

Применение встроенного строчного типа чревато значительным количеством ошибок и не очень удобно из-за того, что он реализован на очень низком уровне. Поэтому достаточно распространена разработка собственного класса или классов для представления строчного типа. Это порождало проблемы совместимости и переносимости программ. Решением проблемы явилась реализация стандартного класса *string* стандартной библиотекой C++.

Минимальный набор операций, которыми должен обладать класс *string*:

- инициализация массивом символов (строкой встроенного типа) или другим объектом типа *string*. Встроенный тип не обладает второй возможностью;
- копирование одной строки в другую. Для встроенного типа приходится использоваться функция **strcpy()**;
- доступ к отдельным символам строки для чтения и записи. Во встроенном массиве для этого применяется операция взятия индекса или косвенная адресация;
- сравнение двух строк на равенство. Для встроенного типа используется функция **strcmp()**;
- конкатенация двух строк, получая результат либо как третью строку, либо вместо одной из исходных. Для встроенного типа применяется функция **strcat()**, однако чтобы получить результат в новой строке, необходимо последовательно задействовать функции **strcpy()** и **strcat()**;
- вычисление длины строки. Узнать длину строки встроенного типа можно с помощью функции **strlen()**;
- возможность узнать, пуста ли строка.

Класс *string* стандартной библиотеки C++ реализует все перечисленные операции (и гораздо больше, как мы увидим в [главе 6](#)).

Для того чтобы использовать объекты класса *string*, необходимо включить соответствующий заголовочный файл:

```
#include <string>
```

ОБЪЯВЛЕНИЕ И ИНИЦИАЛИЗАЦИЯ СТРОК

Первая форма определения строки, представлена объектом типа *string* и инициализированной строкой символов:

```
string st ( "Цена пакета молока\n" );
```

Вторая форма определения строки задает пустую строку:

```
string st2; // пустая строка
```

Третья форма

```
string st ="Цена пакета молока\n" ;
```

Четвертая форма конструктора позволяет установить длину строки и инициализировать ее копиями одного символа

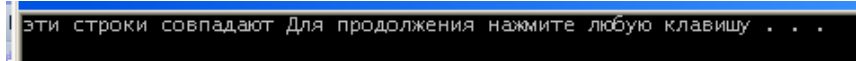
```
string st(10, '\n');
```

Пятая форма конструктора инициализирует объект типа *string* другим объектом того же типа:

```
string st3( st );//Строка st3 инициализируется строкой st.
```

Чтобы убедиться, что эти строки совпадают, воспользуемся оператором сравнения (==):

```
if ( st == st3 ) cout << "эти строки совпадают ";
```



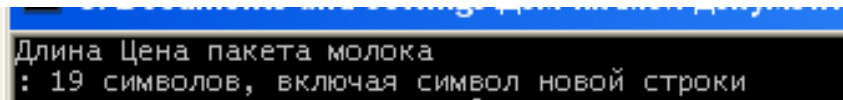
Память для переменной типа *string* может быть также выделена динамически:

```
string* ps= new string;  
string * ps1= new string("Новая строка");  
string& ps2= *new string;//ссылка на строку
```

ОПРЕДЕЛЕНИЕ И ИЗМЕНЕНИЕ ВЕЛИЧИНЫ СТРОКИ

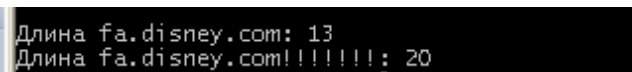
1. Длину строки возвращает функция-член **size()** (длина не включает завершающий нулевой символ).

```
cout << "Длина " << st << ": " << st.size() << " символов, включая символ новой строки\n";
```



2. Функция-член **length()** – это синоним для **size()**;
3. Функция-член **resize()** изменяет величину строки либо отсекая символы в конце, либо вставляя новые. Необязательный второй аргумент может использоваться для указания символа, который будет вставляться в новые позиции.

