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».

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.

К примеру, если x – целое число, цифра 2 будет допустимым значением x, а число 2,5 – нет.

Каждой переменной одного из основных типов (см. раздел «Знакомьтесь, основные типы переменных» позже в этой главе) сопутствует три фрагмента информации: *идентификатор*, или имя, по которому к переменной обращается программист; *размер*, который сообщает компьютеру объем хранимых данных; и собственно *данные*. Имя и размер присваиваются переменной в зависимости от ее предназначения.

В мире компьютеров переменная – это определенный сегмент компьютерной памяти. Данные, связанные с конкретной переменной, хранятся в присвоенном ей сегменте. Представьте себе, что несколько коробок равного размера выстроены в ряд и пронумерованы последовательно (первая имеет номер 1, вторая – номер 2 и т. д.). Так работают переменные. Каждая переменная занимает одну или несколько коробок, в которых и содержатся данные, хранимые в переменной. Этими коробками являются ячейки памяти компьютера. В них хранятся данные переменных. Присвоив переменной имя, мы получаем удобный доступ к самой переменной и связанной с ней памяти (рис. 2.1).

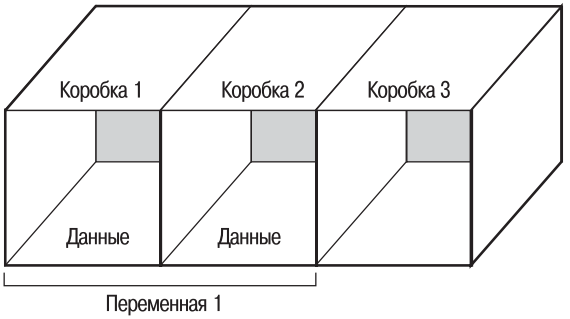

Рис. 2.1. *Первые две коробки диаграммы (байты) принадлежат памяти, отведенной под хранение Переменной 1. Третья коробка – свободное пространство, поскольку она не зарезервирована под хранение данных других переменных*

Разбираемся в отношениях переменных и памяти

По мере накопления опыта вы будете все отчетливее понимать, что большая часть жизни программиста связана с обработкой данных и принятием решений по их обработке. Данные, информация – это сердце мира компьютеров. Если вы знаете, как манипулировать данными, вы умеете программировать.

В целом, ссылаться на данные можно двумя способами: как на литералы или как на основные типы данных. *Литералы* – это буквальные значения данных. Например, число `2` и слово `Привет` являются литера-

лами. В главе 1 «Путешествие начинается» вы встретили много литералов. "Приближаются драконы!" – это строковый литерал, а цифра 8 – целочисленный литерал. Если вы говорите о данных в традиционной манере, то, вероятнее всего, используете литералы.

С другой стороны, *основные типы данных* являются уникальными для программирования. Идея основных типов заключается в обращении к данным без использования литералов. Этот подход является более выгодным, поскольку создает уровень абстракции между кодом и данными, с которыми он работает; это означает, что строка кода может при каждом вызове обрабатывать различные данные.

Основные типы разделены на четыре категории, каждая из которых соответствует определенному виду данных: *логический тип*, *символьные типы*, *целочисленные типы* и *типы с плавающей точкой*. С помощью этих четырех категорий можно представить любые данные в программе.

Но прежде чем продолжить изучение переменных, вы должны узнать немного больше о памяти и о хранении данных.

Существует два базовых типа компьютерной памяти: *оперативная память* (Random-Access Memory, RAM) и *постоянная память* (Read-Only Memory, ROM). Оперативная память состоит из микросхем памяти, расположенных на материнской плате и периферийных картах (графических, звуковых и т. д.). Оперативная память является временным хранилищем данных. Это память, которая используется для работы приложений, и именно этот вид памяти более всех других интересует программистов. ROM-память является *энергонезависимой* и не может изменять состояние; в ней хранятся инструкции загрузки и автоматического тестирования, которые выполняются при включении компьютера. Наш разговор ограничится оперативной памятью (RAM).

*Другим видом памяти является **дисковая память**. Дисковыми накопителями являются жесткие и гибкие диски, компакт-диски и другие устройства временного и долговременного хранения данных. Дисковые носители сохраняют записанную информацию при выключении компьютера.*

Компьютерная память состоит из миллионов крохотных электрических ключей, которые называются битами. (*Бит* – самая маленькая единица хранения информации на компьютере.) Каждый ключ, или бит, имеет два состояния для различных уровней напряжения (например, 0 и +5 вольт). Таким образом, каждый бит может представлять одно из двух значений – назовем их 0 и 1 или истиной и ложью. Следовательно, один ключ хранит один бит (0 или 1). Сочетание двух битов дает четыре варианта комбинаций значений (00, 01, 10 и 11). Так рождается *двоичная* система счисления, или система счисления по основанию два.

Наиболее распространенной системой счисления является десятичная (или система счисления по основанию десять). В ней каждая цифра

имеет одно из десяти возможных состояний (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). В табл. 2.1 приведены числа от 0 до 10, их двоичные и десятичные варианты.

