $W\{4\}$, $X\{8\}$, $Y\{5\}$, $Z\{10\}$. Вывести их на экран. Вычислить и вывести на экран значение $S = \frac{P(W) - P(Y)}{S(X)S(Z)}$, где $P(W)$,

$P(Y)$ — произведения всех элементов массивов W и Y соответственно; $S(X)$, $S(Z)$ — суммы элементов массивов X и Z соответственно. Использовать подпрограммы.

22. Сформировать массив P из 20 элементов. Вычислить значение выражения $S = (\max - \min)/Sp$, где \min — минимальный элемент массива P , \max — максимальный элемент массива P , Sp — сумма положительных элементов массива P .

23. Сформировать массив B из 18 элементов. Вычислить значение выражения $K = (\max + \min)/kn$, где \min — минимальный элемент массива B , \max — максимальный элемент массива B , kn — количество ненулевых элементов массива B .

24. Сформировать массивы $P\{13\}$, $B\{9\}$. Вывести их на экран. Вычислить и вывести на экран значение $Z = \frac{\max(P) - \min(P)}{\max(B) - \min(B)}$, где $\max(P)$, $\max(B)$ — максимальные элементы массивов P , B соответственно; $\min(P)$, $\min(B)$ — минимальные элементы массивов P и B соответственно. Использовать подпрограммы.

25. Сформировать массивы $A\{3\}$, $B\{7\}$, $C\{9\}$. Вывести их на экран. Вычислить и вывести на экран значение $S = \frac{S(A)}{S(B) \cdot S(C)}$, где $S(A)$, $S(B)$, $S(C)$ — суммы ненулевых элементов массивов A , B и C соответственно. Использовать подпрограммы.

26. Сформировать массивы $A\{5\}$, $B\{10\}$, $C\{7\}$. Вывести их на экран. Вычислить и вывести на экран значение $P = \frac{P(A)}{P(B) \cdot P(C)}$, где $P(A)$, $P(B)$, $P(C)$ — произведения положительных элементов массивов A , B , C соответственно. Использовать подпрограммы.

27. Сформировать массивы $X\{10\}$, $Y\{20\}$, $Z\{30\}$. Вывести их на экран. Вычислить и вывести на экран значение $T = \frac{P1(X) + P1(Z)}{P2(Y) \cdot P2(Z)}$, где $P1(X)$, $P1(Z)$ — произведения положительных элементов массивов X и Z соответственно; $P2(Y)$, $P2(Z)$ — произведения отрицательных элементов массивов Y и Z соответственно. Использовать подпрограммы.

28. Сформировать массивы $A\{10\}$, $B\{20\}$, $C\{30\}$. Вывести их на экран. Вычислить и вывести на экран значение $H = \frac{S(A) + S(B) + S(C)}{P(A) \cdot P(B) \cdot P(C)}$, где

$S(A)$, $S(B)$, $S(C)$ — суммы положительных элементов массивов A , B и C соответственно; $P(A)$, $P(B)$, $P(C)$ — произведения отрицательных элементов массивов A , B и C соответственно. Использовать подпрограммы.

29. Сформировать массив M . Из ненулевых элементов массива M сформировать массив K . Вывести оба массива на экран. Для каждого массива найти среднее арифметическое всех элементов. Использовать функции.

30. Сформировать массив A . Из отрицательных элементов массива A сформировать массив P . Их положительных элементов массива A сформировать массив N . Вывести все массивы на экран. Для каждого массива найти максимальный и минимальный элементы. Использовать функции.

31. Дан массив $A\{20\}$. Сформировать массив B , каждый элемент которого есть среднее арифметическое элементов массива A , стоящих справа и слева от отрицательного элемента массива A , первый и последний элементы — неотрицательные. Использовать функции.

32. Вычислить $x = \frac{(a!)^2}{1 + b!}$, при этом $n!$ вычислить в виде функции.

33. Ввести массивы $A = \{a_i | i = 1, 2, \dots, 8\}$, $B = \{b_j | j = 1, 2, \dots, 7\}$. Вычислить элементы массива $C = (\max_{1 \leq i \leq 8} \{a_i\} + \min_{1 \leq i \leq 8} \{a_i\}, \max_{1 \leq i \leq 7} \{b_i\} + \min_{1 \leq i \leq 7} \{b_i\})$.

34. Ввести массивы $A\{8\}$ и $B\{8\}$. Вычислить массив $C = \{c_k | k = 1, 2, \dots, 8\}$:

$$c_i = \frac{\prod_{j=1}^8 (a_j + b_j)}{b_i}, \text{ где } \prod_{j=1}^8 (a_j + b_j) = (a_1 + b_1) \cdot (a_2 + b_2) \cdot \dots \cdot (a_8 + b_8).$$

35. Вычислить $r = \sqrt{e^a + e^b}$. Оформить вычисление e^x как функцию по формуле $e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$ с точностью 10^{-5} .

36. Найти сумму 10 членов ряда, общий член которого $a_n = \frac{\ln(n!)}{(n!)^2}$, при этом $n!$ вычислить в виде функции.

37. Найти с точностью 10^{-3} сумму 10 членов ряда, общий член которого $a_n = \frac{(n!)^2}{1 + (n+1)!}$, при этом $n!$ вычислить в виде функции.

38. Вычислить $x = (\sin(a^2) + \sin(b^2) + \sin(c^2))$. Оформить вычисление $\sin(y^2)$ как функцию.

39. Вычислить $x = (\cos(a^3) + \cos(b^3) + \cos(c^3))$. Оформить вычисление $\cos(y^3)$ как функцию.

40. С помощью генератора случайных чисел задать N чисел (N вводится с клавиатуры). Для каждого числа вычислить $\sqrt[4]{}$ и $\sqrt[3]{}$. Вычисление $\sqrt[4]{}$ и $\sqrt[3]{}$ оформить в виде функций.

41. Вычислить $x = \ln(a^2) + \ln(b^2) + \ln(c^2)$, при этом вычисление $\ln(y^2)$ оформить в виде функции.

42. Вычислить $x = (\sin(a) * \sin(b) * \sin(c))^2 + (\sin(a) * \sin(b) * \sin(c))^3$. Оформить вычисление $(\sin(a) * \sin(b) * \sin(c))$ как функцию.

43. Вычислить $r = \sqrt{e^{2a} + e^{2b} + e^{2c}}$. Оформить вычисление e^x как функцию по формуле $e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$ с точностью 10^{-5} .

44. Вычислить $r = \sqrt{\cos(a) * \cos(b) * \cos(c)} + (\cos(a) * \cos(b) * \cos(c))^3$. Оформить вычисление $(\cos(a) * \cos(b) * \cos(c))$ как функцию.

45. Вычислить $x = \frac{\sqrt{abc} + (\sqrt{abc} + abc)^2}{(\sqrt{abc} + 1)^2}$. Вычисление abc оформить в виде функции.

46. Вычислить $x = (\text{tg}(a) * \text{tg}(b) * \text{tg}(c))^2 + (\text{tg}(a) * \text{tg}(b) * \text{tg}(c))^3$. Оформить вычисление $(\text{tg}(a) * \text{tg}(b) * \text{tg}(c))$ как функцию.

Контрольные вопросы

1. Для чего используются методы структурного программирования?
2. В чем заключается структурирование программы? Приведите примеры.
3. Что такое программирование «сверху вниз»? Приведите примеры.
4. Для чего используется технология нисходящего программирования? Объясните на примере.
5. Для чего в программировании используются подпрограммы?
6. Что такое подпрограмма (функция)? Приведите примеры.
7. Что такое возвращаемое значение? Приведите примеры.
8. Чем стандартные функции отличаются от функций, определенных пользователем? Приведите примеры стандартных функций.
9. Напишите структуру функции в общем виде и объясните, для чего используется каждый раздел функции. Приведите примеры определения функции.
10. Что такое «вызов функции»? Чем вызов функции с возвращаемым значением отличается от вызова функции, не имеющей возвращаемого значения?

11. Можно ли из функции вызвать другую функцию? Приведите примеры.
12. Можно ли внутри функции описать другую функцию? Приведите примеры.
13. Что такое формальные параметры?
14. Что такое фактические параметры?
15. Какие условия должны выполняться при указании фактических параметров в вызове подпрограммы? Приведите примеры.
16. Какие механизмы передачи через параметры вы знаете? Чем эти механизмы отличаются друг от друга?
17. Что такое передача по значению? Объясните на примерах.
18. Что такое передача по ссылке? Объясните на примерах.
19. Что такое передача по указателю? Объясните на примерах.
20. Что может быть фактическим параметром для формального параметра при передаче по значению? Объясните на примерах.
21. Что может быть фактическим параметром для формального параметра при передаче по ссылке? Объясните на примерах.
22. Что может быть фактическим параметром для формального параметра при передаче по указателю? Объясните на примерах.
23. Сколько результатов может иметь подпрограмма? Приведите примеры.
24. Что такое глобальные переменные?
25. Что такое локальные переменные?
26. Могут ли внутри функции использоваться глобальные переменные? Объясните на примере.
27. Могут ли во всей программе использоваться локальные переменные? Объясните.
28. Что произойдет, если глобальная и локальная переменные имеют одинаковые имена? Объясните на примерах.

Глава 12

ФУНКЦИИ

12.1. Локальные и глобальные переменные

Под областью действия переменной понимают область программы, в которой переменная доступна для использования. Если переменная объявлена внутри функции или блока программы, то такая переменная называется **локальной**. Она доступна или видима только в пределах этой функции или блока.

Пример использования локальных переменных.

```
#include <stdio.h>
void func(void)
{
    int var1=11; //локальная переменная var1, доступна внутри func()
    printf("Внутри func() значение var1=%d\n",var1);
}
void main()
{
    int var2=22; //локальная переменная var2, доступна внутри main()
    printf("Внутри main() значение var2=%d\n",var2);
    func();      //переменная var1 не доступна внутри функции main()
}
```

Переменная, объявленная вне любой из функций, называется глобальной переменной и доступна в области от точки ее объявления до конца файла. Глобальные переменные необходимы в тех случаях, когда программисту нужно сделать данные доступными для многих функций, а передавать данные из функции в функцию как параметры проблематично.

Пример использования глобальных переменных.

```
#include <stdio.h>
int var1=11;                                     //глобальная переменная var1
void func(void)
{
    printf("Внутри func() значение var1=%d\n",var1);
                                                //будет напечатано var1=11
}
int main()
{
    int var2=22; //локальная переменная var2, доступна внутри main()
    printf("Внутри main() значение var2=%d\n",var2);
                                                //будет напечатано var2=22

    func();
    printf("Внутри main() значение var1=%d\n",var1);
                                                //будет напечатано var1=11

    return 0;
}
```

12.2. Видимость переменных

Область видимости — часть текста программы, в которой может быть использован данный объект. Объект считается видимым в блоке или исходном файле, если известны тип и имя объекта в блоке или исходном файле. Под блоком подразумевается область программы, заключенная в фигурные скобки { }. Описание локальных переменных возможно не только в начале функции. В C++ можно объявить локальную переменную в любом месте внутри функции, тогда эта локальная переменная доступна от точки определения до следующей закрывающей фигурной скобки (конца блока).

Областью видимости переменной является блок, в котором она определена. Таким образом, если в теле функции будет блок, выделенный парой фигурных скобок, и в этом блоке объявляется переменная, то она будет доступна только в пределах этого блока и в блоках, вложенных в него, а не во всей функции. Переменная, объявленная в блоке, «прячет» любую другую переменную с таким же именем, описанную вне блока.

Пример, иллюстрирующий области видимости локальных и глобальных переменных.

```
#include <stdio.h>
int var1=12;                                //глобальная переменная var1
int main()
{
    printf(" Входим во внешний блок  \n");
    {
        int var1=34;
        printf("Во внешнем блоке значение var1
        =%d\n",var1);                      //будет напечатано var1=34
        {
            printf(" Входим во внутренний блок  \n");
            int var1=56;
            printf("Во внутреннем  значении var1 =%d\n",var1);
            //будет напечатано var1=56
        }
    }
    printf("В main() значение var1 = %d \n",var1);
    //будет напечатано var1=12
    return 0;
}
```

12.3. Время жизни переменной

Время жизни — это интервал выполнения программы, в течение которого программный объект (переменная или функция) существуют — доступна для использования. Время жизни переменной может быть глобальным или локальным. Объект с глобальным временем жизни имеет распределенную для него компилятором память и определенное значение на протяжении всего выполнения программы. Память для локальных переменных выделяется, когда начинается выполнение функции или блока, где они были объявлены. Как только происходит возврат из функции или выход из блока, эта память освобождается локальными переменными, снова становится доступной и может быть использована другими функциями и другими переменными. Таким образом, данные, содержащиеся в локальной переменной, могут быть изменены только при последующих вызовах функции. Функции в программе имеют глобальное время жизни, т. е. существуют на протяжении всего выполнения программы.

12.4. Модификаторы переменных

В языке C++ имеется несколько модификаторов (`auto`, `register`, `extern`, `static`), которые изменяют область действия и время жизни переменных (табл. 12.1). Модификаторы `auto`, `register` определяют объекты с локальным временем жизни, а `extern`, `static` объекты с глобальным временем жизни.

Таблица 12.1. Модификаторы переменных

Модификатор	Время жизни	Область видимости	Примечание
<code>auto</code>	Локальное	Блок	Временное выделение памяти
<code>register</code>	Локальное	Блок	Временное выделение памяти в машинном регистре, убыстряет работу программы
<code>static</code>	Глобальное	Блок	Выделяется постоянная память, значение сохраняется при выходе из блока
<code>extern</code>	Глобальное. Переменная объявляется как глобальная в одном файле, чтобы ее «увидеть» в другом файле, можно использовать модификатор — <code>extern</code>	Блок	Определение переменной, объявленной в другом файле, становится видимым внутри блока

Пример применения модификатора `static`.

```
#include <stdio.h>
void func1(void);
//прототипы функций func1(), func2() и func3()
void func2(void);
void func3(void);
int main()
{
    int i;
    /* значения трех разных статических переменных с одинаковым
       именем nfunc сохраняются при выходе из функций, но видимы
       они только внутри соответствующих функций */
    for (i=1; i<=3; i++) func1();    printf("\n");
```

```
    for (i=1; i<=4; i++) func2();          printf("\n");
    for (i=1; i<=5; i++) func3();
    return 0;
}
void func1(void)
{
    static int nfunc=0;
    /* память под переменную nfunc выделяется постоянная, поэтому
       значение nfunc сохраняется при выходе из функции */
    nfunc+=1;
    printf("Функция func1 вызывалась %d раз\n", nfunc);
}
void func2(void)
{
    static int nfunc=0;
    nfunc+=1;
    printf("Функция func2 вызывалась %d раз\n", nfunc);
}
void func3(void)
{
    static int nfunc=0;
    nfunc+=1;
    printf("Функция func3 вызывалась %d раз\n", nfunc);
}
```

Результат выполнения программы:

```
Функция func1 вызывалась 1 раз
Функция func1 вызывалась 2 раз
Функция func1 вызывалась 3 раз
Функция func2 вызывалась 1 раз
Функция func2 вызывалась 2 раз
Функция func2 вызывалась 3 раз
Функция func2 вызывалась 4 раз
Функция func3 вызывалась 1 раз
Функция func3 вызывалась 2 раз
Функция func3 вызывалась 3 раз
Функция func3 вызывалась 4 раз
Функция func3 вызывалась 5 раз
```

12.5. Функции с переменным числом параметров

В языке C++ допустимы функции, количество параметров у которых при компиляции не определено. Функция с переменным числом параметров имеет следующий формат прототипа:

```
тип имя_функции (список_параметров, ...);
```

Для правильной работы каждая функция с переменным списком параметров должна иметь механизм определения итогового их количества. Для этого обычно один из явных параметров функции (обычно последний в списке явных параметров функции) служит значением числа реальных, но не указанных фактических параметров. Приведем пример функции, вычисляющей сумму переданных ей параметров.

```
# include <stdio.h>
int summ(int k, ...);
//прототип функции с переменным числом параметров

void main()
{
    printf("1+2+3+4+5 = %d\n", summ(5, 1, 2, 3, 4, 5));
    printf("6+5 = %d\n", summ(2, 6, 5));
    printf("4+3+9 = %d\n", summ(3, 4, 3, 9));
}

int summ(int k, ...)
//определение функции с переменным числом параметров
{
    int *p = &k; /*объявление указателя p и инициализация его адресом
                  последнего явного параметра функции summ*/
    int s = 0; //задание начального значения суммы s
    for (int i = 0; i < k; i++) /*в цикле с числом повторений, равным
                               значению последнего явного параметра функции*/
        s += *(++p); //сначала определяем адрес текущего
                    //слагаемого (с помощью префиксной операции инкремента),
                    //разыменовываем его с помощью операции «звездочка»
                    //и складываем полученное значение со старым значением s*/
    return s; //возврат вычисленного значения s в вызывающую функцию
}
```

Фактические параметры хранятся в памяти в соседних ячейках, поэтому объявлен указатель, в котором хранится адрес последнего, передаваемого в функцию явно параметра. Доступ к остальным параметрам получается за счет сдвига указателя.

12.6. Рекурсивные функции

Рекурсия — это такой способ организации вычислительного процесса, при котором функция в ходе выполнения составляющих ее операторов обращается сама к себе. Использование рекурсии обычно выглядит более изящно, чем итерационные алгоритмы, и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызвать переполнение оперативной памяти компьютера.

Пример программы с использованием рекурсии. Вычислить факториал числа N . (Факториалом натурального числа N называют произведение чисел $1 * 2 * 3 * 4 * \dots * N$.) Число N вводится с клавиатуры.

```
#include <stdio.h>
double Faktorial(int k)           //определение рекурсивной функции
{
    if (k == 0)                   //проверка условия завершения рекурсии
        return 1;
    else
        return k*Faktorial(k-1);  //рекурсивное вычисление k!
}
int main() {
    int N;
    double F;
    printf("Введите целое число N< 20 ");
    scanf("%d",&N);
    F=Faktorial(N);
    printf("N!=%7.0lf\n",F);
    getchar();
    return 0;
}
```

Пояснение. С клавиатуры вводится значение N , и в выражении $F=Faktorial(N)$ вызывается функция `Faktorial` с параметром-значением N . В функции проверяется условие « $N=0$ ». Если оно выполняется, то имени функции `Faktorial` присваивается значение «1», на этом выполнение подпрограммы завершается. Если условие « $N=0$ » не выполняется, то вычисляется произведение $N * Faktorial(N-1)$. Вычисление произведения носит рекурсивный характер, так как при этом осуществляется вызов функции `Faktorial(N-1)`. Ее значение вычисляется через вызов функции

`Faktorial(N-2)`, которая, в свою очередь, вызывает функцию `Faktorial(N-3)`, и т. д. до тех пор, пока значение параметра `N` не станет равным 0. Так как базовая часть описания рекурсивной функции `Faktorial` определяет значение `Faktorial=1` для `N=0`, то рекурсивные вызовы функции больше не выполняются, а, наоборот, выполняется вычисление функции `Faktorial` для чисел, возрастающих от 1 до `N`. Причем функция `Faktorial` всякий раз возвращает значение, равное произведению очередного числа `N` на факториал от `(N-1)`. Последнее возвращение результата вычисления функции `N` присвоит переменной `F` значение произведения всех чисел от 1 до `N`, т. е. факториал числа `N`.

Понятие рекурсии также может относиться и к функции `main()`, так как функция `main()` является обычной функцией, выполняющей действия, предписанные программистом. Рассмотрим особенности действия рекурсии на примере функции `main()`

Пример 2. Попросить пользователя ввести с клавиатуры символ. Если введен символ 'y', то завершить программу, в противном случае — попросить ввести новый символ.

```
#include <stdio.h>

int main()
{
    char ch, temp;
    printf("ch = ");           //вывод приглашения к вводу символа
    scanf("%c%c", &ch, &temp); //ввести с клавиатуры два
                               //символа — первый сигнал к продолжению или прекращению ввода,
                               //второй искомый символ*/
    if (ch != 'y' && ch != 'Y') //если было введено y или Y
        main();               /*рекурсивно вызвать функцию main() и т. д., пока
                               не будет введен любой символ, отличный от y или Y*/
    return 0;
}
```

12.7. Перегрузка функций

Перегрузка функций — это возможность определять и использовать в программе несколько разных (но схожих по назначению) функций с одним и тем же именем, но с разными *сигнатурами*. *Сигнатура* функции состоит из имени функции и совокупности типов ее

формальных параметров. *Сигнатура функции* зависит от количества формальных параметров, от их типов и от порядка их следования. Тип результата функции не входит в сигнатуру функции. Распознавание перегруженных функций осуществляется по их сигнатурам. Перегруженные функции должны иметь одинаковые имена, но типы их параметров должны различаться по количеству и (или) по порядку следования в списке формальных параметров.

Пример программы перегрузки функции

```
#include <stdio.h>
int max(int , int);      /*прототип 1-го экземпляра функции max
                          (определение максимума из двух целых чисел)*/
int max(int, int, int):  /*прототип 2-го экземпляра (1-й копии)
                          функции max (определение максимума из трех целых чисел)*/
double max(double, double); /*прототип 3-го экземпляра
                              (2-й копии) функции max (определение макс. из двух числе типа double)*/
int main() {
    int a,b,c, max_ab, max_abc;
    double x,y, max_xy;
    printf("Введите через пробел a, b,c \n");
    scanf("%d%d%d",&a,&b,&c);
    max_ab=max(a,b);
    max_abc=max(a,b,c);
    printf("max_ab=%d max_abc=%d\n",max_ab,max_abc);
    printf("Введите через пробел x,y\n");
    scanf("%lf%lf",&x,&y);
    max_xy=max(x,y);
    printf("max_xy=%lf\n. max_xy);
    getchar();
    return 0;
}
int max(int a1, int b1)
    /*определение первого экземпляра функции max
{ int m1;
  if (a1>=b1)
    m1=a1;
  else m1=b1;
  return m1;
}
int max(int a1, int b1, int c1)
    /*определение 2-го экземпляра (1-й копии) функции max
{
  int m1;
```



```

    if ((a1>=b1)&&(a1>=c1))    m1=a1;
    if ((b1>=a1)&&(b1>=c1))    m1=b1;
    if ((c1>=a1)&&(c1>=b1))    m1=c1;
    return m1;
}
double max(double a1, double b1)
    //определение 3-го экземпляра 3-й копии) функции max
{ double m1;
  if (a1>=b1)
    m1=a1;
  else m1=b1;
  return m1;
}

```

Функция `max` перегружена с тремя разными списками параметров. Первая и третья версии отличаются типами параметров, а вторая — их количеством. Типы возвращаемых значений перегруженных функций могут быть одинаковыми или разными. При создании двух функций с одинаковым именем и одинаковым списком параметров, но с различными типами возвращаемых значений будет диагностирована ошибка компиляции.

При правильной перегрузке функций никакой путаницы при вызове функции не будет, поскольку нужная функция определяется по совпадению используемых параметров. Это позволяет создать функцию, которая сможет, например, усреднять целочисленные значения, значения типа `double` или значения других типов без необходимости создавать отдельные имена для каждой функции. При вызове перегруженной функции компилятор автоматически определит, какой именно вариант функции следует использовать.

Пример. Ввести с клавиатуры числа типа `int` и `float`, найти их абсолютные величины. Вычисление абсолютной величины оформить в виде функции.

```

#include <stdio.h>
int ABS(int);           //прототипы функций ABS
float ABS(float);
int main()
{
    int i; float a;
    printf("Введите целое число: ");
    scanf(" %d",&i);
    printf("Введите вещественное число: ");
}

```

```
scanf(" %f",&a);
printf("abs(%d)=%d; abs(%f)=%f \n",i,ABS(i),a,ABS(a));
return 0;
}
int ABS(int a)                                //определение функции int ABS()
{
    if (a>=0)
        return a;
    else
        return -a;
}
float ABS(float a)                            //определение функции float ABS()
{
    if (a>=0.)
        return a;
    else
        return -a;
}
```

Пояснение. В этом примере перегружена функция вычисления модуля числа. Обе функции имеют одинаковые имена. В одной в качестве параметра и возвращаемого значения используется целочисленная переменная, а во второй — вещественная. В функции `main()` в зависимости от типов используемых фактических параметров вызывается нужная функция.

Задания

1. В программе объявите и проинициализируйте две целочисленные переменные «а» — одну глобальную, другую локальную. Выведите на экран значения этих переменных. Чему будут равны их значения, если убрать инициализацию? Объясните полученные результаты.
2. Напишите функцию, вычисляющую сумму переданных ей параметров. Число параметров может быть произвольным.
3. Напишите функцию, вычисляющую произведение переданных ей параметров. Число параметров может быть произвольным.
4. Напишите функцию вычисления факториала целого числа, введенного с клавиатуры.
5. Напишите программу, демонстрирующую рекурсивный вызов функции `main()`.

6. Напишите функцию, вычисляющую модуль переменных разного типа.

7. Создайте библиотеку, содержащую функции для вычисления суммы, разности, произведения и частного двух переменных различных типов (int, float, double).

Контрольные вопросы

1. Какая переменная называется локальной? Приведите примеры.
2. Как в программе объявить глобальную переменную? Приведите примеры.
3. В чем отличия между локальными и глобальными переменными?
4. Что такое область видимости и время жизни переменных?
5. Какие модификаторы можно использовать для работы с переменными? Приведите примеры.
6. Какие особенности необходимо учитывать при описании функции с переменным числом параметров?
7. Что такое рекурсия? Какие преимущества дает использование рекурсивных функций?
8. Что такое перегрузка функций? Для чего она нужна?

Глава 13

ТИПЫ ДАННЫХ, ВВОДИМЫЕ ПОЛЬЗОВАТЕЛЕМ

13.1. Переименование типов (typedef)

Иногда, чтобы сделать программу более ясной, удобной и эффективной, можно задать стандартному типу новое имя с помощью ключевого слова `typedef`:

```
typedef <тип> <новое_имя>;
```

где `<тип>` — стандартный тип языка C++;

`<новое_имя>` — новое имя типа, придуманное программистом.

Например,

```
typedef unsigned char UN_CHAR;
```

Здесь `unsigned char` — стандартный тип C++ — беззнаковый символьный;

`UN_CHAR` — новое имя типа, которое в программе будет использоваться наравне со старым типом `unsigned char`

```
typedef unsigned int UN_INT;
```

Здесь `unsigned int` — стандартный тип C++ — беззнаковое целое;

`UN_INT` — новое имя типа, которое в программе будет использоваться наравне со старым типом `unsigned int`.

Задавать новое имя стандартному типу необходимо тогда, когда данное этого типа используется в качестве типа возвращаемого значения в функции или при преобразовании типов.

Пример. Определить типы беззнаковый символьный и беззнаковый целый. Ввести символ и число. Определить ASCII код введенного символа, символы следующий и предыдущий для введенного, а также символ ASCII, код которого равен введенному числу.

```
#include <stdio.h>

typedef unsigned char UN_CHAR;
//определение нового типа — беззнаковый символьный
typedef unsigned int UN_INT;
//определение нового типа — беззнаковое целое
UN_INT ORD(UN_CHAR ch)
//заголовок функции — получения кода ASCII символа ch
{
    return UN_INT(ch);
//возврат в вызывающую функцию результата —
//операции преобразования типов
}
UN_CHAR CHR(UN_INT k)
//заголовок функции определения символа по его ASCII коду
{
    return UN_CHAR(k);
//возврат в вызыв. функцию результата операции преобраз. типов
}
UN_CHAR PRED(UN_CHAR ch)
//заголовок функции определения предыдущего символа
{
    if (ORD(ch)>0) //если ASCII код символа больше нуля
        return CHR(ORD(ch)-1);
//возврат символа, код ASCII которого на 1 меньше заданного
    else
        return ch; //в противном случае возврат заданного символа
}
UN_CHAR SUCC(UN_CHAR ch)
//заголовок функции определения следующего символа
{
    if (ORD(ch) < 255) //если ASCII код символа меньше 255
        return CHR(ORD(ch)+1);
//возврат символа, код ASCII которого на 1 больше заданного
    else return ch; //в противном случае возврат заданного символа
}
int main()
{
    UN_CHAR sym;
```

```

UN_INT kod;
printf("Введите символ >");
sym = getchar();
printf("ASCII код введенного символа = %u\n", ORD(sym));
printf("предыдущий символ = %uc следующий символ =
      %uc\n", PRED(sym), SUCC(sym));
printf("Введите число >");
scanf("%u", &kod);
printf("Введенному ASCII коду %u соответствует символ
      %uc\n", kod, CHR(kod));
getchar();
return 0;
}

```

Существует и другой способ задания нового типа:

```
typedef <тип> <новое_имя[размерность]>;
```

Например:

```

typedef char TEXT[100];           /* новый тип TEXT задает тип
                                   одномерного массива из 100 элементов типа char*/

```

Введенные таким образом имена типов можно использовать так же, как и имена стандартных типов:

```

TEXT stroka;                      //одномерный массив из 100 символов
TEXT str[5];                      //массив из 5 строк по 100 символов в каждой

```

Пример. Ввести с клавиатуры строку и разбить ее на слова. Слова в строке разделяются любым количеством пробелов.

```

#include <stdio.h>
typedef char TEXT[100];           /*новый тип TEXT задает тип
                                   одномерного массива из 100 элементов типа char*/

int main()
{
    TEXT stroka, slova[50];
    printf("Введите строку >");
    int i, j, k;
    gets(stroka);
    k=0;                          //задать начальное значение количеству слов в строке
    j=0;                          //задать нач. значение номеру символа в выделяемом слове
    i=0;                          //задать начальное значение номеру символа в строке
}

```



```

while ((i<100)&&(stroka[i]!='\0')) {
    //цикл, пока не конец строки
    if ( stroka[i]!=' ') { //если i-й символ строки не пробел
        slova[k][j]=stroka[i]; //запишем i-й символ строки
                               //на j-е место в слове с номером k*/
        j=j+1; //увеличим на 1 номер символа в слове
    }
    else { //i-й символ строки пробел
        if (j>0){ //если было выделено слово
            slova[k][j]='\0';
            //записать нуль-символ в конец выделенного слова
            k=k+1; //увеличить на 1 количество слов
            j=0; //задать нач. знач.номеру символа в выделяемом слове
        }
        i=i+1; //увеличим на 1 номер символа в строке
    }
    if (j>0){ //если в конце строки нет пробела
        slova[k][j]='\0';
        //записать нуль-символ в конец выделенного слова
        k=k+1; //увеличить на 1 количество слов
    }
    printf("Выделенные слова\n");
    for (i=0; i<k; i++)
        puts(slova[i]);
    getchar();
    return 0;
}

```

Кроме задания типам с длинными описаниями более коротких псевдонимов, `typedef` используется для облегчения переносимости программ: если машинно-зависимые типы объявлять с помощью операторов `typedef`, при переносе программы потребуется внести изменения только в эти операторы.

13.2. Перечисления (enum)

При написании программы часто возникает потребность определения целой переменной, которая может принимать лишь одно значение из определенного множества значений. Для этого рекомендуется использовать спецификацию типа «перечисление» для определения возможных значений переменной. Оператор `enum` должен использоваться согласно следующему синтаксису:

```
enum имя_типа {список_именованных_констант};
```

Константы должны быть целочисленными и могут инициализироваться обычным образом. Например, оператор

```
enum week{mon=1, td=2, wd=3, thd=4, fd=5, st=6, sd =7};
```

определяет тип-перечисление `week` с допустимыми значениями

```
mon=1, td=2, wd=3, thd=4, fd=5, st=6, sd =7
```

Введенное таким образом имя типа можно использовать в программе так же, как и имя любого другого типа:

```
имя_типа имя_переменной;
```

Например, оператор

```
week day1;
```

описывает переменную `day1` типа `week`, которая может принимать только перечисленные значения. На самом деле этими значениями будут константы 1, 2, 3, 4, 5, 6, 7, имеющие имена, указанные в перечислении `week`.

Возможен и другой способ задания переменной типа перечисления

```
enum имя_типа {список_именованных_констант} имя_переменной;
```

Например, оператор

```
enum week{mon=1, td=2, wd=3, thd=4, fd=5, st=6, sd =7}  
day2;
```

описывает переменную `day2` типа-перечисления `week`, которая может использоваться так же, как и переменная `day1`.

Если при задании перечисления отсутствует инициализация именованных констант, то первая константа обнуляется, а каждой следующей присваивается на единицу большее значение, чем предыдущей. Например, оператором

```
enum Err {ERR_READ, ERR_WRITE, ERR_CONVERT} error;
```

вводится переменная `error` перечислимого типа `Err`, которая может принимать значения `ERR_READ`, `ERR_WRITE`, `ERR_CONVERT`. Так как инициализация этих именованных констант отсутствует, то по умолчанию `ERR_READ=0`, `ERR_WRITE=1`, `ERR_CONVERT=2`.

Соответственно ранее вводимый тип-перечисление `week` можно определить по-другому:

```
enum week{mon=1, td, wd, thd, fd, st, sd};
```

В этом перечислении именованные константы получают следующие значения:

```
mon = 1, td = 2, wd = 3, thd = 4, fd = 5, st = 6, sd = 7
```

Если в программе не предполагается использовать имя типа несколько раз, его можно опустить при описании, а переменную перечислимого типа описать сразу:

```
enum {список_именованных_констант} имя_переменной;
```

Например:

```
enum {two=2, three, for, ten=10, eleven, fifty=ten+40}  
primer;
```

Здесь константам `three` и `for` присваиваются значения 3 и 4, константе `eleven` — 11, константе `fifty` — 50. Перечисление может содержать и повторяющиеся значения, а вот идентификаторы в списке перечисления должны быть отличны от всех других идентификаторов в программе. Например, перечисление

```
enum {one=1, edinita=1, five=5, piaterka=5, ot1=5};
```

содержит пару повторяющихся значений 1 и 5, а все идентификаторы — `one`, `edinita`, `five`, `piaterka`, `ot1` являются уникальными и отличными от других идентификаторов программы.

С переменными перечислимого типа допустимы только следующие операции: им можно присваивать значения констант перечисления, одной переменной можно присвоить значения другой переменной, их можно вводить с клавиатуры и выводить на экран.

Например, строка

```
enum week{mon=1, td, wd, thd, fd, st, sd};
```

вводит пользовательский тип, который имеет имя `week`. Теперь можно объявлять переменные этого типа, присваивать им значения, выводить на экран:

```
week day, day1;  
day = thd;  
day1 = sd  
printf("%d\n", day);  
printf("%d\n", day);
```


Пример. По номеру дня недели вывести на экран название дня недели.

```
#include <stdio.h>
int main()
{
    enum {mon=1, td, wd, thd, fd, st, sd} day;
    puts("Введите номер дня недели ");
    scanf("%d", &day);
    switch (day){
    case mon:
        printf("Понедельник"); break;
    case td:
        printf("Вторник"); break;
    case wd:
        printf("Среда"); break;
    case thd:
        printf("Четверг"); break;
    case fd:
        printf("Пятница"); break;
    case st:
        printf("Суббота"); break;
    case sd:
        printf("Воскресенье"); break;
    default: printf("Неверный номер дня недели!\n");
    }
    return 0;
}
```

Пояснение. Переменная `day` перечислимого типа может принимать значения `mon`, `td`, ..., `sd`: `mon=1`, а каждой следующей константе присваивается значение на единицу большее, чем предыдущей. Имя типа не вводится, так как использование других переменных этого типа в программе не предусматривается.

13.3. Структуры (struct)

Реальные данные об объектах часто описываются величинами разных типов. Например, автомобиль в автосалоне имеет следующие характеристики: модель (строка), год выпуска (типа **int**), объем двигателя (типа **double**), наличие подушек безопасности (типа **bool**) и т. д. В этом случае возникает необходимость хранить и обрабатывать совокупности данных различных типов, поэтому приходится использо-

вать отдельные массивы для каждого типа данных, а для установления соответствия между ними вводятся соответствующие индексы. Такой подход не очень удобен и существенно усложняет написание программы. В C++ существует другой способ решения таких задач — использование комбинированного типа данных, который называется структурой (или структурным типом данных).

Объявление комбинированного типа данных. Комбинированный тип данных (*структура*) — это структурированный тип данных, состоящий из фиксированного числа компонентов различного типа. То есть структуры объединяют в одной переменной элементы разных типов. Переменную типа «структура» можно объявить двумя способами.

Способ 1.

```
struct <имя_типа_структуры>{
    <тип_списка_1>    <список_имен_полей_1>;
    <тип_списка_2>    <список_имен_полей_2>;
    ...
    <тип_списка_k>    <список_имен_полей_k>;
};
<имя_типа_структуры> <имя_переменной>;
```

где <имя_типа_структуры> — имя пользовательского типа для структур определенного вида;

struct — ключевое слово, с которого начинается описание типа «структура»;

<список_имен_полей_1>...<список_имен_полей_k> — имена полей структуры (задаются по правилам задания идентификаторов). Имена полей структуры должны быть уникальными в пределах данной структуры, т. е. в разных структурах поля могут иметь одинаковые имена. Однако во избежание ошибок лучше, чтобы имена полей были уникальными в пределах всей программы;

<тип_списка_1>...<тип_списка_k> — любой стандартный или определенный пользователем тип языка C++, включая тип «структура». Если используется тип, определенный пользователем, то он должен быть объявлен до описания структуры.

Способ 2.

```
struct <имя_типа_структуры>{
    <тип_списка_1>    <список_имен_полей_1>;
    <тип_списка_2>    <список_имен_полей_2>;
    ...
    <тип_списка_k>    <список_имен_полей_k>;
}<имя_переменной>;
```

где <имя_переменной> — идентификатор переменной типа «структура»;

<список_имен_полей_1>...<список_имен_полей_k> — имена полей записи (см. выше); <тип_списка_1>...<тип_списка_k> — любой стандартный или определенный пользователем тип языка C++, включая тип «структура».

Имя типа структуры определяет новый тип данных, и его можно использовать наряду со стандартными типами.

Пример 1.

```
struct Avto {                                //объявление типа структуры из 4 полей
    char Model[20];                          //модель автомобиля
    int Year;                                //год выпуска
    double DV;                               //объем двигателя
    int Power;                              //мощность двигателя
};                                           //окончание описания типа структуры
Avto M,V;                                  //объявление переменных типа Avto
```

Пример 2.

```
struct anketa
{
    //мы ввели тип данных anketa, который объединяет три элемента:
    char name[50];                          //строковая переменная — фамилия человека,
    int age;                                //целая переменная — его возраст,
    float pay;                              //вещественная переменная — его оклад
};
struct dinner
{
    //мы ввели тип данных dinner, который объединяет два элемента:
    char cafe_name[70];                     //строковая переменная — название кафе, где можно пообедать,
    float cost;                             //вещественная переменная — стоимость обеда
};
```

Теперь можно ввести структурные переменные, которые описываются с помощью введенных структурных типов данных:

```
имя_структурного_типа имя_переменной;
```

Примеры определения структурных переменных:

```
anketa a1, a2, a3;
//объявление трех переменных структурного типа anketa
dinner d1, d2, d3;
//объявление трех переменных структурного типа dinner
```


Пример 3. Определение структурных переменных одновременно с определением типа.

```
struct anketa                //определяем структурный тип анкета
{
    char name[50];
    int age;
    float pay;
} a1,a2,a3;                  //объявляем 3 переменных нового структурного типа

struct                       //без определения нового структурного типа
{
    char name[50];
    int age;
    float pay;
} a1,a2,a3;                  //сразу определяем переменные структурного типа
```

Если задано имя структурного типа без последующего списка переменных, то такое объявление вводит новый тип данных — шаблон структуры, и память для него не выделяется. Память выделяется только при объявлении структурных переменных.

Работа с переменными структурного типа. Обращение к значению поля осуществляется с помощью составного имени, которое состоит из имени переменной типа «структура» и идентификатора поля, разделенных точкой. Составное имя можно использовать везде, где допустимо применение значения типа поля.

Полям записи можно присваивать значения с помощью оператора присваивания. Например,

а) присваивание значений полям записи типа *Avto* (см. пример выше):

```
V.Year=2010;
V.DV:=1.4;
V.Power:=121;
```

Присваивание значений полям строкового типа осуществляется с помощью стандартных функций библиотеки *string*

```
strcpy(V.Model,"BA3-2115");
```

б) присваивание полям одной структуры значений полей другой структуры того же самого типа:

```
M = V;                //информация из структуры V копируется в структуру M
```

При выполнении этого оператора компилятор создает побитовую копию одной структуры в другую. По умолчанию применение стандартных операций («+», «-», «*», «/» и т. д.) к переменным типа структура не применимо. Приведем пример, демонстрирующий отсутствие возможности сложения двух структур.

```
#include <stdio.h>

struct chisla
{
    int a;
    float x;
};
void prn(struct chisla &ch)
{
    printf("a = %d\tx = %f\n", ch.a, ch.x);
}
int main()
{
    struct chisla ch1 = {3, 5.67}, ch2;
    prn(ch1);
    prn(ch2);
    ch2 = ch1;
    //выполняется побитовая копия структуры ch1 в структуру ch2
    prn(ch2);
    ch2 = ch2 + ch1;
    //на эту строчку компилятором будет сгенерирована ошибка
    return 0;
}
```

Пояснение. В программе введен структурный тип данных для хранения целого и вещественного чисел. Операция «=» осуществляет побитовую копию полей структуры. А на оператор сложения двух структур (`ch2 = ch2 + ch1;`) компилятор сгенерирует ошибку: «error C2676: binary '+' : 'chisla' does not define this operator or a conversion to a type acceptable to the predefined operator», означающую невозможность применения бинарной операции «+» к структурному типу `chisla`.

Чтобы реализовать выполнение стандартных операций для переменных структурного типа, необходимо перегрузить эти операции. Подробно этот материал разбирается в главе 16, параграф 16.7.

Ввод информации в структуру с клавиатуры и вывод данных из структуры осуществляется по полям:

```

//считываем данные
gets(V.Model);
scanf("%d",&V.Year);
scanf("%lf%d",&V.DV, &V.Power);

//выводим данные на экран
printf("%s  %6d V.Model %4.1lf..%d\n", V.Model,V.Year,
V.DV, V.Power);
```

Сравнение записей можно также осуществить по полям:

```

if ((V.Model==M.Model) && (V.Year==M.Year)
&& (V.DV==M.DV) && (V.Power==M.Power))
printf("структуры равны\n");
```

Инициализация структур. Переменная структурного типа инициализируется так же, как и другие переменные. После имени переменной ставится знак равенства, и в фигурных скобках указываются все значения элементов структуры через запятую в порядке их описания. Можно одновременно описать структурный тип данных, объявить переменную этого типа и инициализировать ее.

Примеры инициализации переменных структурного типа.

Пример 1.

```

struct anketa //вводим новый структурный тип данных
{
    char name[50];
    int age;
    float pay;
} a = {"Иванов", 50, 3000. };
//определяем переменную a и задаем ей начальные значения
```

Пример 2.

```

struct dinner //вводим новый структурный тип данных
{
    char cafe_name[70];
    float cos;t
} d={"Ростикс", 150.}; //объявляем и инициализируем переменную
```


Пример 3.

```
/*ранее введенный структурный тип данных anketa используем для объявления и инициализации переменных этого типа*/  
anketa a1 = {"Сидоров", 23, 18000};
```

Пример 4.

```
/*ранее введенный структурный тип данных dinner используем для объявления и инициализации переменных этого типа*/  
dinner dnext={"Колобок", 120.};
```

Доступ к элементам. Доступ к элементам структуры выполняется с помощью операции точка(.) при обращении к полю через имя структуры и операции -> при обращении через указатель:

```
имя_переменной.имя_элемента
```

или

```
указатель_на_структуру->имя_элемента
```

Пример получения доступа к элементам структуры:

```
struct anketa  
{  
    char name[50];  
    int age;  
    double pay;  
} a, *pa;
```

Теперь `a.name` или `pa->name` можно использовать, как и любой другой массив типа `char`, `a.age` или `pa->age` как переменную типа `int`, а `a.pay` или `pa->pay` как переменную типа `double`, например, можно присвоить им начальные значения:

```
a.name[0] = 'П';  
a.name[1] = 'е';  
a.name[2] = 'т';  
a.name[3] = 'р';  
a.name[4] = '\\0';
```

или

```
pa->name[0] = 'П';  
a.age = 25;
```

ИЛИ

```
pa->age = 25;
a.pay = 1525.0;
```

ИЛИ

```
pa->pay = 1525.0;
```

Пример. Инициализировать структуру и вывести на экран ее элементы.

```
#include <stdio.h>
int main()
{
    struct anketa
    {
        char name[50];
        int age;
        float pay;
    };
    struct anketa a={"Петров", 25, 1525.0 };
    puts("Печать элементов структуры типа anketa");
    puts(a.name);
    printf("Возраст %d\n",a.age);
    printf("Оклад %f\n",a.pay);
    return 0;
}
```

Пример. Определить структурный тип данных «турист», содержащий информацию о фамилии туриста, стоимости тура и места отдыха. Написать функции ввода информации о туристе с клавиатуры и вывода введенной информации на экран.

