Максим Динман

C++

ОСВОЙ НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.068+800.92С++
ББК 32.973.26-018.1
Д44

Динман М. И.

Д44 С++. Освой на примерах. — СПб.: БХВ-Петербург, 2006. — 384 с.: ил.

ISBN 5-94157-917-9

Подробно и доступно на занимательных примерах рассмотрены синтаксис, семантика и техника программирования на языке С++. Описаны все этапы проектирования программ, приведены подробные комментарии программного кода, проанализированы результаты вычислений, показаны типичные проблемы и пути их решения. Большое внимание уделяется алгоритмам и примерам решения задач при помощи графов, а также алгоритмам шифрования. Каждая глава содержит упражнения для самостоятельной работы.

Для учащихся и начинающих программистов

УДК 681.3.068+800.92С++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Наталия Першакова*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.06.06.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 24.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-917-9

Оглавление

Введение

Данная книга предназначена для тех, кто хочет освоить азы программирования на C++. Здесь каждый сможет найти много интересных и полезных вещей, которые помогут и при написании простых программ, и при разработке сложных проектов.

Программирование на C++ в современном обществе стало правилом хорошего тона. Большинство программ, предназначенных для работы в средах Windows и UNIX, были написаны на C либо C++.

Программируя на C++, вы не только сумеете автоматизировать свою работу, но и научиться думать по-иному, т. е. подходить к задаче любого рода, даже бытовой, с такой стороны, с которой на ее решение вами будет затрачено минимальное количество времени. Причем решение таких задач не будет содержать сложных для понимания фраз или выдержек из кода, а окажется простым и доступным для любого человека, никак не связанного с программированием.

Приведенные далее правила помогут вам в решении любой задачи:

- ищите решение задачи в ее условии;
- проанализируйте входные данные;
- всегда помните, что вам нужно получить в результате;
- не спешите сразу писать код той или иной задачи;
- решение может стать очевидным, если свести задачу к некоторым бытовым случаям;
- решите задачу в общих чертах;

- предусмотрите все частные случаи, если таковые имеются;
- перечислите компоненты и объекты, которые необходимы при решении задачи;
- определите свойства каждого используемого объекта;
- запрограммируйте составленный алгоритм;
- старайтесь код программы сопровождать понятными комментариями;
- если у вас возникли трудности, произведите пошаговое выполнение алгоритма, это поможет найти и устранить ошибку;
- никогда не думайте, что C++ умнее или в чем-то превосходит вас.

B1. О названии

Название C++ — изобретение совсем недавнее (лето 1983 года). Его придумал Рик Масситти. Название указывает на эволюционную природу перехода к нему от C. "++" — это операция приращения в C. Чуть более короткое имя C+ является синтаксической ошибкой. Кроме того, оно уже было использовано в названии совсем другого языка. Знатоки семантики C находят, что C++ хуже, чем ++C.

Более ранние версии языка использовались, начиная с 1980 года, и были известны как "C с классами". Первоначально язык был придуман потому, что автор хотел написать модели, управляемые прерываниями, для чего был бы идеален Simula67, если не принимать во внимание эффективность. "C с классами" использовался для крупных проектов моделирования, в которых строго тестировались возможности написания программ, требующих минимального пространства памяти и времени на выполнение. В "C с классами" не хватало перегрузки операций, ссылок, виртуальных функций и многих деталей. C++ был впервые введен за пределами исследовательской группы автора в июле 1983 года. Однако тогда многие особенности C++ были еще не придуманы.

B2. О разработке

Изначально C++ был разработан, чтобы автору языка и его коллегам не приходилось программировать на ассемблере, C или других современных языках высокого уровня. Основным его предназначением было сделать удобным написание сложных программ. Плана разработки C++ на бумаге никогда не было — проект, документация и реализация двигались одновременно.

