2 Hello, World!

Программирование изучается
с помощью написания программ.
— Брайан Керниган (Brian Kernighan)

В этой главе приводится простейшая программа на языке C++, которая что-то делает. Цели создания этой программы следующие.

- Дать вам возможность поработать с интегрированной средой разработки программ.
- Дать вам почувствовать, как можно заставить компьютер сделать что-то для вас.

Словом, в этой главе мы представим понятие программы, идею о преобразовании программ с помощью компилятора из текстовой формы, понятной для человека, в машинные команды для последующего выполнения компьютером.

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 от него требуется. 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 уже программисты. В конце концов, мы множество раз получали описательные задания, которые должны были выполнить, например "как идти в школу на уроки" или "как поджарить мясо в микроволновой печи". Разница между такими описаниями и 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 нужный кабинет на верхнем этаже. Однако если вы внимательно посмотрите на эти простые инструкции, то увидите, что они являются грамматически неточными и неполными. Человек может легко восполнить этот недостаток. Представим, например, что вы вошли в здание и спрашиваете, как найти кабинет нужного вам человека. Отвечающий вам совершенно не обязан говорить, чтобы вы прошли через дверь, открыв ее поворотом ручки (а не выбив ее ногой), не толкнули в холле вазон с цветами и т.д. Вам также никто не скажет, чтобы вы были осторожны, поднимаясь по лестнице, и постучали, добравшись до нужной двери. Как открыть дверь в кабинет, прежде чем войти в него, вам, вероятно, также не будут рассказывать.

