БЬЯРНЕ СТРАУСТРУП

Внесено более 750 авторских исправлений и изменений!

Исправленное издание!

ПРОГРАММИРОВАНИЕ

принципы и практика использования C++

Исправленное издание

ББК 32.973.26-018.2.75
С83
УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция докт. физ.-мат. наук *Д.А. Клюшина*
По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Страуструп, Бьярне.

С83 Программирование: принципы и практика использования C++, испр. изд. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2011. — 1248 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1705-8 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Бьярне Страуструп

Программирование: принципы и практика использования C++

Литературный редактор *И.А. Попова*
Верстка *А.В. Плаксюк*
Художественный редактор *Е.П. Дынник*
Корректор *Л.А. Гордиенко*

Подписано в печать 20.12.2010. Формат 70x100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 100,62. Уч.-изд. л. 60,5.
Тираж 1000 экз. Заказ № 0000.

Отпечатано по технологии CtP
в ОАО “Печатный двор” им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15.

ООО “И. Д. Вильямс”, 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1705-8 (рус.)
ISBN 978-0-321-54372-1 (англ.)

Оглавление

Часть I
Основы

2

Hello, World!

“Чтобы научиться программированию,
необходимо писать программы”.

Брайан Керниган (Brian Kernighan)

В этой главе приводится простейшая программа на языке C++, которая на самом деле ничего не делает. Предназначение этой программы заключается в следующем.

- Дать вам возможность поработать с интегрированной средой разработки программ.
- Дать вам почувствовать, как можно заставить компьютер делать то, что нужно.

Итак, мы приводим понятие программы, идею о трансляции программ из текстовой формы, понятной для человека, в машинные инструкции с помощью компилятора для последующего выполнения.

В этой главе...

2.1. Программы

Для того чтобы заставить компьютер сделать что-то, вы (или кто-то еще) должны точно рассказать ему — со всеми подробностями, — что именно хотите. Описание того, "что следует сделать", называется *программой*, а *программирование* — это вид деятельности, который заключается в создании и отладке таких программ. В некотором смысле мы все программисты.

Кроме того, мы сами получаем описания заданий, которые должны выполнить, например "как проехать к ближайшему кинотеатру" или "как поджарить мясо в микроволновой печи". Разница между такими описаниями или программами заключается в степени точности: люди стараются компенсировать неточность инструкций, руководствуясь здравым смыслом, а компьютеры этого сделать не могут. Например, "по коридору направо, вверх по лестнице, а потом налево" — вероятно, прекрасная инструкция, позволяющая найти ванную на верхнем этаже. Однако, если вы посмотрите на эти простые инструкции, то выяснится, что они являются грамматически неточными и неполными. Человек может легко восполнить этот недостаток. Например, допустим, что вы сидите за столом и спрашиваете, как пройти в ванную. Отвечающий вам человек совершенно не обязан говорить вам, чтобы вы встали из-за стола, обошли его (а не перепрыгнули через него или проползли под ним), не наступили на кошку и т.д. Вам также никто не скажет, чтобы вы положили на стол нож и вилку или включили свет, когда будете подниматься по лестнице. Открыть дверь в ванную, прежде чем войти в нее вам, вероятно, также не посоветуют.

В противоположность этому компьютер *действительно* глуп. Ему все необходимо точно и подробно описать. Вернемся к инструкциям "по коридору направо, вверх по лестнице, а потом налево". Где находится коридор? Что такое коридор? Что значит "направо"? Что такое лестница? Как подняться по лестнице? По одной ступеньке? Через две ступеньки? Держась за перила? Что находится слева от меня? Когда это окажется слева от меня? Для того чтобы подробно описать инструкции для компьютера, необходим точно определенный язык, имеющий специфическую грамматику (естественный язык слишком слабо структурирован), а также хорошо определенный словарь для всех видов действий, которые мы хотим выполнить. Такой язык называется *языком программирования*, и язык программирования C++ — один из таких языков, разработанных для решения широкого круга задач.

Более широкие философские взгляды на компьютеры, программы и программирование изложены в главе 1. Здесь мы рассмотрим код, начиная с очень простой программы, а также несколько инструментов и методов, необходимых для ее выполнения.

2.2. Классическая первая программа

Приведем вариант классической первой программы. Она выводит на экран сообщение Hello, World! .

```
// Эта программа выводит на экран сообщение "Hello,World!"
#include "std_lib_facilities.h"
int main() // Программы на C++ начинаются с выполнения функции main
{
     cout << "Hello, World!\n"; // вывод "Hello,World!"
     return 0;
}
```

Этот набор команд, которые должен выполнить компьютер, напоминает кулинарный рецепт или инструкции по сборке новой игрушки. Посмотрим, что делает каждая из строк программы, начиная с самого начала:

```
cout << "Hello, World!\n"; // вывод "Hello,World!"
```

Именно эта строка выводит сообщение на экран. Она печатает символы **Hello, World!**, за которыми следует символ перехода на новую строку; иначе говоря, после вывода символов **Hello,World!** курсор будет установлен на начало новой строки. *Курсор* — это небольшой мерцающий символ или строка, показывающая, где будет выведен следующий символ.

В языке C++ строковые литералы выделяются двойными кавычками (**"**); т.е. **"Hello, Word!\n"** — это строка символов. Символ **\n** — это специальный символ, означающий переход на новую строку. Имя **cout** относится к стандартному потоку вывода. Символы, "выведенные в поток **cout**" с помощью оператора вывода **<<**, будут отображены на экране. Имя **cout** произносится как "see-out", но является аббревиатурой "character output stream" ("поток вывода символов"). Аббревиатуры довольно широко распространены в программировании. Естественно, аббревиатура на первых порах может показаться неудобной для запоминания, но привыкнув, вы уже не сможете от них отказаться, так как они позволяют создавать короткие и управляемые программы.

