STRING

string – класс с методами и переменными для организации работы со строками. Он включен в стандартную библиотеку C++ и является частью стандартного пространства имён – std. Класс string предоставляет более удобный интерфейс доступа, который избавляет от необходимости следить за размером символьного массива. Объект класса string может быть преобразован в С-строку.

Для использования класса string необходимо подключить файл string:

```
#include <string>
```

Объявление и инициализация строки

Объявить объект класса string можно следующими способами:

- объявить экземпляра класса без инициализации:

```
std::string str;
```

- указать С-строку внутри круглых скобок или после оператора =, тем самым произведя инициализацию объекта:

```
std::string str1("Строка1");
std::string str2 = "Строка2";
```

Если во втором параметре передать число, то можно присвоить только часть строки, а не всю строку:

```
std::string str1("Строка1", 3); // Стр
```

- указать объект класса string внутри круглых скобок или после оператора =, тем самым произведя инициализацию объекта:

```
std::string str1("Строка");
std::string str2(str1);
std::string str3 = str1;
```

Можно присвоить не все содержимое объекта, а его часть. Для этого во втором параметре следует передать начальный индекс, а в третьем параметре конечный индекс. Если конечный индекс не указан, то копируется фрагмент до конца строки.

```
std::string str1("Строка");
std::string str2(str1, 3); // ка
std::string str3(str1, 0, 3); // Стро
```

- указать в первом параметре количество символов, а во втором параметре символ-заполнитель:

```
std::string str(5, '*'); // *****
```

Присвоить значение переменной после объявления можно с помощью оператора =. Следить за размером строки не нужно. Если производится присваивание объекта класса string, то создается копия исходного объекта, а не присваивается ссылка на него:

```
std::string str1, str2;
str1 = "Строка";
str2 = str1;
```

Ввод и вывод строк

Объекты класса string можно вводить и выводить с помощью операторов >> и << соответственно. **!!!** С помощью оператора >> можно ввести только фрагмент до первого пробельного символа.

```
std::string str;
cin >> str;
cout << str;
```

Для ввода строки, содержащей пробелы, используется функция getline(). В первом параметре указывается объект std::cin, во втором параметре – объект класса string, а в третьем параметре – символ, до которого производится считывание. Если третий параметр не указан, то считывание производится до символа перевода строки:

```
std::string str;
```

```
std::getline(cin, str, '\n');
cout << str;

std::string str;

str.reserve(100);
for (int i = 0; i < 20; ++i)
{
	str.push_back('A' + i);
	// или
	str += 'A' + rand()%26;
}
std::cout << str << '\n';

std::cout << str.size() << ' ' << str.max_size() << ' ' << str.capacity() << '\n';
               20                  4294967294                  111
std::cout << std::string::npos << ' ' << ULONG_MAX << '\n';
                  4294967295          4294967295

char sstr[] = "0123456789";
str = sstr;

std::cout << sizeof(sstr) << '\n' << sizeof(str) << '\n';
                11                  28
```

Преобразование объекта string в C-строку или массив символов

c_str() – возвращает указатель типа const char*. Его можно использовать в тех случаях, когда необходимо передать C-строку (например, в функцию), вместо объекта класса string. Изменять такую строку нельзя. **!!!** После изменения объекта string указатель может стать некорректным, поэтому после изменения объекта необходимо обновить значение указателя.

Прототип метода:

```
const char *c_str() const;
```

Пример:

```
std::string str = "ABCDE";

const char *sstr = str.c_str();
str += '!';

std::cout << str << '\n' << sstr << '\n';
```

data() – возвращает указатель типа const char* на массив символов (нулевой символ не добавляется). **!!!** После изменения объекта string указатель может стать некорректным, поэтому после изменения объекта необходимо обновить значение указателя.

Прототип метода:

```
const char *data() const;
```

Пример:

```
std::string str = "ABCDE";

const char *sstr = 0;
sstr = str.data();
str += '!';

std::cout << str << '\n' << sstr << '\n';
```

copy() – копирует Count символов, начиная с индекса Off, в символьный массив Ptr. Если индекс не указан, то символы копируются с начала строки. Нулевой символ автоматически не добавляется. В качестве значения метод возвращает количество скопированных символов.

