Часть I. Основы языка C++: подмножество C

Вероятно, читателям уже известно, что язык C++ создан на основе языка C. Фактически язык C++ включает в себя весь язык C, и все программы (за некоторым исключением), написанные на языке C, можно считать программами на языке C++. В процессе разработки языка C++ в качестве отправной точки был выбран язык C. Затем к нему были добавлены новые свойства и возможности, разработанные для поддержки объектно-ориентированного программирования. Однако от языка C при этом не отказались, и стандарт 1989 года ANSI/ISO C Standard стал *базовым документом* при создании международного стандарта языка C++ (International Standard). Таким образом, осваивая язык C++, программисты одновременно овладевают языком C.

Разделение свойств языка C и специфических особенностей языка C++ позволяет достичь трех основных целей.

- Четко провести разделительную линию между языками C и C++.
- Предоставить читателям, владеющим языком C, возможность легко усвоить информацию об особенностях языка C++.
- Выделить и подробно описать свойства языка C++, унаследованные от языка C.

Очень важно точно провести линию, отделяющую язык C от языка C++, поскольку они очень широко распространены, и от программиста иногда требуется поддержка программ, написанных на обоих языках. Если вы пишете программу на языке C, нужно четко понимать, где заканчивается язык C и начинается язык C++. Многие программисты, работающие на языке C++, иногда пишут программы на языке C и не используют специфические свойства языка C++. В особенности это относится к программированию встроенных систем и поддержке существующих приложений. Таким образом, понимание различий между этими языками является необходимым условием профессионального программирования на C++.

Полное и свободное владение языком C абсолютно необходимо для перевода программ на язык C++. Чтобы сделать это на высоком уровне, необходимо хорошо знать язык C. Например, без ясного понимания механизмов ввода-вывода, предусмотренных в языке C, невозможно трансформировать программу, осуществляющую интенсивный обмен данными с внешними устройствами, в эффективную программу на языке C++.

Многие читатели уже владеют языком C. Вследствие этого выделение тем, связанных с языком C, позволяет опытным программистам избежать повторения пройденного и перейти непосредственно к изучению особенностей языка C++. Разумеется, в части I подчеркиваются малейшие отличия языка C++ от языка C. Кроме того, отделение подмножества C от остальных свойств языка C++ позволяет в дальнейшем сосредоточиться на его объектно-ориентированных особенностях.

Несмотря на то что язык C++ полностью содержит язык C, как правило, не все свойства языка C используются в программах, написанных “в стиле C++”. Например, система ввода-вывода, предусмотренная в языке C, по-прежнему доступна в языке C++, хотя в C++ существуют свои объектно-ориентированные механизмы ввода-вывода данных. Еще одним примером такого анахронизма является препроцессор. Он играет чрезвычайно важную роль в языке C и очень скромную — в языке C++. Обсуждение свойств, присущих “стилю языка C”, в первой части книги позволяет избежать путаницы в остальных главах.

Запомните: подмножество C, описанное в части I, является ядром языка C++ и фундаментом, на котором воздвигнуты его объектно-ориентированные конструкции. Все свойства, описанные здесь, являются неотъемлемой частью языка C++ и могут быть использованы в ваших программах.

На заметку

Часть I содержит материалы из моей книги C: The Complete Reference *(McGrawHill-Osborne) (Русский перевод: Г. Шилдт. Полный справочник по языку C. — М.: Изд. дом “Вильямс”, 2001. — Прим. ред.). Если язык C интересует вас как самостоятельный язык программирования, эта книга окажется для вас ценным помощником.*

Глава 1

Обзор языка C

Чтобы понять язык C++, необходимо понять мотивы его создания, идеи, положенные в его основу, и свойства, которые он унаследовал от своих предшественников. Таким образом, история языка C++ начинается с языка C. В главе представлен обзор языка C, а также описана история его возникновения, способы применения и основные принципы. Поскольку язык C++ создан на основе языка C, эту главу можно считать описанием предыстории языка C++. Многое из того, что сделало язык C++ таким популярным, уходит корнями в язык C.