[illegible]

```
int main()
{
    turist t;                //объявляем переменную структурного типа
                              //вызываем функции
    vvod(t);                 //ввода информации
    vyvod(t);                //и вывода информации
    return 0;
}

void vvod(turist &t)         //реализация функции ввода информации
{
    printf("ФИО туриста: ");
    scanf("%s", t.fio);
                              //для доступа к полям структуры используем «.»
    printf("Стоимость тура: ");
    scanf("%d", &t.cost);
    printf("Страна: ");
    scanf("%s", t.country);
}

void vyvod(turist &t)       //реализация функции вывода информации
{
    printf("Турист\n");
    printf("%s\n", t.fio);
    printf("купил тур стоимостью %d", t.cost);
    printf("в страну %s\n", t.country);
}
```

Пояснение. В программе вводится структурный тип данных `turist`. Для ввода данных с клавиатуры используется функция `vvod`, а для вывода информации на экран применяется функция `vyvod`. Передача переменной структурного типа осуществляется по ссылке. В теле функций доступ к полям структуры осуществляется с использованием операции «.».

13.4. Объединения (union)

Объединение (`union`) представляет собой частный случай структуры, все поля которой располагаются по одному и тому же адресу. Формат описания такой же, как и у структуры, только вместо ключевого слова `struct` используется слово `union`. Длина объединения равна наибольшей из длин его полей. В каждый момент времени в пе-

ременной типа `union` хранится только одно значение, и ответственность за его правильное использование лежит на программисте. Объединения применяют в случае необходимости хранить значения различных типов, и при этом заранее известно, что эти значения одновременно не будут использоваться.

Пример использования объединения

```
#include <stdio.h>

enum {CARD, CHECK } pay;           //объявляем тип-перечисление
                                   //0 — оплата по карте
                                   //1 — оплата по чеку

union
{
    char card[25];
    long int check;
} info;                             //объявляем переменную типа объединение

int main()
{
    puts("Введите 0, если оплата по карте, и 1, если
        оплата чеком");
    scanf("%d", &pay);
    switch (pay)                    //в зависимости от типа выбранного платежа
    {                               //в объединении info используем
        case CARD:                 //либо поле card
            puts("Введите Ваше имя");
            scanf("%s", info.card);
            printf("Оплата по карте: %s", info.card); break;
        case CHECK:                //либо поле check
            puts("Введите сумму платежа");
            scanf("%d", &info.check);
            printf("Оплата чеком: %d", info.check); break;
        default:
            printf("Ошибка");
    }
    return 0;
}
```

Пояснение. В программе вводится объединение, в котором может храниться либо массив символов с названием карточки, либо номер чека, по которому будет осуществляться оплата. Доступ к полям объединения осуществляется аналогично структурам. Переменная `info` может использоваться как целочисленная переменная, и как массив символов.

Тесты

1. Задать новое имя стандартному типу можно с помощью ключевого слова:

- а) typedef;
- б) rename;
- в) в языке C++ задать новое имя стандартному типу нельзя.

2. Что представляет переменная `str`, введенная следующим образом:

```
typedef char TEXT[20]  
TEXT str[3];
```

- а) массив из трех строк по 20 символов в каждой;
- б) массив символов из 60 элементов;
- в) в коде содержится ошибка, программа не скомпилируется.

3. Ключевое слово `enum` используется для:

- а) задания перечислений;
- б) определения структур с ограниченными возможностями;
- в) такое ключевое слово не используется в языке C++.

4. Структура — это:

- а) стандартный тип данных;
- б) тип данных, вводимый пользователем, позволяющий объединять элементы одного типа;
- в) тип данных, вводимый пользователем, позволяющий объединять элементы разных типов.

5. Выберите пример правильной инициализации структурной переменной `avto`, содержащей сведения об автомобиле: марку, год выпуска, цвет:

- а)

```
struct AVTO  
{  
    char model[20];  
    int year;  
    char color[20];  
} avto = {  
    "Ford focus", 2006, "blue"  
};
```
- б)

```
struct AVTO  
{  
    char model[20];  
    int year;  
    char color[20];  
};  
AVTO avto = { "Ford focus", 2006, "blue" };
```

в) оба варианта правильные.

6. Укажите правильный вариант выделения памяти под структурную переменную:

- а) `anketa p; p = new anketa;`
- б) `anketa *p; p = struct new anketa;`
- в) `anketa *p; p = new anketa.`

7. Выберите верное утверждение:

- а) объединение занимает меньше места в памяти по сравнению со структурой с таким же набором полей;
- б) все поля объединения не могут использоваться одновременно;
- в) оба утверждения верны.

Задания

1. Используя `typedef`, определить новые имена для всех стандартных типов. Объявить несколько переменных новых типов данных, задать им значения, вывести на экран.

2. С клавиатуры ввести строку, состоящую из нескольких слов, разделенных одним пробелом. Для хранения строки использовать новый тип данных `ТЕХТ`, введенный с помощью `typedef`. Написать функцию подсчета количества слов в строке.

3. Ввести с клавиатуры номер дня недели. Вывести на экран его название. Использовать перечисления (`enum`).

4. Определить структурный тип `adress` (улица, дом, квартира). Ввести указатель на тип `adress`. Выделить память для хранения информации. Проинициализировать, вывести на экран.

5. Определить структурный тип `анкета` (поля: фιο, адрес, год рождения, класс). Ввести информацию с клавиатуры для группы из трех школьников. Вывести школьников, у которых фамилия начинается на букву 'С'.

6. Определить структурный тип `клиент` (поля: фамилия, год рождения, количество покупок, стоимость покупок). Написать функции, генерирующие и выводящие на экран информацию о клиенте.

7. Определить структурный тип `школьник` (поля: фιο(вложенная структура), адрес, год рождения, класс, 10 оценок). Объявить переменную структурного типа `школьник`. Сгенерировать случайным образом информацию для этой переменной. Вывести на экран.

8. Смоделировать диалог покупки произвольного товара в магазине. При этом у покупателя есть возможность расплатиться либо по карточке, либо по чеку. Информацию о платеже хранить в объединении (`union`).

Контрольные вопросы

1. Каким образом в языке C++ можно переименовать стандартные типы данных? Для чего это нужно? Приведите примеры.
2. Как можно задать новый тип данных, предназначенный для хранения массива из нескольких строк ограниченной длины? Приведите примеры.
3. Что такое перечисления? Для чего они используются?
4. Какие варианты определения типа-перечисления можно использовать в языке C++?
5. Как индексируются константы при определении перечислений? Можно ли поменять эту индексацию?
6. Что такое структура? Для чего она предназначена? Какие преимущества предоставляет программисту использование структурных типов данных?
7. Как выполнить инициализацию переменных структурного типа? Приведите примеры.
8. Как можно получить доступ к полям структуры? В чем разница использования операций «.» и «->»?
9. Что такое объединения? Чем они отличаются от структур?

Глава 14

РАБОТА СО СТРУКТУРАМИ

Реальные данные об объектах часто описываются величинами разных типов. Например, автомобиль в автосалоне имеет следующие характеристики: модель (строка), год выпуска (типа **int**), объем двигателя (типа **double**), наличие подушек безопасности (типа **BOOL**) и т. д. В этом случае возникает необходимость хранить и обрабатывать совокупности данных различных типов, поэтому приходится использовать отдельные массивы для каждого типа данных, а для установления соответствия между ними вводятся соответствующие индексы. Такой подход не очень удобен и существенно усложняет написание программы. В C++ существует другой способ решения таких задач — использование *комбинированного типа* данных, который называется *структурой*. Структурный тип данных и действия с данными этого типа были рассмотрены в предыдущей главе.

14.1. Решение задач с использованием переменных комбинированного типа

Обратите внимание, что в C++ в качестве полей структур можно использовать переменные других структурных типов. Однако полем структуры не может быть переменная этого же структурного типа, но допускается использовать указатель на структуру данного типа. Продемонстрируем эти возможности на примерах.

Задача 1. Описать структуру с именем TOVAR, содержащую следующие поля: NAZV — название товара; STOIM — стоимость товара; KOL — количество товара.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив SHOP, состоящий из 5 элементов типа TOVAR;

- вывод на экран информации о товарах, стоимость которых больше введенного с клавиатуры значения;
- если таких товаров нет, выдать на дисплей соответствующее сообщение.

```
#include <stdio.h>
struct TOVAR{           //начало объявления структурного типа TOVAR
    char NAZV[20];       //объявление поля "название товара"
    int STOIM;           //объявление поля "стоимость товара"
    int KOL;             //объявление поля "количество товара"
};                       //конец объявления структурного типа "TOVAR"
const int n=5;          //объявление константы n — числа элементов массива SHOP
int main() {
    TOVAR SHOP[n];       //объявление массива SHOP
    int i;
    int ST;              //объявление цел.переменной— заданного значения стоимости
    int p;               /*если p=0, товаров со стоимостью больше заданной нет,
                        p=1 если есть товары со стоимостью выше заданной*/
    for (i=0; i<n; i++){ //в цикле
        printf("Введите название % товара ", i);
        gets(SHOP[i].NAZV);
        printf("Введите стоимоть % товара ", i);
        scanf("%d",&SHOP[i].STOIM);
        printf("Введите количество % товара ", i);
        scanf("%d",&SHOP[i].KOL);
        getchar();      //считывание с клавиатуры кодов клавиши Enter
    }
    printf("Введите заданную стоимость ");
    scanf("%d",&ST);
    p=0;                //задаем признак "нет товаров стоимостью выше заданной"
    for (i=0; i<n; i++) //в цикле
        if (SHOP[i].STOIM>ST){ //если стоим. i-го товара больше заданной
            p=1;              //задаем признак "есть товары стоимостью выше заданной"
            printf("%s %d %d\n",SHOP[i].NAZV, SHOP[i].STOIM,
                SHOP[i].KOL); //вывод на экран названия, стоимости
                               и количества товара, стоимость которого выше заданной*/
        }
    if (p==0)            //если товаров со стоимостью выше заданной не было
        printf("Товаров со стоимостью выше заданной не
        было\n");
                                                                    //выводим сообщение

    getchar();
    return 0;
}
```


Задача 2. Описать структуру с именем Data, содержащую поля:

- Day — день;
- Month — месяц;
- Year — год

и структуру Student, содержащую следующие поля:

- Full_Name — ФИО студента;
- Date — дата рождения типа Data;
- Group — группа.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив Spisok, состоящий из 30 элементов типа Student;
- вывод на экран отсортированной в алфавитном порядке информации о всех студентах.

```
#include <stdio.h>
#include <string.h> //подключение библиотеки работы со строками
struct Data{       //начало объявления структурного типа Data
    int Day;        //объявление поля Day
    int Month;      //объявление поля Month
    int Year;        //объявление поля Year
};                 //конец описания структурного типа Data
struct Student {   //объявление типа структуры из трех полей
    char Full_Name[50]; //ФИО студента
    Data Date;          //дата рождения
    char Group[10];     //группа
};                     //конец описания типа «структура»
void vvod_student(Student St) //заголовок функции ввода
    с клавиатуры значений одной переменной типа структура Student*/
{ printf("Введите ФИО студента ");
  gets(st.Full_Name);
  printf("Введите день рождения студента ");
  scanf("%d", St.Date.Day);
  printf("Введите месяц рождения студента ");
  scanf("%d", St.Date.Month);
  printf("Введите год рождения студента ");
  scanf("%d", St.Date.Year);
  printf("Введите группу студента ");
  gets(St.Group);
  getchar();
}
void vvod_spisok(Student X[], int n)
    //заголовок функции ввода сведений о группе студентов
{int i;
```

```

for (i=0; i<n;i++){
    {printf("ввод сведений о %d студенте\n",i);
        //вывод информации о номере студента
    vvod_student(X[i]);
        //вызов функции vvod_student для ввода сведений об i-м студенте
    }
}
void sort_spisok(Student X[], int n) /*заголовок функции
    сортировки (методом обмена) по алфавиту списка студентов*/
{int i,j;
    Student d;
    for (i=0; i<n-1; i++) //в цикле по номерам проходов
        for (j=0; j<n-1-i; j++) //в цикле
            if (strcmp(X[j].Full_Name,X[j+1].Full_Name)>0)
                //если ФИО j студ. в алфавите располож. дальше ФИО j+1 студента
                {d=X[j]; //меняем местами сведения о студентах
                    X[j]=X[j+1];
                    X[j+1]=d;
                }
}
void print_spisok(Student X[], int n) /*заголовок функции
    вывода сведений о списке студентов на экран*/
{int i;
    for (i=0; i<n; i++)
        printf("%s %d/%d/%d %s\n", X[i].Full_Name,
            X[i].Date.Day, X[i].Date.Month, X[i].Date.Year,
            X[i].Group);
}
int main() {
    Student Spisok[30];
    vvod_spisok(Spisok,30);
        //вызов функции vvod_spisok для ввода информации о 30 студентах
    printf("Неотсортированный массив студентов\n");
    print_spisok(Spisok,30); /*вызов функции print_spisok для
        вывода неотсортированной информации о 30 студентах*/
    sort_spisok(Spisok,30);
        //сортировка списка студентов в алфавитном порядке
    printf("Отсортированный массив студентов\n");
    print_spisok(Spisok,30); /*вызов функции print_spisok
        для вывода отсортированной информации о 30 студентах*/
    getchar();
    return 0;
}

```

14.2. Динамические массивы структур (только для И-32,33)

Если в программе возникает необходимость использовать массив структур, количество элементов в котором заранее неизвестно, то определить его статическим способом невозможно, так как компилятору неизвестен размер области, которую необходимо зарезервировать. То есть при использовании массивов, количество структур в которых заранее неизвестно, приходится пользоваться динамически распределяемой памятью, а полученные таким образом массивы называются *динамическими массивами структур*.

Объявление динамических массивов структур. Динамические массивы структур объявляются следующим образом:

```
<структурный_тип_элементов_массива> *<имя_массив>'
```

Здесь <структурный_тип_элементов_массива> может быть любым допустимым в языке C++ структурным типом;

<имя_массива> — является указателем, т. е. может содержать адрес некоторой области памяти, в которой может размещаться массив структур.

Выделение и освобождение памяти для динамической матрицы. Для выделения памяти под динамический массив структур надо зарезервировать память с помощью оператора:

```
<имя_массива> = new <структурный_тип_элементов_массива*> [<кол-во_элементов_массива>];
```

После выделения памяти с динамическим массивом структур можно работать точно так же, как с обычным массивом структур, т. е. можно использовать все алгоритмы и методы, описанные выше.

Когда память, выделенная под динамическую матрицу, больше не нужна, ее следует освободить с помощью оператора:

```
delete []имя_динамического_массива_структур;
```

Задача 1. Описать структуру с именем Data, содержащую поля:

- Day — день;
- Month — месяц;
- Year — год

и структуру Anketa, содержащую следующие поля:

- Fam — фамилию туриста;
- Name — имя туриста;
- Otch — отчество туриста;
- Date — дата рождения типа Data;
- Gorod — город, где проживает турист.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в динамический массив A, число элементов которого k вводится с клавиатуры;
- вывод на экран информации о всех туристах.

```
#include <stdio.h>
struct Data{                //начало объявления структурного типа Data
    int Day;                //объявление поля Day
    int Month;              //объявление поля Month
    int Year;               //объявление поля Year
};                          //конец описания структурного типа Data
struct Anketa {            //начало объявления структурного типа Anketa
    char Fam[20];           //фамилия
    char Name[15];          //имя
    char Otch[20];          //отчество
    Data Date;              //дата рождения
    char Gorod;             //город
};                          //конец описания структурного типа Anketa
void vvod_turist(Anketa t)  /*заголовок функции ввода с
                             клавиатуры значений одной переменной типа структура Anketa*/
{ printf("Введите фамилию туриста  ");
  gets(t.Fam);
  printf("Введите имя туриста  ");
  gets(t.Name);
  printf("Введите отчество туриста  ");
  gets(t.Otch);
  printf("Введите день рождения туриста ");
  scanf("%d",t.Date.Day);
  printf("Введите месяц рождения студента ");
  scanf("%d",t.Date.Month);
  printf("Введите год рождения туриста ");
  scanf("%d",t.Date.Year);
  printf("Введите город, откуда приехал турист ");
  gets(t.Gorod);
  getchar();
}
```

```

void vvod_spisok(Anketa *&X, int &n) /*заголовок функции ввода
сведений о группе туристов, число которых вводится с клавиатуры*/
{int i;
printf("Введите число туристов ");
scanf("%d",&n);
x=new Anketa[n]; //выделение памяти под массив структур Anketa
for (i=0; i<n;i++){
{printf("ввод сведений о %d туристе\n",i);
//вывод информации о номере туриста
vvod_turist(X[i]);
//вызов функции vvod_turist для ввода сведений об i-м туристе
}
}
void print_spisok(Anketa X[], int n)
//заголовок функции вывода сведений о списке туристов на экран
{int i;
for (i=0; i<n; i++)
printf("%s %s %s %d/%d/%d %s\n", X[i].Fam, X[i].Name,
X[i].Otch, X[i].Date.Day, X[i].Date.Month,
X[i].Date.Year, X[i].Gorod);
}
int main() {
Anketa *A=NULL; //указатель на динамический массив структ. типа
int k=0; //объявление целой переменной k — число туристов
vvod_spisok(A,k); /*вызов функции vvod_spisok для ввода
динамического массива сведений о k туристах*/
print_spisok(A,k); /*вызов функции print_spisok для вывода
динамического массива сведений о k туристах*/
delete []A;
A=NULL;
getchar();
return 0;
}

```

14.3. Динамические структуры данных (списки)

Динамическая память позволяет реализовать широко используемую организацию данных в виде списков. На рис. 14.1 показан однонаправленный список.

Список состоит из звеньев. Каждое звено списка имеет информационное поле и указатель на следующее звено (адрес следующего звена). В указатель последнего звена списка записывается NULL (пустое



Рис. 14.1. Однонаправленный (односвязный) список

значение) — это является признаком конца списка. Тип каждого звена однонаправленного списка определяется следующим образом:

```
Struct <имя_типа_звена_списка>
{<тип_1>      <имена_полей_информационной_части_1> ;
  <тип_2>      <имена_полей_информационной_части_2> ;
  ...
  <тип_k>      <имена_полей_информационной_части_k>;
  <имя_типа_указателя_на_звено_списка>
      <имя_указателя_на_следующее_звено>;
};
```

где <имя_типа...>, <имена_полей...>, <имя_указателя...> — идентификаторы пользователя;

<тип_звена> — структурный тип, определяемый пользователем; одним из полей в структуре должен быть <указатель на следующее звено списка>; количество информационных полей не ограничено;

<тип_указателя_на_звено_списка> — ссылочный тип, базовым для него является <тип_звена>.

Например, тип списка с целой информационной частью звеньев задается следующим образом:

```
struct Spisok{                               //структурный тип "звено списка"
    int Elem;                                //информационное поле звена
    Spisok *Next; //указатель на следующее звено списка
};
```

Базовым типом указателя на звено является тип звена, который, в свою очередь, содержит в адресной части звена (указателе на следующее звено) тип указателя на звено.

Использование указателей в такой организации данных позволяет достаточно просто вносить в нее изменения. Так, например, легко вставить новое звено на любое место, расширить список, удалить звено с любого места и т. д. Подобные операции для элементов массива могут встретить огромные трудности.

14.4. Формирование списков

Списки могут быть сформированы разными способами, наиболее часто используются следующие два способа:

- по принципу стека (звенья в списке будут следовать в порядке, обратном порядку их поступления);
- по принципу «очереди» (звенья в списке будут следовать в порядке их поступления).

Пример программы формирования списка по принципу стека (рис. 14.2)

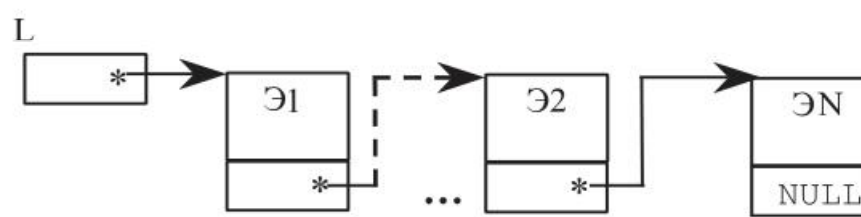


Рис. 14.2. Список, сформированный по принципу стека

```
#include <stdio.h>
struct Spisok{                               //структурный тип "звено списка"
    int Elem;                                //информационное поле звена
    Spisok *Next; //указатель на следующее звено списка
};
void create_stack(Spisok *&L1)
    //заголовок функции создания списка по принципу стека
{
    Spisok *x;
    char sym;
    x=NULL; //присвоить x пустое значение (создание конца списка)
    do{                                           //начало цикла с постусловием
        L1=new Spisok; //выделение памяти под очередное звено списка
        printf("Введите информационную часть звена списка ");
        scanf("%d",&L1->Elem);
        //ввод информационной части звена списка
        L1->Next=x; //запись в адресную часть очередного звена списка
        //адреса предыдущего звена списка или признака конца списка*/
        x=L1; //делаем очередное звено предыдущим для следующей итерации
        printf("Будете вводить еще (y/n)? ");
        sym=getchar();
    }while ((sym=='y') || (sym=='Y')); //конец цикла с постусловием
}
```

```

int main() {
    Spisok *L=NULL;           //объявление и инициализация указателя L
    create_stack(L);         //вызов функции create_stack для созд.списка L
    ...
    return 0;
}

```

Пояснение. В разделе описания типов объявлен структурный тип `Spisok`, содержащий следующие поля: `Elem` — элемент звена (информационное поле), `Next` — указатель на следующее звено. Формирование списка из двух звеньев по принципу стека (функция `create_stack`) показано на рис. 14.3.

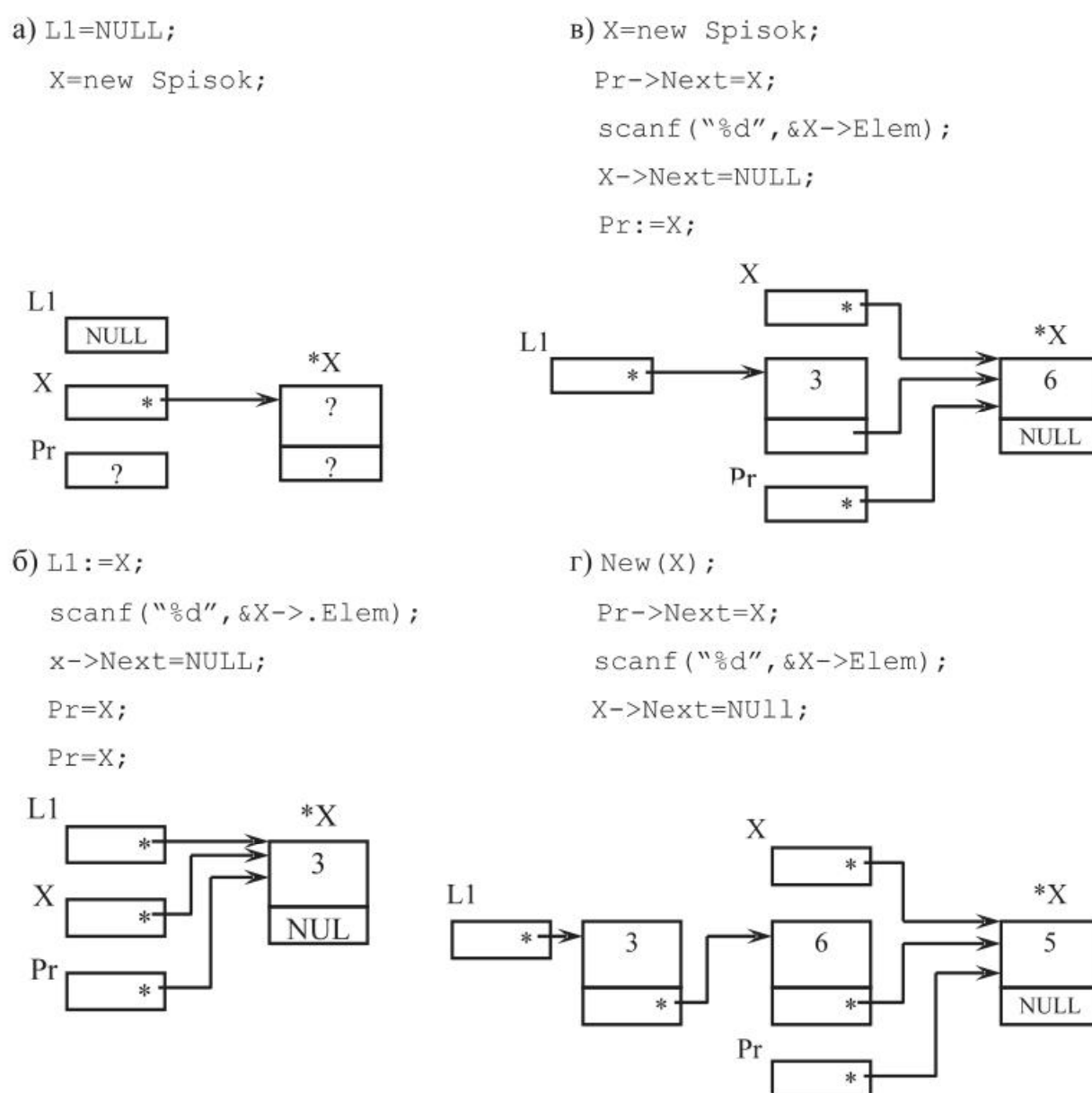


Рис. 14.3. Формирование списка по принципу стека

В функции `create_stack` вначале рабочая переменная `X` полагается равной `NULL`. С помощью процедуры `New` выделяется динамическая память под 1-е звено (переменная `L1` — указатель на 1-е звено)

и в поле `Elem` этого звена с клавиатуры вводится 1-й элемент. В поле `Next` 1-го звена копируется значение переменной `X`, т. е. в поле `Next` 1-го звена заносится указатель `NULL`. После этого в переменной `X` запоминается указатель на 1-е звено. Далее формируется 2-е звено (переменная `L1` — указатель на 2-е звено). В поле `Next` 2-го звена копируется значение переменной `X`, т. е. в это поле заносится указатель на 1-е звено. После этого в переменной `X` запоминается указатель на 2-е звено.

Этот процесс повторяется до тех пор, пока все звенья не будут сформированы. Таким образом, список будет содержать звенья, связанные между собой, причем последнее звено будет стоять в начале списка, а 1-е — в конце. В переменной `L1` будет находиться указатель на начало списка.

Пример программы формирования списка по принципу «очереди» (рис. 14.4)

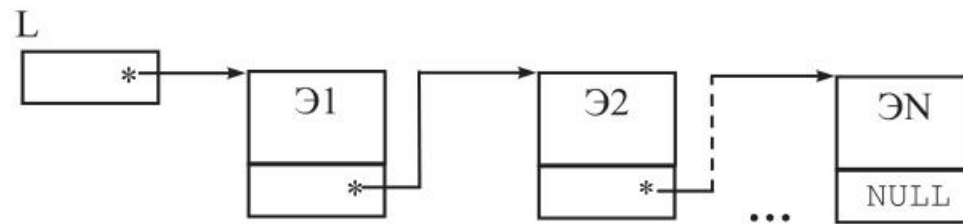


Рис. 14.4. Список, сформированный по принципу очереди

```
#include <stdio.h>
struct Spisok{                               //структурный тип "звено списка"
    int Elem;                                //информационное поле звена
    Spisok *Next; //указатель на следующее звено списка
};
void create_ochered(Spisok *&L1)
    //заголовок функции создания списка по принципу очереди
{Spisok *x, *pr;
char sym;
L1=NULL; //присвоить L1 пустое значение (признак отсут. списка)
do{                                           //начало цикла с постусловием
    x=new Spisok; //выделение памяти под очередное звено списка
    if (L1==NULL) //если начало списка еще не создано
        L1=x; //записываем в начало списка только что созданное звено
    else pr->next=x; /*запись в адресную часть предыдущего звена
        указателя на только что созданное звено*/
    printf("Введите информационную часть звена списка ");
    scanf("%d",&x->Elem);
    //ввод информационной части звена списка
    x->Next=NULL; /*запись пустого значения в адресную часть
        очередного звена списка, так как оно может быть последним*/
}
```



```

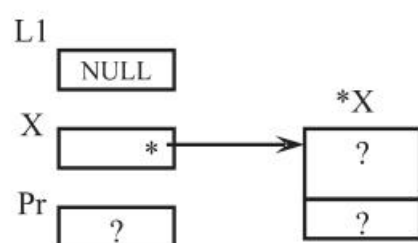
pr=x;      //делаем очередное звено предыдущим для следующей итерации
printf("Будете вводить еще (y/n)?  ");
sym=getchar();
} while ((sym=='y')||(sym=='Y')); //конец цикла с постусловием
}
int main() {
Spisok *L=NULL;      //объявление и инициализация указателя L
create_ochered(L);    //вызов функ. create_ochered
                      // для созд. списка L

...
return 0;
}

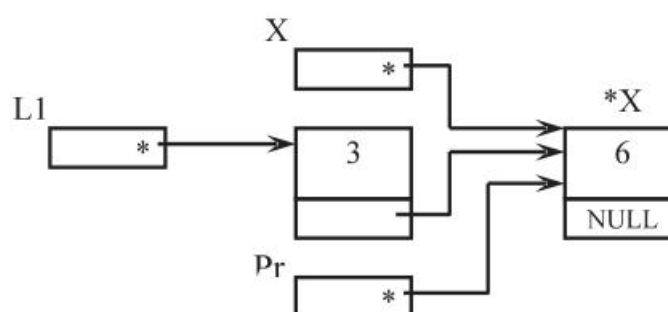
```

Пояснение. В разделе описания типов объявлен тип `Spisok`, содержащий следующие поля: `Elem` — элемент звена (информационное поле), `Next` — указатель на следующее звено. Формирование списка из трех звеньев по принципу «очереди» (рис. 14.5).

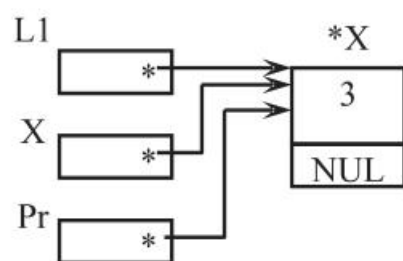
a) `L1=NULL;`
`X=new Spisok;`



b) `X=new Spisok;`
`Pr->Next=X;`
`scanf("%d",&X->Elem);`
`X->Next=NULL;`
`Pr:=X;`



б) `L1:=X;`
`scanf("%d",&X->.Elem);`
`x->Next=NULL;`
`Pr=X;`
`Pr=X;`



г) `New(X);`
`Pr->Next=X;`
`scanf("%d",&X->Elem);`
`X->Next=NULL;`

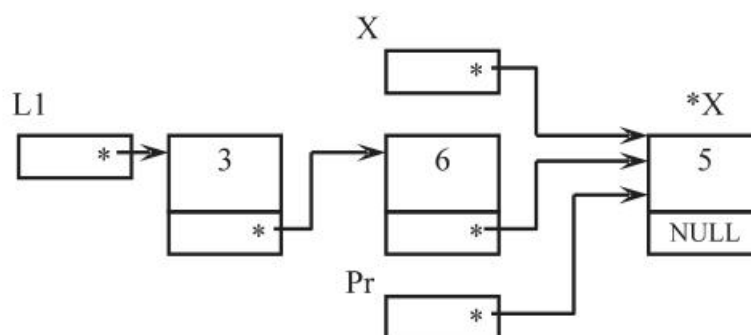


Рис. 14.5. Формирование списка по принципу «очереди»

В функции `create_ochered` вначале переменная `L1` полагается равной `NULL`, и выделяется динамическая память под 1-е звено (переменная `X` — указатель на 1-е звено). Далее, если `L1=NULL`, в `L1` заносится указатель на 1-е звено, т. е. в `L1` будет находиться указатель на начало списка. В поле `Elem` этого звена с клавиатуры вводится 1-й элемент, в поле `Next` этого звена заносится значение `NULL`. В переменной `Pr` запоминается указатель на 1-е звено.

После этого выделяется память под 2-е звено (переменная `X` — указатель на 2-е звено). В поле `Next` 1-го звена копируется значение переменной `X`, т. е. в это поле заносится указатель на 2-е звено. Затем заполняется поле `Elem` 2-го звена, а в поле `Next` 2-го звена заносится значение `NULL`. После этого в переменной `Pr` запоминается указатель на 2-е звено.

Аналогично формируется 3-е звено. Таким образом, список будет содержать звенья, связанные между собой, причем 1-е звено будет стоять в начале списка, а последнее — в конце. В переменной `L1` будет находиться указатель на начало списка.

Примечание. После того как список сформирован, для дальнейшей работы с ним способ его формирования (стек или очередь) не имеет никакого значения. Поэтому все последующие задачи будут рассматриваться вне зависимости от способа создания списка, за исключением тех случаев, где это будет указано в условии.

Пример программы вывода списка на экран.

```
#include <stdio.h>
struct Spisok{                               //структурный тип "звено списка"
    int Elem;                                //информационное поле звена
    Spisok *Next;                            //указатель на следующее звено списка
};
void create_spisok(Spisok *&L1)
...
void print_spisok(Spisok *L1) //заголовок функции вывода списка
{Spisok *p;
  P=L1;                                     //записать в p указатель на начало списка
  while (p!=NULL){                          //цикл, пока не достигнут признак конца списка
    printf("%d ",p->Elem);
                                           //вывод информационной части очередного звена списка
    p=p->Next;                              //переход к следующему звену списка
  }
  printf("\n");
}
```

```
int main() {
    Spisok *L=NULL;
    create_spisok(L);
    printf("Вывод исходного списка\n");
    print_spisok(L);
    ...
    return 0;
}
```

14.5. Решение задач (работа со списками)

Задача 1. Написать программу, включающую в себя подпрограммы формирования линейного списка по принципу стека, поиска в списке L звена с 1-м вхождением элемента E, если такой есть, вывода списка на экран. Значения информационных полей и искомый элемент E — значения типа int.

```
#include <stdio.h>
struct Spisok{                               //структурный тип "звено списка"
    int Elem;                                //информационное поле звена
    Spisok *Next; //указатель на следующее звено списка
};

void create_stack(Spisok *&L1)
    //заголовок функции создания списка по принципу стека
{
    Spisok *x;
    char sym;
    x=NULL; //присвоить x пустое значение (создание конца списка)
    do{                                           //начало цикла с постусловием
        L1=new Spisok; //выделение памяти под очередное звено списка
        printf("Введите информационную часть звена списка ");
        scanf("%d",&L1->Elem); //ввод информационной части звена списка
        L1->Next=x; //запись в адресную часть очередного звена списка
        //адреса предыдущего звена списка или признака конца списка*/
        x=L1; //делаем очередное звено предыдущим для следующей итерации
        printf("Будете вводить еще (y/n)? ");
        sym=getchar();
    }while ((sym=='y')||(sym=='Y')); //конец цикла с постусловием
}

void print_spisok(Spisok *L1) //заголовок функции вывода списка
{
    Spisok *p;
    p=L1; //записать в p указатель на начало списка
    while (p!=NULL){ //цикл, пока не достигнут признак конца списка
```



```

printf("%d ", p->Elem);
//вывод информационной части очередного звена списка
p=p->Next; //переход к следующему звену списка
}
printf("\n");
}
void search(Spisok *L1, int E1)
//заголовок функции поиска элемента E1 в списке L1
{
    Spisok *p;
    Bool flag;
    p=L1; //записать в p указатель на начало списка
    flag=FALSE; //установить признак «элемент в списке отсутствует»
    while ((p!=NULL)&&!flag) /*цикл, пока не достигнут конец списка
                             не найден искомый элемент*/
    {
        if (p->Elem==E1) //если информ.часть очередного элемента равна E1
        {
            flag=TRUE; //установить признак «элемент в списке найден»
        }
        else p=p->Next; //в противном случае переход к следующ. элементу
    }
    if (flag)
        printf("элемент %d в списке найден\n", E1);
    else printf("элемент %d в списке не найден\n", E1);
}

int main() {
    Spisok *L=NULL;
    int E;
    create_stack(L);
    printf("Исходный список\n");
    print_spisok(L);
    printf("Введите искомый элемент ");
    scanf("%d",&E);
    search(L,E);
    getchar();
    return 0;
}

```

Пояснение. В разделе описания типов объявлен тип `Spisok`, содержащий следующие поля: `Elem` — элемент звена (информационное поле), `Next` — указатель на следующее звено. Функции `create_stack` (формирование списка по принципу стека) и `print_spisok` (вывод списка на экран) были подробно рассмотрены выше. В процедуре `Search` осуществляется последовательный перебор элементов списка, пока не будет найден искомый элемент или не достигнут конец списка (линейный поиск). В конце процедуры на экран выводится соответствующее сообщение.

Задача 2. Написать программу, включающую в себя подпрограммы формирования линейного списка по принципу «очереди», поиска максимального элемента списка L, вывода списка на экран. Значения информационных полей — типа int.

```
#include <stdio.h>
struct Spisok{                                //структурный тип "звено списка?
    int Elem;                                //информационное поле звена
    Spisok *Next; //указатель на следующее звено списка
};
void create_ochered(Spisok *&L1)
    //заголовок функции создания списка по принципу «очереди»
{Spisok *x, *pr;
char sym;
L1=NULL; //присвоить L1 пустое значение (признак отсут. списка)
do{                                           //начало цикла с постусловием
    x=new Spisok; //выделение памяти под очередное звено списка
    if (L1==NULL) //если начало списка еще не создано
        L1=x; //записываем в начало списка только что созданное звено
    else pr->next=x; //запись в адресную часть предыдущего звена
        //указателя на только что созданное звено*/
    printf("Введите информационную часть звена списка ");
    scanf("%d",&x->Elem); //ввод информационной части звена списка
    x->Next=NULL; //запись пустого значения в адресную часть
        //очередного звена списка, так как оно может быть последним*/

    pr=x; //делаем очередное звено предыдущим для следующей итерации
    printf("Будете вводить еще (y/n)? ");
    sym=getchar();
}while ((sym=='y') || (sym=='Y')); //конец цикла с постусловием
}
void print_spisok(Spisok *L1) //заголовок функции вывода списка
{Spisok *p;
p=L1; //записать в p указатель на начало списка
while (p!=NULL){ //цикл, пока не достигнут признак конца списка
    printf("%d ",p->Elem);
        //вывод информационной части очередного звена списка
    p=p->Next; //переход к следующему звену списка
}
printf("\n");
}
int Max_Spisok(Spisok *L1) //заголовок функции нахождения
    //максимального элемента списка*/
{Spisok *p;
int Max1;
p=L1; //записать в p указатель на начало списка
```

```

Max1=p->Elem;           //задание начального значения предполагаемому
                        //максимуму
While (p!=NULL){        //в цикле, пока не конец списка
    if (p->Elem>Max1)     /*если информационная часть очередного звена
                        больше предполагаемого максимума*/
        Max1=p->Elem;    //меняем предполагаемый максимум
        p=p->Next;       //переход к следующему звену списка
    }
    return Max1;         /*возврат вычисленного значения Max1
                        в вызывающую функцию*/
}

int main() {
    Spisok *L=NULL;
    int max;
    create_ochered(L);
    printf("Исходный список\n");
    print_spisok(L);
    max=Max_Spisok(L);
    printf("Максимальный элемент списка = %d\n", max);
    getchar();
    return 0;
}

```

Пояснение. Функции `create_ochered` и `print_spisok` (формирование списка по принципу «очереди») рассмотрены выше. В функции `Max_Spisok` осуществляется последовательный перебор элементов списка и сравнение их с предполагаемым максимумом (`Max1`), до тех пор пока не будет достигнут конец списка. В конце функции ее имени присваивается найденное значение максимума.

Задача 3. Вставить звено с информационной частью `E1` в однонаправленный список после 1-го вхождения звена с информационной частью `E`.

```

#include <stdio.h>
struct Spisok{           //структурный тип "звено списка"
    int Elem;            //информационное поле звена
    Spisok *Next;       //указатель на следующее звено списка
};

void create_ochered(Spisok *&L1)
    //заголовок функции создания списка по принципу «очереди»
{
    Spisok *x, *pr;
    char sym;
    L1=NULL;             //присвоить L1 пустое значение (признак отсут. списка)
}

```



```

do{                                     //начало цикла с постусловием
    x=new Spisok;                       //выделение памяти под очередное звено списка
    if (L1==NULL)                       //если начало списка еще не создано
        L1=x;                          //записываем в начало списка только что созданное звено
    else pr->next=x;                    /*запись в адресную часть предыдущего звена
                                       указателя на только что созданное звено*/
    printf("Введите информационную часть звена списка ");
    scanf("%d",&x->Elem); //ввод информационной части звена списка
    x->Next=NULL;                /*запись пустого значения в адресную часть
                                очередного звена списка, так как оно может быть последним*/

    pr=x;                        //делаем очередное звено предыдущим для следующей итерации
    printf("Будете вводить еще (y/n)? ");
    sym=getchar();
}while ((sym=='y')||(sym=='Y')); //конец цикла с постусловием
}

void print_spisok(Spisok *L1) //заголовок функции вывода списка
{Spisok *p;
p=L1;                                //записать в p указатель на начало списка
while (p!=NULL){                    //цикл, пока не достигнут признак конца списка
    printf("%d ",p->Elem);
                                //вывод информационной части очередного звена списка
    p=p->Next;                    //переход к следующему звену списка
}
printf("\n");
}

void insert_spisok(Spisok *L1, int E, int E1) /*заголовок
                                              функции вставки звена с информационной частью E1
                                              после 1-го вхождения звена с информационной частью E*/
{Spisok *p, *X;
p=L1;                                //записать в p указатель на начало списка
while ((p!=NULL)&&(p->Elem!=E))        /*в цикле, пока не закончился
                                      список или не найдено звено с информационной частью, равной E*/
    p=p->Next;                        //переход к следующему звену
if (p->Elem==E){                      //если звено с информационной частью E найдено
    X=new Spisok;                     //выделяем память под новое звено списка
    X->Elem=E1;                       //записываем E1 в информационную часть нового звена
    X->Next=p->Next;                  /*записываем в адресную часть нового звена
                                      указатель на звено, следующее за звеном с информационной
                                      частью, равной E*/
    p->Next=X;                       /*записываем в адресную часть звена с информационной
                                      частью, равной E, указатель на новое звено*/
}
}

```

```

else printf("Элемента с информационной частью %d в
списке нет\n",E);
}
int main() {
    Spisok *L;
    int E, E1;
    create_ochered(L);
    printf("Исходный список\n");
    print_spisok(L);
    printf("Введите E и E1\n");
    scanf("%d%d",&E,&E1);
    insert_spisok(L,E,E1);
    printf("Результирующий список\n");
    print_spisok(L);
    getchar();
    return 0;
}

```

Пояснение. В функции `insert_spisok` осуществляется вставка нового звена с информационной частью `E1`. Сначала просматривается список и определяется место вставки. В переменной `p` будет находиться указатель на звено, после которого произойдет вставка, т. е. поле `Elem` этого звена равно `E`. Потом формируется новое звено `X`, в поле `Elem` которого записывается `E1`. В поле `Next` звена `X` копируется значение поля `Next` звена `p`, а в поле `Next` звена `p` копируется указатель на звено `X`. Таким образом, произойдет вставка звена с информационной частью `E1` после звена с информационной частью `E`.

