В серии: Библиотека ALT Linux

# Программирование на языке C++ в среде Qt Creator

Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк,
О. В. Чеснокова, А. С. Чмыхало

Москва
ALT Linux
2014

УДК 004.43
ББК 32.973.26-018.1
А47

А47 Программирование на языке C++ в среде Qt Creator: / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк,О. В. Чеснокова, А. С. Чмыхало — М. : ALT Linux, 2014. — 236 с. : ил. — (Библиотека ALT Linux).

ISBN 978-5-905167-16-4

Книга является учебником по алгоритмизации и программированию на C++ и пособием по разработке визуальных приложений в среде QT Creator. Также в книге описаны среда программирования Qt Creator, редактор Geany, кроссплатформенная билиотека построения графиков MathGL. При чтении книги не требуется предварительного знакомства с программированием.

Издание предназначено для студентов, аспирантов и преподавателей вузов, а также для всех, кто изучает программирование на C++ и осваивает кроссплатформенный инструментарий Qt для разработки программного обеспечения.

Сайт книги: http://www.altlinux.org/Books:Qt-C++

Книга адресована студентам, преподавателям инженерных и математических специальностей, а также научным работникам.

**УДК 004.43**
**ББК 32.973.26-018.1**

**По вопросам приобретения обращаться: ООО «Альт Линукс»**
**(495)662-38-83 E-mail: sales@altlinux.ru** http://altlinux.ru

Материалы, составляющие данную книгу, распространяются на условиях лицензии GNU FDL. Книга содержит следующий текст, помещаемый на первую страницу обложки: «В серии “Библиотека ALT Linux”». Название: «Программирование на языке C++ в среде Qt Creator». Книга не содержит неизменяемых разделов. Авторы разделов указаны в заголовках соответствующих разделов. ALT Linux — торговая марка компании ALT Linux. Linux — торговая марка Линуса Торвальдса. Прочие встречающиеся названия могут являться торговыми марками соответствующих владельцев.

**ISBN 978-5-905167-16-4**

© Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк, О. В. Чеснокова, А. С. Чмыхало, 2014
© ALT Linux, 2014

# Оглавление

# Предисловие

Книга, которую открыл читатель, является с одной стороны учебником по алгоритмизации и программированию на C++, а с другой — пособием по разработке визуальных приложений в среде QT Creator. В книге описаны среда программирования Qt Creator и редактор Geany. При чтении книги не требуется предварительного знакомства с программированием.

В первой части книги (главы 1–9) на большом количестве примеров представлены методы построения программ на языке C++, особое внимание уделено построению циклических программ, программированию с использованием функций, массивов, матриц и указателей.

Вторая часть книги (глава 10) посвящена объектно-ориентированному программированию на C++.

В третьей части книги (главы 11–15) читатель научиться создавать кроссплатформенные визуальные приложения с помощью Qt Creator и познакомится с библиотекой классов Qt.

В книге присутствуют задания для самостоятельного решения.

Приложениях описан текстовый редактор Geany, а также кроссплатформенная библиотека MathGL предназначена для построения различных двух- и трёхмерных графиков.

Главы 1–9 написаны Е. Р. Алексеевым и О. В. Чесноковой. Автором раздела по объектно-ориентированному программированию является Д. А. Костюк. Главы 11–15, посвящённые программированию с использованием инструментария Qt, написаны Г. Г. Злобиным и А. С. Чмыхало.

Авторы благодарят компанию ALT Linux (www.altlinux.ru) и лично Алексея Смирнова и Владимира Чёрного за возможность издать очередную книгу по свободному программному обеспечению.

# Глава 1

# Знакомство с языком C++

В этой главе читатель напишет свои первые программы на языке C(C++), познакомится с основными этапами перевода программы с языка C++ в машинный код. Второй параграф главы посвящён знакомству со средой Qt Creator.

## 1.1 Первая программа на C++

Знакомство с языком C++ начнём с написания программ, предназначенных для решения нескольких несложных задач.

**Задача 1.1.** Заданы две стороны прямоугольника a, b. Найти его площадь и периметр.

Как известно, периметр прямоугольника $P = 2 \cdot (a + b)$, а его площадь $S = a \cdot b$. Ниже приведён текст программы.

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     float a,b,s,p;
6     cout<<"a=";
7     cin>>a;
8     cout<<"b=";
9     cin>>b;
10    p=2*(a+b);
11    s=a*b;
12    cout << "Периметр прямоугольника равен " << p <<endl;
13    cout << "Площадь прямоугольника равна " << s <<endl;
14    return 0;
15 }
```

Давайте построчно рассмотрим текст программы и познакомимся со структурой программы на C++ и с некоторыми операторами языка.

**Строка 1.** Указывает компилятору (а точнее, препроцессору), что надо использовать функции из стандартной библиотеки `iostream`. Библиотека `iostream` нужна для организации ввода с помощью инструкции `cin` и вывода — с помощью `cout`. В программе на языке C++ должны быть подключены все используемые библиотеки.

**Строка 2.** Эта строка обозначает, что при вводе и выводе с помощью `cin` и `cout` будут использоваться стандартные устройства (клавиатура и экран), если эту строку не указывать, то каждый раз при вводе вместо `cin` надо будет писать `std::cin`, а вместо `cout` — `std::cout`.

**Строка 3.** Заголовок главной функции (главная функция имеет имя `main`). В простых программах присутствует только функция `main()`.

**Строка 4.** Любая функция начинается с символа `{`.

**Строка 5.** Описание вещественных (`float`) переменных `a` (длина одной стороны прямоугольника), `b` (длина второй стороны прямоугольника), `s` (площадь прямоугольника), `p` (периметр прямоугольника). Имя переменной[1] состоит из латинских букв, цифр и символа подчёркивания. Имя не может начинаться с цифры. В языке C++ большие и малые буквы различимы. Например, имена `PR_1`, `pr_1`, `Pr_1` и `pR_1` — разные.

**Строка 6.** Вывод строки символов `a=` с помощью `cout`. Программа выведет подсказку пользователю, что необходимо вводить переменную `a`

**Строка 7.** Ввод вещественного числа `a` с помощью `cin`. В это момент программа останавливается и ждёт, пока пользователь введёт значение переменой `a` с клавиатуры.

**Строка 8**. Вывод строки символов `b=` с помощью `cout`.

**Строка 9.** Ввод вещественного числа `b` с помощью `cin`.

**Строка 10.** Оператор присваивания для вычисления периметра прямоугольника (переменная `p`) по формуле $2 \cdot (a + b)$ . В операторе присваивания могут использоваться круглые скобки и знаки операций: + (сложение), - (вычитание), * (умножение), / (деление).

**Строка 11.** Оператор присваивания для вычисления площади прямоугольника.

[1] В литературе равнозначно используются термины «имя переменной» и «идентификатор».

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

**Строка 12.** Вывод строки «Периметр прямоугольника равен » и значения p на экран. Константа endl хранит строку «\n», которая предназначена для перевода курсора в новую строку дисплея[2]. Таким образом строка

cout << "Периметр прямоугольника равен "<< p <<endl;

выводит на экран текст "Периметр прямоугольника равен "[3], значение переменной p, и переводит курсор в новую строку.

**Строка 13.** Вывод строки "Площадь прямоугольника равна ", значения площади прямоугольника s, после чего курсор переводится в новую строку дисплея.

**Строка 14.** Оператор return, который возвращает значение в операционную систему. Об этом подробный разговор предстоит в п. **??** Сейчас следует запомнить, если программа начинается со строки int main(), последним оператором должен быть return 0[4].

**Строка 15.** Любая функция (в том числе и main) заканчивается символом }.

Мы рассмотрели простейшую программу на языке C++, состоящую из операторов ввода данных, операторов присваивания (в которых происходит расчет по формулам) и операторов вывода.

Любая программа на языке C++ представляет собой одну или несколько функций. В любой программе **обязательно** должна быть одна функция main(). С этой функции начинается выполнение программы. Правилом хорошего тона в программировании является разбиение задачи на подзадачи, и в главной функции чаще всего должны быть операторы вызова других функций. Общую структуру любой программы на языке C++ можно записать следующим образом.

```
Директивы препроцессора
Объявление глобальных переменных
Тип_результата  f1 (Список_переменных)
{
Операторы
}
Тип_результата  f2 (Список_переменных)
{
Операторы
}
```

[2]Обращаем внимание читателя, что символ пробел является обычным символом, который ничем не отличается от остальных. Для вывода пробела на экран его надо явно указывать в строке вывода.

[3]С пробелом после слова «равен».

[4]Вообще говоря, вместо нуля может быть любое целое число.

```
...

Тип_результата fn(Список_переменных)
{
Операторы
}
Тип_ результата main(Список_переменных)
{
Операторы
}
```

На первом этапе знакомства с языком мы будем писать программы, состоящие только из функции main, без использования глобальных переменных. Структура самой простой на C(C++) имеет вид.

```
Директивы препроцессора
Тип_ результата main(Список_переменных)
{
Операторы
}
```

Введенная в компьютер программа на языке C++ должна быть переведена в двоичный машинный код (должен быть сформирован исполняемый файл). Для этого существуют специальные программы, называемые трансляторами. Все трансляторы делятся на два класса:

- *интерпретаторы* — трансляторы, которые переводят каждый оператор программы в машинный код, и по мере перевода операторы выполняются процессором;
- *компиляторы* переводят всю программу целиком, и если перевод всей программы прошел без ошибок, то полученный двоичный код можно запускать на выполнение.

Процесс перевода программы в машинный код называется *трансляцией*. Если в качестве транслятора выступает компилятор, то используют термин *компиляция* программы. При переводе программы с языка C++ в машинный код используются именно компиляторы, и поэтому применительно к языку C++ термины «компилятор» и «транслятор» эквивалентны.

Рассмотрим основные этапы обработки компилятором программы на языке C++ и формирования машинного кода.

1. Сначала программа обрабатывается препроцессором[5], который обрабатывает директивы препроцессора, в нашем случае это ди-

[5]Препроцессор — это программа, которая преобразовывает текст директив препроцессора в форму, понятную компилятору. О данных на выходе препроцессора говорят, что они находятся в препроцессированной форме.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

рективы включения заголовочных файлов (файлов с расширением **.h**) — текстовых файлов, в которых содержится описание используемых библиотек. В результате формируется полный текст программы, который поступает на вход компилятора.

2. Компилятор разбирает текст программ на составляющие элементы, проверяет синтаксические ошибки и в случае их отсутствия формирует объектный код (файл с расширением **.o** или .**obj**). Получаемый на этом этапе двоичный код не включает в себя двоичные коды библиотечных функций и функций пользователя.
3. *Компоновщик* подключает к объектному коду программы объектные модули библиотек и других файлов (если программа состоит из нескольких файлов) и генерирует исполняемый код программы (двоичный файл), который уже можно запускать на выполнение. Этот этап называется компоновкой или сборкой программы.

После написания программы ее необходимо ввести в компьютер. В той книге будет рассматриваться работа на языке C++ в среде Qt Creator[6]. Поэтому перед вводом программы в компьютер надо познакомиться со средой программирования.

## 1.2 Среда программирования Qt Creator

Среда программирования Qt Creator (IDE IDE QT Creator) находится в репозитории большинства современных дистрибутивов Linux (ОС Linux Debian, ОС Linux Ubuntu, ОС ROSA Linux, ALT Linux и др.). Установка осуществляется штатными средствами вашей операционной системы (менеджер пакетов Synaptic и др.) из репозитория, достаточно установить пакет qtcreator, необходимые пакеты и библиотеки будут доставлены автоматически. Последнюю версию IDE Qt Creator можно скачать на сайте QtProject (http://qt-project.org/downloads). Скачанный установочный файл имеет расширение **.run**. Для установки приложения, необходимо запустить его на выполнение. Установка проходит в графическом режиме. После запуска програм-

[6]Тексты программ, приведённые в первой части книги (главы 1–9), без серьёзных изменений могут быть откомпилированы с помощью любого современного компилятора с языка C(C++). Авторы протестировали все программы из первой части книги с помощью QT Creator и IDE Geany (с использованием g++ версии 4.8).

мы пользователь увидит на экране окно, подобное представленному на рис. 1.1[7].


Рис. 1.1. Окно Qt Creator

При работе в Qt Creator вы находитесь в одном из режимов:

1. **Welcome** (Начало) — отображает экран приветствия, позволяя быстро загружать недавние сессии или отдельные проекты. Этот режим можно увидеть при запуске Qt Creator без указания ключей командной строки.
2. **Edi**t (Редактор) — позволяет редактировать файлы проекта и исходных кодов. Боковая панель слева предоставляет различные виды для перемещения между файлами.
3. **Debug** (Отладка) — предоставляет различные способы для просмотра состояния программы при отладке;
4. **Projects** (Проекты) — используется для настройки сборки, запуска и редактирования кода.
5. **Analyze** (Анализ) — в Qt интегрированы современные средства анализа кода разрабатываемого приложения.

[7]Окно на вашем компьютере визуально может несколько отличаться от представленного на рис. 1.1, авторы использовали IDE Qt Creator версии 2.6.2, основанную на QT 5.0.1.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

6. **Help** (Справка) — используется для вывода документации библиотеки Qt и `Qt Creator`.
7. **Output** (Вывод) — используется для вывода подробных сведений о проекте.

Рассмотрим простейшие приёмы работы в среде `Qt Creator` на примере создания консольного приложения для решения задачи 1.1. Для этого можно поступить одним из способов:

1. В меню **File** (Файл) выбрать команду **New File or Project** (Новый файл или проект) (комбинация клавиш **Ctrl**+**N**).
2. Находясь в режиме **Welcome** (Начало) главного окна QtCreator (рис. 1.1) щёлкаем по ссылке **Develop** (Разработка) и выбираем команду Create Project (Создать проект).

После это откроется окно, подобное представленному на рис. 1.2. Для создания простейшего консольного приложения выбираем **Non-Qt Project** (Проект без использования Qt) — **Plain C++ Project** (Простой проект на языке `C++`).

Далее выбираем имя проекта и каталог для его размещения (см. рис. 1.3)[8]. Следующие два этапа создания нашего первого приложения оставляем без изменения[9]. После чего окно `IDE Qt Creator` примет вид, подобное, представленное на рис. 1.4. Заменим текст текста стандартного приложения, которое выводит текст «Hello, Word», на текст программы решения задачи 1.1.

Для сохранения текста программы можно воспользоваться **Сохранить** или **Сохранить всё** из меню **Файл**. Откомпилировать и запустить программу можно одним из следующих способов:

1. Пункт меню **Сборка-Запустить**.
2. Нажать на клавиатуре комбинацию клавиш Ctrk+R.
3. Щёлкнуть по кнопке Запустить (▶).

Окно с результатами работы программы представлено на рис. 1.5.

Авторы сталкивались с тем, что в некоторых дистрибутивах Ubuntu Linux и Linux Mint после установки `Qt Creator` не запускались консольные приложения. Если читатель столкнулся с подобной

[8]Рекомендуем для каждого (даже самого простого) проекта выбирать отдельный каталог. Даже самый простой проект — это несколько взаимосвязанных между собой файлов и каталогов.

[9]О назначении этих этапов будет рассказано в дальнейших разделах книги.

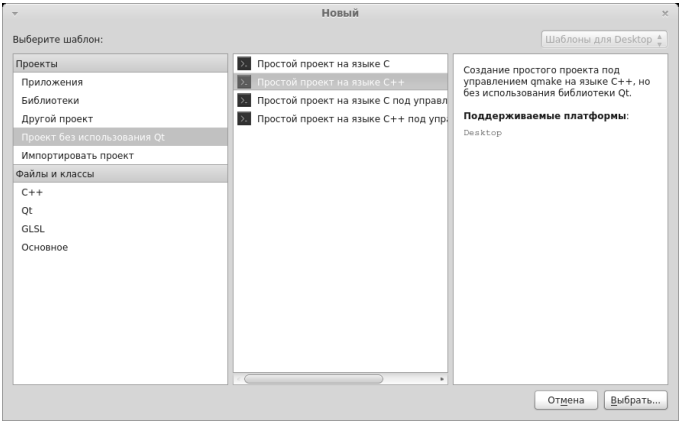


Рис. 1.2. Окно выбора типа приложения в Qt Creator

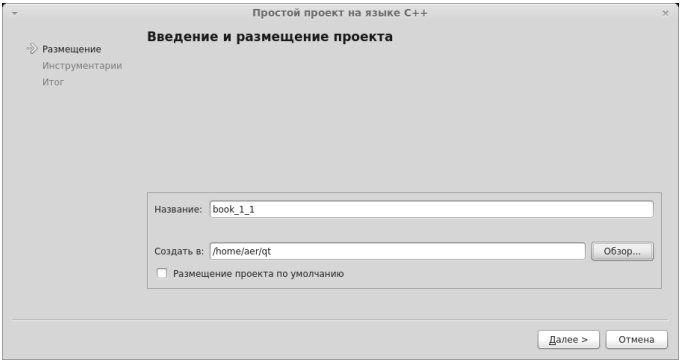


Рис. 1.3. Выбор имени и каталога нового проекта

проблемой, скорее всего надо корректно настроить терминал, который отвечает за запуск приложений в консоли. Для этого вызываем команду Tools — Options — Environment (см. рис. 1.6). Параметр **Terminal** (Терминал) должен быть таким же, как показано на рис. 1.6. Проверьте установлен ли в Вашей системе пакет xterm, и при необходимости доставьте его. После этого не должно быть проблем с запуском консольных приложений.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

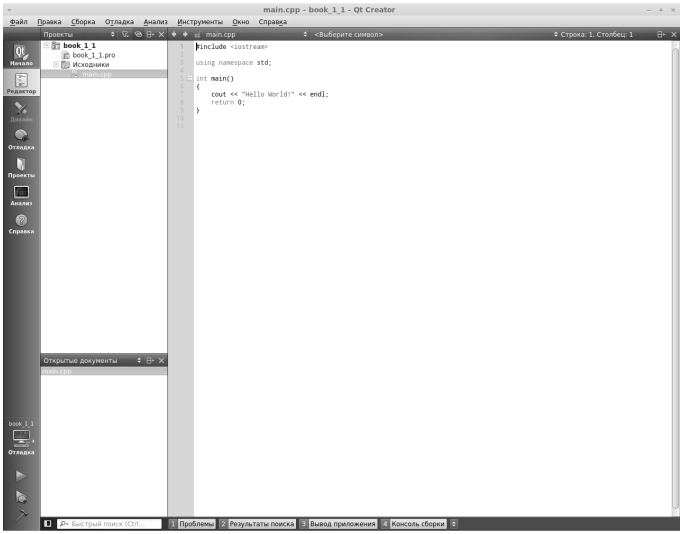


Рис. 1.4. Главное окно создания консольного приложения

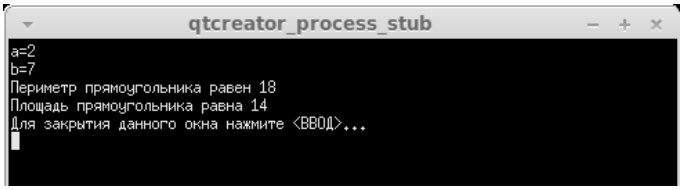


Рис. 1.5. Результаты работы программы решения задачи 1.1

Аналогичным образом можно создавать и запускать любое консольное приложение.

Дальнейшее знакомство со средой `Qt Creator` продолжим, решая следующую задачу.

**Задача 1.2.** Заданы длины трёх сторон треугольника a, b и c (см. рис. 1.7). Вычислить площадь и периметр треугольника

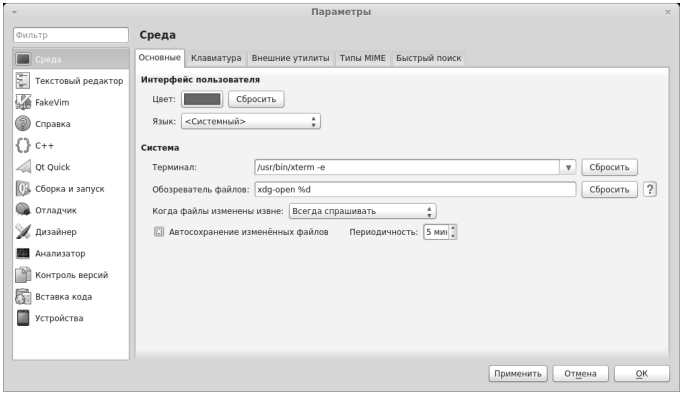


Рис. 1.6. Окно настроек среды Qt Creator

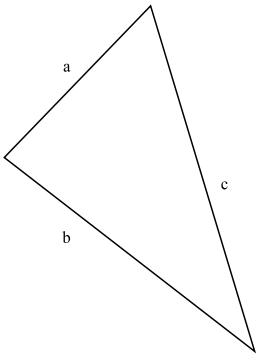


Рис. 1.7. Треугольник

Для решения задачи нам понадобится формула вычисления периметра $p = a + b + c$. Для вычисления площади можно воспользоваться формулой Герона $S = \sqrt{\frac{p}{2}\left(\frac{p}{2} - a\right)\left(\frac{p}{2} - b\right)\left(\frac{p}{2} - a\right)}$.

Решение задачи можно разбить на следующие этапы:

1. Определение значений a, b и c (ввод величин a, b, c с клавиатуры в память компьютера).
2. Расчет значений p и s по приведенным выше формулам.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

3. Вывод p и s на экран дисплея.

Ниже приведен текст программы. Сразу заметим, что в тексте могут встречаться строки, начинающие с двух наклонных (//), являющиеся комментариями. *Комментарии* не являются обязательными элементами программы и ничего не сообщают компьютеру, они поясняют человеку, читающему текст программы, назначение отдельных элементов программы. В книге комментарии будут широко использоваться для пояснения отдельных участков программы.

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    float a,b,c,s,p;
    cout<<"Введите длины сторон треугольника"<<endl;
//Ввод значений длин треугольника a, b, c.
    cin>>a>>b>>c;
//Вычисление периметра треугольника.
    p=a+b+c;
//Вычисление площади треугольника.
    s=sqrt(p/2*(p/2-a)*(p/2-b)*(p/2-c));
//Вывод на экран дисплея значений площади и периметра треугольника.
    cout<<"Периметр треугольника равен "<<p<<", его площадь
        равна "<<s<<endl;
    return 0;
}
```

Кроме используемой в предыдущей программе библиотеки `iostream`, в строке 2 подключим библиотеку `math.h`, которая служит для использования математических функций языка C(C++). В данной программе используется функция извлечения квадратного корня — *sqrt(x)*. Остальные операторы (ввода, вывода, вычисления значений переменных аналогичны используемым в предыдущей программе.

Таким образом, выше были рассмотрены самые простые программы (линейной структуры), которые предназначены для ввода исходных данных расчёта по формулам и вывода результатов.

# Глава 2

# Общие сведения о языке C++

В этой главе читатель познакомится с основными элементами языка C++: алфавитом, переменными, константами, типами данных, основными операциями, стандартными функциями, структурой программы и средствами ввода вывода данных.

## 2.1 Алфавит языка

Программа на языке C++ может содержать следующие символы:

- прописные, строчные латинские буквы A, B, C, . . . , x, y, z и знак подчеркивания;
- арабские цифры от 0 до 9;
- специальные знаки: “ { } , | [ ] ( ) + — / % * . \ ‘ : ? < = > ! & # ~ ; ^
- символы пробела, табуляции и перехода на новую строку.

Из символов алфавита формируют ключевые слова и идентификаторы. Ключевые слова — это зарезервированные слова, которые имеют специальное значение для компилятора и используются только в том смысле, в котором они определены (операторы языка, типы данных и т.п.). Идентификатор — это имя программного объекта, представляющее собой совокупность букв, цифр и символа подчеркивания. Первый символ идентификатора — буква или знак подчеркивания, но не цифра. Идентификатор не может содержать пробел. Прописные и строчные буквы в именах различаются, например, `ABC`, `abc`, `Abc` — три различных имени. Каждое имя (идентификатор) должно быть уникальным в пределах функции и не совпадать с ключевыми словами.

В тексте программы можно использовать комментарии. Если текст начинается с двух символов «косая черта» // и заканчивается символом перехода на новую строку или заключен между символами /* и */, то компилятор его игнорирует.

```
/* Комментарий можетвыглядеть так! */

//А если вы используете такой способ,
//то каждая строка должна начинаться
//с двух символов «косая черта».
```

Комментарии удобно использовать как для пояснений к программе, так и для временного исключения фрагментов программы при отладке.

## 2.2 Данные

Для решения задачи в любой программе выполняется обработка каких-либо данных. Данные хранятся в памяти компьютера и могут быть самых различных типов: целыми и вещественными числами, символами, строками, массивами. Типы данных определяют способ хранения чисел или символов в памяти компьютера. Они задают размер ячейки, в которую будет записано то или иное значение, определяя тем самым его максимальную величину или точность задания. Участок памяти (ячейка), в котором хранится значение определенного типа, называется переменной. У переменной есть имя (идентификатор) и значение. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменить. Перед использованием любая переменная должна быть описана. Оператор описания переменных в языке C++ имеет вид:

```
тип имя_переменной;
```

или

```
тип список_переменных;
```

Типы данных языка C++ можно разделить на основные и составные.

К основным типам данных языка относят:

- `char` — символьный;
- `int` — целый;
- `float` — с плавающей точкой;
- `double` — двойной точности;
- `bool` — логический.

Для формирования других типов данных используют основные типы и так называемые спецификаторы. Типы данных, созданные на базе стандартных типов с использованием спецификаторов, называют составными типами данных. В C++ определены четыре спецификатора типов данных:

- `short` — короткий;
- `long` — длинный;
- `signed` — знаковый;
- `unsigned` — беззнаковый.

Далее будут рассмотрены назначение и описание каждого типа.

### 2.2.1 Символьный тип

Данные типа `char` в памяти компьютера всегда занимают один байт[1]. Это связано с тем, что обычно под величину символьного типа отводят столько памяти, сколько необходимо для хранения любого из 256 символов клавиатуры. Символьный тип может быть со знаком или без знака (табл. 2.1).

Таблица 2.1: Символьные типы данных

| **Тип** | **Диапазон** | **Размер** |
|---|---|---|
| `char` | –128...127 | 1 байт |
| `unsigned char` | 0...255 | 1 байт |
| `signed char` | –128...127 | 1 байт |

Пример описания символьных переменных:

```
char c, str; //Описаны две символьные переменные.
```

При работе с символьными данными нужно помнить, что если в выражении встречается одиночный символ, он должен быть заключен в одинарные кавычки. Например, `'a'`, `'b'`, `'+'`, `'3'`.

Последовательность символов, то есть строка, при использовании в выражениях заключается в двойные кавычки: `"Hello!"`.

### 2.2.2 Целочисленный тип

Переменная типа `int` в памяти компьютера может занимать либо два, либо четыре байта. Это зависит от разрядности процессора.

[1] В кодировке utf каждый символ кириллицы занимает 2 байта.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Диапазоны значений целого типа представлены в таблице 2.2. По умолчанию все целые типы считаются знаковыми, т.е. спецификатор `signed` можно не указывать.

Таблица 2.2: Целые типы данных

| Тип | Диапазон | Размер |
|---|---|---|
| `int` | –2147483647 ... 2147483647 | 4 байта |
| `unsigned int` | 0 ... 4294967295 | 4 байта |
| `signed int` | –2147483647 ... 2147483647 | 4 байта |
| `short int` | –32767 ... 32767 | 2 байта |
| `long int` | –2147483647 ... 2147483647 | 4 байта |
| `unsigned short int` | 0 ... 65535 | 2 байта |
| `signed short int` | –32767 ... 32767 | 2 байта |
| `long long int` | $-(2^{63}-1)$ ... $(2^{63}-1)$ | 8 байт |
| `signed long int` | –2147483647 ... 2147483647 | 4 байта |
| `unsigned long int` | 0 ... 4294967295 | 4 байта |
| `unsigned long long int` | 0 ... $2^{64}-1$ | 8 байт |

Пример описания целочисленных данных:

```
int a, b, c;
unsigned long int A, B, C;
```

### 2.2.3 Вещественный тип

Внутреннее представление вещественного числа в памяти компьютера отличается от представления целого числа. Число с плавающей точкой представлено в экспоненциальной форме $mE \pm p$, где $m$ — мантисса (целое или дробное число с десятичной точкой), $p$ — порядок (целое число). Для того чтобы перевести число в экспоненциальной форме к обычному представлению с фиксированной точкой, необходимо мантиссу умножить на десять в степени порядок. Например,

$-6.42E + 2 = -6.42 \cdot 10^2 = -642$,

$3.2E - 6 = 3.2 \cdot 10^{-6} = 0.0000032$

Обычно величины типа `float` занимают 4 байта, из которых один двоичный разряд отводится под знак, 8 разрядов под порядок и 23 под мантиссу. Поскольку старшая цифра мантиссы всегда равна 1, она не хранится.

Величины типа `double` занимают 8 байт, в них под порядок и мантиссу отводится 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а дли-на порядка его диапазон. Спецификатор типа `long` перед именем типа `double` указывает, что под величину отводится 10 байт.

Диапазоны значений вещественного типа представлены в таблице 2.3.

Таблица 2.3: Вещественные типы данных

| Тип | Диапазон | Размер |
|---|---|---|
| `float` | 3.4E-38 ... 3.4E+38 | 4 байта |
| `double` | 1.7E-308 ... 1.7E+308 | 8 байт |
| `long double` | 3.4E-4932 ... 3.4E+4932 | 10 байт |

Пример описания вещественных переменных:

```
double x1,x2,x3;
float X, Y, Z;
```

### 2.2.4 Логический тип

Переменная типа `bool` может принимать только два значения `true` (истина) или `false` (ложь). Любое значение не равное нулю интерпретируется как `true`, а при преобразовании к целому типу принимает значение равное 1. Значение `false` представлено в памяти как 0.

Пример описания данных логического типа:

```
bool F, T;
```

### 2.2.5 Тип void

Множество значений этого типа пусто. Он используется для определения функций, которые не возвращают значения, для указания пустого списка аргументов функции, как базовый тип для указателей и в операции приведения типов.

## 2.3 Константы

Константы это величины, которые не изменяют своего значения в процессе выполнения программы. Оператор описания константы имеет вид:

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
const тип имя_константы = значение;
```

Константы в языке C++ могут быть целыми, вещественными, символьными или строковыми. Обычно компилятор определяет тип константы по внешнему виду, но существует возможность и явного указания типа, например,

```
const double pi=3.141592653589793
```

Кроме того, константа может быть определена с помощью директивы[2] `#define`. Эта директива служит для замены часто использующихся констант, ключевых слов, операторов или выражений некоторыми идентификаторами. Идентификаторы, заменяющие текстовые или числовые константы, называют именованными константами. Основная форма синтаксиса директивы следующая:

```
#define идентификатор текст
```

Например,

```
#define PI 3.141592653589793
int main()
...
```

## 2.4 Структурированные типы данных

Структурированный тип данных характеризуется множественностью образующих его элементов. В C++ это массивы, строки, структуры и файлы.

Массив — совокупность данных одного и того же типа[3]. Число элементов массива фиксируется при описании типа и в процессе выполнения программы не изменяется.

В общем виде массив можно описать так:

```
тип имя [размерность_1][размерность_2]...[размерность_N];
```

Например,

```
float x[10];     //Описан массив из 10 целых чисел.
int a[3][4];     //Описан двумерный массив,
              //матрица из 3-х строк и 4-х столбцов.
double b[2][3][2];  //Описан трехмерный массив.
```

Для доступа к элементу массива достаточно указать его порядковый номер, а если массив многомерный (например, таблица), то несколько номеров:

[2]Структура программы и директивы описаны в п. 2.8

[3]Подробно работа с одномерными и двумерными массивами описана в главах ?? и ??.

```
имя_массива[номер_1][номер_2]...[номер_N]
```

Например: `x[5]`, `a[2][3]`, `b[1][2][2]`.

Элементы массива в C++ нумеруются с нуля. Первый элемент, всегда имеет номер ноль, а номер последнего элемента на единицу меньше заданной при его описании размерности:

```
char C[5];  //Описан массив из 5 символов,
           //нумерация от 0 до 4.
```

Строка — последовательность символов[4]. В C++ строки описывают как массив элементов типа `char`. Например:

```
char s[25];   // Описана строка из 25 символов.
```

Структура[5] это тип данных, который позволяет объединить разнородные данные и обрабатывать их как единое целое.

Например

```
struct fraction  //Объявлена структура правильная дробь.
{
        //Определяем поля структуры:
  int num;    // поле числитель,
  int den;    // поле знаменатель.
}
...
fraction d, D[20];   //Определена переменная d,
                    //массив D[20], типа fraction.
...
d.num;         //Обращение к полю num переменной d.
D[4].den;     //Обращение к полю den
             //четвертого элемента массива D.
```

## 2.5 Указатели

Указатели широко применяются в C++. Можно сказать, что именно наличие указателей сделало этот язык удобным для системного программирования. С другой стороны это одна из наиболее сложных для освоения возможностей C++. Идея работы с указателями состоит в том, что пользователь работает с адресом ячейки памяти.

Как правило, при обработке оператора объявления переменной

```
тип имя_переменной;
```

[4]Работа со строками описана в главе **??**

[5]Работа со структурами описана в главе **??**.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

компилятор автоматически выделяет память под переменную имя_переменной в соответствии с указанным типом:

```
char C; //Выделена память под символьную переменную C
```

Доступ к объявленной переменной осуществляется по ее имени. При этом все обращения к переменной заменяются на адрес ячейки памяти, в которой хранится ее значение. При завершении программы или функции, в которой была описана переменная, память автоматически освобождается.

Доступ к значению переменной можно получить иным способом — определить собственные переменные для хранения адресов памяти. Такие переменные называют указателями. С помощью указателей можно обрабатывать массивы, строки и структуры, создавать новые переменные в процессе выполнения программы, передавать адреса фактических параметров функциям и адреса функций в качестве параметров.

Итак, указатель это переменная, значением которой является адрес памяти, по которому хранится объект определенного типа (другая переменная). Например, если C это переменная типа char, а P — указатель на C, значит в P находится адрес по которому в памяти компьютера хранится значение переменной C.

Как и любая переменная, указатель должен быть объявлен. При объявлении указателей всегда указывается тип объекта, который будет храниться по данному адресу:

```
тип *имя_переменной;
Например:
int *p;     //По адресу, записанному в переменной p,
        //будет храниться переменная типа int
```

Звездочка в описании указателя, относится непосредственно к имени, поэтому чтобы объявить несколько указателей, ее ставят перед именем каждого из них:

```
float *x, y, *z;   //Описаны указатели на вещественное
//число -- x и z,само вещественное значение -- *x, *z.
//а так же вещественная переменная y, её адрес -- &y.
```

## 2.6 Операции и выражения

Выражение задает порядок выполнения действий над данными и состоит из операндов (констант, переменных, обращений к функциям), круглых скобок и знаков операций. Операции делятся на унар-

ные (например, -c) и бинарные (например, a+b). В таблице 2.4 представлены основные операции языка C++.

Таблица 2.4: Основные операции языка C++

| Операция | Описание |
|---|---|
| **Унарные операции** | |
| ++ | увеличение значения на единицу |
| -- | уменьшение значения на единицу |
| ~ | поразрядное отрицание |
| ! | логическое отрицание |
| - | арифметическое отрицание (унарный минус) |
| + | унарный плюс |
| & | взятие адреса |
| * | разадресация |
| (type) | преобразование типа |
| **Бинарные операции** | |
| + | сложение |
| - | вычитание |
| * | умножение |
| / | деление |
| % | остаток от деления |
| << | сдвиг влево |
| >> | сдвиг вправо |
| < | меньше |
| > | больше |
| <= | меньше или равно |
| >= | больше или равно |
| == | равно |
| != | не равно |
| & | поразрядная конъюнкция (И) |
| ^ | поразрядное исключающее ИЛИ |
| \| | поразрядная дизъюнкция (ИЛИ) |
| && | логическое И |
| \|\| | логическое ИЛИ |
| = | присваивание |
| *= | умножение с присваиванием |
| /= | деление с присваиванием |

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Таблица 2.4 — продолжение

| Операция | Описание |
|---|---|
| += | сложение с присваиванием |
| -= | вычитание с присваиванием |
| %= | остаток от деления с присваиванием |
| <<= | сдвиг влево с присваиванием |
| >>= | сдвиг вправо с присваиванием |
| &= | поразрядная конъюнкция с присваиванием |
| \|= | поразрядная дизъюнкция с присваиванием |
| ^= | поразрядное исключающее ИЛИ с присваиванием |
| **Другие операции** | |
| ? | условная операция |
| , | последовательное вычисление |
| sizeof | определение размера |
| (тип) | преобразование типа |

Перейдем к подробному рассмотрению основных операций языка.

### 2.6.1 Операции присваивания

Обычная операция присваивания имеет вид:

`имя_переменной=значение;`

где `значение` это выражение, переменная, константа или функция. Выполняется операция так. Сначала вычисляется значение выражения указанного в правой части оператора, а затем его результат записывается в область памяти, имя которой указано слева.

Например,

```
b=3;     //Переменной b присваивается значение равное трем.
a=b;     // Переменной a присваивается значение b.
c=a+2*b; // Переменной c присваивается значение выражения.
c=c+1;  // Значение переменой c увеличивается на единицу.
a=a*3;  // Значение переменой a увеличивается в три раза.
```

**Задача 2.1.** Пусть в переменной `a` хранится значение равное 3, а в переменную `b` записали число 5. Поменять местами значения переменных `a` и `b`.

Для решения задачи понадобится дополнительная переменная `c` (см. рис. 2.1). В ней временно сохраняется значение переменной `a`. Затем, значение переменной `b` записывается в переменную `a`, а значение переменной `c` в переменную `b`.

```
c=a;    // Шаг 1. c=3
a=b;    // Шаг 2. a=5
b=c;    // Шаг 3. b=3
```

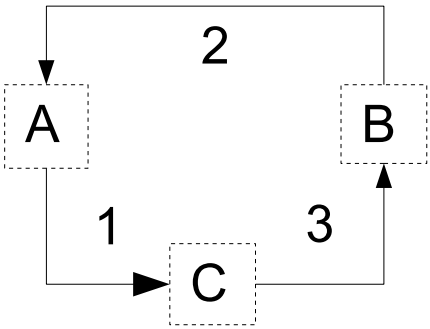


Рис. 2.1. Использование буферной переменной

Если в операторе присваивания левая и правая часть это переменные разных типов, то *происходит преобразование*: значение переменной в правой части преобразуется к типу переменной в левой части. Следует учитывать, что при этом можно потерять информацию или получить другое значение.

В C++ существует возможность присваивания нескольким переменным одного и того же значения. Такая операция называется *множественным присваиванием* и в общем виде может быть записана так:

```
имя_1 = имя_2 = ... = имя_N = значение;
```

Запись `a=b=c=3.14159/6;`

означает, что переменным `a`, `b` и `c` было присвоено одно и то же значение `3.14159/6`.

Операции +=, −=, *=, /= называют *составным присваиванием*. В таких операциях при вычислении выражения стоящего справа используется значение переменной из левой части, например так:

```
x+=p; //Увеличение x на p, то же что и x=x+p.
x-=p; //Уменьшения x на p, то же что и x=x-p.
x*=p; //Умножение x на p, то же что и x=x*p.
x/=p; //Деление x на p, то же что и x=x/p.
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

### 2.6.2 Арифметические операции

Операции +, -, *, / относят к *арифметическим операциям*. Их назначение понятно и не требует дополнительных пояснений. При программировании арифметических выражений следует придерживаться простых правил. Соблюдать очерёдность выполнения арифметических операций. Сначала выполняются операции умножения и деления (1-й уровень), а затем сложения и вычитания (2-й уровень). Операции одного уровня выполняются последовательно друг за другом. Для изменения очерёдности выполнения операций используют скобки. Таблица 2.5 содержит примеры записи алгебраических выражений.

Таблица 2.5: Примеры записи алгебраических выражений

| **Математическая запись** | **Запись на языке C++** |
|---|---|
| $2 \cdot a + b \cdot (c + d)$ | `2*a+b*(c+d)` |
| $3 \cdot \dfrac{a+b}{c+d}$ | `3*(a+b)/(c+d)` |
| $\dfrac{3 \cdot a - 2 \cdot b}{c \cdot d}$ | `(3*a-2*b)/(c*d)` или `(3*a-2*b)/c/d` |
| $\dfrac{(b-a)^2}{c + \dfrac{1}{d-2}} - \dfrac{a^2+1}{b^2+cd}$ | `(b-a)*(b-a)/(c+1/(d-2))-(a*a+1)/(b*b+c*d)` |

Операции *инкремента* ++ и *декремента* -- так же причисляют к арифметическим, так как они выполняют увеличение и уменьшение на единицу значения переменной. Эти операции имеют две формы записи: префиксную (операция записывается перед операндом) и постфиксную (операция записывается после операнда). Так, например оператор `p=p+1;` можно представить в префиксной форме `++p;` и в постфиксной `p++;`. Эти формы отличаются при использовании их в выражении. Если знак декремента (инкремента) предшествует операнду, то сначала выполняется увеличение (уменьшение) значения операнда, а затем операнд участвует в выражении. Например,

```
x=12;
y=++x; //В переменных x и y будет храниться значение 13.
```

Если знак декремента (инкремента) следует после операнда, то сначала операнд участвует в выражении, а затем выполняется увеличение (уменьшение) значения операнда:

```
x=12;
y=x++; //Результат — число 12 в переменной y, а в x — 13.
```

Остановимся на *операциях целочисленной арифметики.*

Операция целочисленного деления / возвращает целую часть частного (дробная часть отбрасывается) в том случае если она применяется к целочисленным операндам, в противном случае выполняется обычное деление: 11/4=2 или 11.0/4=2.75.

Операция остаток от деления % применяется только к целочисленным операндам: 11%4 = 3.

К *операциям битовой арифметики* относятся следующие операции: &, |, ^, ~, <<, >>. В операциях битовой арифметики действия происходят над двоичным представлением целых чисел.

*Арифметическое И* (&). Оба операнда переводятся в двоичную систему, затем над ними происходит логическое поразрядное умножение операндов по следующим правилам:

1&1=1, 1&0=0, 0&1=0, 0&0=0.

Например, если A=14 и B=24, то их двоичное представление — A=0000000000001110 и B=0000000000011000. В результате логического умножения A and B получим 0000000000001000 или 8 в десятичной системе счисления (рис. 2.2). Таким образом, A&B=14&24=8.

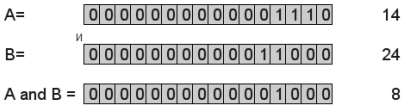


Рис. 2.2. Пример логического умножения

*Арифметическое ИЛИ* (|). Здесь также оба операнда переводятся в двоичную систему, после чего над ними происходит логическое поразрядное сложение операндов по следующим правилам:

1|1=1, 1|0=1, 0|1=1, 0|0=0.

Например, результат логического сложения чисел A=14 и B=24 будет равен A or B=30 (рис. 2.3).

*Арифметическое исключающее ИЛИ* (^). Оба операнда переводятся в двоичную систему, после чего над ними происходит логическая поразрядная операция ^ по следующим правилам:

1^1=0, 1^0=1, 0^1=1, 0^0=0.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.


Рис. 2.3. Пример логического сложения

*Арифметическое отрицание* (~). Эта операция выполняется над одним операндом. Применение операции ~ вызывает побитную инверсию двоичного представления числа (рис. 2.4).

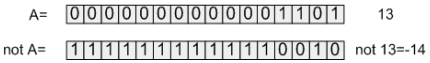


Рис. 2.4. Пример арифметического отрицания

*Сдвиг влево* (`M<<L`). Число `M`, представленное в двоичной системе, сдвигается влево на `L` позиций. Рассмотрим операцию `15 shl 3`. Число 15 в двоичной системе имеет вид `1111`. При сдвиге его на 3 позиции влево получим `1111000`. В десятичной системе это двоичное число равно `120`. Итак, `15 shl 3 =120` (рис. 2.5). Заметим, что сдвиг на один разряд влево соответствует умножению на два, на два разряда — умножению на четыре, на три — умножению на восемь. Таким образом, операция `M<<L` эквивалентна умножению числа `M` на 2 в степени `L`.


Рис. 2.5. Пример операции «Сдвиг влево»

*Сдвиг вправо* (`M>>L`). Число `M`, представленное в двоичной системе, сдвигается вправо на `L` позиций, что эквивалентно целочисленному делению числа `M` на 2 в степени `L`. Например, `15 shr 1=7` (рис. 2.6), `15 shr 3= 2`.


Рис. 2.6. Пример операции «Сдвиг вправо»

### 2.6.3 Логические операции

В C++ определены следующие логические операции || (или), && (и), ! (не). Логические операции выполняются над логическими значениями true (истина) и false (ложь). В языке C++ ложь — 0, истина — любое значение $\neq 0$. В таблице 2.6 приведены результаты логических операций.

Таблица 2.6: Логические операции

| A | B | !A | A&&B | A\|\|B |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |

### 2.6.4 Операции отношения

*Операции отношения* возвращают в качестве результата логическое значение. Таких операций шесть: >, >=, <, <=, ==, !=. Результат операции отношения — логическое значение true (истина) или false (ложь).

### 2.6.5 Условная операция

Для организации разветвлений в простейшем случае можно использовать *условную операцию* ? :. Эта операция имеет три операнда и в общем виде может быть представлена так:

```
условие ? выражение1 : выражение2;
```

Работает операция следующим образом. Если условие истинно (не равно 0), то результатом будет выражение1, в противном случае выражение2. Например, операция y=x<0 ? -x : x; записывает в переменную y модуль числа x.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

### 2.6.6 Операция преобразования типа

Для приведения выражения к другому типу данных в C++ существует *операция преобразования типа*:

`(тип) выражение;`

Например, в результате действий `x=5; y=x/2; z=(float) x/2;` переменная `y` примет значение равное 2 (результат целочисленного деления), а переменная `z = 2.5`.

### 2.6.7 Операция определения размера

Вычислить размер объекта или типа в байтах можно с помощью *операции определения размера*, которая имеет две формы записи:

`sizeof (тип);` или `sizeof выражение;`

Например, предположим, что была описана целочисленная переменная `int k=3;`. Исходя из того, что тип `int` занимает в памяти 4 байта, в переменную `m=sizeof k;` будет записано число 4.

В результате работы команд `double z=123.456; p=sizeof (k+z);` значение переменной `p` стало равно 8, т. к. вещественный тип `double` более длинный (8 байтов) по сравнению с типом `int` (4 байта) и значение результата было преобразовано к более длинному типу. В записи операции `sizeof (k+z)` были использованы скобки. Это связано с тем, что операция определения типа имеет более высокий приоритет, чем операция сложения. При заданном значении `z=123.456`; та же команда, но без скобок `p=sizeof k+z;` вычислит `p=4+123.456=127.456`.

Команда `s = sizeof "Hello"`; определит, что под заданную строку в памяти было выделено `s=6` байтов, т. к. объект состоит из 5 символов и один байт на символ окончания строки.

### 2.6.8 Операции с указателями

При работе с указателями часто используют операции *получения адреса* `&` и *разадресации* `*` (табл. 2.7).

Таблица 2.7: Операции получения адреса `&` и разадресации `*`

| Описание | Адрес | Значение, хранящееся по адресу |
|---|---|---|
| `тип *p` | `p` | `*p` |
| `тип p` | `&p` | `p` |

*Операция получения адреса* `&` возвращает адрес своего операнда. Например:

```
float a;     //Объявлена вещественная переменная a
float *adr_a; //Объявлен указатель на тип float
adr_a=&a;    //Оператор записывает в переменную adr_a
             //адрес переменной a
```

*Операция разадресации* `*` возвращает значение переменной, хранящееся по заданному адресу, т.е. выполняет действие, обратное операции `&`:

```
float a;     //Объявлена вещественная переменная a.
float *adr_a;   //Объявлен указатель на тип float.
a=*adr_a;     //Оператор записывает в переменную a
     //вещественное значение, хранящееся по адресу adr_a.
```

К указателям применяют *операцию присваивания*. Это значит, что значение одного указателя можно присвоить другому. Если указатели одного типа, то для этого применяют обычную операцию *присваивания*:

```
//Описана вещественная переменная и два указателя.
float PI=3.14159,*p1,*p2;
//В указатели p1 и p2 записывается адрес переменной PI.
p1=p2=&PI;
```

Если указатели ссылаются на различные типы, то при присваивании значения одного указателя другому, необходимо использовать преобразование типов. Без преобразования можно присваивать любому указателю указатель `void*`.

Рассмотрим пример работы с указателями различных типов:

```
float PI=3.14159;   //Объявлена вещественная переменная.
float *p1;     //Объявлен указатель на float.
double *p2;    //Объявлен указатель на double.
p1=&PI; //Переменной p1 присваивается значение адреса PI.
p2=(double *)p1;   //Указателю на double присваивается
             //значение, которое ссылается на тип float.
```

В указателях `p1` и `p2` хранится один и тот же адрес (`p1=0012FF7C`), но значения, на которые они ссылаются разные (`*p1=3.14159`, `*p2=2.642140e-308`). Это связано с тем, указатель типа `*float` адресует 4 байта, а указатель `*double` — 8 байт. После присваивания `p2=(double *)p1;` при обращении к `*p2` происходит следующее: к переменной, хранящейся по адресу `p1`, дописывается еще следующих 4 байта из памяти. В результате значение `*p2` не совпадает со значением `*p1`.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Таким образом, при преобразовании указателей разного типа приведение типов разрешает только синтаксическую проблему присваивания. Следует помнить, что операция * над указателями различного типа, ссылающимися на один и тот же адрес, возвращает различные значения.

Над адресами C++ определены следующие *арифметические операции*:

- *сложение и вычитание указателей с константой*;
- *вычитание одного указателя из другого*;
- *инкремент*;
- *декремент*.

