**ВВЕДЕНИЕ В ЯЗЫК СИ**

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ**](http://dfe.karelia.ru/koi/posob/c/c.htm#intr)

[**РАЗДЕЛ 1. ОСНОВНЫЕ ПОНЯТИЯ И ДАННЫЕ**](http://dfe.karelia.ru/koi/posob/c/c.htm#g1)

* + [**Основные понятия языка**](http://dfe.karelia.ru/koi/posob/c/c.htm#g1.1)
  + [**Типы данных**](http://dfe.karelia.ru/koi/posob/c/c.htm#g1.2)
  + [**Переменные и константы**](http://dfe.karelia.ru/koi/posob/c/c.htm#g1.3)
  + [**Как вводить и выводить информацию**](http://dfe.karelia.ru/koi/posob/c/c.htm#g1.4)
  + [**Форматированный вывод данных**](http://dfe.karelia.ru/koi/posob/c/c.htm#g1.5)
  + [**Форматированный ввод данных**](http://dfe.karelia.ru/koi/posob/c/c.htm#g1.6)

[**РАЗДЕЛ 2. ОПЕРАЦИИ И ОПЕРАТОРЫ**](http://dfe.karelia.ru/koi/posob/c/c.htm#g2)

* + [**Операции языка Си**](http://dfe.karelia.ru/koi/posob/c/c.htm#g2.1)
  + [**Преобразование типов**](http://dfe.karelia.ru/koi/posob/c/c.htm#g2.2)
  + [**Указатели и операции с ними**](http://dfe.karelia.ru/koi/posob/c/c.htm#g2.3)
  + [**Операторы цикла**](http://dfe.karelia.ru/koi/posob/c/c.htm#g2.4)
  + [**Операторы условных и безусловных переходов**](http://dfe.karelia.ru/koi/posob/c/c.htm#g2.5)

[**РАЗДЕЛ 3. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3)

* + [**Массивы**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3.1)
  + [**Строки символов**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3.2)
  + [**Структуры**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3.3)
  + [**Оператор typedef**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3.4)
  + [**Битовые поля**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3.5)
  + [**Объединения (union)**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3.6)
  + [**Перечислимый тип данных**](http://dfe.karelia.ru/koi/posob/c/c.htm#g3.7)

[**РАЗДЕЛ 4. ФУНКЦИИ**](http://dfe.karelia.ru/koi/posob/c/c.htm#g4)

* + [**Общие сведения**](http://dfe.karelia.ru/koi/posob/c/c.htm#g4.1)
  + [**Классы памяти**](http://dfe.karelia.ru/koi/posob/c/c.htm#g4.2)
  + [**Указатели на функции**](http://dfe.karelia.ru/koi/posob/c/c.htm#g4.3)
  + [**Аргументы функции main()**](http://dfe.karelia.ru/koi/posob/c/c.htm#g4.4)
  + [**Рекурсия**](http://dfe.karelia.ru/koi/posob/c/c.htm#g4.5)
  + [**Библиотечные функции**](http://dfe.karelia.ru/koi/posob/c/c.htm#g4.6)

[**РАЗДЕЛ 5. ФАЙЛЫ**](http://dfe.karelia.ru/koi/posob/c/c.htm#g5)

[**РАЗДЕЛ 6. ДРУГИЕ ВОЗМОЖНОСТИ**](http://dfe.karelia.ru/koi/posob/c/c.htm#g6)

* + [**Динамическое распределение памяти. Функции malloc( ) и free( )**](http://dfe.karelia.ru/koi/posob/c/c.htm#g6.1)
  + [**Препроцессор**](http://dfe.karelia.ru/koi/posob/c/c.htm#g6.2)
  + [**Использование программно-доступных регистров микропроцессора Intel 8086**](http://dfe.karelia.ru/koi/posob/c/c.htm#g6.3)

[**РАЗДЕЛ 7. ПРИМЕРЫ**](http://dfe.karelia.ru/koi/posob/c/c.htm#g7)

[**ЛИТЕРАТУРА**](http://dfe.karelia.ru/koi/posob/c/c.htm#lit)

**ВВЕДЕНИЕ**

Язык Си, созданный Денисом Ритчи в начале 70-х годов в Bell Laboratory американской корпорации AT&T, является одним из универсальных языков программирования. Язык Си считается языком системного программирования, хотя он удобен и для написания прикладных программ. Среди преимуществ языка Си следует отметить переносимость программ на компьютеры различной архитектуры и из одной операционной системы в другую, лаконичность записи алгоритмов, логическую стройность программ, а также возможность получить программный код, сравнимый по скорости выполнения с программами, написанными на языке ассемблера. Последнее связано с тем, что хотя Си является языком высокого уровня, имеющим полный набор конструкций структурного программирования, он также обладает набором низкоуровневых средств, обеспечивающих доступ к аппаратным средствам компьютера. С 1989 года язык Си регламентируется стандартом Американского института национальных стандартов ANSI С. В настоящее время, кроме стандарта ANSI C разработан международный стандарт ISO C (International Standard Organization C).

В пособии в разделах 1-6 рассматриваются основные конструкции языка Си (общие для Си и Си++). Примеры программ приведены в разделе 7.

|  |  |
| --- | --- |
|  |  |

**РАЗДЕЛ 1. ОСНОВНЫЕ ПОНЯТИЯ И ДАННЫЕ**

***Основные понятия языка***

Программа, написанная на языке Си, состоит из операторов. Каждый оператор вызывает выполнение некоторых действий на соответствующем шаге выполнения программы.

При написании операторов применяются латинские прописные и строчные буквы, цифры и специальные знаки. К таким знакам, например, относятся: точка (.), запятая (,), двоеточие (:), точка с запятой (;) и др. Совокупность символов, используемых в языке, называется алфавитом языка.

В персональном компьютере символы хранятся в виде кодов. Соответствие между каждым символом и его кодом задается специальной кодовой таблицей. На нее разработан стандарт ASCII, поэтому коды символов называют ASCII-кодами.

Различают *видимые* и *управляющие символы*. Первые могут быть отображены на экране дисплея либо отпечатаны на принтере. Вторые вызывают определенные действия в машине, например: звуковой сигнал - код 710, возврат курсора на один шаг - код 810, горизонтальная табуляция - код 910, перевод курсора на новую строку - код 1010, перемещение курсора в начало строки - код 1310 и т.д. Такие управляющие символы имеют десятичные номера 0 - 31, 127.

Для представления каждого символа в персональном компьютере используется один байт, поэтому общее число символов равно 28 = 256. Кодовая таблица, которая устанавливает соответствие между символом и его кодом, имеет 256 строк вида:

код\_символа\_в\_заданной\_системе\_счисления - символ.

Первая половина кодовой таблицы является стандартной, а вторая используется для представления символов национальных алфавитов, псевдографических элементов и т.д.

Важным понятием языка является идентификатор, который используется в качестве имени объекта (функции, переменной, константы и др.). Идентификаторы должны выбираться с учетом следующих правил:

1. Они должны начинаться с буквы латинского алфавита (а,...,z, А,...,Z) или с символа подчеркивания (\_).
2. В них могут использоваться буквы латинского алфавита, символ подчеркивания и цифры (0,...,9). Использование других символов в идентификаторах запрещено.
3. В языке Си буквы нижнего регистра (а,...,z), применяемые в идентификаторах, отличаются от букв верхнего регистра (А,...,Z). Это означает, что следующие идентификаторы считаются разными: name, NaMe, NAME и т.д.
4. Идентификаторы могут иметь любую длину, но воспринимается и используется для различения объектов (функций, переменных, констант и т.д.) только часть символов. Их число меняется для разных систем программирования, но в соответствии со стандартом ANSI C не превышает 32 (в Си++ это ограничение снято). Если длина идентификатора установлена равной 5, то имена count и counter будут идентичны, поскольку у них совпадают первые пять символов.
5. Идентификаторы для новых объектов не должны совпадать с ключевыми словами языка и именами стандартных функций из библиотеки.

В программах на языке Си важная роль отводится комментариям. Они повышают наглядность и удобство чтения программ. Комментарии обрамляются символами /\* и \*/. Их можно записывать в любом месте программы.

В языке Си++ введена еще одна форма записи комментариев. Все, что находится после знака // до конца текущей строки, будет также рассматриваться как комментарий. Отметим, что компилятор языка Си, встроенный в систему программирования Borland C++, позволяет использовать данный комментарий и в программах на Си.

Пробелы, символы табуляции и перехода на новую строку в программах на Си игнорируются. Это позволяет записывать различные выражения в хорошо читаемом виде. Кроме того, строки программы можно начинать с любой позиции, что дает возможность выделять в тексте группы операторов.

|  |  |
| --- | --- |
|  |  |

***Типы данных***

Программы оперируют с различными данными, которые могут быть простыми и структурированными. *Простые данные* - это целые и вещественные числа, символы и указатели (адреса объектов в памяти). Целые числа не имеют, а вещественные имеют дробную часть. *Структурированные данные* - это массивы и структуры; они будут рассмотрены ниже.

В языке различают понятия "тип данных" и "модификатор типа". Тип данных - это, например, целый, а модификатор - со знаком или без знака. Целое со знаком будет иметь как положительные, так и отрицательные значения, а целое без знака - только положительные значения. В языке Си можно выделить пять базовых типов, которые задаются следующими ключевыми словами:

* char - символьный;
* int - целый;
* float - вещественный;
* double - вещественный двойной точности;
* void - не имеющий значения.

Дадим им краткую характеристику:

1. Переменная типа char имеет размер 1 байт, ее значениями являются различные символы из кодовой таблицы, например: 'ф', ':', 'j' (при записи в программе они заключаются в одинарные кавычки).
2. Размер переменной типа int в стандарте языка Си не определен. В большинстве систем программирования размер переменной типа int соответствует размеру целого машинного слова. Например, в компиляторах для 16-разрядных процессоров переменная типа int имеет размер 2 байта. В этом случае знаковые значения этой переменной могут лежать в диапазоне от -32768 до 32767.
3. Ключевое слово float позволяет определить переменные вещественного типа. Их значения имеют дробную часть, отделяемую точкой, например: -5.6, 31.28 и т.п. Вещественные числа могут быть записаны также в форме с плавающей точкой, например: -1.09e+4. Число перед символом "е" называется мантиссой, а после "е" - порядком. Переменная типа float занимает в памяти 32 бита. Она может принимать значения в диапазоне от 3.4е-38 до 3.4e+38.
4. Ключевое слово double позволяет определить вещественную переменную двойной точности. Она занимает в памяти в два раза больше места, чем переменная типа float (т.е. ее размер 64 бита). Переменная типа double может принимать значения в диапазоне от 1.7e-308 до 1.7e+308.
5. Ключевое слово void (не имеющий значения) используется для нейтрализации значения объекта, например, для объявления функции, не возвращающей никаких значений.

Объект некоторого базового типа может быть модифицирован. С этой целью используются специальные ключевые слова, называемые модификаторами. В стандарте ANSI языка Си имеются следующие модификаторы типа:

* unsigned
* signed
* short
* long

Модификаторы записываются перед спецификаторами типа, например: unsigned char. Если после модификатора опущен спецификатор, то компилятор предполагает, что этим спецификатором является int. Таким образом, следующие строки:

long а;

long int а;

являются идентичными и определяют объект а как длинный целый. Табл. 1 иллюстрирует возможные сочетания модификаторов (unsigned, signed, short, long) со спецификаторами (char, int, float и double), а также показывает размер и диапазон значений объекта (для 16-разрядных компиляторов).

Таблица 1

|  |  |  |
| --- | --- | --- |
| Тип | Размер в байтах (битах) | Интервал изменения |
| char | 1 (8) | от -128 до 127 |
| unsigned char | 1 (8) | от 0 до 255 |
| signed char | 1 (8) | от -128 до 127 |
| int | 2 (16) | от -32768 до 32767 |
| unsigned int | 2 (16) | от 0 до 65535 |
| signed int | 2 (16) | от -32768 до 32767 |
| short int | 2 (16) | от -32768 до 32767 |
| unsigned short int | 2 (16) | от 0 до 65535 |
| signed short int | 2 (16) | от -32768 до 32767 |
| long int | 4 (32) | от -2147483648 до 2147483647 |
| unsigned long int | 4 (32) | от 0 до 4294967295 |
| signed long int | 4 (32) | от -2147483648 до 2147483647 |
| float | 4 (32) | от 3.4Е-38 до 3.4Е+38 |
| double | 8 (64) | от 1.7Е-308 до 1.7Е+308 |
| long double | 10 (80) | от 3.4Е-4932 до 3.4Е+4932 |

|  |
| --- |
|  |

***Переменные и константы***

Все переменные до их использования должны быть определены (объявлены). При этом задается тип, а затем идет список из одной или более переменных этого типа, разделенных запятыми. Например:

int a, b, c;

char x, y;

В языке различают понятия объявления переменной и ее определения. Объявление устанавливает свойства объекта: его тип (например, целый), размер (например, 4 байта) и т.д. Определение наряду с этим вызывает выделение памяти (в приведенном примере дано определение переменных).

