Виталий Потопахин

Язык C

ОСВОЙ НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.068+800.92С
ББК 32.973.26-018.1
П64

Потопахин В. В.

П64 Язык С. Освой на примерах. — СПб.: БХВ-Петербург, 2006. — 320 с.: ил.

ISBN 5-94157-950-0

Книга содержит детальное описание языка С, сопровождаемое большим количеством законченных примеров. Рассмотрены указатели и представление структур данных с использованием механизма ссылок. Показано, как с помощью указателей в С создаются строки и такие конструкции данных, как связанные списки и деревья. Рассмотрены работа с файлами, операции ввода-вывода, графические возможности языка и многое другое. В приложении приведены примеры решения задач различной степени сложности.

Для 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	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.07.06.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 20.
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 5-94157-950-0

Оглавление

Введение

Язык C, созданный Денисом Ритчи в начале 70-х годов прошлого века в Bell Laboratory американской корпорации AT&T, считается языком системного программирования и, как говорят, был создан для тех, кто хочет заниматься системным программированием, но не хочет изучать ассемблер. Хотя, безусловно, C удобен и при написании прикладных программ. Первое описание языка дано в книге Б. Кернигана и Д. Ритчи, которая была переведена на русский язык[1]. Долгое время это описание являлось стандартом, однако ряд моментов допускали неоднозначное толкование, которое породило множество трактовок языка. Для исправления этой ситуации при Американском национальном институте стандартов (ANSI) был образован комитет по стандартизации языка C. И в 1983 году утвержден стандарт, получивший название ANSI C.

Язык уже достаточно давно получил всеобщее признание и стал чуть ли не эталоном. Несколько позже с появлением объектно-ориентированного подхода был разработан C++ или C с классами. И за давностью использования постепенно начинает забываться, что язык C и язык C++ — не совсем одно и то же. Наверное, если взять 100 молодых программистов, владеющих C/C++, и спросить, на чем они пишут, то ответ, скорее всего, будет таков: C++, вне зависимости от того, как в действительности выглядят их программы. Между тем, язык C++ — это язык объектно-ориентированного программирования, а далеко не всегда необходима именно объектная программа, и далеко не всегда программы

[1] Брайан У. Керниган, Денис М. Ритчи. Язык программирования C. — М.: Вильямс, 2005.

пишутся как объектные. Автор этой книги уверен, что подавляющая часть разумно написанных на C/C++ программ — это "обычные" программы, т. е. программы без классов. Слово "обычные" взято, правда, в кавычки, т. к. сегодня, наверное, уже спорно, что более обычно: объектная программа или не объектная. Так вот, разница между C и C++ именно в том, что второй нацелен на поддержку объектно-ориентированного программирования, а первый ничего про классы не знает.

Концепция объектного программирования достаточно сложна и для тех, кто не имеет хорошего программистского опыта или просто не хочет слишком уж углубляться в методологию программирования, а таких, кому необходимо быстро создавать небольшие программы, большинство, в этой концепции им нет необходимости. Это означает, что для такого человека необходимо изложить язык C, а не C++, а книга, которая лежит перед вами, представляет именно такое изложение.

Необходимо, правда, сразу заметить, что и сам по себе язык C достаточно сложен по сравнению, например, с языком Паскаль, который изначально создавался как язык для обучения, а уже потом был перехвачен профессионалами, понявшими его изящество и мощность и по сравнению с языком Бейсик, единственная идея которого — это максимальная простота. Язык C создавался сразу как язык для профессионального программирования, причем системного. Поэтому глубокое изучение языка чревато большими сложностями. Но идти до конца, конечно, не обязательно, большинство прикладных программ не нуждается во всех возможностях, прелестях и нюансах этого языка, а, следовательно, бояться его не стоит.

Первый важный вопрос, который должен встать перед изучающим любой язык, — это: чем данный язык выделяется среди остальных. Ответ таков: C дает максимальную гибкость по отношению к аппаратуре компьютера, большую гибкость имеет только ассемблер и при этом C — все-таки язык высокого уровня, располагающий всем привычным набором операторов.