Прототип метода:

```
size_t copy (char *Ptr, size_t Count, size_t Off=0) const;
```

Пример:

```
std::string str="Строка";

char str2[80]={0}, str3[80]={0};

size_t count=str.copy(str2,5);
count=str.copy(str3,3,1);

cout << count << endl; // 5
cout << str2 << endl; // Строк
cout << str3 << endl; // тро
```

В VC++ вместо метода copy() следует использовать метод **_Copy_s()**.

Прототип метода:

```
size_t _Copy_s (char *Dest, size_t Dest_size, size_t Count,
                  size_t Off=0);
```

Пример:

```
size_t count=str._Copy_s(str2, sizeof str2, 5);
```

Получение и изменение размера строки

size() и **length()** – возвращают текущее количество символов в строке.

Прототипы методов:

```
size_t size() const;
size_t length() const;
```

Пример:

```
std::string str="Строка";

cout << str.size() << endl; // 6
cout << str.length() << endl; // 6
```

resize() – задает количество символов в строке, равное числу Newsize. Если указанное количество символов меньше текущего количества, то лишние символы будут удалены. Если количество символов необходимо увеличить, то в параметре Ch можно указать символ, который заполнит новое пространство.

Прототипы методов:

```
void resize(size_t Newsize);
void resize(size_t Newsize, char Ch);
```

Пример:

```
std::string str="Строка";

str.resize(3);
cout << str << endl; // Стр

str.resize(8,'*');
cout << str << endl; // Стр*****
```

clear() – удаляет все символы.

Прототип метода:

```
void clear();
```

Пример:

```
std::string str="Строка";

str.clear();
cout << str.size() << endl; // 0
```

empty() – возвращает значение true, если строка не содержит символов, и false – в противном случае.

Прототип метода:

```
bool empty() const;
```

Пример:

```
std::string str="Строка";
if (str.empty())
	cout << "Строка пустая\n";
else
	cout << "Строка содержит символы\n";
```

Получение и изменение содержимого строки

К любому символу объекта класса string можно обратиться как к элементу массива. Достаточно указать его индекс в квадратных скобках. Нумерация начинается с нуля. Можно как получить символ, так и изменить его. Если индекс выходит за границы диапазона, то возвращаемое значение не определено.

Пример. Перебор символов строки.

```
std::string str = "ABCDE";
size_t len = str.length();
for (size_t i = 0; i < len; ++i)
	std::cout << str[i];
std::cout << '\n';

for (char c : str)
	std::cout << c;
std::cout << '\n';
```

С помощью итераторов (*итераторы* – специальные объекты для доступа к элементам некоторого множества, в нашем случае – строки. С помощью итератора можно перемещаться внутри множества и получать доступ к отдельным элементам):

```
for (std::string::iterator it = str.begin(); it != str.end(); ++it)
	std::cout << *it;
std::cout << '\n';
```

Для объектов класса string определена операция *конкатенации* (объединения строк). Оператор «+» позволяет объединить:

- два объекта класса string:

```
std::string str1="ABC", str2="DE", str3;
str3 = str1 + str2; // ABCDE
```

- объект класса string и символ:

```
std::string str1="ABC", str3;
str3 = str1 + 'D'; // ABCD
```

- объект класса string и С-строку:

```
std::string str1="ABC", str2, str3;
str2 = str1 + "DE"; // ABCDE
str3 = "DE" + str1; // DEABC
```

!!! Оператор «+» для объединения двух С-строк применять нельзя. Чтобы объединить две С-строки их указывают без операторов между ними.

```
std::string str1("ABC"), str2, str3;
str2 = "ABC" + "DE"; // ОШИБКА!!!
str2 = "ABC" + str3 + "DE"; // ABCDE
str2 = string("ABC") + "DE"; // ABCDE
str3 = "ABC" "DE"; // ABCDE
```

Для получения и изменения содержимого строки предназначены следующие методы.

at() – возвращает ссылку на символ, расположенный по индексу Off. Метод позволяет, как получить символ, так и изменить его. Если индекс выходит за границы диапазона, то метод генерирует исключение.

Прототипы метода:

```
reference at(size_t Off);
const_reference at(size_t off) const;
```

Пример:

```
std::string str("Строка");

std::string::reference ref = str.at(0);
ref = 'S';

cout << str << endl; // Strока

cout << str.at(3) << endl; // o
cout << str[3] << endl; // o

str.at(3)='-';
cout << str << endl; // Str-ка
```

front() – возвращает ссылку на первый символ в строке. Метод позволяет, как получить символ, так и изменить его.