Переменные можно разделять по строкам произвольным образом, например:

float a;

float b;

Переменные в языке Си могут быть инициализированы при их определении:

int a = 25, h = 6;

char g = 'Q', k = 'm';

float r = 1.89;

long double n = r\*123;

Выясним теперь, где в тексте программы определяются данные. В языке возможны глобальные и локальные объекты. Первые определяются вне функций и, следовательно, доступны для любой из них. Локальные объекты по отношению к функциям являются внутренними. Они начинают существовать, при входе в функцию и уничтожаются после выхода из нее. Ниже показана структура программы на Си и возможные места в программе, где определяются глобальные и локальные объекты.

int a; /\* Определение глобальной переменной \*/

int function (int b, char c); /\* Объявление функции (т.е. описание

ее заголовка)\*/

void main (void)

{ //Тело программы

int d, e; //Определение локальных переменных

float f; //Определение локальной переменной

...

}

int function (int b, char c) /\* Определение функции и формальных

параметров (по существу - локальных

переменных) b и c \*/

{ //Тело функции

char g; //Определение локальной переменной

...

}

Отметим, что выполнение программы всегда начинается с вызова функции main( ), которая содержит тело программы. Тело программы, как и тело любой другой функции, помещается между открывающей и закрывающей фигурными скобками.

В языке Си все определения должны следовать перед операторами, составляющими тело функции. В языке Си++ это ограничение снято и определения могут находиться в любом месте программы. Если они сделаны в функции, то соответствующие объекты будут локальными, а если вне функций, то глобальными.

Наряду с переменными в языке существуют следующие виды констант:

* *вещественные,* например 123.456, 5.61е-4. Они могут снабжаться суффиксом F (или f), например 123.456F, 5.61e-4f;
* *целые,* например 125;
* *короткие целые,* в конце записи которых добавляется буква (суффикс) H (или h), например 275h, 344H;
* *длинные целые,* в конце записи которых добавляется буква (суффикс) L (или l), например 361327L;
* *беззнаковые,* в конце записи которых добавляется буква U (или u), например 62125U;
* *восьмеричные,* в которых перед первой значащей цифрой записывается нуль (0), например 071;
* *шестнадцатеричные,* в которых перед первой значащей цифрой записывается пара символов нуль-икс (0x), например 0x5F;
* *символьные* - единственный символ, заключенный в одинарные кавычки, например 'О', '2', '.' и т.п. Символы, не имеющие графического представления, можно записывать, используя специальные комбинации, например \n (код 10), \0 (код 0). Эти комбинации выглядят как два символа, хотя фактически это один символ. Так же можно представить любой двоичный образ одного байта: '\NNN', где NNN - от одной до трех восьмеричных цифр. Допускается и шестнадцатеричное задание кодов символов, которое представляется в виде: '\х2В', '\хЗ6' и т.п.;
* *строковые* - последовательность из нуля символов и более, заключенная в двойные кавычки, например: "Это строковая константа". Кавычки не входят в строку, а лишь ограничивают ее. Строка представляет собой массив из перечисленных элементов, в конце которого помещается байт с символом '\0'. Таким образом, число байтов, необходимых для хранения строки, на единицу превышает число символов между двойными кавычками;
* *константное выражение,* состоящее из одних констант, которое вычисляется во время трансляции (например: а=60+301);
* типа *long double,* в конце записи которых добавляется буква L (или l), например: 1234567.89L.

|  |
| --- |
|  |

***Как вводить и выводить информацию***

Операции ввода/вывода в языке Си организованы посредством библиотечных функций (причем их довольно много).

Самый простой механизм ввода - чтение по одному символу из стандартного входного потока (с клавиатуры) с помощью функции getchar( ). Она имеет следующий прототип (т.е. описание заголовка):

int getchar(void);

Здесь определен тип единственного аргумента (void) и тип возвращаемого функцией значения (int).

Оператор вида:

х = getchar( );

присваивает переменной х очередной вводимый символ. Переменная х должна иметь символьный или целый тип.

Другая функция - putchar(х) выдает значение переменной x в стандартный выходной поток (на экран дисплея). Функция putchar( ) имеет прототип:

int putchar(int);

Объявления getchar( ) и putchar( ) сделаны в заголовочном файле stdio.h, содержащем описания заголовков библиотечных функций стандартного ввода/вывода. Чтобы библиотечные функции стали доступны программе, к ней необходимо подключить данный файл. Подключение осуществляется с помощью директивы препроцессора

#include <stdio.h>

помещаемой в начало программы (подробнее см. в разделе 5).

Заметим, что для функции getchar( ) после выбора символа необходимо нажать клавишу <Enter>. Иногда это создает определенные неудобства. Функции getch( ) и getche( ) устраняют их. Они имеют следующие прототипы:

int getch(void);

int getche(void);

Обе эти функции вводят символ сразу же после нажатия соответствующей клавиши (здесь не надо дополнительно нажимать клавишу <Enter>). Отличие между ними заключается в том, что getche( ) отображает вводимый символ на экране дисплея, а getch( ) - нет. Прототипы этих функций содержатся в файле conio.h (консольный ввод/вывод). Для их использования файл conio.h также следует подключить к программе с помощью директивы #include .

|  |
| --- |
|  |

***Форматированный вывод данных***

Функция printf( ) (прототип содержится в файле stdio.h) обеспечивает форматированный вывод. Ее можно записать в следующем формальном виде:

рrintf ("управляющая строка", аргумент \_1, аргумент \_2,...);

Управляющая строка содержит компоненты трех типов: обычные символы, которые просто копируются в стандартный выходной поток (выводятся на экран дисплея); спецификации преобразования, каждая из которых вызывает вывод на экран очередного аргумента из последующего списка; управляющие символьные константы.

Каждая спецификация преобразования начинается со знака % и заканчивается некоторым символом, задающим преобразование. Между знаком % и символом преобразования могут встречаться другие знаки в соответствии со следующим форматом:

% [признаки] [ширина\_поля] [точность] [F|N|h|l|L] c\_n

Все параметры в квадратных скобках не являются обязательными.

На месте параметра c\_n (символ преобразования) могут быть записаны:

**с** - значением аргумента является символ;  
**d** или **i** - значением аргумента является десятичное целое число;   
**е** - значением аргумента является вещественное десятичное число в экспоненциальной форме вида 1.23e+2;  
**Е** - значением аргумента является вещественное десятичное число в экспоненциальной форме вида 1.23E+2;  
**f** - значением аргумента является вещественное десятичное число с плавающей точкой;  
**g** (или **G**) - используется, как е или f, и исключает вывод незначащих нулей;  
**о** - значением аргумента является восьмеричное целое число;  
**s** - значением аргумента является строка символов (символы строки выводятся до тех пор, пока не встретится символ конца строки или же не будет, выведено число символов, заданное точностью);  
**u** - значением аргумента является беззнаковое целое число;  
**х** - значением аргумента является шестнадцатеричное целое число с цифрами 0,..., 9, а, b, с, d, е, f;  
**X** - значением аргумента является шестнадцатеричное целое число с цифрами 0,..., 9, А, В, С, О, Е, F;  
**р** - значением аргумента является указатель;  
**n** - применяется в операциях форматирования. Аргумент, соответствующий этому символу спецификации, должен быть указателем на целое. В него возвращается номер позиции строки (отображаемой на экране), в которой записана спецификация %n.

Необязательные параметры в спецификации преобразования:

* признак минус (-) указывает, что преобразованный параметр должен быть выровнен влево в своем поле;
* признак плюс (+) требует вывода результата со знаком;
* строка цифр, задающая минимальный размер поля (ширина поля). Здесь может так же использоваться символ \*, который тоже позволяет задать минимальную ширину поля и точность представления выводимого числа;
* точка (.), отделяющая размер поля от последующей строки цифр;
* строка цифр, задающая максимальное число выводимых символов, или же количество цифр, выводимых справа от десятичной точки в значениях типов float или double (точность);
* символ F, определяющий указатель типа far;
* символ N, определяющий указатель типа near;
* символ h, определяющий аргумент типа short int (используется вместе с символами преобразования d, i, о, u, х, Х);
* символ l, указывающий, что соответствующий аргумент имеет тип long (в случае символов преобразования d, i, о, u, х, X) или double (в случае символов преобразования е, Е, f, g, G);
* символ L, указывающий, что соответствующий аргумент имеет тип long double (используется вместе с символами преобразований е, Е, f, g, G);
* символ #, который может встречаться перед символами преобразования g, f, е и перед символом х. В первом случае всегда будет выводиться десятичная точка, а во втором - префикс 0x перед соответствующим шестнадцатеричным числом.

Если после знака % записан не символ преобразования, то он выводится на экран. Таким образом, строка %% приводит к выводу на экран знака %.

Функция printf( ) использует управляющую строку, чтобы определить, сколько всего аргументов и каковы их типы. Аргументами могут быть переменные, константы, выражения, вызовы функций; главное, чтобы их значения соответствовали заданной спецификации.

При наличии ошибок, например, в числе аргументов или типе преобразования, результаты будут неверными.

Среди управляющих символьных констант наиболее часто используются следующие:

\а - для кратковременной подачи звукового сигнала;  
\b - для перевода курсора влево на одну позицию;  
\f - для подачи формата;  
\n - для перехода на новую строку;  
\r - для возврата каретки;  
\t - горизонтальная табуляция;  
\v - вертикальная табуляция;  
\\ - вывод символа \;  
\' - вывод символа ' ;  
\" - вывод символа ";  
\? - вывод символа ?.

Например, в результате вызова функции:

printf("\tComputer\n%d\n", i);

сначала выполняется горизонтальная табуляция (\t), т.е. курсор сместится от края экрана, затем на экран будет выведено слово Computer, после этого курсор переместится в начало следующей строки (\n), затем будет выведено целое число i по формату %d (десятичное целое), и, окончательно, курсор перейдет в начало новой строки (\n).

Напечатать строку символов можно и так:

printf("Это строка символов");

|  |
| --- |
|  |

***Форматированный ввод данных***

Функция scanf( ) (прототип содержится в файле stdio.h) обеспечивает форматированный ввод. Ее можно записать в следующем формальном виде:

scanf("управляющая строка", аргумент\_1, аргумент\_2,...);

Аргументы scanf( ) должны быть указателями на соответствующие значения. Для этого перед именем переменной записывается символ &. Назначение указателей будет рассмотрено далее.

Управляющая строка содержит спецификации преобразования и используется для установления количества и типов аргументов. В нее могут включаться:

* пробелы, символы табуляции и перехода на новую строку (все они игнорируются);
* спецификации преобразования, состоящие из знака %, возможно, символа \* (запрещение присваивания), возможно, числа, задающего максимальный размер поля, и самого символа преобразования;
* обычные символы, кроме % (считается, что они должны совпадать с очередными неизвестными символами во входном потоке).

Рассмотрим символы преобразования функции scanf( ) (указываются после символа %):

**с** - на входе ожидается появление одиночного символа;  
**d** или **i** - на входе ожидается десятичное целое число и аргумент является указателем на переменную типа int;  
**D** или **l** - на входе ожидается десятичное целое число и аргумент является указателем на переменную типа long;  
**е** или **Е** - на входе ожидается вещественное число с плавающей точкой;  
**f** - на входе ожидается вещественное число с плавающей точкой;  
**g** или **G** - на входе ожидается вещественное число с плавающей точкой;  
**о** - на входе ожидается восьмеричное целое число и аргумент является указателем на переменную типа int;  
**О** - на входе ожидается восьмеричное целое число и аргумент является указателем на переменную типа long;  
**s** - на входе ожидается появление строки символов;  
**х** - на входе ожидается шестнадцатеричное целое число и аргумент является указателем на переменную типа int;  
**Х** - на входе ожидается шестнадцатеричное целое число и аргумент является указателем на переменную типа long;  
**р** - на входе ожидается появление указателя в виде шестнадцатеричного числа;  
**n** - применяется в операциях форматирования. Аргумент, соответствующий этому символу спецификации, должен быть указателем на целое. В него возвращается номер позиции (после ввода), в которой записана спецификация %n;  
**u** - на входе ожидается беззнаковое целое число и аргумент является указателем на переменную типа unsigned int;  
**U** - на входе ожидается беззнаковое целое число и аргумент является указателем на переменную типа unsigned long;  
**[ ]** - сканирует входную строку для получения символов.

Перед некоторыми символами преобразования могут записываться следующие модификаторы:

**F** - изменяет указатель, заданный по умолчанию, на указатель типа far;  
**N** - изменяет указатель, заданный по умолчанию, на указатель типа near;  
**h** - преобразует аргумент к типу short int (может записываться перед символами d, i, о, u, х);  
**l** - преобразует аргумент к типу long int (может записываться перед символами d, i, o, u, x);  
**L** - преобразует аргумент к типу long double (может записываться перед символами е, f, g).

Ввести целое число (int a;), символ (char b;) и вещественное число (float t;) можно так:

scanf("%d", &a);

scanf("%c", &b);

scanf("%d%c%f",&a, &b, &t);

|  |
| --- |
|  |

**РАЗДЕЛ 2. ОПЕРАЦИИ И ОПЕРАТОРЫ**

***Операции языка Си***

Любое выражение языка состоит из операндов (переменных, констант и др.), соединенных знаками операций. Знак операции - это символ или группа символов, которые сообщают компилятору о необходимости выполнения определенных арифметических, логических или других действий.