Гибкость достается особым отношением к типам данных. Они в языке C сильно похожи на те данные, которые непосредственно

понимает процессор. И, несмотря на то, что называются они поразному, фактически каждый тип для компилятора — не более чем последовательность байтов, забота о смысле которой возлагается на программиста. Компилятор С позволит целому присвоить символьное или сложить два символьных значения, т. к. с байтами данных эти операции вполне законны. Если, с точки зрения программиста, такие операции не имеют смысла, то это проблема его и только его.

Язык С дает программисту инструменты для работы с битами данных, что еще более приближает программиста к нижнему уровню (только необходимо сказать правду: по своим возможностям обработки битовых данных С — все-таки далеко не ассемблер). Но за возможности необходимо платить. И плата достаточно серьезная. Очень многие вещи, которые компилятор, например языка Паскаль, не пропустит как ошибочные, для компилятора С будут безразличны, следовательно, ответственность программиста за действия его программы существенно возрастает.

В отношении данных С допускает большую вольность не только в контроле типов. Данные в С-программе можно объявлять где угодно, вследствие чего понятие локальной переменой заметно усложняется. Для С локальная переменная — это переменная, объявленная в составном операторе. Таким образом, понятие локальной переменной опускается с уровня программного модуля, такого как подпрограмма, функция или процедура, до минимального блока, каким является составной оператор. А для полного понимания, что такое данные в С, придется разобраться еще с довольно нетривиальными понятиями времени жизни, области видимости и класса памяти.

Существенные отличия ожидают новичка не только в структурах данных. Для того, чьим первым языком был Бейсик, Фортран или Паскаль, наверное вызовет некоторое удивление оператор `for`. И в Бейсике, и во многих других языках этот оператор является конструкцией цикла, но С предоставляет программисту совершенно необычную свободу действий с компонентами этого оператора. В языке достаточно большой набор способов организации присваивания значений, и не все из них нуждаются в операторе,

обозначаемом привычным знаком равенства. Без них можно обойтись, традиционное присваивание тоже есть, но использование иных форм при достаточном опыте позволит сэкономить немного места.

Наверное, некоторую сложность вызовет нетребовательность языка к расположению функций, которые можно раскидать по программе как угодно, даже разместить в различных файлах.

В общем, язык интересный, а некоторая его сложность вполне будет компенсирована возможностью написания более изящного и эффективного кода.

Эта книга описывает язык довольно подробно, с большим количеством примеров, которыми иллюстрируется практически все, что может вызвать сложность понимания. Более того, если примеров, приведенных в главах, вам будет недостаточно, то в *приложении 2* вы сможете найти еще 20 достаточно интересных примеров.

Глава 1 написана для обеспечения легкости усвоения материала теми, для кого С — первый язык программирования, хотя, может быть, эта глава окажется полезной и более искушенным читателям. По сути, *глава 1* — это очень короткое описание языка. Его достаточно для того, чтобы написать свою первую простую программу и затем, отталкиваясь от этого важного успеха, уже без особых затруднений двигаться по более сложным разделам книги.

Особое значение среди языковых понятий и конструкций имеют указатели. Этому понятию в различной степени посвящены самые разные главы. Есть глава, которая так и называется "Указатели", это *глава 9*. Но едва ли меньшее значение для понимания этого термина имеют *главы 10* и *11*. В них говорится о строках, связных списках и деревьях. Это структуры данных, построенные с помощью указателей. Необходимо отметить, что в языке С указатели среди видов данных занимают особое место. Практически любые данные так или иначе связаны с указателями или могут быть связаны с пользой для программиста.

Глава 11 немного выбивается из общей логики. Здесь, собственно, рассказ идет не о языке, а специальном классе задач, и сама

глава построена как перечень решенных задач, поэтому ее наравне с *приложением 2* можно рассматривать, как практическое приложение, однако в связи с особой важностью темы этот материал оформлен не как приложение, а как глава основного материала.

Технология ввода/вывода на С имеет два решения. Оба решения одинаково интересны и имеют свои преимущества. Поэтому операциям ввода/вывода посвящена отдельная глава — *глава 3*. Было бы логично эту главу совместить с рассказом о файлах, но все же она стоит отдельно по тем соображениям, что ввод с клавиатуры и вывод на экран монитора все же встречается чаще, чем файловый ввод/вывод. Но если вы внимательно прочитаете *главу 12* о файлах, то сможете заметить, что в С есть общее понятие потока данных, а куда этот поток направлен — это второй, если не третий вопрос.