Таблица 2.1. Двоичная и десятичная системы счисления

Десятичное	Двоичное
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Пусть вас не слишком пугает двоичная система. Старые добрые числа остались на месте; просто в двоичной системе они представлены по-другому. Преобразование числа из двоичной системы в десятичную выполняется достаточно легко. Возьмите первую цифру (крайнюю правую цифру двоичного числа, в нашем случае 1) и умножьте на 2^0 (где *x^a* означает *x* в степени *a*). Затем возьмите следующую цифру и умножьте на 2^1. Повторяйте процесс, пока не закончатся двоичные цифры. Для числа 0110101 преобразование выглядит так:

```
1 x 2^0    =    1
0 x 2^1    =    0
1 x 2^2    =    4
0 x 2^3    =    0
1 x 2^4    =   16
1 x 2^5    =   32
0 x 2^6    =    0
            ------
           =   53
```

К сожалению, преобразование из десятичной системы в двоичную несколько сложнее. Прежде всего, разделите число на 2^(n), где *n* – колонка (в числе 101 три колонки), с которой вы работаете. Затем отнимите от числа полученный при делении остаток. Чтобы найти значение двоичной цифры в колонке, разделите новое число на 2^(n–1). Повторяйте эти шаги, пока при вычитании остатка из числа не получится 0. Вот шаги преобразования числа 9 в двоичную форму:

Колонка номер 1:

```
9  / 2^1  =  4 Остаток: 1
Двоичная цифра = 1 / 2^0 = 1
Двоичное число на данном этапе: 1
9 - 1 = 8
```

Колонка номер 2:

```
8  / 2^2  =  2 Остаток: 0
Двоичная цифра = 0/2^1 = 0
Двоичное число на данном этапе: 01
8 - 0 = 8
```

Колонка номер 3:

```
8  / 2^3  =  1 Остаток: 0
Двоичная цифра = 0/2^2 = 0
Двоичное число на данном этапе: 001
8 - 0 =  8
```

Колонка номер 4:

```
8  / 2^4  =  0 Остаток: 8
Двоичная цифра = 8/2^3 = 1
Двоичное число на данном этапе: 1001
8 - 8 =  0
```

Как мы уже говорили, электрический ключ называется битом и может иметь два значения, 0 или 1. Последовательность из восьми битов называется *байтом*. Байт может иметь 256 различных значений, от 0 до 255. Вспомним аналогию с коробками из начала главы: ряды коробок представляют память. Каждая коробка – один байт памяти. Байт – минимальный объем информации, с которым работает компьютер. Если необходимо хранить более одного байта, компьютер задействует соответствующее число коробок. *Килобайт* (Кбайт) содержит 1024 байта. Помните, не 1000 байт, а именно 1024. Память не до конца подчиняется метрической системе. 1024 килобайта составляют мегабайт (Мбайт). 1024 мегабайта – гигабайт (Гбайт). Эта информация сведена в табл. 2.2.

Таблица 2.2. Единицы измерения памяти

Единица измерения памяти	Число битов	Число значений
бит	1	2
байт	8	256
килобайт (Кбайт)	8192	2^8192
мегабайт (Мбайт)	8388608	2^8388608
гигабайт (Гбайт)	8589934592	2^8589934592
терабайт (Тбайт)	8.796093022 * 10^12	2^(8.796093022 * 10^12)

Считайте микросхему оперативной памяти последовательностью байтов. Каждый байт имеет уникальный адрес. *Адрес* – это число (например, 10345), которое позволяет компьютеру определить, о каком из

байтов идет речь. Однако число, которое выдает компьютер, представлено в *шестнадцатеричной* системе счисления, т. е. в системе с основанием 16. Числа на коробках являются адресами памяти.

Итак, узнав чуть больше о хранении и представлении данных компьютером, вы готовы продолжить исследование переменных.

Идентификаторы переменных

Идентификаторы – имена, которые присваиваются переменным и используются для обращения к ним. Например, переменная, в которой хранится ввод пользователя, может иметь имя `input`. Правила именования переменных таковы:

- Имя должно начинаться с буквы или символа подчеркивания (_).
- Последующие символы могут быть буквами, цифрами или символами подчеркивания.
- Идентификаторы могут быть длинными (более 200 символов).
- Ключевые слова языка запрещено использовать в качестве имен переменных.
- Язык C++ чувствителен к регистру символов (имена *aVariable* и *aVARIABLE* являются различными).

Допустимыми идентификаторами являются `_file5G`, `String7F4`, `input`, `__my_variable` и т. д. Для сравнения приведем примеры недопустимых идентификаторов: `9variable` и `&variable`. Двойное подчеркивание в начале идентификатора – идея не очень хорошая, поскольку такие имена часто резервируются для специальных системных переменных. Кроме того, помните, что идентификатор должен быть не только допустимым, но и полезным. Идентификатор должен описывать информацию, с которой связан, чтобы читая свой код три недели спустя, вы могли четко понять, для чего нужна та или иная переменная.