*Сложение и вычитание указателей с константой* n означает, что указатель перемещается по ячейкам памяти на столько байт, сколько занимает n переменных того типа на который он указывает. Например, пусть указатель имеет символьный тип и его значение равно 100. Результат сложения этого указателя с единицей — 101, так как для хранения переменной типа char требуется один байт. Если же значение указателя равно 100, но он имеет целочисленный тип, то результат его сложения с единицей будет составлять 104, так как для переменной типа int отводится четыре байта.

Эти операции применимы только к указателям одного типа и имеют смысл в основном при работе со структурными типами данных, например массивами.

Фактически получается, что значение указателя изменяется на величину sizeof(тип). Если указатель на определенный тип увеличивается или уменьшается на константу, то его значение изменяется на величину этой константы, умноженную на размер объекта данного типа. Например:

```
//Объявление массива из 10 элементов.
double mas[10]={1.29,3.23,7.98,5.54,8.32,2.48,7.1};
double *p1;   //Объявление указателя на double
p1=&mas[0];   //Присвоение указателю адреса
        //нулевого элемента массива.
p1=p1+3;    //Увеличение значения адреса на 3*8=24
        //(размер типа double), в результате указатель
      //сместится на три ячейки, размером double каждая.
```

*Вычитание двух указателей* определяет сколько переменных данного типа размещается между указанными ячейками. Разность двух

указателей это разность их значений, деленная на размер типа в байтах. Так разность указателей на третий и нулевой элементы массива равна трем, а на третий и девятый — шести. Суммирование двух указателей не допускается.

Операции *инкремента* и *декремента*, соответственно, увеличивают или уменьшают значение адреса:

```
double *p1;
float *p2;
int *i;
p1++;   //Увеличение значения адреса на 8.
p2++;   //Увеличение значения адреса на 4.
i++;     //Увеличение значения адреса на 4.
```

К указателям так же применимы операции отношения `==`, `!=`, `<`, `>`, `<=`, `>=`. Иными словами указатели можно сравнивать. Например, если `i` указывает на пятый элемент массива, а `j` на первый, то отношение `i>j` истинно. Кроме того, любой указатель всегда можно сравнить на равенство с константой нулевого указателя (NULL)[6]. Однако, все эти утверждения верны, если речь идёт об указателях ссылающихся на один массив. В противном случае результат арифметических операций и операций отношения будет не определён.

## 2.7 Стандартные функции

В C++ определены *стандартные функции* над арифметическими операндами[7]. В таблице 2.8 приведены некоторые из них.

Таблица 2.8: Стандартные математические функции

| **Обозначение** | **Действие** |
|---|---|
| `abs(x)` | Модуль целого числа $x$ |
| `fabs(x)` | Модуль вещественного числа $x$ |
| `sin(x)` | Синус числа $x$ |
| `cos(x)` | Косинус числа $x$ |
| `tan(x)` | Тангенс числа $x$ |
| `atan(x)` | Арктангенс числа $x$, $x \in (-\frac{i\pi}{2}; \frac{i\pi}{2})$ |

[6]Константу нулевого указателя можно присвоить любому указателю и такой указатель при сравнении не будет равен любому реальному указателю.

[7]Работа с математическими функциями возможно только при подключении директивы math.h (п. **??**)

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Таблица 2.8 — продолжение

| Обозначение | Действие |
|---|---|
| `acos(x)` | Арккосинус числа $x$ |
| `asin(x)` | Арксинус числа $x$ |
| `exp(x)` | Экспонента, $e^x$ |
| `log(x)` | Натуральный логарифм, $(x > 0)$ |
| `log10(x)` | Десятичный логарифм, $(x > 0)$ |
| `sqrt(x)` | Корень квадратный, $(x > 0)$ |
| `pow(x,y)` | Возведение числа $x$ в степень $y$ |
| `ceil(x)` | Округление числа $x$ до ближайшего большего целого |
| `floor(x)` | Округление числа $x$ до ближайшего меньшего целого |

Примеры записи математических выражений с использованием встроенных функций представлены в таблице 2.9.

Таблица 2.9: Примеры записи математических выражений

| Математическая запись | Запись на языке C++ |
|---|---|
| $\sqrt[3]{(a+b)^2}$ | `pow((a+b)*(a+b),1./3)` или `pow(pow(a+b,2),1./3)` |
| $\cos^4(x)$ | `pow(cos(x), 4)` |
| $e^{2x}$ | `exp(2*x)` |
| $e^{5\sin(\frac{x}{2})}$ | `exp(5*sin(x/2))` |
| $\sin^2(\sqrt{x})$ | `pow(sin(sqrt(x)),2)` |
| $\ln(\lvert x-2\rvert)$ | `log(fabs(x-2))` |
| $\log_b a$ | `log(a)/log(b)` |
| $\frac{lg(x^2+1)}{lg(4)}$ | `log10(x*x+1)/log10(4)` |
| $\sin(x^2+y^2)+\cos\frac{(x^2+y^2)}{2\cdot y}+$ $+\sqrt{x^2+y^2}$ | `z=x*x+y*y;` `sin(z)+cos(z/(2*y))+sqrt(z);` |

Определенную проблему представляет применение функции `pow(x,y)`. При программировании выражений, содержащих возведение в степень, надо внимательно проанализировать значения, которые могут принимать $x$ и $y$, так как в некоторых случаях возведение $x$ в степень $y$ невыполнимо.

Так, ошибка возникает, если $x$ — отрицательное число, а $y$ — дробь. Предположим, что $y$ — правильная дробь вида $\frac{k}{m}$. Если знаменатель $m$ четный, это означает вычисление корня четной степени из отрицательного числа, а значит, операция не может быть выполнена. В противном случае, если знаменатель $m$ нечетный, можно воспользоваться выражением `z = -pow(fabs(x),y)`. Например, вычисление кубического корня из вещественного числа можно представить командой:

```
z=(x<0) ? -pow(fabs(x),(double) 1/3) : pow(x,(double) 1/3);
```

## 2.8 Структура программы

*Программа* на языке C++ состоит из *функций*, *описаний* и *директив процессора*.

Одна из функций должна обязательно носить имя `main`. Элементарное описание функции имеет вид:

```
тип_результата имя_функции (параметры)
{
    оператор1;
    оператор2;
    ...
    операторN;
}
```

Здесь, `тип_результата` — это тип того значения, которое функция должна вычислить (если функция не должна возвращать значение, указывается тип `void`), `имя_функции` — имя, с которым можно обращаться к этой функции, `параметры` — список аргументов функции (может отсутствовать), `оператор1`, `оператор2`,`...`, `операторN` — операторы, представляющие тело функции, они обязательно заключаются в фигурные скобки и каждый оператор заканчивается точкой с запятой. Как правило программа на C++ состоит из одной или нескольких, не вложенных друг в друга функций.

Основному тексту программы предшествуют *директивы препроцессора* предназначенные для *подключения библиотек*, которые в общем виде выглядят так:

```
#include <имя_файла>
```

Каждая такая строка дает компилятору команду присоединить программный код, который хранится в отдельном файле с расширением `.h`. Такие файлы называют *файлами заголовков*. С их помощью можно выполнять ввод-вывод данных, работать с математическими

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

функциями, преобразовывать данные, распределять память и многое другое. Например, описание стандартных математических функций находится в заголовочном файле `math.h`.

Общую структуру программы на языке C++ можно записать следующим образом:

```
директивы процессора
описание глобальных переменных
тип_результата main(параметры)
{
описание переменных главной функции;
  операторы главной функции;
}

тип_результата имя1(параметры1)
{
описание переменных функции имя1;
  операторы1;
}

тип_результата имя2(параметры2)
{
описание переменных функции имя2;
  операторы2;
}

. . . . . .

тип_результата имяN(параметрыN)
{
описание переменных функции имяN;
  операторыN;
}
```

По месту объявления переменные в языке Си можно разделить на три класса: локальные, глобальные и формальные параметры функции.

*Локальные переменные* объявляются внутри функции и доступны только в ней. Например:

```
int main()
{
//В функции main определена вещественная переменная s,
    float s;
    s=4.5;    //и ей присвоено значение 4.5.
}
int f1()
{
//В функции f1 описана другая переменная s,
```

```
    int s;
    s=6;    //ей присвоено значение 6.
}
int f2()
{
//В функции f2 определена еще одна переменная s,
        long int s;
        s=25;        //ей присвоено значение 25.
}
```

*Глобальные переменные* описываются до всех функций и доступны из любого места программы. Например:

```
float s; //Определена глобальная переменная s.
int main()
{
//В главной функции переменной s присваивается значение 4.5.
        s=4.5;
}
int f1()
{
//В функции f1 переменной s присваивается значение 6.
        s=6;
}
int f2()
{
//В функции f2 переменной s присваивается значение 2.1.
    s=2.1;
```

Формальные параметры функций описываются в списке параметров функции. Работа с функциями подробно описана в главе **??**.

## 2.9 Ввод и вывод данных

Ввод-вывод данных в языке C++ осуществляется либо с помощью функций ввода-вывода в стиле C, либо с использованием библиотеки классов C++. Преимущество объектов C++ в том, что они легче в использовании, особенно если ввод-вывод достаточно простой. Функции ввода-вывода унаследованные от C более громоздкие, но более гибко управляют форматированным выводом данных.

Функция

`printf(строка форматов, список выводимых переменных);`

выполняет форматированный *вывод переменных*, указанных в списке, в соответствии со строкой форматов.

Функция

`scanf(строка форматов, список адресов вводимых переменных);`

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

выполняет *ввод переменных*, адреса которых указанны в списке, в соответствии со строкой форматов.

*Строка форматов* содержит символы, которые будут выводиться на экран или запрашиваться с клавиатуры и так называемые спецификации. *Спецификации* это строки, которые начинаются символом `%` и выполняют управление форматированием:

```
% флаг ширина . точность модификатор тип
```

Параметры `флаг`, `ширина`, `точность` и `модификатор` в спецификациях могут отсутствовать. Значения параметров спецификаций приведены в таблице 2.10.

Таблица 2.10: Символы управления

| **Параметр** | **Назначение** |
|---|---|
| Флаги | |
| `-` | Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо. |
| `+` | Перед числом выводится знак «+» или «-» |
| `Пробел` | Перед положительным числом выводится пробел, перед отрицательным «-» |
| `#` | Выводится код системы счисления: `0` — перед восьмеричным числом, `0x` (`0X`) перед шестнадцатеричным числом. |
| Ширина | |
| `n` | Ширина поля вывода. Если `n` позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции заполняются пробелами. |
| `0n` | То же, что и `n`, но незаполненные позиции заполняются нулями. |
| Точность | |
| `ничего` | Точность по умолчанию |
| `n` | Для типов `e`, `E`, `f` выводить `n` знаков после десятичной точки |
| Модификатор | |
| `h` | Для `d`, `i`, `o`, `u`, `x`, `X` тип `short int`. |
| `l` | Для `d`, `i`, `o`, `u`, `x`, `X` тип `long int`. |
| Тип | |

Таблица 2.10 — продолжение

| Параметр | Назначение |
|---|---|
| c | При вводе символьный тип char, при выводе один байт. |
| d | Десятичное int со знаком. |
| i | Десятичное int со знаком. |
| o | Восьмеричное int unsigned. |
| u | Десятичное int unsigned. |
| x, X | Шестнадцатеричное int unsigned, при x используются символы a-f, при X — A - F. |
| f | Значение со знаком вида [-]dddd.dddd. |
| e | Значение со знаком вида [-]d.dddde[+\|-]ddd. |
| E | Значение со знаком вида [-]d.ddddE[+\|-]ddd. |
| g | Значение со знаком типа e или f в зависимости от значения и точности. |
| G | Значение со знаком типа e или F в зависимости от значения и точности. |
| s | Строка символов. |

Кроме того, строка форматов может содержать некоторые специальные символы, которые приведены в таблице 2.11.

Таблица 2.11: Специальные символы

| Символ | Назначение |
|---|---|
| \b | Сдвиг текущей позиции влево. |
| \n | Перевод строки. |
| \r | Перевод в начало строки, не переходя на новую строку. |
| \t | Горизонтальная табуляция. |
| \’ | Символ одинарной кавычки. |
| \’’ | Символ двойной кавычки. |
| \? | Символ ? |

Первой строкой программы, в которой будут применяться функции ввода-вывода языка C, должна быть директива #include <stdio.h>. Заголовочный файл stdio.h содержит описание функций ввода-вывода.

Рассмотрим работу функций на примере следующей задачи.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

**Задача 2.2.** Зная a, b, c — длины сторон треугольника, вычислить площадь S и периметр P этого треугольника.

Входные данные: a, b, c. Выходные данные: S, P.

Для вычисления площади применим формулу Герона:

$S = \sqrt{r \cdot (r-a) \cdot (r-b) \cdot (r-c)}$, где $r = \frac{a+b+c}{2}$ — полупериметр.

Далее приведены две программы для решения данной задачи и результаты их работы (рис. 2.7–2.8). Сравните работу функций printf и scanf в этих программах.

```
//ЗАДАЧА 2.2 Вариант первый
#include <iostream>
#include <stdio.h>
#include <math.h>
using namespace std;
int main()
{
   float a,b,c,S,r;    //Описание переменных.
   printf("a=");       //Вывод на экран символов  a=.
   //В функции scanf для вычисления адреса
   //переменной применяется операция  &.
   scanf("%f",&a);     //Запись в переменную a значения
                       //введенного с клавиатуры.
   printf("b=");       //Вывод на экран символов  b=.
   scanf("%f",&b);     //Запись в переменную  b  значения
                       //введенного с клавиатуры.
   printf("c=");       //Вывод на экран символов  c=
   scanf("%f",&c);     //Запись в переменную с значения
                       //введенного с клавиатуры.
   r=(a+b+c)/2;        //Вычисление полупериметра.
   S=sqrt(r*(r-a)*(r-b)*(r-c)); //Вычисление площади треугольника.
   printf("S=%5.2f \t",S); //Вывод символов  S=,
                           //значения  S  и символа
                           //табуляции  \t.
                           //Спецификация  %5.2f
                           //означает, что будет
                           //выведено вещественное
                           //число из пяти знаков,
                           //два из которых после точки.
   printf("p=%5.2f \n",2*r); //Вывод символов  p=,
                             //значения выражения  2*r
                             //и символа окончания строки.
   //Оператор  printf("S=%5.2f \t p=%5.2f \n",S,2*r);
   //выдаст тот же результат.
   return 0;
}

//ЗАДАЧА 2.2. Вариант второй
#include <iostream>
```

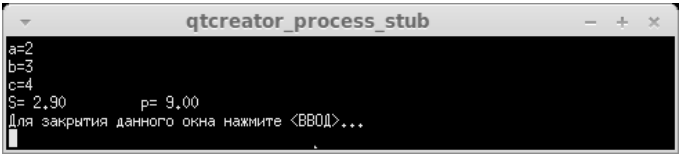


Рис. 2.7. Результаты работы программы к задаче 2.2 (вариант 1)

```
#include <stdio.h>
#include <math.h>
using namespace std;
int main()
{
  float a,b,c,S,r;
  printf("Vvedite a, b, c \n");   //Вывод на экран строки символов.
  scanf("%f%f%f",&a,&b,&c); //Ввод значений.
  r=(a+b+c)/2;
  S=sqrt(r*(r-a)*(r-b)*(r-c));
  printf("S=%5.2f \t p=%5.2f \n",S,2*r); //Вывод результатов.
  return 0;
}
```

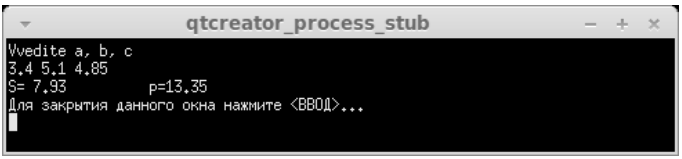


Рис. 2.8. Результаты работы программы к задаче 2.2 (вариант 2)

### 2.9.1 Объектно-ориентированные средства ввода-вывода.

Описание объектов для управления вводом-выводом содержится в заголовочном файле `iostream`. При подключении этого файла с помощью директивы `#include <iostream>` в программе автоматически создаются *объекты-потоки*[8] `cin` для *ввода с клавиатуры* и `cout` для

[8]Поток — виртуальный канал связи, создаваемый в программе для передачи данных

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

*вывода на экран*, а так же операции помещения в поток `<<` и чтения из потока `>>`.

Итак, с помощью объекта `cin` и операции `>>` можно ввести значение любой переменной. Например, если переменная `i` описана как целочисленная, то команда `cin>> i;` означает, что в переменную `i` будет записано некое целое число, введенное с клавиатуры. Если нужно ввести несколько переменных, следует написать `cin>>x>>y>>z;`.

Объект `cout` и операция `<<` позволяют вывести на экран значение любой переменной или текст. Текст необходимо заключать в двойные кавычки, кроме того, допустимо применение специальных символов `\t` и `\n` (таблица 2.11). Запись `cout<<i;` означает вывод на экран значения переменной `i`. А команда `cout<<x<<"\t"<<y;` выведет на экран значения переменных `x` и `y` разделенные символом табуляции.

**Задача 2.3.** Дано трехзначное число. Записать его цифры в обратном порядке и вывести на экран новое число.

Разберем решение данной задачи на конкретном примере. Здесь будут использоваться операции целочисленной арифметики.

Пусть `P=456`. Вычисление остатка от деления числа `P` на `10` даст его последнюю цифру (количество единиц в числе `P`): `456 % 10 =6`.

Операция деления нацело числа `P` на `10` позволит уменьшить количество разрядов и число станет двузначным:

`456 / 10 = 45`.

Остаток от деления полученного числа на `10` будет следующей цифрой числа `P` (количество десятков в числе `P`):

`45 % 10 = 5`.

Последнюю цифру числа `P` (количество сотен) можно найти так:

`456 / 100 = 4`.

Так как в задаче требовалось записать цифры числа `P` в обратном порядке, значит в новом числе будет `6` сотен, `5` десятков и `4` единицы:

`S = 6*100 + 5*10 + 4 = 654`.

Далее приведен текст программы, реализующей данную задачу для любого трехзначного числа.

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
unsigned int P, S;   //Определение целочисленных
                     //переменных без знака.
cout<<"P=";          //Вывод на экран символов P=.
cin>>P;              //Ввод заданного числа P.
S=P%10*100+P/10%10*10+P/100;   //Вычисление нового числа S.
```

```
cout<<"S="<<S<<endl;                    //Вывод на экран символов S=
                                        //и значения переменной S.
return 0;
}
```

**Задача 2.4.** Пусть целочисленная переменная `i` и вещественная переменная `d` вводятся с клавиатуры. Определить размер памяти, отведенной для хранения этих переменных и их суммы, в байтах. Вычислить сколько памяти будет выделено для хранения строки `С Новым Годом!`. Вывести на экран размеры различных типов данных языка `C++` в байтах.

Далее приведён текст программы.

```
#include <iostream>
using namespace std;
int main()
{
int i;      //Определение целочисленной переменной.
double d;   //Определение вещественной переменной.

cout<<"i= "; cin>>i;   //Ввод переменной i.
cout<<"d= "; cin>>d;   //Ввод переменной d.
//Размер памяти, отведенной под переменную i.
cout<<"Размер i: "<<sizeof i<<"\n";
//Размер памяти, отведенной под переменную d.
cout<<"Размер d: "<<sizeof d<<"\n";
//Размер памяти, отведенной под значение выражения i+d.
cout<<"Размер i+d: "<<sizeof (i+d)<<"\n";
cout<<"Размер строки <С Новым Годом!>:";
//Размер памяти, отведенной под строку.
cout<<sizeof "С Новым годом!"<<"\n";
//Вычисление размеров различных типов данных:
cout<<"Размер char: "<<sizeof (char)<<"\n";
cout<<"Размер int: "<<sizeof (int)<<"\n";
cout<<"Размер short int: "<<sizeof (short int)<<"\n";
cout<<"Размер long int: "<<sizeof (long int)<<"\n";
cout<<"Размер long long int:";
cout<<sizeof (long long int)<<"\n";
cout<<"Размер float: "<<sizeof (float)<<"\n";
cout<<"Размер double: "<<sizeof (double)<<"\n";
cout<<"Размер long double: "<<sizeof (long double)<<"\n";
return 0;
}
```

Результаты работы программы[9]

[9]Обратите внимание, что при использовании кодировки utf-16 один кириллический символ занимает в памяти занимает 2 байта.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
i= 23
d= 45.76
Размер i: 4
Размер d: 8
Размер i+d: 8
Размер <С Новым годом!>:26
Размер char: 1
Размер int: 4
Размер short int: 2
Размер long int: 4
Размер long long int:8
Размер float: 4
Размер double: 8
Размер long double: 12
```

## 2.10 Задачи для самостоятельного решения

### 2.10.1 Ввод-вывод данных. Операция присваивания.

Разработать программу на языке C++. Все входные и выходные данные в задачах — *вещественные числа*. Для ввода и вывода данных использовать функции `scanf` и `printf`.

1. Даны катеты прямоугольного треугольника $a$ и $b$. Найти гипотенузу $c$ и углы треугольника $\alpha$, $\beta$, $\chi$.
2. Известна гипотенуза $c$ и прилежащий угол $\alpha$ прямоугольного треугольника. Найти площадь треугольника $S$ и угол $\beta$.
3. Известна диагональ квадрата $d$. Вычислить площадь $S$ и периметр $P$ квадрата.
4. Дан диаметр окружности $d$. Найти ее длину $L$ и площадь круга $S$.
5. Даны три числа — $a$, $b$, $c$. Найти их среднее арифметическое и среднее геометрическое.
6. Даны катеты прямоугольного треугольника $a$ и $b$. Найти его гипотенузу $c$ и периметр $P$.
7. Дан длина окружности $L$. Найти ее радиус $R$ и площадь круга $S$.
8. Даны два ненулевых числа $a$ и $b$. Найти сумму $S$, разность $R$, произведение $P$ и частное $d$ их квадратов.
9. Поменять местами содержимое переменных $A$ и $B$ и вывести новые значения $A$ и $B$.
10. Точки $A$ и $B$ заданы координатами на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$. Найти длину отрезка $AB$.

11. Заданы два катета прямоугольного треугольника $a$ и $b$. Вычислить его площадь $S$ и периметр $P$.
12. Даны переменные $A$, $B$, $C$. Изменить их значения, переместив содержимое $A$ в $B$, $B$ — в $C$, $C$ — в $A$, и вывести новые значения переменных $A$, $B$, $C$.
13. Известна диагональ ромба $d$. Вычислить его площадь $S$ и периметр $P$.
14. Найти значение функции $y = 4 \cdot (x+1)^3 + 5 \cdot (x-1)^5 + 2$ и ее производной при заданном значении $x$.
15. Даны два ненулевых числа $a$ и $b$. Найти сумму $S$, разность $R$, произведение $P$ и частное $D$ их модулей.
16. Известны координаты вершин квадрата $ABCD$: $A(x_1, y_1)$ и $C(x_2, y_2)$. Найти его площадь $S$ и периметр $P$.
17. Даны длины сторон прямоугольника $a$ и $b$. Найти его площадь $S$ и периметр $P$.
18. Известно значение периметра $P$ равностороннего треугольника. Вычислить его площадь $S$.
19. Задан периметр квадрата $P$. Вычислить сторону квадрата $a$, диагональ $d$ и площадь $S$.
20. Дана сторона квадрата $a$. Вычислить периметр квадрата $P$, его площадь $S$ и длину диагонали $d$.
21. Три точки заданы координатами на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$ и $C(x_3, y_3)$. Найти длины отрезков $AB$ и $BC$.
22. Даны переменные A, B, C. Изменить их значения, переместив содержимое $A$ в $C$, $C$ — в $B$, $B$ — в $A$, и вывести новые значения переменных $A$, $B$, $C$.
23. Даны числа — $a_1$, $a_2$, $a_3$, $a_4$, $a_5$. Найти их среднее арифметическое и среднее геометрическое значения.
24. Найти значение функции $y = \frac{3}{2} \cdot (x+3)^4 - \frac{1}{5} \cdot (x-1)^5$ и ее производной при заданном значении $x$.
25. Точки $A$ и $B$ заданы координатами в пространстве: $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$. Найти длину отрезка $AB$.

### 2.10.2 Операции целочисленной арифметики.

Разработать программу на языке C++. Все входные данные в задачах — целые числа. Для ввода и вывода данных использовать объектно-ориентированные средства ввода-вывода.

1. Расстояние $L$ задано в сантиметрах. Найти количество полных метров в нем и остаток в сантиметрах.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

2. Масса $M$ задана в килограммах. Найти количество полных тонн в ней и остаток в килограммах.
3. Дан размер файла $B$в байтах. Найти количество полных килобайтов, которые занимает данный файл и остаток в байтах.
4. Дано двузначное число. Вывести на экран количество десятков и единиц в нем.
5. Дано двузначное число. Найти сумму его цифр.
6. Дано двузначное число. Найти произведение его цифр.
7. Дано двузначное число. Вывести число, полученное при перестановке цифр исходного числа.
8. Дано трехзначное число. Определить сколько в нем единиц, десятков и сотен.
9. Дано трехзначное число. Найти сумму его цифр.
10. Дано трехзначное число. Найти произведение его цифр.
11. Дано трехзначное число. Вывести число, полученное при перестановке цифр сотен и десятков исходного числа.
12. Дано трехзначное число. Вывести число, полученное при перестановке цифр сотен и единиц исходного числа.
13. Дано трехзначное число. Вывести число, полученное при перестановке цифр десятков и единиц исходного числа.
14. С начала суток прошло N секунд. Найти количество полных минут, прошедших с начала суток и остаток в секндах.
15. С начала суток прошло N секунд. Найти количество полных часов, прошедших с начала суток и остаток в секндах.
16. Дано двузначное число $\alpha \leq 88$. Вывести на экран число, которое получится если каждую цифру числа $a$ увеличить на единицу.
17. Дано двузначное число $\alpha \geq 22$. Вывести на экран число, которое получится если каждую цифру числа $a$ уменьшить на единицу.
18. Расстояние $L$ задано в метрах. Найти количество полных километров в нем и остаток в метрах.
19. Масса $M$ задана в граммах. Найти количество полных килограммов в ней и остаток в граммах.
20. Размер файла $B$ дан в килобайтах. Найти количество полных мегабайтов, которые занимает данный файл и остаток в килобайтах.
21. Расстояние $L$ задано в дециметрах. Найти количество полных метров в нем и остаток в сантиметрах.
22. С начала года прошло $K$ дней. Найти количество полных недель, прошедших с начала года и осток в днях.
23. С начала года прошло $K$ часов. Найти количество полных дней, прошедших с начала года и осток в часах.

24. Дано двузначное число $a \leq 44$. Вывести на экран число, которое получится если удвоить каждую цифру числа $a$.
25. Дано двузначное число $\alpha \geq 22$. Вывести на экран число, которое получится если каждую цифру числа $a$ уменьшить вдвое.

### 2.10.3 Встроенные математические функции

Разработать программу на языке `C++`. Все входные и выходные данные в задачах — *вещественные числа*. Для ввода и вывода данных использовать функции `scanf` и `printf`.

Вычислить значение выражения $y = f(x)$ при заданном значении $x$. Варианты заданий представлены в таблице 2.12.

Таблица 2.12: Задачи для самостоятельного решения

| № | **Выражение** $f(x)$ |
|---|---|
| 1 | $\sqrt[7]{x^2 + 2.7 \cdot \pi \cdot \cos\sqrt{\lvert x^3\rvert} - 2} + e^x$ |
| 2 | $\mathrm{tg}^4 x + \sin^2\dfrac{\pi}{x} - e^{2x^2+3.6x-1}$ |
| 3 | $\left\lvert \lvert x^4 - \cos x\rvert - \sqrt[9]{1+\sqrt{x^6}}\right\rvert + \sin^3\dfrac{\pi}{e^x+1}$ |
| 4 | $\log_4\lvert e^x - 4\rvert - \sqrt[7]{\left\lvert\dfrac{2\cdot x}{3.21+\cos^2\frac{\pi}{7}}\right\rvert}$ |
| 5 | $\sqrt[3]{\sqrt{\lvert x\rvert}} + \lvert \mathrm{ctg}^2 x + \dfrac{e^x}{2\cdot\pi} - x^3\rvert$ |
| 6 | $x^5 + \log_3^2(3x^2+5) + \sqrt[9]{(\pi - 6x^2)^2}$ |
| 7 | $\dfrac{1-\log\lvert x - \cos(2x-\pi)\rvert}{6 + x^{4x-1}} + \sqrt[5]{x^3}$ |
| 8 | $e^{x+\frac{\pi}{3}} + \sqrt[3]{tg\left\lvert\dfrac{x^5}{x^2+13.22}\right\rvert} + \cos^3 x$ |
| 9 | $x^{1+\frac{3\cdot\pi}{4}} - 3x^3 - \sqrt[5]{(x+1)^4 + \lg\left\lvert\dfrac{x}{x+1}\right\rvert}$ |
| 10 | $\sqrt[5]{x^3 + \cos\sqrt{\lvert x^3\rvert}} + \dfrac{e^x}{\cos(3\cdot x + \frac{\pi}{15})}$ |
| 11 | $e^{2x} + \sqrt[5]{ctg\dfrac{(x-\pi)^9}{x^4+3.4}} + \sin^2 6.2x$ |
| 12 | $\sqrt[5]{(x+tga)^2} - \dfrac{1-\ln\lvert e^x + \cos\frac{\pi}{8}\rvert}{2}$ |

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Таблица 2.12 — продолжение

| № | **Выражение** $f(x)$ |
|---|---|
| 13 | $\log(e^x+27)-\sqrt{\left\lvert x^3+\dfrac{\sqrt[5]{x^7}+14}{\sin 5x+5.1\cdot\pi}\right\rvert}$ |
| 14 | $\ln\lvert\cos(x-2\cdot\pi)\rvert-\sqrt[3]{1+\dfrac{e^x}{\sin x-3}}$ |
| 15 | $\sqrt{\left\lvert x^3+\dfrac{\sqrt[3]{x^4}-1}{\sin x+\pi+e^x}\right\rvert}$ |
| 16 | $\sqrt[3]{\dfrac{1+3\cdot\pi}{1+x^2}}+\lvert\mathrm{arctg}^2 x^3\rvert$ |
| 17 | $\mathrm{tg}^2\lvert x\rvert+3^{2x^2-e^x}+\dfrac{\sqrt[7]{x^2}}{\cos^2\pi x}$ |
| 18 | $x^4-\sqrt[5]{\pi-\sqrt{\lvert x^3\rvert}+\sin^2\dfrac{x}{x^2+1}}$ |
| 19 | $\log(e^x+6)-\sqrt[3]{(x-4)^2+1.47\sin\sqrt{\lvert\pi\cdot x\rvert}}$ |
| 20 | $\dfrac{x^5}{\sin\lvert x-7\rvert}+\log^2(x^2+2.5)-\sqrt[3]{(\pi-6.1x^2)^2}$ |
| 21 | $\mathrm{ctg}^2\dfrac{x\cdot\pi}{3}-\left(\sqrt{\lvert x\rvert}-3.4\right)^{x^2-10}+\ln(x^2+3)$ |
| 22 | $\left\lvert\log_5\lvert x^3-e^x\rvert-\sqrt[3]{\dfrac{2x}{\cos(x+1.23\cdot\pi)}}\right\rvert$ |
| 23 | $\left\lvert\lvert\cos\dfrac{\pi}{7}-e^x\rvert-\sqrt[7]{2+\sqrt{x^5}}\right\rvert+\ln\dfrac{x^4+1}{6}$ |
| 24 | $\log(x^2+2)-\sin^2 x+\sqrt[5]{2-\sqrt{\lvert x\rvert}+\sin\dfrac{\pi}{e^x+1}}$ |
| 25 | $\log_2 e^x-\cos\dfrac{x}{\pi}+\sqrt[3]{\dfrac{\lvert tg(2x)\rvert}{2.6+x^2+x^3}}$ |

# Глава 3

# Операторы управления

В этой главе описаны основные операторы языка C++: условный оператор `if`, оператор выбора `switch`, операторы цикла `while`, `do...while` и `for`. Изложена методика составления алгоритмов с помощью блок-схем. Приводится большое количество примеров составления программ различной сложности.

## 3.1 Основные конструкции алгоритма

При разработке простейших программ несложно перейти от словесного описания к написанию программы. Однако большинство реально разрабатываемых программ довольно сложные и созданию программы предшествует разработка алгоритма[1]. *Алгоритм* — это чёткое описание последовательности действий, которые необходимо выполнить, для того чтобы при соответствующих исходных данных получить требуемый результат. Одним из способов представления алгоритма является *блок-схема*. При составлении блок-схемы все этапы решения задачи изображаются с помощью различных геометрических фигур. Эти фигуры называют блоками и, как правило, сопровождают надписями. Последовательность выполнения этапов указывают при помощи стрелок, соединяющих эти блоки. Типичные этапы решения задачи изображаются следующими геометрическими фигурами:

- блок начала-конца (рис. 3.1). Надпись внутри блока: «начало» («конец»);

[1] От *algorithmi*, *algorismus*, первоначально латинская транслитерация имени математика аль-Хорезми.

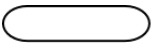

Рис. 3.1. Блок начала-конца алгоритма

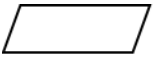

Рис. 3.2. Блок ввода-вывода данных

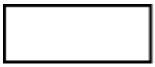

Рис. 3.3. Арифметический блок

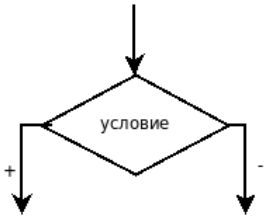


Рис. 3.4. Условный блок

- блок ввода-вывода данных (рис. 3.2). Надпись внутри блока: ввод (вывод или печать) и список вводимых (выводимых) переменных;
- блок решения или арифметический (рис. 3.3). Внутри блока записывается действие, вычислительная операция или группа операций;
- условный блок (рис. 3.4). Логическое условие записывается внутри блока. В результате проверки условия осуществляется выбор одного из возможных путей (ветвей) вычислительного процесса.

Рассмотренные блоки позволяют описать три *основные конструкции алгоритма*: линейный процесс, разветвляющийся процесс и циклический процесс.

*Линейный процесс* это конструкция, представляющая собой последовательное выполнение двух или более операторов (рис. 3.5). *Разветвляющийся процесс* задает выполнение одного или другого оператора в зависимости от выполнения условия (рис. 3.6). *Циклический процесс* задает многократное выполнение оператора или группы операторов (рис. 3.7).

Не трудно заметить, что каждая из основных конструкций алгоритма имеет один вход и один выход. Это позволяет вкладывать конструкции друг в друга произвольным образом и составлять алгоритмы для решения задач любой сложности.

Одним из важных понятий при написании программ на C(C++) является понятие составного оператора.

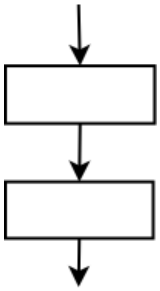

Рис. 3.5. Линейный процесс

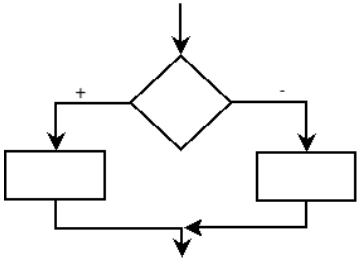


Рис. 3.6. Разветвляющийся процесс

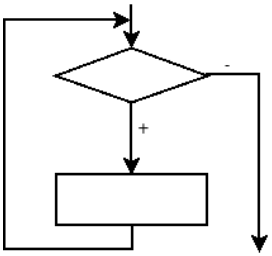


Рис. 3.7. Циклический процесс

## 3.2 Составной оператор

*Составной оператор* — это группа операторов, отделенных друг от друга точкой с запятой, начинающихся с открывающей фигурной скобки { и заканчивающихся закрывающейся фигурной скобкой }:

```
{
  оператор_1;
  ...
  оператор_n;
}
```

Транслятор воспринимает составной оператор как одно целое.

Рассмотрим операторы языка C++, реализующие основные конструкции алгоритма.

## 3.3 Условные операторы

Одна из основных конструкций алгоритма – *разветвляющийся процесс*. Он реализован в языке C++ двумя условными операторами: `if` и `switch`. Рассмотрим каждый из них.

### 3.3.1 Условный оператор

При решении большинства задач порядок вычислений зависит от определенных условий, например, от исходных данных или от промежуточных результатов, полученных на предыдущих шагах программы. Для организации вычислений в зависимости от какого-либо усло-

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

вия в C++ предусмотрен *условный оператор* `if`, который в общем виде записывается следующим образом:

```
if (условие) оператор_1; else оператор_2;
```

где `условие` это логическое (или целое) выражение, переменная или константа.

Работает условный оператор следующим образом. Сначала вычисляется значение выражения, записанного в виде условия. Если оно не равно нулю, т.е. имеет значение истина (`true`), выполняется `оператор_1`. В противном случае, когда выражение равно нулю, т.е. имеет значение ложь (`false`), то — `оператор_2`. Алгоритм, который реализован в условном операторе `if`, представлен на рис. 3.8.

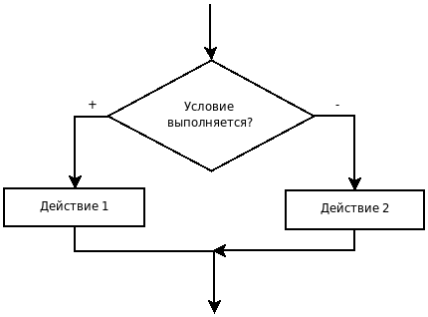


Рис. 3.8. Алгоритм условного оператора `if ... else`

Например, чтобы сравнить значения переменных `a` и `b` нужно написать следующий программный код:

```
cin>>a; cin>>b;
if (a==b) cout<<"a равно b";
else cout<<"a не равно b";
```

**Внимание!** Не путайте знак проверки равенства `==` и оператор присваивания `=`. Например, в записи `if (a=0) b=1;` синтаксической ошибки нет. Операция присваивания `a=0` формирует результат и его значение проверяется в качестве условия. В данном примере присваивание `b=1` не будет выполнено никогда, так как переменная `a` всегда будет принимать значение равное нулю, то есть ложь. Верная запись: `if (a==0) b=1;`.

**Внимание!** Если в задаче требуется, чтобы в зависимости от значения условия выполнялся не один оператор, а несколько, их необхо-

димо заключать в фигурные скобки, как составной оператор. В этом случае компилятор воспримет группу операторов как один:

```
if (условие)
{
  оператор_1;
  оператор_2;
    ...
}
else
{
  оператор_3;
  оператор_4;
    ...
}
```

Альтернативная ветвь `else` в условном операторе может отсутствовать, если в ней нет необходимости:

```
if (условие) оператор;
```

или

```
if (условие)
{
  оператор_1;
  оператор_2;
  ...
}
```

В таком «усеченном» виде условный оператор работает так: `оператор` (группа операторов) либо выполняется, либо пропускается, в зависимости от значения выражения представляющего условие. Алгоритм этого условного процесса представлен на рис. 3.9.

Пример применения условного оператора, без альтернативной ветви `else` может быть таким:

```
if (условие)
cin>>a; cin>>b;
c=0;
//Значение переменной c изменяется
//только при условии, что a не равно b
if (a!=b) c=a+b;
cout<<"c="<<c;
```

Условные операторы могут быть вложены друг в друга. При вложениях условных операторов всегда действует правило: альтернатива `else` считается принадлежащей ближайшему `if`. Например, в записи

```
if (условие_1) if (условие_2) оператор_А; else оператор_Б;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

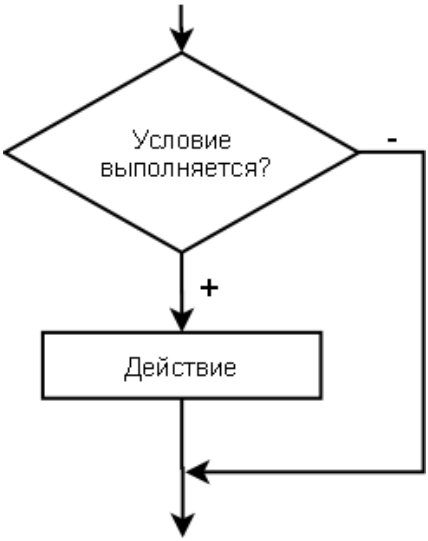


Рис. 3.9. Алгоритм условного оператора `if`

`оператор_Б` относится к `условию_2`, а в конструкции

```
if (условие_1) {if (условие_2) оператор_А;}
else оператор_Б;
```

он принадлежит оператору `if` с `условием_1`.

Рассмотрим несколько задач с применением условных процессов.

**Задача 3.1.** Дано вещественное число $x$. Для функции, график которой приведен на рис. 3.10 вычислить $y = f(x)$.

Аналитически функцию представленную на рис. 3.10 можно записать так:

$$y(x) = \begin{cases} 4, & x \leqslant -2 \\ 1, & x \geqslant 1 \\ x^2, & -2 < x < 1 \end{cases}$$

Составим словесный алгоритм решения этой задачи:

1. Начало алгоритма.
2. Ввод числа $x$ (аргумент функции).
3. Если значение $x$ меньше либо равно -2, то переход к п. 4, иначе переход к п. 5.

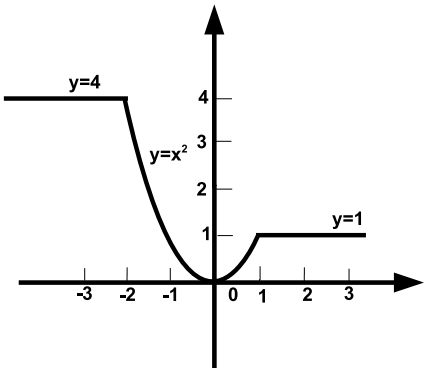


Рис. 3.10. Графическое представление задачи 3.1

4. Вычисление значения функции: $y = 4$, переход к п. 8.
5. Если значение $x$ больше либо равно 1, то переход к п. 6, иначе переход к п. 7.
6. Вычисление значения функции: $y = 1$, переход к п. 8.
7. Вычисление значения функции: $y = x^2$.
8. Вывод значений аргумента $x$ и функции $y$.
9. Конец алгоритма.

Блок-схема соответствующая описанному алгоритму представлена на рис. 3.11.

Текст программы на языке C++ будет иметь вид:

```
#include <iostream>
using namespace std;
int main()
{
  float X,Y;
  cout<<"X="; cin>>X;
  if (X<=-2) Y=4;
  else if (X>=1) Y=1;
  else Y=X*X;
  cout <<"Y=" <<Y<< endl;
  return 0;
}
```

**Задача 3.2.** Даны вещественные числа $x$ и $y$. Определить принадлежит ли точка с координатами $(x; y)$ заштрихованной области (рис. 3.12).

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

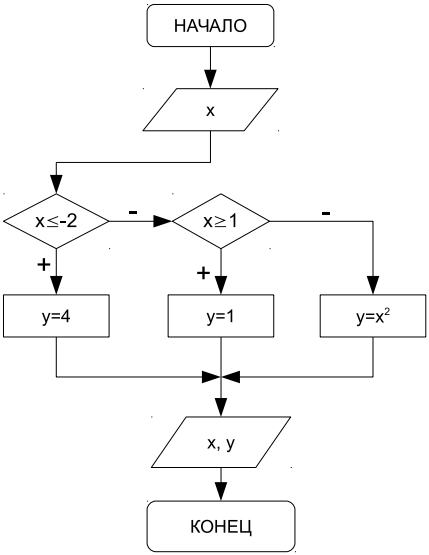


Рис. 3.11. Блок-схема алгоритма решения задачи 3.1

Как показано на рис. 3.12 плоскость ограничена линиями $x = -1$, $x = 3$, $y = -2$ и $y = 4$. Значит точка с координатами $(x; y)$ будет принадлежать этой плоскости, если будут выполняться следующие условия: $x \geq -1$, $x \leq 3$, $y \geq -2$ и $y \leq 4$. Иначе точка лежит за пределами плоскости.

Блок-схема, описывающая алгоритм решения данной задачи представлена на рис. 3.13.

Текст программы к задаче 3.2:

```
#include <iostream>
using namespace std;
int main()
{float X,Y;
cout<<"X="; cin>>X;
cout<<"Y="; cin>>Y;
if (X>=-1 && X<=3 && Y>=-2 && Y<=4)
  cout <<"Точка принадлежит плоскости"<< endl;
else
  cout<<"Точка не принадлежит плоскости"<<endl;
return 0;
}
```

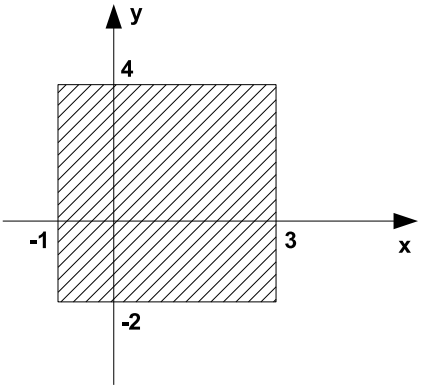


Рис. 3.12. Графическое представление задачи 3.2

**Задача 3.3.** Даны вещественные числа $x$ и $y$. Определить принадлежит ли точка с координатами $(x;\ y)$ заштрихованной области (рис. 3.14).

Составим уравнения линий, ограничивающих заданные плоскости. В общем виде уравнение прямой проходящей через точки с координатами $(x_1, y_1)$ и $(x_2, y_2)$ имеет вид:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}$$

.

Треугольник в первой координатной области ограничен линиями, проходящими через точки:

1. (0, 1) — (4, 3);
2. (4, 3) — (5, 1);
3. (5, 1) — (0, 1).

Следовательно, уравнение первой линии

$$\frac{x-0}{4-0} = \frac{y-1}{3-1} \Rightarrow \frac{x}{4} = \frac{y-1}{2} \Rightarrow y = 1 + \frac{1}{2} \cdot x,$$

уравнение второй линии

$$\frac{x-4}{5-4} = \frac{y-3}{1-3} \Rightarrow x - 4 = \frac{y-3}{-2} \Rightarrow -2 \cdot x + 8 = y - 3 \Rightarrow y = -2 \cdot x + 11$$

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

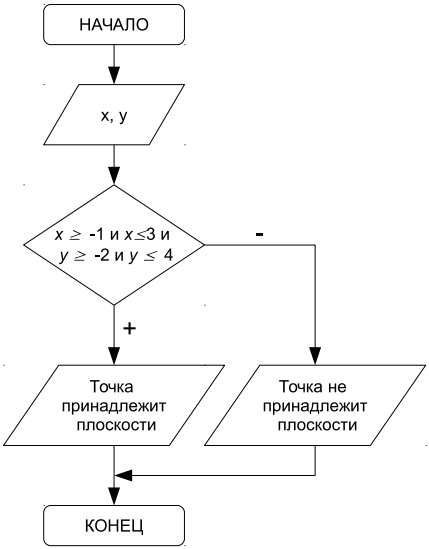


Рис. 3.13. Алгоритм решения задачи 3.2

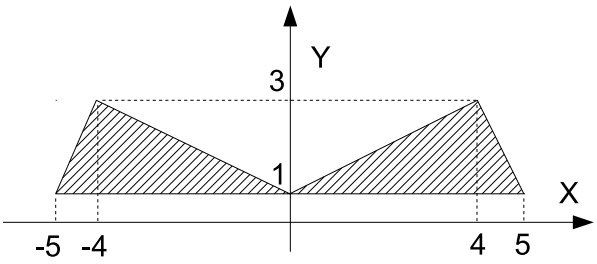


Рис. 3.14. Графическое представление задачи 3.3

и уравнение третьей линии $y = 1$.

Линии, которые формируют треугольник во второй координатной области, проходят через точки:

1. (0, 1) — (-4, 3);
2. (-4, 3) — (-5, 1);
3. (-5, 1) — (0, 1);

Следовательно, уравнение первой линии

$$\frac{x-0}{-4-0}=\frac{y-1}{3-1}\Rightarrow\frac{x}{-4}=\frac{y-1}{2}\Rightarrow y=1-\frac{1}{2}\cdot x,$$

уравнение второй линии

$$\frac{x+4}{-5+4}=\frac{y-3}{1-3}\Rightarrow\frac{x+4}{-1}=\frac{y-3}{-2}\Rightarrow -2\cdot x-8=-y+3\Rightarrow y=2\cdot x+11$$

и уравнение третьей линии $y=1$.

Таким образом, условие попадания точки в заштрихованную часть плоскости имеет вид:

$$\begin{cases} y\leqslant 1+\frac{1}{2}\cdot x \\ y\leqslant -2\cdot x+11 \\ y\geqslant 1 \end{cases} \quad \text{или} \quad \begin{cases} y\leqslant 1-\frac{1}{2}\cdot x \\ y\leqslant 2\cdot x+11 \\ y\geqslant 1 \end{cases}.$$

Далее приведен текст программы для решения задачи 3.3.