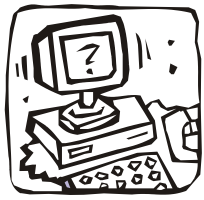
Разумеется, внешний интерфейс среды программирования C++ был написан на C++. Никогда не существовало "Проекта C++" и "Комитета по разработке C++". Поэтому язык развивался и продолжает развиваться во всех направлениях путем преодоления сложностей, с которыми сталкиваются программисты, а также в процессе дискуссий автора с коллегами.

В качестве базового языка для C++ был выбран C, потому что он многоцелевой, лаконичный, имеет относительно низкий уровень, отвечает требованиям большинства задач системного программирования, применим в любой операционной системе.

B3. В общих чертах о C++

C++ — универсальный язык программирования, наиболее подходящий для серьезного программиста. За исключением второстепенных деталей, C++ является надмножеством языка программирования C. Помимо возможностей, которые дает C, C++ предоставляет гибкие и эффективные средства определения новых типов, точно отвечающих концепциям приложения, благодаря чему программист может разделять разрабатываемую программу на легко поддающиеся контролю части. Такой метод построения программ часто называют абстракцией данных. C++ имеет гораздо лучшие, чем в C, средства выражения модульности программы и проверки типов. В языке есть также усовершенствования, не связанные непосредственно с классами, включающие в себя символические константы.

В C++ сохранены возможности языка C по работе с основными объектами аппаратного обеспечения (биты, байты, слова, адреса и т. п.). Это позволяет весьма эффективно реализовывать типы, определяемые пользователем. C++ и его стандартные библиотеки спроектированы так, чтобы обеспечить переносимость. Имеющаяся на текущий момент реализация языка может работать в большинстве систем, поддерживающих C. При написании программ на C++ можно использовать библиотеки языка C, а также бóльшую часть инструментальных средств, поддерживающих программирование на C.



Глава 1

Знакомство с приложением

1.1. Установка C++

Для установки среды разработки C++ запустите файл Setup.exe, который находится в папке инсталляции программы. На экране появится диалоговое окно установки (рис. 1.1). Для продолжения нажмите кнопку **Next** (Далее), оставив все значения без изменения.

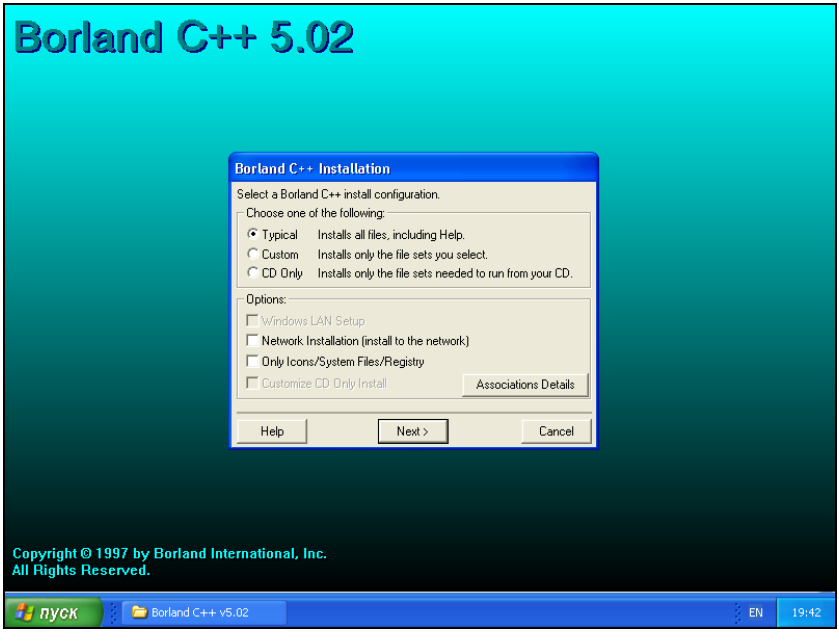


Рис. 1.1. Установка среды разработки C++

Затем укажите папку, в которой вы желаете хранить программу, например, C:\BC5, и нажмите кнопку **Next** (Далее). Значения параметров в следующих двух диалоговых окнах также оставьте без изменений.