Избегайте даже случайного использования ключевых слов в качестве имен переменных. (Возможно, вы помните из главы 1, что ключевые слова – это основной набор команд и функций, применяемых в программировании.) Такое использование ключевых слов может приводить к непредсказуемым результатам и ненужной головной боли. Например слово `int` *зарезервировано для представления целых чисел и его нельзя использовать в качестве имени переменной – компилятор выдаст ошибку в процессе сборки программы.*

Объявления переменных и присвоение значений

Прежде чем использовать переменную, ее необходимо объявить. Объявление переменной является инструкцией компьютеру зарезервиро-

вать определенный объем памяти под хранение данных и присвоить этому объему имя. Различным типам переменных требуются различные объемы памяти. Например, для объявления целочисленной переменной следует использовать такую строку:

```
int x;
```

Общий синтаксис объявления переменных таков:

```
тип_переменной идентификатор;
```

Для присвоения значений переменным используется оператор присваивания (=). *Операторы* – это символы или слова, которые предписывают выполнение... да-да, *операций*. Операторы могут быть математическими, операторами отношений и др. В этой главе мы рассмотрим только математические операторы и оператор присваивания. Оператор присваивания помещает значения из правой части в значения, указанные в левой части. Пример:

```
x = 5;
```

Переменная x теперь хранит значение 5, то есть память, зарезервированная под переменную x, заполнена числом 5. Не забывайте об отличиях знака равенства в математике и программировании. В данном случае число из правой части равенства присваивается в качестве значения переменной из левой части. Эта операция не имеет ничего общего с проверкой равенства частей. Единичный знак равенства *не* имеет значения «равно». Потратьте лишнюю минуту на повторное прочтение этой информации, поскольку она очень важна.

В правой части оператора присваивания также могут фигурировать переменные:

```
x = y;
```

Переменная x получает значение, хранимое в переменной y.

Помните, что в левой части оператора присваивания не могут фигурировать литералы:

```
5 = x;
```

Это недопустимый оператор. Он предписывает сохранить значение, хранимое переменной x, в числе 5, что не имеет смысла.

Чтобы присвоить переменной значение при объявлении, воспользуйтесь такой конструкцией:

```
тип переменной идентификатор = значение;
```

Это называется *инициализацией переменной*, поскольку она получает начальное значение, инициализируется.

Научившись объявлять переменные и присваивать им значения, мы готовы перейти к изучению различных типов переменных.

Знакомьтесь, основные типы данных

Существует довольно много типов переменных, но в этой главе мы рассмотрим лишь четыре из них. Тип переменной определяет объем резервируемой под нее памяти, а значит, и предельные значения хранимых чисел (или объем хранимых данных). В этой главе мы изучим логические (булевые), символьные, целочисленные типы и типы с плавающей точкой. Некоторые из существующих типов переменных представлены в табл. 2.3.

Таблица 2.3. Типы переменных

Тип	Размер	Значения
bool	1 байт	true (1) или false (0)
char	1 байт	от 'a' до 'z', от 'A' до 'Z', от '0' до '9', пробел, табуляция и т. д.
int	4 байта	от –2 147 483 648 до 2 147 483 647
short	2 байта	от –32 768 до 32 767
long	4 байта	от –2 147 483 648 до 2 147 483 647
float	4 байта	+–(1.2 x 10^–38 до 3.4 x 10^38)
double	8 байт	+–(2.3 x 10^–308 до 1.7 x 10^308)

Логический тип

Логический (или булевый) тип является простейшим типом данных. Его размер равен одному байту (минимальный объем данных, с которым работает компьютер). Он может иметь два значения, true (истина) и false (ложь). Переменные этого типа используются для случаев, когда есть лишь два возможных значения. К примеру, может существовать переменная end, которая имеет значение false, пока не закончится программа. Эту переменную можно использовать, чтобы определить, достигнут ли конец программы. Логические переменные объявляются при помощи ключевого слова bool:

```
bool myBool = true;
```

Символьные типы

Переменная *символьного типа* может хранить один из 256 различных символов, перечисленных в приложении С «Стандартная таблица символов ASCII».

Символьный литерал (буквальное представление данных, не сохраненных в переменной) заключается в одинарные кавычки. Примеры символьных литералов: 'a', '5', '%', 'W'.

Переменные символьного типа объявляются при помощи ключевого слова `char`:

```
char myChar = 'a';
cout<< "буква " << myChar;
```

Целочисленные типы

Существует три целочисленных типа: `int`, `short int` (или `short`) и `long int` (или `long`).

С типом `int` мы познакомились в главе 1. Его длина составляет 4 байта (на большинстве компьютеров). Он может хранить числа в диапазоне от –2 147 483 648 до 2 147 483 647. Этот тип данных применяется чаще всего. Переменные целочисленных типов не могут хранить десятичные и рациональные дроби (например, 2.1 или 1/3). Объявление переменных этого типа происходит при помощи ключевого слова `int`:

```
int myInt = -3;
```

Тип `short` в два раза меньше типа `int`, то есть его длина – 2 байта. Он может хранить числа диапазона от –32 768 до 32 767. Этот тип весьма полезен для хранения небольших значений и вдвойне эффективен, поскольку занимает в два раза меньше памяти. Переменные этого типа объявляются при помощи ключевого слова `short` (или `short int`).

```
short myShort = -56;
```

