Класс string

C++ не поддерживает встроенный строковый тип. Однако он предоставляет два способа обработки строк. Во-первых, для представления строк можно использовать традиционный символьный массив с завершающим нулем. Строки, создаваемые таким способом, иногда называют *С-строками*. Во-вторых, можно использовать объекты класса *string.*

В действительности класс *string* представляет собой специализацию более общего шаблонного класса *basic_string*. Существует две специализации типа *basic_string*: тип *string*, который поддерживает 8-битовые символьные строки, и тип *wstring*, который поддерживает строки, образованные двухбайтовыми символами. Чаще всего в обычном программировании используются строковые объекты типа *string*. Для использования строковых классов C++ необходимо включить в программу заголовок <*string*>.

- Существует три причины для включения в C++ стандартного класса *string*: непротиворечивость данных (строка определяется самостоятельным типом данных), удобство (программист может использовать стандартные C++-операторы) и безопасность (границы массивов не будут нарушаться). Следует иметь в виду, что все выше перечисленное не означает, что вы должны отказываться от использования обычных строк с завершающим нулем. Они по-прежнему остаются самым эффективным средством реализации строк. Но если скорость не является для вас определяющим фактором, использование нового класса *string* даст вам доступ к безопасному и полностью интегрированному способу обработки строк.
- Класс *string* традиционно не воспринимается как часть библиотеки STL, тем не менее он представляет собой еще один контейнерный класс, определенный в C++. Это означает, что он поддерживает алгоритмы, описанные в предыдущем разделе. При этом строки имеют дополнительные возможности.

- Класс *string* содержит множество конструкторов и функций-членов. Кроме того, многие функции-члены имеют несколько перегруженных форм. Обратим внимание только на самые популярные его средства.

  Прототипы самых распространенных конструкторов класса *string* имеют следующий вид.

  ```
  string();
  string(const char *str);
  string (const string &str);
  ```

  Первая форма конструктора создает пустой объект класса *string*. Вторая форма создает *string*-объект из строки с завершающим нулем, адресуемой параметром *str*. Эта форма конструктора обеспечивает преобразование из строки с завершающим нулем в объект типа *string*. Третья создает *string*-объект из другого *string*-объекта.

Для объектов класса *string* определены следующие операторы

Оператор	Описание
=	Присваивание
+	Конкатенация
+=	Присваивание с конкатенацией
==	Равенство
!=	Неравенство
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
[]	Индексация
<<	Вывод
>>	Ввод

- В классе *string* определена константа *npos*, которая равна *-1*. Она представляет размер строки максимально возможной длины.
- Строковый класс C++ существенно облегчает обработку строк. Например, используя *string*-объекты, можно применять оператор присваивания для назначения *string*-объекту строки в кавычках, оператор *"+"* — для конкатенации строк и операторы сравнения — для сравнения строк. Выполнение этих операций демонстрируется в следующей программе.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{string str1("Класс string позволяет эффективно ");
string str2("обрабатывать строки.");
string str3;
// Присваивание string-объекта.
str3 = str1; cout << str1 << "\n" << str3 << "\n";
// Конкатенация двух string-объектов.
str3 = str1 + str2; cout << str3 << "\n";
// Сравнение string-объектов.
if(str3 > str1) cout << "str3 > str1\n";
if(str3 == str1 + str2)
cout << "str3 == str1+str2\n";
/* Объекту класса string можно также присвоить обычную строку. */
str1 = "Это строка с завершающим нулем.\n"; cout << str1;
/* Создание string-объекта с помощью другого string-объекта. */
string str4 (str1);
cout << str4;
// Ввод строки.
cout << "Введите строку: "; cin >> str4;
cout << str4;
return 0;}
```

Основные манипуляции над строками

Чтобы присвоить одну строку другой, используют функцию *assign()*.

string &assign(const string &strob, size_type start, size_type num);

string &assign(const char *str, size_type num);

Первый формат позволяет присвоить вызывающему объекту *num* символов из строки, заданной параметром *strob*, начиная с индекса *start*. При использовании второго формата вызывающему объекту присваиваются первые *num* символов строки с завершающим нулем, заданной параметром *str*. В каждом случае возвращается ссылка на вызывающий объект. Конечно, гораздо проще для присвоения одной полной строки другой использовать оператор *"="*. О функции-члене *assign()* вспоминают, в основном, тогда, когда нужно присвоить только часть строки.

С помощью функции-члена *append()* можно часть одной строки присоединить в конец другой. Два возможных формата ее реализации имеют следующий вид.

