Опыт не требуется

Книги этой серии наглядно свидетельствуют, что новичка можно научить языку и хорошему стилю программирования, не повергая его в тоску и уныние.
– Лу Гринзоу, обозреватель Dr. Dobb's Journal.

Исходные тексты всех игр находятся на CD

Дирк Хенкеманс, Марк Ли

Программирование на C++

Premier Press
СИМВОЛ

C++ Programming

for the Absolute Beginner

Dirk Henkemans and Mark Lee

Premier Press

Опыт не требуется

Программирование на C++

Дирк Хенкеманс и Марк Ли

Символ

Санкт-Петербург
2005

Серия «Опыт не требуется»

Дирк Хенкеманс, Марк Ли

Программирование на C++

Перевод М. Зислиса

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Редактор	*А. Лосев*
Художник	*В. Гренда*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Хенкеманс Д., Ли М.
Программирование на C++. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 416 с., ил.
ISBN 5-93286-050-2

Для тех, кто мало знаком с программированием, но ищет хороший учебник по C++, эта книга станет идеальным выбором. Написанная профессиональными разработчиками и отличающаяся легким стилем изложения, она обучает принципам программирования на примерах создания простых игр. Прочитав ее, вы приобретете навыки, необходимые для создания более сложных программ на C++, и узнаете, как использовать их в реальных приложениях. Изучите многочисленные приемы, которые применимы не только к C++, но и к программированию в целом, поэтому полученные знания будут вам полезны при освоении других языков программирования.

Вы узнаете, что такое переменные и управляющие операторы, функции и объектно-ориентированное программирование, пространства имен и массивы. Научитесь программировать для Windows, создавать программы шифрования, отлаживать ошибки и грамотно обрабатывать исключения, эффективно использовать потоки и файлы, а также разрабатывать игры с помощью библиотеки DirectX.

ISBN 5-93286-050-2
ISBN 1-93184-143-8 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 19.11.2004. Формат 70x100 1/16 . Печать офсетная.
Объем 26 печ. л. Доп. тираж 1000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Всем детям двадцать первого века –
вы способны осуществить все,
что можете представить в своих мечтах.

Предисловие

Индустрия видеоигр является уникальной: она регулярно вбирает все новшества основных направлений развития компьютерных наук от трехмерной графики и искусственного интеллекта до теории операционных систем и проектирования баз данных. Если вы разрабатываете коммерческую игру, то рано или поздно столкнетесь с задачами, принадлежащими каждой из этих областей. Некоторые из задач могут потребовать применения специальных языков, но в конечном итоге есть только два языка, столь же привычных для игровой индустрии, как пицца, кофеиносодержащие напитки и критические дни в нашей жизни. Несколько коммерческих игр было написано и на Java (языке, весьма похожем на C++), но практически каждая игра, в которую вам приходилось играть, написана на C или C++. Неважно, работает она на PC, игровой консоли или вовсе на игровом автомате – все шансы за то, что ее сердцем является код C или C++. Даже в случаях, когда требования к производительности диктуют необходимость создания подпрограммы на ассемблере с целью повышения скорости работы, как правило, первый вариант этой подпрограммы пишется на C или C++.

За многие годы работы в этой индустрии я провел более сотни собеседований с претендентами на вакансии, связанные с программированием, и, кроме того, прочитал более тысячи резюме. При отборе я непрестанно ищу у кандидатов комбинацию трех качеств. Первое качество – умение решать проблемы, постоянным источником которых служат непрекращающиеся изменения в технологиях и жесткая конкуренция в среде разработчиков игр. Как следствие, отточенные навыки разрешения проблем – не только роскошь, но и необходимость. Во-вторых, кандидат на должность должен иметь опыт работы во всем спектре компьютерных наук. Ведь будучи хорошим специалистом в одной области, всегда можно столкнуться с проблемой, решение которой лежит за пределами компетенции. И наконец, я требую отличного владения C/C++. Для программиста эти языки равноценны кистям и краскам для художника. Это орудия труда, и значит, они должны быть идеально отточены.

Сегодня C++ повсеместно используется для обучения программированию, но так было не всегда. Я до сих пор помню, как познакомился с программированием на языке C. До этого момента мой опыт в программировании сводился к языкам Basic (на котором я написал свою первую игру), Pascal и Fortran. Но я слышал о C, и, по слухам, этот язык стоило изучать. Я с нетерпением ожидал начала следующего

курса по информатике: «Введение в языки программирования». Я полагал, что в этом курсе меня научат программировать на C, и ошибся. Единственная ссылка на язык C в этом курсе выглядела так: «Вот задание. Напишите программу на C. Сдайте ее в среду». Ну ничего, подумал я. Есть учебник по языку C. Однако выяснилось, что он был посвящен доступу к информации ОС UNIX из программ на языке C. В книге рассказывалось о том, как получать идентификаторы процессов и выполнять команды интерпретатора, и значит, она была для меня бесполезна, т. к. в ней не объяснялось, как прочитать файл или создать функцию.

Кое-как я ухитрился выполнить задание и даже приобрести знания в процессе. Это был не лучший способ изучить новый язык, но моя первая встреча с C++ выглядела еще хуже. Окончив университет, я поступил на работу. В мои задачи входило создание программного обеспечения для исследовательских проектов спортивного факультета. Один из проектов, доставшихся мне от предшественника, был завершен лишь наполовину и написан на языке C++. И снова мне пришлось учиться плавать в боевых условиях. На этот раз у меня был доступ к справочнику по функциям, в котором был описан синтаксис языка, но не рассматривались способы его применения. В то время я готов был пойти на преступление ради книги, которую вы держите в руках. Конечно, я преувеличиваю, но невозможно переоценить достоинства иного метода изучения C++, последовательного и доступного. Читая эту книгу, вспоминайте с сочувствием тех из нас, у кого не было столь замечательного учебника.

Scott Greig
Director of Programming
Bio Ware Corp.

Благодарности

В процессе создания книги участвуют многие замечательные люди, и эта книга не стала исключением. Хотя нам трудно осознать объемы времени и сил, вложенные в эту книгу, мы знаем, что эти объемы были внушительными.

Прежде всего, спасибо нашим родителям за то, что они нас вырастили, и за поддержку, которую мы всегда в них находим.

Спасибо нашему издательству, Premier Press, за эту книгу. Отдельной благодарности заслуживает Мелоди Лейн (Melody Layne), менеджер по работе с авторами, которая поверила в нашу идею и поддержала ее. Мелоди прекрасно известно, что именно эту книгу мы всегда хотели увидеть на полке.

Мы выражаем искреннюю благодарность Грегу Перри (Greg Perry), координирующему редактору и техническому рецензенту. Спасибо за великолепные отклики, Грег, и за тщательную проверку кода в нашей книге.

Мельба Хоппер (Melba Hopper), выпускающий и литературный редактор, заслуживает отдельной страницы благодарностей. Ее рука касалась всех строк этой книги, изменяя и улучшая их. Мельба, ты замечательно с нами ладила, постоянно объясняла, что следует исправить в рукописи, и служила неиссякаемым источником информации и поддержки, в которых мы нуждались, чтобы продолжать работу. И самое главное, ты сделала процесс работы над книгой увлекательным. Спасибо!

Отдельное спасибо всем остальным, кто участвовал в подготовке публикации этой книги. Вот эти люди: Энди Харрис (Andy Harris), редактор серии «Для начинающих», Эрли Хартман (Arlie Hartman), автор компакт-диска; Шон Морнингстар (Shawn Morningstar), дизайнер-верстальщик; художники из Argosy, которые превратили наши наброски в нечто удобоваримое; Дженни Смит (Jeannie Smith), корректор; а также Джонна ВанХуз Динс (Johnna VanHoose Dinse), автор указателя. Все вы сыграли важную роль в достижении полученного результата.

