1

Начало работы с C++

В ЭТОЙ ГЛАВЕ...

- История и философия развития языков C и C++
- Сравнение процедурного и объектно-ориентированного программирования
- Добавление принципов объектно-ориентированного программирования в язык C
- Добавление принципов обобщенного программирования в язык C
- Стандарты языков программирования
- Порядок создания программы

Добро пожаловать в C++! Этот удивительный язык, сочетающий в себе функциональные возможности языка C и принципы объектно-ориентированного и обобщенного программирования, в девяностых годах прошлого столетия занял лидирующую позицию среди существующих языков программирования, продолжая удерживать ее и в двухтысячных годах. Своим происхождением он обязан языку программирования C, поэтому ему свойственны такие характеристики, как эффективность, компактность, быстродействие и переносимость. Благодаря принципам объектной ориентации, язык программирования C++ предлагает новую методологию программирования, которая позволяет решать современные задачи, степень сложности которых постоянно растет. Благодаря возможности использования шаблонов, язык C++ предлагает еще одну новую методологию: обобщенное программирование. Во всем этом есть свои положительные и отрицательные стороны. В конечном счете, C++ является очень мощным языком программирования, но вместе с тем на его изучение нужно потратить довольно много времени.

Мы начнем с небольшого экскурса в историю происхождения языка C++, а затем перейдем к основным правилам создания программ на C++. Оставшаяся часть книги будет сконцентрирована на том, чтобы научить читателя программированию на языке C++: вначале мы рассмотрим простые основы языка, после чего приступим к изучению объектно-ориентированного программирования (ООП), освоим весь его жаргон — объекты, классы, инкапсуляцию, сокрытие данных, полиморфизм и наследование — и напоследок перейдем к изучению обобщенного программирования. (Естественно, по мере изучения C++ эти профессиональные термины перейдут из разряда модных словечек в лексикон опытного разработчика.)

Изучение языка C++: с чем придется иметь дело

В языке программирования C++ объединены три отдельных категории программирования: процедурный язык, представленный C; объектно-ориентированный язык, представленный расширениями в форме классов, которые C++ добавляет к C; и обобщенное программирование, поддерживаемое шаблонами C++. В этой главе как раз и рассматриваются упомянутые категории. Однако, прежде всего давайте разберемся с тем, как эти категории влияют на изучение C++. Одна из причин использования языка C++ связана с желанием получить в свое распоряжение возможности объектной ориентации. Для этого вы должны обладать навыками работы со стандартным языком C, который предлагает для C++ базовые типы, операции, управляющие структуры и синтаксические правила. Поэтому, зная язык программирования C, вы можете смело приступать к изучению C++. Это, однако, вовсе не означает, что вам достаточно будет изучить одни лишь ключевые слова и конструкции. Переход от C к C++ может потребовать стольких же усилий, сколько изучение языка C с самого начала. Кроме этого, если вы знакомы с языком C, то при переходе к C++ вы должны будете отказаться от привычного вам способа написания программ. Если вы не знаете C, то для изучения языка C++ вам придется освоить компоненты языка C, компоненты ООП и обобщенные компоненты, но, по крайней мере, вам не придется забывать привычный способ написания программ. Если у вас начинает складываться впечатление, что для изучения языка программирования C++ придется хорошенько напрячь свои умственные способности, то вы не ошибаетесь. Эта книга послужит хорошим помощником в процессе обучения, поэтому вы сможете сэкономить свои силы.

Настоящая книга построена таким образом, что в ней рассматриваются как элементы языка C, лежащего в основе C++, так и новые компоненты, поэтому она будет полезна даже для неподготовленных читателей. Процесс обучения начинается с рассмотрения функциональных возможностей, общих для обоих языков. Даже если вы знаете язык C, эта часть книги окажется полезным повторением. В ней уделено внимание концепциям, значение которых будет раскрыто позже, и показаны различия между C++ и C. После того как вы усвоите основы языка C, мы перейдем к знакомству с суперструктурой C++. С этого момента мы займемся рассмотрением объектов и классов и вы узнаете, как они реализованы в C++. Затем мы обратимся к шаблонам.

Эта книга не является полным справочником по языку C++, и в ней не рассматриваются все его тонкости. Однако у вас есть возможность изучить большинство основных особенностей языка, в том числе шаблоны, исключения и пространства имен.

А теперь давайте рассмотрим вкратце историю происхождения языка C++.

Истоки языка C++: немного истории

Развитие компьютерных технологий в течение последних нескольких десятков лет происходило удивительно быстрыми темпами. Современные ноутбуки могут производить вычисления гораздо быстрее и хранить намного больше информации, чем вычислительные машины 60-х годов прошлого столетия. (Очень немногие программисты могут вспомнить, как им приходилось возиться с колодами перфокарт, чтобы загрузить их в огромную компьютерную систему, занимающую отдельную комнату и имеющую немыслимый по тем временам объем памяти 100 Кбайт — намного меньше, чем в настоящее время располагает рядовой смартфон.) В ногу со временем развивались и языки программирования. Их развитие не было столь впечатляющим, однако имело огромное значение. Для больших и мощных компьютеров требовались более сложные программы, для которых, в свою очередь, нужно было решать новые задачи управления и сопровождения программ.

В семидесятых годах прошлого столетия такие языки программирования, как C и Pascal, способствовали зарождению эры структурного программирования, привнесшего столь необходимые на то время порядок и дисциплину. Помимо того, что язык C предлагал инструментальные средства для структурного программирования, с его помощью можно было создавать компактные быстро выполняющиеся программы, а также справляться с решением аппаратных задач, например, с управлением коммуникационными портами и дисковыми накопителями. Благодаря этим характеристикам, в восьмидесятых годах язык C стал доминирующим языком программирования. Наряду с ростом популярности языка C зарождалась и новая парадигма программирования: объектно-ориентированное программирование (ООП), которое было реализовано в таких языках, как SmallTalk и C++. Давайте рассмотрим взаимоотношения языка C и ООП более подробно.