На этом этап установки приложения завершен. Открыть среду разработки C++ для работы можно с помощью команды меню **Пуск | Все программы | Borland C++ 5.02 | Borland C++** (рис. 1.2).

Рис. 1.2. Открытие приложения Borland C++ 5.02

1.2. Рабочее окно программы

После открытия на экране вы увидите рабочее окно программы (рис. 1.3), которое состоит из следующих частей:

- меню;
- панель инструментов;
- рабочая область;
- строка состояния.

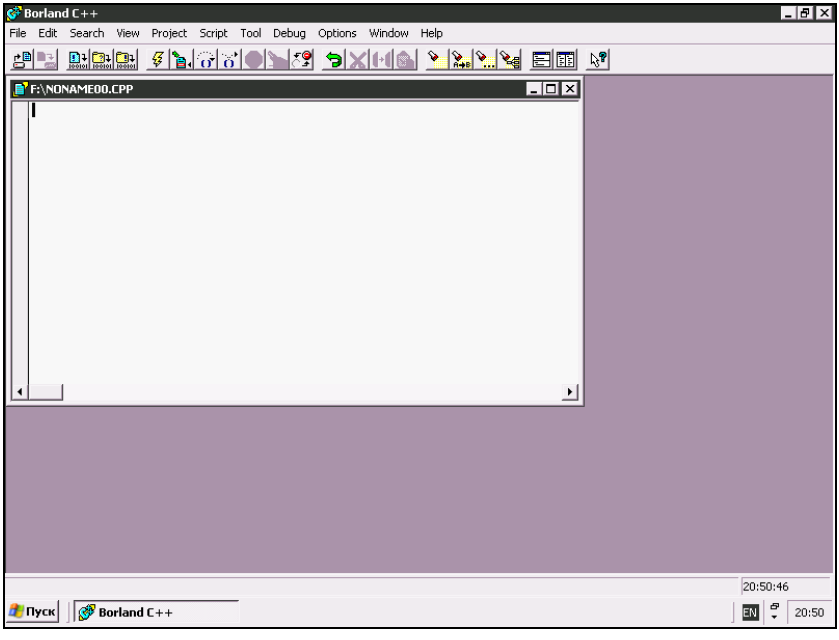


Рис. 1.3. Окно C++

В *строке состояния* программа отображает краткие подсказки, связанные с тем или иным объектом.

Приведем назначение некоторых пунктов меню:

- **File** (Файл) — это один из самых главных пунктов меню; он отвечает за открытие и сохранение проекта;

- **Edit** (Правка) — используется для копирования, вставки, вырезания информации, а также для отмены последних операций над текстом;
- **Search** (Поиск) — отвечает за поиск информации в тексте;
- **View** (Вид) — открывает окна, которые помогают при написании программы; подробно об этом пункте меню будет рассказано позже;
- **Project** (Проект) — предназначен для управления созданным проектом;
- **Options** (Опции) — этот пункт меню служит для настройки параметров программы и проекта;
- **Window** (Окно) — помогает расположить окна каскадом, а также переключаться между ними.

1.3. Создание и сохранение документа

Чтобы создать новый документ, выберите пункт меню **File | New | Text edit** (Файл | Создать | Текстовая область). В рабочей области появится окно — *редактор кода*, в котором и будет содержаться основной код программы (см. рис. 1.3).

Чтобы сохранить документ, выберите пункт меню **File | Save as...** (Файл | Сохранить как...). В результате откроется окно сохранения документа (рис. 1.4).

В этом окне укажите папку, в которой будет размещаться документ, и имя, под которым он будет храниться. Для подтверждения сохранения нажмите кнопку **Открыть**. Данная команда дублируется нажатием комбинации клавиш <Ctrl>+<K>+<S>.