Мы восхваляем Скотта Грейга (Scott Greig), ведущего программиста BioWare Corp. и автора предисловия к этой книге. Скотт, ты наш кумир. Не будь тебя, кем бы мы стремились стать?

И наконец, отдельное спасибо Нолану Барду (Nolan Bard), который в четыре утра помогал нам закончить книгу в срок, а также Джеки Нэги (Jackie Nagy) за его поддержку и за то, что не оставил Дирка, когда тот писал книгу.

Об авторах

Дирк Хенкеманс – создатель любительских руководств по разработке игр и автор веб-сайта EastCoastGames.com. Он также является одним из основателей FireStorm Studios, растущей компании, специализирующейся на мультимедиа-приложениях.

Марк Ли – второй основатель FireStorm Studios, работал компьютерным консультантом и оператором текстовой пользовательской сети. Бегло говорит на C, Java, C++, Visual Basic, разнообразных диалектах ассемблера и систем управления базами данных.

Введение

C++ является одним из наиболее широко применяемых языков программирования, индустриальным стандартом для создания приложений всевозможного рода. Кроме того, это очень рациональный язык, позволяющий использовать ресурсы более эффективно, чем Visual Basic или Delphi. По большому счету, благодаря функциональности и стилю C++ может оказаться единственным из языков, не ориентированных на работу с веб-средой, который вам когда-либо понадобится.

Мы решили обучать читателей C++ на примерах создания игр прежде всего потому, что для многих людей первое знакомство с компьютером связано с играми. А самое главное, это замечательный способ научиться программировать – игры учат отображать интерфейс на экране, обрабатывать команды пользователя и информацию. В конечном итоге они сочетают в себе искусство и науку, проникая в умы творческие и логичные, служат источником визуальных, звуковых и душевных переживаний для программистов и пользователей.

Читая книгу, вы изучите многочисленные приемы программирования, которые применимы не только к C++, но и к программированию в целом. Эти распространенные приемы упростят изучение других языков и создание разнообразных приложений (не только игр).

Структура книги

Сложность материала книги возрастает постепенно – от обычных текстовых программ к играм с полноценной графикой. Новичкам в программировании рекомендуем читать главы в естественной последовательности. Читатели, уже имеющие опыт написания программ, могут бегло пролистать первые шесть глав, которые посвящены основам, и перейти сразу к более сложным темам.

Концептуально книга разбита на четыре части (хотя это деление не соответствует последовательности глав). Первая часть (с главы 1 «Путешествие начинается» по главу 6 «Сложные типы данных») дает базовые знания, необходимые для программирования на C++. Темы в этих главах изложены в определенной последовательности, так что их рекомендуется читать подряд. Так, перед прочтением главы 5 «Боевые качества ООП», скорее всего, придется изучить главу 4 «Пишем функции».

Вторая часть книги (с главы 7 «Градостроение и пространства имен» по главу 11 «Ошибки и обработка исключений») содержит сложные темы C++. Эти главы можно читать в любом порядке.

Третью часть составляют главы с 12 по 14. Здесь читателям предстоит применить все знания, полученные ранее, сначала для разработки Windows-приложений (глава 12 «Программирование для Windows»), затем для работы с DirectX (глава 13 «DirectX») и наконец, для создания потрясающей игры о пиратах с помощью стандартных методов отрасли (глава 14 «Создаем пиратское приключение»).

Четвертая часть содержит приложения с дополнительной информацией, которая будет полезна для читателей.

В каком бы порядке вы ни читали книгу, помните, что существенную часть изучения C++ составляет собственный опыт написания программ. Чем больше программируешь, тем больше приобретаешь навыков в решении задач (очень важное умение в программировании, как станет понятно) и обнаружении ошибок в своем коде. И не исключено, что после многочисленных тренировок вы даже сможете вычислить значение числа π до миллионной цифры после запятой... в уме (хотя авторы книги не дают никаких гарантий этого)!

По мере чтения глав будут встречаться небольшие фрагменты кода, иллюстрирующие понятия, о которых идет речь. В конце каждой главы приводится полноценная игра, демонстрирующая ключевые идеи этой главы, а также резюме главы и набор упражнений, позволяющих испытать приобретенные знания. Мы надеемся, что читатели не поленятся испробовать эти игры и выполнить упражнения, поскольку они существенным образом помогут развить хватку в программировании. Ответы к заданиям приведены в приложении A, а копии всех полноценных программ содержатся на компакт-диске. Но мы настоятельно советуем хотя бы пытаться самостоятельно выполнять задания, не заглядывая в ответы (даже если нужна помощь). Задания достаточно короткие, так что их можно выполнять в компиляторе (опять же, это отличный способ набраться опыта).

Самое необходимое

Изучение программирования – великолепный способ воспользоваться вычислительной мощью компьютера. Но прежде чем вы начнете писать программы, вам понадобится следующее:

- Персональный компьютер с тактовой частотой процессора не менее 75 МГц.
- Операционная система, совместимая с DirectX: Microsoft Windows 95/98/2000, Windows ME/XP.
- Минимум 16 Мбайт оперативной памяти.

- По меньшей мере 125 Мбайт дискового пространства.
- Компилятор (например, CodeWarrior от MetroWerks).
- Устройство для чтения компакт-дисков.
- Знание, время и терпение. Информация, представленная в этой книге, позволит вам эффективно овладеть C++ и компилятором CodeWarrior (да и любым другим).

Специальные обозначения

Помимо полноценных игр и заданий в конце каждой главы книга содержит ряд специальных обозначений:

Примечания. Содержат дополнительную информацию по сложным темам.

Ловушки. Предупреждают об ошибках, которых следует избегать.

Приемы. Содержат советы, облегчающие и совершенствующие программирование.

Истории из жизни

Эти врезки содержат истории о программировании и информацию непосредственно с «передовой».

1

Путешествие начинается

Мысль о программировании может быть пугающей, но волноваться не стоит. Мы написали эту главу так, что вам не придется погружаться в тонкости программирования, чтобы начать писать программы. Глава начинается с рассказа о *CodeWarrior*, компиляторе C++. Затем мы перейдем к основам создания программ, а затем поиграем со строками и числами. С нашей помощью, используя свою изобретательность, вы очень скоро начнете писать собственные программы. Позже они будут постепенно становиться все более сложными, но, чтобы отправиться в путешествие, нужно с чего-то начать. *Ваше* приключение начинается здесь и сейчас! И его началом станут следующие этапы:

- Работа с компилятором CodeWarrior
- Создание кода
- Создание самой первой программы
- Изучение цикла разработки
- Работа с текстом
- Работа с числами

Работа с компилятором CodeWarrior

В этом разделе вы узнаете, как использовать CodeWarrior для создания программы на основе готового шаблона исходного текста. *Исходный текст* – это текст, представляющий собой определенный набор инструкций, которые будут выполняться компьютером. Исходный текст пишется не на русском языке, а на языке программирования. И хотя языков программирования существует великое множество, эта книга научит вас применять C++. Позже в этой главе мы расскажем, как писать исходные тексты программ (вернее, изменять текст, предоставляемый компилятором CodeWarrior).

CodeWarrior облегчает создание программ при помощи *интегрированной среды разработки* (Integrated Development Environment, IDE). Среда разработки является общим графическим интерфейсом для компилятора, навигации в каталогах, изменения настроек, а также *редактора исходных текстов* (то есть окна, в котором происходит редактирование и просмотр текстов программ). Когда мы начинали программировать, то пользовались бесплатным компилятором C++, и все настройки приходилось вводить в приглашении командной строки DOS. На это уходила масса времени. Среда разработки сама позаботится о настройках проектов и файлов, ускоряя и облегчая процесс создания программ.

