ВАЛЕРИЙ ЛАПТЕВ

bhv

# C++

## ЭКСПРЕСС-КУРС

- Синтаксис и семантика конструкций C++
- Основы стандартной библиотеки шаблонов
- Классы, наследование, шаблоны, исключения
- "Подводные камни" программирования
- Основы программирования в Windows

**Валерий Лаптев**

# C++

## ЭКСПРЕСС-КУРС

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1
Л24

**Лаптев В. В.**

Л24 С++. Экспресс-курс. — СПб.: БХВ-Петербург, 2004. — 512 с.: ил.

ISBN 5-94157-358-8

Книга представляет собой руководство по программированию на С++, позволяющее быстро освоиться в данном алгоритмическом языке, и включает как необходимый теоретический материал, так и реализации задуманных программ в виде листингов, поясняющих рисунков, таблиц. Начав с изучения основ языка, читатель знакомится с принципами перехода от формального словесного описания задачи к описанию, понятному для ПК и позволяющему решить ее за короткое время, постепенно осваивает все более сложные конструкции, учится сам использовать богатый арсенал С++. Приводятся примеры не только работающих, "отлаженных" программ, но и наиболее вероятных ошибок, возникающих в процессе написания программы и не всегда распознаваемых компилятором. Рассматриваемые встроенные функции, библиотеки дают возможность при правильном подключении уже готовых функций, макросов значительно сократить программный код.

*Для начинающих программистов*

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Анатолий Адаменко* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Елена Яковлева* |
| Компьютерная верстка | *Ольги Сергиенко* |
| Корректор | *Зинаида Дмитриева* |
| Дизайн обложки | *Игоря Цырульникова* |
| Зав. производством | *Николай Тверских* |

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 02.12.03.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 41,28.
Тираж 5000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-358-8

# Содержание

# Введение

По шутливому замечанию Джеффа Элджера [49], "на рынке продается по крайней мере 2 768 942 книги о C++". И все же вы держите в руках еще одну — 2 768 943-ю книгу о том же самом. Зачем нужна еще одна книга о C++?

Когда я стал преподавать программирование на языке C++ для первокурсников, мне потребовалось выбрать базовый учебник, по которому студенты станут изучать C++ и учиться программировать на этом языке. Перерыв "гору" литературы, я с удивлением констатировал, что нужной мне книги нет. По той или иной причине ни одна из них меня не устроила. Оказалось, что некоторые очень важные свойства языка часто либо вовсе не описаны, либо рассматриваются на элементарном уровне. Наверное, лучшим выбором была бы книга Стенли Липпмана "C++ для начинающих", однако в настоящее время ее просто не найти.

Все книги о C++ можно разделить на несколько категорий: о самом языке C++, об использовании конкретной среды программирования вроде Borland C++ Builder или Visual C++, и о типовых структурах данных и алгоритмах. Однако по собственному опыту знаю, что этого недостаточно, чтобы научиться программировать на C++. В жизни программисту приходится сталкиваться с целым спектром чисто практических проблем, которые по непонятным мне причинам практически ни в одной книге не излагаются.

Во-первых, как я неоднократно ранее убеждался на собственном опыте и еще больше уверился на опыте моих студентов, изучить язык программирования в отрыве от среды программирования нельзя. Можно прекрасно выучить все нюансы C++, но быть совершенно беспомощным при написании конкретной программы в определенной системе. Обучение всегда должно происходить в некоторой среде программирования, да и работать всегда приходится в реальной системе, которая имеет те или иные особенности и недоработки.

Во-вторых, программировать, не обращая внимания на операционную систему и аппаратную платформу, тоже невозможно — хотя бы потому, что реализация встроенных типов данных зависит от аппаратуры. И тут существует множество "подводных камней", на которые постоянно "натыкается" начинающий программист. Ввод/вывод в любом языке программирования всегда зависит от файловой системы ОС. Недаром, чтобы уменьшить эту зависимость, ввод/вывод в C++ полностью вынесен в отдельные библиотеки.

И наконец, проблема русских букв. Естественно, ни одна из книг иностранных авторов, многие из которых просто замечательны и мне очень нравятся, эту проблему не затрагивает. Удивительно, но и в книгах российских авторов тоже нет об этом ни слова! Между тем, это серьезная проблема для начинающего программиста, требующая определенных знаний и навыков.

Такой же значимой проблемой является переход от консольных приложений к оконным. Помню, как мучительно долго я, опытный программист, разбирался в идеологии и особенностях программирования для Windows. Угнетало количество технических деталей и отсутствие четких ориентиров, на что в первую очередь обращать внимание. Тем более такой переход сложен для новичка, несмотря на то, что сейчас существует достаточно много отличных книг, обучающих программированию для Windows.

Однако в экспресс-курсе невозможно изложить все. В процессе работы над книгой мне часто приходилось решать, что оставить, а что выбросить. Должен признать, что отбор материала для изложения оказался большой проблемой — язык C++ очень обширен. Поэтому, может быть, не все читатели найдут в книге то, что хотели найти. Например, в *гл. 4* об указателях я решил не упоминать о функции `malloc` и ее "родственниках" из C, а в *гл. 6* о вводе/выводе ни слова нет о библиотеке stdio.h. Не упоминается и о битовых полях в структуре и совсем мало о битовых операциях. По некоторым темам, только затронутым в книге, можно было бы написать отдельную книгу, в два раза большую по объему (возможно, мне удастся сделать это в будущем). А вот о программировании функций, наоборот, написано значительно больше, чем это обычно принято. Я убежден, что без фундаментальных знаний о функциях невозможно программировать. Довольно много внимания уделено и различным проблемам, возникающим в реальном программировании. Достаточно подробно описана проблема "кириллизации" строк, так важных для начинающих программистов. К сожалению, объем книги не позволял изложить стандартные средства локализации в достаточном объеме, поэтому я решил вообще не касаться этой темы.