Последняя строка может быть переписана следующим образом:

```
short int myShortInt = -56;
```

Следующим в этом замечательном путешествии по типам данных будет тип `long`. Он имеет ту же длину, что и `int`, 4 байта. Диапазоны хранимых значений также совпадают: от –2 147 483 648 до 2 147 483 647. Читатели уже, вероятно, догадываются, что переменные этого типа объявляются при помощи ключевого слова `long` (или `long int`). Этот тип является синонимом для `int`:

```
long myLong = 32056;
```

абсолютно равнозначно

```
long int myLong = 32056;
```

Чтобы ограничить диапазон значений целочисленной переменной только положительными числами, можно предварить ее объявление ключевым словом `unsigned` (беззнаковое). Тип `unsigned short` ограничен диапазоном от 0 до 65 535, а `unsigned long` или `int` – диапазоном от 0 до 4 294 967 295. Беззнаковые целые занимают столько же памяти, сколько и обычные (знаковые) целые. Чтобы гарантировать возмож-

ность хранения отрицательных значений, можно использовать ключевое слово signed, но в этом нет необходимости, поскольку знаковость является стандартом для объявленных переменных.

```
unsigned int anUnsignedInt; // от 0 до 4 294 967 295
unsigned short anUnsignedShort; // от 0 до 65535
signed long aSignedLong; // от -2 147 483 648 до 2 147 483 647
int anInt; // от -2 147 483 648 до 2 147 483 647
```

Циклический возврат целых чисел

Если целое число выходит за пределы допустимого диапазона, происходит особенное событие. Отсчет начинается с начала. Например, если присвоить переменной типа unsigned integer значение 4 294 967 295 (которое на единицу больше максимально допустимого), в действительности будет записано нулевое значение (0). Для беззнаковых целых отсчет начинается с цифры 0, а для знаковых – с нижней границы диапазона. Поясним изложенное таким примером:

```
//2.1 - Циклический возврат целых чисел - Дирк Хенкеманс и Марк Ли - Premier
Press
#include <iostream>
using namespace std;

//демонстрация циклического возврата для целых чисел
int main()
{
     unsigned int unInt = 4294967295;
     signed short aShort = 32767;
     cout << "Значение беззнакового int: " << unInt << endl;
     cout << "Значение short: " << aShort << endl;
     unInt = unInt + 1;
     aShort = aShort + 1;
     cout << endl << "Новое значение беззнакового int: "
          << unInt << endl;
     cout << "Новое значение short: " << aShort << endl;
     return 0;
}
```

Вывод:

```
Значение беззнакового int: 4294967295
Значение short: 32767
Новое значение беззнакового int: 0
Новое значение short: -32768
```

В строках 8 и 9, соответственно, переменным unInt и aShort присваиваются максимальные значения. В строках 12 и 13 значения переменных увеличиваются на единицу. Как можно видеть, произошел циклический переход. Это второстепенный момент, но о нем следует помнить на всякий случай.

Оператор приращения

Чтобы увеличить значение, хранимое целочисленной переменной, ровно на единицу, можно воспользоваться оператором приращения (++), который позволяет существенно сократить код. *Оператор инкремента* является математическим оператором, который показывает, что значение переменной увеличивается на единицу относительно текущего. (*Математическим оператором* является любой, позволяющий производить вычисления.) Итак, вместо строки

```
count = count + 1;
```

можно написать

```
count++;
```

Аналогичным образом можно использовать *оператор декремента* (--) для уменьшения целочисленного значения на единицу. Оператор декремента вычитает единицу из значения числа. Вместо

```
count = count - 1;
```

можно написать

```
count--;
```

Типы с плавающей точкой

Для хранения десятичных или рациональных дробей следует использовать типы данных с плавающей точкой, `float` и `double`.

Тип данных `float` (или тип данных с плавающей точкой одинарной точности) имеет длину 4 байта. Он может хранить положительные и отрицательные целые и действительные значения с максимальной и минимальной точностью в 1.2×10^{-38} и 3.4×10^{38}. Помните, что эти числа могут быть положительными или отрицательными.

Программисты часто записывают действительные числа в *экспоненциальном представлении*. Экспоненциальное представление позволяет емко выражать очень маленькие или очень большие числа.

Экспоненциальное представление – это просто метод записи действительных чисел, с которыми становится неудобно работать. Например, число 0.0000000000000000002. Вот некоторые числа в экспоненциальном представлении:

```
5.6543e17     -4.02934e5     -17.204e-10     2.0e-19
```

Правило преобразования экспоненциальной записи в традиционный формат: сдвиньте десятичную дробь на число позиций, определенное цифрой справа от буквы `e` (она называется *экспонентой*). Чтобы преобразовать запись 5.6543e17 в традиционный формат:

1. Выделите число (17) справа от буквы e.
2. Сдвиньте десятичную запятую на 17 десятичных позиций вправо. Направление однозначно определяется по знаку числа 17 (в данном случае – положительный). Заполните образовавшиеся пустые позиции нулями.