Мы писали эту книгу, исходя из предположения, что читатели пользуются компилятором CodeWarrior Professional 5.0 (разработанным Metrowerks). Если это не так, ничего страшного. Большая часть сведений из этой и других глав будет полезна независимо от того, какой компилятор используется.

Пора начать наши поиски приключений в мире программирования. В процессе чтения разделов, посвященных созданию проекта в CodeWarrior и написанию кода, пробуйте воспроизводить наши действия на своем компьютере. Практика позволяет привыкнуть к незнакомым вещам.

Создаем новый проект

При первом запуске CodeWarrior выглядит так, как показано на рис. 1.1. Как видите, никакого волшебства (ну разве что совсем немного). CodeWarrior – это просто приложение, такое же как Microsoft Word и Netscape Navigator, лишь с той разницей, что CodeWarrior позволяет создавать другие приложения.

Чтобы создать новый проект C++, имея запущенный CodeWarrior, выполните следующие шаги (помните, что элементы меню, диалоговые окна и параметры могут отличаться для вашего компилятора):

1. В строке главного меню CodeWarrior щелкните по элементу File.
2. В появившемся меню выберите пункт New. Откроется диалоговое окно (рис. 1.2), позволяющее создать практически любой тип приложения.
3. На вкладке Project выберите строку «Win32 C/C++ Application Stationery» (Приложение C/C++ для платформы Win32).
4. В поле имени проекта (Project name) наберите имя **Hello**.
5. Нажмите OK. Откроется диалоговое окно нового проекта (рис. 1.3). Оно позволяет выбрать тип *среды времени выполнения* (*run-time environment*), в которой предполагается использовать готовое приложение. Среда времени выполнения определяется условиями, в которых работает программа. Чаще всего эти условия определяют

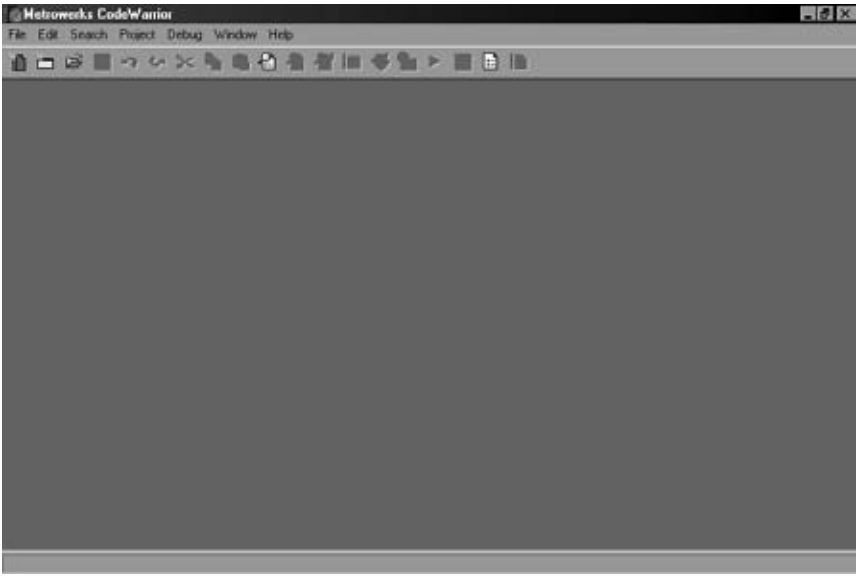

Рис. 1.1. *Так выглядит CodeWarrior при первом запуске*

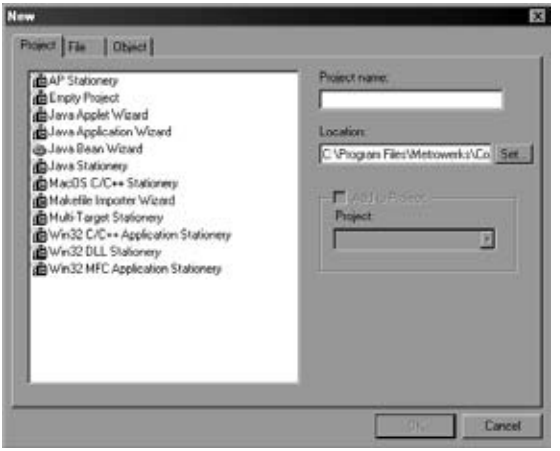

Рис. 1.2. *В окне New выберите тип создаваемого проекта*

лишь операционную систему, для которой будут компилироваться исходные тексты. Например, программы для DOS работают в среде DOS, а приложениям Win32 требуется 32-разрядная среда Windows. Компилятор оптимизирует программу для работы в конкретной среде. В конечном итоге это сокращает размер файлов и повышает скорость работы. Консольные приложения (C++ Console) являются приложениями Windows, работающими в окне, похожем на то, что используется в DOS для выполнения команд. Среда для работы таких приложений является вариантом среды DOS.

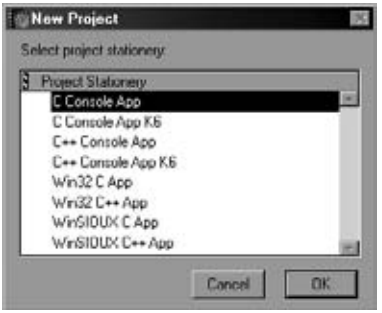

Рис. 1.3. *Диалоговое окно New Project позволяет оптимизировать программы*

6. Выберите из списка в диалоговом окне нового проекта элемент C++ Console App и нажмите OK. Как только CodeWarrior закончит создавать настройки для выбранного проекта, откроется новое окно. Это окно носит имя проекта – hello.mcp. Расширение .mcp связано с файлами проектов CodeWarrior. Файл проекта хранит все настройки проекта, а кроме того содержит перечень всех исходных файлов, составляющих проект. *Исходный файл* имеет текстовый формат и предназначен для хранения исходных текстов. Исходные файлы имеют расширение .cpp.

7. Чтобы раскрыть папку, щелкните по расположенному рядом значку +. В раскрывшейся папке проекта находится файл hello.cpp (расширение .cpp связано с исходными файлами C++).

8. Чтобы открыть файл, дважды щелкните по его имени. На экране появится окно, показанное на рис. 1.4.

Рис. 1.4. *В этом редакторе вы будете работать с исходными текстами*

Итак, вы добрались до сердца подземелий. В следующем разделе мы расскажем о загадочных письменах на рис. 1.4.

Истории из жизни

В начале 1980 года Бьерн Страуструп (Bjarne Stroustrup), работавший в AT&T Bell Laboratories, начал трудиться над языком C++. C++ официально получил свое имя в конце 1983 года, причем имя, подчеркивающее родство с предшественником, языком C. В октябре 1985 года появилась первая коммерческая версия языка и первое издание книги Страуструпа «Язык программирования C++» (The C++ Programming Language).[1]

В восьмидесятые годы язык C++ проходил доработку и в итоге приобрел индивидуальность, сохранив, по большей части, совместимость с языком C и его наиболее важные характеристики. В C++ по-прежнему доступны классические методологии структурного программирования, но дополнительно существует функциональность *объектно-ориентированного программирования* (или ООП; более подробно тема рассмотрена в главе 5 «Боевые качества ООП»). Своим происхождением C++ обязан и другим языкам – BCPL, Simula67, Algol68, Ada, Clu и ML, – из которых он позаимствовал некоторые свои достоинства. К счастью, C++ предоставляет все преимущества этих языков, а потому нет надобности изучать их.