Язык программирования C

В начале семидесятых годов прошлого столетия Деннис Ритчи (Dennis Ritchie), сотрудник компании Bell Laboratories, участвовал в проекте по разработке операционной системы Unix. (*Операционная система* (ОС) представляет собой набор программ, предназначенных для управления аппаратными ресурсами и обслуживания взаимодействий пользователя и компьютера. Так, например, именно ОС выводит на экран монитора системное приглашение в терминальном интерфейсе, управляет окнами и мышью в графическом интерфейсе и запускает программы на выполне-

ние.) В своей работе Ритчи нуждался в языке программирования, который был бы лаконичным, с помощью которого можно было бы создавать компактные и быстро выполняющиеся программы, и посредством которого можно было бы эффективно управлять аппаратными средствами.

По сложившейся на то время традиции программисты использовали для решения этих задач язык ассемблера, тесно связанный с внутренним машинным языком. Однако язык ассемблера является языком *низкого уровня*, т.е. он работает непосредственно с оборудованием (например, напрямую обращается к регистрам центрального процессора и ячейкам памяти). Таким образом, язык ассемблера является специфическим для каждого процессора компьютера. Поэтому если вы хотите, чтобы ассемблерная программа, написанная для компьютера одного типа, могла работать на компьютере другого типа, то, возможно, вам придется полностью переписать программу на другом языке ассемблера. Эта ситуация похожа на то, когда вы приобрели новый автомобиль и обнаружили, что конструкторы изменили расположение рычагов управления и их назначение, поэтому вам теперь нужно заново научиться водить машину.

Однако ОС Unix предназначалась для работы на самых разнообразных типах компьютеров (или платформ). Таким образом, это предполагало использование языка программирования *высокого уровня*. Такой язык ориентирован на решение задач, а не на обслуживание определенного оборудования. Специальные программы, называемые *компиляторами*, переводят язык программирования высокого уровня на внутренний язык определенного компьютера. Поэтому одну и ту же программу, написанную на языке программирования высокого уровня, можно запускать на различных платформах, применяя разные компиляторы. Ритчи необходим был язык, который сочетал бы в себе эффективность языка низкого уровня и возможность доступа к аппаратным средствам с универсальностью и переносимостью языка высокого уровня. Поэтому, основываясь на старых языках программирования, он создал язык C.

Философия программирования на языке C

Поскольку C++ прививает языку C новые принципы программирования, мы должны сначала изучить философию программирования на языке C. В общем случае компьютерные языки имеют дело с двумя концепциями — данные и алгоритмы. *Данные* — это информация, которую использует и обрабатывает программа. *Алгоритмы* — это методы, используемые программой (рис. 1.1). Как и большинство распространенных в то время языков программирования, язык C был *процедурным* языком программирования. Это означает, что в этом языке акцент ставился на обработку данных с помощью алгоритмов. Суть процедурного программирования заключается в том, чтобы запланировать действия, которые должен предпринять компьютер, и с помощью языка программирования их реализовать. В программе предварительно описывается некоторое количество процедур, которые должен будет выполнить компьютер, чтобы получить определенный результат. Здесь в качестве примера можно упомянуть кулинарный рецепт, в котором записан порядок действий, выполнив которые кондитер сможет получить пирог.

В ранних процедурных языках программирования, к которым относятся FORTRAN и BASIC, с увеличением размера программ возникали организационные проблемы. Например, в программах часто используются операторы ветвления, которые передают выполнение тому или иному набору инструкций в зависимости от результата проверки.

Рис. 1.1. *Данные + алгоритмы = программа*

В большинстве старых программ было настолько много запутанных переходов (на профессиональном жаргоне это называется "спагетти-кодом"), что при прочтении программу практически невозможно было понять, и попытка модифицировать такую программу сулила одни неприятности. Чтобы выйти из ситуации такого рода, специалисты по вычислительной технике разработали более дисциплинированный стиль программирования, называемый *структурным программированием*.

Язык программирования C обладает всеми возможностями для реализации этого подхода. Например, структурное программирование ограничивает ветвление (выбор следующей инструкции для выполнения) небольшим набором удобных и гибких конструкций. В языке C эти конструкции (циклы `for`, `while`, `do while` и оператор `if else`) включены в его словарь.

Другим новшеством было так называемое *нисходящее проектирование*. В языке C эта идея состояла в том, чтобы разделить большую программу на небольшие, поддающиеся управлению задачи. Если после разбиения одна из задач все равно остается крупной, этот процесс продолжается до тех пор, пока программа не будет разделена на небольшие модули, с которыми будет просто работать. (Организуйте свой процесс обучения. Ах, нет! Приведите в порядок свой рабочий стол, картотеку и наведите порядок на книжных полках. Ах, нет! Начните со стола, наведите порядок в каждом выдвижном ящике, начиная со среднего. Да, судя по всему, справиться с этой задачей вполне реально.) Этот подход вполне осуществим в языке C, т.к. он позволяет разрабатывать программные модули, называемые *функциями*, которые отвечают за выполнение конкретной задачи. Как вы могли заметить, в структурном программировании отображено процедурное программирования, в том смысле, что программа представляется в виде действий, которые она должна выполнить.

Переход к C++: объектно-ориентированное программирование

Несмотря на то что принципы структурного программирования делают сопровождение программ более ясным, надежным и простым, написать большую программу все еще было непросто. Новый подход к решению этой проблемы был воплощен в

объектно-ориентированном программировании (ООП). В отличие от процедурного программирования, в котором акцент делается на алгоритмах, в ООП во главу угла поставлены данные. Вместо того чтобы пытаться решать задачу, приспосабливая ее к процедурному подходу в языке программирования, в ООП язык программирования приспосабливается к решению задачи. Суть заключается в том, чтобы создать такие формы данных, которые могли бы выразить важные характеристики решаемой задачи.