Происхождение и история языка C

Язык C был изобретен и впервые реализован Деннисом Ритчи (Dennis Ritchie) на компьютере DEC PDP−11 под управлением операционной системы UNIX. Язык C появился в результате развития языка под названием BCPL. В свою очередь, этот язык был разработан Мартином Ричардсом (Martin Richards) под влиянием другого языка, имевшего название B, автором которого был Кен Томпсон (Ken Tompson). Итак, в 1970-х годах развитие языка B привело к появлению языка C.

Многие годы фактическим стандартом языка C была версия для операционной системы UNIX. Впервые она была описана в книге Брайана Кернигана (Brian Kernigan) и Денниса Ритчи *The C Programming Language* в 1978 году. (Русский перевод: Керниган Б., Ритчи Д. Язык программирования C. — СПб: Невский диалект, 2001. — *Прим. ред.*). Летом 1983 года был создан комитет Американского института национальных стандартов (American National Standards Institute — ANSI), целью которого была разработка стандарта языка C. Работа комитета неожиданно растянулась на шесть лет.

В итоге стандарт ANSI C был одобрен в декабре 1989 года и стал распространяться в начале 1990-го. Этот стандарт был также одобрен Организацией международных стандартов (International Standards Organization — ISO), получив название *ANSI/ISO Standard C*. В 1995 году была одобрена Первая поправка, которая помимо всего прочего добавила несколько новых библиотечных функций. В 1989 году стандарт языка C вместе с Первой поправкой стали *базовым документом* для стандарта языка C++, в котором было выделено *подмножество C*. Версию языка C, определенную стандартом 1989 года, обычно называют *C89*.

После 1989 года в центре внимания программистов оказался язык C++. Развитие этого языка на протяжении 1990-х годов завершилось одобрением стандарта в конце 1998 года. Между тем работа над языком C продолжалась, не вызывая излишнего шума. В итоге в 1999 году появился новый стандарт языка C, который обычно называют *C99*. В целом стандарт C99 сохранил практически все свойства стандарта C89, не изменив основных аспектов языка. Таким образом, язык C, описанный стандартом C99, практически совпадает с языком, соответствующим стандарту C89. Комитет по разработке стандарта C99 сосредоточился на двух вопросах: включении в язык нескольких математических библиотек и развитии некоторых специфических и весьма сложных свойств, например, массивов переменной длины и квалификатора указателей **`restrict`**. В стандарт C99 вошли также некоторые свойства, позаимствованные из языка C++, например, однострочные комментарии. Поскольку разработка стандарта языка C++ завершилась до создания стандарта C99, ни одно из новшеств языка C не вошло в стандарт C++.

Сравнение стандартов C89 и C99

Несмотря на то что все новшества, внесенные в стандарт C99, весьма важны с теоретической точки зрения, они имели мало практических последствий, так как до сих пор нет ни одного широко распространенного компилятора, который

поддерживал бы стандарт C99. Большинство программистов до сих пор считают языком C его вариант, определенный стандартом C89. Именно его реализуют все основные компиляторы. Более того, подмножество C языка C++ описывается именно стандартом C89. Хотя некоторые новшества, включенные в стандарт C99, в конце концов обязательно будут учтены следующим стандартом языка C++, пока они несовместимы с языком C++.

Поскольку подмножество C языка C++ соответствует стандарту C89, и эту версию изучают большинство программистов, именно ее мы рассмотрим в части I. Итак, используя название C, мы будем иметь в виду версию языка, определенную стандартом C89. Однако мы будем отмечать важные различия между версиями C89 и C99, поскольку это улучшит совместимость языков C и C++.

C — язык среднего уровня

Язык C часто называют *языком среднего уровня*. Это не означает, что он менее эффективен, более неудобен в использовании или менее продуман, чем языки высокого уровня, такие как Basic или Pascal. Отсюда также не следует, что он запутан, как язык ассемблера (и порождает связанные с этим проблемы). Это выражение означает лишь, что язык C объединяет лучшие свойства языков высокого уровня, возможности управления и гибкость языка ассемблера. В табл. 1.1 показано место языка C среди других языков программирования.