Конец строки

```
// вывод "Hello,World!"
```

является комментарием. Все, что написано после символа **//** (т.е. после двойной косой черты (**/**), которая называется слэшем), считается комментарием. Он игнорируется компилятором и предназначен для программистов, которые будут читать программу. В данном случае мы использовали комментарии для того, чтобы сообщить вам, что именно означает первая часть этой строки.

Комментарии описывают предназначение программы и содержат полезную информацию для людей, которую невозможно выразить в коде. Скорее всего, человеком, который извлечет пользу из ваших комментариев, окажетесь вы сами, когда вернетесь к своей программе на следующей неделе или на следующий год, забыв, для чего вы ее писали. Итак, старайтесь хорошо документировать свои программы. В разделе 7.6.4 мы обсудим, как писать хорошие комментарии.

Программа пишется для двух аудиторий. Естественно, мы пишем программы для компьютеров, которые будут их выполнять. Однако мы долгие годы проводим за чтением и модификацией кода. Таким образом, второй аудиторией для программ являются другие программисты. Поэтому создание программ можно считать формой общения между людьми. Действительно, целесообразно главными читателями своей программы считать людей: если они с трудом понимают, что вы написали, то вряд ли программа когда-нибудь станет правильной. Следовательно, нельзя забывать, что код предназначен для чтения — необходимо делать все, чтобы программа легко читалась. В любом случае комментарии нужны лишь людям, компьютеры игнорируют комментарии.

Первая строка программы — это типичный комментарий, которая сообщает читателям, что будет делать программа.

```
// Эта программа выводит на экран сообщение "Hello,World!"
```

Эти комментарии полезны, так как по коду можно понять, что делает программа, но нельзя выяснить, чего мы на самом деле хотели. Кроме того, в комментариях мы можем намного лаконичнее объяснить цель программы, чем в самом коде (как правило, более подробном). Часто такие комментарии размещаются в первых строках программы. Помимо всего прочего, они напоминают, что мы пытаемся сделать.

Строка

```
#include "std_lib_facilities.h"
```

представляет собой директиву **#include**. Она заставляет компьютер “включить” возможности, описанные в файле **std_lib_facilities.h**. Этот файл упрощает использование возможностей, предусмотренных во всех реализациях языках C++ (стандартной библиотеке языка C++).

По мере продвижения вперед мы объясним эти возможности более подробно. Они написаны на стандартном языке C++, но содержат детали, в которые сейчас не стоит углубляться, отложив их изучение до следующих глав. Важность файла **std_lib_facilities.h** для данной программы заключается в том, что с его помощью мы получаем доступ к стандартным средствам ввода-вывода языка C++. Здесь мы просто используем стандартный поток вывода **cout** и оператор вывода **<<**. Файл, включаемый в программу с помощью директивы **#include**, обычно имеет расширение **.h** и называется *заголовком* (header), или *заголовочным файлом* (header file). Заголовок содержит определения терминов, таких как **cout**, которые мы используем в нашей программе.

Как компьютер находит точку, с которой начинается выполнение программы? Он просматривает функцию с именем **main** и начинает выполнять ее инструкции. Вот как выглядит функция **main** нашей программы "Hello, World!":

```
int main() // Программы на C++ начинаются с выполнения функции main
{
     cout << "Hello, World!\n"; // вывод "Hello,World!"
     return 0;
}
```

Для того чтобы определить отправную точку выполнения, каждая программа на языке C++ должна содержать функцию с именем **main**. Эта функция по существу представляет собой именованную последовательность инструкций, которую компьютер выполняет в порядке перечисления. Эта функция состоит из четырех частей.

- *Тип возвращаемого значения*, в этой функции — тип **int** (т.е. целое число), определяет, какой результат возвращает функция в точку вызова (если она возвращает какое-нибудь значение). Слово **int** является зарезервированным в языке C++ (*ключевым словом*), поэтому его нельзя использовать как имя чего-нибудь еще (см. раздел A.3.1).
- *Имя*, в данном случае **main**.
- *Список параметров*, заключенный в круглые скобки (см. разделы 8.2 и 8.6); в данном случае список параметров пуст.
- *Тело функции*, заключенное в фигурные скобки и перечисляющее действия (называемые *инструкциями*), которые функция должна выполнить.

Отсюда следует, что минимальная программа на языке C++ выглядит так:

```
int main() { }
```

Пользы от этой программы мало, так как она ничего не делает. Тело функции **main** программы "Hello, World!" содержит две инструкции:

```
cout << "Hello, World!\n"; // вывод "Hello,World!"
return 0;
```

Во-первых, она выводит на экран строку **Hello, World!**, а затем возвращает значение `0` (нуль) в точку вызова. Поскольку функция **main()** вызывается системой, мы не будем использовать возвращаемое значение. Однако в некоторых системах (в частности, Unix/Linux) это значение можно использовать для проверки успешности выполнения программы. Нуль (`0`), возвращаемый функцией **main()**, означает, что программа выполнена успешно.

Часть программы на языке C++, определяющая действие и не являющаяся директивой **#include** (или другой директивой препроцессора; см. разделы 4.4 и A.17), называется *инструкцией*.

2.3. Компиляция

C++ — компилируемый язык. Это значит, что для запуска программы сначала необходимо транслировать ее из текстовой формы, понятной для человека, в форму, понятную для машины. Эту задачу выполняет особая программа, которая называется *компилятором*. То, что вы пишете и читаете, называется *исходным кодом*, или *текстом программы*, а то, что выполняет компьютер, называется *выполняемым*, *объектным*, или *машинным кодом.* Обычно файлы с исходным кодом программы на языке C++ имеют расширение **.cpp** (например, **hello_world.cpp**) или **.h** (например, **std_lib_facilities.h**), а файлы с объектным кодом имеют расширение **.obj** (в системе Windows) или **.o** (в системе Unix). Следовательно, простое слово *код* является двусмысленным и может ввести в заблуждение; его следует употреблять с осторожностью и только в ситуациях, когда недоразумение возникнуть не может. Если не указано иное, под словом *код* подразумевается исходный код или даже исходный код, за исключением комментариев, поскольку комментарии предназначены для людей и компилятор не переводит их в объектный код.

