## СТРОКИ

## как одномерные массивы данных типа *char*

(терминальные строки)

В языке Си нет данных типа «*строка*». Работа со строками реализована путем использования одномерных массивов типа *char*, т.е. *строка* — это одномерный массив символов, заканчивающийся нулевым байтом.

В С++ используется тип *string* (изучайте сами).

**Нулевой байт** – это байт, каждый бит которого равен нулю, при этом для него определена символьная константа **\0**′ (признак окончания строки, или *нуль-символ*).

По положению нуль-символа определяется фактическая длина строки.

Если строка должна содержать **k** символов, то в ее описании размер должен быть на 1 больше, например

char s [ 7 ]; – означает, что строка **s** может содержать не более шести символов.

Отсутствие нуль-символа и выход указателя при просмотре строки за ее пределы – распространенная ошибка, признак которой – вывод непонятного «мусора».

Строку можно инициализировать строковой константой, например:

char S[] = "Работа со строками";

будет выделено и заполнено 19 байт — 18 на символы и 19-й на нуль-символ.

В конце строковых констант указывать символ '\0' не нужно, он будет добавлен автоматически.

Символ '\0' **необходимо** добавлять тогда, когда строка формируется посимвольно в процессе выполнения программы.

При работе со строками можно пользоваться указателями, например:

char \*str = "БГУИР";

Такая декларация строки – единственный случай, когда в коде программы можно использовать операцию «=».

Операция char \*str = "БГУИР" создает не строковую переменную, а *указатель на строковую константу.* 

Изменения такой строки (причем это касается не только адреса памяти, но и его размера) приводит к ошибочным результатам.

Знак равенства перед строковым литералом означает инициализацию, а не присваивание.

Для ввода строк обычно используются две стандартные функции:

- 1) **scanf** (**scanf\_s**) выполняет ввод строк при помощи формата %s **до первого пробела** (символ **&** перед *Именем* указывать не надо, т.к. строка это массив, а имя массива это его адрес);
- 2) *gets* выполняет ввод строк, содержащих *пробелы* (ввод завершается по нажатию клавиши *Enter*).

Пример:

char str [ 21 ];

scanf ("%s", str); или gets (str);

Обе функции автоматически ставят в конец строки нулевой байт.

Ввод указанной строки **до первого пробела** можно выполнить в потоке cin >> str;

Для ввода строк, содержащих *пробелы* в потоке cin.get(...) или cin.getline(...)

Вывод строк производится с помощью функций *printf* (*printf\_s*) или *puts* (до нулевого байта).

Функция *printf* не переводит курсор после вывода на начало новой строки, а *puts* после вывода строковой информации автоматически переводит курсор в начало новой строки.

Вывод строк можно выполнять в потоке.

Примеры для строки: char str [ 21 ] = "Minsk";

## Функциями: Аналог в потоке:

```
printf ("%s", str); cout << str;
printf ("%20s", str); cout << setw(20) << str;
puts (str); cout << str << endl;
puts ("Minsk"); cout << "Minsk" << endl;</pre>
```

Большинство действий со строками в Си выполня-ются с помощью стандартных функций, большинство из которых описаны в файле *string.h*.

Приведем наиболее часто используемые функции.

1. Результат функции **strlen** (S) – длина (*len*) строки (*str*) **S** (количество символов без нулевого байта), например:

```
char s1[15] = "Минск!\0", s2[20] = "БГУИР–Ура!";
cout << strlen(s1) << ", " << strlen(s2) << endl;
Результат : 6,10
```

- 2. Функция strcpy (s1, s2) копирует (copy) содержимое строки s2 в строку s1.
- 3. Функция strncpy (s1, s2, n) копирует n символов строки s2 в строку s1 (если n меньше длины строки s2, то окончание строки s1 формируем сами, т.е.  $s1[n] = {0}$ ;

Операция присваивания между строками в языке Си не определена и может быть выполнена либо в цикле посимвольно, либо с использованием функций 2) и 3).

## Примеры:

```
char s1[81], s2[81] = "Minsk";
1) strcpy (s1, s2);
```

- Значение строки s1 будет Minsk
- 2) strncpy (s1, s2, 3); s1[3] = '\0';
- Значение строки s1 будет Min
  - 3) strncpy (s1, s2 + 2, 2);
- (s2 + 2) подстрока строки s2, начинающаяся с позиции 2 s1[2] = '\0';
- Значение строки s1 будет ns

- 4. Функция **strcat** (s1, s2) присоединяет строку s2 к строке s1 (т.е. формально s1 = s1 + s2). Нулевой байт, который завершал строку s1, заменяется первым символом строки s2 (concatenate сцеплять).
- 5. Функция **strncat** (s1, s2, n) присоединяет *n* символов строки *s*2 к строке *s*1. Нулевой байт, который завершал строку *s*1, автоматически сдвигается на *n* позиций.