Задача 4. Сформировать однонаправленный список. Удалить 1-е вхождение звена с элементом `E`.

1-й способ:

```

#include <stdio.h>
struct Spisok{                                //структурный тип "звено списка"
    int Elem;                                //информационное поле звена
    Spisok *Next; //указатель на следующее звено списка
};
void create_stack(Spisok *&L1)
    //заголовок функции создания списка по принципу стека
{
    Spisok *x;
    char sym;
    x=NULL; //присвоить x пустое значение (создание конца списка)
    do{                                         //начало цикла с постусловием
        L1=new Spisok; //выделение памяти под очередное звено списка
        printf("Введите информационную часть звена списка ");
    }
}

```

```

scanf ("%d", &L1->Elem); //ввод информационной части звена списка
L1->Next=x; /*запись в адресную часть очередного звена списка
адреса предыдущего звена списка или признака конца списка*/
x=L1; //делаем очередное звено предыдущим для следующей итерации
printf("Будете вводить еще (y/n)? ");
sym=getchar();
}while ((sym=='y') || (sym=='Y')); //конец цикла с постусловием
}
void delete_spisok(Spisok * &L, int E) /*заголовок функции
удаления из списка звена с информационной частью, равной E*/
{Spisok *p, *x;
if (L1->Elem==E) /*если информационная часть
первого звена списка равна E удалим его*/
{x=L1; //запишем в p указатель на старое начало списка
L1=L1->Next; //сдвинем начало списка на второй элемент списка
delete x; //удаляем старое начало списка
x=NULL;
}
else{ //информационная часть первого звена списка не равна E
p=L1; //запишем в p указатель на начало списка
while ((p->Next!=NULL) && (p->Next->Elem!=E)) /*в цикле
пока не найдено последнее звено списка или не найдено звено,
предшествующее звену с информационной частью, равной E*/
p=p->Next; //переход к следующему звену
if (p->Next!=NULL) { /*если найдено звено, предшествующее
звену с информационной частью, равной E*/
x=p->Next; /*запишем в x указатель на звено
с информационной частью, равной E*/
p->Next=x->Next; /*запишем в адресную часть звена,
предшествующего звену с информационной частью, равной E,
значение адресной части звена с E в информационной части*/
delete x; //удаляем звено с информационной частью, равной E
}
else printf("Элемент %d в списке не найден\n");
}
}
int main() {
Spisok *L;
int E;
create_stack(L);
printf("Исходный список\n");
print_spisok(L);
printf("Введите E >");
scanf ("%d", &E);
delete_spisok(L, E);
}

```



```
printf("Результирующий список\n");
print_spisok(L);
getchar();
return 0;
}
```

Пояснение. В функции `delete_spisok` осуществляется удаление звена с информационной частью `E`. При удалении звена рассматриваются 3 случая: 1) удаление 1-го звена; 2) удаление среднего или последнего звена; 4) элемент `E` не найден.

Сначала проверяется информационное поле 1-го элемента. Если оно равно `E`, то необходимо удалить 1-й элемент. Для этого в указатель `X` записывается адрес 1-го звена (`X=L1;`), в указатель начала списка записывается адрес 2-го звена (`L1=L1->Next;`), после чего 2-й элемент становится 1-м. Далее освобождается память, занимаемая 1-м элементом (`delete x;`) и происходит выход из функции.

В противном случае (информационное поле 1-го элемента не равно `E`) необходимо провести поиск в списке элемента с полем, равным `E`. Для этого в цикле (оператор `while`) последовательно перебираются элементы списка, пока не закончится список или пока не будет найден элемент, предшествующий удаляемому (элементу с информационным полем, равным `E`). Если элемент найден, то в указатель `X` записывается адрес следующего за текущим (удаляемого) элемента, в адресную часть указателя `P` — адрес следующего за удаляемым элементом и освобождается память, занимаемая удаляемым звеном (`delete x;`). Если же элемент найти не удалось (`P->next=NULL`), на экран выводится сообщение «... не найден». Таким образом, произойдет удаление 1-го вхождения звена с информационной частью `E`.

2-й способ (с использованием рекурсии):

```
#include <stdio.h>
struct Spisok{                               //структурный тип "звено списка"
    int Elem;                                //информационное поле звена
    Spisok *Next; //указатель на следующее звено списка
};

void create_stack(Spisok *&L1)
    //заголовок функции создания списка по принципу стека
{
    Spisok *x;
    char sym;
    x=NULL;                                  //присвоить x пустое значение (создание конца списка)
    do{                                       //начало цикла с постусловием
        L1=new Spisok;                       //выделение памяти под очередное звено списка
    }while(1);
```

```

printf("Введите информационную часть звена списка ");
scanf("%d",&L1->Elem); //ввод информационной части звена списка
L1->Next=x; /*запись в адресную часть очередного звена списка
адреса предыдущего звена списка или признака конца списка*/
x=L1; //делаем очередное звено предыдущим для следующей итерации
printf("Будете вводить еще (y/n)? ");
sym=getchar();
}while ((sym=='y') || (sym=='Y')); //конец цикла с постусловием
}
void delete_spisok(Spisok *L, int E) /*заголовок функции
удаления из списка звена с информационной частью, равной E*/
{ if (L1!=NULL) { //если список не пуст
if (L1->Elem==E) /*если информационная часть
первого звена списка равна E, удалим его*/
{x=L1; //запишем в p указатель на старое начало списка
L1=L1->Next; //сдвинем начало списка на второй элемент списка
delete x; //удаляем старое начало списка
x=NULL;
}
else delete_spisok(L1->Next,E); /*рекурсивный вызов функции
с параметром — следующим звеном списка*/
} else printf("Элемент %d не найден \n",E);
}
int main() {
Spisok *L;
int E;
create_stack(L);
printf("Исходный список\n");
print_spisok(L);
printf("Введите E >");
scanf("%d",&E);
delete_spisok(L,E);
printf("Результирующий список\n");
print_spisok(L);
getchar();
return 0;
}

```

Пояснение. В рекурсивной функции `delete_spisok` осуществляется удаление звена с элементом `E`. Если адрес звена (`L1`), переданный в процедуру, не равен `NULL`, происходит сравнение элемента звена с `E`. При равенстве удаляется это звено и происходит выход из функции. Если информационная часть звена не равна `E`, осуществляется рекурсивный вызов процедуры для следующего звена списка

(`delete_spisok(L1->Next, E);`). Остановка рекурсии произойдет, когда будет найдено звено с элементом `E` или когда адрес, переданный в процедуру, станет равен `NULL` (после просмотра последнего звена списка в случае, если звена с элементом `E` нет в списке).

При использовании рекурсии требуется большое количество времени и памяти для повторных (рекурсивных) вызовов подпрограммы и запоминания рабочей информации при каждом вызове. Однако использование рекурсии сильно упрощает алгоритм решения задачи.

Тесты

1. Выберите правильный вариант объявления структуры из трех полей:

- a) `int shop[3];`
- б) `struct shop{
 char Tovar[20];
 double Price;
 int Nal;
};`
- в) `void shop[3];`

2. Как правильно ввести значения полей структуры с клавиатуры?

- a) ...
`printf("Название товара? ");
gets(Z.Tovar);
printf("Цена? ");
scanf ("%lf",&Z.Price);
printf("Наличие? (0/1) ");
scanf ("%d",&Z.Nal);
...`
- б) ...
`printf("Введите название товара, его цену и наличие
 (0/1)");
gets(Z);
...`
- в) ...
`printf("Название товара? ");
gets(Tovar);
printf("Цена? ");
scanf ("%lf",&Price);
printf("Наличие? (0/1) ");
scanf ("%d",&Nal);
...`

3. Что делает следующий фрагмент программы?

```
...
struct Zap{
double X,Y;
int Z;
char S;
};
...
{Zap A[30];
int M,i;
...
M=A[0].Z;
for (i=1; i<30; i++)
if (A[i].Z>M)
M=A[i].Z;
...
}
```

- а) ищет максимальный элемент массива A;
- б) ищет минимальный элемент массива A;
- в) ищет структуру с максимальным значением поля Z;
- г) ищет структуру с минимальным значением поля Z.

4. Что произойдет с массивом X после выполнения следующего фрагмента программы?

```
const int N=20;
struct Zapis{
int A;
char B,C;
char D[20];
};
Zapis X[N];
int Z,i,j;
...
for (i=1; i<N; i++)
for (j=i+1; j<N; j++)
if (X[i].A>X[j].A)
{Z=X[i].A;
X[i].A=X[j].A;
X[j].A=Z;
}
...
}
```

- а) массив X будет отсортирован в порядке убывания по полю A записей (элементов массива);

б) массив X будет отсортирован в порядке возрастания по полю A записей (элементов массива);

в) фрагмент содержит ошибки и работать не будет.

5. Выберите правильный вариант сравнения структур на равенство, если в программе объявлены следующие переменные:

```
Struct Z{
    double A,B;
    char C;
};
Z X,Y;
...
```

а) `if (X==Y)`

...

б) `if ((X.A==Y.A) && (X.B==Y.B) && (X.C==Y.C))`

...

в) `if ((X.A==Y.A) || (X.B==Y.B) || (X.C==Y.C))`

...

Задания

1. Сформировать массив, содержащий сведения о количестве изделий, собранных сборщиками цеха за неделю. Структурный тип содержит поля: фамилия сборщика, количество изделий, собранных им ежедневно в течение шестидневной недели, т. е. отдельно в понедельник, вторник и т. д. Написать программу, выдающую на печать:

- фамилию сборщика и общее количество деталей, собранных им за неделю;
- фамилию сборщика, собравшего наибольшее количество изделий, и день, когда он достиг наивысшей производительности труда.

2. Сформировать массив, содержащий сведения о количестве изделий категорий A , B , C , собранных рабочим за месяц. Структурный тип содержит поля: фамилия сборщика, наименование цеха, количество изделий по категориям, собранных рабочим за месяц.

Считая заданными значения расценок SA , SB , SC за выполненную работу по сборке единицы изделия категорий A , B , C , выдать на печать следующую информацию:

- общее количество изделий категорий A , B , C , собранных рабочим цеха;
- средний размер заработной платы рабочих цеха X .

3. Сформировать массив, содержащий сведения о телефонах абонентов. Структурный тип содержит поля: фамилия абонента, место жительства (название улицы, номер дома), год установки телефона. Написать программу, выдающую следующую информацию:

- номер телефона по вводимой с клавиатуры фамилии абонента;
- количество установленных телефонов с XXXX года;
- список номеров телефонов, принадлежащих жильцам определенного дома и улицы.

4. Сформировать массив, содержащий сведения об ассортименте игрушек в магазине. Структурный тип содержит поля: название игрушки, цена, количество, возрастные границы (2 — 5).

Написать программу, выдающую следующие сведения:

- название игрушек, которые подходят детям от 1 до 3 лет;
- стоимость самой дорогой игрушки и ее название;
- название игрушки, которая по стоимости не превышает X руб. и подходит ребенку в возрасте от A до B лет. Значения A , B , X вводятся с клавиатуры.

5. Сформировать массив, содержащий сведения о сдаче студентами сессии. Структурный тип содержит поля: индекс группы, фамилия студента, оценки по пяти экзаменам.

Написать программу, выдающую информацию:

- фамилии неуспевающих студентов с указанием индексов групп и количества задолженностей;
- средний балл, полученный каждым студентом группы X , и всей группы в целом.

6. Сформировать массив, содержащий сведения о личной коллекции книголюбца. Структурный тип содержит поля: шифр книги, автор, название, год издания, местоположение (номер стеллажа).

Написать программу, выдающую следующую информацию:

- местоположение книги автора X названия Y ;
- список книг автора Z , находящихся в коллекции;
- число книг издания XX года, имеющихся в библиотеке.

7. Сформировать массив, содержащий сведения о наличии билетов на рейсы Аэрофлота. Структурный тип содержит поля: номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест в салоне.

Написать программу, выдающую следующую информацию:

- время вылета самолетов в город X ;
- наличие свободных мест на рейс в город X с временем отправления Y .

8. Сформировать массив, содержащий сведения об ассортименте обуви в магазине фирмы. Структурный тип содержит поля: артикул, наименование, количество, стоимость одной пары. Артикул начинается с буквы Д — для дамской обуви, М — для мужской, П — для детской.

Написать программу, выдающую информацию:

- о наличии и стоимости обуви артикула X ;
- ассортиментный список дамской обуви с указанием наименования и имеющегося в наличии числа пар каждой модели.

9. Сформировать массив, содержащий сведения о нападающих команды «Спартак». Структурный тип содержит поля: имена нападающих, число заброшенных ими шайб, число сделанных голевых передач, заработанное штрафное время.

Написать программу, которая определяет по сумме очков (голы + передачи) четырех лучших игроков.

10. Сформировать массив, содержащий сведения о том, какие из пяти предлагаемых дисциплин по выбору желает изучать студент. Структурный тип содержит поля: фамилия студента, индекс группы, пять дисциплин, средний балл успеваемости. Выбираемая дисциплина отмечается символом 1, иначе — пробелом.

Написать программу, которая печатает список студентов, желающих прослушать дисциплину X . Если число желающих превышает 4 человека, то отобрать студентов, имеющих более высокий средний балл успеваемости.

11. Сформировать массив, содержащий сведения об отправлении поездов дальнего следования с Казанского вокзала. Структурный тип содержит поля: номер поезда, станция назначения, время отправления, время в пути, наличие билетов.

Написать программу, выдающую информацию:

- время отправления поездов в город X во временном интервале от A до B часов;
- наличие билетов на поезд с номером XXX .

12. Сформировать массив, содержащий сведения о сотрудниках института. Структурный тип содержит поля: фамилия работающего, название отдела, год рождения, стаж работы, должность, оклад.

Написать программу, которая позволяет получить информацию:

- список сотрудников пенсионного возраста на сегодняшний день с указанием стажа работы;
- средний стаж работающих в отделе X .

13. Сформировать массив, содержащий сведения о пациентах глазной клиники. Структурный тип содержит поля: фамилия пациента, пол, возраст, место проживания (город), диагноз.

Написать программу, которая выдает информацию:

- количество иногородних, прибывших в поликлинику;
- список пациентов старше X лет с диагнозом J .

14. Сформировать однонаправленный список по принципу стека. Найти количество нулевых элементов и сумму положительных элементов в этом списке. Вывести список на экран.

15. Сформировать однонаправленный список по принципу «очереди». Найти сумму элементов списка, превышающих число K (вводится с клавиатуры), и произведение отрицательных элементов списка, если таковые имеются.

16. Написать функцию, которая находит среднее арифметическое всех элементов (целые числа) непустого списка L , сформированного по принципу стека.

17. Написать функцию, которая заменяет в списке L все вхождения элемента $E1$ на $E2$. Список сформировать по принципу стека. Элементы списка, $E1$, $E2$ — типа Char.

18. Сформировать непустой список L по принципу стека. После каждого вхождения элемента E удалить из списка один элемент, если такой есть и он отличен от E . Вывести список до и после изменений. Использовать подпрограммы.

19. Сформировать непустой список L по принципу стека. Удалить из списка L все вхождения элемента E . Вывести список до и после изменений. Использовать подпрограммы.

20. Сформировать непустой список L . Оставить в списке только первые вхождения одинаковых элементов. Вывести список до и после изменений. Использовать подпрограммы.

21. Сформировать непустой список L . Упорядочить элементы списка по возрастанию.

22. Сформировать линейный однонаправленный список по принципу стека. Вывести элементы списка на экран в обратном порядке.

23. Сформировать линейный однонаправленный список по принципу «очереди». Подсчитать число вхождений элемента E в список.

24. Создать список $L1$ по принципу стека, список $L2$ по принципу «очереди». Вывести их на экран. Найти сумму положительных, количество нулевых, произведение отрицательных элементов списка $L2$ и максимальный элемент списка $L1$. Определить, находится ли элемент с заданным номером списка $L1$ в списке $L2$.

25. Создать список. Вывести его на экран. Удалить из списка все элементы с четными значениями. Вывести на экран полученный список.

26. Создать список. Вывести его на экран. Удалить из списка все элементы с нечетными значениями. Вывести на экран полученный список.

27. Создать список. Вывести его на экран. Удалить из списка все элементы со значением, равным заданному. Вывести на экран полученный список.

28. Создать список. Вывести его на экран. Удалить из списка все элементы со значением меньшим заданного. Вывести на экран полученный список.

29. Создать список. Вывести его на экран. Удалить из списка все элементы со значением, не равным заданному. Вывести на экран полученный список.

30. Создать список. Вывести его на экран. Удалить из списка все элементы со значением большим заданного. Вывести на экран полученный список.

31. Создать список. Вывести его на экран. Вставить значение $-32\,768$ после 1-го элемента списка. Вывести полученный список на экран.

32. Создать список. Вывести его на экран. Вставить значение $30\,000$ перед 1-м элементом списка. Вывести полученный список на экран.

33. Создать список. Вывести его на экран. Вставить значение 9999 после последнего элемента списка. Вывести полученный список на экран.

34. Создать список. Вывести его на экран. Вставить значение 7777 перед последним элементом списка. Вывести полученный список на экран.

35. Создать список. Вывести его на экран. Вставить значение P , введенное с клавиатуры, после элемента с номером NUM . NUM вводится с клавиатуры. Вывести полученный список на экран.

36. Создать список. Вывести его на экран. Вставить значение P , введенное с клавиатуры, перед элементом с номером NUM . NUM вводится с клавиатуры. Вывести полученный список на экран.

37. Создать список. Вывести его на экран. Вставить значение P , введенное с клавиатуры, после элемента со значением Q . Q вводится с клавиатуры. Вывести полученный список на экран.

38. Создать список. Вывести его на экран. Вставить значение P , введенное с клавиатуры, перед элементом со значением Q . Q вводится с клавиатуры. Вывести полученный список на экран.

Контрольные вопросы

1. Для чего в программировании используется комбинированный тип данных? Приведите примеры.
2. Что такое «структура» в программировании с точки зрения типов данных?

3. Каким образом можно объявить в программе переменную комбинированного типа? Приведите примеры.
4. Какие требования предъявляются к именам полей структур? Поясните на примерах.
5. Каких типов могут быть поля структур в C++? Приведите примеры.
6. Можно ли в программе использовать массив структур? В каких задачах это может потребоваться? Приведите примеры.
7. Как правильно присваивать значения полям структуры? Поясните на примерах.
8. Можно ли вводить значения полей структуры с клавиатуры? Приведите примеры.
9. Как правильно сравнить между собой две структуры? Поясните на примерах.
10. Что такое линейный однонаправленный список? Приведите примеры.
11. Каким образом можно сформировать линейный однонаправленный список?
12. Чем список, сформированный по принципу стека, отличается от списка, сформированного по принципу «очереди»? Поясните на примерах.
13. Как формируется линейный однонаправленный список по принципу стека? Поясните на примере.
14. Как формируется линейный однонаправленный список по принципу «очереди»? Поясните на примерах.
15. Чем отличается вывод на экран элементов линейного однонаправленного списка, сформированного по принципу стека, от списка, сформированного по принципу «очереди»?
16. Объясните, как происходит удаление элемента E из линейного однонаправленного списка.
17. Как в линейном однонаправленном списке найти элемент с заданным значением?

Глава 15

КЛАССЫ

В настоящее время при создании прикладного программного обеспечения используется объектно-ориентированное программирование (ООП). Оно позволяет эффективным образом проектировать, создавать, использовать и модифицировать сложные программы. При использовании объектно-ориентированного программирования по сравнению со структурным (процедурным) программированием в несколько раз уменьшается количество ошибок, резко уменьшается сложность разрабатываемых программ, сокращаются сроки их разработки и внедрения, а также повышается эффективность и быстродействие самих программ.

15.1. Объектно-ориентированный подход к программированию. Инкапсуляция

Понятие объектно-ориентированной программы. Развитие ООП обусловлено ограниченностью других методов программирования, разработанных ранее. Проблема структурного (процедурного) подхода заключается в том, что число возможных связей между данными и функциями может быть очень велико. Вследствие этого усложняется структура программы и в нее становится трудно вносить изменения.

Основной особенностью объектно-ориентированного подхода является объединение данных и действий, производимых над этими данными, в единое целое, которое называется *объектом*. Объектно-ориентированная программа представляет собой совокупность объектов и способов их взаимодействия. Например, для многооконного приложения (текстового редактора Word) объектами являются

окна редактирования, диалоговые окна, раскрывающиеся списки, кнопки и т. д. Совокупность этих объектов образует структуру программы, а способы функционирования объектов и способы их взаимодействия определяют работу программы.

Понятие объекта. Принцип инкапсуляции. Под *объектом* понимают совокупность данных, характеризующих этот объект, и методов (способов) обработки и использования данных объекта.

Пользователь и другие объекты программы должны иметь возможность читать данные объекта, обрабатывать их и записывать в объект новые значения данных. Однако если пользователь будет иметь свободный доступ к данным объекта, он может внести в объект неверные данные, и объект начнет функционировать с ошибками. Для безошибочного функционирования программы пользователям должен быть доступен только *пользовательский интерфейс* — описание имеющихся данных и методов объекта и способов использования этих данных и методов. А внутренняя реализация данных и методов объекта — дело разработчика объекта. При таком подходе разработчик может в любой момент модернизировать объект, изменить структуру хранения и форму представления данных, не изменяя пользовательский интерфейс, а пользователи смогут так же успешно и безошибочно работать с этим объектом.

Для обеспечения целостности и непротиворечивости данных объекта и изоляции их от внешнего воздействия в объектно-ориентированном программировании используется принцип *инкапсуляции (сокрытия)* данных. Сущность принципа инкапсуляции заключается в том, что все данные и некоторые методы объекта должны быть недоступными для пользователя.

15.2. Класс как тип данных

Понятие класса. Класс — это производный структурированный тип, введенный программистом на основе уже существующих типов. Класс можно рассматривать как расширение понятия структуры. Он, как и структура, представляет собой абстрактный тип данных (АТД), с помощью которого можно задавать набор функций для работы с ними. Данные класса называются элементами данных или полями (по аналогии с полями структуры), а функции класса — методами. Поля и методы называются элементами класса.

В общем случае класс как тип данных определяется следующим образом:

```
<спецификатор_класса> <имя_класса>
{
    <спецификатор_доступа_1>:
    <тип_списка_1> <список_имен_1>;
    <тип_списка_2> <список_имен_2>;
    ...
    <спецификатор_доступа_2>:
    <тип_списка_1> <список_имен_1>;
    <тип_списка_2> <список_имен_2>;
    ...
    <спецификатор_доступа_N>:
    <тип_списка_1> <список_имен_1>;
    <тип_списка_2> <список_имен_2>;
    ...
};
```

где <спецификатор класса> — одно из ключевых слов —struct, union, class.

<имя_класса> — произвольное имя класса, заданное программистом, оформленное согласно правилам записи идентификаторов;

<тип_списка_1>...<тип_списка_N> — любой стандартный или определенный пользователем тип языка C++, включая тип «структура». Если используется тип, определенный пользователем, то он должен быть объявлен до описания класса;

<список_имен_1>...<список_имен_N> — имена полей класса (задаются по правилам задания идентификаторов). Имена полей класса объединяются в списки по типам: поля, содержащие данные одинакового типа, указываются при объявлении через запятую (аналогично описанию однотипных переменных). Имена полей класса должны быть уникальными в пределах данного класса, т. е. в разных классах поля могут иметь одинаковые имена. Однако во избежание ошибок лучше, чтобы имена полей были уникальными в пределах всей программы;

<спецификатор_доступа> определяет степень (уровень) доступа к члену класса (табл. 15.1).

Спецификатор доступа действует от места своего задания в определении класса до следующего спецификатора доступа или до конца определения класса, поэтому нет необходимости записывать их перед каждым определением члена-переменной или члена-функции класса. Если при описании полей класса спецификатор доступа не указан, то

Таблица 15.1. Квалификаторы доступа

Спецификатор доступа к полям класса	Область доступности полей класса
private	Данные доступны в функциях членах данного класса и в дружественных функциях данного класса
protected	Данные доступны в функциях членах данного класса, в дружественных функциях данного класса и в функциях членах наследников данного класса
public	Данные доступны в функциях членах данного класса, в дружественных функциях данного класса, в функциях членах наследников данного класса и всем остальным функциям программы

по умолчанию будет использован спецификатор `private` для спецификатора класса — `class` или `public` для спецификаторов классов `struct` или `union`.

15.3. Создание объектов (экземпляров) класса

При определении класса место в памяти не резервируется и не создается никаких переменных (точно так же, как и при переопределении типов). Чтобы зарезервировать память и создать переменную, нужно определить объект (экземпляр класса):

```
имя_класса имя_переменной;  
//объявление переменной типа заданного класса
```

Созданную таким образом переменную будем называть экземпляром класса или объектом класса.

Объявление указателя на объект класса осуществляется стандартным образом:

```
имя_класса *имя_указателя = NULL;
```

Для выделения динамической памяти используется оператор `new`.

```
имя_указателя = new имя_класса;
```

Освобождение ранее выделенной памяти осуществляется оператором `delete`:

```
delete имя_указателя;
```

15.4. Доступ к членам класса

Доступ к полям класса осуществляется точно так же, как и к полям структуры, т. е. с использованием операции «.», если обращение идет по имени объекта, и операции «->», если при обращении используется указатель на объект.

Рассмотрим, например, класс CPoint — класс, содержащий координаты X и Y точки, расположенной в декартовой системе координат. Обратим внимание, что первая буква «С» в имени класса говорит о том, что это имя класса (первая буква от ключевого слова class).

```
#include <stdio.h>
class Cpoint
{
    //начало определения класса CPoint
private:
    //спецификатор доступа для членов класса, доступных только этому классу
    int x; //член-переменная класса
    int y; //член-переменная класса
public:
    //спецификатор доступа для членов класса, доступных всей программе
    void SetX(int _x) //функция-член класса, задающая значение
    {
        X = _x; //члену-переменной x
    }
    void SetY(int _y) //функция-член класса, задающая значение
    {
        Y=_y; //члену-переменной y
    }
    int GetX() //функция-член класса, возвращающая значение
    {
        return x; //члена-переменной x
    }
    int GetY() //функция-член класса, возвращающая значение
    {
        return y; //члена-переменной y
    }
}; //конец определения класса CPoint

int main()
{
    CPoint A, pA = &A; //объявление (создание) объекта A класса CPoint
    A.SetX(3); //задание значения 3 члену-переменной x объекта A
```



```
pA->SetY(7);    //задание значения 7 члену-переменной y объекта A
printf("Точка с координатами (%d, %d);\n", A.GetX(),
      A.GetY());
return 0;
}
```

В результате работы программы на экран будет выведено следующее сообщение:

```
Точка с координатами (3, 7);
```

В данном случае поля `x` и `y` объявлены как скрытые (спецификатор `private`). То есть непосредственно доступ к ним мы не имеем, однако в методах класса (функции `SetX()`, `SetY()`, `GetX()`, `GetY()`) можно к ним обращаться. После создания экземпляра класса `CPoint A`; вызов методов класса осуществляется с использованием операции «.», аналогично при обращении по указателю используется операция «->». Отметим, что если метод не содержит параметров (`GetX()`, `GetY()`), то при их вызове использование круглых скобок **обязательно**. При их отсутствии будет сгенерирована ошибка компилятора.

Обратите внимание, что имя формальных параметров не должно совпадать с именами полей класса, так как в случае их совпадения компилятору не понятно, какая из переменных используется в данном контексте.

В приведенном примере методы описываются непосредственно в классе. Возможен другой вариант описания методов (по аналогии с прототипами функций), когда в классе описывается только прототип метода, а вне класса приводится его полное тело. При этом структура описания метода будет выглядеть следующим образом:

```
тип_результата имя_класса::имя_метода(список
    параметров)
{
    //тело метода
}
```

где `<тип_результата>` — тип возвращаемого методом результата;

`<имя_класса>` — имя класса;

`<имя_метода>` — имя описываемого метода;

операция «`::`» — операция расширения области видимости (указывает на принадлежность метода к определенному классу).

Приведем измененную программу класса CPoint.

```
#include <stdio.h>
class CPoint{                                //начало определения класса CPoint
private:
    //спецификатор доступа для членов класса, доступных только этому классу
    int x;                                    //член-переменная класса
    int y;                                    //член-переменная класса
public:
    //спецификатор доступа для членов класса, доступных всей программе
    void SetX(int);
        //прототип метода класса задающего значение члену-переменной x
    void SetY(int);
        //прототип метода класса, задающего значение члену-переменной y
    int GetX();                               /*прототип метода класса, возвращающего
                                                значение члена-переменной x*/
    int GetY()                               /*прототип метода класса, возвращающего
                                                значение члена-переменной y*/
};                                             //конец определения класса CPoint

void CPoint::SetX(int _x)
    //реализация метода SetX для класса CPoint
{
    x =_x;
}
void CPoint::SetY(int _y)
    //реализация метода SetY для класса CPoint
{
    y =_y;
}
int CPoint::GetX()    //реализация метода GetX для класса CPoint
{
    return x;
}
int CPoint::GetY()    //реализация метода GetY для класса CPoint
{
    return y;
}
int main()
{
    CPoint A;          //объявление (создание) объекта A класса CPoint
    A.SetX(3);          //задание значения 3 члену-переменной x объекта A
    A.SetY(7);          //задание значения 7 члену-переменной y объекта A
    printf("Tochka (%d, %d);\n",A.GetX(), A.GetY());
    return 0;
}
```

Приведем пример динамического создания объекта класса. Определение класса при этом не меняется, а функция `main()` приобретет следующий вид:

```
int main()
{
    CPoint *pB = NULL;           //указатель на объект класса CPoint
    pB = new CPoint;             //выделяем память для хранения объекта
    pB->SetX(123);                //задаем параметры объекта
    pB->SetY(456);
                                   //выводим на экран параметры объекта
    printf("Tochka (%d, %d);\n", pB->GetX(), pB->GetY());
    delete pB;                   //освобождаем ранее выделенную память
    return 0;
}
```

15.5. Конструкторы

Обычно среди методов класса имеются некоторые специальные функции, которые позволяют инициализировать члены класса. В нашем предыдущем примере для этих целей использовались функции SetX() и SetY(). В качестве альтернативного и более правильного способа можно использовать **конструктор** — функцию с тем же именем, что и сам класс. При определении конструкторов нельзя указывать тип возвращаемого результата (конструктор не возвращает результат). Однако он может принимать любое число аргументов (включая нулевое), а следовательно, в классе может быть любое число конструкторов, отличающихся параметрами. Приведем пример класса CPoint с конструктором.

[illegible]


```

CPoint::CPoint(int _x, int _y)
    //реализация конструктора класса CPoint
{
    x = _x;
    y = _y;
}
void CPoint::Prn()          //реализация метода Prn класса CPoint
{
    printf("Tochka (%d, %d);\n", x, y);
}
int main()
{
    CPoint A(3,9);
    A.Prn();
    return 0;
}

```

Конструктор — это особый член-функция класса. У конструктора особая роль в классе. Он совместно с компилятором «конструирует» объект класса — выполняет все действия, необходимые для правильного создания объекта. В силу своей особой роли конструктор должен быть объявлен в секции `public`.

Конструктор может вообще не иметь параметров, в этом случае он называется конструктором по умолчанию. Такой конструктор обычно инициализирует элементы класса, присваивая им значения, установленные по умолчанию. Добавим в класс `CPoint` конструктор по умолчанию, который инициализирует координаты точки нулями.

```

#include <stdio.h>
class CPoint
{
private:
    int x;
    int y;
public:
    CPoint();          //прототип конструктора по умолчанию
    CPoint(int, int); //прототип конструктора с двумя параметрами
    void Prn();        //прототип метода вывода на экран координат точки
};
CPoint::CPoint()      //реализация конструктора по умолчанию
{
    x = y = 0;
}
CPoint::CPoint(int _x, int _y)
    //реализация конструктора с параметрами

```

```

{
    x = _x;
    y = _y;
}
void CPoint::Prn()           //реализация метода Prn класса CPoint
{
    printf("Tochka (%d, %d);\n", x, y);
}
void main()
{
    CPoint A(3,9), B; //создаем точку B конструктором по умолчанию
    A.Prn();
    B.Prn();
}

```

Если в определении класса отсутствуют конструкторы, то компилятор сам генерирует, а при создании объекта вызывает конструктор, который не имеет параметров и не выполняет никаких действий по инициализации членов-переменных объекта класса.

Конструктор с параметрами можно объединить с конструктором по умолчанию. Для этого необходимо указать значения параметров, используемые по умолчанию:

```

class CPoint
{
public:
    CPoint(int x = 0, int _y = 0);    /*прототип объединенного
    конструктора с параметрами и конструктора по умолчанию, в котором
    параметры переменные x и y будут иметь нулевые значения*/

    CPoint::CPoint(int _x, int _y)
        //реализация объединенного конструктора
    {
        x = _x;
        y = _y;
    }
};
int main()
{
    CPoint A;           /*при создании объекта A будет вызван конструктор
    по умолчанию и значения полей x и y будут равны 0*/
    Cpoint B(221, 57);   /*при создании объекта B будет вызван
    конструктор с параметрами и значения полей x и y будут
    иметь значения 221 и 57*/

    return 0;
}

```

В этом примере в классе `CPoint` объявлен один конструктор, объединяющий конструктор по умолчанию и конструктор с параметрами. Если при создании объекта не указываются значения, передаваемые в конструктор (например: `CPoint A;`), то используется конструктор по умолчанию. Для создания объекта `B` будет использован конструктор с параметрами.

15.6. Применение конструкторов копирования и преобразования

Рассмотрим специальные свойства конструкторов, имеющих единственный параметр. Если этот параметр является ссылкой на тот же тип данных, что и класс, то конструктор называется **конструктором копирования**. Если же параметр имеет любой тип, отличающийся от класса, то такой конструктор называется **конструктором преобразования**.

Конструкторы с единственным параметром позволяют инициализировать объект, используя знак равенства в самом определении объекта. Такой конструктор определяет правила преобразования определенного типа данных к типу класса.

Например, если класс `CTest` имеет конструктор

```
CTest(int Parm)
{
    операторы;
}
```

то можно создать объект, используя оператор:

```
CTest A(5);
```

или эквивалентный оператор:

```
CTest A = 5;
```

В этом примере конструктор преобразования определяет, каким образом целое число `Param` (тип `int`) будет преобразовано к объекту класса `CTest`. Использование знака равенства — альтернативный способ передачи единственного значения конструктору. Еще раз подчеркнем, что это операция инициализации, а не присваивания.

Приведем пример конструктора преобразования для класса CPoint.

```
#include <stdio.h>
#include <math.h>
class CPoint
{
private:
    int x, y;
public:
    CPoint(int _x, int _y)           //конструктор с параметрами
    {
        x = _x;
        y = _y;
    }
    CPoint()                       //конструктор по умолчанию
    {
        x = y = 0;
    }
    CPoint(int a)                  //конструктор преобразования
    {
                                   x = y = a;
    }
    void print()                  //метод вывода координат точки на экран
    {
        printf("Tochka: (%d, %d).\n", x, y);
    }
};

int main()
{
    CPoint p1 = 5;
    //объект p1 создается с использованием конструктора преобразования
    p1.print();
    return 0;
}
```

Пояснение. После реализации конструктора преобразования становится доступным еще один способ создания нового объекта класса: CPoint p1 = 5. Возможна и такая запись: CPoint p1(5).

Таким образом, конструктор выполняет явное преобразование типов данных, используя разрешенный в языке C++ синтаксис. Другими словами, конструктор преобразования указывает компилятору, как преобразовывать объекты или переменные различных типов в объект данного класса.

15.7. Конструкторы копирования

Конструктор копирования класса — это конструктор с единственным параметром, тип которого определен как ссылка на тип класса:

```
class CTest
{
    // ...

public
CTest (const CTest &Test)
{
    /*использует члены объекта Test для инициализации
      нового объекта класса CTest...*/
}
...
};
```

Обратим внимание на то, что параметр должен быть ссылкой на объект, а не самим объектом. Это связано с тем, что перед передачей параметров в функцию по значению создается их копия, которая в дальнейшем и обрабатывается. А при передаче по ссылке в конструктор передается адрес существующего объекта, а не его копия и необходимость в копировании (рекурсии) отпадает.

Если конструктор копирования класса не определен, то компилятор генерирует его неявно. Конструктор, генерируемый компилятором, инициализирует новый объект, выполняя операцию поэлементного копирования данных объекта класса, передаваемого как параметр. Соответственно, используя объект того же типа, всегда можно инициализировать его, даже если конструктор копирования в классе не определен. Например, даже если класс `CPoint` не содержит конструктор копирования, то следующая инициализация все равно будет корректной:

```
CPoint tochka1(67, 41);
CPoint tochka 2(tochka 1);
CPoint tochka 3 = tochka 1;
```

Инициализация объекта `tochka 2`, как и `tochka 3`, вызывает конструктор копирования, сгенерированный компилятором. В результате данной инициализации оба объекта (`tochka 2` и `tochka 3`) будут содержать те же значения, что и `tochka 1`.

Если операция поэлементного копирования, выполняемая конструктором копирования, сгенерированным компилятором, не подходит для создаваемого класса, но необходимо инициализировать новые объекты с помощью существующих объектов того же типа, то необходимо определить собственный конструктор копирования.

Для класса `CPoint` конструктор копирования будет выглядеть следующим образом:

```
class CPoint
{
    ....
public:
    CPoint(CPoint& p)
    {
        x = p.x;
        y = p.y;
    }
    ...
};
```

После добавления такого конструктора в класс появляется возможность использования следующего оператора:

```
CPoint p2(p1);
```

Также возможен следующий вариант использования конструктора копирования:

```
CPoint p2 = p1;
```

Еще раз подчеркнем, что в данном случае операция «=» является операцией инициализации.

15.8. Деструкторы

Деструктор — специальная функция, именем которой является имя класса с префиксом-тильдой (~), которая вызывается каждый раз при уничтожении объекта. Деструктор сначала выполняет действия, описанные программистом, например освобождает выделенную память, присваивает указателям пустые значения и т. д., а затем выполняет действия по разрушению (деструкции) объекта. В соответствии со своим назначением деструктор не возвращает никакого значения и

не имеет параметров, а следовательно, деструктор нельзя перегрузить и он должен быть в классе один.

Для класса `CPoint` деструктор будет выглядеть следующим образом:

```
class CPoint
{
    ...
    ~CPoint();           //прототип деструктора
    ...
};

CPoint::~CPoint()        //реализация деструктора
{
    //тело деструктора
}
```

В данном примере деструктор не будет выполнять никаких действий. Одно из основных применений деструктора — освобождение выделенной ранее памяти. Приведем пример модернизированного класса `CPoint`, где в качестве данных класса используются указатели на целочисленные переменные.

```
# include <stdio.h>
class CPoint
{
private:
    int *px;                //указатели
    int *py;
public:
    CPoint(int x = 0, int _y = 0);
        //совмещенный конструктор с параметрами и по умолчанию
    ~CPoint();               //деструктор
    void Prn();              //метод вывода координат точки на экран
};

CPoint::CPoint(int _x, int _y)
{
    px = new int;            //выделяем память
    *px = _x;
    py = new int;            //для хранения координат точки
    *py = _y;
}
CPoint::~CPoint()
```

```
{
    delete px;                                //при уничтожении объекта
    delete py;                                //освобождаем ранее выделенную память
}
void CPoint::Prn()
{
    printf("Tochka (%d, %d);\n", *px, *py);
}
void main()
{
    CPoint A(3,9), B;
    A.Prn();
    B.Prn();
}
```

Если деструктор не определен, то компилятор генерирует деструктор, который разрушает объект.

15.9. Когда вызываются конструкторы и деструкторы

В общем случае конструктор вызывается при создании объекта, а деструктор — при его уничтожении. Рассмотрим возможные случаи вызова конструкторов и деструкторов для различного вида объектов:

- объект определен глобально (т. е. вне любой функции). Конструктор вызывается в самом начале программы до вызова функции `main`, деструктор — по окончании программы;
- объект определен локально (т. е. внутри функции). Конструктор вызывается, когда поток управления достигает определения объекта, деструктор — при выходе за пределы блока, в котором определен объект;
- объект определен локально с использованием спецификатора `static`. Конструктор вызывается, когда поток управления впервые достигает определения объекта, деструктор — в конце программы;
- объект создан динамически с использованием оператора `new`. Конструктор вызывается при создании объекта, а деструктор — когда объект явно уничтожается с использованием оператора `delete`. Если этого не происходит, деструктор не будет вызван никогда.

15.10. Передача объектов класса в функции

Объекты классов передаются в функцию как стандартные типы данных. Их можно передавать, используя механизм формальных и фактических параметров (передача по значению), используя указатели или ссылки.

Пример 1. Написать функцию, вычисляющую расстояние между двумя точками. Точки представлены объектами класса CPoint.

```
#include <stdio.h>
#include <math.h>
class CPoint                                //начало определения класса CPoint
{
private:
    int x, y;                                //закрытые члены-переменные класса
public:
    CPoint(int _x, int _y)                  //конструктор класса
    {x = _x; y = _y; }
    int GetX()    //функция, возвращающая значение члена-переменной x
    { return x; }
    int GetY()    //функция, возвращающая значение члена-переменной y
    { return y; }
};                                           //конец определения класса CPoint
float length(CPoint p1, CPoint p2)
    //функции нахождения расстояния между двумя точками
{
    int x1, x2, y1, y2, L;
    x1 = p1.GetX();
        //присвоить x1 значения члена-переменной x точки p1
    y1 = p1.GetY(); //присвоить y1 значения члена-переменной y точки p1
    x2 = p2.GetX();
        //присвоить x2 значения члена-переменной x точки p2
    y2 = p2.GetY();
        //присвоить y2 значения члена-переменной y точки p2
    L = sqrt(pow(x2-x1, 2.0) + pow(y2-y1, 2.0));
        //вычисление расстояния между точками по теореме Пифагора
    return L; //возврат вычисленного значения L в вызывающую функцию
}
int main()
{
    CPoint p1(3, 0), p2(0, 4);
    printf("Length = %3.2f\n", length(p1, p2));
    return 0;
}
```


Пояснение. Обратите внимание на то, что члены-переменные класса `x` и `y` являются закрытыми. Для работы с ними предусмотрены методы `GetX()` и `GetY()`.

Пример 2. Вершины треугольника заданы массивом точек. Точки являются объектом класса `CPoint`. Написать функцию, вычисляющую площадь треугольника по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p — полупериметр; a, b, c — длины сторон треугольника.

```
#include <stdio.h>
#include <math.h>
class CPoint
{
private:
    int x, y;
public:
    CPoint(int _x, int _y)
    {x = _x; y = _y;}
    CPoint()
    {x = 0; y = 0;}
    void SetX(int _x)
    {x = _x;}
    void SetY(int _y)
    {y = _y;}
    int GetX()
    { return x; }
    int GetY()
    { return y; }
};

float length(CPoint p1, CPoint p2)
{
    int x1, x2, y1, y2, L;
    x1 = p1.GetX();
    y1 = p1.GetY();
    x2 = p2.GetX();
    y2 = p2.GetY();
    L = sqrt(pow(x2-x1, 2.0) + pow(y2-y1, 2.0));
    return L;
}
```

```
float square(CPoint *mas)
{
    float L1, L2, L3, p;
    L1 = length(mas[0], mas[1]);
    L2 = length(mas[1], mas[2]);
    L3 = length(mas[2], mas[0]);
    p = (L1 + L2 + L3)/2.0;
    if (p*(p-L1)*(p-L2)*(p-L3)>=0)
        return sqrt(p*(p - L1)*(p - L2)*(p - L3));
    else {printf("Треугольник построить нельзя!\n");
        return -1;
    }
}

int main()
{
    CPoint mas [3];
    int x, y;
    for (int i = 0; i<3; i++)
    {
        printf("First point:");
        scanf("%d%d", &x, &y);
        mas[i].SetX(x);
        mas[i].SetY(y);
    }
    printf("S = %3.2f\n", square(mas));
    return 0;
}
```

Пояснение. Обратите внимание, что координаты треугольника хранятся в массиве из трех объектов класса CPoint. Для объявления такого массива класс CPoint обязательно должен иметь конструктор по умолчанию.