Таблица 1.1. Место языка C среди остальных языков программирования

Высший уровень	Ada
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
Средний уровень	Java
	C#
	C++
	C
	Forth
Низший уровень	Macro-assembler
	Assembler

Будучи языком среднего уровня, язык C позволяет осуществлять манипуляции с битами, байтами и адресами — основными элементами, с которыми работают функции операционной системы. Несмотря на это, программы, написанные на языке C, можно выполнять на разных компьютерах. Это свойство программ называется *машинонезависимостью* (portability). Например, если программу, написанную для операционной системы UNIX, можно легко преобразовать, чтобы она работала на платформе Windows, то говорят, что такая программа является *машинонезависимой* (portable).

Все языки высокого уровня используют концепцию типов данных. *Тип данных* (data type) определяет множество значений, которые может принимать переменная, а также множество операций, которые над ней можно выполнять. К основным типам данных относятся целое число, символ и действительное число. Несмотря на то что в языке C существует пять встроенных типов данных, он не является строго типизи-

рованным языком, как языки Pascal и Ada. В языке C разрешены практически все преобразования типов. Например, в одном и том же выражении можно свободно использовать переменные символьного и целочисленного типов.

В отличие от языков высокого уровня, язык C практически не проверяет ошибки, возникающие на этапе выполнения программ. Например, не осуществляется проверка возможного выхода индекса массива за пределы допустимого диапазона. Предотвращение ошибок такого рода возлагается на программиста.

Кроме того, язык C не требует строгой совместимости типов параметров и аргументов функций. Как известно, языки высокого уровня обычно требуют, чтобы тип аргумента точно совпадал с типом соответствующего параметра. Однако в языке C такого условия нет. Он позволяет использовать аргумент любого типа, если его можно разумным образом преобразовать в тип параметра. Кроме того, язык C предусматривает средства для автоматического преобразования типов.

Особенность языка C заключается в том, что он позволяет непосредственно манипулировать битами, байтами, машинными словами и указателями. Это делает его очень удобным для системного программирования, в котором эти операции широко распространены.

Другой важный аспект языка состоит в том, что в нем предусмотрено очень небольшое количество ключевых слов, которые можно использовать для конструирования выражений. Например, стандарт C89 содержит лишь 32 ключевых слова, а стандарт C99 добавил к ним всего 5 слов. Некоторые языки программирования содержат в несколько раз больше ключевых слов. Скажем, самые распространенные версии языка BASIC предусматривают более 100 таких слов!

C — структурированный язык

Возможно, вы уже слышали словосочетание *блочно-структурированный* (block-structured) по отношению к языку программирования. Хотя этот термин нельзя напрямую применять к языку C, его обычно тоже называют *структурированным*. Он имеет много общего с другими структурированными языками, такими как ALGOL, Pascal и Modula-2.

На заметку *Язык C (как и C++) не считается блочно-структурированным, поскольку не позволяет объявлять одну функции внутри других.*

Отличительной особенностью структурированных языков является *обособление* кода и данных (compartmentalization). Оно позволяет выделять и скрывать от остальной части программы данные и инструкции, необходимые для решения конкретной задачи. Этого можно достичь с помощью подпрограмм (subroutines), в которых используются локальные (временные) переменные. Используя локальные переменные, можно создавать подпрограммы, не порождающие побочных эффектов в других модулях. Это облегчает координацию модулей между собой. Если программа разделена на обособленные функции, нужно лишь знать, что делает та или иная функция, не интересуясь, как именно она выполняет свою задачу. Помните, что чрезмерное использование глобальных переменных (которые доступны в любом месте программы) повышает вероятность ошибок и нежелательных побочных эффектов. (Каждый программист, работавший на языке BASIC, хорошо знает эту проблему.)

На заметку *Концепция обособления широко применяется в языке C++. В частности, каждая часть программы, написанной на этом языке, может четко управлять доступом к ней из других модулей.*

Структурные языки предоставляют широкий спектр возможностей. Они допускают использование вложенных циклов, например **`while`**, **`do-while`** и **`for`**.