Операции выполняются в строгой последовательности. Величина, определяющая преимущественное право на выполнение той или иной операции, называется приоритетом. В табл. 2 перечислены различные операции языка Си. Их приоритеты для каждой группы одинаковы (группы выделены цветом). Чем большим преимуществом пользуется соответствующая группа операций, тем выше она расположена в таблице. Порядок выполнения операций может регулироваться с помощью круглых скобок.

Таблица 2

|  |  |
| --- | --- |
| Знак операции | Назначение операции |
| ( ) | Вызов функции |
| [ ] | Выделение элемента массива |
| **.** | Выделение элемента записи |
| -> | Выделение элемента записи |
| ! | Логическое отрицание |
| ~ | Поразрядное отрицание |
| - | Изменение знака |
| ++ | Увеличение на единицу |
| -- | Уменьшение на единицу |
| & | Взятие адреса |
| \* | Обращение по адресу |
| (тип) | Преобразование типа (т.е. (float) a) |
| sizeof( ) | Определение размера в байтах |
| \* | Умножение |
| / | Деление |
| % | Определение остатка от деления |
| + | Сложение |
| - | Вычитание |
| << | Сдвиг влево |
| >> | Сдвиг вправо |
| < | Меньше, чем |
| <= | Меньше или равно |
| > | Больше, чем |
| >= | Больше или равно |
| = = | Равно |
| != | Не равно |
| & | Поразрядное логическое "И" |
| ^ | Поразрядное исключающее "ИЛИ" |
| | | Поразрядное логическое "ИЛИ" |
| && | Логическое "И" |
| || | Логическое "ИЛИ" |
| ?: | Условная (тернарная) операция |
| = | Присваивание |
| +=, - =, \*=, /=, %=, <<=, >>=, &=, |=, ^= | Составные операции присваивания (например, а \*= b (т.е. a = a \* b) и т.д.) |
| **,** | Операция запятая |

Для исключения путаницы в понятиях "операция" и "оператор", отметим, что оператор - это наименьшая исполняемая единица программы. Различают операторы выражения, действие которых состоит в вычислении заданных выражений (например: a = sin(b)+c; j++;), операторы объявления, составные операторы, пустые операторы, операторы метки, цикла и т.д. Для обозначения конца оператора в языке Си используется точка с запятой. Что касается составного оператора (или блока), представляющего собой набор логически связанных операторов, помещенных между открывающей ({) и закрывающей (}) фигурными скобками ("операторными скобками"), то за ним точка с запятой не ставится. Отметим, что блок отличается от составного оператора наличием определений в теле блока.

Охарактеризуем основные операции языка Си. Сначала рассмотрим одну из них - операцию присваивания (=). Выражение вида

х = у;

присваивает переменной х значение переменной у. Операцию "=" разрешается использовать многократно в одном выражении, например:

x = y = z = 100;

Различают *унарные* и *бинарные операции*. У первых из них один операнд, а у вторых - два. Начнем их рассмотрение с операций, отнесенных к первой из следующих традиционных групп:

1. Арифметические операции.
2. Логические операции и операции отношения.
3. Операции с битами.

Арифметические операции задаются следующими символами (табл. 2): +, -, \*, /, %. Последнюю из них нельзя применять к переменным вещественного типа. Например:

a = b + c;

x = y - z;

r = t \* v;

s = k / l;

p = q % w;

Логические операции отношения задаются следующими символами (см. табл. 2): && ("И"), || ("ИЛИ"), ! ("НЕ"), >, >=, <, <= , = = (равно), != (не равно). Традиционно эти операции должны давать одно из двух значений: истину или ложь. В языке Си принято следующее правило: истина - это любое ненулевое значение; ложь - это нулевое значение. Выражения, использующие логические операции и операции отношения, возвращают 0 для ложного значения и 1 для истинного. Ниже приводится таблица истинности для логических операций.

Таблица 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | x&&y | x||y | !x |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

Битовые операции можно применять к переменным, имеющим типы int, char, а также их вариантам (например, long int). Их нельзя применять к переменным типов float, double, void (или более сложных типов). Эти операции задаются следующими символами: ~ (поразрядное отрицание), << (сдвиг влево), >> (сдвиг вправо), & (поразрядное "И"), ^ (поразрядное исключающее "ИЛИ"), | (поразрядное "ИЛИ").

Примеры: если a = 0000 1111 и b = 1000 1000, то

~a = 1111 0000,

a << 1 = 0001 1110,

a >> 1 = 0000 0111,

a & b = 0000 1000,

a ^ b = 1000 0111,

a | b = 1000 1111.

В языке предусмотрены две нетрадиционные операции инкремента (++) и декремента (--). Они предназначены для увеличения и уменьшения на единицу значения операнда. Операции ++ и -- можно записывать как перед операндом, так и после него. В первом случае (++n или --n) значение операнда (n) изменяется перед его использованием в соответствующем выражении, а во втором (n++ или n--) - после его использования. Рассмотрим две следующие строки программы:

a = b + c++;

a1 = b1 + ++c1;

Предположим, что b = b1 = 2, c = c1 = 4. Тогда после выполнения операций: a = 6, b = 2, c = 5, a1 = 7, b1 = 2, c1 = 5.

Широкое распространение находят также выражения с еще одной нетрадиционной тернарной или условной операцией ?:. В формуле

y = x ? a: b;

y = a, если x не равно нулю (т.е. истинно), и y = b, если х равно нулю (ложно). Следующее выражение

y = (a>b) ? a: b;

позволяет присвоить переменной у значение большей переменной (а или b), т.е. y = max(a, b).

Еще одним отличием языка является то, что выражение вида а = а + 5; можно записать в другой форме: a += 5;. Вместо знака + можно использовать и символы других бинарных операций (см. табл. 2).

Другие операции из табл. 2 будут описаны в последующих параграфах.

|  |
| --- |
|  |

***Преобразование типов***

Если в выражении появляются операнды различных типов, то они преобразуются к некоторому общему типу, при этом к каждому арифметическому операнду применяется такая последовательность правил:

1. Если один из операндов в выражении имеет тип long double, то остальные тоже преобразуются к типу long double.
2. В противном случае, если один из операндов в выражении имеет тип double, то остальные тоже преобразуются к типу double.
3. В противном случае, если один из операндов в выражении имеет тип float, то остальные тоже преобразуются к типу float.
4. В противном случае, если один из операндов в выражении имеет тип unsigned long, то остальные тоже преобразуются к типу unsigned long.
5. В противном случае, если один из операндов в выражении имеет тип long, то остальные тоже преобразуются к типу long.
6. В противном случае, если один из операндов в выражении имеет тип unsigned, то остальные тоже преобразуются. к типу unsigned.
7. В противном случае все операнды преобразуются к типу int. При этом тип char преобразуется в int со знаком; тип unsigned char в int, у которого старший байт всегда нулевой; тип signed char в int, у которого в знаковый разряд передается знак из сhar; тип short в int (знаковый или беззнаковый).

Предположим, что вычислено значение некоторого выражения в правой части оператора присваивания. В левой части оператора присваивания записана некоторая переменная, причем ее тип отличается от типа результата в правой части. Здесь правила преобразования очень простые: значение справа от оператора присваивания преобразуется к типу переменной слева от оператора присваивания. Если размер результата в правой части больше размера операнда в левой части, то старшая часть этого результата будет потеряна.

В языке Си можно явно указать тип любого выражения. Для этого используется операция преобразования ("приведения") типа. Она применяется следующим образом:

(тип) выражение

(здесь можно указать любой допустимый в языке Си тип).

Рассмотрим пример:

int a = 30000;

float b;

........

b = (float) a \* 12;

(переменная a целого типа явно преобразована к типу float; если этого не сделать, то результат будет потерян, т.к. a \* 12 > 32767).

Преобразование типа также может использоваться для преобразования типов аргументов при вызове функций.

|  |
| --- |
|  |

***Указатели и операции с ними***

Указатели - это переменные, показывающие место или адрес памяти, где расположены другие объекты (переменные, функции и др.). Так как указатель содержит адрес некоторого объекта, то через него можно обращаться к этому объекту.

Унарная операция & дает адрес объекта, поэтому оператор

у = &х;

присваивает адрес переменной х переменной у. Операцию & нельзя применять к константам и выражениям; конструкции вида &(х+7) или &28 недопустимы.

Унарная операция \* воспринимает свой операнд как адрес некоторого объекта и использует этот адрес для выборки содержимого, поэтому оператор

z = \*y;

присваивает z значение переменной, записанной по адресу у. Если

y = &x;

z = \*у;

то z = x.

Объекты, состоящие из знака \* и адреса (например, \*а), необходимо определить. Делается это, например, так:

int \*а, \*b, \*с;

char \*d;

Определение вида char \*d говорит о том, что значение, записанное по адресу d, имеет тип char.

Указатели могут встречаться и в выражениях. Если у - указатель на целое, т.е. имело место объявление int \*у, то \*у может появиться там же, где и любая другая переменная, не являющаяся указателем. Таким образом, следующие выражения вполне допустимы:

\*у = 7;

\*x \*=5;

(\*z)++;

Первое из них заносит число 7 в ячейку памяти по адресу у, второе увеличивает значение по адресу х в пять раз, третье добавляет единицу к содержимому ячейки памяти с адресом z. В последнем случае круглые скобки необходимы, так как операции с одинаковым приоритетом выполняются справа налево. В результате если, например, \*z = 5, то (\*z)++ приведет к тому, что \*z = 6, а \*z++ всего лишь изменит сам адрес z (операция ++ выполняется над адресом z, а не над значением \*z по этому адресу).

Указатели можно использовать как операнды в арифметических операциях. Если у - указатель, то унарная операция y++ увеличивает его значение; теперь оно является адресом следующего элемента. Указатели и целые числа можно складывать. Конструкция у + n (у - указатель, n - целое число) задает адрес n-гo объекта, на который указывает у. Это справедливо для любых объектов (int, char, float и др.); транслятор будет масштабировать приращение адреса в соответствии с типом, указанным в определении объекта.

Любой адрес можно проверить на равенство (==) или неравенство (!=) со специальным значением NULL, которое позволяет определить ничего не адресующий указатель.

|  |
| --- |
|  |

***Операторы цикла***

Циклы организуются, чтобы выполнить некоторый оператор или группу операторов определенное число раз. В языке Си три оператора цикла: for, while и do - while. Первый из них формально записывается, в следующем виде:

for (выражение\_1; выражение\_2; выражение\_3) тело\_цикла

Тело цикла составляет либо один оператор, либо несколько операторов, заключенных в фигурные скобки { ... } (после блока точка с запятой не ставится). В выражениях 1, 2, 3 фигурирует специальная переменная, называемая управляющей. По ее значению устанавливается необходимость повторения цикла или выхода из него.

Выражение\_1 присваивает начальное значение управляющей переменной, выражение\_З изменяет его на каждом шаге, а выражение\_2 проверяет, не достигло ли оно граничного значения, устанавливающего необходимость выхода из цикла.

Примеры:

for (i = 1; i < 10; i++)

{ ...

}

for (сh = 'a'; ch != 'p';) scanf ("%c", &ch);

/\* Цикл будет выполняться до тех пор, пока с клавиатуры

не будет введен символ 'p' \*/

Любое из трех выражений в цикле for может отсутствовать, однако точка с запятой должна оставаться. Таким образом, for ( ; ; ) {...} - это бесконечный цикл, из которого можно выйти лишь другими способами.

В языке Си принято следующее правило. Любое выражение с операцией присваивания, заключенное в круглые скобки, имеет значение, равное присваиваемому. Например, выражение (а=7+2) имеет значение 9. После этого можно записать другое выражение, например: ((а=7+2)<10), которое в данном случае будет всегда давать истинное значение. Следующая конструкция:

((сh = getch( )) == 'i')

позволяет вводить значение переменной сh и давать истинный результат только тогда, когда введенным значением является буква 'i'. В скобках можно записывать и несколько формул, составляющих сложное выражение. Для этих целей используется операция запятая. Формулы будут вычисляться слева направо, и все выражение примет значение последней вычисленной формулы. Например, если имеются две переменные типа char, то выражение

z = (х = у, у = getch( ));

определяет следующие действия: значение переменной у присваивается переменной х; вводится символ с клавиатуры и присваивается переменной у; z получает значение переменной у. Скобки здесь необходимы, поскольку операция запятая имеет более низкий приоритет, чем операция присваивания, записанная после переменной z. Операция запятая находит широкое применение для построения выражений цикла for и позволяет параллельно изменять значения нескольких управляющих переменных.

Допускаются вложенные конструкции, т.е. в теле некоторого цикла могут встречаться другие операторы for.

Оператор while формально записывается в таком виде:

while (выражение) тело\_цикла

Выражение в скобках может принимать ненулевое (истинное) или нулевое (ложное) значение. Если оно истинно, то выполняется тело цикла и выражение вычисляется снова. Если выражение ложно, то цикл while заканчивается.