Компилятор читает исходный код и пытается понять, что вы написали. Он проверяет, является ли программа грамматически корректной, определен ли смысл каждого слова. Обнаружив ошибку, компилятор сообщает о ней, не пытаясь выполнить программу. Компиляторы довольно придирчивы к синтаксису. Пропуск какой-нибудь детали, например директивы **#include**, двоеточия или фигурной скобки, приводит к ошибке. Кроме того, компилятор точно так же абсолютно нетерпим к опечаткам. Продемонстрируем это рядом примеров, в каждом из которых сделана небольшая ошибка. Каждая из этих ошибок является довольно типичной.

```
// пропущен заголовочный файл
int main()
{
     cout << "Hello, World!\n";
     return 0;
}
```

Мы не сообщили компилятору о том, что представляет собой объект cout, поэтому он сообщает об ошибке. Для того чтобы исправить программу, следует добавить директиву **#include**.

```
#include "std_facilities.h"
int main()
{
     cout << "Hello, World!\n";
     return 0;
}
```

К сожалению, компилятор снова сообщает об ошибке, так как мы сделали опечатку в строке **std_lib_facilities.h**. Компилятор заметил это.

```
#include "std_lib_facilities.h
int main()
{
     cout << "Hello, World!\n;
     return 0;
}
```

В этом примере мы пропустили закрывающую двойную кавычку (**"**). Компилятор указывает нам на это.

```
#include "std_lib_facilities.h"
integer main()
{
     cout << "Hello, World!\n";
     return 0;
}
```

Теперь мы вместо ключевого слова **int** использовали слово **integer**, которого в языке C++ нет. Компилятор таких ошибок не прощает.

```
#include "std_lib_facilities.h"
int main()
{
     cout < "Hello, World!\n";
     return 0;
}
```

Здесь вместо символов **<<** (оператор вывода) использован символ **<** (оператор “меньше”). Компилятор это заметил.

```
#include "std_lib_facilities.h"
int main()
{
     cout << 'Hello, World!\n';
     return 0;
}
```

Здесь вместо двойных кавычек, ограничивающих строки, по ошибке использованы одинарные. Приведем заключительный пример.

```
#include "std_lib_facilities.h"
int main()
{
     cout << "Hello, World!\n"
     return 0;
}
```

В этой программе мы забыли завершить строку, содержащую оператор вывода, точкой с запятой. Обратите внимание на то, что в языке C++ каждая инструкция завершается точкой с запятой (**;**). Компилятор распознает точку с запятой как символ окончания инструкции и начала следующей. Трудно коротко, неформально и технически корректно описать все ситуации, в которых нужна точка с запятой.

Пока просто запомните правило: точку с запятой следует ставить после каждого выражения, которое не завершается закрывающей фигурной скобкой.

Почему мы посвятили две страницы и несколько минут вашего драгоценного времени демонстрации тривиальных примеров, содержащих тривиальные ошибки? Для того чтобы в будущем вы не тратили много времени на поиск ошибок в исходном тексте программы. Бо́льшую часть времени программисты ищут ошибки в своих программах. Помимо всего прочего, если вы убеждены, что некий код является правильным, то анализ любого другого кода покажется вам пустой тратой времени. На заре компьютерной эры первые программисты сильно удивлялись, насколько часто они делали ошибки и как долго их искали. И по сей день большинство начинающих программистов удивляются этому не меньше.

☑ Компилятор иногда будет вас раздражать. Иногда кажется, что он придирается к несущественным деталям (например, к пропущенным точкам с запятыми) или к вещам, которые вы считаете абсолютно правильными. Однако компилятор, как правило, не ошибается: если он выводит сообщение об ошибке и отказывается создавать объектный код из вашего исходного кода, то это значит, что ваша программа не в порядке; иначе говоря, то, что вы написали, не соответствует стандарту языка C++.

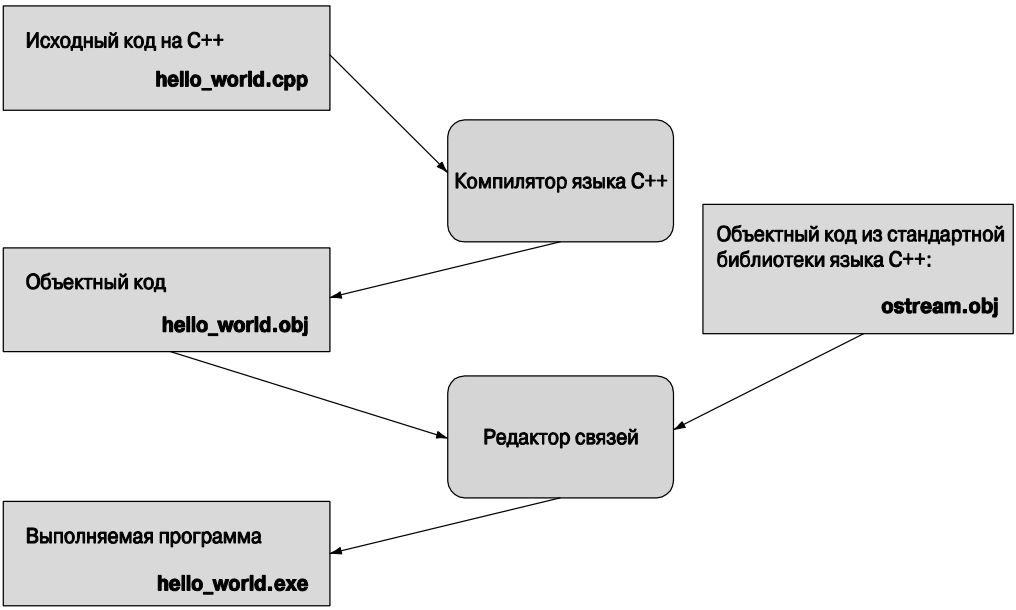
☒ Компилятор не руководствуется здравым смыслом (он — не человек) и очень придирчив к деталям. Поскольку здравый смысл ему не ведом, он не пытается угадать, что на самом деле вы имели в виду, написав фрагмент программы, который выглядит абсолютно правильным, но не соответствует стандарту языка C++. Если бы он угадывал смысл программы и результат оказался бы неожиданным, то вы провели бы очень много времени, пытаясь понять, почему программа не делает то, что вы хотели. После того как все сказано и сделано, компилятор предохраняет нас от множества проблем. Он предотвращает намного больше проблем, чем создает сам.