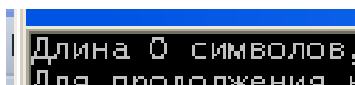
```
string str( "fa.disney.com" );  
cout << "Длина " << str << ": " << str.size() << "\n";  
str.resize (20, '!');  
cout << "Длина " << str << ": " << str.size();
```



4. Чтобы узнать, пуста ли строка, можно

- 4.1. сравнить ее длину с 0:

```
string st2("");  
if ( ! st2.size() )  
cout << "Длина " << st2.size() << " символов \n";
```



- 4.2. использовать специальный метод **empty()**, возвращающий **true** для пустой строки и **false** для непустой:

```
if ( st2.empty() ) cout << "Строка пустая ";
```

КОПИРОВАНИЕ СТРОК И ПОДСТРОКИ

Скопировать одну строку в другую можно с помощью

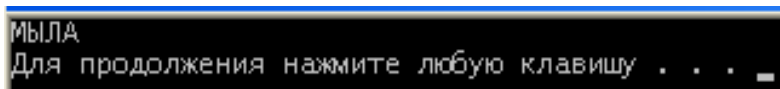
- обычной операции присваивания:

```
string st2 = st3; // копируем st3 в st2
```

- функции-члена **copy()**

s.copy(st2, n, pos) - копирует из строки *s* в символьный массив *st2* *n* символов, начиная с *pos*. В символьный массив *st2* завершающий символ ноль не дописывается. Можно использовать функцию без параметра **pos** (копируются первые *n* символов строки *s*).

```
string s("МАМА МЫЛА "); char st2[100];  
s.copy(st2, 4, 5); // копируем в st2 четыре символа строки s, начиная с 6.  
st2[4]='\0';  
cout << st2 << " \n";
```



- функции-член **substr()**

str.substr(pos, n) возвращает строку, являющуюся подстрокой исходной строки, начиная с позиции **pos** и включая **n** символов, или до конца строки (если **n** не указано).

```
string s("МАМА МЫЛА "), st2;  
st2=s.substr(2, 5); // копируем в st2 пять символов строки s, начиная с 3  
cout << st2 << " \n";
```



Если функция вызывается без параметров **st2=s.substr()**, то в **st2** скопируется вся строка *s*.

ПРИСВАИВАНИЕ, ДОБАВЛЕНИЕ И ОБМЕН СТРОК

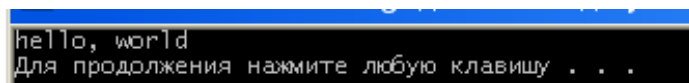
1. Для конкатенации строк используется операция сложения (+) или операция сложения с присваиванием (+=). Этот оператор создает копию левого операнда, а затем добавляет к ней содержимое правого операнда.

Пусть даны две строки:

```
string s1( "hello, " );  
string s2( "world\n" );
```

Можем получить третью строку, состоящую из конкатенации первых двух, таким образом:

```
string s3 = s1 + s2;  
cout << s3;
```

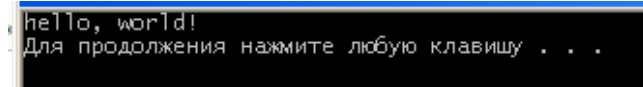


Если необходимо добавить **s2** в конец **s1**, должны написать:

```
s1 += s2;
```

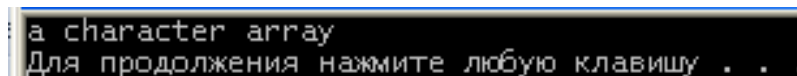
Операция сложения может конкатенировать объекты класса `string` не только между собой, но и со строками встроенного типа. Можно переписать пример, приведенный выше, так, чтобы специальные символы и знаки препинания представлялись встроенным типом, а значимые слова – объектами класса *string*:

```
const char *pc = ",";
const char *pc1 = "!";
string s1( "hello" );
string s2( "world" );
string s3 = s1 + pc + s2 + pc1 + "\n";
cout << s3;
```



Подобные выражения работают потому, что компилятор знает, как автоматически преобразовывать объекты встроенного типа в объекты класса *string*. Возможно и простое присваивание встроенной строки объекту *string*:

```
string s1;
const char *pc = "a character array\n";
s1 = pc; // правильно
cout << s1;
```



Обратное преобразование не работает. Попытка выполнить следующую инициализацию строки встроенного типа вызовет ошибку компиляции:

```
char *str = s1; // ошибка компиляции
```

Чтобы осуществить такое преобразование, необходимо явно вызвать функцию-член `c_str()`:

```
const char *str = s1.c_str(); // правильно
```

Функция `c_str()` возвращает указатель на символьный массив, содержащий строку объекта `string` в том виде, в каком она находилась бы во встроенном строковом типе. Т.о., чтобы занести информацию из объекта *string* во встроенную строку (массив) необходимо явно вызвать функцию-член `c_str()`:

```
char d[256];
strcpy(d, s1.c_str())
```

2. Функции-члены более общего назначения `assign()` и `append()`

`s.append(str, pos,n)` - добавляет к строке *s* *n*-символов строки *str*, начиная с позиции *pos*.