Прототипы метода:

```
reference front();
const_reference front() const;
```

Пример:

```
std::string str("Строка");

cout << str.front() << endl; // C

str.front()='*';

cout << str << endl; // *трока
```

back() – возвращает ссылку на последний символ в строке. Метод позволяет, как получить символ, так и изменить его.

Прототипы метода:

```
reference back();
const_reference back() const;
```

Пример:

```
std::string str("Строка");

cout << str.back() << endl; // a

str.back()='*';
cout << str << endl; // Строк*
```

substr() – возвращает фрагмент строки, состоящий из Count символов, начиная с индекса Off. Если начальный индекс не задан, то предполагается, что индекс равен 0. Если длина не задана, то возвращается фрагмент, начиная с индекса Off до конца строки.

Прототип метода:

```
string substr(size_t Off=0, size_t Count=npos) const;
```

std::string::npos – это специальное значение, равное максимальному значению, которое может предоставить тип size_t. Как правило, это значение используется либо как индикатор конца строки в функциях, которые ожидают позицию символа, либо как индикатор ошибки в функциях, которые возвращают позицию в строке.

Пример:

```
std::string str("ABCDE"), str2;

str2 = str.substr();
cout << str2 << endl; // ABCDE

str2 = str.substr(2);
cout << str2 << endl; // CDE

str2 = str.substr(2, 2);
cout << str2 << endl; // CD
```

push_back() – добавляет символ Ch в конец строки.

Прототип метода:

```
void push_back(char Ch);
```

Пример:

```
std::string str("Строка");

str.push_back('*');
cout << str << endl; // Строка*
```

append() – добавляет символы и строки в конец исходной строки. Метод можно использовать для конкатенации вместо операторов «+» и «+=».

Прототипы метода:

```
// добавляет Count символов Ch в конец строки
string &append(size_t Count, char Ch);
// добавляет С-строку Ptr целиком
string &append(const char *Ptr);
// добавляет Count первых символов С-строки Ptr
string &append(const char *Ptr, size_t Count);
// добавляет класса sting Right целиком
string &append(const string &Right);
// добавляет Count первых символов строки Right, начиная с индекса Roff
string &append(const string &Right, size_t Roff, size_t Count);
```

Пример:

```
std::string str("A");

str.append(2,'*');   // Прототип 1
cout << str << endl; // A**

char str2[]="DEF";
str.append(str2);    // Прототип 2
cout << str << endl; // A**DEF

str.append(str2, 2); // Прототип 3
```

```
cout << str << endl; // A**DEFDE

str="A";
std::string str3="BC";

str.append(str3);    // Прототип 4
cout << str << endl; // ABC

str.append(str3,0 ,1); // Прототип 5
cout << str << endl;   // ABCB
```

insert() – вставляет символы и строки в позицию, указанную индексом или итератором. Остальные символы сдвигаются к концу строки.

Прототипы метода:

```
// Прототип 1
// вставляет Count символов Ch в позицию с индексом Off
string &insert(size_t Off, size_t Count, char Ch);
// Прототип 2
// вставляет символ Ch в позицию, на которую указывает константный итератор
// Where
iterator insert(const_iterator Where, char Ch);
// Прототип 3
// вставляет Count символов Ch в позицию, на которую указывает константный
// итератор Where
void insert(const_iterator Where, size_t Count, char Ch);
// Прототип 4
// вставляет С-строку Ptr целиком в позицию с индексом Off
string &insert(size_t Off, const char *Ptr);
// Прототип 5
// вставляет Count первых символов С-строки Ptr в позицию с индексом Off
string &insert(size_t Off, const char *Ptr, size_t Count);
// Прототип 6
// вставляет объект класса string Right целиком в позицию с индексом Off
string &insert(size_t Off, const string &Right);
// Прототип 7
// вставляет Count первых символов объекта класса string Right, начиная
// с позиции Roff в позицию с индексом Off
string &insert(size_t Off, const string &Right, size_t Roff,
               size_t Count);
```

Пример:

```
std::string str("ABCDE");
str.insert(2, 5, '*'); // Прототип 1
cout << str << endl;   // AB*****CDE

str="ABCDE";
str.insert(str.begin()+2, '*'); // Прототип 2
cout << str << endl;            // AB*CDE

str="ABCDE";
str.insert(str.begin()+2, 5, '*'); // Прототип 3
cout << str << endl;               // AB*****CDE

str="ABCDE";
char str2[]="+++";
str.insert(2, str2); // Прототип 4
cout << str << endl; // AB+++CDE

str="ABCDE";
```

```
str.insert(2, str2, 1); // Прототип 5
cout << str << endl;    // AB+CDE

str="ABCDE";
std::string str3("###");
str.insert(2, str3);  // Прототип 6
cout << str << endl;  // AB###CDE

str="ABCDE";
str.insert(2, str3, 0, 1);  // Прототип 7
cout << str << endl;        // AB#CDE
```

pop_back() – удаляет последний символ в строке.