Обработке файловых данных также посвящена отдельная глава, и это тоже глава не о базовых возможностях языка. Сам по себе язык С не знает, что такое файлы. Надо сказать, что в стандарте языка вообще нет средств ввода/вывода, в том числе и средств ввода/вывода с использованием файлов данных. Изучая ввод/вывод, вы фактически изучаете дополнительные библиотеки, разработанные для большего удобства прикладных программистов. Вообще, в книге будет много материала, который выходит за рамки собственно языка и является описанием каких-то дополнительных библиотек. Но без этого материала книга станет тонкой и неинформативной. Говоря о библиотеках, необходимо заметить: то, что выходит за рамки стандарта, не обязано поддерживаться любым компилятором, поэтому не факт, что любой наш пример будет успешно откомпилирован и выполнен в любой среде. Но утешает тот факт, что в России люди, начинающие работать с языком С, все-таки чаще всего обращаются к тому же компилятору, на который опирается и материал этой книги. То есть к компилятору, разработанному компанией Borland для среды MS-DOS

И конечно, важнейшее значение имеют приложения. Их два. *Приложение 1*, описывающее терминологию, наверное, наименее важно. Оно потребуется только в том случае, если у вас возник-

нет желание поглубже разобраться в понятийном аппарате языка и выстроить для себя систему теоретических знаний, но для практического владения языком это, пожалуй, не так уж важно.

Приложение 2, безусловно, имеет очень большую практическую значимость. В нем 20 работающих программ. Первые две программы логически просты. Следующие 18 невелики по тексту, но имеют достаточно сложную логику. И все 20 программ имеют ярко выраженный прикладной характер. Логически сложные задачи представлены в укороченном варианте: условие, кратко идея решения и текст программы, снабженный комментариями. Задачи взяты из книги В. Потопахина "Turbo-Pascal. Решение сложных задач"[2]. Системщику от этих задач толку будет, наверное, немного. В них не показаны особенности работы с регистровой памятью или битовыми операциями, но для тех, кто хочет писать прикладные программы, эти примеры окажут хорошую услугу. Такой подбор примеров обусловлен тем, что книга в основном рассчитана на прикладников, а не на системщиков. Кроме того, если С — ваш первый язык, не стоит сразу увлекаться системными задачами, это может оказаться слишком сложным делом. Более разумно начать с задач прикладных. А если интерес к системному программированию останется, то в этой книге для вас будет достаточно много информации и в этой области.

О технологии чтения

Конечно, любую книгу желательно читать с самого начала и делать это последовательно. Но что касается языка программирования, то здесь есть небольшая, но очень существенная сложность. Если язык излагать строго так, как это принято в официальных документах, описывающих стандарт языка, то вполне можно добиться жесткой последовательности и полной непротиворечивости. Но это повлечет за собой потерю прозрачности языковых конструкций.

[2] Потопахин В. Turbo-Pascal. Решение сложных задач. — СПб.: "БХВ-Петербург", 2006.

Автор в построении книги исходит из того, что необходимо изучать не формальные определения, а функциональные возможности. Но тогда очень трудно определить, что за чем должно следовать. Например, имя массива в языке С является указателем на область памяти, в которой данный массив хранится. С другой стороны, понятие указателя предоставляет возможность организации массивов. Оба понятия функционально взаимосвязаны настолько сильно, что разорвать их практически невозможно. А разрывать все равно приходится, т. к. какой-то порядок изложения все-таки необходим. Поэтому иногда придется оставлять некоторые вопросы в уме, в надежде получить ответы позже, а иногда возвращаться назад, чтобы лучше понять, о чем идет речь, достаточно часто материал повторяется в связи с введением новых понятий.

Отчасти *глава 1* снимает эту трудность, т. к. уже с первых страниц дает общий обзор языка, такую общую, хотя и очень приблизительную картину. Но главный инструмент в борьбе с трудностями чтения — это примеры. Почти все, что нужно проиллюстрировать, иллюстрируется примерами, которые сразу работают. Но конечно, все наши усилия не решат проблемы восприятия языка без вашей активной работы с компилятором. Это, впрочем, верно в отношении абсолютно любого языка. Язык программирования изучается в практической деятельности, и этого правила еще никто не отменял.

