

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

A47

Программирование на языке C++ в среде Qt Creator:  
A47 / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк, О. В. Чеснокова,  
А. С. Чмыхало — М. : ALT Linux, 2014. — 432 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 библиотека 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++>

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](mailto: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

<b>Предисловие</b>	7
<b>Глава 1. Знакомство с языком C++</b>	8
1.1    Первая программа на C++ . . . . .	8
1.2    Среда программирования Qt Creator . . . . .	11
<b>Глава 2. Общие сведения о языке C++</b>	17
2.1    Алфавит языка . . . . .	17
2.2    Данные . . . . .	18
2.3    Константы . . . . .	21
2.4    Структурированные типы данных . . . . .	21
2.5    Указатели . . . . .	22
2.6    Операции и выражения . . . . .	23
2.7    Стандартные функции . . . . .	31
2.8    Структура программы . . . . .	33
2.9    Ввод и вывод данных . . . . .	35
2.10   Задачи для самостоятельного решения . . . . .	40
<b>Глава 3. Операторы управления</b>	44
3.1    Основные конструкции алгоритма . . . . .	44
3.2    Составной оператор . . . . .	46
3.3    Условные операторы . . . . .	46
3.4    Операторы цикла . . . . .	64
3.5    Решение задач с использованием циклов . . . . .	70
3.6    Задачи для самостоятельного решения . . . . .	86
<b>Глава 4. Использование функций при программировании на C++</b>	100
4.1    Общие сведения о функциях . . . . .	100
4.2    Передача параметров в функцию . . . . .	104
4.3    Возврат результата с помощью оператора return . . . . .	106
4.4    Решение задач с использованием функций . . . . .	106
4.5    Рекурсивные функции . . . . .	121
4.6    Перегрузка функций . . . . .	123
4.7    Шаблоны функций . . . . .	125
4.8    Область видимости переменных в функциях . . . . .	126
4.9    Функция main(). Параметры командной строки . . . . .	127

4.10 Задачи для самостоятельного решения . . . . .	129
<b>Глава 5. Массивы</b> . . . . .	
5.1 Статические массивы в C(C++) . . . . .	134
5.2 Динамические массивы в C(C++) . . . . .	136
5.3 Отличие статического и динамического массива . . . . .	139
5.4 Основные алгоритмы обработки массивов . . . . .	139
5.5 Указатели на функции . . . . .	165
5.6 Совместное использование динамических массивов . . . . .	168
5.7 Задачи для самостоятельного решения . . . . .	174
<b>Глава 6. Статические и динамические матрицы</b> . . . . .	
6.1 Статические матрицы C(C++) . . . . .	184
6.2 Динамические матрицы . . . . .	185
6.3 Обработка матриц в C(C++) . . . . .	186
6.4 Решение некоторых задач линейной алгебры . . . . .	196
6.5 Задачи для самостоятельного решения . . . . .	214
<b>Глава 7. Организация ввода-вывода в C++</b> . . . . .	
7.1 Форматированный ввод-вывод в C++ . . . . .	225
7.2 Работа с текстовыми файлами в C++ . . . . .	229
7.3 Обработка двоичных файлов . . . . .	235
7.4 Функции fscanf() и fprintf() . . . . .	239
<b>Глава 8. Строки в языке C++</b> . . . . .	
8.1 Общие сведения о строках в C++ . . . . .	241
8.2 Операции над строками . . . . .	242
8.3 Тип данных string . . . . .	245
8.4 Задачи для самостоятельного решения . . . . .	247
<b>Глава 9. Структуры в языке C++</b> . . . . .	
9.1 Общие сведения о структурах . . . . .	248
9.2 Библиотеки для работы с комплексными числами . . . . .	255
9.3 Задачи для самостоятельного решения . . . . .	263
<b>Глава 10. Возникновение объектного подхода в программировании</b> . . . . .	
10.1 Классы и объекты в C++ . . . . .	266
10.2 Создание и удаление объектов . . . . .	270
10.3 Наследование . . . . .	284
10.4 Обработка исключений . . . . .	292
10.5 Шаблоны классов . . . . .	304
10.6 Элементы стандартной библиотеки C++ . . . . .	315
10.7 Задачи для самостоятельного решения . . . . .	320
	325

<b>Глава 11. Знакомство с Qt. Подготовка к работе</b>	328
11.1    Знакомство с Qt. Обзор истории . . . . .	328
11.2    Лицензирование Qt . . . . .	331
11.3    Справка и ресурсы . . . . .	332
11.4    Обзор настроек среды Qt Creator . . . . .	333
11.5    Задачи для самостоятельного решения . . . . .	338
<b>Глава 12. Структура проекта. Основные типы</b>	339
12.1    Файлы проекта . . . . .	339
12.2    Компиляция проекта . . . . .	341
12.3    Консольный проект Qt. Вывод сообщений. . . . .	344
12.4    Работа с текстовыми строками в Qt . . . . .	346
12.5    Контейнерные классы в Qt . . . . .	347
12.6    Работа с файлами . . . . .	350
12.7    Задачи для самостоятельного решения . . . . .	353
<b>Глава 13. Создание графического интерфейса средствами Qt</b>	354
13.1    Виджеты (Widgets) . . . . .	354
13.2    Компоновка (Layouts) . . . . .	358
13.3    Политики размера (Size Policies) . . . . .	362
13.4    Сигнально-слотовые соединения . . . . .	364
13.5    Создание сигналов (signals) и слотов (slots) . . . . .	366
13.6    Элементы графического интерфейса. . . . .	370
13.7    Задачи для самостоятельного решения . . . . .	372
<b>Глава 14. Создание элементов графического интерфейса</b>	373
14.1    Класс QObject . . . . .	373
14.2    Управление памятью. Иерархии объектов . . . . .	378
14.3    События (Events). Обработка событий (Event handling) . . . . .	381
14.4    Фильтры событий (Event filters). Пропагирование (Propagation) .	383
14.5    Создание собственного элемента интерфейса . . . . .	387
14.6    Рисование элементов. Класс QPainter . . . . .	390
14.7    Задачи для самостоятельного решения . . . . .	392
<b>Глава 15. Разработка приложений с графическим интерфейсом</b>	393
15.1    Окна. Класс QMainWindow . . . . .	393
15.2    Быстрая разработка с помощью Qt Designer . . . . .	394
15.3    Программирование формы созданной в Qt Designer . . . . .	399
15.4    Стандартные диалоги . . . . .	402
15.5    Ресурсы программы . . . . .	406
15.6    Создание собственных диалогов . . . . .	408
15.7    Сохранение настроек . . . . .	412
15.8    Использование сторонних разработок в собственном проекте . .	413
15.9    Задачи для самостоятельного решения . . . . .	415

<b>Приложение А. Использование компилятора командной строки</b>	417
<b>Сведения об авторах</b>	426
<b>Список литературы</b>	427
<b>Предметный указатель</b>	428

# 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.

В приложениях описан текстовый редактор Geany, а также кроссплатформенная библиотека MathGL предназначенная 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](http://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 С(С++), познакомится с основными этапами перевода программы с языка С++ в машинный код. Второй параграф главы посвящён знакомству со средой Qt Creator.

### 1.1 Первая программа на C++

Знакомство с языком С++ 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 С++ 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 С++ 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<sup>1</sup> 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 переменной `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.

**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<sup>2</sup>. Таким образом строка

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

выводит на экран текст "Периметр прямоугольника равен "<sup>3</sup>, 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. 4.9 Сейчас следует запомнить, если программа начинается со строки `int main()`, последним оператором должен быть `return 0`<sup>4</sup>.

<sup>1</sup>В литературе равнозначно используются термины «имя переменной» и «идентификатор».

<sup>2</sup>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.

<sup>3</sup>С пробелом после слова «равен».

<sup>4</sup>Вообще говоря, вместо нуля может быть любое целое число.

**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.

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 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
}

...
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 программы 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 (формируется 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 **компиля-**

ции 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 с программой работает препроцессор<sup>5</sup>, он обрабатывает директивы, в нашем случае это директивы 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<sup>6</sup>. 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>). Установочный файл имеет расширение .run. Для установки приложения, необходимо запустить его на выполнение. Установка проходит в графическом режиме.

---

<sup>5</sup>Препроцессор — преобразовывает текст директив в форму, понятную компилятору. О данных на выходе препроцессора говорят, что они находятся в препроцессированной форме.

<sup>6</sup>Тексты программы, 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).

После запуска программы пользователь увидит на экране окно, подобное представленному на рис. 1.1<sup>7</sup>.

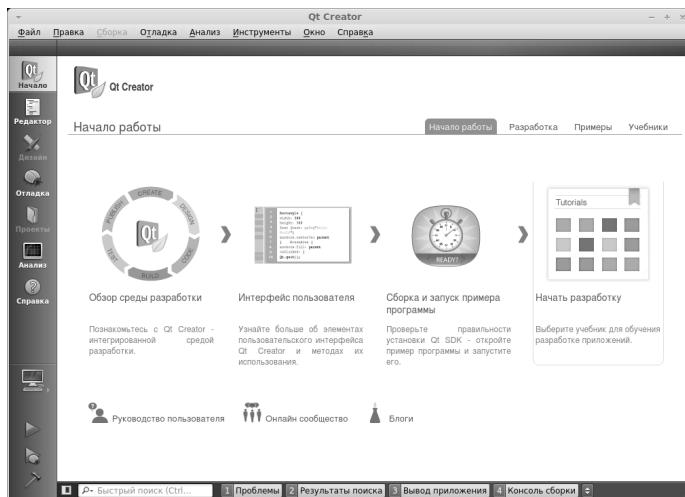


Рис. 1.1: Окно Qt Creator

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

<sup>7</sup>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.

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 этого 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++**)<sup>8</sup>.

Далее выбираем имя проекта и каталог для его размещения (см. рис. 1.3)<sup>9</sup>. Следующие два этапа создания нашего первого приложения оставляем без изменения<sup>9</sup>. После чего окно IDE **Qt Creator** будет подобно представленному на рис. 1.4. Заменим текст «Hello, Word» стандартного приложения, на текст программы решения задачи 1.1.

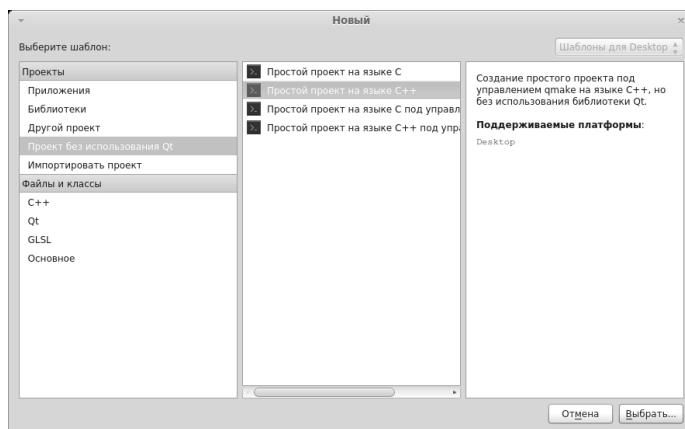


Рис. 1.2: Окно выбора типа приложения в **Qt Creator**

Для сохранения текста программы можно воспользоваться командой **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 **Ctrl+R**.

<sup>8</sup>Рекомендуем для каждого проекта выбирать отдельный каталог. Проект — это несколько взаимосвязанных между собой файлов и каталогов.

<sup>9</sup>О назначении этих этапов будет рассказано в дальнейших разделах книги.

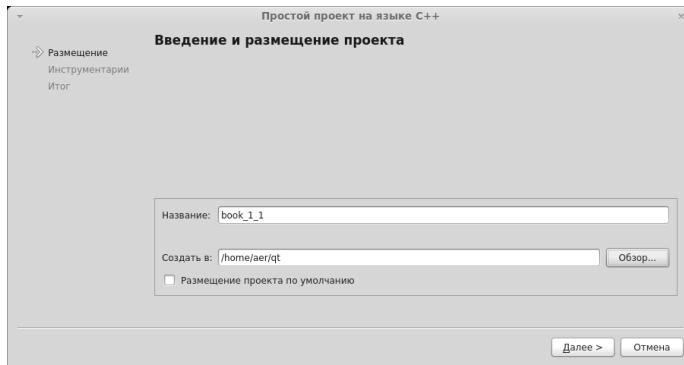


Рис. 1.3: Выбор имени и каталога нового проекта

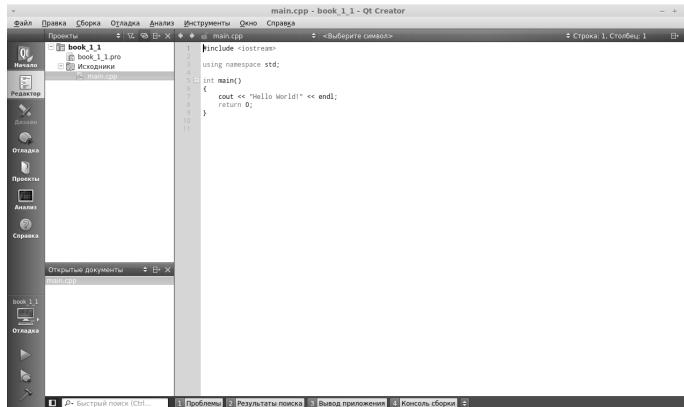


Рис. 1.4: Главное окно создания консольного приложения

### 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 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.

```
a=2
b=7
Периметр прямоугольника равен 18
Площадь прямоугольника равна 14
Для закрытия данного окна нажмите <ВВОД>...
```

Рис. 1.5: Результаты работы программы решения задачи 1.1

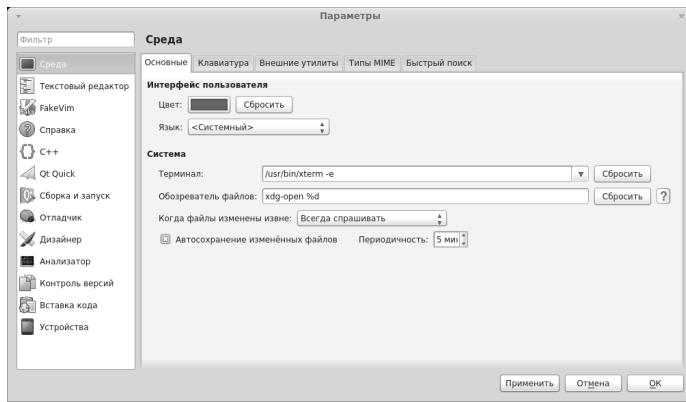


Рис. 1.6: Окно настроек среды Qt Creator

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.

Для решения задачи можно воспользоваться формулой Герона

$$S = \sqrt{\frac{p}{2} \left( \frac{p}{2} - a \right) \left( \frac{p}{2} - b \right) \left( \frac{p}{2} - c \right)}, \text{ где } p = a + b + c \text{ — периметр.}$$

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.
3. Вывод  $p$  и  $s$  на экран дисплея.

Ниже приведен текст программы. Сразу заметим, что в тексте могут встречаться строки, начинающиеся с двух наклонных (//). Это комментарии. Комментарии не

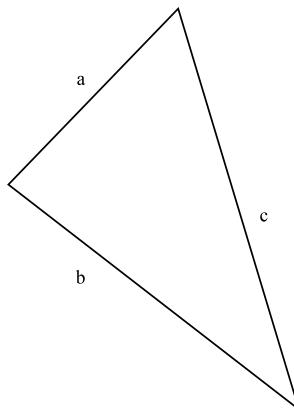


Рис. 1.7: Треугольник

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 С(С++). 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 может
   выглядеть 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<sup>1</sup>. Это связано с тем, что обычно под величину символьного типа отводят столько па-

---

<sup>1</sup>В кодировке utf каждый символ кириллицы занимает 2 байта.

мяти, 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.

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
<code>int</code>	-2147483647 ... 2147483647	4 байта
<code>unsigned int</code>	0 ... 4294967295	4 байта
<code>signed int</code>	-2147483647 ... 2147483647	4 байта
<code>short int</code>	-32767 ... 32767	2 байта
<code>long int</code>	-2147483647 ... 2147483647	4 байта
<code>unsigned short int</code>	0 ... 65535	2 байта
<code>signed short int</code>	-32767 ... 32767	2 байта
<code>long long int</code>	$(2^{63}-1) \dots (2^{63}-1)$	8 байт
<code>signed long int</code>	-2147483647 ... 2147483647	4 байта
<code>unsigned long int</code>	0 ... 4294967295	4 байта
<code>unsigned long long int</code>	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 порядка. 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 мантиссе. Поскольку старшая цифра мантиссы всегда равна 1, она не хранится.

Величины типа `double` занимают 8 байт, в них под порядок и мантиссе отводится 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а длины 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
<code>float</code>	3.4E-38 ... 3.4E+38	4 байта
<code>double</code>	1.7E-308 ... 1.7E+308	8 байт
<code>long double</code>	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:

```
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 явного указания типа, например,

```
const double pi=3.141592653589793
```

Кроме того, константа может быть определена с помощью директивы<sup>2</sup> `#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<sup>3</sup>. 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:

---

<sup>2</sup>Структура программы и директивы описаны в п. 2.8

<sup>3</sup>Подробно работа с одномерными и двумерными массивами описана в главах 5 и 6.

**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<sup>4</sup>. В C++ строки описываются как массив элементов типа **char**. Например:

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

Структура<sup>5</sup> 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**;

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,

<sup>4</sup>Работа со строками описана в главе 8

<sup>5</sup>Работа со структурами описана в главе 9.

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 С это переменная типа char, а Р — указатель на С, значит в Р находится адрес по которому в памяти компьютера хранится значение переменной С.

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 вещественные числа x и z, сами вещественные значения *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 унарные (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
<b>Унарные операции</b>	
<code>++</code>	увеличение значения на единицу
<code>--</code>	уменьшение значения на единицу
<code>~</code>	поразрядное отрицание
<code>!</code>	логическое отрицание
<code>-</code>	арифметическое отрицание (унарный минус)
<code>+</code>	унарный плюс
<code>&amp;</code>	взятие адреса
<code>*</code>	разадресация
<code>(type)</code>	преобразование типа
<b>Бинарные операции</b>	
<code>+</code>	сложение
<code>-</code>	вычитание
<code>*</code>	умножение
<code>/</code>	деление
<code>%</code>	остаток от деления

Таблица 2.4 — продолжение

Операция	Описание
<code>&lt;&lt;</code>	сдвиг влево
<code>&gt;&gt;</code>	сдвиг вправо
<code>&lt;</code>	меньше
<code>&gt;</code>	больше
<code>&lt;=</code>	меньше или равно
<code>&gt;=</code>	больше или равно
<code>==</code>	равно
<code>!=</code>	не равно
<code>&amp;</code>	поразрядная конъюнкция (И)
<code>^</code>	поразрядное исключающее ИЛИ
<code> </code>	поразрядная дизъюнкция (ИЛИ)
<code>&amp;&amp;</code>	логическое И
<code>  </code>	логическое ИЛИ
<code>=</code>	присваивание
<code>*=</code>	умножение с присваиванием
<code>/=</code>	деление с присваиванием
<code>+=</code>	сложение с присваиванием
<code>-=</code>	вычитание с присваиванием
<code>%=</code>	остаток от деления с присваиванием
<code>&lt;&lt;=</code>	сдвиг влево с присваиванием
<code>&gt;&gt;=</code>	сдвиг вправо с присваиванием
<code>&amp;=</code>	поразрядная конъюнкция с присваиванием
<code> =</code>	поразрядная дизъюнкция с присваиванием
<code>^=</code>	поразрядное исключающее ИЛИ с присваиванием
<b>Другие операции</b>	
<code>?</code>	условная операция
<code>,</code>	последовательное вычисление
<code>sizeof</code>	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 переменной c увеличивается на единицу.
a=a*3;   //Значение переменной 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
```

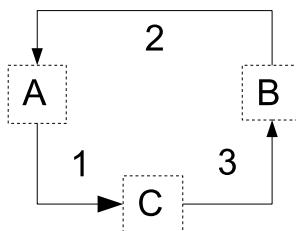


Рис. 2.1: Использование буферной переменной

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; //Уменьшение 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.
```

## 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++
$2 \cdot a + b \cdot (c + d)$	$2*a+b*(c+d)$
$3 \cdot \frac{a + b}{c + d}$	$3*(a+b)/(c+d)$
$\frac{3 \cdot a - 2 \cdot b}{c \cdot d}$	$(3*a-2*b)/(c*d)$ или $(3*a-2*b)/c/d$
$c + \frac{(b - a)^2 - a^2 + 1}{d - 2}$	$(b-a)*(b-a)/(c+1/(d-2)) - (a*a+1)/(b*b+c*d)$

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 `r=r+1;` можно представить в префиксной форме `++r;` и в постфиксной `r++;`. 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  $B=00000000000011000$ . В результате логического умножения  $A$  and  $B$  получим  $0000000000001000$  или 8 в десятичной системе счисления (рис. 2.2). Таким образом,  $A\&B=14\&24=8$ .

$A=$	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	0	0	0	1	1	0	14
0	0	0	0	0	0	0	0	0	0	0	1	1	0			
и																
$B=$	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	0	0	1	1	0	0	24
0	0	0	0	0	0	0	0	0	0	1	1	0	0			

$A=$	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	14
0	0	0	0	0	0	0	0	0	0	0	1	1	1	0			
или																	
$B=$	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	24
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0			

$$A \text{ and } B = \boxed{0000000000001000} \quad 8$$

$$A \text{ or } B = \boxed{0000000000001110} \quad 30$$

Рис. 2.2: Пример логического умножения

Рис. 2.3: Пример логического сложения

*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:

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

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

$A=$	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	0	0	0	1	1	0	1	13
0	0	0	0	0	0	0	0	0	0	1	1	0	1			
not $A=$	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	1	1	1	1	1	1	1	1	1	1	0	0	1	0	not 13=-14
1	1	1	1	1	1	1	1	1	1	0	0	1	0			

Рис. 2.4: Пример арифметического отрицания

*Сдвиг влево ( $M<<L$ ).* Число  $M$ , представленное в двоичной системе, сдвигается влево на  $L$  позиций. Рассмотрим операцию  $15 \text{ shl } 3$ . 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,  $15 \text{ shl } 3 = 120$  (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»

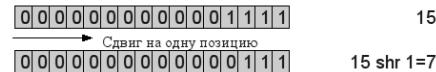


Рис. 2.6: Пример операции «Сдвиг вправо»

*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.

### 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 ≠ 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.

### 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 *р	р	*р
тип р	&р	р

*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`.

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* 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`)<sup>6</sup>. 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 operandами<sup>7</sup>. В таблице 2.8 приведены некоторые из них.

Таблица 2.8: Стандартные математические функции

Обозначение	Действие
<code>abs(x)</code>	Модуль целого числа $x$

<sup>6</sup>Константу нулевого указателя можно присвоить любому указателю и такой указатель при сравнении не будет равен любому реальному указателю.

<sup>7</sup>Работа с математическими функциями возможна только при подключении директивы `math.h` (п. 2.6)

Таблица 2.8 — продолжение

Обозначение	Действие
<code>fabs(x)</code>	Модуль вещественного числа $x$
<code>sin(x)</code>	Синус числа $x$
<code>cos(x)</code>	Косинус числа $x$
<code>tan(x)</code>	Тангенс числа $x$
<code>atan(x)</code>	Арктангенс числа $x$ , $x \in (-\frac{i\pi}{2}; \frac{i\pi}{2})$
<code>acos(x)</code>	Арккосинус числа $x$
<code>asin(x)</code>	Арксинус числа $x$
<code>exp(x)</code>	Экспонента, $e^x$
<code>log(x)</code>	Натуральный логарифм, ( $x > 0$ )
<code>log10(x)</code>	Десятичный логарифм, ( $x > 0$ )
<code>sqrt(x)</code>	Корень квадратный, ( $x > 0$ )
<code>pow(x,y)</code>	Возведение числа $x$ в степень $y$
<code>ceil(x)</code>	Округление числа $x$ до ближайшего большего целого
<code>floor(x)</code>	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++
$\sqrt[3]{(a+b)^2}$	<code>pow((a+b)*(a+b),1./3)</code> или <code>pow(pow(a+b,2),1./3)</code>
$\cos^4(x)$	<code>pow(cos(x), 4)</code>
$e^{2x}$	<code>exp(2*x)</code>
$e^{5 \sin(\frac{x}{2})}$	<code>exp(5*sin(x/2))</code>
$\sin^2(\sqrt{x})$	<code>pow(sin(sqrt(x)),2)</code>
$\ln( x - 2 )$	<code>log(fabs(x-2))</code>
$\log_b a$	<code>log(a)/log(b)</code>
$\frac{\lg(x^2 + 1)}{\lg(4)}$	<code>log10(x*x+1)/log10(4)</code>
$\sin(x^2 + y^2) + \cos \frac{(x^2+y^2)}{2 \cdot y} + \sqrt{x^2 + y^2}$	<code>z=x*x+y*y; sin(z)+cos(z/(2*y))+sqrt(z);</code>

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  $z = -\text{pow}(\text{fabs}(x), y)$ . 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:

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

Здесь, **типа\_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 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
типа_результата main(параметры)
{
    описание переменных главной функции;
    операторы главной функции;
}

типа_результата имя1(параметры1)
{
    описание переменных функции имя1;
    операторы1;
```

```

}

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

.....
тиp_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 4.

## 2.9 Ввод и вывод данных

Ввод-вывод данных в языке C++ осуществляется либо с помощью функций ввода-вывода в стиле С, 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 С 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);
```

выполняет *ввод переменных*, адреса которых указаны 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

Таблица 2.10 — продолжение

Параметр	Назначение
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
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 [-]ddd.dddd.
e	Значение со знаком вида [-]d.dddde[+ -]ddd.
E	Значение со знаком вида [-]d.ddddE[+ -]ddd.
g	Значение со знаком типа е или f в зависимости от значения и точности.
G	Значение со знаком типа е 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 С, должна быть директива #include <stdio.h>. Заголовочный файл stdio.h содержит описание функций ввода-вывода.

Рассмотрим работу функций на примере следующей задачи.

**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:

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

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 а=.
    //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 б=.
    scanf("%f",&b); //Запись в переменную b значения введенного с клавиатуры.
    printf("c="); //Вывод на экран символов с=.
    scanf("%f",&c); //Запись в переменную c 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;
}
```

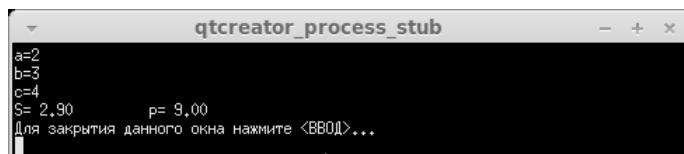


Рис. 2.7: Результаты работы программы к задаче 2.2 (вариант 1)

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

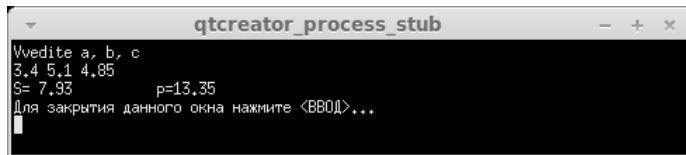


Рис. 2.8: Результаты работы программы к задаче 2.2 (вариант 2)

### 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*<sup>8</sup> `cin` для ввода с клавиатуры и `cout` для 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  $P$ ):

<sup>8</sup>Поток — виртуальный канал связи, создаваемый в программе для передачи данных

$45 \% 10 = 5.$

Последнюю цифру числа Р (количество сотен) можно найти так:

$456 / 100 = 4.$

Так как в задаче требовалось записать цифры числа Р 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<sup>9</sup>

```
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`.

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

---

<sup>9</sup>Обратите внимание, что при использовании кодировки utf-16 один кириллический символ занимает в памяти 2 байта.

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. Данна 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.
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 секундах.
15. С начала суток прошло  $N$  секунд. Найти количество полных часов, прошедших с начала суток и остаток в секундах.
16. Дано двузначное число  $\alpha \leqslant 88$ . Вывести на экран число, которое получится если каждую цифру числа  $a$  увеличить на единицу.
17. Дано двузначное число  $\alpha \geqslant 22$ . 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 остаток в днях.
23. С начала года прошло  $K$  часов. Найти количество полных дней, прошедших с начала года и остаток в часах.
24. Дано двузначное число  $a \leqslant 44$ . Вывести на экран число, которое получится если удвоить каждую цифру числа  $a$ .
25. Дано двузначное число  $\alpha \geqslant 22$ . 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 $f(x)$	№	Выражение $f(x)$
1	$\sqrt[7]{x^2 + 2.7 \cdot \pi \cdot \cos \sqrt{ x^3 } - 2 + e^x}$	2	$\operatorname{tg}^4 x + \sin^2 \frac{\pi}{x} - e^{2x^2+3.6x-1}$
3	$ x^4 - \cos x  - \sqrt[9]{1 + \sqrt{x^6}} + \sin^3 \frac{\pi}{e^x + 1}$	4	$\log_4  e^x - 4  - \sqrt[7]{\frac{2 \cdot x}{3.21 + \cos^2 \frac{\pi}{7}}}$
5	$\sqrt[3]{\sqrt{ x } +  \operatorname{ctg}^2 x + \frac{e^x}{2 \cdot \pi} - x^3 }$	6	$x^5 + \log_3^2(3x^2 + 5) + \sqrt[9]{(\pi - 6x^2)^2}$
7	$\frac{1 - \log  x - \cos(2x - \pi) }{6 + x^{4x-1}} + \sqrt[5]{x^3}$	8	$e^{x+\frac{\pi}{3}} + \sqrt[3]{\operatorname{tg} \frac{x^5}{x^2 + 13.22} + \cos^3 x}$
9	$x^{1+\frac{3+\pi}{4}} - 3x^3 - \sqrt[5]{(x+1)^4 + \lg \frac{x}{x+1}}$	10	$\sqrt[5]{x^3 + \cos \sqrt{ x^3 } + \frac{e^x}{\cos(3 \cdot x + \frac{\pi}{15})}}$
11	$e^{2x} + \sqrt[5]{\operatorname{ctg} \frac{(x-\pi)^9}{x^4 + 3.4} + \sin^2 6.2x}$	12	$\sqrt[5]{(x + \operatorname{tg} a)^2} - \frac{1 - \ln  e^x + \cos \frac{\pi}{8} }{2}$
13	$\log(e^x + 27) - \sqrt{x^3 + \frac{\sqrt[5]{x^7} + 14}{\sin 5x + 5.1 \cdot \pi}}$	14	$\ln  \cos(x - 2 \cdot \pi)  - \sqrt[3]{1 + \frac{e^x}{\sin x - 3}}$
15	$\sqrt{x^3 + \frac{\sqrt[3]{x^4} - 1}{\sin x + \pi + e^x}}$	16	$\sqrt[3]{\frac{1 + 3 \cdot \pi}{1 + x^2}} +  \operatorname{arctg}^2 x^3 $
17	$\operatorname{tg}^2  x  + 3^{2x^2-e^x} + \frac{\sqrt[7]{x^2}}{\cos^2 \pi x}$	18	$x^4 - \sqrt[5]{\pi - \sqrt{ x^3 } + \sin^2 \frac{x}{x^2 + 1}}$
19	$\log(e^x + 6) - \sqrt[3]{(x-4)^2 + 1.47 \sin \sqrt{ \pi \cdot x }}$	20	$\frac{x^5}{\sin  x-7 } + \log^2(x^2 + 2.5) - \sqrt[3]{(\pi - 6.1x^2)^2}$
21	$\operatorname{ctg}^2 \frac{x \cdot \pi}{3} - \sqrt{ x } - 3.4^{x^2-10} + \ln(x^2 + 3)$	22	$\log_5  x^3 - e^x  - \sqrt[3]{\frac{2x}{\cos(x + 1.23 \cdot \pi)}}$
23	$ \cos \frac{\pi}{7} - e^x  - \sqrt[7]{2 + \sqrt{x^5}} + \ln \frac{x^4 + 1}{6}$	24	$\log(x^2 + 2) - \sin^2 x + \sqrt[5]{2 - \sqrt{ x } + \sin \frac{\pi}{e^x + 1}}$
25	$\log_2 e^x - \cos \frac{x}{\pi} + \sqrt[3]{\frac{ \operatorname{tg}(2x) }{2.6 + x^2 + x^3}}$		

# 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<sup>1</sup>. 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»);
- 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.

---

<sup>1</sup>От *algorithmi*, *algorismus*, первоначально латинская транслитерация имени математика аль-Хорезми.



Рис. 3.1: Блок начала-конца алгоритма

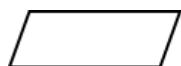


Рис. 3.2: Блок ввода-вывода данных



Рис. 3.3: Арифметический блок

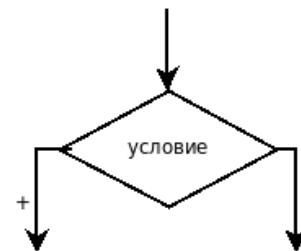


Рис. 3.4: Условный блок

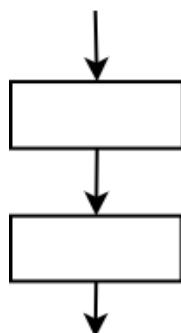


Рис. 3.5: Линейный процесс

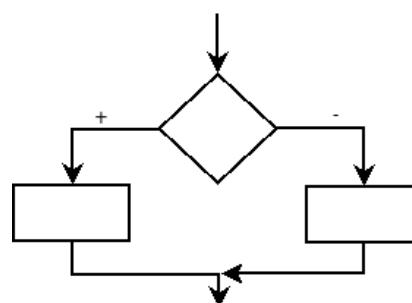


Рис. 3.6: Разветвляющийся процесс

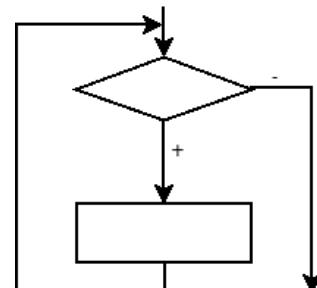


Рис. 3.7: Циклический процесс

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 С(С++) является понятие составного оператора.

## 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 условия 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.

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;.**

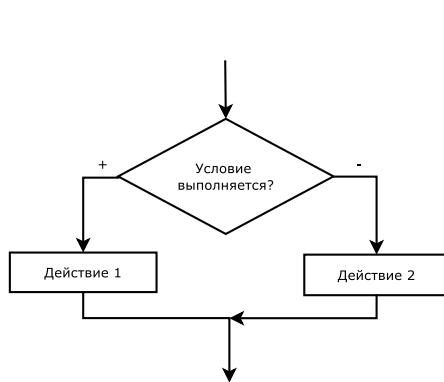


Рис. 3.8: Алгоритм условного оператора `if ... else`



Рис. 3.9: Алгоритм условного оператора `if`

**Внимание!** Если в задаче требуется, чтобы в зависимости от значения условия выполнялся не один оператор, а несколько, их необходимо 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 «усеченному» 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 с изменяется только при условии, что а не равно б
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_A; else оператор_B;
оператор_B относится к условию_2, а в конструкции
if (условие_1) { if (условие_2) оператор_A; }
else оператор_B;

```

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  $y = f(x)$ .

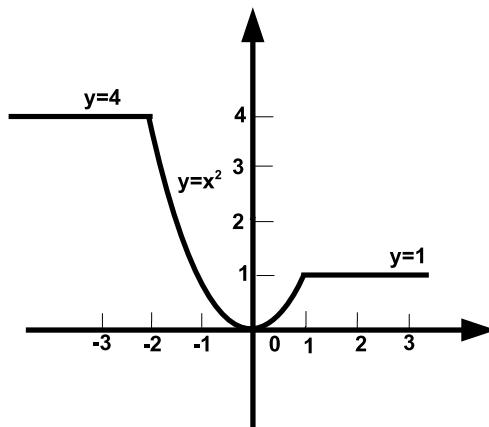


Рис. 3.10: Графическое представление задачи 3.1

Аналитически функцию представленную на рис. 3.10 можно записать так:

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

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.

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.

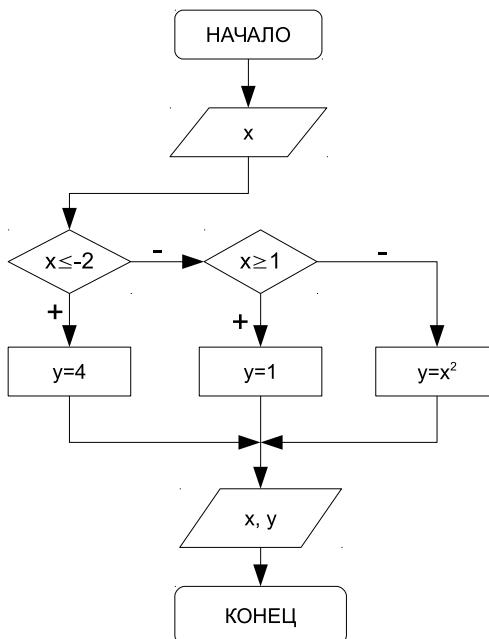


Рис. 3.11: Блок-схема алгоритма решения задачи 3.1

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).

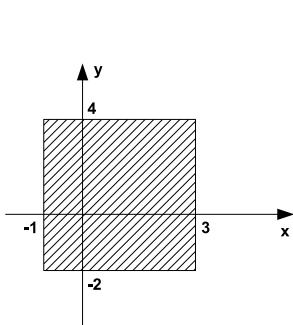


Рис. 3.12: Графическое представление задачи 3.2

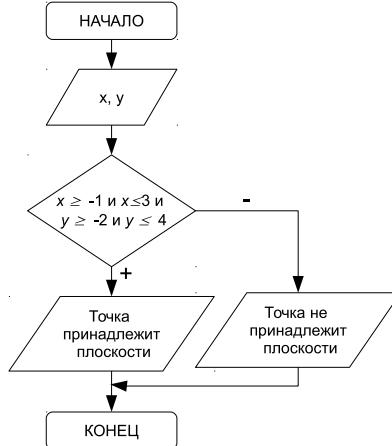


Рис. 3.13: Алгоритм решения задачи 3.2

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;
}
```

**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:

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

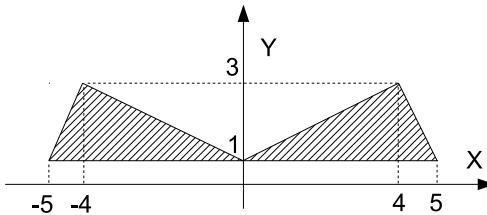


Рис. 3.14: Графическое представление задачи 3.3

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:

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

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:

$$\begin{cases} y \leq 1 + \frac{1}{2} \cdot x \\ y \leq -2 \cdot x + 11 \\ y \geq 1 \end{cases} \quad \text{или} \quad \begin{cases} y \leq 1 - \frac{1}{2} \cdot x \\ y \leq 2 \cdot x + 11 \\ y \geq 1 \end{cases}$$

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 не 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  $x_1$  и  $x_2$  — 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  $c$ .
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  $x_1 = \frac{-b + \sqrt{d}}{2a}$  и  $x_2 = \frac{-b - \sqrt{d}}{2a}$ .
7. Вывод значений  $x_1$  и  $x_2$  на экран
8. Конец алгоритма.

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

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.
```

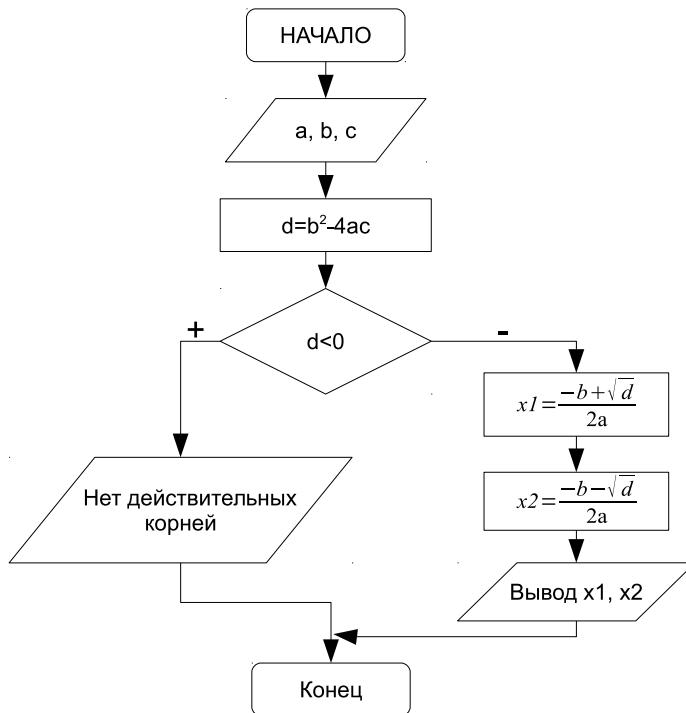


Рис. 3.15: Алгоритм решения квадратного уравнения

```

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  $x_1$  и  $x_2$  — действительные корни квадратного уравнения либо  $x_1$  и  $x_2$  — 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  $d \geq 0$ , то вычисление действительных корней:  $x_1 = \frac{-b + \sqrt{d}}{2a}$  и  $x_2 = \frac{-b - \sqrt{d}}{2a}$  и вывод их на экран. При отрицательном дискриминанте выводится сообщение о том, что действительных корней нет, и вычисляются комплексные корни<sup>2</sup>  $x_1 = \frac{-b}{2a} + i \frac{\sqrt{|d|}}{2a}$ ,  $x_2 = \frac{-b}{2a} - i \frac{\sqrt{|d|}}{2a}$ .

У обоих комплексных корней действительные части одинаковые, а мнимые отличаются знаком. Поэтому можно в переменной  $x_1$  хранить действительную часть числа  $\frac{-b}{2a}$ , в переменной  $x_2$  — модуль мнимой части  $\frac{\sqrt{|d|}}{2a}$ , а в качестве корней вывести  $x_1 + i \cdot x_2$  и  $x_1 - i \cdot x_2$ .

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  $x_1$ , модуль мнимой — в переменную  $x_2$ ), а вывод — в блоке 5 (первый корень  $x_1 + i \cdot x_2$ , второй —  $x_1 - i \cdot x_2$ ). 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 ax2 + 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)
        {
            cout<<x1<<"+"<<x2<<"i\n";
            cout<<x1<<" - "<<x2<<"i\n";
        }
        else
        {
            cout<<x1<<" - "<<fabs(x2)<<"i\n";
        }
    }
}
```

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

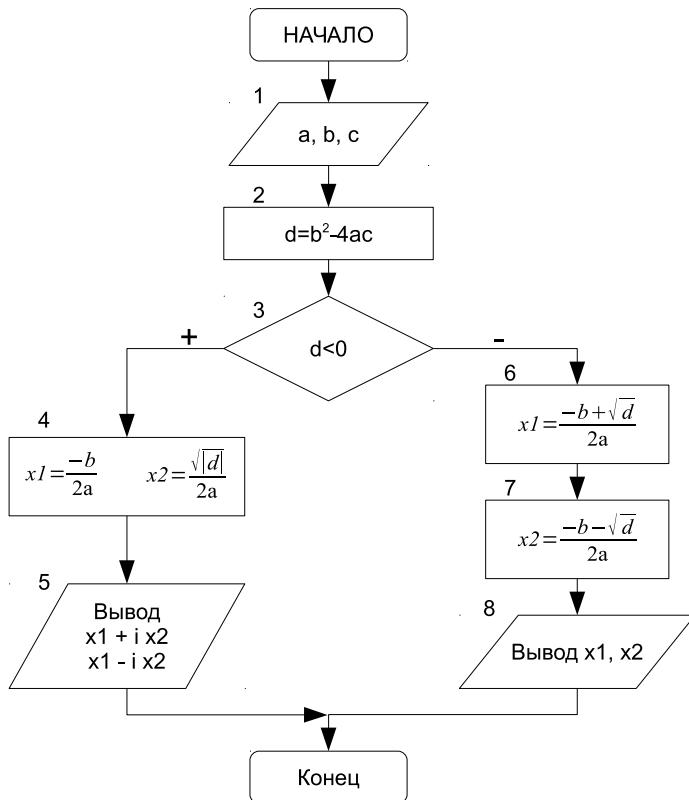


Рис. 3.16: Алгоритм решения задачи 3.5

```

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

$$ax^3 + bx^2 + cx + d = 0 \quad (3.1)$$

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

$$x^3 + rx^2 + sx + t = 0, \quad (3.2)$$

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:

$$y^3 + py + q = 0, \quad (3.3)$$

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
$D \geq 0$	1	2
$D < 0$	3	—

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано:

$$\begin{aligned} y_1 &= u + v \\ y_2 &= \frac{-u+v}{2} + \frac{u-v}{2}i\sqrt{3} \\ y_3 &= \frac{-u+v}{2} - \frac{u-v}{2}i\sqrt{3}, \end{aligned} \quad (3.4)$$

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 ве-

личины. Чтобы избавиться от этого, можно воспользоваться формулами:

$$\begin{aligned}y_1 &= 2\sqrt[3]{\rho} \cos\left(\frac{\phi}{3}\right), \\y_2 &= 2\sqrt[3]{\rho} \cos\left(\frac{\phi}{3} + \frac{2\pi}{3}\right), \\y_3 &= 2\sqrt[3]{\rho} \cos\left(\frac{\phi}{3} + \frac{4\pi}{3}\right),\end{aligned}\quad (3.5)$$

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

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

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

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 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<sup>3</sup>.

```
#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*g/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
```

<sup>3</sup>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

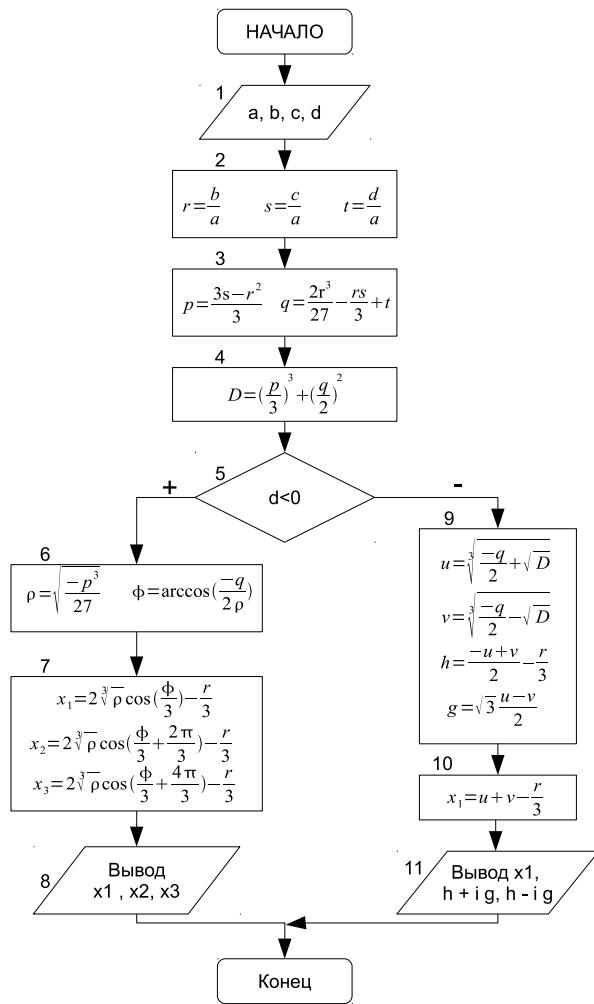


Рис. 3.17: Алгоритм решения кубического уравнения

```

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="<

```

```

//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);
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:*  $x_1$ ,  $x_2$ ,  $x_3$ ,  $x_4$ .

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  $y_1$  и  $y_2$  (блок 5).
4. Если  $y_1 < 0$  и  $y_2 < 0$  (блок 6), то вывод сообщения, что корней нет (блок 7).
5. Если  $y_1 \geq 0$  и  $y_2 \geq 0$  (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  $y_1$ . Если  $y_1 \geq 0$  (блок 11), то вычисляются два корня по формуле  $\pm\sqrt{y_1}$  (блок 12), иначе (если  $y_2 \geq 0$ ) вычисляются два корня по формуле  $\pm\sqrt{y_2}$  (блок 13). Вывод вычисленных значений корней (блок 14).

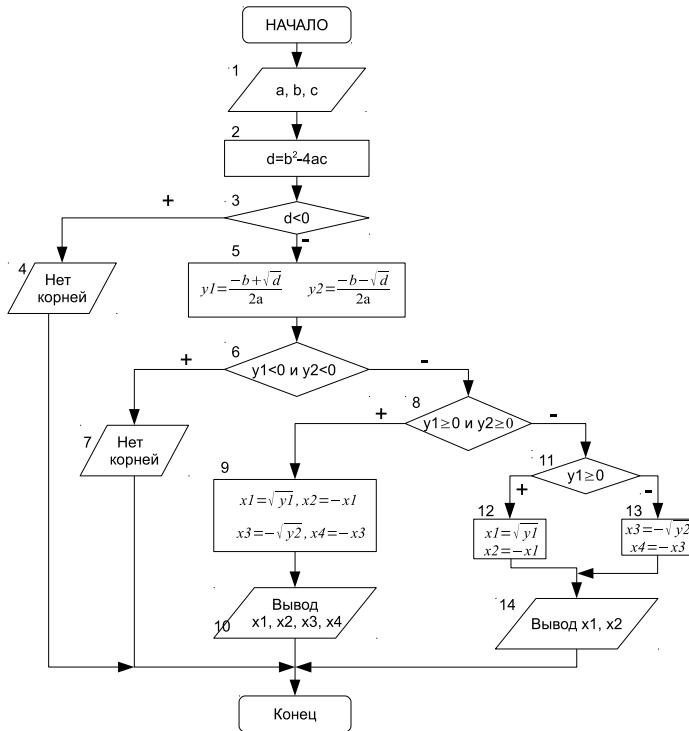


Рис. 3.18: Алгоритм решения биквадратного уравнения

Текст программы решения биквадратного уравнения приведен ниже.

**Внимание!** Если в условном операторе проверяется двойное условие необходимо применять логические операции `||`, `&&`, `!`. Например, условие «если  $y_1$  и  $y_2$  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<<"\n";
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 при-

нимает 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)
{
    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 на 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 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:

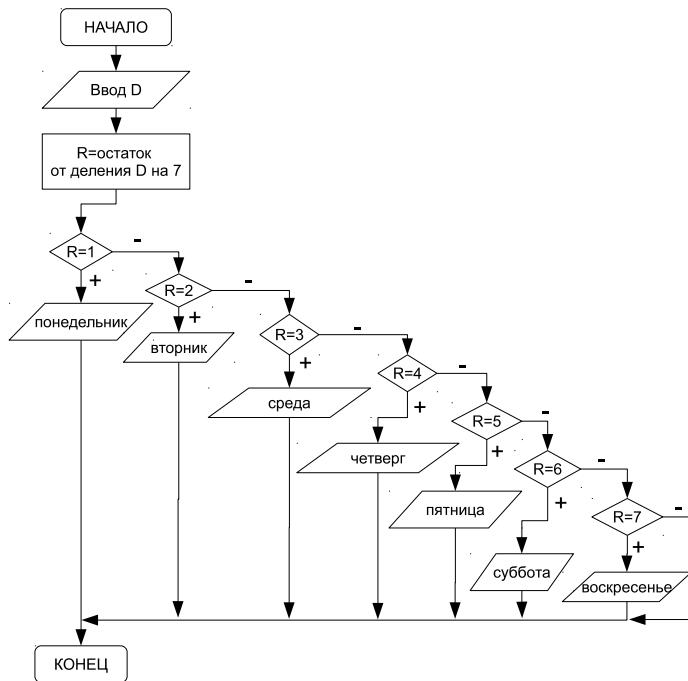


Рис. 3.19: Алгоритм решения задачи 3.8

```

#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;
    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;

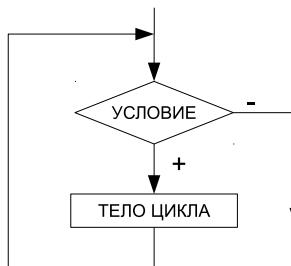


Рис. 3.20: Алгоритм циклической структуры с предусловием

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= 2.10	y=-1.1969
x= 1.10	y=1.1059	x= 2.20	y=-1.3209
x= 1.20	y=0.9203	x= 2.30	y=-1.4045
x= 1.30	y=0.7011	x= 2.40	y=-1.4489
x= 1.40	y=0.4553	x= 2.50	y=-1.4576
x= 1.50	y=0.1918	x= 2.60	y=-1.4348
x= 1.60	y=-0.0793	x= 2.70	y=-1.3862
x= 1.70	y=-0.3473	x= 2.80	y=-1.3172
x= 1.80	y=-0.6017	x= 2.90	y=-1.2334
x= 1.90	y=-0.8328	x= 3.00	y=-1.1400
x= 2.00	y=-1.0331	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.

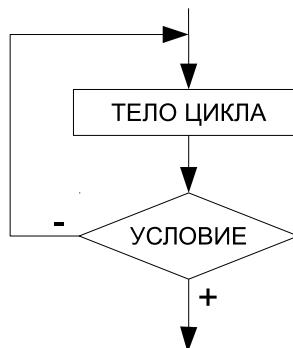


Рис. 3.21: Алгоритм циклической структуры с постусловием

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);

```

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. 66.

### 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<sup>4</sup>, 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.

---

<sup>4</sup>Запятая в C++ это операция последовательного выполнения операторов

**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).



Рис. 3.22: Алгоритм работы цикла с параметром

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:

```
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:

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

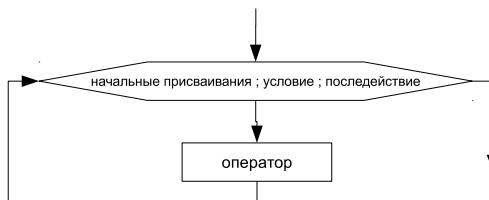


Рис. 3.23: Блок-схема цикла с параметром

```

//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. 66.

#### 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;`<sup>5</sup>.

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.

<sup>5</sup>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

### 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.

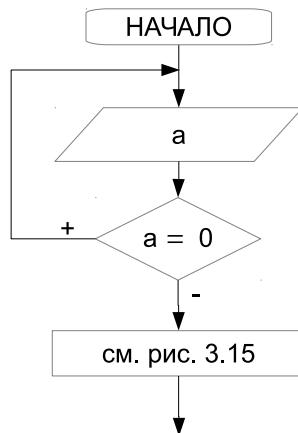


Рис. 3.24: Блок-схема проверки ввода данных

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 a 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()
{
    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()
{

```

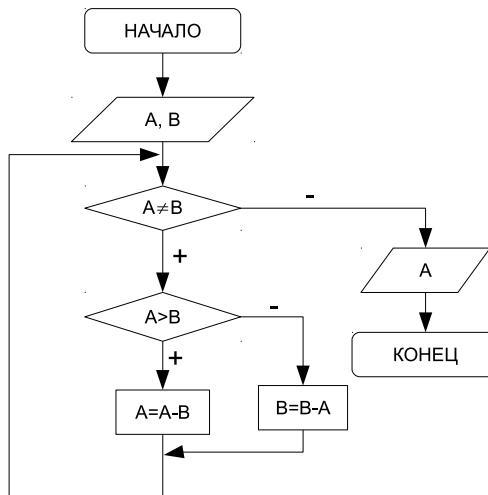


Рис. 3.25: Поиск наибольшего общего делителя двух чисел.

```

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  $N$  ( $N! = 1 \cdot 2 \cdot 3 \cdots N$ ).

*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**, предназначеннной 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 переменной **factorial**, которая была вычислена в теле цикла.

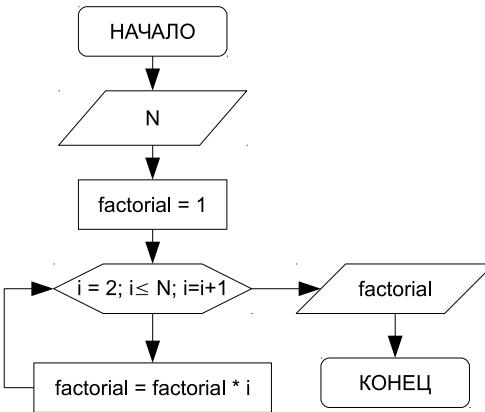


Рис. 3.26: Алгоритм вычисления факториала.

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;
    unsigned int N, i;
    for (cout<<"N=" , cin>>N, factorial=1, i=2; i<=N; factorial*=i , i++)
        cout<<"факториал=" <<factorial <<"\n";
    return 0;
}
  
```

**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, предназначеннной для накапливания суммы, необходимо присвоить значение нуль. Блок-схема решения этой задачи представлена на рис. 3.27.

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

```

//Решение задачи с помощью цикла while
#include <iostream>
  
```

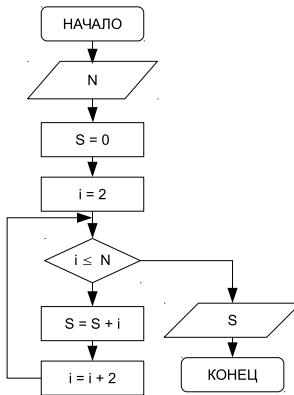


Рис. 3.27: Алгоритм вычисления суммы четных, натуральных чисел.

```

using namespace std;
int main()
{
    unsigned int N, i, S;
    cout<<"N="; cin>>N;
    S=0;
    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.

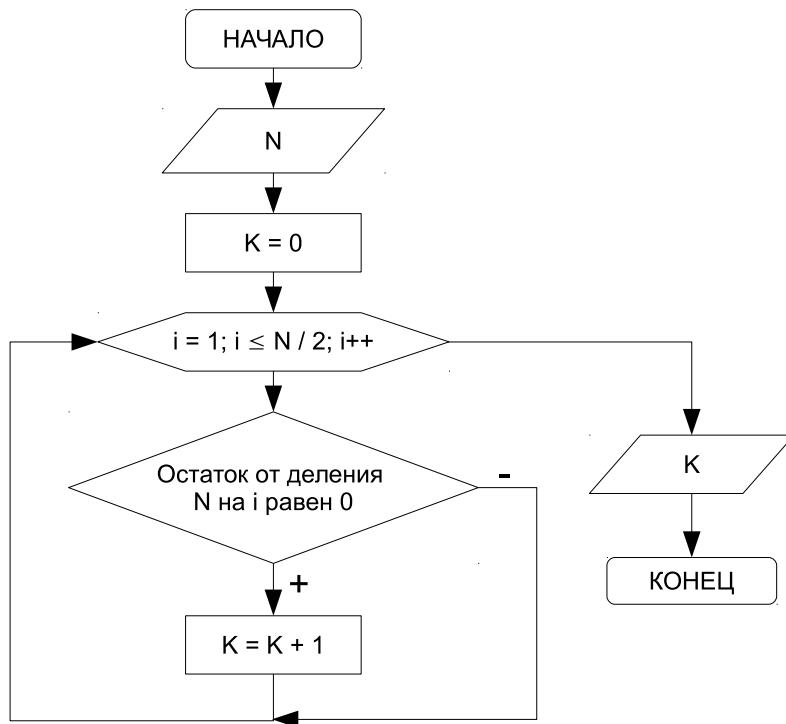


Рис. 3.28: Алгоритм определения делителей натурального числа.

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++;
    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, иначе 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.

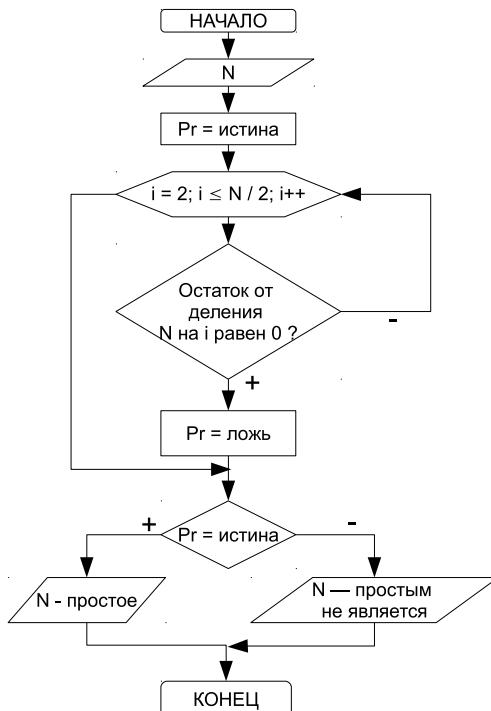


Рис. 3.29: Алгоритм определения простого числа.

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

```

<sup>6</sup>При решении задачи (см. алгоритм на рис. 3.30) исходное число изменяется, поэтому, чтобы его, не потерять, копируем исходное число  $N$  в переменную  $M$ , и делить будем уже  $M$ .

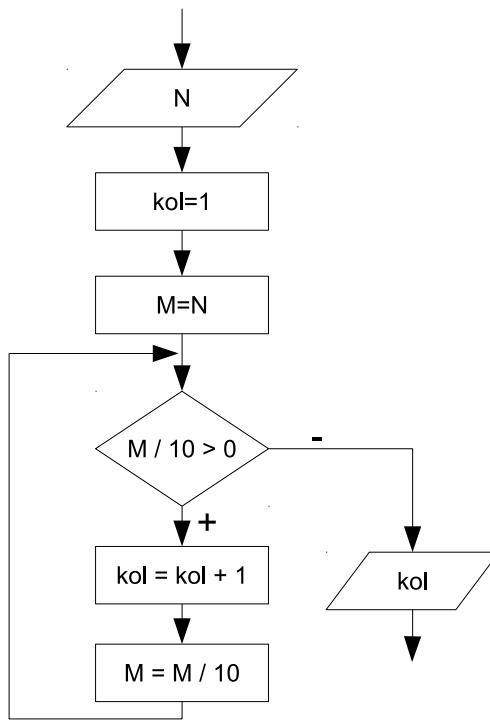


Рис. 3.30: Алгоритм определения количества цифр в числе.

**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<sup>7</sup>. Таким образом, на первом этапе решения задачи необходимо определить  $\text{kol}$  — 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.

<sup>7</sup>Алгоритм нахождения количества цифр в числе был рассмотрен в предыдуще задаче.

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.

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<sup>8</sup>. 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;
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

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

результате в ячейке *Max* сохраниться элемент последовательности с наибольшим значением<sup>9</sup>.

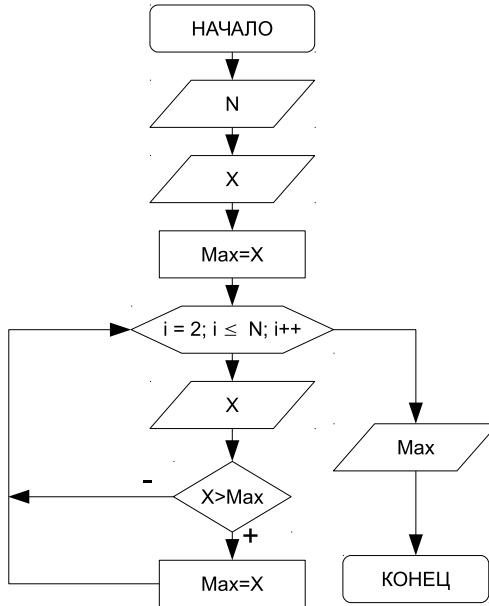


Рис. 3.31: Алгоритм поиска наибольшего числа в последовательности.

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.
    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;
}
  
```

<sup>9</sup> 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.

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

Блок-схема решения задачи приведена на рис. 3.32.

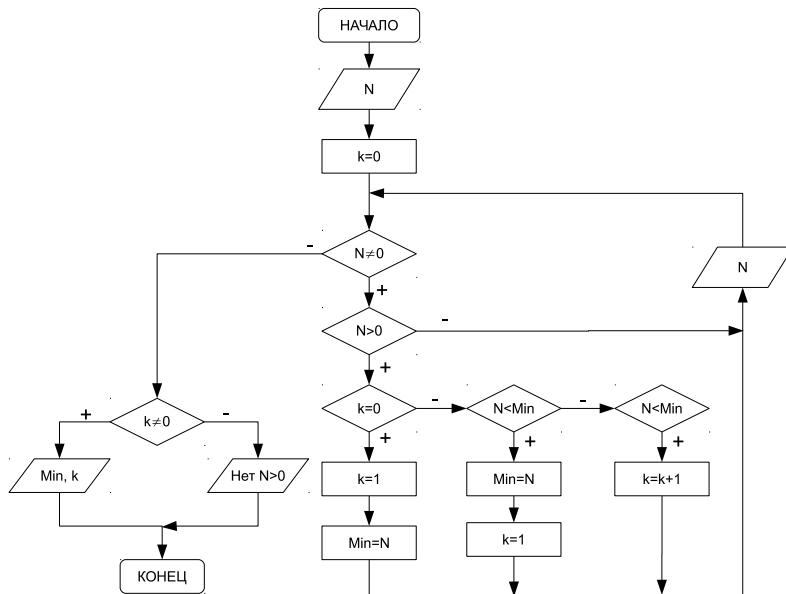


Рис. 3.32: Алгоритм поиска минимального положительного числа в последовательности.

Далее приведен текст подпрограммы с подробными комментариями<sup>11</sup>.

```

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

```

---

<sup>10</sup> 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.

<sup>11</sup> Алгоритм поиска максимального (минимального) элементов последовательности подробно описан в задаче 3.20.

```

    //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).

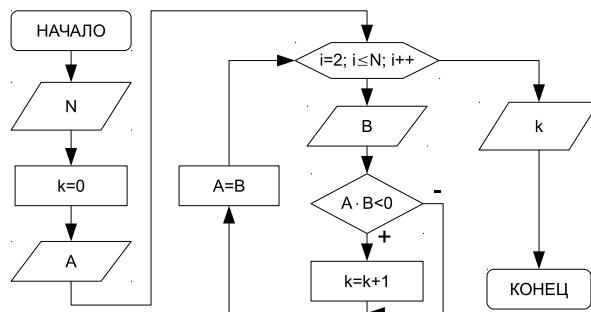


Рис. 3.33: Алгоритм решения задачи 3.22.

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;
}

```

**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).

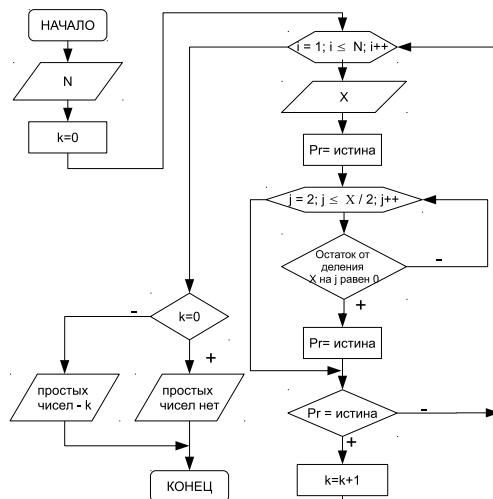


Рис. 3.34: Алгоритм поиска простых чисел в последовательности.

```

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

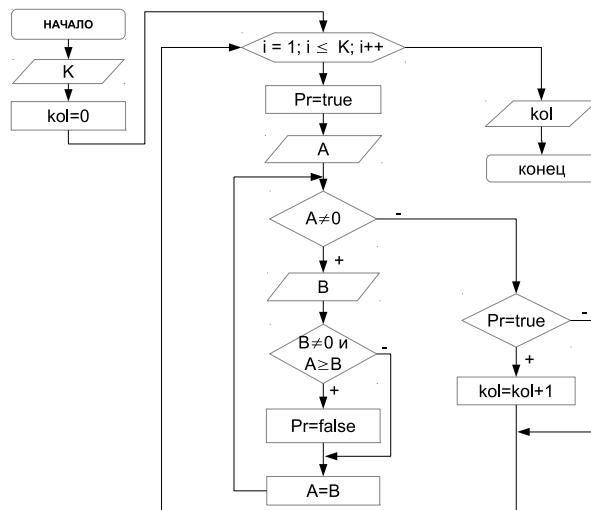


Рис. 3.35: Алгоритм решения задачи 3.24.

Программный код решения задачи 3.24:

```

#include <iostream>
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.

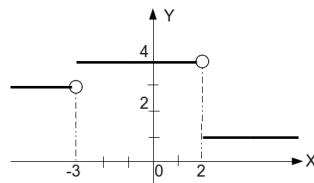


Рис. 3.36: Задание 1

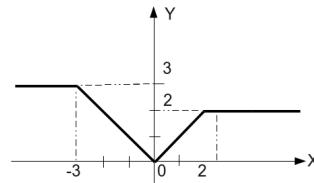


Рис. 3.37: Задание 2

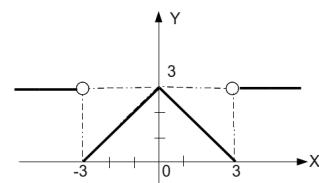


Рис. 3.38: Задание 3

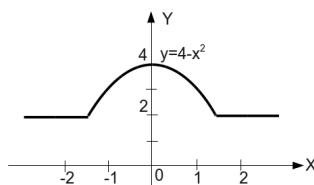


Рис. 3.39: Задание 4

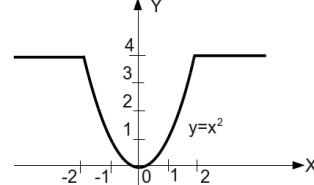


Рис. 3.40: Задание 5

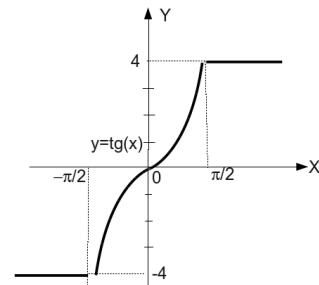


Рис. 3.41: Задание 6

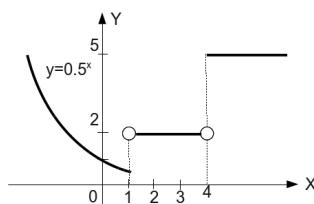


Рис. 3.42: Задание 7

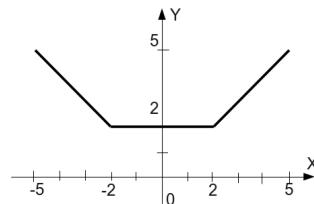


Рис. 3.43: Задание 8

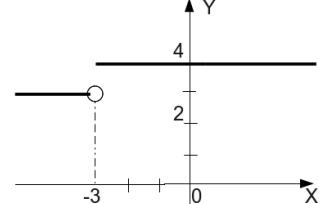


Рис. 3.44: Задание 9

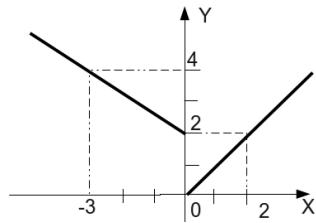


Рис. 3.45: Задание 10

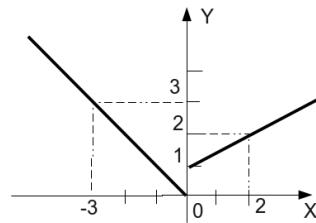


Рис. 3.46: Задание 11

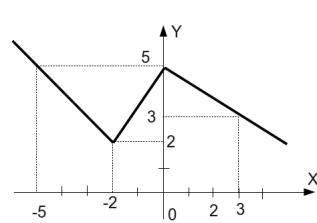


Рис. 3.47: Задание 12

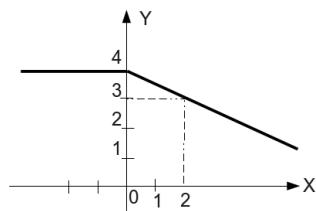


Рис. 3.48: Задание 13

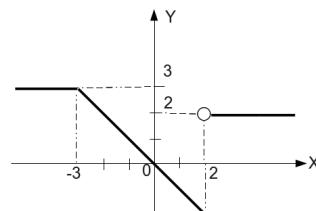


Рис. 3.49: Задание 14

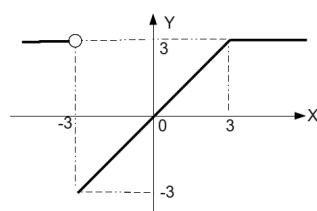


Рис. 3.50: Задание 15

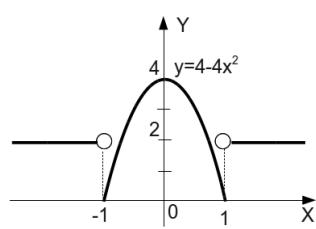


Рис. 3.51: Задание 16

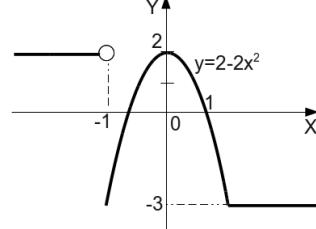


Рис. 3.52: Задание 17

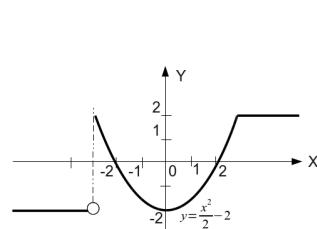


Рис. 3.53: Задание 18

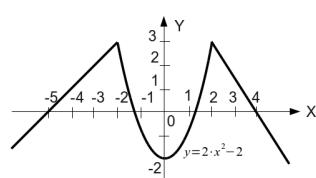


Рис. 3.54: Задание 19

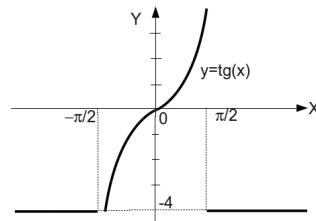


Рис. 3.55: Задание 20

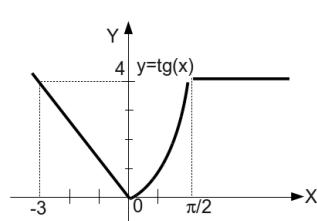


Рис. 3.56: Задание 21

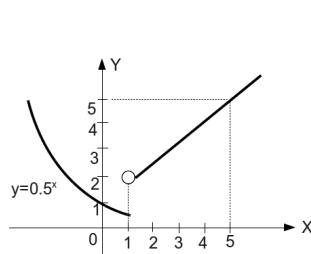


Рис. 3.57: Задание 22

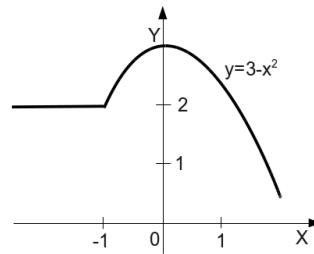


Рис. 3.58: Задание 23

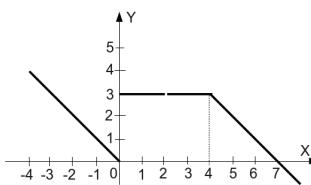


Рис. 3.59: Задание 24

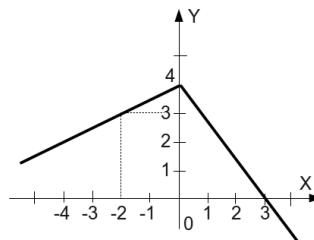


Рис. 3.60: Задание 25

### 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.

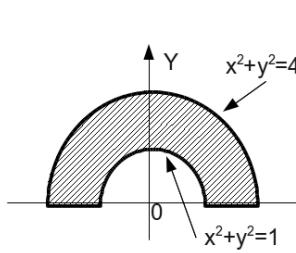


Рис. 3.61: Задание 1

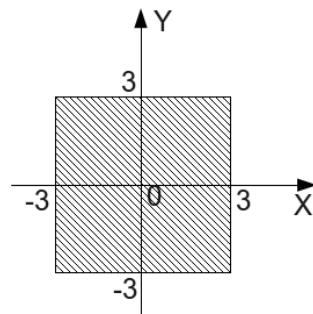


Рис. 3.62: Задание 2

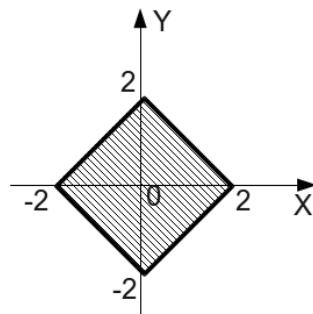


Рис. 3.63: Задание 3

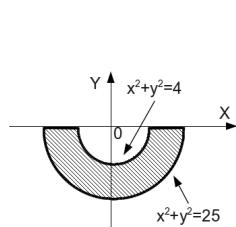


Рис. 3.64: Задание 4

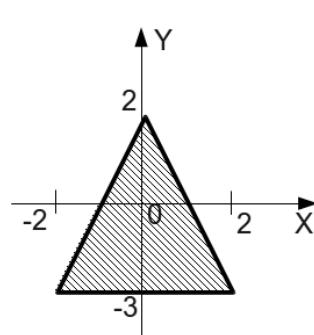


Рис. 3.65: Задание 5

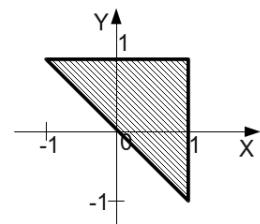


Рис. 3.66: Задание 6

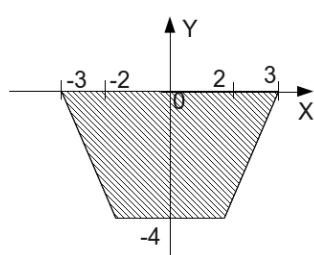


Рис. 3.67: Задание 7

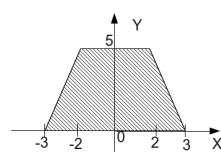


Рис. 3.68: Задание 8

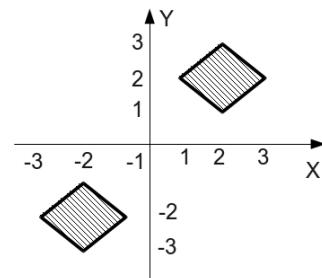


Рис. 3.69: Задание 9

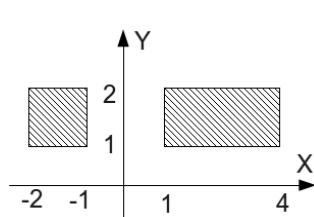


Рис. 3.70: Задание 10

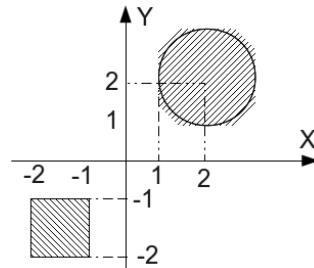


Рис. 3.71: Задание 11

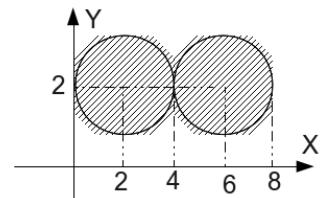


Рис. 3.72: Задание 12

### 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.

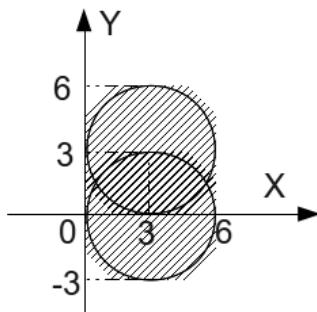


Рис. 3.73: Задание 13

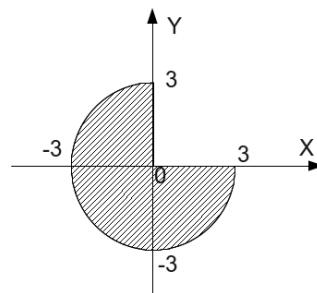


Рис. 3.74: Задание 14

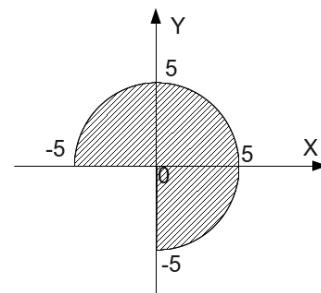


Рис. 3.75: Задание 15

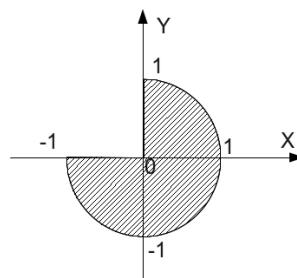


Рис. 3.76: Задание 16

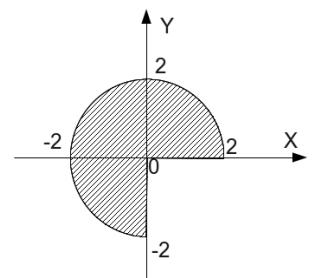


Рис. 3.77: Задание 17

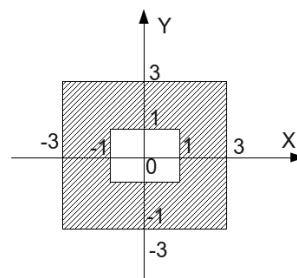


Рис. 3.78: Задание 18

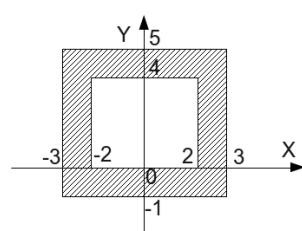


Рис. 3.79: Задание 19

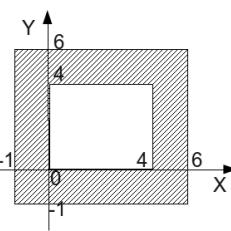


Рис. 3.80: Задание 20

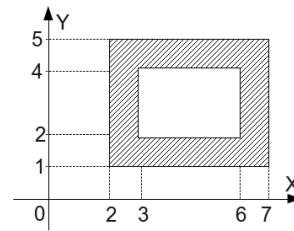


Рис. 3.81: Задание 21

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.

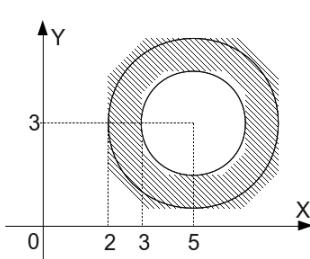


Рис. 3.82: Задание 22

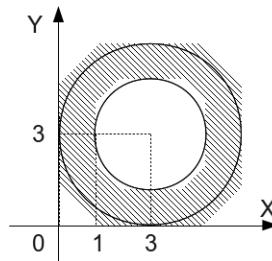


Рис. 3.83: Задание 23

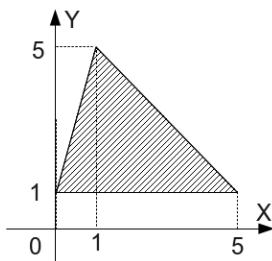


Рис. 3.84: Задание 24

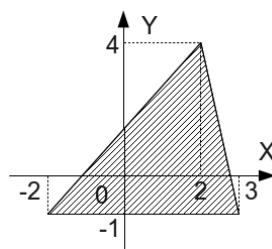


Рис. 3.85: Задание 25

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 ли 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.
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  $1$ .

### 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:

$$P = \frac{n!}{\sum_{i=1}^n i}.$$

5. Вычислить количество натуральных двузначных четных чисел не делящихся на 10.

6. Задано целое положительное число  $n$ . Определить значение выражения:

$$P = \frac{\sum_{i=1}^n i^2}{n!}.$$

7. Вычислить сумму натуральных удвоенных чисел не превышающих 25.

8. Задано целое положительное число  $n$ . Определить значение выражения:

$$P = \frac{\sum_{i=3}^n i - 2}{(n+1)!}.$$

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:  $P = \frac{\sum_{i=0}^5 3^i}{5!}$ .

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:

$$P = \frac{\sum_{i=0}^{n-1} i + 1}{(2n)!}.$$

22. Задано целое положительное число  $n$ . Определить значение выражения:
- $$P = \frac{\sum_{i=5}^{15} i}{(2 \cdot n + 1)!}.$$
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:

- Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить сумму положительных элементов последовательности.
- Вычислить сумму отрицательных элементов последовательности из  $N$  произвольных чисел.
- Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить сколько раз последовательность поменяет знак.
- В последовательности из  $N$  произвольных чисел подсчитать количество нулей.
- Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить наибольшее число в последовательности.
- Вводится последовательность из  $N$  произвольных чисел найти наименьшее число в последовательности.
- Вводится последовательность ненулевых чисел, 0 — конец последовательности. Определить среднее значение элементов последовательности.
- Вводится последовательность из  $N$  произвольных чисел, найти среднее значение положительных элементов последовательности.
- Вводится последовательность ненулевых чисел, 0 — конец последовательности. Подсчитать процент положительных и отрицательных чисел.
- Вводится последовательность из  $N$  произвольных чисел. Определить процент положительных, отрицательных и нулевых элементов.
- Вводится последовательность из  $N$  произвольных чисел. Вычислить разность между наименьшим и наибольшим значениями последовательности.
- Вводится последовательность из  $N$  положительных целых чисел. Найти наименьшее число среди четных элементов последовательности.
- Вводится последовательность из  $N$  положительных целых чисел. Определить является ли эта последовательность знакочередующейся.
- Определить является ли последовательность из  $N$  произвольных чисел строго возрастающей (каждый следующий элемент больше предыдущего).
- 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.
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.

### 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, сколько его значение (например, число 5 выводится 5 раза).
24. Дано целое число  $N > 0$ . Найти сумму  $1! + 2! + 3! + \dots + N!$
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.

## 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 вычисления 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 должна 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;
}
```

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_н(список_переменных)
{
    тело_функции_н;
}
```

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);
```

*Вызывать функцию* можно в любом месте программы. Для *вызыва 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:

```
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<sup>1</sup>.

```
#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 << "Input: " << 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.

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.

---

<sup>1</sup>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.

## 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:

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="<

```

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` выполняет перевод из радианной меры в градусную<sup>2</sup>. 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<<"Input: "<<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;
}

```

---

<sup>2</sup>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.