Для вычисления площади треугольника используем формулу Герона: $S = \sqrt{p(p-a)(p-b)(p-c)}$, где p — полупериметр, a, b, c — длины сторон треугольника. Для вычисления этих длин пользуемся ранее написанной функцией `length()`.

Для вычисления площади треугольника служит функция `square()`, в которую передается указатель на начало массива из объектов класса CPoint.

Для работы с закрытыми элементами класса служат методы `set/get`.

15.11. Класс CVector

В практических задачах часто возникает необходимость работать с массивами и матрицами, размеры которых не всегда определены заранее. Рассмотрим, какие преимущества предоставляют классы при работе с массивами данных. Условимся считать: CVector — класс для работы с одномерными массивами; CMatrix — класс для работы с двумерными массивами (матрицами).

Рассмотрим реализацию и применение класса CVector для работы с вещественными одномерными массивами.

При использовании классов их описание можно размещать в том же файле, где находится функция main (*.cpp). Однако при работе в интегрированной среде разработки Microsoft Visual Studio лучше вынести определение класса в отдельные файлы, а при использовании этих классов останется только подключить эти файлы с использованием директивы #include. Добавление файлов в проект рассмотрено в гл. 2.

Таким образом, проект будет состоять из трех файлов:

- файл с описанием класса CVector (*.h);
- файл с реализацией методов класса CVector (*.cpp);
- файл с основной программой, содержащей функцию main (*.cpp).

Приведем содержимое файла с описанием класса CVector (vector.h).

```
#ifndef _vector_           //обеспечиваем включение описания класса
#define _vector_          //в компилируемый текст только один раз
#include <stdio.h>          //для использования функции printf
#include <stdlib.h>         //для использования генератора
#include <time.h>           //случайных значений
class CVector
{
private:                  //закрытые элементы
    int N;                //количество элементов в массиве
    float *px;            //указатель на начало массива
public:
    CVector(int n = 1);    //конструктор с параметром и по умолчанию
    ~CVector();            //деструктор
    void prn();            //печать значений массива на экран
    void gen();            //генерация случайных элементов массива
    float max();           //нахождение максимального элемента в массиве
};
#endif
```


Приведем содержимое файла с реализацией методов класса CVector (vector.cpp).

```
#include "vector.h"           //для использования класса CVector
CVector::CVector(int n)
{
    N = n;           //в переменной N сохраняем количество элементов массива
    px = new float [N];           //выделяем память для массива
}
CVector::~CVector()           //реализация деструктора
{
    delete [] px;           //освобождаем выделенную память
}
void CVector::gen()           //метод генерации случайных значений
{                           //элементов массива
    for (int i = 0; i<N; i++)
        px[i] = float(rand())/RAND_MAX*5;
}
void CVector::prn()
{
    for (int i = 0; i<N; i++) //в цикле выводим значения элементов
        printf("%3.2f\t", px[i]);           //массива на экран
    printf("\n");
}

float CVector::max()           //метод поиска максимума в массиве
{
    float M;
    int i;
    M = *px;
    for (i = 1; i<N; i++)
        if (px[i]>M)
            M = px[i];           //находим максимальное значение
    return M;           //в массиве и возвращаем его
}
```

Приведем содержимое файла с функцией main (main.cpp).

```
# include "vector.h"           //подключаем описание класса CVector
void main()
{
    srand(time(NULL));           //запускаем генератор случайных чисел
    CVector massiv(3);           //создаем массив из трех элементов
    massiv.gen();           //генерируем случайные значения массива
    massiv.prn();           //выводим значения массива на экран
    printf("max = %3.2f\n", massiv.max());           //находим максимальный элемент массива
}
```

Результат работы программы:

```
2.53      3.62      3.42
max = 3.62
```

В этом примере описание класса вынесено в файл `vector.h`, а реализация методов класса находится в файле `vector.cpp`. В основную программу подключается заголовочный файл `vector.h`. Следует обратить внимание на тот факт, что в основную программу не подключается файл, содержащий реализацию методов класса (`vector.cpp`). В этом нет необходимости, так как все файлы с расширением `*.cpp`, содержащие исходный код программы, будут скомпилированы автоматически.

Благодаря использованию классов основная программа очень компактна, наглядна и понятна. Класс `CVector` позволяет работать с вещественными массивами произвольного размера. Он позволяет быстро и просто создавать массивы различного размера, генерировать случайные значения массивов, выводить их на экран, находить максимальные элементы. Добавление других методов класса позволит продолжить этот список.

15.12. Класс CMatrix

Рассмотрим реализацию и применение класса `CMatrix` для работы с целочисленными матрицами.

Приведем содержимое файла с описанием класса `CMatrix` (`matrix.h`).

```
#ifndef _matrix_
#define _matrix_
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
class CMatrix
{
private:
    int M;
    int N;
    int **ppx;
```

```

public:
    CMatrix(int m = 1, int n = 1);           //конструктор с параметрами и по умолчанию
    ~CMatrix();                             //деструктор
    void Gen_elem();                         //метод генерации значений матрицы
    void Print_elem();                      //вывод элементов матрицы на экран
};
#endif

```

Приведем содержимое файла с реализацией методов класса CMatrix (matrix.cpp).

```

#include "matrix.h"
CMatrix::CMatrix(int m, int n)             //реализация конструктора
{
    M = m;                                //запоминаем, сколько строк
    N = n;                                //и столбцов будет в матрице
    ppx = new int *[M];                   //выделяем память
    for (int i = 0; i<M; i++)              //для хранения
        ppx[i] = new int [N];             //элементов матрицы
}
CMatrix::~~CMatrix()                      //в деструкторе освобождаем
{
    for (int i = 0; i<M; i++)              //ранее выделенную память
        delete [] ppx[i];
    delete [] ppx;
}
void CMatrix::Gen_elem() //реализация метода генерации случайных
{
    for (int i = 0; i<M; i++)              //значений элементов матрицы
        for (int j = 0; j<N; j++)
            ppx[i][j] = rand() % 100 - 50;
}
void CMatrix::Print_elem()
{
    printf("Matrica:\n");                  //выводим элементы матрицы на экран
    for (int i = 0; i<M; i++)
    {
        for (int j = 0; j<N; j++)
            printf("\t%d", ppx[i][j]);
        printf("\n");
    }
}

```


Приведем содержимое файла с функцией `main` (`main.cpp`).

```
# include "matrix.h"
void main()
{
    CMatrix matr(12,7);
        //создаем объект, с помощью конструктора с параметрами
    matr.Gen_elem();           //генерируем случайные значения,
    matr.Print_elem();         //выводим их на экран

    CMatrix matrica;
        //создаем матрицу с помощью конструктора по умолчанию
    matrica.Gen_elem();        //генерируем
    matrica.Print_elem();      //и выводим значения
                                //элементов матрицы на экран
}
```

В этом примере описание класса вынесено в файл `matrix.h`, а реализация методов класса находится в файле `matrix.cpp`. В основную программу подключается заголовочный файл `matrix.h`. Благодаря использованию класса основная программа очень компактна, наглядна и понятна. Класс `CMatrix` позволяет работать с целочисленными матрицами произвольного размера. Он позволяет создавать матрицы различного размера, генерировать случайные значения массивов, выводить их на экран.

15.13. Примеры разработки класса

Пример 1. Разработать класс «Робот». Члены класса: координаты текущего положения робота (x, y). Методы: «шаг робота» — в зависимости от указанного направления перемещает робота. Продемонстрировать применение класса на следующем примере: ввести с клавиатуры последовательность из 50 символов 'N', 'S', 'W', 'E', указывающих направление движения робота. В начальный момент времени робот находится в начале координат (0, 0). Определить, через сколько шагов он вернется в исходное положение.

Файл `robot.h`:

```
#ifndef _robot_
#define _robot_
#include <stdio.h>
class Robot
```

```

{
private:
    int x, y;                //координаты положения робота
public:
    Robot();                 //конструктор по умолчанию
    void Shag(char);         //метод, выполняющий шаг робота
    int Proverka();          //метод проверки нахождения робота в начале координат
};
#endif

```

Файл *robot.cpp*:

```

Robot::Robot()
{
    x = 0; y = 0;           //в начале работы программы робот находится
                             //в центре координат
void Robot::Shag(char c)
{
    switch (c)
    {
        case 'N': y++; break;    //шаг на север — увеличиваем y на 1
        case 'S': y--; break;    //шаг на юг — уменьшаем y на 1
        case 'W': x--; break;    //шаг на запад — уменьшаем x на 1
        case 'E': x++; break;    //шаг на восток — увеличиваем x на 1
        default: printf("\nНеверное направление!\n");
    }
}
int Proverka()
{
    if (x == 0 && y == 0)       //если робот находится в начале координат,
        return 1;              //возвращаем 1
    else
        return 0;              //в противном случае возвращаем 0
}

```

Файл *main.cpp*:

```

#include "robot.h"
#define N 50                //определим число шагов робота

int main()
{
    Robot r;                 //создаем робота
    char mas[N];

```

```

int i;
printf("Vvedite 50 simvolov 'N', 'S', 'W', 'E':");
for (int i = 0; i < N; i++)
{
    char buff;
    printf("\nsimvol[%d] = ", i + 1);
    scanf("%c", &mas[i]);           //считываем символ
    scanf("%c", &buff);           //считываем нажатие клавиши enter
}
int flag = 0;           //флаг расположения робота в начале координат
i = 0;
do
{
    r.Shag(mas[i]);           //робот делает шаг
    flag = r.Proverka();
                               //проверяем, попал ли он в начало координат
    i++;                     //увеличиваем счетчик шагов
}
while (flag == 0 && i < N); /*выполняем цикл до тех пор, пока
    робот не попал в начало координат или не сделал все 50 шагов*/
if (i == 50)
    printf("\nРобот не попал в начало координат!\n");
else
    printf("\nРобот попал в начало координат через %d
        шагов!\n", i);
return 0;
}

```

В этом примере создается класс робот. Члены класса — координаты робота x и y . Для соблюдения принципов инкапсуляции эти данные являются закрытыми. Для работы с ними используется два метода — `Shag` и `Proverka`. В основной программе создается объект класса `CRobot`. После ввода направлений движения с клавиатуры робот начинает выполнение шагов. Движение продолжается до тех пор, когда робот вернется в начало координат либо выполнит все 50 шагов.

15.14. Классы в качестве полей других классов

В C++ в качестве членов-переменных класса можно использовать объекты других классов. Однако членом класса не может быть объект этого же класса, но допускается использовать указатель на объект текущего класса. Продемонстрируем эти возможности на примерах.

Пример 1. Написать класс «окружность». Члены класса: центр окружности (объект класса CPoint, радиус R).

```
#include <stdio.h>
#include <math.h>
class CPoint          //класс точка, в дальнейшем будет использоваться
{                    //в качестве центра окружности
private:
    int x, y;        //координаты точки
public:
    CPoint(int _x, int _y)    //конструктор с параметрами
    {x = _x; y = _y;}
    CPoint()                //конструктор по умолчанию
    {x = 0; y = 0; }
    void SetX(int _x)        //метод задания значения координате x
    {x = _x;}
    void SetY(int _y)        //метод задания значения координате y
    {y = _y;}
    int GetX()               //метод получения значения координаты x
    { return x; }
    int GetY()               //метод получения значения координаты y
    { return y; }
};

class CCircle          //класс окружность
{
private:
    CPoint Centr; //определение члена-переменной — центра окружности
    int R;        //определение члена-переменной — радиуса окружности
public:
    CCircle(int x, int y, int r) //конструктор класса CCircle
    {
        Centr.SetX(x);          /*вызов метода SetX класса CPoint для
                                задания значения x центра окружности*/
        Centr.SetY(y);          /*вызов метода SetY класса CPoint для
                                задания значения y центра окружности*/
        R = r;                  //задание радиуса окружности
    }
    void info() //метод вывода на экран информации об окружности
    {
        printf("Центр: (%d, %d).\n", Centr.GetX(),
            Centr.GetY());
        printf("Радиус окружности: %d.\n", R);
    }
};
```



```

{
    P1.SetX(3);           //задаем координаты начала и конца отрезка,
    P1.SetY(3);           //вызывая методы SetX и SetY
    P2.SetX(8);           //для объекта класса CPoint
    P2.SetY(8);
}
COTrezok(int x1, int y1, int x2, int y2)
                               //конструктор с параметрами
{
    P1.SetX(x1);
    P1.SetY(y1);
    P2.SetX(x2);
    P2.SetY(y2);
}
COTrezok(CPoint p1, CPoint p2) //конструктор с параметрами
{
    P1 = p1;                //задаем значения начала отрезка
    P2 = p2;                //задаем значения конца отрезка
}
void info()
{
    printf("Начало отрезка: (%d, %d).\n", P1.GetX(),
        P1.GetY());
    printf("Конец отрезка: (%d, %d).\n", P2.GetX(),
        P2.GetY());
}
};

int main()
{
    COTrezok otr1(5, 4, 10, 3); //создаем объект класса «отрезок»
    otr1.info();                //выводим координаты отрезка на экран
    CPoint p1(1, 1), p2(7, 7);
        //создаем объект класса «отрезок» с помощью второго конструктора
    COTrezok otr2(p1, p2);
        //создаем объект класса «отрезок» с помощью третьего конструктора
    otr2.info();
    return 0;
}

```

Пояснение. В классе `COTrezok` объявлено три конструктора: 1-й для создания объекта использует четыре целых числа (координаты начала и конца отрезка); 2-й — конструктор по умолчанию: создает

отрезок с началом в точке (3, 3) и концом в точке (8, 8); 3-й конструктор использует для создания нового объекта ранее созданные объекты класса `CPoint`.

Тесты

1. Что означает аббревиатура ООП:

- а) объектный образ в программировании;
- б) объектно-ориентированное программирование;
- в) объективно ориентированное программирование.

2. Принцип инкапсуляции обеспечивает:

- а) объединение данных и методов работы с ними в классе;
- б) доступ к членам класса;
- в) сокрытие данных внутри класса.

3. Укажите правильный вариант определения класса в программе:

а)

```
class Test
{
    public:
    int a;
};
```

б)

```
class Test
{
    private:
    float x;
}
```

- в) оба варианта правильные.

4. Спецификатор доступа `private` обеспечивает:

- а) доступность членов класса в методах данного класса и в дружественных функциях данного класса;
- б) доступность членов класса в методах данного класса, в дружественных функциях данного класса и в методах наследников данного класса;
- в) доступность членов класса во всех функциях программы.

5. Спецификатор доступа `protected` обеспечивает:

- а) доступность членов класса в методах данного класса и в дружественных функциях данного класса;
- б) доступность членов класса в методах данного класса, в дружественных функциях данного класса и в методах наследников данного класса;
- в) доступность членов класса во всех функциях программы.

6. Спецификатор доступа public обеспечивает:

- а) доступность членов класса в методах данного класса и в дружественных функциях данного класса;
- б) доступность членов класса в методах данного класса, в дружественных функциях данного класса и в методах наследников данного класса;
- в) доступность членов класса во всех функциях программы.

7. Что будет выведено на экран в результате выполнения следующей программы?

```
class CLight
{
    int a;
};
...
CLight L;
L.a = 45;
printf("a = %d\n", L.a);
...
```

- а) a = 45;
- б) программа не запустится, так как доступ к полю «a» необходимо получить, используя операцию: L->a = 45;
- в) программа не запустится, так как переменная «a» является закрытой.

8. Конструктор — это:

- а) специальный метод класса с тем же именем, что и сам класс;
- б) специальный метод класса, не имеющий параметров и не возвращающий никакого значения;
- в) механизм создания новых объектов класса.

9. В классе может быть только:

- а) единственный конструктор;
- б) два конструктора — по умолчанию и с параметрами;
- в) произвольное количество конструкторов.

10. Деструктор — это:

- а) специальный метод класса с тем же именем, что и сам класс с префиксом — тильдой (~);
- б) автоматически создаваемый метод класса, предназначенный для удаления объектов класса;
- в) механизм разрушения объектов класса.

Задания

1. Написать класс точка в трехмерном пространстве (CPoint3D). Члены класса — координаты точки x , y , z . В классе предусмотреть следующие методы: конструктор, деструктор, задание значений координатам точки, вывод на экран информации о точке. Создать несколько объектов этого класса.

2. Пусть некоторое устройство содержит в качестве состояния число от 0 до 126. В каждый следующий момент времени число меняется по формуле $x(n+1) = (a * x(n) + b) \bmod 127$, $a = 10$, $b = 11$. Создать класс, отображающий это устройство. В классе должны быть функции-члены: конструктор, функция, переводящая устройство в следующее состояние, функция, отображающая текущее состояние устройства.

3. Написать функции вычисления поэлементной суммы и разности двух объектов класса CPoint.

4. Написать класс «окружность». Члены класса: радиус окружности, координаты центра окружности. Методы класса: установка радиуса, вычисление длины окружности, вычисление площади круга. Продемонстрировать применение класса.

5. Разработать класс «монстр». Члены класса — возраст монстра, здоровье монстра, сила, защита. В классе предусмотреть: конструктор, функцию, выводящую всю информацию о монстре, функцию, наносящую удар по другому монстру (результат работы функции — случайное количество очков от 1 до максимальной силы монстра), и функцию, принимающую удар (результат работы функции: от текущего здоровья монстра отнимается число очков, переданное в качестве параметра функции за минусом защиты монстра). Промоделировать бой монстров.

6. Написать класс «отрезок». Члены класса: две точки — начало и конец отрезка. В классе предусмотреть метод, вычисляющий длину отрезка.

7. Написать функцию, вычисляющую расстояние между двумя точками (объекты класса CPoint), переданных ей в качестве параметров.

8. Написать функцию, вычисляющую площадь круга (объект класса CCircle), переданного ей в качестве параметра.

Контрольные вопросы

1. Что такое объектно-ориентированное программирование? Какие преимущества оно дает?
2. В чем заключается принцип инкапсуляции?

3. Что такое класс? Как объявить класс в программе?
4. Какие спецификаторы доступа используются в языке C++? Перечислите их особенности.
5. С помощью каких операций можно получить доступ к членам-переменным класса? В каких случаях такой доступ возможен?
6. Что такое конструктор? Для чего он предназначен? Сколько конструкторов может быть определено в классе?
7. Для чего служит конструктор копирования? Как его определить в классе?
8. Для чего служит конструктор преобразования? Как его определить в классе?
9. Что такое деструктор? Для чего он предназначен? Сколько деструкторов может быть определено в классе?
10. Какими способами можно передать объекты классов в функции?
11. Можно ли использовать объекты одного класса в качестве членов-переменных другого класса? Какие есть ограничения?

При вызове этого конструктора для каждого объекта данного класса значение члена-переменной *R* будет проинициализировано значением *_r*, после чего будут выполнены все операторы, расположенные в теле конструктора.

Если необходимо проинициализировать несколько констант, то их нужно разделять запятыми, например, так: *R(_r)*, *C(0)* и т. д.

Приведем всю программу, реализующую класс «окружность».

```
# include <stdio.h>
class CCircle                                //класс «окружность»
{
private:
    int x;
    int y;
    const int R;
public:
    CCircle(int _x = 0, int _y = 0, int _r = 1);
                                                //прототип конструктора
    void prn();
};
CCircle::CCircle(int _x, int _y, int _r) : R(_r)
                                                //реализация конструктора
{
    x = _x;
    y = _y;
}

void CCircle::prn()
{
    printf("Centr (%d, %d), R = %d.\n", x, y, R);
}

void main()
{
    CCircle c1(7, 8, 4), c2;
    c1.prn();
    c2.prn();
}
```

Пояснение. В этом примере радиус окружности определен как константа. Для задания значения константе *R* используется список инициализации. При создании объектов класса «окружность» в конструктор передаются три параметра: координаты центра *x*, *y* и радиус. Значение радиуса используется в списке инициализации.

16.2. Статические члены класса

Объект (экземпляр) класса имеет собственную копию переменных-членов, принадлежащих классу. Однако, если переменная-член объявлена с использованием спецификатора `static`, возможно существование единственной копии этой переменной-члена независимо от того, сколько экземпляров класса создается (даже если не создается ни одного экземпляра). Другими словами, поля, объявленные со спецификатором `static` — это такие поля, которые принадлежат не экземплярам класса, а самому классу. Такими полями удобно пользоваться, например, для подсчета количества созданных объектов класса.

Статические члены класса обязательно должны быть проинициализированы:

```
тип имя_класса::имя_переменной = значение;
```

Оператор инициализации статического члена класса должен находиться за пределами определения класса и за пределами определения любой из функций программы и класса.

Приведем пример класса для хранения данных о студентах. В класс добавим возможность подсчета общего количества созданных объектов.

```
# include <stdio.h>
# include <string.h>
class Cstudent                                //начало определения класса Cstudent
{
private:
    char fio[30];                             //ФИО студента
    int bd;                                   //год рождения студента
    int num;                                  //номер текущего объекта
    static int count;                         //общее количество созданных объектов
public:
    CStudent(char FIO[], int);                //прототип конструктора класса Cstudent
    void prn();                               //прототип члена-функции prn класса Cstudent
};                                             //конец определения класса Cstudent
CStudent::CStudent(char FIO[] = "Николенко",
    int BD = 1990) //определение конструктора класса Cstudent
{
    strcpy(fio, FIO);
    //копирование параметра FIO в член-переменную fio класса Cstudent
    bd = BD;
```

```

        //присваивание значения параметра BD члену-переменной bd класса
CStudent
count++;                                //увеличиваем на 1 количество объектов
num = count;                            //сохраняем номер текущего объекта
}
int CStudent::count = 0;                 /*обязательная инициализация
                                         статического члена вне определения класса и вне
                                         всех функций-членов класса и функций программы*/
void CStudent::prn()
    //определение члена-функции prn класса Cstudent
{
    printf("Студент # %d, %s родился %d;\n Всего
           студентов- %d.\n", num, fio, bd, count);
    //вывод информации о текущем студенте и о количестве студентов
}
int main()
{
    CStudent student1("Иванов Николай", 1995);
    CStudent student2("Лершов Владимир", 1994);
    student1.prn();
    student2.prn();
    return 0;
}

```

Пояснение. В классе Cstudent хранится фамилия, год рождения и порядковый номер студента. Для подсчета общего количества созданных объектов класса Cstudent используется статическое поле count. Инициализация этого поля выглядит следующим образом:

```
int CStudent::count = 0;
```

При создании нового объекта вызывается конструктор класса, в котором увеличивается счетчик созданных объектов и присваивается номер текущему объекту.

Таким образом, для создания статического члена класса необходимо использовать ключевое слово `static`. В случае необходимости можно получить доступ к статическому члену через имя класса. Если в приведенном примере объявить `count` в разделе `public`, то можно вставить в программу такую строчку:

```
printf("Всего студентов: %d.\n", CStudent::count);
```


16.3. Статические функции-члены класса

С использованием спецификатора `static` можно задать статическую функцию-член. Такую функцию можно вызывать, используя имя класса. Ее особенностью является то, что она может ссылаться только на статические переменные и функции, принадлежащие классу. В рассматриваемом выше примере статический член класса `count` объявлен со спецификатором `private`, следовательно, доступ к нему можно получить только внутри методов класса. Добавим в этот пример статическую функцию-член класса, выводящую на экран количество созданных объектов.

```
# include <stdio.h>
# include <string.h>
class Cstudent          //начало определения класса Cstudent
{
private:
    char fio[30];        //ФИО студента
    int bd;              //год рождения студента
    int num;             //номер текущего объекта
    static int count;    //общее количество созданных объектов
    static void prn_count(); //статическая функция-член класса
public:
    CStudent(char FIO[], int);
                                //прототип конструктора класса Cstudent

    void prn();              //прототип члена-функции prn класса Cstudent
};
CStudent::CStudent(char FIO[] = "Николенко", int BD =
    1990)                    //определение конструктора класса Cstudent
{
    strcpy(fio, FIO);
    //копирование параметра FIO в член-переменную fio класса Cstudent
    bd = BD;
    //присваивание значения параметра BD члену-переменной bd класса
                                // CStudent
    count++;                  //увеличиваем количество объектов
    num = count;             //сохраняем номер текущего объекта
}
int CStudent::count = 0;
                                //обязательная инициализация статического члена
void CStudent::prn()         //определение члена-функции prn класса
Cstudent
```



```

{
    printf("Студент # %d, %s родился %d;\n Всего
           студентов- %d.\n", num, fio, bd, count);
    //вывод информации о текущем студенте и о количестве студентов
}
void CStudent::prn_count()
    //определение статической члена-функции prn класса Cstudent
{
    printf("Всего %d студентов.\n", count);
    //вывод количества студентов
}
int main()
{
    CStudent student1("Иванов Николай", 1995);
    CStudent student2("Лершов Владимир", 1994);
    student1.prn();
    student2.prn();
    CStudent::prn_count();
    //вызываем статическую функцию, используя имя класса
    return 0;
}

```

Пояснение. В классе объявлен статический метод `void prn_count()`, о чем свидетельствует ключевое слово `static` перед именем метода. В теле этой статической функции можно использовать только статические члены класса (в нашем случае переменную `count`). Попытка обратиться к любой другой переменной вызовет ошибку компилятора.

16.4. Указатель *this*

Когда функция, принадлежащая классу, вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет фиксированное имя **this**, и он автоматически определяется в каждой функции класса следующим образом:

```
имя__класса * const this = адрес__обрабатываемого_объекта;
```

Имя **this** является служебным (ключевым) словом. Явно описать или определить указатель **this** нельзя. Так как **this** является констант-

ным указателем, то изменять его нельзя, однако в каждой принадлежащей классу функции он указывает именно на тот объект, для которого функция вызывается. Указатель **this** является дополнительным (скрытым) параметром каждой нестатической функции.

Приведем пример использования этого указателя.

```
# include <stdio.h>
class CPoint                                     //начало определения класса CPoint
{
private:
    int x;                                       //определение целой закрытой члена-переменной x
    int y;                                       //определение целой закрытой члена-переменной y
public:
    CPoint(int, int);                           //прототип конструктора класса CPoint с двумя параметрами
    void prn();                                 //прототип члена-функции prn класса CPoint
};                                              //конец описания класса CPoint
CPoint::CPoint(int x = 0, int _y = 0)          //определение конструктора класса CPoint с двумя параметрами
{
    this->x = _x;                               //получаем доступ к полям
    this->y = _y;                               //с помощью указателя
}
void CPoint::prn() //определение члена-функции prn класса CPoint
{
    printf("Tochka (%d, %d);\n", this->x, this->y);
                                                //вывод информации о координатах точки
}
void main()
{
    CPoint p1(7, 8), p2;
    p1.prn();
    p2.prn();
}
```

В данном примере использование указателя **this** не дает никаких преимуществ. Однако его можно использовать для снятия возникающих неопределенностей. Например, конструктор для класса CPoint мог бы выглядеть следующим образом.

```
CPoint::CPoint(int x, int y)
{
    this->x = x;
    this->y = y;
}
```

То есть теперь мы можем использовать имена формальных параметров, совпадающие с именами членов класса. Указатель **this** позволяет снять неопределенность и понять компилятору, в каком месте какая переменная используется.

Следует заметить, что эту неопределенность можно устранить, используя операцию расширения видимости (::).

```
CPoint::CPoint(int x, int y)
{
    CPoint::x = x;
    CPoint::y = y;
}
```

Однако есть ситуации, в которых без использования **this** не обойтись. Рассмотрим пример функции, которая возвращает объект, для которого она вызвана.

```
# include <stdio.h>
class CPoint                                     //класс «точка»
{
    int x, y;                                   //определение закрытых членов-переменных
public:
    CPoint(int x = 0, int y = 0)
        //определение конструктора класса CPoint с двумя параметрами
    {
        this->x = x;
        this->y = y;
    }
    CPoint get_point()
        //метод возвращает объект, для которого
    {
        //он вызван
        return *this;
        //согласно определению функции
    }
    //нужно вернуть объект класса CPoint
    //указатель this содержит адрес этого объекта
    //используя операцию разыменования,
    //получаем сам объект
    void info()
        //определение члена-функции info класса CPoint
    {
        printf("x = %d, y = %d\n", x, y);
        //вывод членов-переменных объекта класса CPoint
    }
};
//конец определения класса CPoint
```



```
int main()
{
    CPoint p1(5, 6), p2, p3(4, 89);
    p1.info();           //будет выведено x=5 y=6
    p2.info();           //будет выведено x=0 y=0
    p3.info();           //будет выведено x=4 y=89
    p3 = p1.get_point(); //объекту p3 присвоен объект p1
    p1.info();           //будет выведено x=5 y=6
    p3.info();           //будет выведено x=5 y=6
    return 0;
}
```

Пояснение. В этом примере функция `get_point` используется для получения объекта класса `CPoint`. Получить адрес объекта внутри любого метода класса можно с помощью указателя `this`. Для получения самого объекта остается только применить операцию разыменования к указателю `this`.

16.5. Дружественные функции

Механизм управления доступом позволяет выделять общедоступные (`public`), защищенные (`protected`) и собственные (`private`) члены классов. Защищенные компоненты доступны внутри класса и в производных классах. Собственные компоненты локализованы в классе и недоступны извне. С помощью общедоступных компонентов реализуется взаимодействие класса с любыми частями программы. Однако имеется еще одна возможность расширить возможности доступа к закрытым членам класса. Ее обеспечивают дружественные функции. По определению, дружественной функцией класса называется функция, которая, не являясь его компонентом, имеет доступ к его защищенным и собственным компонентам. Функция не может стать другом класса «без его согласия». Для получения прав друга функция должна быть описана в теле класса со спецификатором `friend`. Именно при наличии такого описания класс предоставляет функции права доступа к защищенным и собственным компонентам.

Пример. Написать функцию, вычисляющую расстояние между двумя точками:

```
# include <stdio.h>
class CPoint           //начало определения класса CPoint
```

```

{
    friend void length(CPoint, CPoint);
        //объявляем функцию length дружественной классу CPoint
private:
    int x;
    int y;
public:
    CPoint(int, int);
    void prn();
};
CPoint::CPoint(int x = 0, int y = 0)
{
    this->x = x;
    this->y = y;
}
void CPoint::prn()
{
    printf("Координаты точки (%d, %d);\n", this->x,
        this->y);
}
void length(CPoint p1, CPoint p2)          /*реализация функции
        вычисления расстояния между двумя точками*/
{
    float L;
    L = sqrt(pow(p1.x-p2.x, 2.0) + pow(p1.y-p2.y, 2.0));
    printf("Length = %3.2f.\n", L);
}
void main()
{
    CPoint p1(7, 8), p2(3, 2);
    p1.prn();
    p2.prn();
    length(p1, p2);
}

```

Пояснение. После того как мы указали, что функция `length` является дружественной классу, она имеет доступ к его членам, т. е. внутри функции мы можем использовать операторы: `p1.x`; `p1.y`; `p2.x`; `p2.y`. Использование дружественной функции позволяет не нарушить принципы инкапсуляции и в то же время сделать программный код более понятным и наглядным.

Особенностью дружественной функции является то, что ей не передается указатель **this**. Поэтому все данные должны передаваться ей

в качестве параметров. Дружественная функция не является членом класса, а соответственно ее нельзя вызвать, используя операцию «.» или «->». Выше приведен пример, когда дружественная функция являлась глобальной. Возможны варианты, когда дружественная функция является методом другого, ранее определенного класса.

```
class CTest1
{
public:
    void metod()
    {}
};
class CTest2
{
    friend void CTest1::metod();
};
```

В этом примере член функция `metod` класса `CTest1` объявлена дружественной классу `CTest2`.

Любая функция может быть дружественная нескольким классам одновременно.

16.6. Дружественные классы

В языке C++ имеется возможность подружить сразу классы. Это означает, что все методы класса будут являться дружественными для другого класса. Описываются дружественные классы следующим образом:

```
class X2
{
    friend class X1;
    ...
};
class X1
{
    ...
    void f1(...);
    void f2(...);
    ...
};
```



```

{
private:
    CPoint p1, p2;          /*определение закрытых членов переменных
                           класса CОтрезок типа CPoint — отрезок
                           задается точками начала и конца*/

public:
    CОтрезок(int x1, int y1, int x2, int y2);
                           //прототип конструктора класса CОтрезок

    void len();             //метод вычисления длины отрезка класса CОтрезок
};                           //конец определения класса CОтрезок
CОтрезок::CОтрезок(int x1, int y1, int x2, int y2)
                           //определение конструктора класса CОтрезок
{
    p1.x = x1;
                           //так как класс «отрезок» является дружественным
    p1.y = y1;
                           //классу «точка», то можно напрямую обратиться к
    p2.x = x2;
                           //закрытым полям x и y
    p2.y = y2;
}
void CОтрезок::len()
                           //определение функции —члена класса CОтрезок
{
    int L;
    L = p2.x-p1.x;          //получаем доступ к закрытым данным
    printf("L = %d\n", L);  //класса «точка»
}
int main()
{
    CОтрезок otr1(1,10, 11, 20);
    otr1.len();
    return 0;
}

```

Пояснение. В этом примере в классе CОтрезок хранятся значения точек начала и конца отрезка. Сами точки являются объектами другого класса — CPoint. Для упрощения доступа к закрытым полям x и y класс CОтрезок объявлен как дружественный классу CPoint. Это означает, что во всех методах класса CОтрезок (в конструкторе и в методе вычисления длины отрезка) можно получить доступ к данным x и y с использованием операции «.».

16.7. Перегрузка стандартных операций

В C++ существует возможность распространения действия стандартных операций на операнды, для которых эти операции первоначально в языке не предполагались. Например, если `s1` и `s2` — объекты нового класса `MyString`, то их конкатенацию (соединение) удобно было бы обозначить как `s1+s2`. Однако бинарная операция `+` в обычном контексте языка C++ предназначена для арифметических операндов и не предусматривает работу с объектами класса `MyString`.

Чтобы иметь возможность использовать стандартную для языка C++ операцию (например, «`+`» или «`*`») с пользовательскими типами данных, необходимо специальным образом определить, каким образом будут выполняться действия с операндами. В языке C++ для этой цели служат специальные операции-функции, которые задаются особым образом.

Операция-функция определяет алгоритм выполнения перегруженной операции, когда эта операция применяется к объектам класса, для которого операция-функция введена. Для обеспечения явной связи с классом операция-функция должна быть:

- 1) либо компонентом класса;
- 2) либо она должна быть определена в классе как дружественная;
- 3) либо у нее должен быть хотя бы один параметр типа класс (или ссылка на класс).

Кроме того, для правильной перегрузки операторов-функций должны соблюдаться следующие правила:

- число операндов оператора-функции должно совпадать с числом операндов стандартной операции (унарные операции и бинарные);
- нельзя перегружать следующие операции:
 - `::` — операцию изменения доступа;
 - `.` — операцию доступа к члену статической структуры;
 - `->` — операцию доступа к члену динамической структуры;
 - `.*` — операцию доступа к динамическому члену статической структуры;
 - `->*` — операцию доступа к динамическому члену динамической структуры;
 - `?:` — условную операцию;
- при использовании нескольких операторов-функций в выражении приоритет выполнения операций должен совпадать с приоритетом выполнения стандартных операций.

Формат определения операции-функции:

```
тип_результата operator знак_операции (список_параметров)
{
    операторы_тела_операции-функции;
}
```

В этом определении используется ключевое слово `operator`.

При необходимости может добавляться и прототип операции-функции с форматом:

```
тип_результата operator знак_операции (список_параметров);
```

Пример. При работе с точками на геометрической плоскости возникает необходимость складывать и вычитать значения координат точек. Перегрузим операции «=», «+», «-» для объектов класса `CPoint`.

```
# include <stdio.h>
# include <string.h>

class CPoint                                     //определение класса «точка»
{
private:
    int x, y;                                     //координаты точки
public:
    CPoint(int x = 0, int y = 0)                 //определение конструктора класса CPoint
    {
        this->x = x;
        this->y = y;
    }
    int GetX()                                   //определение члена-функции GetX класса CPoint
    {
        return x;
    }
    int GetY()                                   //определение члена-функции GetY класса CPoint
    {
        return y;
    }
}
```

```

void SetXY(int x, int y)
    //определение члена-функции SetXY класса CPoint

{
    this -> x = x;
    this -> y = y;
}

void info()          //определение члена-функции info класса CPoint

{
    printf("x = %d, y = %d\n", x, y);
}

CPoint& operator = (CPoint &p)
    //перезгружаем операцию «=»
    //для объектов класса CPoint
    //по первому способу
{
    this -> x = p.x;
    this -> y = p.y;
    return *this;
}

friend CPoint& operator - (CPoint &, CPoint &);
    /*прототип дружественного оператора
    функции «-» объектов класса CPoint по второму способу*/
};

CPoint& operator - (CPoint &p1, CPoint &p2)
    //перезгружаем операцию
    //«-» для объектов класса CPoint по второму способу
{
    static CPoint temp;
    temp.x = p1.x - p2.x;
    temp.y = p1.y - p2.y;
    return temp;
}

CPoint& operator + (CPoint &p1, CPoint &p2)
    //перезгружаем операцию
    //«+» для объектов класса CPoint по третьему способу
{
    static CPoint temp;
    int temp_x, temp_y;
    temp_x = p1.GetX() + p2.GetX();
    temp_y = p1.GetY() + p2.GetY();
}

```

```

    temp.SetXY(temp_x, temp_y);
    return temp;
}
int main()
{
    CPoint p1, p2(4, 9), p3(56, 3);
    p1.info();           //будет выведено x=0 y=0
    p2.info();           //будет выведено x=4 y=9
    p3.info();           //будет выведено x=56 y=3
    p1 = p2;             //проверяем работу перегруженной операции «=»
    p1.info();           //будет выведено x=4 y=9
    p1 = p2 + p3;
                        //сначала выполняется операция «+», а затем «=»
    p1.info();           //будет выведено x=60 y=12
    p2 = p3 - p1;
                        //сначала выполняется операция «-», а затем «=»
    p2.info();           //будет выведено x=-4 y=-9
    return 0;
}

```

Пояснение. В программе определен класс `CPoint`, в котором хранятся координаты точки x и y . Для полноценной работы с этими данными определен полный набор методов.

Операция «=» перегружается как метод класса, так как другими способами она не может быть перегружена. Прототип такой перегрузки имеет вид:

```
CPoint& operator = (CPoint &p);
```

Операция «=» является бинарной, т. е. при ее использовании требуется два операнда. Однако при перегрузке операция-функция имеет единственный параметр, связано это с тем, что вторым операндом выступает сам объект, для которого выполняется эта операция. То есть выражение `p1 = p2`; можно рассматривать как неявный вызов операции «=», компилятор же преобразовывает такую форму записи к явному виду: `p1.operator=(p2)`. Для пользователя первый вариант записи является более привычным, поэтому именно он используется в работе.

Операция «-» перегружается по второму способу. Эта операция-функция объявляется дружественной классу, что позволяет получить доступ к закрытым полям *x* и *y*.

```
CPoint& operator - (CPoint &p1, CPoint &p2)
{
    static CPoint temp;
    temp.x = p1.x - p2.x;
    temp.y = p1.y - p2.y;
    return temp;
}
```

Обратите внимание, что для хранения результата вычисления разности координат двух точек создается временный объект — точка *temp*. При объявлении этой переменной используется модификатор *static*, служащий для изменения времени жизни переменной. Такая необходимость связана с тем, чтобы временный объект *temp* не был сразу уничтожен по окончании выполнения операции функции и данные из него были переписаны в нужный объект.

Операция «+» перегружается по третьему способу. Операция-функция является глобальной, а ее параметры являются ссылками на объекты класса *CPoint*. В этом случае внутри функции нет доступа к закрытым полям класса *CPoint*, поэтому для задания координат точки необходимо пользоваться методами *set/get*.

Аналогичным образом можно перегрузить другие операции для класса *CPoint*. Приведем пример перегрузки унарной операции «++».

```
class CPoint
{
    ...
    CPoint& operator ++()
    {
        x++;
        y++;
        return *this;
    }
    ...
};
```

В основной программе возможно следующее использование этой операции:

```
...
CPoint p(4, 8);
p.info();
p1++;
p.info();
...
```

Так как операция является унарной, то она работает с одним операндом. Поэтому в прототипе операции-функции отсутствуют параметры. Неявный вызов `p1++`; представляется компилятором в явном виде: `p1.operator++()`.

Пример. Создать класс `CVector`. Класс должен включать:

- конструктор с одним параметром — число элементов массива и выделять память под массив;
- деструктор для освобождения памяти под массив;
- член-функцию генерации массива с помощью датчика случайных чисел;
- член-функцию вывода массива на экран;
- перегруженную операцию присваивания;
- перегруженную операцию извлечения элемента из массива;
- перегруженную постфиксную операцию инкремента, увеличивающую на единицу каждый элемент массива;
- перегруженную префиксную операцию инкремента, увеличивающую на 10 каждый элемент массива.

Определить дружественную функцию — перегруженный оператор прибавления к каждому элементу массива вещественного числа, являющегося параметром функции.

```
class CVector                                //начало определения класса CVector
{
private:                                     //закрытые элементы
    int N;                                  //количество элементов в массиве
    float *px;                              //указатель на начало массива
public:
    CVector(int n = 1);                     //конструктор с параметром и по умолчанию
    ~CVector();                             //деструктор
    void prn();                             //печать значений массива на экран
    void gen();                             //генерация случайных элементов массива
```

```

CVector& operator = (CVector &v);
                                //прототип перегруженной операции «=»
float operator[](int k);        /*прототип перегружен-
                                ной операции извлечения элемента из массива*/
CVector& operator++();          /*прототип перегруженной
                                постфиксной операции инкремента*/
CVector& operator++(int k);     //прототип перегруженной
                                //префиксной операции инкремента
friend CVector& operator + (CVector &v1, float c)
    /*прототип перегруженной операции сложения с числом типа float*/

};
CVector::CVector(int n)
                                //определение конструктора класса CVector
{
    N = n;
                                //в переменной N сохраняем количество элементов массива
    px = new float [N];          //выделяем память для массива
}
CVector::~CVector()              //реализация деструктора
{
    delete [] px;                //освобождаем выделенную память
}
void CVector::gen()              //метод генерации случайных значений
{                                //элементов массива
    for (int i = 0; i<N; i++)
        px[i] = float(rand())/RAND_MAX*5;
}
CVector& CVector::operator = (CVector &v)
                                //перегружаем операцию «=»
{                                //для объектов класса CVector
    this->N = v.N;                //по первому способу
    delete []this->px;            //освобождение памяти для указателя
    this->px=new float [N];        //выделение памяти для нового массива
    for (int i=0; i<N; i++)
        this->px[i]=v.px[i];     /*копируем элементы массива объекта v в
                                объект, которому принадлежит метод*/
    return *this;
}
float CVector::operator[](int k)
                                //перегрузка операции извлечения элемента из массива
{ if ((i>=0)&&(i<N))

```



```

    return this->px[k];
    else return MAX RAND;
}

CVector& CVector::operator++()
    //определение перегруженной постфиксной операции инкремента
{for (int i=0; i<this->N; i++)
    this->px[i]++; /*использование стандартной операции инкремента
                    для увеличения элемента массива на 1*/
return *this;
}

CVector& CVector::operator++(int k)
    //определение перегруженной префиксной операции инкремента
{for (int i=0; i<this->N; i++)
    this->px[i]+10;
return *this;
}

void CVector::prn()
{
    for (int i = 0; i<N; i++)
        //в цикле выводим значения элементов
        printf("%3.2f\t", px[i]); //массива на экран
        printf("\n");
}

CVector& operator + (CVector &v1, float c) //перегружаем
                                           //операцию
{
    //«+ число типа float» для объектов класса CVector по второму способу
    static CVector temp;
    temp.N = v1.N ;
    temp = new float [temp.N]
        for (int i=0; i<temp.N; i++)
            temp.px[i] = v1.px[i] +c;
    return temp;
}

void main()
{
    srand(time(NULL)); //запускаем генератор случайных чисел
    CVector massiv(3); //создаем массив из трех элементов
    massiv.gen(); //генерируем случайные значения массива
    massiv.prn(); //выводим значения массива на экран
    CVector mass(0); //создаем массив из 0 элементов
    mass.gen(); //генерируем случайные значения массива из 0 элементов
}

```

```

    mass.prn();
                                //выводим значения массива из 0 элементов на экран
mass=massiv;
                                //присваиваем значение объекта massiv объекту mass
mass.prn();
                                //выводим значения массива mass на экран
    CVector mass1(0);
                                //создаем массив из 0 элементов
    mass1.gen();
                                //генерируем случайные значения массива из 0 элементов
    mass1.prn();
                                //выводим значения массива из 0 элементов на экран
mass1=mass+3.5;
                                //складываем массив mass с числом 3.5
mass1.prn();
                                //выводим значения массива mass1 на экран
mass1++;
                                //применяем постфиксную операцию инкремента к массиву mass1
mass1.prn();
                                //выводим значения массива mass1 на экран
++mass1;
                                //применяем префиксную операцию инкремента к массиву mass1
mass1.prn();
                                //выводим значения массива mass1 на экран

printf("mass1[1]=%f\n",mass1[1]);
}

```

Результат работы программы:

```

2.53      3.62      3.42

2.53      3.62      3.42

6.03      7.12      6.92
7.03      8.12      7. 92
17.03 18.12      17.92
mass[1] = 18.12

```

Пояснение. В этом примере используется операция присваивания, которую можно определить только как член класса. Так как массив в классе является динамическим, необходимо сначала освободить ранее выделенную память, чтобы не возникало сбоев в случае несовпадения количества элементов в объектах слева и справа от знака присваивания.