И последнее замечание. Часть примеров оформлена в виде фрагментов, которые необходимо дополнить прежде, чем они станут программой, а часть примеров — это действительно работающие программы, которые были тщательно отлажены прежде, чем попасть в текст. Поэтому если вы наберете наш пример и он откажется работать, то все-таки посмотрите внимательно, верно ли он набран. Помните, в нашем деле мелочей нет, и один неверно введенный символ может кардинально изменить смысл программы.

Глава 1

Кратко о языке С

Если язык С — не первый ваш язык и вы не считаете себя начинающим программистом, то эту главу можно пропустить, но если вы с языка С начинаете свое программистское образование, то данный текст необходимо прочитать максимально внимательно. Далее, на очень простых примерах мы рассмотрим организацию ввода/вывода в С, самые основные конструкции языка и общую структуру программы. Затем это все будет рассмотрено еще раз и более детально, но, может быть, первый раз лучше обойтись без многочисленных деталей, а увидеть все главное и сразу.

Несмотря на то, что в этой главе подробно разбираются достаточно содержательные примеры и показаны все основные конструкции, не рассчитывайте, что после прочтения вы сможете делать что-то серьезное самостоятельно. Цель главы очень проста — только составить самое общее представление о том, что такое программа, написанная на языке С. Для того чтобы составить быстрое и в то же время достаточно качественное представление об изучаемом языке, мы не будем тщательно изучать типы данных, правила их объявления и преобразования, не станем подробно говорить о свойствах различных библиотечных функций и операций. Мы подойдем к языку с прикладной точки зрения. А именно попробуем решить несколько несложных задач и для каждой из них посмотрим, что необходимо для записи соответствующего алгоритма.

Простые примеры

Рассмотрим следующую задачу. Пусть дано множество чисел. Необходимо найти среди них наибольшее.

Предположим, что наше множество пронумеровано. Это естественное предположение, потому что если даже это и не так, то пронумеровать элементы множества несложно. Можно любой выбранный элемент объявить первым, затем любой из оставшихся — вторым, потом любой из оставшихся — третьим и т. д.

Итак, нумерация проблемы не составляет, поэтому будем считать, что числа множества пронумерованы. Договоримся об обозначениях. Пусть множество обозначено буквой A. Тогда A_1 — первый элемент, A_2 — второй и т. д., т. е. A_i — i-ый элемент. Количество элементов множества будем обозначать буквой N. С учетом принятых договоренностей можно написать алгоритм ввода такого множества чисел. Текст ниже:

```
Ввести N
Для i от 1 до N с шагом 1 делать: Ввести Ai
```

Команда `Ввести` говорит о том, что для ее исполнения необходимо ввести число с клавиатуры компьютера. Вторая команда называется *конструкцией цикла*. Она состоит из двух составляющих: заголовка цикла и набора команд, называемого телом цикла. В нашем примере тело цикла состоит из одной команды.

В заголовке сказано, что величина `i`, имея в начале выполнения цикла значение 1, изменяется на каждом шаге работы цикла на единицу, и это происходит до тех пор, пока ее значение не достигнет значения `N`. Как только значение `i` достигнет `N`, выполнение цикла прекратится. В тексте шаг цикла состоит в выполнении команды `Ввести Ai` — эта команда является телом цикла. В нашем примере в теле цикла только одна команда, но их может быть несколько.

Из сказанного, наверное, уже понятно, что смысл конструкции цикла в том, чтобы многократно выполнить много команд, не повторяя их записи. В нашем цикле на первом шаге будет введено

число с номером 1, на втором — число с номером 2 и т. д. до номера `N`.

Конечно, если вы имеете хотя бы небольшой опыт программирования на любом другом языке, то сказанное ранее должно быть вполне понятно, ничего нового мы здесь не сообщили. Поэтому далее просто запишем, как текст этого алгоритма выглядит на языке С.

```
cin >> N;
for (int i=1;i<=N;i++) cin >> A[i];
```

Нужно пояснить эти строки.

Команда `cin` — это команда ввода значения с клавиатуры. В нашем фрагменте сначала вводится число `N`, а затем в теле цикла многократно выполняется ввод элементов массива.

Массив — это сложная структура данных, которая содержит в себе много чисел (или данных другого типа), имеющих одинаковое имя. В нашем случае — общее имя элементов массива `A`, в квадратных скобках проставляется номер элемента, и только номерами они друг от друга и отличаются.