В языке программирования C++ существуют понятия *класса*, который представляет собой спецификацию, описывающую такую новую форму данных, и *объекта*, который представляет собой индивидуальную структуру данных, созданную в соответствии с этой спецификацией. Например, класс может описывать общие свойства руководителя компании (фамилия, занимаемая должность, годовой доход, необычные способности и т.п.), а объект может представлять конкретного человека (например, Гилфорд Шипблатт, вице-президент компании, годовой доход составляет $925 000, умеет восстанавливать системный реестр Windows). В общем, класс описывает, какие данные используются для отображения объекта *и* какие операции могут быть выполнены над этими данными. Можно, например, определить класс для описания прямоугольника. Часть, касающаяся данных, этой спецификации может включать расположение вершин прямоугольника, высоту и ширину, цвет и стиль линии контура, а также цвет шаблона для закраски прямоугольника. Часть, касающаяся операций, этой спецификации может содержать методы для перемещения прямоугольника, изменения его размеров, вращения, изменения цвета и шаблонов, а также копирования прямоугольника в другое местоположение. Если позже вы воспользуетесь своей программой для рисования прямоугольника, она сможет создать объект в соответствии с его описанием. Объект будет содержать все значения данных, описывающие прямоугольник, и для изменения прямоугольника можно применять методы класса. Если вы нарисуете два прямоугольника, программа создаст два объекта, по одному для каждого прямоугольника.

Подход к проектированию программ, применяемый в ООП, заключается в том, чтобы сначала спроектировать классы, в точности представляющие все элементы, с которыми будет работать программа. Например, программа рисования может работать с классами, представляющими прямоугольники, линии, окружности, кисти, перья и т.п. В описаниях классов, как вы теперь уже знаете, указываются разрешенные операции для каждого класса, такие как перемещение окружности или вращение линии. Затем можно продолжить процесс разработки программы уже с помощью объектов для этих классов. Процесс перехода с нижнего уровня организации, например, с классов, до верхнего уровня — проектирования программы, называется *восходящим* программированием.

ООП позволяет не только связывать данные и методы в единственное определение класса. Например, ООП упрощает создание повторно используемого кода, что позволяет иногда сократить большой объем работы. Сокрытие информации позволяет предотвратить несанкционированный доступ к данным. Благодаря полиморфизму можно создавать множество описаний для операций и функций с программным контекстом, определяющим, какое описание используется. Благодаря наследованию, можно порождать новые классы на основе существующих классов. Как видите, ООП привносит множество новых идей и, в отличие от процедурного программирования, является другим принципом программирования. Вместо того чтобы сосредоточиваться на задачах, вы концентрируете внимание на концепциях представления. В некоторых случаях вместо нисходящего можно применять восходящий подход. Все эти вопросы, сопровождаемые простыми и понятными примерами, будут рассмотрены в данной книге.

Спроектировать полезный и надежный класс не так-то просто. К счастью, языки объектно-ориентированного программирования упрощают встраивание существующих классов в решаемые задачи. Поставщики программного обеспечения предлагают большое количество полезных библиотек классов, в том числе библиотеки, упрощающие написание программ для сред вроде Windows или Macintosh. Одно из замечательных преимуществ языка C++ заключается в том, что он позволяет без особых сложностей использовать повторно и адаптировать существующий надежный код.

C++ и обобщенное программирование

Еще одной парадигмой программирования, реализованной в языке C++, является обобщенное программирование. Как и ООП, обобщенное программирование направлено на упрощение повторного использования кода и на абстрагирование общих концепций. Однако в ООП акцент программирования ставится на данные, а в обобщенном программировании — на независимость от конкретного типа данных. И его цель тоже другая. ООП — это инструмент для управления большими проектами, тогда как обобщенное программирование предлагает инструменты для решения простых задач, подобных сортировке данных или слияния списков. Термин *обобщенный* относится к коду, тип которого является независимым. Данные в C++ могут быть представлены разными способами: в виде целых чисел, чисел с дробной частью, в виде символов, строк символов и в форме определяемых пользователем сложных структур нескольких типов. Например, если вам необходимо сортировать данные различных типов, то в общем случае для каждого типа потребовалось бы создавать отдельную функцию сортировки. При обобщенном программировании язык расширен, поэтому вы можете один раз написать функцию для обобщенного (т.е. не указанного) типа и применять ее для множества существующих типов. Для этого в языке C++ предусмотрены шаблоны.

Происхождение языка программирования C++

Как и C, язык C++ был создан в начале восьмидесятых годов прошлого столетия в Bell Laboratories, где работал Бьярне Страуструп (Bjarne Stroustrup). Вот что об этом говорит сам Страуструп: “C++ был создан главным образом потому, что мои друзья, да и я сам, не имели никакого желания писать программы на ассемблере, C или каком-нибудь языке программирования высокого уровня, существовавшем в то время. Задача заключалась в том, чтобы сделать процесс написания хороших программ простым и более приятным для каждого программиста”.

Домашняя страница Бьярне Страуструпа

Бьярне Страуструп, создатель языка программирования C++, является автором нескольких широко известных справочных руководств — *The C++ Programming Language* и *The Design and Evolution of C++*. Его персональный веб-сайт, размещенный в AT&T Labs Research, должен быть в числе ваших основных закладок:

`www.research.att.com/~bs`

На этом сайте можно найти интересные исторические предпосылки возникновения C++, биографические материалы Бьярне Страуструпа и часто задаваемые вопросы по C++. Удивительно, но чаще всего Страуструпа спрашивают о том, как правильно произносится его имя и фамилия. Просмотрите раздел часто задаваемых вопросов на указанном веб-сайте и загрузите файл `.WAV`, чтобы услышать это самому!

Страуструп стремился больше к тому, чтобы сделать язык C++ полезным, а не внедрить какой-нибудь новый принцип или стиль программирования. Средства языка C++ в первую очередь должны удовлетворять потребностям программиста, а не быть воплощением красивой теории. За основу C++ Страуструп взял язык C, известный своей лаконичностью, пригодностью для системного программирования, широкой доступностью и тесной взаимосвязью с ОС Unix.