   В результате получаем 563430000000000000.0, что представляет собой то же, что и 5.6543e17. Последнее число гораздо удобнее набирать, чем 563430000000000000.0.

Если экспонента числа отрицательная, скажем, 5.6543e–17, десятичную запятую следует сдвинуть на 17 позиций влево; в результате получим 0.00000000000000056343.

При хранении все числа преобразуются в экспоненциальную запись, а затем округляются до ближайшей позиции 10^–38. 38 знаков – это уровень точности для этого типа данных. Например, если сохранить число π (3.14159265358979323846...) с точностью в два десятичных знака, число будет округлено до 3.14.

Переменные с плавающей точкой одинарной точности объявляются при помощи ключевого слова `float`:

```
float aFloat = 3.156;
float negativeFloat = -678.876;
```

Чтобы запомнить рациональную дробь, компьютер должен привести ее из формата дробей a/b к десятичной форме. При следующем обращении к рациональной дроби компьютер выдаст наиболее точное из достижимых для него десятичных представлений этой дроби.

Тип данных `double` имеет длину 8 байт. Он может хранить положительные значения из интервала от 2.2×10^{-308} до 1.7×10^{308} и отрицательные значения из интервала от -2.3×10^{-308} до -1.7×10^{308}. Таким образом, значения `double` округляются до ближайшей позиции 10^–308. Это невероятно высокая точность. В обычных условиях переменные `double` способны сохранить любое значение. Объявление переменных производится при помощи ключевого слова `double`. Запись *<число_a>*e*<число_x>* означает *<число_a>**10^*<число_x>*.

В следующем примере число 2.2e–308 абсолютно эквивалентно 2.2*10^–308:

```
double aDouble = 2.2e-308;
double negDouble = -2.3e-308;
```

Итак, вы познакомились с основными типами данных. Привыкните к тому, чтобы использовать наиболее короткие из подходящих типов. Это сэкономит память и сделает ваши программы более быстрыми.

Оператор sizeof()

Этот относительно простой оператор является частью языка C++. Для любой переменной он возвращает объем зарезервированной для нее памяти в байтах. Формат вызова:

```
sizeof(идентификатор);
```

Чтобы вывести размер переменной типа `double` на экран, воспользуйтесь следующим фрагментом кода:

```
double aDouble;
cout << "Размер переменной типа double: " << sizeof(aDouble); // равен 8
```

Поскольку оператор возвращает целое число (*количество* байтов), его можно использовать в любом контексте, где может быть использовано целочисленное значение.

Кроме того, этот оператор можно выполнить не только для идентификатора, но и для ключевого слова типа, как показано ниже:

```
cout<< "Размер двух чисел типа int равен " << 2 * sizeof(int); // равен 4
```

Игра «Типы данных»

На своем неблизком пути вы встретили поселение эльфов. Тебе повезло, о смелый путешественник, потому что выпал идеальный шанс испытать приобретенные умения. Деревню атакует полчище драконов! Чтобы защитить ее, необходимо вызвать из мистического леса воинов типов данных, которые смогут спасти эльфов.

Для спасения деревни придется использовать знания о целых числах и числах с плавающей точкой различной точности, приобретенные в предыдущем разделе. Готовы принять вызов? Тогда читайте дальше:

```
//2.2 - Игра "Типы данных"- Дирк Хенкеманс и Марк Ли - Premier Press
#include <iostream>
using namespace std;   //используем пространство имен std

//в главных ролях - типы данных
int main( void )
{
     int intWarriors;
     double doubleWarriors;
     float floatWarriors;

     cout << "Поселение эльфов атаковали "
          <<" драконы."<< " Чтобы спасти жителей,  ";
     cout << "необходимо создать воинов каждого типа данных "
          <<"и защитить город." << endl << endl;
     cout << "Сколько воинов типа int послать в атаку?";
     cin >> intWarriors;
```

```
    cout << endl << "К счастью, каждый воин обладает силой"
         << sizeof(intWarriors) << ", " << endl
        << "и этого почти достаточно, чтобы победить синих драконов." << endl;

    cout << endl << "Скорее! Сколько воинов типа double "
          << "послать в атаку?";
    cin >> doubleWarriors;
    cout << endl << doubleWarriors;
     cout << " воинов типа double атакуют оставшихся синих "
           << "драконов" << endl << "Они убили "
           << sizeof(doubleWarriors) << " синих драконов."
          << " Все синие драконы мертвы."
           << endl << endl;

    cout << "Сколько воинов типа  float послать в атаку?";
    cin >> floatWarriors;
    cout << endl << "Каждый из "<< floatWarriors
           <<" воинов типа float выпускает ";
    cout << sizeof(floatWarriors) << " стрел." << endl;
    cout << "Этого как раз достаточно, чтобы убить зеленых драконов."
            << endl << "Поздравляем, ты спас "
           << "эльфов!";
}
```

typedef облегчает жизнь

Периодически становится весьма неудобно раз за разом пользоваться такими ключевыми словами, как `unsigned short int`. Оператор `typedef` позволяет переименовать тип переменной. Можно заменить `unsigned short` на `USHORT` или на еще более удобный вариант. Формат `typedef` следующий:

```
typedef тип_переменной новое_имя;
```

Переименовать `float` в `f` можно так:

```
typedef float f;
```

А теперь объявляем переменную типа `float` с помощью нового ключевого слова:

```
f radius = 4.7639;
```

Помните, что новое имя типа должно быть не только удобным, но и содержательным.