Оператор do-while формально записывается следующим образом:

do {тело\_цикла} while (выражение);

Основным отличием между циклами while и do - while является то, что тело в цикле do - while выполняется по крайней мере один раз. Тело цикла будет выполняться до тех пор, пока выражение в скобках не примет ложное значение. Если оно ложно при входе в цикл, то его тело выполняется ровно один раз.

Допускается вложенность одних циклов в другие, т.е. в теле любого цикла могут появляться операторы for, while и do - while.

В теле цикла могут использоваться новые операторы break и continue. Оператор break обеспечивает немедленный выход из цикла, оператор continue вызывает прекращение очередной и начало следующей итерации.

|  |
| --- |
|  |

***Операторы условных и безусловных переходов***

Для организации условных и безусловных переходов в программе на языке Си используются операторы: if - else, switch и goto. Первый из них записывается следующим образом:

if (проверка\_условия) оператор\_1; else оператор\_2;

Если условие в скобках принимает истинное значение, выполняется оператор\_1, если ложное - оператор\_2. Если вместо одного необходимо выполнить несколько операторов, то они заключаются в фигурные скобки. В операторе if слово else может отсутствовать.

В операторе if - else непосредственно после ключевых слов if и else должны следовать другие операторы. Если хотя бы один из них является оператором if, его называют вложенным. Согласно принятому в языке Си соглашению слово else всегда относится к ближайшему предшествующему ему if.

Оператор switch позволяет выбрать одну из нескольких альтернатив. Он записывается в следующем формальном виде:

switch (выражение)

{

case константа\_1: операторы\_1;

break;

case константа\_2: операторы\_2;

break;

........ ........

default: операторы\_default;

}

Здесь вычисляется значение целого выражения в скобках (его иногда называют селектором) и оно сравнивается со всеми константами (константными выражениями). Все константы должны быть различными. При совпадении выполнится соответствующий вариант операторов (один или несколько операторов). Вариант с ключевым словом default реализуется, если ни один другой не подошел (слово default может и отсутствовать). Если default отсутствует, а все результаты сравнения отрицательны, то ни один вариант не выполняется.

Для прекращения последующих проверок после успешного выбора некоторого варианта используется оператор break, обеспечивающий немедленный выход из переключателя switch.

Допускаются вложенные конструкции switch.

Рассмотрим правила выполнения безусловного перехода, который можно представить в следующей форме:

goto метка;

Метка - это любой идентификатор, после которого поставлено двоеточие. Оператор goto указывает на то, что выполнение программы необходимо продолжить начиная с оператора, перед которым записана метка. Метку можно поставить перед любым оператором в той функции, где находится соответствующий ей оператор goto. Ее не надо объявлять.

|  |
| --- |
|  |

**РАЗДЕЛ 3. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ**

***Массивы***

Массив состоит из элементов одного и того же типа. Ко всему массиву целиком можно обращаться по имени. Кроме того, можно выбирать любой элемент массива. Для этого необходимо задать индекс, который указывает на его относительную позицию. Число элементов массива назначается при его определении и в дальнейшем не изменяется. Если массив объявлен, то к любому его элементу можно обратиться следующим образом: указать имя массива и индекс элемента в квадратных скобках. Массивы определяются так же, как и переменные:

int a[100];

char b[20];

float d[50];

В первой строке объявлен массив а из 100 элементов целого типа: а[0], а[1], ..., а[99] (индексация всегда начинается с нуля). Во второй строке элементы массива b имеют тип char, а в третьей - float.

Двумерный массив представляется как одномерный, элементами которого так же являются массивы. Например, определение char а[10][20]; задает такой массив. По аналогии можно установить и большее число измерений. Элементы двумерного массива хранятся по строкам, т.е. если проходить по ним в порядке их расположения в памяти, то быстрее всего изменяется самый правый индекс. Например, обращение к девятому элементу пятой строки запишется так: а[5][9].

Пусть задан массив:

int a[2][3];

Тогда элементы массива а будут размещаться в памяти следующим образом: a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2].

Имя массива - это константа, которая содержит адрес его первого элемента (в данном примере а содержит адрес элемента а[0][0]). Предположим, что a = 1000. Тогда адрес элемента а[0][1] будет равен 1002 (элемент типа int занимает в памяти 2 байта), адрес следующего элемента а[0][2] - 1004 и т.д. Что же произойдет, если выбрать элемент, для которого не выделена память? К сожалению, компилятор не отслеживает данной ситуации. В результате возникнет ошибка и программа будет работать неправильно.

В языке Си существует сильная взаимосвязь между указателями и массивами. Любое действие, которое достигается индексированием массива, можно выполнить и с помощью указателей, причем последний вариант будет работать быстрее.

Определение

int a[5];

задает массив из пяти элементов а[0], a[1], a[2], a[3], a[4]. Если объект \*у определен как

int \*у;

то оператор у = &a[0]; присваивает переменной у адрес элемента а[0]. Если переменная у указывает на очередной элемент массива а, то y+1 указывает на следующий элемент, причем здесь выполняется соответствующее масштабирование для приращения адреса с учетом длины объекта (для типа int - 2 байта, long - 4 байта, (double - 8 байт и т.д.).

Так как само имя массива есть адрес его нулевого элемента, то оператор у = &a[0]; можно записать и в другом виде: у = а. Тогда элемент а[1] можно представить как \*(а+1). С другой стороны, если у - указатель на массив a, то следующие две записи: a[i] и \*(у+i) - эквивалентны.

Между именем массива и соответствующим указателем есть одно важное различие. Указатель - это переменная и у = а; или y++; - допустимые операции. Имя же массива - константа, поэтому конструкции вида a = y; a++; использовать нельзя, так как значение константы постоянно и не может быть изменено.

Переменные с адресами могут образовывать некоторую иерархическую структуру (могут быть многоуровневыми) типа указатель на указатель (т.е. значение указателя является адресом другого указателя), указатель на указатель на указатель и т.д. Если указатели адресуют элементы одного массива, то их можно сравнивать (отношения вида <, >, = =, != и другие работают правильно). В то же время нельзя сравнивать ли6о использовать в арифметических операциях указатели на разные массивы (соответствующие выражения не приводят к ошибкам при компиляции, но в большинстве случаев не имеют смысла). Любой адрес можно проверить на равенство или неравенство с константой NULL. Указатели на элементы одного массива можно также вычитать. Тогда результатом будет число элементов массива, расположенных между уменьшаемым и вычитаемым объектами.

Язык Си позволяет инициализировать массив при его определении. Для этого используется следующая форма:

тип имя\_массива[...] ... [...] = {список значений};

Примеры:

int a[5] = {0, 1, 2, 3, 4};

char ch[3] = {'d', 'e', '9'};

int b[2][3] = {1, 2, 3, 4, 5, 6};

В последнем случае: b[0][0] = 1, b[0][1] = 2, b[0][2] = 3, b[1][0] = 4, b[1][1] = 5, b[1][2] = 6.

В языке допускаются массивы указателей, которые определяются, например, следующим образом: char \*m[5];. Здесь m[5] - массив, содержащий адреса элементов типа char.

|  |
| --- |
|  |

***Строки символов***

Язык Си не поддерживает отдельный строковый тип данных, но он позволяет определить строки двумя различными способами. В первом используется массив символов, а во втором - указатель на первый символ массива.

Определение char а[10]; указывает компилятору на необходимость резервирования места для максимум 10 символов. Константа а содержит адрес ячейки памяти, в которой помещено значение первого из десяти объектов типа char. Процедуры, связанные с занесением конкретной строки в массив а, копируют ее по одному символу в область памяти, на которую указывает константа а, до тех пор, пока не будет скопирован нулевой символ, оканчивающий строку. Когда выполняется функция типа printf("%s", а), ей передается значение а, т.е. адрес первого символа, на который указывает а. Если первый символ - нулевой, то работа функции printf() заканчивается, а если нет, то она выводит его на экран, прибавляет к адресу единицу и снова начинает проверку на нулевой символ. Такая обработка позволяет снять ограничения на длину строки (конечно, в пределах объявленной размерности): строка может иметь любую длину, но в пределах доступной памяти.

Инициализировать строку при таком способе определения можно следующим образом:

char array[7] = "Строка";

char s[ ] = {'С', 'т', 'р', 'о', 'к', 'а', '\0'};

(при определении массива с одновременной инициализацией пределы изменения индекса можно не указывать).

Второй способ определения строки - это использование указателя на символ. Определение char \*b; задает переменную b, которая может содержать адрес некоторого объекта. Однако в данном случае компилятор не резервирует место для хранения символов и не инициализирует переменную b конкретным значением. Когда компилятор встречает оператор вида b ="IBM PC";, он производит следующие действия. Во-первых, как и в предыдущем случае, он создает в каком-либо месте объектного модуля строку "IBM PC", за которой следует нулевой символ ('\0'). Во-вторых, он присваивает значение начального адреса этой строки (адрес символа 'I') переменной b. Функция printf("%s", b) работает так же, как и в предыдущем случае, осуществляя вывод символов до тех пор, пока не встретится заключительный нуль.

Массив указателей можно инициализировать, т.е. назначать его элементам конкретные адреса некоторых заданных строк при определении.

Для ввода и вывода строк символов помимо scanf( ) и printf() могут использоваться функции gets( ) и puts( ) (их прототипы находятся в файле stdio.h).

Если string - массив символов, то ввести строку с клавиатуры можно так:

gets(string);

(ввод оканчивается нажатием клавиши <Enter>). Вывести строку на экран можно следующим образом:

puts(string);

Отметим также, что для работы со строками существует специальная библиотека функций, прототипы которых находятся в файле string.h.

Наиболее часто используются функции strcpy( ), strcat( ), strlen( ) и strcmp( ).

Если string1 и string2 - массивы символов, то вызов функции strcpy( ) имеет вид:

strcpy(string1, string2);

Эта функция служит для копирования содержимого строки string2 в строку string1. Массив string1 должен быть достаточно большим, чтобы в него поместилась строка string2. Так как компилятор не отслеживает этой ситуации, то недостаток места приведет к потере данных.

Вызов функции strcat( ) имеет вид:

strcat(string1, string2);

Эта функция присоединяет строку string2 к строке string1 и помещает ее в массив, где находилась строка string1, при этом строка string2 не изменяется. Нулевой байт, который завершал первую строку, заменяется первым байтом второй строки.

Функция strlen( ) возвращает длину строки, при этом завершающий нулевой байт не учитывается. Если a - целое, то вызов функции имеет вид:

a = strlen(string);

Функция strcmp( ) сравнивает две строки и возвращает 0, если они равны.

|  |
| --- |
|  |

***Структуры***

Структура - это объединение одного или нескольких объектов (переменных, массивов, указателей, других структур и т.д.). Как и массив, она представляет собой совокупность данных. Отличием является то, что к ее элементам необходимо обращаться по имени и что различные элементы структуры не обязательно должны принадлежать одному типу.

Объявление структуры осуществляется с помощью ключевого слова struct, за которым идет ее тип и далее список элементов, заключенных в фигурные скобки:

struct тип { тип элемента\_1 имя элемента\_1;

.........

тип элемента\_n имя элемента\_n;

};

Именем элемента может быть любой идентификатор. Как и выше, в одной строке можно записывать через запятую несколько идентификаторов одного типа.

Рассмотрим пример:

sruct date { int day;

int month;

int year;

};

Следом за фигурной скобкой, заканчивающей список элементов, могут записываться переменные данного типа, например:

struct date {...} a, b, c;

(при этом выделяется соответствующая память). Описание без последующего списка не выделяет никакой памяти; оно просто задает форму структуры. Введенное имя типа позже можно использовать для объявления структуры, например:

struct date days;

Теперь переменная days имеет тип date.

При необходимости структуры можно инициализировать, помещая вслед за описанием список начальных значений элементов.

Разрешается вкладывать структуры друг в друга, например:

struct man { char name[20], fam[20];

struct date bd;

int age;

};

Определенный выше тип data включает три элемента: day, month, year, содержащий целые значения (int). Структура man включает элементы name, fam, bd и voz. Первые два - name[20] и fam[20] - это символьные массивы из 20 элементов каждый. Переменная bd представлена составным элементом (вложенной структурой) типа data. Элемент age содержит значения целого типа int). Теперь можно определить переменные, значения которых принадлежат введенному типу:

struct man man\_[100];

Здесь определен массив man\_, состоящий из 100 структур типа man.

Чтобы обратиться к отдельному элементу структуры, необходимо указать его имя, поставить точку и сразу же за ней записать имя нужного элемента, например:

man\_[j].age = 19;

man\_[j].bd.day = 24;

man\_[j].bd.month = 2

man\_[j].bd.year = 1987;

При работе со структурами необходимо помнить, что тип элемента определяется соответствующей строкой описания в фигурных скобках. Например, массив man\_ имеет тип man, year является целым числом и т.п. Поскольку каждый элемент структуры относится к определенному типу, его имя может появиться везде, где разрешено использование значений этого типа. Допускаются конструкции вида man\_[i]=man\_[j]; где man\_[i] и man\_[j] - объекты, соответствующие единому описанию структуры. Другими словами, разрешается присваивать одну структуру другой по их именам.

Унарная операция & позволяет взять адрес структуры. Предположим, что определена переменная day:

struct date {int d, m, у;} day;

Здесь day - это структура типа date, включающая три элемента: d, m, у. Другое определение

struct date \*db;

устанавливает тот факт, что db - это указатель на структуру типа date.