`for` — это ключевое слово, обозначающее цикл, точнее один из типов циклов, существующих в С (есть еще два, но о них позже). После него в круглых скобках записано то, что было записано в заголовке цикла алгоритма: `i=1` — параметр цикла имеет начальным значением единицу. `i<=N` — условие продолжения цикла (цикл работает, пока параметр не больше `N`). И, наконец, запись `i++` говорит о том, что `i` изменяется с шагом 1.

Ключевое слово `int` утверждает, что величина `i` — целая. Это называется *объявлением типа*. Выполнить операцию объявления типа надо бы для всех используемых переменных. То есть в полностью завершенной программе должно быть объявление типа для массива `A` и величины `N`.

Учтите, это не текст законченной программы. Чтобы этот фрагмент реально начал работать, необходимо сделать еще достаточно много. Но мы пока вернемся к задаче, решим ее и запишем алгоритм на обычном русском языке. Идея алгоритма такова: объявим первое число наибольшим, а затем для каждого из ос-

тавшихся проверим, не больше ли оно уже найденного наибольшего, и если больше, то поменяем значение наибольшего. Далее представлен алгоритм:

```
Наибольшее = первому
Для всех чисел, начиная от второго и до N—го, делать
    Если Очередное число больше Наибольшего
      То Наибольшее = Очередному
Вывести значение Наибольшего
```

А теперь то же самое запишем на языке С.

```
int max=A[1];
for ( int i =2; i<=N;i++)
  if (max<A[i]) max=A[i];
cout << max;
```

Смысл нового ключевого слова `cout`, наверное, уже ясен: это команда вывода значения на экран монитора. И появилось еще одно ключевое слово — `if`. Но его смысл также должен быть понятен из текста алгоритма. Эта команда проверяет условие (условие обязательно записывается в скобках), записанное после ключевого слова `if`, и если условие верно, то выполняется команда, записанная после условия. Это самая простая форма *условного оператора*, подробности мы сейчас рассматривать не будем. Необходимо пояснить, что в теле цикла многократно выполняется только команда выбора `if`, т. к. цикл `for` считает своим телом только одну команду, записанную после заголовка. Правда, эта единственная команда может быть составным оператором, но об этом позже, сейчас же скажем, что команда `cout` выполняется уже за пределами цикла и, следовательно, выполняется только один раз (листинг 1.1).

Листинг 1.1

```
#include <iostream.h>
#include <conio.h>
void main()
{ int N,A[100];
```

```
  cin >> N;
  for (int i=1;i<=N;i++) cin >> A[i];
  int max=A[1];
  for (i=2;i<=N;i++)
    if (max<A[i]) max=A[i];
  cout << max;
  getch();
}
```

Начинается текст программы с записи `#include <iostream.h>`. Это не команда языка. Это директива компилятору, директивам в нашей книге посвящена *глава 15*, в которой достаточно подробно описано, зачем они нужны и какие директивы существуют. Директивы в отличие от команд выполняются только на этапе компиляции, а в ходе исполнения программы их в тексте кода уже нет. Данная директива подключает к тексту программы так называемый *файл заголовков*. Этот файл состоит из имени и расширения h и содержит объявления различных функций, хранящихся в библиотеках (не все возможности языка известны непосредственно компилятору, часть из них хранится отдельно и подключается к работе по мере необходимости).

Далее записан исполняемый текст. Он представлен одной функцией, состоящей из заголовка и текста, заключенного в фигурные скобки. Заголовок несет в себе информацию о том, данные какого типа функция возвращает по завершении своей работы, тип указывается перед именем. Затем записывается имя функции. Имя — это почти любой набор допустимых символов, но одна функция обязательно должна называться `main()`, с нее начинается выполнение программы. В нашей программе функция только одна, поэтому она в обязательном порядке имеет имя `main()`.

И наконец, в круглых скобках после имени (эти скобки обязательны в любом случае) записываются имена переменных, используемых функцией для получения значений извне. Наша программа состоит только из одной функции, она ничего не получает, и круглые скобки пусты.

Текст программы мы уже анализировали ранее, поэтому скажем только об одной команде:

```
int N,A[100];
```

В ней, как и было обещано, объявлены недостающие типы для переменной `N` и массива, а для массива еще указан его максимально допустимый размер.