### 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 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;
    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%6i==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, сохраним 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, сохраним 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<sup>3</sup>.

<sup>3</sup>Выделение цифры из числа подробно описано в задаче 3.16

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;
    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<sup>4</sup>.

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 int M,R,S;
    int kol, i;
    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) 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++;
}
```

<sup>4</sup>Палиндром — любой симметричный относительно своей середины набор символов.

```

    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:

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

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

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

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.
    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,  $256_{(10)} = 400_{(8)}$ .

$$\begin{array}{r} 256 \\ -256 \\ \hline 0 \end{array} \quad \left| \begin{array}{r} 8 \\ \hline 32 \\ -32 \\ \hline 0 \end{array} \right| \quad \left| \begin{array}{r} 8 \\ \hline 4 \\ \hline 0 \end{array} \right|$$

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

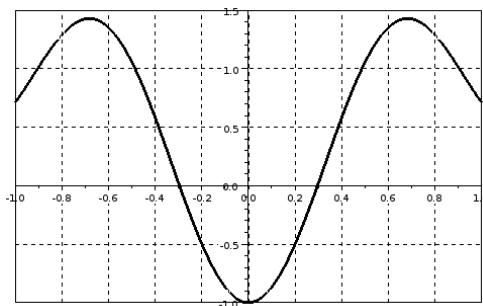


Рис. 4.2: Геометрическое решение задачи 4.7

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  $\varepsilon = 10^{-3}$ .

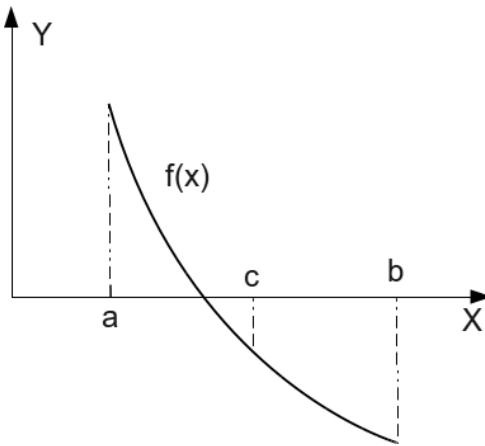


Рис. 4.3: Графическая интерпретация метода половинного деления

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 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  $(b, f(b))$ :

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

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

Найдем точку пересечения хорды с осью  $X$ :

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

итак,  $c = a - \frac{f(a)}{f(b) - f(a)}(b - a)$ .

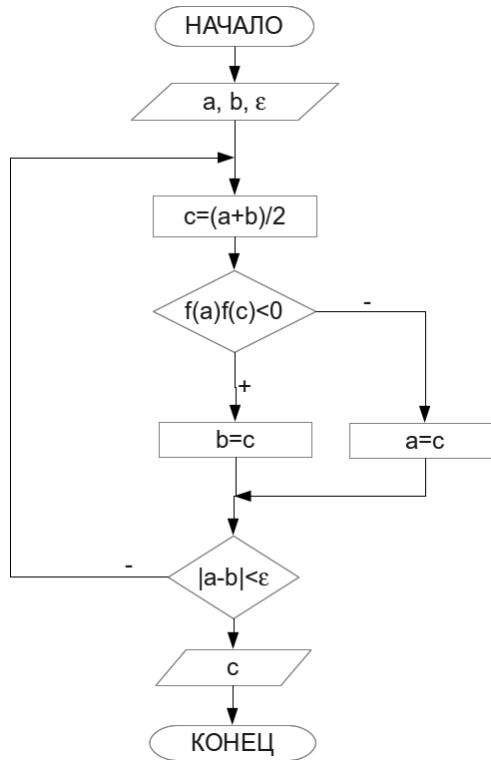


Рис. 4.4: Алгоритм решения уравнения методом дихотомии

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). Запишем уравнение этой прямой:

$$y = k \cdot x + m \quad (4.3)$$

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

Следовательно,

$$\begin{aligned} y &= f'(x) \cdot x + m, f(c) = f'(c) \cdot c + m, m = f(c) - c \cdot f'(c), \\ y &= f'(c) \cdot x + f(c) - c \cdot f'(c), y = f'(c) \cdot (x - c) + f(c). \end{aligned}$$

Найдем точку пересечения касательной с осью  $X$ :

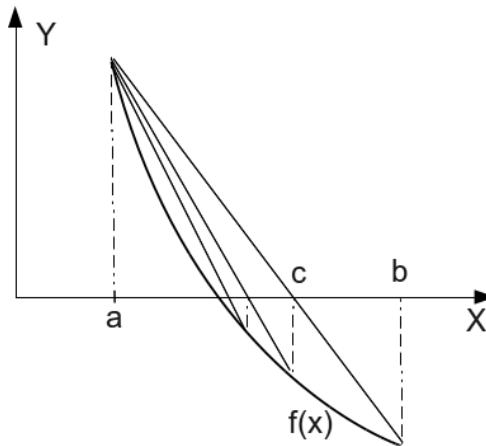


Рис. 4.5: Графическая интерпретация метода хорд

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

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  $\varepsilon = 10^{-3}$ . 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

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

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

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

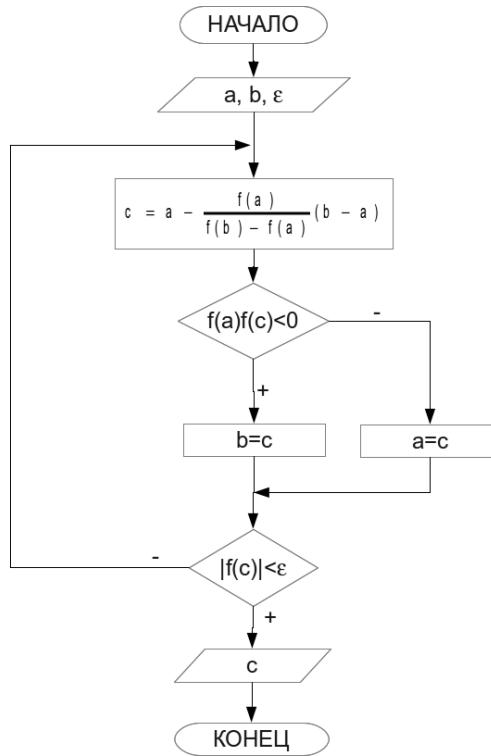


Рис. 4.6: Алгоритм метода хорд

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

$$\phi(x) = x + \lambda \cdot f(x) \quad (4.4)$$

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  $|1 + \lambda \cdot f'(x)| < 1$ :

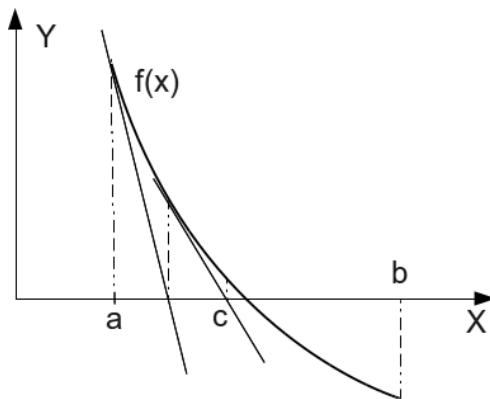


Рис. 4.7: Графическая интерпретация метода касательных

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

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

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

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

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

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.

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)
{
    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
```

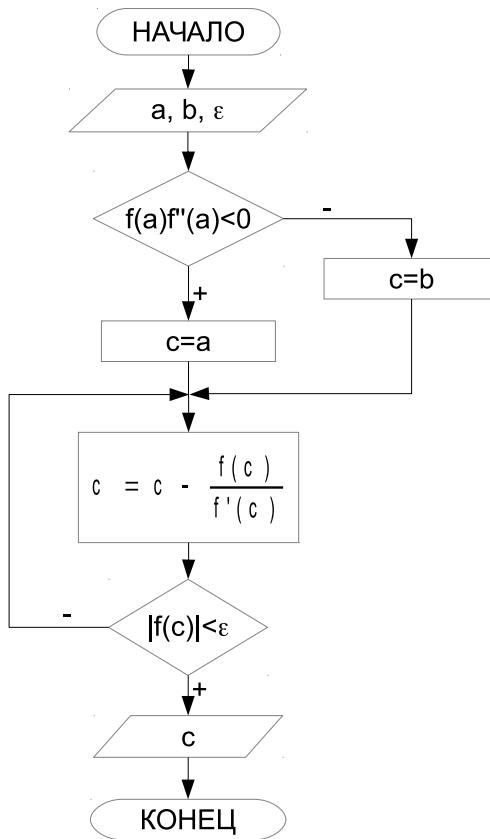


Рис. 4.8: Алгоритм метода Ньютона

```

{
    *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++;
  }
}
  
```

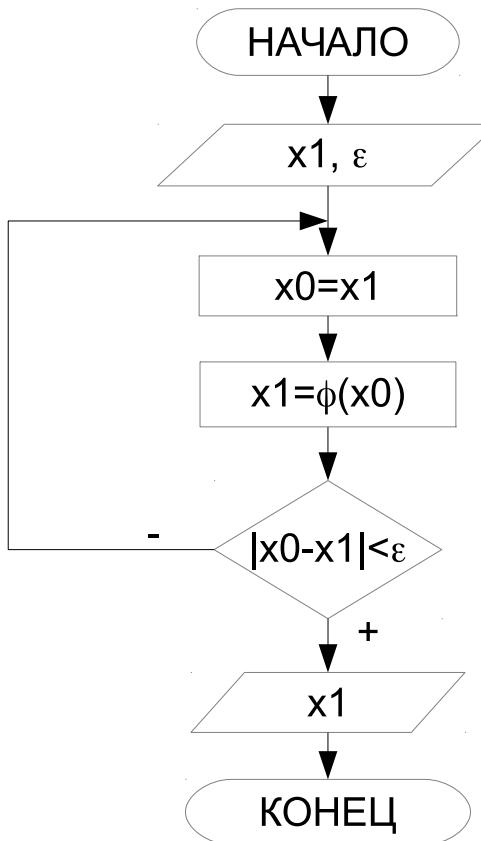


Рис. 4.9: Алгоритм метода простой итерации

```

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));
  }
  while (fabs(f(*c))>=eps);
  return k;
}
  
```

```

        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.
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.

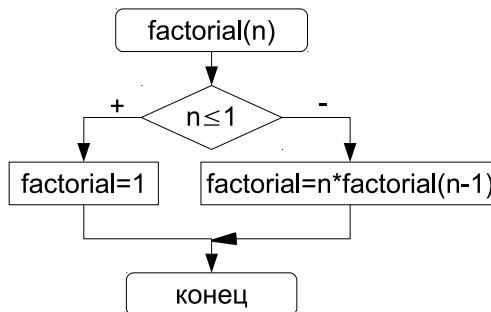


Рис. 4.10: Рекурсивный алгоритм вычисления факториала

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.

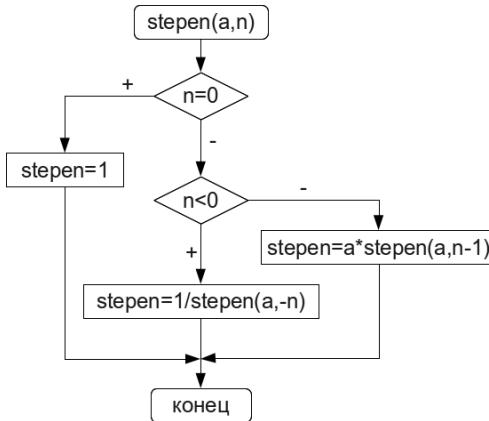


Рис. 4.11: Рекурсивный алгоритм вычисления степени числа

Текст программы с применением рекурсии:

```

#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.

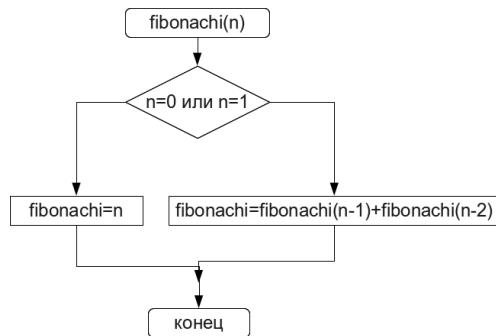


Рис. 4.12: Рекурсивный алгоритм вычисления числа Фибоначчи

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<sup>5</sup>  $n$ . Такую 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).

```
#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)
```

<sup>5</sup>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.

```

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

```

## 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> 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<<"\nVvod 4 thelih chisla\n";
    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<<"\nVvod 4 vecshestvenih chisla\n"; 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* описываются 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:

```

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<sup>6</sup>

```

#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 ;
        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 ;
}

```

<sup>6</sup>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.

```

        else cout<<"неправильный знак операции"<<endl;
    }
    else
        cout<<"недостаточное количество operandov"<<endl;
    return 0;
}

```

Ниже приведены варианты запуска программы и результаты её работы<sup>7</sup>. 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 operandov

```

## 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.
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.

<sup>7</sup>Текст программы хранится в файле 4.cpp. Имя исполняемого файла ./4 (ОС Linux)

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. Прoverить 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  $t$  вычислить и вывести на экран соответствующие *числа Фибоначчи*. Здесь  $t$  — 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 последовательности 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. Заняты 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

Таблица 4.1 — продолжение

Вариант	p	q	C	r
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
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
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

Таблица 4.2 — продолжение

№	Уравнение $f(x) = 0$	Методы решения
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

# Глава 5

## Массивы

Эта глава является ключевой в изучении программирования на С(С++). 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 С(С++)

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. Массив — 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.
```

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;
S=C[0]+C[n-1]; //Сумма первого и последнего элементов массива C.
```

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 компьютера 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 обратиться двумя способами:

1. В соответствии с синтаксисом языка С(С++) 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 массива, необходимо изменить текст программы и перекомпилировать её. Для динамического выделения памяти для массивов в С(С++) можно использовать указатели и операторы (функции) выделения памяти.

## 5.2 Динамические массивы в С(С++)

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`.

### 5.2.1 Функция `malloc`

Функция `malloc` 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<sup>1</sup>, 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;  
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.
```

---

<sup>1</sup>`size_t` — базовый беззнаковый целочисленный тип языка С/С++, 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  $2^{64} - 1$ ).

### 5.2.3 Функция realloc

Функция `realloc` изменяет размер ранее выделенного 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.

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?

Пусть 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;
cin >> k; //Вводим размер массива k.
//Выделение памяти для хранения динамического массива из k чисел.
x = new double[k]; //Адрес начала массива хранится в переменной x.
x = (double *) malloc(k * sizeof(float)); //Память можно будет выделить так
x = (double *) calloc(k, sizeof(float)); //или так
```

Мы имеем указатель на тип данных `double`, вводим  $k$  — размер динамического массива, 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 сначала 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 динамическом массиве выделяется столько элементов, сколько необходимо.

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. Организуется 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.

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.

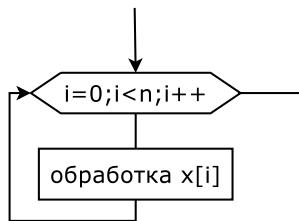


Рис. 5.1: Алгоритм обработки элементов массива

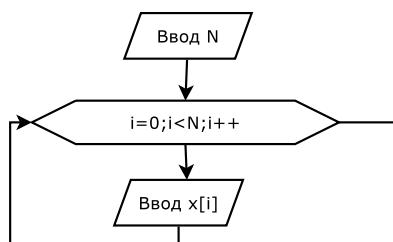
### 5.4.1 Ввод-вывод элементов массива

Ввод и вывод массивов также осуществляется поэлементно. Блок-схемы алгоритмов ввода и вывода элементов массива  $X[N]$  изображены на рис. 5.2–5.3.

Рассмотрим несколько вариантов ввода массива:

```

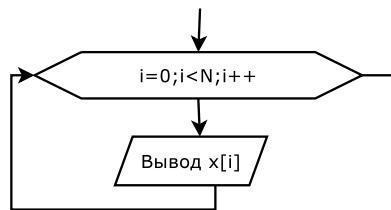
//Вариант 1. Ввод массива с помощью функции scanf.
//При организации ввода используются специальные символы: табуляция — \t
//и переход на новую строку — \n.
#include <stdio.h>
int main()
{
float x[10]; int i,n;
printf("\n N="); scanf("%d",&n); //Ввод размерности массива.
printf("\n Введите элементы массива X \n");
for(i=0;i<n; i++)
scanf("%f",x+i); //Ввод элементов массива в цикле. Обратите внимание!!! Использование 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;
printf("\n N="); scanf("%d",&n); //Ввод размерности массива.
printf("\n Массив X \n");
for (i=0;i<n; i++)
{
    printf("\n Элемент %d \t", i); //Сообщение о вводе элемента.
    scanf("%f",&b); //Ввод переменной b.
    x[i]=b; //Присваивание элементу массива значения переменной b.
}
return 0;
}
  
```

Результат работы программы:

```

N=4
Массив X
Элемент 0 8.7
Элемент 1 0.74
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.
    X=new int [N]; //Выделение памяти для динамического массива из N элементов.
    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 затем к переменной  $S$  последовательно добавляются элементы массива  $X$ . Блок-схема алгоритма расчёта суммы приведена на рис. 5.4.

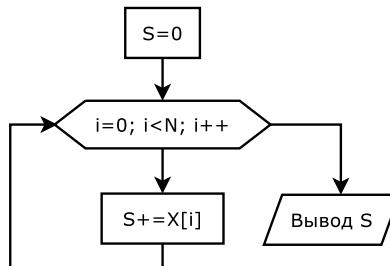


Рис. 5.4: Алгоритм вычисления суммы элементов массива

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  $X[N]$ .
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.

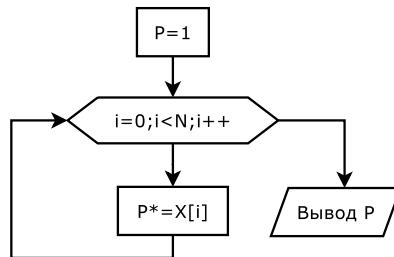


Рис. 5.5: Вычисление произведения элементов массива

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 ;
    cout<<"Введите размер массива " ; cin>>N; //Ввод размерности массива.
    X=new int [N]; //Выделение памяти для хранения динамического массива X.
    cout<<"Введите массив X\n" ; //Ввод массива X.
    for ( i=0 ; i<N ; i++ )
    
```

```

{ cout << "X (" << i << ")" = " ; cin >> X[ i ]; }
for (P=1,S=i=0;i<N; i++) // В цикле перебираем все элементы массива
{
    //Если очередной элемент массива — простое число, добавляем его к сумме.
    if ( prosto(x[ i ]) ) S+=X[ i ];
    //Если очередной элемент массива отрицателен, умножаем его на P.
    if ( X[ i ]<0 ) P*=X[ i ];
}
cout << "S=" << S << endl; //Вывод S и P на экран.
delete [] X; //Освобождение занимаемой массивом X памяти.
return 0;
}

```

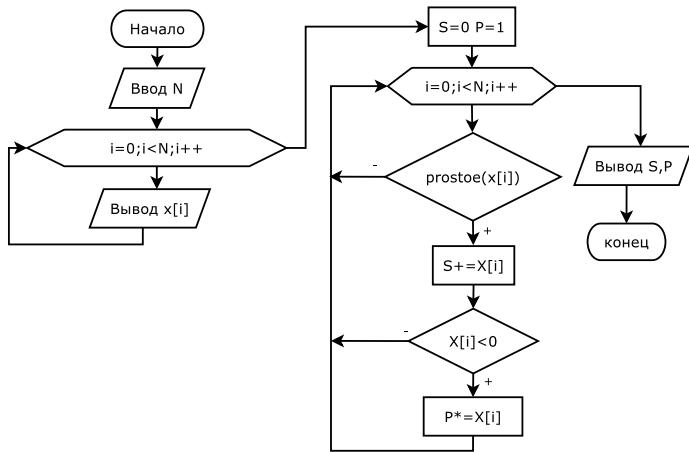


Рис. 5.6: Блок-схема алгоритма решения задачи 5.1

Результаты работы программы представлены ниже.

```

Введите размер массива 10
Введите 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.

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 <i>Max</i>	4	7	7	8	9	9
Значение переменной <i>Nmax</i>	1	2	2	4	5	5

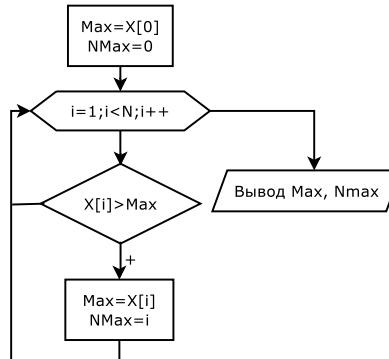


Рис. 5.7: Поиск максимального элемента и его номера в массиве

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;
    cout << "Введите размер массива "; cin >> N; // Ввод размерности динамического массива
    X = new float [N]; // Выделения памяти для хранения динамического массива X.
    cout << "Введите элементы массива X\n"; // Ввод динамического массива X.
    for (i=0; i < N; i++)
        cin >> X[i];
    // В переменной nom будем хранить номер максимального элемента.
    nom = 0; // Предположим, что максимальным элементом, является элемент с номером 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. Будем 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;
    cout << "n="; cin >> n; // Ввод количества элементов в массиве.
```

```

x=new int [ n ]; //Выделяем память для динамического массива x.
cout<<"Введите элементы массива X"; //Ввод элементов массива.
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.
    if ( prostoe( x[ i ] ) ) //Если x[ i ] — простое число.
    {
        k++; //Увеличиваем счётчик количества простых чисел в массиве.
        //Если текущий элемент является первым простым числом в массиве,
        //объявляем его минимумом, а его номер сохраняем в переменную 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.
        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<<"\t nom = "<<nom<<endl;
//Иначе выводим сообщение о том, что в массиве нет простых чисел.
else cout<<"Нет простых чисел в массиве"<<endl;
return 0;
}

```

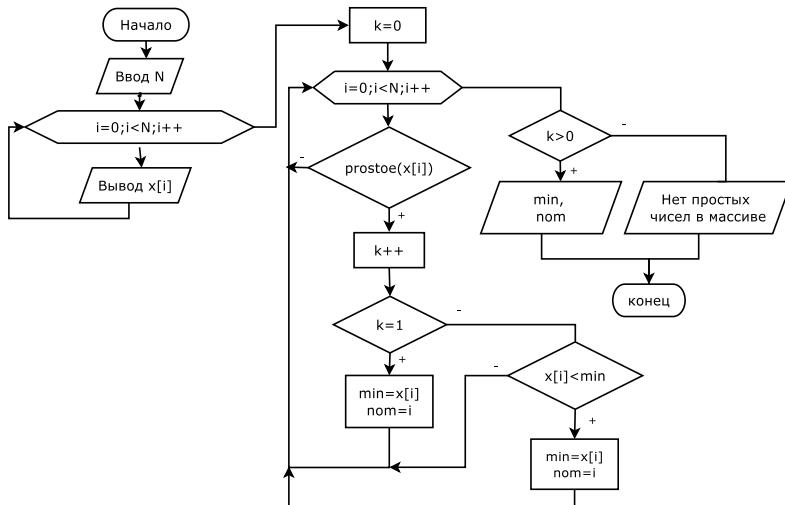


Рис. 5.8: Блок-схема решения задачи 5.2

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++)
        if (i!=nmin1) //Если номер текущего элемента не совпадает с nmin1 ,
    {
        kvo++; //увеличиваем количество таких элементов на 1.
        //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.
            if (X[i]<X[nmin2]) nmin2=i;
    }
    //Вывод двух минимальных элементов и их индексов.
    cout<<"nmin1="<<nmin1<<"\tnmin2="<<X[nmin1]<<endl;
    cout<<"nmin2="<<nmin2<<"\tnmin2="<<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 минималь-

ных 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 ( $\text{if } (x[i] < x[nmin\_temp]) nmin\_temp=i$ ). Блок-схема алгоритма поиска  $k$  минимальных элементов массива представлена на рис. 5.9<sup>2</sup>. Далее приведен текст программы с комментариями.

```
#include <iostream>
using namespace std;
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];
    for (j=0; j<k; j++) //Цикл по переменной j для поиска номера j + 1 минимального элемента
    {
        kvo=0;
        for (i=0; i<n; i++) //Перебираем все элементы массива.
        {
            //Цикл по переменной p проверяет содержит ли номер i в массиве nmin.
            pr=false;
            for (p=0; p<j; p++)
                if (i==nmin[p]) pr=true;
            if (!pr) //Если не содержитя, то количество элементов увеличить на 1.
            {
                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;
            }
        }
        nmin[j]=nmin_temp; //Номер очередного минимального элемента записываем в массив.
    }
    for (j=0; j<k; j++) //Вывод номеров и значений k минимальных элементов массива.
        cout << "nmin[" << j << "]=" << nmin[j] << "\tmin=" << x[nmin[j]] << endl;
    return 0;
}
```

<sup>2</sup> В блок-схеме отсутствует ввод данных и вывод результатов.

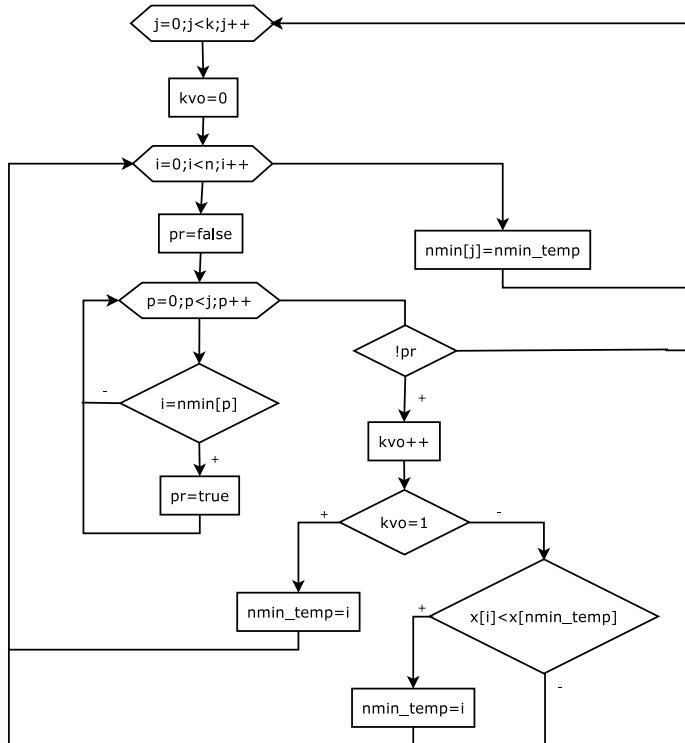


Рис. 5.9: Блок-схема алгоритма поиска  $k$  минимальных элементов в массиве  $x$ .

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;
//Функция проверяет содержится ли число i в массиве x из n элементов.
//Функция возвращает true, если содержитя и false, если не содержитя.
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 ];
for (j=0;j<k; j++) //Цикл по переменной j для поиска номера j + 1 минимального элемента
{
    kvo=0;
    for ( i=0; i<n; i++) //Перебираем все элементы массива.
    {
        //Вызов функции proverka, определяем, содержит 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 ;
}
for (j=0;j<k; j++) //Вывод номеров и значений к минимальных элементов массива.
    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.

```

#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;
cout<<"Преобразованный массив X\n"; //Вывод преобразованного массива.
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 следует 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;
    cout<<"n="; cin>>n; //Ввод количества элементов в массиве.
    x=new int [n]; //Выделяем память для динамического массива x.
    cout<<"Введите элементы массива X"; //Ввод элементов массива.
    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.
    if (prostoe(x[ i ])) //Если x[i] — простое число.
    {
        //Домножаем y[i] на p, а также увеличиваем счётчик 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.
if (prostoe(x[ i ])) //Если x[i] — простое число.
{
    //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 этих 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).

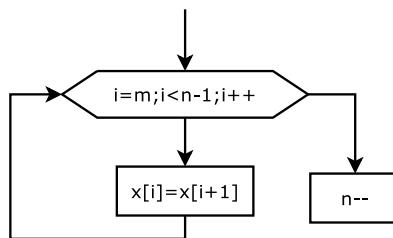


Рис. 5.10: Алгоритм удаления элемента из массива

Фрагмент программы на C++:

```

cout<<"\n m="; cin>>m; //Ввод номера элемента, подлежащего удалению.
for (i=m; i<n-1; X[ i ]=X[ i + 1 ], i++) //Удаление m-го элемента.
n--;
for (i=0; i<n-1; i++) cout<<X[ i ]<<"\t"; //Вывод измененного массива.

```

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 уменьшатся 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 программы приведён далее.

```
#include <iostream>
#include <math.h>
using namespace std;
int main(int argc, char **argv)
{
    int i, j, n=20;
    float x[n]; //Выделяем память для динамического массива x.
    cout<<"Введите элементы массива X\n"; //Ввод элементов массива.
    for (i=0;i<n; i++)
        cin>>x[i];
    for (j=1;j<=6; j++) //Шесть раз повторяем алгоритм удаления элемента с индексом 5.
        for (i=5; i<n-j; i++) //Удаление элемента с индексом 5.
            x[i]=x[i+1];
    cout<<"Преобразованный массив X\n"; //Вывод элементов массива.
    for (i=0; i<n-6; i++)
        cout<<x[i]<<"\t";
    cout<<endl;
    return 0;
}
```

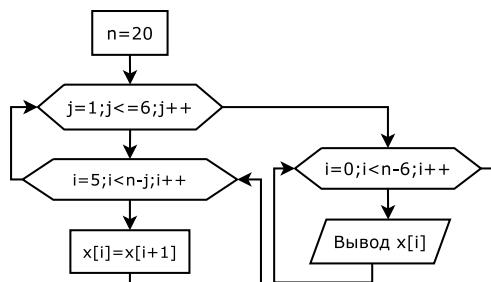


Рис. 5.11: Алгоритм решения задачи 5.6

**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;
    float x[n]; //Выделяем память для динамического массива x.
    cout<<"Введите элементы массива X\n"; //Ввод элементов массива.
    for(i=0;i<n; i++)
        cin>>x[i];
    for(j=1;j<=6; j++) //Шесть раз повторяем алгоритм удаления элемента с индексом 5.
        for(i=5; i<n-j; i++) //Удаление элемента с индексом 5.
            x[i]=x[i+1];
    cout<<"Преобразованный массив X\n"; //Вывод элементов массива.
    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;
    float x[n]; //Выделяем память для динамического массива x.
    cout<<"Введите элементы массива X\n"; //Ввод элементов массива.
    for(i=0; i<n; i++)
        cin>>x[i];
    for(i=0; i<n;)
        if (x[i]>0) //Если текущий элемент положителен,
        { //то удаляем элемент с индексом i.
            for(j=i; j<n-1; j++)
                x[j]=x[j+1];
            n--;
        }
}
```

```

    }
    else i++; //иначе — переходим к следующему элементу массива.
cout<<"Преобразованный массив X\n"; //Вывод элементов массива.
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  $k$   $0, 2, 3, 4, 5, \dots$ , 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 отрицательно ( $x_i \cdot x_{i+1} < 0$ ), 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 положительно ( $x_i \cdot x_{i+1} > 0$ ), 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`)<sup>3</sup>, 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 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;
  cout<<"n="; cin>>n; //Ввод размера массива.
  x=new float [n]; //Выделение памяти для массива.
  cout<<"Введите массив x\n"; //Ввод элементов массива.
  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++;
        if (kgr==1) //Если это первая группа (kgr=1) знакочередующихся элементов,
        {
          max=k; //то max — длина группы (max=k),
          kon_max=i; //kon_max — номер последнего элемента группы.
        }
        else //это не первая группа (kgr ≠ 1), сравниваем max и длину текущей группы.
          if (k>max) //Если k>max,
          {
            max=k; //max — длина группы,
            kon_max=i; //kon_max — номер последнего элемента группы.
          }
        k=1; //В переменную k записываем 1 для формирования новой группы элементов.
      }
    if (k>1) //Если в конце массива была группа.
    {
      kgr++; //Количество групп увеличиваем на 1.
      if (kgr==1) //Если это первая группа,
      {
        max=k; //то max — длина группы,
        kon_max=n-1; //группа закончилась на последнем элементе массива.
      }
    }
}
```

<sup>3</sup>В переменной

```

    }
else
    if (k>max) //Если длина очередной группы больше max.
    {
        max=k; //то в max записываем длину последней группы,
        kon_max=n-1; //группа закончилась на последнем элементе массива.
    }
if (kgr>0) //Если знакочередующиеся группы были,
{ //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;
    for (i=kon_max-max+1;i<=kon_max; i++) //а также саму группу.
        cout<<x[i]<< " ;
    cout<<endl;
}
else //Если знакочередующихся групп не было, то выводим сообщение об этом.
    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

$$Y[0] < Y[1] < \dots < Y[n - 1],$$

и в порядке убывания, если

$$Y[0] > Y[1] > \dots > Y[n - 1].$$

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 расположенных 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;
```

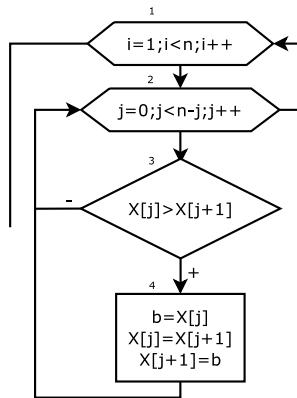


Рис. 5.12: Сортировка массива пузырьковым методом

```

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 ];
    }
    for(j=1; j<n; j++) //Упорядочивание элементов в массиве по возрастанию их значений.
        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 на сохраненный в b
            }
    for (i=0; i<n; i++) cout<<y[ i ]<<"\t"; //Вывод упорядоченного массива
    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 оказаться 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 ];
    }
    for (j=1;j<n; j++) //Упорядочивание элементов массива по убыванию их значений.
    {
        for (pr=false , i=0; i<n-j ; i++) //Предполагаем, что массив уже отсортирован
            (pr=false).
            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
                pr=true; //Если элементы менялись местами, массив ещё неотсортирован (pr=true);
            }
        cout<<"j="<<j<<endl;
        //Если на j-м шаге соседние элементы не менялись, то массив уже отсортирован,
        if (!pr) break; //повторять смысла нет;
    }
    for (i=0;i<n; i++) cout<<y[ i]<<"\t"; //Вывод упорядоченного массива
    return 0;
}

```

Вариант 2.

```

#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
    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 ];
    }
    for (j=1;pr ; j++) //Упорядочивание элементов массива по убыванию их значений.
    { //Вход в цикл, если массив не отсортирован (pr=true).
        for (pr=false , i=0; i<n-j ; i++) //Предполагаем, что массив уже отсортирован
            (pr=false).
            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
                pr=true; //Елементы менялись местами, массив ещё неотсортирован (pr=true)
            }
        for (i=0;i<n; i++) cout<<y[ i]<<"\t"; //Вывод упорядоченного массива
        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.

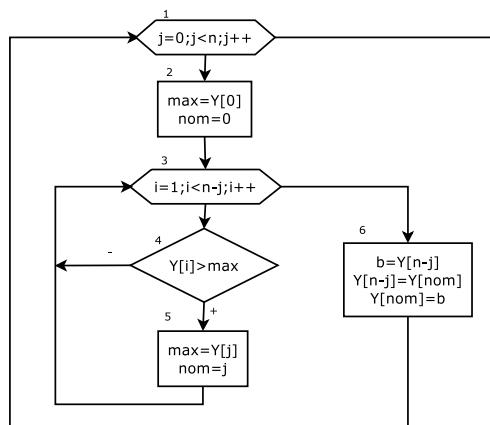


Рис. 5.13: Сортировка массива выбором наибольшего элемента

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;}
  
```

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.

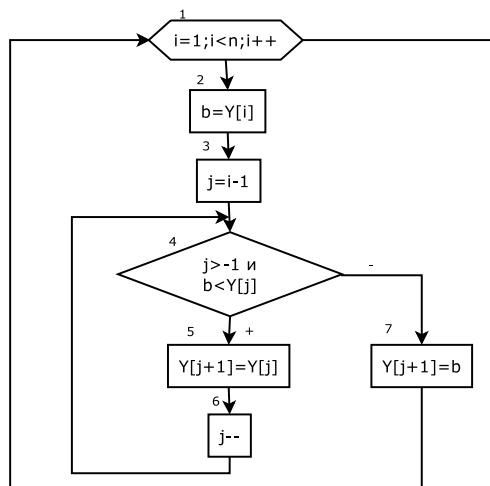


Рис. 5.14: Сортировка массива вставкой

Организуем цикл для просмотра всех элементов массива, начиная с первого (блок 1). Сохраним значение текущего  $i$ -го элемента во 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  $a_0 \geq a_1 \geq a_2 \geq \dots \geq a_{n-3} \geq a_{n-2} \geq a_{n-1}$ .

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<sup>4</sup>.
3. На освободившееся место с номером  $k$  записать число  $b$ .

Текст программы с комментариями приведён ниже.

```
#include <iostream>
1 using namespace std;
2 int main(int argc, char **argv)
3 {
4     int i, k, n;
5     float b;
6     cout<<"n="; cin>>n; //Ввод размера исходного массива.
7     float a[n+1]; //Выделение памяти с учётом предстоящей вставки одного числа в массив.
8     cout<<"Введите массив a\n"; //Ввод исходного упорядоченного по убыванию массива.
9     for(i=0;i<n; i++)
10         cin>>a[i];
11     cout<<"Введите число b="; cin>>b; //Ввод вставляемого в массив числа b .
12     //Если число b меньше всех элементов массива, записываем b в последний элемент массива.
13     if (a[n-1]>=b) a[n]=b;
14     else //Иначе
15     {
16         for(i=0;i<n; i++) //Ищем первое число, меньшее b .
17             if (a[i]<=b)
18             {
19                 k=i; //Запоминаем его номер в переменной k .
20                 break;
21             }
22         for(i=n-1;i>=k; i--) //Все элементы массива от n - 1-го до k-го сдвигаем на один вправо.
23         a[i+1]=a[i];
24         a[k]=b; //Вставляем число b в массив.
25     }
26     cout<<"Преобразованный массив a\n";
27     for(i=0;i<=n; i++)
28         cout<<a[i]<<"\t";
29     return 0;
30 }
```

Обратите внимание, при решении задачи с массивом упорядоченным по возрастанию необходимо во фрагменте со строки 14 по строку 22 заменить все операции отношения на противоположные.

**Задача 5.11.** Проверить является ли массив упорядоченным возрастанию.

Для проверки упорядоченности по возрастанию  $a[n]$ <sup>5</sup> можно поступить следующим образом. Предположим, что массив упорядочен (`pr=true`). Если хотя бы

<sup>4</sup>Очень важно, что сдвиг осуществляем от  $n - 1$ -го до  $k$ -го, в противном случае элементы массива оказались бы испорченными.

<sup>5</sup>Массив является упорядоченным по возрастанию, если выполняются условия  $a_0 \leq a_1 \leq a_2 \leq \dots \leq a_{n-3} \leq a_{n-2} \leq a_{n-1}$ .

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;
    cout<<"n="; cin>>n; //Ввод размера исходного массива.
    float *a=new float [n]; //Выделение памяти для массива.
    cout<<"Введите массив a\n"; //Ввод исходного массива.
    for (i=0;i<n; i++)
        cin>>a[i];
    //Предполагаем, что массив упорядочен (pr=true), перебираем все пары соседних значений
    ////(i — номер пары), при 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

$$\int_{-1}^1 f(x)dx = \sum_{i=1}^n A_i f(t_i), \quad t_i \text{ — точки из интервала } [-1, 1], \quad A_i \text{ — рассчитываемые коэффициенты.}$$

Методика определения  $A_i$ ,  $t_i$  представлена в [3]. Для практического использования 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 от  $a$  до  $b$  квадратурная формула Гаусса может быть записана следующим образом  
 $\int_a^b f(x)dx = \frac{b-a}{2} \sum_{i=1}^n A_i f\left(\frac{b+a}{2} \cdot \frac{b-a}{2} t_i\right)$ , 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  
 $\int_{-1}^1 f(x)dx = \frac{2}{n} \sum_{i=1}^n f(t_i)$ ,  $t_i$  — точки из интервала  $[-1, 1]$ . Формула Чебышева для вычисления интеграла на интервале от  $a$  до  $b$  может быть записана так  $\int_a^b f(x)dx = \frac{b-a}{n} \sum_{i=1}^n f\left(\frac{b+a}{2} \cdot \frac{b-a}{2} t_i\right)$  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
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  $\int_a^b f(x)dx$  методами Гаусса и Чебышева. Далее приведены тексты функций и функция `main()`. В качестве тестовых использовались интегралы  $\int_0^2 \sin^4 x dx \approx 0.9701$ ,  
 $\int_5^{13} \sqrt{2x-1} dx \approx 32.667$ .

```
#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]={-0.9111589, -0.601019, -0.528762, -0.167906, 0, 0.167906, 0.528762,
    0.601019, 0.9111589};
    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]);
    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 интервал 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 С(С++) 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:

```
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
    {
        for (i=k; i<n-1; i++) //Удаляем элемент с номером k.
            x[i]=x[i+1];
        n--;
        return n; //Возвращаем изменённый размер массива.
    }
}
int main(int argc, char **argv)
{
    int i, n;
    cout<<"n="; cin>>n;
    float x[n]; //Выделяем память для динамического массива x.
    cout<<"Введите элементы массива X\n"; //Ввод элементов массива.
    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,
```

```

n=udal(x, i, n); //и возвращает размер массива.
else i++; //иначе (x[i]<=0) — переходим к следующему элементу массива.
cout<<"Преобразованный массив X\n"; //Вывод элементов массива после удаления.
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;
}
int main( int argc, char **argv)
{
    int i,n;
    cout<<"n=" ; cin>>n;
    float x[n]; //Выделяем память для динамического массива x.
    cout<<"Введите элементы массива X\n"; //Ввод элементов массива.
    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);
        else i++; //иначе (x[i]<=0) — переходим к следующему элементу массива.
    cout<<"Преобразованный массив X\n"; //Вывод элементов массива после удаления.
    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 массива.

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 prostoе(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. 6.4);
- `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;
float sr_arifm(int *x, int n) //Функция вычисления среднего значения.
{
    int i; float s=0;
    for (i=0;i<n; s+=x[i], i++);
    if (n>0) return (s/n);
    else return 0;
}
bool prostoе(int n) //Функция для определения простого числа.
{
    bool pr; int i;
    for (pr=true, i=2; i<=n/2; i++)
        if (n%i==0) {pr=false; break;}
    return (pr);
}
void udal(int *x, int m, int *n) //Функция удаления элемента из массива.
{
    int i;
    for (i=m; i<*n-1; *(x+i)=*(x+i+1), i++);
    --*n;
    realloc ((int *)x, *n*sizeof(int));
}
void upor(int *x, int n, bool pr=true) //Функция сортировки массива.
{
    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))
            {
                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 ; )
    {
        if (prostoe(* (a+i ))&& *(a+i)<sr) //Если число простое и меньше среднего,
            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.
    cout<<"Упорядоченный массив A\n" ; //Вывод упорядоченного массива.
    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;
        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 раз мерности  $Y(m)$ , сформировать общий массив  $Z(n + m)$ . Выполнить сорти ровку 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 раз мерности  $B(m)$ , сформировать общий массив  $C(n + m)$ . Преобразовать по лученный массив так, чтобы все его положительные элементы стали отри цательными 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 мас сива.

Из заданного массива  $A(n)$  все положительные числа переписать в мас сив  $B$ , а отрицательные в массив  $C$ . Удалить из массива  $A(n)$  первый ну левой 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

- 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.

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 трём;
- 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.

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 «нuleй» 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 массиве  $U$ .
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.

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 десятичных 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.
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.
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.

# Глава 6

## Статические и динамические матрицы

Данная глава посвящена обработке матриц в C++, на большом количестве примеров будут рассмотрены возможности языка для обработки статических и динамических матриц. В завершающем параграфе будут рассмотрено использование двойных указателей и функций на примере решения задач линейной алгебры.

### 6.1 Статические матрицы C(C++)

Матрица — это двумерный массив, каждый элемент которого имеет два индекса: номер строки —  $i$ ; номер столбца —  $j$ .

Статический двумерный массив (матрицу) можно объявить так:

```
тип имя_переменной [n] [m];
```

где тип определяет тип элементов массива, **имя\_переменной** — имя матрицы,  $n$  — количество строк,  $m$  — количество столбцов в матрице. Строки нумеруются от 0 до  $n - 1$ , столбцы — от 0 до  $m - 1$ .

Например,

```
double x[20] [35];
```

Описана матрица вещественных чисел  $x$ , состоящая из 20 строк и 35 столбцов (строки нумеруются от 0 до 19, столбцы от 0 до 34).

Как и любой другой переменной, матрице можно присвоить начальное значение, например `int A[2][3]={ {1,2,3}, {4,5,6} };`

Для обращения к элементу матрицы необходимо указать ее имя, и в квадратных скобках номер строки, а затем в квадратных скобках — номер столбца. Например,  $x[2][4]$  — элемент матрицы  $x$ , находящийся в третьей строке и пятом столбце<sup>1</sup>.

Для работы с элементами матрицы необходимо использовать два цикла. Для построчной обработки матрицы значениями параметра первого (внешнего) цикла будут номера строк матрицы, значениями параметра второго (внутреннего)

---

<sup>1</sup>Напоминаем, что нумерация строк и столбцов идет с 0.

цикла — номера столбцов (см. рис. 6.1). При построчной обработке матрицы вначале поочерёдно рассматриваются элементы первой строки (столбца), затем второй и т.д. до последней. Если необходимо обрабатывать матрицу по столбцам, то необходимо организовать внешний цикл по столбцам, а внутренний по строкам (см. рис. 6.2).

## 6.2 Динамические матрицы

В предыдущем параграфе мы рассмотрели описание статических матриц, в этом случае память для хранения матрицы выделяется в момент описания. Однако в C++ существует возможность создавать динамические матрицы. Динамические матрицы можно создавать с использованием обычных указателей и с помощью двойных указателей. Рассмотрим оба способа работы с динамическими матрицами последовательно.

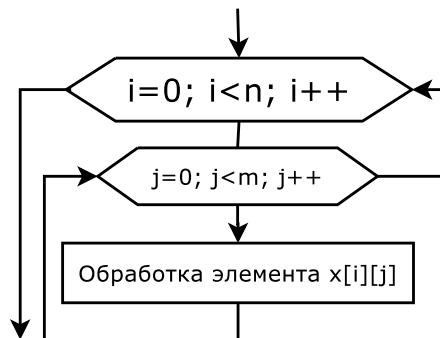


Рис. 6.1: Блок-схема построчной обработки матрицы

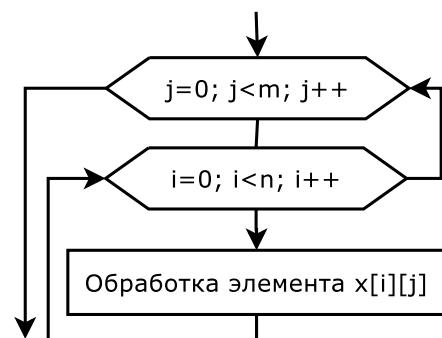


Рис. 6.2: Блок-схема обработки матрицы по столбцам

### 6.2.1 Использование указателей для работы с динамическими матрицами

При работе с динамическими матрицами можно использовать обычные указатели. После описания указателя, необходимо будет выделить память для хранения  $N \times M$  элементов ( $N$  — число строк,  $M$  — число столбцов). Рассмотрим в качестве примера выделение памяти для хранения целочисленной матрицы размером  $N \times M$ .

```
int *A, n, m;
A=(int *) calloc (N*M, sizeof(int));
```

Для выделения памяти можно использовать также и функцию `malloc`

```
A=(int *) malloc (N*M*sizeof(int));
```

или операцию `new`

```
A=new int [N*M];
```

Память мы выделили, осталось найти способ обратиться к элементу матрицы. Все элементы матрицы хранятся в одномерном массиве размером  $N \times M$  элементов. Сначала в этом массиве расположена 0-я строка матрицы, затем 1-я и т.д. Поэтому для обращения к элементу  $A[i][j]$  необходимо, по номеру строки  $i$  и номеру столбца  $j$  вычислить номер этого элемента  $k$  в динамическом массиве. Учитывая, что в массиве элементы нумеруются с нуля  $k = iM + j$ . Обращение к элементу  $A[i][j]$  будет таким  $*(A+i*m+j)$ .

### 6.2.2 Использование двойных указателей для работы с динамическими матрицами

Основной способ работы с динамическими матрицами базируется на использовании двойных указателей. Рассмотрим следующий фрагмент программы.

```
int main()
{
    int N,M;
    float **a;
    a=new float *[N];
}
```

С помощью оператора `new` создан массив из  $n$  элементов<sup>2</sup>, в котором каждый элемент является адресом, где хранится указатель (фактически каждый указатель — адрес строки матрицы). Осталось определить значение элементов массива. Для этого организуем цикл (переменная цикла  $i$  изменяется от 0 до  $N - 1$ ), в котором будет выделяться память для хранения очередной строки матрицы. Адрес этой строки будет записываться в `a[i]`.

```
for ( i=0; i<N; i++)
    a[ i]=new float (M) ;
```

После этого определён массив  $N$  указателей, каждый из которых адресует массив из  $M$  чисел (в данном случае вещественных типа `float`). Фактически создана динамическая матрица размера  $N \times M$ . Обращение к элементу динамической матрицы идет так же, как и к элементу статической матрицы. Для того, чтобы обратиться к элементу  $a_{i,j}$  в программе на C++ необходимо указать ее имя, и в квадратных скобках номер строки и столбца (`a[i][j]`).

## 6.3 Обработка матриц в C(C++)

Рассмотрим основные операции, выполняемые над матрицами (статическими и динамическими) при решении задач.

Матрицы, как и массивы, нужно вводить (выводить) поэлементно. Блок-схема ввода элементов матрицы `x[n][m]` изображена на рис. 6.3.

---

<sup>2</sup>В данном случае массив указателей на `float`.

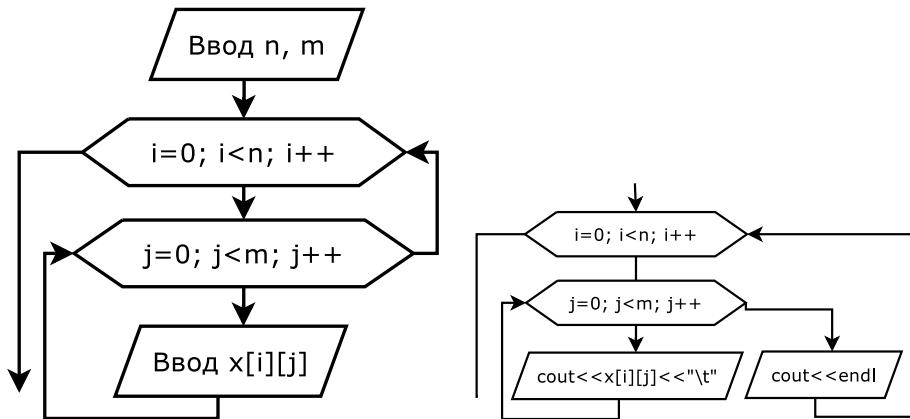


Рис. 6.3: Ввод элементов матрицы

Рис. 6.4: Блок-схема построчного вывода матрицы

При выводе матрицы элементы располагаются построчно, например:

6	-9	7	13
5	8	3	8
3	7	88	33
55	77	88	37

Алгоритм построчного вывода элементов матрицы приведен на рис. 6.4.

Ниже приведен текст программы на С++ ввода-вывода статической матрицы.

```
#include <iostream>
using namespace std;
int main()
{
    int i, j, N, M, a[20][20];
    cout << "N=";
    cin >> N; // Ввод количества строк
    cout << "M=";
    cin >> M; // Ввод количества столбцов
    cout << "Ввод матрицы A" << endl;
    for (i=0; i<N; i++) // Цикл по переменной i, перебираем строки матрицы
        for (j=0; j<M; j++) // Цикл по переменной j, перебираем элементы внутри строки
            cin >> a[i][j]; // Ввод очередного элемента матрицы
    cout << "Вывод матрицы A" << endl;
    for (i=0; i<N; i++) // Цикл по переменной i, перебираем строки матрицы
    {
        for (j=0; j<M; j++) // Цикл по переменной j, перебираем строки матрицы
            cout << a[i][j] << "\t"; // Вывод очередного элемента матрицы
        cout << endl; // По окончанию вывода всех элементов строки — переход на новую строку.
    }
}
```

Цикл для построчного вывода матрицы можно записать и так.

```
for (i=0; i<N; cout << endl, i++)
    for (j=0; j<M; j++)
        cout << a[i][j] << "\t";
```

При вводе матрицы элементы каждой строки можно разделять пробелами, символами табуляции или **Enter**<sup>3</sup>. Ниже представлены результаты работы программы.

```
N=4
M=5
Ввод матрицы A
1 2 3
5 4
3 6 7 8 9
1 2 3 4 5 6 7 8 9 0
Вывод матрицы A
1      2      3      5      4
3      6      7      8      9
1      2      3      4      5
6      7      8      9      0
```

Далее на примерах решения практических задач будут рассмотрены основные алгоритмы обработки матриц и их реализация в C++. Перед этим, давайте, вспомним некоторые свойства матриц (рис. 6.5):

- если номер строки элемента совпадает с номером столбца ( $i = j$ ), это означает что элемент лежит на главной диагонали матрицы;
- если номер строки превышает номер столбца ( $i > j$ ), то элемент находится ниже главной диагонали;
- если номер столбца больше номера строки ( $i < j$ ), то элемент находится выше главной диагонали.
- элемент лежит на побочной диагонали, если его индексы удовлетворяют равенству  $i + j = n - 1$  ;
- неравенство  $i + j < n - 1$  характерно для элемента находящегося выше побочной диагонали;
- соответственно, элементу лежащему ниже побочной диагонали соответствует выражение  $i + j > n - 1$ .

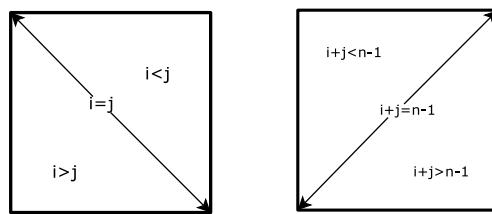


Рис. 6.5: Свойства элементов матрицы

**Задача 6.1.** Найти сумму элементов матрицы, лежащих выше главной диагонали.

---

<sup>3</sup>Элементы каждой строки можно разделять пробелами или символами табуляции, а в конце строки нажимать **Enter**.

Алгоритм решения данной задачи (рис. 6.6) построен следующим образом: обнуляется ячейка для накапливания суммы (переменная  $s$ ). Затем с помощью двух циклов (первый по строкам, второй по столбцам) просматривается каждый элемент матрицы, но суммирование происходит только в том случае если, этот элемент находится выше главной диагонали (при выполнении условия  $i < j$ ).

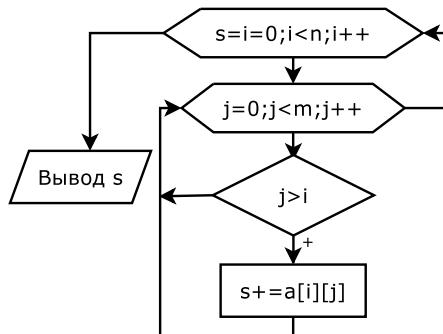


Рис. 6.6: Блок-схема задачи 6.1 (алгоритм 1).

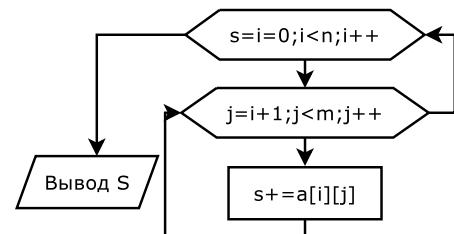


Рис. 6.7: Блок-схема задачи 6.1 (алгоритм 2).

Текст программы:

```

#include <iostream>
using namespace std;
int main()
{
    int s, i, j, n, m, a[20][20];
    cout << "N=" ;
    cin >> n;
    cout << "M=" ;
    cin >> m;
    cout << "Ввод матрицы A" << endl;
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            cin >> a[i][j];
    for (s=i=0; i<n; i++)
        for (j=0; j<m; j++)
            //Если элемент лежит выше главной диагонали, то наращиваем сумму.
            if (j>i) s+=a[i][j];
    cout << "S=" << s << endl;
}
  
```

На рис. 6.7 изображён ещё один вариант решения данной задачи. В нем проверка условия  $i < j$  не выполняется, но, тем не менее, в нем так же суммируются элементы матрицы, находящиеся выше главной диагонали. В нулевой строке данной матрицы необходимо сложить все элементы, начиная с первого. Во первой — все, начиная со второго, в  $i$ -й строке процесс начнётся с  $(i+1)$ -го элемента и так далее. Таким образом, внешний цикл работает от 0 до  $N - 1$ , а второй от  $i+1$  до  $M$ . Авторы надеются, что читатель самостоятельно составит программу, соответствующую описанному алгоритму.

**Задача 6.2.** Вычислить количество положительных элементов квадратной матрицы, расположенных по ее периметру и на диагоналях. Напомним, что в квадратной матрице число строк равно числу столбцов.

Прежде чем приступить к решению задачи рассмотрим рисунок 6.8, на котором изображена схема квадратных матриц различной размерности.

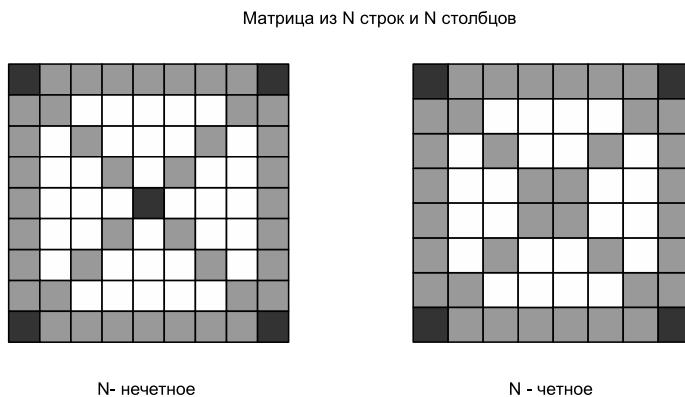


Рис. 6.8: Рисунок к задаче 6.2.

Из условия задачи понятно, что не нужно рассматривать все элементы заданной матрицы. Достаточно просмотреть первую и последнюю строки, первый и последний столбцы, а так же диагонали. Все эти элементы отмечены на схеме, причём чёрным цветом выделены элементы, обращение к которым может произойти дважды. Например, элемент с номером  $(0, 0)$  принадлежит как к нулевой строке, так и к нулевому столбцу, а элемент с номером  $(N - 1, N - 1)$  находится в последней строке и последнем столбце одновременно. Кроме того, если  $N$  – число нечётное (на рисунке 6.8 эта матрица расположена слева), то существует элемент с номером  $(N/2, N/2)$ , который находится на пересечении главной и побочной диагоналей. При нечётном значении  $N$  (матрица справа на рис. 6.8) диагонали не пересекаются.

Итак, разобрав подробно постановку задачи, рассмотрим алгоритм ее решения. Для обращения к элементам главной диагонали вспомним, что номера строк этих элементов всегда равны номерам столбцов. Поэтому, если параметр  $i$  изменяется циклически от 0 до  $N - 1$ , то  $A[i][i]$  – элемент главной диагонали. Воспользовавшись свойством, характерным для элементов побочной диагонали получим:

$i + j + 1 = N \rightarrow j = N - i - 1$ ,  
следовательно, для  $i = 0, 1, \dots, N - 1$  элемент  $A[i][N-i-1]$  – элемент побочной диагонали. Элементы, находящиеся по периметру матрицы записываются сле-

дующим образом:  $A[0][i]$  — нулевая строка,  $A[N-1][i]$  — последняя строка и соответственно  $A[i][0]$  — нулевой столбец,  $A[i][N-1]$  — последний столбец.

Текст программы решения задачи с пояснениями приведён далее.

```
#include <iostream>
using namespace std;
int main()
{
    int k, i, j, N, a[20][20];
    cout<<"N=";
    cin>>N;
    cout<<"Ввод матрицы A"<<endl;
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            cin>>a[i][j];
    //k — количество положительных элементов матрицы,
    //расположенных по ее периметру и на диагоналях.
    for (i=k=0; i<N; i++)
    {
        if (a[i][i]>0) k++; //Элемент лежит на главной диагонали.
        if (a[i][N-i-1]>0) k++; //Элемент лежит на побочной диагонали.
    }
    for (i=1; i<N-1; i++)
    {
        if (a[0][i]>0) k++; //Элемент находится в нулевой строке.
        if (a[N-1][i]>0) k++; //Элемент находится в последней строке.
        if (a[i][0]>0) k++; //Элемент находится в нулевом столбце.
        if (a[i][N-1]>0) k++; //Элемент находится в последнем столбце.
    }
    //Элемент, находящийся на пересечении диагоналей подсчитан дважды,
    //надо уменьшить вычисленное значение k на один.
    if ((N%2!=0)&&(a[N/2][N/2]>0)) k--;
    cout<<"k="<<k<<endl;
}
```

**Задача 6.3.** Проверить, является ли заданная квадратная матрица единичной.

Единичной называют матрицу, у которой элементы главной диагонали — единицы, а все остальные — нули. Например,

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Решать задачу будем так. Предположим, что матрица единичная и попытаемся доказать обратное. Если окажется, что хотя бы один диагональный элемент не равен единице, или любой из элементов вне диагонали не равен нулю, то матрица единичной не является. Воспользовавшись логическими операциями языка С(С++) все эти условия можно соединить в одно и составить программу, текст которой приведён ниже.

```
#include <iostream>
using namespace std;
int main()
{
    int pr, i, j, N, **a;
    cout<<"N=";
    //Ввод размерности матрицы
    cin>>N;
```

```

a=new int *[N]; //Создаём квадратную динамическую матрицу
for (i=0; i<N; i++)
    a[i]=new int [N];
cout<<"Ввод элементов матрицы A "<<endl;
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        cin>>a[i][j];
//Предположим, что матрица единичная и присвоим переменной pr значение 1 (истина).
//Если значение этой переменной при выходе из цикла не изменится, это будет означать,
//что матрица действительно единичная.
for (pr=1, i=0; i<N; i++)
    for (j=0; j<N; j++)
        if (((i==j)&&(a[i][j]!=1)) || ((i!=j)&&(a[i][j]!=0)))
            //Если элемент лежит на главной диагонали и не равен единице или элемент лежит вне
            //главной диагонали и не равен нулю, то
        {
            pr=0; //Переменной pr присвоить значение 0 (ложь), это будет означать,
            break; //что матрица единичной не является, и выйти из цикла.
        }
//Проверка значения переменной pr и вывод соответствующего сообщения.
if (pr) cout<<"Единичная матрица\n";
else cout<<"Матрица не является единичной\n";
}

```

**Задача 6.4.** Преобразовать исходную матрицу так, чтобы нулевой элемент каждой строки был заменён средним арифметическим элементов этой строки.

Для решения данной задачи необходимо найти в каждой строке сумму элементов, которую разделить на их количество. Полученный результат записать в нулевой элемент соответствующей строки. Текст программы приведён далее.

```

#include <iostream>
using namespace std;
int main()
{
    int i, j, N, M;
    double S, **a;
    cout<<"N="; //Ввод размерности матрицы.
    cin>>N;
    cout<<"M=";
    cin>>M;
    a=new double *[N]; //Создаём динамическую матрицу
    for (i=0; i<N; i++)
        a[i]=new double [M];
    cout<<"Ввод элементов матрицы A "<<endl;
    for (i=0; i<N; i++)
        for (j=0; j<M; j++)
            cin>>a[i][j];
    //Цикл по i завершается записью среднего значения в нулевой элемент строки и наращиванием i.
    for (i=0; i<N; a[i][0]=S/M, i++)
        for (S=j=0; j<M; j++) //Вычисление суммы элементов строки.
            S+=a[i][j];
    cout<<"Преобразованная матрица A "<<endl;
    for (i=0; i<N; cout<<endl, i++)
        for (j=0; j<M; j++)
            cout<<a[i][j]<<"\t";
}

```

**Задача 6.5.** Задана матрица  $A(n, m)$ . Поменять местами её максимальный и минимальный элементы.

Алгоритм решения этой задачи следующий: находим максимальный элемент матрицы (`max`) и его индексы (`imax`, `jmax`), а также минимальный (`min`) и его индексы (`imin`, `jmin`). После чего элементы  $A[i_{max}][j_{max}]$  и  $A[i_{min}][j_{min}]$

поменяем местами. Для поиска максимального элемента и его индексов в переменную `max` запишем `A[0][0]`, в переменные `imax`, `jmax`, (номер строки и столбца, где находится максимальный элемент) запишем 0. Затем в двойном цикле (цикл по переменной  $i$  — по строкам, цикл по переменной  $j$  — по столбцам) перебираем все элементы, и каждый из них сравниваем с максимальным (со значением переменной `max`). Если текущий элемент массива оказывается больше максимального, то его переписываем в переменную `max`, а в переменную `imax` — текущее значение индекса  $i$ , в переменную `jmax` — текущее значение  $j$ . Поиск минимального элемента матрицы аналогичен, и отличается только знаком операции сравнения. Далее представлен текст программы решения задачи 6.5.

```
#include <iostream>
using namespace std;
int main()
{
    int i, j, imax, jmax, imin, jmin, N, M;
    double min, max, b, **a;
    cout << "N="; // Ввод размеров матрицы.
    cin >> N;
    cout << "M=";
    cin >> M;
    a = new double *[N]; // Создаём динамическую матрицу
    for (i = 0; i < N; i++)
        a[i] = new double [M];
    cout << "Ввод элементов матрицы A" << endl;
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            cin >> a[i][j];
    // Двойной цикл для поиска максимального, минимального элементов и их индексов.
    for (max = min = a[0][0], imax = jmax = imin = jmin = i = 0; i < N; i++)
        for (j = 0; j < M; j++)
    {
        if (a[i][j] > max) {max = a[i][j]; imax = i; jmax = j;}
        if (a[i][j] < min) {min = a[i][j]; imin = i; jmin = j;}
    }
    // Обмен двух элементов матрицы.
    b = a[imax][jmax];
    a[imax][jmax] = a[imin][jmin];
    a[imin][jmin] = b;
    // Вывод преобразованной матрицы.
    cout << "Преобразованная матрица A" << endl;
    for (i = 0; i < N; cout << endl, i++)
        for (j = 0; j < M; j++)
            cout << a[i][j] << "\t";
}
```

**Задача 6.6.** Преобразовать матрицу  $A(m, n)$  так, чтобы строки с нечётными индексами были упорядочены по убыванию, с чётными — по возрастанию.

В связи с нумерацией строк в C++ с нуля, необходимо помнить, что нулевая, вторая, четвёртая строки упорядочиваются по убыванию, а первая, третья, пятая и т.д. — по возрастанию. Алгоритм решения этой задачи сводится к тому, что уже известный нам по предыдущей главе алгоритм упорядочивания элементов в массиве выполняется для каждой строки матрицы. Блок-схема приведена на рис. 6.9. Текст программы с комментариями приведён ниже.

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

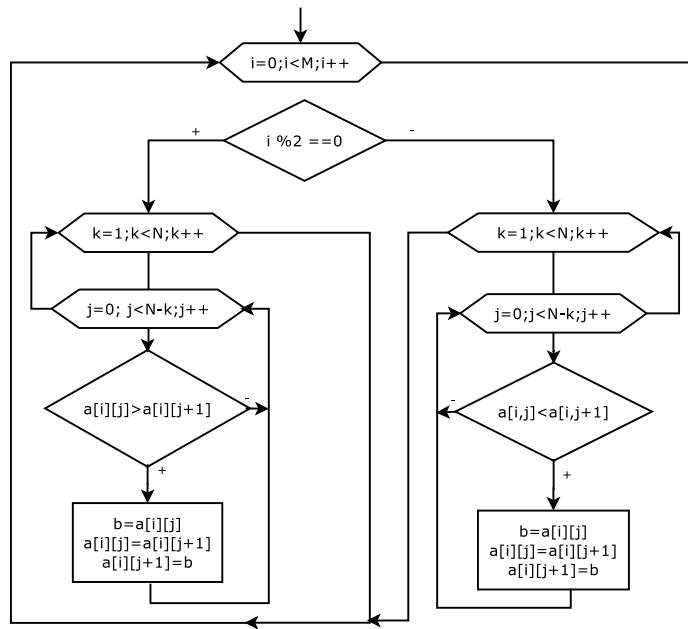


Рис. 6.9: Блок-схема алгоритма задачи 6.6.

```

int i, j, N, M;
double b, **a;
cout << "M="; //Ввод размеров матрицы.
cin >> M;
cout << "N=";
cin >> N;
a = new double * [N]; //Создаём динамическую матрицу
for (i = 0; i < N; i++)
    a[i] = new double [M];
cout << "Ввод элементов матрицы A" << endl;
for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
        cin >> a[i][j];
for (i = 0; i < M; i++) //Цикл по i — для перебора строк матрицы.
    if (i % 2 == 0) //Если строка четна, то
    {
        //упорядочиваем элементы строки по возрастанию,
        for (k = 1; k < N; k++)
            for (j = 0; j < N - k; j++)
                if (a[i][j] > a[i][j + 1])
                {
                    b = a[i][j];
                    a[i][j] = a[i][j + 1];
                    a[i][j + 1] = b;
                }
    }
    else //иначе, нечетные строки, упорядочиваем по убыванию.
        for (k = 1; k < N; k++)
            for (j = 0; j < N - k; j++)
                if (a[i][j] < a[i][j + 1])

```

```

{
    b=a[i][j];
    a[i][j]=a[i][j+1];
    a[i][j+1]=b;
}
cout<<"Преобразованная матрица A"<<endl; //Вывод преобразованной матрицы.
for(i=0;i<M;cout<<endl,i++)
    for(j=0;j<N;j++)
        cout<<a[i][j]<<"\t";
}

```

**Задача 6.7.** Поменять местами элементы главной и побочной диагонали матрицы  $A(k, k)$ .

Алгоритм решения задачи следующий: перебираем все строки матрицы (цикл по переменной  $i$  от 0 до  $k - 1$  в тексте программы), и в каждой строке меняем местами элементы, расположенные на главной и побочной диагоналях (в  $i$ -й строке надо поменять местами элементы  $A[i][i]$  и  $A[i][k-i-1]$ ). Текст программы с комментариями приведён далее.

```

#include <iostream>
using namespace std;
int main()
{
    int i,j,k;
    double b,*a;
    cout<<"k="; //Ввод размера матрицы.
    cin>>k;
    a=new double *[k]; //Создаём динамическую матрицу
    for(i=0;i<k;i++)
        a[i]=new double [k];
    cout<<"Ввод элементов матрицы A"<<endl;
    for(i=0;i<k;i++)
        for(j=0;j<k;j++)
            cin>>a[i][j];
    for(i=0;i<k;i++) //Цикл по строкам.
    {
        //В каждой строке обмен между элементами, лежащими на главной и побочной диагоналях.
        b=a[i][i];
        a[i][i]=a[i][k-1-i];
        a[i][k-1-i]=b;
    }
    cout<<"Преобразованная матрица A"<<endl; //Вывод преобразованной матрицы.
    for(i=0;i<k;cout<<endl,i++)
        for(j=0;j<k;j++)
            cout<<a[i][j]<<"\t";
}

```

**Задача 6.8.** Заполнить матрицу  $A(6, 6)$  числами 1 до 36 следующим образом:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 12 & 11 & 10 & 9 & 8 & 7 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 24 & 23 & 22 & 21 & 20 & 19 \\ 25 & 26 & 27 & 28 & 29 & 30 \\ 36 & 35 & 34 & 33 & 32 & 31 \end{pmatrix}$$

Последовательно построчно заполняем матрицу возрастающей арифметической последовательностью 1, 2, 3, ..., 36. Четные строки заполняем от нулевого элемента к последнему, а нечётные — от последнего к нулевому. Текст программы приведён далее.

```
#include <iostream>
```

```

using namespace std;
int main(int argc, char **argv)
{
    int **a, n=6, k=0, i, j;
    a=new int *[n]; //Выделяем память для хранения матрицы
    for(i=0; i<n; i++)
        a[i]=new int [n];
    for(i=0; i<n; i++) //Перебираем все строки матрицы.
        if (i%2==0) //Строки с чётными номерами заполняем возрастающей последовательностью
            for(j=0; j<n; j++) //чисел слева направо
                a[i][j]=++k;
        else //Строки с нечётными номерами заполняем возрастающей последовательностью чисел
            for(j=n-1; j>=0; j--) //справа налево
                a[i][j]=++k;
    cout<<"Вывод матрицы A "<<endl;
    for(i=0; i<n; cout<<endl, i++)
        for(j=0; j<n; j++)
            cout<<a[i][j]<<"\t";
    return 0;
}

```

## 6.4 Решение некоторых задач линейной алгебры

В этом параграфе рассмотрим использование матриц при решении таких задач линейной алгебры, как сложение, вычитание и умножение матриц, решение систем линейных алгебраических уравнений, вычисление определителя и обратной матрицы. В процессе решения подобных задач будут написаны универсальные функции, которые можно использовать при решении матричной алгебры.

**Задача 6.9.** Заданы четыре матрицы вещественных чисел  $A(N, M)$ ,  $B(N, M)$ ,  $C(M, N)$ ,  $D(M, N)$ . Вычислить матрицу  $C = ((A + B)(C - D))^2$ .

Суммой (разностью) матриц одинаковой размерности  $A$  и  $B$  называется матрица  $C$ , элементы которой получаются сложением  $C_{i,j} = A_{i,j} + B_{i,j}$  (вычитанием  $C_{i,j} = A_{i,j} - B_{i,j}$ ) соответствующих элементов исходных матриц.

Напомним алгоритм умножения матриц на примере двух матриц

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \cdot \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \\ b_{2,0} & b_{2,1} \end{pmatrix}$$

Воспользовавшись правилом «строка на столбец», получим матрицу:

$$\begin{pmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \\ c_{2,0} & c_{2,1} \end{pmatrix} = \begin{pmatrix} a_{0,0} \cdot b_{0,0} + a_{0,1} \cdot b_{1,0} + a_{0,2} \cdot b_{2,0} & a_{0,0} \cdot b_{0,1} + a_{0,1} \cdot b_{1,1} + a_{0,2} \cdot b_{2,1} \\ a_{1,0} \cdot b_{0,0} + a_{1,1} \cdot b_{1,0} + a_{1,2} \cdot b_{2,0} & a_{1,0} \cdot b_{0,1} + a_{1,1} \cdot b_{1,1} + a_{1,2} \cdot b_{2,1} \\ a_{2,0} \cdot b_{0,0} + a_{2,1} \cdot b_{1,0} + a_{2,2} \cdot b_{2,0} & a_{2,0} \cdot b_{0,1} + a_{2,1} \cdot b_{1,1} + a_{2,2} \cdot b_{2,1} \end{pmatrix}$$

Произведением матриц  $A(N, M)$  и  $B(M, L)$  является матрица  $C(N, L)$ , каждый элемент которой  $C_{i,j}$  вычисляется по формуле:

$$C_{i,j} = \sum_{k=0}^{M-1} A_{i,k} B_{k,j}, \text{ где } i = 0, N - 1 \text{ и } j = 0, L - 1.$$

Операция умножения имеет смысл только в том случае, если количество строк левой матрицы совпадает с количеством столбцов правой. Кроме того,  $A \cdot B \neq B \cdot A$ .

При решении задачи будем использовать динамические матрицы и двойные указатели, напишем следующие функции

- `float **sum_m(float **A, float **B, int N, int M)` — функция формирует матрицу, которая является суммой двух матриц. Здесь A, B — указатели на исходные матрицы, N, M — количество строк и столбцов матриц, функция возвращает указатель на сформированную матрицу, которая является суммой двух матриц  $A$  и  $B$ .
- `float **minus_m(float **A, float **B, int N, int M)` — функция формирует матрицу, которая является разностью двух матриц. Здесь A, B — указатели на исходные матрицы, N, M — количество строк и столбцов матриц, функция возвращает указатель на сформированную матрицу, которая является разностью двух матриц  $A$  и  $B$ .
- `float **product_m(float **A, float **B, int N, int M, int L)` — функция формирует матрицу, которая является произведением двух матриц. Здесь A, B — указатели на исходные матрицы. Матрица  $A$  имеет  $N$  строк и  $M$  столбцов, матрица  $B$  имеет  $M$  строка и  $L$  столбцов, функция возвращает указатель на сформированную матрицу, которая является произведением двух матриц  $A$  и  $B$ .
- `float **create_m(int N, int M)` — функция создаёт матрицу, в которой будет  $N$  строк и  $M$  столбцов, осуществляет ввод элементов матрицы, функция возвращает указатель на сформированную матрицу.
- `void output_m(float **A, int N, int M)` — функция построчного вывода на экран матрицы  $A$ , которая имеет  $N$  строк и  $M$  столбцов.

Далее приведен текст программы с комментариями.

```
#include <iostream>
using namespace std;
//функция вычисления суммы двух матриц.
float **sum_m(float **A, float **B, int N, int M)
{
    int i, j;
    float **temp; //указатель для хранения результирующей матрицы
    temp=new float *[N]; //выделение памяти для хранения результирующей матрицы
    for(i=0;i<N; i++)
        temp[i]=new float [M];
    for(i=0; i<N; i++) //Вычисляем сумму двух матриц
        for(j=0; j<M; j++)
            temp[i][j]=A[i][j]+B[i][j];
    return temp; //Возвращаем матрицу как двойной указатель
}
//функция вычисления разности двух матриц.
float **minus_m(float **A, float **B, int N, int M)
{
    int i, j;
    float **temp; //указатель для хранения результирующей матрицы
    temp=new float *[N]; //выделение памяти для хранения результирующей матрицы
    for(i=0; i<N; i++)
        temp[i]=new float [M];
    for(i=0; i<N; i++) //Вычисляем разность двух матриц
        for(j=0; j<M; j++)
            temp[i][j]=A[i][j]-B[i][j];
    return temp; //Возвращаем матрицу как двойной указатель
}
//функция вычисления произведения двух матриц.
float **product_m(float **A, float **B, int N, int M, int L)
{
    int i, j, k;
    float **temp; //указатель для хранения результирующей матрицы
```

```

temp=new float *[N]; //выделение памяти для хранения результирующей матрицы
for(i=0;i<N; i++)
    temp[i]=new float [L];
//Вычисляем произведение двух матриц последовательно формируя все элементы матрицы
for(i=0;i<N; i++)
    for(j=0;j<L; j++)
        //Элемент с индексами  $i, j$  — скалярное произведение  $i$ -й строки матрицы A
        //и  $j$ -го столбца матрицы B
        for(k=0;k<M; k++)
            temp[i][j]+=A[i][k]*B[k][j];
return temp; //Возвращаем матрицу как двойной указатель
}
//функция создаёт динамическую матрицу вещественных чисел размерности N на M,
//в этой же функции осуществляется и ввод элементов матрицы
float **create_m(int N, int M)
{
    int i, j;
    float **temp;
    temp=new float *[N];
    for(i=0;i<N; i++)
        temp[i]=new float [M];
    cout<<"Ввод матрицы\n";
    for(i=0;i<N; i++)
        for(j=0;j<M; j++)
            cin>>temp[i][j];
    return temp;
}
//функция осуществляет построчный вывод матрицы A(N,M)
void output_m(float **A, int N, int M)
{
    int i, j;
    //Цикл по строкам. По окончанию вывода всех элементов строки — переход на новую строку.
    for(i=0;i<N; cout<<endl, i++)
        for(j=0;j<M; j++) //Цикл по переменной j, в котором перебираем строки матрицы
            cout<<A[i][j]<<"\t"; //Вывод очередного элемента матрицы и символа табуляции.
}
int main(int argc, char **argv)
{
    float **A, **B, **C, **D, **result; //указатели для хранения исходных и
                                         //результатирующие матрицы
    int N, M;
    cout<<"N="; cin>>N; //Ввод размерностей матрицы
    cout<<"M="; cin>>M;
    //Выделение памяти и ввод матриц A, B, C, D, обращаясь к функции create_m.
    A=create_m(N, M);
    B=create_m(N, M);
    C=create_m(M, N);
    D=create_m(M, N);
    //Вычисление результатирующей матрицы.
    result=product_m(product_m(sum_m(A, B, N, M), minus_m(C, D, M, N), N, M, N), product_m
        (sum_m(A, B, N, M), minus_m(C, D, M, N), N, M, N), N, N, N);
    output_m(result, N, N); //Вывод результатирующей матрицы.
    return 0;
}

```

Далее без комментариев приведена программа решения задачи 6.9 с помощью динамических матриц и обычных указателей<sup>4</sup>. Рекомендуем читателям самостоятельно разобраться с этой версией программы.

```
#include <iostream>
```

<sup>4</sup> Обращаем внимание читателя, что при использовании одинарных указателей обращение к элементам матрицы происходит быстрее. При обработке матриц большой размерности (более 1000000 элементов) имеет смысл использовать именно одинарные указатели для хранения и обработки матриц. Это позволит ускорить работу программ на 10-15%.