Принципы ООП были позаимствованы в языке моделирования Simula67. Страуструпу удалось реализовать в языке C принципы ООП и поддержку обобщенного программирования без существенного изменения языка. Поэтому язык программирования C++ в первом приближении можно рассматривать как расширенный набор C; в пользу этого свидетельствует тот факт, что любая допустимая программа на языке C — это также допустимая программа на C++. Правда, существуют некоторые незначительные отличия, но не более того. Программы, написанные на C++, могут пользоваться существующими библиотеками программного обеспечения для C. *Библиотека* представляет собой коллекцию программных модулей, которые могут быть вызваны из программы. В этих библиотеках предлагаются надежные и проверенные решения ко многим наиболее часто встречающимся задачам программирования, позволяя тем самым экономить ваши усилия и время. Распространение языка C++ стало возможным во многом благодаря библиотекам.

Название языка C++ происходит от операции инкремента (++) в языке C, которая увеличивает на единицу значение переменной. Таким образом, имя C++ в точности отражает расширенную версию языка C.

Компьютерная программа переводит практическую задачу в последовательность действий, которые должен выполнить компьютер. Благодаря принципам ООП, язык C++ может устанавливать связь с концепциями, составляющими задачу, а благодаря возможностям языка C, он может работать с оборудованием (рис. 1.2). Такое сочетание возможностей способствовало росту популярности языка C++. Процесс переключения с одного аспекта программы на другой можно представить себе как процесс переключения передач в автомобиле. (В действительности некоторые приверженцы чистоты ООП считают, что реализация принципов ООП в языке C сродни попытке научить летать поросенка.) Кроме того, поскольку C++ внедряет принципы ООП в язык C, их можно просто проигнорировать. Однако в этом случае вы утратите массу возможностей.

После того, как C++ действительно стал популярным языком программирования, Страуструп ввел шаблоны, открыв возможности для обобщенного программирования. И только после этого стало ясно, насколько удачным было использование шаблонов — их значение оказалось даже большим, чем реализация ООП, хотя кто-то может и не согласиться с этим. Факт объединения в языке C++ ООП, обобщенного программирования и более традиционного процедурного подхода свидетельствует о том, что в C++ утилитарный подход господствует над идеологическим, и это одна из причин популярности данного языка.

Переносимость и стандарты

Представьте, что у себя на работе вы написали удобную программу на языке C++ для старенького ПК Pentium, функционирующего под управлением ОС Windows 2000, но руководство решило заменить эту машину новым компьютером, на котором установлена другая ОС, скажем, Mac OS X или Linux, и другой процессор, такой как SPARC. Сможете ли вы запустить свою программу на новой платформе? Естественно, вам придется повторно скомпилировать программу, используя компилятор C++ для

новой платформы. А нужно ли что-нибудь изменять в уже написанном коде? Если программу можно перекомпилировать, ничего в ней не меняя, и без помех запустить, то такая программа называется *переносимой*.

Рис. 1.2. *Двойственность языка C++*

Чтобы сделать программу переносимой, нужно справиться с двумя проблемами. Первая — это оборудование. Программа, настроенная на работу с конкретным аппаратным обеспечением, вряд ли будет переносимой. Если программа напрямую управляет платой видеоадаптера IBM PC, то на платформе Sun, например, она выдаст сплошную тарабарщину. (Проблему переносимости можно свести к минимуму, если локализовать части программы, зависящие от оборудования, в модулях функций; затем эти модули можно будет просто переписать.) В этой книге мы не будем рассматривать такой способ программирования.

Второй проблемой является несоответствие языков программирования. Вы наверняка согласитесь с утверждением, что в реальном мире тоже существует проблема с разговорными языками. Жители Бруклина, например, могут не понять сводку новостей на диалекте Йоркшира, и это притом, что все они разговаривают на английском языке. Такая же ситуация и в мире компьютеров: среди языков программирования можно различить характерные "диалекты". Хотя большинство разработчиков хотели бы добиться совместимости их собственных версий C++ с другими версиями, сделать это очень трудно без опубликованного стандарта, описывающего точную работу языка. В Национальном институте стандартизации США (American National Standards Institute — ANSI) в 1990 г. был сформирован комитет (ANSI X3J16), задача которого заключалась в разработке стандарта для языка программирования C++. (Стандарт для языка C уже был создан ANSI.)

Вскоре к этому процессу подключилась и Международная организация по стандартизации (International Organization for Standardization — ISO), у которой на тот момент был сформирован собственный комитет (ISO-WG-21). В результате усилия комитетов ANSI и ISO были объединены, после чего работа по созданию стандарта для языка C++ велась совместно.

Несколько лет напряженной работы, наконец, вылились в международный стандарт (ISO/IEC 14882:1998), который в 1998 г. был принят ISO, Международной электротехнической комиссией (International Electrotechnical Commission — IEC) и ANSI. Этот стандарт, часто называемый C++98, не только уточнял описание существующих средств языка C++, но также и расширений языка: исключений, идентификации типов во время выполнения (Runtime Type Identification — RTTI), шаблонов и стандартной библиотеки шаблонов (Standard Template Library — STL). В 2003 г. было опубликовано второе издание стандарта C++ (ISO/IEC 14882:2003); новое издание представляло собой формально пересмотренную версию. В ней были исправлены ошибки первого издания (ликвидированы опечатки, устранены неточности и т.п.), при этом средства языка программирования не менялись. Это издание стандарта часто называют C++03. Поскольку в C++03 средства самого языка остались не поменялись, мы будем понимать под C++98 как собственно C++98, так и C++03.

Язык C++ продолжает развиваться, и в августе 2011 г. комитет ISO одобрил новый стандарт под названием ISO/IEC 14882:2011, на который неформально ссылаются, как на C++11. Подобно C++98, стандарт C++11 добавляет к языку множество средств. Кроме того, его целями являются удаление противоречий, а также упрощение изучения и применения C++. Это стандарт был назван C++0x, при этом изначально ожидалось, что x будет 7 или 8, однако работа над стандартами — медленный, обстоятельный и утомительный процесс. К счастью, вскоре стало понятно, что 0x может рассматриваться как шестнадцатеричное целое число (см. приложение A), а это означало, что у комитета есть время до 2015 г. на завершение работы. Таким образом, согласно этому измерению, они закончили с опережением графика.