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 же 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;
}
```

Рассматривайте этот текст как набор команд, которые должен выполнить компьютер; это 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 выводит 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 линия, показывающая, где будет выведен следующий символ.

В языке C++ строковые литералы выделяются двойными кавычками ("); т.е. **"Hello, World!\n"** — это строка символов. **\n** — это "специальный символ", означающий переход на новую строку. Имя **cout** относится к стандартному потоку вывода. Символы, "помещенные в поток **cout**" с помощью оператора вывода **<<**, будут отображены на экране. Имя **cout** произносится как "see-out", но является аббревиатурой от "**c**haracter **out**put stream" ("поток вывода символов"). Аббревиатуры довольно широко распространены в программировании. Естественно, все эти сокращения на первых порах могут показаться неудобными 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 комментарий 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 программ являются другие программисты. Поэтому создание программ можно считать формой общения между людьми. В действительности имеет смысл главными читателями своей программы считать людей: если они с трудом понимают, что вы написали, то вряд ли программа когда-нибудь станет правильной. А потому нельзя забывать, что код предназначен для чтения — необходимо делать все, чтобы программа легко читалась. В любом случае комментарии нужны только людям; компьютеры их полностью игнорируют.

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

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

Такие комментарии очень полезны, так как по исходному тексту программы можно понять, что она делает, но нельзя выяснить, что 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 начальной части программы и напоминают, что мы пытаемся сделать в данной программе.

Строка

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

представляет собой директиву **#include**. Она заставляет компьютер сделать доступными ("включить") функциональные 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 используем только стандартный поток вывода **cout** и оператор вывода **<<**. Файл, включаемый в программу с помощью директивы **#include**, обычно имеет суффикс **.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 узнает, с чего начинать выполнение программы? Он ищет функцию с именем **main** и начинает выполнять инструкции, содержащиеся в ней. Вот как выглядит функция **main** нашей программы:

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

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

- *Тип возвращаемого значения*; в этой функции — тип **int** (т.е. integer, 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 в качестве имени чего-нибудь иного (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 и имеет вид **()**.
- *Тело функции*, заключенное в фигурные скобки **{}** и перечисляющее действия (*инструкции*), которые функция должна выполнить.

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

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

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

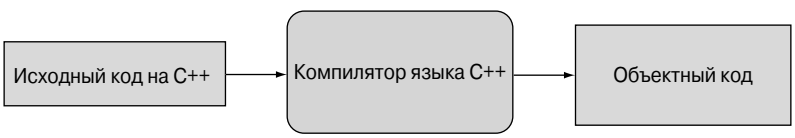
```
cout << "Hello, World!\n"; // Вывод "Hello,World!"
return 0;
```

Сначала она выводит на экран строку **Hello, World!**, а затем возвращает значение 0 (нуль) в точку вызова. Поскольку функция **main()** вызывается системой, мы возвращаемое значение использовать не будем. 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 некоторое действие и не являющаяся директивой **#include** (или другой директивой препроцессора; см. разделы 4.4 и A.17), называется *инструкцией* (statement).

2.3. Компиляция

C++ — компилируемый язык. Это означает, 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 *исходным текстом программы*, а то, что выполняет компьютер, называется *выполняемым*, *объектным* или *машинным кодом*. Обычно файлы с исходным кодом программы на языке C++ имеют суффикс **.cpp** (например, **hello_world.cpp**) или **.h** (например, **std_lib_facilities.h**), а файлы с объектным кодом имеют суффикс **.obj** (в Windows) или **.o** (в 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 одна 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 }.

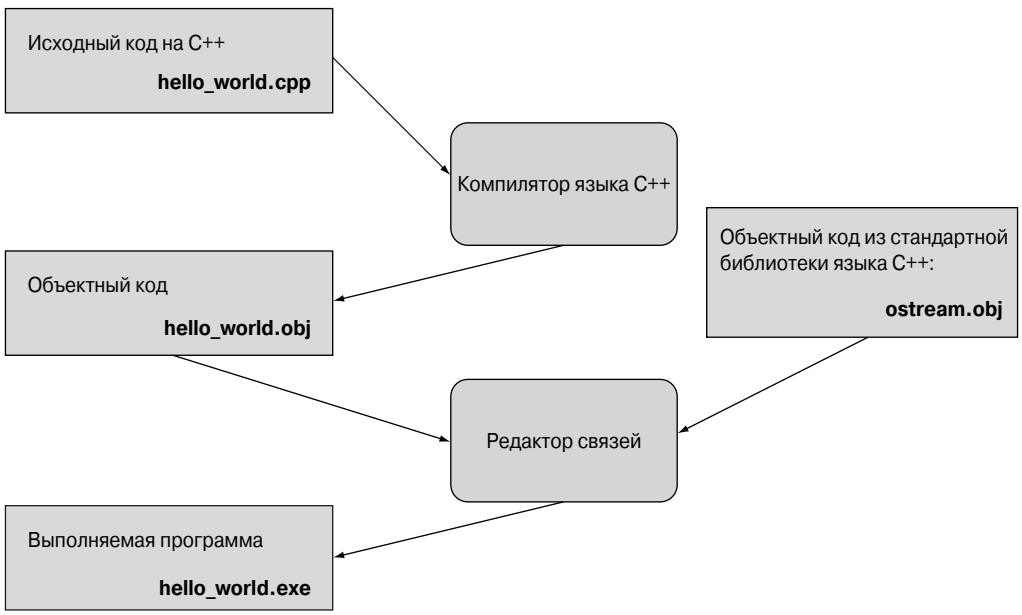
Для чего мы посвятили две страницы и несколько минут вашего драгоценного времени демонстрации тривиальных примеров, содержащих тривиальные ошибки? Для того, чтобы в будущем вы не тратили много времени на поиск ошибок в исходном тексте программы. Большую часть времени программисты ищут ошибки в своих программах. В конце концов, если вы убеждены, что некий код является правильным, то вы, скорее всего, обратитесь к анализу некоторого другого кода, чтобы не тратить время зря. 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 нередко будет вас раздражать. Иногда будет казаться, 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 уж 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 (это не человек!) и очень придирчив к деталям. Поскольку здравый смысл ему чужд, 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, но при этом результат оказался бы для вас совершенно неожиданным, вы провели бы очень много времени, пытаясь понять, почему программа не делает то, что требуется. Компилятор — наш друг, позволяющий избежать множества проблем, которые мы сами же создаем. Никогда не забывайте об этом: компилятор — не просто ваш друг; пожалуй, когда вы занимаетесь программированием, — это наилучший ваш 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 скомпонованы в единое целое, образуя выполняемый файл. Программа, связывающая эти части в одно целое, называется (вполне ожидаемо) компоновщиком или *редактором связей*.