Прототип метода:

```
void pop_back();
```

Пример:

```
std::string str("ABCDE");
str.pop_back();
cout << str << endl;   // ABCD
```

erase() – удаляет один символ или фрагмент.

Прототипы метода:

```
// удаляет символ, на который указывает итератор Where
iterator erase(const_interator Where);
// удаляет фрагмент строки между итераторами First и Last
iterator erase(const_interator First, const_iterator Last);
// удаляет фрагмент длиной Count, начинающийся с индекса Off. Если начальный
// индекс не указан, то предполагается, что индекс равен 0. Если длина
// не указана, то удаляется фрагмент от индекса Off до конца строки
string &erase(size_t Off=0, size_t Count=npos);
```

Пример:

```
std::string str("ABCDE");
str.erase(str.begin()+2);  // Прототип 1
cout << str << endl;       // ABDE

str="ABCDE";
str.erase(str.begin(), str.begin()+2);  // Прототип 2
cout << str << endl;       // CDE

str="ABCDE";
str.erase(3);        // Прототип 3
cout << str << endl; // ABC

str="ABCDE";
str.erase(0, 2);     // Прототип 3
cout << str << endl; // CDE
```

swap() – меняет содержимое двух строк местами.

Прототип метода:

```
void swap(string &Right);
```

Пример:

```
std::string str1="12345", str2="ABC";
str1.swap(str2);
cout << str1 << endl; // ABC
cout << str2 << endl; // 12345
```

replace() – заменяет фрагмент строки отдельным символом или подстрокой. Если вставляемая подстрока меньше фрагмента, то остальные символы сдвигаются к началу строки, а если больше, то раздвигаются таким образом, чтобы вместить всю вставляемую подстроку.

Прототипы метода:

```
// Прототип 1
// вставляет Count символов Ch вместо фрагмента длиной NO, начинающегося
// с индекса Off
string &replace(size_t Off, size_t NO, size_t Count, char Ch);
// Прототип 2
// вставляет Count символов Ch вместо фрагмента ограниченного элементами,
// на которые указывают итераторы First и Last
string &replace(const_iterator First, const_iterator Last, size_t Count, char Ch);
// Прототип 3
// вставляет С-строку Ptr целиком вместо фрагмента длиной NO, начинающегося
// с индекса Off
string &replace(size_t Off, size_t NO, const char *Ptr);
// Прототип 4
// вставляет Count первых символов С-строки Ptr вместо фрагмента длиной NO,
// начинающегося с индекса Off
string &replace(size_t Off, size_t NO, const char *Ptr, size_t Count);
// Прототип 5
// вставляет С-строку Ptr целиком вместо фрагмента ограниченного элементами,
// на которые указывают итераторы First и Last
string &replace(const_iterator First, const_iterator Last, const char *Ptr);
// Прототип 6
// вставляет Count первых символов С-строки Ptr вместо фрагмента ограниченного
// элементами, на которые указывают итераторы First и Last
string &replace(const_iterator First, const_iterator Last, const char *Ptr, size_t
Count);
// Прототип 7
// вставляет объект класса string Right целиком вместо фрагмента длиной NO,
// начинающегося с индекса Off
string &replace(size_t Off, size_t NO, const string &Right);
// Прототип 8
// вставляет Count первых символов объекта string Right вместо фрагмента,
// длиной NO, начинающегося с индекса Off
string &replace(size_t Off, size_t NO, const string &Right, size_t Roff, size_t
Count);
// Прототип 9
// вставляет объект класса string Right целиком вместо фрагмента ограниченного
// элементами, на которые указывают итераторы First и Last
string &replace(const_iterator First, const_iterator Last, const string &Right);
```