В 1990 году комитет ANSI (*American National Standards Institute*, Американский национальный институт стандартов) под названием *X3J16* начал разработку стандарта C++. До публикации его окончательного варианта в ноябре 1997 года язык C++ успел получить широкое распространение и теперь является самым популярным языком для разработки приложений.

---

[1] Б. Страуструп, «Язык программирования C++». – Пер. с англ. – СПб: Невский Диалект, 2001 г.

Что такое исходный текст?

Программирование заключается в передаче компьютеру инструкций, заключенных в исходном коде. Синтаксис исходного кода зависит от того, какой язык программирования используется; в нашем случае это синтаксис языка C++. (*Синтаксис* – это набор правил, определяющих структуру языка.)

Почему нельзя на русском языке объяснить компьютеру, что следует делать? Дело в том, что русский язык очень сложный, и компьютер не сможет понять, что вы пытаетесь сказать. C++ является в некотором роде упрощенным вариантом английского языка, который способен понять CodeWarrior. В следующем разделе вы узнаете, что для непосредственного понимания компьютером слишком сложен даже язык C++. CodeWarrior дол-

*жен перевести код на C++ в **машинный код**. А пока просто сфокусируйтесь на создании исходного кода и существующих для этого правилах.*

C++ очень четко определяет структуру исходных текстов. К примеру, знаки препинания и порядок следования очень важны. Даже регистр символов важен, поскольку C++ *чувствителен к регистру*. Это означает, что компилятор различает прописные и строчные буквы (буквы *K* и *k* с его точки зрения являются различными).

Создание программы начинается с набора кода в редакторе исходных текстов. Затем компилятор преобразует исходный код в язык, на котором говорит компьютер (машинный код). Компилятор и редактор интегрированы в среду CodeWarrior IDE. Компилятор имеет первостепенное значение, и поэтому интегрированные среды разработки часто так и называют – *компиляторами*.

Каждая из строк кода отвечает за определенное действие, примерно как ингредиенты в рецепте. Компилятор разбивает строки кода на инструкции, которые называются *командами*. Команда является атомарной инструкцией для компьютера.

Ранее мы уже говорили, что CodeWarrior представляет собой интегрированную среду разработки и выполняет за вас многие задачи. Текст, который вы видите в окне hello.cpp, свидетельствует об этом. Этот текст – автоматически созданный код, который послужит основой для любой программы. О программе в целом можно думать как о мосте, который вы строите. Сгенерированные строки кода являются опорами. Опоры нужны любому мосту, но мост, состоящий из одних опор, бесполезен.

Взгляните на сгенерированный код: эта программа печатает на экране строку `This is a test` («Это проверка»):

```
#include <iostream>
using namespace std;       //introduces namespace std
int main( void )
{
     cout << "This is a test" ;
     return 0;
}
```

Эти строки содержат некоторые из существующих инструкций, которые могут передаваться компьютеру. Код можно изменять и совершенствовать (либо просто стереть и начать с нуля) с целью создания собственной программы. Но пока что будем считать, что это программа, которую вы написали. В этом случае для создания работающей программы необходимо скомпилировать и выполнить код.

Компиляция

Чтобы запустить программу, следует преобразовать написанный на C++ код в язык, который понимает компьютер. Именно на этом этапе в игру вступает компилятор. Вообразите, что вы эльф, а компьютер – гном. Чтобы компьютер понял ваши указания, необходимо преодолеть языковой барьер. Нужен переводчик, который говорит на языке гномов и на языке эльфов. В мире компьютеров таким переводчиком является *компилятор*. Как мы уже говорили, компилятор преобразует язык программирования в *машинный код*, на котором говорит компьютер. Однако перевод возможен только в одну сторону: компилятор не умеет переводить машинные коды в исходный текст.

Чтобы произвести компиляцию в CodeWarrior, выполните следующие шаги:

1. Из главного меню выберите Project (Проект) и затем пункт Compile (Компилировать). Откроется окно, озаглавленное Building Hello.mcp. (Этот шаг иногда занимает до нескольких минут, поэтому необходимо дождаться завершения работы компилятора.)

   Когда это окно активно, происходит преобразование файла в машинный код и проверка того, что автор программы не нарушил правила языка C++. Если вы изменили текст, созданный средой CodeWarrior, может появиться сообщение об ошибке. Если вы не меняли текст либо внесенные изменения не содержат ошибок, компилятор закроет окно Building Hello.mcp после завершения перевода программы в машинный код. Программа готова к запуску.

2. Из главного меню Project (Проект) выберите пункт Run (Выполнить).

   Откроется окно, в котором отображается вывод программы (рис. 1.5). Вы должны увидеть надпись: `This is a test`.

3. Нажмите любую клавишу, чтобы закрыть окно.

Рис. 1.5. *Вот что вы увидите после компиляции и выполнения программы*

Пишем первую программу

Постойте! Не торопитесь выключать компьютер, все будет гораздо интереснее. Сейчас вы напишете свою первую программу. Она будет печатать на экране строку `Здравствуй, мир!`. Этот проект поможет вам понять язык C++ и механизмы его работы. Но прежде следует выяснить, что уже сделал CodeWarrior.

Начинаем с типовой программы

В начале нового проекта CodeWarrior самостоятельно создает некий код. Этот код является основой для развития программы. Его можно удалить полностью или частично в зависимости от потребностей, но в большинстве случаев сгенерированный код помогает вам начать. Этот код называется *типовой программой*. Он должен выглядеть так:

```
#include <iostream>
using namespace std;       //introduces namespace std
int main( void )
{
     cout << "This is a test" ;
     return 0;
}
```

Рассмотрим строки по очереди. Первая строка сообщает компилятору, что мы будем пользоваться командами из библиотеки iostream. Это существующая библиотека, которая поставляется в составе CodeWarrior и всех остальных компиляторов C++. Она является частью стандартной библиотеки C++. Прежде чем продолжить, нам придется объяснить несколько дополнительных терминов:

- **Команды.** Общее название для строк кода, набираемых в редакторе исходных текстов и представляющих собой инструкции для компьютера. Комментарии не являются командами.
- **Директивы включения.** Это строки, которые начинаются с инструкции `#include`. Они нужны для включения исходных файлов, созданных вами или другими программистами, в текст программы. За инструкцией `#include` следует имя файла, заключенное в символы < и > (для стандартных библиотечных файлов) либо в двойные кавычки (для всех остальных файлов). По странному стечению обстоятельств у файлов стандартной библиотеки нет расширений. Именно поэтому включается iostream, а не iostream.cpp. Однако у большинства включаемых файлов расширения есть, и про них следует помнить. Обычно директивы включения расположены в начале файла. (Пусть вас не беспокоят принципы работы директив включения. На данном этапе достаточно знать, что они используются для включения в программы внешнего кода.)
- **Библиотеки.** Законченные фрагменты кода, для удобства включенные в язык C++. (С различными компиляторами поставляются раз-

личные наборы библиотек. Найти нужную библиотеку можно в Интернете. Один из замечательных ресурсов по бесплатным библиотекам C++ расположен по адресу *http://www.cnet.com*.) Грубо говоря, библиотеки – это скомпилированные включаемые файлы. К примеру, случайное число можно получить с помощью генератора случайных чисел, функции `rand`, поскольку она входит в состав библиотеки cstdlib (C Standard Library, стандартная библиотека C). Для этого необходимо поместить строку `#include <cstdlib>` в начало программы. Эта инструкция подсказывает компилятору, какая библиотека используется. Включаемые файлы библиотек обычно имеют расширение .h (header, файл заголовка), за исключением файлов стандартных библиотек C++, у которых нет расширения.