В итоге получилась книга, которую вы держите в руках. О ее достоинствах и недостатках — судить вам.

# Кому адресована эта книга

Книга представляет собой практическое введение в программирование на C++ в распространенных в России средах фирмы Borland и Visual C++. Книга написана, в первую очередь, для моих студентов — начинающих программистов, которые хотели бы научиться программировать на C++. Желательно (но не обязательно), чтобы имелся опыт программирования (хотя бы учебных программ) на другом языке. Однако и более опытные программисты, надеюсь, найдут в книге немало интересного.

# Структура книги

Книга состоит из 4-х частей. В *части I* излагаются основы языка C++. В *гл. 1* описываются основные операторы, встроенные типы данных и операции. Рассматриваются типичные ошибки и проблемы, возникающие, например, при переполнении или неопределенном поведении программы.

*Гл. 2* посвящена функциям. Описываются способы передачи параметров по значению и по ссылке, константные параметры, параметры по умолчанию. Уделяется внимание некоторым проблемам, связанным с областью видимости переменных. Рассматривается перегрузка и шаблоны функций, основы использования препроцессора.

В *гл. 3* описаны массивы и структуры, приведены многочисленные примеры обработки массивов, в том числе и символьных. Рассматриваются "взаимоотношения" массивов и структур, разбираются основы шаблонов структур.

Без сомнения самой важной темой C++ являются указатели и динамическая память. Об этом довольно подробно повествует *гл. 4*.

В *гл. 5* излагается элементарное введение в библиотеку стандартных шаблонов. Достаточно много внимания уделено строкам.

В *гл. 6* на примере программирования небольшой автоматизированной системы тестирования описываются основы системы ввода/вывода C++.

*Гл. 7* возвращает нас к функциям. Здесь излагаются более сложные вопросы: рекурсия, функции с переменным числом параметров, так называемые "левые" функции.

В *части II* излагаются основы объектно-ориентированного и обобщенного программирования на C++. Объясняется реализация некоторых простых паттернов проектирования. В *гл. 8* подробно разбираются вопросы перегрузки операций, определяется понятие конструктора, реализуется почти законченный класс рациональных чисел.

*Гл. 9* посвящена программированию динамических классов. На примере динамических массивов и списков разбираются проблемы, возникающие при

программировании классов с динамическим распределением памяти. Разбирается понятие интеллектуального указателя.

*Гл. 10* содержит то, что начинающему программисту необходимо знать об исключениях.

В *гл. 11* рассматривается один из трех краеугольных камней объектно-ориентированного программирования — наследование, и связанные с ним вопросы: виртуальные функции и RTTI. Попутно разбирается организация многомодульных программ.

В *гл. 12* излагаются некоторые особенности STL, рассматривается программирование шаблонов и функциональных объектов.

*Часть III* показывает переход к программированию для Windows. В *гл. 13* рассматривается использование Win32 API для создания оконных приложений.

В *гл. 14* приведен пример практической разработки приложения с использованием C++ Builder.

В *приложениях (часть IV)* рассмотрены характеристики систем программирования, приведены ссылки для поиска дополнительной информации в Интернете, а также составлен перечень литературных источников, знакомство с которыми расширит ваш кругозор.

## Используемые программные продукты

В *приложениях 1, 2* приводятся краткие сведения о нескольких системах. Почти все примеры были реально проверены в лицензионной системе Microsoft Visual C++ 6, в которой поставляемая STL от DinkumWare была заменена на STLport версии 4.5.3. Отдельные примеры для сравнения проверялись в нелицензионной версии Visual C++ 7. Однако, по глубокому убеждению автора, квалифицированный программист не должен испытывать затруднений при работе в нескольких разных системах. Поэтому в качестве альтернативы были использованы столь любимые российскими программистами системы фирмы Borland: Borland C++ 3.1, Borland C++ 5, Borland C++ Builder 6. Выбраны системы Borland, как наиболее известные в России. Кроме того, у всех "борландовских" систем прекрасная система встроенной помощи, тогда как Visual C++ 6 требует отдельной установки MSDN.

Может вызвать некоторое недоумение упоминание системы Borland C++ 3.1. Я использую эту версию по следующим причинам.

- Если мы начинаем работать в Borland C++ 3.1, то отсутствует проблема с русскими буквами — все сообщения выводятся именно в том "русском виде", как программист набрал их в редакторе. Практически все примеры первых трех глав проверены именно в этой системе.

