

Занятие № 8. Строки

Массивы символов в C++: обзор

В C++ имеется специальный класс для работы со строками, которого, конечно, не было в языке C. В C строки вводились как массивы символов, ограниченные нуль-символом (ASCII-код которого равен нулю), поэтому большое количество программ, написанных на C, используют символьные массивы. Более того, и в C++, несмотря на то, что он имеет класс для работы со строками, находится применение массивам символов. Поэтому термин «строка» имеет два значения: строка в смысле C++ и строка как массив символов. Весь этот раздел будет посвящен тому, как нужно и не нужно использовать символьные массивы.

Все строки обязательно должны оканчиваться нуль-терминатором, и при объявлении размера массива необходимо это учитывать. Когда вы объявляете строковую переменную как массив символов, увеличьте размер массива на один символ для нуль-терминатора.

Все строки обязательно должны оканчиваться нуль-терминатором, и при объявлении размера массива необходимо это учитывать. Когда вы объявляете строковую переменную как массив символов, увеличьте размер массива на один символ для нуль-терминатора.

Ввод строк

При вводе строк операция извлечения из потока `>>` не всегда будет работать так, как вы ожидаете, поскольку строки часто содержат пробелы, которые игнорируются оператором ввода; поэтому вместо оператора ввода вам нужно использовать функцию `getline`. Эта функция вводит заданное количество символов.

Функция `getline`

Пример

```
#include <iostream.h>
int main()
{
char name[80];
cout << "Enter your name: ";
cin.getline(name, sizeof(name) - 1);
cout << "Hello " << name << ", how are you?";
return 0;
}
```

Присвоение значений строкам

Общий метод инициализации строки:

```
char stringVar [stringSize] = stringLiteral;
```

Пример

```
char aString[81] = "Borland C++ 5 in 21 days";
```

```
char Name[] = "Rene Kinner";
```

Функция strcpy

Прототип функции strcpy таков:

```
char * strcpy(char *target, const char *source);
```

Функция копирует строку source в строку target. Функция предполагает, что целевая строка имеет размер, достаточный для того, чтобы вместить содержимое строки-источника.

Пример

```
char name[41];  
strcpy(name, "Borland C++ 5");
```

Переменная name содержит строку "Borland C++ 5".

Функция `strdup` копирует одну строку в другую, при этом отводит необходимое количество памяти для целевой строки.

Функция `strdup`

Прототип функции `strdup` таков:

```
char* strdup( const char *source) ;
```

Функция копирует строку `source` и возвращает указатель на строку-копию.

Пример

```
char *string1 = "Монархия в Испании";  
char *string2;  
string2 = strdup( string1);
```

После того, как будет отведено необходимое количество памяти для строки `string2`, строка `string1` будет скопирована в строку `string2`.

Так как функция `strdup` выделяет память для новой строки, вы должны не забыть освободить потом эту память.

Библиотека строчковых функций предлагает также функцию `strncpy`, копирующую заданное количество символов из одной строки в другую.

Функция `strncpy`

Прототип функции `strncpy` таков:

```
char * strncpy(char *target, const char *source,  
size_t num);
```

Функция копирует `num` символов из строки `source` в строку `target`. Функция не выполняет ни усечение, ни заполнение строки.

Пример

```
char str1[] = "Pascal";  
char str2[] = "Hello there";  
strncpy(str1, str2, 5);
```

Переменная `str1` содержит строку `"Hello1"`. Заметьте, что символ `'1'` строки-приемника, следующий за скопированной частью строки, сохранился.

Определение длины строки

При работе со строками часто бывает нужно знать длину строки. В `STRING.H` объявляется функция `strlen`, возвращающая количество символов в строке, в которое не включается нуль-терминатор.

Функция `strlen`

Прототип функции `strlen` таков:

```
size_t strlen(const char *string);
```

Функция `strlen` возвращает длину строки `string`.

`size_t` — это имя, приписанное типу `unsigned int` оператором `typedef`.