☑ Итак, помните: компилятор — ваш друг; возможно, лучший друг.

2.4. Редактирование связей

Программа обычно состоит из нескольких отдельных частей, которые часто разрабатываются разными людьми. Например, программа “Hello, World!” состоит из части, которую написали вы, и частей стандартной библиотеки языка C++. Эти отдельные части (иногда называемые *единицами трансляции*) должны быть скомпилированы, а файлы с результирующим объектным кодом должны быть связаны вместе, образуя выполняемый файл. Программа, связывающая эти части в одно целое, называется (вполне ожидаемо) *редактором связей*.

Заметьте, что объектные и выполняемые коды *не* переносятся из одной системы в другую. Например, когда вы компилируете программу под управлением системы Windows, то получите объектный код именно для системы Windows, а не Linux.



Библиотека — это просто некий код (обычно написанный другими), который можно использовать с помощью директивы `#include`. *Объявление* — это инструкция программы, указывающая, как можно использовать фрагмент кода; объявления будут подробно описаны позднее (см., например, раздел 4.5.2).

Ошибки, обнаруженные компилятором, называются *ошибками этапа компиляции*, ошибки, обнаруженные редактором связи, называются *ошибками этапа редактирования связей*, а ошибки, не найденные на этих этапах, называются *ошибками при выполнении программы*, или *логическими ошибками*. Как правило, ошибки этапа компиляции легче понять и исправить, чем ошибки этапа редактирования связей. В свою очередь, ошибки этапа компиляции легче обнаружить и исправить, чем логические. Ошибки и способы их обработки более детально обсуждаются в главе 5.

2.5. Среды программирования

Для программирования необходим язык программирования. Кроме того, для преобразования исходного кода в объектный нужен компилятор, а для редактирования связей нужен редактор связей. Кроме того, для ввода и редактирования исходного текста в компьютера также необходима отдельная программа. Эти инструменты, крайне необходимые для разработки программы, образуют среду разработки программ.

Если вы работаете с командной строкой, как многие профессиональные программисты, то должны самостоятельно решать проблемы, связанные с компилированием и редактированием связей. Если же вы используете среды IDE (интерактивные или интегрированные среды разработки), которые также весьма популярны среди профес-

сиональных программистов, то достаточно щелкнуть на соответствующей кнопке. Описание компиляции и редактирования связей описано в приложении B.

Интегрированные среды разработки включают в себя редактор текстов, позволяющий, например, выделять разным цветом комментарии, ключевые слова и другие части исходного кода программы, а также помогающий отладить, скомпилировать и выполнить программу. *Отладка* — это поиск и исправление ошибок в программе (по ходу изложения мы еще не раз вспомним о ней).

В этой книге в качестве интегрированной среды программирования используется программа Visual C++ компании Microsoft. Если мы говорим просто "компилятор" или ссылаемся на какую-то часть интегрированной среды разработки, то это значит, что мы имеем в виду часть программы Visual C++. Однако вы можете использовать любую другую систему, обеспечивающую современную и стандартную реализацию языка C++. Все, что мы напишем, при очень небольшой модификации, остается справедливым для всех реализаций языка C++, и код будет работать на любом компьютере. В нашей работе мы обычно используем несколько разных реализаций.

Задание

До сих пор мы говорили о программировании, коде и инструментах (например, о компиляторах). Теперь нам необходимо выполнить программу. Это очень важный момент в изложении и в обучении программированию вообще. Именно с этого начинается усвоение практического опыта и овладение хорошим стилем программирования. Упражнения в этой главе предназначены для того, чтобы вы освоились с вашей интегрированной средой программирования. Запустив программу "Hello, World!" на выполнение, вы сделаете первый и главный шаг как программист.

Цель задания — закрепить ваши навыки программирования и помочь вам приобрести опыт работы со средами программирования. Как правило, задание представляет собой последовательность модификаций какой-нибудь простой программы, которая постепенно "вырастает" из совершенно тривиального кода в нечто полезное и реальное.

Для выявления вашей инициативы и изобретательности предлагаем набор традиционных упражнений. В противоположность им задания не требуют особой изобретательности. Как правило, для их выполнения важно последовательно выполнять шаг за шагом, каждый их которых должен быть простым (и даже тривиальным). Пожалуйста, не умничайте и не пропускайте этапы, поскольку это лишь тормозит работу или сбивает с толку.

Вам может показаться, что вы уже все поняли, прочитав книгу или прослушав лекцию преподавателя, но для выработки навыков необходимы повторение и практика. Этим программирование напоминает спорт, музыку, танцы и любое другое занятие, требующее упорных тренировок и репетиций. Представьте себе музыканта, который репетирует от случая к случаю. Можно себе представить, как он играет. Постоянная практика — а для профессионала это означает непрерывную работу

на протяжении всей жизни — это единственный способ развития и поддержания профессиональных навыков.