- Borland C++ 3.1 абсолютно совместима с Borland (Turbo) Pascal 7 по внешнему виду, "горячим клавишам", составу подсистем и т. д. Таким образом, "паскалистам", начинающим изучать C++ (а это практически все студенты 1-го или 2-го курса почти всех университетов России), будет значительно проще освоиться с новым языком.
- После версии 3.1 легко перейти к другим, более мощным "борландовским" системам, т. к. состав клавишных команд остается практически неизменным.
- Borland C++ 3.1 до сих пор реально используется в промышленном программировании. Система достаточно мощная — в ней нет только стандартной библиотеки STL и исключений — это появилось позже. Остальное все есть. Мой друг, работающий в Санкт-Петербурге в одной серьезной организации, написал мне в ответ на вопрос об используемых системах программирования: "Мы пишем все на строгом ANSI C... Мы применяем компилятор Борланд 3.1 и 4.5, еще MS 4.1. Проверены. Я лично ЗАПРЕТИЛ применения Борланда 5 и выше, когда исследовал несколько объектных файлов". Да и в форумах **www.rsdn.ru** частенько попадаются вопросы об этой системе.
- На командных чемпионатах мира по программированию среди студентов, ежегодно проводимых под эгидой IBM, по правилам соревнований допускается использование только двух систем: Borland (Turbo) Pascal 7 и Borland C++ 3.1. Поэтому практически во всех университетах России (и мира тоже) обязательно их изучают как первые системы программирования.
- Еще одна причина — систему легко найти, она бесплатно передается из рук в руки, в отличие от более поздних систем.

Операционной системой, естественно, выбрана Windows, т. к. все среды программирования работают только в ней.

# Благодарности

В первую очередь хотел бы выразить благодарность сотрудникам издательства "БХВ-Петербург", без самоотверженного труда которых эта книга просто не могла родиться. Спасибо моим студентам, работа с ними натолкнула меня на мысль о необходимости написать эту книгу. Не могу не поблагодарить членов команды RSDN, в общении с которыми я провел много часов, выясняя нюансы тех или иных конструкций C++.

# ЧАСТЬ I

# Основы программирования на C++

# ГЛАВА 1

# Как написать программу на C++

"Поехали!"
*Ю. Гагарин*

С момента издания книги "Язык программирования C", написанной Брайаном Керниганом и Денисом Ритчи, стало модным начинать любую книгу по программированию с программы "Hello, World!". Мы немного отступим от этой традиции и сразу "возьмем быка за рога": будем использовать компьютер по прямому назначению — для вычислений. Запустим среду Borland C++ 3.1 и в окне редактора наберем программу "Дважды два", текст которой приведен в листинге 1.1.

**Листинг 1.1. Программа "Дважды два"**

```
//  Программа "Дважды два"
#include <iostream.h>
int main()
{  cout << "2 * 2 =" << 2 * 2 << endl;
   return 0;
}
```

Выполним программу (нажмем клавиши <Ctrl>+<F9>). Если все сделано правильно, то, нажав комбинацию клавиш <Alt>+<F5> и открыв экран пользователя, увидим на экране строку:

```
2 * 2 = 4
```

Разберем программу построчно. Первая строка — это *строчный комментарий*. Такой комментарий всегда начинается с двух символов // и продолжается до конца строки. Начало строчного комментария может быть в любом месте строки, он не обязательно должен начинаться с начала строки. В строчном комментарии, как и в любом другом, можно писать любые символы, доступные на клавиатуре, — транслятор пропустит такую строку.

В C++ есть и *многострочный* комментарий, который появился еще в языке C. Такой комментарий начинается с символов `/*` и заканчивается символами `*/`. Начало и конец комментария могут располагаться на разных строках исходного текста программы, и обычно их пишут на отдельных строках, как показано в следующем примере:

```
/* Это
   многострочный комментарий
*/
```

Внутри многострочного комментария вполне может быть однострочный. Так обычно случается, если программист в процессе работы над программой закомментировал некоторый фрагмент, в котором был написан однострочный комментарий. Однако вложенные многострочные комментарии не допускаются. Такая "дискриминация" возникает потому, что окончанием комментария считается первое встретившееся сочетание символов `*/` — все, что после них, опять считается программой.

**Примечание**

Так требует стандарт. Однако системы фирмы Borland позволяют установить режим работы, при котором вложенные комментарии допускаются.

Следующая строка — это подключение библиотеки ввода/вывода. В языке C++, как и ранее в языке C, отсутствует встроенная система ввода/вывода, и весь обмен данными выполняется внешними программами. В данном случае эта строка обеспечивает нам работу оператора вывода на экран. Имя iostream.h — это имя включаемого файла, который находится в каталоге интегрированной среды include. В языке C++ многие *расширения* подключаются подобным образом. В последней версии C++ все они составляют *стандартную библиотеку*, в состав которой входят две стандартные библиотеки ввода/вывода.

Строка начинается со знака `#` (решетка). Этот символ используется, чтобы обозначить *директивы* препроцессора. *Препроцессор* — это программа, которая выполняет простую обработку текста для последующей трансляции. Существуют и иные директивы препроцессора, например `define`, и ряд других.

Выполнение программы (консольного приложения), написанной на языке C++, всегда начинается с выполнения *главной* функции `main`. Как и всякая другая функция, функция `main` состоит из заголовка

```
int main()
```

и тела в фигурных скобках.