В структурированных языках использование оператора **goto** либо запрещено, либо нежелательно и не включается в набор основных средств управления потоком выполнения программы (как это принято в стандарте языка BASIC и в традиционном языке FORTRAN). Структурированные языки позволяют размещать несколько инструкций программы в одной строке и не ограничивают программиста жесткими полями для ввода команд (как это делалось в старых версиях языка FORTRAN).

Рассмотрим несколько примеров структурированных и неструктурированных языков (табл. 1.2).

Таблица 1.2. Структурированные и неструктурированные языки программирования

Неструктурированные	*Структурированные*
FORTRAN	Pascal
BASIC	Ada
COBOL	Java
	C#
	C++
	C
	Modula-2

Структурированные языки считаются более современными. В настоящее время неструктурированность является признаком устаревших языков программирования, и лишь немногие программисты выбирают их для создания серьезных приложений.

На заметку

Новые версии старых языков программирования (например Visual Basic) включают элементы структурированности. И все же врожденные недостатки этих языков вряд ли будут до конца исправлены, поскольку структурированность не закладывалась в их основу с самого начала.

Основным структурным элементом языка C является *функция*. Именно функции служат строительными блоками, из которых создается программа. Они позволяют разбивать программу на модули, решающие отдельные задачи. Создав функцию, можно не беспокоиться о побочных эффектах, которые она вызовет в других частях программы. Способность создавать отдельные функции чрезвычайна важна при реализации больших проектов, в которых один фрагмент кода не должен взаимодействовать с другими частями программы непредсказуемым образом.

Другой способ структурирования и обособления программы, написанной на языке C, — блоки. *Блок* (code block) — это группа операторов, логически связанных между собой, и рассматриваемых как единое целое. В языке C блок можно создать с помощью фигурных скобок, ограничивающих последовательность операторов. Вот типичный пример блока.

```
if (x < 10) {
  printf("Слишком мало, попробуйте снова.\n";
  scanf("%d", &x);
}
```

Два оператора, расположенных внутри фигурных скобок, выполняются, если значение переменной **x** меньше 10. Эти два оператора вместе с фигурными скобками образуют блок. Блок — это логическая единица, поскольку оба оператора обязательно должны быть выполнены. Блоки позволяют ясно, элегантно и эффективно реализовывать различные алгоритмы. Более того, они помогают программисту лучше выразить природу алгоритма.

C — язык для программистов

Как ни странно, не все языки программирования предназначены для программистов. Рассмотрим классические примеры языков, ориентированных не на программистов, — COBOL и BASIC. Язык COBOL был разработан вовсе не для того, чтобы облегчить участь программистов, улучшить ясность кода и повысить его надежность, и даже не для того, чтобы ускорить выполнение программ. Он был создан, в частности, для того, чтобы люди, не являющиеся программистами, могли читать и (по возможности) понимать (хотя это вряд ли) написанные на нем программы. В свою очередь, язык BASIC был разработан для пользователей, которые решают на компьютере простые задачки.

В противоположность этому, язык C был создан для программистов, учитывал их интересы и многократно проверялся на практике, прежде чем был окончательно реализован. В итоге этот язык дает программистам именно то, чего они желали: сравнительно небольшое количество ограничений, минимум претензий, блочные структуры, изолированные функции и компактный набор ключевых слов. Язык C обладает эффективностью ассемблера и структурированностью языков ALGOL или Modula-2. Поэтому неудивительно, что именно языки C и C++ стали наиболее популярными среди профессиональных программистов высокого уровня.

Тот факт, что язык C часто используют вместо ассемблера, является одной из основных причин его популярности. Язык ассемблера использует символьное представление фактического двоичного кода, который непосредственно выполняется компьютером. Каждая операция, выраженная на языке ассемблера, представляет собой отдельную задачу, выполняемую компьютером. Хотя язык ассемблера предоставляет программисту наибольшую гибкость, разрабатывать и отлаживать программы на нем довольно сложно. Кроме того, поскольку язык ассемблера является неструктурированным, код напоминает спагетти — запутанную смесь переходов, вызовов и индексов. Вследствие этого программы, написанные на языке ассемблера, трудно читать, модифицировать и эксплуатировать. Вероятно, основным недостатком программ на языке ассемблера является их машинозависимость. Программа, предназначенная для конкретного центрального процессора, не может выполняться на компьютерах другого типа.