Итак, никогда не пропускайте заданий, как бы вам этого ни хотелось; они играют важную роль в процессе обучения. Просто начинайте с первого шага и продолжайте, постоянно перепроверяя себя.

Не беспокойтесь, если не поймете все нюансы используемого синтаксиса, и не стесняйтесь просить помощи у преподавателей или друзей. Работайте, выполняйте все задания и большинство упражнений, и со временем все прояснится.

Итак, вот первое задание.

1. Откройте приложение B и выполните все шаги, необходимые для настройки проекта. Создайте пустой консольный проект на C++ под названием **hello_world**.
2. Введите в файл **hello_world.cpp** приведенные ниже строки, сохраните его в рабочем каталоге и включите в проект **hello_world**.

```
#include "std_lib_facilities.h"
int main() // Программы на C++ начинаются с выполнения функции
           // main
{
     cout << "Hello, World!\n"; // вывод строки "Hello, World!"
     keep_window_open(); // ожидание ввода символа
     return 0;
}
```

   Вызов функции **keep_window_open()** нужен при работе под управлением некоторых версий операционной системы Windows для того, чтобы окно не закрылось прежде, чем вы прочитаете строку вывода. Это особенность вывода системы Windows, а не языка C++. Для того чтобы упростить разработку программ, мы поместили определение функции **keep_window_open()** в файл **std_lib_facilities.h**. Как найти файл **std_lib_facilities.h**? Если вы этого не знаете, спросите преподавателя. Если знаете, то загрузите его с сайта **www.stroustrup.com/Programming**. А что, если у вас нет учителя и доступа к веб? В этом (и только в этом) случае замените директиву **#include** строками