```

using namespace std;
float *sum_m(float *A, float *B, int N, int M)
{
    int i, j;
    float *temp;
    temp=new float [N*M];
    for(i=0;i<N; i++)
        for(j=0;j<M; j++)
            temp[i*M+j]=A[i*M+j]+B[i*M+j];
    return temp;
}
float *minus_m(float *A, float *B, int N, int M)
{int i, j;
    float *temp;
    temp=new float [N*M];
    for(i=0;i<N; i++)
        for(j=0;j<M; j++)
            temp[i*M+j]=A[i*M+j]-B[i*M+j];
    return temp;
}
float *product_m(float *A, float *B, int N, int M, int L)
{
    int i, j, k;
    float *temp;
    temp=new float [N*L];
    for(i=0;i<N; i++)
        for(j=0;j<L; j++)
            for(temp[i*L+j]=k=0;k<M; k++)
                temp[i*L+j]+=A[i*M+k]*B[k*L+j];
    return temp;
}
float *create_m(int N, int M)
{
    int i, j;
    float *temp;
    temp=new float [N*M];
    cout<<"Ввод матрицы\n";
    for(i=0;i<N; i++)
        for(j=0;j<M; j++)
            cin>>temp[i*M+j];
    return temp;
}
void output_m(float *A, int N, int M)
{
    int i, j;
    for(i=0;i<N; cout<<endl, i++)
        for(j=0;j<M; j++)
            cout<<A[i*M+j]<<"\t";
}
int main(int argc, char **argv)
{
    float *A, *B, *C, *D,* result;
    int N,M;
    cout<<"N=" ; cin>>N;
    cout<<"M=" ; cin>>M;
    A=create_m(N,M);
    B=create_m(N,M);
    C=create_m(M,N);
    D=create_m(M,N);
    result=product_m(product_m(sum_m(A,B,N,M) , minus_m(C,D,M,N) ,N,M,N) ,
        product_m(sum_m(A,B,N,M) , minus_m(C,D,M,N) ,N,M,N) ,N,N,N);
    output_m(result ,N,N);
    return 0;
}

```

**Задача 6.10.** Решить систему линейных алгебраических уравнений.

При решении этой задачи напишем универсальную функцию решения системы линейных алгебраических уравнений методом Гаусса, а в функции `main()` просто вызовем эту функцию. Вспомним метод Гаусса.

Пусть дана система линейных алгебраических уравнений (СЛАУ) с  $n$  неизвестными

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + \dots + a_{0n-1}x_{n-1} &= b_0, \\ a_{10}x_0 + a_{11}x_1 + \dots + a_{1n-1}x_{n-1} &= b_1, \\ \dots &\dots \\ a_{n-10}x_0 + a_{n-11}x_1 + \dots + a_{n-1n-1}x_{n-1} &= b_{n-1} \end{cases} \quad (6.1)$$

Обозначим через  $A = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} \\ a_{10} & a_{11} & \dots & a_{1n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & \dots & a_{n-1n-1} \end{pmatrix}$  матрицу коэффициентов системы (6.1), через  $b = \begin{pmatrix} b_0 \\ b_1 \\ \dots \\ b_{n-1} \end{pmatrix}$  — столбец ее свободных членов, и через  $x =$

$\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{pmatrix}$  — столбец из неизвестных (искомый вектор). Тогда система (6.1) может быть записана в виде матричного уравнения  $Ax = b$ .

Наиболее распространенным приёмом решения систем линейных уравнений является алгоритм последовательного исключения неизвестных — *метод Гаусса*.

При решении систем линейных алгебраических уравнений этим методом все возможные преобразования производят не над уравнениями системы (6.1), а над так называемой *расширенной матрицей* системы, которая получается путём добавления к основной матрице  $A$  столбца свободных членов  $b$ .

Первый этап решения системы уравнений, называемый *прямым ходом метода Гаусса*, заключается в приведении расширенной матрицы (6.2) к *треугольному виду*. Это означает, что все элементы матрицы (6.2) ниже главной диагонали должны быть равны нулю.

$$A' = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} & b_0 \\ a_{10} & a_{11} & \dots & a_{1n-1} & b_1 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & \dots & a_{n-1n-1} & b_{n-1} \end{pmatrix} \quad (6.2)$$

На первом этапе необходимо обнулить элементы 0-го столбца расширенной матрицы

$$A' = \begin{pmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0n-1} & b_0 \\ 0 & a_{11} & a_{12} & \dots & a_{1n-1} & b_1 \\ 0 & 0 & a_{22} & \dots & a_{2n-1} & b_2 \\ 0 & 0 & 0 & \dots & a_{3n-1} & b_3 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n-1n-1} & b_{n-1} \end{pmatrix} \quad (6.3)$$

Для этого необходимо из каждой строки (начиная со первой) вычесть нулевую, умноженную на некоторое число  $M$ . В общем виде этот процесс можно записать так:

$$\text{1-я строка} = 1\text{-я строка} - M \times 0\text{-я строка}$$

$$\text{2-я строка} = 2\text{-я строка} - M \times 0\text{-я строка}$$

...

$$i\text{-я строка} = i\text{-я строка} - M \times 0\text{-я строка}$$

...

$$n-1\text{-я строка} = n-1\text{-я строка} - M \times 0\text{-я строка}$$

Понятно, что преобразование элементов первой строки будет происходить по формулам:

$$a_{10} = a_{10} - Ma_{00}$$

$$a_{11} = a_{11} - Ma_{01}$$

...

$$a_{1i} = a_{1i} - Ma_{0i}$$

...

$$a_{1n-1} = a_{1n-1} - Ma_{0n-1}$$

$$b_1 = b_1 - Mb_0$$

Так как целью данных преобразований является обнуление первого элемента строки, то  $M$  выбираем из условия:  $a_{10} = a_{10} - Ma_{00} = 0$ . Следовательно,  $M = \frac{a_{10}}{a_{00}}$ .

Элементы второй строки и коэффициент  $M$  можно рассчитать аналогично:

$$a_{20} = a_{20} - Ma_{00}$$

$$a_{21} = a_{21} - Ma_{01}$$

...

$$a_{2i} = a_{2i} - Ma_{0i}$$

...

$$a_{2n-1} = a_{2n-1} - Ma_{0n-1}$$

$$b_2 = b_2 - Mb_0$$

$$a_{20} = a_{20} - Ma_{00} = 0 \Rightarrow M = \frac{a_{20}}{a_{00}}$$

Таким образом, преобразование элементов  $i$ -й строки будет происходить следующим образом:

$$a_{i0} = a_{i0} - Ma_{00}$$

$$a_{i1} = a_{i1} - Ma_{01}$$

...

$$a_{ii} = a_{ii} - Ma_{0i}$$

$$\dots \\ a_{in-1} = a_{in-1} - Ma_{0n-1} \\ b_i = b_i - Mb_0.$$

Коэффициент  $M$  для  $i$ -й строки выбирается из условия  $a_{i0} = a_{i0} - Ma_{00} = 0$  и равен  $M = \frac{a_{i0}}{a_{00}}$ .

После проведения подобных преобразований для всех строк матрица (6.2) примет вид

$$A' = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} & b_0 \\ 0 & a_{11} & \dots & a_{1n-1} & b_1 \\ 0 & a_{21} & \dots & a_{2n-1} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & a_{n-11} & \dots & a_{n-1n-1} & b_{n-1} \end{pmatrix}.$$

Блок-схема обнуления первого столбца матрицы приведена на рис. 6.10.

Очевидно, что если повторить описанный выше алгоритм для следующих столбцов матрицы (6.2), причём начинать преобразовывать первый столбец со второго элемента, второй столбец — с третьего и т.д., то в результате будет получена матрица (6.3). Алгоритм этого процесса изображён на рис. 6.11.

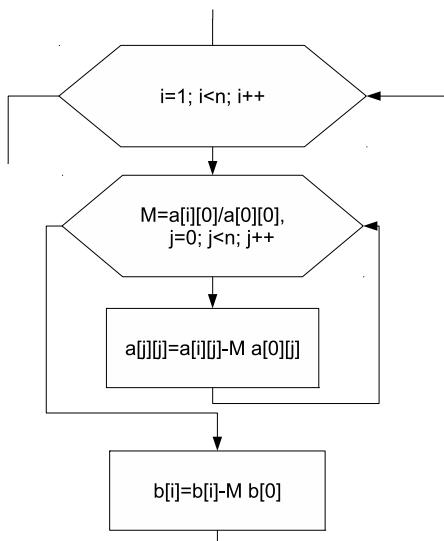


Рис. 6.10: Блок-схема обнуления первого столбца матрицы

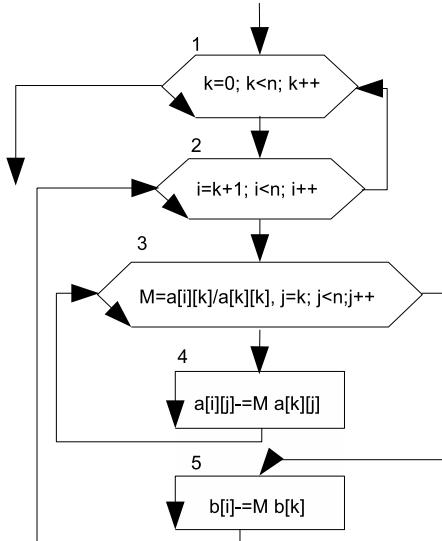


Рис. 6.11: Блок-схема алгоритма преобразования расширенной матрицы к треугольному виду

Заметим, что если в матрице (6.2) на главной диагонали встретится элемент  $a_{kk}$ , равный нулю, то расчёт коэффициента  $M = \frac{a_{ik}}{a_{kk}}$  для  $k$ -й строки будет невозможен. Избежать деления на ноль можно, избавившись от нулевых элементов на главной диагонали. Для этого перед обнулением элементов в  $k$ -м столбце необходимо

димо найти в нем максимальный по модулю элемент (среди расположенных ниже  $a_{k,k}$ , запомнить номер строки, в которой он находится, и поменять ее местами с  $k$ -й. Алгоритм, отображающий эти преобразования, приведён на рис. 6.12.

В результате выполнения прямого хода метода Гаусса матрица (6.2) преобразуется в матрицу (6.3), а система уравнений (6.1) будет иметь следующий вид:

$$\left\{ \begin{array}{ll} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \dots + a_{0n-1}x_{n-1} & = b_0, \\ a_{11}x_1 + a_{21}x_2 + \dots + a_{1n-1}x_{n-1} & = b_1, \\ a_{22}x_2 + \dots + a_{2n-1}x_{n-1} & = b_2, \\ \dots & \\ a_{n-1n-1}x_{n-1} & = b_{n-1} \end{array} \right. \quad (6.4)$$

Решение системы (6.4) называют *обратным ходом метода Гаусса*.

Последнее  $(n-1)$ -е уравнение системы (6.4) имеет вид:  $a_{n-1n-1}x_{n-1} = b_{n-1}$ . Тогда, если  $a_{n-1n-1} \neq 0$ , то  $x_{n-1} = \frac{b_{n-1}}{a_{n-1n-1}}$ . В случае, если  $a_{n-1n-1} = 0$ , и  $b_{n-1} = 0$ , то система (6.4), а следовательно, и система (6.1) имеют бесконечное множество решений.

При  $a_{n-1n-1} = 0$ , и  $b_{n-1} \neq 0$  система (6.4), а значит, и система (6.1) решения не имеет. Предпоследнее  $(n-2)$ -е уравнение системы (6.4) имеет вид  $a_{n-2n-2}x_{n-2} + a_{n-2n-1}x_{n-1} = b_{n-2}$ .

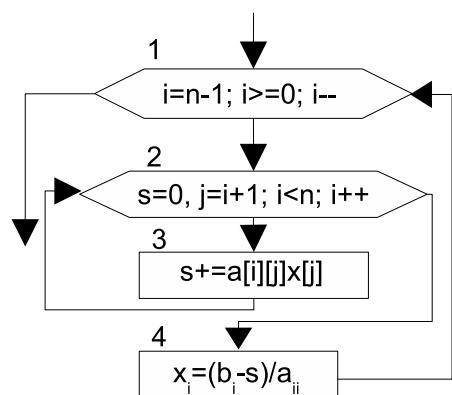
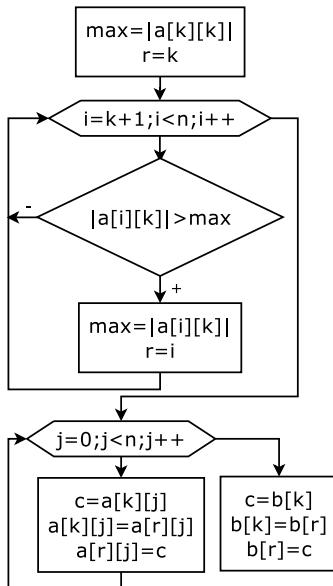


Рис. 6.12: Блок-схема алгоритма перестановки строк расширенной матрицы

Рис. 6.13: Блок-схема алгоритма обратного хода метода Гаусса

Значит,  $x_{n-2} = \frac{b_{n-2} - a_{n-2n-1}x_{n-1}}{a_{n-2n-2}}$ .

Следующее  $(n-3)$ -е уравнение системы (6.4) будет выглядеть так:

$$a_{n-3n-3}x_{n-3} + a_{n-3n-2}x_{n-2} + a_{n-3n-1}x_{n-1} = b_{n-3}.$$

Отсюда имеем

$$x_{n-3} = \frac{b_{n-3} - a_{n-3n-2}x_{n-2} - a_{n-3n-1}x_{n-1}}{a_{n-3n-3}}, x_{n-3} = \frac{b_{n-3} - \sum_{j=n-2}^{n-1} a_{n-3j}x_j}{a_{n-3n-3}}.$$

Таким образом, формула для вычисления  $i$ -го значения  $x$  будет иметь вид:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j}{a_{ii}}.$$

Алгоритм, реализующий обратный ход метода Гаусса, представлен в виде блок-схемы на рис. 6.13.

Объединив блок-схемы, изображенные на рис. 6.11, 6.12 и 6.13, получим общую блок-схему метода Гаусса (рис. 6.14). Блоки 2-6 содержат последовательный ввод данных, где  $n$  — это размерность системы линейных алгебраических уравнений, а сама система задаётся в виде матрицы коэффициентов при неизвестных  $A$  и вектора свободных коэффициентов  $b$ . Блоки 7-18 предусматривают прямой ход метода Гаусса, а блоки 23-27 — обратный. Для вывода результатов предусмотрено несколько блоков вывода. Если результат проверки условий 19 и 20 положительный, то выдается сообщение о том, что система имеет бесконечное множество решений (блок 21). Если условие 19 выполняется, а 20 — нет, то появляется сообщение о том, что система не имеет решений (блок 22). Самые же решения системы уравнений, представленные вектором  $x$ , вычисляются (блоки 23-26) и выводятся экран печать (блок 27) только в случае невыполнения условия.

Теперь алгоритм решения СЛАУ, представленный на рис. 6.14 разобьём на главную функцию `main()` и функцию решения СЛАУ методом Гаусса. В функции `main()` будет находиться ввод исходных данных, обращение к функции `SLAU` решения системы линейных алгебраических уравнений и вывод вектора решения. Функция `SLAU` предназначена для решения системы линейных алгебраических уравнений методом Гаусса.

При написании функции следует учитывать следующее: в методе Гаусса изменяются матрица коэффициентов и вектор правых частей. Поэтому, для того чтобы их не испортить, в функции `SLAU` матрицу коэффициентов и вектор правых частей необходимо скопировать во внутренние переменные, и в функции обрабатывать внутренние переменные-копии.

Функция `SLAU` возвращает значение 0, если решение найдено,  $-1$  — если система имеет бесконечное множество решений,  $-2$  — если система не имеет решений.

Ниже приведено решение задачи 6.10 с подробными комментариями.

```
#include <iostream>
#include <math.h>
using namespace std;
int SLAU(double **matrica_a, int n, double *massiv_b, double *x)
//Функция SLAU возвращает значение типа int: 0, если решение найдено, -1 — если система имеет
//бесконечное
//множество решений, -2 — если система не имеет решений. Формальные параметры функции: n
// — размерность системы,
```

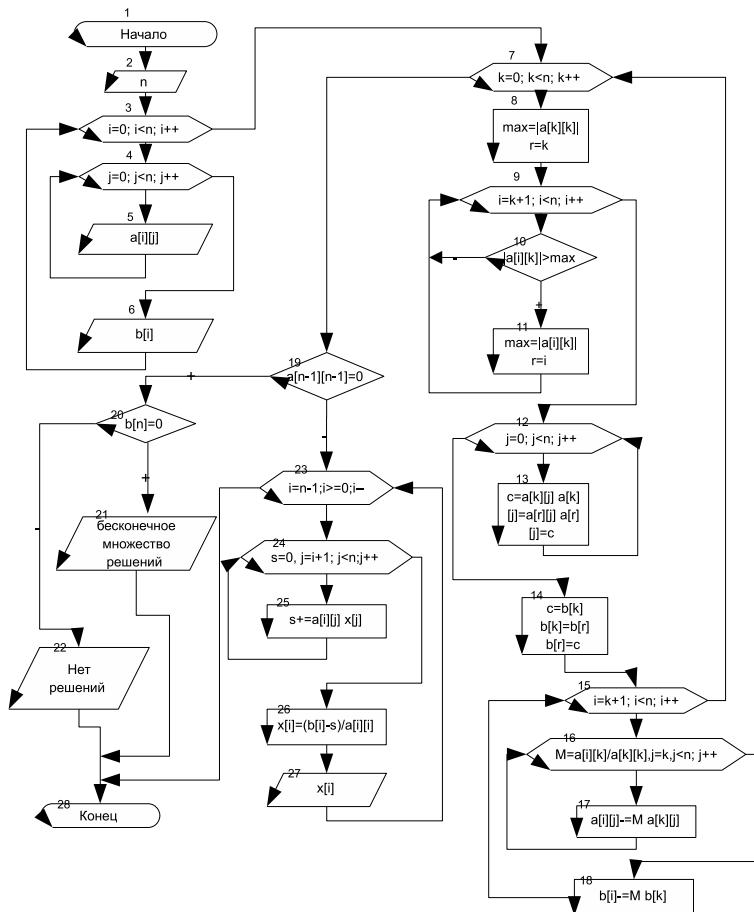


Рис. 6.14: Блок-схема алгоритма решения СЛАУ методом Гаусса

```

//матрица_a — матрица коэффициентов СЛАУ, массив_b — вектор правых частей,
//x — решение СЛАУ, a, b, x передаются как указатели.
{
    int i, j, k, r;
    double c, M, max, s;
    //Матрица a — копия матрицы коэффициентов, массив b — копия вектора правых частей.
    double **a, *b;
    a=new double *[n]; //Выделение памяти для a и b.
    for (i=0;i<n; i++)
        a[i]=new double [n];
    b=new double [n];
    //В a записываем копию матрицы коэффициентов, b копию вектора правых частей.
    for (i=0;i<n; i++)
        for (j=0;j<n; j++)

```

```

a[ i ][ j]=matrica_a[ i ][ j ];
for( i=0; i<n; i++)
    b[ i ]=massiv_b[ i ];
//Прямой ход метода Гаусса: приводим матрицу а (копию матрицы коэффициентов СЛАУ)
//к диагональному виду.
for( k=0; k<n; k++)
{
    //Поиск максимального по модулю элемента в k-м столбце.
    max=fabs( a[ k ][ k ] );
    r=k;
    for( i=k+1; i<n; i++)
        if ( fabs( a[ i ][ k ] )>max )
        {
            max=fabs( a[ i ][ k ] );
            r=i;
        }
    for( j=0; j<n; j++) //Меняем местами k-ю и r-ю (строку, где находится
    { //максимальный по модулю элемент) строки.
        c=a[ k ][ j ];
        a[ k ][ j ]=a[ r ][ j ];
        a[ r ][ j ]=c;
    }
    c=b[ k ];
    b[ k ]=b[ r ];
    b[ r ]=c;
    for( i=k+1; i<n; i++) //Приведение матрицы к диагональному виду.
    {
        for( M=a[ i ][ k ]/a[ k ][ k ], j=k; j<n; j++)
            a[ i ][ j ]-=M*a[ k ][ j ];
        b[ i ]-=M*b[ k ];
    }
}
//Обратный ход метода Гаусса.
if ( a[ n-1 ][ n-1 ]==0 ) //Если последний диагональный элемент равен 0 и
    if ( b[ n-1 ]==0 ) //последний коэффициент вектора свободных членов равен 0,
    return -1; //то система имеет бесконечное множество решений
    else return -2; //последний коэффициент вектора свободных членов не равен 0,
    //система решений не имеет.
else //Последний диагональный элемент не равен 0, начинается обратный ход метода Гаусса.
{
    for( i=n-1; i>=0; i--)
    {
        for( s=0, j=i+1; j<n; j++)
            s+=a[ i ][ j ]*x[ j ];
        x[ i ]=(b[ i ]-s)/a[ i ][ i ];
    }
    return 0;
}
int main()
{
    int result, i, j, N;
    double **a, *b, *x;
    cout<<"N="; //Ввод размерности системы.
    cin>>N;
    a=new double *[N]; //Выделение памяти для матрицы правых частей и вектора свободных
    членов.
    for( i=0; i<N; i++)
        a[ i ]=new double[N];
    b=new double [N];
    x=new double [N];
    cout<<"Ввод матрицы A "<<endl; //Ввод матрицы правых частей
    for( i=0; i<N; i++)
        for( j=0; j<N; j++)
            cin>>a[ i ][ j ];
    cout<<"Ввод вектора B "<<endl; //и вектора свободных членов.
}

```

```

for ( i = 0; i < N; i++)
    cin >> b[ i ];
//Вызов функции решения СЛАУ методом Гаусса. По значению result можно судить сколько
//корней имеет система. Если result=0, то система имеет единственное решение, result=-1 —
//система имеет бесконечное множество решений, result=-2 — система не имеет решений.
result=SLAU(a, N, b, x);
if ( result==0)
{
    //Вывод массива решения.
    cout << "Массив X" << endl;
    for ( i=0; i < N; i++)
        cout << x[ i ] << "\t";
    cout << endl;
}
else if ( result == -1)
    cout << "Бесконечное множество решений\n";
    else if ( result == -2)
        cout << "Нет решений\n";
}

```

**Задача 6.11.** Найти обратную матрицу к квадратной матрице  $A(N, N)$ .

Один из методов вычисления *обратной матрицы* основан на решении систем линейных алгебраических уравнений. Пусть задана некоторая матрица A:

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0n-1} \\ a_{10} & a_{11} & a_{12} & \dots & a_{1n-1} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & a_{n-12} & \dots & a_{n-1n-1} \end{pmatrix} \quad (6.5)$$

Необходимо найти матрицу  $A^{-1}$ , которая является обратной к матрице  $A$ :

$$Y = A^{-1} = \begin{pmatrix} y_{00} & y_{01} & y_{02} & \dots & y_{0n-1} \\ y_{10} & y_{11} & y_{12} & \dots & y_{1n-1} \\ \dots & \dots & \dots & \dots & \dots \\ y_{n-10} & y_{n-11} & y_{n-12} & \dots & y_{n-1n-1} \end{pmatrix} \quad (6.6)$$

Матрица (6.6) будет обратной к матрице (6.5), если выполняется соотношение  $A \cdot A^{-1} = E$ , где  $E$  — это единичная матрица, или более подробно:

$$\begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} \\ a_{10} & a_{11} & \dots & a_{1n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & \dots & a_{n-1n-1} \end{pmatrix} \begin{pmatrix} y_{00} & y_{01} & \dots & y_{0n-1} \\ y_{10} & y_{11} & \dots & y_{1n-1} \\ \dots & \dots & \dots & \dots \\ y_{n-10} & y_{n-11} & \dots & y_{n-1n-1} \end{pmatrix} = E \quad (6.7)$$

Результат перемножения матриц из соотношения (6.7) можно представить по-элементно в виде  $n$  систем линейных уравнений. Умножение матрицы (6.5) на нулевой столбец матрицы (6.6) даст нулевой столбец единичной матрицы:

При умножении матрицы  $A$  на второй столбец обратной матрицы получается следующая система линейных алгебраических уравнений.

$$\left\{ \begin{array}{ll} a_{00}y_{01} + a_{01}y_{11} + \dots + a_{0n-1}y_{n-11} & = 0, \\ a_{10}y_{01} + a_{11}y_{11} + \dots + a_{1n-1}y_{n-11} & = 1, \\ \dots & \dots \\ a_{i0}y_{01} + a_{i1}y_{11} + \dots + a_{in-1}y_{n-11} & = 0, \\ \dots & \dots \\ a_{n-10}y_{01} + a_{n-11}y_{11} + \dots + a_{n-1n-1}y_{n-11} & = 0 \end{array} \right.$$

Система, полученная в результате умножения матрицы (6.5) на  $i$ -й столбец матрицы (6.6), будет выглядеть следующим образом:

$$\left\{ \begin{array}{ll} a_{00}y_{0i} + a_{01}y_{1i} + \dots + a_{0n-1}y_{n-1i} & = 0, \\ a_{10}y_{0i} + a_{11}y_{1i} + \dots + a_{1n-1}y_{n-1i} & = 0, \\ \dots & \dots \\ a_{i0}y_{0i} + a_{i1}y_{1i} + \dots + a_{in-1}y_{n-1i} & = 1, \\ \dots & \dots \\ a_{n-10}y_{0i} + a_{n-11}y_{1i} + \dots + a_{n-1n-1}y_{n-1i} & = 0 \end{array} \right.$$

Понятно, что  $n$ -я система будет иметь вид:

$$\left\{ \begin{array}{ll} a_{00}y_{0n-1} + a_{01}y_{1n-1} + \dots + a_{0n-1}y_{n-1n-1} & = 0, \\ a_{10}y_{0n-1} + a_{11}y_{1n-1} + \dots + a_{1n-1}y_{n-1n-1} & = 0, \\ \dots & \dots \\ a_{i0}y_{0n-1} + a_{i1}y_{1n-1} + \dots + a_{in-1}y_{n-1n-1} & = 0, \\ \dots & \dots \\ a_{n-10}y_{0n-1} + a_{n-11}y_{1n-1} + \dots + a_{n-1n-1}y_{n-1n-1} & = 1 \end{array} \right.$$

Решением каждой из приведенных выше систем будет  $i$ -й столбец обратной матрицы. Количество систем равно размерности обратной матрицы. Для отыскания решений систем линейных алгебраических уравнений можно воспользоваться методом Гаусса.

Описанный алгоритм представлен в виде блок-схемы на рис. 6.15. Блоки 2–5 отражают формирование столбца единичной матрицы. Если условие 3 выполняется и элемент находится на главной диагонали, то он равен единице, все остальные элементы нулевые. В блоке 6 происходит вызов подпрограммы для решения системы уравнений методом Гаусса. В качестве параметров в эту подпрограмму передается исходная матрица  $A$ , сформированный в блоках 2–5 вектор свободных коэффициентов  $B$ , размерность системы  $n$ . Вектор  $X$  будет решением  $i$ -й системы уравнений и, следовательно,  $i$ -м столбцом искомой матрицы  $Y$ .

Как видно из блок-схемы, приведенной на рис. 6.15, при нахождении обратной матрицы понадобится функция SLAU, рассмотренная при решении задачи 6.10. Ниже приведён текст программы с подробными комментариями решения задачи 6.11. В функции `main()` будет находиться ввод исходной матрицы, обращение к функции `INVERSE` для вычисления обратной матрицы. Из функции `INVERSE` будет осуществляться вызов функции `SLAU` для решения системы линейных алгебраических уравнений.

```

#include <iostream>
#include <math.h>
using namespace std;
//Функция решения системы линейных алгебраических уравнений методом Гаусса.
int SLAU(double **matrica_a, int n, double *massiv_b, double **x)
{
    int i, j, k, r;
    double c, M, max, s;
    double **a, *b;
    a=new double *[n];
    for (i=0;i<n; i++)
        a[i]=new double [n];
    b=new double [n];
    for (i=0;i<n; i++)
        for (j=0;j<n; j++)
            a[i][j]=matrica_a[i][j];
    for (i=0;i<n; i++)
        b[i]=massiv_b[i];
    for (k=0;k<n; k++)
    {
        max=fabs(a[k][k]);
        r=k;
        for (i=k+1;i<n; i++)
            if (fabs(a[i][k])>max)
            {
                max=fabs(a[i][k]);
                r=i;
            }
        for (j=0;j<n; j++)
        {
            c=a[k][j];
            a[k][j]=a[r][j];
            a[r][j]=c;
        }
        c=b[k];
        b[k]=b[r];
        b[r]=c;
        for (i=k+1;i<n; i++)
        {
            for (M=a[i][k]/a[k][k], j=k; j<n; j++)
                a[i][j]-=M*a[k][j];
            b[i]-=M*b[k];
        }
    }
    if (a[n-1][n-1]==0)
        if (b[n-1]==0)
            return -1;
        else return -2;
    else
    {
        for (i=n-1; i>=0; i--)
        {
            for (s=0, j=i+1; j<n; j++)
                s+=a[i][j]*x[j];
            x[i]=(b[i]-s)/a[i][i];
        }
        return 0;
    }
    for (i=0; i<n; i++)
        delete [] a[i];
    delete [] a;
    delete [] b;
}
//Функция вычисления обратной матрицы
int INVERSE(double **a, int n, double **y)

```

```

//Формальные параметры: a — исходная матрица, n размерность матрицы, y — обратная матрица.
//Функция будет возвращать 0, если обратная матрица существует, -1 — в противном случае.
{
    int i, j, res;
    double *b, *x;
    //Выделение памяти для промежуточных массивов b и x.
    b=new double [n];
    x=new double [n];
    for (i=0;i<n; i++)
    {
        //Формирование вектора правых частей для нахождения i-го столбца матрицы.
        for (j=0;j<n; j++)
            if (j==i)
                b[j]=1;
            else b[j]=0;
        //Нахождения i-го столбца матрицы путем решения СЛАУ Ax = b методом Гаусса.
        res=SLAU(a ,n ,b ,x);
        //Если решение СЛАУ не найдено, то невозможно вычислить обратную матрицу.
        if (res!=0)
            break;
        else
            //Формирование i-го столбца обратной матрицы.
            for (j=0;j<n; j++)
                y[j][i]=x[j];
    }
    //Проверка существования обратной матрицы, если решение одного из уравнений Ax=b не
    //существует, то невозможно найти обратную матрицу, и функция INVERSE вернет значение -1.
    if (res!=0)
        return -1;
    //Если обратная матрица найдена, то функция INVERSE вернет значение 0,
    //а обратная матрица будет возвращаться через указатель double **y.
    else
        return 0;
}
int main()
{
    int result ,i ,j ,N;
    double **a, **b; //Двойные указатели для хранения исходной a и обратной b матрицы.
    cout<<"N="; //Ввод размера матрицы.
    cin>>N;
    a=new double *[N]; //Выделение памяти для матриц a и b.
    for (i=0;i<N; i++)
        a[i]=new double [N];
    b=new double *[N];
    for (i=0;i<N; i++)
        b[i]=new double [N];
    cout<<"Ввод матрицы A"<<endl; //Ввод исходной матрицы.
    for (i=0;i<N; i++)
        for (j=0;j<N; j++)
            cin>>a[i][j];
    result=INVERSE(a ,N ,b); //Вычисление обратной матрицы.
    if (result==0) //Если обратная матрица существует, то вывести ее на экран.
    {
        cout<<"Обратная матрица"<<endl;
        for (i=0;i<N; cout<<endl , i++)
            for (j=0;j<N; j++)
                cout<<b[i][j]<<"\t";
    }
    else
        //Если обратная матрица не существует, то вывести соответствующее сообщение.
        cout<<"Нет обратной матрицы"<<endl;
}

```

**Задача 6.12.** Найти определитель квадратной матрицы  $A(N, N)$ .

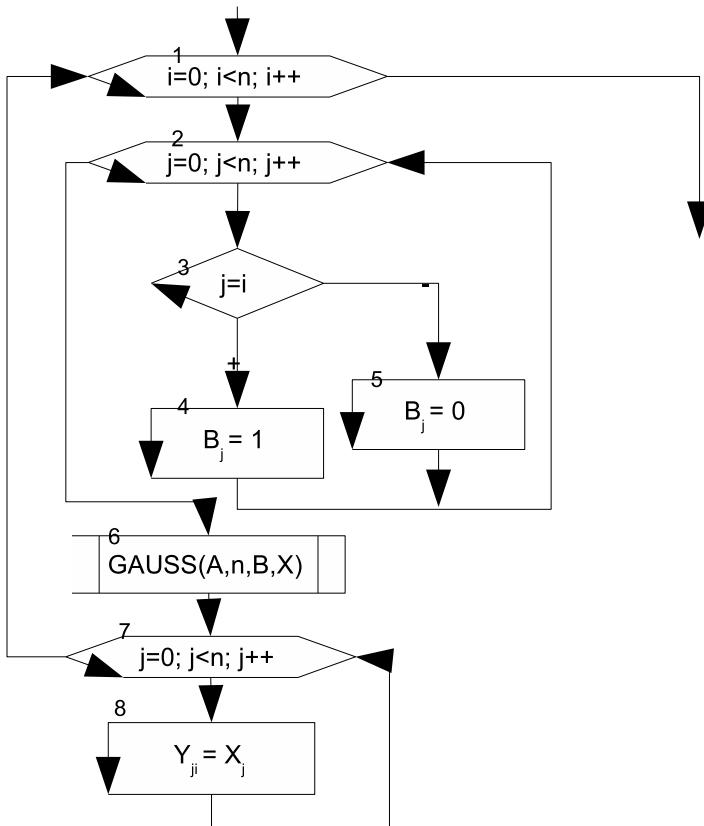


Рис. 6.15: Блок-схема алгоритма вычисления обратной матрицы

Пусть задана матрица (6.2), необходимо вычислить ее *определитель*. Для этого матрицу необходимо преобразовать к треугольному виду (6.3), а затем воспользоваться свойством, известным из курса линейной алгебры, которое гласит, что определитель треугольной матрицы равен произведению ее диагональных элементов:  $\det A = \prod_{i=0}^{n-1} a_{ii}$ .

Преобразование матрицы (6.2) к виду (6.3) можно осуществить с помощью прямого хода метода Гаусса. Алгоритм вычисления определителя матрицы, изображённый в виде блок-схемы на рис. 6.16, представляет собой алгоритм прямого хода метода Гаусса, в процессе выполнения которого проводится перестановка строк матрицы. Эта операция приводит к смене знака определителя. В блок-

схеме момент смены знака отражён в блоках 8–9. В блоке 8 определяется, будут ли строки меняться местами, и если ответ утвердительный, то в блоке 9 происходит смена знака определителя. В блоках 15–16 выполняется непосредственное вычисление определителя путём перемножения диагональных элементов преобразованной матрицы.

На листинге приведен текст программы решения задачи 6.12 с комментариями.

```
#include <iostream>
#include <math.h>
using namespace std ;
//Функция вычисления определителя.
double determinant(double **matrica_a , int n)
//Формальные параметры: matrica_a — исходная матрица, n — размер матрицы,
//функция возвращает значение определителя (тип double.)
{
    int i,j,k,r;
    double c,M,max,s,det=1;
    //a — копия исходной матрицы.
    double **a;
    //Выделение памяти для матрицы a .
    a=new double *[n];
    for(i=0;i<n;i++)
        a[i]=new double[n];
    //В a записываем копию исходной матрицы.
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            a[i][j]=matrica_a[i][j];
    //Прямой ход метода Гаусса.
    for(k=0;k<n;k++)
    {
        max=fabs(a[k][k]);
        r=k;
        for(i=k+1;i<n;i++)
            if (fabs(a[i][k])>max)
            {
                max=fabs(a[i][k]);
                r=i;
            }
        //Если строки менялись местами, то смена знака определителя.
        if (r!=k) det=-det;
        for(j=0;j<n;j++)
        {
            c=a[k][j];
            a[k][j]=a[r][j];
            a[r][j]=c;
        }
        for(i=k+1;i<n;i++)
            for(M=a[i][k]/a[k][k],j=k;j<n;j++)
                a[i][j]-=M*a[k][j];
    }
    //Вычисление определителя.
    for(i=0;i<n;i++)
        det*=a[i][i];
    //Возврат определителя в качестве результата функции.
    return det;
    for(i=0;i<n;i++)
        delete [] a[i];
    delete [] a;
}
int main()
{
    int result,i,j,N;
```

```

double **a, b;
cout<<"N=" ;
cin>>N;
a=new double *[N];
for ( i=0;i<N; i++)
    a [ i ]=new double [ N ];
//Ввод значений исходной матрицы.
cout<<"Ввод матрицы A "<<endl ;
for ( i=0;i<N; i++)
    for ( j=0;j<N; j++)
        cin>>a [ i ] [ j ];
//Обращение к функции вычисления определителя.
cout<<"определитель=" <<determinant ( a ,N )<<endl ;
}

```

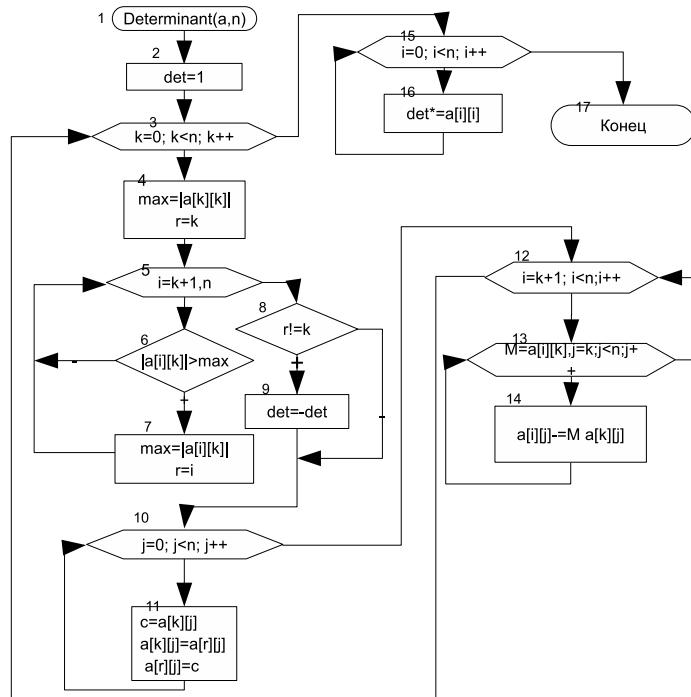


Рис. 6.16: Блок-схема алгоритма вычисления определителя

В этой главе читатель познакомился с обработкой статических и динамических матриц в C++, а также с использованием функций для решения задач обработки динамических матриц.

## 6.5 Задачи для самостоятельного решения

### 6.5.1 Основные операции при работе с матрицами

Разработать программу на языке C++ для решения следующей задачи.

1. В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить:

- наименьший элемент;
- сумму положительных элементов;
- количество простых чисел, расположенных на диагоналях матрицы

Для заданной матрицы  $A(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $B = 2 \cdot A + B^T$ .

2. В двумерном массиве  $C$ , состоящем из  $n \times n$  целых чисел вычислить:

- сумму элементов;
- количество нечетных элементов;
- минимальное простое число среди элементов, расположенных на главной диагонали.

Для заданной матрицы  $C(n \times n)$  и матрицы того же типа и размерности  $B(n \times n)$  найти значение выражения  $A = (B - C) \cdot C^T$

3. В двумерном массиве  $B$ , состоящем из  $m \times m$  целых чисел вычислить:

- номер наибольшего элемента;
- количество отрицательных элементов;
- среднее геометрическое среди простых чисел, расположенных на побочной диагонали.

Для заданной матрицы размерности  $B(n \times n)$  найти значение выражения  $A = 3 \cdot B + B^T$

4. В двумерном массиве  $A$ , состоящем из  $n \times m$  вещественных чисел вычислить:

- сумму элементов;
- произведение ненулевых элементов;
- два наибольших значения матрицы.

Для заданной матрицы  $A(n \times m)$  и матрицы того же типа и размерности  $C(n \times m)$  найти значение выражения  $B = 2 \cdot A + \frac{1}{3} \cdot C$

5. В двумерном массиве  $B$ , состоящем из  $n \times m$  вещественных чисел вычислить:

- произведение элементов;
- сумму положительных элементов;
- два наименьших значения среди расположенных выше побочной диагонали.

Для заданной матрицы  $B(n \times m)$  и матрицы того же типа, но другой размерности  $C(k \times n)$  найти значение выражения  $A = 3 \cdot B \cdot C$ .

6. В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить:

- наименьший элемент;
- количество четных чисел;
- сумму положительных элементов, которые представляют собой возрастающую последовательность цифр.

Для заданной матрицы  $A(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $B = A^2 - C^T$

7. В двумерном массиве  $C$ , состоящем из  $n \times n$  целых чисел вычислить:

- номер наименьшего элемента;
- сумму квадратов отрицательных элементов;
- минимальное простое число среди элементов, расположенных в заштрихованной части матрицы (рис. 6.17).

Для заданной матрицы  $C(n \times n)$  и матрицы того же типа и размерности  $B(n \times n)$  найти значение выражения  $A = (B^T + C)^2$


Рис. 6.17:


Рис. 6.18:

8. В двумерном массиве  $B$ , состоящем из  $n \times n$  целых чисел вычислить:

- среднее арифметическое элементов;
- наименьший четный элемент;
- количество чисел-палиндромов, расположенных в заштрихованной части матрицы (рис. 6.18).

Для заданной матрицы  $B(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $A = \frac{1}{2} \cdot B + C^2$

9. В двумерном массиве  $C$ , состоящем из  $n \times n$  целых чисел вычислить:

- среднее геометрическое элементов;
- наибольший нечетный элемент;
- количество составных чисел среди элементов, расположенных в заштрихованной части матрицы (рис. 6.19).

Для заданной матрицы  $C(n \times n)$  найти значение выражения  $A = C + C^T$ .

10. В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить:

- номер наименьшего элемента;
- среднее арифметическое нечетных чисел;
- количество положительных элементов, которые представляют собой убывающую последовательность цифр.

Для заданной матрицы  $A(n \times n)$  найти значение выражения  $B = \frac{1}{5} \cdot A^2$ .

11. В двумерном массиве  $B$ , состоящем из  $n \times n$  вещественных чисел вычислить:

- среднее арифметическое элементов;

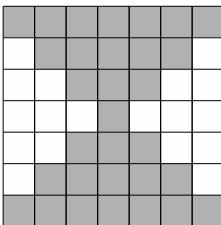


Рис. 6.19:

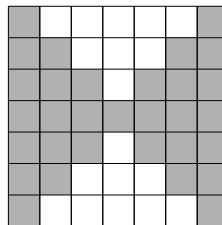


Рис. 6.20:

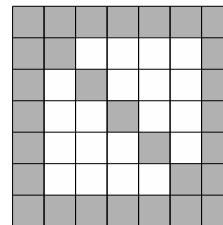


Рис. 6.21:

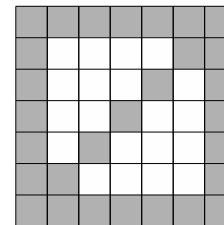


Рис. 6.22:

- элемент наиболее отличающийся от среднего арифметического.

Отразить заданную матрицу относительно побочной диагонали.

Для матрицы  $B(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $A = 2 \cdot B - C^T$ .

12. В двумерном массиве  $C$ , состоящем из  $n \times n$  целых чисел вычислить:

- среднее геометрическое элементов;
- элемент наименее отличающийся от среднего геометрического;
- количество положительных элементов с четной суммой цифр, расположенных в заштрихованной части матрицы (рис. 6.20)

Для матрицы  $C(n \times n)$  и матрицы того же типа и размерности  $B(n \times n)$  найти значение выражения  $A = (B - C) \cdot (B + C)$ .

13. В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить:

- наименьший элемент и его номер;
- среднее арифметическое положительных четных элементов;
- произведение простых чисел-палиндромов, расположенных в заштрихованной части матрицы (рис. 6.21).

Для заданной матрицы  $A(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $B = A^2 - C^2$ .

14. В двумерном массиве  $C$ , состоящем из  $n \times n$  целых чисел вычислить:

- наибольший элемент и его номер;
- среднее арифметическое элементов, расположенных на диагоналях матрицы.

Сформировать новую матрицу  $A(n \times n)$ , каждый элемент которой будет равен сумме цифр элемента матрицы  $C(n \times n)$ . Для матриц  $A(n \times n)$  и  $C(n \times n)$  найти значение выражения  $B = (A + C)^2$ .

15. В двумерном массиве  $B$ , состоящем из  $m \times m$  целых чисел вычислить:

- произведение элементов;
- номер наибольшего четного элемента;
- сумму чисел-палиндромов, расположенных вне диагоналей матрицы.

Для заданной матрицы размерности  $B(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $A = C \cdot B - B^T$ .

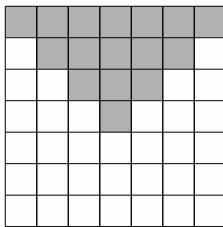


Рис. 6.23:

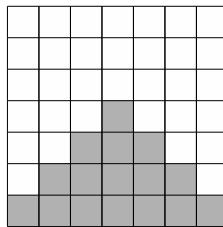


Рис. 6.24:

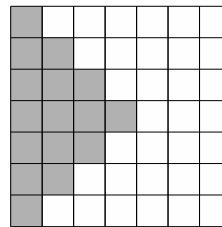


Рис. 6.25:

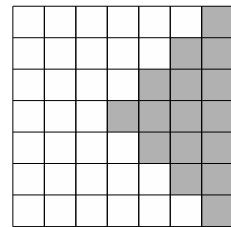


Рис. 6.26:

16. В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить:

- среднее арифметическое элементов;
- номер наименьшего нечетного элемента, расположенного в заштрихованной части матрицы (рис. 6.22).

Сформировать новую матрицу  $B(n \times n)$ , каждый элемент которой равен значению матрицы  $A(n \times n)$ , цифры которого записаны в обратном порядке. Для матриц  $A(n \times n)$  и  $B(n \times n)$  найти значение выражения  $B = A + C^2$ .

17. В двумерном массиве  $A$ , состоящем из  $n \times m$  целых чисел вычислить:

- сумму элементов;
- количество ненулевых элементов, расположенных по периметру матрицы;
- среднее геометрическое чисел, состоящих из различных цифр.

Для заданной матрицы  $A(n \times m)$  и матрицы того же типа и размерности  $C(n \times m)$  найти значение выражения  $B = 2 \cdot A - 3 \cdot C$ .

18. В двумерном массиве  $B$ , состоящем из  $n \times m$  целых чисел вычислить:

- произведение элементов;
- сумму элементов, расположенных вне периметра матрицы;
- наименьшее число, состоящее из одинаковых цифр.

Для заданной матрицы  $B(n \times m)$  и матрицы того же типа, но другой размерности  $C(k \times n)$  найти значение выражения  $A = B \cdot C$ .

19. В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить:

- среднее геометрическое элементов;
- номер наибольшего четного элемента, расположенного в заштрихованной части матрицы (рис. 6.23).

Сформировать новую матрицу  $B(n \times n)$ , каждый элемент которой равен значению матрицы  $A(n \times n)$  в восьмеричной системе счисления. Для матриц  $A(n \times n)$  найти значение выражения  $C = 3 \cdot A^2$ .

20. В двумерном массиве  $B$ , состоящем из  $n \times n$  целых чисел вычислить:

- сумму квадратов элементов;

- количество совершенных чисел, расположенного в заштрихованной части матрицы (рис. 6.24).

Сформировать новую матрицу  $A(n \times n)$ , каждый элемент которой равен количеству делителей соответствующего значения матрицы  $B(n \times n)$ . Для матриц  $A(n \times n)$  и  $B(n \times n)$  найти значение выражения  $C = B^T - A^2$ .

21. В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить:

- наименьшее абсолютное значение элементов;
- произведение ненулевых элементов, расположенного в заштрихованной части матрицы (рис. 6.25).

Сформировать новую матрицу  $B(n \times n)$ , каждый элемент которой равен разряду соответствующего элемента матрицы  $A(n \times n)$ . Для матриц  $A(n \times n)$  найти значение выражения  $C = B^T \cdot A$ .

22. В двумерном массиве  $B$ , состоящем из  $n \times n$  целых чисел вычислить:

- произведение ненулевых элементов;
- наибольшее абсолютное значение элементов, расположенного в заштрихованной части матрицы (рис. 6.26).

Сформировать новую матрицу  $C(n \times n)$ , каждый элемент которой равен значению матрицы  $B(n \times n)$  в пятеричной системе счисления. Для матриц  $B(n \times n)$  найти значение выражения  $A = B \cdot B^T$ .

23. В двумерном массиве  $C$ , состоящем из  $n \times m$  вещественных чисел вычислить:

- сумму модулей элементов;
- количество нулевых элементов, расположенных вне периметра матрицы;
- два наибольших положительных значения.

Для заданной матрицы  $C(n \times m)$  и матрицы того же типа, но другой размерности  $B(k \times n)$  найти значение выражения  $A = C \cdot B$ .

24. В двумерном массиве  $B$ , состоящем из  $n \times m$  вещественных чисел вычислить:

- сумму квадратов элемента;
- номер первого нулевого элемента матрицы;
- два наибольших значения, расположенных вне периметра матрицы;

Для заданной матрицы  $B(n \times m)$  найти значения выражений  $A = B \cdot B^T$  и  $C = B^T \cdot B$ .

25. В двумерном массиве  $A$ , состоящем из  $n \times n$  вещественных чисел вычислить:

- произведение квадратов элемента;
- номер последнего нулевого элемента матрицы;
- два наименьших значения, расположенных вне диагоналей матрицы.

Из элементов заданной матрицы  $A(n \times n)$  сформировать верхнетреугольную матрицу  $V$  и нижнетреугольную матрицу  $U$ . Проверить равенство  $A = V \cdot U$ .

### 6.5.2 Работа со строками и столбцами матрицы

Разработать программу на языке C++ для решения следующей задачи.

1. Задана матрица целых чисел  $A(n \times m)$ . Сформировать массив  $B(m)$ , в который записать среднее арифметическое элементов каждого столбца заданной матрицы. Вывести номера строк матрицы, в которых находится более двух *простых чисел*.
2. Задана матрица вещественных чисел  $B(n \times m)$ . Сформировать массив  $A(n)$ , в который записать среднее геометрическое положительных элементов каждой строки заданной матрицы. Определить количество столбцов, упорядоченных по возрастанию.
3. Задана матрица целых чисел  $A(n \times n)$ . Все *простые числа*, расположенные на побочной диагонали, заменить *суммой цифр* максимального элемента соответствующей строки матрицы. Сформировать массив  $B(n)$ , в который записать произведения элементов нечетных строк заданной матрицы.
4. В матрице целых чисел  $X(n \times n)$  поменять местами диагональные элементы, упорядоченных по убыванию строк. Сформировать массив  $Y(n)$ , в который записать суммы элементов четных столбцов заданной матрицы.
5. Задана матрица целых чисел  $A(n \times n)$ . Максимальный элемент каждого столбца заменить *суммой цифр* максимального элемента матрицы. Сформировать массив  $B(n)$ , в который записать количество четных элементов в каждой строке заданной матрицы.
6. Задана матрица целых чисел  $B(n \times m)$ . Максимальный элемент каждого столбца заменить *суммой цифр* модуля минимального элемента матрицы. Сформировать массив  $A(n)$ , в который записать количество нечетных элементов в каждой строке заданной матрицы.
7. Задана матрица целых чисел  $A(n \times n)$ . Сформировать массив  $B(n)$  из максимальных элементов столбцов заданной матрицы. Вывести номера строк, в которых числа-палиндромы находятся на диагоналях матрицы.
8. Задана матрица вещественных чисел  $P(n \times m)$ . Сформировать массив  $R(k)$  из номеров столбцов матрицы, в которых есть хотя бы один ноль. Найти строку с максимальной суммой элементов и поменять ее с первой строкой.
9. Задана матрица вещественных чисел  $C(k \times m)$ . Сформировать вектор  $D(k)$  из средних арифметических положительных значений строк матрицы, и вектор  $G(n)$  из номеров столбцов, которые представляют собой знакочередующийся ряд.
10. В каждом столбце матрицы вещественных чисел  $P(k \times m)$  заменить минимальный элемент суммой положительных элементов этого же столбца. Сформировать вектор  $D(n)$  из номеров строк, представляющих собой знакочередующийся ряд.
11. В матрице целых чисел  $A(n \times m)$  обнулить строки, в которых более двух *простых чисел*. Сформировать вектор  $D(m)$  из минимальных значений столбцов матрицы.

12. В матрице вещественных чисел  $P(n \times m)$  найти и вывести номера столбцов, упорядоченных по убыванию элементов. Сформировать вектор  $R(n)$  из максимальных значений строк матрицы.
13. В матрице вещественных чисел  $D(n \times m)$  найти и вывести номера строк, упорядоченных по возрастанию элементов. Сформировать вектор  $C(m \times 2)$  из номеров минимальных и максимальных значений столбцов матрицы.
14. В матрице вещественных чисел  $P(n \times m)$  найти и вывести номера столбцов, упорядоченных по возрастанию. Сформировать вектор  $R(n \times 2)$  из номеров минимальных и максимальных значений строк матрицы.
15. В матрице вещественных чисел  $D(n \times m)$  найти и вывести номера строк, упорядоченных по убыванию. Сформировать вектор  $C(m \times 2)$  из максимальных и минимальных значений столбцов матрицы.
16. В матрице вещественных чисел  $X(n \times n)$  найти максимальный и минимальный элементы. Поменять местами элементы строки с максимальным значением и элементы столбца с минимальным значением.
17. Задана матрица целых чисел  $A(n \times n)$ . Сформировать массив  $B(n)$ , каждый элемент которого равен количеству положительных элементов с чётной суммой цифр в соответствующей строке матрицы. В столбцах матрицы поменять местами наибольший и наименьший элементы.
18. Задана матрица целых чисел  $A(n \times m)$ . Сформировать массив  $B(m)$ , каждый элемент которого равен количеству положительных чисел с суммой цифр кратной трем в соответствующем столбце матрицы. Найти строку с максимальным произведением элементов.
19. Задана матрица целых чисел  $A(n \times n)$ . Все *числа-палиндромы*, расположенные на главной диагонали, заменить суммой цифр модуля минимального элемента соответствующего столбца матрицы. Сформировать вектор  $D(n)$  из произведений абсолютных ненулевых значений соответствующих строк матрицы.
20. Задана матрица целых чисел  $A(n \times n)$ . Поменять местами элементы на диагоналях в столбцах, упорядоченных по возрастанию модулей. Сформировать вектор  $B(n)$ , каждый элемент которого равен сумме *составных значений* в соответствующей строке матрицы.
21. Задана матрица целых чисел  $A(n \times n)$ . Минимальный элемент каждой строки заменить суммой цифр максимального *простого элемента* матрицы. Сформировать вектор  $B(n)$ , каждый элемент которого — среднее геометрическое ненулевых элементов в соответствующем столбце матрицы.
22. Задана матрица целых чисел  $A(n \times n)$ . Максимальный элемент каждого столбца заменить суммой цифр минимального *простого элемента* матрицы. Сформировать вектор  $B(n)$ , каждый элемент которого равен количеству четных элементов в соответствующей строке матрицы.
23. Задана матрица целых чисел  $A(n \times n)$ . Обнулить строки, в которых на диагоналях нет *чисел-палиндромов*. Сформировать вектор  $B(n)$ , каждый элемент которого равен количеству нечетных элементов в соответствующем столбце матрицы.

24. Задана матрица вещественных чисел  $P(n \times m)$ . Найти столбец с минимальным произведением элементов. Поменять местами элементы этого столбца и элементы последнего столбца. Сформировать вектор  $R(n)$  из сумм квадратов соответствующих строк матрицы.
25. Задана матрица целых чисел  $A(n \times m)$ . В каждой строке заменить максимальный элемент *суммой цифр* минимального элемента этой же строки. Сформировать вектор  $B(m \times 2)$ , пара элементов которого равна соответственно количеству четных и нечетных чисел в соответствующем столбце матрицы.

### 6.5.3 Решение задач линейной алгебры

Разработать программу на языке C++ для решения следующей задачи.

1. Задана матрицы  $A(n \times n)$  и  $B(n \times n)$ . Вычислить матрицу  $C = 2(A + B^{-1}) - A^T \cdot B$ .
2. Задан массив  $C(n)$ . Сформировать матрицы  $A(n \times n)$  и  $B(n \times n)$  по формулам:  $A_{ij} = C_i \cdot C_j$ ,  $B_{i,j} = \frac{A_{i,j}}{\max(A)}$ .

Решить матричное уравнение  $X(A + E) = 3B - E$ , где  $E$  — единичная матрица.

3. Даны массивы  $C(n)$  и  $D(n)$ . Сформировать матрицы  $A(n \times n)$  и  $B(n \times n)$  по формулам:

$$A_{ij} = C_i \cdot D_j, B_{i,j} = \frac{A_{i,j}}{\min(A)}.$$

Решить матричное уравнение  $(2A - E)X = B + E$ , где  $E$  — единичная матрица.

4. Квадратная матрица  $A(n \times n)$  называется *ортогональной*, если  $A^T = A^{-1}$ . Определить является ли данная матрица ортогональной:

$$\begin{pmatrix} 1 & 0.42 & 0.54 & 0.66 \\ 0.42 & 1 & 0.32 & 0.44 \\ 0.54 & 0.32 & 1 & 0.22 \\ 0.66 & 0.44 & 0.22 & 1 \end{pmatrix}.$$

5. Для матрицы

$$H = E - \frac{vv^T}{|v|^2}, \text{ где } E \text{ — единичная матрица, а } v = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix},$$

проверить свойство ортогональности:  $H^T = H^{-1}$ .

6. Проверить, образуют ли базис векторы

$$f_1 = \begin{bmatrix} 1 \\ -2 \\ 1 \\ 1 \end{bmatrix}, f_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \\ -1 \end{bmatrix}, f_3 = \begin{bmatrix} 5 \\ -2 \\ -3 \\ 1 \end{bmatrix}, f_4 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}.$$

Если образуют, то найти координаты вектора  $x = [1 \ -1 \ 3 \ -1]^T$  в этом базисе. Для решения задачи необходимо показать, что определитель

матрицы  $F$  со столбцами  $f_1, f_2, f_3, f_4$  отличен от нуля, а затем вычислить координаты вектора  $x$  в новом базисе по формуле  $y = F^{-1} \cdot x$ .

7. Найти вектор  $x$ , как решение данной системы уравнений

$$\begin{cases} 3.75x_1 - 0.28x_2 + 0.17x_3 = 0.75 \\ 2.11x_1 - 0.11x_2 - 0.12x_3 = 1.11 \\ 0.22x_1 - 3.17x_2 + 1.81x_3 = 0.05. \end{cases}$$

Вычислить модуль вектора  $x$ .

8. Вычислить скалярное произведение векторов  $x$  и  $y$ . Вектор  $y = |1 \ 1 \ 2 \ -3|$ , а вектор  $x$  является решением СЛАУ:

$$\begin{cases} 5.7x_1 - 7.8x_2 - 5.6x_3 - 8.3x_4 = 2.7 \\ 6.6x_1 + 13.1x_2 - 6.3x_3 + 4.3x_4 = -5.5 \\ 14.7x_1 - 2.8x_2 + 5.6x_3 - 12.1x_4 = 8.6 \\ 8.5x_1 + 12.7x_2 - 23.7x_3 + 5.7x_4 = 14.7. \end{cases}$$

9. Вычислить вектор  $X$ , решив СЛАУ

$$\begin{cases} 4.4x_1 - 2.5x_2 + 19.2x_3 - 10.8x_4 = 4.3 \\ 5.5x_1 - 9.3x_2 - 14.2x_3 + 13.2x_4 = 6.8 \\ 7.1x_1 - 11.5x_2 + 5.3x_3 - 6.7x_4 = -1.8 \\ 14.2x_1 + 23.4x_2 - 8.8x_3 + 5.3x_4 = 7.2. \end{cases}$$

Найти  $Y = X \cdot X^T$ .

10. Вычислить вектор  $X$ , решив СЛАУ

$$\begin{cases} 0.34x_1 + 0.71x_2 + 0.63x_3 = 2.08 \\ 0.71x_1 - 0.65x_2 - 0.18x_3 = 0.17. \\ 1.17x_1 - 2.35x_2 + 0.75x_3 = 1.28 \end{cases}$$

Найти модуль вектора  $|2X - 3|$ .

11. Вычислить угол между векторами  $x$  и  $y = |-1 \ 5 \ -3|$ . Вектор  $x$  является решением СЛАУ:

$$\begin{cases} 1.24x_1 + 0.62x_2 - 0.95x_3 = 1.43 \\ 2.15x_1 - 1.18x_2 + 0.57x_3 = 2.43. \\ 1.72x_1 - 0.83x_2 + 1.57x_3 = 3.88 \end{cases}$$

12. Решив систему уравнений методом Гаусса:

$$\begin{cases} 8.2x_1 - 3.2x_2 + 14.2x_3 + 14.8x_4 = -8.4 \\ 5.6x_1 - 12x_2 + 15x_3 - 6.4x_4 = 4.5 \\ 5.7x_1 + 3.6x_2 - 12.4x_3 - 2.3x_4 = 3.3 \\ 6.8x_1 + 13.2x_2 - 6.3x_3 - 8.7x_4 = 14.3 \end{cases}.$$

Вычислить  $H = E - XX^T$ .

13. Решить СЛАУ  $A^2X = Y^T$ , где  $A = \begin{bmatrix} 2 & 1 & 5 & 2 \\ 5 & 2 & 2 & 6 \\ 2 & 2 & 1 & 2 \\ 1 & 3 & 3 & 1 \end{bmatrix}$ ,  $Y = |3 \ 1 \ 2 \ 1|$ .

14. Решить СЛАУ  $2(A^T)^2X = Y$ , где  $A = \begin{bmatrix} 2 & 1 & 5 & 2 \\ 5 & 2 & 2 & 6 \\ 2 & 2 & 1 & 2 \\ 1 & 3 & 3 & 1 \end{bmatrix}$ ,  $Y = \begin{bmatrix} 3 \\ 1 \\ 2 \\ 1 \end{bmatrix}$ .

15. Заданы матрицы  $A(n \times n)$  и  $B(n \times n)$ .

Найти определитель матрицы  $C = B^T \cdot A$ .

16. Задан массив  $C(n)$ . Сформировать матрицы  $A(n \times n)$  и  $B(n \times n)$  по формулам:

$$A_{ij} = C_i \cdot C_j, B_{ij} = \frac{A_{ij}}{\sum_{i=1}^n A_{ii}}. \text{ Найти определитель } |2E - A \cdot B|.$$

17. Для матрицы  $I = 2P - E$ , где  $E$  — единичная матрица, а

$$P = \begin{bmatrix} -26 & -18 & -27 \\ 21 & 15 & 21 \\ 12 & 8 & 13 \end{bmatrix}$$

проверить свойство  $I^2 = E$ . При помощи метода Гаусса решить СЛАУ  $Ix = [1 \ 1 \ 1]^T$ .

18. Квадратная матрица  $A(n \times n)$  является *симметричной*, если для нее выполняется свойство  $A^T = A$ . Проверить это свойство для матрицы

$$\begin{pmatrix} 1 & 0.42 & 0.54 & 0.66 \\ 0.42 & 1 & 0.32 & 0.44 \\ 0.54 & 0.32 & 1 & 0.22 \\ 0.66 & 0.44 & 0.22 & 1 \end{pmatrix}.$$

Вычислить  $A^{-1}$ . Убедиться, что  $A \cdot A^{-1} = E$ .

19. Ортогональная матрица обладает следующими свойствами:

- модуль определителя ортогональной матрицы равен 1;
- сумма квадратов элементов любого столбца ортогональной матрицы равна 1;
- сумма произведений элементов любого столбца ортогональной матрицы на соответствующие элементы другого столбца равна 0.

Проверить эти свойства для матриц:

$$\begin{pmatrix} -2 & 3.01 & 0.12 & -0.11 \\ 2.92 & -0.17 & 0.11 & 0.22 \\ 0.66 & 0.52 & 3.17 & 2.11 \\ 3.01 & 0.42 & -0.27 & -0.15 \end{pmatrix}, \begin{pmatrix} -2 & 2.92 & 0.66 & 3.01 \\ 2.92 & -2 & 0.11 & 0.22 \\ 0.66 & 0.11 & -2 & 2.11 \\ 3.01 & 0.22 & 2.11 & -2 \end{pmatrix}.$$

20. Проверить, образуют ли базис векторы

$$f_1 = \begin{bmatrix} 0.25 \\ 0.333 \\ 0.2 \\ 0.1 \end{bmatrix}, f_2 = \begin{bmatrix} 0.33 \\ 0.25 \\ 0.167 \\ 0.143 \end{bmatrix}, f_3 = \begin{bmatrix} 1.25 \\ -0.667 \\ 2.2 \\ 3.1 \end{bmatrix}, f_4 = \begin{bmatrix} -0.667 \\ 1.333 \\ 1.25 \\ -0.75 \end{bmatrix}.$$

Если образуют, то найти координаты вектора  $x = [1 \ 1 \ 1 \ 1]^T$  в этом базисе. Для решения задачи необходимо показать, что определитель матрицы  $F$  со столбцами  $f_1, f_2, f_3, f_4$  отличен от нуля, а затем вычислить координаты вектора  $x$  в новом базисе, решив СЛАУ  $F \cdot y = x$ .

21. Решить СЛАУ:

$$\begin{pmatrix} 0.42 & 0.26 & 0.33 & -0.22 \\ 0.74 & -0.55 & 0.28 & -0.65 \\ 0.88 & 0.42 & -0.33 & 0.75 \\ 0.92 & 0.82 & -0.62 & 0.75 \end{pmatrix} \cdot X = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Для матрицы  $C = X \cdot X^T$  проверить условия ортогональности:  $C \cdot C^T = E$  и  $C^T \cdot C = E$ .

22. Найти  $\|A\|_1 = \max \sum_{j=1}^m |a_{ij}|$  и  $\|A\|_{11} = \max \sum_{i=1}^m |a_{ij}|$  для матрицы

$$\begin{pmatrix} 0.75 & 0.18 & 0.63 & -0.32 \\ 0.92 & 0.38 & -0.14 & 0.56 \\ 0.63 & -0.42 & 0.18 & 0.37 \\ -0.65 & 0.52 & 0.47 & 0.27 \end{pmatrix}^{-1}$$

23. Найти  $\|A\|_{111} = \sqrt{\sum_{i,j} a_{ij}^2}$  для матрицы

$$\begin{pmatrix} -1.09 & 7.56 & 3.45 & 0.78 \\ 3.33 & 4.45 & -0.21 & 3.44 \\ 2.33 & -4.45 & 0.17 & 2.21 \\ 4.03 & 1 & 3.05 & 0.11 \end{pmatrix}^{-1}$$

24. Решить СЛАУ методом Гаусса

$$\begin{cases} 8.2x_1 - 3.2x_2 + 14.2x_3 + 14.8x_4 = -8.4 \\ 5.6x_1 - 12x_2 + 15x_3 - 6.4x_4 = 4.5 \\ 5.7x_1 + 3.6x_2 - 12.4x_3 - 2.3x_4 = 3.3 \\ 6.8x_1 + 13.2x_2 - 6.3x_3 - 8.7x_4 = 14.3 \end{cases}$$

Выполнить проверку  $A \cdot x = b$ .

25. Задан массив  $H(k)$ . Сформировать матрицы  $B(k \times k)$  и  $G(k \times k)$  по формулам

$$(B_{ij} = H_i \cdot H_j), G_{ij} = \frac{B_{ij}}{\min(B)}.$$

Решить матричное уравнение  $(G + E) \cdot X = 5B^T - E$ , где  $E$  — единичная матрица.

# Глава 7

## Организация ввода-вывода в C++

### 7.1 Форматированный ввод-вывод в C++

В этом параграфе мы вернемся к рассмотренным ранее *операторам* `cin` и `cout`, и рассмотрим возможности их использования для организации *форматированного ввода-вывода*.

Для управления вводом-выводом в C++ используются:

- *флаги* форматного ввода-вывода;
- *манипуляторы* форматирования.

#### 7.1.1 Использование флагов форматного ввода-вывода

*Флаги* позволяют включить или выключить один из параметров вывода на экран. Для установки *флага вывода* используется следующая конструкция языка C++:

```
cout.setf(ios::flag)
```

Для снятия *флага* применяют конструкцию

```
cout.unsetf(ios::flag)
```

здесь *flag* — имя конкретного флага.

Если при выводе необходимо установить несколько флагов, то можно воспользоваться арифметической операцией «или» (`|`). В этом случае конструкция языка C++ будет такой:

```
cout.setf(ios::flag1|ios::flag2|ios::flag3)
```

В данном случае *flag1*, *flag2*, *flag3* — имена устанавливаемых флагов вывода.

В таблице 7.1 приведены некоторые *флаги форматного вывода* с примерами их использования.

Таблица 7.1: Некоторые флаги форматного вывода

Флаг	Описание	Пример использования <sup>1</sup>	Результат
right	Выравнивание по правой границе	<code>int r=-25; cout.setf(ios::right); cout.width(15); cout&lt;&lt;"r="&lt;&lt;r&lt;&lt;endl;</code>	<code>r=-25</code>
left	Выравнивание по левой границе (по умолчанию)	<code>double r=-25.45; cout.setf(ios::left); cout.width(50); cout&lt;&lt;"r="&lt;&lt;r&lt;&lt;endl;</code>	<code>r=-25.45</code>
boolalpha	Вывод логических величин в текстовом виде ( <code>true</code> , <code>false</code> )	<code>bool a=true; cout&lt;&lt;a&lt;&lt;endl; cout.setf(ios::boolalpha); cout&lt;&lt;a&lt;&lt;endl;</code>	<code>1 true</code>
dec	Вывод величин в десятичной системе счисления (по умолчанию)	<code>int r=-25; cout&lt;&lt;"r="&lt;&lt;r&lt;&lt;endl;</code>	<code>r=-25</code>
oct	Вывод величин в восьмеричной системе счисления	<code>int p=23; //Отменить, установленный по умолчанию, вывод в десятичной системе счисления cout.unsetf(ios::dec); //Установить вывод в восьмеричной системе счисления cout.setf(ios::oct); cout&lt;&lt;"p="&lt;&lt;p&lt;&lt;endl;</code>	<code>p=27</code>
hex	Вывод величин в шестнадцатеричной системе счисления	<code>int p=23; //Отменить, установленный по умолчанию, вывод в десятичной системе счисления cout.unsetf(ios::dec); //Установить вывод в шестнадцатеричной системе счисления cout.setf(ios::hex); cout&lt;&lt;"p="&lt;&lt;p&lt;&lt;endl;</code>	<code>p=17</code>
showbase	Выводить индикатор основания системы счисления	<code>int p=23; cout.unsetf(ios::dec); cout.setf(ios::hex ios::showbase); cout&lt;&lt;"p="&lt;&lt;p&lt;&lt;endl;</code>	<code>p=0x17</code>
uppercase	Использовать прописные буквы в шестнадцатеричных цифрах	<code>int p=29; cout.unsetf(ios::dec); cout.setf(ios::hex ios::uppercase); cout&lt;&lt;"p="&lt;&lt;p&lt;&lt;endl;</code>	<code>p=1D</code>
showpos	Выводить знак «+» для положительных чисел	<code>int p=29; cout.setf(ios::showpos); cout&lt;&lt;"p="&lt;&lt;p&lt;&lt;endl;</code>	<code>p=+29</code>
scientific	Экспоненциальная форма вывода вещественных чисел	<code>double p=146.673; cout.setf(ios::scientific); cout&lt;&lt;"p="&lt;&lt;p&lt;&lt;endl;</code>	<code>p=1.466730e+002</code>

Таблица 7.1 — продолжение

Флаг	Описание	Пример использования	Результат
<code>fixed</code>	Фиксированная форма вывода вещественных чисел (по умолчанию)	<code>double p=146.673; cout.setf(ios::fixed); cout&lt;&lt;"p=""&lt;&lt;p&lt;&lt;endl;</code>	<code>p=146.673</code>

Флаги удобно использовать в тех случаях, когда следует изменить параметры всех последующих операторов ввода-вывода. Использование большого количества флагов для управления одним или несколькими операторами ввода-вывода не совсем удобно, потом все установленные флаги придется отключать.

Еще одним способом форматирования является использование манипуляторов непосредственно в операторах `cin` и `cout`.

### 7.1.2 Использование манипуляторов форматирования

*Манипуляторы* встраиваются непосредственно в операторы ввода-вывода. С одним из манипуляторов (`endl`) читатель уже встречался, начиная с первой главы книги. В таблице 7.2 приведены основные манипуляторы форматирования с примерами их использования. Для корректного использования всех манипуляторов необходимо подключить библиотеку:

```
#include <iomanip>
```

Таблица 7.2: Некоторые манипуляторы форматирования

Манипулятор	Описание	Пример использования	Результат
<code>setw(n)</code>	Определяет ширину поля вывода в <i>n</i> символов	<code>int r=253; cout.setf(ios::fixed); cout&lt;&lt;"r=""&lt;&lt;setw(8)&lt;&lt;r&lt;&lt;endl;</code>	<code>r= 253</code>
<code>setprecision(n)</code>	Определяет количество цифр ( <i>n</i> – 1) в дробной части числа	<code>double h=1234.6578; cout.setf(ios::fixed); cout&lt;&lt;"h=""&lt;&lt;setw(15); cout&lt;&lt;setprecision(3); cout&lt;&lt;h&lt;&lt;endl;</code>	<code>h=1234.658</code>
<code>dec</code>	Перевод числа в десятичную систему (по умолчанию)	<code>int r=0253; cout&lt;&lt;"r=""&lt;&lt;dec&lt;&lt;r&lt;&lt;endl;</code>	<code>r=171</code>
<code>oct</code>	Перевод числа в восьмеричную систему	<code>int r=253; cout&lt;&lt;"r=""&lt;&lt;oct&lt;&lt;r&lt;&lt;endl;</code>	<code>r=375</code>
<code>hex</code>	Перевод числа в шестнадцатиричную систему	<code>int r=253; cout&lt;&lt;"r=""&lt;&lt;hex&lt;&lt;r&lt;&lt;endl</code>	<code>p=fd</code>
<code>right</code>	Выравнивание по правой границе	<code>int r=-25; cout.width(15); cout&lt;&lt;"r=""&lt;&lt;setw(15)&lt;&lt;right; cout&lt;&lt;r&lt;&lt;endl;</code>	<code>r=-25</code>

<sup>1</sup> `cout.width(n)` устанавливает ширину поля вывода, подробнее об этом в п. 7.1.2

Таблица 7.2 — продолжение

Манипулятор	Описание	Пример использования	Результат
<code>left</code>	Выравнивание по левой границе (по умолчанию)	<code>int r=-25; cout.width(15); cout&lt;&lt;"r="&lt;&lt;setw(15)&lt;&lt;left; cout&lt;&lt;r&lt;&lt;endl;</code> <sup>2</sup>	<code>r=-25</code>
<code>boolalpha</code>	Вывод логических величин в текстовом виде ( <code>true</code> , <code>false</code> )	<code>bool a=true; cout&lt;&lt;boolalpha&lt;&lt;a&lt;&lt;endl;</code>	<code>true</code>
<code>noboolalpha</code>	Вывод логических величин в числовом виде (1, 0)	<code>bool a=true; cout&lt;&lt;noboolalpha&lt;&lt;a&lt;&lt;endl;</code>	<code>1</code>
<code>showpos</code>	Выводить знак «+» для положительных чисел	<code>int p=29; cout&lt;&lt;"p="&lt;&lt;showpos&lt;&lt;p&lt;&lt;endl;</code>	<code>p=+29</code>
<code>noshowpos</code>	Не выводить знак «+» для положительных чисел	<code>int p=29; cout&lt;&lt;"p="&lt;&lt;noshowpos; cout&lt;&lt;p&lt;&lt;endl;</code>	<code>p=29</code>
<code>uppercase</code>	Использовать прописные буквы в шестнадцатеричных цифрах	<code>int p=253; cout&lt;&lt;"p="&lt;&lt;hex&lt;&lt;uppercase; cout&lt;&lt;p&lt;&lt;endl;</code>	<code>p=FD</code>
<code>nouppercase</code>	Использовать строчные буквы в шестнадцатеричных цифрах	<code>int p=253; cout&lt;&lt;"p="&lt;&lt;hex&lt;&lt;nouppercase; cout&lt;&lt;p&lt;&lt;endl;</code>	<code>p=fD</code>
<code>showbase</code>	Выводить индикатор основания системы счисления	<code>int p=253; cout&lt;&lt;"p="&lt;&lt;hex&lt;&lt;uppercase&lt;&lt;showbase&lt;&lt;p&lt;&lt;endl;</code>	<code>p=0XFD</code>
<code>noshowbase</code>	Не выводить индикатор основания системы счисления	<code>int p=253; cout&lt;&lt;"p="&lt;&lt;hex&lt;&lt;uppercase; cout&lt;&lt;noshowbase&lt;&lt;p&lt;&lt;endl;</code>	<code>p=FD</code>
<code>setfill(c)</code>	Установить символ <code>c</code> как заполнитель	<code>cout&lt;&lt;"x="&lt;&lt;right&lt;&lt;setw(10)&lt;&lt;setprecision(4); cout&lt;&lt;setfill('!'); cout&lt;&lt;(float)1/7&lt;&lt;endl; cout&lt;&lt;"x="&lt;&lt;left&lt;&lt;setw(10); cout&lt;&lt;setprecision(4); cout&lt;&lt;setfill('!'); cout&lt;&lt;(float)1/7&lt;&lt;endl;</code>	<code>x=!!!!0.1429 x=0.1429!!!!</code>
<code>scientific</code>	Экспоненциальная форма вывода вещественных чисел	<code>double p=146.673; cout&lt;&lt;"p="&lt;&lt;scientific; cout&lt;&lt;p&lt;&lt;endl;</code>	<code>p=1.466730e+002</code>
<code>fixed</code>	Фиксированная форма вывода вещественных чисел (по умолчанию)	<code>cout&lt;&lt;"p="&lt;&lt;fixed&lt;&lt;p&lt;&lt;endl;</code>	<code>p=146.673</code>

Кроме того управлять шириной поля вывода можно с помощью операторов:

- `cout.width(n)` — устанавливает ширину поля вывода —  $n$  позиций;

<sup>2</sup> Еще один пример приведен при использовании манипулятора `setfill`

- `cout.precision(m)` — определяет  $m$  цифр в дробной части числа.

В п. 7.1.1 и 7.1.2 были рассмотрены основные возможности форматированного ввода-вывода. При использовании операторов `cin` и `cout` фактически происходит ввод-вывод в *текстовый файл*. При вводе текстовым файлом является клавиатура ПК, при выводе в качестве текстового файла выступает экран дисплея, `cin` и `cout` фактически являются именами *потоков*<sup>3</sup>, которые отвечают за ввод и вывод в текстовый файл. Поэтому многие рассмотренные возможности форматированного ввода вывода будут использоваться и при обработке текстовых файлов.

Существует два основных типа файлов: *текстовые* и *двоичные*. Файлы позволяют пользователю считывать большие объемы данных непосредственно с диска, не вводя их с клавиатуры.

## 7.2 Работа с текстовыми файлами в C++

*Текстовыми* называют файлы, состоящие из любых символов. Они организуются по строкам, каждая из которых заканчивается символом «конец строки». Конец самого файла обозначается символом «конец файла». При записи информации в текстовый файл, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде.

Для работы с файлами используются специальные типы данных, называемые *потоками*<sup>4</sup>. Поток `ifstream` служит для работы с файлами в режиме чтения. Поток `ofstream` служит для работы с файлами в режиме записи. Для работы с файлами в режиме, как чтения, так и записи служит поток `fstream`.

В программах на C++ при работе с текстовыми файлами необходимо подключать библиотеки `iostream` и `fstream`.

Для того, чтобы записать данные в текстовый файл необходимо:

1. Описать переменную типа `ofstream`.
2. Открыть файл с помощью функции `open`.
3. Вывести информацию в файл.
4. Закрыть файл.

Для того, чтобы считать данные из текстового файла необходимо:

1. Описать переменную типа `ifstream`.
2. Открыть файл с помощью функции `open`.
3. Считать информацию из файла, при считывании каждой порции данных необходимо проверять достигнут ли конец файла.
4. Закрыть файл.

<sup>3</sup>Подробнее о текстовых потоках речь пойдет в п. 7.2

<sup>4</sup>Вообще говоря потоки являются классами, которым будет посвящена специальная глава.

### 7.2.1 Запись информации в текстовый файл

Для того, чтобы начать работать с текстовым файлом необходимо описать переменную типа `ofstream`. Например, с помощью оператора

`ofstream F;`<sup>5</sup>

будет создана переменная `F` для записи информации в файл. На следующем этапе файл необходимо открыть для записи. В общем случае оператор открытия потока будет иметь вид:

`F.open("file mode");`

Здесь `F` — переменная, описанная как `ofstream`, `file` — имя файла на диске, `mode` — режим работы с открываемым файлом.

Файл может быть открыт в одном из следующих режимов:

`ios::in` — открыть файл в режиме чтения данных, этот режим является режимом по умолчанию для потоков `ifstream`;

`ios::out` — открыть файл в режиме записи данных (при этом информация в существующем файле уничтожается), этот режим является режимом по умолчанию для потоков `ofstream`;

`ios::app` — открыть файл в режиме записи данных в конец файла;

`ios::ate` — передвинуться в конец уже открытого файла;

`ios::trunc` — очистить файл, это же происходит в режиме `ios::out`;

`ios::nocreate` — не выполнять операцию открытия файла, если он не существует<sup>6</sup>;

`ios::noreplace` — не открывать существующий файл.

Параметр `mode` может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока<sup>7</sup>.

После удачного открытия файла (в любом режиме) в переменной `F` будет храниться `true`, в противном случае `false`. Это позволит проверять корректность операции открытия файла.

*Открыть файл* (качестве примера возьмем файл `abc.txt`) в режиме записи можно одним из следующих способов:

```
//Первый способ.
ofstream F;
F.open("abc.txt", ios::out);
//Второй способ,
//режим ios::out является режимом по умолчанию для потока ofstream
ofstream F;
F.open("abc.txt");
//Третий способ объединяет описание переменной типа поток
//и открытие файла в одном операторе
ofstream F("abc.txt", ios::out);
```

После открытия файла в режиме записи, будет создан пустой файл, в который можно будет записывать информацию. Если нужно открыть файл, чтобы

<sup>5</sup>Далее термин «поток» будет использоваться для указания переменной, описанной как `ofstream`, `ifstream`, `fstream`, а термин «файл» для указания реального файла на диске.

<sup>6</sup>При открытии файла в режиме `ios::in` происходит как раз обратное, если файл не существует, он создается

<sup>7</sup>`ios::in` — для потоков `ifstream`, `ios::out` — для потоков `ofstream`.

в него что-либо *дописать*, то в качестве режима следует использовать значение `ios::app`.

После открытия файла в режиме записи, в него можно писать точно также, как и на экран, только вместо стандартного устройства вывода `cout` необходимо указать имя открытого для записи файла.

Например, для записи в поток `F` переменной `a`, оператор вывода будет иметь вид:

```
F<<a;
```

Для *последовательного вывода* в поток `G` переменных `b`, `c` и `d` оператор вывода станет таким:

```
G<<b<<c<<d;
```

*Закрытие потока* осуществляется с помощью оператора:

```
F.close();
```

В качестве примера рассмотрим следующую задачу.

**Задача 7.1.** Создать текстовый файл `abc.txt` и записать туда *n* вещественных чисел.

Текст программы с комментариями:

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
int main()
{
    int i, n; double a;
    ofstream f; //Описывает поток для записи данных в файл.
    f.open("abc.txt"); //Открываем файл в режиме записи,
                       //режим ios::out устанавливается по умолчанию.
    cout<<"n="; cin>>n; //Ввод количества вещественных чисел.
    for (i=0;i<n;i++)
    {
        cout<<"a="; cin>>a; //Ввод очередного числа.
        if (i<n-1) //Если число не последнее,
            f<<a<<"\t"; //записать в файл это число и символ табуляции, иначе
        else f<<a; //записать только число.
    }
    f.close(); //Закрытие потока.
    return 0;
}
```

Обратите внимание, что в текстовый файл записываются не только вещественные числа, но и символы табуляции. Таким образом, в конце файла после последнего числа находится символ табуляции. По этой причине может возникнуть проблема при чтении информации (п. 7.2.2), так как символ табуляции будет интерпретирован как вещественное число. Что бы этого избежать в программе была применена следующая конструкция:

```
if (i<n-1) f<<a<<"\t"; else f<<a;
```

Здесь реализована проверка ввода последнего числа. После него символ табуляции отсутствует.

В результате работы программы будет создан текстовый файл `abc.txt`, который можно просмотреть средствами обычного текстового редактора (рис. 7.1–7.2).

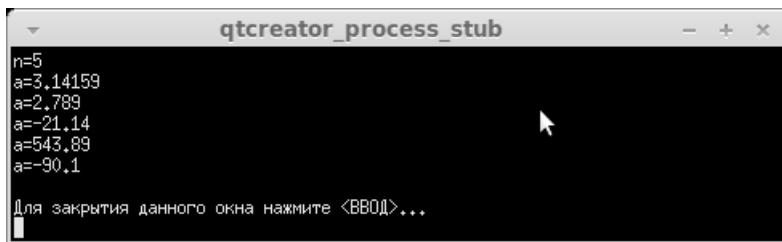


Рис. 7.1: Процесс работы программы к задаче 7.1. Ввод исходных данных.

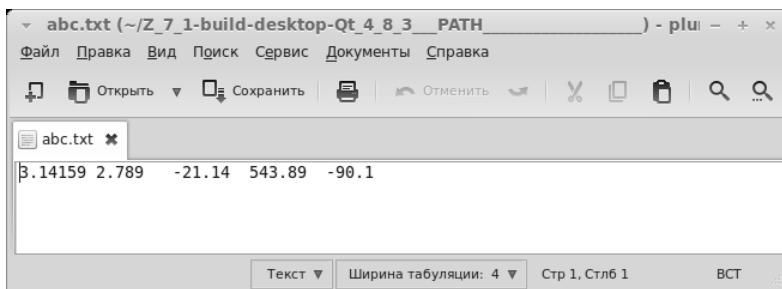


Рис. 7.2: Текстовый файл `abc.txt`, созданный программой к задаче 7.1.

### 7.2.2 Чтение информации из текстового файла

Для того чтобы прочитать информацию из текстового файла необходимо описать переменную типа `ifstream`. После этого файл необходимо открыть для чтения с помощью оператора `open`. Если переменную назвать `F`, то первые два оператора будут такими:

```
ifstream F;
F.open("abc.txt", ios::in);8
```

После открытия файла в режиме *чтения*, из него можно считывать информацию точно так же, как и с клавиатуры, только вместо стандартного устройства ввода `cin` необходимо указать *имя потока*, из которого будет происходить чтение данных.

Например, для чтения из потока `F` в переменную `a`, оператор ввода будет иметь вид:

```
F>>a;
```

<sup>8</sup>Указание режима `ios::in` можно, конечно, опустить, ведь для потока `ifstream` значение `ios::in` является значением по умолчанию, тогда оператор `open` можно будет записать так `F.open("abc.txt");`

Для последовательного ввода из потока **G** в переменные **b**, **c** и **d** оператор ввода станет таким:

```
G>>b>>c>>d;
```

Два числа в текстовом файле считаются разделенными, если между ними есть хотя бы один из символов: пробел, табуляция, символ конца строки.

Хорошо если программисту заранее известно, сколько и каких значений хранится в текстовом файле. Однако часто просто известен тип значений, хранящихся в файле, при этом количество значений в файле может быть различным. При решении подобной проблемы необходимо считывать значения из файла по одному, а перед каждым считыванием проверять достигнут ли конец файла. Для проверки достигнут или нет конец файла, служит функция

```
F.eof();
```

Здесь **F** — имя потока, функция возвращает логическое значение: **true** — если достигнут конец файла, если не достигнут функция возвращает значение **false**.

Следовательно, цикл для чтения содержимого всего файла можно записать так.

```
while (!F.eof()) //Организован цикл, условием окончания цикла
//является достижение конца файла, в этом случае F.eof() вернет true.
{
    F>>a; //Чтение очередного значения из потока F в переменную a.
    ...
    <обработка значения переменной a>
}
```

Рассмотрим следующую задачу.

**Задача 7.2.** В текстовом файле *abc.txt* хранятся вещественные числа (рис. 7.2), вывести их на экран и вычислить их количество.

Текст программы с комментариями приведен ниже.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <iomanip>
using namespace std;
int main()
{
    ifstream f; //Поток для чтения.
    float a; int n=0;
    f.open("abc.txt"); //Открываем файл в режиме чтения.
    if (f) //Если открытие файла прошло корректно, то
    {
        while (!f.eof()) //Организован цикл, выполнение цикла
        //прервется, когда ,будет достигнут конца файла.
        {
            f>>a; //Чтение очередного значения из потока f в переменную a.
            cout<<a<<"\t"; //Вывод значения переменной a
            n++; //Увеличение количества считанных чисел.
        }
        f.close(); //Закрытие потока.
        cout<<"n="<<n<<endl; //Вывод на экран количества чисел.
    }
    else cout<<"File not found"<<endl; //Если открытие файла прошло некорректно, то
    //вывод сообщение, об отсутствии такого файла.
    return 0;
}
```

Если количество вещественных чисел записанных в файл известно заранее, то текст программы можно переписать следующим образом:

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <iomanip>
using namespace std;
int main()
{
    ifstream f;
    float a; int i,n=5;
    f.open("abc.txt");
    if (f)
    {
        for (i=1;i<=n; f>>a, cout<<a<<"\t", i++);
        f.close();
    }
    else cout<<"File not found"<<endl;
    return 0;
}
```

Существует возможность открывать файл с данными таким образом, чтобы в него можно было *дописывать информацию*. Рассмотрим эту возможность на примере решения следующей задачи.