И самое последнее. Функция `getch()` нужна только для того, чтобы окно с выполняемой программой не закрылось с завершением выполнения, и мы успели что-либо увидеть. Эта функция требует ввода символа, и пока ни одна клавиша не нажата, программа будет ждать, выполнение не будет завершено, и мы увидим результаты работы программы.

Все! Программа полностью готова, ее можно откомпилировать и запустить на выполнение.

Наша первая программа состоит из одной функции, но может состоять и из нескольких. Поэтому даже для первого знакомства необходимо показать, как строится такая программа. Для примера возьмем ту же самую задачу. Ввод и вывод результата оставим главной функции, а расчеты перенесем в другую. Для того чтобы показать процесс передачи данных из функции в функцию, передадим в дополнительную функцию величину `N`. Конечно, надо было бы передать и массив `A`. Но так как наша цель — только знакомство, не будем усложнять себе задачу и ограничимся передачей лишь одной величины. Текст программы записан в листинге 1.2.

Листинг 1.2

```
#include <iostream.h>
#include <conio.h>
int maximum(int);
int A[100];
void main()
{ int N;
  cin >> N;
  for (int i=1;i<=N;i++) cin >> A[i];
```

```
  int max=maximum(N);
  cout << max;
  getch();
}
int maximum(int m)
{
  int max=A[1];
  for (int i=2;i<=m;i++)
    if (max<A[i]) max=A[i];
  return max;
}
```

Примечание

В тексте программы появилось одно, на первый взгляд, несущественное изменение. А именно объявление массива сделано до функции `main()`. На самом деле, это очень важный момент, связанный с понятиями локальной и глобальной переменных.

Глобальной называется переменная, которой можно пользоваться в любом месте программы. *Локальная переменная* наоборот доступна только в той структурной единице, в которой она объявлена. Эти два понятия присутствуют почти в любом языке программирования, за исключением, быть может, некоторых версий языка Бейсик. Отличие между языками заключается в том, что понимается под структурной единицей. Например, в языке Паскаль, структурными единицами являются процедура и функция. В языке С структурной единицей считается составной оператор, т. е. группа команд, записанная между фигурными скобками. Проблемы с объявлениями переменных мы еще рассмотрим подробнее, сейчас же будет достаточно сказать, что функция также является составным оператором, поэтому массив, объявленный в предыдущем решении, известен только функции `main()`, т. к. объявлен внутри нее. Если все так и оставить, то на этапе компиляции получим сообщение, что массив функции `maximum()` не известен. Мы же вынесли объявление массива за пределы всех существующих фигурных скобок, что как раз и обеспечивает доступность массива нашей вспомогательной функции, он становится глобальным.

Еще одно важное дополнение — `int maximum(int)`. Это так называемый *"прототип"* или *"заголовок"*. Появление прототипа есть следствие важного принципа, который также присутствует в любом языке. Звучит принцип так: все, что используется в программе, должно быть объявлено до первого своего использования. В нашем же примере функция `maximum()` используется в функции `main()`, а описана ниже ее, следовательно, на момент своего первого использования (т. е. в функции `main()`) функция `maximum()` неизвестна. Для того чтобы выйти из затруднительной ситуации, и описывается прототип, т. е. краткая информация о том, что представляет собой функция `maximum()`. В нашем примере указано, что она возвращает целое значение, на входе получает одну целую переменную и имеет имя `maximum`. После того как записан прототип, саму функцию можно описывать где угодно в тексте программы. Единственный твердый запрет языка C — это запрет на вложенные описания, т. е. функцию нельзя записывать внутри другой функции. Без прототипа можно обойтись, если функцию `maximum()` разместить до функции `main()`, но все же принято размещать функции так, как удобно разработчику, а в начале файла программы указывать все их прототипы.

В самой функции `maximum()` появилась новая команда `return`. Это команда, возвращающая значение, т. е. то значение, которое функцией `maximum()` будет передано переменной `max`.

В рассмотренном примере мы воспользовались целочисленным типом `int`. Конечно, это не единственный тип данных, поддерживаемый языком C. Существуют два типа для чисел с десятичной точкой. Это тип `float` (одинарная точность) и тип `double` (двойная точность). Для представления символьных данных предназначен тип `char`.

Следующий пример (листинг 1.3) показывает использование типа `double`.