`s.assign (str, pos,n)` - присваивает строке *s* *n*-символов строки *str*, начиная с позиции *pos*.

Например.

```
string s("МАМА МЫЛА "), str("МАМА МЫЛА РАМУ");
s.append(str, 10,4); // добавляет к строке s 4 символа строки str, начиная с позиции 11.
cout << s<< " \n";
s.assign (str, 5,9); // присваивает строке s 9-символов строки str, начиная с позиции 6.
cout << s<< " \n";
```

```

МАМА МЫЛА РАМУ
МАМА МЫЛА РАМУ
Для продолжения нажмите любую клавишу . . .

```

Функции-члены **assign()** и **append()** возможно использовать без параметров *pos* и *n*.
s.append(str) - добавляет в конец строки строку *str*. Аналогично записи *s+=str*;

3. Поменять местами содержимое двух строк типа *string* позволяет функция **swap()**:

s.swap(str) меняет содержимое *s* и *str* местами.

СРАВНЕНИЕ СТРОК

Сравнить содержимое двух строк позволяет функция **compare()** :

s.compare(str) - сравнивает строку *s* со строкой *str* и возвращает 0 в случае совпадения, 1 - если больше, -1 – если меньше. Возможно задавать начальную позицию *pos* и число символов для сравнения *n* (**s.compare(pos, n, str)**).

ОПЕРАЦИИ ПОИСКА

Функция-член **find()** определяет первое появление строки, массива символов или символа, переданной ей в качестве первого аргумента, в текущей строке.

s.find(str, pos) - ищет строку *str* в строке *s*, начиная с заданной позиции *pos* (необязательный параметр). Если первое вхождение строки найдено, функция возвращает позицию первого символа в текущей строке, с которой начинается совпадение.

Функция-член **rfind()** аналогична, но выполняет сканирование исходной строки справа налево

```

string s1("mississippi");
cout << s1.find("ss") << " \n";
cout << s1.find("ss", 4) << " \n";
cout << s1.rfind("si") << " \n";

```

```

2
5
6
Для продолжения нажмите любую клавишу . . .

```

Функции **find_first_of()**, **find_last_of()**, **find_first_not_of()**, **find_last_not_of()** выполняют аналогичные операции поиска. Первый их аргумент задает строку, как массив символов или символ для поиска. Другие (необязательные) параметры задают позицию и число символов исходной строки, участвующие в операции. Эти функции находят первый или последний символ, который присутствует или, наоборот, отсутствует в строке.

```

string s1("mississippi");
int i=s1.find_first_of("aeiou");//найти первую гласную
cout << i << " \n";
int j=s1.find_first_not_of("aeiou",4);//найти первую согласную, начиная с пятого
//символа
cout << j << " \n";

```

```

1
5
Для продолжения нажмите любую клавишу . . .

```

ВСТАВКА СИМВОЛОВ В СТРОКУ

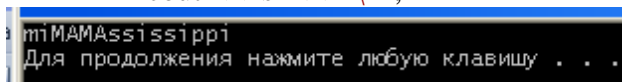
1. Функция-член **insert()** дает возможность вставлять другую строку, массив символов или символ в исходную строку

s.insert (pos, str, beg, count) - вставляет в строку **s**, начиная с заданной позиции **pos** **count** символов строки **str**, начиная с позиции **beg**.

