**Лекция 9**
**ОБРАБОТКА СТРОК В СРЕДЕ VISUAL C++ 2010**

**Цель лекции.** Исследовать особенности обработки строк (массивов символьных переменных) в среде Visual C++ 2010.

**Основные вопросы лекции.**

1. Описание строк в среде Visual C++ 2010.
2. Ввод и вывод строк в среде Visual C++ 2010.
3. Функции обработки строк в среде Visual C++ 2010.
4. Строки в среде Visual C++ 2010.
5. Функции превращения типов.

## 1. Описание строк в среде Visual C++ 2010

В языке C++ массив типа **char** - это одномерный массив, состоящий из символов:

```
char A[11];
```

**Символьная строка** – это последовательность символов, дополненная специальным символом-ограничителем, указывающим конец строки. Ограничивающий символ записывается управляющей последовательностью **"\0"**. Для такой символьной строки применяют название "строка С" (была предложена разработчиком языка). В других ветвях языка C++ существуют другие представления символьных строк.

Каждый символ в строке занимает один байт.

Символьная константа **"\0"** , ограничивающая символьную строку, называется нулевым байтом. Ее следует учитывать при определении соответствующего массива символов: если строка должна содержать **N** символов, то в определении массива следует указать **N + 1** элемент.

Например, определение

```
char A[11];
```

означает, что строка содержит 10 элементов типа **char** (символов), а последний байт зарезервирован для нулевого байта.

В качестве символов могут использоваться.

1. Прописные буквы латинского и российского алфавитов.
2. Строчные буквы латинского и российского алфавитов.
3. Цифры от 0 до 9.
4. Символы пунктуации: **. ;** и т. п.
5. Символьные константы.
6. Управляющие символы.
7. Пробел.
8. Шестнадцатеричные цифры.

Символьные массивы при их определении могут инициализироваться как обычный массив:

```
char A[13]={'K','h','a','r','k','o','v','-','2','0','1','4'};
```

а могут – как символьная строка. Символьная строка – это последовательность символов, заключенных в двойные кавычки:

```
char A[13]="Kharkov-2014";
```

Отличие этих двух способов заключается в том, что во втором случае автоматически будет прибавлен еще и нулевой байт.

Для выделения места в памяти под символьный массив произвольного размера необходимо указать количество символов в строке (если оно известно) или задать явно больший размер массива:

```
char B[80]= "Это инициализация массива символов";
```

В данном случае указан размер массива 80, хотя для размещения этой строки необходимо было указать 35 ( с учетом нулевого байта).

Инициализировать символьный массив можно и без указания его размера:

```
char B[ ]= "Это инициализация массива символов";
```

В этом случае компилятор сам определит необходимый размер памяти под этот массив.

## 2. Ввод и вывод строк в среде Visual C++ 2010

В языке C++ при работе со строками, как и при работе с числовыми переменными, можно использовать операторы ввода в поток >> и вывода из потока <<. Но оператор ввода >> игнорирует пробелы, которые вводятся.

**Пример 1.** В качестве примера использования операторов потоковых ввода и вывода символьных массивов исследуем следующую программу:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
  char stroka[30], A[50];                          //Объявление символьных переменных
  cout<<"Vvedite ctroku  <  30  simvolov:"<<endl;
  cin>>stroka;                                     //Ввод символьной переменной  stroka
  cout<<"Vu vveli stroku:"<<endl;
  cout<<stroka<<endl;                              //Вывод символьной переменной  stroka
  cout<<"Vvedite novuyu ctroku  <  30  simvolov::"<<endl;
  cin>>stroka;                                     //Ввод новой символьной переменной  stroka
  cout<<"Vu vveli novuyu stroku: "<<stroka<<endl;  //Вывод символьной переменной  stroka
  cout<<"Vvedite novuyu ctroku  <  50  simvolov::"<<endl;
  cin>>A;                                          //Ввод символьной переменной  A
  cout<<"Vu vveli novuyu stroku: "<<A<<endl;       //Вывод символьной переменной  A
  cout<<"A0= "<<A[0]<<endl;                        //Вывод 0-го элемента  переменной  A
  cout<<"A8= "<<A[8]<<endl;                        //Вывод 8-го элемента  переменной  A
getch();
return 0;
}
```

После выполнения программы экран будет иметь следующий вид:

```
Vvedite ctroku  <  30  simvolov:
wwwwwwwwwwwqqqqqqqqq
Vu vveli stroku:
wwwwwwwwwwwqqqqqqqqq
Vvedite novuyu ctroku  <  30  simvolov::
aaaaaddddddffff
Vu vveli novuyu stroku: aaaaaddddddffff
Vvedite novuyu ctroku  <  50  simvolov::
ssssddddgggg
Vu vveli novuyu stroku: ssssddddgggg
A0= s
A8= g
```

При вводе символов с пробелами, последние игнорируются операторами ввода >> и вывода <<. Поэтому при работе со строками вместо этих операторов целесообразней использовать следующую функцию:

**getline(ИмяСимвольнойПеременной, РазмерСимвольнойПеременной);**

где **ИмяПеременной** указывает на строку, в которую осуществляется ввод; **РазмерПеременной** – число символов, подлежащих вводу.

Реализация применения этого оператора описана в нижеследующей программе.

**Пример 2.** Исследуем использование функции **getline()**.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
  char stroka[70], A[50];                          //Объявление символьных переменных
  cout<<"Vvedite ctroku  <  30  simvolov:"<<endl;
  cin.getline(stroka,20);                          //Ввод символьной переменной  stroka
```

```
  cout<<"Vu vveli stroku:"<<endl;
  cout<<stroka<<endl;                                   //Вывод символьной переменной  stroka

cout<<"Vvedite novuyu ctroku  <  30  simvolov:"<<endl;
  cin.getline(A,20);                                    //Ввод символьной переменной  A
  cout<<"Vu vveli novuyu stroku: "<<endl<<A;             //Вывод символьной переменной  A
getch();
return 0;
}
```

После выполнения программы экран будет иметь следующий вид:

```
Vvedite ctroku  <  30  simvolov:
www sss ttttt
Vu vveli stroku:
www sss ttttt
Vvedite novuyu ctroku  <  30  simvolov:
qqq yyyyyyy ppppppppp
Vu vveli novuyu stroku:
qqq yyyyyyy ppppppppp_
```

При использовании функции **getline() РазмерПеременной** меньше или равен размеру объявленной символьной строки.

Объявленная в вышеприведенной программе строка **stroka** может принять 70 символов. Например, если в функции **getline(stroka, 20)** указано число 20, то при вводе строки с 37 символами введется строка из 30 символов. Остальные символы будут отброшены.

### 3. Функции обработки строк в среде Visual C++ 2010

Для работы со строками существуют специальные функции, описание которых находится в заглавном файле **string.h**, который необходимо включать в программу оператором **include**:

```
#include <string.h>;
```

Рассмотрим функции, которые используются наиболее часто.

#### 3.1. Определение длины строки

Очень часто при работе со строками необходимо знать, сколько символов содержит строка. Для получения информации о длине строки используется функция **strlen()**. Вызов функции имеет вид:

```
strlen (ИмяСимвольнойПеременной);
```

Функция возвращает значение на единицу меньшее, чем отводится под массив (без учета нулевого байта).

**Пример 3**. Исследуем использование функции **strlen()**.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
  char A[80];
  int k;
  cout<<"Vvedite ctroku  <  30  simvolov:"<<endl;
  cin.getline(A,30);                                    //Вызов функции getline() для ввода массива A
  cout<<"Vu vveli stroku: "<<endl<<A;                   //Вывод символьной переменной  A
  k=strlen(A);                                          //Вызов функции strlen(A) для определения количества
                                                        //символов в массиве A
  cout<<endl<<"k= "<<k<<endl;                           //Вывод переменной k (кол. символов в A)
getch();
return 0;
}
```

Вид экрана после работы программы:

```
Vvedite ctroku  <  30  simvolov:
wwwwwwwww  RRRRRRRRR  xxxxxxxxxx
Vu vveli stroku:
wwwwwwwww  RRRRRRRRR  xxxxxxx
k= 29
```

### 3.2. Копирование строк

Значения строк могут копироваться из одной строки в другую. Копирование осуществляется с помощью следующих функций.

Функция **strcpy(S1,S2)** используется для побайтного копирования строки **S2** в строку **S1**. Копирование прекращается при достижении нулевого байта. Поэтому длина строки **S1** должна быть достаточно большой, чтобы в нее поместилась строка **S2**.

**Пример 4**. Исследуем использование функции **strcpy()**:

```
#include <string.h>                          //Добавление библиотеч. файла для работы со строками
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
  char A[80];
  int k;
  cout<<"Vvedite ctroku  <  30  simvolov:"<<endl;
  cin.getline(A,30);                         //Ввод символьной переменной  A
  cout<<"Vu vveli stroku: "<<endl<<A;
  strcpy(A, "Proverka kopirovaniya");        //Вызов функции strcpy(A) для копирования строки в строку
  cout<<endl<<"Novaya stroka: "<<A;          //Вывод новой символьной переменной  A
getch();
return 0;
}
```

Вид экрана после работы программы:

```
Vvedite ctroku  <  30  simvolov:
aaaaaaaaaaaaaaa ddddddd
Vu vveli stroku:
aaaaaaaaaaaaaaa ddddddd
Novaya stroka: Proverka kopirovaniya_
```

Функция **ctrncpy()** отличается от функции **ctrcpy()** тем, что включает еще один параметр. Он указывает количество символов, которые необходимо копировать из строки **S2** в строку **S1**. Функция имеет вид:

**ctrncpy (S1, S2, n);**

где **n** – количество символов (целое без знака).

Если длина **S1** меньше длины **S2**, то происходит урезание символов.

**Пример 5**. Исследуем использование функции **strncpy()**:

```
#include <string.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
  char A[]="0123456789";                     //Ввод символьной переменной  A
  char B[]="qwertyuiop";                     //Ввод символьной переменной  B
  cout<<"S2= "<<A<<endl;                     //Вывод символьной переменной  A
  cout<<"S1= "<<B<<endl;                     //Вывод символьной переменной  B
  strncpy(B,A,4);
```

```
  cout<<"S2new= "<<B<<endl;                    //Вывод новой символьной переменной  B
getch();
return 0;
}
```

Вид экрана после работы программы:

```
S1= 0123456789
S2= qwertyuiop
S1new= 0123456789
S2new= 0123tyuiop
```

То есть, из строки **S2** в строку **S1** будут скопированы 4 первых символа и размещены в начале короткой строки **S2**.

### 3.3. Присоединение строк

Присоединение (**конкатенация**) строк используется для образования новой строки символов из двух и более исходных строк. Для этой цели используются функции

**strcat (S1, S2) и strncat (S1, S2, n)**;

Функция **strcat (S1, S2)** присоединяет строку **S2** к строке **S1** и помещает ее в массив, где находилась строка **S1**. Строка **S2** не изменяется. Вновь полученная строка S1 автоматически завершается нулевым байтом.

**Пример 6**. Исследуем использование функции **strcat()**:

```
#include <string.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
  char A[30], B[30];
  strcpy(A, "Hello, ");                        //Копирование строки "Hello, " в строку A
  strcpy(B, "World!");                         //Копирование строки "World!" в строку B
  cout<<"A= "<<A<<endl;
  cout<<"B= "<<B<<endl;
  strcat(A, B);                                //Присоединение строки B к строке A
  cout<<endl<<"A= "<<A<<endl;
  cout<<"B= "<<B<<endl;
getch();
return 0;
}
```

Результат выполнения программы следующий

```
A= Hello,
B= World!

A= Hello, World!
B= World!
```

Функция **strncat (S1, S2, n)** также осуществляет присоединение строк, однако присоединяет лишь указанное в третьем параметре количество символов, например:

```
char S1[80]="Dlya prodolgeniya ";
char S2[80]="nagat knopku OK !";
strncat(S1,S2,7);
cout<<S1<<endl;
```

В результате на экран будет выведена строка:

**Dlya prodolgeniya nagat knopku OK !**

### 3.3. Сравнение строк

В библиотеке функций **string.h** есть функции, выполняющие посимвольное сравнение двух строк.

Функция **strcmp (S1, S2)** сравнивает строки **S1** и **S2**. После сравнения строк данная функция возвращает одно из следующих значений:

**<0** – если строка **S1** меньше чем **S2**;
**=0** – если строки эквивалентные;
**>0** – если **S1** больше, чем **S2.**

Эта функция проводит сравнение строк, различая прописные и строчные буквы, например:

```
char S1[]="аааааааавв";
char S2[]="БББББББББГГ";
int к;
k= strcmp(S1,S2);
cout<<"k= "<<k<<endl;
```

Переменной k будет присвоено негативное значение (-1), несмотря на равное количество символов в строках. Строка **S1** меньше строки **S2** по той причине, что прописные буквы имеют код символов меньше чем те же строчные буквы.

На экран будет выведено:

**k= -1**

Функция **stricmp (S1, S2)** сравнивает строки **S1** и **S2,** не различая регистра символов.

Функция **strnimp(S1, S2, n)** проводит сравнение определенного числа (**n**) первых символов двух строк. Регистр символов при этом учитывается.

Кроме рассмотренных функций есть большое количество других функций обработки строк.:

- функции превращения строк (превращение элементов символьной строки из одного регистра в другой);
- функции обращения строк (меняет порядок прохождения символов в строке на обратную);
- функции поиска символов (одного или группы символов в строке).

## 4. Строчные переменные в Visual C++ 2010

Язык C++ позволяет объявлять и применять строчные переменные, т.е. переменные, которые содержат строки:

```
string A;
```

В данном случае строчная переменная **A** инициализирована пустой строкой.

Строчную переменную можно инициализировать строчным литералом, используя оператор

```
string A= "Kharkov-2014";
```

Впоследствии, переменной **A** можно присвоить другую строку, используя оператор присвоения:

```
A = "Visual C++ 2010";
```

Эта строка складывается из 16 символов. В этом случае говорят, что длина строки **A** равна 16. Для вычисления текущей длины строки можно применять или функцию **length()**, или функцию **size()**.

Ссылаться на отдельные символы строки можно с помощью индексов, как будто строка является массивом. Таким образом, в предыдущем примере ячейка **A[0]** содержит символ **“V”**, а ячейка **A[10]** - символ **“+”**.

Строки можно сравнивать, используя обычные операторы сравнения. Причем, можно проверять не только равенство, но и какая из строк предшествует другой.

Строки можно **конкатенировать** (присоединять), используя оператор «+». В итоге образуется новая строка, состоящая из двух частей: первой и второй строк, записанных последовательно. Например, если в программе поместить объявление

```
string B = "Com";
```

то операторы **string C=B+"puter";** и **B+= "puter";**

присвоят переменным **C** и **B** значение строки "**Computer**".

Аналогично, к строке можно приписать отдельный символ: **D+="p";**

## 5. Функции превращения типов

Функции превращения типа используются для превращения чисел, введенных в виде символьных строк, в числовое представление, выполнение определенных математических операций над ними и обратное превращение в строку символов.

Эти функции размещаются в заглавном файле **stdlib.h**.

Функции **atof()**, **atoi()**, **atol()** – превращают строку символов соответственно в число типа **float, int, long**.

Функции **ecvt(), fcvt(), gcvt()** – превращают число с плавающей точкой типа **double** в строку символов.

При использовании функции **ecvt()** десятичная точка и знак числа не включаются в полученную строку.

При использовании функции **fcvt()** округляет приобретенное значение к заданному числу цифр.

При использовании функции **gcvt()** включает символ десятичной точки.

Функции **itoa(), ltoa(), ultoa()** превращают числа типа **int, long и unsigned long** в строку символов.

Функции **strtod() и strtol()** превращают строку символов соответственно в число типа **double** и **long**.

Рассмотрим использование некоторых из этих функций.

Функции

```
int atoi (const char *ptr);
long atol (const char *ptr);
```

превращают строку символов, на которую указывает указатель **ptr** в число типа **int**.

Функция **gcvt** имеет прототип

```
char *gcvt (double val, int sig, char *buf);
```

и превращает число **val** типа **double** в строку с помещением ее в буфер **buf** (**int sig** – число цифр, которые подлежат превращению).

Если число цифр, которые подлежат превращению, меньше числа, указанного в **sig**, то в преобразованном числе указывается знак и десятичная точка. Младшие разряды дробной части отбрасываются. Иначе - число превращается в экспоненциальную форму.

### Выводы.

В языке С++ строка представляется как одномерный массив, элементы которого имеют тип **char**.

Следовательно, символьная строка – это одномерный массив типа **char**, заканчивающийся нулевым байтом.

### Вопросы для самоконтроля.

1. В программе на С++ определен массив **char A [11]**. Это означает, что строка содержит:
   1)10 символов
   2)11 символов
   3) 12 символов
2. В языке С++ для копирования строк используется функция:
   1) **strlen( )**
   2) **strcpy( )**
   3) **strcat( )**
3. В языке С++ конкатенация строк – это:
   1) Копирование одной строки в другой
   2) Сравнение двух строк
   3) Присоединение одной строки к другому
4. Что такое символьная строка?
5. Что такое нулевой байт?
6. Как инициализируется символьный массив?
7. Как объявляется символьный массив?
8. Для чего применяется функция **getline()**?
9. Какие аргументы используются в функции **getline()**?
10. Для чего применяется функция **strcpy()**?
11. Какие аргументы используются в функции **strcpy()**?
12. Для чего применяется функция **strcat()**?
13. Какие аргументы используются в функции **strcat()**?
14. Для чего применяется функция **strncat()**?
15. Какие аргументы используются в функции **strncat()**?