```
#include <iostream>
using namespace std;
int main()
{
float X,Y;
cout<<"X="; cin>>X;
cout<<"Y="; cin>>Y;
if ((Y<=1+(float)1/2*X && Y<=-2*X+11 && Y>=1) ||
            (Y<=1-(float)1/2*X && Y<=2*X+11 && Y>=1))
  cout <<"Точка принадлежит плоскости"<< endl;
else
  cout<<"Точка принадлежит плоскости"<<endl;
return 0;
}
```

**Задача 3.4.** Написать программу решения квадратного уравнения $ax^2+bx+c=0$.

Исходные данные: вещественные числа $a$, $b$ и $c$ — коэффициенты квадратного уравнения.

Результаты работы программы: вещественные числа $x1$ и $x2$ — корни квадратного уравнения либо сообщение о том, что корней нет.

Вспомогательные переменные: вещественная переменная $d$, в которой будет храниться дискриминант квадратного уравнения.

Составим словесный алгоритм решения этой задачи.

1. Начало алгоритма
2. Ввод числовых значений переменных $a$, $b$ и $c$.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

3. Вычисление значения дискриминанта $d$ по формуле $d = b^2 - 4ac$ .
4. Если $d < 0$, то переход к п.5, иначе переход к п.6.
5. Вывод сообщения `"Корней нет"` и переход к п.8.
6. Вычисление корней $x1 = \frac{-b+\sqrt{d}}{2a}$ и $x2 = \frac{-b-\sqrt{d}}{2a}$ .
7. Вывод значений $x1$ и $x2$ на экран
8. Конец алгоритма.

Блок-схема, соответствующая этому описанию представлена на рис. 3.15.

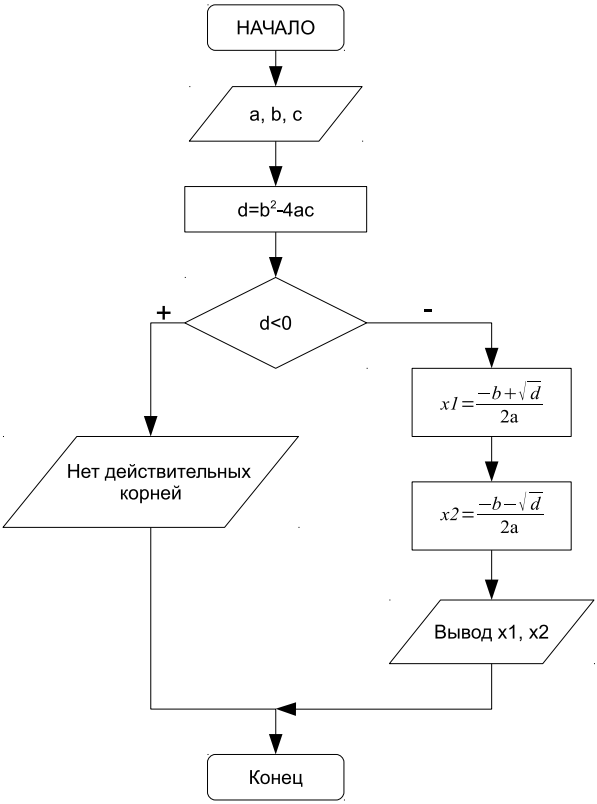


Рис. 3.15. Алгоритм решения квадратного уравнения

Текст программы, которая реализует решение квадратного уравнения:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
float a,b,c,d,x1,x2;
//Ввод значений коэффициентов квадратного уравнения.
cout<<"a=";cin>>a;
cout<<"b=";cin>>b;
cout<<"c=";cin>>c;
d=b*b-4*a*c;      //Вычисление дискриминанта.
if (d<0)
//Если дискриминант отрицательный,
     //то вывод сообщения, о том что корней нет,
  cout<<"Нет вещественных корней";
else
{
//иначе вычисление корней
  x1=(-b+sqrt(d))/2/a;
  x2=(-b-sqrt(d))/(2*a);
//и вывод их значений.
  cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
}
return 0;
}
```

**Задача 3.5.** Составить программу нахождения действительных и комплексных корней квадратного уравнения $ax^2 + bx + c = 0$.

Исходные данные: вещественные числа $a$, $b$ и $c$ — коэффициенты квадратного уравнения.

Результаты работы программы: вещественные числа $x1$ и $x2$ — действительные корни квадратного уравнения либо $x1$ и $x2$ — действительная и мнимая части комплексного числа.

Вспомогательные переменные: вещественная переменная $d$, в которой будет храниться дискриминант квадратного уравнения.

Можно выделить следующие этапы решения задачи:

1. Ввод коэффициентов квадратного уравнения $a$, $b$ и $c$.
2. Вычисление дискриминанта d по формуле $d = b^2 - 4ac$ .
3. Проверка знака дискриминанта. Если $d \geqslant 0$, то вычисление действительных корней: $x1 = \frac{-b+\sqrt{d}}{2a}$ и $x2 = \frac{-b-\sqrt{d}}{2a}$ и вывод их на экран. При отрицательном дискриминанте выводится сооб-

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

щение о том, что действительных корней нет, и вычисляются комплексные корни[2] $x1 = \frac{-b}{2a} + i\frac{\sqrt{|d|}}{2a}$, $x2 = \frac{-b}{2a} - i\frac{\sqrt{|d|}}{2a}$.

У обоих комплексных корней действительные части одинаковые, а мнимые отличаются знаком. Поэтому можно в переменной $x1$ хранить действительную часть числа $\frac{-b}{2a}$ , в переменной $x2$ — модуль мнимой части $\frac{\sqrt{|d|}}{2a}$ , а в качестве корней вывести $x1 + i \cdot x2$ и $x1 - i \cdot x2$.

На рис. 3.16 изображена блок-схема решения задачи. Блок 1 предназначен для ввода коэффициентов квадратного уравнения. В блоке 2 осуществляется вычисление дискриминанта. Блок 3 осуществляет проверку знака дискриминанта, если дискриминант отрицателен, то корни комплексные, их расчет происходит в блоке 4 (действительная часть корня записывается в переменную $x1$, модуль мнимой — в переменную $x2$), а вывод — в блоке 5 (первый корень $x1 + i \cdot x2$, второй — $x1 - i \cdot x2$). Если дискриминант положителен, то вычисляются действительные корни уравнения (блок 6) и выводятся на экран (блок 7).

Текст программы, реализующей поставленную задачу:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
float a,b,c,d,x1,x2;
cout<<"a=";cin>>a;
cout<<"b=";cin>>b;
cout<<"c=";cin>>c;
d=b*b-4*a*c;
if (d<0)
{//Если дискриминант отрицательный, то вывод соответствующего сообщения.
  cout<<"Нет вещественных корней \n";
  x1=-b/(2*a);   //Вычисление действительной части комплексных корней.
  x2=sqrt(fabs(d))/(2*a); //Вычисление модуля мнимой
                          //части комплексных корней
  //Сообщение о комплексных корнях уравнения вида ax^2+bx+c=0.
  cout<<"Комплексные корни уравнения \n";
  cout<<a<<"x^2+"<<b<<"x+"<<c<<"=0 \n";
  //Вывод значений комплексных корней в виде x1 ± ix2
  if (x2>=0)
  {
```

[2]Комплексные числа записываются в виде $a+ib$, где $a$ — действительная часть комплексного числа, $b$ — мнимая часть комплексного числа, $i$ — мнимая единица $\sqrt{-1}$.

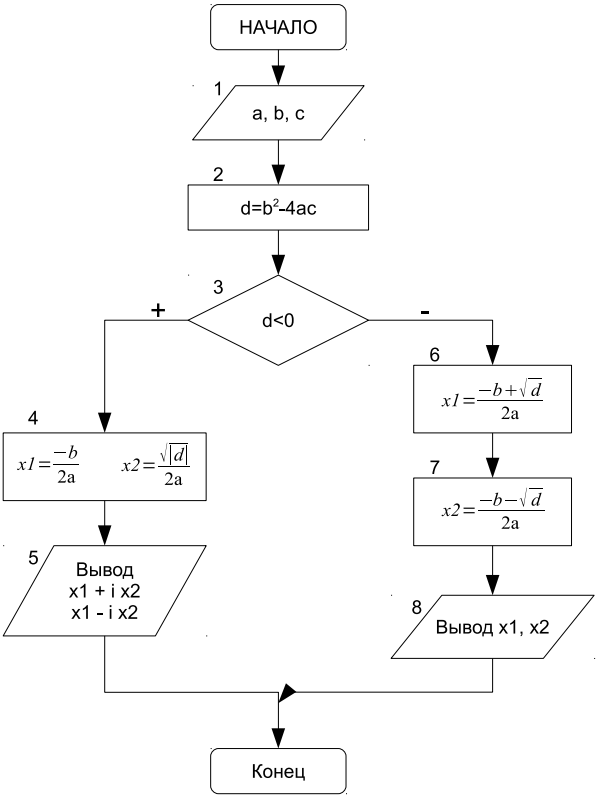


Рис. 3.16. Алгоритм решения задачи 3.5

```
    cout<<x1<<"+"<<x2<<"i\t";
    cout<<x1<<"-"<<x2<<"i\n";
  }
  else
  {
    cout<<x1<<"-"<<fabs(x2)<<"i\t";
    cout<<x1<<"+"<<fabs(x2)<<"i\n";
  }
}
else
{
//Если дискриминант положительный, вычисление действительных корней
//и вывод их на экран.
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
    x1=(-b+sqrt(d))/2/a;
    x2=(-b-sqrt(d))/(2*a);
    cout<<"Вещественные корни уравнения \n";
    cout<<a<<"x^2+"<<b<<"x+"<<c<<"=0 \n";
    cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
}
return 0;
}
```

Результаты работы программы к задаче 3.5 показаны на ниже.

```
a=-5
b=-3
c=-4
Нет вещественных корней
Комплексные корни уравнения
-5x^2+-3x+-4=0
-0.3-0.842615i -0.3+0.842615i
============================
a=2
b=-3
c=1
Вещественные корни уравнения
2x^2+-3x+1=0
X1=1  X2=0.5
```

**Задача 3.6.** Составить программу для решения кубического уравнения $ax^3 + bx^2 + cx + d = 0$.

Кубическое уравнение имеет вид

$$ax^3 + bx^2 + cx + d = 0 \tag{3.1}$$

После деления на $a$ уравнение 3.1 принимает канонический вид:

$$x^3 + rx^2 + sx + t = 0, \tag{3.2}$$

где $r = \frac{b}{a}$, $s = \frac{c}{a}$, $t = \frac{d}{a}$.

В уравнении 3.2 сделаем замену $x = y - \frac{r}{3}$ и получим приведенное уравнение:

$$y^3 + py + q = 0, \tag{3.3}$$

где $p = \frac{3s-r^2}{3}$, $q = \frac{2r^3}{27} - \frac{rs}{3} + t$.

Число действительных корней приведенного уравнения (3.3) зависит от знака дискриминанта (табл. 3.1):

$D = (\frac{p}{3})^3 + (\frac{q}{2})^2$.

Таблица 3.1: Количество корней кубического уравнения

| **Дискриминант** | **Количество действительных корней** | **Количество комплексных корней** |
|---|---|---|
| $D \geqslant 0$ | 1 | 2 |
| $D < 0$ | 3 | - |

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано:

$$\begin{array}{l} y_1 = u + v \\ y_2 = \frac{-u+v}{2} + \frac{u-v}{2} i\sqrt{3} \\ y_3 = \frac{-u+v}{2} - \frac{u-v}{2} i\sqrt{3}, \end{array} \tag{3.4}$$

где $u = \sqrt[3]{\frac{-q}{2} + \sqrt{D}}$, $v = \sqrt[3]{\frac{-q}{2} - \sqrt{D}}$ .

При отрицательном дискриминанте уравнение (3.1) имеет три действительных корня, но они будут вычисляться через вспомогательные комплексные величины. Чтобы избавиться от этого, можно воспользоваться формулами:

$$\begin{array}{l} y_1 = 2\sqrt[3]{\rho}\cos(\frac{\phi}{3}), \\ y_2 = 2\sqrt[3]{\rho}\cos(\frac{\phi}{3} + \frac{2\pi}{3}), \\ y_3 = 2\sqrt[3]{\rho}\cos(\frac{\phi}{3} + \frac{4\pi}{3}), \end{array} \tag{3.5}$$

где $\rho = \sqrt{\frac{-p^3}{27}}$, $\cos(\phi) = \frac{-q}{2\rho}$.

Таким образом, при положительном дискриминанте кубического уравнения (3.3) расчет корней будем вести по формулам (3.4), а при отрицательном — по формулам (3.5). После расчета корней приведенного уравнения (3.3) по формулам (3.4) или (3.5) необходимо по формулам

$$x_k = y_k - \frac{r}{3}, \ k = 1, 2, 3..,$$

перейти к корням заданного кубического уравнения (3.1).

Блок-схема решения кубического уравнения представлена на рис. 3.18.

Описание блок-схемы. В блоке 1 вводятся коэффициенты кубического уравнения, в блоках 2–3 рассчитываются коэффициенты канонического и приведенного уравнений. Блок 4 предназначен для вычисления дискриминанта. В блоке 5 проверяется знак дискриминанта

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

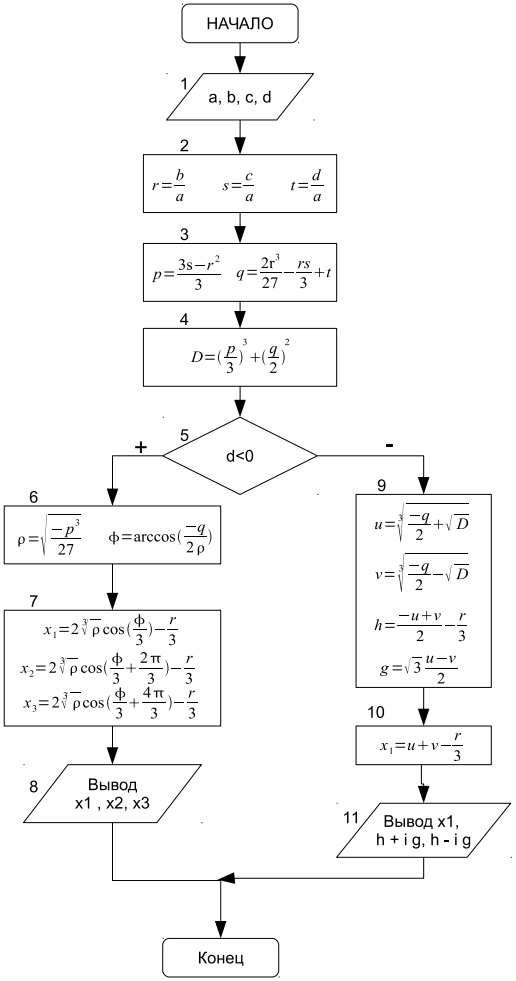


Рис. 3.17. Алгоритм решения кубического уравнения

кубического уравнения. Если он отрицателен, то корни вычисляются по формулам 3.5 (блоки 6–7). При положительном значении дискриминанта расчет идет по формулам 3.4 (блок 9, 10). Блоки 8 и 11 предназначены для вывода результатов на экран.

Текст программы с комментариями приведен ниже[3].

```
#include <iostream>
#include <math.h>
using namespace std;
#define pi 3.14159  //Определение константы
int main()
{
float a,b,c,d,D,r,s,t,p,q,ro,fi,x1,x2,x3,u,v,h,g;
//Ввод коэффициентов кубического уравнения.
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
cout<<"c="; cin>>c;
cout<<"d="; cin>>d;
//Расчет коэффициентов канонического уравнения по формуле 3.2
r=b/a; s=c/a; t=d/a;
//Вычисление коэффициентов приведенного уравнения по формуле 3.3
p=(3*s-r*r)/3; q=2*r*r*r/27-r*s/3+t;
//Вычисление дискриминанта кубического уравнения
D=(p/3)*(p/3)*(p/3)+(q/2)*(q/2);
if (D<0)
{
  //Формулы 3.5
  ro=sqrt((float)(-p*p*p/27));
  fi=-q/(2*ro);
  fi=pi/2-atan(fi/sqrt(1-fi*fi));
  x1=2*pow(ro,(float)1/3)*cos(fi/3)-r/3;
  x2=2*pow(ro,(float)1/3)*cos(fi/3+2*pi/3)-r/3;
  x3=2*pow(ro,(float)1/3)*cos(fi/3+4*pi/3)-r/3;
  cout<<"\n x1="<<x1<<"\t x2="<<x2;
  cout<<"\t x3="<<x3<<"\n";
}
else
{
  //Формулы 3.4
  if (-q/2+sqrt(D)>0) u=pow((-q/2+sqrt(D)),(float)1/3);
  else
  if (-q/2+sqrt(D)<0) u=-pow(fabs(-q/2+sqrt(D)),(float)1/3);
  else u=0;
  if (-q/2-sqrt(D)>0) v=pow((-q/2-sqrt(D)),(float)1/3);
  else
  if (-q/2-sqrt(D)<0) v=-pow(fabs(-q/2-sqrt(D)),(float)1/3);
```

[3]При расчете величин u и v в программе предусмотрена проверка значения подкоренного выражения.

Если $\frac{-q}{2} \mp \sqrt{D} > 0$, то $u = \sqrt[3]{\frac{-q}{2} + \sqrt{D}}$, а $v = \sqrt[3]{\frac{-q}{2} - \sqrt{D}}$.

Если $\frac{-q}{2} \mp \sqrt{D} < 0$, то $u = \sqrt[3]{|\frac{-q}{2} + \sqrt{D}|}$, а $v = \sqrt[3]{|\frac{-q}{2} - \sqrt{D}|}$.

Соответственно, при нулевом значении подкоренного выражения u и v обращаются в ноль

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  else v=0;
    x1=u+v-r/3;   //Вычисление действительного корня
                  //кубического уравнения.
    h=-(u+v)/2-r/3; //Вычисление действительной
    g=(u-v)/2*sqrt((float)3); //и мнимой части комплексных корней
    cout<<"\n x1="<<x1;
    if (x2>=0)
      {
        cout<<x1<<"+"<<x2<<"i\t";
        cout<<x1<<"-"<<x2<<"i\n";
      }
    else
      {
        cout<<x1<<"-"<<fabs(x2)<<"i\t";
        cout<<x1<<"+"<<fabs(x2)<<"i\n";
      }
    }
if (g>=0)
{
cout<<"\t x2="<<h<<"+"<<g<<"i";
cout<<"\t x3="<<h<<"-"<<g<<"i \n";
}
else
{
cout<<"\t x2="<<h<<"-"<<fabs(g)<<"i";
cout<<"\t x2="<<h<<"+"<<fabs(g)<<"i";
}
}
return 0;
}
```

**Задача 3.7.** Заданы коэффициенты $a$, $b$ и $c$ биквадратного уравнения $ax^4 + bx^2 + c = 0$. Найти все его действительные корни.

*Входные данные*: $a$, $b$, $c$.

*Выходные данные*: $x1$, $x2$, $x3$, $x4$.

Для решения биквадратного уравнения необходимо заменой $y = x^2$ привести его к квадратному уравнению $ay^2 + by + c = 0$ и решить это уравнение.

Опишем алгоритм решения этой задачи (рис. 3.18):

1. Ввод коэффициентов биквадратного уравнения $a$, $b$ и $c$ (блок 1).
2. Вычисление дискриминанта уравнения $d$ (блок 2).
3. Если $d < 0$ (блок 3), вывод сообщения, что корней нет (блок 4), а иначе определяются корни соответствующего квадратного уравнения $y1$ и $y2$ (блок 5).
4. Если $y1 < 0$ и $y2 < 0$ (блок 6) , то вывод сообщения, что корней нет (блок 7).

5. Если $y1 \geq 0$ и $y2 \geq 0$ (блок 8), то вычисляются четыре корня по формулам $\pm\sqrt{y_1}, \pm\sqrt{y_2}$ (блок 9) и выводятся значения корней (блок 10).
6. Если условия 4) и 5) не выполняются, то необходимо проверить знак $y1$. Если $y1 \geq 0$ (блок 11), то вычисляются два корня по формуле $\pm\sqrt{y_1}$ (блок 12), иначе (если $y2 \geq 0$) вычисляются два корня по формуле $\pm\sqrt{y_2}$ (блок 13). Вывод вычисленных значений корней (блок 14).

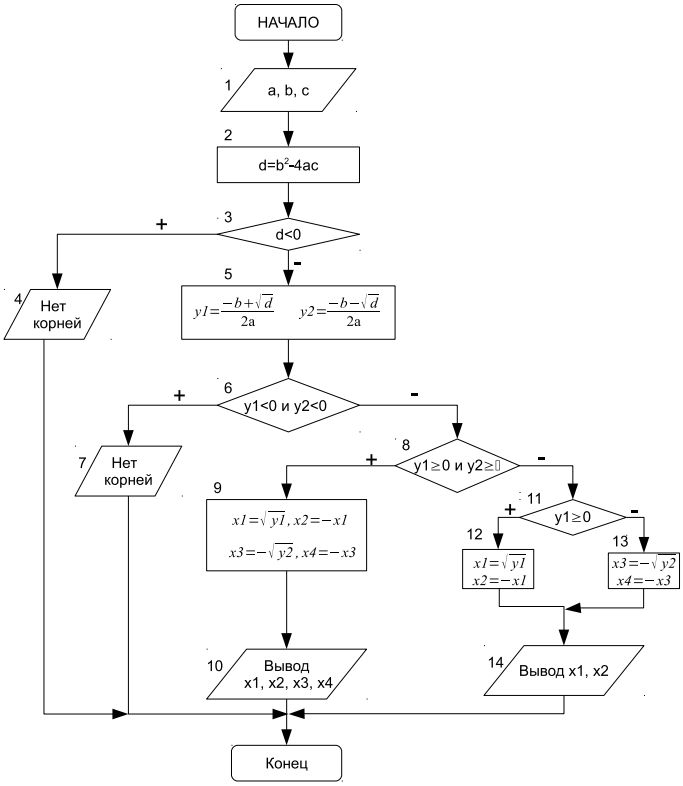


Рис. 3.18. Алгоритм решения биквадратного уравнения

Текст программы решения биквадратного уравнения приведен ниже.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

**Внимание!** Если в условном операторе проверяется двойное условие необходимо применять логические операции `||`, `&&`, `!`. Например, условие «если $y1$ и $y2$ положительны» правильно записать так: `if (y1>=0 && y2>=0)`.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{//Описание переменных:
 //a, b, c – коэффициенты биквадратного уравнения,
 //d – дискриминант,
 //x1, x2, x3, x4 – корни биквадратного уравнения,
 //y1, y2 – корни квадратного уравнения ay^2+by+c=0,
 float a,b,c,d,x1,x2,x3,x4,y1,y2;
 //Ввод коэффициентов уравнения.
 cout<<"a="; cin>>a;
 cout<<"b="; cin>>b;
 cout<<"c="; cin>>c;
 d=b*b-4*a*c;   //Вычисление дискриминанта.
 if (d<0)   //Если дискриминант отрицательный,
            //вывод сообщения «Корней нет».
  cout<<"Нет действительных корней \n";
 else   //Если дискриминант положительный,
 {
 //Вычисление корней соответствующего квадратного уравнения.
   y1=(-b+sqrt(d))/2/a;
   y2=(-b-sqrt(d))/(2*a);
 //Если оба корня квадратного уравнения отрицательные,
  if (y1<0 && y2<0)
 //вывод сообщения «Корней нет»
      cout<<" Нет действительных корней \n";
 //Если оба корня квадратного уравнения положительные,
   else if (y1>=0 && y2>=0)
   {//Вычисление четырех корней биквадратного уравнения
    x1=sqrt(y1);
    x2=-x1;
    x3=sqrt(y2);
    x4=-sqrt(y2);
    //Вывод корней уравнения на экран.
    cout<<"\t X1="<<x1<<"\t X2="<<x2;
    cout<<"\t X3="<<x3<<"\t X4="<<x4<<"\n";
   }
    //Если не выполнились условия
    //1. y1<0 и y2<0
    //2. y1>=0 и y2>=0,
    //то проверяем условие y1>=0.
    else if (y1>=0) //Если оно истинно
    { //вычисляем два корня биквадратного уравнения.
```

```
        x1=sqrt(y1);
        x2=-x1;
        cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
      }
      else
      { //Если условие y1>=0 ложно, то вычисляем два корня
        //биквадратного уравнения
        x1=sqrt(y2);
        x2=-x1;
        cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
      }
  }
return 0;
}
```

Читателю предлагается самостоятельно модифицировать программу таким образом, чтобы она находила все корни (как действительные, так и комплексные) биквадратного уравнения.

### 3.3.2 Оператор варианта

Оператор варианта `switch` необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы:

```
switch (выражение)
{
case значение_1: Операторы_1; break;
case значение_2: Операторы_2; break;
case значение_3: Операторы_3; break;
  ...
case значение_n: Операторы_n; break;
default: Операторы; break;
}
```

Оператор работает следующим образом. Вычисляется значение выражения (оно должно быть целочисленным). Затем выполняются операторы, помеченные значением, совпадающим со значением выражения. То есть, если выражение принимает `значение_1`, то выполняются `операторы_1`. Если выражение принимает `значение_2`, то выполняется `операторы_2` и так далее. Если выражение не принимает ни одно из значений, то выполняются операторы расположенные после ключевого слова `default`.

Альтернативная ветвь `default` может отсутствовать, тогда оператор имеет вид:

```
switch (выражение)
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
{
case значение_1: Операторы_1; break;
case значение_2: Операторы_2; break;
case значение_3: Операторы_3; break;
  ...
case значение_n: Операторы_n; break;
}
```

Оператор `break` необходим для того, чтобы осуществить выход из оператора `switch`. Если оператор `break` не указан, то будут выполняться следующие операторы из списка, не смотря на то, что значение, которым они помечены, не совпадает со значением выражения.

Рассмотрим применение оператора варианта.

**Задача 3.8.** Вывести на печать название дня недели, соответствующее заданному числу $D$, при условии, что в месяце 31 день и 1-е число — понедельник.

Для решения задачи воспользуемся операцией `%`, позволяющей вычислить остаток от деления двух чисел, и условием, что 1-е число — понедельник. Если в результате остаток от деления (обозначим его $R$) заданного числа $D$ на семь будет равен единице, то это понедельник, двойке — вторник, тройке — среда и так далее. Следовательно, при построении алгоритма необходимо использовать семь условных операторов, как показано рис. 3.19. Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта `switch`:

```
#include <iostream>
using namespace std;
int main()
{unsigned int D,R; //Описаны целые положительные числа.
  cout<<"D="; cin>>D;   //Ввод числа от 1 до 31.
  R=D%7;
  switch (R)
  {
    case 1: cout<<"Понедельник \n"; break;
    case 2: cout<<"Вторник \n"; break;
    case 3: cout<<"Среда \n"; break;
    case 4: cout<<"Четверг \n"; break;
    case 5: cout<<"Пятница \n"; break;
    case 6: cout<<"Суббота \n"; break;
    case 0: cout<<"Воскресенье \n"; break;
  }
return 0;
}
```

В предложенной записи оператора варианта отсутствует ветвь `default`. Это объясняется тем, что переменная $R$ может принимать

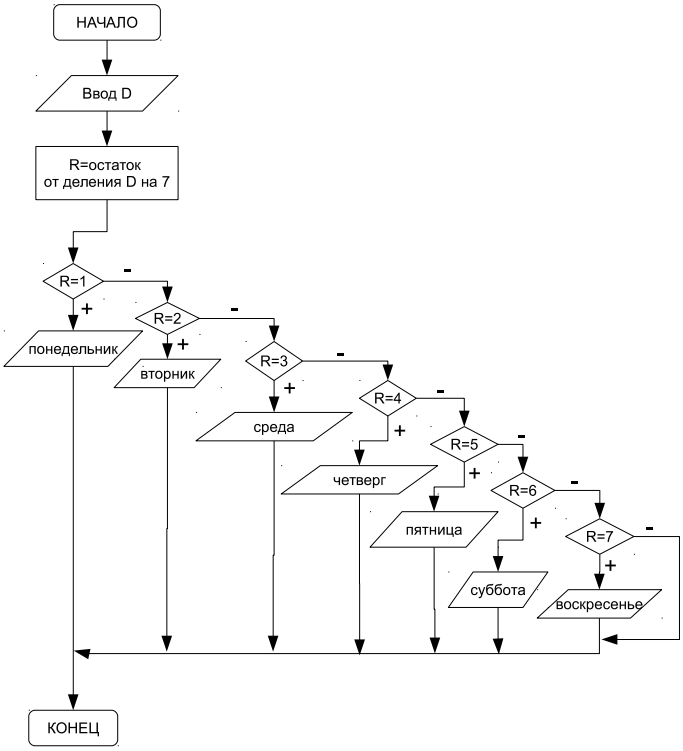


Рис. 3.19. Алгоритм решения задачи 3.8

только одно из указанных значений, т.е. 1, 2, 3, 4, 5, 6 или 0. Однако программа будет работать неправильно, если пользователь введет значение $D$ превышающее 31. Чтобы избежать подобной ошибки лучше сделать дополнительную проверку входных данных:

```
#include <iostream>
using namespace std;
int main()
{
unsigned int D,R;
cout<<"\n D="; cin>>D;
if (D<32)   //Проверка введенного значения.
{
  R=D%7;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  switch (R)
  {
    case 1: cout<<"Понедельник \n"; break;
    case 2: cout<<"Вторник \n"; break;
    case 3: cout<<"Среда \n"; break;
    case 4: cout<<"Четверг \n"; break;
    case 5: cout<<"Пятница \n"; break;
    case 6: cout<<"Суббота \n"; break;
    case 0: cout<<"Воскресенье \n"; break;
  }
}
//Сообщение об ошибке в случае некорректного ввода.
else cout<<"ОШИБКА! \n";
return 0;
}
```

**Задача 3.9.** По заданному номеру месяца $m$ вывести на экран его название.

Для решения данной задачи необходимо проверить выполнение четырех условий. Если заданное число $m$ равно 12, 1 или 2, то это зима, если $m$ попадает в диапазон от 3 до 5, то — весна, лето определяется принадлежностью числа $m$ диапазону от 6 до 8 и, соответственно, при равенстве переменной $m$ 9, 10 или 11 — это осень. Понятно, что область возможных значений переменной $m$ находится в диапазоне от 1 до 12 и если пользователь введет число не входящее в этот интервал, то появится сообщение об ошибке.

```
#include <iostream>
using namespace std;
int main()
{
  unsigned int m;   //Описано целое положительное число.
  cout<<"m="; cin>>m;
  switch (m)
  {
//В зависимости от значения m выводится название месяца.
    case 1: cout<<"Январь \n"; break;
    case 2: cout<<"Февраль \n"; break;
    case 3: cout<<"Март \n"; break;
    case 4: cout<<"Апрель \n"; break;
    case 5: cout<<"Май \n"; break;
    case 6: cout<<"Июнь \n"; break;
    case 7: cout<<"Июль \n"; break;
    case 8: cout<<"Август \n"; break;
    case 9: cout<<"Сентябрь \n"; break;
    case 10:cout<<"Октябрь \n"; break;
    case 11:cout<<"Ноябрь \n"; break;
    case 12:cout<<"Декабрь \n"; break;
```

```
//Если значение переменной m выходит за пределы области
//допустимых значений, то выдается сообщение.
    default: cout<<"ОШИБКА! \n"; break;
  }
return 0;
}
```

## 3.4 Операторы цикла

*Циклический процесс* или просто *цикл* это повторение одних и тех же действий. Последовательность действий, которые повторяются в цикле, называют *телом цикла*. Один проход цикла называют *шагом* или *итерацией*. Переменные, которые изменяются внутри цикла, и влияют на его окончание, называются *параметрами цикла*.

При написании циклических алгоритмов следует помнить следующее. Во-первых, чтобы цикл имел шанс когда-нибудь закончиться, содержимое его тела должно обязательно влиять на условие цикла. Во-вторых, условие должно состоять из корректных выражений и значений, определенных еще до первого выполнения тела цикла.

В C++ для удобства пользователя предусмотрены три оператора, реализующих циклический процесс: `while`, `do...while` и `for`.

### 3.4.1 Оператор цикла с предусловием

На рис. 3.20 изображена блок-схема алгоритма *цикла с предусловием*. Оператор, реализующий этот алгоритм в C++ имеет вид:

```
while (условие) оператор;
```

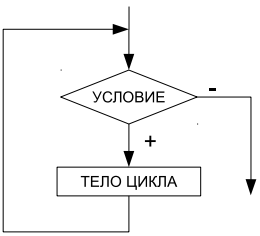


Рис. 3.20. Алгоритм циклической структуры с предусловием

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Работает цикл с предусловием следующим образом. Вычисляется `условие`. Если оно истинно (не равно нулю), выполняется `оператор`. В противном случае цикл заканчивается, и управление передается оператору, следующему за телом цикла. Условие вычисляется перед каждой итерацией цикла. Если при первой проверке выражение равно нулю, цикл не выполнится ни разу. Тип выражения должен быть арифметическим или приводимым к нему.

Если тело цикла состоит более чем из одного оператора, необходимо использовать составной оператор:

```
while (условие)
{
  оператор 1;
  оператор 2;
  ...
  оператор n;
}
```

Рассмотрим пример. Пусть необходимо вывести на экран таблицу значений функции $y = e^{\sin(x)}\cos(x)$ на отрезке $[0;\pi]$ с шагом 0.1. Применив *цикл с предусловием* получим:

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159
using namespace std;
int main()
{
float x, y; //Описание переменных
x=0;          //Присваивание параметру цикла стартового значения
//Цикл с предусловием
while (x<=PI)    //Пока параметр цикла не превышает конечное значение
{ //выполнять тело цикла
  y=exp(sin(x))*cos(x); //Вычислить значение y
  //Вывод на экран пары x и y.
  printf("\t x=%5.2f \t y=%5.4f \n",x,y);
  x+=0.1;  //Изменение параметра цикла
  //(переход к следующему значению x)
} //Конец цикла
return 0;
}
```

В результате работы данного фрагмента программы на экран последовательно будут выводиться сообщения со значениями переменных $x$ и $y$:

```
x= 1.00   y=1.2534
x= 1.10   y=1.1059
```

```
x= 1.20   y=0.9203
x= 1.30   y=0.7011
x= 1.40   y=0.4553
x= 1.50   y=0.1918
x= 1.60   y=-0.0793
x= 1.70   y=-0.3473
x= 1.80   y=-0.6017
x= 1.90   y=-0.8328
x= 2.00   y=-1.0331
x= 2.10   y=-1.1969
x= 2.20   y=-1.3209
x= 2.30   y=-1.4045
x= 2.40   y=-1.4489
x= 2.50   y=-1.4576
x= 2.60   y=-1.4348
x= 2.70   y=-1.3862
x= 2.80   y=-1.3172
x= 2.90   y=-1.2334
x= 3.00   y=-1.1400
x= 3.10   y=-1.0416
```

### 3.4.2 Оператор цикла с постусловием

В цикле с предусловием предварительной проверкой определяется, выполнять тело цикла или нет, до первой итерации. Если это не соответствует логике алгоритма, то можно использовать *цикл с постусловием*. На рис. 3.21. видно, что в этом цикле проверяется, делать или нет очередную итерацию, лишь после завершения предыдущей. Это имеет принципиальное значение лишь на первом шаге, а далее циклы ведут себя идентично.

В C++ *цикл с постусловием* реализован конструкцией

```
do оператор while (условие);
```

здесь `условие` — логическое или целочисленное выражение, или, если тело цикла состоит более чем из одного оператора:

```
do
{
  оператор_1;
  оператор_2;
  ...
  оператор_n;
}
while (условие);
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

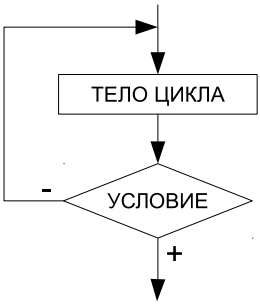


Рис. 3.21. Алгоритм циклической структуры с постусловием

Работает цикл следующим образом. В начале выполняется оператор, представляющий собой тело цикла. Затем вычисляется условие. Если оно истинно (не равно нулю), оператор тела цикла выполняется ещё раз. В противном случае цикл завершается, и управление передаётся оператору, следующему за циклом.

Таким образом, не трудно заметить, что цикл с постусловием всегда будет выполнен хотя бы один раз, в отличие от цикла с предусловием, который может не выполниться ни разу.

Если применить цикл с постусловием для создания программы, которая выводит таблицу значений функции $y = e^{\sin(x)}\cos(x)$ на отрезке $[0;\pi]$ с шагом 0.1, получим:

```
#include <iostream>
#include <stdio.h>
#include <math.h>
#define PI 3.14159
using namespace std;
int main()
{
   float x, y;   //Описание переменных
   x=0; //Присваивание параметру цикла стартового значения
   do   //Цикл с постусловием
   {//Выполнять тело цикла
      y=exp(sin(x))*cos(x);
      printf(" \t x=%5.2f \t y=%5.4f \n",x,y);
      x+=0.1;   //Изменение параметра цикла
   }
   while(x<=PI);   //пока параметр цикла не превышает конечное значение
return 0;
}
```

Результаты работы этой программы будут такими же как на стр. 78.

### 3.4.3 Оператор цикла for с параметром

Кроме того, в C++ предусмотрен *цикл for с параметром*:

```
for (начальные_присваивания;условие;последействие)
оператор;
```

где `начальные_присваивания` — оператор или группа операторов, разделённых запятой[4], применяются для присвоения начальных значений величинам, используемым в цикле, в том числе параметру цикла, и выполняются один раз в начале цикла; целое или логическое `условие` — определяет условие входа в цикл, если условие истинно (не равно нулю), то цикл выполняется; `последействие` — оператор или группа операторов, разделённых запятой, которые выполняются после каждой итерации и служат для изменения параметра цикла; `оператор` — любой оператор языка, представляющий собой тело цикла. `Последействие` или `оператор` должны влиять на условие, иначе цикл никогда не закончится. `Начальные_присваивания`, `выражение` или `приращение` в записи оператора `for` могут отсутствовать, но при этом «точки с запятой» должны оставаться на своих местах.

Опишем алгоритм работы цикла `for`:

1. Выполняются `начальные_присваивания`.
2. Вычисляется `условие`, если оно не равно 0 (`true`), то выполняется переход к п.3. В противном случае выполнение цикла завершается.
3. Выполняется `оператор`.
4. Выполняется оператор `последействие` и осуществляется переход к п.2, опять вычисляется значение `выражения` и т.д.

Понятно, что этот алгоритм представляет собой цикл с предусловием (рис. 3.22).

В дальнейшем, чтобы избежать создания слишком громоздких алгоритмов, в блок-схемах цикл `for` будем изображать, так как показано на рис. 3.23.

В случае если тело цикла состоит более чем из одного оператора, необходимо использовать составной оператор:

[4]Запятая в C++ это операция последовательного выполнения операторов

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

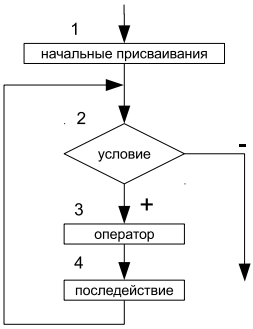


Рис. 3.22. Алгоритм работу цикла с параметром

```
for (начальные_присваивания; условие; приращение)
{
  оператор_1;
  ...
  оператор_n;
}
```

Применение цикла `for` рассмотрим на примере печати таблицы значений функции $y = e^{\sin(x)}\cos(x)$ на отрезке $[0;\pi]$ с шагом 0.1:

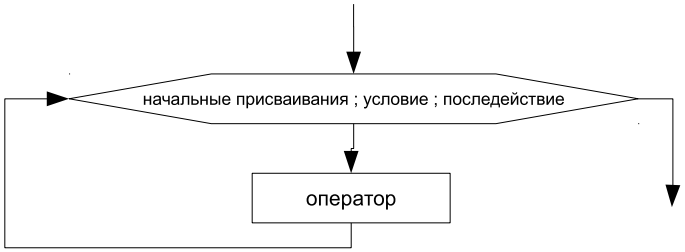


Рис. 3.23. Блок-схема цикла с параметром

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159
using namespace std;
int main()
{
```

```
float x, y;
//Параметру цикла присваивается начальное значение,
//если оно не превышает конечное значение,
//то выполняются операторы тела цикла
//и значение параметра изменяется,
//в противном случае цикл заканчивается.
for (x=0;x<=PI;x+=0.1)
{
  y=exp(sin(x))*cos(x);
  printf("\t x=%5.2f \t y=%5.4f \n",x,y);
}
return 0;
}
```

Программный код выдаст результат представленный на стр. 78.

### 3.4.4 Операторы передачи управления

Операторы передачи управления принудительно изменяют порядок выполнения команд. В C++ таких операторов четыре: `goto`, `break`, `continue` и `return`.

Оператор `goto метка`, где `метка` обычный идентификатор, применяют для безусловного перехода, он передает управление оператору с меткой: `метка: оператор;`[5].

Оператор `break` осуществляет немедленный выход из циклов `while`, `do...while` и `for`, а так же из оператора выбора `switch`. Управление передается оператору, находящемуся непосредственно за циклом или оператором выбора.

Оператор `continue` начинает новую итерацию цикла, даже если предыдущая не была завершена.

Оператор `return выражение` завершает выполнение функции и передает управление в точку ее вызова. Если функция возвращает значение типа `void`, то выражение в записи оператора отсутствует. В противном случае выражение должно иметь скалярный тип.

[5]Обычно применение оператора `goto` приводит к усложнению программы и затрудняет отладку. Он нарушает принцип структурного программирования, согласно которому все блоки, составляющие программу, должны иметь только один вход и один выход. В большинстве алгоритмов применения этого оператора можно избежать

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

## 3.5 Решение задач с использованием циклов

Рассмотрим использование циклических операторов на конкретных примерах.

**Задача 3.10.** Написать программу решения квадратного уравнения $ax^2 + bx + c = 0$. Предусмотреть проверку ввода данных.

Решение квадратного уравнения было подробно рассмотрено в задаче 3.4. Однако алгоритм изображенный на рис. 3.15 не будет работать, если пользователь введет нулевое значение в переменную $a$ (при попытке вычислить корни уравнения произойдет деление на ноль). Чтобы избежать подобной ошибки нужно в программе предусмотреть проверку входных данных, например, так как показано на рис. 3.24. Вводится значение переменной $a$, если оно равно нулю, то ввод повторяется, иначе следует алгоритм вычисления корней квадратного уравнения. Здесь применяется *цикл с постусловием*, так как значение переменной необходимо ввести, а затем проверить его на равенство нулю.

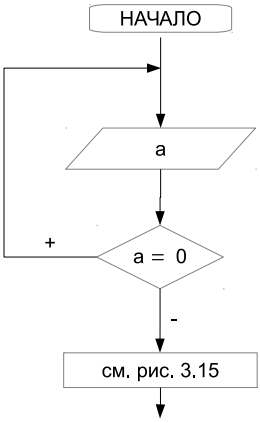


Рис. 3.24. Блок-схема проверки ввода данных

Программа решения задачи:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
```

```
float a,b,c,d,x1,x2;
//Проверка ввода значения коэффициента a.
do  //Выполнять тело цикла пока а равно нулю
{
  cout<<"a=";cin>>a;
}
while (a==0);
cout<<"b=";cin>>b;
cout<<"c=";cin>>c;
d=b*b-4*a*c;
if (d<0) cout<<"Нет вещественных корней";
else
{
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/(2*a);
cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
}
return 0;
}
```

**Задача 3.11.** Найти наибольший общий делитель (НОД) натуральных чисел $A$ и $B$.

*Входные данные*: $A$ и $B$.

*Выходные данные*: $A$ — НОД.

Для решения поставленной задачи воспользуемся алгоритмом Евклида: будем уменьшать каждый раз большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными, так, как показано в таблице 3.2.

Таблица 3.2: Поиск НОД для чисел $A = 25$ и $B = 15$.

| **Шаг** | **A** | **B** |
|---|---|---|
| Исходные данные | 25 | 15 |
| Шаг 1 | 10 | 15 |
| Шаг 2 | 10 | 5 |
| Шаг 3, НОД | 5 | 5 |

В блок–схеме, представленной на рис. 3.25, для решения поставленной задачи используется *цикл с предусловием*, то есть тело цикла повторяется до тех пор, пока $A$ не равно $B$. Следовательно, при создании программы воспользуемся циклом `while`:

```
#include <iostream>
using namespace std;
int main()
{
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

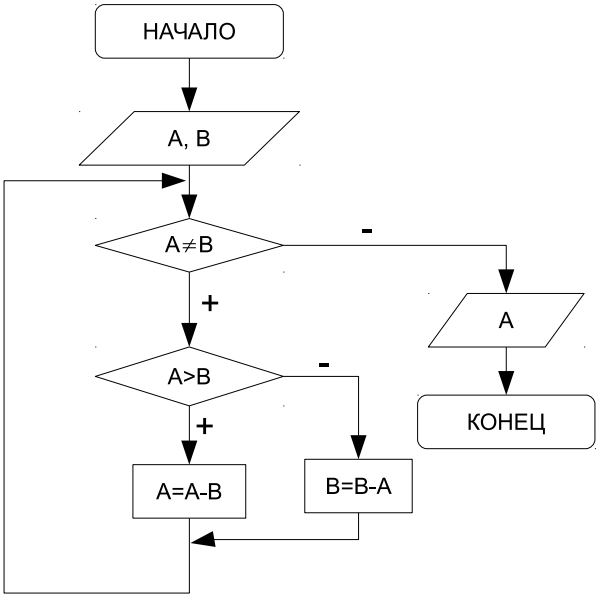


Рис. 3.25. Поиск наибольшего общего делителя двух чисел.

```
  unsigned int a,b;
  cout<<"A="; cin>>a;
  cout<<"B="; cin>>b;
  //Если числа не равны, выполнять тело цикла
  while (a!=b)
  //Если число A больше, чем B, то уменьшить его значение на B,
    if (a>b) a=a-b;
  //иначе уменьшить значение числа B на A
    else b=b-a;
  cout<<"НОД="<<a<<"\n";
return 0;
}
```

Результат работы программы не изменится, если для ее решения воспользоваться циклом с постусловием `do...while`:

```
#include <iostream>
using namespace std;
int main()
{
  unsigned int a,b;
```

```
  cout<<"A="; cin>>a;
  cout<<"B="; cin>>b;
  do
    if (a>b) a=a-b; else b=b-a;
  while (a!=b);
  cout<<"НОД="<<a<<"\n";
  return 0;
}
```

**Задача 3.12.** Вычислить факториал числа $N$ ( $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$).

*Входные данные*: $N$ — целое число, факториал которого необходимо вычислить.

*Выходные данные*: `factorial` — значение факториала числа $N$, произведение чисел от 1 до $N$, целое число.

Промежуточные переменные: $i$ — параметр цикла, целочисленная переменная, последовательно принимающая значения 2, 3, 4 и так далее до $N$.

Блок-схема приведена на рис. 3.26.

Итак, вводится число $N$. Переменной `factorial`, предназначенной для хранения значения произведения последовательности чисел, присваивается начальное значение, равное единице. Затем организуется цикл, параметром которого выступает переменная $i$. Если значение параметра цикла не превышает $N$, то выполняется оператор тела цикла, в котором из участка памяти с именем `factorial` считывается предыдущее значение произведения, умножается на текущее значение параметра цикла, а результат снова помещается в участок памяти с именем `factorial`. Когда параметр $i$ превысит $N$, цикл заканчивается, и на экран выводится значение переменой `factorial`, которая была вычислена в теле цикла.

Обратите внимание, как в программе записан *оператор цикла*. Здесь операторы ввода и операторы присваивания стартовых значений записаны как *начальные присваивания* цикла `for`, а оператор накапливания произведения и оператор модификации параметра цикла представляют собой *приращение*:

```
#include <iostream>
using namespace std;
int main()
{
unsigned long long int factorial;1
unsigned int N, i;
for (cout<<"N=",cin>>N,factorial=1,i=2;i<=N;factorial*=i,i++);
cout<<"факториал="<<factorial<<"\n";
return 0;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

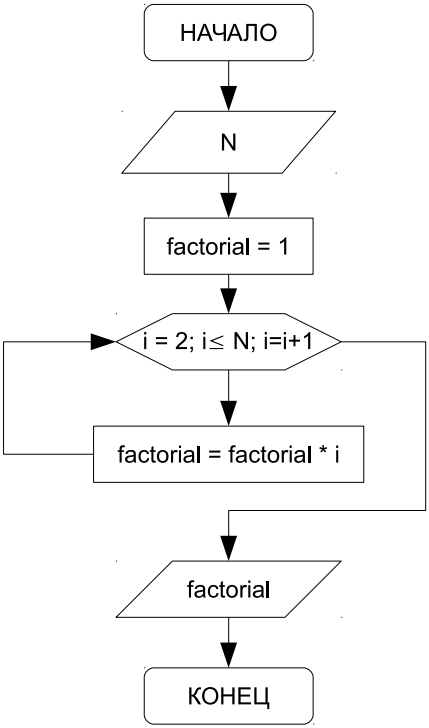


Рис. 3.26. Алгоритм вычисления факториала.

}

**Задача 3.13.** Вычислить сумму натуральных чётных чисел, не превышающих $N$.

*Входные данные*: $N$ — целое число.

*Выходные данные*: $S$ — сумма четных чисел.

Промежуточные переменные: $i$ — параметр цикла, принимает значения 2, 4, 6, 8 и так далее, также имеет целочисленное значение.

При сложении нескольких чисел необходимо накапливать результат в определённом участке памяти (S), каждый раз считывая из этого участка (S) предыдущее значение суммы (S) и прибавляя к нему слагаемое $i$. Для выполнения первого оператора накапливания суммы из участка памяти необходимо взять такое число, которое не влияло

бы на результат сложения. Перед началом цикла переменной, предназначенной для накапливания суммы, необходимо присвоить значение нуль. Блок-схема решения этой задачи представлена на рис. 3.27.

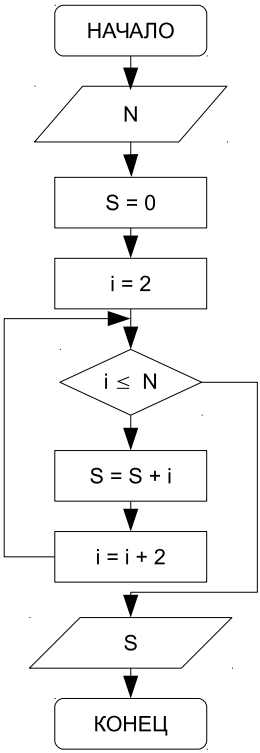


Рис. 3.27. Алгоритм вычисления суммы четных, натуральных чисел.

Решим задачу двумя способами: с применением циклов `while` и `for`:

```
//Решение задачи с помощью цикла while
#include <iostream>
using namespace std;
int main()
{
  unsigned int N,i,S;
  cout<<"N="; cin>>N;
  S=0;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  i=2;
  while (i<=N)
  {
    S=S+i;
    i=i+2;
  }
  cout<<"S="<<S<<"\n";
  return 0;
}
//-----------------------------------
//Решение задачи с помощью цикла for
#include <iostream>
using namespace std;
int main()
{
  unsigned int N,i,S;
  for (cout<<"N=",cin>>N,S=0,i=2;i<=N;S+=i,i+=2);
    cout<<"S="<<S<<"\n";
  return 0;
}
```

**Задача 3.14.** Дано натуральное число $N$. Определить $K$ — количество делителей этого числа, меньших самого числа (Например, для $N$=12 делители 1, 2, 3, 4, 6. Количество $K$=5).

*Входные данные*: $N$ — целое число.

*Выходные данные*: целое число $K$ — количество делителей $N$.

Промежуточные переменные: $i$ — параметр цикла, возможные делители числа $N$.

В блок-схеме, изображенной на рис. 3.28, реализован следующий алгоритм: в переменную $K$, предназначенную для подсчета количества делителей заданного числа, помещается значение, которое не влияло бы на результат, т.е. нуль. Далее организовывается цикл, в котором изменяющийся параметр $i$ выполняет роль возможных делителей числа $N$. Если заданное число $N$ делится нацело на параметр цикла $i$, это означает, что $i$ является делителем $N$, и значение переменной $K$ следует увеличить на единицу. Цикл необходимо повторить $\frac{N}{2}$ раз.

Текст программы на C++:

```
#include <iostream>
using namespace std;
int main()
{
  unsigned int N,i,K;
  cout<<"N="; cin>>N;
  for (K=0,i=1;i<=N/2;i++) if (N%i==0) K++;
```

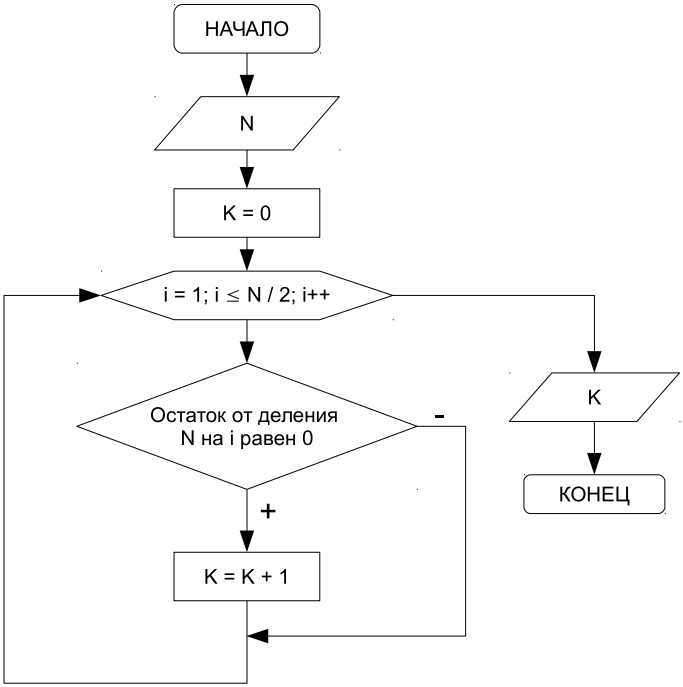


Рис. 3.28. Алгоритм определение делителей натурального числа.