Примеры:

```
std::string str("ABCDE");
str.replace(0,3,4,'*'); // Прототип 1
cout << str << endl;    // ****DE

str="ABCDE";
str.replace(str.begin(),str.begin()+3,3,'+'); // Прототип 2
cout << str << endl;                          // +++DE

str="ABCDE";
char str2[] = "123";
str.replace(0,3,str2); // Прототип 3
cout << str << endl;   // 123DE

str="ABCDE";
```

```
str.replace(0,3,str2,2); // Прототип 4
cout << str << endl;     // 12DE

str="ABCDE";
std::string str3="+++++";
str.replace(str.begin(),str.begin()+3,str3); // Прототип 5
cout << str << endl;    // +++++DE

str="ABCDE";
str.replace(str.begin(),str.begin()+3,str2,2); // Прототип 6
cout << str << endl;    // 12DE

str="ABCDE";
str.replace(1,4,str3);  // Прототип 7
cout << str << endl;  // A+++++

str="ABCDE";
str.replace(2,5,str3,1,2);  // Прототип 8
cout << str << endl;        // AB++

str="ABCDE";
str.replace(1,3,str3);  // Прототип 9
cout << str << endl;        // A+++++E
```

Поиск в строке

find() – производит поиск символа или фрагмента с начала строки. Возвращает индекс первого совпадения, если фрагмент найден, или значение константы std::string::npos – в противном случае. Поиск зависит от регистра символов.

Прототипы метода:

```
// Прототип 1
// ищет в строке символ Ch, начиная с индекса Off
size_t find(char Ch, size_t off=0) const;
// Прототип 2
// ищет в строке С-строку Ptr целиком, начиная с индекса Off
size_t find(const char *Ptr, size_t off=0) const;
// Прототип 3
// ищет в строке первые Count символов С-строки Ptr, начиная с индекса Off
size_t find(const char *Ptr, size_t off, size_t Count) const;
// Прототип 4
// ищет в строке объект класса string Right, начиная с индекса Off,
// если индекс не задан, то полагается значение равное 0
size_t find(const string &Right, size_t off=0) const;
```

Примеры:

```
// Прототип 1
cout << str.find('C') << endl; // 2
cout << str.find('X') << endl; // 4294967295
if (str.find('X') == std::string::npos)
      cout << "Символ не найден\n";
cout << (int)str.find('X') << endl; // -1
cout << (int)str.find('E',3) << endl; // 4
cout << (int)str.find('E',5) << endl; // -1
// Прототип 2
cout << (int)str.find("DED") << endl; // 3
cout << (int)str.find("DED",5) << endl; // -1
// Прототип 3
cout << (int)str.find("ABCXY",0,3) << endl; // 0
cout << (int)str.find("ABCXY",0,4) << endl; // -1
```

```
// Прототип 4
std::string str2="BCD";
cout << (int)str.find(str2) << endl; // 1
cout << (int)str.find(str2,2) << endl; // -1
```

rfind() – производит поиск символа или фрагмента с конца строки. Возвращает индекс первого с конца совпадения, если фрагмент найден, или значение константы std::string::npos – в противном случае. Поиск зависит от регистра символов.

Прототипы метода:

```
// Прототип 1
// ищет в строке символ Ch, начиная с индекса Off
size_t rfind(char Ch, size_t off=npos) const;
// Прототип 2
// ищет в строке C-строку Ptr целиком, начиная с индекса Off
size_t rfind(const char *Ptr, size_t off=npos) const;
// Прототип 3
// ищет в строке первые Count символов C-строки Ptr, начиная с индекса Off
size_t rfind(const char *Ptr, size_t off, size_t Count) const;
// Прототип 4
// ищет в строке объект класса string Right, начиная с индекса Off,
// если индекс не задан, то полагается значение равное индексу последнего
// символа
size_t rfind(const string &Right, size_t off=npos) const;
```

find_first_of() – производит поиск символа или какого-либо символа, входящего в строку. Поиск производится с начала строки. Метод возвращает индекс первого совпадения, если символ найден, или значение константы std::string::npos – в противном случае. Поиск зависит от регистра символов.