В стандарте ISO для языка C++ дополнительно приводится стандарт ANSI для языка C, поскольку считается, что C++ является расширенным набором C. Это означает, что любая допустимая программа на C в идеале также должна быть допустимой программой на C++. Между ANSI C и соответствующими правилами для C++ имеются некоторые различия, однако все они несущественны. Так, например, ANSI C включает некоторые возможности, впервые представленные в C++: прототипирование функций и спецификатор типа `const`.

До выхода ANSI C сообщество программистов на C следовало стандарту де-факто, основанному на книге *Язык программирования C* (Издательский дом "Вильямс", 2005 г.), написанной Брайаном Керниганом и Деннисом Ритчи. Этот стандарт часто назывался как K&R C; после появления ANSI C более простой стандарт K&R C теперь иногда называют *классическим C*.

Стандарт ANSI C не только определяет язык C, но и стандартную библиотеку C, которую должны поддерживать реализации ANSI C. Эта библиотека используется также и в C++; в книге мы называем ее *стандартной библиотекой C* или просто *стандартной библиотекой*. Кроме того, стандарт ISO C++ предоставляет стандартную библиотеку классов C++.

Стандарт C был пересмотрен и в результате получен стандарт C99, который был принят ISO в 1999 г. и ANSI в 2000 г. Этот стандарт добавил к языку C ряд средств, таких как новый целочисленный тип, который поддерживается некоторыми компиляторами C++.

Развитие языка

Первоначально стандарт де-факто для C++ был 65-страничным справочным руководством, включенным в 328-страничную книгу Страуструпа *The C++ Programming Language* (Addison-Wesley, 1986 г.).

Следующим важным опубликованным стандартом де-факто была книга Эллиса и Страуструпа *The Annotated C++ Reference Manual* (Addison-Wesley, 1990 г.). Ее объем составлял 453 страницы; в дополнение к справочному материалу она включала существенные комментарии.

Стандарт C++98 с добавленным множеством средств занимает около 800 страниц, при минимальном числе комментариев.

Описание стандарта C++11 потребовало свыше 1 350 страниц, что существенно превышает объем старого стандарта.

Эта книга и стандарты C++

Современные компиляторы обеспечивают хорошую поддержку для C++98. На момент написания этой книги некоторые компиляторы также поддерживали ряд средств C++11, и можно ожидать, что после принятия нового стандарта уровень поддержки станет быстро возрастать. Эта книга отражает текущую ситуацию, подробно раскрывая C++98 и представляя многие возможности C++11. Некоторые из этих возможностей описываются в рамках соответствующих тем по C++98. В главе 18 все внимание сосредоточено полностью на новых средствах, с подведением итогов по упомянутым ранее возможностям и представлением дополнительных новых средств.

Из-за неполной поддержки на время написания этой книги очень трудно адекватно представить абсолютно все новые возможности C++11. Но даже когда новый стандарт станет поддерживаться полностью, очевидно, что полное его изложение будет выходить далеко за рамки книги разумного объема. Подход, применяемый в настоящей книге, предусматривает концентрацию на средствах, которые уже доступны в ряде компиляторов, и краткое описание множества других возможностей.

Прежде чем вплотную заняться изучением языка C++, давайте рассмотрим порядок создания программ.

Порядок создания программы

Предположим, что вы написали программу на C++. Как вы собираетесь ее запускать? Действия, которые должны быть предприняты, зависят от компьютерной среды и применяемого компилятора C++, однако в общем случае они должны быть примерно следующими (рис. 1.3).

1. С помощью текстового редактора напишите свою программу и сохраните ее в файле. Это будет *исходный код* вашей программы.
2. Скомпилируйте исходный код. Для этого необходимо запустить программу, которая транслирует исходный код во внутренний язык, называемый *машинным языком* рабочего компьютера. Файл, содержащий транслированную программу — это *объектный код* вашей программы.

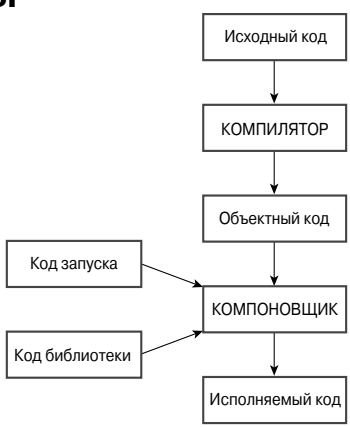

Рис. 1.3. *Этапы программирования*

3. Свяжите объектный код с дополнительным кодом. Например, программы на C++ обычно используют *библиотеки*. Библиотека C++ содержит объектный код для набора компьютерных подпрограмм, называемых *функциями*, которые решают такие задачи, как отображение информации на экране монитора, нахождение квадратного корня числа и т.д. В процессе связывания ваш объектный код комбинируется с объектным кодом для используемых функций и с некоторым стандартным кодом запуска для формирования версии времени выполнения вашей программы. Файл, содержащий этот финальный продукт, называется *исполняемым кодом*.

Термин *исходный код* вы будете постоянно встречать на страницах этой книги, поэтому постарайтесь запомнить его.

Большинство программ в этой книге являются обобщенными и должны выполняться в любой системе, которая поддерживает C++98. Однако некоторые программы, в частности, рассматриваемые в главе 18, требуют определенной поддержки C++11. На момент написания этой книги некоторые компиляторы требовали указания дополнительных флагов для активизации их частичной поддержки C++11. Например, компилятор `g++`, начиная с версии 4.3, в настоящее время при компиляции файла исходного кода использует флаг `-std=c++11`:

```
g++ -std=c++11 use_auto.cpp
```

Этапы сборки программы в одно целое могут варьироваться. Давайте рассмотрим их более подробно.