**Задача 7.3.** В файле *abc.txt* (рис. 7.2) хранится массив вещественных чисел, дописать в файл этот же массив, упорядочив его по возрастанию.

Алгоритм решения задачи очень простой. Считываем в массив данные из текстового файла, упорядочиваем массив, дописываем его в этот же файл.

Для чтения данных из файла описываем поток *ifstream*, открываем его в режиме чтения и последовательно, пока не достигнем конца файла, считываем все элементы в массив. Сортировку проведем методом пузырька. Обратите внимание, что поток нужно открыть так, чтобы была возможность дописать в конец файла упорядоченный массив.

Текст программы с комментариями приведен ниже.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <iomanip>
using namespace std;
int main()
{
    ifstream f; //Поток для чтения.
    ofstream g; //Поток для записи.
    float *a,b;
    a=new float[100];
    int i,j,n=0;
    f.open("abc.txt", ios::in); //Открываем файл в режиме чтения.
    if (f)
    {
        while (!f.eof())
        {
            f>>a[n];
            n++;
        }
        //Сортировка массива.
        for (i=0;i<n-1; i++)
            for (j=0;j<n-i-1; j++)
                if (a[j]>a[j+1])
                    swap(a[j], a[j+1]);
        g.open("abc.txt", ios::out);
        for (i=0;i<n; i++)
            g<<a[i]<<"\t";
        g.close();
    }
}
```

```

{
    b=a[ j ];
    a[ j ]=a[ j +1];
    a[ j +1]=b;
}
f . close () ; //Закрываем поток для чтения.
g . open ("abc . txt " ,ios :: app); //Открываем поток для того чтобы дописать данные.
g << "\n"; //Запись в файл символа конца строки
for ( i =0; i <n; i ++ ) //Запись в файл
    if ( i <n-1) g << a[ i ]<< "\t" ; //Элемента массива и символа табуляции.
    else g << a[ i ]; //Запись последнего элемента массива
g . close () ; //Закрытие файла.
}
else cout<<"File not found"<< endl;
delete [] a;
return 0;
}

```

Содержимое файла `abc.txt` после запуска программы к задаче 7.3

```
3.14159 2.798 -21.14 543.89 -90.1
-90.1 -21.14 2.798 3.14159 543.89
```

## 7.3 Обработка двоичных файлов

Если в файле хранятся только числа или данные определенной структуры, то для хранения таких значений удобно пользоваться двоичными файлами. В *двоичных файлах* информация считывается и записывается в виде блоков определенного размера, в них могут храниться данные любого вида и структуры.

Порядок работы с двоичными и текстовыми файлами аналогичен. Для того, чтобы *записать данные в двоичный файл* необходимо:

1. Описать файловую переменную с помощью оператора `FILE *filename;`  
Здесь, `filename` — имя переменной, где будет храниться указатель на файл.
2. Открыть файл с помощью функции `fopen`.
3. Записать информацию в файл с помощью функции `fwrite`.
4. Закрыть файл с помощью функции `fclose`.

Для того, чтобы *считывать данные из двоичного файла* необходимо:

1. Описать переменную типа `FILE *`.
2. Открыть файл с помощью функции `fopen`.
3. Считать необходимую информацию из файла с помощью функции `fread`, при считывании информации следить за тем, достигнут ли конец файла.
4. Закрыть файл с помощью функции `fclose`.

Рассмотрим основные функции, необходимые для работы с двоичными файлами.

Для *открытия файла* предназначена функция:

`FILE *fopen(const *filename, const char *mode);` где, `filename` — строка, в которой хранится полное имя открываемого файла, `mode` — строка, которая определяет режим работы с файлом; возможны следующие значения:

«`rb`» — открыть двоичный файл в режиме чтения;

«**wb**» — создать двоичный файл для записи, если файл существует, то его содержимое очищается.

«**ab**» — создать или открыть двоичный файл для добавления информации в конец файла;

«**rb+**» — открыть существующий двоичный файл в режиме чтения и записи;

«**wb+**» — открыть двоичный файл в режиме чтения и записи, существующий файл очищается;

«**ab+**» — двоичный файл открывается или создается для исправления существующей информации и добавления новой в конец файла.

Функция `fopen` возвращает в файловой переменной `NULL` в случае неудачного открытия файла.

После открытия файла, в указателе содержится адрес 0-го байта файла. По мере чтения или записи значение указателя смещается на считанное (записанное) количество байт. Текущее значение указателя — номер байта, начиная с которого будет происходить операция чтения или записи.

Для закрытия файла предназначена функция

```
int fclose(FILE *filename);
```

Она возвращает 0 при успешном закрытии файла и `NULL` в противном случае.

Для удаления файлов существует функция

```
int remove(const char *filename);
```

Эта функция удаляет с диска файл с именем `filename`. Удаляемый файл должен быть закрыт. Функция возвращает ненулевое значение, если файл не удалось удалить.

Для переименования файлов предназначена функция

```
int rename(const char *oldfilename, const char *newfilename);
```

здесь, первый параметр — старое имя файла, второй — новое. Возвращает 0 при удачном завершении программы.

Чтение из двоичного файла осуществляется с помощью функции

```
fread (void *ptr, size, n, FILE *filename)
```

Эта функция считывает из файла `filename` в массив `ptr` `n` элементов размера `size`. Функция возвращает количество считанных элементов. После чтения из файла указатель файла смещается на `n*size` байт.

Запись в двоичный файл осуществляется с помощью функции

```
fwrite (const void *ptr, size, n, FILE *filename);
```

Функция записывает в файл `filename` из массива `ptr` `n` элементов размера `size`. Функция возвращает количество записанных элементов. После записи информации в файл указатель файла смещается на `n*size` байт.

Для контроля достижения конца файла есть функция

```
int feof(FILE * filename);
```

Она возвращает ненулевое значение, если достигнут конец файла.

Рассмотрим использование двоичных файлов на примере решения двух стандартных задач.

**Задача 7.4.** Создать двоичный файл `abc.dat`, куда записать целое число `n` и `n` вещественных чисел.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    FILE *f; //Описание файловой переменной.
    int i, n; double a;
    f=fopen("abc.dat", "wb"); //Создание двоичного файла в режиме записи.
    cout<<"n="; cin>>n; //Ввод числа n.
    fwrite(&n, sizeof(int), 1, f); //Запись числа в двоичный файл.
    for (i=0;i<n; i++) //Цикл для ввода n вещественных чисел.
    {
        cout<<"a="; cin>>a; //Ввод очередного вещественного числа.
        fwrite(&a, sizeof(double), 1, f); //Запись числа в двоичный файл.
    }
    fclose(f); //Закрыть файл.
    return 0;
}
```

**Задача 7.5.** Вывести на экран содержимое созданного в задаче 7.4 двоичного файла abc.dat.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    FILE *f; //Описание файловой переменной.
    int i, n; double *a;
    f=fopen("abc.dat", "rb"); //Открыть существующий двоичный файл в режиме чтения.
    fread(&n, sizeof(int), 1, f); //Читать из файла целое число в переменную n.
    cout<<"n=<<n<<\"\\n\"; //Выход n на экран.
    a=new double[n]; //Выделение памяти для массива из n чисел.
    fread(a, sizeof(double), n, f); //Чтение n вещественных чисел из файла в массив a.
    for (i=0;i<n; cout<<a[i]<<"\\t", i++); //Выход массива на экран.
    cout<<endl;
    fclose(f); //Закрыть файл.
    return 0;
}
```

*Двоичный файл* — последовательная структура данных, после открытия файла доступен первый байт. Можно последовательно считывать из файла или записывать их в него. Допустим, необходимо считать пятнадцатое, а затем первое число, хранящееся в файле. С помощью *последовательного доступа* это можно сделать следующим образом.

```
FILE *f;
int i, n; double a;
f=fopen("file.dat", "rb");
for (i=0; i<15; fread(&a, sizeof(double), 1, f), i++);
fclose(f);
f=fopen("file.dat", "rb");
fread(&a, sizeof(double), 1, f);
fclose(f);
```

Как видно, такое чтение чисел из файла, а затем повторное открытие файла — не самый удачный способ. Гораздо удобнее использовать *функцию перемещения указателя файла* к заданному байту:

```
int fseek(FILE *F, long int offset, int origin);
```

Функция устанавливает указатель текущей позиции файла F, в соответствии со значениями начала отсчета origin и смещения offset. Параметр offset равен количеству байтов, на которые будет смешен указатель файла относительно

начала отсчета, заданного параметром `origin`. Если значение `offset` положительно, то указатель файла смещается вперед, если отрицательно — назад. Параметр `origin` должен принимать одно из следующих значений, определенных в заголовке `stdio.h`:

- `SEEK_SET` — отсчет смещения `offset` вести с начала файла;
- `SEEK_CUR` — отсчет смещения `offset` вести с текущей позиции файла;
- `SEEK_END` — отсчет смещения `offset` вести с конца файла.

Функция возвращает нулевое значение при успешном выполнении операции и ненулевое, если возник сбой при выполнении смещения.

Функция `fseek` фактически реализует прямой доступ к любому значению в файле. Необходимо только знать месторасположение (номер байта) значения в файле. Рассмотрим использование прямого доступа в двоичных файлах на примере решения следующей задачи.

**Задача 7.6.** В созданном в задаче 7.4 двоичном файле `abc.dat`, поменять местами наибольшее и наименьшее из вещественных чисел.

Алгоритм решения задачи состоит из следующих этапов:

1. Чтение вещественных чисел из файла в массив `a`.
2. Поиск в массиве `a` максимального (`max`) и минимального (`min`) значения и их номеров (`imax, imin`).
3. Перемещение указателя файла к максимальному значению и запись `min`.
4. Перемещение указателя файла к минимальному значению и запись `max`.

Ниже приведен текст программы решения задачи с комментариями.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    FILE *f; //Описание файловой переменной.
    int i, n, imax, imin;
    double *a, max, min;
    f=fopen ("abc.dat", "rb+"); //Открыть файл в режиме чтения и записи.
    fread(&n, sizeof(int), 1, f); //Считать из файла в переменную n количество элементов в
    //файле.
    a=new double[n]; //Выделить память для хранения вещественных чисел,
    //эти числа будут хранится в массиве a.
    fread(a, sizeof(double), n, f); //Считать из файла в массив a вещественные числа.
    //Поиск максимального, минимального элемента в массиве a, и их индексов.
    for(imax=imin=0, max=min=a[0], i=1; i<n; i++)
    {
        if (a[i]>max)
        {
            max=a[i];
            imax=i;
        }
        if (a[i]<min)
        {
            min=a[i];
            imin=i;
        }
    }
    //Перемещение указателя к максимальному элементу.
    fseek(f, sizeof(int)+imax*sizeof(double), SEEK_SET);
    //Запись min вместо максимального элемента файла.
```

```

fwrite(&min, sizeof(double), 1, f);
//Перемещение указателя к минимальному элементу.
fseek(f, sizeof(int)+imin*sizeof(double), SEEK_SET);
//Запись max вместо минимального элемента файла.
fwrite(&max, sizeof(double), 1, f);
//Закрытие файла.
fclose(f);
//Освобождение памяти, выделенной под массив a.
delete [] a;
return 0;
}

```

## 7.4 Функции fscanf() и fprintf()

Чтение и запись данных в файл можно выполнять с помощью функций **fscanf()** и **fprintf()**. Эти функции подобны функциям **scanf()** и **printf()**, описанным в п. 2.9, за тем исключением, что работают не с клавиатурой и экраном, а с файлами. Функции имеют следующие прототипы.

*Функция чтения*

**fscanf(указатель на файл, строка форматов, адреса переменных);**

*Функция записи*

**fprintf(указатель на файл, строка форматов, список переменных);**

Далее приведен фрагмент программного кода, который демонстрирует пример записи информации в файл **my.txt**.

```

char fio[15] = "Махарадзе В.";
int a=5, b=5, c=4;
float s = (float) (a+b+c)/3;
FILE *f;
f=fopen("my.txt", "w");
fprintf(f, "Оценки студента %s \n", fio);
fprintf(f, "математика %d, физика %d, химия %d \n", a, b, c);
fprintf(f, "Средний балл = %.2f \n", s);
fprintf(f, "\n");
fclose(f);

```

В результате будет сформирован текстовый файл:

```

Оценки студента Махарадзе В.
математика 5, физика 5, химия 4
Средний балл = 4.67

```

Рассмотрим пример чтения данных из файла. Пусть в файле **test.txt** хранится следующая информация:

```

1 Иванов Петр 170 78.1
2 Петров Иван 180 89.6
3 Карпов Борис 167 56.7

```

Тогда с помощью следующих команд можно считать информацию из файла и вывести ее на экран.

```

int i, nom;
float Ves;
int Rost;
char fio[7], name[6];
FILE *f;
f=fopen("test.txt", "r");

```

```
for ( i=0; i<3; i++)
{
    //Чтение из файла
    fscanf( f , "%d %s %s %f \n" ,&nom , &fio ,&name,&Rost ,&Ves ) ;
    //Вывод на экран
    printf( "%d %s %s %d %.2f \n" ,nom , fio ,name , Rost , Ves ) ;
}
fclose ( f ) ;
```

# Глава 8

## Строки в языке C++

В главе дано общее представление о строках в C++. Описана их структура, способы инициализации, возможности ввода-вывода, приведены примеры обработки строк и текстов.

### 8.1 Общие сведения о строках в C++

*Строка* — последовательность символов. Для работы с символами в языке C++ предусмотрен тип данных `char`. Если в выражении встречается одиночный символ, он должен быть заключен в одинарные кавычки. При использовании в выражениях строка заключается в двойные кавычки. Признаком конца строки является нулевой символ '\0'. В C++ строки можно описать с помощью массива символов (массив элементов типа `char`), в массиве следует предусмотреть место для хранения признака конца строки ('\0').

Например,

```
char s[25]; //Описана строка из 25 символов.  
//Элемент s[25] предназначен для хранения символа конца строки.  
char s[7] = "Привет"; //Описана строка из 7 символов и ей присвоено значение.  
//Определен массив из 3 строк по 25 байт в каждой.  
char m[3][25] = { "Пример" , "использования" , " строк" }
```

Для работы со строками можно использовать указатели (`char *`). Адрес первого символа будет начальным значением указателя.

Рассмотрим пример объявления и ввода строк.

```
#include <iostream>  
using namespace std;  
int main()  
{  
    char s2[20], *s3, s4[30]; //Описываем 3 строки, s3 — указатель.  
    cout << "Введите строку:" << endl;  
    cout << "s2="; cin >> s2; //Ввод строки s2.  
    cout << "Была введена строка:" << endl;  
    cout << "s2=" << s2 << endl;  
    s3=s4; //Запись в s3 адреса строки s4. Теперь в указателях s3 и s4  
           //хранится один адрес.  
    cout << "Введите строку:" << endl;  
    cout << "s3="; cin >> s3; //Ввод строки s3.
```

```

cout<<"Была введена строка: "<<endl;
cout<<"s3 = "<<s3<<endl; //Вывод на экран s3 и s4,
cout<<"Сформирована новая строка: "<<endl;
cout<<"s4 = "<<s4<<endl; //s3 и s4 — одно и тоже.
return 0;
}

```

Если запустить эту программу на выполнение, то в консольном окне приложения будет получен следующий результат.

```

Введите строку:
s2=Привет!
Была введена строка:
s2=Привет!
Введите строку:
s3=Программируем?
Была введена строка:
s3=Программируем?
Сформирована новая строка:
s4=Программируем?

```

Однако, если во вводимых строках появятся пробелы, программа будет работать не так, как ожидает пользователь:

```

Введите строку:
s2=Привет, Вася!
Была введена строка:
s2=Привет,
Введите строку:
s3=Была введена строка:
s3=Вася!
Сформирована новая строка:
s4=Вася!

```

Дело в том, что функция `cin` вводит строки до встретившегося пробела. Более универсальной функцией является функция

```
cin.getline(char *s, int n);
```

она предназначена для *ввода с клавиатуры строки s с пробелами, причем в строке не должно быть более n символов*. Например,

```

char s[20];
cout<<"Введите строку: "<<endl;
cout<<"s2 = "; cin.getline(s, 20);
cout<<"Была введена строка: "<<endl;
cout<<"s2 = "<<s2<<endl;

```

Результат:

```

Введите строку:
s2=Привет, Вася!
Была введена строка:
s2=Привет, Вася!

```

## 8.2 Операции над строками

Строчку можно обрабатывать как массив символов, используя алгоритмы обработки массивов, или с помощью специальных *функций обработки строк*, некоторые из которых приведены в таблицах 8.1–8.2.

Таблица 8.1: Функции работы со строками, библиотека `string.h`

Прототип функции	Описание функции	Пример использования	Результат
<code>size_t strlen(const char *s)</code>	Вычисляет длину строки <code>s</code> в байтах	<code>char s[80]; cout &lt;&lt; "s="; cin.getline(s,80); cout &lt;&lt; "s=" &lt;&lt; s &lt;&lt; endl &lt;&lt; "Длина строки" &lt;&lt; strlen(s) &lt;&lt; endl;</code>	<code>s&gt;Hello, Russia! s&gt;Hello, Russia! Длина строки 14</code>
<code>char *strcat(char *dest, const char *src)</code>	Присоединяет строку <code>src</code> в конец строки <code>dest</code> , полученная строка возвращается в качестве результата	<code>char s1[80],s2[80]; cout &lt;&lt; "s1="; cin.getline(s1,80); cout &lt;&lt; "s2="; cin.getline(s2,80); cout &lt;&lt; "s=" &lt;&lt; strcat(s1,s2);</code>	<code>s1&gt;Hello, s2=Russia! s&gt;Hello, Russia!</code>
<code>char *strcpy(char *dest, const char *src)</code>	Копирует строку <code>src</code> в место памяти, на которое указывает <code>dest</code>	<code>char s1[80],s2[80]; cout &lt;&lt; "s1="; cin.getline(s1,80); strcpy(s2,s1); cout &lt;&lt; "s2=" &lt;&lt; s2;</code>	<code>s1&gt;Hello,Russia! s2&gt;Hello,Russia!</code>
<code>char *strncat(char *dest, const char *src, size_t maxlen)</code>	Присоединяет строку <code>src</code> в конец строки <code>dest</code> <code>maxlen</code> символов строки <code>src</code>	<code>char s1[80],s2[80]; cout &lt;&lt; "s1="; cin.getline(s1,80); cout &lt;&lt; "s2="; cin.getline(s2,80); cout &lt;&lt; "s=" &lt;&lt; strncat(s1,s2,6);</code>	<code>s1&gt;Hello, s2=Russia! s&gt;Hello, Russia</code>
<code>char *strncpy(char *dest, const char *src, size_t maxlen)</code>	Копирует <code>maxlen</code> символов строки <code>src</code> в место памяти, на которое указывает <code>dest</code>	<code>char s1[80],s2[80]; cout &lt;&lt; "s1="; cin.getline(s1,80); strncpy(s2,s1,5); cout &lt;&lt; "s2=" &lt;&lt; s2;</code>	<code>s1&gt;Hello,Russia! s2&gt;Hello</code>
<code>int strcmp(const char *s1, const char *s2)</code>	Сравнивает две строки в лексикографическом порядке с учетом различия прописных и строчных букв, функция возвращает 0, если строки совпадают, возвращает -1, если <code>s1</code> располагается в упорядоченном по алфавиту порядке раньше, чем <code>s2</code> , и 1 в противоположном случае.	<code>char s1[80],s2[80]; cout &lt;&lt; "s1="; cin.getline(s1,80); cout &lt;&lt; "s2="; cin.getline(s2,80); cout &lt;&lt; strcmp(s1,s2) &lt;&lt; endl;</code>	<code>s1=RUSSIA s2=Russia -1</code>

Таблица 8.1 — продолжение

Прототип функции	Описание функции	Пример использования	Результат
<code>int strcmp(const char *s1, const char *s2, size_t maxlen)</code>	Сравнивает maxlen символов двух строк в лексикографическом порядке, функция возвращает 0, если строки совпадают, возвращает -1, если s1 располагается в упорядоченном по алфавиту порядке раньше, чем s2, и 1 — в противоположном случае.	<code>char s1[80],s2[80]; cout&lt;&lt;"s1="; cin.getline(s1,80); cout&lt;&lt;"s2="; cin.getline(s2,80); cout&lt;&lt;strcmp(s1,s2,6);</code>	<code>s1&gt;Hello,Russia! s2&gt;Hello, 0</code>

Таблица 8.2: Функции работы со строками, библиотека `stdlib.h`

Прототип функции	Описание функции	Пример использования	Результат
<code>double atof(const char*s)</code>	Преобразует строку в вещественное число, в случае неудачного преобразования возвращается число 0.0	<code>char a[10]; cout&lt;&lt;"a="; cin&gt;&gt;a; cout&lt;&lt;"a="&lt;&lt;atof(a) &lt;&lt;endl;</code>	<code>a=23.57 a=23.57</code>
<code>int atoi(const char*s)</code>	Преобразует строку в целое число, в случае неудачного преобразования возвращается число 0	<code>char a[10]; cout&lt;&lt;"a="; cin&gt;&gt;a; cout&lt;&lt;"a="&lt;&lt;atoi(a) &lt;&lt;endl;</code>	<code>a=23 a=23</code>
<code>long atol(const char*s)</code>	Преобразует строку в длинное целое число, в случае неудачного преобразования возвращается число 0	<code>char a[10]; cout&lt;&lt;"a="; cin&gt;&gt;a; cout&lt;&lt;"a="&lt;&lt;atol(a) &lt;&lt;endl;</code>	<code>a=23 a=23</code>

Для преобразования числа в строку можно воспользоваться функцией `sprintf` из библиотеки `stdio.h`.

```
 sprintf(s, s1, s2);
```

Она аналогична описанной ранее функции `printf`, отличие состоит в том, что осуществляется вывод не на экран, а в выходную строку `s`.

Например, в результате работы следующих команд

```
char str[80];
sprintf (str, "%s %d %s", "С новым ", 2014, "годом!!!");
```

в переменную `str` будет записана строка С новым 2014 годом!!!.

### 8.3 Тип данных `string`

Кроме работы со строками, как с массивом символов, в C++ существует специальный тип данных `string`. Для ввода переменных этого типа можно использовать `cin`<sup>1</sup> или специальную функцию:

```
getline(cin,s);
```

Здесь `s` — имя вводимой переменной типа `string`.

При описании переменной типа `string` можно сразу присвоить ей значение:

```
string var(s);
```

Здесь `var` — имя переменной типа `string`, `s` — строковая константа. В результате этого оператора создается переменная `var` типа `string` и в нее записывается значение из строковой константы `s`. Например,

```
string v("Hello");
```

Создается строка `v`, в которую записывается значение `Hello`.

Доступ к *i*-му элементу строки осуществляется стандартным образом:

```
имя_строки[номер_элемента];
```

Над строками типа `string` определены следующие *операции*:

- *присваивания*, например `s1=s2`;
- *обединение строк* (`s1+=s2` или `s1=s1+s2`) — добавляет к строке `s1` строку `s2`, результат хранится в строке `s1`, например:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string a,b;
    cout<<"a="; getline(cin,a);
    cout<<"b="; getline(cin,b);
    a+=b;
    cout<<"a="<<a<<endl;
    return 0;
}
```

- *сравнение строк* на основе лексикографического порядка: `s1==s2`, `s1!=s2`, `s1<s2`, `s1<=s2`, `s1>s2`, `s1>=s2` — результатом операций сравнения будет логическое значение;

При *обработке строк* типа `string` можно использовать следующие функции<sup>2</sup>:

`s.length()` — возвращает длину строки `s`;

`s.substr(pos, length)` — возвращает подстроку из строки `s`, начиная с номера `pos` длиной `length` символов;

`s.empty()` — возвращает значение `true`, если строка `s` пуста, `false` — в противном случае;

`s.insert(pos, s1)` — вставляет строку `s1` в строку `s`, начиная с позиции `pos`;

`s.remove(pos, length)` — удаляет из строки `s` подстроку `length` длиной `pos` символов;

---

<sup>1</sup>При работе с командой `cin`, как отмечалось ранее, ввод осуществляется до пробела.

<sup>2</sup>В описанных ниже функциях строки `s` и `s1` должны быть типа `string`.

`s.find(s1, pos)` — возвращает номер первого вхождения строки `s1` в строку `s`, поиск начинается с номера `pos`, параметр `pos` может отсутствовать, в этом случае поиск идет с начала строки;

`s.findfirst(s1, pos)` — возвращает номер первого вхождения любого символа из строки `s1` в строку `s`, поиск начинается с номера `pos`, параметр `pos` может отсутствовать, в этом случае поиск идет с начала строки.

**Задача 8.1.** Некоторый текст хранится в файле `text.txt`. Подсчитать количество строк и слов в тексте.

Предлагаем читателю самостоятельно разобраться в приведенном программном коде.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <iomanip>
using namespace std;
int main()
{
    ifstream f;
    int p, j, i, kol, m, n=0;
    string S[10];
    f.open("text.txt");
    if (f)
    {
        while (!f.eof())
        {
            getline(f, S[n]);
            cout<<S[n]<<"\n";
            n++;
        }
        f.close();
        cout<<endl;
        cout<<"Количество строк в тексте - "<<n<<endl;
        for (kol=0, i=0; i<n; i++)
        {
            m=S[i].length();
            S[i]+=" ";
            for (p=0; p<m;)
            {
                j=S[i].find(" ", p);
                if (j!=0) {kol++; p=j+1;}
                else break;
            }
        }
        cout<<"Количество слов в тексте - "<<kol<<endl;
    }
    else cout<<"File not found"<<endl;
    return 0;
}
```

Результаты работы программы:

Если видим, что с картины  
Смотрят кто-нибудь на нас,  
Или принц в плаще старинном,  
Или в робе верхолаз,  
Лётчик или балерина,  
Или Колыка, твой сосед,  
Обязательно картина  
Называется портрет.

Количество строк в тексте - 8

Количество слов в тексте - 29

## 8.4 Задачи для самостоятельного решения

Разработать программу на языке C++ для следующих заданий:

1. Подсчитать количество слов в каждой строке текста.
2. Подсчитать количество символов в тексте.
3. Подсчитать количество точек в тексте.
4. Подсчитать количество пробелов в тексте.
5. Удалить из теста все пробелы.
6. Удалить из теста все точки.
7. Вставить вместо каждого пробела восклицательный знак.
8. Вставить перед каждым восклицательным знаком вопросительный.
9. Определить содержит ли текст хотя бы один восклицательный знак и в какой строке.
10. Подсчитать количество слов в четных строках текста.
11. Найти номер самой длинной строки текста.
12. Променять местами первую и последнюю строки текста.
13. Определить, есть ли в тексте пустые строки.
14. Определить содержит ли текст хотя бы пару соседних одинаковых строк.
15. Найти самую короткую строку текста и заменить ее фразой «С новым годом!».
16. Найти самую длинную строку текста и заменить ее пустой строкой.
17. Определить количество слов в нечетных строках текста.
18. Определить количество пробелов в четных строках текста.
19. Определить количество предложений в тексте, учитывая, что предложение заканчивается точкой, вопросительным или восклицательным знаком.
20. Поменять местами самую длинную и самую короткую строки текста.
21. Вывести на печать первое предложение текста, учитывая что оно заканчивается точкой.
22. Определить количество пробелов в нечетных строках текста.
23. Удалить из теста все восклицательные и вопросительные знаки.
24. Определить содержит ли текст хотя бы один вопросительный знак и в какой строке.
25. Добавить в начало каждой строки текста ее номер и пробел.

# Глава 9

## Структуры в языке C++

В этой главе дано описание структурного типа данных. Этот тип предназначен для представления сложных данных и создания новых типов. Приведены примеры использования структур для работы с комплексным числом.

Описана библиотека языка C++, позволяющая работать с комплексными числами.

### 9.1 Общие сведения о структурах

Из предыдущих глав известно, что массив это переменная для хранения множества данных одного типа. Если возникает необходимость обрабатывать разнородную информацию как единое целое, то применяют тип данных *структурь*. Он позволяет сгруппировать объекты различных типов данных под одним именем.

Для того, чтобы объявить переменные *структурного типа* в начале нужно задать новый тип данных, указав *имя структуры* и ее *элементы*. Элементы структуры называются *полями*, и могут иметь любой тип данных кроме типа этой же структуры. Далее приведен пример создания структурного типа **student** полями которого являются фамилия студента, шифр группы, год начала обучения и оценки по четырем предметам:

```
struct student
{
    //Поля структуры:
    char fio [30];
    char group [8];
    int year;
    int informatika , math , fizika , history ;
}
```

На основании созданного структурного типа данных можно *описать переменные* типа **student**:

```
student Vasya; //Переменная Vasya типа student.
student ES[50]; //Массив, элементы которого имеют тип student.
student *x; //Указатель на тип данных student.
```

*Обращаются к полям* переменной структурного типа так:

**имя\_структурь.поле**

Например,

```
Vasya.year; //Обращение к полю year переменной Vasya.  
ES[4].math; //Обращение к полю math элемента ES[4].
```

**Задача 9.1.** Задано  $n$  комплексных чисел, которые хранятся в двоичном файле. Найти значение наибольшего модуля среди заданных чисел.

Напомним, что *комплексные числа* это числа вида  $z = a + b \cdot i$ , где  $a$  и  $b$  — *действительные* числа, а  $i$  — *мнимая единица*,  $i^2 = -1$ . Комплексное число расширяет понятие *действительного числа*. Если действительное число — это любая точка на числовой прямой, то под комплексным числом понимают точку на плоскости (рис. 9.1). Модуль комплексного числа  $z$  вычисляют по формуле  $|z| = \sqrt{a^2 + b^2}$ .

Для решения задачи 9.1 разработаны две программы. Первая создает файл исходных данных, вторая получает из него информацию и обрабатывает ее в соответствии с поставленной задачей.

Далее приведен текст программы создания двоичного файла с  $n$  комплексными числами. В файл `complex.dat` будет записано число  $n$ , а затем последовательно комплексные числа.

```
#include <iostream>  
#include <fstream>  
using namespace std;  
int main()  
{  
    //Структура Комплексное число.  
    struct complex  
    {  
        //Поля структуры:  
        double Re; //Действительная часть.  
        double Im; //Мнимая часть.  
    };  
    complex p; //Переменная для хранения комплексного числа.  
    int i, n;  
    FILE *f;  
    cout << "n="; cin >> n;  
    f=fopen ("complex.dat", "wb");  
    fwrite (&n, sizeof(int), 1, f);  
    for (i=0; i<n; i++)  
    {  
        cout << "Введите комплексное число\n";  
        //Ввод комплексного числа:  
        cin >> p.Re; //действительная часть,  
        cin >> p.Im; //мнимая часть.  
        //Вывод комплексного числа.  
        cout << p.Re << " + " << p.Im << "i" << endl;  
        //Запись комплексного числа в двоичный файл.  
        fwrite (&p, sizeof(complex), 1, f);  
    }  
    fclose (f);  
    return 0;  
}
```

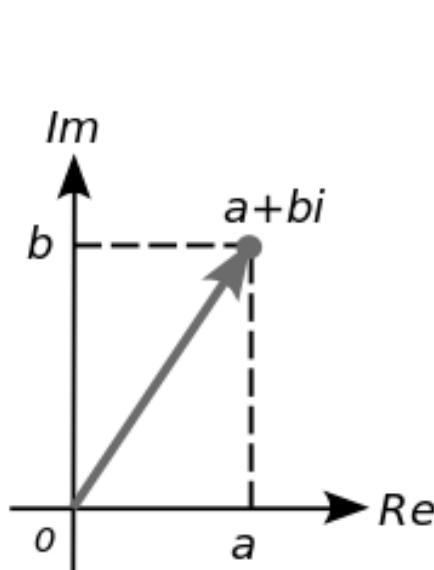


Рис. 9.1: Геометрическая модель комплексного числа  $a + b \cdot i$

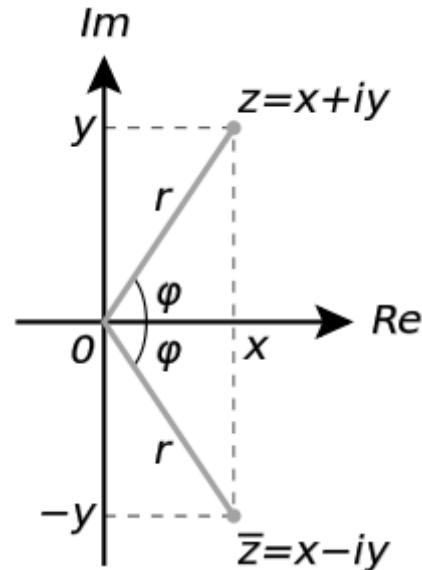


Рис. 9.2: Геометрическая интерпретация комплексно-сопряженного числа

Следующая программа считывает информацию из файла `complex.dat` — количество комплексных чисел в переменную `n`, а сами комплексные числа в массив `p`. Затем происходит поиск комплексного числа с максимальным модулем в массиве `p`.

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    struct complex
    {
        double Re;
        double Im;
    };
    complex *p;
    int i, n, nmax;
    double max;
    FILE *f;
    f=fopen("complex.dat","rb");
    fread(&n, sizeof(int), 1, f);
    p=new complex[n];
    fread(p, sizeof(complex), n, f);
    //Поиск комплексного числа с максимальным модулем
    max=sqrt(p[0].Re*p[0].Re+p[0].Im*p[0].Im);
    for(i=1,nmax=0;i<n;i++)
        if (sqrt(p[i].Re*p[i].Re+p[i].Im*p[i].Im)>max)
    {
        max=sqrt(p[i].Re*p[i].Re+p[i].Im*p[i].Im);
        nmax=i;
    }
}
```

```

    }
    cout<<"max = "<<max<<" \t  nmax = "<<nmax<<endl;
    fclose(f);
    return 0;
}

```

**Задача 9.2.** Даны два комплексных числа  $z_1$  и  $z_2$ . Выполнить над ними *основные операции*:

- сложение  $z_1 + z_2$ ,
- вычитание  $z_1 - z_2$ ,
- умножение  $z_1 \cdot z_2$ ,
- деление  $\frac{z_1}{z_2}$ ,
- возвведение в степень  $n$   $z_1^n$ ,
- извлечение корня  $n$ -й степени  $\sqrt[n]{z_1}$
- вычисление комплексного сопряженного числа  $\bar{z}_1$ .

Суммой двух комплексных чисел  $z_1 = a+i\cdot b$  и  $z_2 = c+i\cdot d$  называется комплексное число  $z = z_1 + z_2 = (a+c) + i \cdot (b+d)$ .

Разностью двух комплексных чисел  $z_1 = a+i\cdot b$  и  $z_2 = c+i\cdot d$  называется комплексное число  $z = z_1 - z_2 = (a-c) + i \cdot (b-d)$ .

Произведением двух комплексных чисел  $z_1 = a+i\cdot b$  и  $z_2 = c+i\cdot d$  называется комплексное число  $z = z_1 \cdot z_2 = (a \cdot c - b \cdot d) + i \cdot (b \cdot c + a \cdot d)$ .

Частным двух комплексных чисел  $z_1 = a+i\cdot b$  и  $z_2 = c+i\cdot d$  называется комплексное число

$$z = \frac{z_1}{z_2} = \frac{ac+bd}{c^2+d^2} + i \cdot \frac{bc-ad}{c^2+d^2}$$

Числом, сопряженным комплексному числу  $z = x+i\cdot y$ , называется число  $\bar{z} = x-i\cdot y$  (рис. 9.2).

Всякое комплексное число, записанное в алгебраической форме  $z = x+i\cdot y$ , можно записать в тригонометрической  $z = r(\cos \phi + i \cdot \sin \phi)$  или в показательной форме  $z = r \cdot e^{i \cdot \phi}$ , где  $r = \sqrt{x^2 + y^2}$  — модуль комплексного числа  $z$ ,  $\phi = \arctan \frac{y}{x}$  — его аргумент (рис. 9.2).

Для возведения в степень комплексного числа, записанного в тригонометрической форме  $z = r(\cos \phi + i \cdot \sin \phi)$ , можно воспользоваться формулой Муавра

$$z^n = r^n \cdot (\cos(n \cdot \phi) + i \cdot \sin(n \cdot \phi)).$$

Формула для извлечения корня  $n$ -й степени из комплексного числа  $z = r \cdot (\cos \phi + i \cdot \sin \phi)$  имеет вид

$$\sqrt[n]{z} = \sqrt[n]{r} \left( \cos \frac{\phi + 2\pi \cdot k}{n} + i \cdot \sin \frac{\phi + 2\pi \cdot k}{n} \right),$$

где  $n > 1$ ,  $k = 0, 1, \dots, n - 1$ .

Далее приведен текст программы реализующий алгоритм решения задачи 9.2. В программе описаны две структуры для работы с комплексными числами: структура `complex1` для представления комплексных чисел в алгебраической форме (`Re` — действительная часть комплексного числа, `Im` — его мнимая часть) и структура `complex2` для представления комплексных чисел в показательной

или тригонометрической форме (`Modul` — модуль комплексного числа, `Argum` — его аргумент). Кроме того в программе созданы функции, реализующие основные действия над комплексными числами, переход между различными формами представления комплексных чисел, а также ввод-вывод комплексных чисел.

```
#include <iostream>
#include <math.h>
using namespace std;
struct complex1
{
    float Re;
    float Im;
};
struct complex2
{
    float Modul;
    float Argum;
};
//Ввод числа в алгебраической форме
complex1 vvod1()
{
    complex1 temp;
    cout<<"Введите действительную часть числа\n";
    cin>>temp.Re;
    cout<<"Введите мнимую часть комплексного числа\n";
    cin>>temp.Im;
    return temp;
}
//Ввод числа в тригонометрической или показательной форме
complex2 vvod2()
{
    complex2 temp;
    cout<<"Введите модуль комплексного числа\n";
    cin>>temp.Modul;
    cout<<"Введите аргумент комплексного числа\n";
    cin>>temp.Argum;
    return temp;
}
//Вывод числа в алгебраической форме
void vivod(complex1 chislo)
{
    cout<<chislo.Re;
    if (chislo.Im>=0)
        cout<<" +"<<chislo.Im<<" i"<<endl;
    else
        cout<<" "<<chislo.Im<<" i"<<endl;
}
//Вывод числа в тригонометрической форме
void vivod(complex2 chislo)
{
    cout<<chislo.Modul<<" ("<<chislo.Argum<<" ) + i sin ("<<chislo.Argum<<" ))"<<endl;
}
//Перевод числа из тригонометрической формы в алгебраическую,
//pr определяет выводить или нет полученное число на экран.
complex1 perevod(complex2 chislo, bool pr=false)
{
    complex1 temp;
    temp.Re=chislo.Modul*cos(chislo.Argum);
    temp.Im=chislo.Modul*sin(chislo.Argum);
    if (pr) vivod(temp);
    return temp;
}
//Перевод числа из алгебраической формы в тригонометрическую,
```

```

//pr определяет выводить или нет полученное число на экран.
complex2 perevod(complex1 chislo, bool pr=false)
{
    complex2 temp;
    temp.Modul=sqrt(chislo.Re*chislo.Re+
    chislo.Im*chislo.Im);
    temp.Argum=atan(chislo.Im/chislo.Re);
    if (pr) vivod(temp);
    return temp;
}
//Функция сложения двух чисел в алгебраической форме,
//pr определяет выводить или нет число на экран.
complex1 plus1(complex1 chislo1,complex1 chislo2,bool pr=true)
{
    complex1 temp;
    temp.Re=chislo1.Re+chislo2.Re;
    temp.Im=chislo1.Im+chislo2.Im;
    if (pr) vivod(temp);
    return temp;
}
//Функция вычитания двух чисел в алгебраической форме,
//pr определяет выводить или нет число на экран.
complex1 minus1(complex1 chislo1,complex1 chislo2,bool pr=true)
{
    complex1 temp;
    temp.Re=chislo1.Re-chislo2.Re;
    temp.Im=chislo1.Im-chislo2.Im;
    if (pr) vivod(temp);
    return temp;
}
//Функция умножения двух чисел в алгебраической форме,
//pr определяет выводить или нет число на экран.
complex1 mult1(complex1 chislo1,complex1 chislo2,bool pr=true)
{
    complex1 temp;
    temp.Re=chislo1.Re*chislo2.Re-chislo1.Im*chislo2.Im;
    temp.Im=chislo1.Im*chislo2.Re+chislo1.Re*chislo2.Im;
    if (pr) vivod(temp);
    return temp;
}
//Функция деления двух чисел в алгебраической форме,
//pr определяет выводить или нет число на экран.
complex1 divide1(complex1 chislo1,complex1 chislo2,bool pr=true)
{
    complex1 temp;
    temp.Re=(chislo1.Re*chislo2.Re+chislo1.Im*chislo2.Im)/(chislo2.Re*chislo2.
    Re+chislo2.Im*chislo2.Im);
    temp.Im=(chislo1.Im*chislo2.Re-chislo1.Re*chislo2.Im)/(chislo2.Re*chislo2.
    Re+chislo2.Im*chislo2.Im);
    if (pr) vivod(temp);
    return temp;
}
//Функция возведения комплексного числа в алгебраической форме
//в целую степень n, pr определяет выводить или нет полученное число на экран.
complex1 pow1(complex1 chislo1, int n, bool pr=true)
{
    complex1 temp;
    complex2 temp2;
    float p=1;
    int i=1;
    temp2=perevod(chislo1, true); //Перевод числа в тригонометрическую форму.
    for (; i<=n; p*=temp2.Modul, i++);
    temp.Re=p*cos(n*temp2.Argum);
    temp.Im=p*sin(n*temp2.Argum);
    if (pr) vivod(temp);
}

```

```

    return temp;
}
//Функция извлечения корня степени n из комплексного числа
//в алгебраической форме, pr определяет выводить или нет
//полученные значения на экран. Функция возвращает ro и fi.
void sqrtN1(complex1 chislo1, int n, float * ro, float * fi, bool pr=true)
{
    complex1 temp;
    complex2 temp2;
    int i=0;
    temp2=perevod(chislo1, true); //Перевод числа в тригонометрическую форму.
    *ro=pow(temp2.Modul, (float)1/n);
    *fi=temp2.Argum;
    if (pr)
    {
        for (i=0; i<n; i++)
        {
            cout<<i<<"-е значение корня\n";
            temp.Re=*ro*cos((*fi+2*M_PI*i)/n);
            temp.Im=*ro*sin((*fi+2*M_PI*i)/n);
            vivod(temp);
        }
    }
}
int main()
{
    complex1 chislo1, chislo2; //Описание комплексных
    complex1 chislo5; //чисел в алгебраической форме.
    complex2 chislo3, chislo4; //Описание комплексных чисел в тригонометрической форме.
    float ro1, fi1;
    chislo1=vvod1(); //Ввод исходных данных
    chislo2=vvod1(); //в алгебраической форме.
    vivod(chislo1); //Вывод исходных данных
    vivod(chislo2); //в алгебраической форме.
    chislo3=perevod(chislo1, true); //Перевод чисел
    chislo4=perevod(chislo2, true); //в тригонометрическую форму и вывод их на экран.
    cout<<"Сумма чисел ";
    chislo5=plus1(chislo1, chislo2, true);
    cout<<"Разность чисел ";
    chislo5=minus1(chislo1, chislo2, true);
    cout<<"Произведение чисел ";
    chislo5=mult1(chislo1, chislo2, true);
    cout<<"Частное чисел ";
    chislo5=divide1(chislo1, chislo2, true);
    chislo5=powl(chislo1, 5, true); //Возведение числа в пятую степень.
    sqrtN1(chislo1, 5, &ro1, &fi1, true); //Извлечение корня пятой степени.
    return 0;
}

```

Результаты работы программы к задаче 9.2.

```

Введите действительную часть числа
5
Введите мнимую часть комплексного числа
-7
Введите действительную часть числа
11
Введите мнимую часть комплексного числа
1.85
5 -7 i
11 +1.85 i
8.60233 ( cos (-0.950547) + i sin (-0.950547))
11.1545 ( cos (0.166623) + i sin (0.166623))
Сумма чисел 16 -5.15 i
Разность чисел -6 -8.85 i

```

```

Произведение чисел 67.95 -67.75 i
Частное чисел 0.337961 -0.693203 i
8.60233 ( cos (-0.950547) + i sin (-0.950547))
1900 +47068 i
8.60233 ( cos (-0.950547) + i sin (-0.950547))
0-е значение корня
1.51018 -0.290608 i
1-е значение корня
0.743054 +1.34646 i
2-е значение корня
-1.05094 +1.12277 i
3-е значение корня
-1.39257 -0.652552 i
4-е значение корня
0.190285 -1.52606 i

```

## 9.2 Библиотеки для работы с комплексными числами

Работа с комплексными числами в C++ реализована с помощью библиотеки `complex`. Подключение этой библиотеки дает возможность применять операции `+`, `-`, `*`, `/` для работы не только с вещественными, но и с комплексными числами.

Перед подключением библиотеки `complex` обязательно необходимо подключить библиотеку `math.h`.

Для определения переменной типа комплексное число используется оператор `complex <тип_переменной> имя_переменной;`

Здесь `тип_переменной` — это любой допустимый в C++ числовой тип данных (`int`, `long int`, `double`, `float` и т. д.), описывающий действительную и мнимую части комплексного числа. Например,

```

complex <float> x,y,z[5],*r;
complex <double> a;
complex <int> a,b,c;

```

Для организации ввода-вывода комплексных чисел можно использовать библиотеку `iostream` и стандартные конструкции `cin`, `cout`. Например,

```

#include <iostream>
#include <math.h>
#include <complex>
using namespace std;
int main(int argc, char **argv)
{
    complex <double> b, c; //Описание комплексных чисел.
    cout<<"b=" ; cin>>b; //Ввод комплексного числа b.
    cout<<"c=" ; cin>>c; //Ввод комплексного числа c.
    cout<<"b/c=" <<b/c; //Вывод частного комплексных чисел
    return 0;
}

```

В результате получим:

```

b=(1.24,-6.12)
c=(9.01,-11.22)
b/c=(0.385567,-0.199105)

```

Обратите внимание, что при вводе комплексных чисел с клавиатуры действительная и мнимая части вводятся в скобках через запятую:

(действительная\_часть, мнимая\_часть)

Далее приведен пример присваивания комплексным переменным реальных значений при их описании:

```
complex <double> z (4.0, 1.0);
complex <int> r (4, -7);
```

Следующий пример демонстрирует как из двух числовых значений можно составить комплексное число:

```
#include <iostream>
#include <math.h>
#include <complex>
using namespace std;
int main(int argc, char **argv)
{
    double x1,y1;
    x1=-2.3;
    y1=8.1;
    complex <double> b (x1,y1); //Формирование комплексного числа b
                                //с действительной частью x1 и мнимой y1.
    cout<<"b^2=<<b*b; //Вывод квадрата комплексного числа.
    return 0;
}
```

В табл. 9.1 представлены основные математические функции для работы с комплексными числами.

Таблица 9.1: Основные функции комплексного аргумента

Прототип функции	Описание функции
double abs(complex z)	Возвращает модуль комплексного числа $z$ .
double arg(complex z)	Возвращает значение аргумента комплексного числа $z$ .
double asin(complex z)	Возвращает арксинус комплексного числа $z$ .
double atan(complex z)	Возвращает арктангенс комплексного числа $z$ .
complex conj(complex z)	Возвращает число комплексно сопряженное числу $z$ .
double cos(complex z)	Возвращает косинус комплексного числа $z$ .
double cosh(complex z)	Возвращает гиперболический косинус комплексного числа $z$ .
double cosh(complex z)	Возвращает экспоненту комплексного числа $z$ .
double imag(complex z)	Возвращает мнимую часть комплексного числа $z$ .
double log(complex z)	Возвращает натуральный логарифм комплексного числа $z$ .
double log10(complex z)	Возвращает десятичный логарифм комплексного числа $z$ .
double norm(complex z)	Возвращает квадрат модуля комплексного числа $z$ .
complex pow(complex x, complex y)	Возвращает степень комплексного числа $z$ .
complex polar(double mag, double angle)	Формирует комплексное число с модулем <i>mag</i> и аргументом <i>angle</i> .
double real(complex z)	Возвращает действительную часть комплексного числа $z$ .
complex sin(complex z)	Возвращает синус комплексного числа $z$ .
complex sinh(complex z)	Возвращает гиперболический синус комплексного числа $z$ .
complex sqrt(complex z)	Возвращает квадратный корень комплексного числа $z$ .
complex tan(complex z)	Возвращает тангенс комплексного числа $z$ .
complex tan(complex z)	Возвращает гиперболический тангенс комплексного числа $z$ .

Далее приведен текст программы демонстрирующей работу с некоторыми функциями из табл. 9.1. После текста программы показаны результаты ее работы.

```
#include <iostream>
#include <math.h>
#include <complex>
using namespace std;
int main()
{
    complex <double> x (4, -6);
    complex <double> y (-7, 2);
    cout<<"x*y="<<x*y<<endl;
    cout<<"sin(x)*cos(y)="\<<sin(x)*cos(y)<<endl;
    cout<<"conj(x)*ln(y)="\<<conj(x)*log(y)<<endl;
    cout<<"sh(y)="\<<sinh(y)<<endl;
    return 0;
}
```

Результаты работы программы с некоторыми функциями комплексного аргумента

```
x*y=(-16,50)
sin(x)*cos(y)=(-747.159,10.2102)
conj(x)*ln(y)=(-9.23917,23.364)
sh(y)=(228.18,498.583)
```

**Задача 9.3.** Вычислить  $y = (\sqrt{3} - i)^{20}$ ,  $z = \left(\frac{1+i\sqrt{3}}{1-i}\right)^{40}$ .

Если провести аналитические преобразования, то получим следующее:

$$y = 2^{19} \cdot (-1 + i \cdot \sqrt{3}), z = -2^{19} \cdot (1 + i \cdot \sqrt{3}).$$

Проверим эти вычисления с помощью программы на C++. Результаты работы программы подтверждают аналитические вычисления.

```
#include <iostream>
#include <math.h>
#include <complex>
using namespace std;
int main()
{
    complex <double> b(sqrt(3), -1), y;
    cout<<"b="\<<b;
    y=pow(b,20);
    cout<<"y="\<<y<<endl;
    cout<<real(y)/pow(2,19)<<"\t";
    cout<<imag(y)/pow(2,19)<<"\n";
    complex <double> a(1,sqrt(3)), c(1,-1), z;
    z=pow(a/c,40);
    cout<<"z="\<<z<<endl;
    cout<<real(z)/pow(2,19)<<"\t";
    cout<<imag(z)/pow(2,19)<<"\n";
    return 0;
}
```

Результаты работы программы к задаче 9.3:

```
b=(1.73205,-1)y=(-524288,908093)
-1 1.73205
z=(-524288,-908093)
-1 -1.73205
```

*Операции с массивами*, элементами которых являются комплексные числа, осуществляются также, как и с обычными переменными. В качестве примера рассмотрим следующие задачи.

**Задача 9.4.** Написать программу умножения матриц комплексных чисел. Матрицы  $A$  и  $B$  имеют вид:

$$A = \begin{pmatrix} 1 + 2 \cdot i & 2 + 3 \cdot i & 3 + 1.54 \cdot i & 4 - 7.2 \cdot i \\ 2 + 5 \cdot i & 3 + 7 \cdot i & 4 + 10 \cdot i & 5 + 14 \cdot i \\ 1.5 + 3.25 \cdot i & 1.7 - 3.94 \cdot i & 6.23 + 11.17 \cdot i & -4.12 + 3.62 \cdot i \end{pmatrix},$$

$$B = \begin{pmatrix} 6.23 - 1.97 \cdot i & 0.19 + 0.22 \cdot i & 0.16 + 0.28 \cdot i & 3.4 + 1.95 \cdot i & 2.20 - 0.18 \cdot i \\ 0.22 + 0.29 \cdot i & 11 + 12 \cdot i & 6.72 - 1.13 \cdot i & 16 + 18 \cdot i & 34 + 66 \cdot i \\ 5 + 1 \cdot i & 1.4 - 1.76 \cdot i & 4.5 + 2.3 \cdot i & 296 + 700 \cdot i & 4.2 + 1.03 \cdot i \\ -3.4 - 2.61 \cdot i & 1 + 11 \cdot i & 2 + 23 \cdot i & 3 - 35 \cdot i & 4 + 47 \cdot i \end{pmatrix}.$$

Пусть исходные данные хранятся в файле `abc.txt`. Данные к задаче 9.4, содержащиеся в файле `abc.txt`:

```
3 4 5
(1,2) (2,3) (3,1.54) (4,-7.2)
(2,5) (3,7) (4,10) (5,14)
(1.5,3.25) (1.7,-3.94) (6.23,11.17) (-4.12,3.62)

(6.23,-1.97) (0.19,0.22) (0.16,0.28) (3.4,1.95) (2.20,-0.18)
(0.22,0.29) (11,12) (6.72,-1.13) (16,18) (34,66)
(5,1) (1.4,-1.76) (4.5,2.3) (296,700) (4.2,1.03)
(-3.14,-2.61) (1,11) (2,23) (3,-35) (4,47)
```

Далее приведен текст программы реализующий алгоритм решения задачи 9.4.

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <complex>
using namespace std;
int main()
{
    int i, j, p, N, M, K;
    complex<float> **A, **B, **C;
    ifstream f;
    ofstream g;
    f.open("abc.txt");
    f>>N>>M>>K;
    cout << "N=" << N << "M=" << M << "K=" << K << endl;
    A = new complex<float> *[N];
    for (i = 0; i < N; A[i] = new complex<float> [M], i++);
    B = new complex<float> *[M];
    for (i = 0; i < M; B[i] = new complex<float> [K], i++);
    C = new complex<float> *[N];
    for (i = 0; i < N; C[i] = new complex<float> [K], i++);
    for (i = 0; i < N; i++)
        for (j = 0; j < M; f >> A[i][j], j++);
    cout << "Матрица A\n";
    for (i = 0; i < N; cout << endl, i++)
        for (j = 0; j < M; cout << A[i][j] << "\t", j++);
    for (i = 0; i < M; i++)
        for (j = 0; j < K; f >> B[i][j], j++);
    cout << "Матрица B\n";
    for (i = 0; i < M; cout << endl, i++)
        for (j = 0; j < K; cout << B[i][j] << "\t", j++);
    for (i = 0; i < N; i++)
        for (j = 0; j < K; j++)
            for (C[i][j] = p = 0; p < M; p++)
                C[i][j] += A[i][p] * B[p][j];
    f.close();
}
```

```

cout<<"Матрица C\n";
for(i=0;i<N;cout<<endl,i++)
    for(j=0;j<K;cout<<C[i][j]<<"\t",j++);
g.open("result.txt");
g<<"Матрица C=A*B\n";
for(i=0;i<N;g<<endl,i++)
    for(j=0;j<K;g<<C[i][j]<<"\t",j++);
g.close();
return 0;
}

```

Результат умножения матриц из задачи 9.4 (файл `result.txt`):

```

Матрица C=A*B
(-8.152,34.598) (75.8604,91.276) (199.988,109.93) (-452.5,2486.99) (237.974,406.978)
(51.78,26.61) (-177.52,190.35) (-290.01,242.21) (-5391.95,5813.9) (-986.2,783.76)
(59.6291,78.3851) (49.9912,-59.0193) (-82.8542,-50.3838) (-5763.7,7803.92) (149.766,-140.709)

```

**Задача 9.5.** Заданы матрицы  $A$  и  $B$ . Необходимо вычислить матрицу  $A^{-1}$  обратную к матрице  $A$ , найти определитель  $|A|$  матрицы  $A$  и решить матричное уравнение  $A \cdot X = B$ , где  $X = A^{-1} \cdot B$ . Матрицы  $A$  и  $B$  имеют вид:

$$A = \begin{pmatrix} 1 + 2 \cdot i & 2 + 3 \cdot i & 3 + 1.54 \cdot i \\ 2 + 5 \cdot i & 3 + 7 \cdot i & 4 + 10 \cdot i \\ 1.5 + 3.25 \cdot i & 1.7 - 3.94 \cdot i & 6.23 + 11.17 \cdot i \end{pmatrix},$$

$$B = \begin{pmatrix} 1.5 + 3.25 \cdot i & 1.7 - 9.34 \cdot i & 6.23 + 11.17 \cdot i \\ 0.11 + 8.22 \cdot i & 0.34 - 18.21 \cdot i & 1 - 7 \cdot i \\ 1 + 5 \cdot i & 7 - 13 \cdot i & 12 + 89 \cdot i \end{pmatrix}.$$

Для хранения исходных данных создадим текстовый файл `abc2.txt` следующего содержания:

```

3
(1,2) (2,3) (3,1.54)
(2,5) (3,7) (4,10)
(1.5,3.25) (1.7,-9.34) (6.23,11.17)

(1.5,3.25) (1.7,-9.34) (6.23,11.17)
(0.11,8.22) (0.34,-18.21) (1,-7)
(1,5) (7,-13) (12,89)

```

Текст программы, реализующий поставленную задачу представлен ниже.

```

#include <iostream>
#include <fstream>
#include <math.h>
#include <complex>
using namespace std;
//Решение СЛАУ с комплексными коэффициентами
int SLAU(complex <float> **matrica_a, int n, complex <float> *massiv_b,
          complex <float> **x)
{
    int i,j,k,r;
    complex <float> c,M,s;
    float max;
    complex <float> **a, *b;
    a=new complex <float> *[n];
    for(i=0;i<n;i++)
        a[i]=new complex <float> [n];
    b=new complex <float> [n];
    for(i=0;i<n;i++)

```

```

for ( j=0; j<n; j++)
    a[ i ][ j]=matrica_a[ i ][ j ];
for ( i=0; i<n; i++)
    b[ i]=massiv_b[ i ];
for (k=0; k<n; k++)
{
    max= abs(a[ k ][ k ]) ;
    r=k;
    for ( i=k+1; i<n; i++)
        if ( abs(a[ i ][ k ])>max)
        {
            max=abs(a[ i ][ k ]) ;
            r=i;
        }
    for ( j=0; j<n; j++)
    {
        c=a[ k ][ j ];
        a[ k ][ j]=a[ r ][ j ];
        a[ r ][ j]=c;
    }
    c=b[ k ];
    b[ k ]=b[ r ];
    b[ r ]=c;
    for ( i=k+1; i<n; i++)
    {
        for ( M=a[ i ][ k ]/a[ k ][ k ], j=k; j<n; j++)
            a[ i ][ j]-=M*a[ k ][ j ];
        b[ i ]-=M*b[ k ];
    }
}
if ( abs(a[ n-1 ][ n-1 ])==0)
    if ( abs(b[ n-1 ])==0)
        return -1;
    else return -2;
else
{
    for ( i=n-1; i>=0; i--)
    {
        for ( s=0, j=i+1; j<n; j++)
            s+=a[ i ][ j ]*x[ j ];
        x[ i ]=(b[ i ]-s)/a[ i ][ i ];
    }
return 0;
}
for ( i=0; i<n; i++)
    delete [] a[ i ];
delete [] a;
delete [] b;
}
//Вычисление обратной матрицы с комплексными коэффициентами
int INVERSE( complex <float> **a, int n, complex <float> **y)
{
    int i,j,res;
    complex <float> *b, *x;
    b=new complex <float> [n];
    x=new complex <float> [n];
    for ( i=0; i<n; i++)
    {
        for ( j=0; j<n; j++)
            if ( j==i )
                b[ j ]=1;
            else b[ j ]=0;
            res=SLAU(a,n,b,x);
        if ( res!=0 )
            break;
    }
}

```

```

    else
        for(j=0;j<n;j++)
            y[j][i]=x[j];
    }
    delete [] x;
    delete [] b;
    if(res!=0)
        return -1;
    else
        return 0;
}
//Вычисление определителя матрицы с комплексными коэффициентами
complex <float> determinant(complex <float> **matrica_a, int n)
{
    int i,j,k,r;
    complex <float> c,M,s,det=1;
    complex <float> **a;
    float max;
    a=new complex <float> *[n];
    for(i=0;i<n;i++)
        a[i]=new complex <float> [n];
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            a[i][j]=matrica_a[i][j];
    for(k=0;k<n;k++)
    {
        max=abs(a[k][k]);
        r=k;
        for(i=k+1;i<n;i++)
            if(abs(a[i][k])>max)
            {
                max=abs(a[i][k]);
                r=i;
            }
        if(r!=k) det=-det;
        for(j=0;j<n;j++)
        {
            c=a[k][j];
            a[k][j]=a[r][j];
            a[r][j]=c;
        }
        for(i=k+1;i<n;i++)
            for(M=a[i][k]/a[k][k],j=k;j<n;j++)
                a[i][j]-=M*a[k][j];
    }
    for(i=0;i<n;i++)
        det*=a[i][i];
    return det;
    for(i=0;i<n;i++)
        delete [] a[i];
    delete [] a;
}
//Умножение матриц с комплексными коэффициентами
void umn(complex <float> **a, complex <float> **b, complex <float> **c, int n, int m, int k)
{
    int i,j,p;
    for(i=0;i<n;i++)
        for(j=0;j<k;j++)
            for(c[i][j]=p=0;p<m;p++)
                c[i][j]+=a[i][p]*b[p][j];
}
int main()
{
    int i,j,N;

```

```

complex <float> **A,**B,**X, **Y;
ifstream f;
ofstream g;
f.open("abc2.txt");
f>>N;
cout<<"N="<<N<<endl;
A=new complex <float> *[N];
for(i=0;i<N; i++)
    A[i]=new complex <float> [N];
B=new complex <float> *[N];
for(i=0;i<N; i++)
    B[i]=new complex <float> [N];
X=new complex <float> *[N];
for(i=0;i<N; i++)
    X[i]=new complex <float> [N];
Y=new complex <float> *[N];
for(i=0;i<N; i++)
    Y[i]=new complex <float> [N];
for(i=0;i<N; i++)
    for(j=0;j<N; j++)
        f>>A[i][j];
cout<<"Матрица A\n";
for(i=0;i<N; cout<<endl , i++)
    for(j=0;j<N; j++)
        cout<<A[i][j]<<"\t";
for(i=0;i<N; i++)
    for(j=0;j<N; j++)
        f>>B[i][j];
cout<<"Матрица B\n";
for(i=0;i<N; cout<<endl , i++)
    for(j=0;j<N; j++)
        cout<<B[i][j]<<"\t";
if(!INVERSE(A, N, X))
{
    cout<<"Обратная матрица\n";
    for(i=0;i<N; cout<<endl , i++)
        for(j=0;j<N; j++)
            cout<<X[i][j]<<"\t";
}
else cout<<"Не существует обратной матрицы\n";
cout<<"Определитель="<<determinant(A,N);
umn(X,B,Y,N,N,N);
cout<<"\n Решение матричного уравнения \n";
for(i=0;i<N; cout<<endl , i++)
    for(j=0;j<N; j++)
        cout<<Y[i][j]<<"\t";
return 0;
}

```

Результат работы программы к задаче 9.5:

```

N=3
Матрица А
(1,2) (2,3) (3,1.54)
(2,5) (3,7) (4,10)
(1.5,3.25) (1.7,-9.34) (6.23,11.17)
Матрица В
(1.5,3.25) (1.7,-9.34) (6.23,11.17)
(0.11,8.22) (0.34,-18.21) (1,-7)
(1,5) (7,-13) (12,89)
Обратная матрица
(-0.495047,-0.748993) (0.325573,0.182901) (-0.0340879,-0.0958618)
(0.125154,0.0765918) (-0.058179,-0.0728342) (0.00208664,0.0685887)
(0.157733,0.322512) (-0.0859214,-0.127174) (0.0143863,-0.000518244)
Определитель=(7.50219,-208.261)

```

**Решение матричного уравнения**  
 $(0.669246, -0.302366) (-5.88068, -2.74393) (15.0106, -16.4762)$   
 $(0.190248, 0.114415) (0.488295, 0.448942) (-6.72319, 3.21833)$   
 $(0.241332, 0.347549) (1.02932, 0.405788) (-3.37716, 5.51956)$

## 9.3 Задачи для самостоятельного решения

### 9.3.1 Структуры. Операции над комплексными числами

Разработать программу на языке C++ для решения следующей задачи. Даны комплексные числа  $a = \alpha + \beta \cdot i$ ,  $b = \gamma + \delta \cdot i$  и  $c = \lambda + \mu \cdot i$ . Найти комплексное число  $d = \varphi + \psi \cdot i$  по формуле представленной в табл. 9.2.

Таблица 9.2: Задания для решения задачи о комплексных числах

Вариант	Формула для вычислений
1	$d = a^2 \cdot \frac{a+b}{a-b \cdot c}$
2	$d = a^2 \cdot \frac{(a+b-c)}{b}$
3	$d = \frac{a^3 \cdot b}{b+c} \cdot  a - b $
4	$d = (a - c)^2 \cdot \frac{(a+b)}{a}$
5	$d = \frac{a^2 \cdot b}{a+c} \cdot (a - b)$
6	$d = (a + c)^2 \cdot \frac{(a-b)}{(a-c)}$
7	$d = \frac{a \cdot b^2 + c}{a-b}$
8	$d = (a + b - c)^2 \cdot \frac{b}{a}$
9	$d = \frac{a \cdot b^3 - c}{a+b}$
10	$d = (a + b - c) \cdot \frac{b^2}{c}$
11	$d = \frac{a^3 \cdot b + c}{a-b}$
12	$d = \frac{(a^2 + b - c^3)}{a}$
13	$d = \frac{a + b^2 - c}{a + b + c}$
14	$d = \frac{(a + b^2 - c)}{(a + b^2)}$
15	$d = \left( \frac{a+b+c}{a-b+c} \right)^2$
16	$d = (a - b - c) \cdot \frac{(b+c)}{(b-c)}$
17	$d = \left( \frac{a+b^3+c}{a-b^2-c} \right)$
18	$d = (a + b + c) \cdot \frac{(b-a)}{(b-c)}$
19	$d = \left( \frac{a-b-c}{a-b^2+c^3} \right)$
20	$d = \frac{(a^2 - b + c)}{(b - c^3)}$
21	$d = \frac{(a+b+c)^2}{a-b-c}$

Таблица 9.2 — продолжение

Вариант	Формула для вычислений
22	$d = (a + b + c) \cdot \frac{(b+c)^2}{(b-c)^3}$
23	$d = \frac{(a^2+b-c) \cdot a}{b}$
24	$d = \frac{(\frac{b}{c}+b \cdot c)}{(a-c)}$
25	$d = (a^2 - \frac{b}{c}) \cdot \frac{(a+c)}{(a-c)}$

### 9.3.2 Работа с библиотекой комплексных чисел

Разработать программу на языке C++ для решения следующей задачи:

- Для заданной матрицы комплексных чисел  $A(n \times n)$  найти  $B = 3 \cdot A^2 + A^T$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = (2 - 3 \cdot i) \cdot A \cdot B + B^T$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = \Delta \cdot A - A^2$ , где  $\Delta = |B|$ .
- Для заданной матрицы комплексных чисел  $A(n \times n)$  найти  $C = (3.2 + 1.8 \cdot i) \cdot A^T - A^2$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = (3.5 \cdot i) \cdot A \cdot B^T - B$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = \frac{\Delta}{2} \cdot A^T + A^2$ , где  $\Delta = |B|$ .
- Для заданной матрицы комплексных чисел  $D(k \times k)$  найти  $C = (3.2 + 1.8 \cdot i) \cdot D^2 - (5.2 \cdot i) \cdot D^T$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = A \cdot B^T + A \cdot B$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = (\Delta \cdot A - A^T) \cdot A$ , где  $\Delta = |B|$ .
- Для заданной матрицы комплексных чисел  $F(m \times m)$  найти  $C = 2.3 \cdot (F^2 + F)^T$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = (-2 + 3.5 \cdot i) \cdot (A - B^T)^2$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = \Delta \cdot (A^2 + A^T)$ , где  $\Delta = |B|$ .
- Для заданной матрицы комплексных чисел  $D(k \times k)$  найти  $C = (8.1 \cdot i) \cdot (D^2 - (1.2 \cdot i) \cdot D^T)$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = (-1.5 \cdot i) \cdot (A^T + B^T)^2$ .
- Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = \Delta \cdot (A^T + A)^2$ , где  $\Delta = |B|$ .
- Для заданной матрицы комплексных чисел  $D(k \times k)$  найти  $C = (D^T - (1.2 \cdot i)) \cdot D$ .

17. Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = (A^2 + B^2)^T$ .
18. Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = \Delta \cdot (A^T + A) \cdot A$ , где  $\Delta = |B|$ .
19. Для заданной матрицы комплексных чисел  $F(m \times m)$  найти  $C = -3.3 \cdot (F^T - (2 \cdot i) \cdot F)^2$ .
20. Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = (A \cdot B + B \cdot A)^T$ .
21. Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = A - \Delta \cdot A \cdot A^T$ , где  $\Delta = |B|$ .
22. Для заданной матрицы комплексных чисел  $F(m \times m)$  найти  $C = F^T + (3 \cdot i) \cdot F^2$ .
23. Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(n \times n)$  найти  $C = ((A + B)^2)^T$ .
24. Для заданных матриц комплексных чисел  $A(n \times n)$  и  $B(m \times m)$  найти  $C = \Delta \cdot (A^2 - A^T)$ , где  $\Delta = |B|$ .
25. Для заданной матрицы комплексных чисел  $D(k \times k)$  найти  $C = (D^T + (5 - 1.3 \cdot i) \cdot D)^2$ .

## Глава 10

### Возникновение объектного подхода в программировании

Первые программы, создававшиеся в те времена, когда значения битов в регистрах переключались тумблерами на системной консоли и тут же отображались загораящимися индикаторами — эти первые программы были чрезвычайно просты. Писали их непосредственно в машинных кодах, или, в лучшем случае, на ассемблере — языке, заменяющем коды машинных команд буквенными мнемониками. В последствии, по мере усложнения компьютеров и увеличения размеров программ, отслеживать возникающие ошибки становилось все труднее. Поэтому стала возрастать популярность языков программирования высокого уровня, а число программ, написанных целиком на языке машинных команд, наоборот, начало сокращаться. Языки высокого уровня обеспечивали более высокий уровень абстракции, приближая конструкции и операторы языка к понятиям, которыми оперирует человек.

Исторически одним из первых языков высокого уровня был Фортран, завоевавший огромную популярность и до сих пор используемый иногда в научных и инженерных расчетах. Подход к программированию, на котором был основан и он, и многие другие ранние языки, получил название *процедурного программирования*. В рамках этого подхода в программе отдельно хранятся *процедуры* — блоки кода, каждый из которых выполняет какое-то самостоятельное действие, и *переменные* — блоки данных (см. рис. 10.1), к которым обращаются процедуры для получения исходных значений и для сохранения результата. Такая четко структурированная программа создавала меньше возможностей не заметить ошибку. Поэтому производительность труда программистов, освоивших процедурную парадигму, ощутимо вырастала, а вместе с производительностью труда вырастали размеры программ и их функциональные возможности. Код серьезных программных продуктов все чаще писался коллективно, и скоро процедурный подход перестал казаться таким уж защищенным от ошибок. Например, нередко возникали ситуации, когда несколько программистов одинаково назы-

вали свои переменные, т. е. фактически использовали одну и ту же глобальную переменную в разных целях, в результате чего ее значение хаотично менялось при вызове разных процедур.

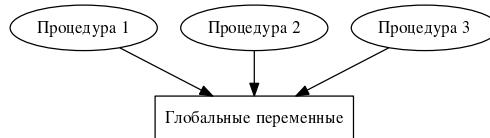


Рис. 10.1: Процедурный подход к программированию

Процедурный подход претерпел ряд модернизаций, более современные языки высокого уровня заимствовали некоторые принципы *функционального программирования* (одним из удачных примеров такого синтеза является язык С), большие программы делились на модули, а фирмы вводили собственные строгие политики в области оформления программного кода. Но в большой программе по-прежнему было слишком трудно разобраться и слишком просто запутаться, поэтому проблема все равно оставалась.

*Объектный подход* родился как следующий важный шаг на пути качественного написания больших программ. В нем предлагается разделять программу на самостоятельные части — *объекты*, наделенные собственными свойствами, текущим состоянием, и умеющие взаимодействовать друг с другом и с окружающей средой — примерно так, как это происходит у объектов реального мира.

В упрощенном виде такая парадигма получила название *объектно-ориентированного программирования* (ООП) — подхода, который позволяет использовать в программе объекты и даже поощряет эту практику, но не требует, чтобы программа состояла из одних только *объектов*.

Классическое определение объекта звучит следующим образом:

*Объект — это осозаемая сущность, которая четко проявляет свое поведение.*

Читателю, для которого объектный подход к программированию внове, такое определение наверняка покажется слишком туманным. Позже конкретные примеры прояснят ситуацию, а пока поговорим о внутреннем устройстве объекта.

Объект состоит из следующих трех частей:

- имя объекта;
- состояние (переменные состояния);
- методы (операции).

На рисунке 10.2 изображены два объекта с именами «Объект 1» и «Объект 2». Первый объект имеет две переменные состояния и три метода, в то время как второй объект обходится одной единственной переменной состояния и двумя методами.

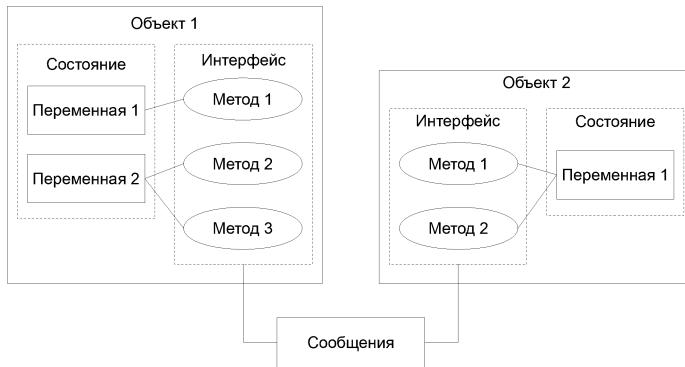


Рис. 10.2: Объектный подход к программированию

Интерфейс объекта с окружающей средой (пользователем, остальной частью программы, операционной системой и т. д.) полностью осуществляется методами: к состоянию объекта нет другого доступа извне, кроме как через его методы. Например, если объект должен передавать окружающей среде информацию о значении одной из своих переменных состояния — для этого создают специальный метод.

Закрытость внутреннего состояния объекта от окружающей среды известна также как свойство *инкапсуляции*. Инкапсуляция означает, что объект содержит внутри себя данные и методы, оперирующие этими данными. Фактически, для окружающей среды объект представляет собой аналог «черного ящика»: принимает входные воздействия и выдает в качестве реакции на них выходные, но при этом никак не проявляет свою внутреннюю структуру.

Для взаимодействия друг с другом объекты обмениваются *сообщениями*, причем объект, получивший сообщение, может либо проигнорировать сообщение, либо выполнить содержащуюся в нем команду (с помощью какого-либо из своих методов).

Однотипные объекты образуют *класс*. Под однотипными объектами мы понимаем такие объекты, у которых одинаковы наборы методов и переменных состояния. При этом объекты, принадлежащие одному классу, имеют разные имена и, вероятно, разные значения переменных состояния. Например, можно придумать класс «студент», объектами которого будут конкретные студенты вуза. Объект класса «студент» должен иметь переменные состояния, в которых содержится информация о конкретном студенте: Ф.И.О., номер студенческой группы, домашний адрес, список изучаемых дисциплин и т. д. Конкретный список переменных зависит от задачи, для решения которой создается программа. Так, если поставлена задача автоматизировать работу университетской библиотеки, то объекты класса «студент» определенно должны содержать информацию о книгах, взятых на абонемент конкретным студентом. Если автоматизируется учет успеваемости,

то в списке книг нет необходимости, но зато в состояние объекта обязательно должны быть включены оценки по изучаемым дисциплинам.

Более того, когда мы рассматриваем сущности реального мира, с которыми должна иметь дело создаваемая программа, мы можем назначить некую сущность на роль объекта или на роль класса объектов, также в зависимости от конкретной задачи. Представим себе, что одна из подзадач программы — систематизировать социальные роли, такие как «домохозяйка», «пенсионер», «студент» (например, для учета доходов и льгот). Вполне вероятно, что в такой программе «социальная роль» будет объявлена классом, а сущность «студент» будет всего лишь одним из объектов.

Иными словами, нет одинакового для всех ситуаций правила, по которому сущность делают объектом или классом объектов. Всегда необходимо исходить из большей целесообразности.

В реальном мире из родственных по смыслу сущностей часто можно составить иерархию «от общего к частному». Такие отношения в объектно-ориентированном подходе называются *наследованием*. Из двух классов, находящихся в отношении наследования, более общий класс называется *базовым* или *родительским* классом, а класс, представляющий собой более частный случай, называется *дочерним* или *производным* классом. Производный класс может заимствовать атрибуты (свойства и методы) базового класса. Это означает, что если в программе используются родственные по смыслу классы, обладающие некоторыми одинаковыми свойствами и методами — лучше определить один базовый класс, находящийся в вершине иерархии, и разместить дублирующиеся свойства и методы в нем. В этом случае производные классы смогут автоматически унаследовать эти атрибуты от базового класса, и поэтому их не придется описывать снова и снова. Например, если программа оперирует классами «студент», «преподаватель» и «инженер», логично ввести дополнительный базовый класс «человек», переместив в него атрибуты, содержащие имя, адрес, другие личные данные, а также методы, манипулирующие этими данными.

Помимо двух уже рассмотренных качеств — инкапсуляции и наследования — у объектов есть еще третье основополагающее качество: *полиморфизм*. Это означает, что объекты могут вести себя по-разному в зависимости от ситуации. Одно из основных проявлений полиморфного поведения — перегрузка функций. Объект может содержать в себе несколько методов с одинаковыми именами, принимающими разные наборы параметров. В результате, передавая объекту данные, можно обращаться к одному и тому же имени метода, не заботясь о типе, в котором представлены данные. Правильно сконструированный объект автоматически выполнит наиболее подходящий метод из группы.

Инкапсуляция, наследование и полиморфизм являются тремя основополагающими принципами объектно-ориентированного программирования и в том или ином виде реализуются во всех объектно-ориентированных языках. В следующих разделах мы увидим, как конкретно эти принципы применены в C++.

## 10.1 Классы и объекты в C++

Хотя C++ и не первая попытка создать объектно-ориентированную версию языка C, среди попыток такого рода он оказался наиболее успешным. Очевидно, одна из причин успешности — то, каким образом объектная парадигма была встроена в синтаксис языка.

Ранние версии C++ были созданы в начале 1980-х Бьёрном Страуструпом для собственных нужд (в качестве универсального языка программирования, удобного для компьютерного моделирования). В создаваемый язык были заложены следующие основные принципы:

- поддержка различных стилей программирования, включая процедурный и объектно-ориентированный подходы;
- предоставление программисту полной свободы выбора — в т. ч. реализовать программные решения, которые могут казаться концептуально неверными;
- сохранение обратной совместимости с языком C, чтобы программист мог использовать только те дополнительные возможности C++, которые он сочтет нужным, или не использовать их вовсе;
- сохранение переносимости и высокой производительности, характерных для языка C.

Эти принципы заслужили высокую оценку программистов-практиков. В результате на текущий момент C++ является одним из наиболее распространенных языков программирования — как системного, так и прикладного.

### 10.1.1 Реализация ООП в C++. Классы и структуры

Синтаксис описания класса во многом копирует синтаксис описания структуры. В простейшем случае класс описывается так:

```
class имя_класса
{
    закрытые члены класса
    public:
        открытые члены класса
};
```

Как и при объявлении структуры, *имя\_класса* становится новым именем типа данных, которое можно использовать для объявления переменных (объектов класса) [5, 6]. Членами класса будут переменные и функции, объявленные внутри класса. Функции-члены класса называют *методами* этого класса, а переменные-члены класса называют *свойствами* класса.

В C++ понятия ООП используются следующим образом [5, 6]:

- «*класс*»: пользовательский тип данных, во многом аналогичный структуре;
- «*объект класса*» или «*переменная-экземпляр класса*»: переменная, в описании которой какой-то класс указан в качестве типа данных;
- «*свойство*» или «*переменная-член класса*»: переменная, объявленная внутри класса, (как поле внутри структуры); на практике чаще говорят не о

свойстве класса, а о *свойстве объекта*, так как для конкретных объектов переменные — члены класса обладают конкретными значениями и потому имеют конкретный смысл.

- «*метод*»: функция, объявленная внутри класса.

По умолчанию все функции и переменные, объявленные в классе, являются *закрытыми*, т. е. принадлежат *закрытой секции класса*. Это значит, что они доступны для обращения только изнутри членов этого класса и недоступны извне. Для объявления *открытых* членов класса используется ключевое слово `public` с двоеточием, обозначающее начало *открытой секции класса*. Все члены класса, объявленные после слова `public`, доступны для обращения как изнутри этого же класса, и для любой другой части программы, в которой доступен класс.

Открытых и закрытых секций в классе может быть несколько, и они могут произвольно чередоваться. При необходимости обозначить очередную закрытую секцию, ее начало обозначается ключевым словом `private`.

Более того, структуры в C++ были существенно доработаны (по сравнению с классическим вариантом структур языка С). В C++ структура может иметь помимо переменных-членов (т. е. полей структуры) также и функции-члены, а еще в структурах можно вводить открытые и закрытые секции. В сущности, структуры отличаются от классов двумя вещами:

- в структурах вместо ключевого слова `class` пишется ключевое слово `struct`;
- в структурах по умолчанию все члены являются открытыми (иначе перестали бы работать программы, написанные на С).

Рассмотрим в качестве примера объект, представляющий собой геометрический вектор в трехмерном пространстве. Для простоты ограничимся хранением в объекте трех координат и функции, вычисляющей модуль вектора. С учетом различий между структурами и классами, приведенные ниже варианты аналогичны.

<code>class spatial_vector</code>	<code>struct spatial_vector</code>
{	{
<b>public</b> :	<b>double</b> abs();
<b>double</b> abs();	<b>private</b> :
<b>private</b> :	<b>double</b> x, y, z;
<b>double</b> x, y, z;	}
}	

Добавив в структуру или в класс какой-нибудь метод, программист может потом вызвать этот метод для конкретного объекта. Обращение к содержимому объекта выполняется так же, как к содержимому структурной переменной: с использованием операции `«.»` (либо операции `«->»`, если нужно обратиться по указателю на объект).

```
main()
{
    spatial_vector a, b;
    double d;
```

```
    . . . .
} d = a.abs();
```

Очевидно, что функция `abs()`, объявленная в классе `spatial_vector`, возвращает абсолютное значение вектора. Однако для того, чтобы программа скомпилировалась, после объявления функцию `abs()` нужно еще определить (т. е. написать тело этой функции). Определение метода выполняется так же, как обычной функции, только в имени метода нужно указать, что он принадлежит конкретному классу. Для этого используется оператор *расширения области видимости* `<::>`. Имя класса записывается перед именем функции, отделенное двойным двоеточием. Например, в следующем примере мы объявим все тот же класс `spatial_vector` с двумя методами (установить значения координат вектора и посчитать его модуль) и опишем эти методы:

```
#include <iostream>
#include <math.h>
using namespace std;
class spatial_vector
{
    double x, y, z;
public:
    void set(double a, double b, double c);
    double abs();
};
void spatial_vector::set(double a, double b, double c)
{
    x=a; y=b; z=c;
}
double spatial_vector::abs()
{
    return sqrt (x*x + y*y + z*z);
}
main()
{
    spatial_vector a;
    a.set(1,2,3);
    cout << a.abs() << endl;
}
```

### 10.1.2 Создание и удаление объекта: конструкторы и деструкторы

Как читатель безусловно помнит, принципы ООП гласят, что свойства, описывающие состояние объекта, должны находиться в закрытой секции, чтобы доступ к ним осуществлялся через вызов методов объекта. Из-за этого в приведенном выше примере для класса `spatial_vector` мы использовали метод `set`, устанавливающий значения его переменных. Вообще, традиционным способом доступа к закрытым переменным класса будет добавление пар методов с именами, состоящими из имени переменной и префиксов `«get»` для чтения и `«set»` для записи (т. н. `«геттеры»` и `«сеттеры»`):

```
class spatial_vector
{
    double x, y, z;
public:
    double get_x();
```

```

void set_x(double x);
...
double spatial_vector::get_x() { return x; }
...
}

```

Этот способ является неудобным, т. к. при большом количестве переменных требует множества тривиальных описаний. Его следует применять только для тех свойств класса, внешний доступ к которым действительно необходим.

Однако есть особая ситуация, когда требуется за один раз присвоить значения переменным-членам класса (всем или большинству): это момент создания объекта, т. е. переменной-экземпляра класса.

C++ позволяет создать специальный метод, который будет автоматически вызываться для инициализации переменных-членов объекта при его создании. Такой метод называется *конструктором*. Программист, написавший класс, может по своему усмотрению включить в конструктор код для присваивания элементам начальных значений, динамического выделения памяти и т. д. Если программист не определил конструктор класса, компилятор самостоятельно генерирует конструктор по умолчанию (пустой и без входных параметров).

Конструктор может вызываться явно или неявно. Компилятор сам вызывает конструктор в том месте программы, где создается объект класса.