Вторая строка приведенного кода, `using namespace std`, заставляет стандартные библиотеки работать так, как нам нужно. Подробности, связанные с этой строкой, входят в состав более сложных тем, которые мы изучим в главе 7 «Градостроение и пространства имен». Вторая половина строки, `//introduces namespace std`, является *комментарием*. Комментарий не влияет на выполнение программы. Пара прямых слэшей (//) предписывает компилятору игнорировать оставшуюся часть строки. Назначение комментариев – делать код более понятным. Комментарии можно писать двумя способами. Однострочные комментарии (как предыдущий) занимают только одну строку. Все символы после // в этой строке игнорируются:

```
// В моей армии один человек
```

Второй способ позволяет растянуть одну фразу либо длинный комментарий на произвольное количество строк:

```
/* драконы правят миром */
```

или

```
/* драконы правят
    миром */
```

И хотя это обычно снижает читаемость, можно размещать многострочные комментарии практически в любой точке кода, как показано здесь:

```
using namespace /*introduces namespace std*/ std;
```

Однако не стоит пользоваться возможностью только потому, что она существует. Такой стиль использования комментариев может быстро сделать код совершенно нечитаемым. Основное правило таково: пользуйтесь комментариями, чтобы делать код более понятным. Если комментарий не приносит пользы, удаляйте его.

Повторимся: содержание комментариев не влияет на код программы. Применяйте их только для пояснения сложных или крупных частей программы, используя обычный русский язык. Компилятор игнорирует комментарии, когда переводит программу в машинный код.

Истории из жизни

Комментарии полезны, поскольку облегчают понимание кода не только для других, но и для самих авторов.

Некоторое время назад мы занимались воскрешением классической игры для Nintendo, которая называлась «Iron Tank» (Железный танк). Через некоторое время после начала проекта у нас было более 30 страниц кода. Затем нам пришлось на некоторое время прервать разработку. Когда мы снова вернулись к этой игре, то потратили почти неделю, пытаясь разобраться, что делает код. Если бы мы сразу озаботились написанием комментариев, эта работа заняла бы максимум один день.

Следующая строка, `int main( void )`, отмечает начало функции `main`. Большая часть кода содержится в функции `main`. В главе 4 «Пишем функции» мы расскажем о том, как помещать код в другие функции, но на данном этапе практически каждая написанная строка кода будет располагаться в теле главной функции. Функция `main` начинается символом `{` и заканчивается символом `}`. Внутри фигурных скобок может располагаться практически любая строка кода. В каждой программе должна быть функция `main` (и только одна).

Следующая строка отмечает начало кода функции `main` открывающейся фигурной скобкой (`{`).

Следующая строка печатает сообщение `This is a test`. Слово `cout` позволяет отобразить текст на экране. И поскольку это слово не встроено в язык, а является частью библиотеки iostream, чтобы воспользоваться им, необходимо включить эту библиотеку в код. Подробнее мы рассмотрим оператор `cout` в разделе «Работа с текстом» позже в этой главе. (*Операторы* – это атомарные мысли или команды; считайте их эквивалентами предложений. Обычно оператор заканчивается точкой с запятой.)

`return 0;` сообщает компьютеру, что все задачи в функции `main` выполнены и следует завершить ее работу.

Закрывающаяся фигурная скобка в последней строке (`}`) сообщает компилятору, что в функции `main` больше нет строк кода. `return 0;` – это исполняемый оператор, который завершает работу функции, а закрывающаяся скобка показывает, что в функции больше нет исполняемых операторов.

Не забывайте о точках с запятыми – они являются частью кода. Большинство операторов *должны* заканчиваться точкой с запятой. В главе 3 «Принимайте командование: управляющие операторы» мы расскажем об операторах, которые не требуют точки с запятой, но до тех пор каждый из встретившихся нам операторов должен завершать-

ся этим символом. Точное правило звучит так: каждый исполняемый оператор должен заканчиваться точкой с запятой. Она отмечает конец оператора, а не конец строки, поскольку в одной строке может присутствовать несколько операторов.

Теперь, когда первое препятствие преодолено, ваше приключение станет еще проще.

Здороваемся с миром

Итак, вы готовы создать свою самую первую программу, которую мы предлагаем назвать «Здравствуй, мир!» (хотя вы можете выбрать любое другое название). Программа будет выводить на экран сообщение `Здравствуй, мир!`. В процессе создания этой программы вы научитесь редактировать исходный текст и поближе познакомитесь с работой кода, сгенерированного средой CodeWarrior. Прежде всего, замените строку

```
cout << "This is a test";
```

строкой

```
cout << "Здравствуй, мир!";
```

Если теперь скомпилировать и выполнить программу, мы увидим на экране сообщение `Здравствуй, мир!`.

Ниже приведен законченный код. (Обратите внимание, мы добавили в начало программы комментарий, который содержит название программы и имя автора. Такие комментарии не являются обязательными, но лишними они тоже не будут.)

```
//1.1 - Здравствуй, мир - Дирк Хенкеманс - Premier Press
#include <iostream>
using namespace std;     //introduces namespace std
int main( void )
{
    cout << "Здравствуй, мир!" ;
    return 0;
}
```

Повторите действия, которые уже выполняли для компиляции кода, сгенерированного CodeWarrior, либо (что работает в большинстве компиляторов) просто нажмите <F5>, чтобы скомпилировать и выполнить программу. Программа выводит сообщение `Здравствуй, мир!`.

Цикл разработки

Цикл разработки – это процесс, через который необходимо пройти при создания программы. Рано или поздно вы поймете, что процесс этот незатейлив и довольно прост. Блок-схема на рис. 1.6 иллюстрирует цикл разработки.

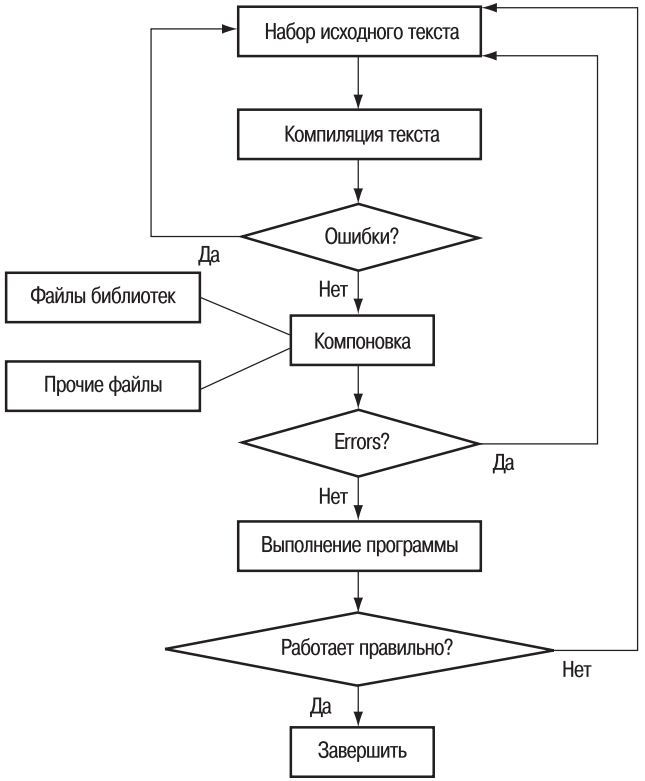

Рис. 1.6. *Схема процесса создания работающей программы*

В общем случае создание программы включает следующие шаги:

1. Набор кода в редакторе исходных текстов (CodeWarrior или другой редактор). В предыдущем разделе «Пишем первую программу» вы немного этим занимались, а до конца книги еще вдоволь попрактикуетесь.