Создание файла исходного кода

В оставшейся части этой книги мы будем говорить обо всем, что связано с файлом исходного кода; в этом разделе речь пойдет о механизме создания этого файла. В некоторых реализациях языка C++, таких как Microsoft Visual C++, Embarcadero C++ Builder, Apple Xcode, Open Watcom C++, Digital Mars C++ и Freescale CodeWarrior, предлагается так называемая *интегрированная среда разработки* (Integrated Development Environment — IDE), которая позволяет управлять всеми этапами создания программы, включая редактирование, из одной главной программы. В других реализациях C++, таких как GNU C++ на платформах Unix и Linux, IBM XL C/C++ на платформе AIX, а также в свободно распространяемых версиях компиляторов Borland 5.5 (распространяемого Embarcadero) и Digital Mars, поддерживаются только этапы компиляции и компоновки, и все команды должны вводиться в командной строке. В этих случаях для создания и изменения исходного кода можно использовать любой доступный текстовый редактор. В системе Unix можно применять редакторы `vi`, `ed` или `emacs`. В режиме командной строки системы Windows можно работать в `edlin` либо `edit` или любом другом доступном текстовом редакторе. Можно даже использовать текстовый процессор при условии, что искомый файл будет сохраняться в формате текстового файла ASCII, а не в специальном формате текстового процессора. В качестве альтернативы может быть доступны IDE-среды для работы с упомянутыми компиляторами командной строки.

При назначении имени файлу исходного кода должен использоваться подходящий суффикс, с помощью которого файл можно идентифицировать как файл кода C++. Благодаря этому суффиксу не только вы, но и компилятор будет знать, что данный файл содержит исходный код C++. (Если Unix-компилятор пожалуется на неверное магическое число (“bad magic number”), следует иметь в виду, что это его излюбленный способ указывать на неверный суффикс.) Суффикс начинается с точки, за которой следует символ или группа символов, называемых *расширением* (рис. 1.4).

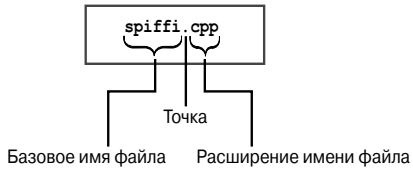

Рис. 1.4. *Элементы имени файла исходного кода*

Выбор расширения будет зависеть от реализации C++. В табл. 1.1 перечислены некоторые распространенные варианты. Например, `spiffy.C` является допустимым именем Unix-файла исходного кода. Обратите внимание, что в Unix важно соблюдать регистр символов: символ `C` должен быть записан в верхнем регистре. Следует отметить, что расширения, записанные в нижнем регистре, тоже допускаются, однако в стандартной версии языка C используется именно такой формат расширения. Поэтому во избежание путаницы в Unix-системах следует применять `c` для программ на языке C и `C` — для программ на языке C++. Можно также использовать один или два дополнительных символа: в некоторых Unix-системах, например, можно использовать расширения `cc` и `cxx`. В DOS, более простой системе по сравнению с Unix, нет разницы в том, в каком регистре будет введен символ, поэтому для различия программ на C и C++ в DOS-реализациях применяются дополнительные символы (см. табл. 1.1).

Таблица 1.1. Расширения файла исходного кода

Реализация C++	Расширения файла исходного кода
Unix	`C`, `cc`, `cxx`, `c`
GNU C++	`C`, `cc`, `cxx`, `cpp`, `c++`
Digital Mars	`cpp`, `cxx`
Borland C++	`cpp`
Watcom	`cpp`
Microsoft Visual C++	`cpp`, `cxx`, `cc`
Freestyle CodeWarrior	`cpp`, `cp`, `cc`, `cxx`, `c++`

Компиляция и компоновка

Первоначально Страуструп реализовал C++ с программой компилятора из C++ в C, и не стал разрабатывать прямой компилятор из C++ в объектный код. Эта программа, называемая `cfront` (сокращение от *C front end*), транслировала исходный код C++ в исходный код C, который затем можно было компилировать с помощью стандартного компилятора C. Этот подход способствовал росту популярности C++ в среде программистов на C. В других реализациях этот подход использовался для переноса C++ на другие платформы. По мере совершенствования языка C++ и роста его популярности, все большее число разработчиков предпочитали создавать компиляторы C++, которые генерировали объектный код непосредственно из исходного кода C++. Такой прямой подход ускорял процесс компиляции и обособлял язык C++.

Способ компиляции зависит от реализации, и в последующих разделах мы рассмотрим некоторые варианты. В этих разделах будут описаны только основные этапы, однако это вовсе не означает, что вам не следует изучать документацию по своей системе.

Компиляция и связывание в Unix

Первоначально команда CC в Unix вызывала cfront. Однако cfront не удалось двигаться наравне с эволюцией C++, и его последний выпуск датируется 1993 г. В наши дни компьютеры на платформе Unix могут вообще не иметь компилятора, а могут иметь собственный компилятор или сторонний компилятор, будь-то коммерческий или свободно распространяемый вроде GNU g++. Во многих этих случаях (но не тогда, когда компилятора вообще нет) команда CC будет работать, вызывая компилятор, используемый в конкретной системе. Для простоты мы предполагаем, что команда CC доступна, но обратите внимание, что при последующем обсуждении она может быть заменена другой командой.

Команда CC используется для компиляции вашей программы. Имя команды вводится в верхнем регистре для того, чтобы отличить его от обычного компилятора C в Unix — cc. Компилятор CC является компилятором командной строки, что означает ввод команд компиляции в командной строке Unix.