Прототипы метода:

```
// Прототип 1
// ищет в строке символ Ch, начиная с индекса Off
size_t find_first_of(char Ch, size_t off=0) const;
// Прототип 2
// ищет совпадение с каким-либо символом из C-строки Ptr, начиная с индекса
// Off
size_t find_first_of(const char *Ptr, size_t off=0) const;
// Прототип 3
// учитывает только первые Count символов C-строки Ptr, начиная с индекса Off
size_t find_first_of(const char *Ptr, size_t off, size_t Count) const;
// Прототип 4
// ищет совпадение с каким-либо символом объекта класса string Right, начиная
// с индекса Off, если индекс не задан, то полагается значение равное индексу
// последнего символа
size_t find_first_of(const string &Right, size_t off=0) const;
```

find_first_not_of() – возвращает индекс первого символа, который не совпадает ни с одним из символов, входящих в указанную строку. Поиск производится, начиная с позиции Off. Если индекс Off не задан, то предполагается, что его значение равно 0. Метод возвращает индекс первого несовпадения, в противном случае – значение константы std::string::npos. Поиск зависит от регистра символов.

Прототипы метода:

```
// Прототип 1
// ищет в строке символ, который не совпадает с символом Ch
size_t find_first_not_of(char Ch, size_t off=0) const;
// Прототип 2
// ищет в строке символ, который не совпадает ни с одним из символов, входящих
// в C-строку Ptr
size_t find_first_not_of(const char *Ptr, size_t off=0) const;
```

```
// Прототип 3
// учитывает только первые Count символов С-строки Ptr
size_t find_first_not_of(const char *Ptr, size_t off, size_t Count) const;
// Прототип 4
// ищет в строке символ, который не совпадает ни с одним из символов, входящих
// в объект класса string Right
size_t find_first_not_of(const string &Right, size_t off=0) const;
```

find_last_of() – метод аналогичен методу find_first_of(), но поиск производится с конца строки, а не с начала.

find_last_not_of() – метод аналогичен методу find_first_not_of(), но поиск производится с конца строки, а не с начала.

Пример. Проверить, что заданная строка состоит только из цифр.

```
std::string str="1275378";
//int index = str.find_last_not_of("0123456789");
int index = str.find_first_not_of("0123456789");
if (index == std::string::npos)
	cout << "Строка содержит только цифры\n";
else
	cout << "Строка содержит не только цифры\n";
```

Сравнение строк

Для сравнения двух объектов класса string или объекта класса string и С-строки предназначены операторы ==, !=, < , >, <=, >=. (!!! С-строки можно сравнивать между собой только функцией strcmp()).

Пример:

```
std::string str = "ABC";
char str2[] = "ABC";
if (str == str2)
	cout << "Строки равны\n";
else
	cout << "Строки не равны\n";
```

Для сравнения объекта класса string с другим объектом класса string или С-строкой можно использовать метод compare(). Сравнение производится с учетом регистра символов. В качестве значения метод возвращает:

- отрицательное число – если объект класса string меньше строки, переданной в качестве параметра;
- ноль – если строки равны;
- положительное число – если объект класса string больше строки, переданной в качестве параметра.

Прототипы метода:

```
// Прототип 1
// сравнивает объект класса string с С-строкой
int compare(const char *Ptr) const;
// Прототип 2
// сравнивает фрагмент объекта класса string длиной NO, начиная с индекса Off,
// с С-строкой
int compare(size_t Off, size_t NO, const char *Ptr) const;
// Прототип 3
// сравнивает фрагмент объекта класса string длиной NO, начиная с индекса Off,
// с фрагментом С-строки длиной Count
int compare(size_t Off, size_t NO, const char *Ptr, size_t Count) const;
```

```
        // Прототип 4
        // сравнивает два объекта класса string
        int compare(const string &Right) const;
        // Прототип 5
        // сравнивает фрагмент объекта класса string длиной NO, начиная с индекса Off,
        // с другим объектом класса string
        int compare(size_t Off, size_t NO, const string &Right) const;
        // Прототип 6
        // сравнивает фрагмент объекта класса string длиной NO, начиная с индекса Off,
        // с другим объектом класса string. Позволяет дополнительно указать длину
        // Count и начальный индекс Roff в строке Right
        int compare(size_t Off, size_t NO, const string &Right, size_t Roff, size_t Count)
const;
```

Примеры:

```
std::string str1 = "ABC", str2 = "ABD";
// Прототип 1
cout << str1.compare("ABE") << endl; // -1
// Прототип 2
cout << str1.compare(0,2,"ABE") << endl; // -1
// Прототип 3
cout << str1.compare(0,2,"ABE",2) << endl; // 0
// Прототип 4
cout << str1.compare(str2) << endl; // -1
// Прототип 5
cout << str1.compare(0,2,str2) << endl; // -1
// Прототип 6
cout << str1.compare(0,2,str2,0,2) << endl; // 0
```

string как контейнер

```
#include <algorithm>

std::string str = "ABCDE";

std::sort(str.begin(), str.end(), [](char x, char y) {return x > y; });
std::cout << str << '\n';

std::for_each(str.begin(), str.end(), [](char &c) { c+=1; });
std::cout << str << '\n';
```