```
  cout<<"K="<<K<<"\n";
  return 0;
}
```

**Задача 3.15.** Дано натуральное число $N$. Определить, является ли оно простым. Натуральное число $N$ называется простым, если оно делится без остатка только на единицу и на само себя. Число 13 — простое, так как делится только на 1 и 13, а число 12 таковым не является, так как делится на 1, 2, 3, 4, 6 и 12.

*Входные данные*: $N$ — целое число.

*Выходные данные*: сообщение.

Промежуточные переменные: $i$ — параметр цикла, возможные делители числа $N$.

Необходимо проверить есть ли делители числа $N$ в диапазоне от 2 до $N/2$ (рис. 3.29). Если делителей — нет, $N$ — простое число, ина-

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

че оно таковым не является. Обратите внимание на то, что в алгоритме предусмотрено два выхода из цикла. Первый — естественный, при исчерпании всех значений параметра, а второй — досрочный. Нет смысла продолжать цикл, если будет найден хотя бы один делитель из указанной области изменения параметра.

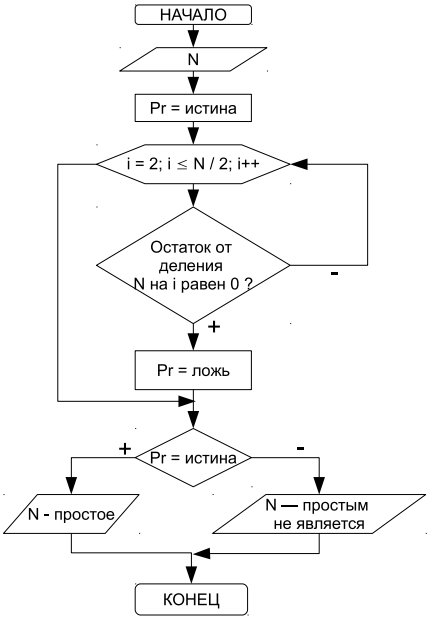


Рис. 3.29. Алгоритм определения простого числа.

При составлении программы на языке C++ досрочный выход из цикла удобно выполнять при помощи оператора `break`:

```
#include <iostream>
using namespace std;
int main()
{
unsigned int N,i;
bool Pr;
cout<<"N="; cin>>N;
Pr=true;   //Предположим, что число простое
for (i=2;i<=N/2;i++)
  if (N%i==0) //Если найдется хотя бы один делитель, то
  {
```

```
    Pr=false; //число простым не является и
    break; //досрочный выход из цикла
  }
  if (Pr) //Проверка значения логического параметра и
    //вывод на печать соответствующего сообщения
    cout<<N<<" - простое число \n";
  else
    cout<<N<<" - не является простым \n";
return 0;
}
```

**Задача 3.16.** Дано натуральное число N. Определить количество цифр в числе.

*Входные данные*: $N$ — целое число.

*Выходные данные*: $kol$ — количество цифр в числе.

*Промежуточные данные*: $M$ — переменная для временного хранения значения $N$[6].

Для того, чтобы подсчитать количество цифр в числе, необходимо определить, сколько раз заданное число можно разделить на десять нацело. Например, пусть $N = 12345$, тогда количество цифр $kol = 5$. Результаты вычислений сведены в таблицу 3.3.

Таблица 3.3: Определение количества цифр числа

| kol | N |
|---|---|
| 1 | 12345 |
| 2 | 12345 / 10 = 1234 |
| 3 | 1234 / 10 = 123 |
| 4 | 123 / 10 = 12 |
| 5 | 12 / 10 = 1 |
| | 1 / 10 = 0 |

Алгоритм определения количества цифр в числе представлен на рис. 3.30.

```
#include <iostream>
using namespace std;
int main()
{
        unsigned long int N, M;
```

[6]При решении задачи (см. алгоритм на рис. 3.30) исходное число изменятся, поэтому, чтобы его, не потерять, копируем исходное число $N$ в переменную $M$, и делить будем уже $M$.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

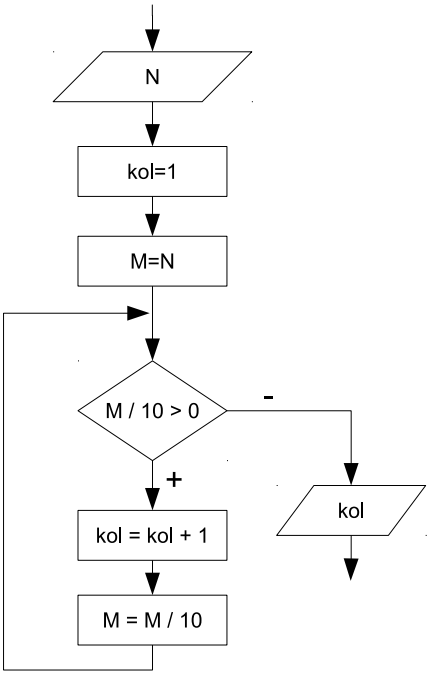


Рис. 3.30. Алгоритм определения количества цифр в числе.

```
        unsigned int kol;
        cout<<"N="; cin>>N;
        for (M=N, kol=1; M/10>0; kol++,M/=10);
        cout<<"kol="<<kol<<endl;
return 0;
}
```

**Задача 3.17.** Дано натуральное число $N$. Определить содержит ли это число нули и в каких разрядах они расположены (например, число 1101111011 содержит ноль в третьем и восьмом разрядах, а число 120405 — во втором и четвертом).

*Входные данные*: $N$ — целое число.

*Выходные данные*: *pos* — позиция цифры в числе.

*Промежуточные данные*: $i$ — параметр цикла, $M$ — переменная для временного хранения значения $N$.

В связи с тем, что разряды в числе выделяются начиная с последнего, то для определения номера разряда в числе, необходимо знать количество цифр в числе[7]. Таким образом, на первом этапе решения задачи необходимо определить *kol* — количество цифр в числе. Затем нужно выделять из числа цифры, если очередная цифра равна нулю, вывести на экран номер разряда, который занимает эта цифра. Процесс определения текущей цифры числа $N = 120405$ представлен в таблице 3.4.

Таблица 3.4: Определение текущей цифры числа

| **i** | **Число M** | **Цифра** | **Номер позиции** |
|---|---|---|---|
| 1 | 120405 | 120405 % 10 = 5 | 6 |
| 2 | 12040/10 = 1204 | 12040 % 10 = 0 | 5 |
| 3 | 1204/10 = 120 | 1204 % 10 = 4 | 4 |
| 4 | 120/10 = 12 | 120 % 10 = 0 | 3 |
| 5 | 12/10 = 1 | 12 % 10 = 2 | 2 |
| 6 | 1/10 = 0 | 1 % 10 = 1 | 1 |

Программный код к задаче 3.17.

```
#include <iostream>
using namespace std;
int main()
{
  unsigned long int N,M; int kol, i;
  cout<<"N="; cin>>N;
  for (kol=1,M=N;M/10>0; kol++,M/=10);
    for (M=N, i=kol; i>0;M/=10,i--)
      if (M%10==0) cout<<"Позиция = "<<i<<endl;
  return 0;
}
```

**Задача 3.18.** Дано натуральное число $N$. Получить новое число, записав цифры числа $N$ в обратном порядке. Например, 12345 — 54321.

*Входные данные*: $N$ — целое число.

*Выходные данные*: $S$ — целое число, полученное из цифр числа $N$, записанных в обратном порядке.

*Промежуточные данные*: $i$ — параметр цикла, $M$ — переменная для временного хранения значения $N$, *kol* — количество разрядов в заданном числе, $R = 10^{kol}$ — старший разряд заданного числа.

[7] Алгоритм нахождения количества цифр в числе был рассмотрен в предыдуще задаче.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Рассмотрим пример. Пусть $N = 12345$, тогда $S = 5 \cdot 10^4 + 4 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0 = 54321$

Значит, для решения поставленной задачи, нужно знать количество разрядов в заданном числе *kol* и его старший разряд $R = 10^{kol}$. Новое число $S$ формируют как сумму произведений последней цифры заданного числа на старший разряд $S+ = M\%10 * R$. Цикл выполняют *kol* раз, при каждой итерации уменьшая само число и старший разряд в десять раз.

```
#include <iostream>
using namespace std;
int main()
{unsigned long int N,M,R,S; int kol, i;
cout<<"N="; cin>>N;
for (R=1,kol=1,M=N;M/10>0; kol++,R*=10,M/=10);
  for(S=0,M=N, i=1;i<=kol;S+=M%10*R,M/=10,R/=10,i++);
    cout<<"S="<<S<<endl;
return 0;
}
```

**Задача 3.19.** Проверить является ли заданное число $N$ палиндромом[8]. Например, числа 404, 1221 — палиндромы.

*Входные данные*: $N$ — целое число.

*Выходные данные*: сообщение.

*Промежуточные данные*: $i$ — параметр цикла, $M$ — переменная для временного хранения значения $N$, *kol* — количество разрядов в заданном числе, $R = 10^{kol}$ — старший разряд заданного числа, $S$ — целое число, полученное из цифр числа $N$, записанных в обратном порядке.

Можно предложить следующий алгоритм решения задачи. Записать цифры заданного числа $N$ в обратном порядке (задача 3.18), получится новое число $S$. Сравнить полученное число $S$ с исходным $N$. Если числа равны, то заданное число является палиндромом.

Текст программы на языке C++:

```
#include <iostream>
using namespace std;
int main()
{unsigned long int N,M,R,S;
int kol, i;
cout<<"N="; cin>>N;
```

[8] Палиндром — это число, слово или фраза одинаково читающееся в обоих направлениях, или, другими словами, любой симметричный относительно своей середины набор символов.

```
for (R=1,kol=1,M=N;M/10>0; kol++,R*=10,M/=10);
  for(S=0,M=N, i=1;i<=kol;S+=M%10*R,M/=10,R/=10,i++);
    if (N==S) cout<<"Число - палиндром"<<endl;
    else cout<<"Число не является палиндромом"<<endl;
return 0;
}
```

**Задача 3.20.** Поступает последовательность из $N$ вещественных чисел. Определить наибольший элемент последовательности.

*Входные данные*: $N$ — целое число; $X$ — вещественное число, определяет текущий элемент последовательности.

*Выходные данные*: $Max$ — вещественное число, элемент последовательности с наибольшим значением.

*Промежуточные переменные*: $i$ — параметр цикла, номер вводимого элемента последовательности.

Алгоритм поиска наибольшего элемента в последовательности следующий (рис. 3.31). Вводится $N$ — количество элементов последовательности и $X$ — первый элемент последовательности. В памяти компьютера отводится ячейка, например с именем $Max$, в которой будет храниться наибольший элемент последовательности — максимум. Далее предполагаем, что первый элемент последовательности наибольший и записываем его в $Max$. Затем вводится второй элемент последовательности и сравниваем его с предполагаемым максимумом. Если окажется, что второй элемент больше, его записывают в ячейку $Max$. В противном случае никаких действий не предпринимаем. Потом переходим к вводу следующего элемента последовательности ($X$), и алгоритм повторяется с начала. В результате в ячейке $Max$ сохранится элемент последовательности с наибольшим значением[9].

Текст программы на C++:

```
#include <iostream>
using namespace std;
int main()
{
  unsigned int i,N;
  float X,Max;
  cout<<"N="; cin>>N;
  cout<<"X="; cin>>X; //Ввод первого элемента последовательности
  //Параметр цикла принимает стартовое значение i=2, т.к. первый элемент
  //уже введен предположим, что он максимальный, т.е. Max=X.
```

[9] Для поиска наименьшего элемента последовательности (минимума), предполагают, что первый элемент — наименьший, записывают его в ячейку min, а затем среди элементов последовательности ищут число, значение которого будет меньше чем предполагаемый минимум.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

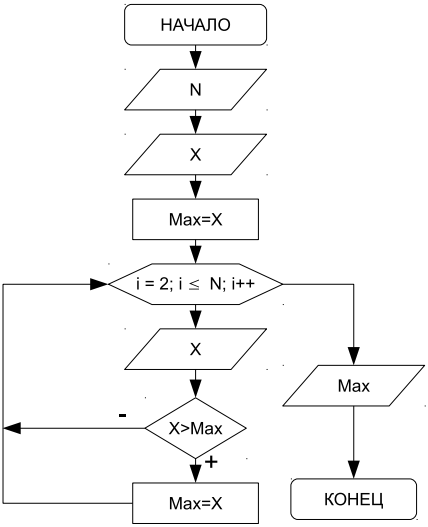


Рис. 3.31. Алгоритм поиска наибольшего числа в последовательности.

```
  for (i=2, Max=X; i<=N; i++)
  {
    cout<<"X="; cin>>X; //Ввод следующих элементов
        последовательности.
  //Если найдется элемент превышающий максимум,
  //записать его в ячейку Max, теперь он предполагаемый максимум.
  if (X>Max) Max=X;
  }
  //Вывод наибольшего элемента последовательности.
  cout<<"Max="<<Max<<"\n";
  return 0;
}
```

**Задача 3.21.** Вводится последовательность целых чисел, 0 — конец последовательности. Найти наименьшее число среди положительных, если таких значений несколько[10], определить, сколько их.

Блок-схема решения задачи приведена на рис. 3.32.

[10]Предположим вводится последовательность чисел 11, -3, 5, 12, -7, 5, 8,-9, 7, -6, 10, 5, 0. Наименьшим положительным числом является 5. Таких минимумов в последовательности 3.

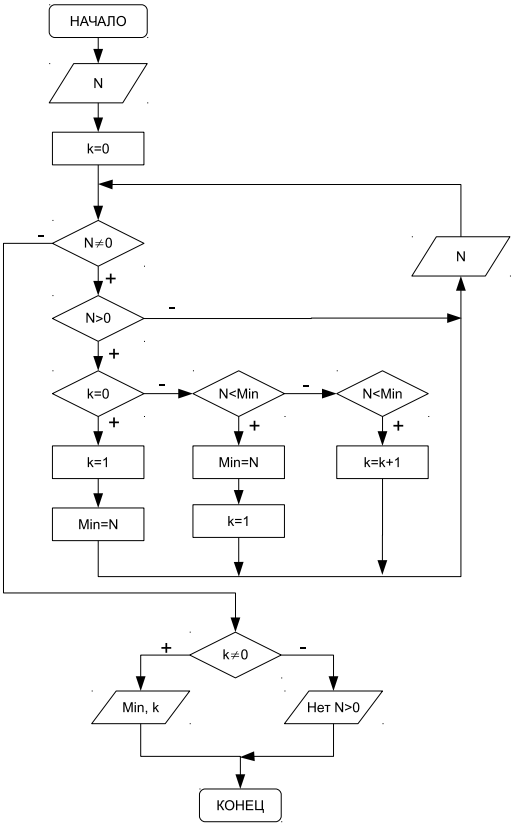


Рис. 3.32. Алгоритм поиска минимального положительного числа в последовательности.

Далее приведен текст подпрограммы с подробными комментариями[11].

```
#include <iostream>
using namespace std;
int main()
{
  float N,Min; int K;
```

[11] Алгоритм поиска максимального (минимального) элементов последовательности подробно описан в задаче 3.20

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  //Предположим, что в последовательности нет положительных чисел, K=0.
  //Вводим число и если оно не равно нулю
  for (cout<<"N=",cin>>N,K=0;N!=0;cout<<"N=",cin>>N)
    //проверяем является ли оно положительным.
    if (N>0)
      //если K=0, поступил 1-й элемент, предположим, что он минимальный.
        if (K==0) {K=1;Min=N;}
        //если элемент не первый сравниваем его с предполагаемым
            минимумом,
        //если элемент меньше записываем его в Min и сбрасываем счетчик
        else if (N<Min) {Min=N;K=1;}
        //если элемент равен минимуму увеличиваем значение счетчика.
          else if (N==Min) K++; //Конец цикла
  //Если значение счетчика не равно нулю, печатаем значение
  //минимального элемента и количество таких элементов.
  if (K!=0) cout<<"Min="<<Min<<"\n"<<"K="<<K<<"\n";
  //в противном случае выдаем сообщаем.
  else cout<<"Positive elements are absent \n";
  return 0;
}
```

**Задача 3.22.** Определить сколько раз последовательность из $N$ произвольных чисел меняет знак.

Чтобы решить задачу нужно попарно перемножать элементы последовательности. Если результат произведения пары чисел — отрицательное число, значит, эти числа имеют разные знаки.

Пусть в переменной $B$ хранится текущий элемент последовательности, в $A$ — предыдущий. Введём первое число $A$ (до цикла) и второе $B$ (в цикле). Если их произведение отрицательно, то увеличиваем количество смен знака на 1 (k++). После чего сохраняем значение B в переменную A и повторяем цикл (рис. 3.33).

Предлагаем читателю самостоятельно разобраться с текстом программы на C++:

```
#include <iostream>
using namespace std;
int main()
{
  float A,B; int i,K,N;
  cout<<"N=";cin>>N;
  for (K=0,cout<<"A=",cin>>A, i=2;i<=N;i++)
  {
    cout<<"B=";cin>>B;
    if (A*B<0) K++;
    A=B;
  }
  cout<<"K="<<K<<"\n";
  return 0;
```

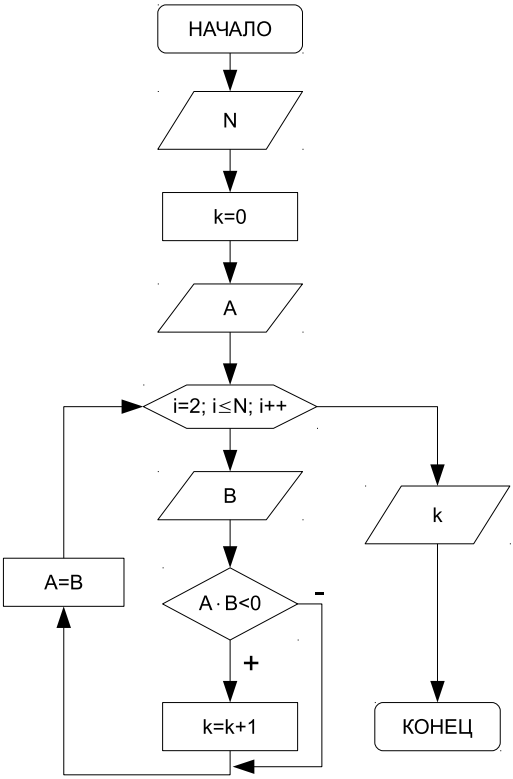


Рис. 3.33. Алгоритм решения задачи 3.22.

}

**Задача 3.23.** Поступает последовательность из $N$ вещественных чисел. Определить количество простых чисел в последовательности.

Блок-схема алгоритма изображена на рис. 3.34. Обратите внимание, что для решения задачи было организовано два цикла. Первый цикл обеспечивает ввод элементов последовательности. Второй цикл, находится внутри первого и определяет, является ли поступившее число простым (задача 3.15).

```
#include <iostream>
using namespace std;
int main()
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

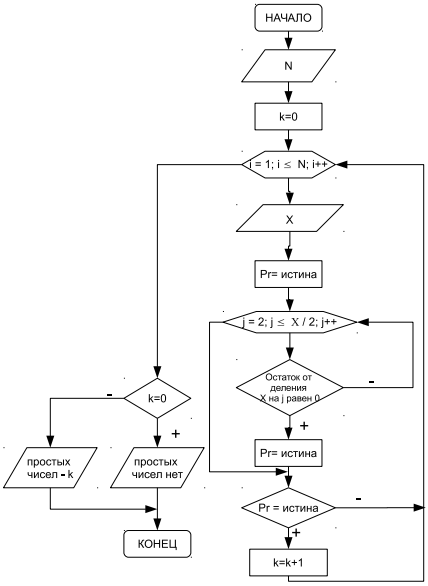


Рис. 3.34. Алгоритм поиска простых чисел в последовательности.

```
{
  unsigned long int X;
  unsigned int N;
  int i,k,j;
  bool Pr;
  for (k=0,cout<<"N=", cin>>N, i=1;i<=N;i++)
  {
    for (cout<<"X=", cin>>X,Pr=true,j=2;j<=X/2;j++)
      if (X%j==0)
      {
        Pr=false;
        break;
      }
      if (Pr) k++;
  }
  if (k==0) cout<<"Prime numbers are not \n";
  else  cout<<"Prime numbers k="<<k<<"\n";
return 0;
}
```

**Задача 3.24.** Дано K наборов ненулевых целых чисел. Каждый набор содержит не менее двух элементов, признаком его завершения является число 0. Найти количество наборов, элементы которых возрастают.

Блок-схема алгоритма решения задачи показана на рис. 3.35. Не трудно заметить, что алгоритм реализован с помощью двух циклических процессов. Внутренний цикл проверяет является ли последовательность возрастающей, а внешний повторяет алгоритм для новой последовательности.

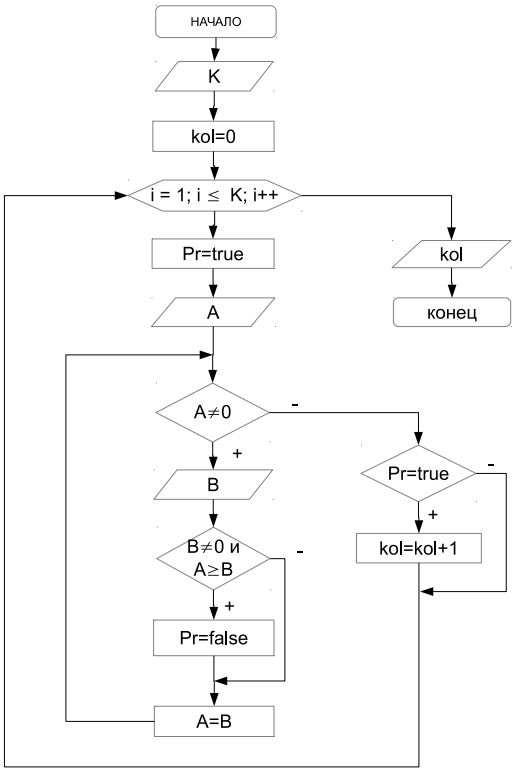


Рис. 3.35. Алгоритм решения задачи 3.24.

Программный код решения задачи 3.24:

```
#include <iostream>
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
using namespace std;
int main()
{
  unsigned int K, i,kol, A, B; bool pr;
  for (cout<<"K=", cin>>K, kol=0,i=1;i<=K;i++)
  {
    for (pr=true,cout<<"A=",cin>>A;A!=0; A=B)
    {
      cout<<"B="; cin>>B;
      if (B!=0 && A>=B) pr=false;
    }
    if (pr) kol++;
  }
  cout << "kol=" << kol<<endl;
  return 0;
}
```

## 3.6 Задачи для самостоятельного решения

### 3.6.1 Разветвляющийся процесс. Вычисление значения функции.

Разработать программу на языке C++. Дано вещественное число $a$. Для функции $y = f(x)$, график которой приведен ниже вычислить $f(a)$. Варианты заданий представлены на рис. 3.36–3.60.

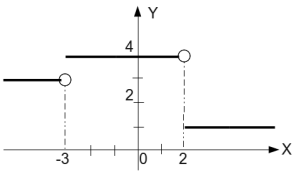


Рис. 3.36. Задание 1

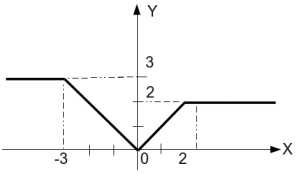


Рис. 3.37. Задание 2

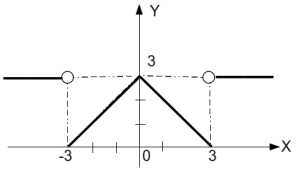


Рис. 3.38. Задание 3

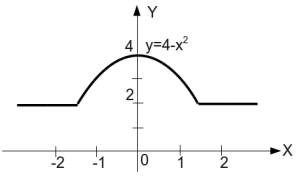


Рис. 3.39. Задание 4

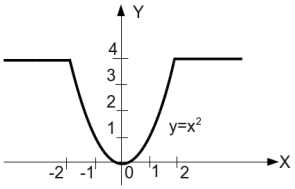


Рис. 3.40. Задание 5

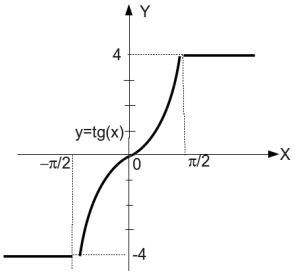


Рис. 3.41. Задание 6

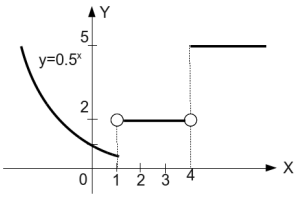


Рис. 3.42. Задание 7

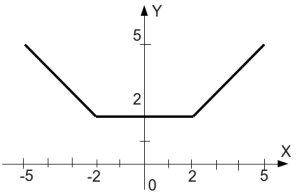


Рис. 3.43. Задание 8

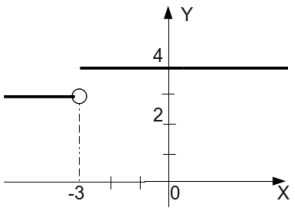


Рис. 3.44. Задание 9

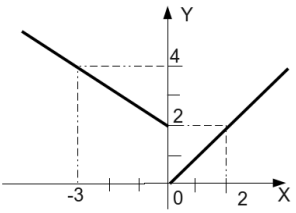


Рис. 3.45. Задание 10

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

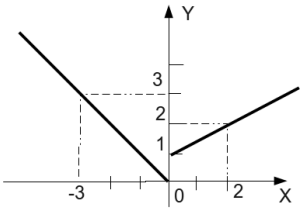


Рис. 3.46. Задание 11

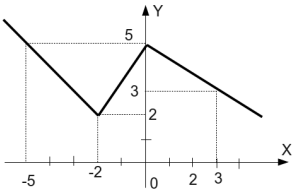


Рис. 3.47. Задание 12

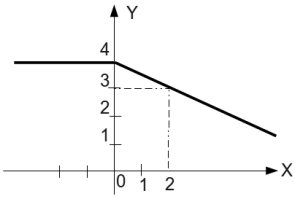


Рис. 3.48. Задание 13

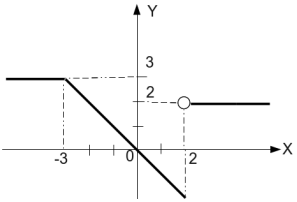


Рис. 3.49. Задание 14

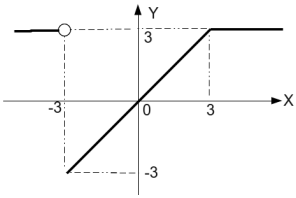


Рис. 3.50. Задание 15

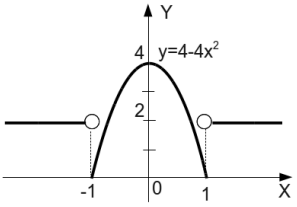


Рис. 3.51. Задание 16

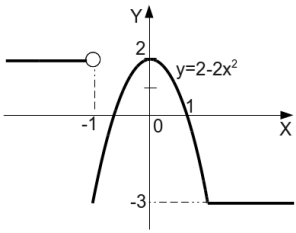


Рис. 3.52. Задание 17

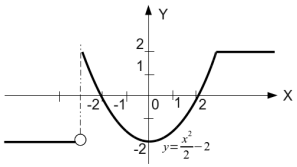


Рис. 3.53. Задание 18

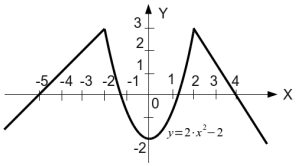


Рис. 3.54. Задание 19

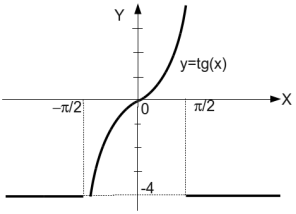


Рис. 3.55. Задание 20

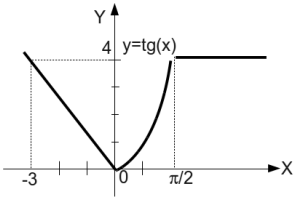


Рис. 3.56. Задание 21

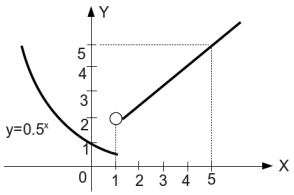


Рис. 3.57. Задание 22

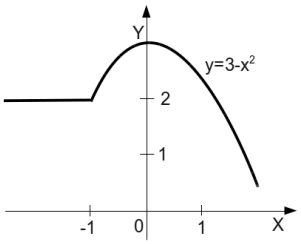


Рис. 3.58. Задание 23

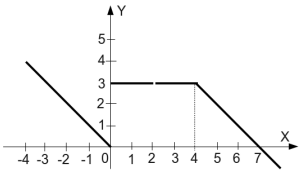


Рис. 3.59. Задание 24

### 3.6.2 Разветвляющийся процесс. Попадание точки в плоскость.

Разработать программу на языке C++. Даны вещественные числа $x$ и $y$. Определить принадлежит ли точка с координатами $(x; y)$ заштрихованной части плоскости. Варианты заданий представлены на рис. 3.61–3.85.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

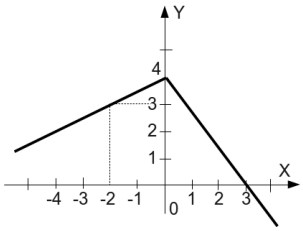


Рис. 3.60. Задание 25

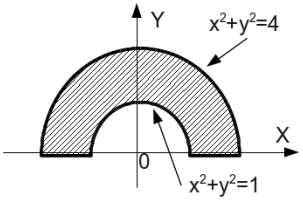


Рис. 3.61. Задание 1

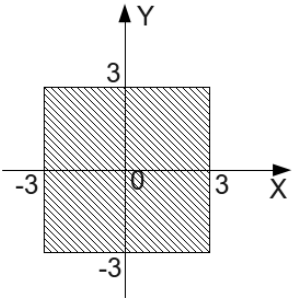


Рис. 3.62. Задание 2

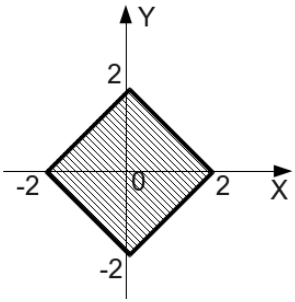


Рис. 3.63. Задание 3

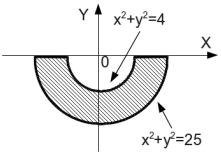


Рис. 3.64. Задание 4

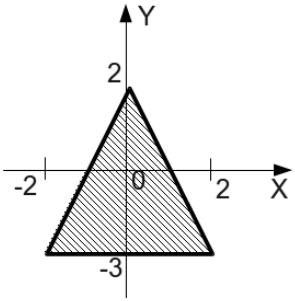


Рис. 3.65. Задание 5

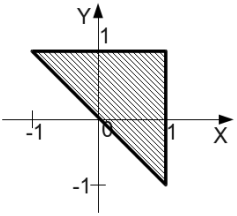


Рис. 3.66. Задание 6

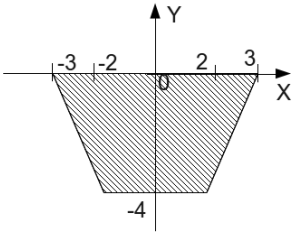


Рис. 3.67. Задание 7

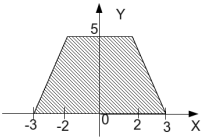


Рис. 3.68. Задание 8

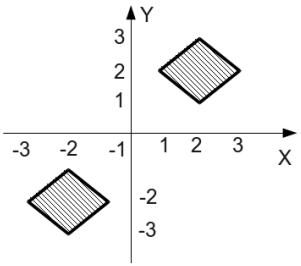


Рис. 3.69. Задание 9

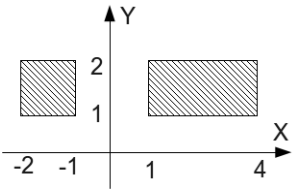


Рис. 3.70. Задание 10

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

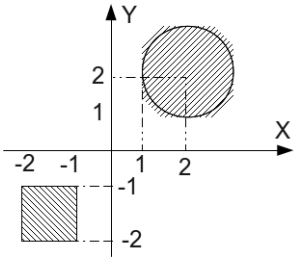


Рис. 3.71. Задание 11

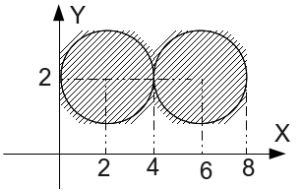


Рис. 3.72. Задание 12

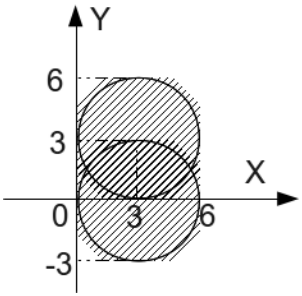


Рис. 3.73. Задание 13

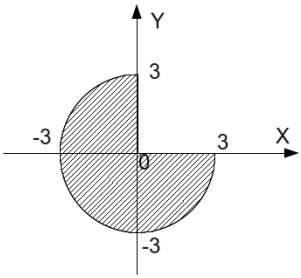


Рис. 3.74. Задание 14

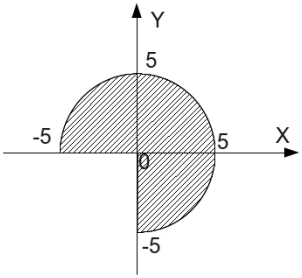


Рис. 3.75. Задание 15

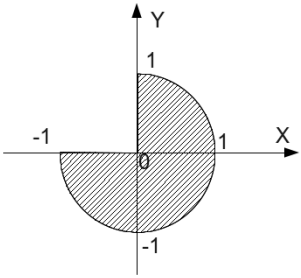


Рис. 3.76. Задание 16

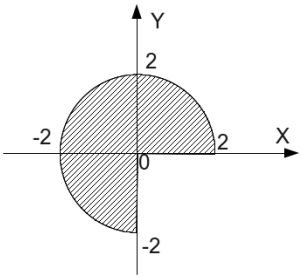


Рис. 3.77. Задание 17

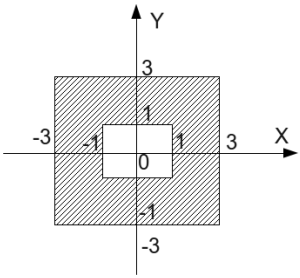


Рис. 3.78. Задание 18

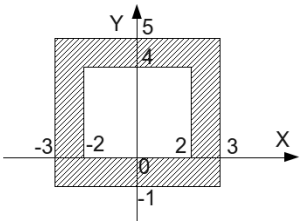


Рис. 3.79. Задание 19

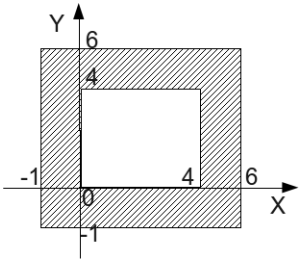


Рис. 3.80. Задание 20

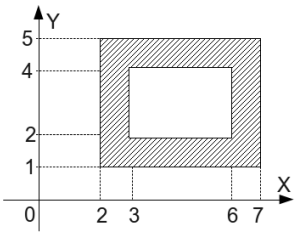


Рис. 3.81. Задание 21

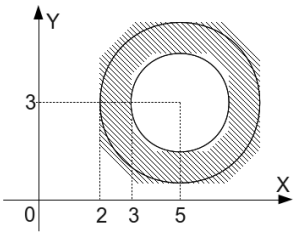


Рис. 3.82. Задание 22

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

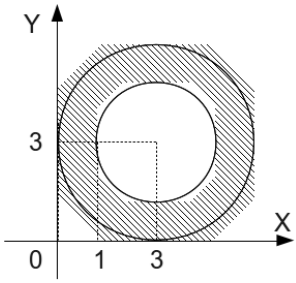


Рис. 3.83. Задание 23

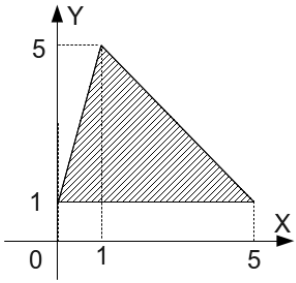


Рис. 3.84. Задание 24

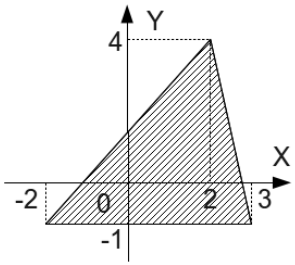


Рис. 3.85. Задание 25

### 3.6.3 Разветвляющийся процесс. Пересечение линий и решение уравнений.

Разработать программу на языке C++ для следующих заданий:

1. Задан круг с центром в точке $O(x_0, y_0)$, радиусом $R_0$ и точка $A(x_1, y_1)$. Определить, находится ли точка внутри круга.
2. Задана окружность с центром в точке $O(x_0, y_0)$ и радиусом $R_0$. Определить пересекается ли заданная окружность с осью абсцисс, если пересекается найти точки пересечения.
3. Задана окружность с центром в точке $O(x_0, y_0)$ и радиусом $R_0$. Определить пересекается ли заданная окружность с осью ординат, если пересекается найти точки пересечения.

4. Задана окружность с центром в точке $O(0,0)$ и радиусом $R_0$ и прямая $y = ax + b$. Определить, пересекаются ли прямая и окружность. Если пересекаются, найти точки пересечения.
5. Заданы окружности. Первая с центром в точке $O(x_1, y_1)$ и радиусом $R_1$, вторая с центром в точке $O(x_2, y_2)$ и радиусом $R_2$. Определить пересекаются окружности, касаются или не пересекаются.
6. Заданы три точки $A(x_1, y_1)$, $B(x_2, y_2)$,$C(x_3, y_3)$. Определить какая из точек наиболее удалена от начала координат.
7. Заданы три точки $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Определить какая из точек $B$ или $C$ наименее удалена от точки $A$.
8. Определить, пересекаются ли линии $y = ax + b$ и $y = kx + m$. Если пересекаются, найти точку пересечения.
9. Определить, пересекает ли линия $y = ax + b$ ось абсцисс. Если пересекает, найти точку пересечения.
10. Определить, пересекаются ли линии $y = ax^3 + bx^2 + cx + d$ и $y = kx + m$. Если пересекаются, найти точки пересечения.
11. Определить, пересекаются ли линии $y = ax^3 + bx^2 + cx + d$ и $y = kx^3 + mx^2 + nx + p$. Если пересекаются, найти точки пересечения.
12. Определить, пересекаются ли линии $y = ax^3 + bx^2 + cx + d$ и $y = ax^3 + mx^2 + nx + p$. Если пересекаются, найти точки пересечения.
13. Определить, пересекаются ли линии $y = ax^3 + bx^2 + cx + d$ и $y = mx^2 + nx + p$. Если пересекаются, найти точку пересечения.
14. Определить, пересекает ли линия $y = ax^3 + bx^2 + cx + d$ ось абсцисс. Если пересекает, найти точку пересечения.
15. Определить, пересекаются ли параболы $y = ax^2 + bx + c$ и $y = dx^2 + mx + n$. Если пересекаются, то найти точки пересечения.
16. Определить, пересекаются ли линии $y = bx^2 + cx + d$ и $y = kx + m$. Если пересекаются, найти точки пересечения
17. Найти точки пересечения линии $y = ax^2 + bx + c$ с осью абсцисс. Если линии не пересекаются выдать соответствующее сообщение.
18. Определить, пересекаются ли линии $y = ax^4 + bx^3 + cx^2 + dx + f$ и $y = bx^3 + mx^2 + dx + p$. Если пересекаются, найти точки пересечения.
19. Определить, пересекаются ли линии $y = ax^4 + bx^2 + kx + c$ и $y = mx^2 + kx + p$. Если пересекаются, найти точки пересечения.
20. Определить, пересекает ли линия $y = ax^4 + bx^2 + c$ ось абсцисс. Если пересекает, найти точки пересечения.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

21. Найти комплексные корни уравнения $y = ax^4 + bx^2 + c$. Если в уравнении нет комплексных корней вывести соответствующее сообщение.
22. Найти комплексные корни уравнения $y = ax^3 + bx^2 + cx + d$. Если в уравнении нет комплексных корней вывести соответствующее сообщение.
23. Найти комплексные корни уравнения $y = ax^2 + bx + c$. Если в уравнении нет комплексных корней вывести соответствующее сообщение.
24. Даны координаты точки на плоскости. Если точка совпадает с началом координат, то вывести 0. Если точка не совпадает с началом координат, но лежит на оси $OX$ или $OY$, то вывести соответственно 1 или 2. Если точка не лежит на координатных осях, то вывести 3.
25. Даны координаты точки, не лежащей на координатных осях OX и OY. Определить номер координатной четверти, в которой находится данная точка.

### 3.6.4 Циклический процесс. Вычисление значений функции

Разработать программу на языке C++. Для решения задачи использовать операторы `for`, `while`, `do`. Варианты заданий:

1. Вывести на экран таблицу значений функции синус в диапазоне от $-2 \cdot \pi$ до $2 \cdot \pi$ с шагом $\frac{\pi}{8}$.
2. Вывести на экран таблицу квадратов первых десяти целых положительных чисел.
3. Вывести на экран таблицу значений функции косинус в диапазоне от $-2 \cdot \pi$ до $2 \cdot \pi$ с шагом $\frac{\pi}{8}$ .
4. Вывести на экран таблицу кубов первых десяти целых положительных чисел.
5. Вывести на экран таблицу значений квадратов синусов в диапазоне от $-\pi$ до $\pi$ с шагом $\frac{\pi}{12}$.
6. Вывести на экран таблицу значений квадратов косинусов в диапазоне от 0 до $2 \cdot \pi$ с шагом $\frac{\pi}{10}$.
7. Вывести на экран таблицу квадратов первых десяти целых четных положительных чисел.
8. Вывести на экран таблицу квадратов первых десяти целых нечетных положительных чисел.

9. Вывести на экран таблицу значений удвоенных синусов в диапазоне от $-a$ до $a$ с шагом $h$. Значения $a$ и $h$ вводятся с клавиатуры.
10. Вывести на экран таблицу значений удвоенных косинусов в диапазоне от $a$ до $b$ с шагом $h$. Значения $a$, $b$ и $h$ вводятся с клавиатуры.
11. Вывести на экран таблицу кубов первых десяти целых нечетных положительных чисел.
12. Вывести на экран таблицу кубов первых десяти целых четных положительных чисел.
13. Вывести на экран таблицу значений функции $y = e^{2x}$ в диапазоне от $-a$ до $a$ с шагом $h$. Значения $a$ и $h$ вводятся с клавиатуры.
14. Вывести на экран таблицу значений функции $y = 5 \cdot e^{-3x}$ в диапазоне от $a$ до $b$ с шагом $h$. Значения $a$ , $b$ и $h$ вводятся с клавиатуры.
15. Вывести на экран таблицу квадратов первых десяти целых отрицательных чисел.
16. Вывести на экран таблицу кубов первых десяти целых отрицательных чисел.
17. Вывести на экран таблицу квадратных корней первых десяти целых положительных чисел.
18. Вывести на экран таблицу кубических корней первых десяти целых положительных чисел.
19. Вывести на экран таблицу значений функции $y = 2 \cdot x^2 + 3 \cdot x - 1$ в диапазоне от $-a$ до $a$ с шагом $h$ . Значения $a$ и $h$ вводятся с клавиатуры.
20. Вывести на экран таблицу значений функции $y = 5.4 \cdot x^3 - 2.8 \cdot x^2 - x + 1.6$ в диапазоне от $a$ до $b$ с шагом $h$. Значения $a$, $b$ и $h$ вводятся с клавиатуры.
21. Вывести на экран таблицу квадратных корней первых десяти целых положительных чётных чисел.
22. Вывести на экран таблицу квадратных корней первых десяти целых положительных нечётных чисел.
23. Вывести на экран таблицу значений функции $y = -1.8 \cdot x^3 - e^2 x + \frac{1}{6}$ в диапазоне от $-3$ до 4 с шагом $\frac{1}{2}$ .
24. Вывести на экран таблицу значений функции $y = -1.3 \cdot x^2 - \frac{e^x}{4}$ в диапазоне от $-2$ до 2 с шагом $\frac{1}{4}$ .
25. Вывести на экран таблицу степеней двойки в диапазоне от 0 до 10 с шагом 1.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

### 3.6.5 Циклический процесс. Последовательности натуральных чисел

Разработать программу на языке C++ для следующих заданий:

1. Дано целое положительное число $N$. Вычислить сумму натуральных нечетных чисел не превышающих это число.
2. Дано целое положительное число $N$. Вычислить произведение натуральных четных чисел не превышающих это число.
3. Дано целое положительное число $N$. Вычислить количество натуральных чисел кратных трем и не превышающих число $N$.
4. Задано целое положительное число $n$. Определить значение выражения:
$$P = \frac{n!}{\sum\limits_{i=1}^{n} i}.$$
5. Вычислить количество натуральных двузначных четных чисел не делящихся на 10.
6. Задано целое положительное число $n$. Определить значение выражения:
$$P = \frac{\sum\limits_{i=1}^{n} i^2}{n!}.$$
7. Вычислить сумму натуральных удвоенных чисел не превышающих 25.
8. Задано целое положительное число $n$. Определить значение выражения:
$$P = \frac{\sum\limits_{i=3}^{n} i - 2}{(n+1)!}.$$
9. Дано целое положительное число $N$. Вычислить сумму квадратов натуральных чётных чисел не превышающих это число.
10. Дано целое положительное число $N$. Вычислить количество натуральных чисел кратных пяти и не превышающих число $N$.
11. Определить значение выражения:
$$P = \frac{\sum\limits_{i=0}^{5} 3^i}{5!}.$$

12. Дано целое положительное число $N$. Вычислить сумму удвоенных натуральных нечётных чисел не превышающих это число.
13. Задано целое положительное число $n$. Определить значение выражения:
$$P = \sum_{i=2}^{n} i^2 - i.$$
14. Найти сумму нечётных степеней двойки. Значение степени изменяется от 1 до 9.
15. Задано целое положительное число $n$. Определить значение выражения:
$$P = \frac{1}{3} \cdot \sum_{i=1}^{n} 2 \cdot i^2 - i + 1.$$
16. Дано целое положительное число $N$. Вычислить произведение натуральных чисел кратных трем и не превышающих число $N$.
17. Задано целое положительное число $n$. Определить значение выражения:
$$P = \sum_{i=3}^{n+2} 2 \cdot i - 4.$$
18. Вычислить сумму натуральных трёхзначных чисел кратных пяти и не делящихся на десять.
19. Определить значение выражения:
$$P = \sum_{i=0}^{10} 2^i.$$
20. Вычислить количество натуральных двузначных нечётных чисел не делящихся на 5.
21. Задано целое положительное число $n$. Определить значение выражения:
$$P = \frac{\sum\limits_{i=0}^{n-1} i + 1}{(2n)!}.$$
22. Задано целое положительное число $n$. Определить значение выражения:
$$P = \frac{\sum\limits_{i=5}^{15} i}{(2 \cdot n + 1)!}.$$

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

23. Найти произведение чётных степеней двойки. Значение степени изменяется от 0 до 8.
24. Вычислить произведение натуральных чисел не превышающих 15.
25. Вычислить произведение натуральных двузначных чисел кратных трем и не делящихся на 10.

### 3.6.6 Циклический процесс. Последовательности произвольных чисел

Разработать программу на языке C++ для следующих заданий:

1. Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить сумму положительных элементов последовательности.
2. Вычислить сумму отрицательных элементов последовательности из $N$ произвольных чисел.
3. Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить сколько раз последовательность поменяет знак.
4. В последовательности из $N$ произвольных чисел подсчитать количество нулей.
5. Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить наибольшее число в последовательности.
6. Вводится последовательность из $N$ произвольных чисел найти наименьшее число в последовательности.
7. Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить среднее значение элементов последовательности.
8. Вводится последовательность из $N$ произвольных чисел, найти среднее значение положительных элементов последовательности.
9. Вводится последовательность ненулевых чисел, 0 — конец последовательности. Подсчитать процент положительных и отрицательных чисел.
10. Вводится последовательность из $N$ произвольных чисел. Определить процент положительных, отрицательных и нулевых элементов.

11. Вводится последовательность из $N$ произвольных чисел. Вычислить разность между наименьшим и наибольшим значениями последовательности.
12. Вводится последовательность из $N$ положительных целых чисел. Найти наименьшее число среди четных элементов последовательности.
13. Вводится последовательность из $N$ положительных целых чисел. Определить является ли эта последовательность знакочередующейся.
14. Определить является ли последовательность из $N$ произвольных чисел строго возрастающей (каждый следующий элемент больше предыдущего).
15. Вводится последовательность произвольных чисел, 0 — конец последовательности. Определить является ли эта последовательность строго убывающей (каждый следующий элемент меньше предыдущего).
16. Вводится последовательность ненулевых целых чисел, 0 — конец последовательности. Определить среднее значение чётных элементов последовательности.
17. Вводится последовательность из $N$ произвольных чисел, найти среднее значение отрицательных элементов последовательности.
18. В последовательности из $N$ целых чисел подсчитать четных и нечетных чисел.
19. Вводится последовательность целых чисел, 0 — конец последовательности. Определить процент чётных и нечётных чисел в последовательности.
20. Вводится последовательность из $N$ целых чисел. Определить содержит ли последовательность хотя бы два соседних одинаковых числа.
21. Вводится последовательность целых чисел, 0 — конец последовательности. Определить наибольшее число среди нечетных элементов последовательности.
22. Вводится последовательность произвольных чисел, 0 — конец последовательности. Определить сумму и количество чисел в последовательности.
23. Вводится последовательность из $N$ произвольных чисел. Найти сумму положительных и сумму отрицательных элементов последовательности.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

24. Вводится последовательность произвольных чисел, 0 — конец последовательности. Определить отношение минимального и максимального элементов друг к другу.
25. Вводится последовательность из $N$ целых чисел. Определить количество одинаковых рядом стоящих чисел.

### 3.6.7 Циклический процесс. Работа с цифрами в числе

Разработать программу на языке C++ для следующих заданий:

1. Определить является ли целое положительное число *совершенным*. Совершенное число равно сумме всех своих делителей, не превосходящих это число. Например, 6=1+2+3 или 28=1+2+4+7+14.
2. Проверить является ли пара целых положительных чисел *дружественными*. Два различных натуральных числа являются дружественными, если сумма всех делителей первого числа (кроме самого числа) равна второму числу. Например, 220 и 284, 1184 и 1210, 2620 и 2924, 5020 и 5564.
3. Определить является ли целое положительное число *недостаточным*. Недостаточное число всегда больше суммы всех своих делителей за исключением самого числа.
4. Вводится целое положительное число. Определить количество четных и нечетных цифр в числе.
5. Вводится целое положительное число. Найти число, которое равно сумме кубов цифр исходного числа.
6. Задача о «*счастливом*» билете. Вводится целое положительное шестизначное число. Определить совпадает ли сумма первых трех цифр с суммой трех последних.
7. Задача о «*встречном*» билете. Вводится целое положительное шестизначное число. Убедиться, что разница между суммой первых трех цифр и суммой последних трех цифр равна единице.
8. Задано целое положительное число. Определить количество его четных и нечетных делителей.
9. Проверить является ли два целых положительных числа *взаимно простыми*. Два различных натуральных числа являются взаимно простыми, если их наибольший общий делитель равен единице.
10. Определить является ли целое положительное число *составным*. Составное число имеет более двух делителей, то есть не является *простым*.

11. Вводится целое положительное число. Найти наименьшую цифру числа.
12. Задано целое положительное число. Определить является ли оно *числом Армстронга*. Число Армстронга — натуральное число, которое равно сумме своих цифр, возведённых в степень, равную количеству его цифр. Например, десятичное число 153 — число Армстронга, потому что: $1^3 + 3^3 + 5^3 = 1 + 27 + 125 = 153$.
13. Вводится целое положительное число. Найти произведение всех ненулевых цифр числа.
14. Вводится целое положительное число. Найти наибольшую цифру числа.
15. Вводится целое положительное число. Определить позицию наибольшей цифры в числе.
16. Вводится целое положительное число. Найти число, которое равно сумме удвоенных цифр исходного числа.
17. Вводится целое положительное число. Найти число, которое равно сумме квадратов цифр исходного числа.
18. Задано целое положительное число. Определить сумму его делителей.
19. Вводится целое положительное число. Определить позицию наименьшей цифры в числе.
20. Проверить, что два целых положительных числа не являются *взаимно простыми*. Различные натуральные числа не являются взаимно простыми, если их наибольший общий делитель отличен от единицы.
21. Убедиться, что заданное целое положительное число не является *палиндромом*. Числа палиндромы симметричны относительно своей середины, например, 12021 или 454.
22. Убедиться, что заданное целое положительное число не является *совершенным*. Совершенное число равно сумме всех своих делителей, не превосходящих это число. Например, 6=1+2+3 или 28=1+2+4+7+14.
23. Проверить, что два целых положительных числа не являются *дружественными*. Два различных натуральных числа являются дружественными, если сумма всех делителей первого числа (кроме самого числа) равна второму числу. Например, 220 и 284, 1184 и 1210, 2620 и 2924, 5020 и 5564.
24. Вводится целое положительное число. Найти число, которое равно сумме утроенных цифр исходного числа.
25. Вводятся два целых положительных числа. Найти сумму их цифр.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

### 3.6.8 Вложенные циклы

Разработать программу на языке C++ для следующих заданий:

1. Дано натуральное число $P$. Вывести на печать все простые числа не превосходящие $P$.
2. Дано натуральное число $P$. Вывести на печать все совершенные числа не превосходящие $P$.
3. Вводится последовательность положительных целых чисел, 0 — конец последовательности. Определить количество совершенных чисел в последовательности.
4. Вводится последовательность положительных целых чисел, 0 — конец последовательности. Определить количество простых чисел в последовательности.
5. Вводится последовательность из $N$ положительных целых чисел. Для каждого элемента последовательности вычислить факториал.
6. Вводится последовательность из $N$ положительных целых чисел. Вывести на печать все числа — палиндромы. Если таких чисел нет, выдать соответствующее сообщение.
7. Вводится последовательность из $N$ положительных целых чисел. Определить разрядность каждого числа.
8. Вводится последовательность из $N$ положительных целых чисел. Вывести на печать количество делителей каждого числа.
9. Вводится последовательность положительных целых чисел, 0 — конец последовательности. Определить сумму цифр каждого элемента последовательности.
10. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Для каждого набора вывести количество его элементов. Вычислить общее количество элементов.
11. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Для каждого набора вычислить среднее арифметическое его элементов.
12. Даны $K$ наборов целых чисел по $N$ элементов в каждом наборе. Для каждого набора найти наибольшее значение его элементов.
13. Даны $K$ наборов целых чисел по $N$ элементов в каждом наборе. Определить есть ли среди наборов данных знакочередующиеся последовательности.
14. Даны $K$ наборов целых чисел по $N$ элементов в каждом наборе. Определить есть ли среди наборов данных строго возрастающие последовательности.

15. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Для каждого набора найти наименьшее значение его элементов.
16. Даны $K$ наборов целых чисел по $N$ элементов в каждом наборе. Для каждого набора вычислить произведение ненулевых элементов.
17. Даны $K$ наборов целых чисел по $N$ элементов в каждом наборе. Найти наибольшее число для всех наборов.
18. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Вычислить среднее арифметическое всех элементов во всех наборах.
19. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Найти количество возрастающих наборов.
20. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Найти количество убывающих наборов.
21. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Найти количество наборов не являющихся знакочередующимися.
22. Дано $K$ наборов ненулевых целых чисел. Признаком завершения каждого набора является число 0. Найти количество наборов элементы которых не возрастают и не убывают.
23. Даны целые положительные числа $N$ и $M$ ($N < M$). Вывести все целые числа от $N$ до $M$ включительно; при этом каждое число должно выводиться столько раз, каково его значение (например, число 5 выводится 5 раза).
24. Дано целое число $N > 0$. Найти сумму $1! + 2! + 3! + \ldots + N!$
25. Даны целые числа $N$ и $M$ ($N < M$). Вывести все целые числа от $N$ до $M$ включительно; при этом число $N$ должно выводиться 1 раз, число $N + 1$ должно выводиться 2 раза и т. д.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

# Глава 4

# Использование функций при программировании на C++

В практике программирования часто складываются ситуации, когда одну и ту же группу операторов, реализующих определенную цель, требуется повторить без изменений в нескольких местах программы. Для избавления от столь нерациональной траты времени была предложена концепция подпрограммы.

*Подпрограмма* — именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке C++ подпрограммы реализованы в виде *функций* [4].

## 4.1 Общие сведения о функциях. Локальные и глобальные переменные

*Функция* — это поименованный набор описаний и операторов, выполняющих определенную задачу. Функция может принимать параметры и возвращать значение. Информация, передаваемая в функцию для обработки, называется *параметром*, а результат вычислений функции ее *значением*. Обращение к функции называют *вызовом*. Как известно (п. 2.8) любая программа на C++ состоит из одной или нескольких функций. При запуске программы первой выполняется функция `main`. Если среди операторов функции `main` встречается вызов функции, то управление передается операторам функции. Когда

все операторы функции будут выполнены, управление возвращается оператору, следующему за вызовом функции.

Перед вызовом функция должны быть обязательно описана. *Описание функции* состоит из заголовка и тела функции:

```
тип имя_функции(список_переменных)
{
  тело_функции
}
```

*Заголовок функции* содержит:

- `тип`, возвращаемого функцией значения, он может быть любым; если функция не возвращает значения, указывают тип `void`;
- `имя_функции;`
- `список_переменных` — перечень передаваемых в функцию величин (*аргументов*), которые отделяются друг от друга запятыми; для каждой переменной из списка указывается `тип` и `имя`; если функция не имеет аргументов, то в скобках указывают либо тип `void`, либо ничего.

*Тело функции* представляет собой последовательность описаний и операторов, заключенных в фигурные скобки.

В общем виде *структура программы* на C++ может иметь вид:

```
директивы компилятора
тип имя_1(список_переменных)
{
  тело_функции_1;
}

тип имя_2(список_переменных)
{
  тело_функции_2;
}

...

тип имя_n(список_переменных)
{
  тело_функции_n;
}

int main(список_переменных)
{
  //Тело функции может содержать операторы вызова функций имя_1, имя_2,
      ..., имя_n
  тело_основной_функции;
}
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Однако допустима и другая *форма записи программного кода*:

```
директивы компилятора
тип имя_1(список_переменных);
тип имя_2(список_переменных);
...
тип имя_n(список_переменных);
int main(список_переменных)
{
  //Тело функции может содержать операторы вызова функций имя_1, имя_2,
      ..., имя_n
  тело_основной_функции;
}
тип имя_1(список_переменных)
{
  тело_функции_1;
}

тип имя_2(список_переменных)
{
  тело_функции_2;
}

...

тип имя_n(список_переменных)
{
  тело_функции_n;
}
```

Здесь функции описаны после функции `main()`, однако до нее перечислены заголовки всех функций. Такого рода опережающие заголовки называют *прототипами функций*. Прототип указывает компилятору тип данных, возвращаемых функцией, тип переменных, выступающих в роли аргументов и порядок их следования. Прототипы используются для проверки правильности вызова функций в основной программе. Имена переменных, указанные в прототипе функции, компилятор игнорирует:

```
//Записи равносильны.
int func(int a,int b);
int func(int ,int);
```

*Вызвать функцию* можно в любом месте программы. Для *вызова функции* необходимо указать ее `имя` и в круглых скобках, через запятую перечислить `имена` или `значения аргументов`, если таковые имеются:

```
имя_функции(список_переменных);
```

Рассмотрим пример. Создадим функцию `f()`, которая не имеет входных значений и не формирует результат. При вызове этой функции на экран выводится строка символов `"Happy new year, "`.

```
#include <iostream>
using namespace std;
void f()   //Описание функции.
{
  cout << "Happy new year, ";
}
int main()
{
  f();        //Вызов функции.
  cout <<"Students!" << endl;
  f();        //Вызов функции.
  cout <<"Teachers!" << endl;
  f();        //Вызов функции.
  cout <<"People!" << endl;
}
```

Результатом работы программы будут три строки:

```
Happy new year, Students!
Happy new year, Teachers!
Happy new year, People!
```

Далее приведен пример программы, которая пять раз выводит на экран фразу `"Hello World!"`. Операция вывода строки символов оформлена в виде функции `fun()`. Эта функция так же не имеет входных значений и не формирует результат. Вызов функции осуществляется в цикле:

```
#include <iostream>
using namespace std;
void fun()
{
  cout << "Hello World!" << endl;
}

int main()
{
  for(int i=1;i<=5;fun(),i++);
}
```

Если тип возвращаемого значения не `void`, то функция может входить в состав выражений. *Типы и порядок следования переменных в определении и при вызове функции должны совпадать.* Для того чтобы *функция вернула* какое-либо значение, в ней должен быть оператор:

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
return (выражение);
```

Далее приведен пример программы, которая вычисляет значение выражения $\sin^2(\alpha) + \cos^2(\alpha) = 1$ при заданном значении $\alpha$. Здесь функция **radian** выполняет перевод градусной меры угла в радианную[1].

```
#include <iostream>
#include <math.h>
#define PI 3.14159
using namespace std;
double radian(int deg, int min, int sec)
{
  return (deg*PI/180+min*PI/180/60+sec*PI/180/60/60);
}
int main()
{
  int DEG,MIN,SEC; double RAD;
  //Ввод данных.
  cout<<"Inpout:"<<endl;     //Величина угла:
  cout<<"DEG="; cin>>DEG;    //градусы,
  cout<<"MIN="; cin>>MIN;    //минуты,
  cout<<"SEC="; cin>>SEC;    //секунды.
  //Величина угла в радианах.
  RAD=radian(DEG,MIN,SEC); //Вызов функции.
  cout << "Value in radian A="<<RAD << endl;
  //Вычисление значения выражения и его вывод.
  cout << "sin(A)^2+cos(A)^2=";
  cout << pow(sin(RAD),2)+pow(cos(RAD),2) << endl;
  return 0;
}
```

Переменные, описанные внутри функции, а так же переменные из списка аргументов, являются *локальными*. Например, если программа содержит пять разных функций, в каждой из которых описана переменная $N$, то для C++ это пять различных переменных. Область действия локальной переменной не выходит за рамки функции. Значения локальных переменных между вызовами одной и той же функции не сохраняются.

Переменные, определенные до объявления всех функций и доступные всем функциям, называют *глобальными*. В функции глобальную переменную можно отличить, если не описана локальная переменная с теми же именем.

[1] Чтобы найти радианную меру какого-нибудь угла по заданной градусной мере, нужно помножить число градусов на $\frac{\pi}{180}$, число минут на $\frac{\pi}{180 \cdot 60}$, число секунд на и $\frac{\pi}{180 \cdot 60 \cdot 60}$ найденные произведения сложить.

Глобальные переменные применяют для передачи данных между функциями, но это затрудняет отладку программы. Для обмена данными между функциями используют параметры функций и значения, возвращаемые функциями.

## 4.2 Передача параметров в функцию

Обмен информацией между вызываемой и вызывающей функциями осуществляется с помощью *механизма передачи параметров*. `Список_переменных`, указанный в заголовке функции называется *формальными параметрами* или просто *параметрами* функции. `Список_переменных` в операторе вызова функции — это *фактические параметры* или *аргументы*.

*Механизм передачи параметров обеспечивает замену формальных параметров фактическими параметрами*, и позволяет выполнять функцию с различными данными. Между фактическими параметрами в операторе вызова функции и формальными параметрами в заголовке функции устанавливается взаимно однозначное соответствие. *Количество, типы и порядок следования формальных и фактических параметров должны совпадать.*

*Передача параметров* выполняется следующим образом. Вычисляются выражения, стоящие на месте фактических параметров. В памяти выделяется место под формальные параметры, в соответствии с их типами. Затем формальным параметрам присваиваются значения фактических. Выполняется проверка типов и при необходимости выполняется их преобразование. При несоответствии типов выдается диагностическое сообщение.

Передача параметров в функцию может осуществляться *по значению* и *по адресу*.

При передаче данных по значению функция работает с копиями фактических параметров, и доступа к исходным значениям аргументов у нее нет. При передаче данных по адресу в функцию передается не переменная, а ее адрес, и, следовательно, функция имеет доступ к ячейкам памяти, в которых хранятся значения аргументов. Таким образом, данные, переданные по значению, функция изменить не может, в отличие от данных, переданных по адресу.

Если требуется запретить изменение параметра внутри функции, используют модификатор `const`. Заголовок функции в общем виде будет выглядеть так:

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
тип имя_функции(const тип_переменной* имя_переменной, ...)
```

Например:

```
#include <iostream>
using namespace std;
int f1(int i)   //Данные передаются по значению
{
  return(i++);
}
int f2(int* j) //Данные передаются по адресу.
                //При подстановке фактического параметра,
                //для получения его значения,
                //применяется операция разадресации *.
{
  return((*j)++);
}
int f3(const int* k)   //Изменение параметра не предусмотрено.
{
  return(*k);
}
int main()
{
  int a;
  cout<<"a=";cin>>a;
  f1(a);
  cout<<"a="<<a<<"\n";
  f2(&a);   //Для передачи фактического параметра
            //используется операция взятия адреса &.
  cout<<"a="<<a<<"\n";
  f3(&a);
  cout<<"a="<<a<<"\n";
  return 0;
}
```

Результат работы программы:

```
Введено значение переменной a.
a=5
Значение переменной a после вызова функции f1 не изменилось.
a=5
Значение переменной a после вызова функции f2 изменилось.
a=6
Значение переменной a после вызова функции f3 не изменилось.
a=6
```

Удобно использовать передачу данных по адресу, если нужно чтобы функция изменяла значения переменных в вызывающей программе.

Далее приведен пример программы, в которой исходя из радианной меры некоторого угла вычисляется величина смежного с ним угла. На экран выводятся значения углов в градусной мере. Функция `degree` выполнят перевод из радианной меры в градусную[2]. Эта функция ничего не возвращает. Ее аргументами являются *значение* переменной `rad`, определяющее величину угла в радианах и *адреса* переменных `deg`, `min`, `sec`, в которых будут храниться вычисленные результаты — градусная мера угла.

```
#include <iostream>
#include <math.h>
#define PI 3.14159
using namespace std;
void degree (double rad, int *deg, int * min, int *sec)
{
  *deg= floor(rad*180/PI);
  *min=floor((rad*180/PI-(*deg))*60);
  *sec=floor(((rad*180/PI-(*deg))*60-(*min))*60);
}
int main()
{
  int DEG,MIN,SEC; double RAD;
  cout<<"Inpout:"<<endl;
  cout << "Value in radian A="; cin>>RAD;
  degree(RAD,&DEG,&MIN,&SEC);
  cout << DEG<<" "<<MIN<<" "<<SEC << endl;
  degree(PI-RAD,&DEG,&MIN,&SEC);
  cout << DEG<<" "<<MIN<<" "<<SEC << endl;
return 0;
}
```

## 4.3 Возврат результата с помощью оператора return

*Возврат результата* из функции в вызывающую ее функцию осуществляется оператором

`return (выражение);`

Работает оператор следующим образом. Вычисляется значение `выражения`, указанного после `return` и преобразуется к типу возвращаемого функцией значения. Выполнение функции завершается, а

[2] Чтобы найти градусную меру угла по заданной радианной нужно помножить число радиан на $\frac{180}{\pi}$ ; если из полученной дроби выделить целую часть, то получим градусы; если из числа полученного умножением оставшейся дробной части 60 выделить целую часть получим минуты; секунды вычисляются аналогично из дробной части минут.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

вычисленное значение передается в вызывающую функцию. Любые операторы, следующие в функции за оператором `return`, игнорируются. Программа продолжает свою работу с оператора следующего за оператором вызова данной функции.

Оператор `return` может отсутствовать в функциях типа `void`, если возврат происходит перед закрывающейся фигурной скобкой, и в функции `main`.

Также функция может содержать несколько операторов `return`, если это определено потребностями алгоритма. Например, в следующей программе, функция `equation` вычисляет корни квадратного уравнения. Если $a = 0$ (уравнение не является квадратным), то в программу передается значение равное -1, если дискриминант отрицательный (уравнение не имеет действительных корней), то 1, а если положительный, то вычисляются корни уравнения и в программу передается 0.

```
#include <iostream>
#include <math.h>
using namespace std;
int equation( float a,float b,float c,float *x1,float *x2)
{ float D=b*b-4*a*c;
  if (a==0) return -1;
  else if (D<0) return 1;
    else {
      *x1=(-b+sqrt(D))/2/a;
      *x2=(-b-sqrt(D))/2/a;
      return 0;
   }
}

int main()
{
  float A, B, C, X1, X2; int P;
  cout<<"Enter the coefficients of the equation:"<<endl;
  cout<<"A="; cin>>A;
  cout<<"B="; cin>>B;
  cout<<"C="; cin>>C;
  P=equation( A, B, C, &X1, &X2);
  if (P==-1)  cout<<"input Error"<<endl;
  else if (P==1)  cout<<"No real roots"<<endl;
  else  cout<<"X1="<<X1<<" X2="<<X2<<endl;
  return 0;
}
```

## 4.4 Решение задач с использованием функций

Рассмотрим несколько задач с применением функций.

**Задача 4.1.** Вводится последовательность из $N$ целых чисел, найти среднее арифметическое совершенных чисел и среднее геометрическое простых чисел последовательности.

Напомним, что целое число называется *простым*, если оно делится нацело только на само себя и единицу. Подробно алгоритм определения простого числа описан в задаче 3.15 (рис. 3.29). В этой задаче кроме простых чисел фигурируют совершенные числа. Число называется *совершенным*, если сумма всех делителей, меньших его самого равна этому числу. Алгоритм, с помощью которого можно определить делители числа, подробно рассмотрен в задаче 3.14 (рис. 3.28).

Для решения поставленной задачи понадобятся две функции:

- `prostoe` — определяет, является ли число простым, аргумент функции целое число $N$; функция возвращает 1, если число простое и 0 — в противном случае.;
- `soversh` — определяет, является ли число совершенным; входной параметр целое число $N$; функция возвращает 1, если число является совершенным и 0 — в противном случае.

```
#include <iostream>
#include <math.h>
unsigned int prostoe(unsigned int N) //Описание функции.
{
  //Функция определяет, является ли число простым.
  int i,pr;
  for(pr=1,i=2;i<=N/2;i++)
  if (N%i==0) {pr=0;break;}
  return pr;
}
unsigned int soversh(unsigned int N) //Описание функции.
{
  //Функция определяет, является ли число совершенным.
  unsigned int i,S;
  for(S=0,i=1;i<=N/2;i++)
    if (N%i==0) S+=i;     //Сумма делителей.
  if (S==N) return 1;
  else return 0;
}
using namespace std;
int main()
{
  unsigned int i,N,X,S,kp,ks;
  long int P;
  cout <<"N="; cin>>N;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  for(kp=ks=S=0,P=1,i=1;i<=N;i++)
  {
    cout <<"X="; cin >> X; //Вводится элемент последовательности.
    if (prostoe(X)) // X — простое число.
    {
      kp++; //Счетчик простых чисел.
      P*=X; //Произведение простых чисел.
    }
    if (soversh(X)) //X — совершенное число.
    {
      ks++; //Счетчик совершенных чисел.
      S+=X; //Сумма совершенных чисел.
    }
  }
  if (kp>0) //Если счетчик простых чисел больше нуля,
            //считаем среднее геометрическое и выводим его,
    cout<<"Geometric mean="<<pow(P,(float) 1/kp)<<endl;
  else      //в противном случае — сообщение об отсутствии простых
      чисел.
    cout<<"No prime numbers \n";
  if (ks>0) //Если счетчик совершенных чисел больше нуля,
            //считаем среднее арифметическое и выводим его,
    cout<<"Arithmetical mean="<<(float) S/ks<<endl;
  else     //в противном случае — сообщение об отсутствии совершенных
      чисел.
    cout<<"No perfect numbers \n";
return 0;
}
```

**Задача 4.2.** Вводится последовательность целых чисел, 0 — конец последовательности. Найти минимальное число среди простых чисел и максимальное, среди чисел, не являющихся простыми.

Для решения данной задачи создадим функцию `prostoe`, которая будет проверять, является ли число `N` простым. Входным параметром функции будет целое положительное число `N`. Функция будет возвращать значение `1`, если число простое и `0` — в противном случае. Алгоритм поиска максимума (минимума) подробно описан в задаче 3.19 (рис. 3.31).

Текст программы:

```
#include <iostream>
using namespace std;
unsigned int prostoe(unsigned int N)
{
  int i,pr;
  for(pr=1,i=2;i<=N/2;i++)
    if (N%i==0) {pr=0;break;}
  return pr;
```

```
}
int main()
{
  int kp=0,knp=0,min,max,N;
  for (cout << "N=", cin>>N; N!=0; cout<<"N=", cin>>N)
  //В цикле вводится элемент последовательности N.
    if (prostoe(N))  //N — простое число,
    {
      kp++;  //счетчик простых чисел.
      if (kp==1) min=N; //Предполагаем, что первое
                        //простое число минимально,
      else if (N<min) min=N; //если найдется меньшее
                             //число, сохраняем его.
    }
    else  //N — простым не является,
    {
      knp++;  //счетчик чисел не являющихся простыми.
      if (knp==1) max=N;  //Предполагаем, что первое
                          //не простое число максимально,
      else if (N>max) max=N; //если найдется большее число,
                             //сохраняем его.
    }
    if (kp>0) //Если счетчик простых чисел больше нуля,
      cout <<"min= "<<min<<"\t";  //выводим значение минимального
                                  //простого числа,
    else  //в противном случае —
      cout <<"Net prostih"; //сообщение об отсутствии простых чисел.
      if (knp>0) //Если счетчик чисел не являющихся простыми больше
          нуля,
        cout <<"max="<<max<<endl; //выводим значение максимального
                                  //числа,
      else //в противном случае —
        cout <<"Net ne prostih";  //сообщение об отсутствии чисел
                                  //не являющихся простыми.
  return 0;
}
```

**Задача 4.3.** Вводится последовательность из $N$ целых положительных чисел. В каждом числе найти наименьшую и наибольшую цифры[3].

Программный код к задаче 4.3.

```
#include <iostream>
using namespace std;
unsigned int Cmax(unsigned long long int P)
{
  unsigned int max;
  if (P==0) max=0;
```

[3]Выделение цифры из числа подробно описано в задаче 3.16

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  for (int i=1 ; P!=0;P/=10)
  {
    if (i==1) {max=P%10;i++;}
    if (P%10>max) max=P%10;
  }
  return max;
}
unsigned int Cmin(unsigned long long int P)
{
  unsigned int min;
  if (P==0) min=0;
  for (int i=1; P!=0;P/=10)
  {
    if (i==1) {min=P%10;i++;}
    if (P%10<min) min=P%10;
  }
  return min;
}
int main()
{
  unsigned int N, k;
  unsigned long long int X;
  for (cout<<"N=",cin>>N,k=1;k<=N;k++)
  {
    cout<<"X=";cin>>X;
    cout<<"Максимальная цифра="<<Cmax(X);
    cout<<"   Минимальная цифра="<<Cmin(X)<<endl;
  }
  return 0;
}
```

**Задача 4.4.** Вводится последовательность целых положительных чисел, 0 — конец последовательности. Определить сколько в последовательности чисел — палиндромов[4].

Алгоритм определения палиндрома подробно описан в задаче 3.18. Далее приведен программный код к задаче 4.4

```
#include <iostream>
using namespace std;
bool palindrom(unsigned long long int N)
{//Функция определяет является ли число N палиндромом,
 //возвращает true, если N — палиндром,
 //и false, в противном случае
  unsigned long long int M,R,S;
  int kol, i;
  for (R=1,kol=1,M=N;M/10>0; kol++,R*=10,M/=10);
```

[4]Палиндром — любой симметричный относительно своей середины набор символов.

```
  for(S=0,M=N, i=1;i<=kol;S+=M%10*R,M/=10,R/=10, i++);
    if (N==S) return true;
    else return false;
}
int main()
{ unsigned long long int X;
  int K;
  for (K=0,cout<<"X=",cin>>X;X!=0;cout<<"X=",cin>>X)
    if (palindrom(X)) K++;
    cout<<"Количество палиндромов равно K="<<K<<endl;
  return 0;
}
```

**Задача 4.5.** Заданы два числа — $X$ в двоичной системе счисления, $Y$ в системе счисления с основанием пять. Найти сумму этих чисел. Результат вывести в десятичной системе счисления.

Любое целое число $N$, заданное в $b$-ичной системе счисления, можно записать в виде:

$$N = P_n \cdot b^b + P_{n-1} \cdot b^{n-1} + ... + P_2 \cdot b^2 + P_1 \cdot b + P_0 = \sum_{i=0}^{n} P_i \cdot b^i,$$

где $b$ — *основание системы счисления* (целое положительное фиксированное число), $P_i$ — *разряд числа:* $0 \le P_i \le b-1, i = 0, 1, 2, ..., n$.

Например, $743_{10} = 7 \cdot 10^2 + 4 \cdot 10^1 + 3 \cdot 10^0 = 700 + 40 + 3 = 743_{10}$ $1011101_2 = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 16 + 8 + 4 + 1 = 93_{10}$

Создадим функцию для перевода целого числа $N$ заданного в $b$-ичной системе счисления в десятичную систему счисления.

```
#include <iostream>
using namespace std;
unsigned long long int DecNC(unsigned long long int N,
    unsigned int b)
{
//Функция выполняет перевод числа N, заданного в b-ичной
//системе счисления, в десятичную систему счисления
  unsigned long long int S,P;
  //for (S=0,P=1, i=1;N/10>0;S+=N%10*P, P*=b, N/=10);
  for (S=0,P=1;N!=0;S+=N%10*P,P*=b,N/=10);
  return S;
}
int main()
{
  unsigned long long int X,Y; unsigned int bX,bY;
  cout<<"X=";cin>>X;   //Ввод числа X.
  cout<<"b=";cin>>bX;  //Ввод основания с/с.
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  cout<<"Y=";cin>>Y;   //Ввод числа X.
  cout<<"b=";cin>>bY;  //Ввод основания с/с.
  //Вывод заданных чисел в десятичной с/с.
  cout<<X<<"("<<bX<<")="<<DecNC(X,bX)<<"(10)"<<endl;
  cout<<Y<<"("<<bY<<")="<<DecNC(Y,bY)<<"(10)"<<endl;
  //Вычисление суммы и вывод результата.
  cout<<X<<"("<<bX<<")+"<<Y<<"("<<bY<<")=";
  cout<<DecNC(X,bX)+DecNC(Y,bY)<<"(10)"<<endl;
  return 0;
}
```

**Задача 4.6.** Задано число $X$ в десятичной системе счисления. Выполнить перевод числа в системы счисления с основанием 2, 5 и 7.

Вообще, для того чтобы перевести целое число из одной системы счисления в другую необходимо выполнить следующие действия:

1. поделить данное число на основание новой системы счисления;
2. перевести остаток от деления в новую систему счисления; получается младший разряд нового числа;
3. если частное от деления больше основания новой системы, продолжать деление, как указано в п.1; новый остаток, переведенный в новую систему счисления, дает второй разряд числа и т. д.

На рис. 4.1 приведен пример «ручного» перевода числа 256 заданного в десятичной системе счисления в восьмеричную. В результате получим, $256_{(10)} = 400_{(8)}$.

$$\begin{array}{r|l} 256 & 8 \\ \hline -256 & 32 \quad \vert\, 8 \\ \hline 0 & -32 \quad \vert\, 4 \\ \hline & 0 \end{array}$$

Рис. 4.1. Пример перевода числа в новую систему счисления

Далее приведен текст программы, реализующей решение задачи 4.6.

```
#include <iostream>
using namespace std;
unsigned long long int NC(unsigned long long int N, unsigned
    int b)
{
  unsigned long long int S,P;
  for (S=0,P=1;N!=0;S+=N%b*P,P*=10,N/=b);
    return S;
```

```
}
int main()
{
  unsigned long long int X;
  cout<<"X=";cin>>X;   //Ввод числа X.
  //Перевод числа X в заданные системы счисления.
  cout<<X<<"(10)="<<NC(X,2)<<"(2)"<<endl;
  cout<<X<<"(10)="<<NC(X,5)<<"(5)"<<endl;
  cout<<X<<"(10)="<<NC(X,7)<<"(7)"<<endl;
  return 0;
}
```

**Задача 4.7.** Найти корни уравнения $x^2 - \cos(5 \cdot x) = 0$.

Для решения задачи использовать:

1. *метод половинного деления*,
2. *метод хорд*,
3. *метод касательных* (метод Ньютона),
4. *метод простой итерации*.

Оценить степень точности предложенных *численных методов*, определив за сколько итераций был найден корень уравнения. Вычисления проводить с точностью $\varepsilon = 10^{-3}$.

Вообще говоря, *аналитическое решение уравнения*

$$f(x) = 0 \tag{4.1}$$

можно найти только для узкого класса функций. В большинстве случаев приходится решать уравнение (4.1) *численными методами*. Численное решение уравнения (4.1) проводят в два этапа: сначала необходимо *отделить корни уравнения*, т.е. найти достаточно тесные промежутки, в которых содержится только один корень, эти промежутки называют *интервалами изоляции корня*; на втором этапе проводят уточнение отделенных корней, т.е. *находят корни с заданной точностью*.

Интервал можно выделить, изобразив график функции, или каким-либо другим способом. Но все способы основаны на следующем свойстве непрерывной функции: если функция $f(x)$ непрерывна на интервале $[a, b]$ и на его концах имеет различные знаки, $f(a) \cdot f(b) < 0$, то между точками имеется хотя бы один корень. Будем считать интервал настолько малым, что в нем находится только один корень. Рассмотрим самый простой способ уточнения корней.

Графическое решение задачи 4.7 показано на рис. 4.2. Так как функция $f(x) = x^2 - \cos(5 \cdot x)$ дважды пересекает ось абсцисс, можно сделать вывод о наличии в уравнении $x^2 - \cos(5 \cdot x) = 0$ двух корней.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Первый находится на интервале $[-0.4; -0.2]$ , второй принадлежит отрезку $[0.2; 0.4]$ .

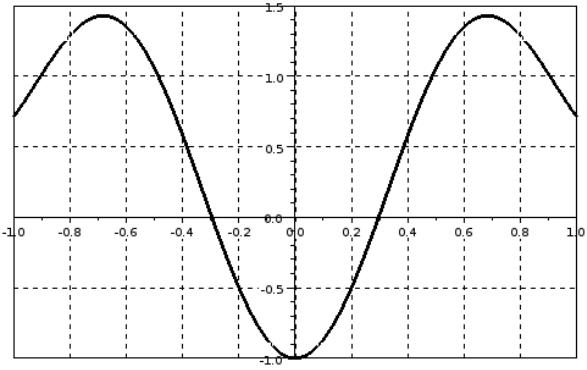


Рис. 4.2. Геометрическое решение задачи 4.7

Рассмотрим, предложенные в задаче *численные методы* решения нелинейных уравнений.

*Метод половинного деления (дихотомии).* Пусть был выбран интервал изоляции $[a, b]$ (рис. 4.3). Примем за первое приближение корня точку $c$, которая является серединой отрезка $[a, b]$ . Далее будем действовать по следующему алгоритму:

1. Находим точку $c = \frac{a+b}{2}$;
2. Находим значение $f(c)$;
3. Если $f(a) \cdot f(c) < 0$ , то корень лежит на интервале $[a, c]$ , иначе корень лежит на интервале $[c, b]$ ;
4. Если величина интервала меньше либо равна $\varepsilon$, то найден корень с точностью $\varepsilon$ , иначе возвращаемся к п.1.

Итак, для вычисления одного из корней уравнения $x^2 - \cos(5 \cdot x) = 0$ методом половинного деления достаточно знать интервал изоляции корня $a = 0.2; b = 0.4$ и точность вычисления $\varepsilon = 10^{-3}$ .

Блок-схема алгоритма решения уравнения методом дихотомии приведена на рис. 4.4. Понятно, что здесь $c$ — корень заданного уравнения.

Однако, несмотря на простоту, такое последовательное сужение интервала проводится редко, так как требует слишком большого количества вычислений. Кроме того, этот способ не всегда позволяет

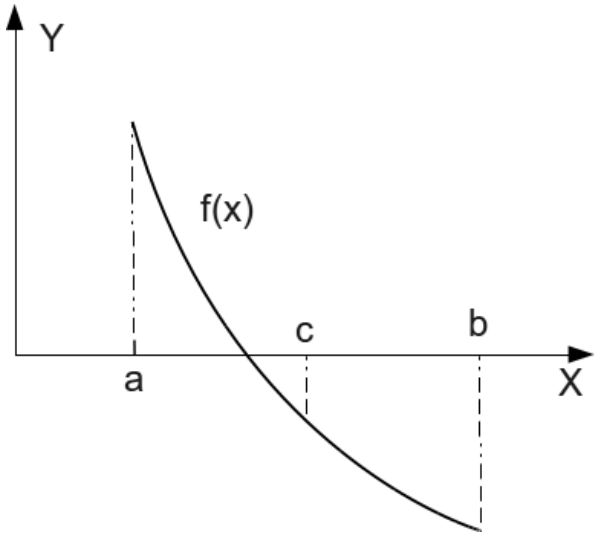


Рис. 4.3. Графическая интерпретация метода половинного деления

найти решение с заданной точностью. Рассмотрим другие способы уточнения корня. При применении этих способов будем требовать, чтобы функция $f(x)$ удовлетворяла следующим условиям на интервале $[a, b]$ :

1. функция $f(x)$ непрерывна вместе со своими производными первого и второго порядка. Функция $f(x)$ на концах интервала $[a, b]$ имела разные знаки $f(a) \cdot f(b) < 0$ ;
2. первая и вторая производные $f'(x)$ и $f''(x)$ сохраняют определённый знак на всем интервале $[a, b]$ .

*Метод хорд.* Этот метод отличается от метода дихотомии тем, что очередное приближение берем не в середине отрезка, а в точке пересечения с осью $X$ (рис. 4.5) прямой, соединяющей точки $(a, f(a))$ и $(b, f(b))$.

Запишем уравнение прямой, проходящей через точки с координатами $(a, f(a))$ и $(b, f(b))$ :

$$\frac{y - f(a)}{f(b) - f(a)} = \frac{x - a}{b - a}, \quad y = \frac{f(b) - f(a)}{b - a} \cdot (x - a) + f(a) \tag{4.2}$$

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

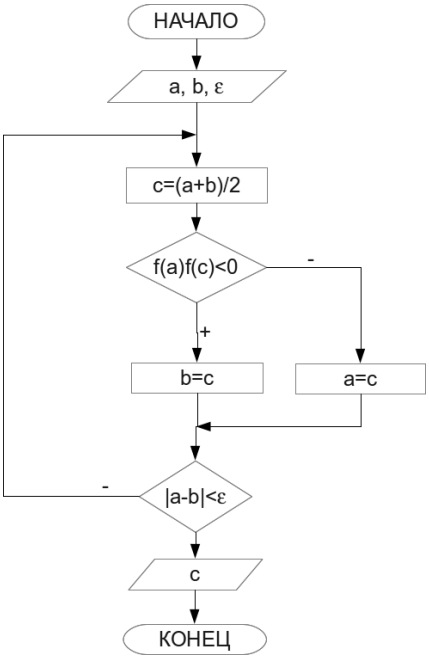


Рис. 4.4. Алгоритм решения уравнения методом дихотомии

Прямая, заданная уравнением (4.2), пересекает ось $X$ при условии $y = 0$.

Найдем точку пересечения хорды с осью $X$:
$y = \frac{f(b)-f(a)}{b-a} \cdot (x-a) + f(a)$, $x = a - \frac{f(a)\cdot(b-a)}{f(b)-f(a)}$,
итак, $c = a - \frac{f(a)}{f(b)-f(a)}(b-a)$.

Далее необходимо вычислить значение функции в точке $c$. Это и будет приближенное значение корня уравнения.

Для вычисления одного из корней уравнения $x^2 - \cos(5 \cdot x) = 0$ методом хорд достаточно знать интервал изоляции корня, например, $a = 0.2; b = 0.4$ и точность вычисления $\varepsilon = 10^{-3}$ . Блок-схема метода представлена на рис. 4.6.

*Метод касательных* (*метод Ньютона*). В одной из точек интервала $[a; b]$, пусть это будет точка $a$, проведем касательную (рис. 4.7).

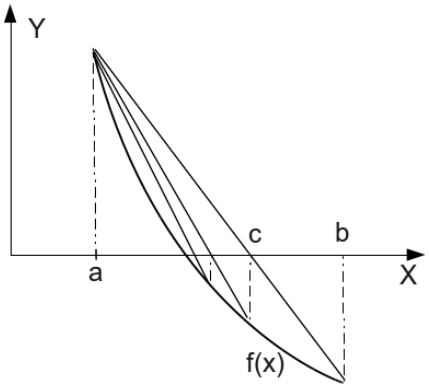


Рис. 4.5. Графическая интерпретация метода хорд

Запишем уравнение этой прямой:

$$y = k \cdot x + m \tag{4.3}$$

Так как эта прямая является касательной, и она проходит через точку $(c, f(c))$ , то $k = f'(c)$ .

Следовательно,

$y = f'(x) \cdot x + m, f(c) = f'(c) \cdot c + m, m = f(c) - c \cdot f'(c),$

$y = f'(c) \cdot x + f(c) - c \cdot f'(c), y = f'(c) \cdot (x - c) + f(c)$ .

Найдем точку пересечения касательной с осью $X$:

$f'(c) \cdot (x - c) + f(c) = 0, \;\; x = c - \frac{f(c)}{f'(c)}$

Если $|f(x)| < \varepsilon$, то точность достигнута, и точка $x$ — решение; иначе необходимо переменной $c$ присвоить значение $x$ и провести касательную через новую точку $c$; так продолжать до тех пор, пока $|f(x)|$ не станет меньше $\varepsilon$ . Осталось решить вопрос, что выбрать в качестве точки начального приближения $c$.

В этой точке должны совпадать знаки функции и ее второй производной. А так как нами было сделано допущение, что вторая и первая производные не меняют знак, то можно проверить условие $f(x) \cdot f''(x) > 0$ на обоих концах интервала и в качестве начального приближения взять ту точку, где это условие выполняется.

Здесь, как и в предыдущих методах, для вычисления одного из корней уравнения $x^2 - \cos(5 \cdot x) = 0$ достаточно знать интервал изоляции корня, например, $a = 0.2; b = 0.4$ и точность вычисления $\varepsilon = 10^{-3}$.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

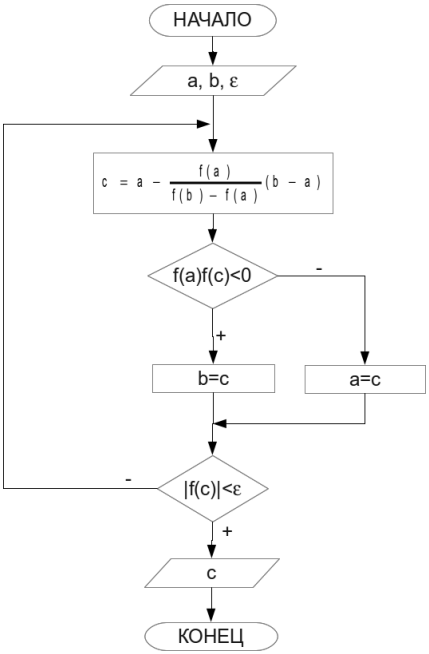


Рис. 4.6. Алгоритм метода хорд

Блок-схема метода Ньютона представлена на рис. 4.8. Понятно, что для реализации этого алгоритма нужно найти первую и вторую производные функции $f(x) = x^2 - \cos(5 \cdot x)$: $f'(x) = 2 \cdot x + 5 \cdot \sin(5 \cdot x)$ , $f(x) = 2 + 25 \cdot cos(5 \cdot x)$.

*Метод простой итерации.* Для решения уравнения этим методом необходимо записать уравнение (4.1) в виде $x = \phi(x)$, задать начальное приближение $x_0 \in [a; b]$ и организовать следующий итерационный вычислительный процесс

$x_{k+1} = \phi(x_k), k = 0, 1, 2, ...$

Вычисление прекратить, если $|x_{k+1} - x_k| < \varepsilon$ ($\varepsilon$ — точность).

Если неравенство $|\phi'(x)| < 1$ выполняется на всем интервале $[a; b]$, то последовательность $x_0, x_1, x_2, ..., x_n, ...$ сходится к решению $x^*$ (т.е. $\lim\limits_{k \to \infty} x_k = x^*$).

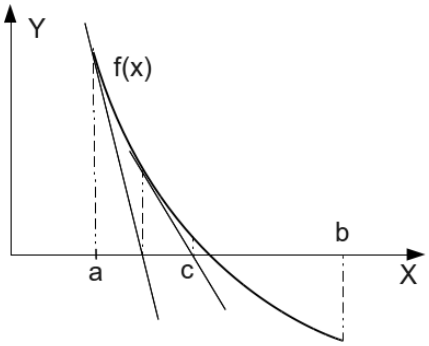


Рис. 4.7. Графическая интерпретация метода касательных

Значение функции $\phi(x)$ должно удовлетворять условию $|\phi'(x)| < 1$ для того, чтобы можно было применить метод простых итераций. Условие $|\phi'(x)| < 1$ является *достаточным условием сходимости* метода простой итерации.

Уравнение (4.1) можно привести к виду $x = \phi(x)$ следующим образом. Умножить обе части уравнения $f(x) = 0$ на число $\lambda$. К обеим частям уравнения $\lambda \cdot f(x) = 0$ добавить число $x$. Получим $x = x + \lambda \cdot f(x)$. Это и есть уравнение вида $x = \phi(x)$, где

$$\phi(x) = x + \lambda \cdot f(x) \tag{4.4}$$

Необходимо чтобы неравенство $|\phi'(x)| < 1$ выполнялось на интервале $[a; b]$, следовательно, $|\phi'(x)| = |1 + \lambda \cdot f'(x)|$ и $|1 + \lambda \cdot f'(x)| < 1$ ($|1 + \lambda \cdot f'(a)| < 1$, $|1 + \lambda \cdot f'(b)| < 1$), а значит, с помощью *подбора параметра* $\lambda$ можно добиться выполнения *условия сходимости*.

Для вычисления корней уравнения $x^2 - \cos(5 \cdot x) = 0$ воспользуемся графическим решением (рис. 4.2) и определим интервал изоляции одного из корней, например, $a = 0.2; b = 0.4$. Подберем значение $\lambda$ решив неравенство $|1 + \lambda \cdot f'(x)| < 1$:

$|1 + \lambda \cdot f'(a)| < 1$ и $|1 + \lambda \cdot f'(b)| < 1$,

$f(x) = x^2 - \cos(5 \cdot x), f'(x) = 2 \cdot x + 5 \cdot \sin(5 \cdot x)$,

$f'(a) = 2 \cdot 0.2 + 5 \cdot \sin(5 \cdot 0.2) \approx 4.6$ , $f'(b) = 2 \cdot 0.4 + 5 \cdot \sin(5 \cdot 0.4) \approx 5.35$,

$|1 + \lambda \cdot 4.6| < 1$ и $|1 + \lambda \cdot 5.35| < 1$.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

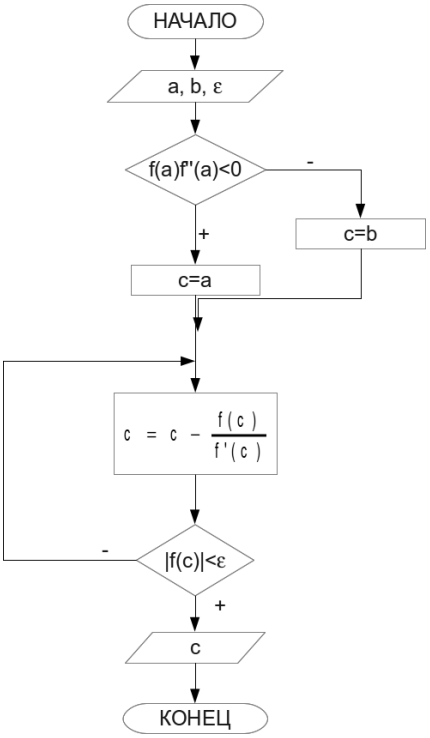


Рис. 4.8. Алгоритм метода Ньютона

$$\left\{\begin{array}{l}\left\{\begin{array}{l}1+4.6\cdot\lambda<1\\1+4.6\cdot\lambda>-1\end{array}\right.\\\left\{\begin{array}{l}\lambda<0\\\lambda>-0.37\end{array}\right.\end{array}\right. \Rightarrow \left\{\begin{array}{l}\left\{\begin{array}{l}\lambda<0\\\lambda>-0.4\end{array}\right.\\\left\{\begin{array}{l}\lambda<0\\\lambda>-0.37\end{array}\right.\end{array}\right. \Rightarrow \left\{\begin{array}{l}\lambda\in(-0.4;0)\\\lambda\in(-0.37;0)\end{array}\right.$$

и, следовательно, $\lambda \in (-0.37; 0)$.

Таким образом, исходными данными для программы будут начальное значение корня уравнения $x_0 = 0.2$ , значение параметра $\lambda$ (пусть $\lambda = -0.2$), и точность вычислений $\varepsilon = 0.001$ .

Для вычисления второго корня заданного уравнения параметр $\lambda$ подбирают аналогично.

Блок-схема метода простой итерации приведена на рис. 4.9.

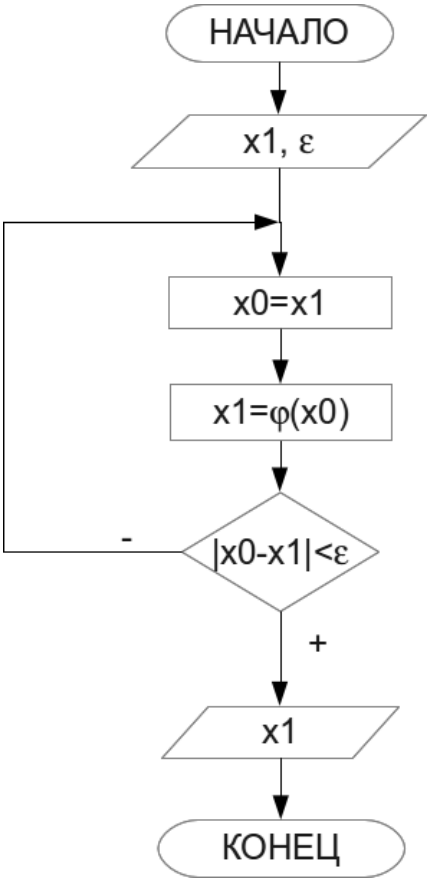


Рис. 4.9. Алгоритм метода простой итерации

Далее представлен текст программы, реализующий решение задачи 4.7.