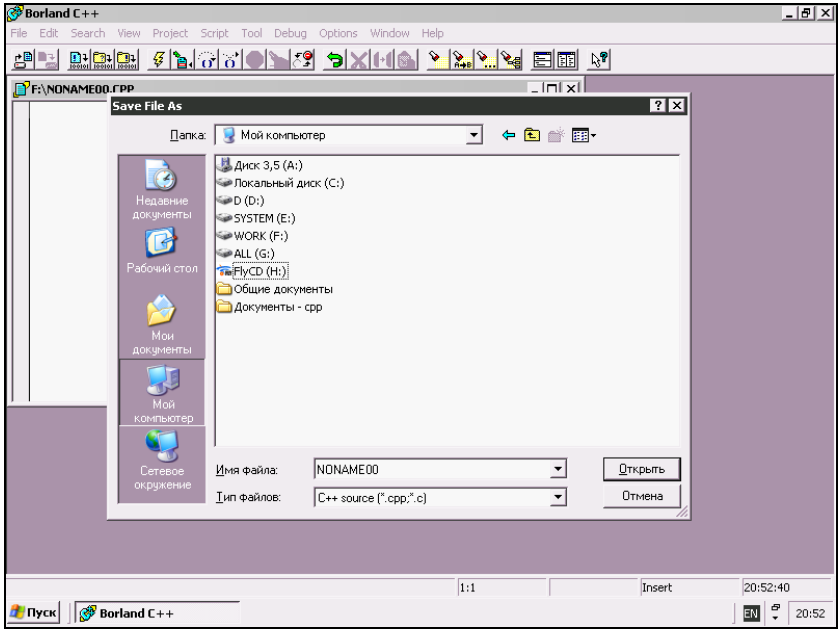


Рис. 1.4. Сохранение документа

1.4. Компиляция программы

После того как исходный текст набран (это можно сделать в любом текстовом редакторе, хотя существуют и специальные приложения), его необходимо преобразовать в программу, которая и будет выполняться на компьютере. Важно понять, что сам текст только формально описывает алгоритм вычислений — он не является программой.

Дело в том, что процессор понимает только двоичный код, представляющий собой очень простые машинные команды. В машинном коде написать более-менее сложную программу практически невозможно. В обычном языке программирования для операции над парой переменных достаточно одной команды, а процессору на это могут потребоваться десятки машинных команд.

В том, что машинные команды очень просты, а написать приличную программу с их помощью невероятно трудно, нет противоречия. Все детали механической части современного автомобиля совершают очень простые вращательные или поступательные движения, но невозможно создать автомобиль из куска металла с помощью молотка и напильника, нужны специальные инструментальные средства.

Точно так же дело обстоит и в вычислительной технике. Здесь тоже нужны специальные инструментальные средства. Перевод из текста в двоичный код осуществляется специальными программами, которые называются *трансляторами*. Они транслируют, т. е. переводят тексты, написанные на языке программирования, в машинный код.

В вычислительной технике существуют два класса трансляторов: интерпретаторы и компиляторы. *Интерпретатор* работает как синхронный переводчик — просматривает исходный текст строку за строкой, переводит каждую строку в промежуточный или в машинный код и передает его на исполнение. Если ошибок нет, интерпретатор приступает к следующей строке. *Компилятор* (а именно к этому классу относится рассматриваемая нами система программирования Borland C++) работает как литературный переводчик. Сначала он просматривает весь текст, иногда и не один раз, находит общие повторяющиеся места, тщательно готовит стратегию перевода, подбирает самые эффективные аналоги и только после этого переводит весь исходный текст целиком и полностью, создав при этом новый документ, который называется *объектным кодом*. Объектный код можно считать законченной программой, хотя и не вполне.

Итак, *компиляция* — это процесс преобразования исходной программы в исполняемую. Процесс компиляции состоит из двух этапов. На первом этапе выполняется проверка текста программы на отсутствие ошибок, на втором — генерируется исполняемая программа (exe-файл).

После ввода текста функции обработки события и сохранения проекта можно из меню **Project** (Проект) выбрать команду **Compile**

(Компиляция) или выполнить компиляцию, нажав клавишу <F9>. Диалоговое окно компилятора представлено на рис. 1.5.