Приведение типов

Иногда бывает удобно произвести преобразование из одного типа в другой. Такое преобразование получило название приведения. (*Приве-*

дение, называемое еще *приведением типов*, – это процесс преобразования данных из одного типа в другой с сохранением почти такого же значения.) При работе с некоторыми библиотеками может требоваться применение конкретных типов данных (это часто происходит при работе с DirectX). Приведение выполняется следующим образом:

```
float pi = 3.14;
int roundedPi = (int) pi;
```

В этом фрагменте кода `roundedPi` получает значение `3`. Дробная часть отсекается. Обратите внимание, что в данном случае не происходит округление до ближайшего целого – дробная часть просто удаляется.

Применение констант

В C++ существует два вида констант. С первым из них вы уже знакомы, это *литеральные константы*. Литеральная константа и литерал – это одно и то же (более подробно о литералах рассказано в разделе «Разбираемся в отношениях переменных и памяти» ранее в этой главе). В качестве примеров литеральных констант можно привести число `2` и строку `"Hello"`. Они являются константами по определению, ведь их значения невозможно изменить.

Второй тип констант – *символьные константы*. Символьная константа во многом похожа на переменную, но ее значение нельзя менять. В начале программы можно создать константу `PI` со значением `3.14`. При компиляции кода каждое вхождение идентификатора `PI` будет заменено числом `3.14`, и это означает, что по крайней мере для компьютера нет разницы между символьной константой и литеральной константой. Но прежде чем вы начнете задаваться вопросом, зачем и когда следует использовать символьные константы, необходимо научиться их использовать.

Существует два способа объявления констант. Первый связан с директивой `#define`. (*Директива* – это строка кода, которая исполняется компилятором непосредственно перед компиляцией оставшейся части кода.) Эта директива имеет следующий синтаксис:

```
#define ИМЯКОНСТАНТЫ значение
```

`ИМЯКОНСТАНТЫ` – это идентификатор, по которому можно будет ссылаться на константу, а `значение` – значение константы. Обратите внимание, как и в случае с директивой `#include`, отсутствует точка с запятой в конце строки. Директива не является исполняемым оператором C++; она лишь предписывает компилятору выполнить определенные действия перед компиляцией. Директива `#define` предписывает компилятору заменить в программе все вхождения идентификатора `ИМЯКОНСТАНТЫ` значением этой константы. Компилятор, выполнив подстановку, переходит к компиляции кода. В программе при этом можно использо-

вать ИМЯКОНСТАНТЫ вместо литеральных констант. К примеру, можно объявить константу для числа π:

```
#define PI 3.14159
```

И в коде вместо подобных конструкций:

```
y = 3.14159 * x;
```

использовать такие:

```
y = PI * x;
```

Первое и самое заметное преимущество использования констант – повышение читаемости исходного текста. Число 3.14159 само по себе не подразумевает значение π; но если создать константу с именем PI, смысл происходящего становится предельно ясным.

Второй способ определения констант связан с ключевым словом const. Синтаксис почти такой же, как для #define:

```
const типКонстанты ИМЯКОНСТАНТЫ значение;
```

Ключевое различие заключается в том, что const определяет тип константы, и в качестве такового может использоваться любой существующий тип данных. Это отличие говорит в пользу const, поскольку хорошо сочетается с проверкой типов. Компилятор имеет возможность убедиться, что константа используется в контексте, где ее тип не вызовет конфликта. К примеру, если создать строковую константу

```
const string HELLO "Привет";
```

или так:

```
#define HELLO "Привет"
```

именем константы можно пользоваться в коде вместо строки "Привет".

Но следует помнить, что const string HELLO дает возможность проверки типа, и константу HELLO можно будет использовать только там, где можно использовать обычные строки. Однако #define HELLO не имеет подобных ограничений, и это может приводить к странным результатам:

```
int x = HELLO;
```

Для константы, определенной посредством const, возникнет ошибка в процессе компиляции, но этого не произойдет для константы #define. Переложив поиск возможных проблем на компилятор, вы избавитесь от необходимости самостоятельно искать ошибку. А это очень существенное преимущество при разработке больших приложений.

Определим константу PI вторым способом:

```
const float PI 3.14159;
```

Не забывайте инициализировать константы при их создании. После создания ее значение невозможно изменить. Если не инициализировать ее при объявлении, возникнет ошибка при попытке установить значение позже.

Применение констант имеет много положительных сторон. Представьте, что в своей программе вы набирали `3.14` всякий раз, когда требовалось значение числа π. И вдруг вы осознали, что для работы приложения требуется более высокая точность. Придется заменить каждое вхождение числа 3.14 более точным значением, скажем, 3.14159. Использование директивы для создания константы `PI` позволило бы менять точность легким изменением `#define` и перекомпиляцией программы.