У описания конструкторов в C++ есть следующие особенности:

- имя конструктора в C++ совпадает с именем класса;
- конструктор не возвращает никакое значение, но при описании конструктора не используется и ключевое слово `void`.

Поскольку конструктор удобно использовать для динамического выделения памяти, должен быть также и способ освобождать эту память при уничтожении объекта (напомним, что локальные объекты удаляются тогда, когда они выходят из области видимости, а глобальные объекты удаляются при завершении программы). Действительно, функцией, обратной конструктору, является *деструктор*. Эта функция вызывается при удалении объекта.

В C++ деструкторы имеют имена, состоящие из имени класса с префиксом тильдой: «~имя\_класса». Как и конструктор, деструктор не возвращает никакое значение, но в отличие от конструктора он не может быть вызван явно. Причины такого ограничения очевидны: код, предположительно освобождающий динамическую память, будет обязательно вызван при выходе из области видимости. Его явный вызов привел бы к тому, что память уже освободилась заранее, а при уничтожении объекта программа попыталась бы сделать это снова и сгенерировала бы ошибку.

Конструктор не может быть описан в закрытой части класса. В общем случае то же ограничение накладывают и на деструктор. В следующем примере мы создаем объект, вызываем его метод, а затем разрушаем при завершении программы:

```

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

```

```

class spatial_vector
{
    double x, y, z;
public:
    spatial_vector();
    ~spatial_vector() { cout << "Работа деструктора\n"; }
    double abs() { return sqrt (x*x + y*y + z*z); }
};
spatial_vector::spatial_vector()
{
    //конструктор класса vector
    x=y=z=0;
    cout << "Работа конструктора\n";
}
main()
{
    spatial_vector a; //создается объект а с нулевыми значениями
    cout << a.abs() << endl;
}

```

Будучи выполнена, программа выводит следующие сообщения:

```

Работа конструктора
0
Работа деструктора

```

Обратите внимание на то, что тела функции `abs()` и деструктора были описаны непосредственно при объявлении, внутри описания класса. Такой подход обычно применяют для очень простых и коротких методов с тривиальным содержимым. В соответствии с традицией, унаследованной от языка С, описания классов в больших программах на С++ обычно выносятся в заголовочные файлы, в отличие от описания методов. Однако помещение описания простых методов внутрь описания класса имеет дополнительный практический смысл. Компилятор пытается сделать код таких методов *встроенным* (англ. *inline*). Это означает, что при обращении к методу вызов соответствующей функции будет заменен непосредственно на ее код. Благодаря такому трюку массовое обращение к свойствам класса через его методы (геттеры или сеттеры) не обязательно снижает производительность в сравнении с тем, если бы свойства находились в открытой секции.

### 10.1.3 Передача параметров в конструкторы

В рассмотренном примере мы использовали конструктор по умолчанию, т. е. без параметров. Однако, как любая другая функция, конструкторы могут иметь параметры. Значения параметров можно передать конструктору при создании объекта, в качестве аргументов:

```

#include <iostream>
#include <math.h>
using namespace std;
class spatial_vector
{
    double x, y, z;
public:
    spatial_vector(double x, double y, double z);
    ~spatial_vector() { cout << "Работа деструктора\n"; }
}

```

```

    double abs() { return sqrt (x*x + y*y + z*z); }
};

spatial_vector::spatial_vector(double x1, double y1, double z1)
{
    x = x1;
    y = y1;
    z = z1;
    cout << "Работа конструктора\n";
}

main()
{
    spatial_vector a(3,4,0);
}

```

В отличие от конструктора, деструктор не может иметь параметров. Это не удивительно: поскольку деструктор не вызывается явно, передавать ему параметры некому.

#### 10.1.4 Указатель `this`

Понятно, что свойства занимают место в памяти для каждого объекта (*собственно, значениями свойств объекты и отличаются друг от друга*). Однако нет никакой причины создавать для каждого нового объекта копии всех методов класса. Поэтому методы класса хранятся в единственном экземпляре.

Вместо бессмысленного расхода памяти на идентичные дубликаты методов, мы обращаемся к коду метода, передавая ему *контекст вызова* — указание на то, для какого объекта этот метод в данный момент вызван. Контекст передается с помощью дополнительного скрытого параметра, который функция-член класса получает в момент вызова: это указатель на переменную-экземпляр класса, для которого функция вызвана. Этот указатель имеет имя `this`. Если в теле метода используется переменная, которая не описана в нем и не является глобальной, автоматически считается, что она является членом класса и принадлежит переменной `this`.

При желании программист может явно использовать этот указатель — например, если имена аргументов метода совпадают с именами переменных-членов класса. Посмотрим как это выглядит на примере конструктора для некоторого класса `point`, содержащего координаты двумерной точки:

```

...
class point
{
    int x, y;
public:
    point(int x, int y)
    {
        this->x=x; this->y=y;
    }
...
}

```

### 10.1.5 Дружественные функции

Иногда использование методов для доступа к защищенным элементам класса из внешней среды может оказаться неудобным. На такой случай предусмотрен специальный обходной путь.

Чтобы класс мог предоставлять внешним функциям доступ к своей закрытой части, используется механизм объявления дружественных функций (**friend**) класса. Внутрь описания класса помещается прототип функции, перед которым ставится ключевое слово **friend**. В качестве примера, рассмотрим все тот же класс **point**. Объявим дружественную этому классу функцию **find\_point**, выполняющую поиск точки с заданными координатами в массиве объектов. Пусть функция принимает три аргумента: указатели на первый и последний элементы массива, среди которых нужно выполнять поиск, а также собственно аргумент поиска, т. е. точку с искомыми координатами.

```
class point
{
private:
    int x, y;
    ...
    friend void find_point(point* first, point *last, point arg);
};

void find_point(point* first, point *last, point arg)
{
    for (point *p=first; p<=last; p++)
        if ((p->x == arg.x) && (p->y == arg.y))
            cout << "Точка с координатами " << p->x << ", " << p->y << " найдена\n";
}
```

Важно понимать, что функция **find\_point()** не является членом класса **point**, хотя и имеет доступ к его закрытой части.

Одна и та же функция может быть другом двух и более классов. Инкапсуляция при этом не страдает, т. к. исключительные права для внешней функции предусмотрены в самом классе.

Функция-член одного класса может быть дружественной другому классу. Например:

```
class x
{
    ...
public:
    void f() {};
};

class y
{
    ...
    friend void x::f();
};
```

Если все функции одного класса дружественны другому классу, можно использовать сокращенную запись:

```
class y
{
    //...
    friend class x;
};
```

### 10.1.6 Статические свойства и методы класса

В C++ предусмотрен дополнительный способ совместного использования элемента данных несколькими объектами — статические члены класса.

Одно из типичных применений этого механизма — быстрый и эффективный обмен информацией между однотипными объектами за счет общей переменной. Другой причиной применения может оказаться желание уменьшить расход памяти в случае, если какое-то свойство класса может менять свое значение только одновременно для всех объектов, и таких объектов в программе много.

Чтобы объявить статический элемент класса, перед ним необходимо указать ключевое слово **static**. Для примера добавим в класс **point** статическое свойство **count** — счетчик, указывающий, сколько экземпляров данного класса существует в памяти в настоящий момент. Очевидно, что управлять содержимым счетчика будут конструктор и деструктор класса.

```
#include<iostream>
using namespace std;
class point
{
    int x, y;
    //...
    static int count;
public:
    point() {cout << "Создается точка с номером " << ++count << endl;}
    ~point() {cout << "Разрушается точка с номером " << count-- << endl;}
};
int point::count;
main()
{
    point a,b,c;
}
```

Помимо соответствующего описания в классе, статическая переменная-член класса должна быть дополнительно объявлена в программе в качестве глобальной переменной с указанием ее принадлежности классу (см. в примере строку перед описанием функции **main**). В сущности, статические свойства и являются глобальными переменными, с ограниченной областью видимости. В результате выполнения программы сначала создаст, а потом разрушит три объекта класса **point**, выведя об этом соответствующие сообщения:

```
Создается точка с номером 1
Создается точка с номером 2
Создается точка с номером 3
Разрушается точка с номером 3
Разрушается точка с номером 2
Разрушается точка с номером 1
```

Метод класса также можно объявить статическим. Такой метод будет вести себя одинаково для всех объектов, т. е. не будет различать, для какого именно объекта он вызван. По этой причине статическим методам не передается скрытый указатель **this**. Однако взамен статические методы получают преимущество: их можно вызывать, не создавая объект класса. Например, статическими могут быть содержащиеся в классе сервисные функции, если они не использует никаких данных объекта, т. е. сам *контекст вызова* им по сути не нужен.

На этот раз примером будет класс `alarm`, предназначенный для работы с будильником и среди прочего содержащий в себе служебный метод `current_time()`, выводящий на экран текущее время. Поскольку этот метод использует служебную функцию операционной системы и не нуждается в свойствах объекта, мы можем сделать его статическим. Остальные методы класса для простоты опустим.

```
#include<iostream>
#include<time.h>
using namespace std;
class alarm
{
    time_t alarm_t;
public:
    static void current_time();
    // ...
};
void alarm::current_time()
{
    time_t t = time(NULL); //получаем текущее время в нотации Unix, в виде числа секунд,
    //прощедших с 1 января 1970 г.
    struct tm tm = *localtime(&t); //переводим в местное текущее время
    cout << tm.tm_hour << ':' << tm.tm_min << ':' << tm.tm_sec << endl;
}
main()
{
    alarm::current_time();
}
```

Как видно из примера, для доступа к статическому методу класса без указания объекта достаточно всего лишь написать перед именем метода имя класса и поставить оператор разрешения области видимости.

### 10.1.7 Перегрузка операторов

Как уже упоминалось, перегрузка, т. е. возможность создавать функции (например, методы класса) с одинаковыми именами и разными наборами параметров, вызываемые в разных ситуациях для решения однотипных задач — это одно из ключевых проявлений полиморфизма в C++. Однако кроме перегрузки функций C++ позволяет проделывать то же самое с большинством стандартных операторов.

На самом деле, можно считать, что перегрузка операторов для стандартных типов данных в неявном виде присутствовала еще в языке С. Например, оператор деления может выполнять разные действия в зависимости от того, какой тип имеют его аргументы: для целочисленных аргументов будет выполнено деление нацело, а для вещественных — деление чисел с плавающей точкой. С точки зрения процессора деление чисел с плавающей точкой кардинально отличается от деления нацело: задействована другая машинная команда, операнды должны быть загружены в совсем другие регистры (ячейки памяти процессора), после чего выполняется совсем другая микропрограмма. На более высоком уровне абстракции операции целочисленного и вещественного деления могут казаться оди-

наковыми; однако использование для них одного и того же оператора допускают далеко не все языки.

В C++ это явление довели до логического завершения, и теперь многие встроенные операторы можно перегрузить для работы с новыми типами данных. Чтобы перегрузить оператор, программист объявляет новую функцию, имя которой состоит из ключевого слова **operator** и знака операции. Например, перегрузим оператор **+** для сложения двух объектов класса **spatial\_vector**. Объявление функции будет выглядеть следующим образом:

```
spatial_vector operator+ ( spatial_vector a, spatial_vector b )
{
    .....
}
```

Нам понадобится предусмотреть в классе **spatial\_vector** геттеры и сеттеры для всех трех координат, чтобы внешняя функция могла выполнить покоординатное сложение двух векторов (либо мы могли бы объявить функцию дружественной классу). Также мы предусмотрим в классе конструктор, инициализирующий координаты заданными значениями, и метод **info**, выводящий координаты вектора на экран.

```
#include <iostream>
#include <math.h>
using namespace std;
class spatial_vector
{
    double x, y, z;
public:
    spatial_vector(double x,double y,double z){this->x=x;this->y=y;this->z=z;}
    double get_x() { return x; }
    double get_y() { return y; }
    double get_z() { return z; }
    void set_x(double x) { this->x=x; }
    void set_y(double y) { this->y=y; }
    void set_z(double z) { this->z=z; }
    void info() { cout << "Координаты вектора: <<x<<" ,<<y<<" ,<<z<<endl; }
};
spatial_vector operator+ ( spatial_vector a, spatial_vector b )
{
    spatial_vector c(0,0,0);
    c.set_x(a.get_x() + b.get_x());
    c.set_y(a.get_y() + b.get_y());
    c.set_z(a.get_z() + b.get_z());
    return c;
}
main()
{
    spatial_vector a(1,2,3), b(10,20,30), c(0,0,0);
    c=a+b;
    c.info();
}
```

- оператор должен уже существовать в языке (нельзя добавить в программу новые, не существовавшие ранее операторы);
- нельзя изменить количество операндов, которое принимает перегружаемый оператор;
- нельзя переопределять действия встроенных в C++ операторов при работе со встроенными типами данных: например, нельзя перегрузить оператор

«+» для работы с целыми числами типа `int` (а если вдруг это зачем-то понадобится, можно создать класс-обертку, например `integer`, и перегружать для него все что угодно);

- нельзя перегружать операторы «.», «.\*», «?:», «::»;
- по вполне очевидной причине нельзя перегружать знак директивы препроцессора «#».

### 10.1.8 Перегрузка членов класса

Члены класса можно перегружать так же, как любые другие функции. Особенно часто перегрузку используют для объявления нескольких конструкторов. Главный смысл перегрузки конструкторов состоит в том, чтобы предоставить программисту наиболее удобный для каждой конкретной ситуации способ инициализации объекта. Например, мы можем объявить два конструктора в классе `spatial_vector`: один конструктор по умолчанию, создающий вектор с нулевыми значениями, а другой — принимающий конкретные параметры:

```
class spatial_vector
{
    double x, y, z;
public:
    spatial_vector (double x, double y, double z);
    spatial_vector ();
...
};
```

Однако заметим, что если код двух конструкторов практически идентичен и различается лишь использованием переданных значений в одном конструкторе и констант — в другом, то запись можно упростить. В такой ситуации можно оставить только конструктор с параметрами и задать для этих параметров (для всех или для нескольких, идущих последними в списке) значения по умолчанию:

```
spatial_vector (double x=0, double y=0, double z=0);
```

Параметры, имеющие значение по умолчанию, можно не указывать при вызове.

Операторы тоже можно перегружать как члены класса, но с некоторыми интересными особенностями. Если мы вызываем в программе метод класса — его вызов будет указан после имени конкретного объекта. Как читатель безусловно помнит, при этом методу передается скрытый указатель на объект. Если перегруженный оператор объявлен как член класса, то компилятор, встретив его вызов, должен определить, для какого объекта вызвана перегружающая оператор функция, и тоже передать ей скрытый указатель на объект. Таким объектом всегда является левый операнд. По этой причине в объявлении перегруженного оператора внутри класса нет необходимости упоминать собственный объект — ведь он передается скрытым указателем `this`. Поэтому описание бинарного оператора, перегруженного как член класса, имеет всего один операнд (правый), а описание унарного оператора не имеет ни одного операнда.

Следует отметить, что нельзя объявить оператор как статический метод (поскольку статическим методам указатель `this` при вызове не передается) или использовать с оператором аргументы по умолчанию.

Для одного и того же оператора можно объявить несколько перегруженных операторов-функций. Но, как и в случае перегрузки обычных функций, компилятор должен различать их по типу и количеству аргументов. Когда компилятор сталкивается с перегруженным оператором для класса `X`, он ищет подходящую функцию-оператор для класса `X`, используя обычные для перегруженных функций правила сопоставления аргументов. Если поиск завершился неудачей, компилятор не пытается самостоятельно применить к аргументам перегруженных операторов преобразования типов.

Механизм дружественных функций часто используется при перегрузке операторов для работы с объектами, когда по какой-то причине перегруженный оператор невозможно или нецелесообразно объявлять членом класса. Для сравнения изменим фрагмент примера из п. 10.1.7, переопределив оператор «+» как функцию-член класса (слева) и как дружественную функцию (справа):

<pre>class spatial_vector {     ...     spatial_vector operator+(spatial_vector b); };  spatial_vector spatial_vector::operator+(spatial_vector b) {     spatial_vector c;     c.x = x + b.x;     c.y = y + b.y;     c.z = z + b.z;     return c; }</pre>	<pre>class spatial_vector {     ...     friend spatial_vector operator+(spatial_vector a, spatial_vector b); };  spatial_vector operator+(spatial_vector a, spatial_vector b) {     spatial_vector c;     c.x = a.x + b.x;     c.y = a.y + b.y;     c.z = a.z + b.z;     return c; }</pre>
---	--

Классический пример перегрузки оператора как дружественной функции — средства стандартного ввода-вывода в C++. Как известно, операции `<<` и `>>` выполняют ввод и вывод, если левым аргументом указан один из стандартных объектов ввода-вывода. Предопределенные объекты `cin` (клавиатура) и `cout` (дисплей) — экземпляры классов `istream` и `ostream`. Этим способом можно вывести любой базовый тип данных C++. Однако, на самом деле при выводе вызывается функция, перегружающая оператор. В частности, для объекта `cout` будет вызвана функция, имеющая приблизительно следующий вид:

```
ostream operator<<(ostream, int)
```

В результате, выражение

```
cout << "Значение переменной i равно " << i << '\n';
```

благодаря такой подстановке будет заменено на

```
operator<<(operator<<(operator<<(cout, "Значение переменной i равно "), i), '\n');
```

Понятно, что библиотека `iostream` содержит функции только для встроенных типов. Если требуется перегрузить операторы стандартного ввода-вывода для нового класса, чтобы программист мог вводить с консоли его информационное содержимое или выполнять его вывод на экран, необходимо перегрузить оператор для нового типа.

Использовать перегрузку оператора как члена класса невозможно, т. к. левым аргументом должен быть не объект нового класса, а уже существующие объекты `cin` и `cout`. Таким образом, перегружать оператор приходится как внешнюю функцию. Однако поскольку эту функцию создает автор нового класса, он вполне может объявить ее в структуре класса как дружественную, упростив ей доступ к закрытой части класса.

В общем виде операция вывода имеет следующую форму

```
ostream& operator << ( ostream& stream , имя_класса& obj )
{
    stream << ... //вывод элементов объекта obj в поток stream
    return stream;
}
```

Аналогичным образом может быть определена функция ввода:

```
istream& operator >> ( istream& stream , имя_класса& obj )
{
    stream >> ... //ввод элементов объекта obj из потока stream
    return stream;
}
```

Знак «`&`» в списке формальных параметров означает, что компилятор обеспечивает скрытую передачу параметра не по значению, а по ссылке (передача объектов по ссылке детально рассмотрена далее, в п. 10.2.2). Первый аргумент функций ввода и вывода определен как ссылка на поток, второй аргумент — ссылка на объект, выводящий или получающий информацию, а возвращаемое значение — тот же самый объект потока, который был передан в качестве первого аргумента.

Приведем пример с перегрузкой операторов стандартного ввода и вывода для уже знакомого нам класса `spatial_vector`.

```
#include <iostream>
using namespace std;
class spatial_vector
{
    double x, y, z;
public:
    spatial_vector() { x=y=z=0; }
    friend ostream& operator<< ( ostream& stream , spatial_vector& b );
    friend istream& operator>> ( istream& stream , spatial_vector& b );
};
ostream& operator<< ( ostream& stream , spatial_vector& b )
{
    stream << "x = " << b.x << " ; y = " << b.y << " ; z = " << b.z << endl;
    return stream;
}
istream& operator>> ( istream& stream , spatial_vector& b )
{
    stream >> b.x >> b.y >> b.z;
    return stream;
}
main()
```

```
{
    spatial_vector a;
    cin >> a;
    cout << "Был введен вектор: " << a << endl;
}
```

### 10.1.9 Перегрузка постфиксных операторов

Большинство операций, поддерживаемых C++, являются префиксными, т. е. оператор применяется до вычисления выражения. Исключение составляют операторы инкремента и декремента `++` и `--`, которые могут быть как префиксными, так и постфиксными. При перегрузке постфиксных операций возникают определенные неудобства: например, программист должен иметь возможность как-то *показать компилятору*, что перегружает именно постфиксный оператор.

Объявление члена класса с именем `operator++` без аргументов перегружает префиксный оператор инкремента. Чтобы перегрузить функцию-член класса как постфиксный оператор, его нужно объявить с одним аргументом типа `int`. Этот аргумент не несет никакой полезной нагрузки и нужен только, чтобы можно было различить префиксные и постфиксные операторы. При выполнении этот аргумент будет иметь нулевое значение. Следующий пример показывает разницу в описаниях, и дополнительно выводит в консоль информацию о том, префиксный или постфиксный оператор был использован. В примере использован класс `integer`, являющийся оберткой над переменной типа `int`, т. е. просто хранящий целое число:

```
#include <iostream>
using namespace std;
class integer
{
    int value;
public:
    integer() { value = 0; }
    integer& operator++(); //префиксный оператор
    integer& operator++(int); //постфиксный оператор
};
integer& integer::operator++()
{
    value+=1;
    cout << "Использован префиксный оператор\n";
    return *this;
}
integer& integer::operator++(int)
{
    value+=1;
    cout << "Использован постфиксный оператор\n";
    return *this;
}
main()
{
    integer i;
    i++; //используется постфиксный оператор
    ++i; //используется префиксный оператор
}
```

Заметим, что в приведенном примере оператор постфиксного инкремента реализован не совсем корректно и по действию не отличается от префиксной формы.

Если требуется реализовать в программе его полноценный функционал, т. е. изменение аргумента после возврата его исходного значения, для этого в теле оператора создается и потом возвращается временный объект — копия исходного аргумента. Подробнее особенности передачи временной копии объекта рассматриваются в следующих подразделах, а пример с полнофункциональной формой постфиксных операторов с дополнительными пояснениями можно найти в конце п. 10.2.3.

## 10.2 Создание и удаление объектов

### 10.2.1 Присваивание объектов, передача в функцию и возвращение объекта

Можно заметить, что в ряде случаев (например, при перегрузке операторов) мы использовали передачу объекта в функцию и возвращение не по значению, а с использованием ссылки. Для этого существует достаточно веская причина: передача объектов по значению, равно как и их присваивание, может приводить к нежелательным последствиям.

Один объект можно присвоить другому, если оба объекта имеют одинаковый тип (если объекты имеют разные типы, то компилятор выдаст сообщение об ошибке). По умолчанию, когда объект A присваивается объекту B, то осуществляется побитовое копирование всех элементов-данных A в соответствующие элементы-данные B. Именно это копирование и является потенциальным источником проблем. Особенно внимательным нужно быть при присваивании объектов, имеющих свойства-указатели.

Рассмотрим в качестве примера класс `matrix`, хранящий в себе прямоугольную матрицу из элементов типа `double`. Размерность матрицы будет передаваться конструктору класса, после чего будет выполняться динамическое выделение памяти под нужное количество элементов. В классе будут также предусмотрены методы `get_val()` чтобы получить элемент матрицы с индексами `(i,j)` и `set_val()` чтобы установить в заданный элемент новое значение.

Однако присвоив просто так одну переменную типа `matrix` другой, мы не сможем избежать побочных эффектов.

```
#include <iostream>
using namespace std;
class matrix
{
    double *m; //элементы матрицы
    size_t width, height; //число строк и столбцов в матрице
public:
    matrix(size_t w, size_t h);
    double get_val(size_t i, size_t j);
    void set_val(size_t i, size_t j, double val);
    ~matrix();
};
matrix::matrix(size_t w, size_t h)
{
    m = new double [w*h];
    width = w;
```

```

    height = h;
}
matrix::~matrix()
{
    delete [] m;
}
double matrix::get_val(size_t i, size_t j)
{
    return m[ i * width + j ]; //получить значение элемента матрицы в позиции [i,j]
}
void matrix::set_val(size_t i, size_t j, double val)
{
    //устанавливаем значение элемента матрицы в позиции [i,j]
    //если координаты не превышают размер матрицы
    if ((i < width) && (j < height)) m[ i * width + j ] = val;
}
main()
{
    matrix a(2, 2); //объявляем матрицу размерности 2 x 2
    a.set_val(0, 0, 100); //устанавливаем a[0,0] = 100
    matrix b=a; //присваиваем матрицу
    b.set_val(0, 0, 200); //устанавливаем b[0,0] = 200
    cout << "a[0,0] = " << a.get_val(0,0) << " ; " << "b[0,0] = " << a.get_val(0,0) << endl;
}

```

При запуске программа выдает сообщение «`a[0,0] = 200; b[0,0] = 200`» вместо ожидаемого «`a[0,0]=100`», после чего и вовсе аварийно завершается с сообщением о попытке дважды освободить память. На самом деле это происходит по вполне очевидной причине. При побитовом копировании скопировался адрес указателя `m`, а содержимое блока памяти, динамически выделенного по этому адресу. В результате оба объекта получают указатель на одну и ту же последовательность вещественных чисел.

Аналогично присваиванию, объекты можно передавать в функции в качестве аргументов, в частности так, как передаются данные других типов. Однако следует помнить, что в C++ по-умолчанию параметры передаются по значению. Это означает, что внутри функции (а точнее, в стеке) создается копия объекта-аргумента, и эта копия, а не сам объект, будет далее использоваться функцией. Благодаря этому функции могут произвольно изменять переданные значения, не влияя на оригинал.

Итак, при передаче объекта в функцию создается новый объект, а когда работает функция завершается, копия переданного объекта будет разрушена. Как всегда при разрушении объектов, при этом будет вызван деструктор копии. И здесь может наблюдаться очередной побочный эффект: если переданный в качестве параметра объект содержит в себе указатель на динамически выделенную область памяти, деструктор копии ее освободит. Но так как копия создавалась побитовым копированием, деструктор копии высвободит область памяти, на которую указывал объект-оригинал. Исходный объект будет по-прежнему «видеть» свои данные по указанному адресу, однако для системы эта память будет считаться свободной. Рано или поздно она будет выделена какому-то другому объекту, и данные окажутся затерты.

Кроме возможности преждевременного разрушения объекта-оригинала, к аварийной ситуации приведет вызов его деструктора (в конце работы программы или при выходе из соответствующей области видимости), который попытается освободить уже свободную память. Та же проблема возникает при использовании объекта в качестве возвращаемого значения.

Во всех трех случаях (при присваивании объекта, при использовании его как параметра и при передаче в качестве возвращаемого значения) если деструктор высвобождает динамически выделенную память, то разрушение временного объекта приведет к преждевременному разрушению данных оригинала.

Частично проблема может быть решена перегрузкой оператора присваивания для данного класса. Кроме того, для объектов, которым противопоказано побитовое копирование, рекомендуется создавать особую разновидность конструктора — т. н. *конструктор копирования* (в некоторых источниках также можно встретить название «конструктор копии»). Конструктор копирования выполняет именно то действие, которое заложено в его названии: позволяет программисту лично проконтролировать процесс создания копии.

Любой конструктор копирования имеет следующую форму:

```
имя_класса (const имя_класса & obj)
{
    ... //тело конструктора
}
```

Читатель должен помнить, что в таком описании `&obj` — это ссылка на объект, известная еще как скрытый указатель.

Оператор присваивания, перегруженный как член класса, связан со своим классом настолько же тесно, как конструктор и деструктор. Эту связь подчеркивает то, что оператор копирования разрешено перегружать только как функцио-член класса, и запрещено — как дружественную функцию. Приведем в качестве иллюстрации две почти одинаковые записи:

```
point p1, p2; //объявляем два объекта класса point
point p3 = p2; //используем конструктор копирования
p1 = p2; //используем оператор присваивания
```

Во второй строке примера переменная `p3` и объявляется и определяется, а в третьей строке переменной `p1` всего лишь присваивается значение. Иными словами, конструктор копирования вызывается для конкретной переменной за время ее жизни только один раз, а присваивать значения ей можно многократно. В логике работы конструктора копирования и оператора присваивания настолько много общего:, что часто рекомендуют описывать одну операцию в терминах другой. Фактически операция присваивания неявно используется в конструкторе копирования. Однако конструктор копирования может добавлять дополнительные действия по инициализации переменных в довесок к тем действиям, которые должен выполнять оператор присваивания.

Если оператор присваивания для класса не был определен, то в случае необходимости (если для объектов этого класса в тексте программы выполняется присваивание) компилятор автоматически генерирует *оператор присваивания по умолчанию*, выполняющий то самое побитовое копирование объекта.

### 10.2.2 Подробнее об указателях и ссылках на объекты

Передача объектов по указателю имеет ряд преимуществ. В стек копируется только ячейка памяти, содержащая адрес объекта, и автоматически исчезает проблема корректного создания копии. Конечно в результате функция взаимодействует с оригиналом объекта, что требует осторожного изменения его данных (а с копией можно делать все что угодно). Еще одно дополнительное преимущество передачи по адресу в сравнении с передачей по значению — экономия ресурсов на копирование. Если объект занимает достаточно большой объем памяти, его передача в функцию и возвращение из нее, даже будучи выполнены корректно, приведут к неоправданным расходам ресурсов.

Недостаток передачи по указателю — худшая читаемость программы, когда приходится часто взаимодействовать с адресами объектов. Многочисленные операции адресации и разадресации (взятия адреса объекта и взятия содержимого по адресу) могут ухудшать визуальное восприятие текста программы, особенно в сочетании со скобками.

По этой причине в C++ был введен специальный тип данных — *ссылка* или *скрытый указатель*. На понятийном уровне ссылку можно воспринимать как другое имя ( псевдоним ) переменной. Фактически же это указатель на переменную, который выглядит так, как будто к переменной обращаются по значению: программист объявляет такую ссылку, присваивает ей какую-либо переменную и далее пользуется ссылкой как еще одной переменной. Компилятор же сам автоматически подставляет ко всем обращениям к ссылке операции адресации и разадресации.

Удобнее всего использовать ссылки для передачи параметров и возвращаемых значений.

Ссылка объявляется так же как указатель, только с использованием знака «&» вместо «звездочки». Сравним, например, как выглядит код при передаче аргумента по указателю и по ссылке, на примере функции `zero()`, устанавливающей в ноль координаты переданного ей объекта класса `point`:

```
//использование указателей
void zero ( point *p )
{
    p->set ( 0,0 );
//мы использовали ">->"
}
main ()
{
    point a(3,4);
    zero(&a);
}
```

```
//использование ссылки
void zero ( point &p )
{
    p.set ( 0,0 );
//мы использовали "."
}
main ()
{
    point a(3,4);
    zero(a);
}
```

В приведенном примере при применении параметра-ссылки, компилятор передает адрес переменной, но везде кроме объявления функции код выглядит так, как будто переменная передана по значению. Аналогично ссылки могут использоваться в качестве возвращаемого значения функции. Однако нельзя забывать,

что функция, в которую передан параметр по ссылке, будет манипулировать *не копией*, а самим оригинальным объектом.

Часто ссылки применяют в сочетании с указателем `this`. Рассмотрим в качестве примера переопределение оператора присваивания: для класса `point`:

```
point& point::operator= (point& p)
{
    x = p.x;
    y = p.y;
    return *this;
}
```

В объявлении функции мы указали ссылочный тип в качестве как аргумента, так и возвращаемого значения. Оператор присваивания должен возвращать результат операции, чтобы стало возможным каскадное присваивание наподобие `a=b=c=0`. В качестве возвращаемого значения мы указываем разадресованный указатель `this`, однако возвращен в качестве результата будет тот объект, который вызывал операцию `<=`, а не его копия

Приведем модифицированный вариант класса `matrix`, имеющий как конструктор копирования, так и оператор присваивания, и выдающий на экран правильный результат.

```
#include <iostream>
using namespace std;
class matrix
{
    double *m; //элементы матрицы
    size_t width, height; //число строк и столбцов в матрице
public:
    matrix(size_t w, size_t h);
    matrix(const matrix& m1); //конструктор копирования
    matrix& operator=(matrix & m1); //оператор присваивания
    double get_val(size_t i, size_t j);
    void set_val(size_t i, size_t j, double val);
    ~matrix();
};
matrix::matrix(size_t w, size_t h)
{
    m = new double [w*h];
    width = w;
    height = h;
}
matrix::matrix(const matrix& m1)
{
    //устанавливаем размер матрицы и выделяем под нее память:
    width = m1.width;
    height = m1.height;
    int size=width*height;
    m = new double [size];
    //копируем элементы матрицы:
    for (int i=0; i < size; i++)
        m[i]=m1.m[i];
}
matrix& matrix::operator=(matrix& m1)
{
    int size=m1.width*m1.height;
    if (size > width*height)
        //зацищаемся от переполнения буфера
        size=width*height;
    m = new double [size];
    //копируем элементы матрицы:
```

```

    for (int i=0; i < size; i++)
        m[ i]=m1.m[ i ];
    return *this;
}
matrix::~matrix()
{
    delete [] m;
}
double matrix::get_val(size_t i, size_t j)
{
    //получить значение элемента матрицы в позиции [i,j]
    return m[ i*width+j ];
}
void matrix::set_val(size_t i, size_t j, double val)
{
    //устанавливаем значение элемента матрицы в позиции [i,j]...
    //...если координаты не превышают размер матрицы
    if ((i<width)&&(j<height)) m[ i*width+j ]=val;
}
main()
{
    matrix a(2, 2); //объявляем матрицу размерности 2 x 2
    a.set_val(0,0,100); //устанавливаем a[0,0] = 100
    matrix b=a; //присваиваем матрицу
    b.set_val(0,0,200); //устанавливаем b[0,0] = 200
    cout << "a[0,0] = " << a.get_val(0,0) << " ; " << "b[0,0] = " << a.get_val
        (0,0) << endl;
}

```

Внимательный читатель может заметить в коде примера необычную особенность. И конструктор копирования, и оператор присваивания получают доступ к закрытой части переданного объекта `m1`. На самом деле это вполне естественно. Вспомним: переменные, объявленные в закрытой секции класса, доступны только для методов этого же класса (а не «этого же объекта»). Иными словами, объекты одного класса могут получать доступ к закрытым членам друг друга, хотя используется это не так часто.

### 10.2.3 Пример: класс `spatial_vector` в сборе

Прежде чем закончить разговор о перегрузке операторов и передаче объектов, приведем еще один пример, наглядно иллюстрирующий, что иногда передача объекта по значению может нести практическую пользу. Рассмотрим финальный вид класса `spatial_vector` с перегруженными операторами ввода-вывода, инкремента и декремента обеих форм, а также операторами сложения и вычитания.

```

#include <iostream>
#include <math.h>
using namespace std;
class spatial_vector
{
    double x, y, z;
public:
    spatial_vector(double x=0, double y=0, double z=0);
    double abs() { return sqrt (x*x + y*y + z*z); }
    double get_x() { return x; }
    double get_y() { return y; }
    double get_z() { return z; }
    void set_x(double x) { this->x=x; }

```

```

void set_y(double y) { this->y=y; }
void set_z(double z) { this->z=z; }
void info();
spatial_vector& operator++(); //префиксная форма
spatial_vector& operator--();
spatial_vector operator++(int); //постфиксная форма
spatial_vector operator--(int);
friend spatial_vector operator+(spatial_vector a, const spatial_vector& b);
friend spatial_vector operator-(spatial_vector a, const spatial_vector& b);
friend ostream& operator<<(ostream& stream, const spatial_vector& b);
friend istream& operator>>(istream& stream, spatial_vector& b);
};

spatial_vector::spatial_vector(double x1, double y1, double z1)
{
    x = x1;
    y = y1;
    z = z1;
}
void spatial_vector::info()
{
    cout << "Координаты вектора: x=" << x << "; y=" << y << "; z=" << z << endl;
    cout << "Модуль вектора равен " << abs() << endl;
}
spatial_vector& spatial_vector::operator++()
{
    x++; y++; z++;
    return *this;
}
spatial_vector& spatial_vector::operator--()
{
    x--; y--; z--;
    return *this;
}
spatial_vector spatial_vector::operator++(int)
{
    spatial_vector temp=*this;
    ++(*this);
    return temp;
}
spatial_vector spatial_vector::operator--(int)
{
    spatial_vector temp=*this;
    --(*this);
    return temp;
}
spatial_vector operator+ (spatial_vector a, const spatial_vector& b)
{
    //передаем первый аргумент по значению,
    //поэтому можем изменять его, не влияя на исходный объект:
    a.x += b.x;
    a.y += b.y;
    a.z += b.z;
    //возвращаем измененную копию первого аргумента:
    return a;
}
spatial_vector operator-(spatial_vector a, const spatial_vector& b)
{
    a.x -= b.x;
    a.y -= b.y;
    a.z -= b.z;
    return a;
}
ostream& operator<<(ostream& stream, const spatial_vector& b)
{
    stream << "x=" << b.x << "; y=" << b.y << "; z=" << b.z << endl;
}

```

```

    return stream;
}
istream& operator>>(istream& stream, spatial_vector& b)
{
    stream >> b.x >> b.y >> b.z;
    return stream;
}
main()
{
    spatial_vector a, b(1,2,3);
    cout << "\n1. Заполнение вектора через стандартный ввод\n";
    cout << "Введите координаты вектора: ";
    cin >> a;
    a.info();
    cout << "\n2. Вычитание векторов\n";
    spatial_vector c = a-b;
    cout << "Координаты вектора c=a-b(1,2,3): " << c;
    cout << "\n3. Изменение координат вектора с помощью геттеров и сеттеров\n";
    c.set_x(c.get_x() + 1);
    cout << "После инкремента координаты x, координаты вектора c: " << c;
    cout << "\n4. Инкремент:\nвывод c++: " << c++;
    cout << "Вывод ++c: " << ++c;
}

```

Функция `main()` просит пользователя ввести с клавиатуры три координаты вектора, а затем выполняет несколько тестов, демонстрирующих работу методов класса. Например, при вводе значений «1 2 3» выводится следующий результат:

```

1. Заполнение вектора через стандартный ввод
Введите координаты вектора: 1 2 3
Координаты вектора: x=1; y=2; z=3
Модуль вектора равен 3.74166

2. Вычитание векторов
Координаты вектора c=a-b(1,2,3): x=0; y=0; z=0

3. Изменение координат вектора с помощью геттеров и сеттеров
После инкремента координаты x, координаты вектора c: x=1; y=0; z=0

4. Инкремент:
вывод c++: x=1; y=0; z=0
Вывод ++c: x=3; y=2; z=2

```

Как можно заметить, в программе не перегружен оператор присваивания, а в классе `spatial_vector` не задан конструктор копирования. В данном случае класс не работает с динамической памятью и не нуждается в какой-то особой предварительной инициализации и деинициализации, поэтому выполняемое по умолчанию побитовое копирование объектов оказывается полностью приемлемым. Более того, передача параметра по значению активно используется в перегруженных операторах сложения и вычитания, а также в постфиксной форме инкремента и декремента. В первых двух случаях первый параметр (левый операнд) передается по значению, чтобы можно было изменить его и вернуть, не затронув исходный объект. В перегруженных постфиксных операторах используется возврат исходной побитовой копии объекта, снятой до того, как оригинальный объект был изменен.

Комбинирование в выражениях перегруженных операторов, некоторые из которых используют передачу по ссылке, а некоторые — по значению, требует тща-

тельности в оформлении списка параметров. Если возвращенный по значению результат работы одного оператора может быть принят другим по ссылке, во избежание конфликтов соответствующий аргумент следует явно объявить константным (см. например, второй параметр оператора потокового вывода). На самом деле использование модификатора `const` в подобных случаях более чем логично, поскольку оператор не изменяет принятый аргумент, а передача по ссылке используется исключительно для уменьшения накладных расходов на копирование объекта.

### 10.3 Наследование

Наследование классов позволяет строить иерархию, наверху которой находятся более общие классы, а внизу — более специализированные. Попробуем привести наглядный пример иерархии наследования. Предположим, мы создаем объектно-ориентированную систему работы с графикой, и предусмотрели класс `point`, описывающий отдельную двумерную точку на экране. В этом классе хранятся координаты точки, ее цвет, а также методы для управления этими данными. При необходимости можно легко создать на базе класса `point` производный класс, хранящий трехмерную вершину (например, `vertex`): добавить в нем третью координату, соответствующие конструкторы, модифицировать некоторые методы. Однако не следует путать отношение наследования с отношением включения. Например, будет нелогичным строить на базе класса `point` или класса `vertex` класс `region`, описывающий объекты с произвольным количеством вершин: скорее, это должен быть класс-контейнер, содержащий в себе массив объектов `point` или `vertex`.

Таким образом, есть смысл создавать на базе существующего класса производный, если мы хотим получить частный случай с модифицированной функциональностью.

В C++ новый класс строится на базе уже существующего с помощью конструкции следующего вида:

```
class parent {.....};  
class child : [модификатор наследования] parent {.....};
```

При определении класса-потомка, за его именем следует разделитель двоеточие «`:`», затем необязательный *модификатор наследования* и имя родительского класса. Модификатор наследования определяет видимость наследуемых переменных и методов для класса-потомка и его возможных потомков. Таким способом определяется, какие права доступа к переменным и методам класса-родителя будут «делегированы» классу-потомку.

При реализации наследования область видимости принадлежащих классу данных и методов можно определять выбором одного из следующих *модификаторов доступа*:

- `private` (закрытый);
- `public` (общедоступный);
- `protected` (защищенный).

Модификаторы наследования:	Модификаторы доступа:
<ul style="list-style-type: none"> <li>• public</li> <li>• protected</li> <li>• private</li> </ul>	<ul style="list-style-type: none"> <li>• public</li> <li>• protected</li> <li>• private</li> </ul>
<pre>class point {...}; class vertex: public point {...};</pre>	<pre>class point { public:     int color;     ... };</pre>

Таблица 10.1: Разница между модификаторами наследования и доступа

Модификатор доступа:	Модификатор наследования:		
	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	нет доступа	нет доступа	нет доступа

Таблица 10.2: Сочетание модификаторов наследования и доступа

Эти модификаторы могут произвольно чередоваться внутри описания класса и уже использовались нами для обозначения открытых и закрытых секций класса.

Модификатор **private** описывает закрытые члены класса, доступ к которым имеют только методы-члены этого класса. Модификатор **public** предназначен для описания общедоступных элементов, доступ к которым возможен из любого места в программе, где доступен объект данного класса. Модификатор **protected** используется тогда, когда необходимо, чтобы некоторые члены базового класса оставались закрытыми, но были бы доступны из класса-потомка.

Иными словами, одни и те же ключевые слова могут использоваться и в качестве модификаторов наследования, и в качестве модификаторов доступа:

То, как изменяется доступ к элементам базового класса из методов производного класса в зависимости от значения модификаторов наследования, показано в таблице 10.2.

Из таблицы видно, в производном классе доступ к элементам базового класса может быть сделан более ограниченным, но никогда нельзя сделать его менее ограниченным.

### 10.3.1 Конструкторы и деструкторы при наследовании

Базовый класс, производный класс или оба могут иметь конструкторы и деструкторы. Если и у базового и у производного классов есть конструкторы и деструкторы, то они срабатывают по-очереди: конструкторы выполняются в порядке наследования, а деструкторы — в обратном порядке.

```
#include<iostream>
using namespace std;
class parent
{
public:
    parent() {cout << "Работа конструктора базового класса\n"; }
    ~parent() {cout << "Работа деструктора базового класса\n"; }
};
class child: public parent
{
public:
    child() {cout<<"Работа конструктора производного класса\n"; }
    ~child() {cout<<"Работа деструктора производного класса\n"; }
};
main()
{
    child c1;
}
```

Выполнение этой предельно простой программы иллюстрирует порядок срабатывания конструкторов и деструкторов. Результат ее работы следующим образом отображается на экране:

```
Работа конструктора базового класса
Работа конструктора производного класса
Работа деструктора производного класса
Работа деструктора базового класса
```

При реализации наследования может возникнуть ситуация, когда конструктор базового или производного класса требует параметры. Если параметры нужно передать только конструктору производного класса, они передаются обычным способом. Если требуется передать какие-либо аргументы родительскому конструктору, для этого используется расширенная запись конструктора производного класса:

```
конструктор_производного_класса (список формальных параметров)
: конструктор_базового_класса (список фактических параметров)
{
    ... //тело конструктора производного класса
}
```

Списки параметров родительского и дочернего конструкторов могут совпадать, а могут и различаться. Например, конструктор производного класса часто принимает некоторые аргументы только для того, чтобы передать их конструктору базового класса. Приведем пример с уже обсуждавшимися классами `point` и `vertex`.

```
#include<iostream>
using namespace std;
class point
{
protected:
    int x, y, color;
```

```

public:
    point ( int p1, int p2, int c );
};

point::point( int p1, int p2, int c )
{
    x=p1;
    y=p2;
    color=c;
}
class vertex: public point
{
    int z;
public:
    vertex ( int p1, int p2, int p3, int c );
};
vertex::vertex( int p1, int p2, int p3, int c ): point(p1, p2, c)
{
    z=p3;
}
main()
{
    vertex c1(2,3,4,0);
}

```

Допускается также использовать эту краткую запись передачи параметров родительскому конструктору, чтобы компактно проинициализировать переменные дочернего класса. Например, мы могли бы написать в предыдущем примере:

```

vertex::vertex( int p1, int p2, int p3 )
    : point(p1, p2), z(p3)
{
    // z=p3;
}

```

С наследованием конструкторов класса связана еще одна специфическая особенность: если в базовом классе есть перегруженный оператор присваивания, он не наследуется производными классами. Если оператор присваивания был перегружен в родительском классе, а в производном — нет, то присваивание объектов производного класса не будет вызывать ошибку, однако выполняться при этом будет не перегруженный оператор, а присваивание по умолчанию, т. е. побитовое копирование свойств объекта.

### 10.3.2 Раннее и позднее связывание

Обрабатывая вызов метода какого-либо класса, компилятор сначала ищет метод с указанным именем внутри данного класса. Если метод с таким именем не определен внутри класса, то компилятор обращается к базовому классу и ищет его там. Если найдет, то подставит в точки вызова адрес метода из родительского класса. Если не найдет, то поднимается все выше по иерархии наследования.

Методы, которые вызываются так, являются статическими — в том смысле, что компилятор разбирает ссылки на них во время компиляции. Этот подход экономит ресурсы в момент выполнения программы, однако иногда приводит к нежелательным результатам. Рассмотрим для примера иерархию из двух классов: класса `vector`, представляющего собой двумерный вектор, и производный от него класс `spatial_vector`, уже знакомый нам по прежним примерам. Нам

будут нужны два метода у каждого из классов: метод `info()`, выводящий текстовое сообщение и сообщающий, чему равен модуль вектора, и метод `abs()`, собственно вычисляющий значение модуля. Наследование одного класса от другого в данном случае представляется вполне логичным: в производном классе достаточно будет добавить еще одну переменную, модифицировать конструктор и функцию вычисления модуля.

```
#include <iostream>
#include <math.h>
using namespace std;
class vector
{
protected:
    double x, y;
public:
    vector(double x, double y) { this->x=x; this->y=y; }
    double abs() { return sqrt (x*x + y*y); }
    void info() { cout << "Модуль вектора равен " << abs() << endl; }
};
class spatial_vector : public vector
{
protected:
    double z;
public:
    spatial_vector(double x, double y, double z);
    double abs() { return sqrt (x*x + y*y + z*z); }
};
spatial_vector::spatial_vector(double x, double y, double z) :vector(x, y)
{
    this->z=z;
}
main()
{
    cout << "Создаем вектор на плоскости с координатами 1,2\n";
    vector a(1,2);
    a.info();
    cout << "Создаем пространственный вектор с координатами 1,2,3\n";
    spatial_vector b(1,2,3);
    b.info();
}
```

В действительности же данный код генерирует весьма странный результат:

```
Создаем вектор на плоскости с координатами 1,2
Модуль вектора равен 2.23607
Создаем пространственный вектор с координатами 1,2,3
Модуль вектора равен 2.23607
```

Мы корректно переопределили метод `abs()` в производном классе (можно легко в этом убедиться, вызвав его непосредственно), однако для производного класса функция `info()` выдала явно неверное значение, не посчитав в модуле третью координату. Проблема в том, что родительский метод `info()` «не знает», что функция `abs()` переопределена в классе-потомке.

Для того чтобы это стало возможным нужен специальный механизм, и в языке C++ это — *позднее связывание*, реализующее механизм *виртуальных методов*. Виртуальные методы реализуют полиморфизм.

*Виртуальный метод* — это метод, который, будучи описан в потомках, замещает собой соответствующий метод *везде*, даже в методах, описанных для предка, если он вызывается для потомка.

Адрес виртуального метода известен только в момент выполнения программы. Когда происходит вызов виртуального метода, его адрес берется из таблицы виртуальных методов своего класса. Таким образом, вызывается то, что нужно.

Виртуальные методы описываются с помощью ключевого слова `virtual` в базовом классе. Это означает, что в производном классе этот метод может быть замещен методом, более подходящим для этого производного класса. Объявленный виртуальным в базовом классе, метод остается виртуальным для всех производных классов. Если в производном классе виртуальный метод не будет переопределен, то при вызове будет найден метод с таким именем вверх по иерархии классов (т. е. в базовом классе).

```
#include <iostream>
#include <math.h>
using namespace std;
class vector
{
protected:
    double x, y;
public:
    vector(double x, double y) { this->x=x; this->y=y; }
    virtual double abs() { return sqrt (x*x + y*y); }
    void info() { cout << "Модуль вектора равен " << abs() << endl; }
};
class spatial_vector : public vector
{
protected:
    double z;
public:
    spatial_vector(double x, double y, double z);
    double abs() { return sqrt (x*x + y*y + z*z); }
};
spatial_vector::spatial_vector(double x, double y, double z):vector(x, y)
{
    this->z=z;
}
main()
{
    cout << "Создаем вектор на плоскости с координатами 1,2\n";
    vector a(1,2);
    a.info();
    cout << "Создаем пространственный вектор с координатами 1,2,3\n";
    spatial_vector b(1,2,3);
    b.info();
}
```

Будучи выполнен, пример наконец выдает ожидаемый ответ:

```
Создаем вектор на плоскости с координатами 1,2
Модуль вектора равен 2.23607
Создаем пространственный вектор с координатами 1,2,3
Модуль вектора равен 3.74166
```

### 10.3.3 Множественное наследование

В списке базовых классов можно указывать несколько классов-родителей, через запятую, каждого со своим модификатором наследования:

```
class A {...};
class B {...};
class C: public A, public B {...};
```

При этом класс С унаследует как содержимое класса А, так и класса В. При вызове конструктора будут сначала вызваны конструкторы базовых классов (в порядке следования). Деструкторы, как всегда, имеют противоположный порядок вызова.

При множественном наследовании автоматически включается позднее связывание.

#### 10.3.4 Указатели на базовые классы

Другая сфера приложения виртуальных функций связана с использованием указателей на объекты. Указатель, объявленный в качестве указателя на базовый класс, также может использоваться как указатель на любой класс, производный от этого базового:

```
point * p = new vertex();
```

По указателю на объект базового класса можно вполне корректно вызвать те методы класса-потомка, которые уже существовали в описании базового класса.

Вызвать по такому указателю метод, присутствующий лишь в производном классе напрямую нельзя, но можно косвенно, с использованием приведения типов:

```
class parent
{
public:
    void parent_method() {}
};

class child : public parent
{
public:
    void child_method() {}
};

main()
{
    parent *p = new child();
    p->parent_method();
    ((child*)p)->child_method();
}
```

Типичное использование указателя на базовый класс, которому присвоен адрес объекта производного класса — хранение либо передача нескольких разнотипных объектов, имеющих общий класс-предок. Например, во многих библиотеках виджетов (графических элементов управления) инструментальная панель, которая может содержать в себе кнопки, надписи, выпадающие списки и т. д., является универсальным контейнером, хранящим указатели на объект базового класса (например, класса `widget`), от которого унаследованы конкретные элементы управления (классы `button`, `text`, `list` и т. д.). Благодаря возможности использовать указатель на базовый класс, панель реализует один единственный набор методов для добавления и удаления разнотипных элементов, а также для обращения к ним.

Однако при использовании указателей на базовый класс требуется соблюдать осторожность в отношении разрушения объектов. Если объект-потомок выйдет из области видимости и будет разрушен по указателю на базовый класс, то без

дополнительных мер вызовется деструктор базового класса. Если деструкторы базового и производного классов имеют важные различия в своем поведении (например, когда деструктор-потомок должен освободить дополнительные блоки памяти) — такое поведение является недопустимым. В этом случае деструктор родительского класса необходимо объявлять виртуальным — так же, как это делается с любым другим методом.

### 10.3.5 Абстрактные классы

Иногда, когда функция объявляется в базовом классе, она не выполняет никаких значимых действий, поскольку часто базовый класс не определяет заданный тип, а нужен чтобы построить иерархию. Например, метод `paint()`, объявленный в классе `widget` и выполняющий отрисовку виджета на экране, должен переопределяться в классах-потомках, с тем, чтобы выводить на экран изображение кнопки в классе `button` или текстовую надпись в классе `text`. Изображение же абстрактного виджета тоже абстрактно, и метод в базовом классе не несет практической нагрузки.

Методы, которые нужны только для того, чтобы их обязательно переопределить в производных классах, называются *чисто виртуальными* методами.

Чисто виртуальные методы не определяются в базовом классе. У них нет тела, а есть только декларации об их существовании.

Чисто виртуальная функция выглядит в описании класса следующим образом:

```
virtual тип имя_функции (список параметров) = 0;
```

Как можно заметить, функцию делает чисто виртуальной приравнивание ее описания к нулю.

Класс, содержащий хотя бы один чисто виртуальный метод, называется абстрактным классом. Поскольку у чисто виртуального метода нет тела, то создать объект абстрактного класса невозможно.

### 10.3.6 Пример иерархии классов — библиотека потокового ввода-вывода

В заключение рассмотрим, как используется наследование для решения реальных задач программирования.

Технологии наследования классов, входящие в стандарт C++, предсказуемо были использованы и при разработке библиотек, входящих в стандарт этого языка. Включенная в стандарт реализация сама по себе является прекрасным примером, т. к. в данном случае не приходится сомневаться в грамотном и целесообразном проектировании как иерархии классов, так и их внутренней реализации. По этой причине рассмотрим в качестве примера сложной структуры классов, созданных для решения конкретных практических задач, объектно-ориентированную библиотеку потокового ввода-вывода.

В предыдущих разделах мы касались использования операторов `<<` и `>>` вместе с объектами `cin` и `cout`, а также отметили, что эти объекты на самом деле являются экземплярами классов `istream` и `ostream`. Однако прежде чем рассматривать внутреннее устройство этих и других классов, вовлеченных в реализацию ввода-вывода, необходимо разобраться в самой предметной области. Поэтому рассмотрим подробнее, какие стадии включает в себя принятая в C++ кросс-платформенная реализация ввода-вывода.

При разработке средств потокового ввода-вывода в C++ была использована следующая двухуровневая модель (см. рис. 10.3), предназначенная для передачи символьных данных между программой и каким-либо внешним устройством.



Рис. 10.3: Процедурный подход к программированию

При этом представление данных в программе и на внешнем устройстве может отличаться: по необходимости данные могут отображаться в форме, удобной для восприятия человеком, либо преобразовываться в какой-либо формат обмена данными. Как видно из рисунка, собственно обработка текстовых данных выполняется на двух уровнях: *форматирования* и *транспортном*. Под форматированием понимается преобразование внутренних данных программы в человекочитаемую последовательность символов. Например, значение вещественной переменной на этом этапе должно быть преобразовано в последовательность цифр, а управляющие символы в строковых данных должны быть заменены соответствующими им символьными последовательностями (например, код табуляции «\t» превращается в заданное число пробелов). Под кодированием понимается трансляция из одной кодировки символов в другую. Например, транспортный уровень может выполнять преобразование символов в кодировку Unicode для их использования за пределами программы (данная кодировка является стандартом де-факто, но неудобна для непосредственного использования в программе, т. к. символы разных языков имеют в ней различное количество байт).

В рамках двухуровневой модели ввода-вывода, принятой в C++, уровень форматирования делает возможным выполнение следующих процедур:

- преобразование вещественных значений в последовательность цифр с заданной точностью и формой представления;
- преобразование целочисленных значений в последовательность цифр в шестнадцатиричной, восьмиричной либо десятичной системе счисления;
- исключение лишних пробелов из входных данных;

- задание ширины поля (т. е. максимального количества знаков) для выводимых данных;
- адаптация представления чисел к конкретной локали (т. е. учет национальной специфики их отображения).

Транспортный уровень отвечает непосредственно за получение и выдачу символов. На этом уровне инкапсулируется специфика конкретных внешних устройств. К числу таковых помимо возможности преобразования в многобайтные кодировки относится также блочный вывод в файлы с использованием системных вызовов операционной системы, под которую компилируется программа.

Для уменьшения числа обращений к внешнему устройству используется потоковый буфер. При выводе последовательность символов после форматирования попадает в потоковый буфер, а реальная передача данных внешнему устройству выполняется, когда буфер оказывается заполнен, или когда принудительно вызвана операция опустошения буфера. При вводе данных транспортный уровень считывает последовательность символов из внешнего устройства в буфер, после чего уровень форматирования извлекает данные из буфера. Когда буфер оказывается пуст, задачей транспортного уровня является его повторное наполнение данными.

Реализованный в C++ форматированный потоковый ввод-вывод можно разделить на две группы: *файловый ввод-вывод* и *ввод-вывод в памяти*. Файловый ввод-вывод предполагает передачу данных между программой и внешним устройством. При этом внешнее устройство только представлено файлом; помимо обычного файла на диске оно может в действительности быть каналом обмена данными или любым реальным устройством, файловая абстракция которого реализована в операционной системе.

Ввод-вывод в памяти в действительности не существует никакого внешнего устройства. Благодаря этому отпадает необходимость в уровне кодирования и передачи, а уровень форматирования просто формирует строку символов.

Расширяемость библиотеки потокового ввода-вывода позволяет программисту добавлять свои элементы на любом из ее уровней. Например, операторы ввода-вывода могут быть перегружены для новых типов данных, программист может создавать собственные элементы, управляющие форматированием (т. н. манипуляторы). Можно создавать собственные локали для специфического представления чисел и т. д.

Теперь мы можем рассмотреть, как выглядит иерархия классов потокового ввода-вывода с точки зрения программиста.

Мы будем рассматривать упрощенное представление для случая, когда символы представлены в программе в однобайтной кодировке с использованием типа `char`. В реальности библиотека `iostream` реализует более универсальное представление данных на основе *шаблонов*, позволяющее не указывать заранее при описании классов тип данных, используемый для хранения символа. Благодаря этому подходу тот же самый код может применяться, например, для многобайтных кодировок, представленных специальным типом `wchar_t`. Также мы на-

данном этапе опустим специальные средства обработки ошибок и других исключительных ситуаций, примененные в данной библиотеке. Подробнее о шаблонах и обработке исключительных ситуаций можно будет узнать в следующих разделах; там же будут пояснены опущенные на данном этапе элементы, и в т. ч. то, как на самом деле объявлены типы библиотеки `iostream`.

Пока достаточно знать, что иерархия классов `iostream` выглядит для программиста, использующего обычные символы типа `char`, следующим образом (см. рис. 10.4).

Из приведенной иерархии можно заметить, что структура классов уровня форматирования заметно более разветвленная, хотя основные отличия между файловыми потоками и потоками в памяти кроются на транспортном уровне. Каждующийся дисбаланс легко объясним, если вспомнить, что программист, пользующийся библиотекой `iostream`, непосредственно взаимодействует в основном с уровнем форматирования. Использование универсальных классов, которые после существенной предварительной настройки выполняли бы ввод-вывод с любыми объектами транспортного уровня, менее удобно, чем специализированные классы для каждого типа ввода-вывода, не требующие или почти не требующие настройки для выполнения требуемых операций.

Будучи базовым для всех потоковых классов уровня форматирования, класс `ios` содержит информацию, присущую любым потокам: управляющую информацию для разбора и форматирования, возможности для расширения иерархии собственными потоками пользователя, а также локали. Здесь же объявляются некоторые типы, используемые остальными классами: флаги форматирования, биты состояния, режим открытия и т. д. Здесь же содержится указатель на потоковый буфер (который в свою очередь включает собственно символьный буфер и служебную информацию, отражающую состояние буфера и обеспечивающую целостность информации).

Как читатель успел заметить из собственной практики программирования на C++, активнее всего потоки используются для стандартного ввода-вывода (т. е. ввода с клавиатуры и вывода на дисплей). Для обработки стандартного ввода предусмотрен класс `istream`, а для обработки стандартного вывода — `ostream`; оба класса наследуются от `ios`, приобретая благодаря этому всю специфику, связанную с форматированием, и указатель на потоковый буфер. Для взаимодействия с потоковым буфером в классе `istream` объявлен перегруженный оператор потокового ввода `>>`, а в классе `ostream` — перегруженный оператор потокового вывода `<<`. Для возможности неформатированного ввода и вывода в этих классах объявлен также ряд методов — таких как `read()` и `write()`. Наконец, для случаев, когда необходим двунаправленный ввод-вывод (по аналогии с тем, как файл может открываться одновременно для чтения и записи) с помощью множественного наследования от этих двух классов порожден класс `iostream`, автоматически приобретающий свойства как входных, так и выходных потоков и используемый как базовый для классов, в которых двунаправленный ввод-вывод действительно востребован.

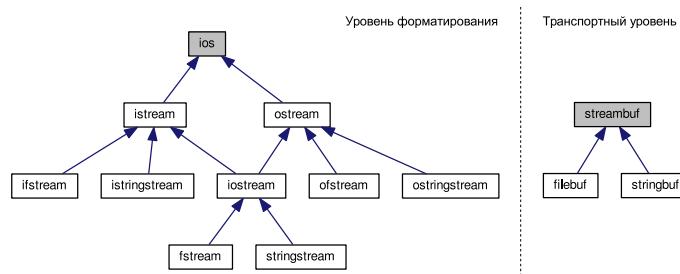


Рис. 10.4: Иерархия классов потокового ввода-вывода

Строковые потоки, осуществляющие ввод-вывод в памяти, представлены классами `istringstream` и `ostringstream`, порожденными соответственно от `istream` и `ostream`, а также универсальным классом двунаправленного ввода-вывода `stringstream`, порожденным от `iostream`. Эти классы включают функции (геттеры и сеттеры) для использования строки в качестве буфера.

Файловый ввод-вывод осуществляют классы `ifstream` и `ofstream`, порожденные соответственно от `istream` и `ostream`, а также универсальный класс `fstream`, порожденный от `iostream`. Эти классы содержат методы для открытия и закрытия файлов, аналогичные функциям `fopen()` и `fclose()` языка C.

Очевидно, что значительная часть различий между стандартным вводом-выводом, файловыми и строковыми потоками скрыта на транспортном уровне.

Базовый класс `streambuf` олицетворяет собой универсальный потоковый буфер. Будучи абстрактным классом, он не содержит в себе специфики конкретных оконечных устройств; однако в нем объявлены две чисто виртуальных функции: `overflow()` и `underflow()`, которые должны быть перегружены в производных классах, чтобы выполнять действительную передачу символов между символьным буфером и конкретными оконечными устройствами.

Класс потокового буфера поддерживает две символьные последовательности: область получения (`get`), представляющую последовательность символов, получаемых из оконечного устройства, и область выдачи (`put`), т. е. выходную последовательность для записи на устройство. Также в классе предусмотрены функции, извлекающие очередной символ из буфера (`sgetc()` и т. д.) и помещающие очередной символ в буфер (`sputc()` и т. д.), которые обычно используются уровнем форматирования. Дополнительно потоковый буфер содержит также объект локали.

Производный от `streambuf` класс `filebuf` используется для работы с файлами и содержит для этого ряд функций, таких как `open()` и `close()`. Он также наследует объект локали от базового класса для перекодирования между внешней и внутренней кодировками (например, как уже упоминалось, между кодировкой

Unicode и внутренним представлением мультиязычных символов значениями типа `wchar_t`).

Класс `stringbuf` также является производным от `streambuf`. Поскольку он предназначен для работы со строками, внутренний буфер одновременно является и оконечным устройством. По мере необходимости внутренний буфер может динамически изменять свой размер, чтобы принять все записанные в него символы. Класс позволяет получить копию внутреннего буфера, а также скопировать в него строку.

Взаимодействие между уровнем форматирования и транспортным уровнем осуществляется следующим образом. Класс `ios`, как мы уже упоминали, содержит в себе указатель на потоковый буфер. В производных от него классах (таких как `fstream` или `stringstream`) содержатся указатели на объекты соответствующих классов транспортного уровня (`filebuf` или `stringbuf`). Классы транспортного уровня можно также использовать и непосредственно, для неформатированного ввода-вывода — точно так же, как их использует уровень форматирования.

Представленная на рисунке иерархия могла бы быть описана следующим образом:

```
class ios {...};
class ostream : public ios {...};
class istream : public ios {...};
class ofstream : public ostream {...};
class ifstream : public istream {...};
class ostringstream : public ostream {...};
class istringstream : public istream {...};
class iostream : public ostream, public istream {...};
class fstream : public iostream {...};
class stringstream : public iostream {...};
```

## 10.4 Обработка исключений

### 10.4.1 Общие понятия

Классический подход к обработке ошибок в программе, разработанной в рамках процедурного подхода, предполагает анализ значений, возвращаемых функциями. Например, многие библиотечные функции в случае возникновения ошибки или какой-либо непредвиденной ситуации возвращают нулевое значение, интерпретируемое как «ложно», а в случае успешной работы — ненулевое, т. е. «истина». При необходимости передачи более детальной информации об ошибке, код ошибки может сохраняться в некую глобальную переменную (библиотечные функции, унаследованные из языка С, для этой цели используют глобальную переменную `errno`).

У этого подхода есть два существенных недостатка. Во-первых, он не поощряет программиста проверять, была ли корректной работа вызванной функции. Фактически, наоборот: программа, в которой половина вызовов функций обернуты условным оператором, анализирующем возвращаемое значение, выглядит менее стройной и хуже читается из-за насыщенности однотипными конструкциями, не имеющими отношения к основному алгоритму работы. По этой причине

программисту инстинктивно хочется выполнять проверку корректности работы как можно реже, что повышает вероятность неотслеживаемых сбоев в работе создаваемого программного продукта.

Второй недостаток связан с крайней лаконичностью информации об ошибке, передаваемой стандартными средствами. Например, по коду ошибки, возникшей где-то на одном из внутренних уровней вложенности, программа определяет проблему с открытием файла, и выдает пользователю сообщение «файл не найден». Однако если бы механизм обработки ошибок позволял программисту для каждого типа ошибочной ситуации стандартными средствами передавать произвольную дополнительную информацию, было бы гораздо проще организовать информативные сообщения в духе «не найден файл такой-то», или «не удалось создать файл такого-то размера», или «выделение в памяти такого-то количества байт завершилось неудачей».

Для решения этих проблем в C++ был включен новый механизм — *обработка исключительных ситуаций*.

Исключительная ситуация (англ. «exception») или *исключение* — это что-то особенное или ненормальное, случившееся в работе программы. Чаще всего исключение — это какая-то возникшая ошибка, но не обязательно: например, это может быть нестандартное стечение обстоятельств или просто ситуация, требующая нетиповой обработки.

Если в программе (или в библиотечной функции, которую использует программа) возникает какая-то неразрешимая ситуация, то *генерируется исключение*. Это означает, что вместо нормального продолжения программы управление передается на другую ветку алгоритма, специально предназначенному для обработки такой исключительной ситуации. Эта другая ветвь программы — *обрабочик исключения* — может просто прервать программу, а может что-то изменить, чтобы программа могла продолжить выполнение. Причем, даже если исключение возникает в библиотечной функции, обработчик исключения все равно находится в программе, использующей эту библиотеку — в той части кода, которая непосредственно вызвала конкретную нестандартную ситуацию и потому лучше может справиться с ее обработкой.

Чтобы исключение, сгенерированное в одном блоке кода, могло найти нужный обработчик, находящийся в другом блоке, при генерации исключения *выбрасывается индикатор исключения*. Индикатором служит объект или переменная некоторого конкретного типа. При возникновении исключения будет выбран тот обработчик, в описании которого указан тот же тип ожидаемого индикатора. Обработчики различают исключения по типам данных индикатора, и поэтому в наиболее распространенном случае в качестве индикатора указывают объект некоторого класса, специально предусмотренного для этой цели.

Программист, желающий использовать исключения, должен поместить вызов кода, в котором исключение может возникнуть, в специальный блок `try{}`. Следом за этим блоком должен следовать блок `catch(){}`, внутрь которого помещают код, обрабатывающий исключительную ситуацию. Например, если ис-

исключение может возникнуть в некой функции `f()`, для его обработки нужно написать следующую конструкцию:

```
f()
{
    //генерируем исключение, если возникла соответствующая ситуация
    if (....) throw индикатор;
    ....
}
....
try
{
    //вызываем код, который может сгенерировать исключение:
    f();
}
catch(индикатор)
{
    //обрабатываем исключение
    ....
}
```

Обработчик исключения всегда следует сразу за блоком `try{}`. В круглых скобках после `catch` указывается индикатор исключения, которое этот блок обрабатывает. Чтобы сгенерировать исключение, нужно использовать специальную конструкцию `throw`, после которой указывается индикатор.

В следующем примере, показывающем, как можно применить обработку исключений для организации диалогового режима работы, мы объявим два пустых класса для использования в качестве индикаторов: класс `unknown_exception`, означающий получение неизвестного ответа от пользователя, и класс `abort_exception`, сигнализирующей о необходимости немедленного выхода из программы. Сама программа задает пользователю вопрос о выполнении последовательно 100 неких абстрактных пронумерованных команд. Диалог реализуется функцией `confirm()`, спрашивающей у пользователя подтверждение на выполнение команды с заданным номером и анализирующей ответ ('`y`' — подтверждение, '`n`' — отказ, '`a`' — немедленный выход).

```
#include <iostream>
#include <math.h>
using namespace std;
class unknown_exception{};
class abort_exception{};
bool confirm(int i)
{
    char c;
    cout << "Подтвердите команду " << i << " (y/n/a - да/нет/выход): ";
    cin >> c;
    cin.ignore(); //очищаем буфер если введены лишние символы
    switch (c) {
        case 'y': return true;
        case 'n': return false;
        case 'a': throw abort_exception();
        default: throw unknown_exception();
    }
}
main()
{
    cout << "Демонстрация диалога подтверждения при выполнении" << " 100 команд\n";
    for (int i=1;i<=100;i++) {
        try{
            if (confirm(i)) cout << "КОМАНДА " << i << " ВЫПОЛНЕНА\n";
        }
```

```

    else cout << "КОМАНДА " << i << " ОТМЕНЕНА\n";
}
catch(unknown_exception) {
    cout << "Неизвестный ответ пользователя\n";
    i--; // возвращаемся к предыдущей команде
}
catch(abort_exception) {
    cout << "Выполняется немедленный выход из программы\n";
    return 0;
}
cout << "Продолжение демонстрации диалога\n";
}
}

```

Как видно из примера, обработчики исключений должны следовать друг за другом каждый в своем блоке `catch()`. После того, как отработает один из обработчиков, управление передается на код, следующий за последним блоком `catch()` в данной цепочке.

Обратите внимание, что в блоке `catch()` мы указали в качестве параметра только тип данных — класс-индикатор. Это допустимо с учетом того, что обработчик исключения не собирается извлекать никаких данных из переданного индикатора, да и сами классы-индикаторы, созданные в программе, являются пустыми и используются только для того, чтобы различать исключительные ситуации.

#### 10.4.2 Передача информации в обработчик

Как уже упоминалось, чтобы передать в обработчик дополнительную информацию, нужно принимать в блоке `catch()` не тип данных, а переменную. Перед выбрасыванием исключения можно ее проинициализировать конкретными данными, а потом прочитать эти данные в обработчике.

Приведем в качестве иллюстрации еще один пример, в котором реализован класс `array`, предоставляющий пользователю массив с возможностью добавления и удаления элементов в стиле работы со стеком. Для этого класса будет перегружен оператор индекса `[]`, возвращающий значение элемента с заданным номером, а также две функции для изменения размера массива: `push()`, позволяющая добавить новый элемент в конец массива, и `pop()`, забирающая из массива последний добавленный элемент. При создании объекта для массива будет резервироваться память, для чего конструктору будет передаваться параметр `capacity` — емкость массива, т. е. максимально допустимое число элементов.

```

#include <iostream>
#include <math.h>
using namespace std;
class general_error
{
public:
    char *message;
    general_error(char* message) { this->message=message; }
};
class out_of_range
{
public:
    size_t i;

```

```

    out_of_range(size_t i) { this->i=i; }
};

class overflow {};
class underflow {};
class array
{
    size_t size; //реальный размер массива
    size_t capacity; //максимально-допустимый размер
    double *data;
public:
    array(size_t capacity);
    ~array();
    double operator[](size_t i); //получить значение i-го элемента
    void push(double v); //добавить элемент
    double pop(); //убрать последний добавленный элемент
};
array::array(size_t capacity)
{
    if (capacity==0)
        throw general_error("массив нулевой вместимости");
    this->capacity=capacity;
    size=0;
    data = new double[capacity];
}
array::~array()
{
    delete [] data;
}
double array::operator[](size_t i)
{
    if (i < size) return data[i];
    else throw out_of_range(i);
}
void array::push(double v)
{
    if (size < capacity) data[size++]=v;
    else throw overflow();
}
double array::pop()
{
    if (size > 0) return data[--size];
    else throw underflow();
}
main()
{
    char c;
    size_t i;
    double v;
    cout << "Введите емкость массива: ";
    cin >> v;
    array a(v);
    for (;;)
    {
        cout << "Введите "+\ для добавления элемента, " " \ "-\" для извлечения, \"i\" для
        просмотра " "i-го элемента, \"a\" для выхода: ";
        cin >> c;
        try
        {
            switch (c)
            {
                case '+':
                    cout << "Введите добавляемое число: ";
                    cin >> v;
                    a.push(v);
                    break;

```

```

    case '‐':
        v=a.pop();
        cout << "Извлечено число " << v << endl;
        break;
    case 'i':
        cout << "Введите индекс: ";
        cin >> i;
        v=a[i];
        cout << "Искомый элемент равен " << v << endl;
        break;
    case 'a':
        return 0;
        break;
    }
}
catch(const out_of_range& e)
{
    cout << "Попытка доступа к элементу с недопустимым индексом "<< e.i << endl;
}
catch(overflow)
{
    cout << "Операция не выполнена, так как массив переполнен\n";
}
catch(underflow)
{
    cout << "Операция не выполнена, так как массив пуст\n";
}
}
}

```

В этом примере использованы четыре класса-индикатора исключений: `general_error` для ошибок неопределенного типа (класс содержит строку `message`, описывающую суть возникшей проблемы), `out_of_range` для выхода индекса за границу массива (свойство `i` предусмотрено для значения индекса), а также классы `overflow` для ошибки переполнения емкости массива и `underflow` для попыток удалить элемент из пустого массива. Обработчик `out_of_range` принимает объект класса-индикатора и сообщает пользователю, какое именно значение индекса оказалось недопустимым. Диалог с пользователем ведется в бесконечном цикле, на каждой итерации которого предлагается выбрать одно из четырех действий: добавление элемента, удаление элемента, просмотр элемента с заданным индексом или выход из программы.

### 10.4.3 Иерархия исключений

Классы-индикаторы исключения могут принадлежать к общей иерархии наследования, т. е. быть в отношениях «родитель-потомок». При этом обработчики индикаторов-родительских классов могут перехватывать исключения с индикаторами-потомками (можно считать такое поведение проявлением полиморфизма). Поэтому родительские обработчики нужно обязательно указывать *после* дочерних в цепочке блоков `catch` — иначе дочерний обработчик никогда не получит управление. В самом конце цепочки можно указать `catch...`, у которого в круглых скобках вместо индикатора три точки. Такой блок будет перехватывать абсолютно любые исключения:

```
class general_error{};
```

```

class out_of_range: public general_error {};
.....
try { ..... }
catch (out_of_range)
{ cout << "Выход индекса за границу массива\n"; }
catch (general_error)
{ cout << "Общий сбой в работе программы\n"; }
catch (...) {cout << "Неизвестная ошибка\n"; }

```

В приведенном схематичном примере мы объявили два различных класса-индикатора, один базовый, для исключений общего типа, и один производный от него, для исключительной ситуации типа «недопустимый индекс при обращении к в массиву». Если бы порядок следования обработчиков был другим, обработчик индикатора `general_error` никогда не смог бы активироваться.

Если обработчик перехватил исключение, но обнаружил, что не сможет справиться с его обработкой, он может вызвать `throw` без аргументов: это передаст исключение дальше по цепочке уровней вложенности, на случай если на более высоком уровне есть обработчик, способный так или иначе решить возникшую ситуацию. Проиллюстрируем повторное возбуждение исключения, изменив пример из п. 10.4.2. Цикл обработки событий мы поместим в отдельную функцию `main_loop()`, принимающую в качестве аргумента ссылку на массив. Соответственно, создание объекта массива и передачу его в цикл обработки событий поместим в еще один блок `try`, с обработчиком, принимающим исключение типа `general_error`. В первую очередь это позволит корректно обрабатывать ошибку нулевой емкости массива. Для иллюстрации передачи повторно генерированного исключения из внутреннего обработчика внешнему специально добавим инструкцию `throw` без аргументов в обработчик события `out_of_range` (таким образом, выход индекса за границу массива станет фатальной ошибкой, приводящей к остановке программы). Чтобы исключение могло быть успешно перехвачено на внешнем уровне вложенности, сделаем класс `general_error` родительским для остальных классов-индикаторов исключений.

```

#include <iostream>
using namespace std;
class general_error
{
public:
    char *message;
    general_error(char* message) { this->message=message; }
};
class out_of_range : public general_error
{
public:
    size_t i;
    out_of_range(size_t i);
};
out_of_range::out_of_range(size_t i)
:general_error("Выход индекса за границу массива")
{ this->i=i; }
class overflow : public general_error
{
public:
    overflow();
};
overflow::overflow():general_error("Переполнение массива") {}
class underflow : public general_error

```

```

{
public:
    underflow();
};

underflow::underflow() : general_error("Массив пуст") {}

class array
{
    size_t size;           //реальный размер массива
    size_t capacity;      //максимально-допустимый размер
    double *data;

public:
    array(size_t capacity) throw(general_error);
    ~array();
    double operator[](size_t i) throw(out_of_range); //получить значение i-го
                                                       //элемента
    void push(double v) throw(overflow); //добавить элемент
    double pop() throw(underflow); //убрать последний добавленный элемент
};

array::array(size_t capacity) throw(general_error)
{
    if(capacity==0) throw
        general_error("массив нулевой вместимости");
    this->capacity=capacity;
    size=0;
    data = new double[capacity];
}

array::~array()
{
    delete[] data;
}

double array::operator[](size_t i) throw(out_of_range)
{
    if(i < size) return data[i];
    else throw out_of_range(i);
}

void array::push(double v) throw(overflow)
{
    if(size < capacity) data[size++]=v;
    else throw overflow();
}

double array::pop() throw(underflow)
{
    if(size > 0) return data[--size];
    else throw underflow();
}

void main_loop(array& a)
{
    char c;
    double v;
    size_t i;
    for(;;)
    {
        cout << "Введите "+\"
            " \\"-\" для извлечения, \"i\" для просмотра "
            "i-го элемента, \"a\" для выхода: ";
        cin >> c;
        try {
            switch(c) {
                case '+':
                    cout << "Введите добавляемое число: ";
                    cin >> v;
                    a.push(v);
                    break;
                case '-':
                    v=a.pop();
                    cout << "Извлечено число " << v << endl;
            }
        }
    }
}

```

```

        break;
    case 'i':
        cout << "Введите индекс: ";
        cin >> i;
        v=a[i];
        cout << "Искомый элемент равен " << v << endl;
        break;
    case 'a':
        return;
        break;
    }
}
catch(out_of_range& e)
{
    cout << "Попытка доступа к элементу с недопустимым индексом " << e.i << endl;
    throw;
}
catch(overflow)
{
    cout<<"Операция не выполнена, так как массив переполнен\n";
}
catch(underflow) {
    cout << "Операция не выполнена, так как массив пуст\n";
}
}
main()
{
    double v;
    try
    {
        cout << "Введите емкость массива: ";
        cin >> v;
        array a(v);
        main_loop(a);
    }
    catch(general_error& e)
    {
        cout << "Произошла неустранимая ошибка следующего типа: " << e.message << endl;
    }
}
}

```

#### 10.4.4 Спецификация исключений

Если некоторая функция содержит инструкции `throw`, в ее заголовке можно явно прописать, какие исключения она может генерировать:

```
void f() throw (x,y,z);
```

В примере функция `f()` может генерировать исключения с классами-индикаторами `x`, `y`, `z` (и производными от них). Если заголовок описан как `<void f() throw ()>` — функция не генерирует исключений вообще. И, наконец, если в заголовке функции ничего не указано, она может генерировать любые исключения (без этого последнего правила не смогли бы работать программы, не использующие обработку исключений).

Если функция попытается сгенерировать исключение, отсутствующее в ее списке — вызовется специальная функция `void unexpected()`, которая в свою очередь вызовет функцию `void terminate()`, а та вызовет функцию `abort()`, аварийно завершающую программу. Программист имеет возможность заменить

функцию `unexpected()` или функцию `terminate()` — или сразу обе — на свои собственные, изменив таким образом обработку неспецифицированных исключений. Для такой замены нужно вызвать специальные библиотечные функции `set_unexpected()` и `set_terminate()`, передав им адреса новых функций в качестве аргументов. Наглядно можно увидеть действие этих функций в примере из раздела 10.4.2, в котором не перехватывается исключение `general_error`. Это исключение выбрасывается в примере конструктором класса `array` в том случае, если пользователь попытается создать массив нулевой емкости. Оно оказывается необработанным, т. к. создание массива находится за пределом блока `try`. К сожалению, стандартная функция не может знать о структуре класса-индикатора, и потому текстовое сообщение, оставленное конструктором, оказывается невостребованным. Программа сообщает имя неперехваченного класса-индикатора, после чего выполняет аварийное завершение работы.

#### 10.4.5 Стандартные классы — индикаторы исключений

Стандартная библиотека C++ содержит иерархию стандартных классов-индикаторов исключений (рис. 10.5), объявленных в заголовочном файле `stdexcept`. Эти индикаторы можно использовать в собственных программах.

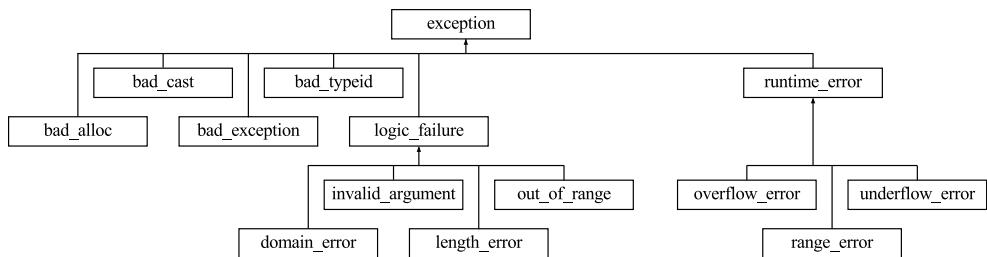


Рис. 10.5: Предопределенные индикаторы исключений

Назначение каждого класса-индикатора представлено в табл. 10.3.

Таблица 10.3: Стандартные классы-индикаторы исключений

Исключение	Описание
<code>exception</code>	базовый класс для всех стандартных исключений C++
<code>bad_alloc</code>	неудача выделения памяти; может генерироваться оператором <code>new</code>
<code>bad_cast</code>	ошибка динамического приведения типов, генерируется <code>dynamic_cast</code>

Таблица 10.3 — продолжение

Исключение	Описание
<code>bad_exception</code>	предназначено для обработки непредусмотренных исключений в программе
<code>bad_typeid</code>	генерируется оператором <code>typeid</code> (оператор, возвращающий имя типа, которому принадлежит аргумент), если не удается определить тип объекта
<code>logic_error</code>	исключение, связанное с ошибкой в логике работы программы, которая, теоретически, может быть обнаружена при чтении кода
<code>domain_error</code>	генерируется при выходе из математической области допустимых значений
<code>invalid_argument</code>	генерируется при получении недопустимого аргумента
<code>length_error</code>	генерируется при попытке создания слишком длинной строки
<code>out_of_range</code>	выход индекса за допустимую границу
<code>runtime_error</code>	исключение, связанное с ошибкой, которая, теоретически, не может быть обнаружена при чтении кода
<code>overflow_error</code>	генерируется при обнаружении математического переполнения
<code>range_error</code>	генерируется при попытке сохранить значение, выходящее за границы допустимого диапазона
<code>underflow_error</code>	генерируется при математической ошибке исчезновения порядка

Базовый класс `exception` содержит конструктор по умолчанию, конструктор копирования, деструктор, перегруженный оператор присваивания, а также единственный метод `what()`, возвращающий ASCII-строку с человеко-читаемым описанием исключительной ситуации. Классы-потомки могут добавлять к этому свой собственный функционал, в зависимости от типа ошибок, для обработки которых они предназначены. Однако стандартные классы-индикаторы по существу являются простыми обертками над `exception`, ограничиваясь возможностью указать конструктору класса-индикатора сообщение, которое должен возвращать метод `what()`.