Классы istringstream, ostringstream и stringstream

Классы istringstream, ostringstream и stringstream позволяют работать со строкой как с потоком. Класс istringstream предназначен для чтения данных из строки, класс ostringstream – для записи данных в строку, класс stringstream позволяет, как читать из строки, так и записывать в строку. Прежде чем использовать эти классы необходимо подключить заголовочный файл <sstream>.

При создании экземпляра класса чаще всего используются четыре формата:

```
<Класс> <Переменная>;
<Класс> <Переменная> (<Режим>);
<Класс> <Переменная> (<С-строка>[, <Режим>]);
<Класс> <Переменная> (<Объект класса string>[, <Режим>]);
```

Название класса зависит от необходимой операции со строкой. Параметр режим задает режим открытия:

- ios::in – открытие строкового потока для чтения. Если в строке существуют символы, то содержимое не удаляется;

- ios::out – открытие строкового потока для записи. Если в строке существуют символы, то содержимое удаляется;
- ios::trunc – если в строке существуют символы, то содержимое удаляется. Режим используется по умолчанию, если установлен режим ios::out, а режимы ios::in, ios::app или ios::ate не указаны;
- ios::app – запись производится в конец строки, даже если указана другая позиция с помощью метода seek();
- ios::ate – при открытии потока курсор устанавливается на конец строки. Запись производится в текущую позицию курсора, а не обязательно в конец строки, как в режимеios::app.

Если параметр не указан, то используются следующие режимы по умолчанию:

- istringstream – ios::in;
- ostringstream – ios::out;
- stringstream – ois::in | ios::out.

Записать строку из экземпляра класса std::string после создания объекта, а также получить строковое представление объекта позволяет метод str().

Прототипы метода:

```
void str(const basic_string &Newstr);
basic_string str() const;
```

Первый прототип очищает содержимое объекта и записывает в него строку из экземпляра класса basic_string. Второй прототип возвращает экземпляр класса basic_string. (std::string это псевдоним для std::basic_string).

Примеры создания строкового потока:

```
// Первый формат
std::ostringstream str;
str << "Строка";
cout << str.str() << endl; // Строка
// Второй формат позволяет указать режим открытия
std::ostringstream str(std::ios::out);
str << "Строка";
cout << str.str() << endl; // Строка
// Третий формат копирует С-строку в объект
std::ostringstream str("Строка",std::ios::out | std::ios::ate);
str << " ABC";
cout << str.str() << endl; // Строка ABC
// Четвертый формат копирует строку из объекта класса string
std::string s("Строка");
std::ostringstream str(s, std::ios::out | std::ios::ate);
str << " ABC";
cout << str.str() << endl; // Строка ABC

std::stringstream ss("ABCDE 123");
std::string str=ss.str();
std::cout << str << '\n'; // ABCDE 123

ss.str("QWERTY");
std::cout << ss.str() << '\n'; // QWERTY

ss >> str;
std::cout << str << '\n'; // QWERTY

std::cout << ss.eof() << '\n'; // 1
```

Для записи в поток используется оператор <<, а также методы put() и write(). Чтение производится с помощью оператора >>, а также методов get(), getline() и read(). Кроме того, можно перемещаться внутри потока и проверять состояние потока.

Преобразование числа в строку и наоборот

```
#include <sstream>

template <typename T>
std::string toString (T x)
{
	std::ostringstream oss;
	oss << x;
	return  oss.str();
}

template <typename T>
T fromString (const std::string &str)
{
	std::istringstream iss(str);
	T x;
	iss >> x;
	return  x;
}

int main()
{
	std::string str;
	int iValue = 123;
	float fValue = 123.45f;

	str = toString(iValue);
	cout << str << endl; // "123"
	str = toString(fValue);
	cout << str << endl; // "123.45"

	return 0;
}

std::stringstream ss("12-3");
int x;
ss >> x;
std::cout << x << '\n' << ss.str() << '\n' << ss.rdbuf() << '\n';
// 12
// 12-3
// -3
ss >> x;
std::cout << ss.fail() << '\n'; // 1
```