Изначально язык C предназначался для системного программирования. *Системная программа* (system program) представляет собой часть операционной системы или является одной из ее утилит. Рассмотрим некоторые из них.

- Операционные системы
- Интерпретаторы
- Редакторы
- Компиляторы
- Файловые утилиты
- Оптимизаторы
- Диспетчеры реального времени
- Драйверы

По мере роста популярности языка C многие программисты стали применять его для программирования всех задач, используя его машинонезависимость и эффективность, а кроме того, он им просто нравился! К моменту появления языка C языки

программирования прошли сложный и трудный путь совершенствования. Разумеется, вновь созданный язык вобрал в себя все лучшее.

С появлением языка C++ некоторые программисты посчитали, что язык C потеряет самостоятельность и сойдет со сцены. Однако этого не произошло. Во-первых, не все программы должны быть объектно-ориентированными. Например, программное обеспечение встроенных систем по-прежнему создается на языке C. Во-вторых, существует огромное множество программ на языке C, которые активно эксплуатируются и нуждаются в модификации. Поскольку язык C является основной языка C++, он продолжает широко использоваться, имея блестящие перспективы.

Структура программы на языке C

В табл. 1.3 перечислены 32 ключевых слова, которые используются при формировании синтаксиса языка C, стандарта C89 и подмножества C языка C++. Все они, конечно, являются и ключевыми словами языка C++.

Таблица 1.3. Ключевые слова подмножества C языка C++

`auto`	`double`	`int`	`struct`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

Кроме того, многие компиляторы для более эффективного использования среды программирования вносят в язык C дополнительные ключевые слова. Например, некоторые компиляторы предусматривают ключевые слова для управления памятью процессоров семейства 8086, поддержки многоязычного программирования и доступа к системным прерываниям. Перечислим некоторые из этих расширенных ключевых слов.

```
asm         _cs       _ds       _es
_ss         cdecl     far       huge
interrupt   near      pascal
```

Ваш компилятор может изменить этот список, стремясь наиболее эффективно использовать конкретную среду программирования.

Обратите внимание на то, что все ключевые слова набраны строчными буквами. Язык C/C++ *чувствителен к регистру* (case sensitive), т.е. прописные и строчные буквы в нем различаются. Это значит, что слово **`else`** является ключевым, а слово **`ELSE`** — нет. Ключевые слова нельзя использовать в программе для иных целей, например, в качестве имени переменной или функции.

Все программы на языке C состоят из одной или нескольких функций. В любом случае программа должна содержать функцию **`main()`**, которая при выполнении программы вызывается первой. В хорошо написанном коде функция **`main()`** должна содержать, по существу, схему работы всей программы. Несмотря на то что имя **`main()`** не включено в список ключевых слов, по своей природе оно является именно таковым. Скажем, назвать переменную именем **`main`** нельзя, так как компилятор сразу выдаст сообщение об ошибке.

Общий вид программы на языке C показан в листинге 1.1. Функции с именами **f1()**, ..., **fN()** определяются пользователем.

Листинг 1.1. Общий вид программы на языке C

```
объявления глобальных переменных
тип_возвращаемого_значения main (список параметров)
{
   последовательность операторов
}

тип_возвращаемого_значения f1 (список параметров)
{
   последовательность операторов
}

тип_возвращаемого_значения f2 (список параметров)
{
   последовательность операторов
}

.
.
.

тип_возвращаемого_значения fN (список параметров)
{
   последовательность операторов
}
```

Библиотека и связывание

С формальной точки зрения можно написать законченную и вполне осмысленную программу на языке C, не используя ни одной стандартной функции. Однако это довольно затруднительно, так как в языке C нет ключевых слов, имеющих отношение к вводу-выводу, математическим операциям высокого уровня или обработке символов. В результате большинство программ вынуждены вызывать различные функции, содержащиеся в *стандартной библиотеке* (standard library).