Воспользуемся в следующем примере двумя стандартными классами-индикаторами:

- выбрасываемым автоматически при выходе индекса за пределы строки (вспользуемся классом `string` из стандартной библиотеки);
- выбрасываемым при ошибке в ходе выполнения (соответствующее исключение будем генерировать сами).

```
#include <string>
```

```

#include <stdexcept>
#include <iostream>
using namespace std;
int main ()
{
    //перехват выхода индекса за границу массива:
    try
    {
        string s(" sdf ");
        s.replace (100, 1, 1, 'c');
    }
    catch (out_of_range &e)
    {
        cout << "Обнаружен выход индекса за границу массива: " << e.what () << endl;
    }
    catch (exception &e)
    {
        cout << "Обнаружена ошибка неопределенного вида: " << e.what () << endl;
    }
    catch (...)
    {
        cout << "Неизвестная ошибка\n";
    }
    //перехват ошибки, возникающей в момент выполнения:
    try
    {
        throw runtime_error ("ошибка в ходе выполнения");
    }
    catch (runtime_error &e)
    {
        cout << "Обнаружена ошибка при выполнении программы: " << e.what () << endl;
    }
    catch (exception &e)
    {
        cout << "Обнаружена ошибка неопределенного вида: " << e.what () << endl;
    }
    catch (...)
    {
        cout << "Неизвестная ошибка\n";
    }
    return 0;
}

```

В первом случае исключение с индикатором `out_of_range` генерируется автоматически, когда мы допускаем выход индекса за границу строки (такая возможность реализована классом `string` из стандартной библиотеки). Во втором случае исключительную ситуацию с индикатором `runtime_error` (обозначающим ошибку, возникающую в момент выполнения программы) мы порождаем принудительно, передавая конструктору объекта — индикатора исключения строку с описанием подробностей.

## 10.5 Шаблоны классов

Шаблоны классов во многом похожи на шаблоны функций, рассмотренные ранее. Точно также, как и в случае шаблонов функций, шаблоны классов позволяют отдельить общий алгоритм от его реализации под конкретные типы данных.

Классический пример ситуации, когда выгодно применять шаблоны классов — это так называемые контейнеры, т. е. классы, содержащие наборы некоторых

значений (динамические списки, массивы, множества). Закладывая тип данных элемента как параметр шаблона, можно создать универсальный класс-контейнер, а на его основе порождать объекты для хранения наборов элементов конкретного типа — контейнер целых чисел, контейнер строк и т. д.

Описание шаблона класса также имеет много общего с шаблоном функции. Описание класса точно также начинают с ключевого слова `template`, за которым следует список формальных параметров шаблона в угловых скобках. Этот же заголовок повторяется и перед описанием методов класса. В качестве параметров шаблона можно передавать типы данных или константы, но перед идентификатором, обозначающим тип данных, в списке формальных параметров ставится ключевое слово `class`.

В отличие от шаблонов функций, для которых фактические параметры шаблона (т. е. конкретные типы данных) определяются по типам аргументов, переданных функции, для шаблонов классов фактические параметры необходимо передавать явно. Список фактических параметров шаблона указывается после имени класса в угловых скобках во всех случаях, когда имя шаблонного класса используется в программе — например, при порождении объектов, или при указании принадлежности элемента-члена класса. Дальнейшее обращение с порожденными объектами не отличается от обычных классов.

Рассмотрим в качестве простого примера уже знакомый класс `point`, который хранит пару координат точки и на этот раз имеет метод `info()`, выводящий координаты на экран.

```
#include<iostream>
using namespace std;
template <class Type>
class point
{
    Type x, y;
    //...
public:
    point(Type x, Type y) { this->x=x; this->y=y; }
    void info();
};
template <class Type>
void point<Type>::info()
{
    cout << "Координаты точки: x=" << x << ", y=" << y << endl;
}
main()
{
    point<float> f(10.1, 20.5);
    f.info();
}
```

Как видим, конкретный тип (в нашем случае `float`) мы указали при создании объекта в угловых скобках. Точно так же мы могли указать любой стандартный тип данных, а могли — пользовательский тип, объявленный в программе. Однако необходимо помнить, что шаблоны функций и шаблоны классов могут работать только для тех типов данных (в т. ч. классов), которые поддерживают необходимые операции. Например, если мы захотим создать экземпляр класса `point` для хранения пары объектов какого-то собственного класса `X`, этот класс

должен содержать конструктор копирования, а также поддерживать перегрузку двух использованных в `point` операторов:

```
class X
{
    .....
public:
    X(X &); //конструктор копирования
    friend ostream& operator<<(X &);
    .....
};
```

Как упоминалось, в качестве параметров шаблона можно передавать и константы. Рассмотрим пример, где константа передается шаблонному классу `square_matrix`, хранящему квадратную матрицу заданной размерности. Такой шаблон позволит легко создавать объекты типов «матрица 20x20 целых чисел», или «матрица 10x10 типа `double`».

```
#include <iostream>
using namespace std;

template <class Type, int n>
class square_matrix
{
    Type *data;
public:
    square_matrix(){ data = new Type[n*n]; }
    void print();
    // ...
};

template <class Type, int n>
void square_matrix<Type, n>::print()
{
    for (int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            cout << data[i*n+j] << '\t';
        }
        cout << endl;
    }
}

int main()
{
    cout << "Матрица 5x5 целых чисел: \n";
    square_matrix<int, 5> m1;
    m1.print();
    cout << "Матрица 10x10 значений типа double: \n";
    square_matrix<double, 10> m2;
    m2.print();
    return 0;
}
```

Для простоты приведенный пример умеет только порождать матрицу заданного размера с нулевыми элементами, а также построчно выводить ее на экран.

Шаблоны классов, как и классы, поддерживают механизм наследования. Все принципы наследования при этом остаются неизменными, что позволяет построить иерархическую структуру шаблонов, аналогичную иерархии классов.

### 10.5.1 Типаж шаблона

Иногда шаблонная функция должна реализовать нестандартное поведение для какого-либо определенного типа данных. В этом случае в дополнение к шаблону объявляют отдельную перегруженную версию функции для нужного типа. Например, если функция, возвращающая минимальный из двух аргументов, не может нормально работать для строковых данных, переданных указателем на тип `char`, проблема решается так:

```
template<class Type>
Type min(Type a, Type b)
{
    return a<b?a:b;
}
char *min (char *a, char *b)
{
    strcmp(a,b)<0?a:b;
}
```

С шаблонами классов может возникать аналогичная проблема, но решается она обычно несколько иначе.

Предположим, при создании шаблонного класса `matrix`, работающего с матрицами элементов произвольного типа, оказалось, что нет возможности создать код, одинаково обрабатывающий все нужные типы данных — т. е. работа некоторых методов класса *должна* зависеть от типа элементов матрицы. В довершение ко всему, типы элементов матрицы могут быть встроенными типами данных C++, поэтому нет никакой возможности заставить тип элемента нести дополнительную информацию об особенностях его обработки. В результате написать единственный шаблонный класс оказывается невозможно, а его переписывание под каждый конкретный тип данных возможно, но по определению лишено смысла:

```
template <class Type>
class matrix
{
    // ...
};

template<>
class matrix<long>
{
    // ...
};

template<>
class matrix<int>
{
    // ...
};

...
```

Фактически это означает несколько раз переписать класс заново.

Ситуация еще более осложняется, если разработчик класса хочет оставить возможность для использования в нем новых типов элементов (без переписывания класса с нуля).

В такой ситуации все, что зависит от изменений типа данных, собирают в одном месте и называют *типажом* (англ. «trait»). Типаж передают классу как еще один параметр шаблона. Типаж является небольшим по объему классом, содержащим все операции, зависящие от типа данных, и при необходимости его

несложно переписать под нужный тип. Более того, поскольку для большинства типов никакого специфического поведения не требуется, в типаж сам по себе может быть шаблоном класса. В этом случае как дополнительный параметр шаблона будет передаваться шаблонный класс, основанный на том же типе, что передан в качестве основного параметра — кроме тех редких случаев, когда требуется реализовать специфическое поведение:

```
template<class Type>
class MatrixTraits
{
    // ...
};

template<class Type>
class MatrixTraits<int>
{
    // ...
};

template<class Type, class Traits>
class matrix
{
    // ...
};

matrix <int, MatrixTraits<int>> m1;
```

Как правило, члены класса-типажа являются статическими функциями, поэтому его обычно используют без создания объекта.

### 10.5.2 Пример реальной иерархии шаблонов

Стандартная библиотека C++ практически полностью построена на шаблонах и потому представляет достаточно примеров профессионального использования данного механизма. Рассмотрим в качестве наглядной иерархии шаблонов уже знакомые нам классы потокового ввода-вывода.

Изначально библиотека потокового ввода-вывода действительно представляла собой такую иерархию классов, которая изображена на рис. 10.4. Однако по мере увеличения спроса на приложения, работающие с текстом сразу на нескольких языках, встал вопрос о поддержке кодировки Unicode, позволяющей совмещать в одной строке символы разных национальных алфавитов. В зависимости от языка, символ в Unicode может кодироваться различным количеством байт — от одного до четырех. В C++ для поддержки таких символов существует тип `wchar_t` (от англ. wide characters — «широкие символы»). Фактически понадобилось создать иерархию классов, аналогичную классам `iostream`, но работающих с типом данных `wchar_t` вместо `char`. В итоге библиотека `iostream` была переработана на основе механизма шаблонов.

Классы, основанные на шаблонах, носят имена, аналогичные описанным в разделе 10.3.6, с добавлением приставки «`basic`» и знака подчеркивания: `basic_ios`, `basic_istream`, `basic_ostream` и т. д. Привычные программисту имена классов для работы с символами типа `char` (как, впрочем, и с `wchar_t`) реализованы через подстановку имени типа в конструкции `typedef`:

```
typedef basic_ios<char> ios;
```

```
typedef basic_ios<wchar_t> wios;
typedef basic_istream<char> istream;
typedef basic_istream<wchar_t> wistream;
....
```

Используя базовые шаблоны библиотеки, можно реализовать потоковый ввод-вывод на любом собственном типе символьных данных вместо существующих, подставив его в качестве параметра шаблона и обеспечив работу соответствующих операторов.

Хотя по виду конструкции `typedef` может показаться, что шаблоны библиотеки потокового ввода-вывода имеют один параметр, на самом деле это не совсем так. Второй параметр — это как раз *типаж* символов, т. е. отдельный шаблонный класс, который реализует базовые операции с символами и строками для заданного типа символов. Эти базовые операции — присваивание символов, копирование и сравнение их последовательностей, приведение к целому типу и др. Данный параметр имеет значение по умолчанию, и потому может не использоваться при объявлении экземпляров шаблона. В оригинале же шаблоны классов потокового ввода-вывода выглядят следующим образом (в виду однотипности, приведем по одному примеру для стандартного ввода-вывода, работы с файлами и со строками):

```
template <class charT, class traits=char_traits<charT>> basic_istream;
template <class charT, class traits=char_traits<charT>> basic_ifstream;
template <class charT, class traits = char_traits<charT>, class Allocator =
allocator<charT>> basic_istringstream;
```

Именно эти потоковые шаблоны определяют на самом деле методы для разбора и форматирования, являющиеся перегруженными версиями операторов ввода `operator>>` и вывода `operator<<`.

Аналогично реализованы шаблоны для потоковых буферов:

```
template <class charT, class traits = char_traits<charT>> basic_streambuf;
template <class charT, class traits = char_traits<charT>> basic_filebuf;
```

## 10.6 Элементы стандартной библиотеки C++

### 10.6.1 Базовые понятия

Стандартная библиотека C++ — это общий набор шаблонов классов и алгоритмов, позволяющий программистам легко реализовывать стандартные структуры данных, такие как очереди, списки и стеки.

В библиотеке выделяют пять основных компонентов:

- Контейнер (`container`) — хранение набора объектов в памяти.
- Итератор (`iterator`) — обеспечение средств доступа к содержимому контейнера.
- Алгоритм (`algorithm`) — определение вычислительной процедуры.
- Адаптер (`adaptor`) — адаптация компонентов для обеспечения различного интерфейса.

- Функциональный объект (**functor**) — сокрытие функции в объекте для использования другими компонентами.

### 10.6.2 Контейнеры

Будем считать, что стандартная библиотека реализует следующие контейнеры:

- **vector** — линейный массив (особенно эффективен при добавлении элементов в конец);
- **list** — двухсвязанный список (более эффективен при вставке и перестановке элементов);
- **set** — ассоциативный массив уникальных ключей (математический тип множества);
- **multiset** — ассоциативный массив с возможность дублирования ключей;
- **bitset** — массив, обеспечивающий компактное хранение заданного количества битов;
- **map** — ассоциативный массив с уникальными ключами и значениями (хорош при поиске по ключу);
- **multimap** — ассоциативный массив с возможность дублирования ключей и значений;
- **stack** — структура данных типа стек;
- **queue** — структура данных типа очередь;
- **priorityqueue** — очередь с приоритетами;
- **deque** — очередь с двухсторонним доступом.

Например, стек целых чисел можно объявить так:

```
stack<int> s;
```

С чуть большими затратами можно заставить работать контейнеры с собственным типом данных.

Основные методы, которые присутствуют почти во всех коллекциях стандартной библиотеки, приведены ниже.

- **empty** — определяет, является ли коллекция пустой.
- **size** — определяет размер коллекции.
- **begin, end** — указывают на начало и конец коллекции.
- **rbegin, rend** — то же но для желающих пройти коллекцию от конца к началу
- **clear** — удаляет все элементы коллекции (если в коллекции сохранены указатели, нужно еще удалить все элементы вручную вызовом **delete**).
- **erase** — удаляет элемент или несколько элементов из коллекции.
- **capacity** — вместимость коллекции определяет реальный размер - то есть размер буфера коллекции, а не то, сколько в нем хранится элементов. Когда вы создаете коллекцию, то выделяется некоторое количество памяти. Как только размер буфера оказывается меньшим, чем размер, необходимый для хранения всех элементов коллекции, происходит выделение памяти для нового буфера, а все элементы старого копируются в новый буфер.

- `insert` — вставка элемента в произвольном месте последовательности

Есть также и необязательные операции: `front`, `back`, `push_front`, `push_back`, `pop_front`, `pop_back`, и оператор `[]`.

### 10.6.3 Итераторы

Итератор (`iterator`) согласно Википедии — объект, позволяющий программисту перебирать все элементы коллекции без учета особенностей ее реализации. В простейшем случае итератором в низкоуровневых языках является указатель.

В C++ есть несколько разных типов итераторов (табл. 10.4):

Таблица 10.4: Типы итераторов

Итератор	Описание
<code>input_iterator</code> (для чтения)	Читают значения с движением вперед. Могут быть инкрементированы, сравнены и разыменованы.
<code>output_iterator</code> (для записи)	Пишут значения с движением вперед. Могут быть инкрементированы и разыменованы.
<code>forward_iterator</code> (однонаправленные)	Читают или пишут значения с движением вперед. Комбинируют функциональность предыдущих двух типов с возможностью сохранять значение итератора.
<code>bidirectional_iterator</code> (двунаправленные)	Читают и пишут значения с движением вперед или назад. Похожи на однонаправленные, но их также можно инкрементировать и декрементировать.
<code>random_iterator</code> (с произвольным доступом)	Читают и пишут значения с произвольным доступом. Самые мощные итераторы, сочетающие функциональность двунаправленных итераторов и возможность выполнения арифметики указателей и сравнений указателей.
<code>reverse_iterator</code> (обратные)	Или итераторы с произвольным доступом, или двунаправленные, движущиеся в обратном направлении.

Итераторы обычно используются парами: один для указания текущей позиции, а второй для обозначения конца коллекции элементов. Итераторы создаются объектом-контейнером, используя стандартные методы `begin()` и `end()`. Функция `begin()` возвращает указатель на первый элемент, а `end()` — на воображаемый несуществующий элемент, следующий за последним.

К элементам контейнера — например, `vector` — можно обращаться и по номерам, как к элементам классического массива — и с помощью итераторов:

Необъектный подход	Правильный (объектный) подход
<pre>#include &lt;iostream&gt; #include &lt;vector&gt; using namespace std; main() {     vector&lt;int&gt; a;     //добавляем элементы     a.push_back(1);     a.push_back(4);     a.push_back(8);     for(int y=0;y&lt;a.size();y++)     {         //выводим 1 4 8         cout&lt;&lt;a[y]&lt;&lt;" ";     } }</pre>	<pre>#include &lt;iostream&gt; #include &lt;vector&gt; using namespace std; main() {     vector&lt;int&gt; a;     vector&lt;int&gt;::iterator it;     //добавляем элементы     a.push_back(1);     a.push_back(4);     a.push_back(8);     for(it=a.begin();it!=a.end();it++)     {         //выводим 1 4 8         cout&lt;&lt;*it&lt;&lt;" ";     } }</pre>

В отличие от счетчика цикла, итератор обеспечивает доступ к элементу, а не только его перебор. В отличие от операции индексации, итератор «не портится» при добавлении в контейнер новых элементов. Кроме того, индексация иногда вообще неприменима — например, в коллекциях с медленным произвольным доступом, таких как списки (`list`).

Рассмотрим, как использовать контейнеры на примере класса `vector`:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
main()
{
    //Объявляем вектор из целых чисел
    vector<int> k;
    //Добавляем элементы в конец вектора
    k.push_back(22);
    k.push_back(11);
    k.push_back(4);
    k.push_back(100);
    vector<int>::iterator p;
    cout << "Вывод неотсортированного вектора:\n";
    for(p = k.begin(); p<k.end(); p++) {
        cout << *p << ',';
    }
    //Сортировка вектора.
    sort(k.begin(), k.end());
    cout << "\nВывод отсортированного вектора:\n";
    for(p = k.begin(); p<k.end(); p++) {
        cout << *p << ',';
    }
    cout << endl;
}
```

Как видно, пример сначала заполняет вектор целых чисел четырьмя значениями, затем поэлементно выводит содержимое вектора на экран, сортирует с

использованием алгоритма `sort`, а затем снова выводит на экран. Вывод программы выглядит следующим образом:

```
Выход неотсортированного вектора:  
22 11 4 100  
Выход отсортированного вектора:  
4 11 22 100
```

#### 10.6.4 Алгоритмы

В состав стандартной библиотеки входит группа функций, выполняющих некоторые стандартные действия, например поиск, преобразование, сортировку, поштучный перебор элементов, копирование и т. д. Они называются *алгоритмами*. Параметрами для алгоритмов, как правило, служат итераторы. Алгоритму нет никакого дела до типа переданного ему итератора, лишь бы итератор подпадал под определенную категорию. Так, если параметром алгоритма должен быть односторонний итератор, то подставляемый итератор должен быть либо односторонним, либо двунаправленным, либо итератором произвольного доступа.

Например, алгоритм простого поиска `find` просматривает элементы подряд, пока нужный не будет найден. Для такой процедуры вполне достаточно итератора ввода. С другой стороны, алгоритм более быстрого бинарного поиска `binary_search` должен иметь возможность переходить к любому элементу последовательности, и поэтому требует итератора с произвольным доступом.

Многие алгоритмы получают в качестве параметра различные функции. Эти функции используются для сравнения элементов, их преобразования и т. д. Однако вызов функции по указателю — ресурсоемкая операция. Вместо указателя на функцию можно передать объект любого класса с перегруженным оператором вызова функции `operator()`. Одно из преимуществ обращения к переопределенному оператору вызова функции вместо собственно ее вызова заключается в том, что переопределенный оператор может быть реализован в классе как встроенная функция, т. е. код оператора будет подставлен вместо вызова.

Рассмотрим пример алгоритма `find`, который находит первое вхождение заданного значения в коллекцию. Алгоритм принимает в качестве аргументов пару итераторов и искомое значение; соответственно возвращается итератор, указывающий на первое вхождение заданного значения. Благодаря универсальности механизма итераторов, алгоритм будет работать со структурой любого типа, в том числе и с обычными массивами языка С. Например, чтобы найти первое вхождение числа 7 в массив целых, требуется выполнить следующий код, использующий в качестве итераторов обычные указатели:

```
int data[100];  
...  
int *where;  
where = find(data, data+100, 7);
```

Поиск первого значения в целочисленном векторе выглядит приблизительно так же:

```
vector<int> a;  
...  
vector<int>::iterator where;  
where = find(a.begin(), a.end(), 7);
```

## 10.7 Задачи для самостоятельного решения

### 10.7.1 Иерархия классов

Определить иерархию наследования из двух классов в соответствии с номером задания. Каждый класс снабдить свойствами и методами в соответствии с предметной областью, указанной в варианте задания. В базовом классе предусмотреть метод `info()`, выводящий на экран информацию об объекте. Предусмотреть конструкторы, инициализирующие свойства объектов переданными данными либо значениями по умолчанию. Написать демонстрационную программу, создающую 4-5 объектов и выводящую на экран информацию о них. Варианты классов:

1. «Водный транспорт», «Грузовое судно»
2. «Летательный аппарат», «Дирижабль»
3. «Здание», «Коттедж»
4. «Двигатель», «Двигатель внутреннего сгорания»
5. «Устройство печати», «Струйный принтер»
6. «Устройство ввода», «Цифровая камера»
7. «Растровое изображение», «Репродукция картины»
8. «Млекопитающее», «Собака»
9. «Транспортное средство», «Легковой автомобиль»
10. «Печатное издание», «Номер журнала»
11. «Документ», «Квитанция об оплате»
12. «Пищевой продукт», «Йогурт»
13. «Корпусная мебель», «Книжный шкаф»
14. «Проверка знаний», «Экзамен»
15. «Носитель информации», «Компакт-диск»
16. «Аудиозапись», «файл в формате MP3»
17. «Видеозапись», «Художественный фильм»
18. «Транспортное средство», «Маршрутный автобус»
19. «Средство связи», «Сотовый телефон»
20. «Человек», «Член клуба»
21. «Птица», «Почтовый голубь»
22. «Электронная карта», «Аbonемент на проезд»
23. «Дата», «День рождения»
24. «Удостоверение», «Паспорт»
25. «Сотрудник компании», «Начальник отдела»

### 10.7.2 Перегрузка операторов

Реализовать класс, содержащий коллекцию объектов, методы для включения и удаления элементов, вывода содержимого коллекции на экран, а также перегруженный в соответствии с заданием оператор. Написать программу, заполняющую коллекцию несколькими элементами и демонстрирующую пользователю работу перегруженного оператора для элементов коллекции:

1. «--» (вычитание одной коллекции из другой), класс «множество символов»
2. «+» (объединение коллекций), класс «множество целых чисел»
3. «\*» (пересечение коллекций), класс «множество целых чисел»
4. «!=» (сравнение коллекций на неравенство), класс «неупорядоченный массив вещественных чисел»
5. «==» (сравнение коллекций на неравенство), класс «упорядоченный массив символов»
6. «[]» (получение элемента по его номеру в коллекции), класс «неупорядоченный массив целых чисел»
7. «[]» (получение элемента по его номеру в коллекции), класс «упорядоченный массив вещественных чисел»
8. «%» (проверка элемента на принадлежность коллекции), класс «множество целых чисел»
9. «%» (проверка элемента на принадлежность коллекции), класс «упорядоченный массив символов»
10. «<<<» (удаление элемента из коллекции с его выводом на экран), класс «множество целых чисел»
11. «>>>» (добавление введенного с клавиатуры элемента в коллекцию), класс «множество символов»
12. «>=» (проверка на включение коллекции, заданной вторым аргументом, в начальную часть коллекции, заданной первым аргументом), класс «упорядоченный массив символов»
13. «<=» (проверка на включение коллекции, заданной первым аргументом, в начальную часть коллекции, заданной вторым аргументом), класс «неупорядоченный массив символов»
14. «++» (добавление элемента со значением, на единицу больше последнего добавленного элемента), класс «упорядоченный массив целых чисел»
15. «++» (добавление элемента со значением, на единицу больше последнего добавленного элемента), класс «стек целых чисел»
16. «--» (удаление последнего добавленного элемента), класс «упорядоченный массив целых чисел»
17. «--» (удаление элемента), класс «очередь вещественных чисел»
18. «--» (опустошение коллекции), класс «множество вещественных чисел»
19. «>>>» (добавление введенного с клавиатуры элемента в коллекцию), класс «очередь целых чисел»
20. «<<<» (удаление элемента из коллекции с его выводом на экран), класс «очередь вещественных чисел»

21. «>>» (добавление введенного с клавиатуры элемента в коллекцию), класс «стек символов»
22. «<<» (удаление элемента из коллекции с его выводом на экран), класс «стек целых чисел»
23. «\*» (умножение всех элементов коллекции на заданное число), класс «неупорядоченный массив вещественных чисел»
24. «/» (деление всех элементов коллекции на заданное число), класс «множество вещественных чисел»
25. «~» (смена регистра), класс «множество символов»

### 10.7.3 Обработка исключительных ситуаций

Снабдить класс из задания п. 10.7.1 проверкой на допустимость значений, передаваемых конструктору. В случае передачи недопустимых значений генерировать исключительную ситуацию. Предусмотреть не менее двух различных классов-индикаторов исключения, позволяющих передать обработчику необходимую информацию. Расширить демонстрационную программу показом обработки некорректной инициализации объектов.

# Глава 11

## Знакомство с Qt. Подготовка к работе

### 11.1 Знакомство с Qt. Обзор истории

Кроссплатформенный инструментарий разработки **Qt** появился впервые в 1995 году благодаря своим разработчикам Хаарварду Норду и Айрику Чеймб-Ингу. С самого начала создавался как программный каркас, позволяющий создавать кроссплатформенные программы с графическим интерфейсом. Первая версия **Qt** вышла 24 сентября 1995. Программы, разработанные с **Qt**, работали как под управлением операционных систем семейства Microsoft Windows™ так и под управлением Unix-подобных систем.

За годы разработки возможности **Qt** значительно выросли. Работа с сетью, базами данных, графикой, мультимедиа, Интернет и другие расширения превратили его в универсальный инструментарий для создания программ. **Qt** превратился в полноценный и мощный инструмент разработки, который значительно превзошел свои первоначальные возможности.

В июне 1999 года вышла вторая версия — **Qt 2.0**. А в 2000 году состоялся выпуск версии для встраиваемых систем, который назывался **Qt Embedded**. Версия **Qt 3.0** — 2001 год — работала в ОС семейства Windows™ и многих Unix-подобных ОС, таких как MacOS, xBSD, в различных вариантах Linux для персональных компьютеров и встраиваемых систем. Он имел 42 дополнительных класса, объем вырос до более чем 500 000 строк кода. Летом 2005 года состоялся выпуск **Qt 4.0**, который включал в совокупности около 500 классов и имел огромное количество существенных улучшений. Вместе с выпуском **Qt 4.5** вышло и специализированная интегрированная среда разработки **QtCreator**.

В декабре 2012 состоялся официальный выпуск **Qt5**. Эта версия кроссплатформенного средства разработки совместима с **Qt4**. Перенос кода с **Qt4** на **Qt5** не требует много усилий. В то же время, **Qt5** отличается рядом особенностей, улучшений и большим количеством новых возможностей.

Современное программное обеспечение достаточно сложное и должно соответствовать многим требованиям. Кроме пользовательских требований, налага-

емых на удобство и возможности программного продукта, есть и другие требования, касающиеся разработки программного обеспечения. Большую роль здесь играют средства, которыми программист пользуется в процессе своей работы. Во многих случаях бывает удобно владеть инструментарием, который имеет достаточно широкую область применения и может служить для решения большого количества задач разного масштаба: от построения небольших программ для создания мощных программных пакетов. Также часто возникает вопрос о поддержке нескольких программных платформ, ведь, ориентируясь только на одну платформу, можно потерять большое количество потенциальных пользователей.

Инструментарий разработки **Qt** используют для создания *кроссплатформенных программ*. Здесь под этим утверждением мы подразумеваем программы, исходный текст которых можно скомпилировать на разных программных платформах (различные разновидности Linux, Windows, MacOS и т.д.) *практически без изменений или с незначительными изменениями*. Кроме того **Qt** используют и для разработки программ, имеющих характерный («родной», native) для программного окружения или даже собственный стилизованный интерфейс. Все это благодаря открытому свободному программному коду, удобному и логическому API и широким возможностям применения.

**Qt** расширяет возможности программиста с помощью набора макросов, метаинформации и сигнально-слотовых соединений, но использует при этом лишь средства языка C++ и является совместимым со всеми распространенными современными его компиляторами.

Наряду с традиционным для предыдущих версий **Qt** способом создания пользовательских интерфейсов, основанный на *виджетах* — визуальных элементах интерфейса (кнопки, флажки, выпадающие списки, поля ввода, слайдеры и т.д.), **Qt5** ставит большой акцент на использовании технологии **QtQuick**. В **Qt5** некоторые нововведения коснулись и синтаксиса для создания сигнально-слотовых соединений.

Программный код, зависящий от оконной системы в **Qt5**, был отделен и реорганизован в отдельные библиотеки расширения, что позволило упростить перенос **Qt** на новые платформы и адаптации для поддержки других оконных систем. Благодаря **QPA** (**Qt Platform Abstraction**) в **Qt5** реализована поддержка многих платформ для мобильных устройств.

Несмотря на эти изменения и усовершенствования, большинство программного кода созданного для **Qt4** совместимо с **Qt5** и компилируется с новой версией почти без изменений. Почти весь материал следующих разделов и примеры подходят для изучения как **Qt4**, так и **Qt5**. Большая часть изменений в **Qt5** касается разделения на модули.

### 11.1.1 Основные составляющие Qt

Рассмотрим основные составляющие кроссплатформенного средства разработки **Qt**: модули и инструменты.

На рис. 11.1 изображены основные составляющие Qt. Модули и инструменты доступны для разработки под целевые (Reference) и другие (Other) платформы. Средства Qt разделены по назначению на отдельные части — модули. Каждый из модулей выполнен в виде отдельной библиотеки. Разработчик имеет возможность выбрать модули, которые он использует в программе. Модули имеют взаимозависимости: одни модули используют возможности, которые предоставляют другие. Основу составляют основные (Essentials) модули:

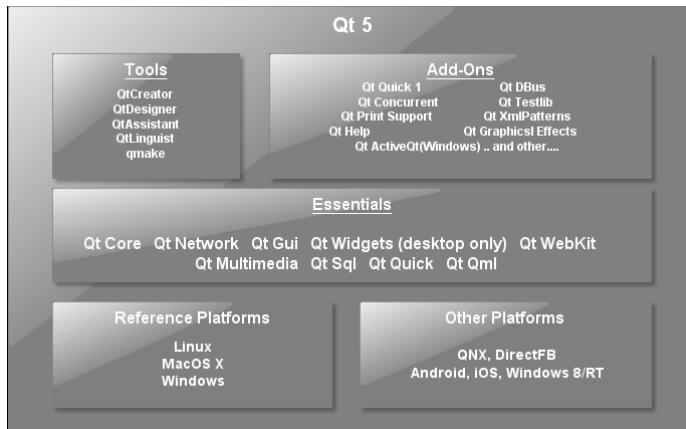


Рис. 11.1: Состав Qt5

- **Qt Core** — основной модуль, который содержит все базовые средства Qt. На его основе построены все другие модули. Каждая программа созданная с использованием Qt, использует этот модуль;
- **Qt Network** — модуль для работы с сетевыми средствами;
- **Qt Gui** — модуль поддержки графического вывода на экран. В Qt4 он также содержит содержит набор виджетов для создания графического интерфейса пользователя. В Qt5 виджеты вынесены в отдельный модуль;
- **Qt Widgets** — модуль, который содержит набор виджетов для создания графического интерфейса пользователя (Qt5)
- **Qt WebKit** — средства работы с Веб;
- **Qt WebKit Widgets** — виджеты для работы с Веб (Qt5);
- **Qt Multimedia** — средства работы с мультимедийными устройствами и файлами;
- **Qt Multimedia Widgets** — виджеты для работы с мультимедийными устройствами и файлами (Qt5);
- **Qt Sql** — средства работы с базами данных;
- **Qt Qml** — поддержка декларативной языка QML для разработки динамических визуальных интерфейсов (Qt5);

- **Qt Quick** — поддержка создания динамических визуальных интерфейсов (Qt5);
- **Qt Quick Controls** — использование технологии QtQuick для создания традиционного для рабочих столов графического интерфейса (Qt5);
- **Qt Quick Layouts** — компоновка для элементов QtQuick (Qt5).

Существует также много дополнительных (Add-On) модулей. Стоит заметить, что разделение на основные и дополнительные модули характерно Qt5 в отличие от предыдущих версий. Названия некоторых модулей в Qt5 по сравнению с Qt4 были изменены, а некоторые средства были вынесены в отдельные или перенесены в другие модули. Эти изменения необходимо учитывать при переносе программ, которые были разработаны с использованием Qt4. Почти все примеры, которые мы будем рассматривать, работают как с Qt4 так и Qt5. В случаях, когда это существенно, мы будем указывать на отличия.

Кроме модулей, в состав инструментария входят инструменты разработки, исходные тексты Qt, примеры программ и документация.

## 11.2 Лицензирование Qt

Qt распространяется по условиям трех различных лицензий: GNU GPL v3, GNU LGPL v3 и по коммерческой лицензии компании Digia. Здесь мы лишь кратко осмотрим основные положения этих лицензий и что это означает для программ, которые используют соответственно лицензированный Qt.

### 11.2.1 GPL

Программа должна быть открыта, свободно распространяться, исходные тексты программы и все изменения в исходных текстах Qt должны пребывать в свободном доступе.

### 11.2.2 LGPL

Исходные тексты программы могут быть как открытыми так и закрытыми. В случае, если программа является закрытой и планируется коммерческое использование программы — Qt должен связываться с программой в виде динамических библиотек. Конечно, в этом случае нельзя вставлять и использовать любые исходные тексты Qt в программе. Также любые изменения в исходных текстах Qt должны быть пребывать в свободном доступе.

### 11.2.3 Commercial

В случае коммерческой лицензии, кроме возможности закрывать, модифицировать любым образом текст программы, модифицировать или закрывать изменения в коде Qt и произвольно выбирать лицензию и способ распространения

программы, предоставляется также поддержка и консультации по использованию Qt.

### 11.3 Справка и ресурсы

Важнейшей помощницей при разработке с использованием Qt является интегрированная справка. Документация Qt удивительно удобна в использовании и создана для быстрого поиска среди богатого инструментария Qt. Она содержит не только описания классов, входящих в состав модулей, но и краткие примеры использования методов и классов, полные тексты демонстрационных программ, освещающих возможности Qt. Также здесь можно найти несколько пошаговых инструкций для начинающих и статьи, посвященные описанию и объяснению механизмов работы и различных аспектов использования инструментария.

Для просмотра интегрированной справки можно воспользоваться как средой Qt Creator, так и специальной отдельной программой, которая называется Qt Assistant и является частью инструментария Qt.

Для вызова встроенной справки вы можете воспользоваться одним из следующих способов:

- перейдите в режим справки среды Qt Creator — Help (комбинация клавиш **Ctrl+7**);
- установите курсор на название класса или метода и нажмите F1 — сразу выполнит поиск и откроет соответствующий раздел справки в боковой панели.

В режиме справки или в случае использования Qt Assistant слева от окна документации расположена панель, которая может переключаться в несколько различных режимов: Закладки (Bookmarks), Содержание (Contents), Указатель (Index) и Поиск (Search). Режим панели определяется выпадающим списком сверху. Особенно удобно пользоваться режимом Указатель (Index) при работе: как только пользователь вводит начало названия класса, метода или статьи, в справке выполняется поиск и отображение совпадений. Это особенно пригодится для быстрой навигации и поиска в справке.

Следует помнить, что эта книга, как и любая другая, не может быть исчерпывающим обзором Qt, поэтому дальнейшая работа с ней будет требовать параллельного исследования документации. Вот несколько советов:

- не пытайтесь запомнить все названия методов, классов и т. п. Сконцентрируйтесь на осмотре возможностей, основных концепциях и практике. Используйте справку для быстрого поиска и восстановления в памяти тех или иных деталей использования инструментов Qt;
- обратите внимание на большое количество примеров. Рассматривайте примеры параллельно с рассмотрением материала в книге;
- попробуйте сразу же находить классы и методы из следующих глав книги в справке и исследовать их, как только вы начинаете их изучение. Для этого особенно пригодится быстрая навигация и поиска в справке.

В сети Интернет существует большое количество ресурсов, статей, учебных видео посвященных Qt. Вот важнейшие из них:

- **Qt Project** (<http://qt-project.org/>) — главный сайт свободного инструментария разработки Qt;
- **Qt Diggia** (<http://qt.diggia.com/>) — официальный сайт коммерческой версии Qt;
- **Planet Qt** (<http://planet.qt-project.org/>) — сайт, который собрал десятки блогов посвященных Qt;
- **Qt Centre** (<http://www.qtcentre.org/>) — форум посвященный вопросам разработки;
- **Qt-Apps.org** (<http://qt-apps.org/>) — сайт посвященный открытому программному обеспечению созданному с использованием Qt.

## 11.4 Обзор настроек среды Qt Creator

Для разработки программ с использованием библиотеки Qt была создана интегрированная среда разработки Qt Creator. Ее первая версия была представлена одновременно с официальным выпуском Qt 4.5.0. Это полноценная кроссплатформенная среда для создания новых проектов и работы с ними.

Мы рассмотрим работу со средой Qt Creator версии 2.8.0, которая позволяет управлять целым рядом этапов разработки программы такими как: управление сеансами и проектами, редактирование и создание программного кода, конструирование пользовательского интерфейса программы, анализ быстродействия, анализ использования ресурсов, отладка, построение проекта, запуск программы.

Одно из первых действий, которое необходимо выполнить разработчику перед началом работы с Qt Creator — это настроить среду таким образом, чтобы с ней было удобно работать. Конечно, Qt Creator имеет стандартные настройки, которые уже достаточно удобны для работы. Несмотря на это, мы хотели бы обратить ваше внимание на некоторые настройки, которые особенно полезны в работе. Среди большого количества настроек мы рассмотрим лишь наиболее важные для работы, а именно настройки компиляции и настройки редактора кода. Для доступа к диалогу настройки используем главное меню (пункт Tools->Options..., см. рис. 11.2).

Сначала коснемся настроек компиляции. Для управления настройками, относящимися к построению проекта, Qt Creator использует понятие комплекта (Kit). Комплект (Kit) — это конфигурация, которую составляют версия Qt, компилятор и еще некоторые дополнительные настройки. Таким образом, QtCreator позволяет работать с несколькими различными версиями Qt, несколькими компиляторами в системе, выбирать и настраивать их комбинацию для построения проекта.

Стандартным для Linux и Mac OS X является компилятор GCC. Для Windows можно воспользоваться его свободным аналогом — MinGW, или компилятором MSVC, который входит в состав Microsoft Windows SDK или Visual Studio (SDK для Windows 7 можно получить бесплатно на официальном сайте Microsoft).

Если среда **Qt Creator** была установлена отдельно, то возможно необходимо будет добавить комплект самостоятельно. Для этого необходимо выполнить следующие шаги:

1. Перейдите на вкладку **Compilers** (Компиляторы) раздела **Build and Run** (Сборка и запуск) и проверьте наличие доступных компиляторов. Обычно наличие компиляторов MSVC (Windows) и GCC (Linux, MacOS) определяется автоматически. Для того, чтобы добавить компилятор MinGW, необходимо воспользоваться кнопкой **Add->MinGW**(Добавить->MinGW). Затем для добавленного компилятора ввести имя (поле **Name**) и указать полный путь к компилятору C++ — g++ (поле **Compiler path**);
2. Перейдите на вкладку **Qt Versions** (Версии Qt) и проверьте наличие доступных версий Qt. Установленную версию можно легко добавить в список воспользовавшись кнопкой **Add...** (Добавить). Для добавленной версии укажите имя (поле **Version name**) и полный путь к программе qmake (поле **qmake location**). Обычно данная программа содержится в папке **bin** в месте, куда был установлен Qt;
3. Перейдите на вкладку **Kits** (Комплекты). Добавьте новый комплект с помощью кнопки **Add** (Добавить). Выделите в списке новый комплект и задайте для него комбинацию из установленных компилятора (выпадающий список **Compiler**) и версии Qt (выпадающий список **Qt Version**). Далее задайте имя инструментария (поле **Name**) и сохраните изменения.

После выполнения этих действий, если компилятор и версия Qt, которые есть в составе инструментария, были установлены правильно, вы можете использовать комплект для построения проекта.

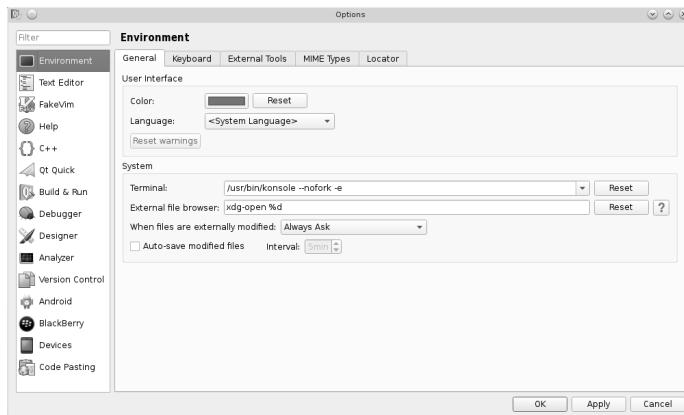


Рис. 11.2: Окно диалога настроек Qt Creator

Среди настроек редактора стоит отметить настройки горячих клавиш. Qt Creator предоставляет множество комбинаций клавиш для выполнения различ-

ных действий. Приведем некоторые из них (см. табл. 11.1), которые используются чаще всего.

Таблица 11.1: Некоторые важные горячие клавиши Qt Creator

Комбинация клавиш	Описание
Esc	Выполняет переход к редактированию кода. Несколько последовательных нажатий этой клавиши переключают пользователя в режим редактирования, закрывают панели вывода справки, отладки.
F4	Переключает редактор между файлом реализации (.cpp) и соответствующим заголовочным файлом (.h), которые содержат объявления интерфейса и реализации класса в соотвенно.
F2	Выполняет переход к месту объявления переменной, функции, класса, на имени которых стоял курсор при нажатии.
F1	Показывает справку для класса или метода Qt, на имени которого стоит курсор.
Ctrl+Shift+R	Переименование переменной, метода, класса, на имени которых стоит курсор. Имя будет изменено во всех местах, где встречается его использование: не только в текущем файле, но и в других файлах проекта. При замене имени будет учитываться область видимости имени, поэтому замена произойдет только в местах обращения к имени. Именно этим это действие отличается от обычного поиска и замены текста.
Ctrl+Shift+U	Поиск всех мест обращения к переменной, методу, классу на имени которого стоит курсор.
Ctrl+K	Открывает поле быстрого поиска (Locator).
Alt+Enter	Позволяет открыть доступные дополнительные действия для переменной, метода, класса, оператора в позиции курсора. Это дополнительные действия для рефакторинга (реорганизации и улучшения существующего кода) могут содержать изменение порядка параметров, изменения в текущем фрагменте кода, добавление фрагментов кода и т.д.
Ctrl+Space	Вызывает выпадающий список автозавершения код.
Ctrl+F	Поиск текста в текущем открытом файле.
Ctrl+Shift+F	Расширенный поиск текста в файле, проекте или группе проектов (доступны дополнительные настройки).

#### 11.4.1 Первый оконный проект

Для создания простого оконного проекта выберите в главном меню **File->New File or Project...** (Файл->Новый файл или проект...) или нажмите **Ctrl+N**. На экране появится окно мастера новых файлов и проектов (см. рис. 11.3).

Для создания нашего проекта выберем раздел **Applications** (Приложения) в списке **Projects** (Проекты) и выберем **Qt Widgets Application** как тип

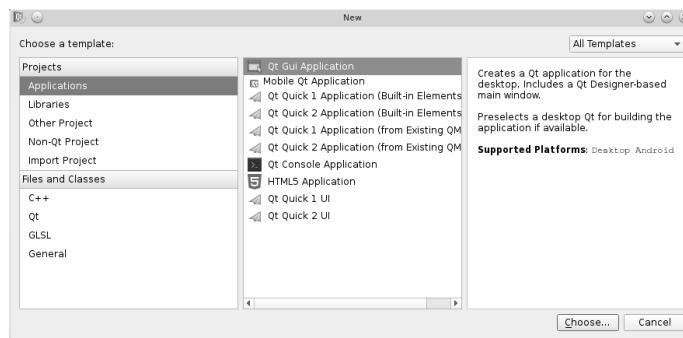


Рис. 11.3: Окно мастера новых файлов и проектов (шаг 1)

проекта (приложение на Qt с использованием виджетов). Нажмем кнопку **Choose...** (Выбрать...).

Далее нам необходимо ввести имя проекта в поле ввода **Name** (например **FirstGuiProject**). Поле ввода **Create in** содержит путь, где будет создан каталог с новым проектом. Флажок **Use as default project location** (Использовать как место по умолчанию для размещения проектов) означает, что путь расположения проекта сохраняется и для последующих новых проектов (рис. 11.4)

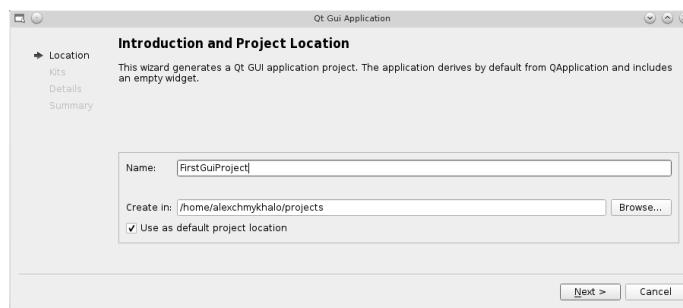


Рис. 11.4: Окно мастера новых файлов и проектов (шаг 2)

Во время следующего шага (рис. 11.5) необходимо указать комплект, с помощью которого среда будет строить новый проект. Оставим выбор инструментария по умолчанию и нажмем кнопку **Next** (Далее).

Следующее окно мастера (см. рис. 11.6) информирует о классах, которые будут сгенерированы автоматически для нового проекта. Для главного окна программы будет сгенерирован класс **MainWindow** (поле **Class name**), который будет



Рис. 11.5: Окно мастера новых файлов и проектов (шаг 3)

унаследован от класса `QMainWindow` (поле `Base class`). Класс `QMainWindow` обладает особенностями, характерными для главного окна программы, такими как главное меню, панель инструментов и т.п. Для кода класса `MainWindow` мастер создаст заголовочный файл — `mainwindow.h` (поле `Header file`) и файл реализации — `mainwindow.cpp` (поле `Source file`), а также добавит их в проект.

Флажок `Generate form` указывает на необходимость сгенерировать и добавить к проекту файл формы для главного окна.

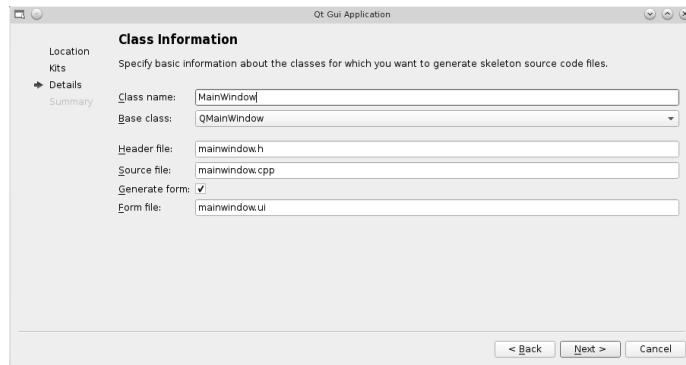


Рис. 11.6: Окно мастера новых файлов и проектов (шаг 4)

В завершающем окне мастера (см. рис. 11.7) нажмите `Finish` (Завершить).

Файлы формы позволяют редактировать вид окна с помощью визуального редактора интерфейса — `Qt Designer`. Оболочка `Qt Creator` также дает воз-

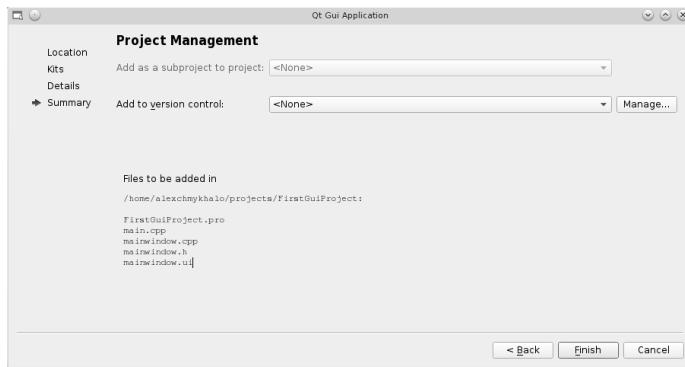


Рис. 11.7: Окно мастера новых файлов и проектов (шаг 5)

можность редактировать файлы формы. Файлы формы главного окна будет сгенерирован автоматически — `mainwindow.ui` (поле `Form file`).

После завершения работы мастера проект откроется в окне оболочки **Qt Creator**. В левой части окна теперь можно исследовать структуру проекта, который состоит из всех файлов, входящих в проект и были сгенерированы мастером. Для компиляции и запуска проекта нажмите кнопку запуска программы или комбинацию клавиш **Ctrl R**. После запуска появится пустое окно — главное окно нашей программы. Вот таким образом выглядит создание простого оконного проекта. Все файлы проекта были сгенерированы мастером, который использует обычно для удобства и чтобы ускорить работу. Конечно, есть и другой путь — можно создать каждый из файлов отдельно и, без каких-либо трудностей, добавить к проекту. Мы подробно рассмотрим как это делать, для того, чтобы исследовать основные составляющие проекта **Qt** и понять как происходит его компиляция.

## 11.5 Задачи для самостоятельного решения

- Повторите описанные шаги для создания собственного проекта. Назовите проект **MyGuiProject**. Класс главного окна программы назовите **MyMainWindow**. Просмотрите структуру проекта. Скомпилируйте и запустите проект на выполнение.
- Попробуйте использовать любую из горячих клавиш, описанных в таблице «Некоторые важные горячие клавиши».
- Используйте документацию **Qt** и найдите описание для классов **QMainWindow** и **QApplication**.

## Глава 12

# Структура проекта. Основные типы

### 12.1 Файлы проекта

Теперь давайте рассмотрим из чего состоит проект Qt. В общем, проект Qt имеет такую структуру:

- файл проекта — описывает файлы, которые входят в проект и содержит необходимые настройки;
- файлы, входящие в проект (или другие подпроекты, если проект разбит на несколько частей).

Ключевую роль имеет файл проекта с расширением `.pro`. Он содержит списки файлов: исходных кодов, файлов ресурсов, файлов локализации, форм, других файлов, которые входят в проект, а также файлов подпроектов, если проект состоит из нескольких частей. Этот файл также содержит некоторые настройки программы.

Теперь рассмотрим создание своего проектного файла. Создадим новую папку, где будет размещаться проект (например: `custom_project`). Создайте файл (это будет файл проекта) введите его имя с расширением `.pro` (например: `custom_project.pro`). Наш файл пока что пустой, но его уже можно открыть в `Qt Creator` (воспользуйтесь главным меню: `File->Open File or Project...`).

Создать пустой проект можно с помощью мастера построения проектов. Для этого надо воспользоваться главным меню `File->New File or Project...` либо комбинацией клавиш `Ctrl+Shift+N`. В окне мастера нужно выбрать раздел `Other Project` (Другой проект) и тип проекта — `Empty Qt Project`.

После того, как мы открыли проект, `Qt Creator` предлагает выбрать комплект для его компиляции. В разделе `Projects` (Проекты) выберем комплект по умолчанию и нажмем `Configure Project`. В дереве проекта выберем и откроем файл проекта. Теперь настало время исследовать синтаксис проектных файлов `Qt`.

Проектный файл обычно содержит несколько настроек в виде специальных переменных, каждая из которых играет свою особую роль. Среди большого количества настроек, которые задают в .pro-файле:

- тип проекта (приложение, динамическая или статическая библиотека, проект, который состоит из подпроектов);
- общие настройки проекта;
- настройки компиляции;
- путь, где будет размещен исполняемый файл, библиотека или бинарный файл во время процесса компиляции;
- пути к файлам, библиотекам и другим частям проекта необходимым для компиляции;
- файлы, входящие в проект;
- дополнительные действия, которые будут выполняться в процессе компиляции проекта.

Откройте проектный файл и добавьте к нему содержимое. Обратите внимание: символ `#` можно использовать для обозначения комментариев.

```
# Указываем тип проекта
TEMPLATE = app # app – Application , прикладная программа
# Используемые модули Qt
QT -= gui # Удаляем из списка модуль gui
# это означает отказ от использования графического интерфейса,
# то есть – консольную программу
CONFIG += console # Конфигурируем создание консольного проекта
# (нужно только для консольных проектов в Windows, в Linux и Mac OS X не выполняет никаких
# действий)
CONFIG -= app_bundle # Предотвращает создание Application bundle в Mac OS X
# (нужно только для консольных проектов в Mac OS X)
TARGET = custom_project # Название исполняемого файла
```

Теперь нам осталось добавить в проект файл с текстом программы. Для этого мы снова можем воспользоваться мастером. В категории **Files and Classes** (Файлы и классы) выберем раздел C++ и выберем тип файла «**C++ Source File**» (Файл исходных текстов C++). Поскольку это будет главный файл программы, то дадим ему привычное для этого случая название: `main.cpp`. Текст программы является обычным:

```
int main(int argc, char *argv[])
{
    return 0;
}
```

После создания `main.cpp`, вновь откроем файл проекта и обратим внимание на несколько дополнительных строк:

```
SOURCES += \
    main.cpp
```

Переменная `SOURCES` хранит список .cpp файлов. В табл. 12.1 мы предоставляем список переменных, которые часто участвуют в описании проекта:

Таблица 12.1: Некоторые важные переменные для описания настроек проекта

Переменная	Описание	Пример
CONFIG	Разнообразные настройки конфигурации проекта (например: режим отладки, вывод предупреждений, компиляция динамической библиотеки и т.п.).	CONFIG += dll plugin \ warn_on release
DEFINES	Макроопределения в проекте. Работает так же, как директива препроцессора <code>#define</code> .	DEFINES += DEBUG_OUTPUT \ CUSTOM_DEFINE
DESTDIR	Путь к папке, где будет создан исполняемый файл.	DESTDIR = ./bin
INCLUDEPATH	Путь к папкам с заголовочными файлами.	INCLUDEPATH += ./includes \ ./my_header_files
FORMS	Файлы форм <b>Qt Designer</b> .	FORMS += mainwindow.ui
HEADERS	Заголовочные файлы программы <code>*.h</code> .	HEADERS += mainwindow.h
LIBS	Пути к динамическим библиотекам и библиотеки, которые используются в программе.	LIBS += -L./libs \ -L./my_libs \ -lmymcustomlib
QT	Модули <b>Qt</b> , которые используются в программе.	QT += core gui widgets \ network sql xml
RESOURCES	Файл ресурсов.	RESOURCES = resources.qrc
SOURCES	Исходные тексты программы <code>*.cpp</code> .	SOURCES += main.cpp \ mainwindow.cpp
TARGET	Название исполняемого файла или динамической библиотеки.	TARGET = MyFirstProject
TEMPLATE	Тип проекта (приложение, библиотека, составленный из подпроектов ...)	TEMPLATE = lib

## 12.2 Компиляция проекта

Компиляция проекта проходит в два этапа. Сначала выполняется предварительная обработка проекта с помощью программы `qmake`. Этот инструмент **Qt** несет ответственность за весь процесс компиляции проекта. Он читает содержание проектного файла и генерирует необходимые промежуточные файлы (дополнительные файлы с исходным кодом и `make`-файлы для компиляции). Это необходимо для того, чтобы превратить все особые расширения **Qt**, которые были использованы в программе, в код на языке **C++** и использовать дополнительные настройки для проекта, описанные в `pro`-файле. После этого проект готов к обработке компилятором. Вторым этапом является непосредственно процесс компиляции. Все эти действия выполняются автоматически в среде **Qt Creator**.

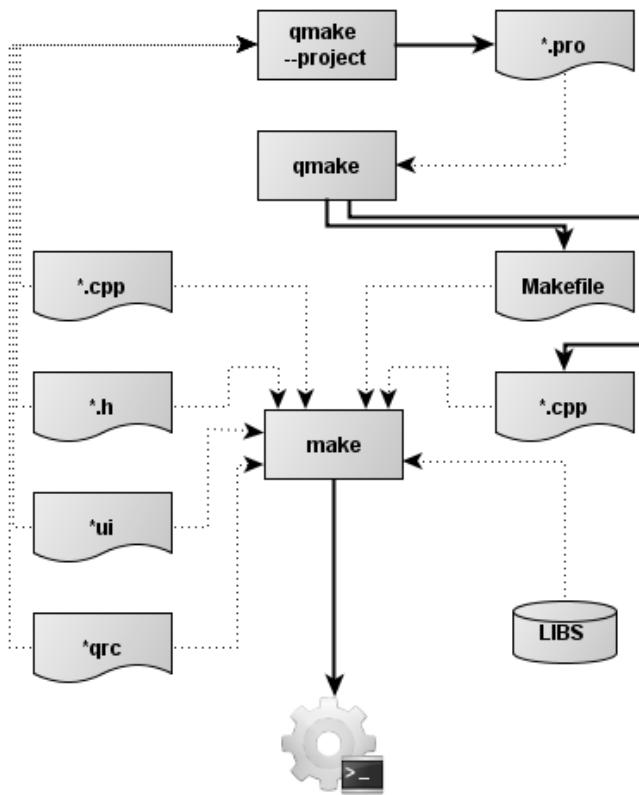


Рис. 12.1: Процесс компиляции проекта

После успешной компиляции мы получаем исполняемый файл программы. Откройте папку проекта. Она будет содержать исполняемый файл и все промежуточные файлы, сгенерированные в процессе.

Таким образом, процесс построения проекта руководит `.pro`-файл. При наличии исходных текстов программы и при отсутствии `.pro`-файла, его можно сгенерировать. Для этого из командной строки необходимо перейти в папку, которая содержит исходные тексты программы и вызвать `qmake` с параметром `--project`. Этим приемом удобно воспользоваться, чтобы сгенерировать файл проекта и использовать оболочку `QtCreator` для работы над программой (даже для обычных программ на C++ без Qt).

Раздел «**Projects**» (Проекты) содержит набор необходимых настроек для процесса компиляции и для настройки среды запуска проекта. Одной из таких

настроек есть опция **Shadow Build**, которая позволяет включить режим при котором для промежуточных файлов, make-файлов и продуктов компиляции создается отдельная папка вне папки с исходным кодом проекта (настройки размещения для нее — в поле **Build directory**). Это позволяет построить и хранить одновременно несколько вариантов построенного проекта для различных инструментариев. Также это сохраняет папку с исходным кодом от засорения файлами, созданных в процессе построения проекта. При выключенном **Shadow build** промежуточные файлы и папка с построенной программе сохраняются в папке, которая содержит файл проекта.

Конечно, созданные промежуточные файлы не являются непосредственной частью проекта. Они были сгенерированы, и будут перезаписываться при необходимости во время компиляции. Поэтому не стоит добавлять их к **pro**-файлу или делать любые изменения в них. Также не стоит их добавлять в систему контроля версий, если ее используют при разработке.

Иногда сгенерированные файлы вместе с объектными и make-файлами бывают необходимо удалить. Это необходимо делать перед тем как заархивировать проект для сохранения, поскольку сгенерированные файлы занимают довольно много места на диске по сравнению с объемом исходного кода. Порой также могут возникать проблемы с компиляцией, когда после значительных изменений в структуре программы промежуточные файлы не были достаточно хорошо заново сгенерированные. В таких случаях возникает необходимость очистить проект. Для этого выберите в главном меню **Build->Clean Project** (Сборка->Очистить проект). Это позволит удалить сгенерированные файлы, кроме скомпилированного исполняемого файла и make-файлов.

Для того, чтобы очистить проект полностью, необходимо изменить некоторые настройки. Откройте раздел **Projects**(Проекты) и в разделе **Clean Steps**(Этапы очистки) нажмите кнопку **Details**(Подробнее) и измените параметр **Make arguments** (Аргументы make) с **clean** на **distclean** (рис. 12.2).

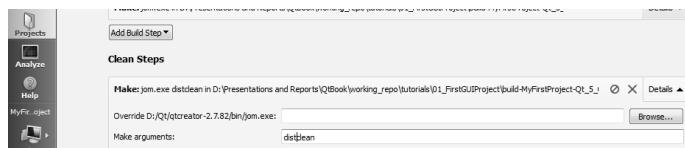


Рис. 12.2: Настройки очистки проекта

Снова очистите проект — все сгенерированные файлы, включительно с исполняемым файлом и make-файлами, будут удалены.

### 12.3 Консольный проект Qt. Вывод сообщений.

Несмотря на то, что **Qt** почти всегда рассматривают как инструментарий для создания программ с графическим интерфейсом, его также можно использовать и в таких программах, которые работают как фоновые процессы, а также в консольных проектах. Для нескольких следующих примеров мы будем использовать последний созданный нами в предыдущем разделе проект.

В таком консольном проекте можно использовать почти все привычные для **Qt** средства и классы. В следующих нескольких примерах мы рассмотрим работу с некоторыми важными типами **Qt** именно на примере консольного проекта. А пока что ограничимся только обзором средства, которое позволяет выводить в консоль сообщения и разнообразную информацию для отладки в процессе работы программы.

Для вывода информации в консольном проекте можно использовать все привычные средства стандартной библиотеки C++. Но в **Qt** для этого есть удобный инструмент — функция **qDebug()**. Рассмотрим пример ее использования:

```
#include <QDebug>
//Собственный тип данных — структура для комплексных чисел
struct complex
{
    double re;
    double im;
};
//Определение потокового оператора для поддержки вывода собственного типа
//complex с помощью qDebug()
QDebug operator<<(QDebug dbg, const complex &c)
{
    dbg.nospace() << "(" << c.re << " + i*" << c.im << ")";
    return dbg.space();
}
int main(int lArgc, char *lArgv[])
{
    //Вывод разнообразных типов данных
    qDebug() << "Hello, " << "this is debug output";
    qDebug() << "Integer values: " << 1 << 10 << 100;
    qDebug() << "Doubles and floats: " << .1 << .123 << 0.112345;
    qDebug() << "Characters: " << 'c' << 't' << '$' << '\n' << "newline";
    qDebug() << "Booleans: " << true << false;
    qDebug() << "Pointers: " << lArgv;
    qDebug() << " and much more... ";
    //Вывод собственного типа данных
    complex c;
    c.re = 0.2;
    c.im = 1.5;
    qDebug() << "including custom types: " << c;
    return 0;
}
```

После выполнения программы в консоли увидим текст:

```
Hello, this is debug output
Integer values: 1 10 100
Doubles and floats: 0.1 0.123 0.112345
Characters: c $ newline
Booleans: true false
Pointers: 0x3278fc8
and much more...
```

```
including custom types: (0.2 + i*1.5)
```

Кроме qDebug() существуют другие функции для вывода сообщений разного уровня. Описание и примеры этих функций рассмотрим в таблице 12.2.

Таблица 12.2: Функции для вывода сообщений

Функция	Описание	Особенности	Пример
qDebug()	Вывод сообщений для отладки, разнообразной информации при работе программы.	Сообщения могут быть выключены с помощью специального макроопределения QT_NO_DEBUG_OUTPUT например, в файле проекта: DEFINES += QT_NO_DEBUG_OUTPUT	<pre>int error_num = 59; std::string error_string(     "unknown error"); qDebug("result: %d,         description: %s",         error_num, error_string         .c_str());</pre> <hr/> <pre>#include &lt;QDebug&gt; ... qDebug() &lt;&lt; "result: " &lt;&lt; error_num &lt;&lt; ", " description: " &lt;&lt; error_string.c_str();</pre>
qWarning()	Вывод сообщений при работе программы.	Сообщения могут быть выключены с помощью специального макроопределения QT_NO_WARNING_OUTPUT например, в файле проекта: DEFINES += QT_NO_WARNING_OUTPUT	<pre>qWarning("warning: %d,         description: %s",         error_num, error_string         .c_str());</pre> <hr/> <pre>#include &lt;QDebug&gt; ... qWarning() &lt;&lt; "warning: " &lt;&lt; error_num &lt;&lt; ", " description: " &lt;&lt; error_string.c_str();</pre>
qCritical()	Вывод сообщений о критических ошибках.		<pre>qCritical("critical error : %d, description: %s",         error_num, error_string         .c_str());</pre> <hr/> <pre>#include &lt;QDebug&gt; ... qCritical() &lt;&lt; "critical error: " &lt;&lt; error_num &lt;&lt; ", description: " &lt;&lt; error_string.c_str();</pre>
qFatal()	Вывод сообщений о фатальных для программы ошибках.	После вывода сообщения происходит аварийное завершение работы программы.	<pre>qFatal("fatal error: %d,         description: %s",         error_num, error_string         .c_str());</pre>

## 12.4 Работа с текстовыми строками в Qt. Класс `QString`. Списки строк `QStringList`.

Обычные строки С довольно просты в использовании, но работать с ними не очень удобно в ряде случаев. Один из них, это поддержка выбора кодировок для текста. Ведь, как известно, существует много разных стандартов кодирования символов текста, которые отличаются поддержкой разного диапазона кодируемых символов.

В Qt для работы со строками есть мощный и специализированный класс — `QString`. Он имеет поддержку Unicode, возможность преобразования текста между разными кодировками и в обычные строки С и `std::string`. А также он имеет хорошее быстродействие и богатый набор инструментов для работы. Поддержка Unicode позволяет работать с текстом на любом языке мира, что очень важно при локализации графического интерфейса программы.

Рассмотрим методы работы с текстовыми строками в Qt. Перед началом работы с текстом в Qt нужно подключить файл описания `QString`:

```
#include <QString>
```

Как и почти для всех классов Qt, название класса совпадает с названием файла описания класса, который необходимо подключить с помощью директивы `#include`.

Существует большое количество разных способов добавления строк и символов к существующей строке:

```
QString lMainStr = "string"; // lMainStr == "string"
lMainStr += ' '; // lMainStr == "string "
(lMainStr += "is") += ' '; // lMainStr == "string is"
QString lHelperStr1("composed");
lMainStr += lHelperStr1; // lMainStr == "string is composed"
QString lHelperStr2 = + ' ', +QString("from") + ' ';
lMainStr.append(lHelperStr2);
// lMainStr == "string is composed from"
lMainStr.push_back("fragments");
// lMainStr == "string is composed from fragments"
lMainStr.prepend("This ");
// lMainStr == "This string is composed from fragments"
lMainStr.insert(lMainStr.length(), ".");
// lMainStr == "This string is composed from fragments."
lMainStr += QString(2, '.');
// lMainStr == "This string is composed from fragments..."
lMainStr = lMainStr.rightJustified(lMainStr.length() + 8, ' ');
// lMainStr == "This string is composed from fragments..."
```

Также есть возможность выделения части строки либо разделения ее на части:

```
QString lQuote = "This is sentence one. This is sentence two.";
// Новая строка с пятью символами
QString lFragment1 = lQuote.left(5); // lFragment1 == "This "
qDebug() << "lFragment1 is: " << lFragment1;
// Первое предложение: Все символы до первой точки
QString lSentence = lQuote.section('.', 0, 0);
qDebug() << "lSentence is: " << lSentence;
// lSentence == "This is sentence one"
// Список слов в строке
QStringList lWordsList = lSentence.split(' ', QString::SkipEmptyParts);
qDebug() << lWordsList;
```

```
// lWordsList == ("This", "is", "sentence", "one", "This",
// "is", "sentence", "two")
```

Для проверки на пустую строку используют метод `isEmpty()`. Его не следует путать с методом `isNull()`, который возвращает значение `true` только для еще не инициализированной строки. Например:

```
QString().isNull();           //true (нулевая строка)
QString().isEmpty();          //true (нулевая строка тоже пустая)
QString("").isNull();         //false (пустая строка не является нулевой)
QString("").isEmpty();        // true
QString("abc").isNull();      // false
QString("abc").isEmpty();     // false
```

`QString` имеет инструменты для преобразования из `std::string` и наоборот. Например:

```
QString lQtStringInitial = "I am a standard STL string.";
std::string lStdString = lQtStringInitial.toStdString();
QString lQtString = QString::fromStdString(lStdString);
```

Также `QString` имеет средства для работы с числовой информацией:

```
//преобразование целого числа в строку
int x = 16;
QString lXStr = QString::number(x);
// x = 7; lXStr = 7
//преобразование строки в целое число
int y = lXStr.toInt();
//преобразование дробного числа в строку
double teta = 12099.10012021210102109991;
QString lTetaStr = QString::number(teta);
// lTetaStr == 12099.1
lTetaStr.setNum(teta);
// lTetaStr == 12099.1
//вывод с 4-мя знаками после запятой
lTetaStr = QString::number(teta, 'f', 4);
// lTetaStr == 12099.1001
//форматирование с использованием символа 'e'
lTetaStr = QString::number(teta, 'e');
// lTetaStr == 1.209910e+04
//Запись числа в строку в разных системах счисления
lXStr = QString(int %1 is %L2 in decimal system, %L3 in binary system, and %
L4 in hexadecimal)
    .arg(x)
    .arg(x, 0, 10)
    .arg(x, 0, 2)
    .arg(x, 0, 16);
```

Для работы со списком строк в Qt предусмотрен специализированный тип `QStringList`. `QStringList` относят к контейнерным классам Qt. Подробнее классы-контейнеры мы рассмотрим в следующем параграфе.

## 12.5 Контейнерные классы в Qt

Список строк `QStringList`, который мы рассматривали, является представителем семейства контейнерных классов Qt, которые являются аналогом контейнеров STL. Но в то же время они имеют свои собственные различия в реализации и возможностях. Мы же будем в дальнейшем использовать в наших примерах исключительно контейнеры Qt.

Простейшим контейнерным классом является `QList`. `QList` — список общего назначения. Для добавления элементов в начало и в конец списка используют методы `prepend()` и `append()`. Также можно добавить несколько элементов за раз используя потоковые операторы. например:

```
#include <QList>
...
QList<QString> lList;
QString lStr1("string1");
QString lStr2("string2");
lList << lStr1 << lStr2; //Добавляем несколько элементов за раз
lList.prepend(lStr1); //Добавляем элемент в начало (повторно)
lList.append("string3"); //Добавляем в конец списка
lList << "string4"; //То же, что и lList.append("string4");
```

Последние две строки возможны за счет неявного преобразования `char*` в `QString`. Класс `QStringList`, которого мы коснулись в примерах предыдущего раздела посвященного текстовым строкам, наследует от `QList<QString>` и реализует ряд дополнительных методов для работы со строками в списке.

Для доступа к элементам используют метод `at()`, который принимает индекс элемента в списке в качестве аргумента. Количество элементов в списке возвращают методы `size()` и `count()`.

```
qDebug() << lList.first();
qDebug() << lList.last();
if(lList.size()>3) qDebug() << lList.at(3);
```

Также стоит отметить два других важных разновидности контейнеров: хэш (`QHash`) и словарь ( `QMap`). Хэш — контейнер, в котором элементы добавляют парами (ключ-значение). Значение в хэше находят по ключу, а для поиска используют хэш-функцию, которая преобразует ключ в значение. В словаре элементы также добавляют парами ключ-значение, но значение сортируют по ключу. Для доступа к значению, используют метод `value()`, который принимает два параметра: ключ и значение по умолчанию, которое метод вернет, если значение не будет найдено. Например:

```
#include <QMap>
...
QMap<QString, QString> lSurnameByName;
lSurnameByName.insert("Bill", "Hunter");
lSurnameByName.insert("Marry", "Lee");
//Поиск значения по ключу
qDebug() << "Bill" << lSurnameByName.value("Bill");
qDebug() << "Marry" << lSurnameByName.value("Marry", "Doe");
//Прибавляем другое значение с уже существующим ключом
lSurnameByName.insert("Marry", "Hunter");
qDebug() << "Marry" << lSurnameByName.value("Marry");
//Ключи не существуют — вывод значений по умолчанию
qDebug() << "James" << lSurnameByName.value("James");
qDebug() << "John" << lSurnameByName.value("John", "Doe");
```

После выполнения получим вывод:

```
Bill "Hunter"
Marry "Lee"
Marry "Hunter"
James ""
John "Doe"
```

Обратите внимание: после того, как мы добавили еще одно значение с тем же ключом (Marry), предыдущее значение было перезаписано новым значением. Для того, чтобы добавить несколько значений с одним и тем же ключом можно воспользоваться методом `insertMulti()`.

```
#include <QHash>
...
QHash<QString, QString> lClassificationHash;
//Добавляем несколько значений с одинаковыми ключами
lClassificationHash.insertMulti("fruits", "apple");
lClassificationHash.insertMulti("fruits", "orange");
lClassificationHash.insertMulti("vegetables", "potato");
lClassificationHash.insertMulti("vegetables", "cabbage");
lClassificationHash.insertMulti("vegetables", "tomato");
qDebug() << lClassificationHash.value("fruits"); //Вывод одного значения за ключом
qDebug() << lClassificationHash.values("fruits"); //Вывод значений с ключом
qDebug() << lClassificationHash.values("vegetables");
```

Получим следующий вывод в консоль:

```
"orange"
("orange", "apple")
("tomato", "cabbage", "potato")
```

Для итерации по списку можно воспользоваться макросом `foreach`. Также можно воспользоваться итератором в стиле Java. Например:

```
QList<int> lList; //Создаем список целых чисел
lList.append(3); //Добавляем элементы
lList.append(6);
lList.append(9);
QListIterator<int> lIt(lList); //Создаем итератор для списка
while (lIt.hasNext()) //Пока следующий элемент существует
{
    qDebug() << lIt.next(); //...вывести следующий элемент
}
```

Другой пример — итерация в обратном направлении. На этот раз используем `xep`.

```
QHash<QString, int> lNumberByName;
lNumberByName.insert("twelve", 12);
lNumberByName.insert("thirty three", 33);
lNumberByName.insert("one hundred and twenty five", 125);
QHashIterator<QString, int> lHashIterator(lNumberByName);
lHashIterator.toBack(); //Перейти к концу контейнера — итератор указывает после
//последнего элемента
while (lHashIterator.hasPrevious())
{
    lHashIterator.previous(); //Переходим к предыдущему элементу
    //Выводим ключ и значение
    qDebug() << lHashIterator.key() << " - " << lHashIterator.value();
}
```

Следующий пример — с итератором в стиле STL.

```
QHash<QString, int>::const_iterator lStlLikeIterator;
for (lStlLikeIterator = lNumberByName.begin();
     lStlLikeIterator != lNumberByName.end();
     lStlLikeIterator++)
{
    qDebug() << lStlLikeIterator.key() << " - "
    //Тоже самое, что и *lStlLikeIterator
    << lStlLikeIterator.value();
}
```

В таблице 12.3 приведены разновидности контейнеров Qt.

Таблица 12.3: Контейнеры Qt

Переменная	Описание особенностей
<code>QList</code>	Список общего назначения для использования в большинстве ситуаций, которые возникают при разработке. Имеет оптимальное быстродействие в большинстве случаев.
<code>QLinkedList</code>	Реализует связный список в Qt. Отсутствует операция доступа по индексу элемента (такая как <code>at(int pos)</code> ).
<code> QVector</code>	Реализует вектор элементов в Qt.
<code>QStack</code>	Реализует стек. Стек размещает элементы по принципу LIFO (Last In, First Out) — элемент, добавленный первым будет последним элементом в стеке.
<code>QQueue</code>	Реализует очередь. Очередь размещает элементы по принципу FIFO (First In, First Out) — элемент, добавленный первым, будет первым элементом в очереди.
<code>QSet</code>	Множество элементов. Гарантирует, что все элементы будут уникальными.
<code> QMap</code>	Контейнерный класс для словаря. Элементы добавляют парами: ключ-значение. Словарь всегда сортирует элементы по ключу. Позволяет найти элемент по ключу.
<code>QMultiMap</code>	Контейнерный класс словаря создан для удобной работы с тем, чтобы каждому ключу соответствовало несколько значений. Метод <code>insert()</code> не заменяет значение ключа, если ключ уже существует, а добавляет новую пару ключ-значение.
<code>QHash</code>	Контейнерный класс для хеша. Элементы добавляют парами: ключ — значение. Элементы хранятся в хеше в произвольном порядке. Позволяет выполнять очень быстрый поиск элемента по ключу.
<code>QMultiHash</code>	Контейнерный класс хеша, оздан для удобной работы с тем, чтобы каждому ключу соответствовало несколько значений. Метод <code>insert()</code> не заменяет значение ключа, если ключ уже существует, а добавляет новую пару ключ-значение.

## 12.6 Работа с файлами

Инструментарий Qt содержит большое количество средств, которые позволяют разработчику абстрагироваться от деталей реализации на той или иной программной платформе. В этом разделе мы рассмотрим средства, Qt предоставляет для работы с файловой системой.

Все устройства ввода/вывода в Qt наследуют от абстрактного класса `QIODevice`. Среди его потомков: буфер для данных (`QBuffer`), процесс — программа которая выполняется в системе (`QProcess`), сетевой сокет (`QAbstractSocket`) и другие. Мы же подробно рассмотрим работу с другим его потомком — классом для работы с файлом ( `QFile`).

Для работы с файлом необходимо создать объект класса `QFile` и задать для него путь к файлу (абсолютный или относительный), с которым вы будете работать. Путь и имя передают как параметр конструктора или с помощью метода `setFileName()`.

Далее файл необходимо открыть и задать режим доступа к нему. Метод `open()` принимает флаги доступа и возвращает `true`, если файл удалось открыть. Доступные флаги доступа:

- `QIODevice::ReadOnly` — открыть для чтения;
- `QIODevice::WriteOnly` — открыть для записи;
- `QIODevice::ReadWrite` — открыть для чтения и записи;
- `QIODevice::Append` — все данные будут добавляться в конец файла (после уже существующих данных);
- `QIODevice::Truncate` — если возможно, стереть содержимое файла перед открытием;
- `QIODevice::Text` — режим работы с текстовым файлом (важно для текстовых файлов для корректной обработки символов завершения строки в Windows и Linux).

Флаги (класс `QFlags`) часто используют в `Qt` для задания комбинации настроек. Для комбинаций нескольких настроек, так же как и бинарной арифметике, используют операцию побитового `OR`.

Для записи и чтения используют методы `read()` и `write()`, которые перегружены в нескольких вариантах. Для чтения одной строки текстового файла используют методы `canReadLine()` и `readLine()`. Для чтения всего содержимого можно воспользоваться методом `readAll()`. Текущую позицию при чтении из файла определяют с помощью метода `pos()`. Установить позицию можно с помощью метода `seek()`. Метод `atEnd()` позволяет определить достигли ли мы конца файла при чтении. После завершения работы с файлом, его нужно закрыть вызовом метода `close()`. Следующий пример демонстрирует чтение текстового файла и вывод его в консоль.

```
#include <QDebug>
#include <QFile>
int main(int argc, char *argv[])
{
    const QString lFileName("file.txt");
    //Проверяем существование файла
    if (!QFile::exists(lFileName))
    {
        qCritical("File %s does not exists.", qPrintable(lFileName));
        return 1;
    }
    QFile lFile;
    //Устанавливаем имя файла
    lFile.setFileName(lFileName);
    //Открываем файл — текстовый, только для чтения
    if (!lFile.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        //Если открыть файл не удалось — выводим сообщение об ошибке
        qCritical("Error %d : %s.", lFile.error(),
                  qPrintable(lFile.errorString()));
        return 2;
    }
```

```

    }
    //Пока можно прочесть строку
    while ( lFile . canReadLine() )
    {
        // ... выводить ее в консоль
        qDebug() << lFile . readLine() ;
    }
    //Заканчиваем работу с файлом
    lFile . close();
    return 0;
}

```

Рассмотрим работу с файлами в **Qt** на другом примере записи и чтения текстовой информации. В этом примере мы используем класс **QTextStream** для получения введенной пользователем информации в стандартный поток ввода. Конструктор **QTextStream** может принимать в качестве параметра указатель на потомок **QIODevice**, указатель на **QString** или **QByteArray**, а также файловую переменную. В примере мы перенаправляем поток ввода в **QTextStream**. Далее мы читаем строку из потока ввода и записываем ее в файл.

```

#include <QDebug>
#include <QIODevice>
#include <QFile>
#include <QTextStream>
int main(int lArgc , char *lArgv [])
{
    QTextStream in(stdin);
    QFile lFile("in.txt");
    if (lFile.open(QIODevice::WriteOnly | QIODevice::Truncate))
    {
        QString lData = in.readLine();
        lFile.write(qPrintable(lData));
        lFile.close();
    }
    else
    {
        qDebug() << "Cannot open file!";
    }
    return 0;
}

```

В следующем фрагменте демонстрируем обратный процесс — чтение строки из файла и вывод прочитанного строки в консоль. Опять же для чтения из файла мы используем **QTextStream**.