```
#include <iostream>
#include <math.h>
using namespace std;
//Функция, определяющая левую часть уравнения f(x) = 0.
double f(double x)
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
{
  return(x*x-cos(5*x));
}
//Функция, реализующая метод половинного деления.
int Dichotomy(double a, double b, double *c, double eps)
{int k=0;
  do
  {
    *c=(a+b)/2;
    if (f(*c)*f(a)<0) b=*c;
    else a=*c;
    k++;
  }
  while (fabs(a-b)>=eps);
  return k;
}
//Функция, реализующая метод хорд.
int Chord(double a, double b, double *c, double eps)
{int k=0;
  do
  {
    *c=a-f(a)/(f(b)-f(a))*(b-a);
    if (f(*c)*f(a)>0) a=*c;
    else b=*c;
    k++;
  }
  while (fabs(f(*c))>=eps);
  return k;
}
double f1(double x)   //Первая производная функции f(x).
{
  return(2*x+5*sin(5*x));
}
double f2(double x)   //Вторая производная функции f(x).
{
  return(2+25*cos(5*x));
}
//Функция, реализующая метод касательных.
int Tangent(double a, double b, double *c, double eps)
{int k=0;
  if (f(a)*f2(a)>0) *c=a;
  else *c=b;
  do
  {
    *c=*c-f(*c)/f1(*c);
    k++;
  }
  while (fabs(f(*c))>=eps);
  return k;
}
```

```
double fi(double x,double L) //Функция, заданная выражением 4.4.
{
  return(x+L*f(x));
}
//Функция, реализующая метод простой итерации.
int Iteration(double *x, double L, double eps)
{int k=0; double x0;
  do
  {
    x0=*x;
    *x=fi(x0,L);
    k++;
  }
  while (fabs(x0-*x)>=eps);
  return k;
}
int main()
{
  double A, B, X, P;
  double ep=0.001;  //Точность вычислений.
  int K;
  cout<<"a="; cin>>A;  //Интервал изоляции корня.
  cout<<"b="; cin>>B;
  cout<<"Решение уравнения x^2-cos(5*x)=0."<<endl;
  cout<<"Метод дихотомии:"<<endl;
  K=Dichotomy(A,B,&X,ep);
  cout<<"Найденное решение x="<<X;
  cout<<", количество итераций k="<<K<<endl;
  cout<<"Метод хорд:"<<endl;
  K=Chord(A,B,&X,ep);
  cout<<" Найденное решение x="<<X;
  cout<<", количество итераций k="<<K<<endl;
  cout<<"Метод касательных:"<<endl;
  K=Tangent(A,B,&X,ep);
  cout<<" Найденное решение x="<<X;
  cout<<", количество итераций k="<<K<<endl;
  cout<<"Метод простой итерации:"<<endl;
  X=A;
  cout<<"L="; cin>>P;
  K=Iteration(&X,P,ep);
  cout<<" Найденное решение x="<<X;
  cout<<", количество итераций k="<<K<<endl;
  return 0;
}
```

Результаты работы программы:

```
a=0.2
b=0.4
Решение уравнения x^2-cos(5*x)=0.
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
Метод дихотомии:
Найденное решение x=0.296094, количество итераций k=8
Метод хорд:
Найденное решение x=0.296546, количество итераций k=2
Метод касательных:
Найденное решение x=0.296556, количество итераций k=2
Метод простой итерации:
L=-0.2
Найденное решение x=0.296595, количество итераций k=3
```

## 4.5 Рекурсивные функции

Под *рекурсией* в программировании понимают функцию, которая вызывает сама себя. *Рекурсивные функции* чаще всего используют для компактной реализации рекурсивных алгоритмов. Классическими рекурсивными алгоритмами могут быть возведение числа в целую положительную степень, вычисление факториала. С другой стороны любой рекурсивный алгоритм можно реализовать без применения рекурсий. Достоинством рекурсии является компактная запись, а недостатком расход памяти на повторные вызовы функций и передачу параметров, существует опасность переполнения памяти.

Рассмотрим применение рекурсии на примерах [7, 8].

**Задача 4.8.** Вычислить факториал числа $n$.

Вычисление факториала подробно рассмотрено в задаче 3.11 (рис. 3.25). Для решения этой задачи с применением рекурсии создадим функцию `factoial`, алгоритм которой представлен на рис. 4.10.

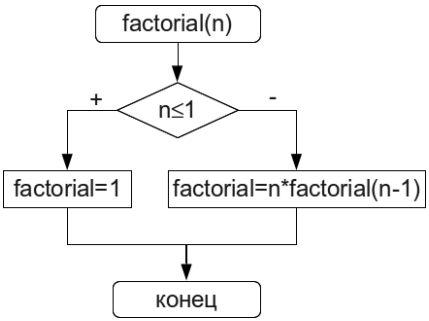


Рис. 4.10. Рекурсивный алгоритм вычисления факториала

Текст программы с применением рекурсии:

```
#include <iostream>
using namespace std;
long int factorial(int n)
{
  if (n<=1)
    return(n);
  else
  return(n*factorial(n-1));
}
int main()
{
  int i; long int f;
  cout<<"i="; cin>>i;
  f=factorial(i);
  cout<<i<<"!="<<f<<"\n";
  return 0;
}
```

**Задача 4.9.** Вычислить $n$-ю степень числа $a$ ($n$ — целое число).

Результатом возведения числа $a$ в целую степень $n$ является умножение этого числа на себя $n$ раз. Но это утверждение верно, только для положительных значений $n$. Если $n$ принимает отрицательные значения, то $a^{-n} = \frac{1}{a^n}$. В случае $n = 0$, $a^0 = 1$.

Для решения задачи создадим рекурсивную функцию `stepen`, алгоритм которой представлен на рис. 4.11.

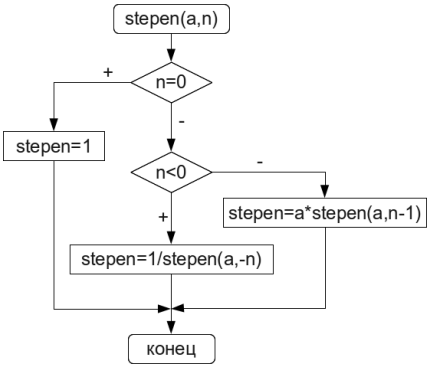


Рис. 4.11. Рекурсивный алгоритм вычисления степени числа

Текст программы с применением рекурсии:

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
#include <iostream>
using namespace std;
float stepen(float a, int n)
{
  if (n==0)
    return(1);
  else if (n<0)
    return(1/stepen(a,-n));
  else
    return(a*stepen(a,n-1));
}
int main()
{
  int i; float s,b;
  cout<<"b="; cin>>b;
  cout<<"i="; cin>>i;
  s=stepen(b,i);
  cout<<"s="<<s<<"\n";
  return 0;
}
```

**Задача 4.10.** Вычислить $n$-е число Фибоначчи.

Если нулевой элемент последовательности равен нулю, первый — единице, а каждый последующий представляет собой сумму двух предыдущих, то это *последовательность чисел Фибоначчи* (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... ).

Алгоритм рекурсивной функции `fibonachi` изображен на рис. 4.12.

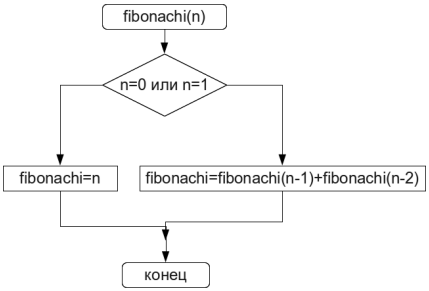


Рис. 4.12. Рекурсивный алгоритм вычисления числа Фибоначчи

Текст программы:

```
#include <iostream>
```

```
using namespace std;
long int fibonachi(unsigned int n)
{
  if ((n==0)||(n==1))
    return(n);
  else
    return(fibonachi(n-1)+fibonachi(n-2));
}
int main()
{
  int i; long int f;
  cout<<"i="; cin>>i;
  f=fibonachi(i);
  cout<<"f="<<f<<"\n";
  return 0;
}
```

## 4.6 Перегрузка функций

Язык C++ позволяет связать с одним и тем же именем функции различные определения, то есть возможно существование нескольких функций с одинаковым именем. У этих функций может быть разное количество параметров или разные типы параметров. Создание двух или более функций с одним и тем же именем называется *перегрузкой имени функции*. Перегруженные функции создают, когда одно и то же действие следует выполнить над разными типами входных данных.

В приведённом далее тексте программы три функции с именем `Pow`. Первая выполняет операцию возведения вещественного числа $a$ в дробную степень $n = \frac{k}{m}$, где $k$ и $m$ — целые числа. Вторая возводит вещественное число $a$ в целую степень $n$, а третья — целое число $a$ в целую степень[5] $n$. Какую именно функцию вызвать компилятор определяет по типу фактических параметров. Так, если $a$ — вещественное число, а $k$ — целое, то оператор `Pow(a,k)` вызовет вторую функцию, так как она имеет заголовок `float Pow(float a, int n)`. Команда `Pow((int)a,k)` приведет к вызову третьей функции `float Pow(int a, int n)`, так как вещественная переменная $a$ преобразована к целому типу. Первая функция `float Pow(float a, int k, int m)` имеет три параметра, значит, обращение к ней осуществляется командой `Pow(a,k,m)`.

[5]Как известно операция $a^n$ не определена при $a = 0$ и $n = 0$, а так же при возведении отрицательного значения $a$ в дробную степень $n = \frac{k}{m}$, где $m$ — четное число (п. 2.7).Пусть наши функции в этих случаях возвращают 0.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
#include <iostream>
using namespace std;
#include <math.h>
float Pow(float a, int k, int m) //Первая функция
{
  cout<<"Функция 1 \t";
  if (a==0)
    return 0;
  else if (k==0)
        return 1;
       else if (a>0)
              return exp((float)k/m*log(a));
            else if (m%2!=0)
              return -(exp((float)k/m*log(-a)));
}
float Pow(float a, int n)  //Вторая функция
{
  float p; int i;
  cout<<"Функция 2 \t";
  if (a==0)
    return 0;
  else if (n==0)
    return 1;
    else if (n<0)
    {
      n=-n;
      p=1;
      for(i=1;i<=n;i++)
        p*=a;
      return (float)1/p;
    }
      else
      {
        p=1;
        for(i=1;i<=n;i++)
          p*=a;
        return p;
      }
}
float Pow(int a, int n) //Третья функция
{
  int i,p;
  cout<<"Функция 3 \t";
  if (a==0)
    return 0;
  else if (n==0)
    return 1;
    else if (n<0)
   {
     n=-n;
```

```
      p=1;
      for(i=1;i<=n;i++)
        p*=a;
      return (float)1/p;
    }
      else
      {
        p=1;
        for(i=1;i<=n;i++)
          p*=a;
        return p;
      }
}
int main()
{
  float a; int k,n,m;
  cout<<"a="; cin>>a;
  cout<<"k="; cin>>k;
  cout<<"s="<<Pow(a,k)<<"\n";          //Вызов 2-й функции.
  cout<<"s="<<Pow((int)a,k)<<"\n";     //Вызов 3-й функции.
  cout<<"a="; cin>>a;
  cout<<"k="; cin>>k;
  cout<<"m="; cin>>m;
  cout<<"s="<<Pow(a,k,m)<<endl;        //Вызов 1-й функции.
  return 0;
}
```

Результаты работы программы

```
a=5.2
k=3
Функция 2  s=140.608
Функция 3  s=125
a=-8
k=1
m=1
Функция 1  s=-8

a=5.2
k=-3
Функция 2  s=0.00711197
Функция 3  s=0.008
a=-8
k=1
m=3
Функция 1  s=-2
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

## 4.7 Шаблоны функций

*Шаблон* — это особый вид функции. С помощью шаблона функции можно определить алгоритм, который будет применяться к данным различных типов. Механизм работы шаблона заключается в том, что на этапе компиляции конкретный тип данных передаётся в функцию в виде параметра.

Простейшую функцию–шаблон в общем виде можно записать так [6]:

```
template <class Type> \Sys{заголовок}
{
  тело функции
}
```

Обычно в угловых скобках указывают список используемых в функции типов данных. Каждый тип предваряется служебным словом `class`. В общем случае в списке могут быть не только типы данных, но и имена переменных.

Рассмотрим пример шаблона поиска наименьшего из четырех чисел.

```
#include <iostream>
using namespace std;
//Определяем абстрактный тип данных с помощью служебного слова Type.
template <class Type>
Type minimum(Type a, Type b, Type c, Type d)
{ //Определяем функцию с использованием типа данных Type.
  Type min=a;
  if (b<min) min=b;
  if (c<min) min=c;
  if (d<min) min=d;
  return min;
}
int main()
{
  int ia,ib,ic,id,mini; float ra,rb,rc,rd,minr;
  cout<<"Vvod 4 thelih chisla\t";
  cin>>ia>>ib>>ic>>id;
  mini=minimum(ia,ib,ic,id); //Вызов функции minimum,
                    //в которую передаем 4 целых значения.
  cout<<"\n"<<mini<<"\n";
  cout<<"Vvod 4 vecshestvenih chisla\t"; cin>>ra>>rb>>rc>>rd;
  minr=minimum(ra,rb,rc,rd); //Вызов функции minimum,
                  //в которую передаем 4 вещественных значения.
  cout<<"\n"<<minr<<"\n";
  return 0;
}
```

## 4.8 Область видимости переменных в функциях

Как известно (п. 2.8), по месту объявления переменные в языке C++ делятся на три класса: локальные, глобальные и переменные, описанные в списке формальных параметров функций. Все эти переменные имеют разную область видимости.

*Локальные переменные* объявляются внутри функции и доступны только в ней. О таких переменных говорят, что они имеют локальную видимость, то есть, видимы только внутри функции.

*Глобальные переменные* описывают вне всех функций. Поскольку они доступны из любой точки программы, то их область видимости охватывает весь файл.

Одно и тоже имя может использоваться при определении глобальной и локальной переменной. В этом случае в теле функции локальная переменная имеет преимущество и «закрывает» собой глобальную. Вне этой функции «работает» глобальное описание переменной.

Из функции, где действует локальное описание переменной можно обратиться к глобальной переменной с таким же именем, используя *оператор расширения области видимости*

`::переменная;`

Рассмотрим пример:

```
#include <iostream>
using namespace std;
float pr=100.678; //Переменная pr определена глобально.
int prostoe (int n)
{
  int pr=1,i; //Переменная pr определена локально.
  if (n<0) pr=0;
  else
    for (i=2;i<=n/2;i++)
      if (n%i==0){pr=0;break;}
  cout<<"local pr="<<pr<<"\n";   //Вывод локальной переменной.
  cout<<"global pr="<<::pr<<"\n";   //Вывод глобальной переменной.
  return pr;
}
int main()
{
  int g;
  cout<<"g="; cin>>g;
  if (prostoe(g)) cout<<"g - prostoe \n";
  else cout<<"g - ne prostoe \n";
  return 0;
}
```

Результаты работы программы:

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
g=7
local pr=1  //Локальная переменная.
global pr=100.678  //Глобальная переменная.
g - prostoe
```

## 4.9 Функция main(). Параметры командной строки

Итак, любая программа на C++ состоит из одной или нескольких функций, причем одна из них должна обязательно носить имя `main` (основной, *главный*). Именно это функции передается управление после запуска программы. Как любая функция, `main` может принимать параметры и возвращать значения. У функции `main` две формы записи:

- без параметров: `тип main() {тело функции}`,
- и с двумя параметрами: `тип main(int argc, char *argv[]) {тело функции}`.

Первый параметр `argc` определяет количество параметров, передаваемых в функцию `main` из командной строки. Второй параметр `argv` — указатель на массив указателей типа `char` (массив строк). Каждый элемент массива ссылается на отдельный параметр командной строки. При стандартном запуске программы `argc` равно 1, `argv` — массив из одного элемента, этим элементом является — имя запускаемого файла.

Рассмотрим следующую программу.

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main(int argc, char *argv[])
{
  int i;
  cout<<"В командной строке "<<argc<<" аргументов\n";
  for(i=0;i<argc;i++)
  cout<<"Аргумент № "<<i<<" "<<argv[i]<<endl;
  return 0;
}
```

Текст программы хранится в файле `1.cpp`. При стандартном запуске программа выведет следующую информацию:

```
В командной строке 1 аргументов
Аргумент № 0 ./1
```

Программа выводит количество параметров командной строки и последовательно все параметры. При стандартном запуске – количество аргументов командной строки — 1, этим параметром является имя запускаемого файла (в нашем случае, имя запускаемого файла — `./1`).

Запустим программу следующим образом

```
./1 abc 34 6 + 90 Вася Маша
```

Результаты работы программы представлены ниже.

```
В командной строке 8 аргументов
Аргумент № 0 ./1
Аргумент № 1 abc
Аргумент № 2 34
Аргумент № 3 6
Аргумент № 4 +
Аргумент № 5 90
Аргумент № 6 Вася
Аргумент № 7 Маша
```

Рассмотрим приложение, в которое в качестве параметров командной строки передается `число1, операция, число2`. Функция выводит

`число1 операция число2`.

Текст программы приведен на ниже[6]

```
#include <iostream>
#include <stdlib.h>
#include <cstring>
using namespace std;
int main(int argc, char **argv)
{
//Если количество параметров больше или равно 4, то были
//введены два числа и знак операции.
  if (argc>=4)
//Если операция *, то выводим число1*число2.
  {
    if (!strcmp(argv[2],"*")) cout<<atof(argv[1])*atof(argv
        [3])<<endl;
    else
//Если операция +, то выводим число1+число2.
      if (!strcmp(argv[2],"+")) cout<<atof(argv[1])+atof(argv
          [3])<<endl;
```

[6]Функция `atof` преобразовывает строку символов в вещественное число, если преобразование невозможно, то результатом функции `atof` будет число 0.0. Функция `strcmp` сравнивает две строки и возвращает 0, в случае совпадения строк. Подробнее об этих функциях можно прочесть в главе, посвящённой строкам.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
      else
//Если операция -, то выводим число1-число2.
        if (!strcmp(argv[2],"-")) cout<<atof(argv[1])-atof(
          argv[3])<<endl;
        else
//Если операция /, то выводим число1/число2.
          if (!strcmp(argv[2],"/")) cout<<atof(argv[1])/atof(
            argv[3])<<endl;
          else cout<<"неправильный знак операции"<<endl;
  }
  else
    cout<<"недостаточное количество операндов"<<endl;
  return 0;
}
```

Ниже приведены варианты запуска программы и результаты её работы[7]. Предлагаем читателю самостоятельно разобраться с результатами всех тестовых запусков приложения.

```
./4 1.3 + 7.8
9.1
./4 1.3 - 7.8
-6.5
./4 1.3 / 7.8
0.166667
./4 1.3 \* 7.8
10.14
./4 1.3 % 7.8
неправильный знак операции
./4 1.3+ 7.8
недостаточное количество операндов
```

## 4.10 Задачи для самостоятельного решения

### 4.10.1 Применение функций при работе с последовательностями чисел

Разработать программу на языке C++ для следующих заданий:

1. Вводится последовательность целых положительных чисел, 0 — конец последовательности. Для каждого элемента последовательности определить и вывести на экран число, которое получится после записи цифр исходного числа в обратном порядке.

[7] Текст программы хранится в файле `4.cpp`. Имя исполняемого файла `./4` (ОС Lnux)

2. Вводится последовательность целых чисел, 0 — конец последовательности. Определить содержит ли последовательность хотя бы одно *совершенное число*. Совершенное число равно сумме всех своих делителей, не превосходящих это число. Например, 6=1+2+3 или 28=1+2+4+7+14.
3. Вводится последовательность из N целых положительных элементов. Определить содержит ли последовательность хотя бы одно *простое число*. Простое число не имеет делителей, кроме единицы и самого себя.
4. Вводится последовательность из N целых положительных элементов. Посчитать количество чисел *палиндромов*. Числа палиндромы симметричны относительно своей середины, например, 12021 или 454.
5. Вводится последовательность из N целых положительных элементов. Подсчитать количество совершенных и простых чисел в последовательности.
6. Поступает последовательность целых положительных чисел, 0 — конец последовательности. Определить в каком из чисел больше всего делителей.
7. Поступает последовательность целых положительных чисел, 0 — конец последовательности. Определить в каком из чисел больше всего цифр.
8. Вводится последовательность из N целых положительных элементов. Проверить содержит ли последовательность хотя бы одну пару соседних *дружественных чисел*. Два различных натуральных числа являются дружественными, если сумма всех делителей первого числа (кроме самого числа) равна второму числу. Например, 220 и 284, 1184 и 1210, 2620 и 2924, 5020 и 5564..
9. Поступает последовательность целых положительных чисел, 0 — конец последовательности. Для элементов последовательности, находящихся в диапазоне от единицы до $m$ вычислить и вывести на экран соответствующие *числа Фибоначчи*. Здесь $m$ — целое положительное число, которое необходимо ввести.
10. Вводится последовательность из N целых положительных элементов. Найти число с минимальным количеством цифр.
11. Вводится последовательность из N целых элементов. Для всех положительных элементов последовательности вычислить значение *факториала*. Вывести на экран число и его факториал.
12. Поступает последовательность целых положительных чисел, 0 — конец последовательности. Вывести на экран все числа по-

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

следовательности являющиеся *составными* и их делители. Составное число имеет более двух делителей, то есть не является *простым*.

13. Вводится последовательность из N целых положительных элементов. Определить содержит ли последовательность хотя бы одно *число Армстронга*. Число Армстронга — натуральное число, которое равно сумме своих цифр, возведенных в степень, равную количеству его цифр. Например, десятичное число 153 — число Армстронга, потому что: $1^3+3^3+5^3 = 1+27+125 = 153$.
14. Поступает последовательность целых положительных чисел, 0 — конец последовательности. Найти среднее арифметическое *простых* чисел в этой последовательности. Простое число не имеет делителей, кроме единицы и самого себя.
15. Вводится последовательность из N целых положительных элементов. Определить сколько в последовательности пар соседних *взаимно простых чисел*. Различные натуральные числа являются взаимно простыми, если их наибольший общий делитель равен единице.
16. В последовательности из $N$ целых положительных элементов найти сумму всех *недостаточных чисел*. Недостаточное число всегда больше суммы всех своих делителей за исключением самого числа.
17. Вводится последовательность из N целых положительных элементов. Посчитать количество элементов последовательности, имеющих в своем представлении цифру 0.
18. Вводится N пар целых положительных чисел $a$ и $b$. В случае, если $a > b$ вычислить: $C = \frac{a!}{b!\cdot(a-b)!}$.
19. Вводится последовательность из $N$ целых элементов. Для каждого элемента последовательности найти среднее значение его цифр.
20. Вводится последовательность целых положительных чисел, 0 — конец последовательности. Для каждого элемента последовательности определить и вывести на экран число, которое получится, если поменять местами первую и последнюю цифры исходного числа.
21. Вводится последовательность из $N$ целых элементов. Для каждого элемента последовательности вывести на экран количество цифр и количество делителей.
22. Вводится последовательность из $N$ целых положительных элементов. Среди элементов последовательности найти наибольшее

число — *палиндром*. Числа палиндромы симметричны относительно своей середины, например, 12021 или 454.

23. Поступает последовательность целых положительных чисел, 0 — конец последовательности. Для каждого элемента последовательности вывести на экран сумму квадратов его цифр.
24. Вводится последовательность из $N$ целых положительных элементов. Для *простых* элементов последовательности определить сумму цифр. Простое число не имеет делителей, кроме единицы и самого себя.
25. Вводится последовательность целых положительных чисел, 0 — конец последовательности. Среди элементов последовательности найти наименьшее *составное число*. Составное число имеет более двух делителей, то есть не является *простым*.

### 4.10.2 Применение функций для вычислений в различных системах счисления

Разработать программу на языке C++ для решения следующей задачи. Заданы два числа — $A$ и $B$, первое в системе счисления с основанием $p$, второе в системе счисления с основанием $q$. Вычислить значение $C$ по указанной формуле и вывести его на экран в десятичной системе счисления и системе счисления с основанием $r$. Исходные данные для решения задачи представлены в табл. 4.1.

Таблица 4.1: Задания для решения задачи о различных системах счисления

| Вариант | p | q | C | r |
|---|---|---|---|---|
| 1 | 2 | 8 | $A^2 \cdot (A + B)$ | 3 |
| 2 | 3 | 7 | $2 \cdot (A^2 + B^2)$ | 4 |
| 3 | 4 | 6 | $2 \cdot B^2 \cdot (A + B)$ | 5 |
| 4 | 5 | 2 | $(A - B)^2 + 3 \cdot A$ | 6 |
| 5 | 6 | 4 | $A^2 + A \cdot B$ | 7 |
| 6 | 7 | 3 | $(5 \cdot B - 2 \cdot A)^2$ | 8 |
| 7 | 8 | 2 | $(2 \cdot A - 3 \cdot B)^2$ | 5 |
| 8 | 3 | 8 | $(B - A)^2 + 2 \cdot A$ | 6 |
| 9 | 4 | 7 | $B^3 - B^2 + 2 \cdot A$ | 2 |
| 10 | 5 | 6 | $A^3 - A^2 + 3 \cdot B$ | 8 |
| 11 | 6 | 5 | $(2 \cdot A - 3 \cdot B)^2$ | 3 |

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Таблица 4.1 — продолжение

| **Вариант** | **p** | **q** | **C** | **r** |
|---|---|---|---|---|
| 12 | 7 | 4 | $A^2 + 2 \cdot A + B^2$ | 5 |
| 13 | 8 | 3 | $A^2 + 3 \cdot B + B^2$ | 7 |
| 14 | 4 | 2 | $A^2 - 2 \cdot A + B$ | 6 |
| 15 | 5 | 8 | $3 \cdot B^2 - 2 \cdot B + A$ | 3 |
| 16 | 6 | 7 | $A^2 + (B - A)^2$ | 2 |
| 17 | 7 | 6 | $3 \cdot B^2 + 2 \cdot A \cdot B$ | 8 |
| 18 | 8 | 5 | $2 \cdot A^2 + 3 \cdot A \cdot B$ | 7 |
| 19 | 2 | 4 | $B^3 - 2 \cdot B + A$ | 3 |
| 20 | 3 | 8 | $A^3 - 2 \cdot A + B$ | 4 |
| 21 | 4 | 7 | $(5 \cdot A - 2 \cdot B)^2$ | 5 |
| 22 | 5 | 6 | $(B^2 - 3 \cdot A)^2$ | 7 |
| 23 | 6 | 5 | $(A^2 - 2 \cdot B)^2$ | 8 |
| 24 | 7 | 4 | $A^2 \cdot B^2 - A \cdot B$ | 6 |
| 25 | 8 | 3 | $A \cdot B + A^2 - B$ | 2 |

### 4.10.3 Применение функций для решения нелинейных уравнений

Разработать программу на языке C++ для вычисления одного из корней уравнения $f(x) = 0$ методами, указанными в задании. Для решения задачи предварительно определить интервал изоляции корня графическим методом. Вычисления проводить с точностью $\varepsilon = 10^{-4}$. Оценить степень точности путем подсчета количества итераций, выполненных для достижения заданной точности. Исходные данные для решения задачи представлены в табл. 4.2.

Таблица 4.2: Задания к задаче о решении нелинейных уравнений

| **№** | **Уравнение** $f(x) = 0$ | **Методы решения** |
|---|---|---|
| 1 | $x - 0.2 \cdot \sin(x + 0.5) = 0$ | метод половинного деления, метод хорд |
| 2 | $x^2 - \lg(x + 2) = 0$ | метод касательных, метод простой итерации |
| 3 | $x^2 - 20 \cdot \sin(x) = 0$ | метод хорд, метод касательных |
| 4 | $\ln(x) + (x + 1)^3 = 0$ | метод дихотомии, метод простой итерации |
| 5 | $x^2 - \sin(5x) = 0$ | метод половинного деления, метод касательных |

Таблица 4.2 — продолжение

| № | **Уравнение** $f(x) = 0$ | **Методы решения** |
|---|---|---|
| 6 | $e^x + x^2 - 2 = 0$ | метод хорд, метод простой итерации. |
| 7 | $0.8 \cdot x^2 - \sin(10 \cdot x) = 0$ | метод половинного деления, метод хорд |
| 8 | $\sin(7 \cdot x) + 2 \cdot x - 6 = 0$ | метод касательных, метод простой итерации |
| 9 | $x \cdot \ln(x) - 1 = 0$ | метод хорд, метод касательных |
| 10 | $2 \cdot \lg(x) + 0.5 \cdot x = 0$ | метод дихотомии, метод простой итерации |
| 11 | $e^{-x} - x^2 = 0$ | метод половинного деления, метод касательных |
| 12 | $x^2 - 3 \cdot \cos(x^2) = 0$ | метод хорд, метод простой итерации |
| 13 | $\sin(7 \cdot x) - x^2 + 15 = 0$ | метод половинного деления, метод хорд |
| 14 | $(x-1)^2 - 0.5 \cdot e^x = 0$ | метод касательных, метод простой итерации |
| 15 | $2 \cdot \ln(x) - 0.2 \cdot x + 1 = 0$ | метод хорд, метод касательных |
| 16 | $2 - x \cdot e^x = 0$ | метод дихотомии, метод простой итерации |
| 17 | $0.1 \cdot x^3 + 3 \cdot x^2 - 10 \cdot x - 7 = 0$ | метод половинного деления, метод касательных |
| 18 | $0.1 \cdot x^2 - e^x = 0$ | метод хорд, метод простой итерации |
| 19 | $e^{-2 \cdot x} - 2 \cdot x + 1 = 0$ | метод половинного деления, метод хорд |
| 20 | $x^2 - 3 + 0.5^x = 0$ | метод касательных, метод простой итерации |
| 21 | $\lg(4 \cdot x) - \cos(x) = 0$ | метод хорд, метод касательных |
| 22 | $\ln(x) - \cos^2(x) = 0$ | метод дихотомии, метод простой итерации |
| 23 | $\frac{4}{x} - 0.2 \cdot e^x = 0$ | метод половинного деления, метод касательных |
| 24 | $\sqrt{x + 6.5} - e^x = 0$ | метод хорд, метод простой итерации. |
| 25 | $0.5^x + 1 - (x-2)^2 = 0$ | метод половинного деления, метод хорд |

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

# Глава 5

# Массивы

Эта глава является ключевой в изучении программирования на C(C++). В ней описаны методы построения алгоритмов и программ с использованием статических и динамических массивов. В заключительном параграфе главы на большом количестве примеров рассматривается совместное использование указателей, динамических массивов и функций пользователя при решении сложных задач обработки массивов.

## 5.1 Статические массивы в C(C++)

Часто для работы с множеством однотипных данных (целочисленными значениями, строками, датами и т.п.) оказывается удобным использовать массивы. Например, можно создать массив для хранения списка студентов, обучающихся в одной группе. Вместо создания переменных для каждого студента, например `Студент1`, `Студент2` и т.д., достаточно создать один массив, где каждой фамилии из списка будет присвоен порядковый номер. Таким образом, можно дать следующее определение. — структурированный тип данных, состоящий из фиксированного числа элементов одного типа.

Массив в табл. 5.1 имеет 8 элементов, каждый элемент сохраняет число вещественного типа. Элементы в массиве пронумерованы (нумерация массивов начинается с нуля). Такого рода массив, представляющий собой просто список данных одного и того же типа, называют простым или одномерным массивом. Для доступа к данным, хранящимся в определенном элементе массива, необходимо указать

имя массива и порядковый номер этого элемента, называемый индексом.

Таблица 5.1: Одномерный числовой массив

| № элемента массива | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Значение | 13.65 | -0.95 | 16.78 | 8.09 | -11.76 | 9.07 | 5.13 | -25.64 |

Если возникает необходимость хранения данных в виде матриц, в формате строк и столбцов, то необходимо использовать двумерные массивы. В табл. 5.2 приведен пример массива, состоящего из четырех строк и пяти столбцов. Это двумерный массив. Строки в нем можно считать первым измерением, а столбцы вторым. Для доступа к данным, хранящимся в этом массиве, необходимо указать имя массива и два индекса, первый должен соответствовать номеру строки, а второй номеру столбца в которых хранится необходимый элемент.

Таблица 5.2: Двумерный числовой массив

| | | | | |
|---|---|---|---|---|
| 1.5 | -0.9 | 1.8 | 7.09 | -1.76 |
| 3.6 | 0.5 | 6.7 | 0.09 | -1.33 |
| 13.65 | -0.95 | 16.78 | 8.09 | -11.76 |
| 7.5 | 0.95 | 7.3 | 8.9 | 0.11 |

Если при описании массивов определён его размер, то массивы называются статическими, рассмотрим работу с одномерными статическими массивами в языке C(C++). Двумерные массивы подробно описаны в следующей главе.

### 5.1.1 Описание статических массивов

Описать статический массив в C(C++) можно так:

`тип имя_переменной [размерность];`

размерность — количество элементов в массиве. Например:

```
int x[10];   //Описание массива из 10 целых чисел. Первый
             //элемент массива имеет индекс 0, последний 9.
float a[20]; //Описание массива из 20 вещественных чисел.
       //Первый элемент массива имеет индекс 0, последний 19.
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Размерность массива и тип его элементов определяют объем памяти, который необходим для хранения массива. Рассмотрим ещё один пример описания массива:

```
const int n=15; //Определена целая положительная константа.
double B[n];    //Описан массив из 15 вещественных чисел.
```

При описании статического массива в качестве размерности можно использовать целое положительное число или предопределённую константу.

Элементы массива в C(C++) нумеруются с нуля. Первый элемент, всегда имеет номер ноль, а номер последнего элемента на единицу меньше заданной при его описании размерности:

```
char C[5];  //Описан массив из 5 символов, нумерация от 0 до 4.
```

### 5.1.2 Основные операции над массивами

Доступ к каждому элементу массива осуществляется с помощью индекса — порядкового номера элемента. Для обращения к элементу массива указывают его имя, а затем в квадратных скобках индекс:

```
имя_массива [индекс]
```

Например:

```
const int n=15;
double C[n],S;
//Сумма первого и последнего элементов массива C.
S=C[0]+C[n-1];
```

Массиву, как и любой другой переменной, можно присвоить начальное значение (инициализировать). Для этого значения элементов массива нужно перечислить в фигурных скобках через запятую:

```
тип имя_переменной[размерность]={элемент_0, элемент_1, ...};
```

Например:

```
float a[6]={1.2, (float)3/4, 5./6, 6.1};
//Формируется массив из шести вещественных чисел, значения
//элементам присваиваются по порядку, элементы значения,
//которых не указаны (в данном случае a[4], a[5])
//обнуляются:
// a[0]=1.2, a[1]=(float)3/4, a[2]=5./6, a[3]=6.1, a[4]=0,
// a[5]=0, для элементов a[1] и a[2] выполняется
// преобразование типов.
```

Рассмотрим, как хранится массив в памяти на примере массива

```
double x[30];
```

В памяти компьютере выделяется место для хранения 30 элементов типа `double`. При этом адрес этого участка памяти хранится в переменной `x`. Таким образом получается, что к элементу массива с индексом 0. можно обратится двумя способами:

1. В соответствии с синтаксисом языка `C(C++)` можно записать `x[0]`.
2. Адрес начала массива хранится в переменной `x` (по существу`x` — указатель на double), поэтому с помощью операции `*x` мы можем обратиться к значению нулевого элемента массива.

Следовательно, `x[0]` и `*x` являются обращением к нулевому элементу массива.

Если к значению добавить единицу (число 1), то мы сместимся на один элемент типа `double`. Таким образом, `x+1` — адрес элемента массива `x` с индексом 1. К первому элементу массива `x` также можно обратиться двумя способами `x[1]` или `*(x+1)`. Аналогично, к элементу с индексом 2 можно обращаться либо `x[2]`, либо `*(x+2)`. Таким образом, получается, что к элементу с индексом `i` можно обращаться `x[i]` или `*(x+i)`.

При этом при обработке массива (независимо от способа обращения `x[i]` или `*(x+i)`) программист сам должен контролировать существует ли элемент массива `x[i]` (или `*(x+i)`) и не вышла ли программа за границы массива.

Особенностью статических массивов является определение размера статическим образом при написании текста программы. При необходимости увеличить размер массив необходимо изменить текст программы и перекомпилировать её. Для динамического выделения памяти для массивов в `C(C++)` можно указатели и использовать операторы (функции) выделения памяти.

## 5.2 Динамические массивы в C(C++)

Для создания динамического массива необходимо [1, 8]:

- описать указатель (`тип * указатель;`);
- определить размер массива;
- выделить участок памяти для хранения массива и присвоить указателю адрес этого участка памяти.

Для выделения памяти в `C++` можно воспользоваться оператором `new` или функциями языка `C` — `calloc`, `malloc`, `realloc`. Все функции находятся в библиотеке `stdlib.h`.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

### 5.2.1 Функция malloc

Функция выделяет непрерывный участок памяти размером `size` байт и возвращает указатель на первый байт этого участка. Обращение к функции имеет вид:

```
void *malloc(size\_t size);
```

где `size` — целое беззнаковое значение[1], определяющее размер выделяемого участка памяти в байтах. Если резервирование памяти прошло успешно, то функция возвращает переменную типа `void *`, которую можно преобразовать к любому необходимому типу указателя. Если выделить память невозможно, то функция вернёт пустой указатель `NULL`.

Например,

```
double *h; //Описываем указатель на double.
int k;
cin>>k; //Ввод целого числа k.
//Выделение участка памяти для хранения k элементов типа
//double. Адрес этого участка хранится в переменной h.
h=(double *) malloc(k*sizeof(double));
//h — адрес начала участка памяти, h+1, h+2, h+3 и т. д. —
//адреса последующих элементов типа double.
```

### 5.2.2 Функция calloc

Функция `calloc` предназначена для выделения и обнуления памяти.

```
void *calloc (size_t num, size_t size);
```

С помощью функции будет выделен участок памяти, в котором будет храниться `num` элементов по `size` байт каждый. Все элементы выделенного участка обнуляются. Функция возвращает указатель на выделенный участок или `NULL` при невозможности выделить память.

Например,

```
float *h; //Описываем указатель на float.
int k;
```

[1] `size_t` — базовый беззнаковый целочисленный тип языка C/C++, который выбирается таким образом, чтобы в него можно было записать максимальный размер теоретически возможного массива любого типа. В 32-битной операционной системе `size_t` является беззнаковым 32-битным числом (максимальное значение $2^{32} - 1$), в 64-битной — 64-битным беззнаковым числом (максимальное значение $2^{64} - 1$).

```
cin>>k;     //Ввод целого числа k.
//Выделение участка памяти для хранения k элементов типа
//float. Адрес этого участка хранится в переменной h.
h=(float *) calloc(k,sizeof(float));
//h — адрес начала участка памяти, h+1, h+2, h+3 и т. д. —
// адреса последующих элементов типа float.
```

### 5.2.3 Функция realloc

Функция realloc изменяет размер ранее выделеннго участка памяти. Обращаются к функции так:

```
char *realloc(void *p, size_t size);
```

где `p` — указатель на область памяти, размер которой нужно изменить на `size`. Если в результате работы функции меняется адрес области памяти, то новый адрес вернется в качестве результата. Если фактическое значение первого параметра `NULL`, то функция `realloc` работает, так же как и функции `malloc`, то есть выделяет участок памяти размером `size` байт.

### 5.2.4 Функция `free`

Для освобождения выделенной памяти используется функция `free`. Обращаются к ней так:

```
void free(void *p);
```

где `p` — указатель на участок память, ранее выделенный функциями `malloc`, `calloc` или `realloc`.

### 5.2.5 Операторы `new` и `delete`

В языке `C++` есть операторы `new` для выделения и `free` для освобождения участка памяти.

Для выделения памяти для хранения $n$ элементов одного типа оператор `new` имеет вид [5]:

```
x=new type [n];
```

`type` — тип элементов, для которых выделяется участок памяти;

`n` — количество элементов;

`x` — указатель на тип данных `type`, в котором будет храниться адрес выделенного участка памяти.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

При выделении памяти для одного элемента оператор `new` имеет вид.

```
x=new type;
```

Например,

```
float *x; //Указатель на тип данных float.
int n;
cin>>n;   //Ввод n
//Выделение участка памяти для хранения n элементов типа
//float. Адрес этого участка хранится в переменной x;
// x+1, x+2, x+3 и т. д. — адреса последующих элементов
// типа float.
```

Освобождение выделенного с помощью `new` участка памяти осуществляется с помощью оператора `delete` следующей структуры:

```
delete [] p;
```

$p$ — указатель (адрес участка памяти ранее выделенного с помощью оператора `new`).

## 5.3 Отличие статического и динамического массива

В чём же отличие статического и динамического массива?

Если у нас есть статический массив, например.

```
double x[75];
```

Мы имеем участок памяти для хранения 75 элементов типа `double` (массив из 75 элементов типа `double`). Адрес начала массива хранится в переменной `x`. Для обращения к `i`-му элементу можно использовать конструкции `x[i]` или `*(x+i)`. Если понадобится обрабатывать массив более, чем из 75 элементов, то придётся изменить описание и перекомпилировать программу. При работе с массивами небольшой размерности, большая часть памяти, выделенная под статический массив будет использоваться вхолостую.

Если имеется динамический массив, например

```
double *x;//Указатель на double
int k;
//Вводим размер массива k.
cin>>k;
//Выделение памяти для хранения динамического массива из k
//чисел. Адрес начала массива хранится в переменной x.
x=new double[k];
//Память можно будет выделить так
```

```
x=(double *) calloc(k,sizeof(float));
//или так
x=(double *) malloc(k*sizeof(float));
```

Мы имеем указатель на тип данных `double`, вводим размер динамического $k$, выделяем участок памяти для хранения $k$ элементов типа *`double`* (массив из $k$ элементов типа `double`). Адрес начала массива хранится в переменной $x$. Для обращения к $i$-му элементу можно использовать конструкции `x[i]` или `*(x+i)`. В случае динамического массива мы с начала определяем его размер (в простейшем случае просто вводим размер массива с клавиатуры), а потом выделяем память для хранения реального количества элементов. Таким образом основное отличие статического и динамического массивов состоит в том, что в статическом массиве выделяется столько элементов, сколько необходимо.

Таким образом, при использовании как статического, так и динамического массива, имя массива — адреса начала участка памяти, обращаться к элементам массива можно двумя способами — `x[i]`, `*(x+i)`.

## 5.4 Основные алгоритмы обработки массивов

Все манипуляции с массивами в C++ осуществляются поэлементно. Организовывается цикл, в котором происходит последовательное обращение к нулевому, первому, второму и т.д. элементам массива. В общем виде алгоритм обработки массива выглядит, так как показано на рис. 5.1.

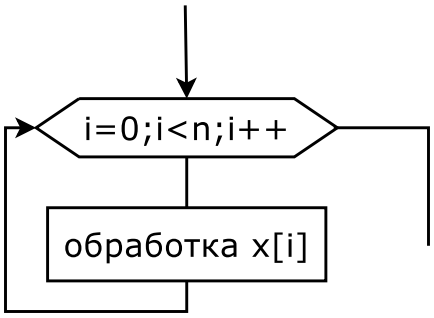


Рис. 5.1. Алгоритм обработки элементов массива

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Алгоритмы, с помощью которых обрабатывают одномерные массивы, похожи на обработку последовательностей (вычисление суммы, произведения, поиск элементов по определенному признаку, выборки и т. д.). Отличие заключается в том, что в массиве одновременно доступны все его компоненты, поэтому становится возможной, например, сортировка его элементов и другие, более сложные преобразования.

### 5.4.1 Ввод-вывод элементов массива

Ввод и вывод массивов так же осуществляется поэлементно. Блок-схемы алгоритмов ввода и вывода элементов массива `X[N]` изображены на рис. 5.2–5.3.

Рассмотрим несколько вариантов ввода массива:

```
//Вариант 1. Ввод массива с помощью функции scanf.
//При организации ввода используются специальные символы:
//табуляция — \t и переход на новую строку — \n.
#include <stdio.h>
int main()
{
float x[10]; int i,n;
//Ввод размерности массива.
printf("\n N="); scanf("%d",&n);
printf("\n Введите элементы массива X \n");
for(i=0;i<n;i++)
//Ввод элементов массива в цикле.
//Обратите внимание!!! Использование x+i.
scanf("%f",x+i);
return 0;
}
```

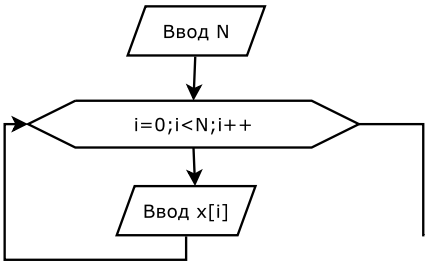


Рис. 5.2. Алгоритм ввода массива `X[N]`

Результат работы программы:

```
N=3
Введите элементы массива X
1.2
-3.8
0.49
```

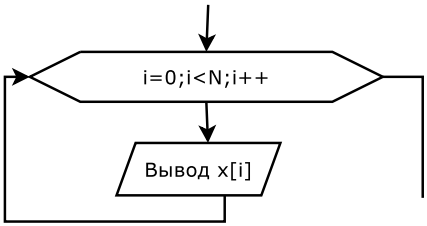


Рис. 5.3. Алгоритм вывода массива `X[N]`

```
//Вариант 2. Ввод массива с помощью функции scanf
//и вспомогательной переменной b.
#include <stdio.h>
int main()
{
float x[10],b; int i,n;
//Ввод размерности массива.
printf("\n N="); scanf("%d",&n);
printf("\n Массив X \n");
for(i=0;i<n;i++)
{
//Сообщение о вводе элемента.
printf("\n Элемент %d \t",i);
scanf("%f",&b);        //Ввод переменной b.
//Присваивание элементу массива значения переменной b.
x[i]=b;
}
return 0;
}
```

Результат работы программы:

```
N=4
Массив X
Элемент 0  8.7
Элемент 1  0.74
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
Элемент 2  -9
Элемент 3  78
```

```
//Вариант 3. Ввод динамического массива с помощью оператора  cin.
#include <iostream>
using namespace std;
int main()
{
int *X,N,i;
cout<<"\n N="; cin>>N; //Ввод размерности массива.
//Выделение памяти для динамического массива из N элементов.
X=new int [N];
for (i=0;i<N;i++)
{
cout<<"\n X["<<i<<"]="; //Сообщение о вводе элемента.
cin>>X[i];              //Ввод элементов массива в цикле.
}
delete [] X;
return 0;
}
```

Результат работы программы:

```
N=4
X[0]=1
X[1]=2
X[2]=4
X[3]=5
```

Вывод статического или динамического массива можно осуществить несколькими способами:

```
//Вариант 1. Вывод массива в виде строки.
for(i=0;i<n;i++) printf("%f \t",X[i]);
//Вариант 2. Вывод массива в виде столбца.
for(i=0;i<n;i++) printf("\n %f ",X[i]);
//Вариант 3. Вывод массива в виде строки.
for (i=0;i<N;i++) cout <<"\t X["<<i<<"]="<<X[i];
//Вариант 4. Вывод массива в виде столбца.
for (i=0;i<N;i++) cout <<"\n X["<<i<<"]="<<X[i];
```

### 5.4.2 Вычисление суммы элементов массива

Дан массив $X$, состоящий из $N$ элементов. Найти сумму элементов этого массива. Процесс накапливания суммы элементов массива достаточно прост и практически ничем не отличается от суммирования значений некоторой числовой последовательности. Переменной $S$

присваивается значение равное нулю, затем последовательно суммируются элементы массива $X$. Блок-схема алгоритма расчёта суммы приведена на рис. 5.4.