Возможные варианты использования функции:

- **s.insert (pos, str)** - вставляет в строку **s**, начиная с заданной позиции **pos** строку **str**

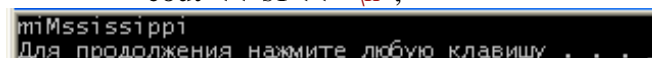
```
string s1("mississippi");
s1.insert (2, "MAMA");
cout << s1<< " \n";
```



```
miMAMAssissippi
Для продолжения нажмите любую клавишу . . .
```

- **s.insert (pos, str, count)** - вставляет в строку **s**, начиная с заданной позиции **pos**, **count** символов строки **str**.

```
string s1("mississippi");
s1.insert (2, "MAMA",1);
cout << s1<< " \n";
```



```
miMssissippi
Для продолжения нажмите любую клавишу . . .
```

- **s.insert (pos, const char* str, count)** - вставляет в строку **s**, начиная с заданной позиции **pos**, **count** символов массива, на который указывает параметр **str**.

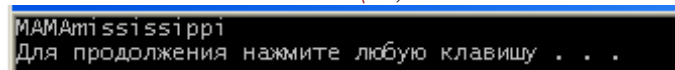
```
string s1("mississippi");
const char* str="MAMA";
s1.insert (1, str, 2);
cout << s1<< " \n";
```



```
mMAssissippi
Для продолжения нажмите любую клавишу . . .
```

- **s.insert (pos, const char* str)** - вставляет в строку **s**, начиная с заданной позиции **pos** символы массива, на который указывает параметр **str**.

```
string s1("mississippi");
const char* str="MAMA";
s1.insert (0, str);
cout << s1<< " \n";
```



```
MAMAmissippi
Для продолжения нажмите любую клавишу . . .
```

- **s.insert (pos, count, char c)** - вставляет в строку **s**, начиная с заданной позиции **pos**, **count** копий символа **c**.

```
string s1("mississippi");
s1.insert (3, 5, '*');
```

```
cout << s1 << " \n";
1 mis****sippi
Для продолжения нажмите любую клавишу . . .
```

2. Функция-член **s.push_back(symbol)** - добавляет в конец строки символ

```
string s1("mississippi");
s1.push_back('*'); //добавляет в конец строки символ
cout << s1 << " \n";
```

```
mississippi*
Для продолжения нажмите любую клавишу . . . _
```

ЗАМЕНА И УДАЛЕНИЕ СИМВОЛОВ ИЗ СТРОКИ

ЗАМЕНА. Функция-член **replace()** заменяет символы в строке содержимым другой строки, массива символов или просто символом.

Возможные варианты использования функции:

- **s.replace(pos, n, str)** - заменяет **n** символов строки **s**, начиная с позиции **pos**, строкой **str**

```
string s1("mississippi"),str="****";
s1.replace(4, 2,str);
cout << s1 << " \n";
```

```
mis****sippi
Для продолжения нажмите любую клавишу . . .
```

- **s.replace(pos1, n1, str, pos2, n2)** - заменяет **n1** символов строки **s**, начиная с позиции **pos1**, **n2** символами строки **str**, начиная с позиции **pos2**.

```
string s1("mississippi"), str="*@#%";
s1.replace(4, 2,str,1,2);
cout << s1 << " \n";
```

```
miss@#sippi
Для продолжения нажмите любую клавишу . . .
```

- **s.replace(pos1, n1, const char* str, n2)** - заменяет **n1** символов строки **s**, начиная с позиции **pos1**, **n2** символами массива, на который указывает параметр **str**.

```
string s1("mississippi");
const char *str = "*@#%";
s1.replace(4, 2,str,3);
cout << s1 << " \n";
```

```
mis*#@sippi
Для продолжения нажмите любую клавишу .
```

- **s.replace(pos1, n1, const char* str)** - заменяет **n1** символ строки **s**, начиная с позиции **pos1**, символами массива, на который указывает параметр **str**.

```
string s1("mississippi");
const char *str = "*@#%";
s1.replace(4, 2,str);
cout << s1 << " \n";
```

```
miss*@#%sippi
Для продолжения нажмите любую клавишу . . .
```

- **s.replace(pos1, n1, n2, char c)** - заменяет **n1** символ строки **s**, начиная с позиции **pos1**, символами **c** **n2** раз.

```
string s1("mississippi");
s1.replace(4,2,5,'&');
cout << s1<< " \n";
```

```
miss&&&&&sippi
Для продолжения нажмите любую клавишу . . .
```

УДАЛЕНИЕ.

1. Функция-член **erase()** удаляет символы.