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;
inline void keep_window_open() { char ch; cin>>ch; }
```

   В этих строках стандартная библиотека используется непосредственно. Подробности этого кода изложены в главе 5 и разделе 8.7.

3. Скомпилируйте и выполните программу “Hello, World!”. Вполне вероятно, что у вас это сразу не получится. Очень редко первая попытка использовать новый язык программирования или новую среду разработки программ завершается успехом. Найдите источник проблем и устраните его! В этот момент целесообраз-

но заручиться поддержкой более опытного специалиста, но перед этим следует убедиться, что вы сами сделали все, что могли.

- Возможно, вы нашли несколько ошибок и исправили их. На этом этапе следует поближе ознакомиться с тем, как компилятор находит ошибки и сообщает о них программисту! Посмотрите, как отреагирует компилятор на шесть ошибок, сделанных в разделе 2.3. Придумайте еще как минимум пять ошибок в вашей программе (например, пропустите вызов функции **`keep_window_open()`**, наберите ее имя в верхнем регистре или поставьте запятую вместо точки с запятой) и посмотрите, что произойдет при попытке скомпилировать и выполнить эту программу.

Контрольные вопросы

Основная идея контрольных вопросов — дать вам возможность выяснить, насколько хорошо вы усвоили основные идеи, изложенные в главе. Вы можете найти ответы на эти вопросы в тексте главы; это нормально и вполне естественно, можете перечитать все разделы, и это тоже нормально и естественно. Но если даже после этого вы не можете ответить на контрольные вопросы, то вам следует задуматься о том, насколько правильный способ обучения вы используете? Возможно, вы слишком торопитесь. Может быть, имеет смысл остановиться и попытаться поэкспериментировать с программами? Может быть, вам нужна помощь друга, с которым вы могли бы обсуждать возникающие проблемы?

1. Для чего предназначена программа “Hello, World!”?
2. Назовите четыре части программы.
3. Назовите функцию, которая должна существовать в каждой программе, написанной на языке C++.
4. Для чего предназначена строка **`return 0`** в программе “Hello,World!”?
5. Для чего предназначен компилятор?
6. Для чего предназначена директива **`#include`**?
7. Что означает расширение **`.h`** после имени файла в языке C++?
8. Что делает редактор связей?
9. В чем заключается различие между исходным и объектным файлом?
10. Что такое интегрированная среда разработки и для чего она предназначена?
11. Если вам все понятно, то зачем нужны упражнения?

Обычно контрольный вопрос имеет ясный ответ, явно сформулированный в главе. Однако иногда мы включаем в этот список вопросы, связанные с информацией, изложенной в других главах и даже в других книгах. Мы считаем это вполне допустимым; для того чтобы научиться писать хорошие программы и думать о последствиях их использования, мало прочитать одну главу или книгу.

Термины

Приведенные термины входят в основной словарь по программированию и языку C++. Если вы хотите понимать, что люди говорят о программировании, и озвучивать свои собственные идеи, следует понимать их смысл. Можете пополнять этот словарь самостоятельно, например, выполнив упр. 5

`#include`	библиотека	компилятор
//	вывод	объектный код
<<	выполняемый	ошибка на этапе компиляции
C++	заголовок	программа
`cout`	инструкция	редактор связей
IDE	исходный код	функция
`main()`	комментарий	

Упражнения

Мы приводим задания отдельно от упражнений; прежде чем приступать к упражнениям, необходимо выполнить все задания. Тем самым вы сэкономите время.

1. Измените программу так, чтобы она выводила две строки:

   ```
   Hello, programming!
   Here we go!
   ```

2. Используя приобретенные знания, напишите программу, содержащую инструкции, с помощью которых компьютер нашел бы ванную на верхнем этаже, о которой шла речь в разделе 2.1. Можете ли вы указать большее количество шагов, которые подразумевают люди, а компьютер — нет? Добавьте эти команды в ваш список. Это хороший способ научиться думать, как компьютер. Предупреждаем: для большинства людей “иди в ванную” — вполне понятная команда. Для людей, у которых нет собственного дома или ванной (например, для неандертальцев, каким-то образом попавших в гостиную), этот список может оказаться *очень* длинным. Пожалуйста, не делайте его больше страницы. Для удобства читателей можете изобразить схему вашего дома.
3. Напишите инструкции, как пройти от входной двери вашего дома до двери вашей аудитории (будем считать, что вы студент; если нет, выберите другую цель). Покажите их вашему другу и попросите уточнить их. Для того чтобы не потерять друзей, неплохо бы сначала испытать эти инструкции на себе.
4. Откройте хорошую поваренную книгу и прочитайте рецепт изготовления булочек с черникой (если в вашей стране это блюдо является экзотическим, замените его каким-нибудь более привычным). Обратите внимание на то, что, несмотря на небольшое количество информации и инструкций, большинство людей вполне способны выпекать эти булочки, следуя рецепту. При этом никто не считает этот рецепт сложным и доступным лишь профессиональным поварам или искусным кулинарам. Однако, по мнению автора, лишь некоторые упражнения из нашей

книги можно сравнить по сложности с рецептом по выпечке булочек с черникой. Удивительно, как много можно сделать, имея лишь небольшой опыт!

- Перепишите эти инструкции так, чтобы каждое отдельное действие было указано в отдельном абзаце и имело номер. Подробно перечислите все ингредиенты и всю кухонную утварь, используемую на каждом шаге. Не пропустите важные детали, например желательную температуру, предварительный нагрев духовки, подготовку теста, время выпекания и средства защиты рук при извлечении булочек из духовки.
- Посмотрите на эти инструкции с точки зрения новичка (если вам это сложно, попросите об этом друга, ничего не понимающего в кулинарии). Дополните рецепт информацией, которую автор (разумеется, опытный кулинар) счел очевидной.
- Составьте словарь использованных терминов. (Что такое противень? Что такое предварительный разогрев? Что подразумевается под духовкой?)
- Теперь приготовьте несколько булочек и насладитесь результатом.

5. Напишите определение каждого из терминов, включенных в раздел "Термины". Сначала попытайтесь сделать это, не заглядывая в текст главы (что маловероятно), а затем перепроверьте себя, найдя точное определение в тексте. Возможно, вы обнаружите разницу между своим ответом и нашей версией. Можете также воспользоваться каким-нибудь доступным глоссарием, например, размещенным на сайте `www.research.att.com/~bs/glossary.html`. Формулируя свое описание, вы закрепите полученные знания. Если для этого вам пришлось перечитать главу, то это только на пользу. Можете пересказывать смысл термина своими словами и уточнять его по своему разумению. Часто для этого полезно использовать примеры, размещенные после основного определения. Целесообразно записывать свои ответы в отдельный файл, постепенно добавляя в него новые термины.

Послесловие

Почему программа "Hello, World!" так важна? Ее цель — ознакомить вас с основными инструментами программирования. Мы стремились использовать для этого максимально простой пример.

Мы разделяем обучение на две части: сначала изучаем основы новых инструментов на примере тривиальных программ, а затем исследуем более сложные программы, уже не обращая внимания на инструменты, с помощью которых они написаны. Одновременное изучение инструментов программирования и языка программирования намного сложнее, чем овладение этими предметами по отдельности. Этот подход, предусматривающий разделение сложной задачи на ряд более простых задач, не ограничивается программированием и компьютерами. Он носит универсальный характер и используется во многих областях, особенно там, где важную роль играют практические навыки.

3 Объекты, типы и значения

“Фортуна благоволит подготовленному уму”.

Луи Пастер (Louis Pasteur)

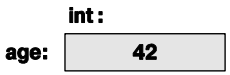
В этой главе излагаются основы хранения и использования данных в программе. Сначала мы сосредоточим внимание на вводе данных с клавиатуры. После введения основных понятий объектов, типов, значений и переменных рассмотрим несколько операторов и приведем много примеров использования переменных типов `char`, `int`, `double` и `string`.

В этой главе...

3.1. Ввод

Программа “Hello, World!” просто записывает текст на экран. Она осуществляет вывод. Она ничего не считывает, т.е. не получает ввода от пользователя. Это довольно скучно. Реальные программы, как правило, производят результаты на основе каких-то данных, которые мы им даем, а не делают одно и то же каждый раз, когда мы их запускаем.

Для того чтобы считать данные, нам необходимо место, куда можно ввести информацию; иначе говоря, нам нужно какое-то место в памяти компьютера, чтобы разместить на нем то, что мы считаем. Мы называем такое место объектом. Объект — это место в памяти, имеющее тип, который определяет вид информации, разрешенной для хранения. Именованный объект называется переменной. Например, строки символов вводятся в переменные типа **string**, а целые числа — в переменные типа **int**. Объект можно интерпретировать как “коробку”, в которую можно поместить значение, имеющее тип объекта.



Например, на рисунке изображен объект типа **int** с именем **age**, содержащий целое число 42. Используя строковую переменную, мы можем считать строку с устройства ввода и вывести ее на экран, как показано ниже.

```
// считать и записать имя
#include "std_lib_facilities.h"
int main()
{
  cout << "Пожалуйста, введите ваше имя (затем нажмите 'enter'):\n";
  string first_name; // first_name — это переменная типа string
  cin >> first_name; // считываем символы в переменную first_name
  cout << "Hello, " << first_name << "!\n";
}
```

Директива **#include** и функция **main()** известны нам из главы 2. Поскольку директива **#include** необходима во всех наших программах (вплоть до главы 12), мы отложим ее изучение, чтобы не запутывать ситуацию. Аналогично иногда мы будем

демонстрировать код, который работает, только если поместить его в тело функции **main()** или какой-нибудь другой.

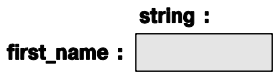
```
cout << "Пожалуйста, введите ваше имя (затем нажмите 'enter'):\n";
```

Будем считать, что вы понимаете, как включить этот код в полную программу, чтобы провести ее тестирование.

Первая строка функции **main()** просто выводит на экран сообщение, предлагающее пользователю ввести свое имя. Такое сообщение называется *приглашением* (prompt), поскольку оно предлагает пользователю предпринять какое-то действие. Следующие строки определяют переменную типа **string** с именем **first_name**, считывают данные с клавиатуры в эту переменную и выводят на экран слово Hello. Рассмотрим эти строки по очереди.

