

Севастопольский государственный университет  
Кафедра «Информационные системы»

Курс лекций по дисциплине

**"АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ"**

(АиТ)

2 семестр

Лектор: Сметанина Татьяна Ивановна

# Лекция 1

## Строки в C/C++

# Строки

В C++ существуют 2 вида строк:

- С-строки, унаследованные от своего предшественника – языка С
- строки класса **string**.

Рассмотрим каждый из этих видов строк более подробно.

# C-строки

В тексте программы строка описывается как последовательность символов, заключенная в двойные кавычки.

Например: `"Hello world"`

Размер памяти, отводимой под строку, равен

**1 байт \* количество символов + 1**

Представление в памяти:

H	e	l	l	o		w	o	r	l	d	\0
---	---	---	---	---	--	---	---	---	---	---	----

C-строки не содержат никакой дополнительной информации !!!

В конце строковой переменной обязательно должен быть служебный символ `'\0'`.

# C-строки

Объявление строковых переменных:

```
char Имя_массива[Максимальный_размер_строки + 1];
```

Пример:

```
char my_cstring[11];
```

В объявленной выше строковой переменной **my\_string** можно хранить строку длиной не более 10 символов. Символ '**\0**' называется нуль-символом (или терминирующим нулем).

Символ '**\0**' используется как специальная метка, обозначающая окончание строки.

Отличие C-строки от обычного массива , — это то, что строковая переменная должна содержать символ '**\0**'.

# C-строки

Важно отличать длину строки от размера памяти под переменную:

```
char my_cstring[11]="Hello";
```

```
cout<< sizeof(my_cstring)<<" "<<strlen(my_cstring) ;  
// 11  5
```

Вспомним, что C-строка – это указатель на начало строки:

```
int strlen(char *s) {  
    int n;  
    for (n = 0; *s != '\0' ; s++)  
        n++;  
    return n;  
}
```

# C-строки

## Особенности ввода-вывода строк:

Выводить строки в стандартный поток вывода можно с помощью:

```
printf("%s",my_cstring);  
puts(my_cstring);  
cout << my_cstring;
```

Считать строку можно следующим образом:

```
scanf("%s",my_cstring);           // до пробел. символа  
scanf("%10c",my_cstring);         // ровно 10 символов  
cin >> my_cstring;                 // до пробел. символа  
cin.getline(my_cstring,10);       // до 10 символов
```

# C-строки

Для C-строк **не определены** операции присваивания и конкатенации, для C-строк **нет операций сравнения**, в отличие от других языков программирования. Для этих операций созданы специальные функции в библиотеке **string.h**.

**strlen(строка)** – возвращает целое число, равное длине строки без учета '**\0**';

**strcpy(строка1, строка2)** – копирует значение **строка2** в переменную **строка1**;

**strcat(строка1, строка2)** – присоединяет значение **строка2** к окончанию значения переменной **строка1**;

**strcmp(строка1, строка2)** –  
возвращает 0 - если **строка1** и **строка2** совпадают,  
возвращает *отрицательное* число - если **строка1**  
лексикографически меньше, чем **строка2**,

возвращает *положительное* число - если **строка1**  
лексикографически больше **строка2**,.



# C-строки

Для C-строк существуют функции преобразования строк в числа, которые находятся в библиотеке **stdlib**:

**atoi (строка\_цифр)** – возвращает значение типа **int**;

**atol (строка\_цифр)** – возвращает значение типа **long**;

**atof (строка\_цифр\_с\_разделителем\_точка)** – возвращает значение типа **double**.

Если строковый аргумент не допускает такого преобразования, функция возвращает нулевое значение.

Работа с C-строками достаточно сложная, кроме того, для этих строк работа с существующими функциями не всегда корректна, а обнаружить эту опасность достаточно сложно.

Рассмотрим другой вид строк – строки класса **string**.

# Строки класса **string**

Описание строковых переменных:

```
string s;  
string s1="world";  
string s2("Hello");
```

Переменная **s** инициализируется пустой строкой, переменная **s1** инициализируется **"world"**, переменная **s2** - **"Hello"**

Для работы со строками класса **string** требуется подключать библиотеку **string**. Для класса **string** определены операции

- присваивания (=)
- копирования
- операция конкатенации (+):

Пример:

```
s=s1+" "+s2;  
cout<<s;
```

# Строки класса `string`: ВВОД-ВЫВОД

Ввод/вывод осуществляется с помощью операторов `<<` и `>>`.

Оператор `>>` точно так же игнорирует пробелы. Если в строке встречаются пробелы, то перегруженный оператор `>>` считывает не всю строку, а только ту ее часть, которая идет до пробела (игнорируя сам пробел).

