Тема 5

Строки

Строки

Строка – конечная последовательность символов. Количество символов в строке называется её длиной (текущей длиной). Допустимы строки нулевой длины.

Основные операции над строками:

- поиск символов в строке;
- замена символов в строке;
- поиск, замена, удаление подстрок;
- вставка в строку новой подстроки;
- сцепление (конкатенация) двух строк.

В результате выполнения этих операций длина строки может измениться!

Средства C++ для работы со строками

Для работы со строками символов язык C++ предоставляет две возможности:

- работу с массивами данных типа char (функции для работы с такими данными описаны в файлах <string.h> или <cstring>
- класс string, описанный в файле <string>

Работа со строками фиксированной длины

Если длина строки известна, не равна нулю и не изменяется, для её хранения можно использовать массив соответствующего размера.

Пример: номер зачётной книжки всегда состоит из 7 символов

```
char nz[7];
nz[0] = '1'; nz[1] = '5'; nz[2] = '2'; nz[3] = '3';
nz[4] = '1'; nz[5] = '6'; nz[6] = '9';
…
for (int i=0; i<7; i++)
 cout << nz[i];
```

Неудобства такого подхода – строку можно обрабатывать только посимвольно!

Нуль-терминированные строки

Нуль-терминированные строки – строки, в которых символ с кодом 0 является признаком конца строки и не входит в её состав.

Нуль-терминированные строки используются:

- в строковых константах;
- при вводе-выводе;
- в стандартных функциях, описанных в файле <string.h> или <cstring>

Особенности работы с нуль-терминированными строками

Если мы сами формируем строку, то для совместимости со стандартными средствами языка должны правильно размещать признак конца строки. Так, результат вывода следующей программы не определен:

```
char s[20];
s[0]=1;
cout << strlen(s) << endl;
```

Другой особенностью работы со строками является отсутствие каких-либо проверок на действительную длину строки. Так, написав

```
char *s;
cin >> s;
```

мы, скорее всего, что-то куда-то введем…

Функции для работы со строками из заголовочного файла <string.h>

- **`char* strcat(char *s1, const char *s2)`**

Функция добавляет s2 к s1 и возвращает s1. В конец результирующей строки добавляется нуль-символ.

- **`char* strchr(char *s, int ch)`**

Функция возвращает указатель на первое вхождение символа ch в строку s, если его нет, то возвращается NULL.

- **`int strcmp(const char *s1, const char *s2)`**

Функция сравнивает строки и возвращает отрицательное (если s1 меньше s2), нулевое (если s1 равно s2) или положительное (если s1 больше s2) значение. Строки сравниваются лексикографически.

- **`char* strcpy(char *s1, const char *s2)`**

Функция копирует s2 в s1 и возвращает s1.

Функции для работы со строками из заголовочного файла <string.h> (часть 2)

- **`size_t strlen(const char *s)`**

Функция возвращает длину строки (без учета символа завершения строки).

- **`char* strncat(char *s1, const char *s2, size_t n)`**

Функция добавляет не более n символов из s2 к s1 и возвращает s1. Первый символ s2 пишется на место завершающего нуль-символа строки s1. Если длина строки s2 меньше n, переписываются все символы s2. К строке s1 добавляется нуль-символ. Если строки перекрываются, поведение не определено.

- **`int strncmp(const char *s1, const char *s2, size_t  n)`**

Функция сравнивает первую строку и первые n символов второй строки и возвращает отрицательное (если s1 меньше s2), нулевое (если s1 равно s2) или положительное (если s1 больше s2) значение.

Функции для работы со строками из заголовочного файла <string.h> (часть 3)

- **`int strncmp(const char *s1, const char *s2, size_t  n)`**

Функция сравнивает первую строку и первые n символов второй строки и возвращает отрицательное (если s1 меньше s2), нулевое (если s1 равно s2) или положительное (если s1 больше s2) значение.

- **`char* strncpy(char *s1, const char *s2, size_t n)`**

Функция копирует не более n символов из s2 в s1 и возвращает s1. Если длина исходной строки превышает или равна n, нуль-символ в конец строки s1 не добавляется. В противном случае строка дополняется нуль-символами до n-го символа. Если строки перекрываются, поведение не определено.

Функции для работы со строками из заголовочного файла <string.h> (часть 3)

- **`size_t strspn(const char *s1, const char *s2)`**

Функция возвращает индекс первого символа из s1, отсутствующего в s2.

Примеры :

```
strspn(”abracadabra”, ”abr”) // результат - 4
strspn(”abracadabra”, ”abc”) // результат - 2
strspn(”abracadabra”, ”abcdr”) // результат - 11
```

- **`char* strstr(char *s1, const char *s2)`**

Функция выполняет поиск первого вхождения подстроки s2 в строку s1. В случае удачного поиска, возвращает указатель на элемент из s1, с которого начинается s2, и NULL в противном случае.

Общие замечания по использованию стандартных функций

- Все функции, изменяющие строки, не проверяют правильность выделения памяти. Ошибки не вызывают никаких сообщений компилятора!
- MS Visual Studio считает большинство перечисленных функций небезопасными (deprecated) и предлагает свои, более надёжные функции(например, strcpy_s вместо strcpy). Однако это может привести к непереносимости программы на другие компиляторы!

  Для устранения этих предупреждений рекомендуется использовать директиву

  ```
  #pragma warning(disable: 4996)
  ```

Пример программы, работающей со строками (постановка задачи)

Ввести строку длиной не более 100 символов и вывести ее в обратном порядке (т.е. вместо строки «мама мыла раму» вывести строку «умар алым амам») Обратите внимание на особенность ввода строк, которые могут содержать пробелы!

Пример программы, работающей со строками (решение)

```
#include <iostream>
#include <cstring>
using namespace std;

int main (void) {
 char *s = new char[101], c;
 cin.getline(s, 101);
 // cin >> s; - неверно!
 int len=strlen(s);
 if (len>0) {
  for (int i=0; i<len-1; i++, len--) {
   c=s[len-1];
   s[len-1]=s[i];
   s[i]=c;
  }
 }
 cout << s << endl;
 delete [] s;
 return 0;
}
```