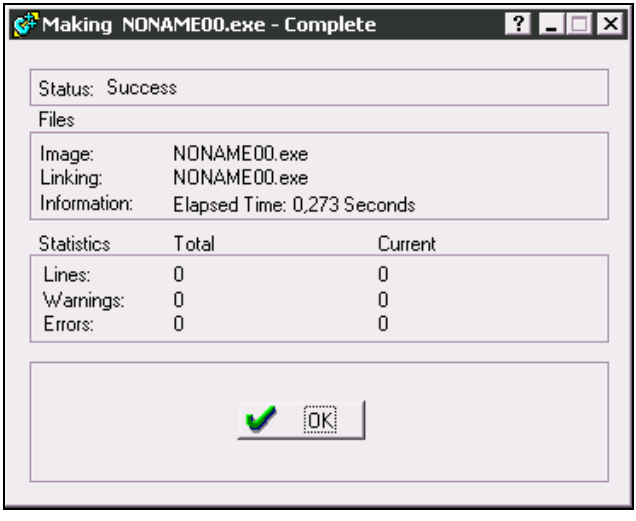


Рис. 1.5. Результаты компиляции в диалоговом окне

Если ошибок не найдено (**Errors** = 0), то можно запустить приложение, нажав комбинацию клавиш <Ctrl>+<F9>.

Компиляцию можно осуществлять и в пошаговом режиме с помощью клавиши <F7>. После ее нажатия строка, с которой начинается программа, будет выделена синим цветом. При повторном нажатии клавиши операции, находящиеся в этой строке, будут выполнены, и синим цветом подсветится следующая строка.

Запуск программы можно начать с любой желаемой строки, установив в ней курсор и нажав клавишу <F4>.

Глава 2

Основы программирования

В данной главе вы познакомитесь с основами программирования на C++, получите представление о программах и разработке алгоритмов для них.

2.1. Этапы разработки программы

Программа, которую выполняет компьютер, нередко отождествляется с самим компьютером, т. к. человек, использующий программу, вводит в компьютер исходные данные (например, при помощи клавиатуры), а компьютер выводит результат на экран, на принтер или в файл. Процессор преобразует исходные данные в результат по определенному алгоритму, который, будучи записан на специальном языке, называется *программой*. Таким образом, чтобы компьютер решил некоторую задачу, необходимо разработать последовательность команд, обеспечивающую решение данной задачи, или, как говорят, написать программу.

Выражение "написать программу" означает только один из этапов создания компьютерной программы, когда разработчик (программист) действительно пишет команды (инструкции) на бумаге или при помощи текстового редактора.

Программирование — это процесс создания (разработки) программы, который может быть представлен с помощью последовательности шагов:

1. Спецификация (определение, формулирование требований к программе).

2. Разработка алгоритма.
3. Кодирование (запись алгоритма на языке программирования).
4. Отладка.
5. Тестирование.

Спецификация

Спецификация — определение требований к программе — один из важнейших этапов, на котором подробно описывается исходная информация, формулируются требования к результату, поведение программы в особых случаях (например, при вводе неверных данных).

Разработка алгоритма

На этапе *разработки алгоритма* необходимо определить последовательность действий, которые надо выполнить для получения результата. Если задача может быть решена несколькими способами, то программист, используя некоторый критерий (например, скорость работы), выбирает наиболее подходящий алгоритм. Результатом этапа разработки алгоритма является его подробное словесное описание или блок-схема.

Кодирование

После того как определены требования к программе и составлен алгоритм решения, он записывается на выбранном языке программирования. В результате получается исходная программа.

Отладка

Отладка — это процесс поиска и устранения ошибок. Ошибки в программе разделяют на две группы: синтаксические (ошибки в тексте) и алгоритмические. Синтаксические ошибки устранить довольно легко, а алгоритмические ошибки обнаружить труднее. Этап отладки можно считать законченным, если программа правильно работает при любом правильном наборе входных данных.