```

QFile lFile("in.txt");
if (lFile.open(QIODevice::ReadOnly | QIODevice::Truncate))
{
    QTextStream in(&lFile);
    QString lData = in.readLine();
    qDebug() << lData;
    lFile.close();
}
else
{
    qDebug() << "Cannot open file!";
}

```

## 12.7 Задачи для самостоятельного решения

1. Создайте пустой консольный проект Qt и скомпилируйте его. Определите потоковый оператор для вывода класса Person, который имеет поля Name, Phone number, Address для вывода в консоль с помощью функции qDebug().
2. Создайте программу табуляции функции и записи в текстовый файл с использованием средств, предоставляемых Qt. Используйте классы QFile, QTextStream для записи. Адрес текстового файла жестко задайте в тексте программы.
3. Создайте программу чтения значений протабулированной функции из текстового файла с использованием средств Qt и вывода значений в консоль. Используйте классы QFile, QTextStream для чтения. Адрес текстового файла жестко задайте в тексте программы. Если файл невозможно открыть для чтения — выведите сообщение о критической ошибке с завершением работы программы.

# Глава 13

## Создание графического интерфейса средствами Qt

### 13.1 Виджеты (Widgets)

*Виджеты (Widgets)* — это визуальные элементы, с которых состоит графический интерфейс пользователя.

Примеры виджетов:

- Кнопка (класс QPushButton);
- Метка (класс QLabel);
- Поле ввода (класс QLineEdit);
- Числовое поле-счетчик (класс QSpinBox);
- Стока прокрутки (класс QScrollBar).

В **Qt** есть около 50-ти готовых классов графических элементов доступных для использования. Родительским классом для всех виджетов является класс **QWidget**. От него наследуются все главные свойства визуальных элементов, которые мы тщательно рассмотрим. Исследование способов разработки программ с графическим интерфейсом начнем с примера

Создадим пустой файл проекта. Запустим мастера проектов и выберем в разделе **Projects** (Проекты) пункт **Other Project** (Другой проект). Далее выберем тип проекта **Empty Qt Project** (Пустой проект Qt). К файлу проекта добавим содержимое:

```
TEMPLATE = app
#Модули Qt, которые мы будем использовать
QT += widgets #Добавляем модуль widgets для работы с виджетами (необходимо для Qt5).
TARGET = widget #Название исполняемого файла
SOURCES += \
main.cpp
```

Теперь создадим простую программу с окном, в котором мы будем выводить надпись. Установим размер окна и текст его заголовка, а также установим шрифт для надписи. Для этого создадим файл **main.cpp** со следующим содержанием:

```
#include <QApplication>
#include <QLabel>
int main(int argc, char *argv[])
{
    //Создаем объект QApplication, который инициализирует и настраивает оконную программу,
    //управляет ее выполнением с помощью цикла обработки событий
    QApplication application(argc, argv);
    QLabel lLabel; //Создаем виджет QLabel — метка
    lLabel.setText("I am Widget!"); //Задаем текст для метки
    lLabel.setGeometry(200, 200, 300, 150);
    //Задаем размеры — позицию (x, y) ширину и высоту. Задаем выравнивание текста
    lLabel.setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);
    //Класс QFont используют для настройки параметров шрифта.
    //Выбираем семейство шрифтов Arial Black и размер 12.
    QFont lBlackFont("Arial Black", 12);
    lLabel.setFont(lBlackFont); //Задаем шрифт для метки
    lLabel.show(); //Вызываем метод show() для показа метки на экране.
    return application.exec(); //Запускаем программу на выполнение exec() выполняет
    //цикл обработки событий. Программа ожидает действия пользователя и выполняет их обработку.
}
```

Как видим, элементы, из которых состоят интерфейсы в **Qt**, имеют собственные *позицию* и *размер* — так называемую «*геометрию*» — и, таким образом, занимают соответствующую прямоугольный участок на экране (см. рис. 13.1). Также каждый из элементов имеет настройки, которые определяют его *поведение* и *вид*.



Рис. 13.1: Первый оконный проект

Для создания структуры виджеты организовывают в иерархию по принципу «часть — целое». Каждый из виджетов может содержать другие виджеты. Такой визуальный элемент становится «родителем» (*родительским виджетом*) для элементов, которые он содержит. Отметим, что такие отношения не следует путать с наследованием в C++ — отношениями между классами в программе. Отношения между виджетами являются отношениями между объектами. Такие отношения порождают несколько последствий:

- родительский элемент будет отвечать за удаление дочернего элемента: если родительский виджет удалят — то он автоматически удалит и все дочерние элементы;
- родительский виджет размещает дочерние виджеты внутри себя, части дочерних виджетов, которые выходят за пределы родителя будут невидимыми;

- часть состояния родительского виджета передается дочерним — это касается некоторых свойств (видимость, активность) и стиляй, которые накладываются на визуальный элемент.

Виджеты, которые не имеют родителя (*виджеты верхнего уровня*), имеют вид отдельных окон в программе. Рассмотрим пример. Назовем новый проект **ParentExample**. Файл проекта будет содержать обычные для GUI-проекта настройки:

```
TEMPLATE = app
TARGET = ParentExample
QT += widgets
```

Для виджета, который мы будем использовать в качестве главного окна создадим новый класс. Для этого в категории **Files and Classes** (Файлы и классы) выберем раздел *C++* и выберем *C++ Class* (см. рис. 13.2).

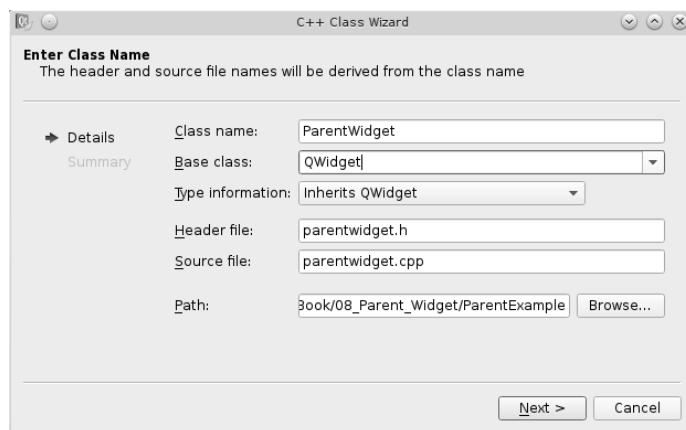


Рис. 13.2: Мастер создания нового класса

Следующим шагом будет создание нескольких элементов на окне. Для этого откроем файл **parentwidget.cpp** и изменим код конструктора класса. Для отображения элементов достаточно создать их в конструкторе класса и задать **ParentWidget** как отца для них. Код **parentwidget.cpp** выглядит так:

```
#include "parentwidget.h"
#include <QLabel>
#include <QPushButton>
#include <QLineEdit>
ParentWidget::ParentWidget(QWidget *parent) :
QWidget(parent)
{
// Создаем метку, указывая родительский виджет — this, то есть экземпляр класса ParentWidget.
QLabel *lLabel=new QLabel(this);
// Позиция относительно левого верхнего угла родительского виджета.
lLabel->setGeometry(50, 0, 100, 30);
lLabel->setText("Text Label"); // Текст на метке.
// Создаем кнопку, задаем «родителя», геометрию и текст
```

```

QPushButton *lPushButton = new QPushButton(this);
lPushButton->setGeometry(50, 50, 100, 30);
lPushButton->setText("PushButton");
//Создаем поле ввода, задаем «родителя», геометрию и текст
QLineEdit *lLineEdit = new QLineEdit(this);
lLineEdit->setGeometry(50, 100, 100, 30);
lLineEdit->setText("LineEdit");
lLineEdit->selectAll(); //Выделяем текст в поле ввода (просто для примера)
//Наконец изменяем размер родительского виджета
setGeometry(x(), y(), 300, 150);
//и задаем текст заголовка окна
setWindowTitle("Parent Widget Example");
}

```

Поскольку родительским элементом является `ParentWidget`, то метка (`QLabel`), кнопка (`QPushButton`) и текстовое поле (`QLineEdit`) находятся в его пределах. Позицию дочерних виджетов задают относительно левого верхнего угла отца. В этом легко убедиться изменив размеры и позицию окна нашей программы. Обратите внимание на то, как мы создавали элементы пользовательского интерфейса в динамической памяти используя оператор `new`. Это гарантирует, что элементы не будут удалены после завершения работы конструктора `ParentWidget`.

Далее добавим в проект файл `main.cpp`. Наш класс наследует от класса `QWidget` — базового класса для всех визуальных элементов пользовательского интерфейса, а следовательно будет обладать всеми его особенностями. Создадим экземпляр нашего класса и вызовем метод `show()` для того, чтобы показать его (см. рис. 13.3). Главный файл программы теперь имеет вид:

```

#include <QApplication>
//Подключаем .h файл с определением нашего класса ParentWidget
#include "parentwidget.h"
int main(int lArgc, char *lArgv[])
{
    QApplication lApplication(lArgc, lArgv);
    //Создаем и показываем окно программы
    ParentWidget lParentWidget;
    lParentWidget.show();
    return lApplication.exec();
}

```



Рис. 13.3: Пример создания дочерних виджетов

## 13.2 Компоновка (Layouts)

Для того, чтобы оптимально разместить виджеты на форме, необходимо учесть ряд деталей:

- размер соседних виджетов;
- визуальные компоненты могут динамически изменять размер, исчезать или появляться в следствии работы логики программы;
- форма, в которой размещают виджеты, может динамически изменять размер при работе программы (когда пользователь меняет размер окна либо раскрывает его на весь экран);
- часто нужно растянуть визуальную компоненту таким образом, чтобы она занимала все пространство формы, или же чтобы несколько компонент занимали пространство формы пропорционально (в соответствующих пропорциях 1:1, 1:2, 3:5 и т.д.);
- часто нужно разместить виджеты либо группы виджетов вертикально либо горизонтально на форме.

Понятно, что такая логика является довольно сложной для реализации при создании пользовательских интерфейсов разной структуры. К счастью в арсенале Qt есть довольно мощный инструмент для упорядочивания виджетов на форме — **компоновщик**.

Компоновщик позволяет упорядочить размещение компонент, оставляя интерфейс гибким для внесения изменений, таких как изменение размера либо количества элементов на форме. Компоновщик также может обеспечивать адекватное изменение размера самого виджета в ответ на перемены в его наполнении.

Компоновщик не принадлежит к виджетам, не наследует от `QWidget` и является невидимым на форме. Он только управляет ее размером и размещением ее содержания.

Обычно используют три основных класса компоновщика:

- вертикальная компоновка (класс `QVBoxLayout`);
- горизонтальная компоновка (класс `QHBoxLayout`);
- компоновка сеткой (класс `QGridLayout`);

Для того, чтобы продемонстрировать работу с компоновками, используем предыдущий пример добавив еще несколько элементов и сделав так, чтобы размещение элементов было пропорциональным и соответствовало изменениям размеров окна. Попытаемся создать такой интерфейс:

- метки и поля ввода разместим горизонтально в одной строке;
- создадим три такие строки с метками и полями;
- добавим еще одну строку с двумя кнопками горизонтально и правым выравниванием.

Создадим проект `LayoutExample`. Для этого воспользуемся мастером новых проектов. Выберем в списке `Qt Widgets Application`. Зададим название для

проекта и настроим класс главного окна: зададим родительский класс (в выпадающем списке **Base Class** установим **QWidget**) и снимем флажок для автоматической генерации файла формы (**Generate Form**) (см. рис. 13.4).

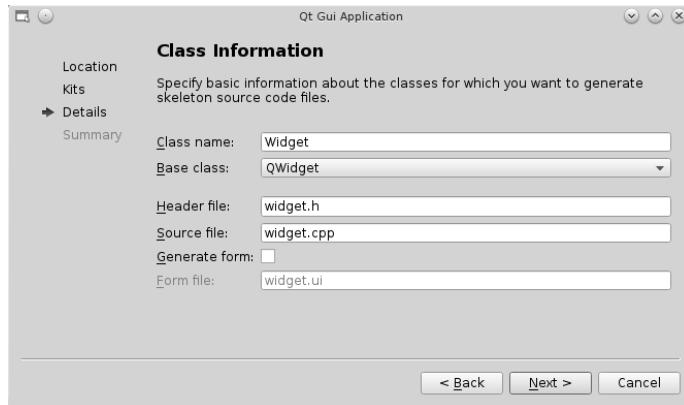


Рис. 13.4: Создание класса окна

Откроем файл **mainwindow.cpp** и изменим код конструктора окна:

```
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
MainWindow :: MainWindow(QWidget *parent) : QWidget(parent)
{
    //Первая горизонтальная строка. Начальный текст в поле ввода
    QLineEdit *lLineEdit = new QLineEdit("Text 1");
    //Задаем текст. & — означает комбинацию клавиш для активации виджета
    QLabel *lLabel = new QLabel("Line Edit &1");
    //Задаем виджет на который будет переключаться фокус ввода при нажатии Alt+1
    lLabel->setBuddy(lLineEdit);
    //Размещаем поле ввода и метку в одной строке
    QHBoxLayout *lHBoxLayout = new QHBoxLayout;
    lHBoxLayout->addWidget(lLabel);
    lHBoxLayout->addWidget(lLineEdit);
    //Вторая горизонтальная строка
    QLineEdit *lLineEdit2 = new QLineEdit("Text 2");
    QLabel *lLabel2 = new QLabel("Line Edit &2");
    lLabel2->setBuddy(lLineEdit2);
    QHBoxLayout *lHBoxLayout2 = new QHBoxLayout;
    lHBoxLayout2->addWidget(lLabel2);
    lHBoxLayout2->addWidget(lLineEdit2);
    //Третий ряд виджетов с кнопками
    QPushButton *IPushButtonOk = new QPushButton("&Ok");
    QPushButton *IPushButtonCancel = new QPushButton("&Cancel");
    QHBoxLayout *lHBoxLayout3 = new QHBoxLayout;
    //Добавим элемент-растяжку он займет все возможное свободное пространство
    //и "прижмет" кнопки к краю
    lHBoxLayout3->addStretch();
    lHBoxLayout3->addWidget(IPushButtonOk);
    lHBoxLayout3->addWidget(IPushButtonCancel);
```

```

//Добавим компоновку вертикально в колонку
QVBoxLayout *IVBoxLayout = new QVBoxLayout;
IVBoxLayout->addLayout(lHBoxLayout);
IVBoxLayout->addLayout(lHBoxLayout2);
IVBoxLayout->addLayout(lHBoxLayout3);
//Задаем компоновщик для окна
setLayout(IVBoxLayout);
}

```

После запуска программы получим главное окно с размещенными в компоновщиках виджетами (см. рис. 13.5).

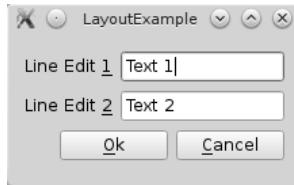


Рис. 13.5: Пример: компоновка виджетов

### 13.2.1 Компоновка сеткой (класс `QGridLayout`)

Рассмотрим работу с компоновщиками на еще одном примере. На этот раз мы создадим простую программу-калькулятор и используем компоновщики `QGridLayout`.

Создадим новый проект так же, как для предыдущего примера. Добавим члены класса — указатели на цифровые кнопки (0-9), кнопку для сброса результата (C) и для суммирования\отображения результата (+ / =). Для создания визуальных элементов в окне — объявим отдельный частный метод `createWidgets()`.

```

#include <QWidget>
//Предварительное объявление классов
class QPushButton;
class QLCDNumber;
class CalculatorMainWindow : public QWidget
{
    Q_OBJECT
public:
    explicit CalculatorMainWindow(QWidget *parent = 0);
    ~CalculatorMainWindow();
private:
    //Отдельный метод для создания интерфейса программы
    void createWidgets();
private:
    //Цифровые кнопки
    QPushButton *pushButton;
    QPushButton *pushButton_2;
    QPushButton *pushButton_3;
    QPushButton *pushButton_4;
    QPushButton *pushButton_5;
    QPushButton *pushButton_6;
    QPushButton *pushButton_7;
    QPushButton *pushButton_8;

```

```

QPushButton *pushButton_9;
QPushButton *pushButton_10;
//Кнопка вывода результата и суммирования
QPushButton *pushButtonPlus;
//Кнопка сброса результата
QPushButton *pushButtonC;
//Виджет — цифровой дисплей
QLCDNumber *lcdNumber;
};


```

В файл реализации включим необходимые .h-файлы:

```

#include <QPushButton>
#include <QGridLayout>
#include <QLCDNumber>

```

Определим метод `createWidgets()`, ответственный за создание интерфейса программы-калькулятора. В нем создадим элемент `QLCDNumber` для отображения следующего слагаемого и результата. Для упрощения примера запрограммируем только операцию сложения целых чисел. Поэтому калькулятор будет содержать только цифровые кнопки (0-9), кнопку добавления/вывода результата и кнопку сброса результата. Все элементы пользовательского интерфейса разместим в одной компоновке `QGridLayout`:

```

void CalculatorMainWindow :: createWidgets()
{
    QGridLayout *lCalcLayout = new QGridLayout;
    setLayout(lCalcLayout);
    lcdNumber = new QLCDNumber;
    pushButton = new QPushButton("1");
    pushButton_2 = new QPushButton("2");
    pushButton_3 = new QPushButton("3");
    pushButton_4 = new QPushButton("4");
    pushButton_5 = new QPushButton("5");
    pushButton_6 = new QPushButton("6");
    pushButton_7 = new QPushButton("7");
    pushButton_8 = new QPushButton("8");
    pushButton_9 = new QPushButton("9");
    pushButton_10 = new QPushButton("0");
    pushButtonC = new QPushButton("C");
    pushButtonPlus = new QPushButton("+/=+");
    lCalcLayout->addWidget(lcdNumber, 0, 0, 1, 4);
    lCalcLayout->addWidget(pushButton, 1, 0);
    lCalcLayout->addWidget(pushButton_2, 1, 1);
    lCalcLayout->addWidget(pushButton_3, 1, 2);
    lCalcLayout->addWidget(pushButton_4, 2, 0);
    lCalcLayout->addWidget(pushButton_5, 2, 1);
    lCalcLayout->addWidget(pushButton_6, 2, 2);
    lCalcLayout->addWidget(pushButton_7, 3, 0);
    lCalcLayout->addWidget(pushButton_8, 3, 1);
    lCalcLayout->addWidget(pushButton_9, 3, 2);
    lCalcLayout->addWidget(pushButton_10, 4, 0, 1, 3);
    lCalcLayout->addWidget(pushButtonC, 1, 3);
    lCalcLayout->addWidget(pushButtonPlus, 2, 3, 3, 1);
}

```

Метод `addWidget()` класса `QGridLayout` принимает в качестве параметров указатель на виджет, а также строку и столбец для размещения виджета. Количество строк и столбцов задают заранее, оно может быть произвольным и определяется в процессе добавления виджетов к компоновщику. Перегруженный вариант метода `addWidget()` используем для добавления кнопок «0», «+/-» и виджета `QLCDNumber` на форму. Он принимает два дополнительных параметра:

ра. Они определяют количество ячеек по горизонтали и по вертикали, которые будет занимать виджет в компоновщике. Обычно каждый элемент занимает одну ячейку в компоновщике. Последний параметр метода `addWidget()` указывает выравнивание элемента внутри ячейки (флажки `Qt::Alignment`).

В заключение добавим к конструктору вызов метода для создания элементов в окне.

```
CalculatorMainWindow::CalculatorMainWindow(QWidget *parent) : QWidget(parent)
{
    resize(300, 300);
    setWindowTitle("Simple Calculator");
    createWidgets();
}
```

### 13.3 Политики размера (Size Policies)

В процессе размещения визуальные элементы и компоновки «договариваются» о размерах и пропорциях в окне. Компоновщики придают размещению структурированный вид: будут ли элементы размещены в ряд (вертикально или горизонтально) либо в сетке. В свою очередь каждый из виджетов предоставляет собственные *политики размера*: какое пространство будет занимать каждый визуальный элемент у ширину и высоту, минимальный и максимальный размер для каждого из них.

Политики размера задают вызовом метода `setSizePolicy()` класса `QWidget`. Метод принимает значение для горизонтальной и вертикальной политики изменения размера. Метод `sizeHint()` возвращает оптимальный размер (класс `QSize`), который был определен для виджета. Ниже приведены возможные значения настройки политик размера:

- `QSizePolicy::Fixed` — `sizeHint` определяет размеры элемента. Размеры элемента фиксированы;
- `QSizePolicy::Minimum` — `sizeHint` определяет минимально возможные размеры;
- `QSizePolicy::Maximum` — `sizeHint` определяет максимально возможные размеры;
- `QSizePolicy::Preferred` — `sizeHint` определяет рекомендованные размеры для элемента;
- `QSizePolicy::Expanding` — так же как `Preferred` с тенденцией к увеличению размера;
- `QSizePolicy::MinimumExpanding` — так же как `Minimum` с тенденцией к увеличению размера;
- `QSizePolicy::Ignored` — `sizeHint` будет игнорирован, элемент должен занимать столько места на форме, сколько возможно.

В предыдущем примере кнопки имеют фиксированный вертикальный размер. Для того, чтобы размеры кнопок менялись как горизонтально так и вертикально, добавим настройки политик размера метода `createWidgets()`.

```
void CalculatorMainWindow :: createWidgets()
{
    ...
    //Зададим политики размера для кнопок: горизонтальное и вертикальное изменение размера
    pushButton->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_2->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_3->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_4->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_5->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_6->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_7->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_8->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_9->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButton_10->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButtonC->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
    pushButtonPlus->setSizePolicy (QSizePolicy :: Preferred , QSizePolicy :: Preferred );
}
```

Также размеры виджета можно жестко ограничивать с помощью методов **setMaximumSize()** (контролирует максимальный размер элемента интерфейса) и **setMinimumSize()** (контролирует минимальный размер). В качестве параметра они принимают объект класса **QSize**, содержащий размеры элемента. Для удобства можно использовать также методы **setMinimumWidth()**, **setMinimumHeight()**, **setMaximumWidth()**, **setMaximumHeight()** для задания минимальной ширины и высоты, а также максимальной ширины и высоты. Зададим фиксированный вертикальный размер для визуального элемента **lcdNumber**:

```
void CalculatorMainWindow :: createWidgets()
{
    ...
    lcdNumber->setFixedHeight (50);
}
```

Результат изменения политик размера показан на рисунке 13.6.

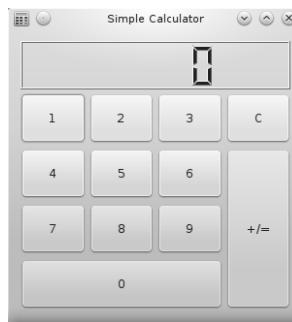


Рис. 13.6: Графический интерфейс программы-калькулятора.

### 13.4 Сигнально-слотовые соединения

Одной из фундаментальных возможностей в Qt является взаимодействие объектов с помощью *сигнально-слотовых соединений*. Для осознания преимуществ, которые предоставляет такая возможность, рассмотрим как устроено взаимодействие между объектами в программе.

Любая объектно — ориентированная программа состоит из объектов, которые взаимодействуют между собой. Каждый из объектов обладает состоянием, которое определяет совокупность данных, которые хранит объект в данный момент. В ответ на взаимодействие с объектом его состояние может измениться. Например, объект реализующий сетевое соединение может получить новые пересланы данные, а объект реализующий кнопку на окне пользовательского интерфейса, может быть нажат пользователем. Таким образом объект изменил свое состояние — и он может сообщить об этом другой объект посыпая ему сообщение об изменении. Практически это взаимодействие обычно реализуют с помощью вызова метода объекта, которому нужно отправить сообщение. Этот метод должен выполнять соответствующие действия в ответ на изменение. Каждый объект выполняет свою роль, а взаимодействие между ними происходит за счет взаимного вызова методов (прямого или косвенного), каждый из которых должен выполнять собственную четкую функцию. Таким разделением получают уменьшение сложности кода — за счет четкого распределения ответственности между объектами — а, следовательно, получают и гибкость, способность классов к повторному использованию, простоту сопровождения кода. Очень важно хорошее понимание этих ключевых идей — разделение ответственности между объектами в программе и организации взаимодействия между ними на основе четко определенных интерфейсов (наборов методов), вытекающих из фундаментальных понятий ООП: абстракции и инкапсуляции.

Таким образом, для реализации такого взаимодействия программисту нужно вызвать метод другого объекта в ответ на изменение состояния. Объект, который сменил состояние, может для этого просто хранить указатель на другой объект, чей метод нужно вызвать. В ответ на изменение состояния он будет обращаться через указатель и вызывать метод. Однако, в таком случае теряется гибкость. Например, в случае с элементами управления в окне программы, придется программировать каждый элемент отдельно — наследовать от класса элемента управления, добавлять указатель и переопределять метод для обработки действий пользователя с целью взаимодействия с другими объектами в программе. К тому же такая связь не может быть динамической и задается жестко на этапе компиляции.

Конечно можно передавать указатель на метод, который будет вызываться в ответ (callback), но такой подход тоже имеет некоторые недостатки (меньшая безопасность, всегда работает как прямой вызов метода, нужны знания о деталях реализации). Именно поэтому в Qt используется концепция сигнально-слотовых соединений.

*Сигнально-слотовые соединения* являются простым, но в то же время важным средством кроссплатформенного инструментарию разработки Qt. Один объект при изменении своего состояния может сообщить других с помощью *сигнала*. Визуальные компоненты тоже вырабатывают сигналы в ответ на действия пользователя (например, нажатия кнопки, установки флага, изменения положения слайдера, редактирования текста в поле ввода и т. п.). Другие объекты могут присоединиться к сигналу *слотом* — специальным методом, который реализует некоторую функциональность и вызывается каждый раз, когда был выпущен присоединенный к нему сигнал,. Как отмечалось ранее, сигнально-слотовые соединения могут использоваться как для взаимодействия объектов в пределах одного потока, так и между потоками (при соответствующих условиях), также возможна организация отложенного выполнения операций.

Для задания соединения используют метод `connect()` класса `QObject`. Метод принимает пять параметров:

- указатель на объект, который посылает сигнал (`sender`);
- название сигнала (`signal`) и его параметры, которые задаются с помощью макрока `SIGNAL()`;
- указатель на объект, который получает сигнал (`receiver`);
- название слота (`slot`) и его параметры, которые задаются с помощью макрока `SLOT()`, или название другого сигнала, будет эмитироваться (выpusкаться) в ответ;
- тип сигнально-слотового соединения (имеет значение по умолчанию `Qt::AutoConnection`).

В следующем примере мы продемонстрируем сигнально-слотовое соединение между кнопкой `QPushButton` и виджетом-окном `QWidget`. В ответ на нажатие кнопки (сигнал `clicked()`), вызывается метод-слот `close()`, который закрывает окно. Заметим, что слоты имеют спецификатор доступа (`private/protected/public`) и вызывают их как и обычные методы класса. Этим они не отличаются от других методов.

```
connect(IPushButton,SIGNAL(clicked()),IWindow, SLOT(close()));
```

*Сигнально-слотовые соединения* могут отличаться за методом вызова слота, либо за механизмом соединения. Тип соединения можно указать при его создании:

- `Qt::AutoConnection` — тип соединения по умолчанию. При соединении объектов в пределах потока ведет себя как `Direct Connection`, иначе — как `Queued Connection`;
- `Qt::DirectConnection` — слот вызывается немедленно после того, как был выпущен сигнал. По сути это напоминает обычный вызов слота как метода;
- `Qt::QueuedConnection` — слот выполняется, как только управления перейдет к очереди обработки сообщений потока получателя;
- `Qt::BlockingQueuedConnection` — то же, что и `Queued Connection`, но поток, из которого был выпущен сигнал блокируется, пока выполнение слота

не будет завершено. Этот тип соединения должен использоваться только когда взаимодействующие объекты находятся в разных потоках.

- `Qt::UniqueConnection` — этот тип соединения такой же как и `Qt::AutoConnection`, но соединение происходит только тогда, когда оно уникально (то есть такое, которое не дублирует уже существующие соединений).

Также с помощью соединений между слотами и сигналами может происходить передача параметров. Например, визуальный элемент `QCheckBox` выпускает сигнал `toggled()` каждый раз при установке и снятии флашка. Сигнал `toggled()` передает один параметр — булевое значение: `true` — если флашок установлен, `false` — если нет. Мы сможем соединить его с со слотом `setChecked()` кнопки, который также принимает булевое значение и устанавливает кнопку в положение «включено» (`true`) или «выключено» (`false`). Заметьте: мы использовали метод `setCheckable()`, который устанавливает для кнопки режим переключения между двумя состояниями.

```
lPushButton->setCheckable(true);
connect (lCheckBox ,SIGNAL(toggled(bool)) ,lPushButton ,SLOT(setChecked(bool))) ;
```

- порядок и тип параметров должен совпадать у объекта который передает сигнал и у объекта-получателя;
- сигнал или слот получателя может опускать некоторые или все остальные параметры, при этом порядок и тип параметров, которые остались у получателя, должны совпадать с первыми параметрами сигнала.

Один и тот же сигнал объекта может быть подключен к нескольким различным сигналам и слотам другого объекта и наоборот. При этом последовательность, в которой будут вызываться присоединенные сигналы и слоты будет соответствовать порядку в котором их соединили. Также надо помнить, что одно и то же соединение может быть выполнено несколько раз подряд. В таком случае при вызове сигнала слот сработает столько же раз, сколько раз был повторно выполнено соединение. Для того, чтобы избежать этого, необходимо передавать тип соединения `Qt::UniqueConnection` каждый раз в случаях, когда создание повторных соединений не требуется.

Возможность создания перекрестного сигнально-слотового соединения продемонстрирована в следующем примере:

```
connect (lCheckBox ,SIGNAL(toggled(bool)) ,lPushButton ,SLOT(setChecked(bool))) ;
connect (lPushButton ,SIGNAL(toggled(bool)) ,lCheckBox ,SLOT(setChecked(bool))) ;
```

Здесь мы видим взаимодействие между обоими элементами управления. При установке/сбросе флашка кнопка будет устанавливаться во включенное состояние или сбрасываться и наоборот.

### 13.5 Создание сигналов (signals) и слотов (slots)

Для того, чтобы объект мог принимать участие в сигнально-слотовом взаимодействии, нужно удовлетворить несколько условий. Класс, экземпляром кото-

рого является объект, должен наследовать от класса `QObject` (в случае множественного наследования `QObject` должен быть первым в списке классов).

Также нужно добавить к описанию класса (перед описанием любых других членов класса в секции `private`) макрос `Q_OBJECT`, который будет обработан метаобъектным компилятором  `moc`  (программа, которая выполняет предварительную обработку текста программы при запуске `qmake` и генерирует дополнительный код для реализации возможностей, которые предоставляет `Qt`). Далее в описании класса можно указать собственно сигналы и слоты. Сигналы описываются в разделе `signals`, а слоты — в разделе `slots`, где перед названием раздела стоит спецификатор доступа (`public`, `private` либо `protected`).

Слоты являются обычными методами класса, которые имеют реализацию и могут принимать параметры и возвращать значения. Спецификатор доступа касается только использования слота как обычного метода но не сигнально-слотовых соединений (при в получении сигнала слот будет вызван независимо от спецификатора). Также значения, которые возвращает слот, игнорируются при сигнально-слотовом соединении. Сигналы, в отличие от слотов, не имеют реализации. Их реализация обеспечивается метаобъектной системой `Qt`. Сигналы являются «зашитенными» (`protected`) методами — их можно посылать из классов, которые наследуют от класса, содержащего сигнал. Сигнально-слотовые соединения могут происходить как между объектами, так и внутри самого объекта, когда тот же объект является одновременно и отправителем сигнала и получателем.

Создание собственных слотов и использования их в программе продемонстрируем на примере нашего предыдущего проекта. Добавим секцию `private slots` и добавим описания слотов для обработки нажатия кнопки калькулятора.

```
private slots:
void slotClear(); //Обработка нажатия кнопки сброса
void slotButtonPressed (int pNum); //Обработка цифровых кнопок
void slotPlusEqual(); //Обработка кнопки суммирования/вывода результата
```

Для сохранения результата и слагаемого, добавим частный член класса:

```
private:
int mSum; //Результат
int mNextNumber; //Следующее слагаемое
```

Для упрощения примера, наш калькулятор будет выполнять только одну операцию — сложение чисел от 0 до 9 с предыдущим результатом и вывод результата при нажатии кнопки вывода результата и суммирования. Реализацию слотов добавим в файл реализации класса `CalculatorMainWindow`.

```
void CalculatorMainWindow :: slotClear ()
{
    lcdNumber->display (0);
    mSum = 0;
    mNextNumber = 0;
}
void CalculatorMainWindow :: slotButtonPressed (int pNum)
{
    mNextNumber = pNum;
    lcdNumber->display (pNum);
}
void CalculatorMainWindow :: slotPlusEqual ()
```

```
{
    mSum += mNextNumber;
    lcdNumber->display(mSum);
    mNextNumber = 0;
}
```

Теперь выполним соединения для кнопок суммирования и сброса.

```
connect(pushButtonC, SIGNAL(clicked()), this, SLOT(slotClear()), Qt::UniqueConnection);
connect(pushButtonPlus, SIGNAL(clicked()), this, SLOT(slotPlusEqual()), Qt::UniqueConnection);
```

Для реализации обработки цифровых кнопок можно использовать следующие подходы:

- отдельные слоты для каждого случая
  - скорее всего будут содержать почти одинаковый код;
  - трудно поддерживать программный код;
  - трудно расширять возможности — требует еще больше дополнительных слотов.
- наследовать кнопку и добавить дополнительный сигнал к классу, который будет передавать значение
- дополнительный специализированный класс.

Однако возможен и другой подход — использование класса `QSignalMapper`. `QSignalMapper` привязывает некоторое значение к каждому сигналу и позволяет избежать чрезмерного дополнительного создания слотов или специализированных классов. По сути, он выполняет роль посредника между объектами, которые посылают сигналы, и слотом, который принимает параметр, изменяющийся в зависимости от объекта, который его выполнил. Добавим описание нового поля класса, содержащий указатель на `QSignalMapper`:

```
//Предварительное объявление классов
class QSignalMapper;
...
private:
    QSignalMapper *mMapper;
```

В конструкторе класса создадим объект и свяжем каждую из кнопок с заданным для нее значением:

```
#include <QSignalMapper>
...
connect(pushButton, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_2, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_3, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_4, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_5, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_6, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_7, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_8, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
```

```
connect(pushButton_9, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
connect(pushButton_10, SIGNAL(clicked()), mMapper, SLOT(map()), Qt::UniqueConnection);
```

После этого соединим **QSignalMapper** сигнально-слотовым соединением с нашим слотом для обработки нажатий цифровых кнопок:

```
connect(mMapper, SIGNAL(mapped(int)), this, SLOT(slotButtonPressed(int)), Qt::UniqueConnection);
```

Теперь программа-калькулятор готова к работе.

Заметим, что для создания соединения обычно используют метод **connect()** но существует способ установить соединение автоматически. Обычно такие соединения можно использовать в случае, когда необходимо соединить элементы на форме созданной как UI-файл с программным кодом, реализующий обработку действий пользователя. Автоматическое соединение задают следующим образом: **on\_<имя объекта>\_<имя сигнала> (параметры)**

Соединение происходит с помощью вызова статического метода **QMetaObject::connectSlotsByName**.

Несмотря на относительную простоту этот метод не всегда является удобным (учитывая некоторую неочевидность, а также способ именования слотов — особенно когда это касается многократного их использования).

Подытожим наши знания о сигналах, слотах и сигнально-слотовых соединениях:

Слоты:

- слот реализуют как обычный метод класса;
- определяют в одной из секций для слотов (**private slots**, **protected slots**, **public slots**);
- слот может возвращать значение, но это нельзя каким-либо образом использовать в сигнально-слотовом соединении;
- произвольное количество сигналов может быть присоединено к одному слоту;
- слот можно вызвать, как обычный метод класса.

Сигналы:

- определяют в секции для сигналов (**signals**);
- сигналы всегда возвращают **void**;
- сигнал должен быть без реализации (реализацию для сигнала предоставляет макрообъектный компилятор **moc**);
- сигнал может быть присоединен к произвольному количеству слотов;
- обычно эмитирование (выпускание) сигнала приводит к прямому вызову слота, но вызов может также быть косвенным (зависит от типа соединения);
- слоты при этом могут вызываться в произвольном порядке;
- для посыпания сигнала достаточно простого вызова (как в случае с методами), но предпочтительно использовать перед вызовом макрос **emit** (используется для различия вызова метода и эмитирования сигнала, но фактически не выполняет никакой специальной роли).

### 13.6 Элементы графического интерфейса и их использование.

Все виджеты в Qt наследуют от класса `QWidget`. Класс `QWidget` предоставляет базовую функциональность общую для всех виджетов. Среди свойств, которые наследуют от `QWidget`, — свойство `enabled`, которое позволяет разрешить или запретить взаимодействие пользователя с элементом управления (методы `void setEnabled(bool)` и `bool isEnabled()`). Когда свойство установлено логическое значение `false` — визуальный элемент деактивирован и пользователь больше не может с ним взаимодействовать. Обычно такие деактивированные элементы изменяют внешний вид, чтобы пользователь смог их отличить от активных. Свойство `visible` (методы `void setVisible(bool)` и `bool isVisible()`) определяет видимый виджет (значение `true`) или нет (значение `false`). Эти свойства влияют не только на сам визуальный элемент, но и на дочерние элементы.

Кроме этих общих свойств, каждый виджет обладает собственными уникальными особенностями, которые позволяют создавать удобные и практические в использовании пользовательские интерфейсы.

Так, например, *кнопки (Buttons)*, очень часто употребляемый элемент управления. В общем поведение для кнопок определяет абстрактный класс `QAbstractButton`. Эти элементы могут находиться в включенном или выключенном состояниях. Состояние можно определять с помощью свойства `checked` (метод `isChecked()`). Переключением можно управлять с помощью свойства `checkable` (`bool isChecked()`, `setChecked(bool)`). От класса `QAbstractButton` наследуют классы `QCheckBox`, `QPushButton`, `QRadioButton`, `QToolButton`.

`QPushButton` чаще используют соединяя его сигнал `clicked()`, который вызывается при нажатии кнопки, с другими слотами.

`QToolButton` является кнопкой для быстрого доступа к действиям или настройкам, которую обычно используют внутри панели инструментов.

`QCheckBox` — элемент-флажок. Может находиться в включенном или выключенном состоянии. Также может иметь третье промежуточное состояние, поддержку которого можно активировать с помощью метода `setTristate()`.

`QRadioButton` — кнопка-переключатель. Так как и флажок может находиться в включенном или выключенном состоянии. Ее используют в группе с другими переключателями (см. класс `QButtonGroup`) для обозначения нескольких взаимоисключающих вариантов выбора, где только один переключатель включен, а все остальные выключены.

К *виджетам-контейнерам (Containers)* относят `QFrame`, `QGroupBox`, `QTabWidget`, `QToolBox`. `QFrame` — наиболее общий элемент. Это базовый класс для виджетов, которые имеют обрамление. От него наследуют такие классы визуальных элементов как `QLabel`, `QLCDNumber`, `QSplitter`, `QToolBox`, `QStackedWidget`, `QAbstractScrollArea`. Может использоваться самостоятельно для отображения различных рамок.

**QGroupBox** используют для выделения группы виджетов рамкой с надписью.

Есть возможность задать клавиатурную комбинацию, чтобы перевести фокус ввода на виджеты в группе. Не создает компоновку для группы виджетов автоматически.

**QTabWidget** — виджет для отображения виджетов внутри отдельных страниц.

Предоставляет панель вкладок и показывает виджет для текущей страницы внутри себя. Для работы необходимо создать виджет и добавить его как страницу, а также задать имя страницы.

**QToolBox** используют для создания вертикальной колонки виджетов со вкладками. Каждый виджет имеет отдельную вкладку. Текущий видимый виджет соответствует текущей открытой вкладке.

Также следует выделить виджеты-виды (*Views*), к которым относят **QListView**, **QListWidget**, **QTableView**, **QTableWidget**, **QTreeView**, **QTreeWidget**. Они добавляют возможность выводить информацию в виде списков, таблиц и деревьев. **QListView**, **QTableView**, **QTreeView** используют модель, как источник данных (см. абстрактным класс **QAbstractItemModel**). **QListWidget**, **QTableWidget** и **QTreeWidget** используют как самостоятельный виджет с данными, данные добавляют поэлементно (см. **QListWidgetItem**, **QTableWidgetItem**, **QTreeWidgetItem**).

К элементам вывода информации (*Display widgets*) относят **QLabel**, **QLCDNumber** и **QProgressBar**. **QProgressBar** позволяет вывести текущий прогресс в виде заполненной линии. **QLCDNumber** выводит целые и числа с плавающей запятой в стиле семисегментного дисплея. **QLabel** используют для вывода различной текстовой информации. Этот виджет также поддерживает разметку HTML4, которую можно использовать для оформления текста.

Наиболее многочисленная группа — элементы ввода (*Input widgets*). К ним относят **QComboBox**, **QDateEdit**, **QDial**, **QDoubleSpinBox**, **QFontComboBox**, **QLineEdit**, **QScrollBar**, **QSlider**, **QSpinBox**, **QTextEdit**, **QTimeEdit**.

**QComboBox** — выпадающий список, используемый для выбора элемента из списка альтернатив.

**QDateTimeEdit** — поле ввода даты и времени. Позволяет вводить и показывать время в заданном формате. Вид этого элемента наследуют также **QTimeEdit** и **QdateEdit**.

**QDial** — изменяет числовое значение, по тому же принципу, что и регуляторы на панели приборов. Наследует от абстрактного класса **QAbstractSlider**.

**QLineEdit** — поле ввода. Дает возможность не только вводить текст, но и проверять допустимость ввода (см. класс **QValidator**). Имеет режим для ввода пароля. Также возможно задать маску для ввода значений.

**QScrollBar** — элемент управления скроллингом. Наследует от абстрактного класса **QAbstractSlider**. Каждому положению указателя соответствует значение, в заданных пределах. Часто используют для прокрутки содержимого других виджетов.

**QSlider** — элемент, который использует перетаскивания мышкой для ввода значений. Каждому положению указателя соответствует значение, в заданных пределах. Наследует от абстрактного класса **QAbstractSlider**.

### 13.7 Задачи для самостоятельного решения

1. Добавьте к примеру калькулятора поддержку нескольких дополнительных арифметических действий (вычитание, умножение, деление).
2. Создайте проект с графическим интерфейсом. Разместите на окне в компоновщиках 5 различных виджетов. Соедините их (по 2–3 между собой) сигнально-слотовыми соединениями, таким образом, чтобы они реагировали на изменения состояния друг друга. (Например, чтобы **QScrollBar** реагировал на перемещение **QSlider**).
3. Создайте поле для игры в крестики-нолики. Для этого разместите кнопки в компоновщиком **QGridLayout** сеткой 3x3. Также разместите в окне надпись **QLabel**, которая будет показывать текст «Player1» или «Player2» в зависимости от того, ходит первый игрок (крестики) или второй игрок (нолики). При нажатии на кнопку она должна менять текст в зависимости от игрока на символ «X» или «O». Используйте для этого класс **QSignalMapper**. Также добавте кнопку «Clear», которая будет очищать поле (устанавливать в качестве текста для всех кнопок пустую строку). После каждого нажатия кнопки «Clear» порядок хода меняется.

# Глава 14

## Собственные классы в Qt. Создание элементов графического интерфейса

### 14.1 Класс QObject

`QObject` является базовым классом для почти всех классов Qt. Исключением являются только классы, которые должны быть достаточно «легкими» (экземпляры которых должны занимать как можно меньше памяти) и классы, объекты которых должны копироваться (`QObject` не поддерживает копирования), а также контейнерные классы. Все виджеты Qt наследуют `QObject` (класс `QWidget` является потомком `QObject`). `QObject` реализует все базовые особенности, которыми обладают классы Qt:

- мощный механизм взаимодействия между объектами с помощью сигнально-слотовых соединений;
- иерархические взаимосвязи между объектами, позволяющие объединять их в объектные деревья;
- управление памятью;
- «умные» указатели, позволяющие отслеживать уничтожение объекта;
- поддержка свойств;
- таймеры;
- обработка событий и фильтры событий;
- метаинформация о типе объекта, его свойства и т. п.

Каждый объект типа `QObject` обладает метаданными, которые хранятся внутри специального метаобъекта. Этот метаобъект создается для каждого потомка `QObject` и сохраняет различную информацию об объекте (так называемые метаданные). Среди доступных для программиста метаданных есть:

- имя класса (метод `const char *Qobject::className()`);
- наследование (метод `bool QObject::inherits(const char *className)`);
- информация о свойствах;

- информация о сигналах и слотах;
- общая информация о классе (`QObject::classInfo`).

Метаданные собираются во время компиляции (предварительной обработки проекта с помощью `qmake`) метаобъектным компилятором `moc`, который анализирует содержание заголовочных файлов программы. Эти метаданные позволяют получить информацию про любого потомка `QObject`. Эта информация может быть полезна как для отладки программы, так и для создания различных механизмов взаимодействия между объектами в программе.

При разработке с использованием Qt часто возникает необходимость наследовать класс `QObject` непосредственно или его потомка. Объекты, которые наследуют `QObject`:

- объекты имеющие имя (`QObject :: objectName ()`), которое используется в Qt для реализации различных возможностей (стили, QML и т.д.);
- объекты, которые могут занимать место в иерархии других объектов `QObject`;
- объекты, которые могут иметь сигнально-слотовые соединения с другими `QObject`.

Рассмотрим создание собственного класса на примере создания виджета, который можно будет многократно использовать в разных программах. Разрабатаем поле ввода с пиктограммой, которая может реагировать на действия пользователя. Такая пиктограмма:

- может быть статически изображена в поле ввода (например, для обозначения обязательных полей в форме ввода данных пользователем);
- обозначать пустое поле или поле с введенными данными (например, для обозначения некорректно введенных данных);
- выполнять заранее заданное действие после нажатия (например, для открытия диалога выбора файла для поля ввода пути к файлу).

Создадим новый проект и добавим к нему новый класс, который будет наследовать от `QLineEdit` (создание новых классов с использованием мастера описано в разделе 13.1). Назовем новый класс `IconizedLineEdit`. После создания нового класса откроем файл описания (`IconizedLineEdit.h`).

Для того, чтобы корректно наследовать класс от `QObject` необходимо выполнить несколько условий:

- во-первых, `QObject` (или потомок `QObject`, от которого наследуют) должен стоять первым в списке классов, от которых наследуют;
- во-вторых, перед объявлением интерфейса класса сразу после фигурной скобки в частной секции размещают макрос `Q_OBJECT`.

Созданный класс выполняет эти условия (наследует от `QLineEdit`, который наследует от `QWidget`, а тот в свою очередь от `QObject` и содержит макрос `Q_OBJECT` в частной секции класса).

```
#ifndef QRCLEARABLELINEEDIT_H
#define QRCLEARABLELINEEDIT_H
#include <QLineEdit>
class QLabel;
class IconizedLineEdit : public QLineEdit
{
    Q_OBJECT
public:
    //Режимы отображения пиктограммы, которые определяют ее поведение
    enum IconVisibilityMode {
        //Всегда отображать пиктограмму
        IconAlwaysVisible =0,
        //Отображать пиктограмму после наведения курсора на поле ввода
        IconVisibleOnHover ,
        //Отображать пиктограмму при присутствии текста
        IconVisibleOnTextPresence ,
        //Отображать пиктограмму при отсутствии текста
        IconVisibleOnEmptyText ,
        //Всегда прятать пиктограмму
        IconAlwaysHidden
    };
    explicit IconizedLineEdit(QWidget *parent = 0);
    bool isIconVisible() const;
    void setIconPixmap(const QPixmap &pPixmap);
    void setIconTooltip(const QString &pToolTip);
private:
    void updateIconPositionAndSize();
private:
    QLabel *mIconLabel; //Указатель на метку, которая отображает пиктограмму
};
#endif // QRCLEARABLELINEEDIT_H
```

В конструкторе класса создадим метку `mIconLabel` с помощью которой мы будем отображать значок. Также добавим реализацию для нескольких вспомогательных методов.

```
#include "iconizedlineedit.h"
#include <QStyle>
#include <QLabel>
//Конструктор класса
IconizedLineEdit::IconizedLineEdit(QWidget *parent) : QLineEdit(parent)
{
    mIconLabel = new QLabel(this); //Создаем метку для того, чтобы показать пиктограмму
}
//Возвращает true, если пиктограмма видима
bool IconizedLineEdit::isIconVisible() const
{
    return mIconLabel->isVisible();
}
//Устанавливает пиктограмму
void IconizedLineEdit::setIconPixmap(const QPixmap &pPixmap)
{
    //Устанавливаем пиктограмму для метки
    mIconLabel->setPixmap(pPixmap);
    //Обновляем позицию и размеры
    updateIconPositionAndSize();
}
//Устанавливаем подсказку для пиктограммы
void IconizedLineEdit::setIconTooltip(const QString &pToolTip)
{
    //Подсказка будет видимой после наведения курсора на метку с пиктограммой
    mIconLabel->setToolTip(pToolTip);
}
void IconizedLineEdit::updateIconPositionAndSize()
```

```

//Обновить размер пиктограммы
mIconLabel->setScaledContents(true);
mIconLabel->setFixedSize(height(), height());
//Обновить размещение пиктограммы
QSize lSize = mIconLabel->size();
mIconLabel->move(rect().right() - lSize.width(), rect().bottom() + 1 -
lSize.height()) / 2);
}

```

Чтобы показать значок, мы используем метку, изображения для которой можно передать с помощью метода `setPixmap()`. Этот метод принимает экземпляр `QPixmap` класса для работы с растровыми изображениями.

Метод `updateIconPositionAndSize()` обновляет размер и размещение для метки. Для того, чтобы растянуть\сжать изображения мы передаем `true` методу метки `setScaledContents()`. Это позволяет игнорировать размеры изображения и изменить размер для значков. Далее мы устанавливаем фиксированный размер для метки, таким образом, чтобы ее пропорции подходили для размещения в поле. Затем размещаем метку в правом конце текстового поля.

Для того, чтобы установить тот или иной режим отображения значка запрограммируем метод `setIconVisibility()`. Добавим объявление метода в файл описания, а также добавим описание перечисления `IconVisibilityMode`, содержащее режимы отображения пиктограммы.

```

public :
    //Режимы отображения пиктограммы, которые определяют ее поведение
enum IconVisibilityMode
{
    //Всегда отображать пиктограмму
    IconAlwaysVisible = 0,
    //Отображать пиктограмму после наведения на поле ввода
    IconVisibleOnHover ,
    //Отображать пиктограмму при присутствии текста
    IconVisibleOnTextPresence ,
    //Отображать пиктограмму при отсутствии текста
    IconVisibleOnEmptyText ,
    //Всегда прятать пиктограмму
    IconAlwaysHidden
};
void setIconVisibility(IconVisibilityMode pIconVisibilityMode);
...
private slots:
    void slotTextChanged(const QString &pText);
private:
    void updateIconPositionAndSize();
    void setIconVisible(bool pIsVisible);
private:
    IconVisibilityMode mIconVisibilityMode; //Режим отображения

```

Для того, чтобы слот `slotTextChanged(QString)` работал, добавим в конструктор сигнально-слотовое соединение. Также добавим начальную инициализацию для поля `mIconVisibilityMode`.

```

//Конструктор класса
IconizedLineEdit :: IconizedLineEdit(QWidget *parent) : QLineEdit(parent),
    mIconVisibilityMode(IconAlwaysVisible) //Инициализация
{
    mIconLabel = new QLabel(this); //Создаем метку для того, чтобы показать пиктограмму
    //Обработка изменения текста в поле
    connect(this, SIGNAL(textChanged(QString)), this, SLOT(slotTextChanged(
        QString)), Qt::UniqueConnection);
}

```

```
}
```

Чтобы использовать наш виджет в программе, добавим файл описания класса и создадим несколько экземпляров, которые разместим на форме. Ответственным за создание интерфейса окна будет отдельный метод `createUi()`.

В файле описания класса главного окна напишем:

```
#include <QWidget>
class IconizedLineEdit;
class MainWindow : public QWidget
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
private:
    void createUi();
private:
    IconizedLineEdit *iconizedLineEdit;
    IconizedLineEdit *iconizedLineEdit_2;
    IconizedLineEdit *iconizedLineEdit_3;
    IconizedLineEdit *iconizedLineEdit_4;
    IconizedLineEdit *iconizedLineEdit_5;
};
```

В файле реализации главного окна разместим код:

```
include "mainwindow.h"
#include "iconizedlineedit.h"
#include <QVBoxLayout>
MainWindow::MainWindow(QWidget *parent) :
QWidget(parent)
{
    createUi();
}
void MainWindow::createUi()
{
    QVBoxLayout *lMainLayout = new QVBoxLayout;
    setLayout(lMainLayout);
    iconizedLineEdit = new IconizedLineEdit;
    iconizedLineEdit->setPlaceholderText("Click to open file");
    iconizedLineEdit->setIconPixmap(QPixmap("Folder.png"));
    iconizedLineEdit->setIconVisibility(IconizedLineEdit::IconAlwaysVisible);
    lMainLayout->addWidget(iconizedLineEdit);
    iconizedLineEdit_2 = new IconizedLineEdit;
    iconizedLineEdit_2->setPlaceholderText("Enter IP address");
    iconizedLineEdit_2->setIconPixmap(QPixmap("Checkmark.png"));
    iconizedLineEdit_2->setIconVisibility(IconizedLineEdit::IconAlwaysVisible);
    lMainLayout->addWidget(iconizedLineEdit_2);
    iconizedLineEdit_3 = new IconizedLineEdit;
    iconizedLineEdit_3->setPlaceholderText("");
    iconizedLineEdit_3->setIconPixmap(QPixmap("Questions.png"));
    iconizedLineEdit_3->setIconVisibility(IconizedLineEdit::
        IconVisibleOnTextPresence);
    lMainLayout->addWidget(iconizedLineEdit_3);
    iconizedLineEdit_4 = new IconizedLineEdit;
    iconizedLineEdit_4->setPlaceholderText("Cannot be empty...");
    iconizedLineEdit_4->setIconPixmap(QPixmap("Warning.png"));
    iconizedLineEdit_4->setIconVisibility(IconizedLineEdit::
        IconVisibleOnEmptyText);
    lMainLayout->addWidget(iconizedLineEdit_4);
    iconizedLineEdit_5 = new IconizedLineEdit;
    iconizedLineEdit_5->setPlaceholderText("Clearable");
    iconizedLineEdit_5->setIconPixmap(QPixmap("X.png"));
    iconizedLineEdit_5->setIconVisibility(IconizedLineEdit::
        IconVisibleOnTextPresence);
```

```
lMainLayout->addWidget(iconizedLineEdit_5);
}
```

Пути к файлам изображений для значков мы передаем в конструктор `QPixmap`. Изображения должны находиться в текущей папке (папке, в которой расположен файл проекта).

После запуска программы мы увидим главное окно и пять полей с пиктограммами. Нашему виджету еще не хватает нескольких важных возможностей. Значки не двигаются при изменении размера текстовых полей. Так же значок не реагирует на нажатие мышкой. Совершенствование этого примера, а также дальнейшее изучение свойств объектов и виджетов мы продолжим в следующих параграфах, посвященных обработке событий.

## 14.2 Управление памятью. Иерархии объектов

Как мы уже знаем, каждый объект может иметь объектов «родительские» и «дочерние» объекты. Таким образом объекты организуют в иерархические структуры напоминающие дерево. Каждый из объектов может содержать дочерние объекты, а также иметь родительский объект. Отношение между объектами можно задать либо воспользовавшись параметром конструктора (при создании), или же с помощью метода `QObject::setParent()`. Обычно пользуются первым способом, то есть задают родительский объект при конструировании.

```
//Задаем родительский виджет с помощью конструктора
QLabel *lLabel = new QLabel(lSomeWidget);
//Задаем родительский виджет с помощью метода QObject::setParent()
QPushbutton *lPushButton = new QPushbutton;
lPushButton->setParent(lSomeWidget);
```

Если виджет содержит компоновщики, то и компоновщики и все размещенные в нем виджеты автоматически получат «родителя» (ведь каждый визуальный элемент наследует от `QWidget`, а тот в свою очередь наследует от `QObject`). Объекты также могут менять «родителя», для этого достаточно вызвать метод `QObject::setParent (QObject *)` с указателем на другой родительский объект. Объектную иерархию можно увидеть наглядно, воспользовавшись методом `dumpObjectTree()` класса `QObject`. Он выводит в стандартный поток вывода тип и имя всех дочерних объектов. Зададим имена для объектов `lLabel` и `lPushButton`:

```
lLabel->setObjectName("ChildLabel");
lPushButton->setObjectName("ChildPushButton");
```

Если теперь вызвать метод `dumpObjectTree()` для объекта `lSomeWidget`, то получим вывод:

```
SomeWidget :: 
    QLabel :: ChildLabel
    QPushbutton :: ChildPushButton
```

Когда родительский объект удаляют, дочерние объекты тоже будут удалены из памяти перед удалением родительского. То же самое происходит и при удалении виджетов — все дочерние виджеты в визуальной иерархии тоже удаляются. Это позволяет управлять высвобождением памяти в программе.

Управление памятью важно, поскольку неуправляемое выделение памяти приводит к *утечке памяти в программе* — ситуации, когда программа не освобождает память. Если такое неуправляемое выделение памяти повторяется периодически, а сама программа выполняется относительно долго, то со временем программа будет занимать все больше места в оперативной памяти. Наконец, таким образом можно исчерпать всю доступную память, что приведет к аварийному завершению программы и исчерпания ресурсов операционной системы.

Например, при использовании ключевого слова `new` память выделяется в динамически распределяемой памяти (так называемой «куче» — `heap`). Динамическая память должна быть освобождена от созданных объектов с помощью ключевого слова `delete` для того, чтобы избежать утечки памяти (`memory leak`). Объекты, созданные в динамически распределяемой памяти могут существовать столько, сколько необходимо. Динамическая память не будет освобождена автоматически, поэтому программист должен самостоятельно следить за высвобождением выделенной памяти. Рассмотрим следующий пример:

```
#include < QApplication >
#include < QWidget >
#include < QLabel >
#include < QPushButton >
int main( int argc , char * argv [] )
{
    QApplication lApplication( argc , argv );
    // (1) Память в динамически-распределяемой памяти (heap)
    QWidget * lSomeWidget = new QWidget( 0 ); // Окно
    // Задаем родительский виджет с помощью конструктора
    QLabel * lLabel = new QLabel( lSomeWidget );
    // Задаем родительский виджет с помощью метода QObject::setParent()
    QPushButton * lPushButton = new QPushButton;
    lPushButton->setParent( lSomeWidget );
    lLabel->setText( "Label" );
    lLabel->move( 10 , 10 );
    lPushButton->setText( "Button" );
    lPushButton->move( 50 , 10 );
    // (1) Память в динамически-распределяемой памяти (heap)
    lSomeWidget->resize( 150 , 50 );
    lSomeWidget->show();
    return lApplication.exec();
```

Хотя с первого взгляда, программа может показаться корректной (она компилируется и выполняется), в ней есть ошибка, связанная с некорректным управлением памятью. Для того, чтобы проанализировать использование памяти в программе, воспользуемся программой `Valgrind`. Это свободно-распространяемая программа для `Linux` с открытым кодом. Среда `Qt Creator` поддерживает работу с этой программой для анализа использования памяти и оптимизации программ на `Qt`.

Перейдите в режим анализа `Analyse` (*Анализ*) воспользовавшись кнопкой на панели переключения режимов работы слева или воспользуйтесь комбинацией клавиш `Ctrl+6`. Снизу под окном редактирования появится дополнительная панель. В выпадающем списке выберите `Valgrind Memory Analyzer` (*Анализатор памяти Valgrind*) и запустите процесс анализа (нажмите на кнопку с пиктограммой треугольника на панели или выберите в главном меню `Analyze->Valgrind Memory Analyzer` (*Анализ->Анализатор памяти Valgrind*)). Начнется анализ работы программы и программа запустится. Затем закройте окно программы. По-

сле завершения работы на панели появится сообщение об утечке памяти (см. рис. 14.1).

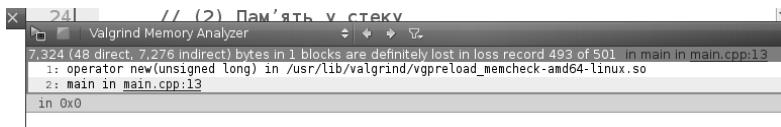


Рис. 14.1: Анализ памяти с помощью Valgrind.

Сообщение указывает на строку:

```
QWidget *lSomeWidget = new QWidget(0); //Окно
```

Это происходит из-за того, что для `lSomeWidget` была выделена динамическая память, но не была корректно высвобождена с помощью оператора `delete`. Для того, чтобы корректно освободить память, модифицируем последние строки программы:

```
int exitCode = QApplication::exec();
//(1) Память в динамически-распределяемой памяти (heap)
delete lSomeWidget;
return exitCode;
```

Скомпилируем программу — после повторного анализа сообщение об утечке памяти исчезнет. Конечно, в этом примере утечка памяти не приводит к негативным последствиям. Выделение памяти происходит только раз и после завершения работы вся использованная оперативная память высвобождается операционной системой. Но в крупных проектах утечки памяти могут стать серьезной проблемой. Инструменты для анализа памяти такие как Valgrind, позволяют локализовать и исправить их.

Можно выделить память для родительского объекта в стеке. Память для объектов, созданных в стеке, освобождается автоматически как только объект выходит за пределы области видимости. Поэтому объекты, созданные в стеке, удаляются как только они выходят из области видимости. Для того, чтобы освободить память автоматически, достаточно только родительский объект создать в стеке. Как только родительский объект удаляется, будут автоматически удалены и все дочерние объекты. Мы можем модифицировать последний пример таким образом, чтобы память для родительского объекта была выделена в стеке. Для этого закомментируйте все части программы обозначенные (1) и измените текст программы:

```
//(2) Память в стеке
SomeWidget lSomeWidget;
//Задаем родительский виджет с помощью конструктора
QLabel *lLabel = new QLabel(&lSomeWidget);
//Задаем родительский виджет с помощью метода QObject::setParent()
QPushButton *lPushButton = new QPushButton;
lPushButton->setParent(&lSomeWidget);
...
//(2) Память в стеке
lSomeWidget.resize(150, 50);
```

```
lSomeWidget.show();
```

При описании собственных классов, необходимо помнить следующие рекомендации:

- задавать для конструкторов класса параметр, который принимает указатель на родительский объект и имеет значение по умолчанию 0;
- создавать дополнительный конструктор, который принимает только параметр с указателем на родительский объект;
- параметр с указателем на «родителя» желательно быть первым параметром среди параметров со значением по умолчанию (если параметров со значением по умолчанию несколько);
- используйте ключевое слово `this` как указатель на родительский объект при создании объектов внутри своего класса. Например:

```
#include <QWidget>
class CustomWidget : public QWidget
{
    Q_OBJECT
public:
    explicit CustomWidget(QWidget *parent = 0);
};

//Конструктор
CustomWidget::CustomWidget(QWidget *parent) : QWidget(parent)
{
    //Задаем родительский виджет — this то есть экземпляр класса CustomWidget
    QPushButton *lPushButton = new QPushButton(this);
    lPushButton->setGeometry(50, 50, 200, 30);
}
```

## 14.3 События (Events). Обработка событий (Event handling)

В Qt существует механизм, который позволяет обрабатывать различные события, происходящие в системе и в самой программе. Визуальные элементы пользовательского интерфейса получают события от устройств ввода информации, которыми управляет пользователь: мышки, клавиатуры и т.п.. Также события могут поступать от объектов изнутри приложения. Таким образом бывают:

- события, возникшие спонтанно и поступающие от оконной системы, такие как нажатие клавиш, движения мышкой, манипуляции с окном программы (spontaneous events);
- события, отправленные изнутри программы, созданные объектами в программе и направленные в цикл обработки событий (posted events) или напрямую к другому объекту (sent events).

Спонтанные события и большинство внутренних событий поступают в цикл обработки событий (event loop). Этот цикл последовательно рассматривает каждое из событий и отправляет на обработку конкретному объекту. Каждый объект в свою очередь имеет возможность обрабатывать событие, которое поступает к нему.

Каждое событие в Qt существует в виде объекта, унаследованного от `QEvent`. Когда событие поступает на обработку, то для объекта-обработчика выполняется виртуальный метод `QObject::event()`. Этот метод обычно вызывает другой метод-обработчик события, который для большинства событий является отдельным. Это необходимо для удобства и гибкости, поскольку таким образом есть возможность запрограммировать обработку конкретного класса событий в отдельном методе. Почти все основные типы событий реализованы в виде отдельного класса. Например, события для движений и нажатий клавиш мышки имеют класс `QMouseEvent`, события для нажатий клавиш клавиатуры имеют класс `QKeyEvent` и т. п. В таблице 14.1 приведены события, которые часто необходимо обрабатывать при разработке программ с графическим интерфейсом.

Таблица 14.1: Часто употребляемые классы событий и их обработчики

Название класса события	Описание	Обработчик события
<code>QMouseEvent</code>	Событие для движений мышкой и нажатия клавиш мышки. Посыпается виджетам. Выполняется только при нажатии клавиши мышки. Для того, чтобы виджет постоянно отслеживал этот тип событий, необходимо передать <code>true</code> в метод виджета <code>setMouseTracking()</code> .	клавиша мышки нажата <code>QWidget::mousePressEvent()</code> клавишу мышки отпустили <code>QWidget::mouseReleaseEvent()</code> клавишу мышки нажали два раза <code>QWidget::mouseDoubleClickEvent()</code> курсор мышки изменил позицию <code>QWidget::mouseMoveEvent()</code>
<code>QKeyEvent</code>	Событие для нажатия клавиш клавиатуры	клавиша нажата <code>QWidget::keyPressEvent()</code> клавишу отпустили <code>QWidget::keyReleaseEvent()</code>
<code>QResizeEvent</code>	Сообщает об изменении размера виджета (изменение уже произошло)	<code>QWidget::resizeEvent()</code>
<code>QPaintEvent</code>	Сообщает о необходимости перерисовки виджета	<code>QWidget::paintEvent()</code>
<code>QTimerEvent</code>	Посыпается через заданные интервалы времени объектом, который запустил один или несколько таймеров с помощью метода <code>QObject::startTimer()</code> . Каждый таймер имеет уникальный идентификатор, который возвращает метод <code>QObject::startTimer()</code> при создании нового таймера. Таймер можно остановить с помощью вызова метода <code>QObject::killTimer()</code> .	<code>QObject::timerEvent()</code>
<code>QCloseEvent</code>	Посыпается окну, которое пользователь пытается закрыть. Позволяет контролировать закрыто окно после обработки события или нет.	<code>QWidget::closeEvent()</code>

Таким образом, для того, чтобы выполнить обработку события, необходимо переопределить нужный обработчик события, выполнить необходимые для обработки действия и, если необходимо, сохранить стандартное поведение для

обработчика — вызвать обработчика события из родительского класса. Теперь вернемся к одному из предыдущих примеров — собственному виджету для поля ввода с пиктограммой. После изменения размера окна (и поля ввода, которое было размещено в компоновке), нам необходимо обновить позицию значка для того, чтобы она всегда находилась в правом конце поля ввода. Для этого мы можем переопределить обработчик события `QResizeEvent` для класса `IconizedLineEdit`:

```
//Событие изменения размера виджета
void IconizedLineEdit :: resizeEvent ( QResizeEvent * pEvent )
{
//Если изменение размера состоялось, обновить позицию и размер пиктограммы
    updateIconPositionAndSize ();
    QWidget :: resizeEvent ( pEvent );
}
```

Для обновления позиции пиктограммы достаточно еще раз вызвать метод `IconizedLineEdit::updateIconPositionAndSize()`. После изменения размера поля ввода позиция пиктограммы будет автоматически обновлена.

## 14.4 Фильтры событий (Event filters). Пропагирование (Propagation)

Когда цикл обработки посыпает событие объекту-адресату, он ожидает, что объект соответствующим образом обработает событие или сообщит о том, что он не будет выполнять обработку. Во втором случае некоторые события могут быть *пропагандируемые* (propagated) к родительскому объекту, то есть пересланы ему на обработку, если дочерний объект проигнорировал событие. Примером событий, для которых работает пропагандирование — это события мышки и нажатий клавиш клавиатуры. Они будут передаваться от родительского объекта к дочернему, пока один из этих объектов не обработает событие или все не проигнорирует его.

Для управления этим процессом используют методы `QEvent::accept()` (для обозначения события как обработанного) и `QEvent::ignore()` (для игнорирования события). Их используют только в виртуальных методах-обработчиках. Обычно `QEvent::accept()` явно не вызывают, поскольку большинство пропагандируемых событий обозначаются как обработанные по умолчанию.

Несмотря на гибкую систему контроля обработки событий, иногда возникает потребность выполнить обработку вместо объекта-адресата. Также иногда возникает необходимость заблокировать (отфильтровать) некоторые события, которые поступают на обработку к объекту. Этого можно достичь с помощью *фильтров событий* (event filters).

Фильтры событий являются обычными объектами, унаследованными от `QObject`. Но перед тем, как событие поступит к адресату, оно поступает к фильтру событий. Он может передать его дальше на обработку адресату, отсеять или выполнить необходимые действия в ответ. Для обработки событий, фильтру необходимо переопределить виртуальный метод `QObject::eventFilter()`. Этот метод возвращает логическое значение, которое означает, будет ли событие

отфильтровано (значение `true`) или поступит на дальнейшую обработку (значение `false`). Фильтр событий для объекта можно задать с помощью метода `QObject::installEventFilter()`, который принимает указатель на `QObject`, выступающий в роли фильтра.

Наш предыдущий пример содержит поле ввода с пиктограммой. Для того чтобы реализовать реакцию на нажатие клавиши мыши на пиктограмму, но не создавать своего класса, унаследованного от класса метки с пиктограммой, мы добавим фильтр событий, который будет посыпать сигнал, если для пиктограммы поступит событие типа `QEvent::MouseButtonDown`. Добавим объявление класса `QEvent`:

Установим фильтр событий для метки: `mIconLabel->installEventFilter(this);`

И добавим фильтр событий для метки в файле `IconizedLineEdit.h`:

```
class IconizedLineEdit : public QLineEdit
{
    Q_OBJECT
public:
    ...
    void setIconClickable(bool pIsIconClickable);
    ...
signals:
    void iconPressed();
protected:
    ...
    bool eventFilter(QObject *pObject, QEvent *pEvent);
private:
    ...
    bool mIsIconCklickable;
};
```

В файле `IconizedLineEdit.cpp`:

```
bool IconizedLineEdit::eventFilter(QObject *pObject, QEvent *pEvent)
{
    if (mIsIconCklickable)
    {
        if ((pObject==mIconLabel) && (pEvent->type()== QEvent::MouseButtonDown))
        {
            emit iconPressed();
            return true;
        }
    }
    return false;
}
//Установить режим реакции на нажатие мышкой на пиктограмму
void IconizedLineEdit::setIconClickable(bool pIsIconClickable)
{
    mIsIconCklickable = pIsIconClickable;
    //Устанавливаем вид курсора при наведении на метку с пиктограммой
    if (mIsIconCklickable)
    {
        mIconLabel->setCursor(Qt::PointingHandCursor);
    }
    else
    {
        mIconLabel->setCursor(Qt::ArrowCursor);
    }
}
```

Наш класс для поля ввода с пиктограммой одновременно выступает фильтром событий для метки.

Добавим реакцию на нажатие к двум полям в `mainwindow.cpp`. Первое поле будет открывать диалог выбора файла и записывать путь к файлу. Последнее поле будет стирать свое содержимое в ответ на нажатие пиктограммы:

```
#include <QFileDialog>
...
void MainWindow::createUi()
{
    ...
iconizedLineEdit->setIconClickable(true);
connect(iconizedLineEdit, SIGNAL(iconPressed()), this, SLOT(slotChooseFile()), Qt::UniqueConnection);
    ...
iconizedLineEdit_5->setIconClickable(true);
connect(iconizedLineEdit_5, SIGNAL(iconPressed()), iconizedLineEdit_5, SLOT(clear()), Qt::UniqueConnection);
    ...
}
void MainWindow::slotChooseFile()
{
    QString lFileName = QFileDialog::getOpenFileName(this, "Open file");
    iconizedLineEdit->setText(lFileName);
}
```

Объявления слота `slotChooseFile()` добавим к описанию класса главного окна:

```
class MainWindow : public QWidget
{
    Q_OBJECT
    ...
private slots:
    void slotChooseFile();
    ...
};
```

После этого наш пример окончательно готов (см. рис. 14.2).

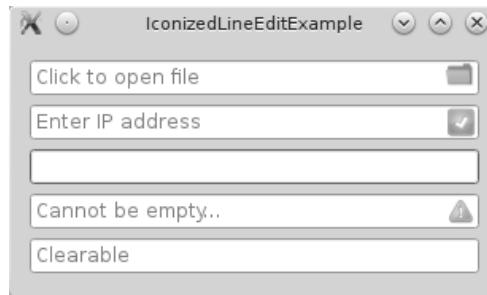


Рис. 14.2: Пример: поле ввода с пиктограммой.

Просуммируем наши знания об обработке событий в Qt:

- события дают возможность проводить обработку различных ситуаций, возникающих во время работы программы;

- есть два источника поступления событий. Спонтанные события поступают от оконной системы в ответ на действия пользователя (движения мышкой, нажатия клавиш клавиатуры, действия с окном программы и т. д.). События также возникают внутри самой программы и поступают от объектов, которые участвуют в работе программы;
- события направляются на обработку объектам, участвующим в работе программы. За сбор и перенаправление событий к обработчикам ответственным является цикл обработки событий в программе;
- каждое событие существует в виде объекта типа `QEvent`. Для большинства событий существуют специализированные классы этих событий, унаследованные от `QEvent`, которые содержат информацию о событии и данные необходимые для ее корректной обработки;
- для участия в обработке событий объект должен быть унаследован от класса `QObject`. События, которые поступают к объекту на обработку, автоматически передаются к виртуальному методу `QObject::event()`;
- для удобства, для большинства событий существуют отдельные виртуальные методы обработки, которые вызываются виртуальным методом `QObject::event()`;
- объекты имеют возможность быть фильтрами событий для других объектов в приложении. Для того, чтобы установить объект в качестве фильтра событий, пользуются методом `installEventFilter()`. Объект-фильтр должен переопределить виртуальный метод `QObject::eventFilter()`, который возвращает логическое значение, было событие отфильтровано (то есть, оно не будет направляться на обработку) или нет.

А вот общий алгоритм для программной обработки событий:

- в случае, когда нет возможности унаследовать класс объекта и переопределить методы обработки, а также в случае, когда необходимо фильтровать, блокировать или предварительно обработать событие перед тем, как она будет отправлена на обработку к объекту — воспользуйтесь фильтром событий
  - переопределите виртуальный метод `QObject::eventFilter()` для объекта, который будет выступать в роли фильтра событий;
  - в методе `eventFilter()` проверьте объект к которому на обработку будет направляться событие;
  - дальше, определите тип события (`QEvent::Type`) и класс события, проверьте является ли полученное событие событием нужного вам типа;
  - приведите объект события до нужного вам класса событий;
  - выполните предварительную обработку события, если необходимо. Для того, чтобы отфильтровать событие (после этого оно не будет передаваться объекту на обработку) верните из метода логическое значение `true`. Для того, чтобы отправить событие на обработку объекту-адресату верните логическое значение `false`.

- для некоторых типов событий, которые не имеют отдельного метода-обработчика, обработку можно запрограммировать, переопределив метод `QObject::event()`
  - переопределите виртуальный метод для объекта `QObject::event()`, который должен выполнять обработку;
  - определите тип события (`QEvent::Type`) и класс события, проверьте является ли полученное событие событием нужного вам типа;
  - приведите объект события до нужного вам класса событий;
  - выполните обработку или вызовите метод, который ее выполнит, верните значение `true`, которое означает, что событие распознано и обработано, или `false` в случае, если объект не будет выполнять обработку. В этом случае событие может быть переслано на обработку родительскому объекту.
  - вызовите метод `event()` для базового класса, чтобы провести обработку всех остальных видов событий. Верните из метода значение, которое вернет метод `event()` базового класса после выполнения.
- если объект имеет виртуальный метод-обработчик события, используйте его
  - переопределите виртуальный метод для объекта `QObject::event()`, который должен выполнять обработку;
  - выполните обработку события. Вызовите метод `accept()` для события, если во время обработки убедились, что событие должно быть проработано данным объектом. В противном случае, вызовите `ignore()` для объекта события (в этом случае событие может быть направлено к родительскому объекту на обработку).

## 14.5 Создание собственного элемента интерфейса. Создание свойств

`QObject` предоставляет механизм для объявления свойств. *Свойства* — это специальные общедоступные поля, через которые можно получить доступ к данным объекта. Свойства обычно имеют методы для установки и получения значения, вызываемые при чтении из свойства и записи данных. В `Qt` свойства выполняют специальную роль, ведь делают доступными для метаобъектной системы способы чтения и записи данных объектов, что часто используется в `Qt` для реализации различных механизмов (анимации, описание динамических пользовательских интерфейсов в `QtQuick`, и т. д.).

Определение свойства обычно состоит из макроса `Q_PROPERTY`, который описывает свойство и содержит описания:

- метода для установки начального значения;
- метода для получения значения;
- метода для установки значения;
- сигнала об изменении значения свойства;
- дополнительных настроек.

Метод для установки значения принимает одно значение (того же типа, что и свойство) и возвращает `void`. Метод для получения значения возвращает значение свойства.

Обычно метод для получения значения называют так же как и свойство, а метод для установки значения еще и имеет префикс «`set`». Метод для получения значения, который возвращает тип `bool` также обычно имеет префикс «`is`». например:

```
QString text() const;
void setText(const QString &text);
bool isVisible() const;
void setVisible(bool isVisible);
```

Наличие отдельного метода для установки значения позволяет запрограммировать дополнительные действия, а также проверить допустимость значения, которое устанавливается. В свою очередь, отдельные методы для установки значения дают возможность косвенного использования уже существующих свойств. Изменение значения свойства возможно, как через прямое использование методов установки и получения значения свойства, так и используя средства мета-объектной системы Qt. Также есть возможность определения доступных свойств во время выполнения программы.

Мы рассмотрим определение свойств на примере создания собственного виджета-индикатора `LedIndicator`, который будет находиться в одном, включенном или выключенном, состоянии в зависимости от значения свойства.

Для начала, создадим оконный проект и добавим к нему класс `LedIndicator`, унаследованный от `QWidget`:

```
class LedIndicator : public QWidget
{
    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE setText)
    Q_PROPERTY(bool turnedOn READ isTurnedOn WRITE setTurnedOn NOTIFY
               stateToggled)
public:
    explicit LedIndicator(QWidget *parent = 0);
    QString text() const;
    bool isTurnedOn() const;
signals:
    void stateToggled(bool);
public slots:
    void setText(const QString &);
    void setTurnedOn(bool);
private:
    QString mText;
    bool mIsTurnedOn;
};
```

Мы определили два свойства: `text` типа `QString` для надписи, а также `turnedOn` типа `bool` для состояния индикатора. Специальные слова `READ` и `WRITE` в описании свойства обозначают названия методов для установления и изменения значения свойства. В описании второго свойства использовано слово `NOTIFY` для обозначения названия сигнала, который будет сообщать об изменении свойства. Добавим реализацию для класса:

```
#include <QPainter>
#include <QPaintEvent>
```

```
//Радиус индикатора
const int cLedRadius = 7;
//Отступ между индикатором и надписью
const int cLedSpacing = 5;
LedIndicator :: LedIndicator (QWidget *parent) : QWidget (parent),
    mIsTurnedOn (false) //Инициализируем начальным значением!
{
}
//Метод получения значения — свойство text
QString LedIndicator :: text () const
{
    return mText;
}
//Метод получения значения — свойство turnedOn
bool LedIndicator :: isTurnedOn () const
{
    return mIsTurnedOn;
}
//Метод установки значения — свойство text
void LedIndicator :: setText (const QString &pText)
{
    mText = pText;
}
//Метод установки значения — свойство turnedOn
void LedIndicator :: setTurnedOn (bool pIsTurnedOn)
{
    //Проверка уже установленного значения
    if (isTurnedOn () == pIsTurnedOn)
    {
        return;
    }
    mIsTurnedOn = pIsTurnedOn;
    //Выпускаем сигнал про изменение
    emit stateToggled (mIsTurnedOn);
    //Вызываем метод QWidget :: update(), который добавляет в очередь событий QPainterEvent
    //для того чтобы перерисовать наш виджет в соответствии с установленным isTurnedOn()
    update ();
}
```

Обратите внимание: в большинстве случаев при создании собственных слотов, которые получают и устанавливают значения, следует сравнивать переданное значение с текущим и запрещать выполнение тела слота, если переданное и текущее значения равны. Особенно за этим необходимо следить, если слот выдает сигнал об изменении значения. Об этом необходимо помнить, чтобы избежать бесконечной рекурсии и аварийного завершения программы в случае перекрестного сигнально-слотового соединения между такими слотами. Именно такую проверку мы выполнили в методе `setTurnedOn()` перед отправкой сигнала об изменении и перерисовкой.

Теперь осталось создать для нашего индикатора соответствующий вид. Как мы отметили, существует два способа создать собственный виджет: составить из готовых виджетов, упорядочив их размещение и геометрию, или нарисовать виджет средствами Qt. В следующем параграфе мы познакомимся со средствами для прорисовки виджетов на экране.

## 14.6 Рисование элементов. Класс QPainter

Каждый виджет занимает прямоугольную область на экране, в соответствии с позицией и размерами. В пределах этой области виджет прорисовывает себя когда становится видимым или когда часть этой области была перекрыта или изменилась геометрия виджета. Для рисования визуальных элементов пользуются примитивами, такими как линии, окружности, прямоугольники, градиенты и т. п. Из примитивов складываются сложные элементы, такие как разнообразные рамки, поля, панели и т. д. Для рисования примитивов в Qt пользуются специальным классом `QPainter`.

`QPainter` обладает богатым набором методов для рисования различных геометрических примитивов, надписей и частей изображений. Для рисования он использует объекты класса `QPaintDevice`, реализующих область для вывода графической информации (на экран, область памяти, на принтер и т. д.). Класс `QWidget` как раз наследует от классов `QObject` и `QPaintDevice`, что позволяет использовать `QPainter` для рисования интерфейса.

Оконная система и Qt следят за изменениями размеров, позиции, видимости окон и отдельных виджетов в программе и направляют специальные события, которые сообщают каждый виджет о необходимости обновления вида. В виртуальном обработчике `paintEvent()` виджеты имеют возможность использовать `QPainter` для собственного отражения. В нашем примере мы определим обработчик события `QPaintEvent`.

```
#include <QWidget>
class LedIndicator : public QWidget
{
    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE setText)
    Q_PROPERTY(bool turnedOn READ isTurnedOn WRITE setTurnedOn NOTIFY
               stateToggled)
public:
    ...
    QSize minimumSizeHint() const;
    ...
protected:
    void paintEvent(QPaintEvent * );
    ...
};
```

И добавим реализацию для него:

```
void LedIndicator::paintEvent(QPaintEvent *pEvent)
{
    // Создаем объект QPainter и указываем QPaintDevice текущий виджет
    QPainter IPainter(this);
    // Используем сглаживание при рисовании для лучшего вида
    IPainter.setRenderHint(QPainter::Antialiasing);
    // Центр окружности индикатора QPoint — класс для описания точки
    QPoint ILedCenter(cLedRadius + 1, height() / 2);
    // Фигура, которую мы будем рисовать QPainterPath — класс для описания фигуры
    // состоящей из нескольких примитивов
    QPainterPath IPath;
    // Добавляем окружность
    IPath.addEllipse(ILedCenter, cLedRadius, cLedRadius);
    IPainter.save(); // Сохраняем настройки после всех изменений мы восстановим их для
    // рисования подписи
    // Создаем радиальный (окружностями) градиент указываем центр для градиента и радиус
```

```

QRadialGradient lGradient(lLedCenter, cLedRadius);
if (mIsTurnedOn) //Устанавливаем цвет границы и градиент
{
    //для включенного и выключенного состояний
//Задаем объект QPen — настройки рисования контуров
//Используем константу для задания цвета контура в конструкторе QPen
    lPainter.setPen(QPen(Qt::darkGreen));
//Задаем цвет в разных точках (0 — центр, 1 — край) цвет будет равномерно изменяться
//Для задания цвета пользуемся текстовым шестнадцатеричным RGB обозначением
//— неявное преобразование в QColor
    lGradient.setColorAt(0.2, "#70FF70");
    lGradient.setColorAt(1, "#00CC00");
}
else
{
//Здесь задаем черный цвет. Конструктор QColor, значение красной,
//зеленой и синей (0-255) компонент цвета
    lPainter.setPen(QPen(QColor(0, 0, 0)));
    lGradient.setColorAt(0.2, Qt::gray);
    lGradient.setColorAt(1, Qt::darkGray);
}
//Заполняем фигуру индикатора градиентом
lPainter.fillPath(lPath, QBrush(lGradient));
//Рисуем границу индикатора
lPainter.drawPath(lPath);
//Восстанавливаем настройки перед последним сохранением
lPainter.restore();
//Устанавливаем шрифт для рисования текста используем QWidget::font(),
//чтобы иметь возможность стилизовать надпись
lPainter.setFont(font());
//Квадрат, в котором будет рисоваться текст. QRect — класс для обозначения прямоугольной области
QRect lTextRect(cLedRadius*2+cLedSpacing, 0, width() - (cLedRadius*2 +
    cLedSpacing), height());
//Рисуем текст в заданном прямоугольнике, выравнивание по левому краю и вертикально по
//центру
lPainter.drawText(lTextRect, Qt::AlignVCenter | Qt::AlignLeft, mText);
}
//Переопределяем виртуальный метод minimumSizeHint() для передачи корректных минимальных
//размеров
QSize LedIndicator::minimumSizeHint() const
{
    return QSize(cLedRadius * 2 //Диаметр индикатора
    + fontMetrics().width(mText) //Ширина текста mText
    + cLedSpacing, //Отступ
    cLedRadius * 2);
}

```

Теперь создадим наш виджет и добавим его в главное окно программы вместе с флагком. Соединим флагок и индикатор сигнально-слотовым соединением, таким образом, что состояние флагка будет ???устанавливаться индикатору.