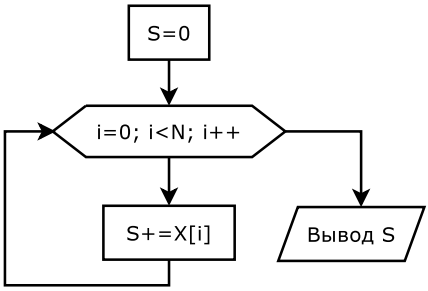


Рис. 5.4. Алгоритм вычисления суммы элементов массива

Соответствующий алгоритму фрагмент программы будет иметь вид:

```
for (S=i=0;i<N;i++)
    S+=X[i];
cout<<"S="<<S<<"\n";
```

### 5.4.3 Вычисление произведения элементов массива

Дан массив $X$, состоящий из $N$ элементов. Найти произведение элементов этого массива. Решение этой задачи сводится к тому, что значение переменной $P$, в которую предварительно была записана единица, последовательно умножается на значение $i$–го элемента массива. Блок-схема алгоритма приведена на рис. 5.5.

Соответствующий фрагмент программы будет иметь вид:

```
for (P=1,i=0;i<N;i++)
    P*=X[i];
cout<<"P="<<P<<"\n";
```

**Задача 5.1.** Задан массив целых чисел. Найти сумму простых чисел и произведение отрицательных элементов массива.

Алгоритм решения задачи состоит из следующих этапов.

1. Вводим массив $X[N]$.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

2. Для вычисления суммы в переменную $S$ записываем значение 0, для вычисления произведения в переменную $P$ записываем 1.
3. В цикле($i$ изменяется от 0 до $N$-1 с шагом 1) перебираем все элементы массива $X$, если очередной элемент массива является простым числом, добавляем его к сумме, а если очередной элемент массива отрицателен, то умножаем его на $P$.
4. Выводим на экран значение суммы и произведения.

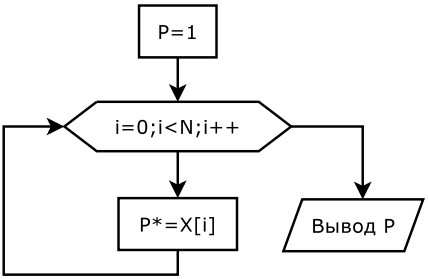


Рис. 5.5. Вычисление произведения элементов массива

Блок-схема решения задачи представлена на рис. 5.6. Для решения задачи применим функцию (prostoe) проверки является ли число простым. Текст программы с подробными комментариями приведён далее.

```
#include <iostream>
using namespace std;
//Текст функции prostoe.
bool prostoe (int N)
{
    int i;
    bool pr;
    if (N<2) pr=false;
    else
    for(pr=true, i=2;i<=N/2;i++)
        if (N%i==0)
        {
            pr=false;
            break;
        }
    return pr;
}
int main()
```

```
{
    int *X,i,N,S,P;
//Ввод размерности массива.
    cout<<"Введите размер массива ";cin>>N;
//Выделение памяти для хранения динамического массива X.
 X=new int [N];
//Ввод массива X.
    cout<<"Ведите массив X\n";
    for(i=0;i<N;i++)
    {cout<<"X("<<i<<")=";cin>>X[i];}
//В цикле перебираем все элементы массива
    for(P=1,S=i=0;i<N;i++)
    {
//Если очередной элемент массива — простое число, добавляем его к сумме.
        if (prostoe(X[i])) S+=X[i];
//Если очередной элемент массива отрицателен, умножаем его на P.
        if (X[i]<0) P*=X[i];
    }
//Вывод S и P на экран.
    cout << "S=" <<S<<"\tP="<<P<< endl;
//Освобождение занимаемой массивом X памяти.
    delete [] X;
    return 0;
}
```

Результаты работы программы представлены ниже.

```
Введите размер массива 10
Ведите массив X
X(0)=-7
X(1)=-9
X(2)=5
X(3)=7
X(4)=2
X(5)=4
X(6)=6
X(7)=8
X(8)=10
X(9)=12
S=14 P=63
```

### 5.4.4 Поиск максимального элемента в массиве и его номера

Дан массив `X`, состоящий из `n` элементов. Найти максимальный элемент массива и номер, под которым он хранится в массиве.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

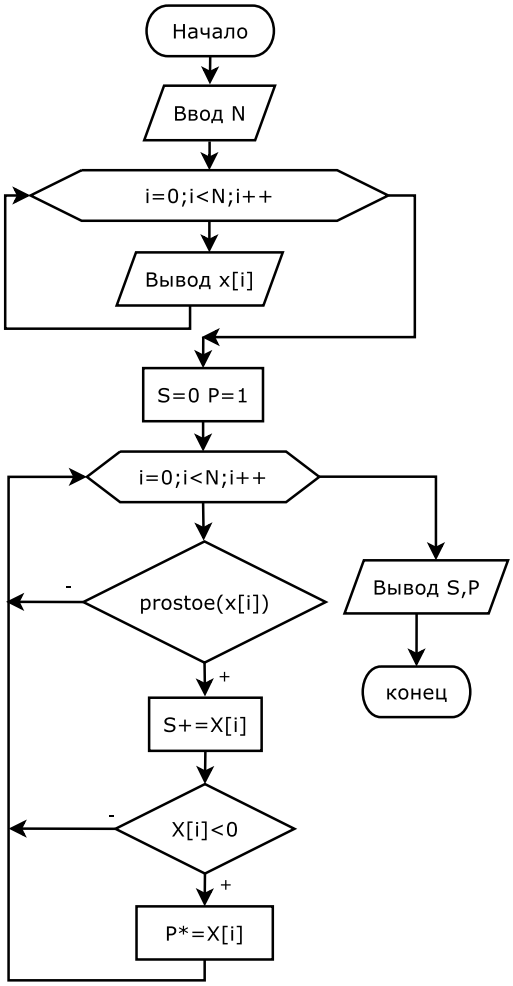


Рис. 5.6. Блок-схема алгоритма решения задачи 5.1

Алгоритм решения задачи следующий. Пусть в переменной с именем `Max` хранится значение максимального элемента массива, а в переменной с именем `Nmax` – его номер. Предположим, что нулевой элемент массива является максимальным и запишем его в переменную

`Max`, а в `Nmax` — его номер (то есть ноль). Затем все элементы, начиная с первого, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную `Max`, а в переменную `Nmax` – текущее значение индекса `i`. Процесс определения максимального элемента в массиве приведен в таблице 5.3 и изображен при помощи блок-схемы на рис. 5.7. Соответствующий фрагмент программы имеет вид:

```
for (Max=X[0],Nmax=i=0;i<n;i++)
  if (Max<X[i])
  {
    Max=X[i];
    Nmax=i;
  }
cout<<"Max="<<Max<<"\n";
cout<<"Nmax="<<Nmax<<"\n";
```

Таблица 5.3: Определение максимального элемента и его номера в массиве

| Номера элементов | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Исходный массив | 4 | 7 | 3 | 8 | 9 | 2 |
| Значение переменной $Max$ | 4 | 7 | 7 | 8 | 9 | 9 |
| Значение переменной $Nmax$ | 1 | 2 | 2 | 4 | 5 | 5 |

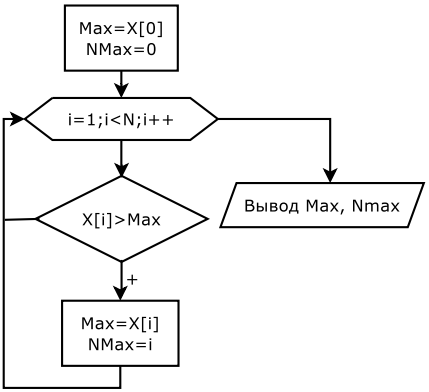


Рис. 5.7. Поиск максимального элемента и его номера в массиве

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

При поиске максимального элемента и его номера, можно найти номер максимального элемента, а потом по номеру извлечь значение максимального элемента из массива.

Текст программы поиска номера максимального элемента:

```
#include <iostream>
using namespace std;
int main()
{
  float *X;
  int i,N,nom;
  //Ввод размерности динамического массива
  cout<<"Введите размер массива "; cin>>N;
  //Выделения памяти для хранения динамического массива X.
  X=new float[N];
  //Ввод динамического массива X.
  cout<<"Введите элементы массива X\n";
  for(i=0;i<N;i++)
    cin>>X[i];
  //В переменной nom будем хранить номер максимального
  //элемента. Предположим, что максимальным элементом,
  //является элемент с номером 0.
  nom=0;
  for(i=1;i<N;i++)
  //Если очередной элемент больше X[nom], значит nom не
  //является номером максимального элемента, элемент с
  //номером i больше элемента X[nom], поэтому переписываем
  //число i в переменную nom.
    if (X[i]>X[nom]) nom=i;
  cout << "Максимальный элемент="<<X[nom]<<", его
       номер=" << nom << endl;
  return 0;
}
```

**Совет.** Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в условном блоке и, соответственно, в конструкции `if` текста программы знак поменяется с $<$ на $>$.

Рассмотрим несколько задач.

**Задача 5.2.** Найти минимальное простое число в целочисленном массиве x[N].

Эта задача относится к классу задач поиска минимума (максимума) среди элементов, удовлетворяющих условию. Подобные задачи рассматривались в задачах на обработку последовательности чисел. Здесь поступим аналогично. Блок-схема приведена на рис. 5.8.

Необходимо первое простое число объявить минимумом, а все последующие простые элементы массива сравнивать с минимумом. Бу-

дем в цикле последовательно проверять, является ли элемент массива простым числом (функция `prostoe`). Если `X[i]` является простым числом, то количество простых чисел (`k`) увеличиваем на 1 (`k++`), далее, проверяем, если `k` равен 1 (`if (k==1)`), то этот элемент объявляем минимальным (`min=x[i]; nom=i;`), иначе сравниваем его с минимальным (`if (x[i]<min) {min=x[i];nom=i;}`).

Текст программы:

```
#include <iostream>
using namespace std;
bool prostoe (int N)
{ int i;  bool pr;
  if (N<2) pr=false;
  else
  for(pr=true, i=2;i<=N/2;i++)
    if (N%i==0)
    {
       pr=false;
       break;
    }
  return pr;
}
int main(int argc, char **argv)
{
  int i,k,n,nom,min,*x;
//Ввод количества элементов в массиве.
  cout<<"n="; cin>>n;
//Выделяем память для динамического массива x.
  x=new int [n];
//Ввод элементов массива.
  cout<<"Введите элементы массива X";
  for(i=0;i<n;i++)
    cin>>x[i];
//С помощью цикла по переменной i, перебираем все элементы
// в массиве x, k — количество простых чисел в массиве.
  for(i=k=0;i<n;i++)
//Проверяем, является ли очередной элемент массива
// простым числом. Если x[i] — простое число.
  if (prostoe(x[i]))
  {
//Увеличиваем счётчик количества простых чисел в массиве.
    k++;
//Если текущий элемент является первым простым числом в массиве,
// объявляем его минимумом, а его номер сохраняем в перемнной nom.
    if (k==1) {min=x[i];nom=i;}
    else
//Все последующие простые числа в массиве сравниваем с минимальным
//простым числом. Если текущее число меньше min, перезаписываем
//его в переменную min, а его номер — в переменную nom.
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
      if (x[i]<min) {min=x[i];nom=i;}
   }
//Если в массиве были простые числа, выводим значение и
//номер минимального простого числа.
   if (k>0)
      cout<<"min="<<min<<"\tnom="<<nom<<endl;
//Иначе выводим сообщение о том, что в массиве нет простых чисел.
   else cout<<"Нет простых чисел в массиве"<<endl;
   return 0;
}
```

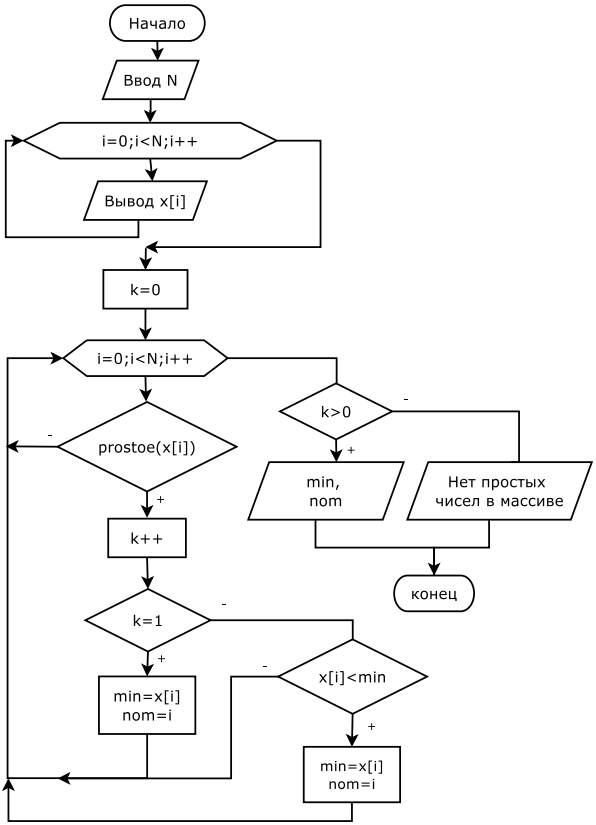


Рис. 5.8. Блок-схема рещения задачи 5.2

Аналогичным образом можно написать программу любой задачи поиска минимума (максимума) среди элементов, удовлетворяющих какому-либо условию (минимум среди положительных элементов, среди чётных и т.д.).

**Задача 5.3.** Найти $k$ минимальных чисел в вещественном массиве.

Перед решением этой довольно сложной задачи рассмотрим более простую задачу.

Найти два наименьших элемента в массиве. Фактически надо найти номера (`nmin1`, `nmin2`) двух наименьших элементов массива. На первом этапе надо найти номер минимального (`nmin1`) элемента массива. На втором этапе надо искать минимальный элемент, при условии, что его номер не равен `nmin1`. Вторая часть очень похожа на предыдущую задачу (минимум среди элементов, удовлетворяющих условию, в этом случае условие имеет вид `i!=nmin`).

Решение задачи с комментариями:

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
  int kvo,i,n,nmin1,nmin2;
  double *X;
  cout<<"n="; cin>>n;
  X=new double [n];
  cout<<"Введите элементы массива X\n";
  for(i=0;i<n;i++)
    cin>>X[i];
//Стандартный алгоритм поиска номера первого минимального
//элемента (nmin1).
  for(nmin1=0,i=1;i<n;i++)
    if (X[i]<X[nmin1]) nmin1=i;
//Второй этап — поиск номера минимального элемента,
//среди элементов, номер которых не совпадает nmin1.
//kvo — количество таких элементов.
  for(kvo=i=0;i<n;i++)
//Если номер текущего элемента не совпадает с nmin1,
    if (i!=nmin1)
    {
//увеличиваем количество таких элементов на 1.
      kvo++;
//Номер первого элемента, индекс которого не равен nmin1,
//объявляем номером второго минимального элемента.
      if (kvo==1) nmin2=i;
      else
//очередной элемент индекс которого не равен nmin1
//сравниваем с минимальным, если он меньше, номер
//перезаписываем в переменную nmin2.
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
        if (X[i]<X[nmin2]) nmin2=i;
    }
//Вывод двух минимальных элементов и их индексов.
        cout<<"nmin1="<<nmin1<<"\tmin1="<<X[nmin1]<<endl;
        cout<<"nmin2="<<nmin2<<"\tmin2="<<X[nmin2]<<endl;
  return 0;
}
```

По образу и подобию этой задачи можно написать задачу поиска трёх минимальных элементов в массиве. Первые два этапа (поиск номеров двух минимальных элементов в массиве) будут полным повторением кода, приведённого выше. На третьем этапе нужен цикл, в котором будем искать номер минимального элемента, при условии, что его номер не равен `nmin1` и `nmin2`. Авторы настоятельно рекомендуют читателям самостоятельно написать подобную программу. Аналогично можно написать программу поиска четырёх минимальных элементов. Однако при этом усложняется и увеличивается код программы. К тому же, рассмотренный приём не позволит решить задачу в общем случае (найти `k` минимумов).

Для поиска $k$ минимумов в массиве можно поступить следующим образом. Будем формировать массив `nmin`, в котором будут храниться номера минимальных элементов массива `x`. Для его формирования организуем цикл по переменной $j$ от 0 до `k-1`. При каждом вхождении в цикл в массиве `nmin` элементов будет `j-1` элементов и мы будем искать $j$-й минимум (формировать $j$-й элемент массива). Алгоритм формирования $j$-го элемента состоит в следующем: необходимо найти номер минимального элемента в массиве `x`, исключая номера, которые уже хранятся в массиве `nmin`. Внутри цикла по $j$ необходимо выполнить такие действия. Для каждого элемента массива `x` (цикл по переменной $i$) проверить содержится ли номер в массиве `nmin`, если не содержится, то количество (переменная `kvo`) таких элементов увеличить на 1. Далее, если `kvo` равно 1, то это первый элемент, который не содержится в массиве `nmin`, его номер объявляем номером минимального элемента массива (`nmin_temp=i;`). Если `kvo>1`, сравниваем текущий элемент `x[i]` с минимальным (`if (x[i]<X[nmin_temp]) nmin_temp=i;`). Блок-схема алгоритма поиска $k$ минимальных элементов массива представлена на рис. 5.9[2]. Далее приведен текст программы с комментариями.

```
#include <iostream>
using namespace std;
```

[2]В блок-схеме отсутствует ввод данных и вывод результатов.

```
int main(int argc, char **argv)
{
  int p,j,i,n,*nmin,k,kvo,nmin_temp;
  bool pr;
  double *x;
  cout<<"n=";cin>>n;
  x=new double [n];
  cout<<"Введите элементы массива X\n";
  for(i=0;i<n;i++)
    cin>>x[i];
  cout<<"Введите количество минимумов\n";cin>>k;
  nmin=new int[k];
//Цикл по переменной  j  для поиска номера  j+1  минимального элемента
  for(j=0;j<k;j++)
  {
    kvo=0;
//Перебираем все элементы массива.
    for(i=0;i<n;i++)
    {
//Цикл по переменной p проверяет содержится ли номер i в массиве nmin.
      pr=false;
      for(p=0;p<j;p++)
        if (i==nmin[p]) pr=true;
//Если не содержится, то количество элементов увеличить на 1.
      if (!pr)
      {
        kvo++;
//Если kvo=1, то найден первый элемент, который не содержится в массиве
//nmin, его номер объявляем номером минимального элемента массива
        if (kvo==1) nmin_temp=i;
        else
//Если kvo>1, сравниваем текущий элемент x[i] с минимальным.
          if (x[i]<x[nmin_temp]) nmin_temp=i;
      }
    }
//Номер очередного минимального элемента записываем в массив nmin.
    nmin[j]=nmin_temp;
  }
//Вывод номеров и значений k минимальных элементов массива.
  for(j=0;j<k;j++)
    cout<<"nmin1="<<nmin[j]<<"\tmin1="<<x[nmin[j]]<<endl;
  return 0;
}
```

Проверку содержится ли число `i` в массиве `nmin`, можно оформить в виде функции, тогда программа может быть записана следующим образом:

```
#include <iostream>
using namespace std;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

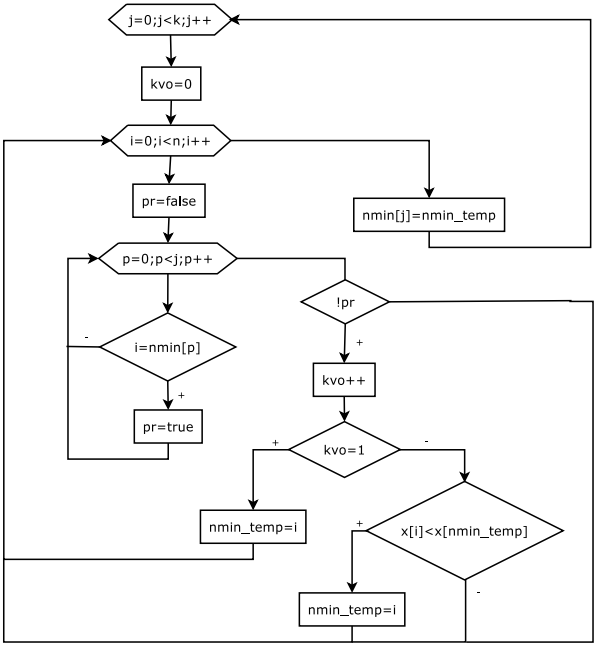


Рис. 5.9. Блок-схема алгоритма поиска $k$ минимальных элементов в массиве $x$.

```
//Функция проверяет содержится ли число i в массиве x из n элементов.
//Функция возвращает true, если содержится false, если не содержится.
bool proverka(int i, int *x, int n)
{
  bool pr;
  int p;
  pr=false;
  for(p=0;p<n;p++)
    if (i==x[p]) pr=true;
  return pr;
}
int main(int argc, char **argv)
{
  int   j,i,n,*nmin,k,kvo,nmin_temp;
  double *x;
  cout<<"n=";cin>>n;
  x=new double [n];
```

```
  cout<<"Введите элементы массива X\n";
  for(i=0;i<n;i++)
    cin>>x[i];
  cout<<"Введите количество минимумов\n";cin>>k;
  nmin=new int[k];
//Цикл по переменной j для поиска номера j+1 минимального элемента
  for(j=0;j<k;j++)
  {
    kvo=0;
//Перебираем все элементы массива.
    for(i=0;i<n;i++)
    {
//Вызов функции proverka, чтобы проверить содержится ли число i
//в массиве nmin из j элементов
      if (!proverka(i,nmin,j))
      {
        kvo++;
        if (kvo==1) nmin_temp=i;
        else
          if (x[i]<x[nmin_temp]) nmin_temp=i;
      }
    }
    nmin[j]=nmin_temp;
  }
//Вывод номеров и значений k минимальных элементов массива.
  for(j=0;j<k;j++)
    cout<<"nmin1="<<nmin[j]<<"\tmin1="<<x[nmin[j]]<<endl;
  return 0;
}
```

Авторы настоятельно рекомендуют читателю разобрать все версии решения задачи 5.3.

**Задача 5.4.** Поменять местами максимальный и минимальный элементы в массиве X.

Алгоритм решения задачи можно разбить на следующие этапы.

1. Ввод массива.
2. Поиск номеров максимального (`nmax`) и минимального (`nmin`) элементов массива.
3. Обмен элементов местами. Не получится записать «в лоб» (`X[nmax]=X[nmin]; X[nmin]=X[nmax];`). При таком присваивании мы сразу же теряем максимальный элемент. Поэтому нам понадобится временная (буферная) переменная temp. Обмен элементов местами должен быть таким:

   ```
   temp=X[nmax];X[nmax]=X[nmin]; X[nmin]=temp;
   ```

Далее приведён текст программы с комментариями.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
  int i,N,nmax,nmin;
  float temp;
  cout<<"N=";cin>>N;
  float X[N];
  cout<<"Введите элементы массива X\n";
  for(i=0;i<N;i++)
    cin>>X[i];
//Поиск номеров максимального и минимального элементов массива.
  for(nmax=nmin=0,i=1;i<N;i++)
  {
    if (X[i]<X[nmin]) nmin=i;
    if (X[i]>X[nmax]) nmax=i;
  }
//Обмен максимального и минимального элементов местами.
  temp=X[nmax];X[nmax]=X[nmin]; X[nmin]=temp;
//Вывод преобразованного массива.
  cout<<"Преобразованный массив X\n";
  for(i=0;i<N;i++)
    cout<<X[i]<<" ";
  cout<<endl;
  return 0;
}
```

**Задача 5.5.** Найти среднее геометрическое среди простых чисел, расположенных между максимальным и минимальным элементами массива.

Среднее геометрическое $k$ элементов ($SG$) можно вычислить по формуле $SG = \sqrt[k]{P}$, $P$ — произведение $k$ элементов. При решении этой задачи необходимо найти произведение и количество простых чисел, расположенных между максимальным и минимальным элементами.

Алгоритм решения задачи состоит из следующих этапов:

1. Ввод массива.
2. Поиск номеров максимального (`nmax`) и минимального (`nmin`) элементов массива.
3. В цикле перебираем все элементы массива, расположенные между максимальным и минимальным элементами. Если текущий элемент является простым числом, то необходимо увеличить количество простых чисел на 1, и умножить $P$ на значение элемента массива.
4. Вычислить $SG = \sqrt[k]{P}$ .

При решении этой задачи следуем учитывать, что неизвестно какой элемент расположен раньше максимальный или минимальный.

Текст программы с комментариями приведён ниже.

```
#include <iostream>
#include <math.h>
using namespace std;
bool prostoe (int N)
{
  int i;
  bool pr;
  if (N<2) pr=false;
  else
    for(pr=true, i=2;i<=N/2;i++)
      if (N%i==0)
      {
          pr=false;
          break;
      }
  return pr;
}
int main(int argc, char **argv)
{
  int i,k,n,nmax,nmin, p, *x;
//Ввод количества элементов в массиве.
  cout<<"n="; cin>>n;
//Выделяем память для динамического массива x.
  x=new int [n];
//Ввод элементов массива.
  cout<<"Введите элементы массива X";
  for(i=0;i<n;i++)
    cin>>x[i];
//Поиск номеров максимального и минимального элементов в массиве.
  for(nmax=nmin=i=0;i<n;i++)
  {
    if (x[i]<x[nmin]) nmin=i;
    if (x[i]>x[nmax]) nmax=i;
  }
  if (nmin<nmax)
    for(p=1,k=0,i=nmin+1;i<nmax;i++)
//Обратите особое внимание на использование в следующей строке
//фигурной скобки (составного оператора). В цикле всего один оператор!!!
//при этом, при отсутствии составного оператора, программа
//начинает считать с ошибками!!!
  {
//Проверяем, является ли очередной элемент массива простым числом.
//Если x[i] — простое число.
    if (prostoe(x[i]))
    {
//Домножаем y[i] на p, а также увеличиваем счётчик
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
//количества простых чисел в массиве.
        k++;p*=x[i];
      }
    }
    else
      for(p=1,k=0,i=nmax+1;i<nmin;i++)
//Проверяем, является ли очередной элемент массива простым
//числом. Если x[i] — простое число.
        if (prostoe(x[i]))
        {
//Домножаем y[i] на p, а также увеличиваем счётчик количества
//простых чисел в массиве.
          k++;p*=x[i];
        }
//Если в массиве были простые числа, выводим среднее геометрическое
//простых чисел на экран
        if (k>0)
          cout<<"SG"<<pow(p,1./k)<<endl;
//Иначе выводим сообщение о том, что в массиве нет простых чисел.
        else cout<<"Нет простых чисел в массиве"<<endl;
    return 0;
}
```

### 5.4.5 Удаление элемента из массива

Для удаления элемента с индексом $m$ из массива $X$, состоящего из $n$ элементов нужно записать $(m+1)$-й элемент на место элемента $m$, $(m+2)$-й — на место $(m+1)$-го и т.д., $(n-1)$-й — на место $(n-2)$-го. После удаления количество элементов в массиве уменьшилось на 1 (рис. 5.10).

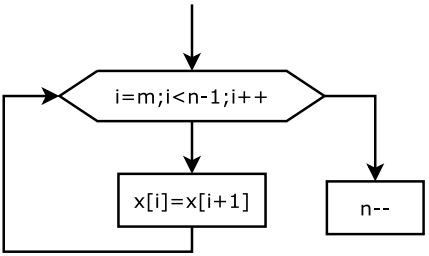


Рис. 5.10. Алгоритм удаления элемента из массива

Фрагмент программы на C++:

```
//Ввод номера элемента, подлежащего удалению.
cout<<"\n m=";  cin>>m;
for  (i=m; i<n-1;X[i]=X[i+1],i++);  //Удаление m-го элемента.
n--;
//Вывод измененного массива.
for  (i=0;i<n-1;i++)cout<<X[i]<<"\t";
```

При написании программ, в которых удаляются элементы из массива следует учитывать тот факт, что после удаления элемента все элементы, расположенные после удалённого изменяют свои номера (индексы увеличиваются на один). Это особенно важно при удалении нескольких элементов из массива. Рассмотрим несколько задач.

**Задача 5.6.** Удалить из массива `x[20]` все элементы с пятого по десятый.

При решении задач, связанных с удалением подряд идущих элементов, следует понимать, что после удаления очередного элемента следующий переместил на место удалённого. Поэтому далее нужно будет удалять элемент с тем же самым номером. В нашем случае подлежит удалению 6 элементов с пятого по десятый. Однако, реально надо будет 6 раз удалить элемент с номером 5. Блок-схема алгоритма представлена на рис 5.11, текст программе приведён далее.

```
#include <iostream>
#include <math.h>
using namespace std;
int main(int argc, char **argv)
{
  int i,j,n=20;
//Выделяем память для динамического массива x.
  float x[n];
//Ввод элементов массива.
  cout<<"Введите элементы массива X
      n";
  for(i=0;i<n;i++)
  cin>>x[i];
//Шесть раз повторяем алгоритм удаления элемента с индексом 5.
  for(j=1;j<=6;j++)
//Удаление элемента с индексом 5.
  for(i=5;i<n-j;i++)
    x[i]=x[i+1];
//Вывод элементов массива.
  cout<<"Преобразованный массив X
      n";
  for(i=0;i<n-6;i++)
    cout<<x[i]<<"\t";
  cout<<endl;
  return 0;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

}

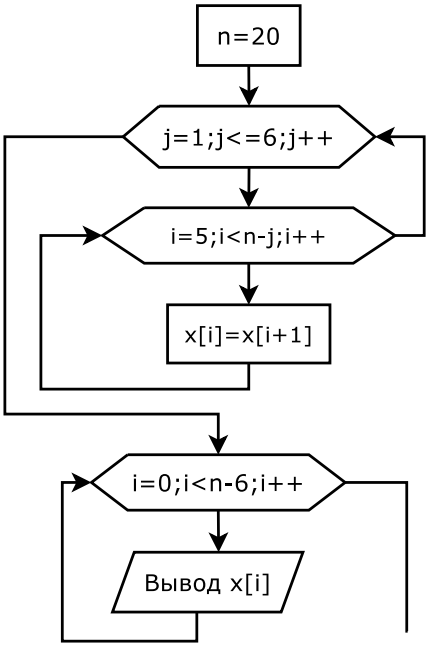


Рис. 5.11. Алгоритм решения задачи 5.6

**Задача 5.7.** Удалить из массива `X[n]` все положительные элементы.

При удалении отдельных элементов из массива следует учитывать: при удалении элемента (сдвиге элементов влево и уменьшении $n$) не надо переходить к следующему, а если элемент не удалялся, то, наоборот, надо переходить к следующему.

Далее приведён текст программы решения задачи 5.7.

```
#include <iostream>
#include <math.h>
using namespace std;
int main(int argc, char **argv)
{
  int i,j,n=20;
//Выделяем память для динамического массива x.
  float x[n];
//Ввод элементов массива.
```

```
   cout<<"Введите элементы массива X\n";
   for(i=0;i<n;i++)
   cin>>x[i];
//Шесть раз повторяем алгоритм удаления элемента с индексом 5.
   for(j=1;j<=6;j++)
//Удаление элемента с индексом 5.
   for(i=5;i<n-j;i++)
   x[i]=x[i+1];
//Вывод элементов массива.
   cout<<"Преобразованный массив X\n";
   for(i=0;i<n-6;i++)
      cout<<x[i]<<"\t";
   cout<<endl;
   return 0;
}
```

**Задача 5.8.** Удалить из массива все отрицательные элементы, расположенные между максимальным и минимальным элементами массива `X[n]`.

Решение этой задачи можно разделить на следующие этапы:

1. Ввод массива.
2. Поиск номеров максимального (`nmax`) и минимального (`nmin`) элементов массива.
3. Определение меньшего (`a`) и большего (`b`) из чисел `nmax` и `nmin`.
4. Далее, необходимо перебрать все элементы массива, расположенные между числами с номерами `a` и `b`. Если число окажется отрицательным, то его необходимо удалить. Однако, на этом этапе нужно учитывать тонкий момент. Если просто организовать цикл от `a+1` до `b-1`, то при удалении элемента, изменяется количество элементов и номер последнего удаляемого элемента, расположенных между `a` и `b`. Это может привести к тому, что не всегда корректно будут удаляться отрицательные элементы, расположенные между `a` и `b`. Поэтому этот цикл для удаления организован несколько иначе.

Текст программы:

```
#include <iostream>
#include <math.h>
using namespace std;
int main(int argc, char **argv)
{
   int i,j,n;
   cout<<"n=";cin>>n;
//Выделяем память для динамического массива x.
   float x[n];
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
//Ввод элементов массива.
   cout<<"Введите элементы массива X\n";
   for(i=0;i<n;i++)
      cin>>x[i];
   for(i=0;i<n;)
//Если текущий элемент положителен,
      if (x[i]>0)
//то удаляем элемента с индексом i.
      {
         for(j=i;j<n-1;j++)
            x[j]=x[j+1];
         n--;
      }
//иначе — переходим к следующему элементу массива.
      else i++;
//Вывод элементов массива.
   cout<<"Преобразованный массив X\n";
   for(i=0;i<n;i++)
      cout<<x[i]<<"\t";
   cout<<endl;
   return 0;
}
```

В качестве тестового можно использовать следующий массив 34 4 -7 -8 -10 7 -100 -200 -300 1. Здесь, приведенная выше программа работает корректно, а вариант

```
for(i=a+1;i<b;)
   if (x[i]<0)
   {
      for(j=i;j<n-1;j++)
         x[j]=x[j+1];
      n--;
   }
   else i++;
```

приводит к неправильным результатам. Рекомендуем читателю самостоятельно разобраться в особенностях подобных алгоритмов удаления.

**Задача 5.9.** В массиве `X[n]` найти группу наибольшей длины, которая состоит из знакочередующихся чисел.

Группа подряд идущих чисел внутри массива можно определить любыми двумя из трёх значений:

- `nach` — номер первого элемента в группе;
- `kon` — номер последнего элемента в группе;
- `k` — количество элементов в группе.

Зная любые два из выше перечисленных значений, мы однозначно определим группу внутри массива. Минимальное число элементов в группе 2.

В начале количество элементов в знакочередующейся группе равно 1. Дело в том, что если мы встретим первую пару знакочередующихся элементов, то количество их в группе сразу станет равным 2. Однако все последующие пары элементов будут увеличивать `k` на 1. И чтобы не решать проблему построения последовательности значений `k` 0,2,3,4,5,..., первоначальное значение `k` примем равным 1. Когда будем встречать очередную пару подряд идущих соседних элементов, то `k` необходимо будет увеличить на 1.

Алгоритм поиска очередной группы состоит в следующем: попарно $(x_i, x_{i+1})$ перебираем все элементы массива `for(i=0;i<n-1;i++)`.

Если произведение соседних элементов $(x_i \cdot x_{i+1})$ отрицательно, то это означает, что они имеют разные знаки и являются элементами группы. В этом случае количество $(k)$ элементов в группе увеличиваем на 1 (`k++`). Если же произведение соседних элементов $(x_i \cdot x_{i+1})$ положительно, то эти элементы не являются членами группы. В этом случае возможны два варианта:

1. Если $k > 1$, то только что закончилась группа, в этом случае `kon=i` — номер последнего элемента в группе, $k$ — количество элементов в только что закончившейся группе.
2. Если $k = 1$, то это просто очередная пара незнакочередующихся элементов.

После того как закончилась очередная группа знакочередующихся элементов, необходимо количество групп (`kgr`) увеличить на 1 (`kgr++`). Если это первая группа (`kgr=1`) знакочередующихся элементов, то в переменную `max` записываем длину этой группы (`max=k`)[3], а в переменную `kon_max` номер последнего элемента группы (`kon_max=i`). Если это не первая группа (`kgr=1`), то сравниваем `max` и длину текущей группы (`k`). Если `k>max`, то в переменную `max` записываем длину этой группы (`max=k`), а в переменную `kon_max` номер последнего элемента группы (`kon_max=i`).

После этого в переменную `k` опять записываем 1 (в группе нет элементов) для формирования новой группы элементов.

По окончанию цикла значение `k` может быть больше 1. Это означает, что в самом конце массива встретилась ещё одна группа. Для

[3] В переменной

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

неё надо будет провести все те же действия, что и для любой другой группы. Далее приведён текст программы.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{ float *x;
  int i,k,n,max,kgr,kon_max;
//Ввод размера массива.
  cout<<"n="; cin>>n;
//Выделение памяти для массива.
  x=new float [n];
//Ввод элементов массива.
  cout<<"Введите массив x\n";
  for(i=0;i<n;i++)
    cin>>x[i];
//Попарно перебираем элементы массива. Количество знакочередующихся
//групп в массиве kgr=0, количество элементов в текущей группе — 1.
  for(kgr=i=0,k=1;i<n-1;i++)
//Если соседние элементы имеют разные знаки, то количество (k)
//элементов в группе увеличиваем на 1.
    if (x[i]*x[i+1]<0) k++;
    else
      if (k>1)
      {
//Если k>1, то только что закончилась группа, i — номер последнего
//элемента в группе, k — количество элементов в группе. Увеличиваем kgr на 1.
        kgr++;
//Если это первая группа (kgr=1) знакочередующихся элементов,
        if (kgr==1)
        {
//то max — длина группы (max=k),
          max=k;
//kon_max — номер последнего элемента группы.
          kon_max=i;
        }
        else
//Если это не первая группа (kgr=1), то сравниваем max и длину
//текущей группы. Если k>max,
          if (k>max)
          {
//max — длина группы,
            max=k;
//kon_max — номер последнего элемента группы.
            kon_max=i;
          }
//В переменную k записываем 1 для формирования новой группы элементов.
          k=1;
      }
//Если в конце массива была группа.
```

```
    if (k>1)
    {
//Количество групп увеличиваем на 1.
      kgr++;
//Если это первая группа,
    if (kgr==1)
    {
//то max — длина группы,
      max=k;
//группа закончилась на последнем элементе массива.
      kon_max=n-1;
    }
    else
//Если длина очередной группы больше max.
      if (k>max)
      {
//то в max записываем длину последней группы,
        max=k;
//группа закончилась на последнем элементе массива.
        kon_max=n-1;
      }
    }
//Если знакочередующиеся группы были,
    if (kgr>0)
    {
//то выводим информацию о группе наибольшей длины,
      cout<<"В массиве "<<kgr<<" групп знакочередующихся
          элементов\n";
      cout<<"Группа максимальной длины начинается с элемента Номер
          "<<kon_max-max+1<<", её длина "<<max<<" ,номер последнего
          элемента группы " <<kon_max<<endl;
//а также саму группу.
      for(i=kon_max-max+1;i<=kon_max;i++)
        cout<<x[i]<<" ";
      cout<<endl;
    }
//Если знакочередующихся групп не было, то выводим сообщение об этом.
    else
      cout<<"В массиве нет групп знакочередующихся элементов\n";
  return 0;
}
```

### 5.4.6 Сортировка элементов в массиве

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Например, массив $Y$ из $n$ элементов будет отсортирован в порядке возрастания значений его элементов, если

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

$Y[0] < Y[1] < \ldots < Y[n-1]$,
и в порядке убывания, если
$Y[0] > Y[1] > \ldots > Y[n-1]$.

Существует большое количество алгоритмов сортировки, но все они базируются на трех основных:

- сортировка обменом;
- сортировка выбором;
- сортировка вставкой.

Представим, что нам необходимо разложить по порядку карты в колоде. Для сортировки карт *обменом* можно разложить карты на столе лицевой стороной вверх и менять местами те карты, которые расположены в неправильном порядке, делая это до тех пор, пока колода карт не станет упорядоченной.

Для *сортировки выбором* из разложенных на столе карт выбирают самую младшую (старшую) карту и держат ее в руках. Затем из оставшихся карт вновь выбирают наименьшую (наибольшую) по значению карту и помещают ее позади той карты, которая была выбрана первой. Этот процесс повторяется до тех пор, пока вся колода не окажется в руках. Поскольку каждый раз выбирается наименьшая (наибольшая) по значению карта из оставшихся на столе карт, по завершению такого процесса карты будут отсортированы по возрастанию (убыванию).

Для *сортировки вставкой* из колоды берут две карты и располагают их в необходимом порядке по отношению друг к другу. Каждая следующая карта, взятая из колоды, должна быть установлена на соответствующее место по отношению к уже упорядоченным картам.

Итак, решим следующую задачу. Задан массив $Y$ из $n$ целых чисел. Расположить элементы массива в порядке возрастания их значений.

#### 5.4.6.1 Сортировка методом «пузырька»

Сортировка пузырьковым методом является наиболее известной. Ее популярность объясняется запоминающимся названием, которое происходит из-за подобия процессу движения пузырьков в резервуаре с водой, когда каждый пузырек находит свой собственный уровень, и простотой алгоритма. Сортировка методом «пузырька» использует метод обменной сортировки и основана на выполнении в цикле операций сравнения и при необходимости обмена соседних элементов. Рассмотрим алгоритм пузырьковой сортировки более подробно.

Сравним нулевой элемент массива с первым, если нулевой окажется больше первого, то поменяем их местами. Те же действия выполним для первого и второго, второго и третьего, $i$–го и $(i+1)$–го, предпоследнего и последнего элементов. В результате этих действий самый большой элемент станет на последнее $(n-1)$-е место. Теперь повторим данный алгоритм сначала, но последний $(n-1)$-й элемент, рассматривать не будем, так как он уже занял свое место. После проведения данной операции самый большой элемент оставшегося массива станет на $(n-2)$-е место. Так повторяем до тех пор, пока не упорядочим весь массив.

В табл. 5.4 представлен процесс упорядочивания элементов в массиве.

Таблица 5.4: Процесс упорядочивания элементов

| **Номер элемента** | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| Исходный массив | 7 | 3 | 5 | 4 | 2 |
| Первый просмотр | 3 | 5 | 4 | 2 | 7 |
| Второй просмотр | 3 | 4 | 2 | 5 | 7 |
| Третий просмотр | 3 | 2 | 4 | 5 | 7 |
| Четвертый просмотр | 2 | 3 | 4 | 5 | 7 |

Нетрудно заметить, что для преобразования массива, состоящего из $n$ элементов, необходимо просмотреть его $n-1$ раз, каждый раз уменьшая диапазон просмотра на один элемент. Блок–схема описанного алгоритма приведена на рис. 5.12.

Обратите внимание на то, что для перестановки элементов (рис. 5.12, блок 4) используется буферная переменная $b$, в которой временно хранится значение элемента, подлежащего замене. Текст программы, сортирующей элементы в массиве по возрастанию методом «пузырька» приведён далее.

```
#include <iostream>
using namespace std;
int main()
{
  int n,i,b,j;
  cout<<" n="; cin>>n;
  float y[n];
  for (i=0;i<n;i++)    //Ввод массива.
  {
    cout<<"\n Y["<<i<<"]=";
    cin>>y[i];
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

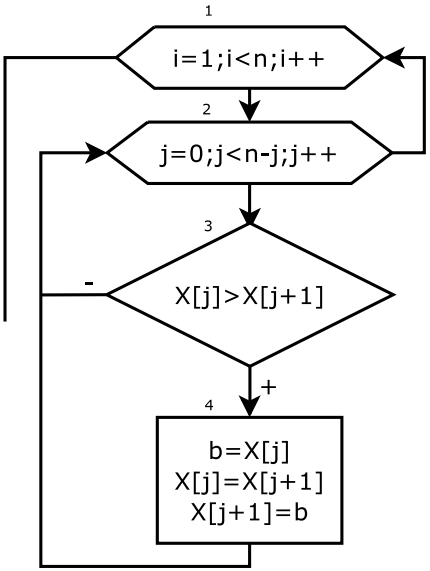


Рис. 5.12. Сортировка массива пузырьковым методом

```
  }
//Упорядочивание элементов в массиве по возрастанию их значений.
  for(j=1;j<n;j++)
    for(i=0;i<n-j;i++)
      if (y[i]>y[i+1]) //Если текущий элемент больше следующего
      {
        b=y[i]; //Сохранить значение текущего элемента
        y[i]=y[i+1]; //Заменить текущий элемент следующим
        y[i+1]=b; //Заменить следующий элемент текущим
      }
//Вывод упорядоченного массива
  for (i=0;i<n;i++) cout<<y[i]<<"\t";
  return 0;
}
```

Для перестановки элементов в массиве по убыванию их значений необходимо в программе и блок-схеме при сравнении элементов массива заменить знак $>$ на $<$.

Однако, в этом и во всех ниже рассмотренных алгоритмах не учитывается то факт, что на каком-то этапе (или даже в начале) массив

уже может оказать отсортированным. При большом количестве элементов (сотни и даже тысячи чисел) на *сортировку «в холостую»* массива тратится достаточно много времени. Ниже приведены тексты двух вариантов программы сортировки по убыванию методом пузырька, в которых алгоритм прерывается, если массив уже отсортирован.

Вариант 1.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
  int n,i,b,j;
  bool pr;
  cout<<" n="; cin>>n;
  float y[n];
  for (i=0;i<n;i++)  //Ввод массива.
  {
    cout<<"\n Y["<<i<<"]=";
    cin>>y[i];
  }
//Упорядочивание элементов в массиве по убыванию их значений.
  for(j=1;j<n;j++)
  {
//Предполагаем, что массив уже отсортирован (pr=false).
    for(pr=false, i=0;i<n-j;i++)
      if (y[i]<y[i+1])//Если текущий элемент меньше следующего
      {
        b=y[i]; //Сохранить значение текущего элемента
        y[i]=y[i+1]; //Заменить текущий элемент следующим
        y[i+1]=b; //Заменить следующий элемент текущим
//Если элемент менялись местами, массив ещё неотсортирован (pr=true);
        pr=true;
      }
    cout<<"j="<<j<<endl;
//Если на j-м шаге соседние элементы не менялись,
//то массив уже отсортирован, повторять смысла нет;
    if (!pr) break;
  }
//Вывод упорядоченного массива
  for (i=0;i<n;i++) cout<<y[i]<<"\t";
  return 0;
}
```

Вариант 2.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
{
  int n,i,b,j;
  bool pr=true;
  cout<<" n="; cin>>n;
  float y[n];
  for (i=0;i<n;i++) //Ввод массива.
  {
    cout<<"\n Y["<<i<<"]=";
    cin>>y[i];
  }
//Упорядочивание элементов в массиве по убыванию их значений.
//Вход в цикл, если массив не отсортирован (pr=true).
  for(j=1;pr;j++)
  {
//Предполагаем, что массив уже отсортирован (pr=false).
    for(pr=false, i=0;i<n-j;i++)
      if (y[i]<y[i+1])//Если текущий элемент меньше следующего
      {
        b=y[i]; //Сохранить значение текущего элемента
        y[i]=y[i+1]; //Заменить текущий элемент следующим
        y[i+1]=b; //Заменить следующий элемент текущим
//Если элемент менялись местами, массив ещё неотсортирован (pr=true)
        pr=true;
      }
  }
//Вывод упорядоченного массива
  for (i=0;i<n;i++) cout<<y[i]<<"\t";
  return 0;
}
```

### 5.4.6.2 Сортировка выбором

Алгоритм сортировки выбором приведён в виде блок-схемы на рис. 5.13. Идея алгоритма заключается в следующем. В массиве $Y$ состоящем из $n$ элементов ищем самый большой элемент (блоки 2–5) и меняем его местами с последним элементом (блок 7). Повторяем алгоритм поиска максимального элемента, но последний $(n-1)$-й элемент не рассматриваем, так как он уже занял свою позицию.

Найденный максимум ставим на $(n-2)$-ю позицию. Описанную выше операцию поиска проводим $n-1$ раз, до полного упорядочивания элементов в массиве. Фрагмент программы выполняет сортировку массива по возрастанию методом выбора:

```
for(j=1;j<n;b=y[n-j],y[n-j]=y[nom],y[nom]=b,j++)
for(max=y[0],nom=0,i=1;i<=n-j;i++)
if (y[i]>max) {max=y[i];nom=i;}
```

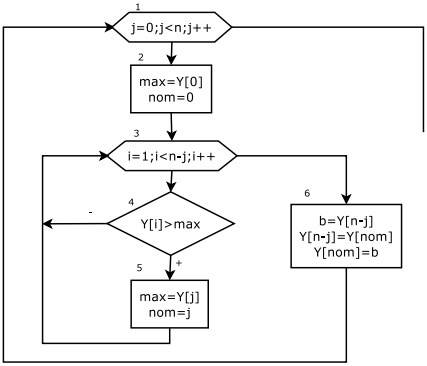


Рис. 5.13. Сортировка массива выбором наибольшего элемента

Для упорядочивания массива по убыванию необходимо менять минимальный элемент с последним элементом.

### 5.4.6.3 Сортировка вставкой

Сортировка вставкой заключается в том, что сначала упорядочиваются два элемента массива. Затем делается вставка третьего элемента в соответствующее место по отношению к первым двум элементам. Четвертый элемент помещают в список из уже упорядоченных трех элементов. Этот процесс повторяется до тех пор, пока все элементы не будут упорядочены.

Прежде чем приступить к составлению блок–схемы рассмотрим следующий пример. Пусть известно, что в массиве из десяти элементов первые шесть уже упорядочены (с нулевого по пятый), а шестой элемент нужно вставить между вторым и четвертым. Сохраним шестой элемент во вспомогательной переменной, а на его место запишем пятый. Далее четвертый переместим на место пятого, а третий на место четвертого, тем самым, выполнив сдвиг элементов массива на одну позицию вправо. Записав содержимое вспомогательной переменной в третью позицию, достигнем нужного результата.

Составим блок–схему алгоритма (рис. 5.14), учитывая, что возможно описанные выше действия придется выполнить неоднократно.