2. Компиляция кода. Если обнаружены ошибки, следует вернуться к предыдущему шагу и исправить их. Пошаговые инструкции по компиляции кода в CodeWarrior даны выше по тексту в разделе «Компиляция».

3. *Компоновка* кода. Это процесс проверки его работоспособности со всеми включенными файлами. Если возникнет ошибка, следует вернуться к шагу 1 и исправить ее. CodeWarrior выполняет компоновку автоматически, и она вас не должна интересовать, пока не возникнет ошибка.

4. Тестирование программы. Следует убедиться, что программа работает правильно. Другими словами, тестирование предназначено для обнаружения семантических ошибок. *Семантические ошибки*

связаны с результатами, которые показывает программа. Если программа компилируется и выполняется, но не делает то, что должна делать, речь обычно идет о семантической ошибке. Допустим, вы написали программу для вывода на экран слова «Привет», а вместо этого выводится слово «Пока». Это семантическая ошибка, которую следует исправить. Тем не менее, наличие семантических ошибок не мешает компиляции кода.

Если программа успешно прошла через все шаги, цикл разработки завершен. Тестирование программ и исправление ошибок описано в главе 5.

Работа с текстом

До появления графики основой всех программ был текст. Текстовые приключения и текстовые электронные доски объявлений (BBS) – вот наши первые опыты программирования. Текст по-прежнему является очень важной составляющей программирования. В этом разделе вы узнаете, как создавать и хранить его.

Текст – самое простое средство вывода результатов работы программ. В первых главах книги используется только текст, поскольку в большинстве случаев его достаточно для отображения выходных данных. Этот подход также избавляет нас от необходимости вдаваться в сложности, связанные с графикой, и отложить их до более поздних глав.

В мире компьютеров у текста есть техническое название – *строка* (*string*). Так, в нашей первой программе текст `Здравствуй, мир!` является строкой.

Компоновка строк

Строка – это набор символов. Обычно мы считаем *символом* знак, который можно набрать с клавиатуры (включая пробелы). Компьютер считает строку, например `"За короля!!!"`, последовательностью символов, каждый из которых занимает одну позицию (`'3'` – первый символ, `'a'` – второй и т. д.). Строки заключаются в двойные кавычки, а отдельные символы – в одинарные. Скажем, `"a"`, `" "` (пробел), `"4"` и `"%"` – это строки, и каждая состоит из одного символа. Однако `'a'`, `' '`, `'4'` и `'%'` – это отдельные символы. За редким исключением символы можно использовать везде, где используются строки.

Но символы не так просты, как кажется. Не все символы можно набрать при помощи клавиатуры. Существует 256 различных символов.

Доступ ко многим другим символам можно получить, удерживая клавишу <Alt> и набирая их численные значения. К примеру, численным значением буквы *A* является 65. В приложении C «Стандартная таблица символов ASCII» содержится справочник по стандартизированным символам и их численным значениям.

Очень важно понимать разницу между строками и символами. Строки состоят из символов, но значительно отличаются от них. Кроме того, символ можно легко преобразовать в строку, но он не является строкой.

Тема может показаться сложной, но на деле строки очень легко создавать. Ниже приведены примеры строк. Как видите, строка – это просто текст, заключенный в двойные кавычки.

```
"За короля!!!"
"Передайте мне мой меч."
"Кто положил мой посох на повозку?"
```

Хранение строк

Строки можно хранить в компьютере, чтобы их не приходилось набирать многократно. Точнее, строки хранятся в *памяти* (о памяти мы подробно поговорим в главе 2 «Продолжаем погружение: переменные»). Код для работы со строками хранится в подходящим образом названной *библиотеке строк*. Чтобы хранить строки, необходимо включить библиотеку в текст программы. Это можно сделать, добавив в начало исходного текста такую директиву:

```
#include <string>
```

Когда вы решаете сохранить строку в памяти, то должны придумать для нее имя, чтобы в будущем можно было объяснить компьютеру, о какой из строк идет речь. К примеру, мы сохраним строку `"Дракон приближается"` в памяти и назовем ее `yell`.

Существует и небольшая сложность. Компьютер умеет хранить в памяти различную информацию, не только строки, и различного рода информация хранится по-разному. Число хранится иначе, чем строка. Поэтому необходимо указать компьютеру, какого рода информацию мы намереваемся хранить. Вот что мы должны указать:

- Тип информации, которую необходимо хранить (в нашем случае – строка `string`)
- Символы, составляющие строку
- Имя строки

И хотя C++ не обеспечивает возможности хранить строки, такую возможность обеспечивает стандартная библиотека. Чтобы воспользоваться ею, необходимо включить в текст программы библиотеку `<string>`.

После включения библиотеки строку можно сохранить так:

```
string yell = "Дракон приближается";
```

Прежде всего указывается тип хранимой информации: строка. Затем идет имя строки: `yell`. Знак равенства сообщает компьютеру, что строка `"Дракон приближается"` должна быть сохранена в `yell`. Последний символ, точка с запятой, сообщает компьютеру, что команда закончена и за ней начинается следующая.

Вывод строк: cout

Теперь вы знаете, как записать строку в память, и вам наверняка захочется ее отобразить. В этом разделе мы расскажем, как печатать строки разнообразными способами.

Вывод строк производится с помощью команды `cout`. Команда `cout` принадлежит файлу iostream стандартной библиотеки C++. Вот команда `cout`, которую мы встречали раньше:

```
cout<< "Здравствуй, мир!";
```

При помощи `cout` можно отобразить любую строку:

```
cout<< "Вперед, на подвиги!";
```

Во-первых, необходимо набрать команду `cout`, затем два знака «меньше» (`<<`), а затем строку для печати и за ней точку с запятой (которая, как вы, вероятно, помните, сообщает компилятору, что команда закончилась).

Вывод нескольких строк: cout

Вы овладели основами `cout`, но на этом разговор о `cout` и строках не заканчивается. Еще эта команда позволяет, например, выводить на печать несколько строк одновременно. Предположим, мы хотим отобразить два слова, но они хранятся в различных строках. С учетом полученных знаний мы могли бы написать примерно такой код:

```
cout<<"Красный";
cout<<" Дракон";
```

Вывод:

```
Красный Дракон
```

Но есть и более простой способ. Можно отобразить любое число строк, разместив их рядом и разделив парами символов `<<`. Вот эквивалент предыдущего фрагмента кода:

```
cout<<"Красный"<<" Дракон";
```

Вывод:

```
Красный Дракон
```

Вывод в обоих случаях одинаковый, но второй фрагмент кода немного проще. Структура языка C++ позволяет записывать команду в несколько строк. Последний фрагмент кода и приведенный ниже эквивалентны:

```
cout
<<"Красный"
<<" Дракон";
```

Вывод:

```
Красный Дракон
```

Разумеется, пример тривиальный, поскольку в данном случае мы могли бы воспользоваться единственной строкой (`"Красный Дракон"`).

Рекомендуется структурировать текст программы так, чтобы его было легко читать. В начале каждой фразы следует использовать отдельную команду `cout`. К примеру, чтобы напечатать предложение о драконах и предложение об эльфах, следует поместить всю информацию о драконах в одну команду `cout`, а всю информацию об эльфах – во вторую команду. Таким образом, код будет более организованным и легким для восприятия.