Затем объекту приемнику (*this) присваивается число элементов объекта источника (стоящего справа от знака присваивания) и происходит поэлементное копирование массива.

Операция извлечения элемента массива по его индексу «[]» также является бинарной, первым операндом является объект, вызвавший эту операцию, а вторым — индекс элемента в массиве (mass1[1]).

По своему назначению и по синтаксису записи эта операция, так же как и операция присваивания, должна перегружаться как член класса и возвращать значение типа, совпадающего с типом элементов массива.

Унарные операции, в частности операции инкремента и декремента, можно перегрузить всеми тремя указанными ранее способами. В данном примере показана особенность перегрузки префиксных и постфиксных операций инкремента как членов класса. Если необходимо перегрузить их разным образом (как в примере), то префиксные операции члены класса обязательно должны иметь фиктивный параметр целого типа, который никак не используется и передается в префиксную операцию со значением 0. Постфиксные операции (члены класса) инкремента и декремента перегружаются без параметров.

Тесты

1. Укажите правильный вариант объявления константы в качестве члена класса:

- а) `const int A;`
- б) `const int A = 34;`
- в) `const int A (34).`

2. Что такое список инициализации?

- а) такое понятие не используется в ООП;
- б) это операторы, предназначенные для задания начальных значений объекту класса;
- в) это операторы, предназначенные для задания начальных значений констант, используемых в классе.

3. Укажите правильный вариант инициализации целочисленных констант A и B в классе:

- а)

```
klass::klass(int a, int b) : A(a), B(b)
{
    ...
}
```
- б)

```
klass::klass(int a, int b) : A= a, B = b
{
    ...
}
```
- в)

```
klass::klass(int a, int b) : A(a) B(b)
{
    ...
}
```


4. Выберите верное утверждение:

- а) статический член класса не может менять своего значения;
- б) статический член класса обязательно должен быть проинициализирован;
- в) оба утверждения верны.

5. Что будет выведено на экран в результате выполнения следующего кода?

```
class klass
{
public:
    static int a;
    klass()
    {
        a++;
    }
    ...
};
int klass::a = 0;
...
klass k1, k2, k3;
printf("a = %d\n", klass::a);
...
```

- а) a = 2;
- б) a = 3;
- в) программа не запустится, так как получить доступ к полю a, используя оператор klass::a, нельзя.

6. Можно ли в классе объявить следующий метод?

```
static void klass::info()
{
    printf("a = %d b = %d.\n", A, B);
}
```

- а) да, если переменные A и B являются статическими;
- б) да, в любом случае;
- в) нет, в любом случае.

7. Выберите верное утверждение:

- а) указатель this содержит адрес объекта класса, для которого вызван метода класса;
- б) указатель this содержит адрес первого созданного объекта класса;

в) указатель `this` содержит адрес объекта, записанный в него программистом.

8. Какое ключевое слово используется для объявления дружественной функции?

- а) `friend`;
- б) `friendship`;
- в) `friendly`.

9. Укажите правильный вариант прототипа перегрузки операции-функции «+» как метода класса `CPoint`:

- а) `friend CPoint& operator + (CPoint &, CPoint &);`
- б) `CPoint& operator + (CPoint &, CPoint &);`
- в) `CPoint& operator + (CPoint &, CPoint &).`

10. Может ли прототип перегрузки унарной операции-функции «++» иметь следующий вид?

`CComplex& operator ++();`

- а) нет, в прототипе должны быть указаны типы параметров;
- б) да, в любом случае прототип будет иметь такой вид;
- в) да, если операция-функция перегружается как метод класса.

Задания

1. Разработать класс «студент». Члены класса: ФИО, номер группы, год рождения. Поле «год рождения» объявить как константу. Привести примеры использования разработанного класса.

2. Разработать класс «геометрическая фигура». Члены класса: тип фигуры ('о' — окружность, 'к' — квадрат, 'т' — треугольник и т. д.). Методы класса: конструктор, деструктор, вычисление площади фигуры. Добавить в класс статический член, служащий для подсчета количества созданных объектов этого класса, и статическую функцию, выводящую на экран количество объектов этого класса.

3. Написать две функции вычисления длины отрезка, заданного двумя точками. Одну из этих функций объявить дружественной классу `CPoint`. Объяснить особенности реализации этих функций.

4. Написать класс, предназначенный для хранения и обработки информации о фамилии, имени и отчестве человека. С использованием этого класса создать класс для хранения и обработки информации об автомобилях (поля: ФИО владельца, год выпуска и т. д.). Для получения доступа к закрытым полям класса использовать дружбу классов.

5. Перегрузить операции «=», «+», «-», «*», «/» для класса `CVector`, предназначенного для работы с одномерными целочисленными массивами.

6. Перегрузить операции «=», «+», «-», «*», «/» для класса `CMatrix`, предназначенного для работы с вещественными матрицами.

7. Написать класс для работы с комплексными числами. Перегрузить стандартные операции для этого класса.

Контрольные вопросы

1. Каким образом можно использовать константу в качестве члена класса? Как проинициализировать константу?
2. Как в списке инициализации задать значения нескольким константам?
3. Что такое статический член класса? Чем он отличается от обычных членов?
4. Как объявить статический метод класса? Какими особенностями обладают такие методы?
5. Что такое указатель `this`? Как он определяется в классе? Какая информация хранится в этом указателе? Приведите примеры использования указателя `this`.
6. Как объявить функцию дружественной классу? Какие преимущества имеет дружественная функция?
7. Как объявить класс дружественным другому классу? Что означает «дружба» классов?
8. Для чего нужна перегрузка стандартных операций? Сколькими способами можно реализовать перегрузку операций?
9. Приведите пример синтаксиса перегрузки стандартных операций.
10. Что такое явный и неявный вызов перегруженных операций-функций?

Глава 17

НАСЛЕДОВАНИЕ, ПОЛИМОРФИЗМ

Объектно-ориентированное программирование (ООП) базируется на трех основных принципах (трех китах ООП):

- 1) инкапсуляция;
- 2) полиморфизм;
- 3) наследование.

Инкапсуляция — основной принцип абстракции данных — сокрытие данных внутри объекта. Инкапсуляция была подробно рассмотрена в предыдущей главе.

Полиморфизм — возможность программного кода работать с разными объектами одинаковым образом. Другими словами — это свойство, которое позволяет одно и то же имя использовать для решения нескольких схожих, но технически разных задач.

Наследование — создание новых классов и их иерархии на основе уже имеющихся. Другими словами — это процесс, посредством которого один объект может приобретать свойства другого. Наследование (inheritance) — механизм получения нового класса из существующего. Существующий класс может быть дополнен или изменен для создания производного класса. Наследование — это мощный механизм повторного использования кода. С помощью наследования может быть создана иерархия родственных типов, которые совместно используют код и интерфейсы.

17.1. Наследование

Наследованием называется механизм создания производных классов (классов-наследников) на основе базового класса. Производные классы унаследуют доступные им поля и методы родительского класса. Дополнительно добавляются поля и методы, присущие только

производным классам, и, возможно, переопределяются некоторые методы родительского класса.

Определение типа производного класса. Синтаксис при определении производного класса (класса-наследника) следующий:

```
class <имя_производного_класса>:<спецификатор_доступа>
имя_базового_класса
{...};
```

Здесь <имя_производного_класса> — имя класса-наследника, унаследовавшего от <базового_класса> некоторые поля и методы, а также имеющего дополнительные поля и методы по сравнению с базовым классом. Имена производного и базового класса задаются по правилам задания идентификаторов.

<спецификатор_доступа> определяет способ доступа к элементам базового класса, функциям-членам производного классов, а также другим функциям программы (табл. 17.1).

Таблица 17.1. Действия спецификаторов доступа при наследовании

Доступ в базовом классе	Спецификатор доступа	Доступ в производном классе
public	Отсутствует	private
protected	Отсутствует	private
private	Отсутствует	Недоступны
public	public	public
protected	public	protected
private	public	Недоступны
public	protected	protected
protected	protected	protected
private	protected	Недоступны
public	private	private
protected	private	private
private	private	Недоступны

Например, создадим класс CCircle на базе класса CPoint.

```
#include <stdio.h>
class CPoint{                               //начало определения базового класса CPoint
protected:                                /*объявление членов класса, доступных
                                           только этому классу и его наследникам*/
```

```

int x;                //член-переменная класса CPoint
int y;                //член-переменная класса CPoint
public:
    //объявление методов класса, доступных всей программе
void SetX(int _x)      /*функция-член класса CPoint, задающая
                        значение члену-переменной x*/
void SetY(int _y)      //функция-член класса, задающая значение
                        //члену-переменной y
{y=_y;}
int GetX()             //функция-член класса, возвращающая значение
                        //члена-переменной x
{return x;}
int GetY()             //функция-член класса, возвращающая значение
                        //члена-переменной y
{return y;}
};                    //конец определения базового класса CPoint
class CCircle: public CPoint{    /*начало определения
                                производного класса CCircle, построенного
                                на основе базового класса CPoint*/
protected:              /*квалификатор доступа для членов класса,
                        доступных только этому классу и его наследникам*/
    int r;                //член-переменная класса
public:                  /*квалификатор доступа для членов
                        класса, доступных всей программе*/
void SetR(int _r)        //функция-член класса, задающая значение
                        //члену-переменной r
{r=_r;}
int GetR()               //функция-член класса, возвращающая значение
                        //члена-переменной r
{return r;}
};                    //конец определения производного класса CCircle
int main() {
    CPoint A;             //объявление (создание) объекта A класса CPoint
    A.SetX(3);             //задание значения 3 члену-переменной x объекта A
    A.SetY(7);             //задание значения 7 члену-переменной y объекта A
    printf("Tochka (%d, %d);\n", A.GetX(), A.GetY());
    CCircle B;            //объявление объекта B класса CCircle
    B.SetX(30);            //задание значения 30 члену-переменной x объекта B
    B.SetY(20);            //задание значения 20 члену-переменной y объекта B

    printf("Okrugnost (%d, %d, %d);\n", B.GetX(),
        B.GetY(), B.GetR());
    return 0;
}

```

В этом примере класс `CCircle` наследует поля `x`, `y`, методы `SetX`, `SetY`, `GetX`, `GetY` класса `CPoint`. Поле `r`, методы `SetR` и `GetR` присущи только классу `CCircle`.

При наследовании важную роль играет спецификатор доступа компонентов. В табл. 17.1 указывается статус доступа при наследовании. Как видно из таблицы, отсутствие спецификатора эквивалентно спецификатору `private`.

Применение различных спецификаторов доступа показано в следующей программе.

Пример 2. Различные спецификации при наследовании.

```
# include <stdio.h>

class CBase                                //начало определения базового класса
{private:
    int pri;                                //объявление закрытой члена-переменной pri
    protected:
    int pro;                                //объявление защищенной члена-переменной pro
public:
    int pub;                                //объявление общедоступной члена-переменной pub
};                                           //конец определения базового класса
class C1 : CBase                            //начало определения класса C1 наследника класса CBase
{
public:
    void set(int i);                        //прототип общедоступного метода set
    void prn();                             //прототип общедоступного метода prn
};
void C1::set(int i)                         //общедоступный метод set класса C1
{                                           /*pri = i; -член-переменная базового класса pri
                                           недоступна нигде, кроме класса CBase*/

    pro = i;
    pub = i;}
void C1::prn()                             //общедоступный метод prn класса C1
{
    printf("Class C1 (без спецификатора):\n");
    printf("pri - недоступна.\npro = %d.\npub = %d.\n",
        pro, pub);
}
class C2 : public CBase                    //начало определения класса C2 наследника класса CBase
{public:
    void set(int i);                        //прототип общедоступного метода set
    void prn();                             //прототип общедоступного метода prn
    void C2::set(int i)
```

```
{
//pri = i; — член-переменная pri недоступна нигде, кроме класса CBase
    pro = i;
    pub = i;
}
void C2::prn()
{
    printf("Class C2 (спецификатор -public):\n");
    printf("pri - недоступна.\npro = %d.\npub = %d.\n",
        pro, pub);
}
class C3 : protected CBase
    //начало определения класса C3 наследника класса CBase

{public:
    void set(int i);
    void prn();};
    void C3::set(int i)
{
//pri = i; — член-переменная pri недоступна нигде, кроме класса CBase
    pro = i;
    pub = i;
}
void C3::prn()
{printf("Class C3 (спцификатор - protected):\n");
printf("pri - недоступна.\npro = %d.\npub = %d.\n",
pro, pub);
}
class C4 : private CBase
    //начало определения класса C4 наследника класса CBase

{public:
    void set(int i);
    void prn();};
    void C4::set(int i)
{
//pri = i; — член-переменная pri недоступна нигде, кроме класса CBase
    pro = i;
    pub = i;
}
void C4::prn()
{
    printf("Class C4 (спецификатор - private):\n");
    printf("pri - недоступна.\npro = %d.\npub = %d.\n",
        pro, pub);
}
```

```

int main() {
    CBase b1;          /*b1.pri = 0; — член-переменная pri объекта b1
                       недоступна нигде, кроме объекта b1 класса CBase*/

    /*b1.pro = 0; — член-переменная pro объекта b1 недоступна нигде,
                       кроме объекта b1 класса CBase и объектов-наследников
                       класса CBase*/

    b1.pub = 0;        //общедоступная член-переменная объекта b1
    C1 c1;

    /*c1.pri = 0; — член-переменная pri объекта c1
                       недоступна нигде, кроме объекта c1 класса CBase*/
    /*c1.pro = 0; — член-переменная pro объекта c1
                       недоступна нигде, кроме объекта c1 класса C1*/
    /*c1.pub = 0; — член-переменная pub объекта c1
                       недоступна нигде, кроме объекта c1 класса C1*/

    c1.set(1);
    c1.pri();
    C2 c2;              /*c2.pri = 0; — член-переменная pri объекта c2
                       недоступна нигде, кроме объекта c2 класса CBase*/
    /*c2.pro = 0; — член-переменная pro объекта c2 недоступна
                       нигде, кроме объекта c2 класса CBase и объектов-наследников класса C2*/
    c2.pub = 0;
    c2.set(10);
    c2.pub = 9;
    c2.pri();
    C3 c3;

    /*c3.pri = 0; — член-переменная pri объекта c3
                       недоступна нигде, кроме объекта c3 класса CBase*/
    /*c3.pro = 0; — член-переменная pro объекта c3 недоступна
                       нигде, кроме объекта c3 класса C3 и наследников объекта c3*/
    /*c3.pub = 0; — член-переменная pub объекта c3 недоступна
                       нигде, кроме объекта c3 класса C3 и наследников объекта c3*/

    c3.set(5);
    c3.pri();
    C4 c4;

    /*c4.pri = 0; — член-переменная pri объекта c4
                       недоступна нигде, кроме объекта c4 класса CBase*/
    /*c4.pro = 0; — член-переменная pro объекта c4
                       недоступна нигде, кроме объекта c4 класса C4*/
    /*c4.pub = 0; — член-переменная pub объекта c4
                       недоступна нигде, кроме объекта c4 класса C4*/

    c4.set(15);
    c4.pri();
    return 0;
}

```


17.2. Конструкторы производного и базового классов

Перед выполнением программы компилятор резервирует память под объект производного класса, т. е. под члены-переменные и члены-функции объекта. Во время выполнения программы в момент создания объекта производного класса вызывается конструктор производного класса. Первым делом в этом конструкторе вызывается конструктор базового класса. После выполнения всех операторов конструктора базового класса начинают выполняться операторы конструктора производного класса. Операторы в конструкторах служат для задания начальных значений переменных, используемых в классе, настройке различных связей и прочих необходимых действий.

Конструкторы не наследуются. Перед вызовом конструктора производного класса необходимо вызвать конструктор базового класса. Если такой конструктор не определен, то он будет сгенерирован автоматически.

```
имя_производного_класса(параметры_конструктора) :  
имя_базового_класса(параметры_базового_конструктора)  
{  
    ...  
}
```

Например:

```
CCircle::CCircle(int _r) : CPoint(0,0)  
{...};
```

Вызываем конструктор базового класса с параметрами (0, 0).

Пример 1. На базе класса `Chisla` породить новый класс «данные», предназначенный для хранения переменных целого, вещественного и символьного типов.

```
#include <stdio.h>  
  
class chisla  
{  
public:  
    int a;  
    float x;  
    chisla()  
{  
    printf("Выполняется конструктор базового класса\n");  
}  
};
```

```
class data : public chisla
{
public:
    char c;
    data() : chisla()
    {
        printf("Выполняется конструктор производного
            класса\n");
    }
};
void prn(chisla &ch)
{
    printf("a = %d\tx = %f\n", ch.a, ch.x);
}
int main()
{
    data m1;
    return 0;
}
```

Результат работы программы:

Выполняется конструктор базового класса
Выполняется конструктор производного класса

Пояснение. При создании объекта производного класса вызывается конструктор базового класса, а после окончания его работы выполняется конструктор производного класса. Об этом свидетельствуют выводимые на экран информационные сообщения.

Пример 2. Создать класс CCircle на основе класса CPoint.

```
#include <stdio.h>
#include <math.h>
class CPoint
{
protected:
    int x, y;
public:
    CPoint(int _x = 0, int _y = 0)
    {
        x = _x; y = _y;
    }
};
```

```
class CCircle: public CPoint
{
protected:
    int R;
public:
    CCircle(int _x, int _y, int _r): CPoint(_x, _y)
    {
        R = _r;
    }
    void print()
    {
        printf("Centr: (%d, %d). R = %d.\n", x, y, R);
    }
};
int main()
{
    CCircle c1(4, 7, 8);
    c1.print();
    return 0;
}
```

Пояснение. Обратим внимание на то, что в исходном классе CPoint члены x и y объявлены со спецификатором protected, это позволяет получить к ним доступ в производном классе.

В строке

```
CCircle(int _x, int _y, int _r): CPoint(_x, _y)
```

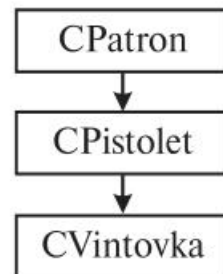
происходит вызов конструктора базового класса. Если вызов конструктора базового класса убрать из этой строчки, то автоматически будет вызван конструктор по умолчанию (не имеющий параметров и задающий координатам x и y нулевые значения) для класса CPoint. В основной программе создается объект класса «окружность» с координатами центра (4, 7) и радиусом, равным 8.

17.3. Иерархия классов

Отметим, что язык C++ позволяет создавать иерархию классов, т. е. производный класс может служить базовым для другого класса.

Пример. Создать базовый класс «патрон». Члены класса: калибр патрона. Методы класса: вывод калибра патрона на экран. Создать производные классы — «пистолет» и «винтовку». Класс «пистолет» со-

держит дополнительные данные: скорострельность и прицельную дальность, «винтовка» также должна содержать информацию о ее размере и весе. Иерархия классов будет иметь вид:



```

#include <stdio.h>
class CPatron                                //начало определения базового класса
{
protected:
    float kalibr;                            //защищенная член-переменная
public:
    CPatron(float);                          //прототип конструктора класса CPatron
    void Fire();                             //прототип члена-функции класса CPatron
};                                           //конец определения класса CPatron

CPatron::CPatron(float k)                   //определение конструктора класса CPatron
{
    kalibr = k;
}
void CPatron::Fire() //определение функции-члена класса CPatron
{
    printf("Kalibr patrona = %3.2f\n", kalibr);
}
class CPistolet : public CPatron
    //начало определения класса CPistolet наследника класса CPatron
{
protected:
    int skorost;                            //защищенная член-переменная класса CPistolet
    int dalnost;                            //защищенная член-переменная класса CPistolet
public:
    CPistolet(float kal, int sk, int dal);
                                           //прототип конструктора класса CPistolet
    void Fire();                            //прототип члена-функции класса CPistolet
};                                           //конец определения класса CPistolet

/*определение конструктора класса CPistolet с инициализирующим
вызовом конструктора класса CPatron*/
CPistolet::CPistolet(float kal, int sk, int dal) :
CPatron(kal)
  
```

```

{
    skorost = sk;
    dalnost = dal;
}
void CPistolet::Fire()
    //определение члена-функции класса CPistolet
{
    CPatron::Fire();
    //вызов одноименной члена-функции класса CPatron
    printf("Skorostrelnost = %d\n", skorost);
    printf("Pricelnaya dalnost = %d\n", dalnost);
}

class CVintovka : public CPistolet
    //начало определения класса CVintovka наследника класса CPistolet
{
protected:
    float razmer;    //защищенная член-переменная класса CVintovka
    float ves;       //защищенная член-переменная класса CVintovka
public:
    CVintovka(float kal, int sk, int dal, float raz,
        float v);    //прототип конструктора класса CVintovka
    void Fire();     //прототип члена функции класса CVintovka
};
    //конец определения класса CVintovka
    /*определение конструктора класса CVintovka с
    инициализирующим вызовом конструктора класса CPistolet*/
CVintovka::CVintovka(float kal, int sk, int dal, float
    raz, float v):CPistolet(kal, sk, dal)
{
    razmer = raz;
    ves = v;
}
void CVintovka::Fire()
    //определение члена-функции класса CVintovka
{
    CPistolet::Fire();
    //вызов одноименной функции-члена класса CPistolet
    printf("Razmer vintovki = %3.2f\n", razmer);
    printf("Ves vintovki = %3.2f\n", ves);
}
int main()
{
    printf("Class CPatron:\n\n");
    CPatron pat1(9.1);

```

```
    pat1.Fire();
    printf("\nClass CPistolet:\n\n");
    CPistolet pist1(6.1, 10, 500);
    pist1.Fire();
    printf("\nClass CVintovka:\n\n");
    CVintovka vin1(5.1, 5, 2500, 1.6, 2.6);
    vin1.Fire();
    return 0;
}
```

Пояснение. Данные в исходном классе `CPatron` объявлены со спецификатором `protected`. То есть доступ к ним в наследуемых классах должен осуществляться путем использования методов класса. Использование родительских конструкторов и одноименных методов позволяет эффективно использовать ранее написанный код, тем самым уменьшая возможность появления ошибок в программе.

17.4. Вызов конструкторов и деструкторов

Так как при создании объекта производного класса сначала выполняются операторы конструктора базового класса, а уже потом операторы конструктора производного, то необходимо учитывать эту особенность при работе с динамической памятью. Необходимо в правильной последовательности освобождать ранее выделенную память.

Деструкторы, аналогично конструкторам, не наследуются, т. е. если в производном классе отсутствует деструктор, то он не будет унаследован из базового класса, а будет автоматически сформирован компилятором.

При уничтожении объекта производного класса сначала выполняются операторы деструктора производного класса, а потом операторы деструктора базового класса. То есть вызов деструкторов выполняется в противоположной последовательности по сравнению с конструкторами.

Приведем пример, отображающий последовательность выделения памяти для хранения переменных и ее освобождения при разрушении объектов.

```
#include <stdio.h>

class pointer
{
private:
    int *pa;
```



```
public:
    pointer()
    {
        printf("Выделяем память для членов базового
               класса\n");
        pa = new int;
        *pa = 456;
    }
    ~pointer()
    {
        printf("Освобождаем память в базовом классе\n");
        delete pa;
    }
    void info()
    {
        printf("*pa = %d\n", *pa);
        //выводим на экран значение переменной
    }
};

class pointer_s : public pointer //производный класс
{
private:
    float *px;
public:
    pointer_s() : pointer()
        //в конструкторе производного класса вызываем конструктор базового
    {
        printf("Выделяем память в производном классе\n");
        px = new float;
        *px = 5.67;
    }
    ~pointer_s()
    {
        printf("Освобождаем память в производном классе\n");
        delete px;
    }
    void info()
    {
        pointer::info(); //вызываем метод базового класса
        printf("*px = %4.2f\n", *px);
        //выводим на экран значение переменной
    }
};
```

```
int main()
{
    pointer_s pp;
    pp.info();
    return 0;
}
```

Результат работы программы:

```
Выделяем память для членов базового класса
Выделяем память для членов производного класса
*pa = 456
*px = 5.67
Освобождаем память в производном классе
Освобождаем память в базовом классе
```

Пояснение. При создании объекта производного класса (`pointer_s pp;`) сначала выполняются операторы конструктора базового класса. В базовом классе `pointer` объявлен указатель на целочисленный тип данных. В конструкторе выделяется память и в нее записывается значение переменной, равное 456. После этого в конструкторе производного класса выделяется память для хранения вещественной переменной. После вывода на экран значений переменных `pa` и `px` программа завершается и вызывается деструктор производного класса, а после окончания его работы выполняются операторы деструктора базового класса. Порядок уничтожения объекта противоположен по отношению к порядку его конструирования. Результаты работы программы наглядно демонстрируют очередность вызова конструкторов и деструкторов базового и производного классов.

17.5. Множественное наследование

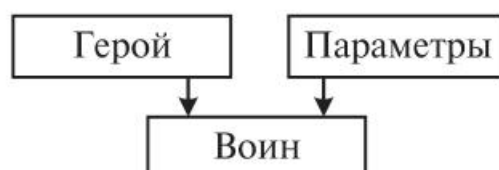
В общем случае класс может быть порожден от любого числа базовых классов, например:

```
class X1 {...};           //определение 1-го базового класса
class X2 {...};           //определение 2-го базового класса
class X3 {...};           //определение 3-го базового класса
class Y : public X1, public X2, public X3 {};
                        //определение класса Y наследника классов X1, X2, X3
```

Такую процедуру создания производных классов называют множественным наследованием.

Пример 1. Написать класс «герой». Члены класса: имя героя, его возраст. Написать класс «параметры». Члены класса: сила, защита, скорость. На базе этих двух классов написать класс «воин», включающий в себя все перечисленные члены.

Иерархия классов будет выглядеть следующим образом:



```

#include <stdio.h>
#include <string.h>
class Hero          //начало определения 1-го базового класса Hero
{
protected:
    char name[32];    //защищенная член-переменная name класса Hero
    int age;          //защищенная член-переменная age класса Hero
public:
    Hero()            //определение конструктора класса Hero без параметров
    {
        strcpy(name, "NoName");
                        //копирование в член-переменную name текста NoName
        age = 0;       //присвоить члену-переменной age значения 0
    }
    Hero(char* name, int age)
        //определение конструктора класса Hero с двумя параметрами
    {
        strcpy(this->name, name);    /*копирование члена-переменной
                                        name (this->name) значения параметра name*/
        this->age = age;
        //присвоить члену-переменной age (this->age) значение параметра age
    }
    void info()        //определения функции-члена info
    {
        printf("Hero:%s, %d\n", name, age);
        //вывод на экран названия класса и значений членов-переменных name и age
    }
};
                        //конец определения базового класса Hero
class Parametry       //начало определения базового класса Parametry
{
protected:
    int sila;          //защищенная член-переменная sila класса Parametry
    int zashita;
                        //защищенная член-переменная zashita класса Parametry

```



```

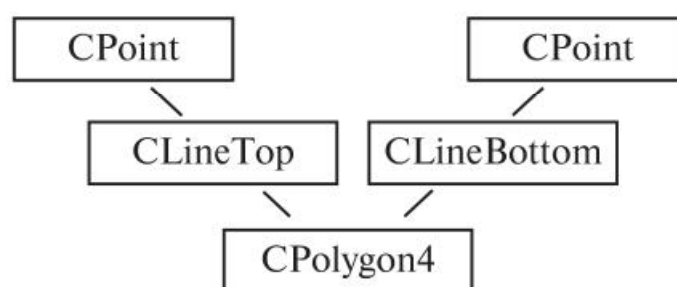
    int skorost;
        //защищенная член-переменная skorost класса Parametry
public:
    Parametry(int _sila = 0, int _zashita = 0,
        int _skorost = 0)
        //конструктор класса Parametry со значениями по умолчанию
    {
        sila = _sila;
        zashita = _zashita;
        skorost = _skorost;
    }
void info() //определение члена-функции info класса Parametry
{
    printf("Parametry: %d, %d, %d\n", sila, zashita,
        skorost); /*вывод на экран названия класса и значений
        членов-переменных sila, zashita, skorost*/
}
}; //конец определения класса Parametry
class Warrior : public Hero, Parametry
    //начало определ. класса Warrior наследника классов Hero, Parametry
{
public:
    /*определение конструктора класса Warrior инициализацией
    «родительских» конструкторов классов Hero и Parametry*/
    Warrior(char* _name, int _age, int _sila, int _zashita,
        int _skorost): Hero(_name, _age), Parametry(_sila,
        _zashita, _skorost)
    {
    }
void info() //определение члена-функции info класса Warrior
{
    printf("Warrior:\n");
        //вывод на экран названия класса Warrior
    Hero::info(); //вызов функции-члена info класса Hero
    Parametry::info();
        //вызов функции-члена info класса Parametry
}
}; //конец определения класса Warrior
int main()
{
    Warrior w("Ivan", 30, 100, 300, 15);
    w.info();
    return 0;
}

```

Обратим внимание на то, что создание нового класса «воин» полностью основывается на уже существующих классах. Создание нового объекта происходит с помощью конструктора, при этом вызываются конструкторы базовых классов. Так как в данном случае число базовых классов равно двум, то происходит вызов обоих конструкторов, при этом они отделяются друг от друга запятой.

Пример 2. На базе класса CPoint породить два класса CLineTop и CLineBottom, на основе которых породить класс «четырёхугольник».

Иерархия классов:



```

#include <stdio.h>
class CPoint          //начало определения базового класса CPoint
{
protected:
    int x, y;
public:
    CPoint(int x = 0, int y = 0)          //конструктор
    {
        this -> x = x;
        this -> y = y;
    }
    void prn()
    {
        printf(" (%d, %d)\n", x, y);
    }
    void setxy(int x, int y)
    {
        this -> x = x;
        this -> y = y;
    }
};          //конец определения базового класса CPoint

class CLineTop : public CPoint
          //начало определения класса CLineTop
{
protected:
    CPoint p;

```

```

public:
    /*определение конструктора класса CLineTop инициализацией координат
        одной точки с помощью базового конструктора CPoint(x1,y1)*/
    CLineTop(int x1, int y1, int x2, int y2) : CPoint(x1,
        y1)
    {
        p.setxy(x2, y2);
        //вызов унаследованной члена-функции для задания координат
                                                //второй точки
    }
    void prn() //определение функции-члена класса CLineTop
    {
        printf("Координаты 1-го угла: ");
        CPoint::prn(); /*вызов функции-члена класса CPoint
                        (CPoint::prn()) для вывода координат первой точки*/
        printf("Координаты 2-го угла: ");
        p.prn(); /*вызов функции-члена класса CLineTop
                  для вывода координат второй точки*/
    }
}; //конец определения класса CLineTop
class CLineBottom : public CPoint
    //начало определения класса CLineBottom
{
protected:
    CPoint p;
public:
    /*определение конструктора класса CLineBottom инициализацией
        координат одной точки с помощью базового
        конструктора CPoint(x1,y1)*/
    CLineBottom(int x1, int y1, int x2, int y2) :
    CPoint(x1, y1)
    {
        p.setxy(x2, y2); /*вызов унаследованной члена-функции
                            для задания координат второй точки*/
    }
    void prn() //определение функции-члена класса CLineBottom
    {
        printf("Координаты 3-го угла: ");
        CPoint::prn(); /*вызов функции-члена класса CPoint
                        (CPoint::prn()) для вывода координат первой (третьей) точки*/
        printf("Координаты 4-го угла: ");
        p.prn(); /*вызов функции-члена класса CLineBottom для
                  вывода координат второй (четвертой) точки*/
    }
}

```



```

}
};                                     //конец определения класса CLineBottom
class CPolygon4 : public CLineTop, public CLineBottom
                                     //начало определения класса CPolygon4
{
public:
    /*определение конструктора класса CPolygon4 инициализацией
    «родительских» конструкторов классов CLineTop и CLineBottom*/
    CPolygon4(int x1, int y1, int x2, int y2, int x3,
              int y3, int x4, int y4): CLineTop(x1, y1, x2, y2),
              CLineBottom(x3, y3, x4, y4)
    {
    }
    void prn()
    {
        CLineTop::prn();
        CLineBottom::prn();
    }
};                                     //конец определения класса CPolygon4

int main()
{
    CLineTop l1(1, 2, 5, 4);
    l1.prn();
    CLineBottom l2(2, 1, 3, 1);
    l2.prn();
    CPolygon4 p(1, 2, 5, 4, 2, 1, 3, 1);
    printf("Четырехугольник:\n");
    p.prn();
    printf("\n\nSizeof(CPoint) \t\t= %d\n",
          sizeof(CPoint));
    printf("Sizeof(CLineTop) \t= %d\n",
          sizeof(CLineTop));
    printf("Sizeof(CLineBottom) \t= %d\n",
          sizeof(CLineTop));
    printf("Sizeof(CPolygon4) \t= %d\n",
          sizeof(CPolygon4));
    return 0;
}

```

Результаты работы программы:

Координаты 1-го угла: (1, 2).
 Координаты 2-го угла: (0, 0).
 Координаты 3-го угла: (2, 1).
 Координаты 4-го угла: (0, 0).

```
Четырехугольник:  
Координаты 1-го угла: (1, 2).  
Координаты 2-го угла: (0, 0).  
Координаты 3-го угла: (2, 1).  
Координаты 4-го угла: (0, 0).  
  
Sizeof(CPoint)           = 8  
Sizeof(CLineTop)         = 16  
Sizeof(CLineBottom)     = 16  
Sizeof(CPolygon4)        = 32
```

Пояснение. В классе `CPoint` хранятся два целых числа, поэтому размер под объект этого класса выделяется восемь байт. Объект класса `CLineTop` (и класса `CLineBottom`) занимает 16 байт — четыре целых числа. Для хранения объекта класса «четырёхугольник» требуется 32 байта. То есть в случае обычного множественного объекта все члены базовых классов включаются в новый класс, и соответственно увеличивается объем требуемой памяти.

17.6. Виртуальные функции

Полиморфизм. Полиморфизмом называется возможность использовать в базовом и производном классах методы с одними и теми же именами и списком типов формальных параметров (одинаковой сигнатурой), причем для базового и производного классов эти методы определяются по-разному.

Имеется три вида методов: *статические*, *виртуальные* и *абстрактные*.

По умолчанию все методы *статические*. Если в классе-наследнике переопределен такой метод, то для объектов класса-наследника будет вызываться переопределенный метод, а для объектов базового класса будет вызываться метод класса-родителя. В приведенном выше примере переопределенный метод `prn` является статическим. Для статических методов определено *раннее связывание* объектов и методов. При раннем связывании связь объектов и методов устанавливается сразу же после компиляции и не изменяется во время выполнения программы.

Для *виртуальных* методов определено *позднее связывание* объектов и методов. В этом случае после компиляции только указывается, что объект имеет метод с таким именем и такими параметрами, а сам ме-

тод подключается к объекту во время выполнения программы при вызове данного метода. При вызове данного метода каким-то объектом сначала происходит определение типа объекта, а затем в соответствии с типом вызывается метод класса, к которому относится объект. При определении виртуального метода в родительском классе в начале объявления ставится ключевое слово **virtual**, а в классах-наследнике при объявлении виртуального метода ключевого слова **virtual** ставить не обязательно. Например, для базового класса **CRectangle** и производного класса **CBlock** определены виртуальные методы **Draw**, которые выполняются по-разному для разных классов. Тогда, например, допустимы следующие действия:

```
CRectangle *PRect;    //объявление указателя на класс CRectangle
```

В C++ разрешается присвоить этому указателю адрес как экземпляра класса **CRectangle**, так и экземпляра производного класса для **CRectangle**, без применения операции преобразования типов.

```
CRectangle *PRect1, *PRect2;
                                //объявление указателей на класс CRectangle
CRectangle Rect;                //объект класса CRectangle
CBlock Block;                   //объект производного класса CBlock
PRect1 = &Rect;
                                //PRect1 содержит адрес объекта класса CRectangle
PRect2 = &Block;
                                //PRect2 содержит адрес объекта класса CBlock
```

Теперь можно вызывать любой метод класса:

```
PRect1->Draw();    //будет выполняться метод класса CRectangle
PRect2->Draw();    //будет выполняться метод класса CBlock
```

Если эти методы были определены как в базовом, так и в производном классе как статические, то возникает неопределенность: компилятор не знает, метод какого класса ему вызывать. По умолчанию вызывается метод базового класса. Для устранения этой неопределенности следует применять виртуальные функции. То есть метод **Draw()** должен быть объявлен со спецификатором **virtual**. Отметим, что указание этого ключевого слова в производных классах необязательно, т. е. если метод объявлен виртуальным в базовом классе, то во всех производных он будет таким же. Приведем несколько примеров использования виртуальных и неvirtуальных функций.

Пример 1

```
# include <stdio.h>
class CBase                                     //начало определения базового класса CBase
{
public:
    virtual void fun(int i);
                                //прототип виртуальной функции fun базового класса CBase
};
                                //конец определения базового класса CBase
void CBase::fun(int i)
    //определение виртуальной функции fun базового класса CBase
{
    printf("Внутри CBase, i = %d.\n", i);
}
class CNas1 : public CBase
    //начало определения класса-наследника CNas1 базового класса CBase
{
public:
    virtual void fun(int i);
                                //прототип виртуальной функции fun класса CNas1
};
                                //конец определения класса CNas1
void CNas1::fun(int i)
    //определение виртуальной функции fun класса CNas1
{
    printf("Внутри CNas1, i = %d.\n", i);
}
class CNas2 : public CBase
    //начало определения класса-наследника CNas2 базового класса CBase
{
public:
    void fun(int i);
                                //прототип виртуальной функции fun класса CNas2
};
                                //конец определения класса CNas2
void CNas2::fun(int i)
    //определение виртуальной функции fun класса CNas2
{
    printf("Внутри CNas2, i = %d.\n", i);
}

int main()
{
    CBase b, *pb = &b;
    CNas1 n1, *pn1 = &n1;
    CNas2 n2, *pn2 = &n2;
    CBase *p1 = &n1, *p2 = &n2;
```

```

pb->fun(7);
pb->info();           //будет выведено «Внутри CBase: i=7»
pn1->fun(6);
pn1->info();          //будет выведено «Внутри CNas1: i=6»
pn2->fun(5);
pn2->info();          //будет выведено «Внутри CNas2: i=5»

p1->info();           //будет выведено «Внутри CNas1: i=6»
p2->info();           //будет выведено «Внутри CNas2: i=5»
p1->fun(4);
p1->info();           //будет выведено «Внутри CNas1: i=4»
pn1->info();          //будет выведено «Внутри CNas1: i=4»
p2->fun(3);
p2->info();           //будет выведено «Внутри CNas2: i=3»
pn2->info();          //будет выведено «Внутри CNas2: i=3»
return 0;
}

```

В этом примере для указателей `p1` и `p2` компилятор (система программирования) определит, что они указывают на объекты классов `CNas1`, `CNas2`, и вызовет соответствующие методы этих классов.

Пример 2. Пусть объявлен класс `CPoint`, у которого есть не виртуальный метод `print()` — выводит координаты точки на экран. На основе этого класса создается новый класс `CCircle`. В этом классе также есть не виртуальный метод `print()`, только теперь он должен выводить на экран координаты центра и радиус окружности. В результате выполнения следующего кода:

```

CPoint p1(2, 8);
CCircle c1(1, 5, 9);
CPoint *P;
P = &c1;
P->print();

```

на экран будут выведены координаты точки, а не центр окружности и ее радиус, даже несмотря на то, что указатель `P` содержит адрес объекта класса `CCircle`. Если же метод `print()` в базовом классе объявить с ключевым словом `virtual`, то все будет работать так, как хотелось. При этом в производных классах слово `virtual` можно не писать, достаточно указать его только в базовом классе.

Выводы. Виртуальные функции реализуют одно из основных понятий ООП — полиморфизм. Ранее мы уже сталкивались с примерами полиморфизма — перегрузкой функций.

17.7. Виртуальные базовые классы

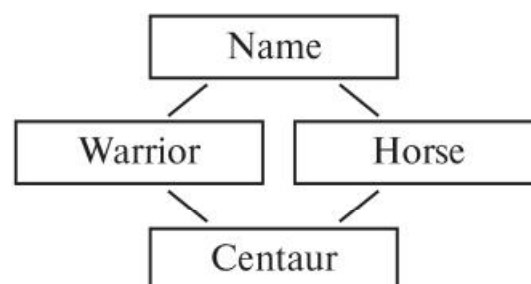
Иногда при множественном наследовании приходится строить производные классы, которые включают в себя несколько экземпляров одного и того же непрямого базового класса. Например, класс `CPolygon4` (см. параграф 15.2) включал в себя два экземпляра непрямого базового класса `CPoint`. В другом случае при более сложном производном классе число экземпляров непрямого базового класса может возрасти в несколько раз, что может внести путаницу в создание иерархии производных классов и написания программы. Для устранения дублирования объектов непрямого базового класса при множественном наследовании и служит виртуальный базовый класс, устраняющий возможную путаницу.

При наследовании виртуальный базовый класс объявляется с ключевым словом `virtual`.

```
class C1 : public virtual CBase
```

Пример. Написать класс «имя», служащий для хранения имени игрока компьютерной игры. На основе класса «имя» породить класс «воин», содержащий информацию и силу воина, и класс «конь», содержащий информацию и скорость коня. На базе этих классов породить новый класс «кентавр». Использовать виртуальные базовые классы, чтобы исключить двойное включение объекта класса «имя» в новый класс «кентавр».