*Заголовок* главной функции `main` имеет стандартный вид:

- сначала указывается *тип возвращаемого значения*. В данном случае тип возвращаемого значения — стандартный *целый* тип `int`, это слово является *зарезервированным словом* языка C++ и его нельзя писать по-другому. Язык C++ включает и иные зарезервированные слова, которые мы будем узнавать по мере изучения;
- `main` — обязательное имя главной функции; другое имя указывать нельзя. Несмотря на это, слово `main` не является зарезервированным словом языка C++;
- скобки после имени показывают, что `main` — это именно имя функции.

Все слова в заголовке написаны маленькими английскими буквами. Если мы попробуем использовать в заголовке хотя бы одну большую букву, программа не будет транслироваться. В языке C++ все зарезервированные слова пишутся маленькими буквами.

В теле функции записывается последовательность действий, которые требуется выполнить. Эта последовательность действий записывается с помощью *операторов*, каждый из которых завершается символом `;` (точка с запятой). В нашей программе в теле функции задан *оператор вывода* на экран

```
cout << "2 * 2 =" << 2 * 2 << endl;
```

и *оператор возврата* значения

```
return 0;
```

Английское слово `return` является зарезервированным словом языка C++; его надо писать именно так, как показано, и нельзя использовать никаким другим образом. Число `0`, заданное в операторе, и есть целое возвращаемое значение, которое может быть проверено с помощью команд операционной системы. Обычно значение `0` означает нормальное завершение программы. Это только соглашение, которого стараются придерживаться все программисты в мире. Однако вы можете написать программу, в которой в качестве нормального возвращаемого значения используете значение `1` или `-1`. Тем не менее постарайтесь придерживаться общепринятого соглашения, потому что соглашения (поверьте моему опыту) облегчают программисту жизнь.

Оператор вывода имеет более сложную структуру:

- `cout` — это *системный объект*, обеспечивающий *вывод* результатов из программы во внешнюю среду, в нашем случае — на экран. Данное слово не является зарезервированным словом C++, тем не менее, во избежание возможной путаницы, не рекомендуется использовать `cout` в каком-нибудь другом смысле;
- в нашем операторе вывода задано три аргумента: строковая константа `"2 * 2 ="`, выражение `2 * 2` и манипулятор `endl`. Перед каждым аргумен-

том прописан двойной знак меньше <<. Это одна из операций языка, которая в данном случае означает "вывод в поток".

*Строковая константа* выводится точно в таком виде, как написана, только без кавычек. При написании программы мы можем внутри кавычек писать любые символы (кроме двойных кавычек, одиночной кавычки-апострофа и символа \ — обратной косой черты), которые есть на клавиатуре. В частности, без ограничений можно использовать русские и английские буквы. Вне кавычек русские буквы можно применять только в комментариях.

*Выражение* содержит три элемента: две целых константы и знак * (звездочка), который обозначает операцию умножения. Выражение вычисляется, и значение выводится на экран.

*Манипулятор* `endl` обеспечивает нам перевод курсора экрана на следующую строку.

Теперь мы легко можем написать и выполнить программу, вычисляющую и выводящую на экран полную таблицу умножения на 2 — мы просто скопируем 10 раз оператор вывода, исправив множители в строках и в выражениях. Понятно, что таким же способом мы можем создать программы для вычисления каких угодно таблиц умножения. Однако лучше написать программу таким образом, чтобы ее можно было использовать с различными данными. В самом деле, таблица умножения на 3 отличается от таблицы умножения на 2 только множителем. Этот множитель можно было бы сообщать программе каждый раз при запуске. Для этого в программе (ее текст приведен в листинге 1.2) необходимо объявить переменную и написать оператор ввода.

**Листинг 1.2. Таблица умножения с вводом переменной**

```
#include <iostream.h>
int main()
{  int k;       //  объявление переменной
   cout << "Введите множитель от 1 до 9: ";
   cin >> k;    //  ввод числа
   cout << k << "* 1 =" << k * 1 << endl;
   cout << k << "* 2 =" << k * 2 << endl;
   cout << k << "* 3 =" << k * 3 << endl;
   cout << k << "* 4 =" << k * 4 << endl;
   cout << k << "* 5 =" << k * 5 << endl;
   cout << k << "* 6 =" << k * 6 << endl;
   cout << k << "* 7 =" << k * 7 << endl;
   cout << k << "* 8 =" << k * 8 << endl;
   cout << k << "* 9 =" << k * 9 << endl;
   cout << k << "* 10 =" << k * 10 << endl;
   return 0;
}
```

В этой программе оператор

```
int k;
```

объявляет целую *переменную* с именем `k`. Имя переменной называется *идентификатором*. Для написания идентификатора программист может использовать большие и маленькие английские буквы, цифры и знак подчеркивания _. Идентификаторы в языке C++ должны начинаться либо с буквы, либо с подчеркивания. С цифры начинать идентификатор нельзя. Подчеркивание тоже первым писать не рекомендуется, хотя и разрешается. Мы вместо слова "идентификатор" часто будем употреблять слово "имя" — оно короче. В C++ различаются большие и маленькие буквы, поэтому следующие имена считаются различными: `ab`, `Ab`, `aB`, `AB`. Количество символов в имени язык не ограничивает, но понятно, что бесконечно длинным имя быть не может. В каждой интегрированной среде обычно устанавливается некоторое ограничение длины имен.