Специальные символы

Некоторые символы, такие как двойные кавычки (`"`) и разрыв строки, невозможно выразить в строке. Если включить двойные кавычки в строку, компилятор посчитает этот символ признаком конца этой строки. К примеру, чтобы создать строку с цитатой, мы могли бы написать:

```
"Он сказал: "Это цитата"."
```

Но компилятор интерпретирует нашу строку как две самостоятельные строки `"Он сказал: "` и `"."`, а поскольку слова `Это цитата` не будут сочтены строкой, мы получим ошибку синтаксиса. Сходная проблема связана с разрывом строк. Текст строки должен располагаться в одной строке программы, и разрыв строки в любом месте также приведет к возникновению ошибки синтаксиса. К счастью, существует решение всех этих проблем.

Речь идет о специальных символах. *Специальные символы* (или метасимволы, escape-последовательности) используются для представления символов, которые не могут быть явно включены в текст строки. Специальный символ является комбинацией обратного слэша (`\`) и конкретного символа. К примеру, специальным символом для двойных кавычек будет `\"`. Вот так это выглядит в коде:

```
"Он сказал: \"Это и впрямь цитата\"."
```

Такая строка приведет к получению желаемого результата. Если вывести строку на печать, мы увидим на экране текст `Он сказал: "Это и впрямь цитата".`. Существуют и другие специальные символы, а самые важные из них перечислены в табл. 1.1.

Мы советуем поближе познакомиться с большинством из этих специальных символов, потому что они бывают полезными, а некоторые из них, скажем, разрыв строки, являются жизненно необходимыми.

Таблица 1.1. Специальные символы

Название	Символ
Новая строка	\n
Горизонтальная табуляция	\t
Забой (backspace)	\b
Звуковой сигнал	\a
Обратный слэш	\\
Знак вопроса	\?
Одинарная кавычка	\'
Двойная кавычка	\"

Вывод хранимых строк

До этого момента мы выводили на печать непосредственно символьные строки. Теперь мы готовы к отображению хранимых строк. Чтобы отобразить хранимую строку, следует использовать вместо символьной строки имя хранимой строки. Напомним определение строки, с которым мы уже встречались:

```
string yell = "Приближается дракон";
```

Чтобы отобразить строку, воспользуемся ее именем, `yell`, а не хранимым текстом:

```
cout<<yell.c_str();
```

Эта строка делает то же, что и следующая, которую мы использовали в разделе «Хранение строк».

```
cout<<"Приближается дракон";
```

Не беспокойтесь пока о суффиксе `.c_str()` после имени `yell`: это просто «волшебный» код, позволяющий правильно отобразить строку с текстом. Подробнее о волшебном коде мы расскажем в главе 6 «Сложные типы данных».

Программа «Глашатай»

Вы узнали, как создавать строки и выводить их на экран, пользуясь `cout`. Настало время испытать знания, создав программу «Глашатай».

Представьте, что к деревне приближается дракон и необходимо прокричать (yell) предупреждение всем жителям. Если вы прокричите предупреждение в четыре раза быстрее обычного, деревня будет спасена. Очевидно, это работа для наших героев... хранимых строк и команды `cout` – только они смогут спасти деревню. Таким образом, ваше задание заключается в создании программы, которая выводит текст предупреждения четыре раза подряд.

Ниже приводится один из вариантов готовой программы, но мы предлагаем обращаться к нему только в самом крайнем случае – если вы будете испытывать сложности с созданием собственной программы:

```
//1.2 - Программа "Глашатай" - Дирк Хенкеманс - Premier Press
#include<iostream>
#include<string>
using namespace std;      //introduces namespace std

string yell = "Приближается дракон, спасайтесь!!!";

int main(void)
{
    cout<< yell.c_str() <<endl
        << yell.c_str() <<endl
        << yell.c_str() <<endl
        << yell.c_str() <<endl;

    return 0;
}
```

Вывод:

```
Приближается дракон, спасайтесь!!!
Приближается дракон, спасайтесь!!!
Приближается дракон, спасайтесь!!!
Приближается дракон, спасайтесь!!!
```

Применение cin

При разработке программ часто бывает необходимо предоставить пользователям возможность вводить и хранить информацию. Например, набирая свои имена, пользователи должны иметь возможность обращаться к ним впоследствии. Следовательно, необходим способ получения и применения информации от пользователя в процессе работы программы (на этапе ее выполнения), а не в процессе написания (разработки) программы, поскольку невозможно предсказать ввод пользователей. В этом разделе мы расскажем, как получать ввод пользователя и сохранять его для последующего применения.

Сохраняем строки с помощью cin

Сохранение строк с помощью `cin` во многом похоже на то, как мы сохраняли строки раньше, с одним исключением: сначала объявляется имя строки, а значение присваивается позже. Для присвоения значения используется объект `cin`. (*Объект* – это конструктивный элемент программирования, представляющий собой сущность или понятие. Объект `cin` представляет клавиатуру или иное устройство ввода, а объект `cout` – экран или иное устройство вывода. (Подробнее мы расскажем об объектах в главе 5.) Вот пример сохранения строки, введенной пользователем:

```
string name;
cin>>name;
```

Обратите внимание, применение `cin` во многом сходно с применением `cout`. Знаки «меньше» сменились на знаки «больше». Они означают, что компьютер читает данные, а не печатает их.

Вот пример полноценной программы, в которой используется `cin`:

```
//1.3 - Программа "Приветствие" - Дирк Хенкеманс - Premier Press
#include <iostream>
#include <string>
using namespace std;      //introduces namespace std
string name = ""; //  "" означает пустую строку
int main( void )
{
    cout<< "Как тебя зовут?";
    cin>>name;
    cout <<endl<<  "Привет, " << name.c_str() ;
    return 0;
}
```

Вывод (ввод пользователя выделен):

```
Как тебя зовут?
Джеки
Привет, Джеки
```

В некоторых ситуациях и для отдельных компиляторов система Windows закрывает окно программы раньше, чем будет отображена последняя информация. Действительно, как только система Windows поймет, что осталось лишь отобразить текст, после его отображения она немедленно закрывает окно. Чтобы увидеть окончание вывода программы, можно добавить в конец программы оператор `cin` *(что мы и сделаем, когда чуть позже будем писать игру «Пираты и мушкетеры»). Windows не закроет окно, пока вы не нажмете клавишу <Enter> по завершении программы.*

В пятой строке приведенного кода `string name` сообщает компьютеру, что следует выделить в памяти место для хранения строки по имени `name`. Помните, что для работы со строками необходимо включить библиотеку `string`.

В строке 8 оператор `cout<< "Как тебя зовут?";` отображает для пользователя приглашение, в котором предлагается набрать имя.

В строке 9 оператор `cin<<name;` предписывает компьютеру остановиться, чтобы пользователь мог набрать информацию. Когда пользователь нажимает клавишу <Enter>, компьютер делает весь текст, набранный до нажатия <Enter>, значением строки `name`.

В строке 10 оператор `cout<< endl << "Привет, " << name.c_str();` начинается с указания компьютеру начать вывод с новой строки. Затем происходит вывод строки `"Привет, "` и имени пользователя. Если в строке 9 пользователь ввел `Джо` или `Джейн` в качестве своего имени, строка 10

приведет к отображению `Привет, Джо` или `Привет, Джейн`. Между словом `Привет,` и именем присутствует пробел, потому что он включен в конец строки `"Привет, "` (до закрывающей двойной кавычки).

И хотя о тексте можно еще многое рассказать, базовую информацию по использованию текста вы уже получили, и теперь мы обратим свое внимание на числа.

Работа с числами

Компьютеры работают только благодаря числам. Даже текст, с которым мы работали ранее, состоит из них (если прибавить единицу к букве *B*, мы получим букву *C*). Числа являются основой всего, что происходит в компьютере, и хорошее их понимание просто необходимо. В этом разделе вы узнаете об основах математики, операторе взятия остатка и о целых числах (дополнительная информация по целым числам содержится в главе 2).