```

#include <QHBoxLayout>
#include <QCheckBox>
#include "ledindicator.h"
MainWindow::MainWindow(QWidget *parent) : QWidget(parent)
{
//Главное компонование
QHBoxLayout *lLayout = new QHBoxLayout;
setLayout(lLayout);
//Создаем наш индикатор и добавляем его к компоновщику
LedIndicator *lLedIndicator = new LedIndicator;
lLedIndicator->setText("LED Indicator");
lLayout->addWidget(lLedIndicator);
//Создаем и добавляем флагок
QCheckBox *lCheckBox = new QCheckBox("Led ON");
lLayout->addWidget(lCheckBox);

```

```
//Соединяем флагок и индикатор
connect(lCheckBox, SIGNAL(toggled(bool)),
lLedIndicator, SLOT(setTurnedOn(bool)), Qt::UniqueConnection);
}
```

Окончательно, окно нашей программы имеет вид (см. рис. 14.3).

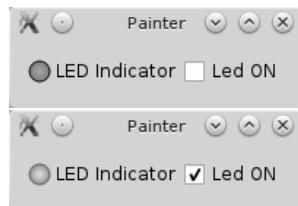


Рис. 14.3: Пример: виджет-индикатор.

## 14.7 Задачи для самостоятельного решения

1. Создайте окно для ввода имени пользователя и пароля. Используйте для этого класс `IconizedLineEdit`.
2. Добавьте возможность проверки правильности введенных данных с помощью класса `QValidator`.
3. Создайте программу, которая открывает изображение в окне. Для ввода пути к изображению добавьте к окну поле ввода. Для поля ввода пути к изображению используйте класс `IconizedLineEdit`. При нажатии на пиктограмму должен открываться диалог выбора файла.
4. Добавьте к классу `LedIndicator` поддержку третьего промежуточного состояния. Для включения поддержки этого состояния, добавьте метод `setTristate()`. Для установки и чтения состояния индикатора добавьте методы `setState`.

## Глава 15

# Разработка приложений с графическим интерфейсом

### 15.1 Окна. Класс QMainWindow

Как уже отмечалось ранее, виджеты, родительский виджет для которых не задан, становятся окнами. Обычно для окон приложения используют следующие классы:

- `QMainWindow` — окно приложения, которое может содержать меню, панели, строку статуса;
- `QDialog` — диалоговое окно;
- `QWidget` — простое, обычно немодальное окно;

Окно обычно имеет обрамление и заголовок. Текст для заголовка окна устанавливают с помощью метода `QWidget::setWindowTitle()`. Конструктор класса `QWidget` принимает дополнительный параметр, для типа окна — `Qt::WindowFlags`. С помощью этого параметра можно управлять типом обрамления, типом окна (для оконной системы). Например, можно создать окно без обрамления (это полезно в некоторых случаях для оформления, например, для окна загрузки программы) или деактивировать кнопки для минимизации и максимизации окна.

*Окно диалога* — это особый вид окна, который может использоваться для различных целей, но всегда предоставляет пользователю возможность взаимодействия с программой. Диалоги, как правило, не имеют кнопок для минимизации и максимизации окна. Окна диалога также часто бывают *модальными*. *Модальность* окна определяется его поведением. Модальные окна блокируют доступ к другим окнам, пока пользователь не завершит работу с окном (не закроет его). Задать модальность окна можно с помощью метода `QWidget::setWindowModality()`, если передать ему логическое значение `true`.

`QMainWindow` — класс, реализующий функциональность главного окна приложения. Для этого он дополнительно имеет специальные средства работы:

- Главное меню (Main menu);
- Панели инструментов (Toolbars);
- Панель статуса (Status bar);
- Присоединяемые панели (Docks);

Несколько элементов пользовательского интерфейса могут выполнять одно и то же *действие* (например: меню, кнопка на панели инструментов и т.д.). Класс `QAction` используют для того, чтобы привязать заданное действие к нескольким элементам управления. Благодаря группировке действия и связанных с ней данных (названия, подсказки, пиктограммы и т.д.), а также ее многократного использования (в главном меню, на панели инструментов и т.д.), можно избежать дублирования кода.

*Присоединяемые* панели `QDockWidget` (dock widgets), монтируются в крае окна, и могут быть перенесены и перегруппированы пользователем, или даже разделены и размещены как отдельные окна. Обычно содержат группу элементов пользовательского интерфейса, объединенных общей целью и назначением или группу инструментов для работы с текущим открытым файлом.

*Панель статуса* `QStatusBar` (Status bar) обычно используют для изображения текстовых сообщений о статусе или текущие действия программы, но она может содержать пиктограммы, а также другие виджеты (например, индикаторы прогресса, метки).

Таким образом, воспользовавшись возможностями главного окна и создав несколько диалогов, можно получить программу, которая будет соответствовать стандартам современных пользовательских интерфейсов. Сих пор для создания интерфейса программы нам приходилось самостоятельно создавать и компоновать виджеты. В следующем параграфе мы используем для этой цели программу `Qt Designer`, которая позволяет создать интерфейс средствами визуального проектирования.

## 15.2 Быстрая разработка с помощью `Qt Designer`

Как мы отмечали ранее, есть два подхода, которые можно использовать при построении графического пользовательского интерфейса, используя виджеты `Qt`:

- создать, настроить виджеты и разместить их на форме в соответствующих компоновках с помощью программного кода;
- воспользоваться визуальным редактором форм `Qt Designer`, который создаст файл формы (он будет описывать ее внешний вид, размещение, размеры, настройки, компонование и т.д.). В дальнейшем из файла формы на этапе компиляции будет создан файл с кодом программы, будет программно создавать этот интерфейс и предоставлять программисту доступ к элементам на форме.

*Файлы формы* имеют расширение `.ui`. `Qt Designer` позволяет редактировать файлы форм, содержащих настройки вида виджетов. `Qt Designer` можно ис-

пользовать как отдельную программу или воспользоваться интеграцией с оболочкой **Qt Creator** — редактором форм.

Визуальный редактор форм позволяет воспользоваться фактически всеми стандартными элементами управления имеющимися в **Qt**, настроить значение для их свойств, стилизовать их внешний вид и скомпоновать их на форме. Также он содержит большое количество инструментов: поле для редактирования формы, редактор сигнально-слотовых соединений, редактор свойств объектов, средства для работы с компоновками, стилями и т. п.

Для того, чтобы продемонстрировать работу редактора форм, создадим новый пример — простой редактор текста:

1. Для того, чтобы создать форму главного окна, мы воспользуемся настройками мастера новых проектов. Вызовите мастер создания файлов и проектов, и выберите тип проекта **Qt Widgets Application**(Приложение **Qt Widgets**). В окне мастера введите имя для нового проекта (например: «**SimpleTextEditor**»), выберите также инструментарий для проекта. В окне **Class Information** (Информация о классе) выберите **QMainWindow** в качестве базового класса главного окна. Также установите флажок **Generate Form** (Создать форму) — это укажет мастеру на необходимость создания .ui-файла для главного окна (см. рис. 15.1).

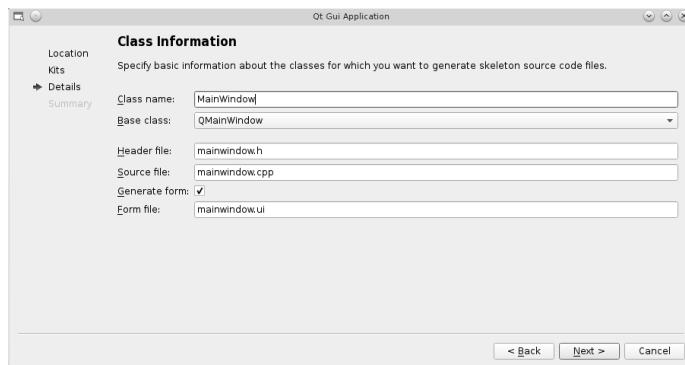


Рис. 15.1: Окно мастера проектов: создание класса главного окна и формы.

2. После создания проекта откройте дерево проекта, и в разделе **Forms**(Формы) выберите файл формы для главного окна (**mainwindow.ui**). **Qt Creator** сразу перейдет в режим редактирования пользовательского интерфейса **Design** (Дизайн). Интерфейс редактора форм состоит из нескольких панелей (см. рис. 15.2).
3. Выберите на панели виджетов «**Plain Text Edit**» и перетащите его мышкой на форму. Для удобного поиска виджетов вы можете воспользоваться фильтром в верхней части панели. Для поиска виджета введите его название.

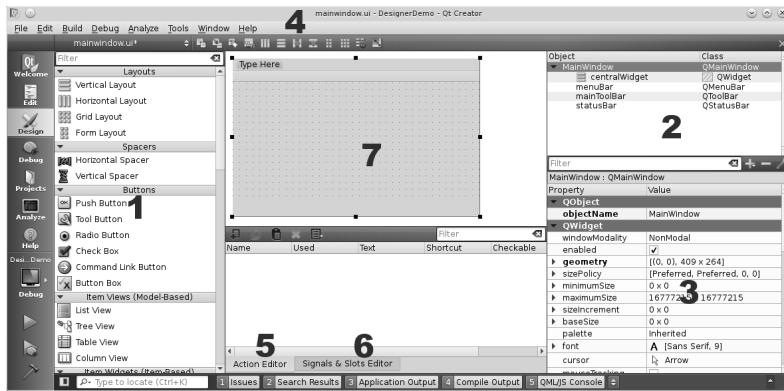


Рис. 15.2: Интерфейс редактора форм: 1. Панель виждетов (Widget box) 2. Окно дерева объектов (Object inspector) 3. Редактор свойств (Property editor) 4. Панель переключения режимов работы редактора форм 5. Редактор действий (Action editor) 6. Редактор сигнально-слотовых соединений (Signals & Slots editor) 7. Центральная часть окна, в которой размещена форма.

4. Для того чтобы разместить **Plain Text Edit** в компоновке внутри главного окна, нажмите правой кнопкой мыши на свободной от виджетов части формы и выберите в контекстном меню тип компоновки (подменю **Lay out**). Для примера мы используем вертикальную компоновку. После выбора компоновки текстовое поле займет все свободное пространство формы.
5. Добавим главное меню программы. Поскольку для главного окна был избран класс **QMainWindow**, то панель главного меню (**QMenuBar**) , панель статуса (**QStatusBar**) и панель инструментов (**QToolBar**) автоматически добавлены к проекту (убедитесь в этом просмотрев дерево объектов). Главное меню расположено в верхней части формы, и пока не содержит ни одного элемента. Нажмите два раза мышкой на надписи «Type here» (Пишите здесь) в главном меню, и введите «&File». По окончании ввода нажмите **Enter** — на форме появится меню «File». Откройте меню «File» и введите так же еще пункты: «&New», «&Open...» и «&Save...». Добавьте разделитель в меню нажав «Add separator» (Добавить разделитель). После этого добавьте еще один пункт — **&Exit**.
6. Так же добавьте меню «Edit» и «About», добавьте подпункты для этих меню (смотрите на рисунке ниже). Для того, чтобы к пунктам главного меню можно получить доступ с помощью комбинации клавиш (**Alt+<подчеркнутая буква в названии пункта>**), используют символ «&». Например, для того чтобы открыть меню File с помощью комбинации **Alt+F**, название пункта меню задают как «&File (см. рис. 15.3–15.5). Все пункты меню и разделители можно упорядочить перетащив мышкой. До-



Рис. 15.3: Меню «File» после редактирования.

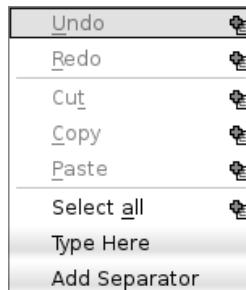


Рис. 15.4: Меню «Edit» после редактирования.

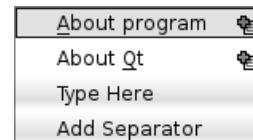


Рис. 15.5: Меню «About» после редактирования.

дополнительные подменю можно создать для каждого из пунктов меню, которые уже существуют, нажав на значок справа от названия пункта.

7. Для каждого из пунктов автоматически было создано соответствующее действие (**QAction**). Список действий можно просмотреть в редакторе действий (Action editor) на одной из страниц нижней панели редактора форм. Нажмите два раза мышкой на действие — появится диалоговое окно, в котором можно отредактировать свойства для действия: текст (Text), имя объекта **QAction** (Object name), подсказку (ToolTip) — для панели инструментов, куда будет добавлен действие, значок (Icon) и комбинацию клавиш для вызова действия (Shortcut). Например, для того, чтобы ввести комбинацию клавиш, просто нажмите на поле Shortcut и нажмите выбранную комбинацию. Пока мы не будем добавлять горячие комбинации клавиш. Окончательный вид списка действий после редактирования менюсмотрите на рис. 15.6 ниже.

Name	Used	Text	Shortcut	Checkable	ToolTip
action_Open	✓	&Open...	<input type="checkbox"/>		Open
action_Save	✓	&Save...	<input type="checkbox"/>		Save
action_Exit	✓	&Exit	<input type="checkbox"/>		Exit
actionUndo	✓	&Undo	<input type="checkbox"/>		Undo
actionRedo	✓	&Redo	<input type="checkbox"/>		Redo
actionCut	✓	Cu&t	<input type="checkbox"/>		Cut
actionC_opy	✓	&Copy	<input type="checkbox"/>		Copy
action_Paste	✓	&Paste	<input type="checkbox"/>		Paste
actionSelect_all	✓	Select &all	<input type="checkbox"/>		Select all
actionAbout_program	✓	&About program	<input type="checkbox"/>		About program
actionAbout_Qt	✓	About &Qt	<input type="checkbox"/>		About Qt
action_New	✓	&New	<input type="checkbox"/>		New

Рис. 15.6: Вид списка действий в редакторе действий (Action Editor) после редактирования меню.

8. Уже на этапе конструирования пользовательского интерфейса в редакторе форм мы можем создать сигнально-слотовые соединения между объектами на форме. Для этого перейдите на вкладку Signals & Slots Editor на нижней панели. Для того, чтобы добавить новое соединение нажмите на значок с символом «+». Появится новая строка, в которой двойным щелчком мыши в каждом из столбцов можно вызвать меню со списком доступных вариантов. Выберите в качестве объекта (Sender) actionExit, который будет посылать сигнал (Signal) triggered(), а в качестве адресата (Receiver) выберите MainWindow и слот (Slot) close(). Также добавьте остальные сигнально-слотовые соединения для действий, как на рис. 15.7.

Sender	Signal	Receiver	Slot
plainTextEdit	copyAvailable(bool)	actionC_opy	setEnabled(bool)
plainTextEdit	copyAvailable(bool)	actionCut	setEnabled(bool)
plainTextEdit	undoAvailable(bool)	actionUndo	setEnabled(bool)
plainTextEdit	redoAvailable(bool)	actionRedo	setEnabled(bool)
plainTextEdit	modificationChanged(bool)	MainWindow	setWindowModified(bool)
action_Paste	triggered()	plainTextEdit	paste()
action_Exit	triggered()	MainWindow	close()
actionUndo	triggered()	plainTextEdit	undo()
actionSelect_all	triggered()	plainTextEdit	selectAll()
actionRedo	triggered()	plainTextEdit	redo()
actionCut	triggered()	plainTextEdit	cut()
actionC_opy	triggered()	plainTextEdit	copy()

Action Editor   Signals & Slots Editor

Рис. 15.7: Вид редактора сигналов и слотов (Signals & Slots Editor) после редактирования

9. Такие действия как Undo (Отменить), Redo (Повторить), Cut (Вырезать), Copy (Копировать), теперь присоединены к соответствующим сигналам Plain Text Edit, которые будут сигнализировать о возможности их выполнение действий пользователя в редакторе. Запустим созданную форму в режиме просмотра — для этого выберите Tools->Form Editor->Preview... (Инструменты->Редактор форм->Предпросмотр) или нажмите комбинацию клавиш (Alt+Shift+R). Редактор форм загрузит и покажет на экране форму без предварительной компиляции всей программы. Если мы сразу же откроем меню, то увидим что пункты Undo, Redo, Cut, Copy уже активны, хотя изменения, отмены и выделение текста еще не происходило. Это связано с тем, что по умолчанию, созданные действия являются активными. Для того чтобы это исправить, откройте дерево объектов (Object Tree), выберите любую из действий для пунктов Undo, Redo, Cut, Copy и снимите флажок enabled для них в редакторе свойств (Property Editor). Редактор свойств позволяет настроить каждый из виджетов на форме, изменив значение для их свойств.
10. Скомпилируйте программу и запустите. Главное окно будет иметь вид, который мы задали на этапе проектирования формы. Некоторые пункты ме-

нию уже работают благодаря сигнально-слотовым соединениям, которые мы сделали.

В следующем параграфе мы продолжим работу над примером и продемонстрируем, как получить доступ к элементам формы из кода программы.

### 15.3 Программирование формы созданной в Qt Designer

Рассмотрим, как файлы форм интегрируются в проект. Как мы уже отмечали ранее, для того чтобы добавить файл формы в проект существует специальная переменная FORMS. Если мы откроем .pro-файл, то увидим:

```
FORMS += mainwindow.ui
```

Файлы форм добавленые в проект, считаются и превращаются в эквивалентный C++ код с помощью специальной программы `uic` (это происходит во время предварительной обработки проекта с помощью `qmake`). Код сгенерированный `uic` создает виджеты и компоновки, содержит настройки свойств, сигнально-слотовые соединения и стили, необходимые для получения визуального отображения содержимого .ui-файла. Если мы откроем папку с проектом (или папку для shadow build), то увидим среди исходных текстов и сгенерированных файлов файл `ui_mainwindow.h`, содержащий сгенерированный для формы код. Конечно, этот файл должен быть добавлен где-то в коде программы с помощью директивы `#include` для того, чтобы можно было воспользоваться сгенерированным кодом. Но нет необходимости делать это самостоятельно — гораздо легче воспользоваться мастером создания файлов и проектов.

При создании проекта для нашего примера мы установили флагок «Generate form» для того, чтобы автоматически сгенерировать форму для главного окна и добавить необходимый код для доступа к элементам на форме в программе. Среди кода, мастер автоматически добавил к объявлению и реализации класса `MainWindow`:

- предварительное объявление класса формы:

```
//Предварительное объявление класса формы созданной из .ui-файла
namespace Ui {
    class MainWindow;
}
```

- указатель на объект формы, позволяющий получить доступ к элементам на форме:

```
class MainWindow : public QMainWindow
{
    ...
private:
    //Указатель на объект формы созданной из .ui-файла
    Ui::MainWindow *ui;
```

- Конструктор главного окна создает объект класса формы и инициализирует переменную `ui`, а также вызывает метод `ui->setupUi(this)`, который создает все элементы которые есть на форме и устанавливает текущий виджет как родительский для них;

```
//Файл сгенерированный ui.c при обработке файла формы
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) //Создаем объект формы, созданной в QtDesigner
{
    ui->setupUi(this); //Применяем дизайн, который мы создали в QtDesigner к текущему
    окну
    ...
}
```

- Деструктор: удаляет объект на который ссылается указатель ui

```
MainWindow::~MainWindow()
{
    //Удалить форму из памяти
    delete ui;
}
```

Таким образом, через указатель ui мы можем получить доступ к созданной форме и элементам на ней. Например, мы можем получить доступ к действиям добавленным к главному меню нашего редактора и задать горячие клавиши для них:

```
//Задаём комбинации клавиш для действий
ui->actionUndo->setShortcut(QKeySequence::Undo);
ui->actionRedo->setShortcut(QKeySequence::Redo);
ui->actionCut->setShortcut(QKeySequence::Cut);
ui->actionCopy->setShortcut(QKeySequence::Copy);
ui->action_Paste->setShortcut(QKeySequence::Paste);
ui->action_Select_all->setShortcut(QKeySequence::SelectAll);
ui->action_New->setShortcut(QKeySequence::New);
ui->action_Open->setShortcut(QKeySequence::Open);
ui->action_Save->setShortcut(QKeySequence::Save);
ui->action_Exit->setShortcut(QKeySequence::Quit);
```

Обратите внимание на то, как мы используем класс `Qt QKeySequence` для того, чтобы задать горячие клавиши для действий. Помимо возможности задать комбинацию клавиш текстовой строкой (например `QKeySequence (Ctrl + X)`) или с помощью специально определенных констант для клавиш (например `QKeySequence (Qt :: CTRL + Qt :: Key_X)`), можно воспользоваться набором стандартных клавиатурных сокращений, которые будут соответствовать стандартам системы. Так для повторения отмененного действия в ОС Linux пользователь сможет воспользоваться комбинацией клавиш `Ctrl+Shift+Z`, а в ОС Windows привычной комбинацией `Ctrl+Y`.

Нашему текстовому редактору еще хватает функциональности: нам необходимо запрограммировать создание, открытие и сохранение текстового файла, а также пункты меню `Help`.

Для начала добавим объявления слота для создания нового файла и добавим поле для сохранения пути текущего открытого файла:

```
private slots:
    void slotNew();

private:
    QString mFileName;
```

Мы используем частный метод `updateTitle()` для того, чтобы обновить заголовок окна и вывести название программы и путь к текущему открытому файлу:

```
private:
void updateTitle();
```

Реализация метода обновления заголовка:

```
//Метод обновления заголовка окна
void MainWindow::updateTitle()
{
//Подставляем в название заголовке имя поточного открытого файла. Комбинацией
//символов "/*"
//обозначаем место, где будет выводиться знак "*" в случае, когда содержимое окна
//модифицировано.QString lTitle=QString("TextEditor- // устанавливаем заголовок
//окнаsetWindowTitle(lTitle);
```

Метод **setWindowTitle()** позволяет не только задать текст заголовка для окна, но и отметить несохраненные изменения в текущем открытом документе. Для этого, после заголовка мы добавили шаблон **[\*]**. Теперь, если сообщить главному окну про редактирование содержимого посредством вызова слота **QWidget::setWindowModified(bool)** со значением **true**, то после текста заголовка появится символ **«\*»**, означающий, что содержание главного окна изменено и его следует сохранить. Слот **setWindowModified(bool)** мы соединили с сигналом **modificationChanged(bool)** текстового поля **QPlainTextEdit** с помощью редактора сигнально-слотовых соединений в редакторе форм (см. предыдущий пункт).

Добавим реализацию слота **MainWindow::slotNew()**:

```
//Слот для создания нового документа
void MainWindow::slotNew()
{
    mFileName = "UntitledFile"; //Задать имя для нового файла по умолчанию
    ui->plainTextEdit->clear(); //Очистить текстовое поле
    setWindowModified(false); //Установить — содержание не модифицировано
    updateTitle(); //Обновить заголовок окна
}
```

Теперь присоединим объект **QAction** для создания нового документа к слоту:

```
connect(ui->action_New, SIGNAL(triggered()),this, SLOT(slotNew()), Qt::UniqueConnection);
```

Сигнал выпускается при выполнении действия (вызыва пункта меню, нажатие кнопки на панели инструментов, для которой была добавлено действие и т.д.). В конце конструктора вызовем слот **slotNew()**:

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
//Создаем объект, который содержит дизайн созданный в редакторе форм
{
    ...
//В конце вызываем слот для нового документа. Таким образом, пользователь сможет сразу же
//приступить к работе
    slotNew();
}
```

Остальные пункты меню мы запрограммируем в следующем параграфе, в котором мы рассмотрим работу со стандартными системными диалогами в **Qt**.

## 15.4 Стандартные диалоги

Диалог выбора файла для открытия и сохранения, диалог выбора шрифта, окна сообщений об ошибках являются примерами диалоговых окон, с которыми часто приходится сталкиваться при работе с программами. Такие диалоги обычно имеют стандартные для всех программ в системе вид и функциональность. Qt позволяет воспользоваться готовыми диалогами для этих целей, которые легко вызвать в программе. Классы для работы с диалогами, которые часто используют в программе, приведены в табл. 15.1.

Таблица 15.1: Некоторые классы готовых диалогов Qt

Класс	Описание особенностей
QInputDialog	Используют для удобства в случае когда необходимо ввести числовое значение или строку текста. Имеет несколько сигналов, которые сигнализируют об изменении значения в поле ввода. Класс имеет статические методы для вызова диалога ввода числа ( <code>getDouble()</code> , <code>getInt()</code> ), ввода ( <code>getText()</code> ) выбора элемента из списка ( <code>getItem()</code> ).
QColorDialog	Стандартный диалог выбора цвета. Класс имеет статический метод <code>getColor()</code> для удобного вызова диалога.
QFontDialog	Стандартный диалог выбора шрифта. Класс имеет статический метод <code>getFont()</code> для удобного вызова диалога.
QFileDialog	Стандартный диалог выбора файла. Имеет большое количество настроек, возможность фильтрации файлов по расширению. Класс имеет статические методы ( <code>getExistingDirectory()</code> , <code>getOpenFileName()</code> , <code>getOpenFileNames()</code> , <code>getSaveFileName()</code> ) для вызова диалога.
QMessageBox	Диалог сообщение. Используют для вывода информации, сообщений об ошибках и вопросов. Класс имеет статические методы для удобного вызова в программе информационных окон ( <code>about()</code> , <code>aboutQt()</code> ), сообщений об ошибках ( <code>critical()</code> , <code>warning()</code> ), вопросов ( <code>question()</code> ) и сообщений ( <code>information()</code> ).

В нашем примере мы будем использовать класс `QFileDialog` для выбора файла при открытии и сохранении, а также класс `QMessageBox`. Добавим описание слотов для открытия и сохранения файла:

```
private slots:
void slotOpen();
void slotSave();
```

Подключим необходимые заголовочные файлы в `mainwindow.cpp`:

```
#include <QFileDialog>
#include <QMessageBox>
#include <QDir>
```

Добавим реализацию для слотов `slotOpen()` и `slotSave()`:

```
//Слот для открытия файла в редакторе
void MainWindow::slotOpen()
```

```

{
    //Вызывать системный диалог открытия файла в домашней папке пользователя
    QString lFileName=QFileDialog::getOpenFileName(this, "Open file...", QDir::
        homePath(), "Text files (*.txt);;All files (*.*)"); //указываем фильтры для
        //просмотра файлов
    if (lFileName.isEmpty())//Если пользователь не выбрал ни одного файла
    {
        return; //выйти из метода
    }
    //Спросить пользователя о сохранении документа
    if (!askForFileSaveAndClose())
    {
        //Если пользователь нажал «Отмена» игнорировать вызов — продолжать работу
        return;
    }
    QFile lFile(lFileName); //Устанавливаем имя открытого файла
    //Если текстовый файл открыт только для чтения...
    if (lFile.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        mFileName = lFileName; //задать имя файла
        //читаем все содержимое и устанавливаем текст для редактора
        ui->plainTextEdit->setPlainText(lFile.readAll());
        lFile.close(); //закрываем открытый файл
        //устанавливаем состояние окна — содержимое не модифицировано
        setWindowModified(false);
        //и обновляем заголовок окна для демонстрации названия текущего открытого файла
        updateTitle();
    }
    else
    {
        //Если при открытии файла возникла ошибка выводим диалоговое окно с сообщением,
        //содержащим имя файла, одну кнопку «Ок» и заголовок «Ошибка»
        QMessageBox::warning(this, "Error", QString("Could not open file %1 for
            reading").arg(lFile.fileName()), QMessageBox::Ok);
    }
}
//Слот для сохранения изменений в текущем файле
void MainWindow::slotSave()
{
    //Если содержимое не модифицировано...
    if (!isWindowModified()) //Если содержимое не модифицировано
    {
        return; //Выйти из метода — продолжить работу
    }
    //Вызывать системный диалог сохранения файла в домашней папке пользователя
    QString lFileName=QFileDialog::getSaveFileName(this, tr("Save file..."),
        QDir::homePath(), tr("Text files (*.txt);;All files (*.*)"));
    //Если пользователь не выбрал имя файла для сохранения...
    if (lFileName.isEmpty())
    {
        return; //... выйти из метода
    }
    QFile lFile(lFileName); //Устанавливаем имя открытого файла
    //Если текстовый файл открыт для записи
    if (lFile.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        mFileName = lFileName; //Задать имя файла
        //Создаем временный QByteArray для записи данных
        QByteArray lData;
        //Читаем текст из редактора и добавляем QByteArray, записываем в файл и закрываем файл
        //после записи
        lData.append(ui->plainTextEdit->toPlainText());
        lFile.write(lData);
        lFile.close();
        //Устанавливаем состояние окна — содержимое не модифицировано
    }
}

```

```

        setWindowModified( false ) ;
}
else
{
    //Если при открытии файла возникла ошибка выводим диалоговое окно с сообщением,
    //содержащим имя файла, одну кнопку «Ок» и заголовок «Error»
    QMessageBox::warning( this , "Error" , QString("Could not open file %1 for
        writing").arg( lFile.fileName() ) , QMessageBox::Ok );
}
}

```

Здесь в начале каждого из слотов мы вызываем диалог для выбора файла с помощью статических методов класса `QFileDialog`. Диалог для открытия файла мы вызываем с помощью метода `QFileDialog::getOpenFileName()`, а для сохранения — с помощью `QFileDialog ::getSaveFileName()`. Для каждого из случаев диалог будет иметь соответствующий вид. Для того, чтобы добавить фильтр для текстовых файлов мы передаем строку с описанием фильтра — «Text files (\*.txt );All files (\*.\*)». В описании — названия для фильтров и шаблоны для фильтрации (в скобках). Можно задавать несколько шаблонов через пробел при необходимости. Фильтры в списке разделяем с помощью двойной точки с запятой.

После выбора пользователем файла, статический метод вернет полный путь к нему. В случае, когда пользователь закрыл диалог или нажал «Отмена», метод вернет пустую строку. Для выбранного файла мы выполняем чтение содержимого и сохранения. Если при открытии возникла ошибка, мы выводим ее с помощью статического метода `QMessageBox::warning()`.

При открытии файла мы использовали собственный метод `askForFileSaveAndClose()`, который должен проверить текущий открытый файл на изменения и предложить пользователю сохранить их перед открытием другого файла. Добавим описание этого метода к описанию класса `MainWindow`:

```
private:
bool askForFileSaveAndClose();
```

и его реализацию в файл `mainwindow.cpp`:

```
//Метод для проверки текущего файла на изменения и вывода диалога для пользователя, с
//предложением
//сохранить изменения. Метод возвращает логическое значение, содержащее false в случае,
//когда пользователь нажал в диалоге кнопку «Cancel»
bool MainWindow :: askForFileSaveAndClose()
{
    if (isWindowModified()) //Если содержимое окна модифицировано
    {
        //вызываем диалог с вопросом, нужно ли сохранять изменения: подставляем в текст диалога
        //название
        //текущего открытого файла, задаем кнопки: «Да», «Нет» и «Отмена».
        //Результат работы диалога (нажатой кнопки) записываем в переменную
        int lResult = QMessageBox::question( this , tr("Save changes") ,
            QString(tr("File %1 is modified. Do you want to save your changes?")).arg
                (mFile.fileName() ) , QMessageBox::Yes , QMessageBox::No , QMessageBox::Cancel );
        if (QMessageBox :: Yes == lResult) //Если нажали кнопку «Да»
        {
            slotSave(); //сохранить изменения
        }
        else
        {
            if (QMessageBox :: Cancel == lResult) //Если нажали кнопку «Отменить»
```

```

    {
        return false;
    }
}
return true;
}

```

В этом фрагменте программы мы использовали статический метод `QMessageBox :: question`, и задали заголовок, текст и кнопки на диалоге с помощью специальных констант (`QMessageBox::Yes`, `QMessageBox::No`, `QMessageBox::Cancel`). Он, как и почти все статические методы `QMessageBox`, возвращает значение нажатой кнопки. Это значение мы сравниваем со значениями констант для кнопок, чтобы определить, какие дальнейшие действия выбрал пользователь. Используем метод `askForFileSaveAndClose()` также и в слоте для нового текстового файла:

```

//Спросить пользователя о сохранении документа
if (!askForFileSaveAndClose())
{
    //Если пользователь нажал «Отменить» игнорировать вызов — продолжать работу
    return;
}

```

Осталось реализовать вывод информации о программе. Для этого сначала добавим к .pro-файлу информацию о версии. Например, добавим переменные с большим и меньшим номерами версии, а потом передадим их в переменную `DEFINES` таким образом, чтобы они были объявлены в программе. Это может быть удобно для дальнейшего изменения версии при разработке программы:

```

MAJOR_VERSION = 1
MINOR_VERSION = 0
DEFINES += \
    MAJOR_VERSION=$$MAJOR_VERSION \
    MINOR_VERSION=$$MINOR_VERSION

```

После изменений в файле проекта не забудьте вызвать `qmake` еще раз, чтобы программа обработала все изменения в файле проекта (выберите в главном меню `Build->Run qmake`) Теперь в файле `main.cpp` зададим версию, а также название для `QApplication`:

```

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    a.setApplicationName("TextEditor");
    a.setApplicationVersion(QString("%1.%2")
        .arg(MAJOR_VERSION)
        .arg(MINOR_VERSION));
    MainWindow w;
    w.show();
    return a.exec();
}

```

Теперь используем объект `QApplication` для получения версии и названия программы в слоте отображения информации. Добавим объявление слота, а также следующую его реализацию:

```

//Слот для отображения информации о программе
void MainWindow::slotAboutProgram()
{

```

```
//Выводим диалоговое информационное окно с сообщением, куда подставляем версию и название
//программы возвращаемых QApplication. Указываем — окно содержит заголовок «About».
QMessageBox::about(this, tr("About"),
QString("%1 v. %2").arg(qApp->
applicationName()).arg(qApp->applicationVersion()));
}
```

В конце подсоединяем сигналы от пунктов главного меню к созданным слотам:

```
//Присоединяя действия к созданным слотам
connect(ui->action_New, SIGNAL(triggered()), this, SLOT(slotNew()), Qt::UniqueConnection);
connect(ui->action_Open, SIGNAL(triggered()), this, SLOT(slotOpen()), Qt::UniqueConnection);
connect(ui->action_Save, SIGNAL(triggered()), this, SLOT(slotSave()), Qt::UniqueConnection);
connect(ui->actionAbout_Qt, SIGNAL(triggered()), qApp, SLOT(aboutQt()), Qt::UniqueConnection);
connect(ui->actionAbout_program, SIGNAL(triggered()), this, SLOT(slotAboutProgram()), Qt::UniqueConnection);
```

## 15.5 Ресурсы программы

Часто бывает необходимо добавить в программу дополнительные файлы такие как изображения и значки, используемые при оформления интерфейса, звуковые файлы для уведомлений пользователя, сценарии, выполняемые программой и т.д. В таких случаях можно воспользоваться преимуществами, которые предоставляет ресурсная система **Qt**.

Добавляя файл как ресурс в программу, мы указываем, что хотим включить данные, содержащиеся в этом файле в исполняемый файл. Таким образом скомпилированная программа будет содержать все ресурсные файлы внутри. Средства **Qt** позволяют обращаться к этим данным, и считывать файлы в ресурсах так же, как и обычные файлы в файловой системе.

Для демонстрации работы с ресурсами в **Qt** добавим изображения для пунктов меню и кнопки на панель инструментов текстового редактора.

1. В папке проекта создадим подкаталог **resources**. Внутри добавим еще одну вложенную папку с названием **images** и добавим туда значки для меню.
2. Вызовем мастер создания новых файлов и проектов. Выберем раздел **Qt** и создадим **Qt Resource File** (Файлресурсов **Qt**). В следующем окне мастера, введем для файла имя в поле **Name (Имя)**: **resources.qrc**. В поле **Path (Путь)** нажмем на кнопку и в диалоге выберем созданную папку **resources**. Нажмем **Next (Далее)** и в следующем окне **Finish (Завершить)**.
3. Выберем созданный файл ресурсов в дереве проекта (раздел **Resources**) и откроем его. В главном окне **Qt Creator** появится редактор ресурсов, который состоит из браузера ресурсов (сверху) и панели редактирования (снизу). На нижней панели выберите **Add-> Add Prefix** (Добавить->Добавить префикс). Сразу же в редакторе появится раздел для ресурсов и станет доступным для редактирования его название (поле **Prefix**). Установите название для раздела: **actions**.

4. Выберите раздел **actions** в редакторе ресурсов и нажмите внизу на панели Add-> **Add Files** (Добавить->Добавить файлы). В диалоге выбора файлов откройте папку **resources/images** внутри папки с проектом, выберите все файлы в папке и нажмите **Open**. Пиктограммы сразу будут добавлены в раздел и показаны в редакторе.
5. Для каждой из пиктограмм можно выбрать псевдоним — второе имя по которому к пиктограмме можно получить доступ. Это делают для того, чтобы при изменении файла не нужно было менять путь к изображению в коде программы. Для этого достаточно добавить другое изображение и установить такой же псевдоним как для старого, а старому изображению, в свою очередь, удалить псевдоним. Установите псевдоним для каждого из изображений согласно действию для которого они предназначены (смотрите на рис. 15.8).
6. Изображение для действия можно добавить в код программы вызвав для действия метод **QAction::setIcon()**. Например: `ui->action_New->setIcon (QIcon(":/actions/new"));`

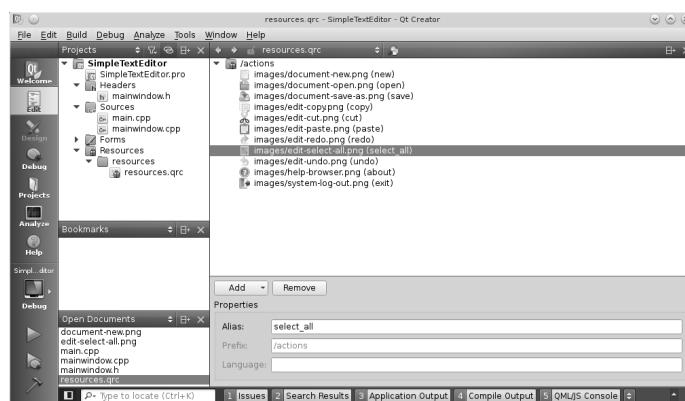


Рис. 15.8: Редактирование файла ресурсов

Обратите внимание на то, как мы указываем путь к файлу с изображением в ресурсах программы. Для того чтобы получить доступ к ресурсам мы начинаем путь с символов `:`. Далее мы указываем раздел, куда мы добавили пиктограммы — **actions**, и затем указываем псевдоним для самого изображения в ресурсной системе — **new**. Для того, чтобы установить изображение для действия, мы создаем временный объект **QIcon**, которому передаем путь к изображению.

Для того, чтобы добавить изображение для действий, мы воспользуемся другим способом и добавим изображение в редакторе действий. Для этого перейдите к редактору действий, два раза нажмите на действие и в диалоге редактирования нажмите на кнопку «...» в поле **Icon** (Значок). Выберите нужное изображение из ресурсов. Так же измените и другие действия.

7. Добавим действий теперь также и в панель инструментов. Для этого перетащите действие из редактора действий на панель инструментов, которая расположена прямо под главным меню на форме. Для того, чтобы добавить разделитель нажмите правой кнопкой на панели и выберите Append Separator. Выберите в редакторе свойств для панели инструментов свойство `toolButtonStyle` и установите его в значение `ToolButtonTextUnderIcon`. Запустите просмотр для формы (см. рис. 15.9).



Рис. 15.9: Редактор с панелью инструментов

## 15.6 Создание собственных диалогов

Для нашего примера мы создадим диалог настроек, с помощью которого мы будем изменять вид главного окна.

1. Вызовем мастер новых файлов и проектов. Выберем раздел **Files and Classes->Qt** (Файлы и классы -> Qt) и **Qt Designer Form Class** (Класс формы Qt Designer) для создания класса окна вместе с файлом формы. Выберем в окне **Choose a Form Template** (Выбор шаблона формы) шаблон для нового окна — **Dialog with Buttons Bottom** (Диалог с кнопками внизу) и нажмите **Next**. Этим мы укажем мастеру создать окно, унаследовав класс от **QDialog** и добавить внизу формы дополнительно две кнопки **Ok** и **Cancel** (см. рис. 15.10).
2. **QDialog** может возвращать значение после завершения, которое можно получить с помощью метода **int result()**. Перед выходом из диалога должен вызываться один из двух методов: **QDialog::accept()**

или `QDialog::reject()`. Они изменяют значение `QDialog::result()` на `QDialog::Accepted` и `QDialog::Rejected` соответственно и закрывают диалог. Обратите внимание, что после создания формы для диалога, кнопки внутри `QDialogButtonBox` были автоматически подключены к слотам диалога `accept()` и `reject()` (убедитесь в этом открыв редактор сигнално-слотовых соединений в редакторе форм для диалога). Также `QDialog` можно вызвать с помощью метода `exec()`, который также возвращает результат его выполнения. В этом случае диалог вызывается как модальный.

3. В следующем окне (см. рис. 15.11) напишем имя класса для окна: `SettingsDialog` (поле `Class name`). Нажмем кнопку `Next` и в следующем окне нажмем `Finish`. Откроем форму в редакторе. На форме мастер автоматически добавил `Dialog Button Box` с двумя кнопками «`Ok`» и «`Cancel`».

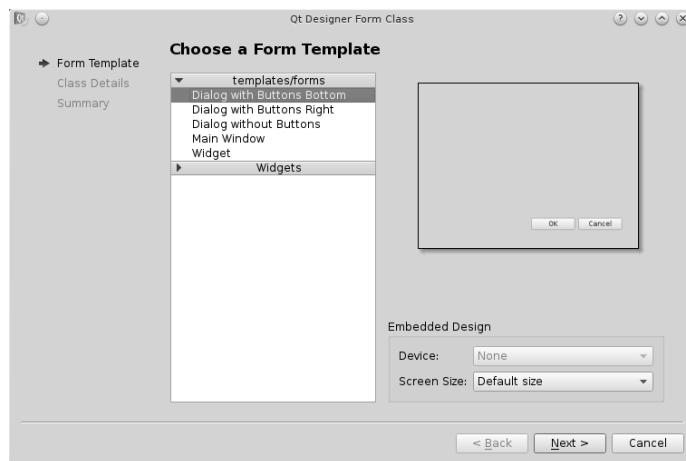


Рис. 15.10: Создание диалога настроек (шаг 1).

4. Наш диалог будет содержать одну группу настроек. Это будут настройки вида окна, а именно настройки видимости панели инструментов и строки статуса. Для этого перетащим на форму элемент `Group Box`. Для того чтобы отредактировать заголовок два раза нажмите мышкой на тексте надписи или выберите свойство `title` в редакторе свойств. Измените заголовок на `View Settings`.
5. Перетащите два флажка в `Group Box` и назовите их `Show Toolbar` и `Show Status Bar` (см. рис. 15.12). Выберите свойство `objectName` для каждого из них и установите в `showToolbarCheckBox` и `showStatusBarCheckBox` соответственно. Нажмите правой кнопкой мыши внутри элемента `Group Box` и выберите `Lay out->Lay out Horizontally` (Компоновка->Скомпоновать по горизонтали). Флажки расположатся в горизонтальной компоновке внутри элемента `Group Box`.

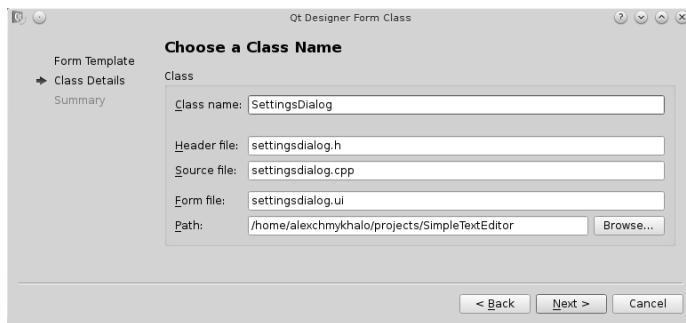


Рис. 15.11: Создание диалога настроек (шаг 2).

6. Нажмите на свободном от элементов пространстве формы и выберите в контекстном меню **Layout->Lay out Vertically** (Компоновка->Скомпоновать по вертикали). Элементы **Group Box** и **Button Box** расположатся в вертикальной компоновке на форме.
7. Измените размер формы. Для этого выберите **SettingsDialog** в дереве объектов и подведите указатель мышки к краю формы или выберите свойство **geometry** и измените. Сделайте размер формы меньше. В редакторе свойств выберите флажок **modal** и установите его. Измените свойство **windowTitle** на «**Settings**».



Рис. 15.12: Пример: Диалог настроек для проекта SimpleTextEditor.

Объявление класса диалога настроек (**settingsdialog.h**):

```
class SettingsDialog : public QDialog
{
    Q_OBJECT
public:
    explicit SettingsDialog(QWidget *parent = 0);
    ~SettingsDialog();
    bool isShowToolBar() const;
    void setShowToolBar(bool pShow);
    bool isShowStatusBar() const;
    void setShowStatusBar(bool pShow);
private:
    Ui::SettingsDialog *ui;
};
```

Также реализуем методы для установки и получения значения флагков (**settingsdialog.h**):

```

bool SettingsDialog :: isShowToolBar() const
{
    return ui->showToolbarCheckBox->isChecked();
}
void SettingsDialog :: setShowToolBar(bool pShow)
{
    ui->showToolbarCheckBox->setChecked(pShow);
}
bool SettingsDialog :: isShowStatusBar() const
{
    return ui->showStatusBarCheckBox->isChecked();
}
void SettingsDialog :: setShowStatusBar(bool pShow)
{
    ui->showStatusBarCheckBox->setChecked(pShow);
}

```

Добавим вызов диалога из программы. Для этого в файле `mainwindow.h` сделаем предварительное объявление:

```

class SettingsDialog;
и добавим объявление слота, который будет показывать диалог и переменную,
указывающую на объект диалога настроек:
private slots:
    void showPreferencesDialog();
private:
    SettingsDialog *mSettingsDialog;

```

В файле `mainwindow.cpp` подключите файл описания класса `SettingsDialog`:

```
#include "settingsdialog.h"
```

Создаем диалог настроек

```

MainWindow :: MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    mSettingsDialog(new SettingsDialog(this))
//Создаем диалог настроек
{
    ui->setupUi(this); //Применяем дизайн, который мы создали в редакторе форм в текущем
    окне

```

и добавляем реализацию для слота, который вызовет диалог:

```

void MainWindow :: showPreferencesDialog()
{
    //Показываем диалог настроек
    mSettingsDialog->show();
}

```

В конце добавляем сигнально-слотовое соединение, которое присоединит сигнал от меню к слоту:

```
connect(ui->actionPr_ eferences, SIGNAL(triggered()), mSettingsDialog, SLOT(
    showPreferencesDiaLog()), Qt :: UniqueConnection);
```

Конечно, мы могли присоединить сигнал к слоту `show()` диалога напрямую, но в следующем параграфе мы рассмотрим, как хранить в системе и считывать настройки нашей программы, и перед тем как вызвать диалог мы добавим код, который будет инициализировать диалог настроек.

А пока суммируем наши знания о дизайнере форм. Алгоритм работы с дизайнером форм выглядит примерно так:

1. Примерно разместить виджеты на форме;

2. Применить к виджетам компоновки, начиная с наиболее вложенных виджетов;
3. Добавить меню, панели, настроить действия;
4. Настроить объекты на форме с помощью редактора свойств;
5. Сделать сигнально-слотовые соединения, если необходимо;
6. Реализовать логику работы в исходном коде программы.

## 15.7 Сохранение настроек

Для сохранения настроек в программе мы воспользуемся классом `QSettings`. Он позволяет сохранять настройки программы стандартным для программной платформы способом. В ОС Linux и Mac OS X это конфигурационные файлы (`.ini` и `.plist`), а под управлением ОС Windows настройки программы можно сохранять в системный реестр. Также класс `QSettings` часто используют для редактирования произвольных `.ini` и `.plist` файлов и ключей системного реестра Windows.

Чтобы добавить поддержку сохранения настроек в программе объявим несколько дополнительных методов.

```
private slots:
    void slotPreferencesAccepted();
private:
    void readSettings();
    void writeSettings();
    void applySettings();
```

Метод `readSettings` будет читать настройки программы и инициализировать настройками наш диалог. Метод `writeSettings` будет читать настройки, установленные в диалоге, и записывать их. Наконец метод `applySettings` будет получать текущие настройки от диалога и настраивать соответствующим образом программу.

```
void MainWindow::readSettings()
{
    //Указываем, где хранились настройки. QSettings::NativeFormat — в формате определенном
    //системой
    //QSettings::UserScope — настройки для каждого пользователя отдельно.
    //Также устанавливаем имя организации и название программы
    QSettings lSettings(QSettings::NativeFormat, QSettings::UserScope, "", qApp
        ->applicationName());
    //Открываем группу настроек
    lSettings.beginGroup(SETTINGS_GROUP_VIEW);
    //Читаем настройки
    bool lShowToolBar=lSettings.value(SETTING_SHOW_TOOLBAR, true).toBool();
    mSettingsDialog->setShowToolBar(lShowToolBar);
    bool lShowStatusBar = lSettings.value(SETTING_SHOW_STATUS_BAR, true).toBool();
    mSettingsDialog->setShowStatusBar(lShowStatusBar);
}
void MainWindow::writeSettings()
{
    //Указываем как сохранять настройки. QSettings::NativeFormat — в формате определенном
    //системой
    //QSettings::UserScope — настройки для каждого пользователя отдельно.
    //Также устанавливаем имя организации и название программы.
    QSettings lSettings(QSettings::NativeFormat, QSettings::UserScope, "", qApp
        ->applicationName());
```

```

//Открываем группу настроек
lSettings.beginGroup(SETTINGS_GROUP_VIEW);
//Записываем настройки
lSettings.setValue(SETTING_SHOW_TOOLBAR, mSettingsDialog->isShowToolBar());
lSettings.setValue(SETTING_SHOW_STATUS_BAR, mSettingsDialog->
    isShowStatusBar());
}
void MainWindow::applySettings()
{
//Читаем настройки установленные в диалоге и применяем их
ui->mainToolBar->setVisible(mSettingsDialog->isShowToolBar());
ui->statusBar->setVisible(mSettingsDialog->isShowStatusBar());
}

```

Модифицируем также метод showPreferencesDialog, чтобы он вызывал считывание настроек. Также добавим реализацию метода slotPreferencesAccepted.

```

void MainWindow::showPreferencesDialog()
{
    readSettings(); //Считываем настройки и устанавливаем их для диалога
    mSettingsDialog->show(); //Показываем диалог настроек
}
void MainWindow::slotPreferencesAccepted()
{
    writeSettings(); //Записать новые настройки
    applySettings(); //Применить настройки
}

```

В конструкторе главного окна соединим сигнал accepted от диалога к слоту slotPreferencesAccepted:

```

connect(mSettingsDialog, SIGNAL(accepted()), this, SLOT(slotPreferencesAccepted()), Qt::
UniqueConnection);

```

## 15.8 Использование сторонних (third party) разработок в собственном проекте

Программная реализация уже существует для множества функциональных возможностей, для большинства распространенных стандартов и для решения большого количества типичных задач. Например, реализация протокола SMTP (Simple Mail Transfer Protocol) для электронной почты или библиотека для быстрого преобразования Фурье, или даже визуальный элемент для вывода графика функции — являются примерами задач, с которыми может сталкиваться разработчик. Поэтому программист может столкнуться с необходимостью исследовать предметную область, спроектировать и разработать собственный вариант решения для них, выполняя работу заново с самого начала, хотя для этих задач уже существует реализация в других программных продуктах (библиотеках, инструментариях, и т. п.).

Использование сторонних разработок в собственном проекте может значительно сократить время создания программы, обогатить функциональные возможности, и, возможно, обеспечить совместимость со стандартами и другими программами, которые уже существуют. Конечно, в таком случае приходится иметь дело с кодом, созданным другим программистом, который может содержать свои собственные особенности, дефекты, требования к выполнению. Но в

большинстве случаев (если сторонняя разработка достаточно хорошо протестирована и рассчитана на подобное использование, и поставляется как библиотека или фреймворк) это предоставляет серьезные преимущества, поскольку дает возможность избежать повторного решения задачи, которая была решена другим разработчиком ранее. Особенно это важно для свободного программного обеспечения с открытым кодом, которое на полную мощность пользуется возможностями повторного использования.

Конечно, играет важную роль способ лицензирования программного кода, который будет предоставлять возможности для использования текста или готовой программы в собственных целях. Лицензия может также давать возможность использовать программное обеспечение или его исходный код в коммерческих проектах. Поэтому особенно важно убедиться в лицензионной чистоте кода, который Вы будете использовать в своем проекте и в ограничениях, которые лицензия накладывает на его использование.

В нашем примере мы используем библиотеку **Qwt**, которая дает дополнительный набор виджетов для вывода графиков, гистограмм, значений, а также несколько дополнительных элементов управления. Исходный код и документацию можно получить на сайте <http://qwt.sourceforge.net>.

Откроем папку с исходным кодом и файл проекта **qwt.pro**. Обратите внимание: файл **qwt.pro** содержит строку

```
include(qwtconfig.pri)
```

Файл **qwtconfig.pri** служит для настройки компиляции **Qwt**. Для компиляции примеров и тестового проекта для исследования **Qwt** откомментируйте:

```
QWT_CONFIG += QwtExamples
QWT_CONFIG += QwtPlayground
```

Также закомментируйте строку для того, чтобы построить отдельную динамическую библиотеку для **Qwt**, которую мы будем использовать (этую строку необходимо комментировать только под Windows):

```
QWT_CONFIG += QwtDesignerSelfContained
```

Далее отключите **Shadow Build** и запустите компиляцию в режиме **Release**. После завершения компиляции мы получим такое размещение файлов :

- **qwt/designer/plugins/designer** — расширение для **QtDesigner**, которое даст нам возможность использовать виджеты **Qwt** при визуальном проектировании интерфейса;
- **qwt/lib** — содержит динамически загружаемую библиотеку **Qwt**;
- **qwt/src** — содержит заголовочные файлы необходимые для разработки ;

Создадим в папке нового GUI проекта подпапку и назовем ее **3rdparty**. Добавим внутри нее еще одну папку с названием **qwt**. Наконец создадим внутри **qwt** еще две папки: **lib** и **include**. Внутрь папки **lib** скопируем содержимое **qwt/lib** — построенных библиотек **qwt**. А в папку **include** скопируем все файлы **qwt/src**, которые имеют расширение **.h**. Наконец для удобства создадим файл для включения в проект **3rdparty/qwt/qwt.pri** и добавим содержание:

```
INCLUDEPATH += $$PWD/include
LIBS += -L$$PWD/lib
LIBS += -lqwt
```

Первая строка .pri-файла указывает путь к размещению .h-файлов. Переменная \$\$PWD хранит значение текущей папки, то есть той, где находится файл qwt.pri. Вторая строка указывает размещение библиотек (-L <путь\_к\_библиотекам>). Последняя строка указывает на необходимость связывать программу во время компиляции с библиотекой qwt (-l <название\_библиотеки>, где название\_библиотеки — название файла библиотеки без расширения и префикса lib).

Отметим, что для того чтобы пристроить проект в Debug режиме необходимо добавить соответствующие (тоже построенные в Debug режиме) библиотеки. То же касается и Release режима.

Теперь добавим этот файл к проекту PlotExample.pro. После строк

```
QT      += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

добавим include(3rdparty/qwt/qwt.pri)

Теперь мы можем воспользоваться библиотекой Qwt в проекте. Создадим виджет для графика и добавим его на окно.

```
setCentralWidget(new QWidget); //Центральный виджет
QLayout *lLayout = new QVBoxLayout;
centralWidget()->setLayout(lLayout); //компоновщик для размещения графика
QwtPlot *lPlot = new QwtPlot; //Визуальный элемент — График
lLayout->addWidget(lPlot);
QwtPlotCanvas *lCanvas = new QwtPlotCanvas();
//Объект для отображения
lCanvas->setFrameStyle(QFrame::Box|QFrame::Plain); //нашего графика
lPlot->setCanvas(lCanvas);
lPlot->setAxisTitle(QwtPlot::xBottom, "x"); //Название оси — x
lPlot->setAxisScale(QwtPlot::xBottom, -10.0, 10.0); //Границы оси x
lPlot->setAxisTitle(QwtPlot::yLeft, "y"); //Название оси — y
lPlot->setAxisScale(QwtPlot::yLeft, -10.0, 10.0); //Границы оси y
new QwtPlotPanner(lCanvas); //Добавляем инструмент для перетягивания смещения
//графика указателем мышки
new QwtPlotMagnifier(lCanvas); //Добавляем инструмент для увеличения уменьшения
//графика роликом мышки
```

После запуска программы мы увидим пустое окно графика на экране (рис. 15.13).

## 15.9 Задачи для самостоятельного решения

1. Добавьте вывод на панель статуса текущей позиции курсора в текстовом редакторе. Используйте класс `QStatusBar` и его метод `showMessage()`, который принимает как параметр сообщение и количество милисекунд в течении которых сообщение видимо. Используйте сигнал `QPlainTextEditor::currentPositionChanged()` для определения изменения позиции курсора.
2. Добавьте к настройкам опцию для сохранения геометрии главного окна после закрытия программы. Если флагок опции включен, то после запуска

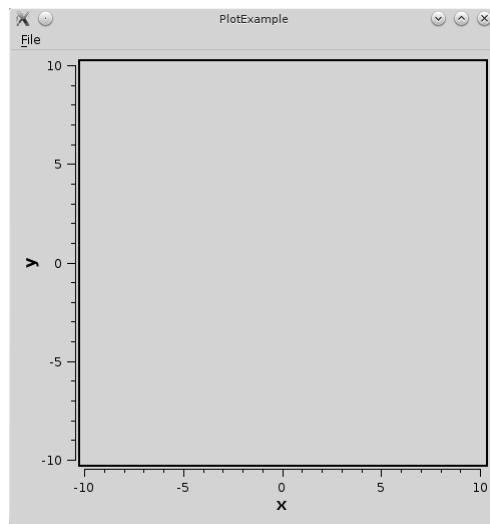


Рис. 15.13: Пример: программа для вывода графика функции.

главное окно будет на той же позиции и тех же размеров, как и в момент перед закрытием программы.

3. Создайте проект с графическим интерфейсом. Разместите на окне в компоновщиках 5 различных виджетов. Соедините их (по 2–3 между собой) сигнально-слотовыми соединениями, таким образом, чтобы они реагировали на изменения состояния друг друга. Выполните это задание исключительно с помощью редактора форм. Все виджеты должны быть расположены в компоновщиках и пропорционально изменять размеры при изменении размеров окна.
4. Попробуйте добавить вывод простого графика функции приведенному выше примеру использования компоненты *Qwt*.
5. Запрограммируйте вывод протабулированной функции на график из файла. Для этого добавьте пункт главного меню «File-> Load...», который будет показывать диалог открытия файла. После того, как пользователь выберет файл, программа должна загрузить данные из него и построить график функции.
6. Попробуйте использовать в качестве посторонней компоненты для вывода графика — *QCustomPlot* (сайт проекта : <http://www.qcustomplot.com/>, лицензия GPL). Использование этого проекта не требует компиляции библиотеки — просто добавьте файлы *qcustomplot.h* и *qcustomplot.cpp* в проект. Обратите внимание на документацию и примеры, которые поставляются вместе с проектом.

## Приложение А

### Использование компилятора командной строки и текстового редактора Geany

Одним из самых мощных современных кроссплатформенных компиляторов языка C++ является свободно-распространяемый компилятор g++. Для его установки в debian-подобных ОС семейства Linux (Debian, Ubuntu, Mint и их клоны) достаточно выполнить команду терминала `apt-get install g++` с правами суперпользователя (администратора) или воспользоваться менеджером пакетов `synaptic`.

Установка компилятора g++ в ОС семейства Windows несколько сложнее, поэтому рассмотрим этот процесс более подробно.

1. На сайте [mingw.org](http://mingw.org) переходим в раздел **Download** (Загрузки), в появившемся окне щелкаем по ссылке `Download mingw-get-setup.exe` (рис. А.1). Загружаем установочный файл и запускаем его.
2. На первом этапе установки необходимо выбрать команду **Install** (см. рис. А.2), затем папку для установки (рис. А.3).
3. После этого начнётся процесс доустановки инсталлятора (рис. А.4).
4. Далее выбираем компиляторы для установки (**C++, Fortran, Ada**) (рис. А.5) и ждем пока будут скачаны и установлены необходимые компиляторы. После завершения процесса установки компиляторы (в нашем случае компилятор C++) готовы к использованию. Но для вызова их (из командной строки Windows, из текстового редактора Geany) необходимо указывать полное имя файла с компилятором. При установке компилятора C++ в стандартный каталог `C:\MinGW` — полное имя компилятора C++ будет таким `C:\MinGW\bin\g++`. Для того, чтобы каждый раз не писать полное имя компиляторы, можно добавить путь `C:\MinGW\bin` в список путей системной переменной `Path`.
5. Для изменения значения системы переменной `Path` необходимо в панели управления выбрать *Система -> Дополнительные параметры системы -> Дополнительно -> Переменные среды -> Системные переменные*

-> *Path* -> *Изменить*. В открывшемся диалоговом окне добавить путь `C:\MinGW\bin\`.

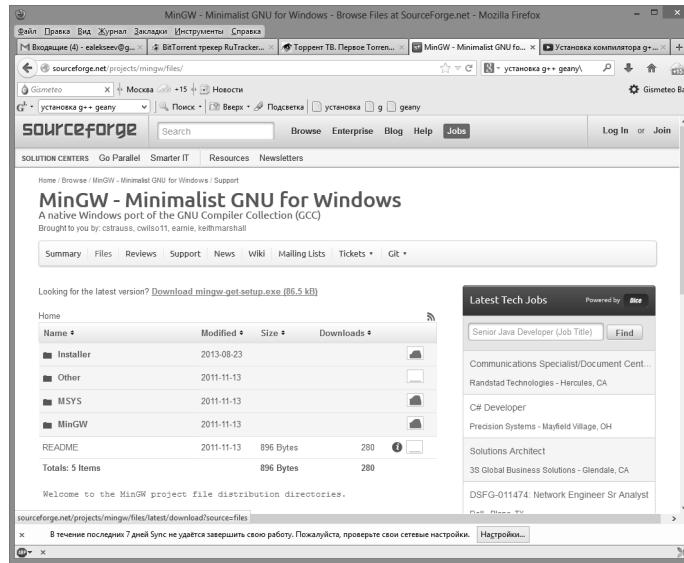


Рис. А.1: Окно загрузки `mingw`.

После перезагрузки ОС Windows для обращения к компилятору достаточно будет указывать его имя — `g++`.

Таким образом, в ОС Linux для работы с компилятором в командной строке необходимо запустить Терминал, а в ОС Windows — командную строку. После чего работа с компилятором `g++` с ОС Windows и Linux идентична.

Рассмотрим опции компилятора командной строки, необходимые для компиляции и запуска простейших программ.

Для того, чтобы создать исполняемый файл из текста программы на C++, необходимо выполнить команду

```
g++ name.cpp
```

Здесь `name.cpp` — имя файла с текстом программы. В результате будет создан исполняемый файл со стандартным именем `a.out`. Для того, чтобы создать исполняемый файл с другим именем, необходимо выполнить команду

```
g++ -o nameout name.cpp
```

Здесь `name.cpp` — имя файла с текстом программы, `nameout` — имя исполняемого файла.

При использовании компилятора `g++` после компиляции программы автоматически происходит компоновка программы (запуск компоновщика `make`). Чтобы

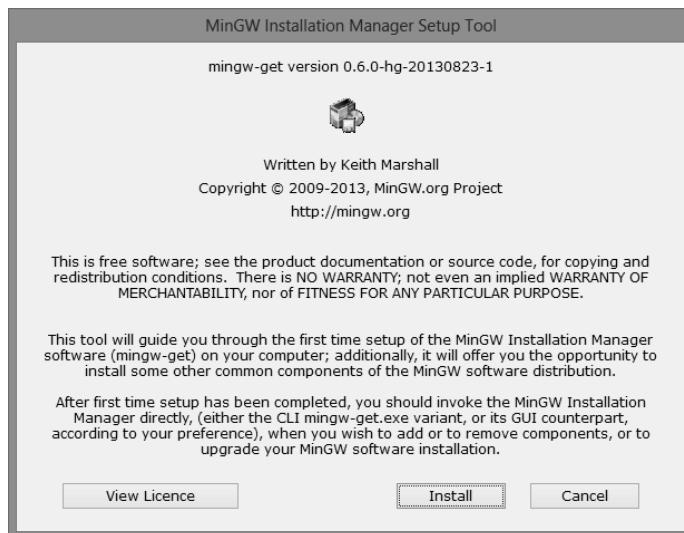


Рис. А.2: Первое окно установки mingw.



Рис. А.3: Выбор папки для установка компилятора.

исключить автоматическую компоновку программы следует использовать опцию `-c`. В этом случае команда будет иметь вид `g++ -c name.cpp`

Технология работы с компилятором `g++` может быть такой: набираем текст программы в стандартном текстовом редакторе, потом в консоли запускаем компилятор, после исправления синтаксических ошибок, запускаем исполняемый файл. После каждого изменения текста программы, надо сохранить изменения в

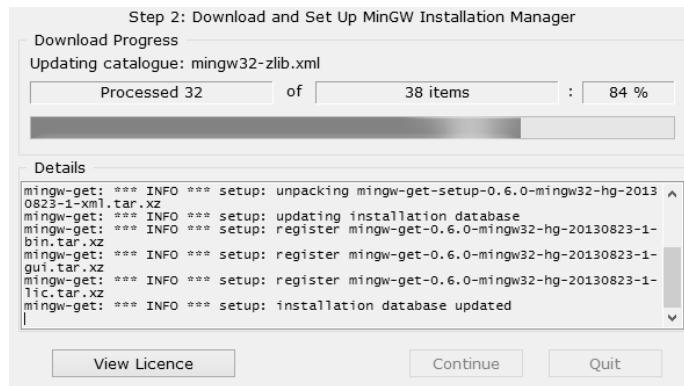


Рис. А.4: Загрузка инсталлятора.

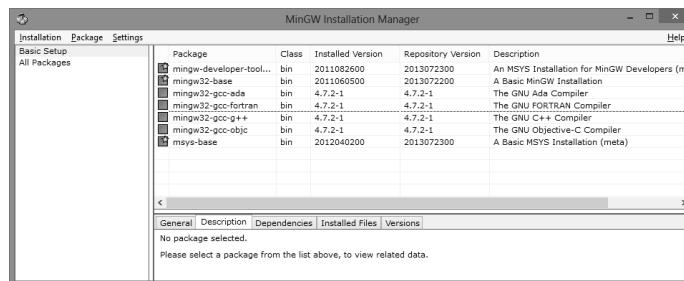


Рис. А.5: Выбор устанавливаемых компиляторов.

файле на диске, запустить компилятор, и только после этого запускать программу (исполняемый файл). Очень важно не забывать сохранять текст программы, иначе при запуске компилятора будет компилироваться старая версия текста программы.

Компилятор `g++` эффективен при разработке больших комплексов программ, он позволяет собирать приложения из нескольких файлов, создавать библиотеки программ. Рассмотрим процесс создания и использования библиотеки решения задач линейной алгебры (см. п. 6.4, задачи 6.10 – 6.12):

`int SLAU(double **matrica_a, int n, double *massiv_b, double *x)` — функция решения системы линейных алгебраических уравнений;

`int INVERSE(double **a, int n, double **y)` — функция вычисления обратной матрицы;

`double determinant(double **matrica_a, int n)` — функция вычисления определителя.

Для создания библиотеки создадим заголовочный файл **slau.h** и файл **slau.cpp**, в который поместим тексты всех трёх функций решения задач линейной алгебры.

Текст файла **slau1.h**:

```
int SLAU(double **matrica_a, int n, double *massiv_b, double **x);
int INVERSE(double **a, int n, double **y);
double determinant(double **matrica_a, int n);
```

Текст файла **slau1.cpp**:

```
#include <math.h>
int SLAU(double **matrica_a, int n, double *massiv_b, double **x)
{
    int i, j, k, r;
    double c, M, max, s;
    double **a, *b;
    a=new double *[n];
    for(i=0;i<n;i++)
        a[i]=new double[n];
    b=new double[n];
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            a[i][j]=matrica_a[i][j];
    for(i=0;i<n;i++)
        b[i]=massiv_b[i];
    for(k=0;k<n;k++)
    {
        max=fabs(a[k][k]);
        r=k;
        for(i=k+1;i<n;i++)
            if(fabs(a[i][k])>max)
            {
                max=fabs(a[i][k]);
                r=i;
            }
        for(j=0;j<n;j++)
        {
            c=a[k][j];
            a[k][j]=a[r][j];
            a[r][j]=c;
        }
        c=b[k];
        b[k]=b[r];
        b[r]=c;
        for(i=k+1;i<n;i++)
        {
            for(M=a[i][k]/a[k][k], j=k; j<n; j++)
                a[i][j]-=M*a[k][j];
            b[i]-=M*b[k];
        }
    }
    if(a[n-1][n-1]==0)
        if(b[n-1]==0)
            return -1;
        else return -2;
    else
    {
        for(i=n-1;i>=0;i--)
        {
            for(s=0,j=i+1;j<n;j++)
                s+=a[i][j]*x[j];
            x[i]=(b[i]-s)/a[i][i];
        }
    }
    return 0;
}
```

```

        }
    }

int INVERSE(double **a, int n, double **y)
{
    int i,j,res;
    double *b, *x;
    b=new double [n];
    x=new double [n];
    for (i=0;i<n; i++)
    {
        for (j=0;j<n; j++)
            if (j==i)
                b[j]=1;
            else b[j]=0;
            res=SLAU(a,n,b,x);
            if (res!=0)
                break;
            else
                for (j=0;j<n; j++)
                    y[j][i]=x[j];
    }
    if (res!=0)
        return -1;
    else
        return 0;
}
double determinant(double **matrica_a, int n)
{
    int i,j,k,r;
    double c,M,max,s,det=1;
    double **a;
    a=new double *[n];
    for (i=0;i<n; i++)
        a[i]=new double [n];
    for (i=0;i<n; i++)
        for (j=0;j<n; j++)
            a[i][j]=matrica_a[i][j];
    for (k=0;k<n; k++)
    {
        max=fabs(a[k][k]);
        r=k;
        for (i=k+1;i<n; i++)
            if (fabs(a[i][k])>max)
            {
                max=fabs(a[i][k]);
                r=i;
            }
            if (r!=k) det=-det;
        for (j=0;j<n; j++)
        {
            c=a[k][j];
            a[k][j]=a[r][j];
            a[r][j]=c;
        }
        for (i=k+1;i<n; i++)
            for (M=a[i][k]/a[k][k], j=k; j<n; j++)
                a[i][j]-=M*a[k][j];
    }
    for (i=0;i<n; i++)
        det*=a[i][i];
    return det;
    for (i=0;i<n; i++)
        delete [] a[i];
    delete [] a;
}

```

В качестве тестовой задачи напишем главную функцию, которая предназначена для решения системы линейных алгебраических уравнений.

```
#include <iostream>
#include <math.h>
//Подключение личной библиотеки slau
#include "slau1.h"
using namespace std;
int main()
{
    int result , i , j , N;
    double **a , *b , *x;
    //Ввод размерности системы.
    cout<<"N=" ;
    cin>>N;
    //Выделение памяти для матрицы правых частей и вектора свободных членов.
    a=new double *[N];
    for ( i=0; i<N; i++)
        a[ i]=new double [N];
    b=new double [N];
    x=new double [N];
    //Ввод матрицы правых частей и вектора свободных членов .
    cout<<"Input Matrix A "<<endl;
    for ( i=0; i<N; i++)
        for ( j=0; j<N; j++)
            cin>>a[ i ][ j ];
    cout<<"Input massiv B "<<endl;
    for ( i=0; i<N; i++)
        cin>>b[ i ];
    //Вызов функции решения СЛАУ методом Гаусса из библиотеки slau.h
    result=SLAU(a , N, b , x );
    if ( result==0)
    {
        //Вывод массива решения.
        cout<<"Massiv X "<<endl;
        for ( i=0; i<N; i++)
            cout<<x[ i]<<"\t";
        cout<<endl;
    }
    else if ( result== -1)
        cout<<"Бесконечное множество решений\n ";
    else if ( result== -2)
        cout<<"Нет решений\n ";
}
}
```

Теперь необходимо из этих текстов создать работающее приложение. Рассмотрим это поэтапно.

1. Компиляция библиотеки `slau1.h` с помощью команды `g++ -c slau1.cpp`.
2. Компиляция главной функции `main.cpp` с помощью команды `g++ -c main.cpp`.
3. Создание исполняемого файла с именем `primer` из двух откомпилированных файлов `main.o` и `slau1.o` с помощью команды `g++ main.o slau1.o -o primer`.
4. Запуск исполняемого файла.

После разработки библиотеки линейной алгебры пример `slau1`, можно использовать её в различных программах при вычислении определителя, обратной матрицы и решения систем линейных алгебраических уравнений.

При разработки программ с большим количеством вычислений, компилятор `g++` позволяет оптимизировать программы по быстродействию. Для получения оптимизированных программ можно использовать ключи `-O0`, `-O1`, `-O2`, `-O3`, `-Os`:

- при использовании ключа `-O0` оптимизация отключена, достигается максимальная скорость компиляции, опция задействована по умолчанию;
- при использовании ключа «мягкой» оптимизации `-O1` происходит некоторое увеличение времени компиляции, этот ключ оптимизации позволяет одновременно уменьшать занимаемую программой память и уменьшить время выполнения программы;
- при использовании ключа `-O2` происходит существенное уменьшение времени работы программы, при этом не происходит увеличение памяти занимаемой программой, не происходит развертка циклов и автоматическое встраивание функций;
- ключ «агрессивной» оптимизации `-O3` нацелен в первую очередь на уменьшение времени выполнения программы, при этом может произойти увеличение объёма кода и времени компиляции, в этом случае происходит развертка циклов и автоматическое встраивание функций;
- ключ `-Os` ориентирован на оптимизацию размера программы, включаются те опции из набора `-O2`, которые обычно не увеличивают объём кода, применяются некоторые другие оптимизации, направленные на снижение его объёма.

Для разработки программ на различных языках программирования можно использовать текстовый редактор `Geany`. Редактор `Geany` входит в репозитории большинства дистрибутивов Linux, его установка осуществляется стандартным для вашего дистрибутива образом (в debian-подобных ОС с помощью команды `apt-get install geany`). Для установки его в Windows необходимо скачать со страницы <http://www.geany.org/Download/Releases> инсталляционный файл и установить программу стандартным способом.

Разработка программ с использованием `Geany` более эффективна. Окно `Geany` представлено на рис. А.6.

Последовательно рассмотрим основные этапы разработки программы с использованием `Geany`.

1. Необходимо создать шаблон приложения на C/C++ (или другом языке программирования) с помощью команды *Файл -> Создать из шаблона -> main.cxx*. После чего необходимо ввести текст программы и сохранить его.
2. Для компиляции и запуска программы на выполнение служит пункт меню *Сборка*. Для компиляции программы следует использовать команду *Сборка -> Скомпилировать* (F8). В этом случае будет построен объектный код программы (файл с расширением .o или .obj). Для создания исполняемого кода программы служит команда *Сборка -> Собрать* (Shift+F9). Для запуска программы следует выполнить команду *Сборка -> Выполнить* (F5).

Параметры компилятора определяются автоматически после выбора шаблона (*Файл -> Создать из шаблона*). Однако, команды компиляции и сборки по

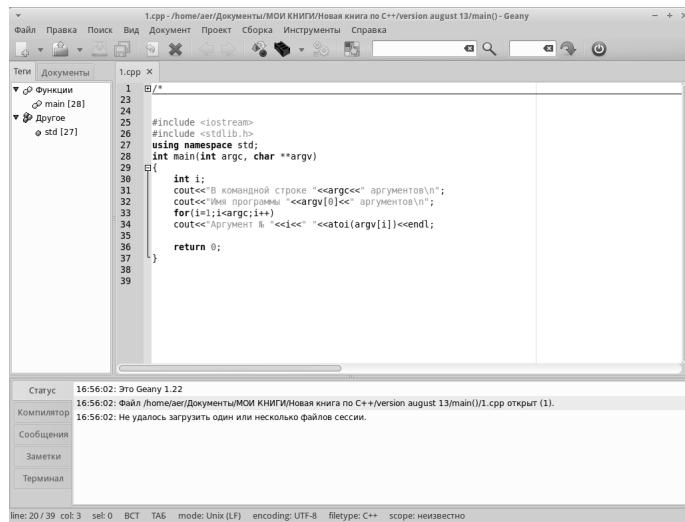


Рис. А.6: Окно Geany.

умолчанию можно изменить, используя команду *Сборка -> Установить параметры сборки* (см. рис. А.7). Здесь %f — имя компилируемого файла, %e — имя файла без расширения.



Рис. А.7: Настройка компиляции программ на языке С++ в Geany.

## Сведения об авторах

**Алексеев Евгений Ростиславович** — кандидат технических наук, доцент, профессор кафедры вычислительной математики и программирования Донецкого национального технического университета, специалист в области программирования, вычислительной математики, свободного программного обеспечения. Преподавательский стаж Алексеева Е.Р. Более 20 лет.

Евгений Ростиславович — автор пятнадцати книг, выпущенных в Москве и Донецке, общий тираж которых превышает 65 тыс. экземпляров. Е.Р. Алексеев — автор более 100 научных и методических работ. Область его научных интересов — программирование, вычислительная математика, Интернет-технологии, использование свободно распространяемого программного обеспечения.

**Чеснокова Оксана Витальевна** — старший преподаватель кафедры «Вычислительная математика и программирование» Донецкого национального технического университета. Преподавательский стаж Чесноковой О.В. — 18 лет. Оксана Витальевна — автор десяти книг, общий тираж которых превышает 40 тыс. экземпляров, а также более 50 научных и методических работ. Специалист в области программирования и вычислительной математики.

**Злобин Григорий Григорьевич** — кандидат технических наук, доцент кафедры радиофизики и компьютерных технологий Львовского национального университета имени Ивана Франко, автор 13 книг, изданных во Львове и Киеве общим тиражом около 12 тыс. экземпляров, автор более 180 научных статей и докладов на конференциях, специалист в области автоматизации эксперимента, программирования и использования свободного программного обеспечения.

**Чмыхало Александр Сергеевич** — выпускник факультета электроники Львовского национального университета им. И. Франко. Разработчик программного обеспечения и сертифицированный Qt специалист.

**Костюк Дмитрий Александрович** — кандидат технических наук, доцент кафедры электронных вычислительных машин и систем Брестского государственного технического университета, специалист в области системного и прикладного программирования, свободного программного обеспечения. Преподавательский стаж Костюка Д.А. около 15 лет. Автор более 180 научных публикаций.

# Список литературы

- [1] Алексеев Е.Р. Программирование на Microsoft Visual C++ и Turbo C++ Explorer. — М.: НТ Пресс, 2007. – 352 с.
- [2] Алексеев Е.Р., Чеснокова О.В, Кучер Т.В. Самоучитель по программированию на Free Pascal и Lazarus. Донецк: Унитех, 2009. – 502 с.
- [3] Б.П. Демидович, И.А. Марон. Основы вычислительной математики. — М.: Наука, 1966. – 664 с.
- [4] Керниган Б., Ритчи Д. Язык программирования Си. — М.: Вильямс, 2013. – 304 с.
- [5] Лаптев В.В. C++. Объектно-ориентированное программирование. — СПб.: Питер, 2008. – 464 с.
- [6] Лаптев В.В. Морозов А.В., Бокова. C++. Объектно-ориентированное программирование. Задачи и упражнения. — СПб.: Питер, 2007. – 288 с.
- [7] Подбелльский В.В. Язык C++. — М.: Финансы и статистика, 2001. – 560 с.
- [8] Шиманович Е.Л. С/C++ в примерах и задачах. — Минск: Новое знание, 2004. – 528 с.

# Предметный указатель

- Valgrind, 385
- Алгоритм, 45, 330  
линейный, 46  
основные конструкции, 46  
Поиск максимального (минимального) элемента массива и номера, 146  
Произведение элементов массива, 144  
разветвляющийся, 46  
Сортировка элементов массива, 162  
Сумма элементов массива, 143  
Удаление элемента из массива, 156  
цикл с параметром, 68  
цикл с постусловием, 67  
цикл с предусловием, 66  
циклический, 46, 65
- Библиотека  
complex, 262  
iostream, 8, 262  
math.h, 17, 262  
комплексных чисел, 262  
операции с массивами, 264  
определение переменной, 262  
организация ввода-вывода, 262  
основные функции, 263
- Блок-схема, 45
- Виджет  
позиция и размер, 369
- Виджеты, 335, 360
- Виджеты верхнего уровня, 362
- Виджеты-контейнеры, 376
- Виртуальный метод, 303
- Выражение, 24
- Горячие клавиши  
**Qt Creator**, 340
- Динамически распределяемая память, 385
- Директивы, 34
- Дочерние виджеты, 361
- Запись файла с использованием **Qt**, 358
- Идентификатор, 18
- Инструменты Qt
- Qt Assistant**, 338  
**Qt Creator**, 339  
qmake, 340, 347  
**Qt Designer**, 343
- Интерпретатор, 10
- Исключение, 312  
индикатор, 312  
обработчик, 312
- Итератор, 328
- Класс, 275  
**QMainWindow**, 343  
**QAbstractButton**, 376  
**QAction**, 400, 403, 407  
**QApplication**, 360, 411  
**QCheckBox**, 372, 376  
**QCloseEvent**, 388  
**QColor**, 397  
**QColorDialog**, 408  
**QComboBox**, 377  
**QDateTimeEdit**, 377  
**QDial**, 377  
**QDialog**, 399, 415  
**QDockWidget**, 400  
**QEvent**, 388, 389  
**QFile**, 356  
**QFileDialog**, 408  
**QFlags**, 357  
**QFont**, 361  
**QFontDialog**, 408  
**QFrame**, 376  
**QGridLayout**, 364, 366, 367  
**QGroupBox**, 377  
**QHash**, 354, 356  
**QHBoxLayout**, 364  
**QIcon**, 413  
**QInputDialog**, 408  
**QIODevice**, 356  
**QKeyEvent**, 388  
**QKeySequence**, 406  
**QLabel**, 360, 377  
**QLCDNumber**, 377  
**QLineEdit**, 377

- QLinkedList, 356  
QList, 354, 356  
QMainWindow, 399  
 QMap, 354, 356  
 QMessageBox, 408, 411  
 QMouseEvent, 388  
 QMultiHash, 356  
 QMultiMap, 356  
 QObject, 379, 384  
 QPaintDevice, 396  
 QPainter, 396  
 QPainterPath, 396  
 QPaintEvent, 388  
 QPen, 397  
 QPixmap, 382, 384  
 QPoint, 396  
 QProgressBar, 377  
 QPushButton, 376  
 QQueue, 356  
 QRadioButton, 376  
 QRect, 397  
 QResizeEvent, 388  
 QScrollBar, 377  
 QSet, 356  
 QSettings, 418  
 QSignalMapper, 374  
 QSize, 368, 369, 397  
 QSlider, 378  
 QStack, 356  
 QStatusBar, 400  
 QString, 352  
 QStringList, 353  
 QTabWidget, 377  
 QTextStream, 358  
 QTimerEvent, 388  
 QToolBox, 377  
 QToolButton, 376  
 QVBoxLayout, 364  
 QVector, 356  
 QWidget, 360, 376, 396, 399  
 абстрактный, 306  
 базовый, 276  
 деструктор, 280  
 дочерний, 276  
 конструктор, 280  
 метод, 278  
 модификаторы доступа, 300  
 модификаторы наследования, 300  
 наследование, 276, 299  
 переменная-член класса, 277  
 производный, 276  
 родительский, 276  
 свойство, 277  
 шаблон класса, 322  
 Ключевые слова, 18  
 Комментарии, 18  
 Компилятор, 10  
 Компиляция проекта, 347  
 Комплект (Kit), 339  
 Компоновщик, 364  
 Консольное приложение  
     запуск, 13  
     создание, 13  
 Консольный проект, 350  
 Константа, 22  
     описание, 22  
 Конструктор  
     копирования, 293  
 Контеинер, 327  
 Контеинерные классы Qt, 353  
 Кроссплатформенная программа, 335  
 Макрос  
     foreach, 355  
     Q\_PROPERTY, 393  
     QT\_NO\_DEBUG\_OUTPUT, 351  
     QT\_NO\_WARNING\_OUTPUT, 351  
 Манипулятор, 234  
     endl, 234  
 Массив, 22, 135  
 Метаобъект, 379  
 Метод Гаусса  
     вычисление обратной матрицы, 214  
     вычисление определителя, 218  
     решение систем линейных уравнений, 206  
 Модальность, 399  
 Модули Qt  
     дополнительные (Add-On), 337  
     основные (Essentials), 336  
 Наследование  
     множественное, 304  
 Настройка  
     Shadow Build, 349  
     версии Qt, 340  
     компиллятора, 340  
     комплектов, 340  
     очистки проекта, 349  
 Объект, 274  
     инкапсуляция, 275  
     полиморфизм, 276  
 Объектная иерархия, 384  
 Операнд, 24  
 Оператор  
     delete, 140  
     new, 139  
     ввода, 232  
     вывода, 232  
     разветвляющийся, 47  
     составной, 47  
     условный, 47

- цикл с параметром, 68
- цикл с постусловием, 67
- цикл с предусловием, 66
- циклический, 65
- Операции, 24
  - арифметические, 26, 31
  - бинарные, 24
  - битовой арифметики, 27
  - декремента, 27
  - инкремента, 27
  - логические, 29
  - множественного присваивания, 26
  - отношения, 29
  - получение адреса, 30
  - преобразования типа, 30
  - присваивания, 25, 31
  - разадресации, 30
  - составного присваивания, 26
  - унарные, 24
  - условная, 29
  - целочисленной арифметики, 27
- Панель
  - присоединяемая, 400
  - статуса, 400
- Перегрузка операторов, 285
- Передача параметров, 105
  - по адресу, 105
  - по значению, 105
- Переменная, 19
  - глобальная, 35, 104
  - значение, 19
  - имя, 19
  - локальная, 35, 104
  - описание, 19
- Переменные qmake
  - CONFIG, 347
  - DEFINES, 347, 411
  - DESTDIR, 347
  - FORMS, 347
  - HEADERS, 347
  - INCLUDEPATH, 347
  - LIBS, 347, 421
  - PWD, 421
  - QT, 347
  - RESOURCES, 347
  - SOURCES, 346, 347
  - TARGET, 347
  - TEMPLATE, 347
- Перечисления
  - QSizePolicy::Policy, 368
  - Qt::ConnectionType, 371
- Подпрограмма, 101
- Позднее связывание, 303
- Позиция и размер виджета, 361
- Политики размера (size policies), 368
- Потоки, 236
- Программа, 34
  - структура, 34
- Программная платформа, 335
- Расширение файла
  - .pro, 345
- Редактор форм, 401
- Рекурсия, 122
- Ресурсы, 412
- Родительский виджет, 361, 384
- Свойства, 393
- Сигналы, 371, 373, 375
- Сигнально-слотовые соединения, 370
  - автоматические, 375
- Слоты, 371, 373, 375
  - creation, 373, 395
- События, 387
- Создание
  - Класса формы Qt Designer, 414
  - нового класса, 362
  - оконного проекта, 341
  - проекта с графическим интерфейсом, 364
  - проектного файла, 345
  - пустого проекта, 345, 360
  - файла исходных текстов, 346
- Среда программирования Qt Creator, 11
- Строка, 23, 248
- Строки
  - ввод, 249
  - операции, 252
  - функции, 252
  - функции обработки, 249
- Структура, 23
- Структура программы, 10
- Структуры, 255
- Типы данных, 19
  - вещественный, 21
  - логический, 21
  - основные, 19
  - символьный, 20
  - составные, 19
  - структурированные, 22
  - целый, 20
- Транслятор, 10
- Указатель, 23
  - вычитание, 31
  - декремент, 31
  - инкремент, 31
  - получение адреса, 30
  - присваивание, 31
  - разадресация, 30
  - скрытый, 294
  - сложение с константой, 31
- Утечки памяти, 385

**Файл**

.ui, 405  
make-файл, 347  
двоичный, 242  
закрыть, 243  
запись, 236, 237, 242, 243  
открыть, 237, 242  
переименование, 243  
последовательный доступ, 244  
проекта, 345  
ресурсов, 412  
текстовый, 236  
удаление, 243  
формы, 343, 400, 405  
чтение, 236, 242, 243

**Фильтр событий**, 389, 392**Флаги**

Qt::Allignment, 368  
Qt::WindowFlags, 399

**Функции**, 32

ввод данных, 36  
вывод данных, 36  
стандартные, 32

**Функция**, 101

calloc, 138  
main, 10  
malloc, 138  
qCritical, 351  
qDebug, 350, 351  
qFatal, 351  
qWarning, 351  
realloc, 139  
sqrt(x), 17  
возврат результата, 104, 107  
вызов, 103  
заголовок, 102  
механизм передачи параметров, 105  
описание, 101  
перегрузка имени, 124  
прототип, 103  
рекурсивная, 122  
тело функции, 102  
фактические параметры, 105  
формальные параметры, 105  
шаблон, 126

**Цикл обработки событий**, 387

Чисто виртуальный метод, 306  
Чтение файла с использованием Qt, 357,  
358

*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>