Иерархия классов:



```
#include <stdio.h>
#include <string.h>
class Name      //начало определения виртуального базового класса Name
{
protected:
    char name[32];      //определение защищенного члена-переменной
public:
    Name()              //определение конструктора без параметров
    { strcpy(name, "Noname"); }
```



```

    Name(char *n)
        //определение конструктора с одним параметром-строкой
    { strcpy(name, n); }
    virtual void prn()
        //определение виртуальной функции базового класса
    { printf("%s\n", name); }
    friend class Centaur;
        //объявление класса Centaur дружественного классу Name
};
        //конец определения виртуального базового класса Name
class Warrior : public virtual Name /*начало определения
        класса Warrior наследника виртуального непрямого
        базового класса Name*/
{
protected:
    int sila;
        //определение защищенного члена-переменной класса Warrior
public:
        /*определение конструктора класса Warrior с инициализирующим
        вызовом конструктора класса Name*/
    Warrior(char *n, int s) : Name(n)
    {
        sila = s;
    }
    void prn()
        //определение виртуальной функции производного класса Warrior
    {
        Name::prn(); //вызов виртуальной функции базового класса
        printf("Sila: %d\n", sila);
    }
    friend class Centaur;
        //объявление класса Centaur дружественного классу Warrior
};
        //конец определения производного класса Warrior
class Horse : public virtual Name /*начало определения класса
        Horse наследника виртуального непрямого базового класса Name*/
{protected:
    int skorost;
        //объявление защищенной члена-переменной класса Horse
public:
        /*определение конструктора класса Horse с инициализирующим
        вызовом конструктора класса Name*/
    Horse(char *n, int s) : Name(n)
    {
        skorost = s;
    }
}

```

```

void prn()
    //определение виртуальной функции производного класса Horse
{
    Name::prn();          //вызов виртуальной функции базового класса
    printf("Skorost: %d\n", skorost);
}
friend class Centaur;
    //объявление класса Centaur дружественного классу Horse

};          //конец определения класса Horse
class Centaur : public Warrior, public Horse
    //начало определения класса Centaur наследника классов Warrior
    //и Horse
{
protected:
public:
    /*определение конструктора класса Centaur с инициализирующим
    вызовом конструкторов классов Warrior, Horse и Name*/
    Centaur(char *na, int si, int sk) : Warrior(na, si),
        Horse(na, sk), Name(na)
    {
    }
void prn()
    //определение виртуальной функции производного класса Centaur
{
    Warrior::prn();
    //вызов виртуальной функции класса Warrior
    Horse::prn();      //вызов виртуальной функции класса Horse
}
};
int main()
{
    printf("Воин:\n");
    Warrior w("Ivan", 39);    w.prn();
    Horse h("Sivka", 20);    h.prn();
    printf("\n\nКентавр:\n");
    Centaur c("Terminator", 40, 50);    c.prn();
    printf("\n\nSizeof(Name): \t\t%d\n", sizeof(Name));
    printf("Sizeof(Warrior): \t%d\n", sizeof(Warrior));
    printf("Sizeof(Horse): \t\t%d\n", sizeof(Horse));
    printf("Sizeof(Centaur): \t%d\n", sizeof(Centaur));
    return 0;
}

```

Результат работы программы:

```
Воин:  
Ivan  
Sila: 39  
Sivka  
Skorost: 20
```

```
Кентавр:  
Terminator  
Sila: 40  
Terminator  
Skorost: 50
```

```
Sizeof(Name) :           36  
Sizeof(Warrior) :        48  
Sizeof(Horse) :          48  
Sizeof(Centaur) :        56
```

Пояснение. В данном случае объект класса `Name` занимает 36 байт, хотя в нем хранится массив из 32 символов. Дополнительные 4 байта требуются для хранения информации о виртуальной функции `print()`. Для хранения объекта класса `Warrior` требуется 48 байт, хотя в нем к существующему массиву из 32 символов добавлено одно целое число. Дополнительные 4 байта аналогично нужны для хранения информации о виртуальной функции и еще 8 байт для хранения информации о том, что класс `Warrior` является виртуальным. Аналогично объясняются объемы памяти, необходимые для других классов.

Таким образом, при использовании виртуальных классов требуется больше памяти, однако они позволяют создать логически оправданную иерархию классов.

17.8. Абстрактные классы

Чисто виртуальный метод — это виртуальный метод, реализация которого не определена в том классе, в котором он объявлен. Предполагается, что такой метод будет перегружен в классах-наследниках. Чисто виртуальный метод можно вызывать только в тех классах-наследниках, где этот метод перегружен.

Чистая виртуальная функция (метод) — это функция, объявленная со спецификатором `virtual` и не выполняющая никаких действий. Чисто виртуальная функция определяется следующим образом:

```
virtual тип имя_функции(список_параметров) = 0;
```

Например, в объявлении

```
virtual void prn()=0;
```

объявляется чисто виртуальная функция `prn`, не имеющая параметров и не возвращающая никакого значения вызываемой функции.

Если в классе есть хотя бы один чисто виртуальный метод, данный класс считается абстрактным классом. Объект такого класса создать невозможно. Создать можно только объекты наследников этого класса, в которых реализованы все чисто виртуальные методы данного класса. Абстрактный класс обычно служит началом иерархии классов программы.

Абстрактный класс может иметь явно определенный конструктор, из которого могут вызываться методы класса. Однако любые прямые обращения к конструктору приведут к ошибкам.

Рассмотрим пример использования абстрактного класса.

```
#include <stdio.h>
class CBase
    //начало определения абстрактного базового класса CBase
{
protected:
    virtual void prn() = 0;
    //определение чисто виртуальной функции
};
    //конец определения абстрактного класса CBase
class CPoint : public CBase
    //начало определения производного класса CPoint наследника класса
    CBase
{
protected:
    int x, y;
    //определение защищенных членов-переменных класса CPoint
public:
    CPoint(int x = 0, int y= 0)
        //определение конструктора класса CPoint
    {
        this->x = x;
        this->y = y;
    }
}
```

```
void prn()    //определение виртуальной функции-члена класса CPoint
{
    printf("Tochka (%d, %d).\n", x, y);
}
};           //конец определения класса CPoint
int main()
{
    CPoint p;
    p.prn();
    printf("\n\nSizeof(CBase) = %d\n", sizeof(CBase));
    printf("Sizeof(CPoint) = %d\n", sizeof(CPoint));
    return 0;
}
```

Обращения к чистым виртуальным функциям не допускаются, следовательно, в производном классе такие функции должны быть обязательно перегружены. Отсутствие описания в производных классах приводит к ошибке компилятора.

17.9. Классы и шаблоны

Шаблоны, которые иногда называют родовыми или параметризованными типами, позволяют создавать (конструировать) семейство родственных функций и классов.

Шаблоны семейства функций и классов определяют потенциально неограниченное множество родственных функций и классов.

Шаблон семейства классов определяет способ построения классов, подобно тому как класс определяет правила построения, поведения и формат отдельных объектов. В определении класса, входящего в шаблон, особую роль играет имя класса. Оно является не именем отдельного класса, а параметризованным именем семейства классов. Определение шаблона может быть только глобальным.

Рассмотрим, например, класс `CPoint_int` с членами-переменными типа `int`, который мы рассматривали ранее, и аналогичные ему классы `CPoint_float`, с членами-переменными типа `float` и `CPoint_double` с членами-переменными типа `double`:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
```

```
class CPoint_int          //начало определения класса CPoint_int
{
private:
    int x, y;             //определение закрытых целых членов-переменных
public:
    CPoint_int(int x = 0, int y = 0)
        //определение конструктора класса CPoint_int по умолчанию
    {
        this->x = x;
        this->y = y;
    }
    void set_X(int x)      //определение члена-функции класса CPoint_int
    {
        this->x = x;
    }
    void set_Y(int y)      //определение члена-функции класса CPoint_int
    {
        this->y = y;
    }
    int get_X()            //определение члена-функции класса CPoint_int
    {
        return this->x;
    }
    int get_Y()            //определение члена-функции класса CPoint_int
    {
        return this->y;
    }
};                          //конец определения класса CPoint_int

class CPoint_float        //начало определения класса CPoint_float
{
private:
    float x, y;           //определение закрытых вещественных членов-переменных
public:
    CPoint_float(float x = 0, float y = 0)
        //определение конструктора класса CPoint_float по умолчанию
    {
        this->x = x;
        this->y = y;
    }
    void set_X(float x)
        //определение члена-функции класса CPoint_float
    {
        this->x = x;
    }
}
```



```
void set_Y(float y)
    //определение члена-функции класса CPoint_float
{
    this->y = y;
}
float get_X()    //определение члена-функции класса CPoint_float
{
    return this->x;
}
float get_Y()    //определение члена-функции класса CPoint_float
{
    return this->y;
}
};                //конец определения класса CPoint_float

class CPoint_double //начало определения класса CPoint_double
{
private:
    double x, y;
    //определение закрытых вещественных членов-переменных
public:
    CPoint_double(double x = 0, double y = 0)
        //определение конструктора класса CPoint_double по умолчанию
    {
        this->x = x;
        this->y = y;
    }
void set_X(double x)
    //определение члена-функции класса CPoint_double
{
    this->x = x;
}
void set_Y(double y)
    //определение члена-функции класса CPoint_double
{
    this->y = y;
}
double get_X() //определение члена-функции класса CPoint_double
{
    return this->x;
}
double get_Y() //определение члена-функции класса CPoint_double
{
    return this->y;
}
};                //конец определения класса CPoint_double
```

```

int main()
{
    CPoint_int p1(5, 8);
    CPoint_float p2(1.8, 5.3);
    CPoint_double p3(3.1, 56.63);
    print("p1.x=%d p1.y=%d\n", p1.getX(), p1.getY());
    print("p2.x=%f p2.y=%f\n", p2.getX(), p2.getY());
    print("p3.x=%lf p3.y=%lf\n", p3.getX(), p3.getY());
    return 0;
}

```

Как видно из примера, все рассмотренные классы являются родственными — у них одинаковые члены-переменные, члены-функции, поведение и они одинаковым образом могут использоваться в программе. Разные в этих классах только типы членов-переменных. Без использования шаблона класса пришлось давать полные определения практически одинаковых классов, что сильно увеличивает объем программы и вероятность появления ошибок. Использование шаблонов снимает эти проблемы.

Пример 1. Написать шаблон для класса CPoint.

```

#include <iostream>
#include <stdlib.h>
#include <time.h>
template <class T> class CPoint    /*начало определения шаблона
                                   класса CPoint с параметром — классом (типом) T */
{
private:
    T x, y;                      //определение закрытых членов-переменных типа T
public:
    CPoint(T x = 0, T y = 0)      //определение конструктора класса CPoint по умолчанию
    {
        this->x = x;
        this->y = y;
    }
    void set_X(T x)               //определение члена-функции класса CPoint
    {
        this->x = x;
    }
    void set_Y(T y)               //определение члена-функции класса CPoint
    {
        this->y = y;
    }
}

```

```

T get_X()                //определение члена-функции класса CPoint
{
    return this->x;
}
T get_Y()                //определение члена-функции класса CPoint
{
    return this->y;
}
};                        //конец определения шаблона класса CPoint
int main()
{
    CPoint<int>  p1(5, 8);
    CPoint<float> p2(1.8, 5.3);
    CPoint<double> p3(3.1, 56.63);
    print("p1.x=%d p1.y=%d\n", p1.getX(), p1.getY());
    print("p2.x=%f p2.y=%f\n", p2.getX(), p2.getY());
    print("p3.x=%lf p3.y=%lf\n", p3.getX(), p3.getY());
    return 0;
}

```

Пример 2. Написать шаблон класса CVector.

Описание шаблона вынесем в *.h файл.

Содержимое файла vector.h:

```

template <class Type> class CVector    /*начало определения
    шаблона класса CVector с параметром — классом (типом) Type*/

{
private:
    int N;                /*объявление закрытой члена-переменной — числа
                           элементов динамического массива*/
    Type *p;              //объявление указателя на динамический массив типа Type
public:
    CVector(int n)         //определение конструктора шаблона
                           //класса CVector
    {
        N = n;            //присвоить члену-переменной значение параметра
        if (N > 0)
            p = new Type [N];    /*выделение памяти под N элементов
                                   массива типа Type — программист должен быть убежден,
                                   что система программирования может это сделать*/
    }
}

```

```

~CVector()           //определение деструктора шаблона класса CVector
{
    if (N > 0) delete [] p;           /*освобождение памяти под
                                     одномерный массив типа Type — программист должен быть
                                     убежден, что система программирования сможет это сделать*/
}
void gen()
    //определение члена-функции генерации одномерного массива
{
    for (int i = 0; i<N; i++)
        p[i] = Type(rand())/RAND_MAX * 256 + 20;
}
int GetN()
{return N;
}
Type operator[](int k)
    //определение перегруженного оператора извлечения элемента из массива
{if ((k>=0)&&(k<N)
return p[k];
else return Type (RAND_MAX);
}
};                               //конец определения шаблона класса CVector

int main()
{
    srand(time(NULL));
    CVector <int> v1(5);
    v1.gen();
    for (int i=0; i<v1.GetN(); i++)
        printf("%d ", v1[i]);
    printf("\n");
    CVector <float> v2(3);
    v2.gen();
    for (int i=0; i<v2.GetN(); i++)
        printf("%f ", v2[i]);
    printf("\n");
    CVector <char> v3(10);
    v3.gen();
    for (int i=0; i<v3.GetN(); i++)
        printf("%c ", v[i]);
    printf("\n");
    return 0;
}

```

В списке параметров шаблона могут присутствовать формальные параметры, не определяющие тип, — это параметры, для которых тип фиксирован.

Пример 3. Альтернативный вариант шаблона класса CVector.

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

template <class Type, int N> class CVector /*начало
    определения шаблона класса CVector с параметрами — классом (типом)
    Type и целым параметром N — числом элементов одномерного массива*/
{
private:
    Type mass [N];
    //объявление закрытого члена-переменной — статического массива
public:
    CVector(); //прототип конструктора без параметров
    CVector(Type initValue); //прототип конструктора с одним параметром
    void GenRandom();
    int GetN()
    {return N;
    }
    Type operator[] (int k)
    //прототип перегруженного оператора извлечения элемента из массива

    Type max();
};

template <class Type, int N> CVector <Type,
    N>::CVector() //шаблон определения конструктора класса CVektor
    //без параметров
{
    for (int i = 0; i<N; i++)
        mass[i] = 0;
}

template <class Type, int N> CVector <Type,
    N>::CVector(Type initValue)
{
    //шаблон определения конструктора класса CVektor с одним
    for (int i = 0; i<N; i++) //параметром
        mass[i] = initValue;
}
```

```

template <class Type, int N> void CVector <Type,
    N>::GenRandom()
    /*шаблон определения члена-функции генерации одномерного
      массива с помощью датчика случайных чисел*/
{
    for (int i = 0; i<N; i++)
        mass[i] = (Type) (rand() * 10) / RAND_MAX;
}

template <class Type, int N> Type CVector <Type,
    N>::operator(int k) /*шаблон определения перегруженного
      оператора извлечения элемента из массива*/
{if ((k>0)&&(k<N))
    return p[k];
else return Type (RAND_MAX);
}

template <class Type, int N> Type CVector <Type,
N>::max() /*шаблон определения функции-члена класса — нахождения
      максимального элемента массива*/
{
    Type m;
    m = mass[0];
    for (int i = 1; i<N; i++)
        if (mass[i] > m)
            m = mass[i];
    return m;
} /*конец определения шаблона класса CVector
      с параметрами — классом (типом) Type и целым
      параметром N — числом элементов одномерного массива*/

int main()
{
    srand(time(NULL));
    CVector <float, 5> v;
    for (int i=0; i<v.GetN(); i++)
        printf("%f ", v[i]);
    printf("\n");
    v.GenRandom();
    for (int i=0; i<v.GetN(); i++)
        printf("%f ", v[i]);
}

```



```
printf("\n");
cout << "Max = " << v.max() << endl;
CVector <int, 5> v1(8.5);
for (int i=0; i<v1.GetN(); i++)
printf("%d ", v1[i]);
printf("\n");
v1.GenRandom();
for (int i=0; i<v1.GetN(); i++)
printf("%d ", v1[i]);
printf("\n");
cout << "Max = " << v1.max() << endl;
return 0;
}
```

В этом примере память под элементы массива выделяется не динамически, так как количество элементов задано жестко параметром N.

В заключение можно сказать, что шаблоны классов позволяют избавиться от дублирования кода для таких классов, чьи объекты отличаются только типом их элементов. Это позволяет повысить эффективность программирования, избежать ненужных ошибок и уменьшить сроки разработки ПО.

Тесты

1. Наследование — это:

- а) возможность использования базовых библиотек языка C++ в своих программах;
- б) условия, описывающие последовательность вызова конструкторов для объектов классов, используемых в программе;
- в) механизм создания производных классов, на базе уже имеющихся.

2. Укажите правильный вариант определения порожденного класса на базе уже существующего:

- а) `class new_class : base {};`
- б) `class new_class : public base {};`
- в) оба варианта правильные.

3. Сколько ошибок допущено в следующем коде?

```
class base
{
    int x, y;
};
```

```
class child : public base
{
void info()
{
    printf("x = %d, y = %d\n", x, y);
}
};
```

а) 0;

б) 1;

в) 2.

4. Укажите верную последовательность выполнения конструкторов:

а) сначала выполняются операторы конструктора базового класса, затем операторы конструктора порожденного класса;

б) сначала выполняются операторы конструктора порожденного класса, затем операторы конструктора базового класса;

в) операторы конструктора порожденного класса выполняются одновременно с операторами конструктора базового класса.

5. Укажите верную последовательность выполнения деструкторов:

а) сначала выполняются операторы деструктора базового класса, затем операторы деструктора порожденного класса;

б) сначала выполняются операторы деструктора порожденного класса, затем операторы деструктора базового класса;

в) операторы деструктора порожденного класса выполняются одновременно с операторами деструктора базового класса.

6. Выберите правильный вариант вызова конструктора базового класса Base, в конструкторе порожденного класса Child:

а) Child(Base()) {}

б) Child() : Base() {}

в) Child() : public Base() {}

7. Можно ли в языке C++ породить новый класс на основе нескольких базовых классов?

а) да;

б) нет;

8. Полиморфизм — это:

а) возможность программного кода работать с разными объектами одинаковым образом;

б) возможность изменения программного кода в зависимости от решаемых задач;

в) возможность доработки программного кода в случае необходимости.

9. С помощью какого ключевого слова задается виртуальная функция?

а) virt;

б) virtual function;

в) virtual.

10. Что такое чистый виртуальный метод?

а) это функция, объявленная со спецификатором virtual и не выполняющая никаких действий;

б) это функция, объявленная со спецификатором virtual и не возвращающая никакого значения (даже типа void);

в) это функция, объявленная со спецификатором virtual и предназначенная для очистки памяти.

11. Укажите верное утверждение:

а) конструкторы и деструкторы не наследуются;

б) абстрактный базовый класс имеет хотя бы один чисто виртуальный метод;

в) оба утверждения верны.

12. Что такое родовые классы?

а) это классы, полученные в результате наследования;

б) это классы, имеющие одинаковые члены-переменные и члены-функции, отличающиеся только типами параметров;

в) это набор базовых классов, используемых при множественном наследовании.

13. Сколько параметров типа может быть в шаблоне?

а) один;

б) два — один стандартный тип и один пользовательский;

в) произвольное количество.

Задания

1. Разработайте базовый класс «геометрическая точка» в трехмерном пространстве (члены класса — координаты x , y , z). На базе этого класса создайте порожденный класс — «сфера», с методами вычисления площади поверхности и объема сферы.

2. Разработать базовый класс «животное». Члены класса: вид животного. В классе предусмотреть конструктор, деструктор и набор методов для работы с данными. На основе этого класса породить классы: травоядные и хищники. Создайте несколько объектов производных классов.

3. Разработать базовый класс «заготовка», содержащий сведения о материале заготовки для последующего изготовления детали. На основе этого класса породить класс «деталь», добавив в нее структуру с техническими характеристиками детали (масса, габаритные размеры и т. д.). Написать класс «изделие», содержащий в себе пять деталей. Создать объект «велосипед» класса «изделие» (детали: переднее колесо, заднее колесо, рама, руль, сиденье). Вывести информацию о велосипеде на экран.

4. Написать базовый класс Name, в котором хранится имя героя игры. На основе этого класса создать классы игроков: Warrior — воин, характеризуется именем и силой. Horse — лошадь, характеризуется именем и скоростью. На базе классов Warrior и Horse создать новый класс игрока: Centaur (Кентавр), который характеризуется именем, силой и скоростью. Вывести на экран размеры созданных классов. Продемонстрировать разницу в классе Centaur при использовании абстрактных базовых классов.

5. Создать иерархию классов: абстрактный класс «фигура», от него порождается абстрактный класс «замкнутая фигура». От класса «фигура» порождается классы: «точка» и «линия», от класса «замкнутая фигура» — «прямоугольник», «окружность». В каждом классе должны быть методы вывода на экран информации о фигуре, сдвига фигуры на указанные величины, вычисления площади (для замкнутых фигур).

Контрольные вопросы

1. На каких основных принципах базируется ООП? Кратко поясните суть каждого из них.
2. Что такое наследование? Какие преимущества дает использование наследования?
3. Приведите синтаксис создания производного класса. На что влияют спецификаторы доступа, используемые при наследовании?
4. Какими особенностями обладают конструкторы при наследовании? В какой последовательности вызываются конструкторы при создании объектов производного класса?
5. Какими особенностями обладают деструкторы при наследовании? В какой последовательности вызываются деструкторы при удалении объектов производного класса?

-
6. Что такое иерархия классов? Как построить иерархию классов в языке C++? Приведите примеры.
 7. Приведите синтаксис создания порожденного класса на основе нескольких базовых классов.
 8. Что такое полиморфизм? Какие виды методов бывают у классов?
 9. Что такое виртуальный метод класса? Как сделать метод класса виртуальным?
 10. Какие преимущества дает использование виртуальных базовых классов?
 11. Что такое чистый виртуальный метод? Для чего он нужен?
 12. Что такое абстрактный базовый класс?
 13. Какие классы можно считать родственными? Как определить шаблон для родственных функций или классов? Приведите примеры.
 14. Какие ограничения накладываются на список параметров шаблона? Приведите примеры.

Глава 18

ПОТОКОВЫЙ ВВОД/ВЫВОД

18.1. Пространство имен

Часто программа использует одновременно несколько библиотек. Каждая из библиотек вводит свою систему понятий и основанную на ней иерархию классов, которую можно назвать системой сущностей данной библиотеки. Например, прототипы функций `rand()`, `srand()` и некоторых других, константа `RAND_MAX` являются сущностями библиотеки `stdlib`. Названия сущностей библиотеки выбираются таким образом, чтобы соответствовать назначению этих сущностей, например математические функции библиотеки `math` или функции и константы библиотеки `stdlib`.

Поскольку библиотеки в общем случае разрабатываются независимо друг от друга, нет никаких гарантий того, что названия сущностей разных библиотек не вступят в конфликт друг с другом. Например, библиотека классов трехмерного геометрического моделирования для `Windows Application` может предоставлять класс `Point` для представления точки в трехмерном пространстве. В то же самое время графическая библиотека интерфейса пользователя для `Windows Application` может предоставлять класс `Point` для отображения точки в двухмерном окне на экране. В программе, которая одновременно использует эти две библиотеки, почти наверняка возникнут трудности с использованием класса `Point`. Для решения этой проблемы необходимо вводить пространства имен.

Пространство имен задается таким способом:

```
namespace имя_пространства
{
    определения классов, функции и т. д.
}
```


После того как разработчик ввел пространство имен, программист может использовать методы, классы и т. д., описанные в нем. Для этого необходимо указать имя используемого пространства:

```
using namespace имя;
```

Пример.

```
#include <stdio.h>
namespace myPoint //начало определения пространства имен myPoint
{
    class CPoint //начало определения класса CPoint
    {
        int x, y; //закрытые целые переменные класса CPoint
    public:
        CPoint(int _x = 0, int _y = 0)
            //определение конструктора класса CPoint с двумя параметрами
        { x = _x; y = _y; }
        void print() //определение члена-функции класса CPoint
        { printf("Tochka:)(%d. %d) \n".x,y); }
    }; //конец определения класса CPoint
} //конец определения пространства имен myPoint
using namespace myPoint;
//указание имени используемого пространства имен myPoint
int main()
{
    CPoint p1, p2(3, 7);
    p1.print();
    p2.print();
    return 0;
}
```

Пояснение. Описание класса `CPoint` включено в пространство имен `myPoint`. В дальнейшем для работы с объектами класса `CPoint` используется строка:

```
using namespace myPoint;
```

В пространстве имен могут быть расположены определения произвольных пользовательских типов данных. Если описание типов данных расположено в заголовочном файле, то и границы определения пространства имен должны находиться в нем же.

Пример. Разработать класс, предназначенный для работы с рейсами авиакомпании. Описание класса вынести в заголовочный файл, а реализацию методов класса — в отдельный *.cpp файл.

Файл *reis.h*

```
#ifndef _reis_                                //заголовочный файл, только один раз
#define _reis_    //будет включен в компилируемый текст программы
#include <string.h>
#include <stdio.h>
namespace air_space                          //объявляем пространство имен
{
    class Reis
    {
    protected:
        char country[20];
        int cost;
    public:
        Reis(char* str, int);
        void info();
    };
}
#endif
```

Файл *reis.cpp*

```
#include "reis.h"
using namespace air_space;    /*перед реализацией методов класса
                               указываем, что будем работать с классами из пространства имен
                               air_spase*/
Reis::Reis(char *str = "Россия", int cost = 100)
{
    strcpy(country, str);
    this->cost = cost;
}

void Reis::info()
{
    printf("Country = %s, Cost = %d\n", country, cost);
}
```

Файл с основной программой:

```
#include <stdio.h>
#include "reis.h"
```

```
int main()
{
    using namespace air_space;
    //указываем, что будем работать с типами данных,
    Reis r("Турция", 1200);
    //определенными в пространстве имен air_space
    r.info();
    return 0;
}
```

Пояснение. Определение класса помещено в пространство имен с именем `air_space`. В файле, содержащем реализацию методов класса, необходимо указать, что будем работать с типами данных, введенными в пространство имен `air_space`. Аналогично указывается пространство используемых имен перед созданием экземпляра класса `Reis`.

18.2. Понятие потока

При работе программы практически всегда возникает обмен информацией между программой и различными внешними устройствами — экраном, клавиатурой, файлами на дисках и т. п. Причем почти всегда внешние устройства (экран, клавиатура и т. д.) принимают или выдают информацию посимвольно, а в программе нужен обмен числами из нескольких символов, массивами чисел, строками и т. п. Если бы обмен данными между программой и внешними устройствами происходил посимвольно, как того требуют внешние устройства, то все время работы программы и операционной системы уходило бы на обслуживание этого обмена и выполнение программы занимало бы в десятки раз больше времени, чем это необходимо.

Для устранения этого недостатка в компьютере используются буферы — специальные области операционной памяти:

- буфер клавиатуры — обслуживает обмен между клавиатурой и программой;
- буфер экрана — обслуживает обмен между экраном и программой;
- файловый буфер — обслуживает обмен между файлом на диске и программой.

Перечисленные буферы используются во всех языках программирования высокого уровня (Object Pascal, C++ и т. д.), но в каждом из языков имеют свои особенности организации и использования.

В языке C++ в связи с обменом данными существует понятие **потока** (*stream*), в которое входит:

- наличие специального буфера для приема, хранения и выдачи информации;
- наличие указателей считывания из потока и записи в поток, или, другими словами, номеров байтов (адресов), откуда происходит считывание из буфера или запись в буфер;
- сведения о состоянии буфера (пуст, полон или еще есть свободное место);
- сведения о способе использования буфера — чтение, запись;
- возможность проверки, не полон ли буфер;
- возможность проверки, можно ли записать на это место символ;
- возможность проверки, можно ли считать с этого места символ;
- возможность передвинуть на один байт указатель считывания или записи;
- возможность дозаписи в буфер при его частичном или полном освобождении и т. д.

Более точное определение: **поток** — это специальная область памяти со средствами буферизации, записи, хранения и выдачи информации в необходимом виде.

Некоторые модели потоков представлены на рис. 18.1, 18.2.



Рис. 18.1. Модель выходного потока

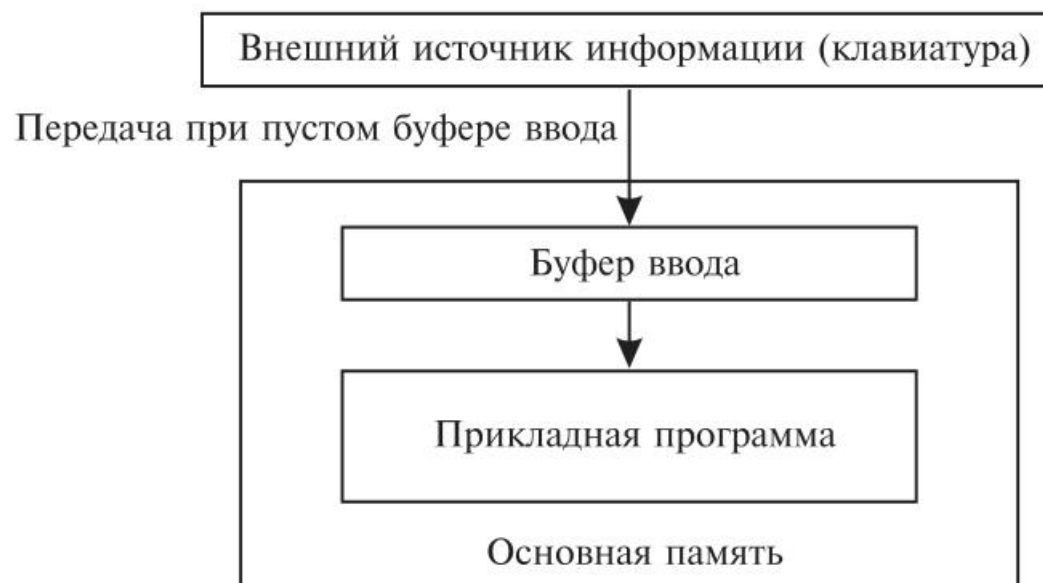


Рис. 18.2. Модель входного потока

Аналогичным образом можно представить модели потоков при работе с файлами, расположенными на диске компьютера. Информация из файла поступает в буфер ввода, после чего становится доступна в основной программе. Соответственно в модели выходного потока информация из программы будет записываться в буфер, из которого попадет непосредственно в файл.

18.3. Классы потоков. Иерархия классов потоков

`ios` — базовый потоковый класс;
`istream` — класс входных потоков;
`ostream` — класс выходных потоков;
`iostream` — класс двунаправленных потоков ввода-вывода;
`istream` — класс входных строковых потоков;
`ostream` — класс выходных строковых потоков;
`stringstream` — класс двунаправленных строковых потоков ввода-вывода;
`ifstream` — класс входных файловых потоков;
`ofstream` — класс выходных файловых потоков;
`fstream` — класс двунаправленных файловых потоков ввода-вывода;
`constream` — класс консольных потоков.

Класс `ios` является основой иерархии и содержит все основные члены-переменные и методы, принадлежащие любому потоку.

Классы `istream` (класс входных потоков), `ostream` (класс выходных потоков), `iostream` (класс двунаправленных потоков ввода-вывода) не

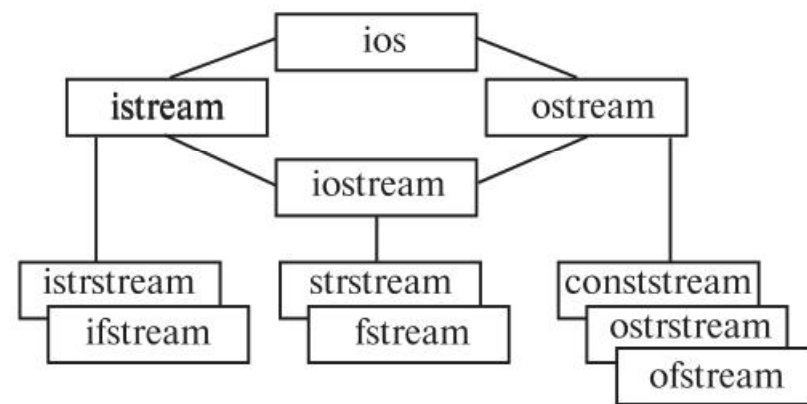


Рис. 18.3. Упрощенная схема иерархии потоковых классов

связаны с источниками (клавиатура, файл, строка) или приемниками (экран, файл, строка) информации. Эти классы содержат члены-переменные и методы, необходимые для организации соответствующего обмена вне зависимости от источника или приемника данных.

Классы `istrstream` (класс входных строковых потоков), `ostrstream` (класс выходных строковых потоков), `strstream` (класс двунаправленных строковых потоков ввода-вывода) содержат члены-переменные и методы, необходимые для обмена между программой и строкой как областью оперативной памяти.

Классы `ifstream` (класс входных файловых потоков), `ofstream` (класс выходных файловых потоков), `fstream` (класс двунаправленных файловых потоков ввода-вывода) содержат члены-переменные и методы, необходимые для обмена между программой и файлом на диске.

18.4. Класс консольных потоков. Объекты `cin` и `cout`

Консоль (Console) — физически представляет собой совокупность клавиатуры и экрана. И так как физически эти устройства очень сложны и важны для компьютера, был выделен специальный класс `constream` — класс консольных потоков.

Также были выделены специальные стандартные объекты для работы с клавиатурой и экраном:

`cout` — объект класса `ostream`, связанный со стандартным буферизированным выходным потоком (как правило — дисплей);

`cin` — объект класса `istream`, связанный со стандартным буферизированным входным потоком (как правило — клавиатура);

`cerr` — объект класса `ostream`, связанный со стандартным небуферизированным выходным потоком (как правило — дисплей), в который направляются сообщения об ошибках;

`clog` — объект класса `ostream`, связанный со стандартным буферизированным выходным потоком (как правило — дисплей), в который с буферизацией направляются сообщения об ошибках.

18.5. Ввод/вывод стандартных типов данных

Для того чтобы для ввода-вывода данных можно было воспользоваться стандартными объектами потоков, необходимо подключить специальную библиотеку ввода-вывода — `iostream`, название которой расшифровывается следующим образом — `I(nput)O(utput)STREAM` — потоковая библиотека ввода-вывода.

Потоковая библиотека ввода-вывода `iostream` является библиотекой классов, и поэтому в директиве `include` расширение файла писать не надо:

```
#include <iostream>
```

Библиотека `iostream` содержит классы и функции, которые могут работать в консольном режиме (Console Application) и в Windows режиме (Windows Application). Причем некоторые функции могут выглядеть одинаково, но выполняться по-разному, а некоторые объекты, например `cin` и `cout`, неизвестны в Windows режиме. Поэтому, чтобы пользоваться библиотекой `iostream` в консольных приложениях, необходимо указывать, что используется пространство имен с названием `std`:

```
#include <iostream>
using namespace std;
```

Ввод данных любых стандартных типов осуществляется следующим оператором:

```
cin>>a;;
```

где `cin` — объект класса `istream`, связанный со стандартным буферизированным входным потоком (как правило — клавиатура);

`>>` — функция ввода;

`a` — имя переменной, значение которой нужно ввести с клавиатуры.

Если требуется с помощью одного оператора ввести значения нескольких переменных любых стандартных типов языка C++, то оператор ввода записывается следующим образом:

```
cin>>a>>b>>c;
```

При вводе первым записывается значение переменной *a*, затем — значение переменной *b* и самым последним — значение переменной *c*.

Вывод данных любых стандартных типов осуществляется оператором

```
cout<<a;
```

где *cout* — объект класса *ostream*, связанный со стандартным буферизированным выходным потоком (как правило — дисплей);

<< — функция ввода;

a — имя переменной, значение которой нужно вывести на экран.

Если требуется с помощью одного оператора вывести значения нескольких переменных любых стандартных типов языка C++, то оператор вывода записывается следующим образом:

```
cout<<"a="<<a<<" b="<<b<<" c="<<c;
```

Последовательности символов, заключенные в кавычки, являются строковыми константами и выводятся без изменения. Имена переменных не заключены в кавычки, поэтому будут выводиться их значения, например:

```
a=3.1 b=6 c=-0.75
```

Курсор при этом останется справа от выведенного значения. Если необходимо перевести курсор на начало следующей строки экрана, необходимо воспользоваться квалификатором *endl* (END of Line), например:

```
cout<<"a="<<a<<" b="<<b<<" c="<<c<<endl;
```

Пример 1. Вывод на экран и чтение с клавиатуры стандартных типов данных.

```
#include <iostream>           //подключение потоковой библиотеки
                                //ввода-вывода
using namespace std;         //указание используемого пространства имен
std
int main()
{
    int a;
    float x;
    char c;
    char stroka[32];
    cout << "a = ";           //оператор вывода на экран текста "a = "
    cin >> a;                 //оператор ввода с клавиатуры значения переменной a
```

```

cout << "x = ";           //оператор вывода на экран текста "x="
cin >> x;                 //оператор ввода с клавиатуры значения переменной x
cout << "c = ";           //оператор вывода на экран текста "c="
cin >> c;                 //оператор ввода с клавиатуры значения переменной c
cout << "stroka = ";      //оператор вывода на экран
                           //текста "stroka="
cin >> stroka; /*оператор ввода с клавиатуры последовательности
                символов до первого пробела и запись этой
                последовательности в переменную stroka*/
cout << "Result:" << endl; /*оператор вывода на экран текста
                           "Result" и перевода курсора на начало следующей строки экрана*/

cout << a << " " << x << " " << c << " " << stroka <<
endl;
/*оператор вывода на экран через пробел значений a, x, c, stroka
и перевода курсора на начало следующей строки экрана*/
cout << "Stroka s probelami = ";
/*оператор вывода на экран текста "Stroka s probelami = "
cin.ignore(1, '\n'); //пропуск одного символа конца строки \n
cin.getline(stroka, 256, '\n'); //оператор ввода
                                с клавиатуры или строки символов со всеми пробелами,
                                если длина строки меньше 256, или первых 255 символов строки,
                                если длина строки больше или равна 256 */
cout << stroka << endl; /*оператор вывода на экран значения
                        переменной stroka и перевода курсора
                        на начало следующей строки экрана*/

return 0;
}

```

Пояснение. Для работы с объектами `cin` и `cout` подключаем библиотеку `iostream` и указываем, что будем использовать пространство имен `std`. Как видно из примера, теперь при вводе значений с клавиатуры нет необходимости задумываться, какой тип будет введен, т. е. строка-спецификация теперь не используется. По аналогии с функцией `scanf()` мы считываем строку до первого пробела: `cin >> stroka`. Чтобы считать строку, содержащую пробелы, пользуемся методом класса `istream::getline()`. Этот метод имеет три параметра: указатель на массив символов; максимальное число считываемых символов; символ, до которого происходит считывание. Нас интересует ввод до нажатия клавиши «Enter», поэтому указываем: `'\n'`.

При этом строка считывается до символа `'\n'`, НЕ включая его, т. е. он остается в буфере. Чтобы пропустить его, для ввода следующих переменных служит метод `ignore()`.

Приведем пример использования еще одного метода класса `istream`: `peek()`. Этот метод возвращает код следующего в буфере символа.

```
#include <iostream>
//подключение библиотеки потокового ввода-вывода
using namespace std;
//указание на то, что используется пространство имен std
int main()
{
    char c; //объявление символьной переменной
    char stroka[32]; //объявление массива символов — строки
    cout << "stroka = "; //вывод на экран приглашения к вводу строки
    cin >> stroka; //считываем последовательность символов до
    //1-го пробела и записываем ее в переменную stroka*/
    cout << "Result:" << endl; //вывод на экран текста Results:
    //и перевод курсора на начало следующей строки экрана*/
    cout << stroka << endl;
    //вывод на экран значения переменной stroka
    char s = cin.peek();
    //определяем следующий символ в буфере ввода
    cout << "Next simbol = " << (int)s << endl; //выводим
    //ASCII код введенного символа и переводим
    //курсор на начало следующей строки экрана */
    cin.ignore(1, '\n'); //пропуск одного символа конца строки \n
    cout << "Stroka s probelami = ";
    //оператор вывода на экран текста "Stroka s probelami = "
    cin.getline(stroka, 256, '\n'); //оператор ввода
    //с клавиатуры или строки символов со всеми пробелами,
    //если длина строки меньше 256, или первых 255 символов
    //строки, если длина строки больше или равна 256*/
    cout << stroka << endl; //оператор вывода на экран
    //значения переменной stroka и перевода курсора
    //на начало следующей строки экрана*/

    return 0;
}
```

Результат работы программы:

```
stroka = Hello_world!
Result:
Hello_world!
Next simbol = 10
Stroka s probelami = Hello world!
Hello world!
Для продолжения нажмите любую клавишу . . .
```

Пояснение. С помощью операции извлечения из потока с клавиатуры считывается последовательность символов, не включающая нажатую пользователем клавишу Ввод (Enter). Используя метод `peek()`, считывается следующий символ в буфере. Этим символом будет являться Enter, оставшийся в буфере после ввода строки. Убеждаемся в этом после вывода на экран его кода, который равен 10.

Ограничить число знаков, выводимых после запятой, можно, используя метод `precision()` класса `ostream`. Метод имеет один параметр — число знаков после запятой.

```
#include <iostream>
using namespace std;
int main()
{
    float x = 1.0/3;
    cout.precision(3);
    cout << x << endl;          /*значение x будет выведено с тремя
                                цифрами (знаками) после запятой — 0.333*/
    return 0;
}
```

18.6. Ввод/вывод пользовательских типов данных

Из приведенных примеров видно, что использование стандартных объектов `cin` и `cout` библиотеки `iostream`, а также операций извлечения из потока «>>» и вставки в поток «<<» делает очень удобным потоковый ввод и вывод стандартных типов данных в любой программе.

Однако это не распространяется на типы данных, созданные пользователем, например при попытке вставки в поток структурного типа данных в следующей программе:

```
#include <iostream>
using namespace std;
struct Anketa
{
    char name[32];
    int age;
};
```

```
int main()
{
    Anketa a1 = {"Ivanov", 23};
    cout << a1 << endl;           //ошибка в этой строке!
    return 0;
}
```

Система программирования выдаст ошибку, ссылаясь на то, что не знает, каким образом ей работать с объектом `a1`. Для того чтобы устранить эту ошибку и сделать удобными операции вставки в поток и извлечения из потока, нужно перегрузить эти операции.

Например:

```
#include <iostream>           //подключение потоковой библиотеки
                               //ввода-вывода

using namespace std;
    //указание на то, что используется пространство имен с именем std
struct Anketa                //начало определения структурного типа Anketa
{
    char name[32];            //определение поля name — имя
    int age;                  //определение поля age — возраст
};                            //конец определения структурного типа Anketa

ostream& operator << (ostream& out, Anketa& a) /*перегрузка
    операции вставки в поток для данного типа Anketa;
    out — ссылка на объект класса поток ostream*/
{
    out << a.name << ", " << a.age << endl;    /*оператор
    вставки в поток out значений a.name (типа строка) и a.age
    (типа int)*/
    return out;
    //возврат ссылки на поток out после вставки необходимых значений
}

istream& operator >> (istream& inp, Anketa& a) /*перегрузка
    операции извлечения из потока для данного типа Anketa;
    inp — ссылка на объект класса поток istream*/
{
    inp >> a.name >> a.age;    /*оператор извлечения из потока inp
    значений a.name (типа строка) и a.age (типа int)*/
    return inp;
    //возврат ссылки на поток inp после извлечения необходимых значений
}

int main()
{
    Anketa a1 = {"Иванов", 23};
```



```

    cout << a1;
    cin >> a1;
    cout << a1;
    return 0;
}

```

Результат работы программы:

```

Иванов, 23
Сидоров
34
Сидоров, 34
Для продолжения нажмите любую клавишу . . .

```

Пояснение. Для того чтобы не создавать копии потоков, параметрами функций являются ссылки на эти потоки. Отметим также, что при перегрузке операции «<<» при передаче параметра типа `anketa` ссылку можно опустить. А вот при перегрузке операции «>>» ссылка должна присутствовать обязательно, так как при ее отсутствии данные будут считаны во временную копию передаваемого объекта и после выхода из функции будут потеряны.

Аналогичным образом перегружаются операции вставки в поток и извлечения из потока для классов.

Пример. Вывод на экран и чтение с клавиатуры значений объектов класса `CPoint`.

```

#include <iostream>           //подключение потоковой библиотеки
                               //ввода-вывода
using namespace std;
    //указание на то, что используется пространство имен с именем std
class CPoint                  //начало определения класса CPoint
{
private:
    int x, y; //объявление закрытых членов-переменных класса CPoint
public:
    CPoint(int _x = 0, int _y = 0)
        //определение конструктора класса CPoint
    {
        x = _x;
        y = _y;
    }
friend ostream& operator << (ostream&, CPoint&);
    //прототип дружественного оператора вставки в поток класса ostream

```

```

friend istream& operator >> (istream&, CPoint&);
    //прототип дружественного оператора извлечения из потока класса
    istream

};
    //конец определения класса CPoint
ostream& operator << (ostream& out, CPoint& p)
//определение дружественного оператора вставки в поток класса ostream

{
    out << "Tochka: (" << p.x << ", " << p.y << ")." <<
        endl;
    /*оператор вставки в поток out текста "Tochka:
    (" и членов-переменных p.x и p.y класса CPoint*/
    return out;
    /*возврат ссылки на поток out после вставки
    в него информации класса CPoint*/
}
istream& operator >> (istream& inp, CPoint& p)
    //определение дружественного оператора извлечения
    //из потока класса istream

{
    inp >> p.x >> p.y;
    //оператор извлечения из потока inp значений
    //членов-переменных класса CPoint
    return inp;
    /*возврат ссылки на поток inp после вставки
    в него значений членов-переменных класса CPoint*/
}
int main()
{
    CPoint p1(4, 7);
    cout << p1;
    cin >> p1;
    cout << p1;
    return 0;
}

```

Результат работы программы:

Tochka: (4, 7).

