**Строковые константы**

Строковая константа представляется последовательностью символов кода *ASCII*, заключённой в кавычки: "...". Она имеет тип char[].

Примеры:

"This is character string"

"Это строковая константа"

"A" "1234567890" "0" "$"

В конце каждой строки *компилятор* помещает нулевой символ ' \0 ', отмечающий конец данной строки.

**Каждая строковая константа, даже если она идентична другой строковой константе, сохраняется в отдельном месте памяти**.

Если необходимо ввести в строку символ кавычек ( " ), то перед ним надо поставить символ обратной косой ( \ ). В строку могут быть введены любые *специальные символьные константы*, перед которыми стоит символ \.

Символ \ и следующий за ним *символ новой строки* игнорируется.

Строковые *константы* размещаются в **статической** памяти. Вся фраза в кавычках является указателем на *место* в памяти, где записана строка. Это аналогично использованию имени массива, служащего указателем на расположение массива. Если это действительно так, то как выглядит оператор, который выводит строку?

/\* Строки в качестве указателей \*/

main( )

{

printf("%s, %u, %c\n", "We", "love", \*"Pascal");

}

Итак, формат %s выводит строку We. Формат %u выводит целое без знака. Если *слово* "love" является указателем, то выдается его *значение*, являющееся адресом первого символа строки. Наконец, \*"Pascal" должно выдать *значение*, на которое ссылается *адрес*, т.е. первый символ строки "Pascal".

Вот что выдаст наша *программа*:

We, 34, P

**Массивы символьных строк и их инициализация**

При определении массива символьных строк необходимо сообщить компилятору требуемый размер памяти. Один из способов сделать это - инициализировать *массив* при помощи строковой *константы*. Например, оператор

char m1[ ]="Только ограничьтесь одной строкой.";

инициализировал внешний по умолчанию *массив* m1 для указанной строки. Этот вид инициализации является краткой формой стандартной инициализации массива

char m1[ ]={

'T','o','л','ь','k','o','

','o','г'','p','a','н','и','ч','ь','т','e','c','ь','

'o','д','н','o','й','

','c','т','p','o','k','o','й','.','\0'

}

Без символа 0 мы имеем *массив* символов, а не строку. Для той и другой формы *компилятор* подсчитывает символы и таким образом получает размер памяти. Как и для других массивов, имя m1 является указателем на первый элемент массива:

m1 == &m1[0], \*m1 == 'T', и \*(m1+1) == m1[1] == 'o'

и т.д.

Действительно, мы можем использовать *указатель* для создания строки. Например:

char \*m3="\n Символьная строка.";

Это почти то же самое, что и

static char m3[ ]="\n Символьная строка.";

Оба описания говорят об одном: m3 является указателем строки со словами "*Символьная строка*". В том и другом случае сама строка определяет размер памяти, необходимой для ее размещения. Однако вид их не идентичен.

### Массив и указатель: различия

В нижеследующем тексте мы обсудим различия в использовании описаний этих двух видов:

static char heart[ ] = "Я люблю язык Cи!";

char \*head = "Я люблю язык Pascal!";

Основное отличие состоит в том, что *указатель* heart является константой, в то время как *указатель* head - переменной. Посмотрим, что на самом деле дает эта разница.

Во-первых, и в том и в другом случае можно использовать операцию сложения с указателем:

for(i=0;i<7;i++)

putchar(\* (heart+i));

putchar('\n');

for(i=0;i<7;i++)

putchar(\* (head+i));

putchar('\n');

В результате получаем

Я люблю

Я люблю

Но операцию увеличения можно использовать только с указателем:

while ((\*head) != '\0') /\* останов в конце строки \*/

putchar(\*(head++)); /\* печать символа и перемещение указателя \*/

В результате получаем:

Я люблю язык Pascal!

Предположим, мы хотим изменить head на heart. Можно так:

head=heart; /\* теперь head указывает на массив heart \*/

но теперь можно и так

heart = head; /\* запрещенная конструкция \*/

Ситуация аналогична x = 5 или 5 = x. Левая часть оператора присваивания должна быть именем переменной. В данном случае head = heart, не уничтожит строку про язык Cи, а только изменит *адрес* , записанный в head.

Вот каким путем можно изменить обращение к *head* и проникнуть в сам *массив*:

heart[13] = 'C';

или

\*(heart+13)='C';

Переменными являются элементы массива, но не имя!