Например, чтобы скомпилировать файл исходного кода C++ по имени spiffy.C, в строке приглашения Unix необходимо ввести следующую команду:

```
CC spiffy.C
```

Если, благодаря опыту или удаче в вашей программе не окажется ошибок, компилятор сгенерирует файл объектного кода с расширением o. В нашем случае компилятор создаст файл с именем

```
spiffy.o
```

Затем компилятор автоматически передает файл объектного кода системному редактору связей — программе, которая комбинирует ваш код с кодом библиотеки для построения исполняемого файла. По умолчанию исполняемому файлу назначается имя a.out. Если вы используете только один файл исходного кода, тогда редактор связей удалит файл spiffy.o, поскольку в нем больше нет необходимости. Чтобы запустить программу, достаточно ввести имя исполняемого файла:

```
a.out
```

Обратите внимание, что если вы компилируете новую программу, то новый исполняемый файл a.out заменит предыдущий файл a.out, который мог существовать. (Это происходит потому, что исполняемые файлы занимают достаточно много места, и замена старых исполняемых файлов позволяет сократить объем занимаемого дискового пространства.) Но если вы разрабатываете исполняемую программу, которую необходимо сохранить на будущее, воспользуйтесь Unix-командой mv, чтобы изменить имя исполняемого файла.

В языке C++, как и в C, одна программа может состоять из нескольких файлов. (Таковыми являются многие программы, рассматриваемые в главах 8–16 этой книги.) В этом случае компилировать программу можно, перечислив все эти файлы в командной строке:

```
CC my.C precious.C
```

При наличии нескольких файлов исходного кода компилятор не будет удалять файлы объектного кода. Так, если вы просто измените файл my.C, то повторную компиляцию программы можно будет выполнить с помощью следующей команды:

```
CC my.C precious.o
```

В результате файл my.C будет повторно скомпилирован и связан с ранее скомпилированным файлом precious.o.

Иногда возникает необходимость явным образом указывать некоторые библиотеки. Например, чтобы обратиться к функциям, определенным в библиотеке математических операций, может понадобиться добавить флаг -lm в командной строке:

```
CC usingmath.C -lm
```

Компиляция и связывание в Linux

В системах Linux чаще всего используется g++ — компилятор GNU C++, разработанный Фондом свободного программного обеспечения (Free Software Foundation). Этот компилятор входит в состав большинства дистрибутивов Linux, однако может устанавливаться и отдельно. Компилятор g++ работает подобно стандартному компилятору Unix. Например, в результате выполнения команды

```
g++ spiffy.cxx
```

будет создан исполняемый файл с именем a.out.

В некоторых версиях компилятора необходимо связываться с библиотекой C++:

```
g++ spiffy.cxx -lg++
```

Чтобы скомпилировать несколько файлов исходного кода, их достаточно перечислить в командной строке:

```
g++ my.cxx precious.cxx
```

В результате будет создан исполняемый файл по имени a.out и два файла объектного кода, my.o и precious.o. Если впоследствии вы измените только один файл исходного кода, например, my.cxx, то повторную компиляцию можно будет выполнить, используя my.cxx и precious.o:

```
g++ my.cxx precious.o
```

Компилятор GNU может работать на различных платформах, в том числе в режиме командной строки на ПК под управлением Windows и на различных платформах Unix-систем.

Компиляторы командной строки для режима командной строки Windows

Наименее затратный вариант компиляции программ на C++ в системах Windows заключается в том, чтобы загрузить свободно распространяемый компилятор командной строки, работающий в режиме командной строки Windows, при котором открывается MS-DOS-подобное окно. Бесплатно загружаемыми программами для Windows, которые включают компилятор GNU C++, являются Cygwin и MinGW; именем используемого в них компилятора является g++.

Для запуска компилятора g++ сначала потребуется открыть окно командной строки. Программы Cygwin и MinGW делают это автоматически при запуске. Чтобы скомпилировать файл исходного кода по имени great.cpp, в командной строке введите следующую команду:

```
g++ great.cpp
```

В случае успешной компиляции будет сформирован исполняемый файл по имени a.exe.

Компиляторы для Windows

Компиляторов для Windows так много, при этом их модификации появляются настолько часто, что нет смысла рассматривать их все по отдельности. В настоящее время самым популярным является Microsoft Visual C++ 2010, который доступен также в виде бесплатной версии Microsoft Visual C++ 2010 Express. В Wikipedia (`http://en.wikipedia.org/wiki/List_of_compilers`) представлен исчерпывающий список компиляторов для множества платформ, включая Windows. Несмотря на разные проектные решения и цели, большинство компиляторов C++ для Windows обладают общими характеристиками.

Как правило, для программы требуется создать проект и добавить в него один или несколько файлов, составляющих программу. Каждый производитель предлагает IDE-среду с набором меню и, возможно, с программой автоматизированного помощника, которую удобно использовать в процессе создания проекта. Очень важно определиться с тем, к какому типу будет относиться создаваемая вами программа. Обычно компилятор предлагает несколько вариантов, среди которых приложение для Windows, приложение Windows MFC, динамически подключаемая библиотека, элемент управления ActiveX, программа, выполняемая в режиме DOS или в символьном режиме, статическая библиотека или консольное приложение. Некоторые из них могут быть доступны в форме 64- и 32-разрядных версий.

Поскольку программы в этой книге являются обобщенными, вы должны избегать вариантов, требующих кода для определенной платформы, например, приложение для Windows. Вместо этого нужно запускать программу в символьном режиме. Выбор режима зависит от компилятора. В общем случае необходимо искать такие варианты, как консольное, символьное или DOS-приложение, и пробовать их. Например, в среде Microsoft Visual C++ 2010 выберите опцию Win32 Console Application (Консольное приложение Win32), щелкните на Application Settings (Настройки приложения) и выберите вариант Empty Project (Пустой проект). В C++Builder XE выберите вариант Console Application (Консольное приложение) в разделе C++Builder Projects (Проекты C++Builder).

После того как проект настроен, вы должны скомпилировать и скомпоновать свою программу. В IDE-среде обычно предлагается несколько вариантов, такие как Compile (Компилировать), Build (Построить), Make (Построить), Build All (Построить все), Link (Скомпоновать), Execute (Выполнить), Run (Выполнить) и Debug (Отладить) (но не обязательно все варианты сразу).