Запишем выражение:

db = &day;

В этом случае для выбора элементов d, m, у структуры необходимо использовать конструкции:

(\*db).d; (\*db).m; (\*db).y;

Действительно, db - это адрес структуры, \*db - сама структура. Круглые скобки здесь необходимы, так как точка имеет более высокий, чем звездочка, приоритет. Для аналогичных целей в языке Си предусмотрена специальная операция ->. Эта операция выбирает элемент структуры и позволяет представить рассмотренные выше конструкции в более простом виде:

db -> d; db -> m; db -> у;

|  |
| --- |
|  |

***Оператор typedef***

Рассмотрим описание структуры:

struct data {int d, m, у;};

Здесь фактически вводится новый тип данных - data. Теперь его можно использовать для объявления конкретных экземпляров структуры, например:

struct data а, b, с;

В язык Си введено специальное средство, позволяющее назначать имена типам данных (переименовывать). Таким средством является оператор typedef. Он записывается в следующем виде:

typedef тип имя;

Здесь "тип" - любой разрешенный тип данных и "имя" - любой разрешенный идентификатор.

Рассмотрим пример:

typedef int INTEGER;

После этого можно сделать объявление:

INTEGER а, b;

Оно будет выполнять то же самое, что и привычное объявление int a,b;. Другими словами, INTEGER можно использовать как синоним ключевого слова int.

|  |
| --- |
|  |

***Битовые поля***

Особую разновидность структур представляют собой битовые поля. Битовое поле - это последовательность соседних битов внутри одного, целого значения. Оно может иметь тип signed int или unsigned int и занимать от 1 до 16 битов. Поля размещаются в машинном слове в направлении от младших к старшим разрядам. Например, структура:

struct prim { int a:2;

unsigned b:3;

int c:5;

int d:1;

unsigned d:5; } i, j;

обеспечивает размещение данных в двух байтах (в одном слове). Если бы последнее поле было задано так: unsigned d:6, то оно размещалось бы не в первом слове, а в разрядах 0 - 5 второго слова.

В полях типа signed крайний левый бит является знаковым.

Поля используются для упаковки значений нескольких переменных в одно машинное слово с целью экономии памяти. Они не могут быть массивами и не имеют адресов, поэтому к ним нельзя применять унарную операцию &.

|  |
| --- |
|  |

***Объединение (union)***

Объединение - это некоторая переменная, которая может хранить (в разное время) объекты различного типа и размера. В результате появляется возможность работы в одной и той же области памяти с данными различного вида. Для описания объединения используется ключевое слово union, а соответствующий синтаксис аналогичен структурам.

Пусть задано определение:

union r {int ir; float fr; char cr;} z;

Здесь ir имеет размер 2 байта, fr - 4 байта, cr - 1 байт. Размер переменной z будет равен размеру самого большого из трех приведенных типов (т.е. 4 байтам). В один и тот же момент времени z может иметь значение только одной из переменных ir, fr или cr.

|  |
| --- |
|  |

***Перечислимый тип данных***

Перечислимый тип данных предназначен для описания объектов из некоторого заданного множества. Он задается ключевым словом enum. Рассморим пример:

enum seasons (spring, summer, autumn, winter);

Здесь введен новый тип данных seasons. Теперь можно определить переменные этого типа:

enum seasons а, b, с;

Каждая из них (а, b, c) может принимать одно из четырех значений: spring, summer, autumn и winter. Эти переменные можно было определить сразу при описании типа:

enum seasons (spring, summer, autumn, winter) a, b, с;

Рассмотрим еще один пример:

enum days {mon, tues, wed, thur, fri, sat, sun} my\_week;

Имена, занесенные в days (также как и в seasons в предыдущем примере), представляют собой константы целого типа. Первая из них (mon) автоматически устанавливается в нуль, и каждая следующая имеет значение на единицу больше, чем предыдущая (tues=1, wed=2 и т.д.).

Можно присвоить константам определенные значения целого типа (именам, не имеющим их, будут, как и раньше, назначены значения предыдущих констант, увеличенные на единицу). Например:

enum days (man=5, tues=8, wed=10, thur, fri, sat, sun} my\_week;

После этого mon=5, tues=8,wed=10, thur=11, fri=12, sat=13, sun=14.

Тип enum можно использовать для задания констант true=1 и false=0, например:

enum t\_f (false, true) а, b;

|  |
| --- |
|  |

**РАЗДЕЛ 4. ФУНКЦИИ**

***Общие сведения***

Программы на языке Си обычно состоят из большого числа отдельных функций (подпрограмм). Как правило, эти функции имеют небольшие размеры и могут находиться как в одном, так и в нескольких файлах. Все функции являются глобальными. В языке запрещено определять одну функцию внутри другой. Связь между функциями осуществляется через аргументы, возвращаемые значения и внешние переменные.

В общем случае функции в языке Си необходимо объявлять. Объявление функции (т.е. описание заголовка) должно предшествовать ее использованию, а определение функции (т.е. полное описание) может быть помещено как после тела программы (т.е. функции main( )), так и до него. Если функция определена до тела программы, а также до ее вызовов из определений других функций, то объявление может отсутствовать. Как уже отмечалось, описание заголовка функции обычно называют прототипом функции.

Функция объявляется следующим образом:

тип имя\_функции(тип имя\_параметра\_1, тип имя\_параметра\_2, ...);

Тип функции определяет тип значения, которое возвращает функция. Если тип не указан, то предполагается, что функция возвращает целое значение (int).

При объявлении функции для каждого ее параметра можно указать только его тип (например: тип функция (int, float, ...), а можно дать и его имя (например: тип функция (int а, float b, ...) ).

В языке Си разрешается создавать функции с переменным числом параметров. Тогда при задании прототипа вместо последнего из них указывается многоточие.

Определение функции имеет следующий вид:

тип имя\_функции(тип имя\_параметра\_1, тип имя\_параметра\_2,...)

{

тело функции

}

Передача значения из вызванной функции в вызвавшую происходит с помощью оператора возврата return, который записывается следующим образом:

return выражение;

Таких операторов в подпрограмме может быть несколько, и тогда они фиксируют соответствующие точки выхода. Например:

int f(int a, int b)

{

if (a > b) { printf("max = %d\n", a); return a; }

printf("max = %d\n", b); return b;

}

Вызвать эту функцию можно следующим образом:

c = f(15, 5);

c = f(d, g);

f(d, g);

Вызвавшая функция может, при необходимости, игнорировать возвращаемое значение. После слова return можно ничего не записывать; в этом случае вызвавшей функции никакого значения не передается. Управление передается вызвавшей функции и в случае выхода "по концу" (последняя закрывающая фигурная скобка).

В языке Си аргументы функции передаются по значению, т.е. вызванная функция получает свою временную копию каждого аргумента, а не его адрес. Это означает, что вызванная функция не может изменить значение переменной вызвавшей ее программы. Однако это легко сделать, если передавать в функцию не переменные, а их адреса. Например:

void swap(int \*a, int \*b)

{

int \*tmp = \*a;

\*a = \*b;

\*b = \*tmp;

}

Вызов swap(&b, &c) (здесь подпрограмме передаются адреса переменных b и с) приведет к тому, что значения переменных b и c поменяются местами.

Если же в качестве аргумента функции используется имя массива, то передается только адрес начала массива, а сами элементы не копируются. Функция может изменять элементы массива, сдвигаясь (индексированием) от его начала.

Рассмотрим, как функции можно передать массив в виде параметра. Здесь возможны три варианта:

1. Параметр задается как массив (например: int m[100];).
2. Параметр задается как массив без указания его размерности (например: int m[];).
3. Параметр задается как указатель (например: int \*m;). Этот вариант используется наиболее часто.

Независимо от выбранного варианта вызванной функции передается указатель на начало массива. Сами же элементы массива не копируются.

Если некоторые переменные, константы, массивы, структуры объявлены как глобальные, то их не надо включать в список параметров вызванной функции.

|  |
| --- |
|  |

***Классы памяти***

В языке Си различают четыре основных класса памяти: внешнюю (глобальную), автоматическую (локальную), статическую и регистровую память.

Внешние (глобальные) переменные определены вне функций и, следовательно, доступны для любой из них. Они могут быть определены только один раз. Выше уже говорилось, что сами функции всегда глобальные. Язык не позволяет определять одни функции внутри других. Область действия внешней переменной простирается от точки во входном файле, где она объявлена, до конца файла. Если на внешнюю переменную нужно ссылаться до ее определения или она определена в другом входном файле, то в подпрограмме или файле она должна быть объявлена как extern.

Например:

extern int a; /\* Объявление a; память под переменную не

резервируется \*/

Автоматические переменные по отношению к функциям являются внутренними или локальными. Они начинают существовать при входе в функцию и уничтожаются при выходе из нее (для них можно использовать ключевое слово auto). Однако оно практически не используется, так как при отсутствии ключевого слова переменные по умолчанию принадлежат к классу auto.

Статические переменные объявляются с помощью ключевого слова static. Они могут быть внутренними (локальными) или внешними (глобальными). Внутренние статические переменные, как и автоматические, локальны по отношению к отдельной функции. Однако они продолжают существовать, а не возникают и не уничтожаются при каждом ее вызове. Другими словами, они являются собственной постоянной памятью для функции. Внешние статические переменные доступны внутри оставшейся части файла после того, как они в нем объявлены, однако в других файлах они неизвестны. Это, в частности, позволяет скрыть данные одного файла от другого файла.

Регистровые переменные относятся к последнему классу. Ключевое слово register говорит о том, что переменная, о которой идет речь, будет интенсивно использоваться. Если возможно, значения таких переменных помещаются во внутренние регистры микропроцессора, что может привести к более быстрой и короткой программе (разработчики компиляторов фирмы Borland утверждают, что оптимизация компиляторов данной фирмы по использованию регистровых переменных сделана так хорошо, что указание использовать переменную как регистровую может только снизить эффективность создаваемого машинного кода). Для регистровых переменных нельзя взять адрес; они могут быть только автоматическими с допустимыми типами int или char.

Таким образом, можно выделить четыре модификатора класса памяти: extern, auto, static, register. Они используются в следующей общей форме:

модификатор\_класса\_памяти тип список\_переменных;

Выше уже говорилось об инициализации, т.е. о присвоении различным объектам начальных значений. Если явная инициализация отсутствует, гарантируется, что внешние и статические переменные будут иметь значение нуль, а автоматические и регистровые - неопределенное значение.

|  |
| --- |
|  |

***Указатели на функции***

В языке Си сама функция не может быть значением переменной, но можно определить указатель на функцию. С ним уже можно обращаться как с переменной: передавать его другим функциям, помещать в массивы и т.п.

Код функции в персональном компьютере занимает физическую память. В этой памяти есть точка входа, которая используется для того, чтобы войти в функцию и запустить ее на выполнение. Указатель на функцию как раз и адресует эту точку входа. Это уже будет обычная переменная и с ней можно делать все, что можно делать с переменной.

Через указатель можно войти в функцию, т.е. запустить ее на выполнение. Объявление вида:

int (\*f)( );

говорит о том, что f - это указатель на функцию, возвращающую целое значение. Первая пара скобок необходима, без них int \*f( ); означало бы, что f - функция, возвращающая указатель на целое значение. После объявления указателя на функцию в программе можно использовать объекты: \*f - сама функция; f - указатель на функцию. Для любой функции ее имя (без скобок и аргументов) является указателем на эту функцию.

|  |
| --- |
|  |

***Аргументы функции main( )***

В программы на языке Си можно передавать некоторые аргументы. Когда вначале вычислений производится обращение к main( ), ей передаются три параметра. Первый из них определяет число командных аргументов при обращении к программе. Второй представляет собой массив указателей на символьные строки, содержащие эти аргументы (в одной строке - один аргумент). Третий тоже является массивом указателей на символьные строки, он используется для доступа к параметрам операционной системы (к переменным окружения).

Любая такая строка представляется в виде:

переменная = значение\0

Последнюю строку можно найти по двум заключительным нулям.

Назовем аргументы функции main( ) соответственно: argc, argv и env (возможны и любые другие имена). Тогда допустимы следующие описания:

main( )

main(int argc)

main(int argc, char \*argv[ ] )

main(int argc, char \*argv[ ], char \*env[ ] )

Предположим, что на диске A: есть некоторая программа prog.exe. Обратимся к ней следующим образом:

A:\>prog.exe file1 file2 file3 <Enter>

Тогда argv[0] - это указатель на строку A:\prog.exe, argv[1] - на строку file1 и т.д. На первый фактический аргумент указывает argv[1], а на последний - argv[3]. Если argc=1, то после имени программы в командной строке параметров нет. В нашем примере argc=4.

|  |
| --- |
|  |

***Рекурсия***

Рекурсией называется такой способ вызова, при котором функция обращается к самой себе.