3 4

Tochka: (3, 4).

Для продолжения нажмите любую клавишу . . .

Пояснение. В этом примере для класса CPoint перегружаются операторы «извлечения из потока» и «вставка в поток». Для доступа к закрытым полям класса перегруженные операции функции объявлены как дружественные классу. После загрузки операций становится возможен потоковый ввод/вывод объектов класса CPoint.

Тесты

1. Объявить пространство имен можно с помощью ключевого слова:

- а) workspace;
- б) namespace;
- в) userspace.

2. Операция «>>» называется:

- а) вставка в поток;
- б) извлечение из потока.

3. Базовым потоковым классом является:

- а) baseios;
- б) basestream;
- в) ios.

4. Что будет выведено на экран после выполнения кода

```
cout >> "Hello world";
```

- а) Hello;
- б) Hello world;
- в) программа не скомпилируется.

5. Что будет выведено на экран после выполнения кода

```
float x = 1.0 / 3;  
cout.precision(3);  
cout << x << endl
```

- а) 0.33333333;
- б) 0.333;
- в) программа не скомпилируется.

6. Можно ли в программе использовать следующие операторы?

```
Anketa a1 = {"Иванов", 23};  
cout << a1;
```

- а) да, можно в любом случае;
- б) нет, в любом случае;
- в) да, если оператор «<<» перегружен для структуры Anketa.

7. Укажите правильный вариант перегрузки операции «<<» для класса CPoint:

- а) ostream& operator << (ostream&, CPoint&);

- б) ostream operator << (ostream, CPoint);
- в) оба варианта правильные.

Задания

1. Разработайте два класса: Point2D (координаты точки на плоскости) и Point3D (координаты точки в пространстве). Поместите описание этих классов в разные пространства имен. Продемонстрируйте применение этих классов.
2. Перегрузите операции «>>» и «<<» для структурного типа Anketa, содержащей данные об учениках школы.
3. Перегрузите операции «>>» и «<<» для структурного типа Reis, содержащей данные о рейсах авиакомпании.
4. Перегрузите операции «>>» и «<<» для класса CVector, предназначенного для работы с одномерными массивами.
5. Перегрузите операции «>>» и «<<» для класса CMatrix, предназначенного для работы с матрицами.

Контрольные вопросы

1. Что такое пространство имен? Для чего оно используется?
2. Как в программе ввести собственное пространство имен? Приведите примеры.
3. Что такое поток данных? Что входит в поток? Приведите примеры потоков.
4. Какие стандартные классы потоков доступны для использования?
5. Нарисуйте иерархию потоковых классов.
6. Каким классам принадлежат объекты cin и cout?
7. Что необходимо сделать для ввода/вывода пользовательских типов данных с помощью операций извлечение из потока (>>) и вставка в поток (<<)?
8. Приведите примеры перегрузки операций «>>» и «<<» для структурных типов данных.

Глава 19

РАБОТА С ФАЙЛАМИ

19.1. Поточковый ввод/вывод файлов

Для работы с дисковыми файлами необходимо подключение заголовочного файла `<fstream>`, содержащего наборы специальных классов:

`ifstream` — для ввода;

`ofstream` — для вывода;

`fstream` — для работы с двунаправленными файловыми потоками.

Чтобы получить возможность работать с дисковым файлом, нужно открыть его с указанием режима доступа, который определяется значением константы `open-mode` класса `ios` (табл. 19.1).

Таблица 19.1. Режимы доступа при открытии файла

Режим доступа	Стандарт	Действие
<code>app</code>	Нет	Открывает файл для дозаписи
<code>ate(attend)</code>	Да	При открытии файла устанавливает файловый указатель на конец файла
<code>binary(bin)</code>	Да	Открыть файл в двоичном представлении
<code>in</code>	Да	Открыть файл только для чтения
<code>nocreate</code>	Нет	Если файл не существует, то новый файл не создается
<code>noreplace</code>	Нет	Если файл уже существует, файл не перезаписывается
<code>out</code>	Да	Открыть файл только для записи

Примечание. Если режим доступа является стандартом, то он будет осуществлять указанное действие вне зависимости от используемой среды разработки. В противном случае действие режима доступа может быть изменено.

Допускается использовать дизъюнкцию флагов. Например, для открытия файла одновременно для чтения и записи необходимо использовать следующее выражение: `ios::in | ios::out`. Если файл открывается с флагом `ios::in` и при этом файла с указанным именем не существует, то он будет создан, но не будет содержать никакой информации. Чтобы этого не происходило, необходимо использовать два флага одновременно: `ios::in | ios::nocreate`. В случае открытия файла для записи, чтобы не происходила перезапись ранее созданного файла, необходимо использовать флаг `ios::noreplace`. Операция «|» выполняет операцию побитового «или».

19.2. Работа с текстовыми файлами. Запись/чтение стандартных типов данных. Запись/чтение пользовательских типов данных

Для создания файловых потоков определяют объекты класса `ofstream` (`ifstream` или `fstream`). Так как эти классы являются производными от классов `ostream`, `istream`, `stream`, то при работе с ними можно использовать рассмотренные в предыдущей главе методы. После создания выходного файлового потока необходимо открыть соответствующий файл и связать его с потоком.

```
ofstream outputFile; //создали файловый поток
outputFile.open("Out.txt", ios::out);
//попытались связать его с файлом Out.txt,
```

Благодаря наличию различных конструкторов у класса `ofstream` можно объединить эти два действия:

```
ofstream outputFile("Out.txt", ios::out);
//создали файловый поток и попытались связать его с файлом Out.txt
```

При попытке открытия файла на запись могут возникнуть следующие ошибочные ситуации:

- нет места на диске;
- неправильно задано имя файла.

В обеих этих ситуациях выполнение программы прерывается и на экран выводится сообщение об ошибке без указания того, в каком месте программы эта ошибка произошла.

При этом метод `<имя_выходного_файлового_потока>.open` действует так:

- если файл можно открыть (и он был открыт), то экземпляру выходного файлового потока присваивается адрес созданного в оперативной памяти потока, связанного с файлом;
- если файл нельзя открыть, то экземпляру выходного файлового потока присваивается пустое значение `NULL`.

Поэтому после того, как предпринималась попытка открыть файл на запись (связать его с выходным файловым потоком), следует убедиться в том, что файл открыт и готов для записи (или перезаписи).

```
if (!outputFile)           /*если экземпляр выходного файлового потока
                           равен пустому значению NULL*/
{
    cout<<"Error: unable to write to out.txt"<<endl;
    //вывод сообщения, что нельзя открыть файл out.txt на запись
    exit(1);                //выход из программы
}
```

Все сказанное верно и для файлов, открываемых для чтения (или входных файлов).

После создания входного файлового потока необходимо открыть соответствующий файл и связать его с потоком.

```
ifstream inputFile;        //создали файловый поток
inputFile.open("Input.txt", ios::in);
                           //попытались связать его с файлом Input.txt,
```

Благодаря наличию различных конструкторов у класса `ifstream` можно объединить эти два действия:

```
ifstream inputFile("Input.txt", ios::in);
//создали файловый поток и попытались связать его с файлом Input.txt
```

При этом метод `<имя_выходного_файлового_потока>.open` действует так:

- если файл можно открыть (и он был открыт), то экземпляру выходного файлового потока присваивается адрес созданного в оперативной памяти потока, связанного с файлом;
- если файл нельзя открыть, то экземпляру выходного файлового потока присваивается пустое значение `NULL`.

При попытке открытия файла на чтение могут возникнуть следующие ошибочные ситуации:

- открывается несуществующий файл;
- неправильно задано имя файла.


```

    if (!inputFile)
        //если экземпляр входного файлового потока равен пустому значению NULL
    {
        cout << "Error open file!" << endl;
        //вывод сообщения об ошибке
        return 1;
        //выход из функции main
    }
    char c, next; /*объявление символьных переменных c — очередного
                  символа, next — следующего символа*/
    int count = 0; //объявление и инициализация числа символов в файле
    while(1)
        //цикл со всегда истинным условием
    {
        inputFile.get(c); //чтение очередного символа из файла
        next = inputFile.peek();
        //попытка чтения следующего символа из файла
        if (next == EOF)
            //если следующий символ находится за концом файла
        {
            cout << c; //вывести на экран последний символ файла
            count++; //увеличить на 1 число символов в файле
            break; //выйти из цикла
        }
        cout << c; //вывести на экран очередной символ файла
        count++; //увеличить на 1 число символов в файле
    }
    cout << endl; //перевод курсора на начало следующей строки экрана
    inputFile.close(); //закрыть доступ к файлу
    cout << "Count = " << count << endl;
    //вывод числа символов в файле

    return 0;
}

```

Результат работы программы:

```

Enter file name: input.txt
1 2 3 4 //после 4 в файле нет пробелов и нажат enter
a b c d //в конце этой строки тоже нет пробелов и enter не нажат
Count = 15 //в файле 15 символов
Для продолжения нажмите любую клавишу . . .

```

Пояснение. Для посимвольного чтения файла в бесконечном цикле (`while(1)`) используется ранее применяемый метод `get()`. В цикле происходит постоянная проверка следующего за текущим (очередным) символа: `next = inputFile.peek()`; . Как только доходим до символа «конец файла» (EOF), то происходит выход из цикла с помощью оператора `break`.

Пример 2. Написать программу, которая просит ввести имя файла с клавиатуры и записывает в указанный файл заглавные буквы английского алфавита, разделяя их пробелом.

```
#include <fstream>      //подключение библиотеки файловых потоков
#include <iostream>      //подключение потоковой библиотеки
                        //ввода-вывода
using namespace std; //указание на использование пространства имен
std
int main()
{
    ofstream outputFile;    //создание выходного файлового потока
    char name[32];
    cout << "Enter file name: ";
                        //вывод приглашения к вводу имени файла
    cin >> name;           //ввод с клавиатуры имени файла
    outputFile.open(name, ios::out);
                        //попытка открыть файл на запись
    if (!outputFile)
        //если экземпляр выходного файлового потока равен пустому значению NULL
    {
        cout << "Error open file!" << endl;
                        //вывод на экран сообщения об ошибке
        return 1;        //выход из функции main
    }
    char c;                //объявление символьной переменной c
    for (int i = 0; i<26; i++)
        //в цикле для всех букв латинского алфавита
    {
        c = 'A' + i;        /*любая латинская буква получается суммой
        латинской буквы 'A' и номера в латинском алфавите заданной буквы*/
        outputFile.put(c);    //запись в файл символа c
        outputFile.put(' '); //запись в файл пробела
    }
    outputFile.close();    //закрывать доступ к файлу
    return 0;
}
```

В результате работы программы в файл будет записана строка:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Пояснение. После считывания имени файла с клавиатуры с помощью метода `Open` файл открывается на запись. Запись букв англий-

ского алфавита происходит в цикле `for`. Счетчик цикла меняет свое значение от 0 до 25 включительно. Для последовательного получения символов алфавита к счетчику цикла прибавляется символ 'А'. Для записи символа в файл используется метод `put()`. После окончания работы с файлом необходимо закрыть поток, используя метод `close`.

Пример 3. С клавиатуры ввести имя файла. Вывести его содержимое на экран, подсчитать количество строк в файле.

```
#include <fstream>      //подключение библиотеки потоков fstream
#include <iostream>      //подключение потоковой библиотеки
                        //ввода-вывода
using namespace std;    //указание на использование пространства
                        //имен std

int main()
{
    ifstream inputFile;  //создание входного файлового потока
    char name[32];
    cout << "Enter file name: ";
                        //вывод приглашения к вводу имени файла
    cin >> name;         //ввод с клавиатуры имени файла
    inputFile.open(name, ios::in);
                        //попытка открыть файл на чтение
    if (!inputFile)
//если экземпляр входного файлового потока равен пустому значению NULL
    {
        cout << "Error open file!" << endl;
                        //вывод сообщения об ошибке
        return;        //выход из функции main
    }
    char str[256], next;
    //описание строки, считываемой из файла, и следующего символа файла
    int count = 0;
    //объявление и задание начального значения числу строк в файле
    while (1)           //цикл со всегда истинным условием
    {
        inputFile.getline(str, 256);
                        //чтение очередной (текущей) строки из файла
        next = inputFile.peek();
                        //попытка чтения следующего символа
        if (next == EOF)
            //если следующий символ находится за концом файла
```

```

    {
        cout << str << endl;          /*вывести на экран последнюю
строку файла и перевести курсор на начало следующей строки экрана*/
        count++;                      //увеличить на 1 число строк файла
        break;                        //выход из цикла
    }
    count++;                          //увеличить на 1 число строк файла
    cout << str << endl;              /*вывести на экран очередную
(текущую) строку файла и перевести курсор
на начало следующей строки экрана*/
}
cout << "Count = " << count << endl;
//вывести на экран количество строк в файле
inputFile.close();                  //закрыть доступ к файлу
return 0;
}

```

Результат работы программы:

```

Enter file name: input.txt
1 2 3 4
a b c d
5 6 7 8 9
e f g h g
Count = 4

```

Для продолжения нажмите любую клавишу . . .

Пояснение. Построковое чтение файла практически идентично посимвольному чтению. Только вместо метода `get()` используется метод `getline()`. При этом нужно не забывать, что этот метод читает строку до символа новой строки `'\n'`, не включая его.

Пример 4. Демонстрация построковой записи информации в файл.

```

#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    ofstream outputFile;
    outputFile.open("str.txt", ios::out);
    if (!outputFile)
    {
        cout << "Error open file!" << endl;
        return;
    }
}

```



```
outputFile <<"Я не знаю, где встретиться\n";  
                                     //записать в выходной файл первую строку  
outputFile <<"Нам придется с тобой,\n";  
                                     //записать в выходной файл вторую строку  
outputFile <<"Глобус крутится-вертится,\n";  
                                     //записать в выходной файл третью строку  
outputFile <<"Словно шар голубой\n";  
                                     //записать в выходной файл четвертую строку  
outputFile.close();                 //закрывать доступ к файлу  
}
```

Пояснение. Для записи в файл строки пользуемся операцией вставки в поток «<<». Записываемые строки являются строками типа `*char`, завершающимися символом `'\n'`.

19.4. Признак конца файла

Признак конца файла приходится искать в файлах, открытых для чтения. Этот признак устанавливается в тот момент, когда в файле не осталось больше данных, которые можно считать. Количество байт существующего файла записано в файловой системе компьютера. При связывании входного потока с конкретным файлом размер файла, т. е. количество байт в файле, автоматически переписывается в соответствующую член-переменную потока.

Признак конца файла анализируется в выражении вида

```
while (! inputFile.eof())  
{ ... }
```

где `.eof()` – `EndOfFile` — метод проверки на конец файла.

Однако такая проверка на конец файла не анализирует ошибки, которые могут встретиться в процессе чтения файла. Например, если файл во время работы с ним был поврежден сторонней программой, или часть файла была физически потеряна на жестком диске, но его прежний размер сохранился в файловой системе компьютера, то такие ситуации не будут корректно обработаны методом `eof`.

Для проверки как конца файла, так и наличия ошибок при его чтении можно воспользоваться методом `good()`:

```
while (inputFile.good())  
{ ... }
```

Этот метод в отличие от метода `eof` не только проверяет, что при чтении файла достигнут его конец, но также проверяет, чтобы файл не был испорчен сторонней программой, что информация, хранящаяся в файле, доступна и может быть прочитана.

Пример 5. С клавиатуры ввести имя текстового файла, создать файл с именем `copy.txt` записать в него копию исходного файла.

```
#include <fstream>      //подключение библиотеки файловых потоков
#include <iostream>      //подключение потоковой библиотеки
                        //ввода-вывода

#include <string.h>
using namespace std; //указание на использование пространства имен
std
int main()
{
    ifstream inputFile;           //исходный файл
    ofstream outputFile;         //файл-копия
    char name[32];
    cout << "Enter file name: ";
                                //вывод приглашения к вводу имени файла
    cin >> name;                 //ввод имени файла с клавиатуры
    inputFile.open(name, ios::in);
                                //попытка открыть файл на чтение
    if (!inputFile)
//если экземпляр входного файлового потока равен пустому значению NULL
    {
        cout << "Error open file!" << endl;
                                //вывод сообщения об ошибке
        return 0;               //выход из функции main
    }
    outputFile.open("copy.txt", ios::out);
    if (!outputFile)
//если экземпляр входного файлового потока равен пустому значению NULL
    {
        cout << "Error open file!" << endl;
                                //вывод сообщения об ошибке
        return 0;               //выход из функции main
    }
    char str[256];
    while(inputFile.good())
                                //выполняем цикл до тех пор, пока с файлом все хорошо
    {
        inputFile.getline(str, 256);
```

```
                                //читаем строку из исходного файла
    outputFile << str << endl;    //и пишем ее в файл-копию
}
cout << endl; //перевод курсора на начало следующей строки экрана
inputFile.close();                //закрывать доступ к файлу
outputFile.close();
return 0;
}
```

Пояснение. Исходный файл открывается для чтения (флаг `ios::in`), а файл, в который будет копироваться информация, — на запись. Оба файла проверяются на корректное открытие. Чтение данных осуществляется в цикле `while` с условием `inputFile.good()`. Использование этого метода позволяет не только отследить достижение конца файла, но и проверить его целостность.

19.5. Чтение и запись в файл стандартных типов данных

Как было показано в примере 4 предыдущего параграфа, массивы символов можно записывать в файл, используя операцию «<<». Аналогичным образом можно считывать и записывать любые стандартные типы данных. То есть если мы знаем, что в файле находятся только целые числа,

```
int x;
```

разделенные пробелами, то можем считать их следующим образом:

```
inputFile >> x;
```

Пример 6. Дан числовой файл `inp.txt`. Выбрать все значения, которые делятся нацело на 2 и 4, но не делятся на 6. Записать эти значения в файл `out1.txt`, а все остальные — в файл `out2.txt`.

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <conio.h>
using namespace std;
int main()
{
    int x;
```



```

    ifstream in_file("inp.txt", ios::in); /*создание входного
    потока in_file и попытка связать его с файлом inp.txt, открытым
                                         на чтение*/

    if (!in_file)
//если экземпляр входного файлового потока равен пустому значению NULL
    {
        cout<<"Error input file"<<endl;
//вывод сообщения об ошибке во входном файле
        exit(1); //выход из программы
    }
    ofstream out_file1("out1.txt", ios::out); /*создание
    выходного потока out_file1 и попытка связать его
    с файлом out1.txt, открытым на запись*/
    if (!out_file1) /*если экземпляр 1-го выходного файлового потока
    равен пустому значению NULL*/
    {
        cout<<"Error output file1"<<endl;
//вывод сообщения об ошибке в 1-м выходном файле
        exit(1); //выход из программы
    }
    ofstream out_file2("out2.txt", ios::out); ofstream
    out_file1("out1.txt", ios::out); /*создание выходного потока
    out_file2 и попытка связать его с файлом out2.txt, открытым
                                         на запись*/
    if (!out_file2) /*если экземпляр 1-го выходного файлового
    потока равен пустому значению NULL*/
    {
        cout<<"Error output file2"<<endl;
//вывод сообщения об ошибке в 1-м выходном файле

        exit(1); //выход из программы
    }
    while (in_file) //пока из файла можно что-то считать
    {
        in_file >> x; //считать из файла целое число
        if ( x%2 == 0 && x%4 == 0 && x%6 != 0 )
/*если остаток от деления x на 2 и на 4 равен 0,
    но остаток от деления x на 6 не равен 0*/
            out_file1 << x << " "; //записать x в outfile1
        else
            out_file2 << x << " ";
//иначе записать x в outfile1
    }
    in_file.close();

```

```
outfile1.close();
outfile2.close();
return 0;
}
```

19.6. Чтение и запись в файл пользовательских типов данных

Так как классы `ifstream`, `ofstream` и `fstream` являются производными соответственно от классов `istream`, `ostream` и `stream`, то по умолчанию при использовании следующего кода:

```
Anketa a1 = {"Sidorov", 30};
ofstream outF("info.txt", ios::out);
outF << a1;
```

компилятор выдаст ошибку. Решить эту проблему можно путем перегрузки операций «<<» и «>>»:

```
ofstream& operator << (ofstream& outF, Anketa& a)
{
    outF << a.name << ", " << a.age << endl;
    return outF;
}
```

Пример 7. В текстовом файле содержатся сведения о телефонах абонентов. Структурный тип содержит поля: фамилия абонента, место жительства (название улицы, номер дома), год установки телефона.

Написать программу, которая выводит на экран информацию об абонентах, фамилия которых начинается на введенную букву.

```
#include <fstream>
#include <iostream>
using namespace std;
struct Abonent //начало определения структурного типа Abonent
{
    char surname[32]; //определение поля surname — фамилия
    char street[32]; //определение поля street — название улицы
    int numberHouse;
    int year;
};

//запись структуры в файл (в программе не используется)
```

```

ofstream& operator << (ofstream& outF, Abonent& a)
    /*перезгрузка операции вставки в файловый поток для данного типа
      Abonent ; outF — ссылка на объект класса поток ofstream*/
{
    outF << a.surname << ", " << a.street << ", " <<
a.numberHouse << ", " << a.year << endl; /*оператор вставки
      в поток outF значений a.surname(типа строка), a.street
      (типа строка), a.numberHouse (int) и a.a.year (типа int)*/

    return outF;
    //возврат ссылки на файловый поток outF после вставки
      //необходимых значений
}

    //вывод структуры на экран
ostream& operator << (ostream& out, Abonent& a) /*перезгрузка
      операции вставки в поток для данного типа Abonent ;
      out — ссылка на объект класса поток ostream*/
{
    out << a.surname << ", " << a.street << ", " <<
a.numberHouse << ", " << a.year << endl; /*оператор вставки
      в поток out значений a.surname(типа строка),
      a.street (типа строка), a.numberHouse (int) и a.year (типа int)*/
    return out;
    //возврат ссылки на поток out после вставки необходимых значений
}

    //чтение структуры из файла
ifstream& operator >> (ifstream& inpF, Abonent& a)
    /*перезгрузка операции изъятия из файлового потока для данного типа
      Abonent ; inpf — ссылка на объект класса файловый поток
      ifstream*/
{
    inpF >> a.surname >> a.street >> a.numberHouse >>
a.year; /*оператор изъятие из файлового потока inpf значений
      a.surname (типа строка), a.street (типа строка),
      a.numberHouse (int) и a.year (типа int)*/
    return inpF; /*возврат ссылки на файловый поток inpF после
      изъятия необходимых значений*/
}

int main()
{
    Abonent ab; //объявление структуры ab типа Abonent
    ifstream inpF("info.txt", ios::in); /*создание входного
      потока inpF и попытка связать его
      с файлом info.txt, открытым на чтение*/

    char c, next; /*объявление переменных c — очередной (текущий)
      символ и next — следующий символ потока*/

```



```

int count = 0;
//объявление и инициализация числа абонентов в файле
cout << "Enter simbol: "; //вывод приглашения к вводу символа
cin >> c; //ввод символа с клавиатуры
while(1) //в цикле со всегда истинным условием продолжения
{   inpF >> ab; //чтение из файла сведений об абоненте
    next = inpF.peek();
    //попытка чтения следующего символа из файла
    if (ab.surname[0] == c)
        //если первая буква фамилии абонента совпадает с заданным символом
        {   cout << ab; //вывести на экран сведения об абоненте
            count++; //увеличить на 1 число абонентов с фамилией
                //на заданный символ*/
        }
    if (next == EOF)
        //если следующий символ находится за концом файла
        break; //выход из цикла
}
if (count == 0)
    cout << "Abonentov net!" << endl;
return 0;
}

```

19.7. Произвольный доступ к элементам файлов

Каждый файл имеет два связанных с ним значения: указатель чтения и указатель записи, по-другому называемые файловым указателем (указатель чтения) или текущей позицией (указатель записи).

При последовательном доступе к элементам файлов перемещение файлового указателя чтения (записи) происходит автоматически от начала файла к его продолжению. Но иногда возникает необходимость принудительно изменить или узнать его текущее положение. Для этого используются следующие функции:

Методы принудительной установки указателей чтения и записи:

`seekg()` — установить текущий указатель чтения на заданную позицию;

`seekp()` — установить текущий указатель записи на заданную позицию.

Методы имеют два параметра: 2-й — точка отсчета, от которой отсчитывается смещение. 2-й параметр может принимать одно из значений из табл. 19.2.

Таблица 19.2. Допустимые точки отсчета при произвольном доступе к файлу

Именная константа	Числовое значение	Положение в файле
beg	0	Начало файла
cur	1	Текущая позиция файла
end	2	Конец файла

1-й параметр метода — смещение в байтах относительно заданной точки отсчета (2-й параметр метода). Тип первого параметра эквивалентен типу long, но лучше использовать следующее определение: `typedef long streampos; typedef long streamoff`.

Методы определения текущего положения указателей чтения и записи:

`tellg()` — проверить текущий указатель чтения;

`tellp()` — проверить текущий указатель записи.

Буква «g» обозначает getting (метод относится к указателю чтения), а буква «p» — printing (метод относится к указателю записи).

Эти методы не имеют параметров, а возвращают смещение файлового указателя относительно начала файла.

Пример 8. Написать свой класс для работы с двунаправленными файловыми потоками. Продемонстрировать применение класса.

```
#include <fstream>           //подключение библиотеки файловых потоков
class CPoint                 //начало определения класса CPoint
{
private:
    int x, y;
    //определение целых закрытых членов-переменных класса CPoint
public:
    CPoint(int _x = 0, int _y = 0)
        //определение конструктора с двумя параметрами класса CPoint
    {
        x = _x; y = _y;
    }
    void print()              /*определение члена-функции вывода информации
                               об объекте класса CPoint*/
    {
        cout << "Tochka (" << x << ", " << y << ")." <<
            endl;
    }
}
```

```

friend class MyTxtFile;
    //указание класса MyTxtFile, дружественного классу CPoint
}; //конец определения класса CPoint
class MyTxtFile : public fstream
    //начало определения класса MyTxtFile — наследника класса fstream
{
public:
    /*определение конструктора класса MyTxtFile, инициализированного
    конструктором класса fstream*/
    MyTxtFile(char *name) : fstream(name, ios::out |
        ios::in)
    {}
    MyTxtFile& operator << (CPoint& p)
        //перезгрузка операции вставки в поток MyTxtFile для типа CPoint ;
    {
        fstream(*this) << " " << p.x << " " << p.y <<
            endl; /*операция вставки в поток типа fstream значений
                членов-переменных объекта класса CPoint*/
        return *this; /*возврат ссылки на тот объект, с которым
                происходит работа, после вставки в поток
                членов-переменных объекта класса CPoint*/
    }
    MyTxtFile& operator >> (CPoint& p)
        //перезгрузка операции извлечения из потока MyTxtFile для типа CPoint
    ;
    {
        fstream(*this) >> p.x >> p.y; /*операция извлечения из
            потока типа fstream значений членов-переменных объекта
            класса CPoint*/
        return *this; /*возврат ссылки на тот объект, с которым
            происходит работа, после извлечения из потока
            членов-переменных объекта класса CPoint*/
    }
}; //конец определения класса MyTxtFile

int main()
{
    MyTxtFile file("test.txt");
        //создание потока и попытка связать его с файлом test.txt
    if (!file)
    {
        cout << "Error open file!" << endl;
        return -1;
    }
}

```



```

CPoint p1(3, 5), p2(7, 9);
                                //создание объектов p1 и p2 класса Cpoint
p1.print();                     //будет напечатано Toчка (3, 5)
p2.print();                     //будет напечатано Toчка (7, 9)
    file << p1 << p2;          //запись объектов p1 и p2 в начало файла
file.seekg(0L, ios::beg);
                                //установить указатель чтения на начало файла
file >> p2 >> p1;              //чтение объектов p2, p1 из начала файла
p1.print();                     //будет напечатано Toчка (7, 9)
p2.print();                     //будет напечатано Toчка (3, 5)
file.close();                   //закрывать доступ к файлу
return 1;
}

```

19.8. Работа с двоичными файлами

Вся информация хранится в компьютере в виде 0 и 1, т. е. в двоичном виде. Двоичные файлы отличаются от текстовых только методами работы с ними. Например, если мы записываем в текстовый файл цифру «4», то она записывается как символ, и для ее хранения нужен один байт. Соответственно и размер файла будет равен одному байту. Текстовый файл, содержащий запись: «145687», будет иметь размер шесть байт.

Если же записать целое число 145 687 в двоичный файл, то он будет иметь размер четыре байта, так как именно столько необходимо для хранения данных типа `int`. То есть двоичные файлы более компактны и в некоторых случаях более удобны для обработки.

19.9. Запись стандартных типов данных в двоичные файлы

Для того чтобы открыть двоичный файл, необходимо задать режим доступа `ios::binary` (в некоторых компиляторах C++ — `ios::bin`).

Для создания выходного файла создают объект:

```

ofstream outBinFile("out.bin", ios::out | ios::binary);
/*создание объекта класса ofstream и попытка связать его
с файлом out.bin в режиме записи двоичного файла */

```

```

if (!out_fil)                                     //стандартная проверка
{
    cout << "Error!"<<endl;
    exit(1);
}

```

Запись данных происходит с помощью метода `write()`, который имеет два параметра: первый — указатель на начало (адрес начала) записываемых данных, второй — количество записываемых байтов. При этом указатель необходимо явно преобразовать к типу `char`.

Пример 1. Записать в двоичный файл переменные различного типа:

```

#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    ofstream outBinFile("test.bin", ios::out |
        ios::binary);    /*создание объекта класса ofstream и попытка
        связать его с файлом test.bin в режиме записи двоичного файла */

    int a = 145687;        //объявление целой переменной a
    outBinFile.write((char*)&a, sizeof(a));    /*запись в файл
        переменной a как потока байтов, т. е. запись
        в файл внутреннего представления целой переменной a*/
    float x = 123.25;    //объявление вещественной переменной x
    outBinFile.write((char*)&x, sizeof(x));    /*запись в файл
        переменной x как потока байтов, т. е. запись в файл
        внутреннего представления целой переменной x*/

    char c = 'g';
    //определение символьной переменной c и инициализация ее символом g
    outBinFile.write((char*)&c, sizeof(c));
    //запись символа g в файл

    outBinFile.close();
    return 0;
}

```

Если открыть содержимое файла `test.bin` текстовым редактором, то он будет иметь вид:

```
9  AЎBg
```

а размер файла составит 9 байт.

19.10. Чтение стандартных типов данных из двоичных файлов

Для того чтобы открыть существующий двоичный файл для чтения, нужно создать объект:

```
ifstream inpBinFile("inp.bin", ios::in | ios::binary);
    /*используем дизъюнкцию флагов, указывающую на то
    что файл открывается на чтение в двоичном виде*/
if (! inpBinFile)
{
    cout<<"Error!"<<endl;
    exit(2);
}
```

Для чтения данных используем функцию `read()`, имеющую аналогичные функции `write()` параметры.

Пример 2. Считать данные из файла «test.bin», вывести их на экран.

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    ifstream inpBinFile("test.bin", ios::in |
        ios::binary);    //открываем файл на чтение в двоичном виде
    int a;
    float x;
    char c = 'g';
    inpBinFile.read((char*)&a, sizeof(a));
        //читаем целочисленную переменную
    inpBinFile.read((char*)&x, sizeof(x));
        //читаем вещественную переменную
    inpBinFile.read((char*)&c, sizeof(c));
        //читаем символьную переменную

    inpBinFile.close();
    cout << "a = " << a << endl;
    cout << "x = " << x << endl;
    cout << "c = " << c << endl;
    return 0;
}
```


Результат работы программы:

```
a = 145687
x = 123.25
c = g
Для продолжения нажмите любую клавишу . . .
```

Обратите внимание, что при использовании функции `write` и `read` не происходит никакого преобразования информации. В файл записывается и считывается внутреннее представление данных. Именно поэтому две предыдущие программы дали правильный результат.

19.11. Запись и чтение пользовательских типов данных в двоичные файлы

В отличие от текстовых файлов, работа с пользовательскими типами данных с использованием двоичных файлов ничем не отличается от стандартных типов данных. Аналогично используются методы `write()` и `read()`. Программисту только остается указать адрес записываемого участка памяти и количество записываемых байтов, учтя при этом, что никакого преобразования данных не происходит, записывается и считывается только внутреннее представление информации.

Также при работе с двоичными файлами могут использоваться методы `seekg()`, `tellg()`, `seekp()`, `tellp()`.

Пример 3. Написать программу, которая записывает сведения о группе туристов в двоичный файл.

```
#include <fstream>
#include <iostream>
using namespace std;
struct Anketa
{
    char name[32];
    int age;
};
int main()
{
    fstream BinFile("ankety.bin", ios::in | ios::out |
        ios::binary);
```

```

    if (!BinFile)
    {
        cout << "Error open file!" << endl;
        return -1;
    }
    Anketa Gruppa[3] = {"Ivanov", 23}, {"Sidorov", 21},
        {"Petrov", 22}};
    for (int i = 0; i<3; i++)
        BinFile.write((char*)&Gruppa[i], sizeof(Anketa));
    BinFile.close();
    return 0;
}

```

Пример 4. В файле «ankety.bin» содержатся данные о группе туристов, необходимо считать их и вывести на экран.

```

#include <fstream>
#include <iostream>
using namespace std;
struct Anketa
{
    char name[32];
    int age;
};

    /*перезгрузка операции вставки в поток для вывода пользовательского
    структурного типа данных Anketa на экран*/
ostream& operator << (ostream& out, Anketa& a)
{
    out << a.name << ", " << a.age << endl;
    return out;
}

int main()
{
    fstream BinFile("ankety.bin", ios::in | ios::out |
        ios::binary);
    if (!BinFile)
    {
        cout << "Error open file!" << endl;
        return -1;
    }
    Anketa Gruppa[3];
    for (int i = 0; i<3; i++)
    {
        //сразу читаем все байты, занимаемые переменной типа Anketa
        BinFile.read((char*)&Gruppa[i], sizeof(Anketa));
    }
}

```

```

        cout << Gruppa[i];
    }
    BinFile.close();
    return 0;
}

```

Результат работы программы:

```

Ivanov, 23
Sidorov, 21
Petrov, 22
Для продолжения нажмите любую клавишу . . .

```

19.12. Разработка собственных классов для работы с файлами

Постоянно пользоваться методами `write()` и `read()` неудобно, гораздо приятнее иметь возможность пользоваться операциями «<<» и «>>» по аналогии с текстовыми файлами. Приведем пример реализации своего класса для работы с двоичными файлами.

```

#include <fstream>
#include <iostream>
using namespace std;

struct Anketa          //объявляем структуру для хранения информации
{
    char name [32];
    int age;
    char telefon [32];
};

/*перезгрузка операции вставки в поток для вывода пользовательского
структурного типа данных Anketa на экран*/
ostream& operator << (ostream& out, Anketa& ank)
{
    out << ank.name << ", " << ank.age << ", " <<
        ank.telefon << "." << endl;
    return out;
}

class outBinaryFile : public ofstream /*определяем свой класс
для работы с выходными бинарными файлами. Порождаем его от
класса работы с выходными файловыми потоками*/

```



```

{
public:
    /*при описании конструктора порожденного класса не забываем вызвать
       конструктор базового, передав ему необходимые параметры*/
    outBinaryFile(char* name) : ofstream(name, ios::out |
        ios::binary)
    {
        //перезгружаем необходимые операции как методы класса
        outBinaryFile& operator << (int chislo)
        {
            write((char*)&chislo, sizeof(chislo));
            return *this;
        }
        outBinaryFile& operator << (Anketa ank)
        {
            write((char*)&ank, sizeof(ank));
            return *this;
        }
    };
}

class inpBinaryFile : public ifstream /*определяем свой класс
                                         для работы с входными бинарными файлами. Порождаем его от
                                         класса работы с входными файловыми потоками*/
{
public:
    inpBinaryFile(char* name) : ifstream(name, ios::in |
        ios::binary)
        /*вызова конструктора базового класса с необходимыми параметрами,
           достаточно для конструктора порожденного класса*/
    {
        //перезгружаем необходимые операции
        inpBinaryFile& operator >> (int& chislo)
        {
            read((char*)&chislo, sizeof(chislo));
            return *this;
        }
        inpBinaryFile& operator >> (Anketa& ank)
        {
            read((char*)&ank, sizeof(ank));
            return *this;
        }
    };
}

int main()
{
    Anketa anketa = {"Kolya", 1990, "582-78-95"};
}

```

```
int a = 111, b = 112;
outBinaryFile outFile("dannye.bin");
                                     //открываем файл на чтение
if (!outFile)
{
    cout << "Error output file!" << endl;
    return 1;
}
outFile << a << b << anketa;
outFile.close();

a = 0; b = 0;
inpBinaryFile inFile("dannye.bin");
if (!inFile)
{
    cout << "Error output file!" << endl;
    return 1;
}
for (int i = 0; i<2; i++)
{
    inFile >> a;
    cout << a << endl;
                                     //читаем анкету из файла
                                     //и выводим ее на экран
}
inFile >> anketa;
cout << anketa;
inFile.close();
return 0;
}
```

Результат работы программы:

```
111
112
Kolya, 1990, 582-78-95.
Для продолжения нажмите любую клавишу . . .
```

Тесты

1. Можно ли в программе использовать операцию?

`ios::in | ios::out`

- а) да, в любом случае;
- б) да, но только при работе с текстовыми файлами;
- в) нет, в любом случае.

2. Укажите правильный вариант открытия текстового файла для чтения:

- а) `ifstream inpF("input.txt", ios::in);`
- б) `ifstream inpF("input.txt", ios::input);`
- в) `ifstream inpF(ios::in, "input.txt").`

3. Что будет выведено на экран в результате выполнения следующего кода?

```
while(1)
{
    inputFile.get(c);
    next = inputFile.peek();
    if (next == EOF)
    {
        cout << c;
        break;
    }
    cout << c;
}
```

- а) содержимое файла, связанного с потоком `inputFile`, выведется на экран один раз;
- б) содержимое файла, связанного с потоком `inputFile`, будет выводиться на экран бесконечное число раз;
- в) на экран ничего не будет выведено.

4. Сколько символов содержится в файле?

```
1 2 3
4 5 6
```

- а) 6;
- б) 7;
- в) 11.

5. Какие методы позволяют определить конец файла?

- а) `eof();`
- б) `good();`
- в) оба указанных метода.

6. Для чего предназначена функция `getline()`?

- а) считывает слово из файла;
- б) считывает все содержимое файла;
- в) считывает строку из файла.

7. Чтобы записывать/считывать пользовательские типы данных в файл, необходимо:

- а) перегрузить операции «>>» и «<<»;
- б) запись и чтение пользовательских типов данных доступны без дополнительных действий;
- в) запись и чтение пользовательских типов данных в файл невозможны.

8. Какие функции используются для записи/чтения информации в двоичном виде?

- а) printf / scanf;
- б) write / read;
- в) put / get.

Задания

1. Написать программу, которая записывает в файл буквы английского алфавита.

2. В файле input.txt записана информация из нескольких текстовых строк. Вывести содержимое этого файла на экран, посчитать количество строк в файле.

3. На диске находится файл result.txt с результатами химических экспериментов. Написать программу, создающую копию этого файла с именем copy_result.txt.

4. С клавиатуры ввести имя файла. В указанном файле удалить все четные строки.

5. Написать программу, которая в текстовом файле, заменяет все строчные буквы прописными, и наоборот.

6. В исходном текстовом файле находятся числа, разделенные пробелами. Сформировать два новых файла: первый должен содержать только четные числа, а второй — нечетные.

7. В файл записаны вещественные числа. Написать программу, которая отбрасывает дробную часть у этих чисел и записывает их в новый файл.

8. В текстовом файле записана информация о рейсах авиакомпании. Выбрать из этих данных рейсы, вылетающие после обеда, и вывести их на экран.

9. Перегрузить операторы >> и << так, чтобы можно было записывать в файл информацию об объектах классов CPoint, CVector, CMatrix.

10. Написать собственный класс для работы с бинарными файлами.

11. Записать список 10 учеников класса в текстовый файл и в двоичный файл. Сравнить эти файлы. Объяснить полученное отличие.

12. Разработать класс, который записывает в файл информацию об автомобилях (год выпуска, марку, цвет и т. д.) в текстовый файл. При этом каждый символ информации заменяется своим ASCII-кодом. Полученный файл вывести на экран.

Контрольные вопросы

1. Какие классы используются для работы с файловыми потоками?
2. Какие режимы доступа могут использоваться при работе с файлами? Приведите примеры.
3. Какой метод служит для открытия файла? Приведите примеры.
4. Какие операции доступны для работы с файлами? Какие функции предназначены для выполнения этих операций?
5. Какие методы позволяют определить конец файла при чтении из него информации? В чем отличие этих методов? Приведите примеры.
6. Каким образом можно считать переменные стандартных типов данных из текстовых файлов?
7. Можно ли считывать из текстовых файлов переменные пользовательских типов данных?
8. Какие функции предназначены для произвольного чтения информации из файла? Приведите примеры.
9. Назовите особенности двоичных файлов. В чем преимущества использования таких файлов?
10. С помощью каких функций можно записывать/считывать информацию в двоичные файлы?
11. Как считать переменные стандартных типов данных из двоичного файла?
12. Какие особенности нужно учитывать при чтении пользовательских типов данных из двоичных файлов?

Глава 20

ОБРАБОТКА ИСКЛЮЧЕНИЙ

20.1. Конструкция try — catch

Исключительной ситуацией, или исключением (exception), называется прерывание нормального потока выполнения программы в ответ на непредвиденное или аварийное событие, которое порождается ошибками в аппаратуре или в самой программе, например делением на нуль или обращением к памяти по некорректному адресу, а также генерируется программно, например, функциями динамической библиотеки или Win32 при возникновении ситуации, не позволяющей завершить задачу.

Язык C++ предоставляет простой механизм обработки исключений перечисленных типов. Если обработчик для какого-либо исключения не предусмотрен, то стандартный обработчик генерирует сообщение об ошибке и завершает выполнение программы. При обработке исключения можно либо исправить ошибки и продолжить корректную работу программы, или освободить все используемые ресурсы и корректно завершить программу.

Обработка программных исключений. При тестировании и отладке программы программист может объявить любую ее часть контролируемой, например:

- ввод исходных данных;
- вычисления по определенным формулам;
- переход по какой-то ветви по определенному условию;
- проход определенной итерации цикла;
- выход из цикла и т. д.

В этом случае он особым образом выделяет эту часть программы и выбирает любую ситуацию, которую ему необходимо проверить при

выполнении программы, например выход за границы массива, деление на нуль, достижение какой-то переменной некоторого значения и т. д. Выбранная часть программы называется контролируемой ее частью, а выбранная ситуация — исключением, на которую программа должна прореагировать особым образом — вывести на экран специальное сообщение, прервать свою работу и вызвать обработчик исключения.

Покажем на примере, как обрабатывать исключения, генерируемые программно, с использованием оператора `throw`, за которым следует некоторое значение. Значение может быть константой, переменной или объектом (т. е. экземпляром класса, структуры или объединения), оно предназначено для передачи информации об исключении, которая может использоваться его обработчиком. В приведенном ниже фрагменте программы при некорректном выделении памяти генерируется исключение со строкой, поясняющей ошибку.

```
char *Buffer = new char [1000];
if (Buffer == 0)
    throw "out of memory";                                //нехватка памяти
```

Если исключение сгенерировано, но в программе не предусмотрена его обработка, механизм исключений вызывает из стандартной динамической библиотеки языка C++ функцию завершения `terminate`, которая выдает сообщение «*abnormal program termination*», которое не сообщает никаких сведений о причине и месте прерывания программы и прекращает выполнение программы.