Слово `int` — это тип значений, которые могут храниться в нашей переменной. Как мы помним, `int` является зарезервированным словом. Каждый тип определяется множеством значений и набором операций, которые применяются к значениям. В любом языке программирования обычно есть некоторый набор *встроенных* типов, каждый из которых определяется своим зарезервированным словом. Язык C++ включает много различных встроенных типов, большинство из которых являются числовыми.

Таким образом, каждая переменная, объявленная в программе, имеет имя и ей приписан тип. В разные моменты времени в переменной могут храниться разные *значения*, однако имя и тип изменить нельзя. В общем случае имена всех переменных должны быть разные, однако в C++ есть исключения из этого правила.

После объявления переменной мы написали оператор для вывода приглашения. После него оператор

```
cin >> k;
```

выполняет *ввод* значений переменной `k`. Слово `cin` обозначает системный объект, обеспечивающий нам ввод данных из внешней среды в программу. В данном случае выполняется ввод с клавиатуры. Последней клавишей, которую требуется нажать, должна быть клавиша <Enter>.

Так же, как и `cout`, слово `cin` не является зарезервированным словом C++, однако не рекомендуется использовать его в каком-нибудь другом смысле. Мы видим также, что C++ позволяет программисту записывать несколько операторов в одной строке, лишь бы каждый оператор заканчивался точкой с запятой. После последнего оператора в строке можно написать строчный комментарий.

Далее написаны десять однотипных операторов вывода. Если мы запустим программу, то на экране появится сообщение

```
Введите множитель от 1 до 9:
```

и программа остановится, ожидая от нас ввода целого числа. Нажмем цифру 5 и клавишу <Enter>, и увидим на экране следующие десять строк:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Каждый из операторов вывода работает так: сначала на экран выводится значение переменной k (в данном случае — число 5), потом без пропуска — соответствующая строковая константа, затем вычисляется соответствующее выражение и выводится его значение.

# Проверка условий

Если мы вновь запустим программу (см. листинг 1.2) и введем значение 3, то увидим на экране таблицу умножения на 3. Более того, мы можем ввести любое положительное число, например 147, и получим таблицу умножения на 147. Мы даже можем вычислять таблицы умножения отрицательных чисел. Запустим программу и введем значение -2. Программа правильно вычислит и выведет на экран отрицательную таблицу умножения. Таким образом, программа работает, и достаточно универсальна. Тем не менее изначально требовалось иметь программу, вычисляющую таблицу умножения от 1 до 9. Наша программа делает "лишнюю" работу.

"Корифеи" программирования в один голос утверждают: если программа выполняет не предусмотренную первоначальным заданием работу, то такая программа ошибочна. Таким образом, в программе необходимо проверять значения, которые вводит пользователь, и отвергать ошибочные. Пусть программа в этом случае выводит на экран вежливое предупреждение и заканчивает работу. Напишем это более формальным способом:

```
если 1 ≤ к ≤ 9 то вывести таблицу умножения;
иначе вывести сообщение об ошибке;
```

Чтобы запрограммировать эти действия, нам потребуется *условный* оператор. На языке C++ условный оператор записывается так:

```
if (условие) оператор;
```

или так:

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

Слова `if` и `else` являются зарезервированными словами. Первая форма оператора `if` работает так: если условие истинно, то выполняется заданный оператор. Затем выполняется следующий оператор после условного. Если же условие ложно, то заданный оператор пропускается и сразу начинает работать следующий после `if` оператор. Вторая форма оператора работает таким образом: если условие истинно, то выполняется заданный `оператор_1`; если условие ложно, то работает заданный `оператор_2`. Затем выполняется оператор, следующий после условного.

Как мы видим, вторая форма оператора практически совпадает с написанной на русском языке *схемой*. Однако нам надо решить две проблемы. Первая проблема состоит в том, что наша фраза "вывести таблицу умножения" означает запись на языке C++ десяти операторов вывода, в то время как в операторе `if` можно указывать только один оператор. Эта проблема решается очень просто — с помощью фигурных скобок. Оператор `if` в этом случае приобретает вид:

```
if  (условие)
{  //    составной оператор
   cout << k << "* 1 =" << k * 1 << endl;
   cout << k << "* 2 =" << k * 2 << endl;
   cout << k << "* 3 =" << k * 3 << endl;
   cout << k << "* 4 =" << k * 4 << endl;
   cout << k << "* 5 =" << k * 5 << endl;
   cout << k << "* 6 =" << k * 6 << endl;
   cout << k << "* 7 =" << k * 7 << endl;
   cout << k << "* 8 =" << k * 8 << endl;
   cout << k << "* 9 =" << k * 9 << endl;
   cout << k << "* 10 =" << k * 10 << endl;
}
else cout << "Неправильное значение множителя!" << endl;
```

Несколько операторов, заключенных в фигурные скобки, называются *составным оператором*. В тексте программы составной оператор может стоять везде, где может быть указан простой оператор. Если нам потребуется, мы можем задать составной оператор и после `else`.

Теперь разберемся, как писать условие. В языке C++ нет значка ≤ — вместо него необходимо писать <= (меньше или равно). Следовательно, на языке C++ наше условие выглядит так: `1 <= k <=9`. Однако в таком виде условие работать не будет. Чтобы убедиться в этом, напишем простую программу, текст которой приведен в листинге 1.3.