Важным моментом при составлении рекурсивной программы является организация выхода. Здесь легко допустить ошибку, заключающуюся в том, что функция будет последовательно вызывать саму себя бесконечно долго. Поэтому рекурсивный процесс должен шаг за шагом так упрощать задачу, чтобы в конце концов для нее появилось не рекурсивное решение. Использование рекурсии не всегда желательно, так как это может привести к переполнению стека.

|  |
| --- |
|  |

***Библиотечные функции***

В системах программирования подпрограммы для решения часто встречающихся задач объединяются в библиотеки. К числу таких задач относятся: вычисление математических функций, ввод/вывод данных, обработка строк, взаимодействие со средствами операционной системы и др. Использование библиотечных подпрограмм избавляет пользователя от необходимости разработки соответствующих средств и предоставляет ему дополнительный сервис. Включенные в библиотеки функции поставляются вместе с системой программирования. Их объявления даны в файлах \*.h (это так называемые включаемые или заголовочные файлы). Поэтому, как уже упоминалось выше, в начале программы с библиотечными функциями должны быть строки вида:

#include <включаемый\_файл\_типа\_h>

Например:

#include <condio.h>

Существуют также средства для расширения и создания новых библиотек с программами пользователя.

|  |
| --- |
|  |

**РАЗДЕЛ 5. ФАЙЛЫ**

Файлом называют способ хранения информации на физическом устройстве. Файл - это понятие, которое применимо ко всему - от файла на диске до терминала.

В языке Си отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.д. Эти устройства сильно отличаются друг от друга. Однако файловая система преобразует их в единое *абстрактное логическое устройство*, называемое *потоком*.

В Си существует два типа потоков: текстовые (text) и двоичные (binary).

*Текстовый поток* - это последовательность символов. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки).

*Двоичный поток* - это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

Прежде чем читать или записывать информацию в файл, он должен быть открыт и тем самым связан с потоком. Это можно сделать с помощью библиотечной функции fopen( ). Она берет внешнее представление файла (например, c:\my\_prog.txt) и связывает его с внутренним логическим именем, которое используется далее в программе. Логическое имя - это указатель на требуемый файл. Его необходимо определить; делается это, например, так:

FILE \*fp;

Здесь FILE - имя типа, описанное в стандартном заголовочном файле stdio.h, fp - указатель на файл. Обращение к функции fopen( ) в программе осуществляется выражением:

fp = fopen(спецификация файла, "способ использования файла");

Спецификация файла (т.е. имя файла и путь к нему) может, например, иметь вид: "c:\\my\_prog.txt" - для файла my\_prog.txt на диске с:.

Способ использования файла задается следующими символами:

r - открыть существующий файл для чтения;  
w - создать новый файл для записи (если файл с указанным именем существует, то он будет переписан);  
а - дополнить файл (открыть существующий файл для записи информации, начиная с конца файла, или создать файл, если он не существует);  
  
r+ - открыть существующий файл для чтения и записи;  
w+ - создать новый файл для чтения и записи;  
a+ - дополнить или создать файл с возможностью чтения и записи;  
  
rb - открыть двоичный файл для чтения;  
wb - создать двоичный файл для записи;  
аb - дополнить двоичный файл;  
  
r+b - открыть двоичный файл для чтения и записи;  
w+b - создать двоичный файл для чтения и записи;  
а+b - дополнить двоичный файл с предоставлением возможности чтения и записи;  
  
rt - открыть текстовой файл для чтения;  
wt - создать текстовый файл для записи;  
at - дополнить текстовый файл;  
  
r+t - открыть текстовой файл для чтения и записи;  
w+t - создать текстовый файл для чтения и записи;  
a+t - дополнить текстовый файл с предоставлением возможности записи и чтения.

Если режим t или b не задан (например, r, w или а), то он определяется значением глобальной переменной \_fmode. Если fmode=0\_BINARY, то файлы открываются в двоичном режиме, а если \_fmode=0\_TEXT - в текстовом режиме. Константы 0\_BINARY и 0\_ТЕXТ определены в файле fcntl.h.

Строки вида r+b можно записывать и в другой форме: rb+.

Если в результате обращения к функции fopen( ) возникает ошибка, то она возвращает константу NULL.

Рекомендуется использовать следующий способ открытия файла:

if ((fp = fopen("c:\\my\_prog.txt", "rt")) == NULL)

{

puts("Открыть файл не удалось\n");

exit(1);

}

После окончания работы с файлом он должен быть закрыт. Это делается с помощью библиотечной функции fclose( ). Она имеет следующий прототип:

int fclose(FILE \*fp);

При успешном завершении операции функция fclose( ) возвращает значение нуль. Любое другое значение свидетельствует об ошибке.

Рассмотрим другие библиотечные функции, используемые для работы с файлами (все они описаны в файле stdio.h):

1. Функция putc( ) записывает символ в файл и имеет следующий прототип:

int putc(int с, FILE \*fp);

Здесь fp - указатель на файл, возвращенный функцией fopen( ), с - символ для записи (переменная с имеет тип int, но используется только младший байт). При успешном завершении putc( ) возвращает записанный символ, в противном случае возвращается константа EOF. Она определена в файле stdio.h и имеет значение -1.

2. Функция getc( ) читает символ из файла и имеет следующий прототип:

int getc(FILE \*fp);

Здесь fp - указатель на файл, возвращенный функцией fopen( ). Эта функция возвращает прочитанный символ. Соответствующее значение имеет тип int, но старший байт равен нулю. Если достигнут конец файла, то getc( ) возвращает значение ЕОF.

3. Функция feof( ) определяет конец файла при чтении двоичных данных и имеет следующий прототип:

int feof(FILE \*fp);

Здесь fp - указатель на файл, возвращенный функцией fopen( ). При достижении конца файла возвращается ненулевое значение, в противном случае возвращается 0.

4. Функция fputs( ) записывает строку символов в файл. Она отличается от функции puts( ) только тем, что в качестве второго параметра должен быть записан указатель на переменную файлового типа.

Например:

fputs("Ехаmple", fp);

При возникновении ошибки возвращается значение EOF.

5. Функция fgets( ) читает строку символов из файла. Она отличается от функции gets( ) тем, что в качестве второго параметра должно быть указано максимальное число вводимых символов плюс единица, а в качестве третьего - указатель на переменную файлового типа. Строка считывается целиком, если ее длина не превышает указанного числа символов, в противном случае функция возвращает только заданное число символов.

Рассмотрим пример:

fgets(string, n, fp);

Функция возвращает указатель на строку string при успешном завершении и константу NULL в случае ошибки либо достижения конца файла.

6. Функция fprintf( ) выполняет те же действия, что и функция printf( ), но работает с файлом. Ее отличием является то, что в качестве первого параметра задается указатель на переменную файлового типа.

Например:

fprintf(fp, "%х",а);

7. Функция fscanf( ) выполняет те же действия, что и функция scanf(), но работает с файлом. Ее отличием является то, что в качестве первого параметра задается указатель на переменную файлового типа.

Например:

fscanf(fp, "%х", &a);

При достижении конца файла возвращается значение EOF.

8. Функция fseek( ) позволяет выполнять чтение и запись с произвольным доступом и имеет следующий прототип:

int fseek(FILE \*fp, long count, int access);

Здесь fp - указатель на файл, возвращенный функцией fopen( ), count - номер байта относительно заданной начальной позиции, начиная с которого будет выполняться операция, access - способ задания начальной позиции.

Переменная access может принимать следующие значения:

0 - начальная позиция задана в начале файла;  
1 - начальная позиция считается текущей;  
2 - начальная позиция задана в конце файла.

При успешном завершении возвращается нуль, при ошибке - ненулевое значение.

9. Функция ferror( ) позволяет проверить правильность выполнения последней операции при работе с файлами. Имеет следующий прототип:

int ferror(FILE \*fp);

В случае ошибки возвращается ненулевое значение, в противном случае возвращается нуль.

10. Функция remove( ) удаляет файл и имеет следующий прототип:

int remove(char \*file\_name);

Здесь file\_name - указатель на строку со спецификацией файла. При успешном завершении возвращается нуль, в противном случае возвращается ненулевое значение.

11. Функция rewind( ) устанавливает указатель текущей позиции в начало файла и имеет следующий прототип:

void rewind(FILE \*fp);

12. Функция fread( ) предназначена для чтения блоков данных из потока. Имеет прототип:

unsigned fread(void \*ptr, unsigned size, unsigned n, FILE \*fp);

Она читает n элементов данных, длиной size байт каждый, из заданного входного потока fp в блок, на который указывает указатель ptr. Общее число прочитанных байтов равно произведению n\*size. При успешном завершении функция fread( ) возвращает число прочитанных элементов данных, при ошибке - 0.

13. Функция fwrite( ) предназначена для записи в файл блоков данных. Имеет прототип:

unsigned fwrite(void \*ptr, unsigned size, unsigned n, FILE \*fp);

Она добавляет n элементов данных, длиной size байт каждый, в заданный выходной файл fp. Данные записываются с позиции, на которую указывает указатель ptr. При успешном завершении операции функция fwrite( ) возвращает число записанных элементов данных, при ошибке - неверное число элементов данных.

В языке Си имеются пять стандартных файлов со следующими логическими именами:

**stdin** - для ввода данных из стандартного входного потока (по умолчанию - c клавиатуры);  
**stdout** - для вывода данных в стандартный выходной поток (по умолчанию - на экран дисплея);  
**stderr** - файл для вывода сообщений об ошибках (всегда связан с экраном дисплея);  
**stdprn** - для вывода данных на принтер;  
**stdaus** - для ввода и вывода данных в коммуникационный канал.

В языке Си имеется также система низкоуровневого ввода/вывода (без буферизации и форматирования данных), соответствующая стандарту системы UNIX. Прототипы составляющих ее функций находятся в файле io.h. К этим функциям относятся:

open( ) - открыть файл;  
close( ) - закрыть файл;  
read( ) - читать данные;  
write( ) - записать данные;  
lseek( ) - поиск определенного байта в файле;  
unlink( ) - уничтожить файл.

|  |
| --- |
|  |

**РАЗДЕЛ 6. ДРУГИЕ ВОЗМОЖНОСТИ**

***Динамическое распределение памяти. Функции malloc( ) и free( )***

В языке Си принято следующее распределение памяти:

Таблица 4

|  |  |
| --- | --- |
| СТЕК | Верхние адреса |
| СВОБОДНАЯ ПАМЯТЬ |  |
| РАЗДЕЛ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ И КОНСТАНТ |  |
| КОД ПРОГРАММЫ | Нижние адреса |

Для глобальных переменных отводится фиксированное место в памяти на все время работы программы. Локальные переменные хранятся в стеке. Между ними находится область памяти для динамического распределения.

Функции malloc( ) и free( ) используются для динамического распределения свободной памяти. Функция malloc( ) выделяет память, функция free( ) освобождает ее. Прототипы этих функций хранятся в заголовочном файле stdlib.h и имеют вид:

void \*malloc(size\_t size);

void \*free(void \*p);

Функция malloc( ) возвращает указатель типа void; для правильного использования значение функции надо преобразовать к указателю на соответствующий тип. При успешном выполнении функция возвращает указатель на первый байт свободной памяти размера size. Если достаточного количества памяти нет, возвращается значение 0. Чтобы определить количество байтов, необходимых для переменной, используют операцию sizeof( ).

Пример использования этих функций:

#include <stdio.h>

#include <stdlib.h>

void main(void)

{

int \*p, i;

p = (int \*) malloc(100 \* sizeof(int)); /\* Выделение памяти для 100

целых чисел \*/

if (!p)

{

printf("Недостаточно памяти\n");

exit(1);

}

for (i = 0; i < 100; ++i) \*(p+i) = i; /\* Использование памяти \*/

for (i = 0; i < 100; ++i) printf("%d", \*(p++) );

free(p); /\* Освобождение памяти \*/

}

Перед использованием указателя, возвращаемого malloc( ), необходимо убедиться, что памяти достаточно (указатель не нулевой).

|  |
| --- |
|  |

***Препроцессор***

Препроцессор Си - это программа, которая обрабатывает входные данные для компилятора. Препроцессор просматривает исходную программу и выполняет следующие действия: подключает к ней заданные файлы, осуществляет подстановки, а также управляет условиями компиляции. Для препроцессора предназначены строки программы, начинающиеся с символа #. В одной строке разрешается записывать только одну команду (директиву препроцессора).

Директива

#define идентификатор подстановка

вызывает замену в последующем тексте программы названного идентификатора на текст подстановки (обратите внимание на отсутствие точки с запятой в конце этой команды). По существу, эта директива вводит макроопределение (макрос), где "идентификатор" - это имя макроопределения, а "подстановка" - последовательность символов, на которые препроцессор заменяет указанное имя, когда находит его в тексте программы. Имя макроопределения принято набирать прописными буквами.

Рассмотрим примеры:

#define MAX 25

#define BEGIN {

Первая строка вызывает замену в программе идентификатора MAX на константу 25. Вторая позволяет использовать в тексте вместо открывающей фигурной скобки ( { ) слово BEGIN.

Отметим, что поскольку препроцессор не проверяет совместимость между символическими именами макроопределений и контекстом, в котором они используются, то рекомендуется такого рода идентификаторы определять не директивой #define, а с помощью ключевого слова const с явным указанием типа (это в большей степени относится к Си++):

const int MAX = 25;

(тип int можно не указывать, так как он устанавливается по умолчанию).