Все компиляторы языка C++ сопровождаются стандартной библиотекой функций, выполняющих наиболее распространенные задачи. В этой библиотеке определен минимальный набор функций, которые поддерживаются большинством компиляторов. Однако конкретный компилятор может содержать много других функций. Например, стандартная библиотека не содержит графических функций, но в каждом компиляторе всегда есть определенный набор таких функций.

Стандартная библиотека языка C++ разделена на две части: функции и классы. Стандартная библиотека функций представляет собой наследие языка C. Язык C++ поддерживает все функции, предусмотренные стандартом C89. Таким образом, все стандартные функции языка C можно свободно использовать в программах на языке C++.

Кроме стандартной библиотеки функций, язык C++ имеет собственную библиотеку классов. Эта библиотека состоит из объектно-ориентированных модулей, которые можно использовать в собственных программах. Кроме того, существует стандартная

библиотека шаблонов STL, содержащая широкий набор готовых решений многих задач. В части I используется только стандартная библиотека функций, поскольку именно она относится к языку C.

Стандартная библиотека состоит из многих универсальных функций. При вызове библиотечной функции компилятор "запоминает" ее имя. Позднее редактор связей объединит ваш код с объектным кодом этой библиотечной функции. Этот процесс называется *редактированием связей* (linking). Некоторые компиляторы имеют свои собственные редакторы связей, остальные используют редактор связей, предусмотренный операционной системой.

Функции в библиотеке имеют *машинонезависимый* формат (relocatable format). Это значит, что адреса памяти для разных машинных инструкций не являются абсолютными — сохраняется лишь информация о смещении их адреса. Фактические адреса ячеек, используемых стандартными функциями, определяются во время редактирования связей. Подробное описание этих процессов можно найти в соответствующих технических руководствах. Более детальные объяснения были бы излишни.

В стандартной библиотеке содержится много функций, которые могли бы оказаться полезными. Они представляют собой крупные строительные блоки, из которых можно сконструировать свою программу. В частности, если какую-то функцию вы используете в своих программах очень часто, ее следует поместить в библиотеку.

Раздельная компиляция

Большинство коротких программ обычно можно уместить в одном файле. Однако по мере возрастания объема программы увеличивается время ее компилирования. Для решения этой проблемы в языке C/C++ предусмотрена возможность делить программу на файлы и компилировать каждый из них отдельно. Скомпилировав все файлы, отредактировав связи между ними и библиотечными функциями, вы получите завершенный объектный код. Преимущество раздельной компиляции заключается в том, что при изменении кода, записанного в одном из файлов, нет необходимости компилировать заново всю программу. Это существенно экономит время на этапе компиляции. Документация, сопровождающая компиляторы языка C/C++, содержит инструкции, которые позволят вам скомпилировать программу, записанную в нескольких файлах.

Расширения файлов .C и .CPP

Разумеется, программы, приведенные в части I, являются вполне корректными программами на языке C++. Их можно компилировать с помощью любого современного компилятора языка C++. Одновременно они являются корректными программами на языке C и могут компилироваться с помощью компиляторов этого языка. Итак, если вы собираетесь писать программы на языке C, можете рассматривать программы из первой части в качестве примера. Традиционно программы на языке C используют расширения файла **.C**, а программы на языке C++ — **.CPP**. Компилятор языка C++ использует расширение файла для определения типа программы. Это имеет большое значение, поскольку компилятор рассматривает любую программу, использующую расширение **.C**, как программу на языке C, а программу, записанную в файл с расширением **.CPP**, — как программу на языке C++. Если обратное не указано явно, вы можете выбрать любое расширение для файла с программой из первой

части книги. Однако программы, приведенные в остальной части книги, должны быть записаны в файлы с расширением **.CPP**.

И последнее: хотя язык C является подмножеством языка C++, между ними существует несколько отличий. В некоторых случаях программу на языке C нужно компилировать *именно как программу на языке C* (используя расширение **.C**). Все такие случаи мы оговариваем отдельно.