Истории из жизни

Вообразите, что являетесь руководителем крупной корпорации, в которой каждый сотрудник получает $10.78 в час, и решили повысить оплату до $11.78 в час. Очень затруднительно будет редактировать тысячи записей в программе расчета зарплаты. В случае применения константы для хранения информации о ставке понадобится изменить лишь эту константу, что сэкономит вам лично и всей корпорации массу времени и усилий.

Если речь идет об очень длинных числах (скажем, π с точностью до 22 знаков после запятой), весьма утомительно набирать их при каждой необходимости; а если вы похожи на нас, то запросто можете сделать ошибку при наборе. Константы позволяют избежать подобных затруднений.

Игра «Круги»

Ваше домашнее задание по математике на завтра: вычислить площадь круга и длину окружности. Не исключено, что мы сможем посодействовать. В следующей программе мы выполним обе задачи с помощью констант. Воспользуемся константой для хранения значения числа π (3.141592). Пользователю останется лишь ввести радиус круга, а программа вычислит площадь круга и длину окружности. Обратите внимание: ввод пользователя должен храниться в переменной типа `float`, чтобы пользователь мог указывать дробные значения радиуса. Отлично, пора писать код:

```
//2.3 - Игра "Круги" - Дирк Хенкеманс и Марк Ли - Premier Press
#include <iostream>
using namespace std;

//Вычисляет площадь круга и длину окружности
int main()
{
```

```
    typedef float f;
    const f PI = 3.141592;
    f radius, circumference, area;

    cout << "Добро пожаловать в программу создания кругов!" << endl;
    cout << "Какой изволите определить "<<endl
        <<"радиус для круга? ";
    cin >> radius;

    area = PI * radius * radius;
    circumference = PI * (radius * 2);
    cout << "Площадь круга: " << area << endl;
    cout << "Длина окружности: "
        << circumference << endl;
    cout << "Спасибо, что играли в создание кругов!"
         << endl;

    return 0;
}
```

Повторяем синтаксис

К этому моменту мы изучили разнообразные типы слов – ключевые, идентификаторы, директивы и т. д., – а также многие виды операторов – присваивания, включения и пр. В этом разделе мы повторим уже изученные конструкции синтаксиса языка C++. (*Синтаксис* – это грамматические правила языка.)

Начнем с ключевых слов – фундамента языка C++. Эти слова служат основой для создания программ. Компилятор C++ понимает только эти слова, пока программист не «научит» его другим.

В табл. 2.4 перечислены уже изученные нами ключевые слова, описана их функциональность и синтаксис.

Повторимся, ключевые слова – это сердце языка C++. Каждое из них имеет особое значение и смысл. Познав эти слова, вы окажетесь на верном пути к вершинам мастерства в C++.

Следом идут идентификаторы. Вам известно три типа идентификаторов: литералы, идентификаторы констант и идентификаторы переменных. В табл. 2.5 указаны их применения и форматы.

Идентификаторы литералов выбирать не приходится: им даются имена, соответствующие значениям. В идентификаторах констант обычно все буквы заглавные, хотя это необязательно. В идентификаторах переменных обычно все буквы строчные, хотя слова в составных именах выделяются (к примеру, `variableName`). Старайтесь делать идентификаторы как можно более наглядными; это повысит прозрачность кода.

Таблица 2.4. Ключевые слова C++ и их назначение

Ключевое слово	Назначение	Синтаксис
`const`	Объявление констант	`const тип_константы имя_константы значение;`
`int`	Объявление переменных типа int	`int имя_переменной;`
`short`	Объявление переменных типа short	`short имя_переменной;`
`long`	Объявление переменных типа long	`long имя_переменной;`
`float`	Объявление переменных типа float	`float имя_переменной;`
`double`	Объявление переменных типа double	`double имя_переменной;`
`bool`	Объявление логических переменных	`bool имя_переменной;`
`string`	Объявление переменных типа string	`string имя_переменной;`
`unsigned`	Ограничение целочисленных переменных положительными значениями	`unsigned целочисленный_тип имя_переменной;`
`return`	Необходимо в конце функции `main()`	`return 0;`
`sizeof()`	Возвращает размер переменной	`sizeof(тип_переменной)`
`void`	Необходимо для объявления функции `main()`	`int main (void)`
`main`	Начало основной программы	`int main (void) { }`
`typedef`	Переименование типов переменных	`typedef тип_переменной новое_имя;`

Таблица 2.5. Применения и форматы идентификаторов

Идентификатор	Применение	Формат
Литерал	Хранение буквальных значений (3, 2)	Нет правил форматирования (просто используйте значение)
Переменная	Хранение изменяющихся значений	Присвоить литерал идентификатору переменной с помощью оператора присваивания (=)
Константа	Хранение констант	Правило для переменных; присваивание возможно только в момент создания константы

Мы еще не рассматривали в подробностях директивы препроцессора, но две из них вы уже знаете. Речь идет о `#include` и `#define`. (Как мы уже говорили, директивы являются предписаниями компилятору выполнять определенные действия в процессе компиляции.) `#include` сообщает компилятору, что следует добавить внешний файл к тому, который компилируется. Скажем, в программе «Здравствуй, мир» строка

```
#include <iostream>
```

сообщает компилятору, что необходимо добавить библиотеку `iostream` в файл `hello.cpp` при компиляции. Эта библиотека содержит довольно большой объем кода, включая и реализацию оператора `cout`.