Если директива #define имеет вид:

#define идентификатор(идентификатор, ..., идентификатор) подстановка

причем между первым идентификатором и открывающей круглой скобкой нет пробела, то это определение макроподстановки с аргументами. Например, после появления строки вида:

#define READ(val) scanf("%d", &val)

оператор READ(y); воспринимается так же, как scanf("%d",&y);. Здесь val - аргумент и выполнена макроподстановка с аргументом.

При наличии длинных определений в подстановке, продолжающихся в следующей строке, в конце очередной строки с продолжением ставится символ \.

В макроопределение можно помещать объекты, разделенные знаками ##, например:

#define PR(x, у) x##y

После этого PR(а, 3) вызовет подстановку а3. Или, например, макроопределение

#define z(a, b, c, d) a(b##c##d)

приведет к замене z(sin, x, +, y) на sin(x+y).

Символ #, помещаемый перед макроаргументом, указывает на преобразование его в строку. Например, после директивы

#define PRIM(var) printf(#var"= %d", var)

следующий фрагмент текста программы

year = 2006;

PRIM(year);

преобразуется так:

year = 2006;

printf("year""= %d", year);

Опишем другие директивы препроцессора. Директива #include уже встречалась ранее. Ее можно использовать в двух формах:

#include "имя файла"

#include <имя файла>

Действие обеих команд сводится к включению в программу файлов с указанным именем. Первая из них загружает файл из текущего или заданного в качестве префикса каталога. Вторая команда осуществляет поиск файла в стандартных местах, определенных в системе программирования. Если файл, имя которого записано в двойных кавычках, не найден в указанном каталоге, то поиск будет продолжен в подкаталогах, заданных для команды #include <...>. Директивы #include могут вкладываться одна в другую.

Следующая группа директив позволяет избирательно компилировать части программы. Этот процесс называется условной компиляцией. В эту группу входят директивы #if, #else, #elif, #endif, #ifdef, #ifndef. Основная форма записи директивы #if имеет вид:

#if константное\_выражение последовательность\_операторов

#endif

Здесь проверяется значение константного выражения. Если оно истинно, то выполняется заданная последовательность операторов, а если ложно, то эта последовательность операторов пропускается.

Действие директивы #else подобно действию команды else в языке Си, например:

#if константное\_выражение

последовательность\_операторов\_1

#else

последовательность\_операторов\_2

#endif

Здесь если константное выражение истинно, то выполняется последовательность\_операторов\_1, а если ложно - последовательность\_операторов\_2.

Директива #elif означает действие типа "else if". Основная форма ее использования имеет вид:

#if константное\_выражение

последовательность\_операторов

#elif константное\_выражение\_1

последовательность\_операторов\_1

#elif константное\_выражение\_n

последовательность\_операторов\_n

#endif

Эта форма подобна конструкции языка Си вида: if...else if...else if...

Директива

#ifdef идентификатор

устанавливает определен ли в данный момент указанный идентификатор, т.е. входил ли он в директивы вида #define. Строка вида

#ifndef идентификатор

проверяет является ли неопределенным в данный момент указанный идентификатор. За любой из этих директив может следовать произвольное число строк текста, возможно, содержащих инструкцию #else (#elif использовать нельзя) и заканчивающихся строкой #endif. Если проверяемое условие истинно, то игнорируются все строки между #else и #endif, а если ложно, то строки между проверкой и #else (если слова #else нет, то #endif). Директивы #if и #ifndef могут "вкладываться" одна в другую.

Директива вида

#undef идентификатор

приводит к тому, что указанный идентификатор начинает считаться неопределенным, т.е. не подлежащим замене.

Рассмотрим примеры. Три следующие директивы:

#ifdef WRITE

#undef WRITE

#endif

проверяют определен ли идентификатор WRITE (т.е. была ли команда вида #define WRITE...), и если это так, то имя WRITE начинает считаться неопределенным, т.е. не подлежащим замене.

Директивы

#ifndef WRITE

#define WRITE fprintf

#endif

проверяют является ли идентификатор WRITE неопределенным, и если это так, то определятся идентификатор WRITE вместо имени fprintf.

Директива #error записывается в следующей форме:

#error сообщение\_об\_ошибке

Если она встречается в тексте программы, то компиляция прекращается и на экран дисплея выводится сообщение об ошибке. Эта команда в основном применяется на этапе отладки. Заметим, что сообщение об ошибке не надо заключать в двойные кавычки.

Директива #line предназначена для изменения значений переменных \_LINE\_ и \_FILE\_, определенных в системе программирования Си. Переменная \_LINE\_ содержит номер строки программы, выполняемой в текущий момент времени. Идентификатор \_FILE\_ является указателем на строку с именем компилируемой программы. Директива #line записывается следующим образом:

#line номер "имя\_файла"

Здесь номер - это любое положительное целое число, которое будет назначено переменной \_LINE\_, имя\_файла - это необязательный параметр, который переопределяет значение \_FILE\_.

Директива #pragma позволяет передать компилятору некоторые указания. Например, строка

#pragma inline

говорит о том, что в программе на языке Си имеются строки на языке ассемблера. Например:

asm mov ax, 5

asm {

inc dx

sub bl, al

}

и т.д.

Рассмотрим некоторые глобальные идентификаторы или макроимена (имена макроопределений). Определены пять таких имен: \_LINE\_, \_FILE\_, \_DATE\_, \_TIME\_, \_STDC\_. Два из них (\_LINE\_ и \_FILE\_) уже описывались выше. Идентификатор \_DATE\_ определяет строку, в которой сохраняется дата трансляции исходного файла в объектный код. Идентификатор \_TIME\_ задает строку, сохраняющую время трансляции исходного файла в объектный код. Макрос \_STDC\_ имеет значение 1, если используются стандартно - определенные макроимена. В противном случае эта переменная не будет определена.

|  |
| --- |
|  |

***Использование программно-доступных регистров микропроцессора Intel 8086***

В языке Си для IBM-совместимых персональных компьютеров обращение к регистрам микропроцессора Intel 8086 осуществляется с помощью специальных объектов, называемых псевдопеременными. Полный список псевдопеременных включает 21 элемент: \_АХ, \_ВХ, \_СХ, \_DX, \_CS, \_DS, \_SS, \_ES, \_SP, \_ВР, \_DI, \_SI, \_AL, \_AH, \_BL, \_ВH, \_CL, \_СН, \_DL, \_DH, \_FLAGS. Их имена образуются из имен регистров с префиксом \_ (например, переменная \_AX связана с регистром AX). Первые двенадцать псевдопеременных и последняя имеют тип unsigned int, а оставшиеся восемь - unsigned char. Присвоение значения какой-либо переменной, например \_АХ, вызывает занесение этого значения в регистр АХ. Получение значения переменной, например \_ВХ, эквивалентно получению значения из регистра ВХ.

|  |
| --- |
|  |

**РАЗДЕЛ 7. ПРИМЕРЫ**

Рассмотрим [примеры программ](http://dfe.karelia.ru/koi/posob/c/exampl_c.zip), в которых используются различные конструкции языка Си. Первый из них демонстрирует использование управляющих символов *n* в функциях printf( ) и scanf( ).

/\* Пример 1 \*/

#include <stdio.h>

void main(void)

{

int x, n1, n2;

printf("Введите целое число от -32768 до 32767\n");

scanf("%d%n", &x, &n1);

printf("x = %d%n\n", x, &n2);

printf("n1 = %d, n2 = %d\n", n1, n2);

}

Результаты работы этой программы имеют вид:

Введите целое число от -32768 до 32767

234<Enter>

x = 234

n1 = 3, n2 = 7

Значение n1 определяет число введенных цифр, а n2 - число выведенных символов в строке x = 234 (с пробелами).

Следующий пример показывает использование спецификаций %[], а также символов \* и #.

/\* Пример 2 \*/

#include <stdio.h>

void main(void)

{

char str\_b[21], str\_c[21]; /\* Последний элемент резервируем под \0 \*/

int x, n1, n2;

float y;

printf("Введите строку до 20 символов\n");

scanf("%[Computer]%s", str\_b, str\_c);

printf("str\_b = %s, str\_c = %s\n", str\_b, str\_c);

y = 12.345678;

n1 = 8;

n2 = 3;

x = 0x100;

printf("y = %\*.\*f\n", n1, n2, y);

printf("x(16) = %#x, x(16) = %x, x(10) = %i\n", x, x, x);

}

Результаты работы программы имеют следующий вид:

Введите строку до 20 символов

Comp-1-2-3-4-5<Enter>

str\_b = Comp, str\_c = -1-2-3-4-5

у = 12.346

x(16) = 0x100, x(16) = 100, x(10) = 256

Здесь пользователем введена строка Comp-1-2-3-4-5. Из нее только четыре символа (Comp) совпадают с первыми символами, заданными в квадратных скобках рассматриваемой спецификации [Computer]. Поэтому только эти четыре символа попадут в первую строку, а оставшиеся символы (1-2-3-4-5) попадут во вторую строку. Число n1 определяет минимальную ширину поля для вывода, а число n2 = 3 - количество цифр после запятой. В результате число 12.345678 будет смещено относительно левой границы, и после запятой будут выведены три цифры. Шестнадцатеричное число 0х100 выведено функцией printf( ) с префиксом 0x, без префикса 0x и в десятичной форме.

Третья программа демонстрирует использование условного оператора if...else и оператора for для организации цикла.

/\* Пример 3 \*/

#include <conio.h>

#define SYM 'X' /\* Выводимый символ \*/

#define SPACE ' ' /\* Определение пробела \*/

#define LF 10 /\* Перевод строки \*/

#define CR 13 /\* Возврат каретки \*/

#define LEFT 24 /\* Левая граница символа \*/

#define RIGHT 51 /\* Правая граница символа \*/

#define BOTTOM 25 /\* Нижняя граница символа \*/

void main(void)

{

int col, line; /\* col - номер колонки для вывода символа \*/

/\* line - номер линям для вывода символа \*/

clrscr( );

for (line = 1; line <= BOTTOM; line++) /\* Вывод пробелов до левой

границы символа \*/

{

for (col = 1; col < LEFT; col++) putch(SPACE);

for(col = LEFT + 1; col < RIGHT; col++) /\* Вывод символа X

на весь экран \*/

if ((col == (LEFT + line)) || (col == (RIGHT - line)))

putch(SYM);

else putch(SPACE);

putch(LF); /\* Возврат каретки и перевод строки после \*/

putch(CR); /\* вывода каждой линии символа \*/

}

getch( ); /\* Ожидание нажатия клавиши \*/

}

После ее запуска на весь экран будет выведен символ X.

Новая библиотечная функция clrscr( ) имеет следующий прототип:

void clrscr (void);

Она выполняет очистку экрана и объявлена в заголовочном файле conio.h.

Четвертая программа демонстрирует использование рекурсивной функции для вычисления факториала. (Отметим, что определение функции factorial( ) может находиться и после функции main( ), но в этом случае функция factorial( ) должна быть объявлена перед функцией main( ), т.е. до main( ) необходимо поместить строку: long factorial(int);.)

/\* Пример 4 \*/

#include <stdio.h>

#include <values.h>

#include <process.h>

long factorial(int value) /\* Рекурсивная функция \*/

{

long result = 1;

if (value != 0)

{

result = factorial(value - 1);

/\* Проверка возможности вычисления факториала \*/

if (result > MAXLONG / (value + 1))

{

fprintf(stderr, "Очень большое число\n");

getch( ); /\* Ожидание нажатия клавиши \*/

exit (1);

}

result \*= value;

}

return(result);

}

/\* Рекурсивное вычисление факториала числа value \*/

void main(void)

{

int value; /\* Факториал этого значения вычисляется \*/

long result; /\* Переменная для результата \*/

puts("Факториал какого числа?");

scanf("%d", &value);

result = factorial(value);

printf("Результат: %ld\n", result);

getch( ); /\* Ожидание нажатия клавиши \*/

}

Результаты работы этой программы:

Факториал какого числа? 10<Enter>

Результат: 362880

Пятая программа подсчитывает число символов и слов во вводимых строках (новые символы и слова суммируются с предыдущими; пробелы входят в число введенных символов).

/\* Пример 5 \*/

#include <stdio.h>

#include <conio.h>

#define ESC 27 /\* 27 - ASCII-код клавиши ESC \*/

void CountOfLines(void)

{

/\* Статические переменные будут сохранять старые значения при каждом

новом вызове функции CountOfLines \*/

static int words = 0, symbols = 0; /\* words-число слов,

symbols-число символов \*/

char temp, t = 0; /\* Временные переменные \*/

++symbols;

/\* Число символов и слов выдается после нажатия клавиши <Enter> \*/

while ((temp = getche( )) != '\r' )

{

++symbols; /\* Подсчитывается каждый символ \*/

/\* После одного или нескольких пробелов подсчитывается слово \*/

if ((temp == ' ') && (t == 1)) continue;

if (temp == ' ') { t = 1; ++words; }

else t = 0;

}

if (t == 1) --words;

else ++words;

printf ("\n Слов: %d; символов: %d\n", words, symbols);

}

void main(void)

{

puts("Для завершения программы нажмите <ESC> в начале строки");

puts("Строка не должна начинаться с пробела и с нажатия клавиши"

"<Enter>");

puts("Строка не должна завершаться пробелом");

while (getche( ) != ESC) CountOfLines();

putch('\b');

putch(' ');

putch('\b');

}

Результаты работы этой программы:

Для завершения программы нажмите <ESC> в начале строки

Строка не должна начинаться с пробела и с нажатия клавиши <Enter>

Строка не должна завершаться пробелом

Mouse Keyboard <Enter>

Слов: 2 символов: 14

<ESC>

Следующая группа программ демонстрирует работу с файлами. Она позволяет организовать в файле на диске телефонный справочник и выполняет следующие функции:

* занесение фамилии абонента и номера телефона в справочник;
* поиск в справочнике номера телефона по фамилии абонента;
* удаление из справочника фамилии абонента и номера его телефона.