- Compile обычно означает компиляцию кода в открытом в настоящий момент файле.
- Build или Make обычно означает компиляцию кода для всех файлов исходного кода, входящих в состав данного проекта. Часто этот процесс является инкрементным. То есть, если в проекте было три файла, и вы изменили только один из них, то повторно скомпилирован будет только этот файл.
- Build All обычно означает компиляцию всех файлов исходного кода с самого начала.
- Как уже было сказано ранее, Link означает объединение скомпилированного исходного кода с необходимым библиотечным кодом.
- Run или Execute означает запуск программы. Обычно если вы еще не завершили выполнение предыдущих этапов, команда Run выполнит их перед запуском программы.

- Debug означает запуск программы с возможностью ее пошагового выполнения.
- Компилятор может поддерживать создание версий Debug (Отладочная) и Release (Выпуск). Версия Debug содержит дополнительный код, который увеличивает размер программы и замедляет ее выполнение, но зато делает доступными средства отладки.

Если вы нарушите какое-то правило языка, компилятор выдаст сообщение об ошибке и укажет на строку кода, в которой она была найдена. К сожалению, если вы еще недостаточно хорошо знаете язык программирования, то понять смысл этого сообщения порой бывает трудно. В некоторых случаях действительная ошибка может находиться перед указанной строкой, а бывает так, что единственная ошибка порождает цепочку сообщений об ошибках.

Совет

Исправление ошибок начинайте с самой первой. Если вы не можете ее найти в указанной строке, проверьте предыдущую строку кода.

Имейте в виду, что факт принятия программы определенным компилятором вовсе не означает, что эта программа является допустимой программой на C++. И то, что определенный компилятор отклоняет программу, не обязательно означает, что эта программа не является допустимой программой C++. Однако современная компиляторы в большей степени соответствуют принятому стандарту, нежели их предшественники несколько лет тому назад. Кроме того, компиляторы, как правило, имеют опции для управления строгостью компиляции.

Совет

Время от времени компиляторы, не завершив процесс создания программы, генерируют бессмысленные сообщения об ошибках, которые невозможно устранить. В подобных ситуациях следует воспользоваться командой Build All, чтобы начать процесс компиляции с самого начала. К сожалению, отличить эту ситуацию от другой, более распространенной, когда сообщение об ошибке только кажется бессмысленным, достаточно трудно.

Обычно IDE-среда позволяет запускать программу во вспомогательном окне. В некоторых IDE-средах это окно закрывается после завершения выполнения программы, а в некоторых оно остается открытым. Если ваш компилятор закрывает окно, вы не успеете увидеть вывод программы, если только не умеете очень быстро читать и не обладаете фотографической памятью. Чтобы увидеть результат выполнения, в конце программы необходимо ввести следующий код:

```
    cin.get(); // добавьте этот оператор
    cin.get(); // и, возможно, этот тоже
    return 0;
}
```

Оператор `cin.get()` читает следующее нажатие клавиши, поэтому программа будет находиться в режиме ожидания до тех пор, пока вы не нажмете клавишу <Enter>. (Вплоть до нажатия <Enter> никакие нажатия клавиш программе не отправляются, поэтому нажимать другие клавиши не имеет смысла.) Второй оператор необходим на случай, если программа оставит необработанным нажатие клавиши после предыдущего ввода информации в программе. Например, при вводе числа вы нажимаете соответствующую клавишу, а затем <Enter>. Программа может прочитать число, но

оставить необработанным нажатие клавиши <Enter>, и затем читает его в первом операторе `cin.get()`.

C++ в Macintosh

Компания Apple в настоящее время поставляет платформу разработки под названием Xcode вместе с операционной системой Mac OS X. Эта платформа является бесплатной, но обычно предварительно не устанавливается. Ее можно установить с дистрибутивных дисков ОС или загрузить за номинальную плату из Apple. (Имейте в виду, что объем загрузки превышает 4 Гбайт.) Платформа Xcode не только предоставляет IDE-среду, поддерживающую множество языков программирования, она также устанавливает пару компиляторов — `g++` и `clang`, — которые могут использоваться как программы командной строки в режиме Unix, доступном через утилиту Terminal.

Совет

Для IDE-сред: чтобы сэкономить время, можно использовать только один проект для всех примеров программ. Просто удалите файл с предыдущим примером исходного кода из списка проекта и затем добавьте новый исходный код. Это сохраняет время, усилия и дисковое пространство.

Резюме

Компьютеры стали более мощными, а компьютерные программы — большими и сложными. Как результат, компьютерные языки стали более развитыми и теперь они упрощают управление процессом написания программ. Язык C наделен такими средствами, как управляющие структуры и функции, с помощью которых можно расширить возможности управления ходом выполнения программы и реализовать более структурированный, модульный подход к написанию программ. Для данных инструментов в C++ поддерживается объектно-ориентированное программирование (ООП), а также обобщенное программирование. Благодаря этому открываются возможности для еще более высокой степени модульности и повторного использования кода, что позволяет сократить время на разработку и повысить надежность создаваемых программ.

Популярность языка программирования C++ привела к появлению большого количества реализаций для множества компьютерных платформ; стандарты C++ ISO (C++98/03 и C++11) обеспечивают основу для взаимной совместимости этих реализаций. В стандартах указано, какими возможностями должен обладать язык, как он себя должен вести, какими должны быть библиотеки функций, классы и шаблоны. Стандарты поддерживают идею переносимого языка, программы на котором могут работать на множестве различных вычислительных платформ и в различных реализациях языка.

Чтобы создать программу на языке C++, вы создаете один или несколько файлов исходного кода, написанного на C++. Это текстовые файлы, которые необходимо компилировать и компоновать для формирования файлов на машинном языке, содержащих исполняемые программы. Эти задачи часто выполняются в IDE-среде, которая предлагает текстовой редактор для подготовки файлов исходного кода, компилятор и компоновщик для построения исполняемых файлов, а также другие ресурсы наподобие средств управления проектом и отладкой. Однако те же самые задачи могут быть решены также и в командной строке, в которой соответствующие инструменты вызываются по отдельности.