**Листинг 1.3. Проверка заданных условий**

```
#include <iostream.h>
int main()
{  int k;
   cout << "Введите целое значение от 1 до 9: ";
   cin >> k;    //  ввод числа
   if (1 <= k <= 9) cout << "Истинно" << endl;
   else cout << "Ложно" << endl;
   return 0;
}
```

Программа транслируется без ошибок и выполняется. Однако выясняется, что при любых значениях переменной `k` на экран всегда выводится слово "Истинно". Дело в том, что выражение `1 <= k <= 9` вычисляется последовательно слева направо. Поэтому сначала вычисляется выражение `1 <= k`, и вычисленный результат сравнивается с `9`.

**Примечание**

Здесь мы сталкиваемся с очень неприятной для программистов особенностью языка C++: неявным преобразованием типов. Результат вычисления выражения `1 <= k` преобразуется в целое число! Если выражение истинно, то результатом будет `1`; если же выражение ложно, то результат равен `0`. Теперь понятно, почему наше условие всегда истинно: и `0 < 9`, и `1 < 9`.

Вместо привычной математической записи мы должны в данном случае написать *логическое выражение*. Для этого сначала нужно разделить наше условие на два отдельных, а затем объединить их с помощью логической операции И. На языке C++ это выглядит так:

```
(1 <= k) && (k <= 9)
```

Немного переделаем условие в соответствии со стилем C++:

```
(0 < k) && (k < 10)
```

С точки зрения смысла ничего не изменилось, но такое выражение проще писать и легче читать. Скобки ясно выражают наши намерения. Таким образом, в листинге 1.4 представлен окончательный вариант нашей программы.

**Листинг 1.4. Таблица умножения с проверкой множителя**

```
#include <iostream.h>
int main()
{  int k;        //  объявление переменной
   cout << "Введите множитель от 1 до 9: ";
   cin >> k;     //  ввод числа
   if ((0 < k) && (k < 10))    //  внешние скобки обязательны
   {  cout << k << "* 1 =" << k * 1 << endl;    //  составной оператор
      cout << k << "* 2 =" << k * 2 << endl;
      cout << k << "* 3 =" << k * 3 << endl;
      cout << k << "* 4 =" << k * 4 << endl;
      cout << k << "* 5 =" << k * 5 << endl;
      cout << k << "* 6 =" << k * 6 << endl;
      cout << k << "* 7 =" << k * 7 << endl;
      cout << k << "* 8 =" << k * 8 << endl;
      cout << k << "* 9 =" << k * 9 << endl;
      cout << k << "* 10 =" << k * 10 << endl;
   }
   else cout << "Неправильное значение множителя!" << endl;
   return 0;
}
```

Напишем другую программу (ее текст приведен в листинге 1.5), в которой условный оператор используется более интенсивно. Пусть у нас есть кирпич с размерами сторон `a`, `b`, `c` и есть прямоугольное отверстие размером `x` на `y`. Необходимо проверить, пролезет ли кирпич в отверстие. Очевидно, что кирпич пройдет в отверстие, если любые две из его сторон меньше размеров отверстия. Упорядочим размеры отверстия `(x < y)`, и "выстроим" размеры сторон кирпича в возрастающем порядке: `a < b < c`. Тогда для ответа на вопрос, пройдет ли кирпич в отверстие, нужна одна проверка:

```
если (a < x) и (b < y) то ответ "ДА" иначе ответ "НЕТ"
```

Выстроить стороны кирпича в нужном порядке можно, обменяв значения переменных, как показано в следующей схеме:

```
если (a > b) то { t = a; a = b; b = t; }
если (b > c) то { t = b; b = c; c = t; }
если (a > b) то { t = a; a = b; b = t; }
```

Двух перестановок недостаточно. Пусть, например, `a = 10`, `b = 8`, `c = 6`. После первой проверки `a = 8`, `b = 10`, `c = 6`; после второй — `a = 8`, `b = 6`, `c = 10`. Аналогично можно поступить и с размерами отверстия, поэтому считаем, что `x < y`. Наши рассуждения почти "дословно" переводятся на язык C++.

**Листинг 1.5. Программа "Кирпич и отверстие"**

```
#include <iostream.h>
int main()
{  double a, b, c;    //  размеры кирпича
   cout << "Задайте размеры кирпича: ";
   cin >> a >> b >> c;
   double x, y;       //  размеры отверстия
   cout << "Задайте размеры отверстия: ";
   cin >> x >> y;
   if (x > y) { double t = x; x = y; y = t; }
   if (a > b) { double t = a; a = b; b = t; }
   if (b > c) { double t = b; b = c; c = t; }
   if (a > b) { double t = a; a = b; b = t; }
   if ((a < x) && (b < y)) cout << "Кирпич пройдет в отверстие!" << endl;
   else cout << "Кирпич не пройдет в отверстие!" << endl;
   return 0;
}
```

В программе, помимо использования условного оператора, мы видим еще несколько новых особенностей. Переменные объявляются по мере необходимости, а не в начале программы. В операторах ввода указаны сразу несколько переменных:

```
cin >> x >> y;
```

Такая запись просто короче, чем два оператора:

```
cin >> x;
cin >> y;
```

Как в том, так и в другом случае при вводе, значения можно задавать через пробел. Клавиша <Enter> нажимается после второго числа. Аналогично работает и оператор ввода размеров кирпича.

