ВИТАЛИЙ ПОТОПАХИН

Язык C

ОСВОЙ НА ПРИМЕРАХ

БАЗОВЫЕ И СОСТАВНЫЕ ТИПЫ

ОСНОВНЫЕ ЯЗЫКОВЫЕ КОНСТРУКЦИИ

УКАЗАТЕЛИ И ИХ СВЯЗЬ С ДРУГИМИ СТРУКТУРАМИ ДАННЫХ

СТРУКТУРЫ И ПОЛЯ

РАБОТА С ФАЙЛАМИ

СВЯЗАННЫЕ СПИСКИ И ДЕРЕВЬЯ

bhv®

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

© Потопахин В. В., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

Введение **1**

О технологии чтения 6

Глава 1. Кратко о языке С **9**

Простые примеры 10
Условные циклы 19
Указатели 21
Заключение 23

Глава 2. Величины **25**

Типы данных 25
Расположение объявлений 29
Инициализация переменных 30
Более подробно о времени жизни и области видимости 36
Классы памяти 39
  Объявления внутреннего уровня 40
  Объявления внешнего уровня 42
О преобразованиях типов 43
Объявления вида *typedef* 47
Заключение 48

Глава 3. Ввод/вывод **51**

Форматный ввод/вывод 51
Потоковый ввод/вывод 54
  Интересный пример вывода 55

Интересный пример ввода 55
Несколько полезных функций 56
Функции *putchar()* и *getchar()* 56
Функции *textcolor()* и *textbkcolor()* 57
Функция *gotoxy()* 58
Команда перевода курсора на следующую строку 58

Глава 4. Операции в С 61

Список операций 61
Арифметические операции 64
Операция *%* 65
Операции сдвига 66
Битовая логика 68
Логические операции 69
Операция следования 70
Операции индексации и построения выражений 71
Операция определения размера объекта *sizeof* 72
Арифметическое условие 73
Старшинство и порядок вычисления 74
Еще несколько замечаний о порядке выполнения операций 75

Глава 5. Операторы языка С 77

Общие сведения 77
Циклы 77
Цикл с предусловием 80
Цикл с постусловием 81
О структуре цикла 83
Замечания о цикле с постусловием 84
Конструкции выбора 86
Условный оператор 86
Оператор-переключатель 89
Оператор *break* 91
Составной оператор 92
Оператор продолжения *continue* 93
Оператор-выражение 94
Оператор *goto* 95

Пустой оператор *;* 96
Оператор возврата *return* 96

Глава 6. Константы 99

Простые константы 99
Перечисления 104

Глава 7. Массивы 107

Общие правила работы с массивами 107
Инициализация массива символов 115
Заключение 116

Глава 8. Структуры и объединения 117

Общие сведения 117
Структуры 118
Замечание об инициализации 120
Объединения 121
Замечание об инициализации объединений 125
Поля и экономия памяти 126
Заключение 129

Глава 9. Указатели 131

Объявление и создание указателей 131
Доступ к элементам структуры 136
Выделение и освобождение памяти 142
Арифметические операции над указателями 143
Замечание о шаге указателя 148
Указатель в никуда 151
Заключение 151

Глава 10. Строки 153

Нультерминальные строки С 153
О вложении присваивания 161
Пример упорядочения строк 162

Полезные функции 163
Функция *strstr()* 164
Функции *strlow()* и *strupr()* 164
Функция *strset()* 165
Функция *strcmp()* 166
Заключение 166

Глава 11. Связные списки и деревья 167

Построение связного списка 167
Задача 1. Обход связного списка 168
Задача 2. Обход связного списка неизвестной длины 169
Задача 3. Обход связного списка в двух направлениях 171
Несколько замечаний о методах обхода 172
Задача 4. Вставка элемента в связный список 173
Задача 5. Вставка связного списка 175
Задача 6. Сортировка по возрастанию массива, организованного в виде связного списка 176
Деревья 178
Задача 7. Построение двоичного дерева 178
Задача 8. Обмен местами значения левой и правой ветви двоичного дерева 179
Заключение 182

Глава 12. Файлы данных 183

Неформатный ввод/вывод 183
Форматный файловый ввод/вывод 189
Полезные функции 191
Функция *fseek()* 191
Функция *fgetpos()* 192
Функция *rewind()* 192
Функция *creat()* 193
Заключение 194

Глава 13. Построение сложной программы 195

Общие сведения 195
Передача одномерного массива 198
Пример 1. Расчет максимального в массиве 198

Пример 2. Расчет максимального с передачей массива, как параметра 200
Пример 3. Расчет максимального с передачей массива, как указателя 201
Передача многомерных массивов 203
Передача параметров по ссылке 204
Параметр по умолчанию 205
О возвращаемых значениях 206
Перегрузка функций 209
Указатель на функцию 212
Комментарии 215
Заключение 216

Глава 14. Графика языка С 217

Общая информация о графических режимах 217
Некоторые функции 219
Функция *arc()* 219
Функция *bar()* 219
Функция *bar3d()* 220
Функция *circle()* 220
Функция *cleardevice()* 220
Функция *closegraph()* 220
Функция *drawpoly()* 221
Функция *ellipse()* 221
Функция *floodfill()* 222
Функция *getbkcolor()* 222
Функция *getcolor()* 222
Функции *getmaxx()* и *getmaxy()* 222
Функция *getpixel()* 222
Функции *getx()* и *gety()* 223
Функция *linerel()* 223
Функция *lineto()* 223
Функция *moverel()* 223
Функция *moveto()* 223
Функция *outtext()* 224
Функция *outtextxy()* 224
Функция *putpixel()* 224

Функция *rectangle()* 224
Функция *sector()* 225
Функция *setbkcolor()* 225
Функция *setcolor()* 225
Функция *setfillstyle()* 225

Глава 15. Директивы 227

Директива *include* 227
Директива *define* 229
Строкообразующий оператор # 231
Замечание о расположении макросов 232
Заключение 233

ПРИЛОЖЕНИЯ 235

Приложение 1. Терминология 237

Приложение 2. Примеры задач 245

Предметный указатель 299

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 С — 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 С. 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 случаях 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 С 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 С 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 С 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 С 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 С 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
```