Ниже приведен текст головной программы main.c:

// Пример 6

//---------------------------------------------------------

// Головная программа для работы с телефонным справочником

//---------------------------------------------------------

#include "A:\my.h" //Заголовочный файл с глобальными

//переменными и константами

#include "A:\findt.c" //Поиск строки str в файле

#include "A:\choicet.c" //Проверка наличия строки в файле

#include "A:\addt.c" //Добавление строки в файл

#include "A:\subt.c" //Удаление строки из файла

void main(int argc, char \*argv[ ])

{

if (argc == 3)

if (\*argv[1] == '+') //Добавить запись

{

if (Choice(argv[2]) == 0) //Нет ли такой

//записи в файле?

{

puts("Эта фамилия есть в справочнике");

exit(1);

}

Add(argv[2]); //Добавление записи

}

else if (\*argv[1] == '-') Sub(argv[2]); //Удалить запись

else puts("Ошибочное значение аргумента");

else if (argc == 2) Find(argv[1]); //Поиск записи

else puts("Ошибочное число аргументов");

}

С помощью директив #include в головную программу включаются файлы: my.h, findt.c, choicet.c, addt.c и subt.c. Считается, что все они находятся в корневом каталоге диска A:. Если это не так, то необходимо изменить соответствующие директивы #include. В файле my.h определены глобальные переменные и некоторые символьные значения.

//Файл заголовков my.h

//--------------------------------------------------------

//Определения глобальных переменных и символьных значений

//--------------------------------------------------------

#include <stdio.h>

#include <process.h>

#include <string.h>

#define MAX\_NAME 20 //Максимальное число символов в фамилии

#define MAX\_NUMBER 10 //Максимальное число цифр в телеф. номере

char Name[MAX\_NAME]; //Массив фамилий

char Number[MAX\_NUMBER]; //Массив для телефонного номера

char File[ ] = "A:\\tel\\tel\_num.txt"; //Имя файла справочника

int Count; //Число фамилий в справочнике

FILE \*F\_tel; //Логическое имя файла справочника

Файл my.h, в частности, определяет, что телефонный справочник будет организован в каталоге tel диска A:. Поэтому необходимо перед запуском программы main.exe создать этот подкаталог либо использовать другой подкаталог. В последнем случае необходимо изменить строку:

char File[ ] = "A:\\tel\\tel\_num.txt";

которая задает имя файла с телефонным справочником (tel\_num.txt).

Модуль findt.c, текст которого приведен ниже, содержит функцию Find( ) для поиска строки str в файле tel\_num.txt.

//Модуль findt.c

//----------------------------------------------------------

//Функция Find( ) для поиска строки str в файле tel\_num.txt

//----------------------------------------------------------

void Find(char \*str)

{

int i;

//Если файл невозможно открыть для чтения, то программа

//завершает работу

if ((F\_tel = fopen(File, "r")) == NULL)

{

fprintf(stderr, "\"%s\" : невозможно открыть\n", File);

exit(1);

}

//Чтение числа записей (Count) в файле

if (fread(&Count, sizeof(int), 1, F\_tel) != 1)

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

//В цикле for осуществляется поиск нужной записи

for (i = 1; i < Count; i++)

{

fread(Name, 1, MAX\_NAME, F\_tel); //Чтение имени

fread(Number, 1, MAX\_NUMBER, F\_tel); //Чтение номера

if (ferror(F\_tel)) //Проверка отсутствия ошибки

{

fprintf(stderr, "\"%s\" : ошибка чтения\n'', File);

exit(1);

}

if (strcmp(str, Name) == 0) //Если имя совпадает

//с введенным, то фамилия

//и найденный номер

//выводятся на экран

{

printf("Фамилия : %s\n", Name);

printf("Номер телефона: %s\n", Number);

fclose(F\_tel);

return;

}

}

//Если результат поиска отрицательный, то выводится

//следующее сообщение

fprintf(stderr,"\"%s\" : запись в файле отсутствует\n", File);

fclose(F\_tel);

return;

}

Модуль choicet.c содержит функцию Choice( ), позволяющую проверить есть ли заданная строка в файле tel\_num.txt.

// Модуль choicet.c

//----------------------------------------------------------------------

//Функция Choice( ), проверяющая есть ли строка str в файле tel\_num.txt

//----------------------------------------------------------------------

int Choice(char \*str)

{

int i;

char temp[MAX\_NAME + MAX\_NUMBER];

if ((F\_tel = fopen(File, "r")) == NULL)

return 1; //Строки str нет в файле

if (fread(&Count, sizeof(int), 1, F\_tel) != 1)

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

for (i = 0; i < Count; i++)

{

fread(temp, 1, MAX\_NAME + MAX\_NUMBER, F\_tel);

if (ferror(F\_tel))

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

if (strcmp(str, temp) == 0)

{

fclose(F\_tel);

return 0; //Строка str есть в файле

}

}

fclose(F\_tel);

return 1; //Строки str нет в файле

}

Модуль addt.c содержит функцию Add( ), которая добавляет заданную строку в файл tel\_num.txt.

//Модуль addt.c

//-------------------------------------------------------

//Функция Add( ) добавляет строку str в файл tel\_num.txt

//-------------------------------------------------------

void Create(void) //Создает файл, если он не существует

{

if ((F\_tel = fopen(File, "wb+")) == NULL)

{

fprintf(stderr, "\%s\" : невозможно открыть\n", File);

exit(1);

}

Count = 0;

if ( ! fwrite(&Count, sizeof(Count), 1, F\_tel))

{

fprintf(stderr, "\"%s\" : ошибка записи\n", File);

exit(1);

}

}

void Add(char \*s) //Добавляет запись в файл

{

char str[MAX\_NAME], sn[MAX\_NUMBER]; //Временные массивы

int i;

for (i = 0; i < MAX\_NAME; i++)

str[i] = ' '; //Пробелы в str

strcpy(str, s); //Копирование строки в str

if ((F\_tel = fopen(File, "rb+")) = = NULL)

Create(); //Создаем файл, если он не

//существует

else if (fread(&Count, sizeof(Count), 1, F\_tel) != 1)

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

printf("Номер телефона : "); //Запрашивается и вводится номер

if (gets(Number) == NULL || \*Number == '\0')

{

fclose(F\_tel);

return; //Возврат, если номер не введен

}

//Установка указателя в файле на первую свободную запись

if (fseek(F\_tel, (long)((MAX\_NAME+MAX\_NUMBER)\*Count), SEEK\_CUR)!=0)

{

fprintf(stderr, "\"%s\" : ошибка поиска\n", File);

exit(1);

}

fwrite(str, 1, MAX\_NAME, F\_tel); //Запись в файл фамилии

for (i = 0; i < MAX\_NUMBER; i++)

sn[i] = ' '; //Пробелы в sn[ ]

strcpy(sn, Number); //Копирование сроки Number в строку sn

fwrite(sn, 1, MAX\_NUMBER, F\_tel); //Запись в файл номера

if (ferror(F\_tel)) //Проверка наличия ошибки

{

fprintf(stderr, "\"%s\" : ошибка записи\n", File);

exit(1);

}

//Установка указателя в файле на первый байт

if (fseek(F\_tel, 0L, SEEK\_SET) != 0)

{

fprintf(stderr, "\"%s\" : ошибка позиционирования\n", File);

exit(1);

}

++Count; //Увеличение числа записей на единицу

//Запись Count в файл

if (fwrite(&Count, sizeof(int), 1,F\_tel) != 1)

{

fprintf(stderr, "\"%s\" : ошибка записи\n", File);

exit(1);

}

fclose(F\_tel);

return;

}

Модуль subt.c содержит функцию Sub( ), которая удаляет заданную строку из файла tel\_num.txt.

//Модуль subt.c

//------------------------------------------------------------

//Функция Sub( ) удаляет заданную строку из файла tel\_num.txt

//------------------------------------------------------------

void Sub(char \*str)

{

int i, j;

char temp[MAX\_NAME + MAX\_NUMBER]; //Временный массив

if ((F\_tel = fopen(File, "r+")) == NULL)

{

fprintf(stderr, "\"%s\" : невозможно открыть\n", File);

exit(1);

}

if (fread(&Count, sizeof(int), 1, F\_tel) != 1)

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

//В цикле осуществляется поиск удаляемой строки в файле

for (i = 0; i < Count; i++)

{

fread(temp, 1, MAX\_NAME + MAX\_NUMBER, F\_tel);

if (ferror(F\_tel))

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

if (strcmp(str, temp) == 0) //Если срока найдена

{

for (j = i; j < Count; j++) //она удаляется

{

fread(temp, 1, MAX\_NAME + MAX\_NUMBER, F\_tel);

fseek(F\_tel, (long)(j\*(MAX\_NAME+MAX\_NUMBER)+2L), SEEK\_SET);

fwrite(temp, 1, MAX\_NAME + MAX\_NUMBER, F\_tel);

fseek(F\_tel, (long)((j+2)\*(MAX\_NAME+MAX\_NUMBER)+2L), SEEK\_SET);

if (ferror(F\_tel))

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

}

--Count; //При удалении строки декремент Count

fseek(F\_tel, 0L, SEEK\_SET); //Установка указателя

//Запись уменьшенного значения Count в файл

if (fwrite(&Count, sizeof(Count), 1, F\_tel) != 1)

{

fprintf(stderr, "\"%s\" : ошибка записи\n", File);

exit(1);

}

fclose(F\_tel);

puts("Запись удалена из файла");

return;

}

}

fprintf(stderr, "\"%s\" : отсутствует в базе данных\n", File);

fclose(F\_tel);

}

Ниже приводится возможный сценарий работы с программой main.

main + Петров<Enter>

Номер телефона: 77-17-89<Enter>

main + Иванов<Enter>

Номер телефона: 52-98-02<Enter>

main Иванов<Enter>

Фамилия: Иванов

Номер телефона: 52-98-02

main - Петров<Enter>

Запись удалена из файла

main Петров<Enter>

"tel\_num.txt" : запись в файле отсутствует

Последняя программа showt.c позволяет вывести на экран содержимое телефонного справочника.

//Программа showt.c

//-------------------------------------------------

//Выводит на экран все записи из файла tel\_num.txt

//-------------------------------------------------

#include "my.h"

void Show(void)

{

int i;

//Если файл невозможно открыть для чтения, то завершение работы программы

if ((F\_tel = fopen(File, "r")) == NULL)

{

fprintf(stderr, "\"%s\" : невозможно открыть\n",File);

exit(1);

}

//Чтение числа записей (Count) в файле

if(fread(&Count, sizeof(int), 1, F\_tel) != 1)

{

fprintf(stderr, "\"%s\" : ошибка чтения\n", File);

exit(1);

}

//В цикле осуществляется вывод всех записей

for (i=0; i < Count; i++)

{

fread(Name, 1, MAX\_NAME, F\_tel); //Читается имя

fread(Number, 1, MAX\_NUMBER, F\_tel); //Читается номер

if (ferror(F\_tel)) //Проверяется отсутствие ошибки

{

fprintf(stderr, "\"%s\" : ошибка чтения\n'', File);

exit(1);

}

printf("Фамилия: %s; номер телефона: %s\n", Name, Number);

}

fclose(F\_tel);

}

void main(void)

{

Show( );

}

|  |
| --- |
|  |

**ЛИТЕРАТУРА**

1. Скляров В.А. Программирование на языках Си и Си++.- М.: Высшая школа, 1996.
2. Березин Б.И., Березин С.Б. Начальный курс С и С++.- М.: ДИАЛОГ-МИФИ, 1999.
3. Керниган Б., Ритчи Д. Язык программирования Си.- М.: Финансы и статистика, 1992.
4. Подбельский В.В., Фомин С.С. Программирование на языке Си. Учеб. пособие.- М.: Финансы и статистика, 2000.
5. Павловская Т.А. C/C++, Программирование на языке высокого уровня.- СПб.: Питер, 2005.
6. Касаткин А.И. Профессиональное программирование на языке Си. Системное программирование.- Минск: Высшая школа, 1993.
7. Касаткин А.И. Профессиональное программирование на языке Си. Управление ресурсами.- Минск: Высшая школа, 1993.
8. Касаткин А.И., Вальвачев А.Н. Профессиональное программирование на языке Си. От Turbo C к Borland C++.- Минск: Высшая школа, 1995.

(c) Курсков С.Ю., составление, 2006