Самое интересное, что объявление переменной `t` четыре раза не вызывает "протестов" транслятора. Это объясняется тем, что данная переменная объявляется внутри составного оператора. Составной оператор с объявленными внутри переменными по многолетней традиции программирования называют *блоком*. В C++ принято, что в любом месте программы, где может стоять составной оператор, может стоять и блок. Поэтому мы в дальнейшем не будем делать различия между блоком и составным оператором и всегда будем называть конструкцию в фигурных скобках блоком. Переменные, объявленные в блоке, считаются *локальными переменными* в блоке и "живут от скобки до скобки" *(см. гл. 2)*.

## Оператор-переключатель

В C++ есть еще один оператор выбора решения — оператор-переключатель. Напишем программу, представляющую упрощенную версию калькулятора. Программа вводит два числа и знак операции, которую требуется выполнить с этими числами. Знак операции — это один из символов +, –, *, /. Сначала напишем программу-калькулятор на C++, а затем разберем ее работу. Текст программы приведен в листинге 1.6.

**Листинг 1.6. Калькулятор с оператором `switch`**

```
#include <iostream.h >
int main()
{  cout << "Задайте два числа a и в: ";
   double a, b;
   cin >> a >> b;    //  ввод двух значений
   cout << "Задайте операцию [+ – * /]: ";
   char op;
   cin >> op;        //   ввод знака операции
   switch (op)       //  оператор-переключатель
   {  case '+': cout << a + b << endl; break;
      case '-': cout << a – b << endl; break;
      case '*': cout << a * b << endl; break;
      case '/': if (b != 0) cout << a / b << endl;
      else cout << "Делитель равен нулю!" << endl;
      break;
      default: cout << "Неверный знак операции!" << endl;
   }
   return 0;
}
```

Как обычно, в программе сначала вводятся необходимые данные, а потом символ операции, которую надо выполнить. Ввод символа по форме ничем не отличается от ввода чисел. После ввода написан оператор-переключатель. В общем виде оператор-переключатель имеет форму:

```
switch (переключающее_выражение)
{ набор альтернатив }
```

*Переключающим выражением* в нашей программе является переменная типа `char`, значение которой вводится с клавиатуры как символ. В общем виде одна *альтернатива* оператора имеет вид:

```
case константа: операторы;
```

Константа должна быть того же типа, что и переключающее выражение. В отличие от других операторов языка C++, в альтернативе можно задавать несколько операторов без фигурных скобок. Одна из альтернатив — особая, она начинается со слова `default`. Эту альтернативу можно не писать. Все слова — зарезервированные слова языка C++. Оператор-переключатель работает следующим образом:

- вычисляется переключающее выражение;
- значение выражения сравнивается поочередно с константами в альтернативах;
- первое совпадение приводит к выполнению операторов совпавшей альтернативы;
- выполняются все остальные операторы во всех остальных альтернативах, в том числе и в `default`.

После этого выполняется следующий за `switch` оператор. Однако на практике почти всегда нужно выполнять только *операторы совпавшей альтернативы*. Именно это требуется и в нашем случае. Для ограничения действия оператора-переключателя только одной альтернативой используется оператор `break`. Это тоже зарезервированное слово языка C++. Его действие заключается в немедленном выходе из блока — происходит неявный переход на оператор вне фигурных скобок оператора `switch`. Таким образом, оператор-переключатель в нашей программе работает так:

- значение переменной `op` по очереди сравнивается с константами в альтернативах;
- если произошло совпадение с одним из символов, выполняется соответствующий оператор вывода, в котором вычисляется нужное выражение;
- выполняется оператор `break`, что приводит к немедленному выходу из оператора-переключателя.

Если мы введем неправильный символ — знак операции, то совпадения не произойдет, и выполнится оператор в альтернативе `default`, который выведет сообщение об ошибке. В этой альтернативе нет необходимости задавать оператор `break`, т. к. после выполнения операторов этой альтернативы все равно происходит выход из оператора-переключателя.

Оператор-переключатель можно промоделировать с помощью вложенных условных операторов. Мы могли бы в программе-калькуляторе написать следующее:

```
if (op == '+') cout << a + b << endl;
else if (op == '-') cout << a – b << endl;
else if (op == '*') cout << a * b << endl;
else if (op == '/')
```

```
{ if (b != 0) cout << a/b << endl;
  else cout << "Делитель равен нулю!" << endl;
}
else cout << "Неверный знак операции!" << endl;
```

При такой записи необходимость в операторе `break` отпадает.

Следует отметить, что переключатель наиболее часто применяется для распознавания различных клавиш, вводимых с клавиатуры. Очень часто этот оператор используется при реализации *конечных автоматов* [3, 32, 35], где одна альтернатива оператора `switch` обозначает одно состояние автомата.

# Повторение вычислений

Вернемся к программе "Таблица умножения с проверкой множителя", приведенной в листинге 1.4. Теперь наша программа делает именно то, что и требовалось. Сначала просит ввести множитель: если множитель правильный, то вычисляется и выводится на экран соответствующая таблица умножения; если множитель неправильный, то программа выводит сообщение об ошибке и заканчивает работу. Все правильно. Однако рассмотрим вывод таблицы умножения внимательнее. Операторы вывода отличаются один от другого всего в двух местах: в текстовой константе и в выражении умножения. Хотелось бы заменить десять очень похожих операторов одним-единственным, который отработает за все десять.