```
string first_name; // first_name — это переменная типа string
```

Эта строка выделяет участок памяти для хранения строки символов и присваивает ему имя **first_name**.

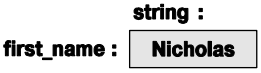


Инструкция, вводящая новое имя в программе и выделяющая память для переменной, называется *определением*.

Следующая строка считывает символы с устройства ввода (клавиатуры) в переменную:

```
cin >> first_name; // считываем символы в переменную first_name
```

Имя **cin** относится к стандартному потоку ввода (читается как “си-ин” и является аббревиатурой от character input), определенному в стандартной библиотеке. Второй операнд оператора **>>** (“ввести”) определяет участок памяти, в который производится ввод. Итак, если мы введем некое имя, например **Nicolas**, а затем выполним переход на новую строку, то строка “**Nicolas**” станет значением переменной **first_name**.



Переход на новую строку необходим для того, чтобы привлечь внимание компьютера. Пока переход на новую строку не будет выполнен (не будет нажата клавиша <Enter>), компьютер просто накапливает символы. Эта “отсрочка” дает нам шанс передумать, стереть некоторые символы или заменить их другими перед тем, как нажать клавишу <Enter>. Символ перехода на новую строку не является частью строки, хранящейся в памяти.

Введя входную строку в переменную **first_name**, можем использовать ее в дальнейшем.

```
cout << "Hello, " << first_name << "!\n";
```

Эта строка выводит на экран слово **Hello**, за которым следует имя **Nicolas** (значение переменной **first_name**) с восклицательным знаком (**!**) и символом перехода на новую строку экрана (**'\n'**).

```
Hello, Nicolas!
```

Если бы мы любили повторяться и набирать лишний текст, то разбили бы эту строку на несколько инструкций.

```
cout << "Hello, ";
cout << first_name;
cout << "!\n";
```

Однако мы не страдаем графоманией и, что еще важнее, — очень не любим лишние повторы (поскольку любой повтор создает возможность для ошибки), поэтому объединили три оператора вывода в одну инструкцию.

Обратите внимание на то, что мы заключили выражение **Hello** в двойные кавычки, а не указали имя **first_name**. Двойные кавычки используются для работы с литеральными строками. Если двойные кавычки не указаны, то мы ссылаемся на нечто, имеющее имя.

```
cout << "Имя" << " — " << first_name;
```

Здесь строка "Имя" представляет собой набор из трех символов, а имя **first_name** позволяет вывести на экран значение переменной **first_name**, в данном случае **Nicolas**. Итак, результат выглядит следующим образом:

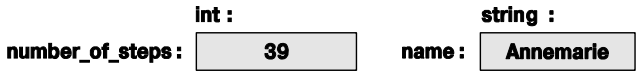
```
Имя — Nicolas
```

3.2. Переменные

В принципе, не имея возможности хранить данные в памяти так, как показано в предыдущем примере, с помощью компьютера невозможно сделать ничего интересного. Место, в котором хранятся данные, называют *объектами*. Для доступа к объекту необходимо знать его имя. Именованный объект называется *переменной* и имеет конкретный *тип* (например, **int** или **string**), определяющий, какую информацию можно записать в объект (например, в переменную типа **int** можно записать число **123**, а в объект типа **string** — строку символов **"Hello, World!\n"**), а также какие операции к нему можно применять (например, переменные типа **int** можно перемножать с помощью оператора *, а объекты типа **string** можно сравнивать с помощью оператора **<=**). Данные, записанные в переменные, называют *значениями*. Инструкция, определяющая переменную, называется (вполне естественно) *определением*, причем в определении можно (и обычно желательно) задавать начальное значение переменной. Рассмотрим следующий пример:

```
string name = "Annemarie";
int number_of_steps = 39;
```

Эти переменные можно изобразить следующим образом:



Мы не можем записывать в переменную значение неприемлемого типа.

```
string name2 = 39; // ошибка: 39 — это не строка
int number_of_steps = "Annemarie"; // ошибка: "Annemarie" —
                                   // не целое число
```

Компилятор запоминает тип каждой переменной и позволяет вам использовать переменную лишь так, как предусмотрено ее типом, указанным в определении.

В языке C++ предусмотрен довольно широкий выбор типов (см. раздел A.8). Однако можно создавать прекрасные программы, обходясь лишь пятью из них.

```
int number_of_steps = 39;  // int — для целых чисел
double flying_time = 3.5;  // double — для чисел с плавающей точкой
char decimal_point = '.';  // char — для символов
string name = "Annemarie"; // string — для строк
bool tap_on = true;        // bool — для логических переменных
```

Ключевое слово **double** используется по историческим причинам: оно является сокращением от выражения “число с плавающей точкой и двойной точностью” (“double precision floating point.”) Числом с плавающей точкой в компьютерных науках называют действительное число.

Обратите внимание на то, что каждый из этих типов имеет свой характерный способ записи.