Чтобы прочитать всю строку используется функция `getline`:

```
getline(cin, s); getline(cin, s, '\t');
```

Первый аргумент – объект типа `istream`, второй – объект типа `string`, а третий (если он есть) — символ, появление которого во входном потоке приводит к прекращению вывода. По умолчанию таким признаком завершения является символ `'\n'`.

```
string s;  
getline(cin, s);  
cout << s << endl;
```

```
string s;  
getline(cin, s, '\t');  
cout << s << endl;
```

```
string s;  
getline(cin, s, ' ');  
cout << s << endl;
```

# Строки класса `string`

Для обращения к одному символу строки используются (как и в массивах) квадратные скобки.

Выражение `s[i]` можно использовать для извлечения символов точно так же, как и символы из обычного символьного массива.

Важно помнить о том, что в перегрузку квадратных скобок для объектов класса `string` не входит проверка выхода значений индексов за дозволенные пределы. *Это означает, что правильность использования значений индексов не проверяется.*

Проверку такого типа выполняет функция-член под названием `at`, которая в основном работает точно так же, как и квадратные скобки, но все же имеет два отличия. Для нее используются обозначения, принятые для функций, поэтому вместо `s[i]` нужно использовать `s.at(i)`.

Второе отличие состоит в том, что функция-член `at` проверяет, является ли значение аргумента `i` правильным индексом.

# Строки класса `string`

В двух приведенных ниже примерах фрагментов кода предпринимается попытка выйти за пределы строки. В первом из них сообщение об ошибке не выведется, несмотря на то что в нем происходит обращение к переменной с несуществующим индексом:

```
string s("Mary"); cout << s[6] << endl;
```

Однако во втором фрагменте произойдет аварийный останов программы, из чего можно сделать вывод, что в программе имеется ошибка.

```
string s("Mary"); cout << s.at(6) << endl;
```

Отдельно взятый символ строки можно изменить, присвоив переменной с индексом новое значение. То же можно сделать и с помощью метода `at(pos)`. Например,

```
s.at(2) = 'X';          s[2] = 'X';
```

# Часто используемые методы класса string

Метод `s[i]`

Пояснение: обращение к символу с индексом `i` или операция индексации

Пример:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s("Mary");
    cout << s[1] << endl;
    return 0;
}
```

Результат: **a**

# Часто используемые методы класса `string`

Метод `s.c_str()`

Пояснение: предоставляет доступ только для чтения к С-строке, представленной объектом `string`

Если нужно использовать значение из `string` с функциями для обычных С-строк, то используйте эту функцию.

Пример:

```
string s="Hello";  
char str[80];  
strcpy(str, s.c_str());  
printf("%s", str);
```

Результат: `Hello`

# Часто используемые методы класса `string`

Метод `s.substr(pos, len);`

Пояснение: возвращает подстроку, начинающуюся с позиции **pos** и имеющую длину **len**

Пример:

```
string s="hello world";  
cout<<s.substr(3, 2);
```

Результат: `lo`



# Часто используемые методы класса `string`

Метод `s.at(i)` ;

Пояснение: обращается к символу строки `s`, который имеет индекс `i` (с проверкой границ)

Пример:

```
string s="hello world";  
cout<<s.at(1) ;
```

Результат: `e`

# Часто используемые методы класса `string`

Метод `s1=s2;`

Пояснение: выделяет для строки `s1` объем памяти, равный длине строки `s2`, и инициализирует строку `s1` значением строки `s2`

Пример:

```
string s1="hello world", s2=s1;  
cout<<s2;
```

Результат: `hello world`

# Часто используемые методы класса `string`

Методы `s += s2;`

Пояснение: символы строки `s2` добавляются в конец строки `s`, для которой выделяется необходимый объем памяти

Пример:

```
string s="hello", s2=" world";  
s += s2;  
cout << s;
```

Результат: `hello world`

# Часто используемые методы класса `string`

Метод `s.empty()`

Пояснение: если строка `s` является пустой, возвращает истину, если `s` не пустая - ложь

Пример:

```
string s="hello", s1;  
cout << s.empty() << " " << s1.empty();
```

Результат: 0 1

# Часто используемые методы класса string

Метод `s.clear()`

Пояснение: удаляет все символы в **s**

Пример:

```
string s="hello";  
s.clear();  
cout<<s;
```

Результат:

# Часто используемые методы класса `string`

Функция `s.insert(pos, s2)`

Пояснение: помещает строку `s2` в строку `s`, начиная с позиции `pos`

Пример:

```
string s="hello",s2=" world";  
    s.insert(2, s2);  
    cout << s;
```

Результат: `he worldllo`

# Часто используемые методы класса `string`

Функция `stoi(s)`

Пояснение: возвращает значение типа `int` (`s` – строка цифр)

Пример:

```
string s="123";  
int x=stoi(s);  
cout << x;
```

Результат: 123

# Часто используемые методы класса `string`

Метод `stod(s)`

Пояснение: возвращает значение типа **double** (**s** – строка цифр и разделитель точка)

Пример:

```
string s="1.234567e+3";  
double x=stod(s);  
cout << x;
```

Результат: **1234.567**



# Часто используемые методы класса `string`

Метод `to_string`

Пояснение: преобразовывает значение в значение `string`.

Пример:

```
string s;  
s = to_string(123ul) ;  
cout<<s<<endl ;  
s = to_string(123.456f) ;  
cout<<s<<endl ;  
s = to_string(0xA) ;  
cout<<s ;
```

Результат:

123

123.456001

10

# Часто используемые методы класса string

Методы `s1 == s2`                      `s1 != s2`

Пояснение: проверяет, равны строки или нет; возвращает соответствующее логическое значение

Пример:

```
string s1="world", s2="world";  
cout<<(s1==s2)<<" " <<(s1!=s2);
```

Результат: 1 0

# Часто используемые методы класса string

Методы      `s1<=s2`      `s1>=s2`

Пояснение: лексикографическое сравнение строк

Пример:

```
string s1="world1", s2="world2";  
cout<<(s1<=s2)<<" "<<(s1>=s2);
```

Результат: 1 0

# Часто используемые методы класса `string`

Метод `s.find(s1)`

Пояснение: возвращает индекс начала подстроки `s1`, входящей в строку `s`

Пример:

```
string s="Hello world", s1="world";  
cout<<s.find(s1);
```

Результат: 6

# Часто используемые методы класса `string`

Метод `s.find(s1, pos)`

Пояснение: возвращает индекс начала подстроки `s1`, входящей в строку `s`; поиск начинается с позиции `pos`

Пример:

```
string s="Hello world! Hello!";  
cout<<s.find(s1, 0)<<endl;  
cout<<s.find(s1, 5);
```

Результат:

**0**

**13**

# Часто используемые методы класса string

Методы `s.length()`

`s.size()`

Пояснение: возвращает текущее количество символов в строке

Пример:

```
string s="Hello";  
cout<<s.length()<<endl;  
cout<<s.size();
```

Результат:

5

5

# Управление потоком ввода

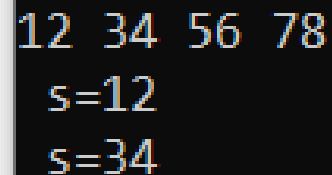
Функция

```
istream& ignore(int count, char delimiter);
```

Пояснение: функция либо считает **count** символов, либо будет считывать их до тех пор, пока ей не встретится символ-ограничитель **delimiter** (в зависимости от того, какое событие наступит раньше), а затем просто проигнорирует считанное.

Пример:

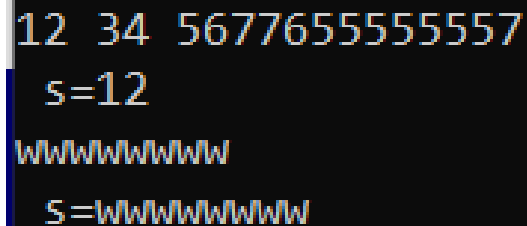
```
string s;  
cin>>s;  
cout<<" s="<<s<<endl;  
cin>>s;  
cout<<" s="<<s<<endl;
```



```
12 34 56 78  
s=12  
s=34
```

# Управление потоком ввода

```
string s;  
cin>>s;  
cout<<" s="<<s<<endl;  
cin.ignore(100, '\n');  
cin>>s;  
cout<<" s="<<s<<endl;
```



```
12 34 5677655555557  
s=12  
wwwwwwwww  
s=wwwwwwwww
```

Вызов функции с одним параметром «выбросит» указанное количество символов, а вызов без параметров отбросит 1 символ.



# P.S. Классификация и преобразование символов

isalnum — проверка на принадлежность символа к алфавитно-цифровым

isalpha — проверка на принадлежность символа к буквам

isblank — проверка пустого символа

isctrl — проверка на принадлежность символа к управляющим

isdigit — проверка на принадлежность символа к цифровым

isgraph — проверка на принадлежность символа к печатным, но не к пробелу

islower — проверка на принадлежность символа к строчным

isprint — проверка на принадлежность символа к печатным

ispunct — проверка на принадлежность символа к знакам пунктуации

isspace — проверка на принадлежность символа к пробельным

isupper — проверка на принадлежность символа к прописным

isxdigit — проверка на принадлежность символа к шестнадцатеричным

tolower — переводит символ в нижний регистр

toupper — переводит символ в верхний регистр

```
#include <stdio.h>

int main() {
    char x;
    FILE *in,*out; // описание указателей на файлы
    in = fopen("c:\\1\\old.txt","rt");
    if (in == NULL) {
        fprintf(stderr," Не могу открыть входной файл \n");
        return 1;
    }
    out = fopen("c:\\1\\new.txt","wt");
    if (out== NULL) {
        fprintf(stderr,"Не могу открыть выходной файл \n");
        return 1;
    }
    while (1) { // вечный цикл
        fscanf(in,"%c",&x);
        if (feof(in)) break; // выход из цикла
        fprintf(out,"%c",x);
    }
    fclose(in);
    fclose(out);
    return 0;
}
```