`#define` предписывает компилятору заменять каждое вхождение имени определяемой константы ее значением в процессе компиляции. Операторы директив препроцессора не заканчиваются точкой с запятой.

Как мы уже рассказывали, операторы – это символы или слова, выполняющие определенные операции. Вы уже изучили девять операторов. Математические – сложение (`+`), вычитание (`-`), умножение (`*`), деление (`/`) и взятие остатка (`%`); оператор приращения (`++`), оператор присваивания (`=`), оператор `sizeof()` и оператор приведения типов (`( )`). В табл. 2.6 перечислены эти операторы и их назначение.

Научились ли вы обращаться с переменными свободно? Если нет, советуем повторно прочитать разделы этой главы и попрактиковаться с примерами кода. Практика поможет решить все проблемы.

Таблица 2.6. Применение и синтаксис операторов

Оператор	Применение	Синтаксис
Сложения (`+`)	Сложение двух чисел	`число1 + число2`
Вычитания (`-`)	Вычитание для двух чисел	`число1 - число2`
Умножения (`*`)	Умножение двух чисел	`число1 * число2`
Деления (`/`)	Деление двух чисел	`число1 / число2`
Взятия остатка (`%`)	Взятие остатка от деления двух чисел	`число1 % число2`
Приращения (`++`)	Увеличение целого числа на единицу	`целое++`
Присваивания (`=`)	Присвоение значения	`переменная1 = число1`
`sizeof()`	Вычисление размера типа данных	`sizeof(идентификатор)`
Приведения типов (`( )`)	Изменение типа значения переменной	`(тип)Переменная`

Пишем игру «Оружейный магазин»

Пробираясь через темный лес, где-то у черта на рогах вы наткнулись на загадочный оружейный магазин. Тебе повезло, о достойнейший из путешественников, ибо это отличная возможность проверить приобретенные в этой главе знания – включая константы, приведение, операторы, типы данных и прочее, прочее... Чтобы посетить оружейный магазин, придется скомпилировать и выполнить эту программу:

```
//2.4  - Игра "Оружейный магазин" - Дирк Хенкеманс и Марк Ли
//Premier Press
#include <iostream>
#include <string>
using namespace std;

//код игры "Оружейный магазин".
int main (void)
{
     string name;
     cout << "Добро пожаловать в оружейный магазин, благородный рыцарь."
          << " Снова настала пора экипировать армию?" <<endl
          << "Как твое имя? ";
     cin >> name;
     cout << "Что ж, сэр " << name.c_str()
           << ", вперед, за покупками!" << endl;

     float gold = 50;
     int silver = 8;
     const float SILVERPERGOLD = 6.7;
     const float BROADSWORDCOST = 3.6;
     unsigned short broadswords;

     cout << "У тебя " << gold << " золотых монет и "
          << silver << " серебряных." <<endl<< "Это будет   ";
     gold += silver / SILVERPERGOLD;
     cout << gold << " золотых монет." << endl;

     cout<< "Сколько палашей ты желаешь приобрести?"
          <<" (каждый стоит 3.6 золотых)";
     cin >> broadswords;
     gold = gold - broadswords * BROADSWORDCOST;
     cout << "\nСпасибо за покупку. У тебя осталось " << gold << ".";
     silver = (gold - (int)gold) * SILVERPERGOLD;
     gold = (int)(gold);
     cout << "Это будет " << gold << " золотых и "
          << silver << " серебряных монет. " << endl
          << "Спасибо, что посетили Оружейный магазин. "
          << "Всего хорошего, сэр " << name.c_str();
     return 0;
}
```

Резюме

В этой главе мы изучили много всего. Вы узнали, как компьютер хранит информацию в памяти, как временно сохранять данные для последующего использования, а также о различных типах данных. Вы узнали о константах, определениях `typedef`, операторе присваивания и `sizeof()`. Кроме того, мы повторили уже изученный синтаксис языка C++.

Вы проделали большой путь и можете без опаски называть себя программистами. Это повод для гордости; очень немногие забираются так далеко.

В следующей главе мы познакомим вас с управляющими операторами, с которых начинается истинная власть над компьютером. Расслабьтесь на минутку, чтобы улеглась полученная информация, а затем экипируйтесь, прыгайте в кресло и не забудьте пристегнуть ремни – впереди изумительное путешествие!

Задания

1. Выберите подходящий тип переменной для хранения следующей информации:

   Число книг на книжной полке

   Стоимость этой книги

   Число людей в мире

   Слово *Привет*

2. Придумайте содержательные имена для переменных из первого задания.
3. Приведите две причины использования констант вместо литералов.
4. Напишите программу, которая вычисляет и отображает размеры всех основных типов.
5. Выясните, что происходит, когда символьный тип объявляется с модификатором `unsigned`. Соответствуют ли результаты ожиданиям? Сформулируйте причину соответствия или несоответствия.