```
39          // int: целое число
3.5         // double: число с плавающей точкой
'.'         // char: отдельный символ, заключенный в одинарные кавычки
"Annemarie" // string: набор символов, выделенный двойными кавычками
true        // bool: либо истина, либо ложь
```

Иначе говоря, последовательность цифр (например, **1234**, **2** или **976**) означает целое число, отдельный символ в одинарных кавычках (например, **'1'**, **'@'** или **'x'**) означает символ, последовательность цифр с десятичной точкой (например, **1.234**, **0.12** или **.98**) означает число с плавающей точкой, а последовательность символов, заключенных в двойные кавычки (например, **"1234"**, **"Howdy!"** или **"Annemarie"**), обозначает строку. Подробное описание литералов приведено в разделе A.2.

3.3. Ввод и тип

Операция ввода **>>** (“извлечь из”) очень чувствительна к типу данных, т.е. она считывает данные в соответствии с типом переменной, в которую производится запись. Рассмотрим пример.

```
// ввод имени и возраста
int main()
{
     cout << "Пожалуйста, введите свое имя и возраст\n";
     string first_name; // переменная типа string
     int age;           // переменная типа integer
     cin >> first_name; // считываем значение типа string
     cin >> age;        // считываем значение типа integer
```

```
    cout << "Hello, " << first_name << " (age " << age << ")\n";
}
```

Итак, если вы наберете на клавиатуре **Carlos 22**, то оператор **>>** считает значение **Carlos** в переменную **first_name**, число **22** — в переменную **age** и выведет на экран следующий результат.

```
Hello, Carlos (age 22)
```

Почему вся строка **Carlos 22** не была введена в переменную **first_name**? Потому что по умолчанию считывание строк прекращается, как только будет обнаружен так называемый *разделитель* (whitespace), т.е. пробел, символ перехода на новую строку или символ табуляции. В других ситуациях разделители по умолчанию игнорируются оператором **>>**. Например, перед считываемым числом можно поместить сколько угодно пробелов; оператор **>>** пропустит их и считает число.

Если вы наберете на клавиатуре строку **22 Carlos**, то увидите нечто неожиданное. Число **22** будет считано в переменную **first_name**, так как, в конце концов, **22** — это тоже последовательность символов. С другой стороны, строка **Carlos** не является целым числом, поэтому она не будет считана. В результате на экран будет выведено число **22**, за которым будет следовать строковый литерал **"(age"** и какое-то случайное число, например **-96739** или **0**. Почему? Потому что вы не указали начальное значение переменной **age** и впоследствии в нее ничего не записали. В итоге получили какое-то "мусорное значение", хранившееся в участке памяти в момент запуска программы. В разделе 10.6 мы покажем способ исправления ошибок, связанных с форматом ввода. А пока просто инициализируем переменную **age** так, чтобы она имела определенное значение и ввод осуществлялся успешно.

```
// ввод имени и возраста (2-я версия)
int main()
{
  cout << "Пожалуйста, введите свое имя и возраст\n";
  string first_name = "???";// переменная типа string
                            // ("???" означает, что "имя неизвестно")
  int age = -1;             // переменная типа int (-1 означает
                            // "возраст неизвестен")
  cin >> first_name >> age;  // считываем строку, а затем целое число
  cout << "Hello, " << first_name << " (age " << age << ")\n";
}
```

Теперь ввод строки **22 Carlos** приводит к следующему результату:

```
Hello, 22 (age -1)
```

Обратите внимание на то, что мы можем одним оператором ввода ввести одновременно несколько значений, а одним оператором вывода — вывести их на экран. Кроме того, оператор **<<**, как и оператор **>>**, чувствителен к типу, поэтому можем вывести переменную **age** типа **int** вместе со строковой переменной **first_name** и строковыми литералами **"Hello, "**, **" (age "** и **"\n"**.

Ввод объекта типа **string** с помощью оператора >> (по умолчанию) прекращается, когда обнаруживается разделитель; иначе говоря, оператор >> считывает отдельные слова. Однако иногда нам необходимо прочитать несколько слов. Для этого существует много возможностей. Например, можно прочитать имя, состоящее из двух слов.

```
int main()
{
     cout << "Пожалуйста, введите свое имя и отчество\n";
     string first;
     string second;
     cin >> first >> second; // считываем две строки
     cout << "Hello, " << first << ' ' << second << '\n';
}
```

Здесь мы просто использовали оператор >> дважды, применив его к каждому из слов. Если требуется вывести эти слова на экран, то между ними следует вставить пробел.

➘ ПОПРОБУЙТЕ

Запустите программу "имя и возраст". Измените ее так, чтобы она выводила возраст, измеренный месяцами: введите возраст, выраженный в годах, и умножьте это число на 12 (используя оператор *). Запишите возраст в переменную типа **double**, чтобы дети могли гордиться, что им пять с половиной, а не пять лет.

3.4. Операции и операторы

Кроме значений, которые могут храниться в переменной, ее тип определяет также операции, которые можно применять к ней, и их смысл. Рассмотрим пример.

```
int count;
cin >> count; // оператор >> считывает целое число в объект count
string name;
cin >> name;  // оператор >> считывает строку в переменную name
int c2 = count+2; // оператор + складывает целые числа
string s2 = name + " Jr. "; // оператор + добавляет символы
int c3 = count-2;           // оператор - вычитает целые числа
string s3 = name - "Jr. ";  // ошибка: оператор - для строк не определен
```

☒ Под ошибкой мы подразумеваем то, что компилятор откажется компилировать программу, пытающуюся вычитать строки. Компилятор точно знает, какие операции можно применять к каждой из переменных, и, следовательно, может предотвратить любые ошибки. Однако компилятор не знает, какие операции имеют смысл для тех или иных переменных, поэтому охотно допускает выполнение легальных операций, приводящих к абсурдным результатам. Рассмотрим пример.

```
int age = -100;
```

Очевидно, что человек не может иметь отрицательный возраст (хотя почему бы и нет?), но никто не сказал компилятору об этом, поэтому он успешно создаст код