Для обработки такого исключения с сообщением причины прерывания и выполнения необходимых только в этом случае действий необходимо предусмотреть операторы `try` и `catch` (задать контролируемую часть программы), как показано в следующем примере.

```
try
{
    //начало контролируемого блока
    //операторы ...
    char *Buffer = new char [1000];
    if (Buffer == 0)
        throw "out of memory";
    //генерация исключения с параметром — строкой
    //операторы...
}
//конец контролируемого блока
//обработка исключения
```

```
catch (char *ErrorMsg) /*определение функции обработки ошибки,  
                        параметром которой является любая строка char *ErrorMsg*/  
{  
    cout << ErrorMsg << '\n'; /*вывод сообщения об ошибке (обработка  
                               ошибки) и вызов диалогового окна для возобновления выполнения  
                               программы или вызов функции exit для остановки работы программы*/  
} //конец блока обработки исключений контролируемого блока  
/*здесь выполнение программы продолжается, если не произошло прерывание  
работы программы в контролируемом блоке...*/
```

Если исключение генерируется в любом месте контролируемого блока, следующем за оператором try (или внутри любой функции в этом блоке), то управление передается за пределы блока try. Если за try следует подходящий блок catch, то управление переходит к нему. Блок catch начинается с объявления в круглых скобках. Если тип параметра в этом объявлении совпадает с типом значения в операторе throw, генерирующем исключение, то управление передается данному блоку catch. При несовпадении типов параметров программа ищет другой обработчик catch, тип параметра которого совпадает с типом параметра исключения, и т. д. до конца блока обработки исключений контролируемого блока. После выполнения кода в блоке catch управление передается первому оператору, следующему за блоком обработки исключений, и программа возобновляет работу в нормальном режиме (если блок catch не содержит оператор return или вызов функции exit). Таким образом, операторы try и catch могут предотвратить завершение программы стандартным обработчиком исключений, заменив обработку исключений специальным, понятным программистам способом с указанием места и причины исключения.

Блок try называют охраняемым (контролируемым) разделом кода. Если исключение не сгенерировано, то поток управления «перепрыгнет» все блоки catch и перейдет на первый, следующий за блоком оператор.

В приведенном выше примере оператор throw содержит строку «out of memory». Так как тип параметра в объявлении catch (char *) такой же, то данный блок catch получает управление при вызове этого исключения.

Обратите внимание: оператор catch объявляет параметр типа char * с именем ErrorMsg, что позволяет внутри блока catch получить доступ к значению, заданному в throw. Данный механизм очень

похож на механизм передачи параметров функции. Чтобы понять это, можно представить оператор `throw` как вызов функции:

```
throw "out of memory";
```

в котором значение («out of memory») передается в функцию.

Также можно представить блок `catch` как вызываемую функцию, а объявление `char * Errormsg` как объявление ее формального параметра. Как и параметры функции, переменная `ErrorMsg` доступна только внутри `catch`.

Заметьте: в операторе `catch` может содержаться только описание типа без имени параметра.

```
catch (char *)
{
    //не использует значение, определенное в операторе throw
}
```

В этом случае блок `catch`, как и прежде, получит управление по оператору `throw "out of memory"`, но без доступа к значению типа `char *`.

Если оператор `catch` определяет какой-либо другой тип параметра (например, `int`), блок не получит управление при возникновении данного исключения.

```
catch (int ErrorCode)
{
    /*управление НЕ будет получено после выполнения
    оператора throw "out of memory"*/
};
```

Если поместить несколько `catch` за блоком `try`, то можно управлять различными типами исключений. Например, в следующем фрагменте программы обрабатываются исключения с аргументом типа `char *` или `int`.

```
try
{
    char *Buffer1 = new char [1000]; /*попытка выделить память
    под 1000 символов и записать указатель на эту область памяти
    в Buffer1*/
    if (Buffer1 == 0)
        //если память не удалось выделить, генерируем исключение
        throw "out of memory Buffer1"; /*генерация исключения
}
```



```

        с параметром-строкой — это исключение обрабатывается
        первым блоком catch*/
char *Buffer2 = new char [1000]; /*попытка выделить память
        под 1000 символов и записать указатель
        на эту область памяти в Buffer2*/
if (Buffer2 == 0)
    //если память не удалось выделить, генерируем исключение
throw "out of memory Buffer2"; /*генерация исключения
        с параметром-строкой это исключение обрабатывается
        первым блоком catch*/

for (i=0; i<1000; i++)
if (Buffer2[i]==0)
throw i; /*это исключение обрабатывается вторым блоком catch
Buffer1[i]=Buffer1[i]/Buffer2[i];
....
{
        //обработка исключений
catch (char *ErrorMsg) /*определение функции обработки ошибки,
        параметром которой является любая строка char *ErrorMsg*/
{
    cout << ErrorMsg << '\n'; /*вывод сообщения об ошибке (обработка
        ошибки) и вызов диалогового окна для возобновления выполнения
        программы или вызов функции exit для остановки работы программы*/
}
catch (int ErrorCode) //обработка любого исключения типа int
{ cout<<"Арифметическая ошибка при i = "<<ErrorCode<<endl;
    /*вывод сообщения об ошибке (обработка ошибки) и вызов
        диалогового окна для возобновления выполнения программы
        или вызов функции exit для остановки работы программы*/
}
}
}

```

Если объявление блока catch содержит многоточие, то он получает управление в ответ на исключение любого типа, сгенерированное предыдущими блоками try.

```

catch (...)
{
    //получает управление в ответ на исключения любого типа
}

```

Так как данный блок не содержит параметров, он не имеет доступа к значению, передаваемому при вызове исключения. Блок catch с многоточием записывается последним. Программа ищет

блоки `catch` в порядке их следования и активизирует первый подходящий по типу исключения. Так как `catch` с многоточием является последним, то в первую очередь управление получает `catch` с точно совпадающим типом исключения, а не универсальный блок с многоточием.

20.2. Программирование блоков `catch`

При разработке блока `catch` выбирается оптимальный вариант для обработки каждого исключения. На обработку исключения влияет как общий тип исключения, так и информация, передаваемая через параметр блока `catch`. Ниже перечислены три основных способа обработки исключений в блоке `catch`.

Продолжение программы. Если проблема не сложная, то `catch` уведомляет о ней и самостоятельно исправляет ошибочную ситуацию. После передачи управления выполнение программы продолжается с оператора, следующего за последним блоком `catch`. Блок `catch` удаляет программные ресурсы (например, блоки памяти или дескрипторы файлов), которые должен был удалить блок `try`, если бы не было сгенерировано исключение.

Завершение программы. Если проблема достаточно серьезна, обработчик `catch` выполняет требуемую «чистку мусора» (например, закрывает дескрипторы файлов), освобождение захваченной памяти сообщает об ошибке, а затем вызывает библиотечную функцию `exit`., которая выполняет выход из программы.

Отказ от обработки исключения. Блок `catch` не обрабатывает исключение. Для этого используется оператор `throw` без конкретизирующего значения.

Например,

```
catch(...) throw;
```

Этот оператор генерирует исключение с таким же типом и значением, как и первоначальное исключение, переданное `catch`. Поток управления разветвляется, и программа ищет другой обработчик исключения (или вызывает системный обработчик исключений (оператор `terminate`), если он не найден, другой обработчик исключений того же типа и значения). Запомните: оператор `throw` используется

без указания типа только в блоке `catch` или внутри вызываемой им функции.

Рассмотрим несколько примеров.

Пример 1. Написать программу, генерирующую исключение о невозможности выделения памяти под массив, обрабатывающую его и задающую 0-й элемент массива заданным способом.

```
# include <iostream>           //подключение потоковой библиотеки
                                //ввода-вывода
using namespace std;           //указание, что используется
                                //пространство имен std

int main()
{
    char *Buffer = new char [100];
        //попытка выделить память под массив Buffer из 100 символов
    try
    {
        //начало контролируемого (охраняемого) блока
        if (Buffer == 0)           //если памяти выделить не удалось
            throw "Out of memory";
        //генерируем исключение со строкой «Out of memory»
    }
    else
        throw 1;           //иначе генерируем исключение с целым числом 1
    catch (char *ErrorMsg)
        //метод catch обработки исключений с параметром-строкой
    {
        cout << ErrorMsg << endl;           //вывод сообщения об ошибке
    }
    catch (int a)
        //метод catch обработки исключений с целым параметром a
    {
        cout << a << " - Vse - Ok" << endl;
        //вывод значения a и сообщения о том, что все в порядке
    }
    Buffer[0] = '2';
        //присвоить 0-му элементу массива Buffer значение символа '2'
    return 0;
}
```

Пример 2. Написать функцию, вычисляющую наибольший общий делитель (НОД) для двух чисел, используя алгоритм Евклида. В программе предусмотреть обработку различных исключительных ситуаций.


```

#include <iostream>
//подключение потоковой библиотеки ввода-вывода
using namespace std;
//указание, что используется пространство имен std
int nod(int x, int y) /*заголовок функции нахождения наибольшего
                     общего делителя целых значений x и y*/
{
    try
    {
        //начало контролируемого (охраняемого) блока

        if (x == 0 || y == 0) throw "zero!";
        //если либо x=0 либо y=0, генерируем исключение со строкой «zero»
        if (x < 0) throw "x - negativ!";
        //если x<0, генерируем исключение со строкой "x- negativ"
        if (y < 0) throw "y - negativ!";
        //если y<0, генерируем исключение со строкой "y- negativ"

        while(x != y) //в цикле, пока x!=y
        {
            if (x < y) //если x<y
                y = y - x; //считаем новое значение y
            else
                x = x - y; //иначе считаем новое значение x
        }
        return x;
        //возврат вычисленного значения x в вызывающую функцию
    }
    catch (char *error) //обработчик ошибки с параметром-строкой
    {
        cout << "Error: " << error << " x = " << x << ",
        y = " << y << endl; //вывод сообщения об ошибке и значений x и y
        return 0; //возврат в вызывающую функцию
    }
}

int main()
{
    cout << "NOD(5, 10) = " << nod(5, 10) << endl;
    //попытка вывести наибольший общий делитель чисел 5, 10
    cout << "NOD(0, 4) = " << nod(0, 4) << endl;
    //попытка вывести наибольший общий делитель чисел 0, 4
    cout << "NOD(-4, 8) = " << nod(-4, 8) << endl;
    //попытка вывести наибольший общий делитель чисел -4, 8
    cout << "NOD(3, -9) = " << nod(3, -9) << endl;
    //попытка вывести наибольший общий делитель чисел 3, -9
    return 0;
}

```

Результат работы программы:

```
NOD(5, 10) = 5
Error: zero! x = 0, y = 4
NOD(0, 4) = 0
Error: x - negativ! x = -4, y = 8
NOD(-4, 8) = 0
Error: y - negativ! x = 3, y = -9
NOD(3, -9) = 0
Для продолжения нажмите любую клавишу . . .
```

Пояснение. Эта программа демонстрирует особенности механизма особых ситуаций (исключений), но в ней нет никаких преимуществ перед стандартными средствами анализа данных и возврата из функций. В этой программе генерация исключений и их обработка происходят в одной функции. Такое использование механизма обработки исключительных ситуаций не типично.

Наиболее правильный подход: функция только генерирует исключение, а его обработку предоставляет другой функции, использующей данную. Рассмотрим другую версию программы.

```
#include <iostream>           //подключение потоковой библиотеки
                               //ввода-вывода
using namespace std;          //указание, что используется
                               //пространство имен std
int nod(int x, int y) /*заголовок функции нахождения наибольшего
                       общего делителя целых значений x и y*/
{
    if (x == 0 || y == 0) throw "zero!";
    //если либо x=0 либо y=0, генерируем исключение со строкой «zero»
    if (x < 0) throw "x - negativ!";
    //если x<0, генерируем исключение со строкой «x- negativ»
    if (y < 0) throw "y - negativ!";
    //если y<0, генерируем исключение со строкой «y- negativ»

    while(x != y)              //в цикле, пока x!=y
    {
        if (x < y)              //если x<y
            y = y - x;          //считаем новое значение y
        else
            x = x - y;          //иначе считаем новое значение x
    }
    return x;
    //возврат вычисленного значения x в вызывающую функцию
}
```

```

int main()
{
    try
    {
        //начало контролируемого (охраняемого) блока
        cout << "NOD(5, 10) = " << nod(5, 10) << endl;
        //попытка вывести наибольший общий делитель чисел 5, 10
        cout << "NOD(0, 4) = " << nod(0, 4) << endl;
        //попытка вывести наибольший общий делитель чисел 0, 4
        cout << "NOD(-4, 8) = " << nod(-4, 8) << endl;
        //попытка вывести наибольший общий делитель чисел -4, 8
        cout << "NOD(3, -9) = " << nod(3, -9) << endl;
        //попытка вывести наибольший общий делитель чисел 3, -9
    }
    catch (char *error) //обработчик ошибки с параметром-строкой
    {
        cout << "Error: " << error << endl;
        //вывод сообщения об ошибке
    }
    return 0;
}

```

Результат работы программы:

```

NOD(5, 10) = 5
Error: zero!
Прервать? Продолжить?

```

Пояснение. В программе генерируются исключения типа `*char`. Из результатов программы непонятно, в каком месте и из-за чего произошла ошибка. Стандартный обработчик `terminate` предлагает только два действия: прервать или продолжить, и тоже непонятно, что делать. Для того чтобы исключение стало более информативно (или для передачи большей информации о возникшей ситуации), оно должно быть объектом класса, который создается специально и содержит всю информацию о причине ошибки и о значении нужных переменных.

Тогда программа примет вид:

```

#include <iostream>
#include <string>
using namespace std;
class DATA //начало определения класса DATA
{

```



```

public:
    int x, y;
    //объявление общедоступных целых членов-переменных x, y класса DATA
    char error[30];
    //объявление общедоступной строки члена-переменной класса DATA
    DATA(int x, int y, char *str)
        //определение конструктора класса DATA
    {
        this->x = x;
        //присваивание 1-го параметра члену-переменной x
        this->y = y;
        //присваивание 2-го параметра члену-переменной y
        strcpy(error, str);
        //копирование параметра-строки в строку член-переменную класса DATA
    }
};
//конец определения класса DATA

int nod(int x, int y)
{
    if (x == 0 || y == 0) throw DATA(x, y, "zero!");
    /*если x =0 или y=0, вызываем исключение с безымянным объектом
    класса DATA и параметрами x, y и строкой «zero»*/
    if (x < 0) throw DATA(x, y, "x - negativ!");
    /*если x =0 или y=0, вызываем исключение с безымянным объектом
    класса DATA и параметрами x, y и строкой «x-negativ»*/
    if (y < 0) throw DATA(x, y, "y - negativ!");
    /*если x =0 или y=0, вызываем исключение с безымянным объектом класса
    DATA и параметрами x, y и строкой «y- negativ»*/

    while(x != y)
    {
        if (x < y)
            y = y - x;
        else
            x = x - y;
    }
    return x;
    //возврат вычисленного значения x в вызывающую функцию
}

int main()
{
    try
    {
        //начало контролируемого (охраняемого) блока
        cout << "NOD(5, 10) = " << nod(5, 10) << endl;
        //попытка вывести наибольший общий делитель чисел 5, 10
    }
}

```

```

    cout << "NOD(0, 4) = " << nod(0, 4) << endl;
        //попытка вывести наибольший общий делитель чисел 0, 4
    cout << "NOD(-4, 8) = " << nod(-4, 8) << endl;
        //попытка вывести наибольший общий делитель чисел -4, 8
    cout << "NOD(3, -9) = " << nod(3, -9) << endl;
        //попытка вывести наибольший общий делитель чисел 3, -9
}

catch (DATA d)
    //обработчик ошибки с параметром-объектом типа DATA
{
    cout << "Error: " << d.error << " x = " << d.x << " ,
    y = " << d.y << endl;
        /*вывод сообщения об ошибке
        членов-переменных — строки, x и y объекта класса DATA*/
    }
    return 0;
}

```

Пример 3. Написать функцию деления двух чисел. Предусмотреть обработку возникающих исключительных ситуаций.

```

#include <iostream>
using namespace std;
class ZeroDivide {};    /*определение пустого класса ZeroDivide,
                        который не имеет ни членов-переменных, ни членов-функций */
class OverFlow {};    /*определение пустого класса OverFlow, который
                        не имеет ни членов-переменных, ни членов-функций*/
float divide(float n, float d)    //определение функции divide
{
    if (d == 0.0)                //если знаменатель равен 0
        throw ZeroDivide();
        //вызываем исключение с безымянным объектом класса ZeroDivide
    float r = n / d;
    if (r > 1e+30)                //если частное больше 1e+30
        throw OverFlow();
        //вызываем исключение с безымянным объектом класса ZeroDivide

    return r;    //возврат вычисленного частного в вызывающую функцию
}
float R;    //объявление глобальной вещественной переменной R
void RR(float a, float b)
    //определение функции RR с двумя параметрами
{
    try
    {
        //начало контролируемого (охраняемого) блока
    }
}

```

```

R = divide(a, b);
           //вызов функции divide с параметрами функции RR
}
catch (ZeroDivide)
           //обработчик ошибки с параметром — типом ZeroDivide

{
    cout << "Деление на ноль!" << endl;
}
catch (OverFlow)
           //обработчик ошибки с параметром — типом OverFlow

{
    cout << "Значение превышает допустимый
           диапазон!" << endl;
}
}
int main()
{
    cout << "4.4 / 0 = ";
    RR(4.4, 0.0);
    cout << "1e+20 / 1e-15 = ";
    RR(1e+20, 1e-15);
    cout << "10.0 / 3.0 = ";
    RR(10.0, 3.0);
    cout << R << endl;
    return 0;
}

```

Результат выполнения программы:

```

4.4/0= Деление на ноль
1e+20/1e-15= Значение превышает допустимый диапазон!
10.0/3.0=3.33333

```

Пример 4. Написать программу, генерирующую исключения различных типов в зависимости от введенного символа.

```

#include <iostream>
using namespace std;
class CExept           //начало определения класса CExept
{
private:
    int ExCode;
           //определение закрытой члена-переменной ExCode класса CExept

```



```

public:
    CExept(int k)           //определение конструктора класса CExept
    {
        ExCode = k;
        //присвоить члену-переменной класса CExept значение параметра
    }
};                          //конец определения класса CExept

int main()
{
    try
    {
        //начало контролируемого (охраняемого) блока
        char sim;
        cout << "char exeption?";
        cin >> sim;
        if (sim == 'y')
            throw 'd';      //генерируем исключение символьного типа
        cout << "int exeption?";
        cin >> sim;
        if (sim == 'y')
            throw 1;        //генерируем исключение целого типа
        cout << "float exeption?";
        cin >> sim;
        if (sim == 'y')
            throw float(2.5);
        //генерируем исключение вещественного типа
        cout << "CExept exeption?";
        cin >> sim;
        if (sim == 'y')
            throw CExept(10);
        cout << "Double exeption?";
        cin >> sim;
        if (sim == 'y')
            throw double(3.14); //генерируем исключение типа double
    }

    //блок обработки исключений

    catch (char s)
    {
        cout << "Char exeption!" << endl;
    }
    catch (int a)
    {
        cout << "Int exeption!" << endl;
    }
}

```

```
    catch (float x)
    {
        cout << "Float exeption!" << endl;
    }
    catch (CExept ce)
    {
        cout << "CExept exeption!" << endl;
    }
    catch (...)
    {
        cout << "Unknown exeption!" << endl;
    }
    return 0;
}
```

Результат выполнения программы:

```
Char exeption?y
Char exeption!
Int exeption?y
Int exeption!
float exeption?y
float exeption!
CExept exeption?y
CExept exeption!
Double exeption?y
CExept exeption!
Unknown exeption!
```

Задания

1. Напишите программу, в которой выделяется память для хранения большого массива вещественных переменных. Предусмотрите в программе обработку исключительной ситуации (нехватку памяти).
2. Напишите программу, генерирующую и обрабатывающую исключения различных типов.
3. Напишите функцию, вычисляющую остаток от деления двух целых чисел. Предусмотрите обработку возможных исключительных ситуаций.
4. Напишите функцию, возвращающую результат деления переданных ей параметров. Предусмотрите обработку возможных исключительных ситуаций (деление на ноль, выход за пределы диапазонов).

5. Напишите программу, вычисляющую корни квадратного уравнения. Предусмотрите обработку различных исключительных ситуаций (деление на ноль, вычисление корня из отрицательного числа).

Контрольные вопросы

1. Что такое исключительная ситуация (исключение)?
2. Какая конструкция в языке C++ предназначена для обработки исключений?
3. Для чего необходимо выполнять обработку исключительных ситуаций?
4. Какие исключительные ситуации могут возникнуть в программе?
5. Приведите пример генерации исключительной ситуации.
6. Исключения каких типов можно сгенерировать в языке C++? Приведите примеры.

Список используемой литературы

1. Брюс Эккель. Философия C++. Введение в стандартный C++. Thinking in C++. Introduction to Standard C++. СПб.: Питер, 2004.
2. Бьярн Страуструп. Программирование. Принципы и практика использования C++. Programming: Principles and Practice Using C++. М.: Вильямс, 2011.
3. Джесс Либерти, Брэдли Джонс. Освой самостоятельно C++ за 21 день. М.: Вильямс, 2007.
4. Майкл Дж. Янг Visual C++6: Полн. рук. : в 2 т. / пер. с англ.; под ред. В.И. Пустоварова, Б.Г. Жадаева. М.: Ирина Спарк, 1999.
5. Подбельский В. В., Фомин С. С. Программирование на языке Си. М.: Финансы и статистика, 2004.
6. Подбельский В.В. Язык СИ++. 5-е изд. М.: Финансы и статистика, 2000.
7. Стенли Б. Липпман, Жози Лажойе, Барбара Му. Язык программирования C++. Вводный курс C++. Primer. М.: Вильямс, 2007.
8. Стенли Б. Липпман, Жози Лажойе. Язык программирования C++. Вводный курс издательства. М.: Невский Диалект, ДМК Пресс, 2001.

Таблица ASCII символов

Таблица П1.1. Управляющие символы

Обозначение символа	ASCII-код	Описание
NUL	0	Null. Пустой
SOH	1	Start Of Heading. Начало заголовка
STX	2	Start of Text. Начало текста
ETX	3	End of Text. Конец текста
EOT	4	End of Transmission. Конец передачи
ENQ	5	Enquire. Запрос
ACK	6	Acknowledgement. Подтверждение
BEL	7	Bell. Звонок, звуковой сигнал
BS	8	Backspace. Возврат на один символ
TAB	9	Tabulation. Обозначался также HT — Horizontal Tabulation. Горизонтальная табуляция
LF	10	Line Feed. Перевод строки
VT	11	Vertical Tab. Вертикальная табуляция
FF	12	Form Feed. Новая страница
CR	13	Carriage Return. Возврат каретки
SO	14	Shift Out. Выключить сдвиг
SI	15	Shift In. Включить сдвиг
DLE	16	Data Link Escape
DC1	17	Device Control 1. Управление устройством 1
DC2	18	Device Control 2. Управление устройством 2
DC3	19	Device Control 3. Управление устройством 3
DC4	20	Device Control 4. Управление устройством 4
NAK	21	Negative Acknowledgment. Отсутствие подтверждения (неподтверждение)
SYN	22	Synchronization. Синхронизация
ETB	23	End of Text Block. Конец текстового блока
CAN	24	Cancel. Отмена

Окончание табл. П1.1

Обозначение символа	ASCII-код	Описание
EM	25	End of Medium
SUB	26	Substitute. Подставить
ESC	27	Escape. Отмена. Следующие символы — что-то специальное
FS	28	File Separator. Разделитель файлов
GS	29	Group Separator. Разделитель групп
RS	30	Record Separator. Разделитель записей
US	31	Unit Separator. Разделитель юнитов

Таблица П1.2. Символы с кодами 32—127

Символ	ASCII-код	Символ	ASCII-код
Пробел	32	P	80
!	33	Q	81
"	34	R	82
#	35	S	83
\$	36	T	84
%	37	U	85
&	38	V	86
'	39	W	87
(40	X	88
)	41	Y	89
*	42	Z	90
+	43	[91
,	44	\	92
-	45]	93
.	46	^	94
/	47	_	95
0	48	`	96
1	49	a	97
2	50	b	98

Окончание табл. П1.2

Символ	ASCII-код	Символ	ASCII-код
3	51	c	99
4	52	d	100
5	53	e	101
6	54	f	102
7	55	g	103
8	56	h	104
9	57	i	105
:	58	j	106
;	59	k	107
<	60	l	108
=	61	m	109
>	62	n	110
?	63	o	111
@	64	p	112
A	65	q	113
B	66	r	114
C	67	s	115
D	68	t	116
E	69	u	117
F	70	v	118
G	71	w	119
H	72	x	120
I	73	y	121
J	74	z	122
K	75	{	123
L	76		124
M	77	}	125
N	78	~	126
O	79	␣	127

Таблица П1.3. Символы с кодами 128—255

Символ	ASCII-код	Символ	ASCII-код
А	128	Т	192
Б	129	⊥	193
В	130	т	194
Г	131	┌	195
Д	132	—	196
Е	133	⊕	197
Ж	134	⌞	198
З	135	⌏	199
И	136	ℓ	200
Й	137	℞	201
К	138	⌞	202
Л	139	⌞	203
М	140	⌏	204
Н	141	=	205
О	142	⌏	206
П	143	⌞	207
Р	144	⌞	208
С	145	⌞	209
Т	146	π	210
У	147	ℓ	211
Ф	148	ℓ	212
Х	149	℞	213
Ц	150	π	214
Ч	151	⌏	215
Ш	152	⌏	216
Щ	153	┘	217
Ъ	154	г	218
Ы	155	■	219
Ь	156	■	220
Э	157	┘	221
Ю	158	┘	222
Я	159	■	223

Окончание табл. П1.3

Символ	АSCII-код	Символ	АSCII-код
а	160	Р	224
б	161	С	225
в	162	Т	226
г	163	У	227
д	164	Ф	228
е	165	Х	229
ж	166	Ц	230
з	167	Ч	231
и	168	Ш	232
й	169	Щ	233
к	170	Ъ	234
л	171	Ы	235
м	172	Ь	236
н	173	Э	237
о	174	Ю	238
п	175	Я	239
▒	176	Ё	240
▓	177	Ё	241
█	178	Є	242
	179	Є	243
┌	180	Ї	244
┐	181	Ї	245
└	182	Ў	246
┘	183	Ў	247
┌	184	°	248
┐	185	·	249
└	186	·	250
┘	187	√	251
┌	188	№	252
┐	189	α	253
└	190	■	254
┘	191		255

Базовые функции из математической библиотеки языка C++ (заголовочный файл *math.h*)

Прототип функции	Краткое описание
int abs(int);	Вычисление абсолютного значения целого аргумента
double acos(double);	Вычисление арккосинуса. Значение аргумента должно находиться в диапазоне от -1 до $+1$
double asin(double);	Вычисление арксинуса. Значение аргумента должно находиться в диапазоне от -1 до $+1$
double atan(double);	Вычисление арктангенса
double atan2(double y, double x);	Вычисление арктангенса отношения первого аргумента функции ко второму
double cabs(struct complex);	Вычисление абсолютного значения комплексного числа. Определение структуры <code>complex</code> находится в файле <code>math.h</code>
double cos(double);	Вычисление косинуса. Значение аргумента должно быть задано в радианах
double cosh(double);	Вычисление гиперболического косинуса
double exp(double);	Вычисление значения экспоненциальной функции
double fabs(double);	Вычисление абсолютного значения вещественного аргумента двойной точности
double floor(double);	Нахождение наибольшего целого, не превышающего значение аргумента функции. Результат возвращается в форме <code>double</code>
double fmod(double, double);	Вычисление остатка от деления первого аргумента функции на второй. Результат возвращается в форме <code>double</code>
double hypot(double, double);	Вычисление гипотенузы прямоугольного треугольника по значениям катетов

Окончание прил. 2

Прототип функции	Краткое описание
double labs(long);	Вычисление абсолютного значения целого аргумента типа <code>long</code> . Результат возвращается в форме <code>double</code>
double log(double);	Вычисление значения натурального логарифма
double log10(double x);	Вычисление значения десятичного логарифма
double poly(double x, int n , double c[]);	Вычисление значения полинома: $C[n] \cdot x^n + C[n-1] x^{(n-1)} + \dots + C[1] \cdot x + C[0]$
double pow(double x, double y);	Вычисление значения x в степени y : x^y
double pow10(int p);	Вычисление значения 10^p
double sin (double);	Вычисление синуса. Значение аргумента должно быть задано в радианах
double sinh(double);	Вычисление гиперболического синуса
double sqrt(double);	Вычисление квадратного корня
double tan(double);	Вычисление тангенса. Значение аргумента должно быть задано в радианах
double tanh(double);	Вычисление гиперболического тангенса

Классы и функции потокового ввода/вывода

Библиотека потоковых классов языка C++ построена на основе базового класса `ios`. Иерархия классов представлена на рис. ПЗ.1.

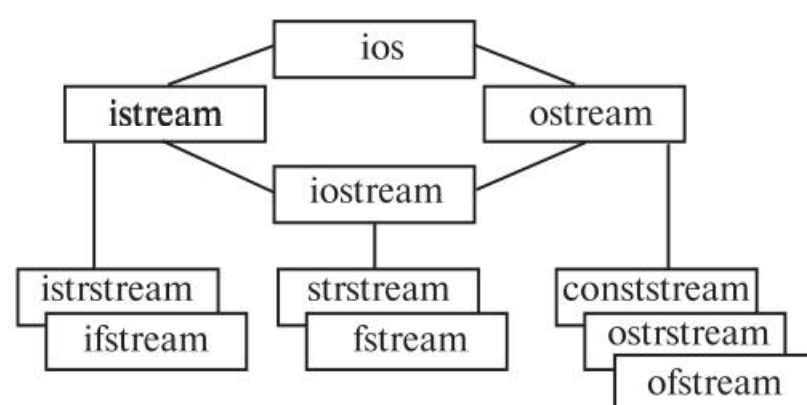


Рис. ПЗ.1. Упрощенная схема иерархии потоковых классов

Таблица ПЗ.1. Список доступных классов потокового ввода/вывода

Имя класса	Краткое описание
<code>ios</code>	Базовый потоковый класс
<code>istream</code>	Потоки ввода
<code>ostream</code>	Потоки вывода
<code>iostream</code>	Двунаправленные потоки
<code>istrstream</code>	Строковые потоки ввода
<code>ostrstream</code>	Строковые потоки вывода
<code>strstream</code>	Двунаправленные строковые потоки
<code>ifstream</code>	Файловые потоки ввода
<code>ofstream</code>	Файловые потоки вывода
<code>fstream</code>	Двунаправленные файловые потоки

Таблица ПЗ.2. Список заранее определенных объектов-потоков

Имя объекта	Краткое описание
<code>istream cin;</code>	Стандартный поток ввода с клавиатуры
<code>ostream cout;</code>	Стандартный поток вывода на экран

Окончание табл. ПЗ.2

Имя объекта	Краткое описание
<code>ostream cerr;</code>	Стандартный поток вывода сообщений об ошибках на экран
<code>ostream clog;</code>	Стандартный буферизованный поток вывода сообщений об ошибках на экран

Таблица ПЗ.3. Список заголовочных файлов, необходимых для использования классов потокового ввода/вывода

Имя файла	Используемые классы
<code>iostream.h</code>	<code>ios, ostream, istream</code>
<code>strstream.h</code>	<code>strstream, istrstream, ostrstream</code>
<code>fstream.h</code>	<code>fstream, ifstream, ofstream</code>

Таблица ПЗ.4. Список базовых функций класса `istream` и `ostream`

Имя функции	Краткое описание
<code>istream &get(char *buffer, int size, char delimiter = '\n');</code>	Функция извлечения символов из потока <code>istream</code> и помещения их в буфер. Операция прекращается при достижении конца файла, либо когда скопировано указанное число <code>size</code> символов, либо при обнаружении указанного разделителя. Последовательность прочитанных символов завершается нулевым символом (<code>'\0'</code>)
<code>istream &read(char *buffer, int size);</code>	Функция чтения символов из потока. В отличие от функции <code>get</code> не поддерживает разделителей, и считанные в буфер символы не завершаются нулевым символом
<code>istream &getline(char *buffer, int size, char delimiter = '\n');</code>	Функция для извлечения строк из потока. Считывается либо <code>size-1</code> символ, либо последовательность символов до символа-ограничителя. Считанные символы завершаются нулевым символом
<code>istream get(char &C);</code>	Функция чтения символа из потока <code>istream</code> в символ <code>C</code> . В случае ошибки <code>C</code> принимает значение <code>0xFF</code>
<code>int get(void);</code>	Функция извлечения из потока <code>istream</code> очередного символа. При обнаружении конца файла возвращает <code>EOF</code>

Окончание табл. ПЗ.4

Имя функции	Краткое описание
<code>int peek(void);</code>	Функция определения очередного символа в потоке <code>istream</code> . Символ не извлекается из потока <code>istream</code>
<code>int gcount(void);</code>	Функция возвращает количество символов, считанных во время последней операции неформатированного ввода
<code>istream &ignore(int count = 1, int target = EOF);</code>	Функция извлечения символов из потока <code>istream</code> до тех пор, пока не произойдет одно из следующих событий: функция не извлечет <code>count</code> символов; не будет обнаружен символ <code>target</code> ; не будет достигнут конец файла
<code>ostream &put(char C);</code>	Функция помещает символ <code>C</code> в поток <code>ostream</code>
<code>ostream &write(const char *buffer, int size);</code>	Функция записи в поток <code>ostream</code> содержимого буфера. Символы копируются до тех пор, пока не возникнет ошибка или не будет скопировано указанное число символов. Буфер записывается без форматирования
<code>istream &seekg(long p);</code>	Функция устанавливает указатель потока <code>get</code> со смещением <code>p</code> от начала потока
<code>istream &seekg(long p, seek_dir point);</code>	Функция указания начальной точки перемещения в потоке. Смещение может принимать одно из значений: <code>enum seek_dir { beg, curr, end }</code> Положительное значение <code>p</code> перемещает указатель <code>get</code> вперед (к концу потока), Отрицательное значение <code>p</code> перемещает указатель <code>get</code> назад (к началу потока)
<code>long tellg(void);</code>	Функция возвращает текущее положение указателя <code>get</code>

Оглавление

Предисловие	3
Глава 1. ОСНОВНЫЕ СВЕДЕНИЯ О ПРОГРАММИРОВАНИИ	5
1.1. Основные этапы решения задач	5
1.2. Схемы алгоритмов	7
1.3. Рекомендации по стилю программирования	10
Глава 2. ОСНОВЫ РАБОТЫ В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКИ MICROSOFT VISUAL STUDIO 2010	13
2.1. Назначение и состав среды программирования Microsoft Visual Studio 2010	13
2.2. Процесс создания и выполнения программы в среде программирования Microsoft Visual Studio 2010	14
2.3. Запуск интегрированной среды разработки Microsoft Visual Studio 2010	15
2.4. Работа с проектами в среде Microsoft Visual Studio 2010	16
2.5. Компиляция, компоновка и выполнение проекта	22
2.6. Отладка проекта	25
Глава 3. ВВЕДЕНИЕ В ЯЗЫК C++	29
3.1. Алфавит, лексемы, разделители	29
3.2. Ключевые слова. Идентификаторы	32
3.3. Константы и переменные	33
3.4. Понятие типа данных	34
3.5. Целые типы данных	34
3.6. Вещественные типы данных	36
3.7. Логический тип данных	37

3.8. Операторы описания и определения переменных	38
3.9. Преобразование типов	39
3.10. Знаки операций	40
3.11. Оператор присваивания. Арифметические выражения. Приоритет операций	43
3.12. Структура программы на языке C++	45
3.13. Форматированный ввод и вывод данных	47
3.14. Особенности ввода и вывода символов и строк	50
 Глава 4. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЕННЫХ АЛГОРИТМОВ	 57
4.1. Условный оператор	57
4.2. Условная операция	66
4.3. Оператор выбора	68
 Глава 5. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ	 81
5.1. Оператор цикла с предусловием <i>while</i>	81
5.2. Оператор цикла с постусловием <i>do-while</i>	84
5.3. Оператор цикла <i>for</i>	86
5.4. Решение задач с использованием операторов цикла	91
 Глава 6. ПРЕПРОЦЕССОРНЫЕ СРЕДСТВА	 120
6.1. Директива <i>#include</i>	120
6.2. Директива <i>#define</i>	121
6.3. Директивы условной компиляции*	123
6.4. Макроподстановки с параметрами	130
6.5. Прагмы	132
 Глава 7. ПАМЯТЬ. АДРЕСА. УКАЗАТЕЛИ	 137
7.1. Организация памяти. Хранение переменных в памяти	137
7.2. Указатели. Объявление. Инициализация	137
7.3. Операции взятия адреса и разыменования	138
7.4. Арифметические операции с указателями	139
7.5. Выделение динамической памяти. Операторы <i>new</i> и <i>delete</i> ..	141
7.6. Ссылки	144

Глава 8. ОДНОМЕРНЫЕ МАССИВЫ	151
8.1. Понятие одномерного массива	151
8.2. Работа с одномерными массивами	153
8.3. Поиск максимального (минимального) элемента в массиве и определение его индекса	158
8.4. Решение задач с использованием одномерных массивов	162
8.5. Динамические одномерные массивы	169
8.6. Массивы указателей	174
Глава 9. РАБОТА СО СТРОКАМИ	184
9.1. Функции работы со строками	185
9.2. Работа со строками как с массивом символов	189
9.3. Стандартные функции обработки строк	191
Глава 10. ДВУМЕРНЫЕ МАССИВЫ (МАТРИЦЫ)	198
10.1. Понятие матрицы (двумерного массива)	198
10.2. Формирование матриц и вывод их на экран	199
10.3. Работа с матрицами	203
10.4. Поиск максимального (минимального) элемента матрицы и определение его координат (индексов)	211
10.5. Формирование одномерных массивов из элементов матриц	214
10.6. Представление двумерного массива (только для И-32,33)	219
10.7. Представление статического двумерного массива (только для И-32,33)	222
10.8. Динамические двумерные массивы (только для И-32,33)	223
Глава 11. ПОНЯТИЕ ФУНКЦИИ	235
11.1. Функции	235
11.2. Определение функции. Объявление (прототип) функции	236
11.3. Формальные и фактические параметры	238
11.4. Передача по значению	240
11.5. Передача по ссылке	244
11.6. Передача по указателю	246
11.7. Локальные и глобальные переменные	247

11.8. Использование одномерных массивов в качестве параметров	250
11.9. Использование двумерных массивов в качестве параметров	255
Глава 12. ФУНКЦИИ	273
12.1. Локальные и глобальные переменные	273
12.2. Видимость переменных	274
12.3. Время жизни переменной	275
12.4. Модификаторы переменных	276
12.5. Функции с переменным числом параметров	278
12.6. Рекурсивные функции	279
12.7. Перегрузка функций	280
Глава 13. ТИПЫ ДАННЫХ, ВВОДИМЫЕ ПОЛЬЗОВАТЕЛЕМ ...	285
13.1. Переименование типов (typedef)	285
13.2. Перечисления (enum)	288
13.3. Структуры (struct)	291
13.4. Объединения (union)	299
Глава 14. РАБОТА СО СТРУКТУРАМИ	304
14.1. Решение задач с использованием переменных комбинированного типа	304
14.2. Динамические массивы структур (только для И-32,33)	308
14.3. Динамические структуры данных (списки)	310
14.4. Формирование списков	312
14.5. Решение задач (работа со списками)	317
Глава 15. КЛАССЫ	334
15.1. Объектно-ориентированный подход к программированию. Инкапсуляция	334
15.2. Класс как тип данных	335
15.3. Создание объектов (экземпляров) класса	337
15.4. Доступ к членам класса	338
15.5. Конструкторы	341
15.6. Применение конструкторов копирования и преобразования	344

15.7. Конструкторы копирования	346
15.8. Деструкторы	347
15.9. Когда вызываются конструкторы и деструкторы	349
15.10. Передача объектов класса в функции	350
15.11. Класс CVector	353
15.12. Класс CMatrix	355
15.13. Примеры разработки класса	357
15.14. Классы в качестве полей других классов	359
 Глава 16. ОСОБЕННОСТИ КЛАССОВ	367
16.1. Константы в качестве полей класса	367
16.2. Статические члены класса	369
16.3. Статические функции-члены класса	371
16.4. Указатель <i>this</i>	372
16.5. Дружественные функции	375
16.6. Дружественные классы	377
16.7. Перегрузка стандартных операций	380
 Глава 17. НАСЛЕДОВАНИЕ, ПОЛИМОРФИЗМ	393
17.1. Наследование	393
17.2. Конструкторы производного и базового классов	399
17.3. Иерархия классов	401
17.4. Вызов конструкторов и деструкторов	404
17.5. Множественное наследование	406
17.6. Виртуальные функции	412
17.7. Виртуальные базовые классы	416
17.8. Абстрактные классы	419
17.9. Классы и шаблоны	421
 Глава 18. ПОТОКОВЫЙ ВВОД/ВЫВОД	434
18.1. Пространство имен	434
18.2. Понятие потока	437
18.3. Классы потоков. Иерархия классов потоков	439
18.4. Класс консольных потоков. Объекты <i>cin</i> и <i>cout</i>	440
18.5. Ввод/вывод стандартных типов данных	441
18.6. Ввод/вывод пользовательских типов данных	445

Глава 19. РАБОТА С ФАЙЛАМИ	451
19.1. Поточковый ввод/вывод файлов	451
19.2. Работа с текстовыми файлами. Запись/чтение стандартных типов данных. Запись/чтение пользовательских типов данных	452
19.3. Примеры программ работы с файлами	454
19.4. Признак конца файла	459
19.5. Чтение и запись в файл стандартных типов данных	461
19.6. Чтение и запись в файл пользовательских типов данных ..	463
19.7. Произвольный доступ к элементам файлов	465
19.8. Работа с двоичными файлами	468
19.9. Запись стандартных типов данных в двоичные файлы	468
19.10. Чтение стандартных типов данных из двоичных файлов	470
19.11. Запись и чтение пользовательских типов данных в двоичные файлы	471
19.12. Разработка собственных классов для работы с файлами ...	473
 Глава 20. ОБРАБОТКА ИСКЛЮЧЕНИЙ	479
20.1. Конструкция try — catch	479
20.2. Программирование блоков catch	484
 Список используемой литературы	495
 Приложение 1. Таблица ASCII символов	496
 Приложение 2. Базовые функции из математической библиотеки языка C++ (заголовочный файл <i>math.h</i>)	501
 Приложение 3. Классы и функции потокового ввода/вывода	503

По вопросам приобретения книг обращайтесь:
Отдел продаж «ИНФРА-М» (оптовая продажа):

127214, Москва, ул. Полярная, д. 31В, стр. 1

Тел. (495) 280-33-86 (доб. 218, 222)

E-mail: bookware@infra-m.ru

•

Отдел «Книга—почтой»:

тел. (495) 280-33-86 (доб. 222)

ФЗ № 436-ФЗ	Издание не подлежит маркировке в соответствии с п. 1 ч. 4 ст. 11
----------------	---

Учебное издание

**Немцова Тамара Игоревна,
Голова Светлана Юрьевна,
Терентьев Алексей Игоревич**

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C++

УЧЕБНОЕ ПОСОБИЕ

Под редакцией *Л.Г. Гагариной*

ООО «Издательский дом ФОРУМ»
127247, Москва, ул. Софьи Ковалевской, д. 1, стр. 51
E-mail: forum2-book@yandex.ru
Тел.: (495) 280-15-96

ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29
E-mail: books@infra-m.ru <http://www.infra-m.ru>

Подписано в печать 31.10.2022.
Формат 60×90/16. Бумага офсетная. Гарнитура *Times*.
Печать цифровая. Усл. печ. л. 32,0.
ППТ30. Заказ № 00000
ТК 165400-1916204-090615

Отпечатано в типографии ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29