Возможные варианты использования функции:

- **s.erase(pos, n)** удаляет **n** элементов строки **s**, начиная с позиции **pos**

```
string s1("mississippi");
s1.erase(2, 4 );
cout << s1<< " \n";
```

```
missippi
Для продолжения нажмите любую клавишу . . .
```

- **s.erase(pos)** удаляет все элементы строки **s**, начиная с позиции **pos**

```
string s1("mississippi");
s1.erase(2);
cout << s1<< " \n";
```

```
mi
Для продолжения нажмите любую клавишу . . .
```

- **s.erase()** удаляет все элементы строки **s**.

2. Функция-член **clear()** - очищает строку.

s.clear() удаляет все элементы строки **s**.

К отдельным символам объекта типа *string*, как и встроенного типа, можно обращаться с помощью операции взятия индекса. Вот, например, фрагмент кода, заменяющего все точки символами подчеркивания:

```
string str( "fa.disney.com" );
int size = str.size();
```

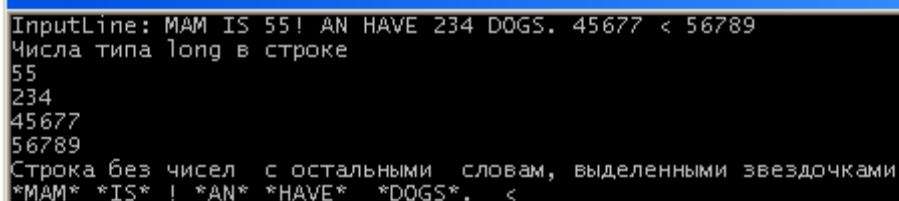
```
for ( int ix = 0; ix < size; ++ix )
    if ( str[ ix ] == '.' )
        str[ ix ] = '_';
cout << str<< " \n";
```

```
fa_disney_com
Для продолжения нажмите любую клавишу . . .
```

ПРИМЕР.

Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими) и символами пунктуации. Вывести те слова строки, которые являются числами типа long, и удалить их из исходной строки. Все остальные слова строки выделить символом '*' с обеих сторон.

```
#include <iostream>
#include <string>
#include <locale>
#include <conio.h>
using namespace std;
int main()
{
    setlocale (LC_CTYPE,"rus");
    string str,s;
    cout<<"InputLine: ";
    getline(cin,str, '*'); // ввод текста
    str+=" ";
    int len = str.size() ; //длина введенного текста
    int i = 0;
    int begin = 0, end = 0;
    cout<<"Числа типа long в строке\n";
    while (i < str.size()-1)
    {
        while ((isspace(str[i])||ispunct(str[i]))&& i < str.size()-1)
            ++i; //пропустить пробелы и любые разделители
        begin = i; // номер первого символа слова
        while ( !isspace(str[i])&&!ispunct(str[i])&& i < str.size()-1)
            ++i; // пропустить все символы слова
        end = i; // номер символа, следующего за последним символом слова
        s=str.substr(begin, end-begin) ;// копируем в s (end-begin)символов строки str,
//начиная с begin
        if (atol(s.c_str())) //если преобразование строки s в целое число типа long
//прошло успешно
        {
            cout<<atol(s.c_str())<<"\n"; //вывод числа
            str.erase(begin, end-begin); i=i-(end-begin); //ПРИ УДАЛЕНИИ
//ДОЛЖНЫ УМЕНЬШИТЬ i НА КОЛИЧЕСТВО УДАЛЕННЫХ СИМВОЛОВ!
        }
        else {str.insert (begin, 1, '*'); i++;str.insert (end+1, 1, '*'); i++;} //ПРИ ВСТАВКЕ
//ДОЛЖНЫ УВЕЛИЧИТЬ i НА КОЛИЧЕСТВО ВСТАВЛЕННЫХ СИМВОЛОВ!
    }
    cout<<"Строка без чисел с остальными словам, выделенными звездочками\n";
    cout<<str<<"\n";
    return 0;
}
```



```
InputLine: MAM IS 55! AN HAVE 234 DOGS. 45677 < 56789
Числа типа long в строке
55
234
45677
56789
Строка без чисел с остальными словам, выделенными звездочками
*MAM* *IS* ! *AN* *HAVE* *DOGS*. <
```