Пример

```
char str[] = "1234567890";
```

```
size_t i;
```

```
i = strlen(str);
```

Переменной `i` будет присвоено значение 10.

Конкатенация строк

Операция **конкатенации** используется достаточно часто, когда новая строка получается **объединением** двух или более строк. Присоединить одну строку к другой можно функцией `strcat`.

Конкатенация строк означает их последовательное присоединение друг к другу.

функция `strcat`

Прототип функции `strcat` таков:

```
char *strcat(char *target, const char *source);
```

Функция добавляет к содержимому целевой строки содержимое строки-источника и возвращает указатель на целевую строку. Функция предполагает, что целевая строка может вместить содержимое объединенной строки.

Пример

```
char string[81];  
strcpy(string, "Borland");  
strcat(string, " C++ 5");
```

Переменная `string` содержит строку "Borland C++ 5".

Функция `strncat` добавляет к содержимому целевой строки указанное количество символов из строки-источника.

Функция `strncat`

Прототип функции `strcat` :

```
char *strncat(char *target, const char *source, size_t num);
```

Функция добавляет к содержимому целевой строки `num` символов из строки-источника и возвращает указатель на целевую строку.

Пример

```
char str1[81] = "Hello I am ";  
char str2[41] = "Keith Thompson";  
strncat(str1, str2, 5);
```

Переменная `str1` теперь содержит строку "Hello I am Keith .

Программа должна выполнять следующие задачи:

- ❑ Предлагает вам ввести строку; ввод не должен превышать 40 символов
- ❑ Предлагает вам ввести вторую строку; ввод не должен превышать 40 символов
- ❑ Выводит число символов, содержащихся в каждой строке
- ❑ Присоединяет вторую строку к первой
- ❑ Выводит результат конкатенации
- ❑ Выводит длину объединенной строки
- ❑ Предлагает вам ввести символ для поиска
- ❑ Предлагает вам ввести символ для замены
- ❑ Выводит содержимое объединенной строки после замены символа

Преобразование строк

Заголовочный файл `STRING.H` библиотеки работы со строками предлагает функции `strlwr` и `strupr` для преобразования символов строк в нижний и верхний регистр соответственно.

Функция `strlwr`

Прототип функции `_strlwr`:

`char* strlwr(char *source)`

Функция преобразует символы верхнего регистра в символы нижнего регистра в строке `source`. Другие символы не затрагиваются. Функция возвращает указатель на строку `source`.

Пример

```
char str [ ] = "HELLO THERE";  
_strlwr(str);
```

Переменная `str` теперь содержит строку "hello there".

Функция `strupr`

Прототип функции `strupr`:

```
char* strupr(char *source)
```

Функция преобразует символы нижнего регистра в символы верхнего регистра в строке `source`. Другие символы не затрагиваются. Функция возвращает указатель на строку `source`.

Пример

```
char str[] = " Borland C++ 5";  
strupr(str);
```

Переменная `str` теперь содержит строку " BORLAND C ++ 5".

Обращение строк

Библиотека `STRING.H` предлагает функцию `strrev` для записи символов в строке в обратном порядке.

Функция `strrev`

Прототип функции `strrev`:

```
char* strrev(char *str)
```

Функция обращает порядок символов в строке `str` и возвращает указатель на строку `str`.

```
char str[] = "Hello";  
strrev(str) ;  
cout << str;
```

Будет выведено "olleH".

Программа выполняет следующие задачи:

- ❑ Запрашивает у вас ввод строки
- ❑ Отображает ваш ввод
- ❑ Выводит вашу строку в нижнем регистре
- ❑ Выводит вашу строку в верхнем регистре
- ❑ Отображает символы, которые вы ввели, в обратном порядке

Поиск символов

Библиотека `STRING.H` предлагает ряд функций для поиска символов в строках. Это функции `strchr`, `strrchr`, `strspn`, `strcspn` и `strpbrk`. Они осуществляют поиск в строках символов и простых символьных шаблонов.