Листинг 1.3

```
#include <iostream.h>
#include <math.h>
void main()
```

```
{ double result[100][3];
  int i=0;
  for (float x=0; x<=3.14; x+=0.1)
    { result[i][0]=x;
      result[i][1]=sin(x);
      result[i][2]=cos(x);
      i++;
    }
  for (int k=0;k<i;k++)
   cout <<result[k][0]<<" "<<result[k][1]<<" "<<re-
sult[k][2]<<"\n";
}
```

Обратите внимание, что в заголовке цикла величина `x`, играющая роль параметра цикла, имеет тип `float`, т. е. является действительным (вещественным) числом. Во многих языках программирования параметр цикла — величина обязательно целая. Язык С свободен от такого ограничения. Оператор

```
x+=0.1
```

это еще одна форма *оператора присваивания*. То же самое можно записать универсальным присваиванием:

```
x=x+0.1
```

Но С предлагает нам целый набор специализированных операторов, которые в некоторых случаях помогают выполнить запись короче. Язык С для каждой арифметической операции предлагает дополнительную форму присвоения: `+=`, `-=`, `*=`, `/=`. Это звучит так: вычислить и прибавить, вычислить и отнять и т. д.

Еще два важных замечания о массивах

Замечание первое. Массив, объявленный в нашем примере, назвать двумерным массивом будет не совсем точно. Более правильно будет сказать, что мы объявили 100 массивов длиной по 3 элемента каждый. В большинстве случаев разницы между многомерным массивом и массивом массивов нет, но все равно запомните этот факт. Хорошо изучив указатели и усвоив связь между указателями и массивами, вы поймете, что это не вполне одно и то же.

Замечание второе. Нумерация элементов массива всегда начинается с нуля. Объявить массив с произвольным интервалом изменения индекса нельзя. В некоторых языках такого ограничения нет. В C это ограничение связано с идеей приближения языка к языками низкого уровня и обеспечения максимальной гибкости и переносимости программ.

Еще два важных понятия, необходимых для хорошего понимания механизма работы C с величинами: это *область видимости* и время жизни величины. Эти два понятия описывают, где, в какой части программы объявленной переменной можно пользоваться, и сколько времени она существует. Язык C разрешает объявлять переменные где угодно, поэтому и появилась необходимость в правилах для определения времени жизни и области видимости. Для простых случаев эти два понятия сложностей не составляют. *Область видимости* для переменной — это тот составной оператор, в котором она объявлена. *Время жизни* — с момента объявления до момента завершения выполнения составного оператора, в котором величина объявлена.

Но в C все не так просто. Прежде всего, объявление может быть сделано на внешнем уровне, т. е. за пределами всех возможных блоков, тогда эта переменная глобальная. И переменная может быть объявлена внутри оператора — это иная ситуация, для которой наше определение полностью бы подходило, если бы не еще один важный нюанс. В C существуют *классы памяти*, определяющие способ хранения переменных и в силу этого влияющие на область видимости и время жизни. Наши обычные переменные, такие, которые мы использовали в приведенных примерах, — это переменные класса `auto`. Для них определение области видимости и времени жизни остаются в силе. Но это не единственный класс. Существуют еще три. О них подробнее позже. Сейчас только упомянем о классе `static`. Область видимости для переменных этого класса определяется так же, а вот время жизни становится глобальным. Это означает, что выход за пределы блока, в котором переменная была объявлена, не приводит к потере значения. И это независимо от того, чем именно является блок — просто сложным оператором или функцией.

Рассмотрим следующий пример (листинг 1.4).

Листинг 1.4

```
#include <iostream.h>
void main()
{ int s;
  for (int i=1; i<=5; i++)
    for (int k=1; k<=5; k++)
    { static int sum=0;
      sum+=k;
      s=sum;
    }
  cout << s;
}
```

Результатом работы будет число 75. Это означает, что величина `sum` накапливается на всех шагах двух циклов и не инициализируется нулем при повторном вхождении в составной оператор, следующий после заголовка второго цикла. Уберите ключевое слово `static`, и вы увидите совершенно иной результат.