Заметьте, что объектные и выполняемые коды являются *не* переносимыми из одной системы в другую. Например, когда вы компилируете программу для Windows, то получите объектный код именно для Windows, и этот код не будет работать в Linux.

Библиотека — это просто некоторый код (обычно написанный другими), доступ к которому можно получить с помощью объявлений, содержащихся в файле, включенном директивой `#include`. *Объявление* — это инструкция программы, указывающая, как можно использовать некоторый фрагмент кода; объявления будут подробно описаны позднее (см., например, раздел 4.5.2).

Ошибки, обнаруженные компилятором, называются *ошибками времени компиляции*; ошибки, обнаруженные компоновщиком, называются *ошибками времени компоновки*, а ошибки, не обнаруженные на этих этапах и проявляющиеся при выполнении программы, называются *ошибками времени выполнения* или *логическими ошибками*. Как правило, ошибки времени компиляции легче понять и исправить, чем ошибки времени компоновки. В свою очередь, ошибки времени компоновки легче обнаружить и исправить, чем ошибки времени выполнения. 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 компьютере 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 IDE (interactive (integrated) development environment — интерактивные (интегрированные) среды разработки), которые также весьма популярны среди профессиональных программистов, то достаточно щелкнуть на соответствующей кнопке. Описание компиляции и редактирования связей приведено в приложении B.

Интегрированные среды разработки включают в себя редактор текстов, позволяющий, например, выделять разными цветами 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).

Работая с этой книгой, вы можете использовать любую систему, предоставляющую современную, соответствующую стандарту реализацию C++. Большинство из того, о чем мы говорим, с очень малыми модификациями справедливо для всех реализаций языка C++, и приводимый нами код будет работать везде. В нашей работе мы 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 важна последовательность пошагового выполнения действий, каждое из которых должно быть простым (и даже тривиальным). Пожалуйста, не умничайте и не пропускайте описанные шаги, 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 вы не понимаете все тонкости 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 текст файла **hello_world.cpp** в точности таким, как показано ниже, сохраните его в рабочем каталоге и включите его 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 учитесь с преподавателем, спросите у него. Если работаете самостоятельно, загрузите его с сайта **www.stroustrup.com/Programming**. А если у вас нет ни преподавателя, ни доступа к вебу? 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 непосредственно используется стандартная библиотека. Подождите до главы 5 и еще более подробного изложения в 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 целесообразно заручиться поддержкой более опытного специалиста, но при этом вы должны убедиться, что понимаете, что именно он сделал и почему, и сможете повторить эти действия в дальнейшем самостоятельно.

4. Возможно, вы столкнетесь с наличием ошибок в программе и будете должны их исправить. Тогда самое время познакомиться с тем, как ваш компилятор находит ошибки и сообщает о них программисту! Посмотрите, как отреагирует компилятор на шесть ошибок, описанных в разделе 2.3, внося их и пытаясь скомпилировать программу. Придумайте еще как минимум пять других 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 функции.
3. Назовите функцию, которая должна иметься в каждой программе 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 суффикс `.h` после имени файла в языке C++?
8. Что делает редактор связей?
9. В чем заключается различие между исходным и объектным файлами?
10. Что такое интегрированная среда разработки и для чего она предназначена?
11. Если в книге вам все понятно, то зачем нужна практическая работа?

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++. Чтобы понимать, что люди говорят о программировании, и озвучивать собственные идеи, следует понимать их смысл.

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

Можете пополнять этот словарь самостоятельно, выполняя приведенное ниже пятое упражнение для каждой прочитанной главы.

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 кабинет на верхнем этаже, о котором 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 указанное место” — вполне понятная команда. Для людей, которые никогда не видели современного строения (например, для перемещенных во времени неандертальцев), этот список может оказаться *очень* длинным. Пожалуйста, не делайте его больше страницы. Для удобства читателей можете изобразить схему строения.

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 по адресу **`www.stroustrup.com/glossary.html`**. Формулируя свое описание, вы закрепите полученные знания. Если для этого вам пришлось перечитать главу, то это пойдет вам 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. Так 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.