Тестирование

Тестирование особенно важно, если вы предполагаете, что вашей программой будут пользоваться другие. На этом этапе следует проверить, как ведет себя программа при как можно большем количестве входных наборов данных, в том числе и заведомо неверных.

2.2. Элементы языка C++

Алфавит

Множество символов языка C++ включает прописные и строчные буквы латинского алфавита и цифры.

C++ чувствителен к регистру. Это означает, что "a" и "A" — совсем разные буквы.

Язык программирования, как и любой другой язык, имеет свой словарь. *Словарь языка программирования* представляет собой совокупность зарезервированных или так называемых "ключевых" слов. Все зарезервированные слова содержат только строчные буквы (символы нижнего регистра) и написаны на английском языке.

Пробел используется не только в качестве разделительного знака, но и как символьный знак.

Приведем список ключевых слов языка C++:

`asm`	`char`	`delete`	`extern`
`auto`	`class`	`do`	`float`
`break`	`const`	`double`	`for`
`case`	`continue`	`else`	`friend`
`catch`	`default`	`enum`	`goto`

```
if          protected   static      typedef
inline      public      struct      union
int         register    switch      unsigned
long        return      template    virtual
new         short       this        void
operator    signed      throw       volative
private     sizeof      try         while
```

Примечание

Использование части ключевых слов будет объяснено в процессе изложения материала, остальные войдут в словарь программиста по мере изучения C++.

Кроме того, в C++ используются следующие символы:

- знаки препинания: `. , : ;`;
- скобки: `( ) [ ] { }`;
- знаки: `| ^ ? _`;
- двойные и одинарные кавычки: `"` и `'`.

Алгоритм заключается в фигурные скобки `{}` после выражения `main()`, в котором круглые скобки показывают, что это программный блок, называемый *функцией*. Обычно программа состоит из нескольких функций, но все же `main()` присутствует всегда, с нее и начинается выполнение программы. Данная функция всегда оканчивается оператором `return 0;`, показывающим, что программа завершена. Вот пример простой программы:

```
main()
{
  return 0;
}
```

Внимание

Зарезервированные (ключевые) слова запрещается использовать в качестве имени переменных.

Комментарии

Комментарии, как и в других языках программирования, не обрабатываются компилятором, и поэтому не влияют на выполнение программы. В языке C++ можно записывать комментарии двух видов:

- комментарии можно разместить после двух подряд идущих слэшей `//`, записанных в любом месте строки. Все, что следует после этих символов до конца строки, воспринимается как комментарий. Для продолжения комментария на следующей строке необходимо в ней повторно записать эти два символа;
- перед началом комментария один раз записываются два подряд идущих символа `/*` (открытие комментария), а в конце — два подряд идущих символа `*/` (закрытие комментария). В этом случае комментарии могут занимать несколько строк.

Совет

Используйте первый способ для коротких комментариев, а второй — для длинных. Для доступности и облегчения восприятия кода другими программистами и пользователями всегда старайтесь снабжать программу подробными комментариями к коду.

Переменные

В процессе написания программы программисту приходится использовать *переменные*. Каждая переменная имеет имя, тип, размер и значение.

Имя переменной в прямом смысле является ее названием. Имя переменной, или идентификатор, может состоять из латинских букв, цифр и символа подчеркивания. Прописные и строчные буквы в именах различаются. Язык C++ не ограничивает длину идентификатора, однако пользоваться слишком длинными именами типа достаточно неудобно. Чтобы текст вашей программы был более понятным, рекомендуется использовать такие идентификаторы, которые несут какой-либо смысл, поясняя назначение объекта в программе, например, `birth_date` или `salary`.

Тип определяет, какие символы или числа записаны в ячейку памяти под этим именем.

Размер указывает максимальную величину или точность задания числа.

Значение определяет содержимое ячейки памяти.

Более общее определение переменной можно дать следующим образом: переменная — это именованная область памяти, к которой программист имеет доступ из программы.