Организуем цикл для просмотра всех элементов массива, начиная с первого (блок 1). Сохраним значение текущего $i$–го элемента во

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

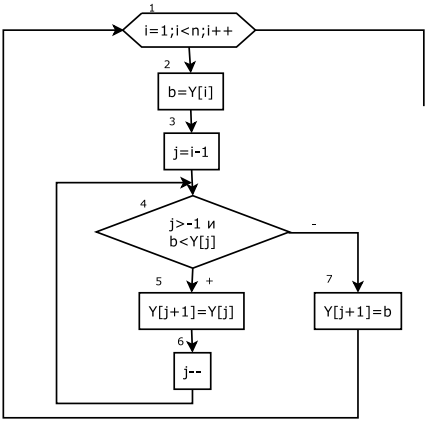


Рис. 5.14. Сортировка массива вставкой

вспомогательной переменной $b$, так как оно может быть потеряно при сдвиге элементов (блок 2) и присвоим переменной $j$ значение индекса предыдущего $(i-1)$–го элемента массива (блок 3). Далее движемся по массиву влево в поисках элемента меньшего, чем текущий и пока он не найден сдвигаем элементы вправо на одну позицию. Для этого организуем цикл (блок 4), который прекратиться, как только будет найден элемент меньше текущего. Если такого элемента в массиве не найдется и переменная $j$ станет равной (-1), то это будет означать, что достигнута левая граница массива, и текущий элемент необходимо установить в первую позицию. Смещение элементов массива вправо на одну позицию выполняется в блоке 5, а изменение счетчика $j$ в блоке 6. Блок 7 выполняет вставку текущего элемента в соответствующую позицию.

Далее приведен фрагмент программы, реализующей сортировку массива методом вставки.

```
for ( i =1; i<n ; y [ j +1]=b , i++)
for (b=y [ i ] , j=i -1;( j >-1 && b<y [ j ] ) ; y [ j +1]=y [ j ] , j --);
```

Рассмотрим несколько несложных задач, связанных с упорядочиванием.

**Задача 5.10.** Задан массив $a[n]$ упорядоченный по убыванию, вставить в него некоторое число b, не нарушив упорядоченности массива.

Массив является упорядоченным по убыванию, если каждое последующий элемент массива не больше предыдущего, т. е. при выполнении следующей совокупности неравенств $a_0 \geq a_1 \geq a_2 \geq ... \geq a_{n-3} \geq a_{n-2} \geq a_{n-1}$.

Для вставки в подобный массив некоторого числа без нарушений упорядоченности, необходимо:

1. Найти номер $k$ первого числа в массиве, которое $a_k \leq b$.
2. Все элементы массива $a$, начиная от `n-1` до $k$-го сдвинуть на один вправо[4].
3. На освободившееся место с номером $k$ записать число $b$.

Текст программы с комментариями приведён ниже.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
  int i,k,n;
  float b;
//Ввод размера исходного массива.
  cout<<"n=";cin>>n;
//Выделение памяти с учётом предстоящей вставки одного числа в массив.
  float a[n+1];
//Ввод исходного упорядлченного по убыванию массива.
  cout<<"Введите массив a\n";
  for(i=0;i<n;i++)
    cin>>a[i];
//Ввод вставляемого в массив числа b.
  cout<<"Введите число b=";cin>>b;
//Если число b меньше всех элементов массива, записываем b
//в последний элемент массива.
  if (a[n-1]>=b) a[n]=b;
  else
//Иначе
  {
    for(i=0;i<n;i++)
//Ищем первое число, меньшее b.
      if (a[i]<=b)
      {
//Запоминаем его номер в переменной k.
        k=i;
        break;
      }
//Все элементы массива от n-1-го до k-го сдвигаем на один вправо.
```

[4]Очень важно, что сдвиг осуществляем от $n-1$-го до $k$-го, в противном случае элементы массива оказались бы испорченными.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
    for(i=n-1;i>=k;i--)
      a[i+1]=a[i];
//Вставляем число b в массив.
    a[k]=b;
  }
  cout<<"Преобразованный массив a\n";
  for(i=0;i<=n;i++)
    cout<<a[i]<<"\t";
  return 0;
}
```

Обратите внимание, при решении задачи с массивом упорядоченным по возрастанию необходимо во фрагменте

```
if (a[n-1]>=b) a[n]=b;
else
{
  for(i=0;i<n;i++)
    if (a[i]<=b)
    {
      k=i;
      break;
    }
```

заменить все операции отношения на противоположные.

**Задача 5.11.** Проверить является ли массив упорядоченным возрастанию.

Для проверки упорядоченности по возрастанию `a[n]`[5] можно поступить следующим образом. Предположим, что массив упорядочен (`pr=true`). Если хотя бы для одной пары соседних элементов выполняется условие $a_i > a_{i+1}$, то массив не упорядочен по возрастанию (`pr=false`). Текст программы с комментариями приведён ниже. Читателю предлагается преобразовать программу таким образом, чтобы осуществлялась проверка, упорядочен ли массив по убыванию.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
  int i,n;
  bool pr;
//Ввод размера исходного массива.
  cout<<"n="; cin>>n;
//Выделение памяти для массива.
  float *a=new float [n];
```

[5] Массив является упорядоченным по возрастанию, если выполняются условия $a_0 \le a_1 \le a_2 \le ... \le a_{n-3} \le a_{n-2} \le a_{n-1}$.

```
//Ввод исходного массива.
   cout<<"Введите массив a\n";
   for(i=0;i<n;i++)
      cin>>a[i];
//Предполагаем, что массив упорядочен (pr=true),
//Перебираем все пары соседних значений (i — номер пары),
//i равном n-2 будем сравнивать последнюю пару a[n-2] и a[n-1].
   for(pr=true,i=0;i<n-1;i++)
//Если для очередной пары соседних элементов выяснилось, что предыдущий
//элемент больше последующего, то массив неупорядочен по возрастанию
//(pr=false), остальные пары соседних значений, можно не проверять
      (оператор break)
      if (a[i]>a[i+1]) {pr=false;break;}
   if (pr) cout<<"Массив упорядочен по возрастанию";
   else cout<<"Массив не упорядочен по возрастанию";
   return 0;
}
```

## 5.5 Указатели на функции

При решении некоторых задач возникает необходимость передавать имя функции, как параметр. В этом случае формальным параметром является указатель на передаваемую функцию. В общем виде прототип указателя на функцию можно записать так.

```
type (*name\_f)(type1, type2, type3,...)
```

Здесь

`name_f` — имя функции

`type` — тип возвращаемый функцией,

`type1, type2, type3,...` — типы формальных параметров функции.

В качестве примера рассмотрим решение широко известной математической задачи.

**Задача 5.12.** Вычислить $\int_a^b f(x)dx$ методами Гаусса и Чебышева.

Кратко напомним читателю методы численного интегрирования.

Метод Гаусса состоит в следующем. Определённый интеграл непрерывной функции на интервале от -1 до 1 можно заменить суммой и вычислить по формуле $\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} A_i f(t_i)$, $t_i$ — точки из интервала [-1,1], $A_i$ — рассчитываемые коэффициенты. Методика определения $A_i$, $t_i$ представлена в [3]. Для практического использо-

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

вания значения коэффициентов при $n = 2, 3, 4, 5, 6, 7, 8$ представлены в табл. 5.5.

Таблица 5.5: Значения коэффициентов в квадратурной формуле Гаусса

| **n** | **Массив t** | **Массив A** |
|---|---|---|
| 2 | −0.57735027, 0.57735027 | 1, 1 |
| 3 | −0.77459667, 0, 0.77459667 | 5/9, 8/9, 5/9 |
| 4 | −0.86113631, −0.33998104, 0.33998104, 0.86113631 | 0.34785484, 0.65214516, 0.65214516, 0.34785484 |
| 5 | −0.90617985, −0.53846931, 0, 0.53846931, 0.90617985 | 0.23692688, 0.47862868, 0.568888889, 0.47862868, 0.23692688 |
| 6 | −0.93246951, −0.66120939, −0.23861919, 0.23861919, 0.66120939, 0.93246951 | 0.17132450, 0.36076158, 0.46791394, 0.46791394, 0.36076158, 0.17132450 |
| 7 | −0.94910791, −0.74153119, −0.40584515, 0, 0.40584515, 0.74153119, 0.94910791 | 0.12948496, 0.27970540, 0.38183006, 0.41795918, 0.38183006, 0.27970540, 0.12948496 |
| 8 | −0.96028986, −0.79666648, −0.52553242, −0.18343464, 0.18343464, 0.52553242, 0.79666648, 0.96028986 | 0.10122854, 0.22238104, 0.31370664, 0.36268378, 0.36268378, 0.31370664, 0.22238104, 0.10122854 |

Для вычисления интеграла непрерывной функции на интервале ог a до b квадратурная формула Гаусса может быть записана следующим образом $\int\limits_a^b f(x)dx = \frac{b-a}{2} \sum\limits_{i=1}^{n} A_i f\left(\frac{b+a}{2} \cdot \frac{b-a}{2} t_i\right)$ , значения коэффициентов $A_i$ и $t_i$ приведены в табл. 5.5.

При использовании квадратурной формулы Чебышева определённый интеграл непрерывной функции на интервале от -1 до 1 записывается в виде следующей формулы $\int\limits_{-1}^{1} f(x)dx = \frac{2}{n} \sum\limits_{i=1}^{n} f(t_i)$, $t_i$ — точки из интервала $[-1, 1]$. Формула Чебышева для вычисления интеграла на интервале от $a$ до $b$ может быть записана так $\int\limits_a^b f(x)dx = \frac{b-a}{n} \sum\limits_{i=1}^{n} f\left(\frac{b+a}{2} \cdot \frac{b-a}{2} t_i\right)$ Методика определения $t_i$ представлена в [3]. Рассмотренные формулы имеют смысл при $n = 2, 3, 4, 5, 6, 7, 9$ , коэффициенты $t_i$ представлены в табл. 5.6.

Таблица 5.6: Значения коэффициентов в квадратурной формуле Чебышева

| **n** | **Массив t** |
|---|---|
| 2 | −0.577350, 0.577350 |

Таблица 5.6 — продолжение

| n | Массив t |
|---|---|
| 3 | −0.707107, 0, −0.707107 |
| 4 | −0.794654, −0.187592, 0.187592, 0.794654 |
| 5 | −0.832498, −0.374541, 0, 0.374541, 0.832498 |
| 6 | −0.866247, −0.422519, −0.266635, 0.266635, 0.422519, 0.866247 |
| 7 | −0.883862, −0.529657, −0.323912, 0, 0.323912, 0.529657, 0.883862 |
| 9 | −0.911589, −0.601019, −0.528762, −0.167906, 0, 0.167906, 0.528762, 0.601019, 0.911589 |

Осталось написать функции вычисления определённого интеграла $\int\limits_a^b f(x)dx$ методами Гаусса и Чебышева. Далее приведены тексты функций и функция main(). В качестве тестовых использовались интегралы $\int\limits_0^2 \sin^4 x dx \approx 0.9701$, $\int\limits_5^{13} \sqrt{2x-1}dx \approx 32.667$ .

```
#include <iostream>
#include <math.h>
using namespace std;
//Функция вычисления определённого интеграла методом Чебышева.
//(a,b) — интервал интегрирования, *fn —
//указатель на функцию типа double f (double).
double int_chebishev(double a, double b,
double (*fn)(double))
{
  int i,n=9;
  double s,
  t[9]={-0.911589, -0.601019, -0.528762, -0.167906, 0,
       0.167906, 0.528762, 0.601019,  0.911589};
  for(s=i=0;i<n;i++)
    s+=fn((b+a)/2+(b-a)/2*t[i]);
  s*=(b-a)/n;
  return s;
}
//Функция вычисления определённого интеграла методом Гаусса.
//(a,b) — интервал интегрирования, *fn —
//указатель на функцию типа double f (double)
double int_gauss(double a, double b, double (*fn)(double))
{
  int i,n=8;
  double s,
  t[8]={-0.96028986, -0.79666648, -0.52553242, -0.18343464,
       0.18343464, 0.52553242,  0.79666648, 0.96028986},
  A[8]={0.10122854, 0.22238104, 0.31370664, 0.36268378,
       0.36268378, 0.31370664, 0.22238104, 0.10122854};
  for(s=i=0;i<n;i++)
    s+=A[i]*fn((b+a)/2+(b-a)/2*t[i]);
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  s*=(b-a)/2;
  return s;
}
//Функции f1 и f2 типа double f(double), указатели на
//которые будут передаваться в int_gauss и int_chebishev.
double f1(double y)
{
  return sin(y)*sin(y)*sin(y)*sin(y);
}
double f2(double y)
{
  return pow(2*y-1,0.5);
}

int main(int argc, char **argv)
{
  double a,b;
  cout<<"Интеграл sin(x)^4=\n";
  cout<<"Введите интервалы интегрирования\n";
  cin>>a>>b;
//Вызов функции int_gauss(a, b, f1), f1 — имя функции,
//  интеграл от которой надо посчитать.
  cout<<"Метод Гаусса:"<<int_gauss(a, b, f1)<<endl;
//Вызов функции int_chebishev(a, b, f1), f1 — имя функции,
//интеграл от которой надо посчитать.
  cout<<"Метод Чебышева:"<<int_chebishev(a,b,f1)<<endl;
  cout<<"Интеграл sqrt(2*x-1)=\n";
  cout<<"Введите интервалы интегрирования\n";
  cin>>a>>b;
//Вызов функции int_gauss(a, b, f2), f2 — имя функции,
//  интеграл от которой надо посчитать.
  cout<<"Метод Гаусса:"<<int_gauss(a, b, f2)<<endl;
//Вызов функции int_chebishev(a, b, f2), f2 — имя функции,
//интеграл от которой надо посчитать.
  cout<<"Метод Чебышева:"<<int_chebishev(a,b,f2)<<endl;
  return 0;
}
```

Результаты работы программы приведены ниже

```
Интеграл sin(x)^4=
Введите интервалы интегрирования
0 2
Метод Гаусса:0.970118
Метод Чебышева:0.970082
Интеграл sqrt(2*x-1)=
Введите интервалы интегрирования
5 13
Метод Гаусса:32.6667
```

```
Метод Чебышева:32.6667
```

## 5.6 Совместное использование динамических массивов, указателей, функций в сложных задачах обработки массивов

Функции в C(C++) могут возвращать только одно скалярное значение, однако использование указателей в качестве аргументов функций позволяет обойти это ограничение и писать сложные функции, которые могут возвращать несколько значений.

Если в качестве аргумента в функцию передаётся указатель (адрес), то следует иметь в виду следующее. *При изменении в функции значений*, хранящегося по этому адресу, *будет происходить глобальное изменение значений*, хранящихся по данному адресу в памяти компьютера. Таким образом, получаем механизм с помощью которого можно возвращать множество значений. Для этого просто надо передавать их, как адреса (указатели). В литературе по программированию подобный механизм зачастую называют *передачей параметров по адресу*. При этом не следует забывать о том, что этот механизм работает без всяких исключений. Любое изменение значений, переданных в функцию по адресу, приводит к глобальному изменению.

В качестве примера рассмотрим задачу *удаления положительных элементов из массива* (см. задачу 5.7). Пользуясь тем, что задача несложная, напишем несколько вариантов функции удаления элемента с заданным номером из массива.

Для решения задачи удалении положительных элементов из массива понадобится функция удаления элемента из массива.

Назовём функцию `udal`. Её входными параметрами будут:

- массив (`x`),
- его размер (`n`),
- номер удаляемого элемента (`k`).

Функция возвращает:

- модифицированный массив (`x`),
- размер массива после удаления (`n`).

При передаче массива с помощью указателя, исчезает проблема возврата в главную программу модифицированного массива, размер массива будем возвращать с помощью обычного оператора `return`.

Заголовок (прототип) функции `udal` может быть таким:

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
int udal (float *x, int k, int n)
```

Здесь `x` — массив, `k` — номер удаляемого элемента, `n` — размер массива.

Весь текст функции можно записать так

```
int udal(float *x, int k, int n)
{
  int i;
  if (k>n-1) return n;
  else
  {
    for(i=k;i<n-1;i++)
      x[i]=x[i+1];
    n--;
    return n;
  }
}
```

Ниже приведён весь текст программы удаления положительных элементов из массива `x` с использованием функции `udal` и комментарии к нему.

```
#include <iostream>
#include <math.h>
using namespace std;
int udal(float *x, int k, int n)
{
  int i;
//Если номер удаляемого элемента больше номера последнего элемента,
//то удалять нечего, в этом случае возвращается неизменённый размер массива
  if (k>n-1) return n;
  else
  {
//Удаляем элемент с номером k.
    for(i=k;i<n-1;i++)
      x[i]=x[i+1];
    n--;
//Возвращаем изменённый размер массива.
    return n;
  }
}
int main(int argc, char **argv)
{
  int i,n;
  cout<<"n=";cin>>n;
//Выделяем память для динамического массива x.
  float x[n];
//Ввод элементов массива.
  cout<<"Введите элементы массива X\n";
```

```
  for(i=0;i<n;i++)
    cin>>x[i];
  for(i=0;i<n;)
    if (x[i]>0)
//Если текущий элемент положителен, то для удаления элемента с индексом i,
//Вызываем функцию udal, которая изменяет элементы, хранящиеся по адресу
    x,
//и возвращает размер массива.
      n=udal(x,i,n);
//иначе (x[i]<=0) — переходим к следующему элементу массива.
    else i++;
//Вывод элементов массива после удаления.
cout<<"Преобразованный массив X\n";
  for(i=0;i<n;i++)
    cout<<x[i]<<"\t";
  cout<<endl;
  return 0;
}
```

Эту функцию можно переписать и по другому, передавая и массив и его размер, как указатели, в этом случае функция будет такой.

```
void udal(float *x, int k, int *n)
{
  int i;
  for(i=k;i<*n-1;i++)
    x[i]=x[i+1];
  if (k<*n) --*n;
}
```

В этом случае изменится и обращение к `udal` в функции `main`.

Ниже приведён модифицированный текст программы удаления положительных элементов из массива `x` с использованием функции `udal(float *x, int k, int *n)` и комментарии к нему.

```
#include <iostream>
#include <math.h>
using namespace std;
void udal(float *x, int k, int *n)
{
  int i;
//Если номер удаляемого элемента больше номера последнего элемента,
//то удалять нечего, в этом случае возвращается неизменённый размер массива
//Удаляем элемент с номером k.
  for(i=k;i<*n-1;i++)
    x[i]=x[i+1];
//Уменьшаем на 1 значение, хранящееся по адресу n.
//Обратите внимание, что надо писать именно --*n,
//*n-- — НЕПРАВИЛЬНО!!!!!!!!!!!!!!
  if (k<*n) --*n;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
}
int main(int argc, char **argv)
{
  int i,n;
  cout<<"n=";cin>>n;
//Выделяем память для динамического массива x.
  float x[n];
//Ввод элементов массива.
  cout<<"Введите элементы массива X\n";
  for(i=0;i<n;i++)
    cin>>x[i];
  for(i=0;i<n;)
    if (x[i]>0)
//Если текущий элемент положителен, то удаления элемента с индексом i,
//Вызываем функцию udal, которая изменяет элементы, хранящиеся по адресу
//x, и изменяет значение переменной n.
      udal(x,i,&n);
//иначе (x[i]<=0) — переходим к следующему элементу массива.
    else i++;
//Вывод элементов массива после удаления.
  cout<<"Преобразованный массив X\n";
  for(i=0;i<n;i++)
    cout<<x[i]<<"\t";
  cout<<endl;
  return 0;
}
```

Авторы рекомендуют разобраться с этими примерами для понимания механизма передачи параметров по адресу.

**Задача 5.13.** Из массива целых чисел удалить все простые числа, значение которых меньше среднего арифметического элементов массива. Полученный массив упорядочить по возрастанию.

Алгоритм решения этой задачи без применения функций будет очень громоздким, а текст программы малопонятным. Поэтому, разобьем задачу на подзадачи:

- вычисление среднего арифметического элементов массива;
- определение простого числа;
- удаление элемента из массива;
- упорядочивание массив.

Прототипы функций, которые предназначены для решения подзадач, могут выглядеть так:

- `float sr_arifm(int *x, int n)` — вычисляет среднее арифметическое массива x из n элементов;

- `bool prostoe(int n)` — проверяет, является ли целое число $n$ простым, результат логическое значение `true`, если число простое и `false`, в противном случае;
- `void udal(int *x, int m, int *n)` — удаляет элемент с номером $m$ в массиве $x$ из $n$ элементов (рис. **??**);
- `void upor(int *x, int N, bool pr=true)` — сортирует массив $x$ из $n$ элементов по возрастанию или по убыванию, направление сортировки зависит от значения параметра `pr`, если `pr=true`, то выполняется сортировка по возрастанию, если `pr=false`, то по убыванию.

Текст программы с комментариями:

```
#include <iostream>
using namespace std;
//Функция вычисления среднего значения.
float sr_arifm(int *x, int n)
{
  int i; float s=0;
  for(i=0;i<n;s+=x[i],i++);
  if (n>0) return(s/n);
  else return 0;
}
//Функция для определения простого числа:
bool prostoe(int n)
{
  bool pr; int i;
  for(pr=true,i=2;i<=n/2;i++)
  if(n%i==0) {pr=false;break;}
  return(pr);
}
//Функция удаления элемента из массива.
void udal(int *x, int m, int *n)
{
int i;
for(i=m;i<*n-1;*(x+i)=*(x+i+1),i++);
  --*n;
realloc((int *)x,*n*sizeof(int));
}
//Функция сортировки массива.
void upor(int *x, int n, bool pr=true)
{
int i,j,b;
if (pr)
{
  for(j=1;j<=n-1;j++)
  for(i=0;i<=n-1-j;i++)
  if (*(x+i)>*(x+i+1))
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
  {
    b=*(x+i);
    *(x+i)=*(x+i+1);
    *(x+i+1)=b;
  }
}
else
  for(j=1;j<=n-1;j++)
for(i=0;i<=n-1-j;i++)
  if (*(x+i)<*(x+i+1))
  {
    b=*(x+i);
    *(x+i)=*(x+i+1);
    *(x+i+1)=b;
  }
}
int main()
{
int *a,n,i; float sr;
cout<<"n="; cin>>n;     //Ввод размерности массива.
a=(int *)calloc(n,sizeof(int)); //Выделение памяти.
cout << "Введите массив A\n";
for(i=0;i<n;i++) cin>>*(a+i); //Ввод массива.
sr=sr_arifm(a,n);     //Вычисление среднего арифметического.
cout<<"sr="<<sr<<"\n";   //Вывод среднего арифметического.
for(i=0;i<n;)
{
//Если число простое и меньше среднего,
  if(prostoe(*(a+i))&& *(a+i)<sr)
    udal(a,i,&n);   //удалить его из массива,
  else i++;          //иначе, перейти к следующему элементу.
}
cout << "Массив A\n";   //Вывод модифицированного массива.
for(i=0;i<n;i++) cout<<*(a+i)<<"\t";
cout<<"\n";
upor(a, n);      //Сортировка массива.
//Вывод упорядоченного массива.
cout<<"Упорядоченный массив A\n";
for(i=0;i<n;i++)cout<<*(a+i)<<"\t";
cout<<"\n";
free(a); //Освобождение памяти.
return 0;
}
```

**Задача 5.14.** Все положительные элементы целочисленного массива $G$ переписать в массив $W$. В массиве $W$ упорядочить по убыванию элементы, которые расположены между наибольшим и наименьшим числами палиндромами.

Для создания этой программы напишем следующие функции:

- `int form (int *a, int n, int *b)`, которая из массива целых чисел $a$ формирует массив положительных чисел $b$, $n$ — количество чисел в массиве $a$, функция возвращает число элементов в массиве $b$.
- `bool palindrom (int n)`, которая проверяет является ли число $n$ палиндромом.
- `sort (int *x, int n, int k, int p)` которая сортирует по возрастанию элементы массива `x[n]`, расположенные между $k$-м и $p$-м элементами массива.

Рекомендуем читателю самостоятельно разобрать текст программы, реализующей решение задачи 5.14.

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
int kvo_razryad(int M)
{
        long int k;
        for(k=1;M>9;M/=10,k++);
        return k;
}
bool palindrom (int n)
{
        int k=kvo_razryad(n),s,p=n;
        for(s=0;p!=0;p/=10,k--)
        s+=(p%10)*pow(10,k-1);
        if (s==n) return true; else return false;

}
int form (int *a, int n, int *b)
{
        int i,k;
        for(i=k=0;i<n;i++)
        if(a[i]>0)
        b[k++]=a[i];
        return k;
}
void sort (int *x, int n, int k, int p)
{
        int i,nom,j;
        int b;
        for(i=k+1;i<p;)
        {
                nom=i;
                for(j=i+1;j<p;j++)
                if (x[j]<x[nom]) nom=j;
```

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

```
                b=x[p-1]; x[p-1]=x[nom]; x[nom]=b;
                p--;
        }
}
int main(int argc, char **argv)
{
        int *G,*W;
        int nmax,nmin,kp,i,N,k;
        cout<<"N=";
        cin>>N;
        G=(int *)calloc(N,sizeof(int));
        W=(int *)calloc(N,sizeof(int));
        cout<<"Ввод массива G\n";
        for(i=0;i<N;i++)
        cin>>G[i];
        k=form(G,N,W);
        cout<<"Вывод массива W\n";
        for(i=0;i<k;i++)
        cout<<W[i]<<" ";
        cout<<endl;
        for(kp=i=0;i<k;i++)
        if (palindrom(W[i]))
        {
                kp++;
                if (kp==1) {nmax=i;nmin=i;}
                else
                {
                        if (W[i]<W[nmin]) nmin=i;
                        if (W[i]>W[nmax]) nmax=i;
                }
        }
        if (nmax<nmin)
        sort(W,k,nmax,nmin);
        else
        sort(W,k,nmin,nmax);
        cout<<"Вывод преобразованного массива W\n";
        for(i=0;i<k;i++)
        cout<<W[i]<<" ";
        cout<<endl;
        return 0;
}
```

Результаты работы программы представлены ниже.

```
N=17
Ввод массива G
-5 -6 191 121 12 -13 14 15 -5 100 666 -666 15251 16261 16262 991 -724
Вывод массива W
191 121 12 14 15 100 666 15251 16261 16262 991
```

```
Вывод преобразованного массива W
191 121 15251 666 100 15 14 12 16261 16262 991
```

## 5.7 Задачи для самостоятельного решения

### 5.7.1 Основные операции при работе с массивами

Разработать программу на языке C++ для решения следующей задачи.

1. Задан массив целых чисел $X(n)$. Найти
   - сумму четных элементов массива;
   - наибольшее из отрицательных чисел массива.

   Из данного массива и некоторого массива того же типа, но другой размерности $Y(m)$, сформировать общий массив $Z(n + m)$. Выполнить сортировку полученного массива по возрастанию модулей. Удалить из массива число с номером $k$.
2. Задан массив вещественных чисел $A(n)$. Найти
   - произведение положительных элементов массива;
   - сумму отрицательных чисел, расположенных после максимального элемента массива.

   Из данного массива и некоторого массива того же типа, но другой размерности $B(m)$, сформировать общий массив $C(n + m)$. Преобразовать полученный массив так, чтобы все его положительные элементы стали отрицательными и наоборот. Удалить предпоследний элемент массива.
3. Задан массив вещественных чисел $A(n)$. Найти
   - произведение ненулевых элементов массива.
   - сумму четных чисел, расположенных до минимального элемента массива.

   Из заданного массива $A(n)$ все положительные числа переписать в массив $B$, а отрицательные в массив $C$. Удалить из массива $A(n)$ первый нулевой элемент.
4. Задан массив целых чисел $X(n)$. Найти
   - сумму положительных четных элементов массива;
   - количество элементов массива, расположенных после первого нулевого элемента.

   Из данного массива и некоторого массива того же типа, но другой размерности $Y(m)$, сформировать общий массив $Z(n + m)$. Удалить из полученного массива наибольший элемент.
5. Задан массив вещественных чисел $X(n)$. Найти

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

- сумму элементов с нечетными номерами;
- произведение элементов массива, расположенных между первым и последним отрицательными элементами.

Из данного массива и некоторого массива того же типа, но другой размерности $Y(m)$, сформировать общий массив $Z(n+m)$. Удалить из полученного массива наименьший элемент.

6. Задан массив вещественных чисел $X(n)$. Найти
   - сумму положительных элементов массива;
   - произведение элементов с нечетными индексами, расположенных во второй половине массива.

   Из данного массива и некоторого массива того же типа, но другой размерности $Y(m)$, сформировать общий массив $Z(n+m)$ таким образом, чтобы в нем сначала располагались все отрицательные элементы, затем элементы равные нулю, и в заключении все положительные. Удалить из массива $Z(n+m)$ максимальный элемент.
7. Задан массив целых чисел $B(n)$. Найти
   - произведение отрицательных элементов с четными индексами.
   - максимальный элемент среди элементов, которые кратны 3.

   Из данного массива и некоторого массива того же типа, но другой размерности $C(m)$, сформировать массив $A$, состоящий только из неотрицательных значений заданных массивов. Удалить из массива $A$ первое число кратное 17.
8. Задан массив целых чисел $X(n)$. Найти
   - сумму чисел, расположенных в первой половине массива;
   - разностью между значениями максимального и минимального элементов массива.

   Из данного массива сформировать новый массив $Y$, в который записать все ненулевые элементы массива $X(n)$. Удалить из массива $X(n)$ последнее четное число.
9. Задан массив целых чисел $X(n)$. Найти
   - произведение элементов массива, кратных трем;
   - сумму чисел, которые расположены между минимальным и максимальными элементами массива.

   Из данного массива сформировать новый массив $Y(n)$, в который переписать все элементы массива $X(n)$ в обратном порядке. Удалить из массива $Y(n)$ минимальный и максимальный элементы.
10. Задан массив целых чисел $X(n)$ Найти

- сумму нечетных положительных элементов массива;
- количество чисел, которые расположены до первого нулевого элемента в массиве.

Записать элементы заданного массива в обратном порядке. Определить положение максимального элемента до и после преобразования. Удалить максимальный элемент.

11. Задан массив целых чисел $X(n)$ Найти
    - сумму четных элементов;
    - количество чисел, которые расположены после минимального элемента массива.

    Заменить нулевые элементы заданного массива значениями их номеров. Определить среднее арифметическое элементов массива до и после преобразования. Удалить минимальный элемент массива $X(n)$
12. Задан массив вещественных чисел $X(n)$ Найти
    - процент отрицательных чисел в массиве;
    - сумму первого и последнего положительных элементов.

    Записать элементы заданного массива в обратном порядке. Определить положение минимального элемента до и после преобразования. Удалить минимальный элемент
13. Задан массив целых чисел $A(n)$ Найти
    - среднее арифметическое элементов массива;
    - минимальный элемент и его индекс в первой половине массива.

    Из данного массива и некоторого массива того же типа, но другой размерности $B(m)$ сформировать общий массив $C$, в который переписать удвоенные положительные значения элементов исходных массивов. Удалить из массива $C$ последний четный элемент.
14. Задан массив целых чисел $A(n)$ Найти
    - сумму элементов массива, кратных 13;
    - количество четных чисел, расположенных до максимального элемента массива.

    Сформировать массив $C$, в который переписать квадраты отрицательных элементов исходного массива $A(n)$ Удалить из массива три последних чётных элемента.
15. Задан массив целых чисел $P(n)$ Найти
    - количество нечетных элементов массива;
    - произведение чисел, расположенных до минимума.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Первую половину массива $P(n)$ переписать в массив $R$, а вторую в массив $Q$. Найти сумму квадратов разностей элементов массивов $R$ и $Q$. Удалить из массива $P(n)$ последнее число кратное 5.

16. Задан массив целых чисел $X(n)$ Найти
    - сумму четных элементов во второй половине массива;
    - количество чисел расположенных между первым и последним отрицательными элементами массива.

    Из заданного массива $X(n)$ все положительные числа переписать в массив $Y$, а отрицательные в массив $Z$. Поменять местами максимальный и минимальный элементы в массиве $X(n)$ Удалить третий элемент массива $X(n)$
17. Задан массив целых чисел $X(n)$ Найти
    - количество четных элементов в массиве;
    - среднее геометрическое положительных элементов массива, расположенных в его первой половине.

    Все отрицательные элементы заданного массива заменить значением его максимального элемента. Удалить из массива первый нулевой элемент.
18. Задан массив целых чисел $P(n)$ Найти
    - сумму модулей элементов массива;
    - номер первого нулевого элемента.

    Из данного массива и некоторого массива того же типа, но другой размерности $R(m)$ сформировать общий массив $Q$, в который переписать положительные значения элементов исходных массивов. Удалить из массива $Q$ наибольший четный элемент.
19. Задан массив целых чисел $X(n)$ Найти
    - произведение чисел, кратных 7;
    - количество чисел, которые расположены между первым и последним четными числами.

    Из данного массива сформировать новый массив $Y$, в который переписать первые $k$ положительных элементов массива $X(n)$ Удалить из массива $X(n)$ число наименее отличающееся от среднего арифметического значения элементов массива.
20. Задан массив целых чисел $A(n)$ Найти
    - произведение ненулевых элементов массива;
    - среднее арифметическое элементов массива, расположенных в его первой половине.

Из данного массива и некоторого массива того же типа и размерности $B(n)$ сформировать массив $C(n)$ каждый элемент которого равен квадрату суммы соответствующих элементов массивов $A(n)$ и $B(n)$ Удалить из массива $C(n)$ наибольший и наименьший элементы.

21. Задан массив вещественных чисел $X(n)$ Найти
    - произведение абсолютных значений элементов массива;
    - количество нечетных элементов массива, расположенных в его второй половине.

    Из данного массива и некоторого массива того же типа и размерности $Y(n)$ сформировать массив $Z(n)$ каждый элемент которого равен квадрату разности соответствующих элементов массивов $X(n)$ и $Y(n)$ Удалить из массива $Z(n)$ минимальный элемент и поменять местами первый и последний элементы.
22. Задан массив целых чисел $A(n)$ Найти
    - сумму элементов массива, кратных трем;
    - произведение ненулевых элементов массива с четными индексами.

    Сформировать массива $B$, в который записать последние $k$ элементов массива $A(n)$ Удалить из массива $A(n)$ максимальный нечетный элемент.
23. Задан массив вещественных чисел $P(n)$ Найти
    - количество положительных элементов массива;
    - номера первого положительного и последнего отрицательного элементов массива.

    В массиве $P(n)$ поменять местами первые и последние пять элементов. Удалить из массива $P(n)$ элемент наименее отличающийся от среднего арифметического.
24. Задан массив целых чисел $B(n)$ Найти
    - среднее геометрическое элементов с четными индексами, кратных трем.
    - минимальный элемент среди положительных четных элементов.

    Из данного массива и некоторого массива того же типа, но другой размерности $C(m)$ сформировать массив $A$, состоящий только из положительных значений заданных массивов. Удалить из массива $B(n)$ первый чётный и последний нечётный элементы.
25. Задан массив вещественных чисел $X(n)$ Найти
    - номер минимального по модулю элемента массива;
    - среднее арифметическое первых $k$ положительных элементов.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

Из данного массива и некоторого массива того же типа, но другой размерности $Y(m)$ сформировать общий массив $Z$ таким образом, чтобы сначала располагались все отрицательные элементы, а затем все положительные. Удалить из массива наибольшее и наименьшее простое число.

### 5.7.2 Применение функций для обработки массивов.

Разработать программу на языке C++ для решения следующей задачи.

1. Задан массив целых чисел $X(n)$ Все *простые числа* переписать в массив $Y$. Из массива $Y$ удалить 5 наибольших элементов массива. Вывести на экран содержимое массива $Y$ в *двоичной системе*.
2. Заданы массивы целых чисел $X(n)$ и $Y(k)$ Все *совершенные числа* из этих массивов переписать в массив $Z$. В массиве $Z$ найти четыре наименьших элемента массива. Удалить из массива $Z$ все нулевые элементы Результаты вывести на экран в *восьмеричной системе*.
3. Заданы массивы целых чисел $X(n)$ и $Y(k)$ Два наибольших элемента из массива $X$ и пять последних *простых чисел* из массива $Y$ переписать в массив $Z$. Проверить содержит ли массив $Z$ числа, в которых есть цифра «7».
4. Заданы массивы целых чисел $X(n)$ и $Y(k)$ Три наименьших *простых числа* из массива $Y$ и числа из массива $X$, в которых есть цифры «1» и «9» переписать в массив $Z$. Из массива $Z$ удалить все нечетные числа.
5. Задан массив целых чисел $X(n)$ Шесть наибольших чисел этого массива переписать в массив $Z$. Удалить из массива $Z$ все четные числа. Вывести на экран элементы массива $Z$в *восьмеричной системе счисления*.
6. Заданы массивы целых чисел $X(n)$ и $Y(k)$ Числа из массива $X$, в которых нет «нулей» и *составные числа* из массива $Y$, переписать в массив $Z$. Найти в массиве $Z$ пять наибольших нечетных чисел. Выполнить сортировку массивов $X$, $Y$ и $Z$ в порядке возрастания их элементов.
7. Заданы массивы целых положительных чисел. $X(n)$ — в двоичной системе счисления, а $Y(k)$ — в восьмеричной. Все числа из массивов $X$ и $Y$ переписать в массив десятичных чисел $Z$. В

массиве $Z$ найти пять наибольших *простых числа*. Удалить из массива $Z$ все *составные числа*.

8. Задан массив целых положительных чисел $X(n)$ Все *простые числа* длиной не более пяти цифр переписать в массив $Y$. Удалить из массива два наибольших и три наименьших числа.
9. Задан массив целых положительных чисел в пятеричной системе $X(n)$ Из массива $X$ сформировать массив десятеричных чисел $Z$. Найти сумму трех наименьших и четырех наибольших чисел массива $Z$.
10. Заданы массивы целых положительных чисел $X(n)$ $Y(k)$ $Z(m)$ Сформировать массив $U$ из таких элементов массивов $X$, $Y$, $Z$, которые в восьмеричной системе образуют возрастающую последовательность цифр. Найти пять наибольших чисел в массива $U$.
11. Задан массив целых положительных чисел $X(n)$ Все числа в которых нет цифр «1», «2» и «3» переписать в массив $Y$. Найти сумму двух наибольших и трех наименьших *простых чисел* в массиве $Y$.
12. Заданы массивы целых положительных чисел $X(n)$ $Y(k)$ $Z(m)$ Сформировать массив $U$ из таких элементов массивов $X$, $Y$, $Z$, которые состоят из одинаковых цифр. Удалить из массива $U$ наибольшее и наименьшее число. Выполнить сортировку массивов $X(n)$ $Y(k)$ $Z(m)$ в порядке возрастания их элементов.
13. Задан массив целых положительных чисел $X(n)$ Все числа, в которых нет цифры ноль, а их длина не менее трех цифр переписать в массив $Z$. Поменять местами наибольшее *составное число* и наименьшее *простое число* в массиве $Z$.
14. Задан массив целых чисел $X(n)$ Все положительные числа, состоящие из одинаковых цифр, переписать в массив $Z$. Удалить из массива $Z$ числа, с четной суммой цифр.
15. Заданы массивы целых чисел $X(n)$ и $Y(k)$ Все числа, с нечетной суммой цифр, переписать в массив $Z$. Найти три наибольших *простых числа* в массиве $Z$.
16. Заданы массивы целых чисел $X(n)$ и $Y(k)$ Три наибольших числа из массива $X$ и числа из массива $Y$, в которых нет чётных цифр переписать в массив $Z$. Элементы массива $Z$ вывести на экран в восьмеричной и десятичной системах счисления.
17. Задан массив целых чисел $X(n)$ Семь наименьших *простых чисел* переписать в массив $Z$. Удалить из массива числа с четной суммой цифр.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

18. Заданы массивы целых чисел $X(n)$ и $Y(k)$ Положительные числа из массива $X$ и пять наибольших чисел из массива $Y$, переписать в массив $Z$. Найти сумму четырехзначных чисел массива $Z$.
19. Заданы массивы целых положительных чисел: $X(n)$ — в пятеричной, а $Y(k)$ в шестеричной системах счисления. Все числа из массивов переписать в массив десятичных чисел $Z$. В массиве $Z$ найти пять наибольших чисел с нечетной суммой цифр.
20. Заданы массив целых положительных чисел $X(n)$ $Z(m)$ Все простые числа из массивов $X$ и $Z$, в которых есть цифры «1», «2» или «3» переписать в массив $Y$. Найти произведение двух наибольших и три наименьших *простых чисел* массива $Y$.
21. Задан массив целых положительных чисел в двоичной системе $X(n)$ Из массива $X$ сформировать массив десятеричных чисел $Z$. Из массива $Z$ удалить четыре наименьших и три наибольших числа.
22. Заданы массивы целых положительных чисел $X(n)$ $Y(k)$ $Z(m)$ Сформировать массив $U$ из элементов массивов $X$, $Y$, $Z$, которые образуют убывающую последовательность цифр. Найти сумму семи наименьших чисел массива $U$.
23. Задан массив целых положительных чисел $X(n)$ Переписать в массив $Y$ все числа-палиндромы, остальные числа переписать в массив $Z$. Удалить из массива $Z$ все числа которые есть нули и сумма цифр нечётна.
24. Заданы массивы целых положительных чисел $X(n)$ $Y(k)$ $Z(m)$ Числа, которые не состоят из одинаковых цифр, переписать в массив $U$. Удалить из массива $U$ числа с четной суммой цифр.
25. Задан массив целых положительных чисел $X(n)$ Все числа с четной суммой цифр переписать в массив $Z$. Элементы массива $Z$ упорядочить в порядке убывания суммы цифр.

### 5.7.3 Работа с группами элементов в массиве

Разработать программу на языке C++ для решения следующей задачи.

1. В массиве вещественных чисел найти предпоследнюю группу, которая состоит только из отрицательных элементов.
2. В массиве вещественных чисел найти первую и последнюю группы знакочередующихся элементов.

3. В массиве целых чисел найти вторую и третью группу, состоящую из нечетных цифр.
4. В массиве целых чисел найти предпоследнюю группу, состоящую из возрастающей последовательности цифр.
5. Из массива целых чисел удалить предпоследнюю группу, состоящую из возрастающей последовательности цифр.
6. Из массива целых чисел удалить последнюю группу, состоящую из убывающей последовательности нечетных цифр.
7. Из массива целых чисел удалить группу наибольшей длины, которая состоит из возрастающей последовательности нечетных цифр.
8. В массиве целых чисел найти группу наименьшей длины, которая состоит из убывающей последовательности четных цифр.
9. Из массива целых чисел удалить две группы наибольшей длины, состоящие из простых чисел, в которых нет четных цифр.
10. Задан массив целых чисел. Вывести на экран первую и последнюю группы, состоящие из простых чисел.
11. Из массива целых чисел удалить три группы наименьшей длины, состоящие из простых чисел, в представлении которых нет цифры семь.
12. Из массива целых чисел удалить группу наибольшей длины, которая состоит из возрастающей последовательности простых чисел.
13. Из массива целых чисел удалить все группы, которые состоят из убывающей последовательности четных чисел.
14. В массиве вещественных чисел найти группу максимальной длины, которая состоит из знакочередующихся чисел.
15. В массиве вещественных чисел найти группу минимальной длины, которая состоит из убывающей последовательности чисел.
16. Из массива вещественных чисел удалить все группы, состоящие из невозрастающей последовательности чисел.
17. Из массива вещественных чисел удалить три группы наибольшей длины, состоящие из возрастающей последовательности чисел.
18. В массиве целых чисел найти две последних группы, состоящие из простых чисел, причем цифры каждого числа образуют возрастающую последовательность.
19. Из целочисленного массива удалить группу простых чисел минимальной длины, цифры которых образуют убывающей последовательность.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

20. Из целочисленного массива удалить группу минимальной длины, состоящую из элементов, представляющих собой возрастающую последовательность четных цифр.
21. В массиве целых чисел найти группы наименьшей и наибольшей длины, которые состоят из простых чисел.
22. В массиве целых чисел найти группу наибольшей длины, которая состоит из неубывающей последовательности нечетных чисел.
23. Из массива целых чисел удалить две группы наименьшей длины, состоящие из составных чисел, в записи которых нет цифр «0» и «2».
24. Задан массив целых чисел. Вывести на экран первую и последнюю группы, состоящие из простых чисел с нечетной суммой цифр в каждом.
25. Из массива целых чисел удалить три группы наибольшей длины, которые состоят из отрицательных чисел с четной суммой цифр в каждом.

### 5.7.4 Сортировка элементов массива

Разработать программу на языке C++ для решения следующей задачи.

1. Упорядочить по убыванию элементы целочисленного массива, расположенные между двумя наибольшими чётными значениями.
2. Упорядочить в порядке возрастания модулей элементы массива, расположенные между наибольшим и наименьшим значениями.
3. Упорядочить в порядке убывания модулей элементы, расположенные между первым и последним отрицательным значениями массива.
4. Упорядочить в порядке убывания элементы, расположенные между вторым положительным и предпоследним отрицательным значениями массива.
5. Упорядочить по возрастанию элементы целочисленного массива, расположенные между первым числом палиндромом и последним отрицательным значением.
6. Упорядочить в порядке возрастания суммы цифр элементы целочисленного массива, расположенные между последним числом-палиндромом и первым простым числом.

7. Упорядочить по возрастанию модулей элементы целочисленного массива, расположенные между третьим и пятым простыми числами.
8. Упорядочить по убыванию элементы целочисленного массива, расположенные после минимального числа-палиндрома.
9. Удалить из целочисленного массива простые числа. В полученном массиве упорядочить по возрастанию модулей элементы, расположенные после наибольшего числа.
10. Удалить из целочисленного массива числа-палиндромы. В полученном массиве упорядочить по возрастанию модулей элементы, расположенные до наименьшего простого числа.
11. Удалить из целочисленного массива все составные числа. Упорядочить элементы массива в порядке возрастания суммы цифр чисел.
12. Удалить из целочисленного массива все числа, состоящие из одинаковых цифр. Упорядочить элементы массива в порядке убывания суммы их цифр.
13. Задан массив целых положительных чисел. Сформировать новый массив, куда записать элементы исходного массива, состоящие из одинаковых цифр. Упорядочить элементы полученного массива в порядке возрастания суммы цифр чисел.
14. Упорядочить по возрастанию модулей элементы, расположенные между двумя наименьшими значениями массива.
15. Упорядочить в порядке возрастания элементы, расположенные между четвёртым и девятым отрицательным числами массива.
16. Упорядочить в порядке возрастания модулей элементы, расположенные между наибольшим и предпоследним положительным значениями массива.
17. Упорядочить в порядке убывания модулей элементы, расположенные между пятым положительным и первым отрицательным значениями массива.
18. Упорядочить в порядке убывания модулей элементы целочисленного массива, расположенные между наибольшим и наименьшим числами-палиндромами.
19. Упорядочить в порядке убывания суммы цифр элементы целочисленного массива, расположенные между последним и предпоследним числами-палиндромами.
20. Упорядочить по возрастанию модулей элементы массива, расположенные между двумя наименьшими положительными числами.

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

21. Упорядочить по возрастанию элементы целочисленного массива, расположенные между двумя наибольшими числами-палиндромами.
22. Удалить из целочисленного массива числа-палиндромы. В полученном массиве упорядочить по возрастанию модулей элементы, расположенные до наименьшего значения.
23. Удалить из целочисленного массива отрицательные числа. В полученном массиве упорядочить по убыванию элементы, расположенные между двумя наибольшими простыми числами.
24. Удалить из целочисленного массива простые числа. Упорядочить элементы массива в порядке убывания суммы цифр чисел.
25. Задан массив целых положительных чисел. Сформировать новый массив, куда записать элементы исходного массива, состоящие из нечетных цифр. Упорядочить элементы полученного массива в порядке убывания суммы цифр чисел.

# Список литературы


[1] Алексеев Е.Р. *Программирование на Microsoft Visual C++ и Turbo C++ Explorer.* — М.: НТ Пресс, 2007. - 352с.

[2] Алексеев Е.Р., Чеснокова О.В, Кучер Т.В. *Самоучитель по программированию на Free Pascal и Lazarus.* Донецк, Унитех, 2009. - 502с.

[3] Б.П. Демидович, И.А. Марон. *Основы вычислительной математики.* — М. Наука, 1966. - 664с.

[4] Керниган Б., Ритчи Д. *Язык программирования Си.* — М., Вильямс, 2013. - 304с.

[5] Лаптев В.В. *C++. Объектно-ориентированное программирование.* — СПб., Питер, 2008. – 464с.

[6] Лаптев В.В. Морозов А.В., Бокова. C++. Объектно-ориентированное программирование. Задачи и упражнения. — СПб., Питер, 2007. – 288с.

[7] Подбельский В.В. *Язык C++.* — М., Финансы и статистика, 2001. – 560с.

[8] Шиманович Е.Л. *C/C++ в примерах и задачах.* — Мн., Новое знание, 2004. - 528с.

# Предметный указатель

© 2014 Алексеев Е. Р., Злобин Г. Г., Костюк Д. А., Чеснокова О. В., Чмыхало А. С.

*Научное издание*

Серия «Библиотека ALT Linux»

Алексеев Евгений Ростиславович, Григорий Григорьевич Злобин,
Дмитрий Александрович Костюк, Чеснокова Оксана Витальевна,
Чмыхало Александр Сергеевич

**Программирование на языке C++ в среде Qt Creator**

Оформление обложки: А. С. Осмоловская
Вёрстка: В. Л. Черный
Редактура: В. Л. Черный

Подписано в печать 26.08.14. Формат 60x90/16.
Гарнитура Computer Modern. Печать офсетная. Бумага офсетная.
Усл. печ. л. 23,0. . Тираж 999 экз. Заказ

ООО «Альт Линукс»
Адрес для переписки: 119334, Москва, 5-й Донской проезд, д. 15,
стр. 6
Телефон: (495) 662-38-83. E-mail: sales@altlinux.ru
`http://altlinux.ru`