Примеры:

char s1[81] = "Minsk";

1) strcat(s1, "-2011");

Значение строки s1 будет Minsk-2011

2) strncat(s1, "-2011", 3);

Значение строки s1 будет Minsk-20

- 6. Функция **strcmp** (s1, s2) сравнивает (*compare*) строки s1 и s2, ее результат (последовательно символ за символом сравниваются их коды до первого различия)
  - Меньше 0, если s1 < s2;
  - Больше 0, если s1 > s2;
- Равен 0, если строки равны, т.е. содержат одинаковое число одинаковых символов.
- 7. Функция *strncmp*(s1, s2, n) сравнивает *n* символов строк s1 и s2, ее результат аналогичен функции *strcmp*.

**Внимание!** Во всех функциях пунктов 1-7 используются **строки**, а не **символы** строк, т.е. например, если к строке "Minsk" нужно присоединить символ «\*»:

```
strcat ("Minsk", "*"); - Правильно, т.к. обе строки strcat ("Minsk", '*'); - Ошибка, т.к. '*' – символ, а не строка
```

- 8. Функции преобразования строки **S** в число:
- целое: int atoi (S);
- длинное целое: long **atol** (S);
- действительное: double **atof** (S);

При возникновении ошибки функции возвращают значение **0**, например

```
atoi ("123") результат 123 atoi ("123asd") результат 123 atoi ("a123") результат 0
```

- 9. Функции преобразования числа **V** в строку **S**:
- целое: *itoa* (V, S, kod);
- длинное целое: *Itoa* (V, S, kod);
- 2 ≤ *kod* ≤ 36, для десятичных чисел со знаком *kod* = 10.

Функции пунктов 7 и 8 описаны в файле *stdlib.h*.

В консольных приложениях не выводятся символы русского алфавита, что вызвано различными стандартами кодировки символов кириллицы.

Для вывода строки, содержащей буквы русского алфавита используйте дополнительные возможности той сред, в которой вы работаете, либо оформляйте свои сообщения с помощью английских текстов, или просто «латиницей».

Если не получается, рассмотрим возможность вывода строки, содержащей русский алфавит с использованием функции *CharToOem*, описанной в файле *windows.h*.

```
#include <iostream.h>
#include <windows.h>
char bufRus [255]; // Глобальная переменная
```

```
char* Rus (const char *text) { - Дополнительная
  CharToOem (text, bufRus); - функция
  return bufRus;
                - преобразования
void main()
  char s[81] = "Muhck!", ss[100];
  cout << Rus("Город ") << Rus(s) <<endl;
  cout << Rus("Введи строку:");
  cin >> ss;
  cout << Rus(" Строка: ") << ss << endl;
  return 0;
```

Рассмотрим некоторые примеры работы с терминальными строками.

Для получения более высоких баллов старайтесь не использовать стандартные функции. Старайтесь написать либо свои функции, либо коды их аналогов.

1. В строке, разделенной пробелами, найти количество слов длинной 3 символа и вывести их на экран:

```
char str [ 81 ], - Исходная строка
   res [ 81 ]; - Строка-результат
int len str, - Длина исходной строки
  len word = 0, - Длина текущего слова
          - Индекс символа в строке
  kol = 0, - Количество найденных слов
   pos; - Начало нужного слова
puts ("Input String");
gets (str); - Ввод исходной фразы
len str = strlen(str);
```

Для обработки строк, состоящих из слов, разделенных пробелами, проще использовать символ-ПРОБЕЛ в качестве признака окончания каждого слова. Но для последнего слова в строке это не сработает, поэтому используем явное добавление пробела ('' – между кавычками символ-пробел) после последнего слова:

```
if(str[len_str-1] != ' ') { - Если последний не пробел, strcat (str, " "); - добавляем строку-пробел и len_str++; - увеличиваем длину строки }
```

```
for( i = 0; i < len str; ++i ) - Анализ символов строки
  if(str[i] != ' ') - Если символ не пробел,
      len word++;
                         увеличиваем длину слова

    Иначе, если – пробел:

  else {
      if (len word == 3) { - Анализ длины
   kol++;
                     - Считаем количество
   pos = i - 3;
                - Начало этого слова
         strncpy(res, str+pos, 3); - Создаем слово
         res[3] = '\0'; - Заканчиваем слово
         puts(res); - Выводим на экран
      len word = 0; - Обнуляем длину для
             поиска следующего слова
printf("\n Kol-vo = %d\n", kol);
```

```
2. В строке str найти максимальное слово и его длину
  (продолжение Примера 1):
  int len max = 0; - Длина максим-го слова
  for(i = 0; i < len str; ++i) - Просмотр строки
    if(str[i] != ' ') - Если не пробел
     len_word++; - Увеличиваем длину
  else { - Иначе, если – пробел
       if(len word > len max) { - Сравниваем длины
          len max = len word; - Меняем на большую
     pos = i - len max; - Начало макс-го слова
```

```
len_word = 0; - Обнуляем длину для поиска следующего слова
```

- Конец else
  strncpy ( res, str + pos, len\_max );
- Создаем строку максимальной длины *len\_max*, которая начинается с индекса *pos*

```
res[len_max] = '\0'; - Заканчиваем строку
printf("\n Max Word = %s, len = %d\n", res, len_max);
```

- Выводим найденную максимальную строку и ее длину