Совет

Как правило, в программах переменные на том или ином шаге принимают разные значения. Для более быстрой и удобной ориентации в их значениях можно использовать окно **Watch** (Часы). Чтобы открыть данное окно, можно воспользоваться пунктами меню **View | Watch** (Вид | Часы). Для добавления некоторой переменной в окно нажмите комбинацию клавиш <Ctrl>+<A> и введите имя переменной. По завершению ввода нажмите клавишу <Enter>. Теперь при пошаговом выполнении программы вы сможете увидеть значение помещенной в окно переменной на данном шаге.

Типы данных

Основным принципом типизации, принятым в языках программирования и базах данных, является то, что любая константа, переменная, выражение и функция относится к некоторому типу, характеризующему прежде всего множество значений, к которым относятся константы, которые могут принимать переменные и выражения и которые могут формировать функции. При описании любых используемых констант, переменных и функций явно или неявно указывается их тип. В первую очередь это дает возможность компилятору и/или системе управления базами данных выделить для хранения объекта данных ровно тот объем памяти, который определяется допустимым диапазоном значений типа.

Однако концепция типа этим не исчерпывается. Следующим исключительно важным свойством типа данных является инкапсуляция внутреннего представления его значений (*инкапсуляция* — это механизм защиты данных от несанкционированного доступа;

она будет рассмотрена в *главе 8*). К значению типа данных (констант, переменных, выражений и функций) можно обращаться только с помощью операций, предопределенных в описании этого типа. Эти операции могут быть явными (например, арифметические операции `+`, `-`, `*` и `/` для числовых типов) или неявными (например, операция преобразования значения целого типа к значению типа с плавающей запятой). В некоторых языках (в C и C++ тоже) допускаются и явные преобразования типов.

Наличие типовых описаний констант, переменных и функций и предписанные правила определения типов выражений вместе с поддержкой свойства инкапсуляции типов дают возможность компиляторам языков программирования и языков баз данных производить существенный контроль допустимости языковых конструкций на этапе компиляции, что позволяет сократить число проверок на стадии выполнения программ и облегчить их отладку.

Перечислим существующие категории типов.

- *Встроенные типы данных*, т. е. типы, предопределенные в языке программирования. Обычно в языке фиксируются внешнее представление значений этих типов (вид литеральных констант) и набор операций с описанием их семантики. Внутреннее представление и реализация операций выбираются в конкретных компиляторах и подсистемах поддержки выполнения программ.
- *Конструируемые типы* (иногда их называют составными) обладают той особенностью, что в языке предопределены средства спецификации таких типов и некоторый набор операций, дающих возможность доступа к компонентам составных значений.
- *Указательные типы* дают возможность работы с типизированными множествами абстрактных адресов переменных, содержащих значения некоторого типа. В языках с более слабой типизацией (например, C и C++) допускаются практически неограниченные манипуляции указателями.

В C++ также реализован набор базовых типов данных, представленных в табл. 2.1.

Таблица 2.1. *Базовые типы данных*

Имя типа	Размер в байтах	Допустимое значение
`char`	1 байт	От –128 до 127
`int`	Зависит от реализации	
`short int`	2 байта	От –32 768 до 32 767
`long int`	4 байта	От –2 147 483 648 до 2 147 483 647
`unsigned char`	1 байт	От 0 до 255
`unsigned int`	Зависит от реализации	
`unsigned short int`	2 байта	От 0 до 65 535
`unsigned long int`	4 байта	От 0 до 4 294 967 295
`float`	4 байта	От ~3.4e – 38 до ~3.4e + 38
`double`	8 байтов	От ~1.7e – 308 до ~1.7e + 308
`long double`	10 байтов	От ~3.4e – 4932 до ~1.1e + 4932

Отметим, что ключевые слова `signed` и `unsigned` необязательны. Они указывают, как интерпретируется нулевой бит объявляемой переменной, т. е. если указано ключевое слово `unsigned`, то нулевой бит интерпретируется как часть числа, в противном случае нулевой бит интерпретируется как знаковый. В случае отсутствия ключевого слова `unsigned` целая переменная считается знаковой. В том случае, если спецификатор типа состоит из ключевого типа `signed` или `unsigned` и далее следует идентификатор переменной, то она будет рассматриваться как переменная типа `int`.