Условные циклы

Предположим, есть необходимость вычислять некоторую величину до тех пор, пока не будет выполнено некоторое условие. Пусть, например, необходимо посчитать сумму элементов ряда 1 + 1/2 + 1/3 + ... до тех пор, пока элемент ряда не станет меньше некоторого малого числа. Ясно, что нужен цикл, но сейчас нам неизвестно, сколько будет выполнено операций. Эту задачу можно решить тремя способами. Вполне сгодится уже известный нам цикл `for` в силу его огромной гибкости, но С для таких ситуаций предоставляет еще две формы цикла: цикл с предусловием вида:

`while (`*условие*`)` *оператор*

и цикл с постусловием, т. е. цикл вида

`do`

{ *перечень операторов*

```
}
while (условие)
```

Решение нашей задачи с помощью цикла `for` представлено в листинге 1.5.

Листинг 1.5

```
#include <conio.h>
#include <iostream.h>
void main()
{ clrscr();
  float sum=0;
  for (float i=1;1/i>=0.001;i++) sum+=1/i;
  cout <<sum;
}
```

Эта же задача для цикла `while` представлена в листинге 1.6.

Листинг 1.6

```
#include <conio.h>
#include <iostream.h>
void main()
{ clrscr();
  float sum=0;
  float i=1;
  while (1/i>=0.001)
   {sum+=1/i;
    i++;
   }
  cout <<sum;
}
```

И эта же задача для цикла `do...while` решена с помощью программы из листинга 1.7.

Листинг 1.7

```
#include <conio.h>
#include <iostream.h>
void main()
{ clrscr();
  float sum=0;
  float i=1;
  do
   {sum+=1/i;
    i++;
   }
  while (1/i>=0.001);
  cout <<sum;
}
```

Указатели

В языке С много любопытных конструкций и объектов. Конечно, в кратком изложении мы не будем говорить обо всем даже в двух словах, но, по крайней мере, об указателях сказать совершенно необходимо. Внешне *указатель* — очень простая вещь. Это переменная, содержащая адрес, по которому находится числовое или какое-либо иное значение. Объявить указатель просто, вот так:

```
int *a
```

Это объявление указателя на целую величину. Указатели полезны во многих отношениях, например, когда необходимо передать величину из одной функции в другую, далее мы поговорим об этом более подробно. Но самое важное в них, с точки зрения автора этой книги, — это то, что указатели представляют собой универсальный механизм выделения памяти под любые переменные и структуры. Указатель может ссылаться как на одиночную переменную, так и на массив, и в обоих случаях он работает одинаково.

Для того чтобы обратиться к адресу, переменная, объявленная как указатель, используется по своему имени, а чтобы обратиться

к значению, на которое ссылается указатель, к его имени достаточно добавить звездочку, вот так:

- `int *a;` — объявлен указатель на целое;
- `*a=8;` — в ячейку памяти, на которую указывает a, помещено число 8;
- `a++;` — выполнен переход на следующую ячейку.

Для указателей в C доступно достаточно много мощных операций, что делает их важным и интересным инструментом. А сейчас перепишем программу поиска максимального с использованием указателей (листинг 1.8).

Листинг 1.8

```
#include <iostream.h>
#include <conio.h>
int maximum(int,int *);
void main()
{ int *a,*b;
  clrscr();
  int n;
  cin >>n;
  a=new int[n+1];
  b=a;
  for (int i=0;i<=n;i++)
    {cin >>*a;
     a++;
    }
  int max=maximum(n,b);
  cout << max;
}
int maximum(int n, int *a)
{ int max=*a;
  for (int i=0;i<=n;i++)
   { if (*a>max) max=*a;
```

```
    a++;
   }
  return max;
}
```

Во-первых, сразу обратите внимание на тот факт, что мы избавились от глобальных переменных. Это хорошо. Во-вторых, обратите внимание на оператор

```
a=new int[n+1];
```

Здесь мы выделили память под массив именно такого размера, какой нам нужен, и ни одного байта больше. Объявить массив как

```
int a[n+1];
```

нельзя, а указатель помогает нам обойти это ограничение.

Заключение

Итак, еще раз перечислим основные объекты языка С, с которыми нам придется иметь дело, в двух словах опишем их функциональное назначение и на этом завершим наше краткое изложение и перейдем к подробному.

Итак. Переменные и константы являются основными объектами, которыми оперирует программа. Описания перечисляют переменные, которые будут использоваться, указывают их тип и, возможно, их начальные значения. Операции определяют, что с переменными и константами будет сделано. Выражения объединяют переменные и константы для получения новых значений. А оператор — это минимальная управляющая команда, используемая в программе.