Знакомьтесь, целые числа

Компьютеры хранят информацию разнообразными способами. Тем не менее, мы остановимся пока лишь на базовых понятиях целых чисел и их применении. *Целые числа* – это все недробные числа, как положительные, так и отрицательные, включая нуль. Числа 5, 0 и –100 являются целыми, а число 0,5 – нет. Если вы попытаетесь сохранить дробное число в качестве целого, компьютер удалит остаток, то есть все, что находится после десятичной запятой.

Выполнение действий с помощью операторов

В общем случае *оператор* – это любой символ или пара символов (скажем `<=`), а в некоторых случаях даже термин вроде `sizeof()`, который предписывает компьютеру выполнить определенное действие. Для выполнения сложения, вычитания, умножения и деления применяются операторы. К примеру, когда требуется, чтобы компьютер сложил два числа, следует использовать оператор сложения (`+`), что, вообще говоря, совершенно естественно. Все мы знаем, что 2 + 2 = 4. Вот как можно выполнить ту же операцию в коде C++:

```
cout<<2 + 2;
```

Эта строка выводит на экран цифру `4`.

Четыре основных оператора абсолютно прозрачны – они делают именно то, что вы, вероятно, предполагаете, но потратьте несколько секунд на изучение символов, соответствующих каждому из них:

Сложение `+`

Вычитание `-`

Умножение	*
Деление	/

Как и в математике, эти операторы выполняются не в порядке следования. Операторы умножения и деления выполняются до операторов сложения и вычитания. Например:

```
1 + 3 * 2
```

Сначала вычисляется `3 * 2`, затем добавляется цифра `1`, и в результате получается число `7`.

Приоритет вычислений можно изменять с помощью простых скобок. Если добавить в предыдущую формулу скобки, как показано здесь:

```
(1 + 3) * 2
```

то сначала будет вычислена сумма `1 + 3` с результатом `4`, и этот результат будет умножен на `2`. Окончательно получаем `8`.

Используйте скобки в больших количествах. Это значительно облегчает отладку. Основное правило таково: если формула может выиграть от применения скобок, используйте их. Следование этому правилу сделает код более понятным и легким для восприятия.

Оператор взятия остатка

Помните, в начальных классах вы выполняли письменное деление столбиком и всегда получали целочисленные ответы? Примерно в то же время вы начали работать с остатками, потому что еще не изучали десятичные дроби. Иногда очень полезно знать остаток, полученный при делении числа. Эту информацию можно извлечь с помощью оператора *взятия остатка*. Он возвращает остаток от деления x на y (`x % y`). Чтобы найти остаток от деления 5 на 2, можно воспользоваться таким кодом:

```
5 % 2
```

Эта строка возвращает цифру `1`. Вот еще один простой пример. Представьте, что пять пиратов пытаются поделить 16 сверкающих золотых монет. Пиратам необходимо выяснить, можно ли разделить сокровище поровну или придется устраивать большую пьяную драку, что вплотную подводит нас к следующей игре. (Впрочем, надо заметить, что пираты в любом случае устроят пьяную драку.)

Пишем игру «Пираты и мушкетеры»

Пришла пора испытать приобретенные навыки. Эта программа доступна на компакт-диске, но мы настоятельно рекомендуем написать ее на компьютере самостоятельно. Это будет первая *полная* програм-

ма, которая позволит проверить приобретенные знания о числах и тексте. Удачной потасовки!

```
//1.4 - Игра "Пираты и мушкетеры" - Дирк Хенкеманс - Premier Press
#include <iostream>
#include <string>
using namespace std;      //introduces namespace std

int main( void )
//расскажет историю о пиратах
{
     int buddies;
     int afterBattle;
     string exit;

     cout<< "Ты пират, и разгуливаешь"
          << " по кишащему преступниками " << endl
          << "городу Гаване  (год 1789).  "
          << "Сколько с тобой  "<<endl
          <<"дружков-пиратов? (побольше)"<<endl;
     //записываем число спутников игрока
     cin>>buddies;
     //вычисляем число пиратов, выживших после схватки.
     afterBattle = 1 + buddies - 10;
     cout<< "Внезапно из ближайшей таверны "
          << "выбегают 10 мушкетеров и " <<endl
          << "обнажают свои шпаги. "
          << "10 мушкетеров и 10 пиратов погибают в " <<endl
          << "схватке. Осталось лишь "
          <<(buddies + 1 - 10)<< " пиратов." <<endl
          <<endl;
    cout<< "Состояние убитых насчитывает 107 золотых монет. Это по "
          <<(107 / afterBattle)
           << " золотых монет на каждого." <<endl;
     cout<< "Пираты устраивают большую пьяную драку из-за оставшихся "
          <<(107 % afterBattle)<< " монет.";
     //делаем паузу, чтобы игрок мог оценить результаты
     cin>>exit;

     return 0;
}
```

Резюме

Самый простой способ выводить текст и числа на экран, а также принимать ввод пользователя – воспользоваться библиотекой iostream. Для обработки ввода пользователя можно применять `cin`, а для вывода информации – `cout`. Чтобы воспользоваться ими, необходимо включить библиотеку `<iostream>` в начале программы. Можно отображать целые числа и сохранять их для последующего использования. Хранение данных позволяет сокращать длину программ и делать их более

эффективными. Кроме того, помните, что в одном операторе `cout` можно выводить сразу несколько строк и целых чисел, но при этом их следует разделять парами символов <<. Вот и все на сегодня; приключение продолжится в главе 2.

Задания

1. Напишите программу, которая выводит изображение домика, похожее на то, что представлено на рис. 1.7.

2. Что выводит следующая программа?

```
#include <iostream>
using namespace std;       //introduces namespace std
int x = 25;
string str2 = "Это проверка";

int main( void )
{
    cout<<"Проверка"<<1<<2<<"3";
    cout<<25 %7<<endl<<str2.c_str();
    return 0;
}
```

3. Напишите программу, которая запрашивает у пользователя его имя, приветствует его, затем запрашивает два числа и отображает их сумму.
4. Что произойдет, если сохранить 10,3 в качестве целого числа? А если 0,6? Можно ли сохранить число –101,8 в качестве целого?
5. Напишите код, умножающий исходное число на 2, если оно принадлежит интервалу от 1 до 100 (включительно) и делится нацело на 3; в противном случае умножает на три, если число принадлежит интервалу от 1 до 100, но не делится нацело на три; и наконец, умножает число на остаток от его деления на 100, если число не принадлежит интервалу от 1 до 100. (Подсказка: используйте вложенные операторы `if`.)

2

Продолжаем погружение: переменные

Для начинающего программиста речи о переменных могут звучать пугающе. Но в этой главе мы устраним все сложности и надеемся, что приведенная информация послужит для читателей факелом в царстве тьмы. К концу главы вы будете уверенно разбираться в переменных и применять их в своих программах.

В этой главе вы узнаете:

- Что такое переменные
- Как хранить данные
- Как объявлять переменные и присваивать им значения
- Об основных типах данных
- Как определять размер переменной
- Как использовать `typedef`
- Как преобразовывать шестнадцатеричные числа в десятичные
- Как выполнять приведение типов
- Как пользоваться константами

Что такое переменная

Помните, еще в школе вам давались формулы вроде $3x + 5$? В них x мог быть любым числом. Часто накладывалось сопутствующее ограничение: x мог быть целым числом, рациональным числом и т. д. Это ограничение определяло спектр возможных значений x. То же справедливо и для переменных.

Переменная – это символ, который представляет численные значения, строковые (текстовые) значения или логические значения (истина и ложь). Определение переменной содержит ее тип (например, целочисленный), и переменная может представлять только числа этого типа.