Функция `strchr` определяет первое вхождение символа в строку.

Функция `strchr`

Прототип функции `strchr`:

```
char* strchr(const char *target, int c)
```

Функция находит первое вхождение символа `c` в строку `target`. Функция возвращает указатель на символ в строке `target`, который соответствует заданному образцу `c`. Если символ `c` в строке не обнаруживается, функция возвращает `0`.

Пример

```
char str[81] = "Borland C++ 5";  
char *strPtr;  
strPtr = strchr (str, ' + ' );
```

Указатель `strPtr` теперь содержит адрес подстроки `"++ 5 "` в строке `str`.

Функции `strrchr` определяет последнее вхождение символа в строке.

Функция `strrchr`

Прототип функции `strrchr`:

```
char* strrchr(const char *target, int c)
```

Функция находит последнее вхождение символа `c` в строку `target`. Функция возвращает указатель на символ в строке `target`, который соответствует заданному образцу `c`. Если символ `c` в строке не обнаруживается, функция возвращает `0`.

Пример

```
char str[81] = "Borland C++ 5 is here";  
char* strPtr;  
strPtr = strrchr(str, '+');
```

Указатель `strPtr` теперь указывает на подстроку "+ 5 is here " в строке `str`.

Функция `strspn` возвращает число символов с начала строки, совпадающих с любым символом из шаблона.

Функция `strspn`

Прототип для функции `strspn`:

```
size_t    strspn(const    char    *target,    const    char
*pattern) ;
```

Функция `strspn` возвращает число символов от начала строки `target`, совпадающих с любым символом из шаблона `pattern`.

Пример

```
char str[] = "Borland C++ 5";  
char substr[] = "narlBod ";  
int index;  
index = strspn(str, substr);
```

Этот оператор присваивает 8 переменной `index`, потому что первые восемь символов из `str` содержатся в подстроке `substr`.

Функция `strcspn` просматривает строку и выдает число первых символов в строке, которые не содержатся в шаблоне.

Функция `strcspn`

Прототип функции `strcspn`:

```
size_t strcspn(const char* str1, const char* str2)
```

Функция `strcspn` просматривает строку `str1` и выдает длину подстроки, отсчитываемой с начала строки, символы которой полностью отсутствуют в строке `str2`.

Пример

```
char strng[] = "The rain in Spain";  
int i ;  
i = strcspn(strng, " in");
```

Этот пример возвращает 3 (расположение первого пробела в строке strng) переменной i.

Функция `strpbrk` просматривает строку и определяет первое вхождение любого символа из образца.

функция `strpbrk`

Прототип функции `strpbrk`:

```
char*    strpbrk(const    char*    target,    const    char*  
pattern);
```

Функция `strpbrk` ищет в строке `target` первое вхождение любого символа из образца `pattern`. Если символы из образца не содержатся в строке, функция возвращает 0.

Пример

```
char *str = "Hello there how are you";  
char *substr = "hr";  
char *ptr;  
ptr = strpbrk(str, substr);  
cout << ptr << endl;
```

Вы увидите на экране строку "here how are you", потому что 'h' встречается в строке str раньше, чем 'r'.

Поиск строк

Функция strstr

Прототип функции strstr:

```
char* strstr(const char *str, const char  
*substr) ;
```

Функция ищет в строке `str` первое вхождение подстроки `substr`. Функция возвращает указатель на первый символ найденной в строке `str` подстроки `substr`. Если строка `substr` не обнаружена в строке `str`, функция возвращает 0.

Пример

```
char str[] = "Hello there! how are you";  
char substr[] = "how";  
char *ptr;  
ptr = strstr(str, substr);  
cout << ptr << endl ;
```

Это приведет к выводу строки "how are you", поскольку в строке `str` была обнаружена подстрока "how". Указатель `ptr` содержит адрес остатка первоначальной строки, начинающегося с подстроки "how".