Чтобы назначить переменной тот или иной тип, необходимо указать его перед первым использованием переменной. Например:

```
unsigned int A;
int B;          /* подразумевается signed int B */
unsigned C;     /* подразумевается unsigned int C*/
```

Так же можно по одному типу определять несколько переменных, тогда список переменных следует указать через запятую.

Простое определение переменной не задает ее начального значения. Если объект определен как глобальный, спецификация C++ гарантирует, что он будет инициализирован нулевым значением. Если же переменная локальная либо динамическая, ее начальное значение не определено, т. е. она может содержать некоторое случайное значение.

Совет

Довольно часто после написания и исполнения программы полученный результат оказывается неверным, хотя алгоритм правильный. Данная ошибка возможна из-за того, что в начале программы не было задано значение некоторой переменной. Поэтому рекомендуется явно указывать начальное значение переменной, по крайней мере, в тех случаях, когда неизвестно, может ли объект инициализировать сам себя.

Преобразование типов

При выполнении операций производится автоматическое преобразование типов, чтобы привести операнды выражений к общему типу или чтобы расширить короткие величины до размера целых величин, используемых в машинных командах. Выполнение преобразования зависит от специфики операций и от типа операндов.

Рассмотрим общие арифметические преобразования:

- операнды типа `float` преобразуются к типу `double`;
- если один операнд — `long double`, то второй преобразуется к этому же типу;
- если один операнд — `double`, то второй также преобразуется к типу `double`;
- любые операнды типа `char` и `short` преобразуются к типу `int`;
- любые операнды `unsigned char` или `unsigned short` преобразуются к типу `unsigned int`;
- если один операнд — `unsigned long`, то второй преобразуется к типу `unsigned long`;

- если один операнд — `long`, то второй преобразуется к типу `long`;
- если один операнд — `unsigned int`, то второй операнд преобразуется к этому же типу.

Таким образом, при вычислении выражений операнды преобразуются к типу того операнда, который имеет наибольший размер.

Константы

Константы, в отличие от переменных, не могут изменяться программой. Записываются они по следующим правилам:

- вещественные константы можно записать в обычной форме, используя символ . (точка) для разделения целой и дробной части (`0.54`) или в экспоненциальной форме (`0.5e1`, `-0.12345e+2`, `987e-5`);
- целочисленные константы можно записать в десятичной (`-26`) или шестнадцатеричной системе счисления (`0x1A`);
- символьные константы записываются в одинарных кавычках (`'5'`, `'A'`, `'\t'`, `'\r'`);
- строковые константы записываются в двойных кавычках (`"Line"`).

Примечание

Грамматический контроль строк не выполняется.

Константы можно определить одним из следующих способов:

- непосредственно записать в выражении;
- с помощью ключевого слова `const`, например, `const Long=5`. Тогда в выражении вместо константы `5` указывается идентификатор `Long`. Этот способ имеет следующее преимущество. Если одна и та же константа в программе встречается несколько раз, то достаточно изменить ее значение один раз при объявлении;
- с помощью директивы препроцессора `#define`, например, `#define Long 5`. Директива заменяет каждое появление сим-

волов `Long` на `5`. Подробней о директивах препроцессора будет рассказано позже.

Арифметические и логические операции

В C++ используются *арифметические* и *логические операции*. Их перечень приведен в табл. 2.2.

Таблица 2.2. *Арифметические и логические операции*

Название операции	**Символ, который используется в коде**
Арифметическое сложение	`+`
Арифметическое вычитание	`-`
Умножение	`*`
Деление	`/`
Отрицание	`!`
Присваивание	`=`
Вычисление остатка	`%`
Логическое сложение	`&&`
Логическое умножение	`||`
Проверка на равенство	
Равно	`= =`
Не равно	`!=`
Проверка отношения	
Больше	`>`
Меньше	`<`
Больше или равно	`>=`
Меньше или равно	`<=`