Это не пустой интерес программиста, которому лень написать десять почти одинаковых строк, это очень важная проблема. Представьте себе, что надо вычислять не десять вариантов, а 573. Или даже миллион! Если 573 еще можно написать вручную, то миллион мы просто не осилим: если в секунду писать по одному оператору, то в течение суток мы напишем 86 400 операторов. На всю программу у нас уйдет 2 недели! А представляете размер такой программы? Поэтому сокращение текста программы за счет однотипных вычислений — это очень важное действие.

Разумеется, это можно сделать, если повторить выполнение оператора вывода десять раз. И при каждом повторении (которые в программировании обычно называются *итерациями*) требуется изменить текстовую константу и второй множитель в выражении умножения. Для решения этих проблем мы заменим явную целую константу переменной. Эта переменная должна увеличиваться на `1` при каждом повторении. Назовем нашу переменную именем `i`. Значение переменной `i` представляет собой номер итерации. Обычно такие переменные называют *счетчиками*. В языке C++ принято (хотя это и не обязательно) начинать считать с нуля. С учетом этого соглашения мы можем написать схему так:

```
i = 0;
пока (условие истинно) увеличить i;
вывод одной строки таблицы умножения
```

Мы специально сдвинули две строки вправо, чтобы подчеркнуть тот факт, что эти строки должны повторяться, пока ... что? Разберемся с условием повторений. Последний раз повторение оператора вывода должно произойти при значении `i`, равном `9`. Как только `i` станет больше `9`, повторения должны прекратиться. Поэтому условие можно записать так: `i < 10`.

На языке C++ повторение выполнения можно организовать, если использовать *оператор цикла*. В C++ имеется три оператора цикла, но в данном случае мы используем *оператор с предусловием*, который записывается так:

```
while (условие) оператор;
```

Слово `while` является зарезервированным словом C++. Этот оператор цикла работает следующим образом: пока условие истинно, выполняется заданный оператор; как только условие стало ложным, заданный оператор пропускается и начинает работать следующий после цикла оператор. Условие записывается точно так же, как и в операторе `if`. И точно так же на месте заданного оператора можно прописать составной оператор в фигурных скобках. Составной оператор называется *телом цикла*. Наша схема на C++ будет выглядеть так:

```
i = 0;              //  начальное значение
while (i < 10)      //  условие продолжения
{  ++i;             //  увеличение переменной на 1
   cout << k << "*" << i << "=" << k * i << endl;
}
```

Мы преобразовали текстовую константу в операторе вывода. По сравнению с первоначальным вариантом от нее осталось только два неизменных символа: звездочка и равно, — остальное превратилось в переменные, имеющие разные значения в разный момент времени. Теперь можно написать полную программу с циклом, текст которой приведен в листинге 1.7.

**Листинг 1.7. Таблица умножения с циклом `while`**

```
#include <iostream.h>
int main()
{  int k;                        //  объявление переменной
   cout << "Введите множитель от 1 до 9: ";
   cin >> k;                     //  ввод числа
   if ((0 < k) && (k < 10))      //  внешние скобки обязательны
   {  int i = 0;                 //  начальное значение
```

```
      while (i < 10)            //  условие продолжения
     {  ++i;                    //  увеличение переменной на 1
        cout << k << "*" << i << "=" << k * i << endl;
     }
   }
   else cout << "Неправильное значение множителя!" << endl;
   return 0;
}
```

Как и в предыдущей версии исполнения программы, на экран выводится приглашение, и программа ждет ввода значения. Если задано недопустимое число, то программа выводит сообщение об ошибке и заканчивает работу. При вводе правильного значения выполняется составной оператор после условия оператора `if`. Сначала объявляется переменная `i`, которая сразу же получает значение `0`. В языке C++ переменные можно *инициализировать* сразу при объявлении.

Потом оператор цикла проверяет условие, которое в данном случае истинно (`0 < 10`). Следовательно, выполняется тело цикла. Сначала значение переменной `i` увеличивается на `1` и становится равным `1` — это делает оператор `++i`. После этого работает оператор вывода. Сначала выводится значение переменной `k`, потом (без пропусков) — символ `*`, значение `i` (в данном случае — единица), знак `=` (равно) и значение выражения `k * i`. Курсор экрана переводится на следующую строку. На этом месте тело цикла заканчивается, и программа возвращается к проверке условия. Условие (`1 < 10`) истинно, поэтому повторяется только что описанный процесс. И так продолжается до тех пор, пока при очередном увеличении значение переменной `i` не станет равным `10`. Тогда при проверке условие (`10 < 10`) окажется ложным (`10` не меньше `10`) и произойдет выход из цикла. И сразу же — выход из оператора `if`. Таким образом, будет выполняться оператор возврата — программа завершит работу.

Можно использовать и другой оператор цикла, в котором условие проверяется после выполнения тела цикла. Форма записи *цикла с постусловием* следующая:

```
do оператор;
while (условие);
```

Как обычно, на месте единственного оператора может стоять составной оператор в фигурных скобках. Важное отличие оператора цикла с постусловием от предыдущего варианта состоит в том, что в данном случае тело цикла обязательно выполнится хотя бы один раз, тогда как при вычислении предусловия может возникнуть такая ситуация, что тело цикла не сработает ни разу. Само условие вычисляется точно так же. Наша программа с циклом `do...while`, текст которой приведен в листинге 1.8.