### Указатели и строки

Большинство операций языка *Си*, имеющих дело со строками, работают с указателями. Рассмотрим, например, приведенную ниже бесполезную, но поучительную программу:

/\* Указатели и строки \*/

#define PX(X) printf("X = %s; значение = %u; &X = %u\n",X,X,&X)

main( )

{

static char \*mesg = "Сообщение";

static char \*copy;

copy = mesg;

printf("%s\n",copy);

PX(mesg);

PX(copy);

}

Мы можем подумать, что эта *программа* копирует строку "Сообщение", и при беглом взгляде на *вывод* может показаться правильным это предположение:

Сообщение

mesg = Сообщение; значение = 14; &mesg = 32

copy = Сообщение; значение = 14; &copy = 34

Но изучим *вывод* PX( ). Сначала X, который последовательно является mesg и copy, печатается как строка ( %s ). Здесь нет сюрприза. Все строки содержат "Сообщение".

Третьим элементом в каждой строке является &X, т. е. *адрес* X. Указатели mesg и copy записаны в ячейках 32 и 34, соответственно.

Теперь о втором элементе, который мы называем значением. Это сам X. Значением указателя является *адрес*, который он содержит. Мы видим, что mesg ссылается на ячейку 14, и поэтому выполняется copy.

Смысл заключается в том, что сама строка никогда не копируется. Оператор copy = mesg; создает второй *указатель*, ссылающийся на ту же самую строку.

Зачем все эти предосторожности? Почему бы не скопировать всю строку? Хорошо, а что эффективнее - копировать один *адрес* или, скажем, 70 отдельных элементов? Часто бывает, что *адрес* - это все, что необходимо для выполнения работы.

### Ввод-вывод строк

fgets - прочитать строку из входного потока, включая *символ новой строки*.

Объявления: char \*fgets (s, n, stream)

char \*s;

int n;

FILE \*stream;

gets - прочитать строку из стандартного файла ввода stdin.

Определение: char \*gets (s)

char \*s;

fputs - записать строку в *поток* *stream*.

Определение: int fputs (s, stream)

char \*s;

FILE \*stream;

puts - записать строку в стандартный *файл* вывода stdout. В конце строк записывается *символ новой строки*.

Определение:

int puts (s)

char \*s;

### Обработка строк

Для выполнения описанных в этом подразделе функций необходимо включить в программу *файл* string.h командой

#include <string.h>

***strcat*** - сцепить две строки.

Определение: char \*strcat(s1,s2)

char \*s1, \*s2;

Пример 1:

/\* сцепить две строки \*/

/\* в головном файле conio.h содержится функция очистки экрана clrscr( ) \*/

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{ clrscr();

char destination[25];

char \*blank = " ", \*c = "C++", \*turbo = "Turbo";

strcpy(destination, turbo);

strcat(destination, blank);

strcat(destination, c);

printf("%s\n", destination);

getch();

return 0;

}

***strncat*** - сцепить две строки, причем из второй строки копировать не более n символов.

Определение: char \*strncat(s1,s2,n)

char \*s1, \*s2;

int n;

Пример 2:

/\* cцепить две строки, причем из второй строки

копировать не более n символов \*/

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

clrscr();

char destination[25];

char \*source = "structured ";

strcpy(destination, "programming");

strncat(destination, source, 11);

printf("%s\n", destination);

getch();

return 0;

}

***strcmp*** - сравнить две строки в лексикографическом порядке.

Определение: int strcmp(s1,s2)

char \*s1, \*s2;

Пример 3:

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

char \*buf1 = "aaa", \*buf2 = "bbb", \*buf3 = "ccc";

int ptr;

clrscr();

ptr = strcmp(buf2, buf1);

if (ptr > 0)

printf("buffer 2 is greater than buffer 1\n");

else

printf("buffer 2 is less than buffer 1\n");

ptr = strcmp(buf2, buf3);

if (ptr > 0)

printf("buffer 2 is greater than buffer 3\n");

else

printf("buffer 2 is less than buffer 3\n");

getch();

return 0;

}

***strncmp*** - сравнить первые n символов двух строк.

Определение: int strncmp(s1,s2, n)

char \*s1, \*s2;

int n;

Пример 4:

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

char \*buf1 = "aaabbb", \*buf2 = "bbbccc", \*buf3 = "ccc";

int ptr;

clrscr();

ptr = strncmp(buf2,buf1,3);

if (ptr > 0)

printf("buffer 2 is greater than buffer 1\n");

else

printf("buffer 2 is less than buffer 1\n");

ptr = strncmp(buf2,buf3,3);

if (ptr > 0)

printf("buffer 2 is greater than buffer 3\n");

else

printf("buffer 2 is less than buffer 3\n");

getch();

return(0);

}

***strcpy*** - копировать строку s2 в строку s1.

Определение: char \*strcpy(s1,s2)

char \*s1, \*s2;

Пример 5:

#include <stdio.h>

#include <string.h>

#include <conio.h>

int main(void)

{

clrscr();

char string[10];

char \*str1 = "abcdefghi";

strcpy(string, str1);

printf("%s\n", string);

getch();

return 0;

}

***strncpy*** - копировать не более n символов строки s2.

Определение: char \*strncpy(s1,s2,n)

char \*s1, \*s2;

int n;

Пример 6:

#include <stdio.h>

#include <string.h>

#include <conio.h>

int main(void)

{

clrscr();

char string[10];

char \*str1 = "abcdefghi";

strncpy(string, str1, 3);

string[3] = '\0';

printf("%s\n", string);

getch();

return 0;

}

***strlen*** - определить длину строки (число символов без завершающего нулевого символа).

Определение: int strlen(s)

char \*s;

Пример 7:

#include <stdio.h>

#include <string.h>

#include <conio.h>

int main(void)

{

clrscr();

char \*string = "Borland International";

printf("%d\n", strlen(string));

getch();

return 0;

}

***strchr*** - найти в строке первое вхождение символа n.

Определение:

char \*strchr(s,n)

char \*s;

int n;

Пример 8:

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

clrscr();

char string[20];

char \*ptr, c = 'r';

strcpy(string, "This is a string");

ptr = strchr(string, c);

if (ptr)

printf("The character %c is at position: %d\n", c, ptr);

else

printf("The character was not found\n");

getch();

return 0;

}

***strrchr*** - найти в строке последнее вхождение символа с.

Определение:

char \*strrchr(s,c)

char \*s;

int c;

Пример 9:

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

clrscr();

char string[20];

char \*ptr, c = 'r';

strcpy(string, "This is a string");

ptr = strrchr(string, c);

if (ptr)

printf("The character %c is at position: %d\n", c, \*ptr);

else

printf("The character was not found\n");

getch();

return 0;

}

***strpbrk*** - найти в строке s1 любой из *множества* символов, входящих в строку s2.

Определение:

char \*strpbrk(s1,s2)

char \*s1, \*s2;

Пример 10:

#include <stdio.h>

#include <string.h>

#include <conio.h>

int main(void)

{

clrscr();

char \*string1 = "abcdefghijklmnopqrstuvwxyz";

char \*string2 = "onm";

int \*ptr;

ptr = strpbrk(string1, string2);

if (ptr)

printf("strpbrk found first character: %c\n", \*ptr);

else

printf("strpbrk didn't find character in set\n");

getch();

return 0;

}

***strspn*** - определить длину отрезка строки s1, содержащего символы из *множества*, входящих в строку s2.

Определение:

int strspn(s1,s2)

char \*s1, \*s2;

Пример 11:

#include <stdio.h>

#include <string.h>

#include <alloc.h>

#include <conio.h>

int main(void)

{

clrscr();

char \*string1 = "1234567890";

char \*string2 = "123DC8";

int length;

length = strspn(string1, string2);

printf("Character where strings differ is at position %d\n", length);

getch();

return 0;

}

***strcspn*** - определить длину отрезка строки s1, не содержащего символы cтроки s2.

Определение:

int strcspn(s1,s2)

char \*s1, \*s2;

Пример 12:

#include <stdio.h>

#include <string.h>

#include <alloc.h>

#include <conio.h>

int main(void)

{

clrscr();

char \*string1 = "1234567890";

char \*string2 = "747DC8";

int length;

length = strcspn(string1, string2);

printf("Character where strings intersect is at position %d\n", length);

getch();

return 0;

}

***strtok*** - выделить из строки s1 лексемы, разделенные любым из *множества* символов, входящих в строку s2.

Определение: char \*strtok(s1,s2)

char \*s1, \*s2;

Пример 13:

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

clrscr();

char input[16] = "abc,d";

char \*p;

p = strtok(input, ",");

if (p) printf("%s\n", p);

p = strtok(NULL, ",");

if (p) printf("%s\n", p);

getch();

return 0;

}