```
string &append(const string &strob, size_type start, size_type num);
string &append(const char *str, size_type num);
```

Здесь при использовании первого формата *num* символов из строки, заданной параметром *strob*, начиная с индекса *start*, будет присоединено в конец вызывающего объекта. Второй формат позволяет присоединить в конец вызывающего объекта первые *num* символов строки с завершающим нулем, заданной параметром *str*. В каждом случае возвращается ссылка на вызывающий объект. Гораздо проще для присоединения одной полной строки в конец другой использовать оператор Функция же *append()* применяется тогда, когда необходимо присоединить в конец вызывающего объекта только часть строки.

Вставку или замену символов в строке можно выполнять с помощью функций-членов *insert()* и *replace()*. Прототипы их наиболее употребительных форматов.

```
string &insert(size_type start, const string &strob);
string &insert(size_type start, const string &strob, size_type insStart, size_type num);
string &replace(size_type start, size_type num, const string &strob);
string &replace(size_type start, size_type orgNum, const string &strob, size_type replaceStart, size_type replaceNum);
```

Первый формат функции *insert()* позволяет вставить строку, заданную параметром *strob*, в позицию вызывающей строки, заданную параметром *start*. Второй формат функции *insert()* предназначен для вставки *num* символов из строки, заданной параметром *strob*, начиная с индекса *insStart*, в позицию вызывающей строки, заданную параметром *start*.

Первый формат функции *replace()* служит для замены num символов в вызывающей строке, начиная с индекса *start*, строкой, заданной параметром *strob*. Второй формат позволяет заменить orgNum символов в вызывающей строке, начиная с индекса *start*, *replaceNum* символами строки, заданной параметром *strob*, начиная с индекса *replaceStart*. В каждом случае возвращается ссылка на вызывающий объект

Удалить символы из строки можно с помощью функции *erase()*. Один из ее форматов выглядит так:

```
string &erase(size_type start = 0,
size_type num = npos);
```

Эта функция удаляет *num* символов из вызывающей строки, начиная с индекса *start*. Функция возвращает ссылку на вызывающий объект.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{string str1("Это простой тест.");
string str2("ABCDEFG");
cout << "Исходные строки:\n";
cout << "str1: " << str1 << endl;
cout << "str2: " << str2 << "\n\n";
// Демонстрируем использование функции insert().
cout << "Вставляем строку str2 в строку str1:\n";
str1.insert(5, str2); cout << str1 << "\n\n";
// Демонстрируем использование функции erase().
cout << "Удаляем 7 символов из строки str1:\n";
str1.erase(5, 7); cout << str1 <<"\n\n";
// Демонстрируем использование функции replace().
cout << "Заменяем 2 символа в str1 строкой str2:\n";
str1.replace(5, 2, str2);
cout << str1 << endl;
return 0;}
```

Результаты выполнения этой программы таковы.

Исходные строки:

str1: Это простой тест.

str2: ABCDEFG

Вставляем строку str2 в строку str1:

Это пABCDEFGростой тест.

Удаляем 7 символов из строки str1:

Это простой тест.

Заменяем 2 символа в str1 строкой str2:

Это пABCDEFGстой тест.

Поиск в строке

В классе *string* предусмотрено несколько функций-членов, которые осуществляют поиск. Это, например, такие функции, как *find()* и *rfind()*. Рассмотрим прототипы самых употребительных версий этих функций.

```
size_type find(const string &strob, size_type start=0) const;
size_type rfind(const string &strob, size_type start=npos) const;
```

Функция *find()*, начиная с позиции *start*, просматривает вызывающую строку на предмет поиска первого вхождения строки, заданной параметром *strob*. Если поиск успешен, функция *find()* возвращает индекс, по которому в вызывающей строке было обнаружено совпадение. Если совпадения не обнаружено, возвращается значение *npos*. Функция *rfind()*выполняет то же действие, но с конца. Начиная с позиции *start*, она просматривает вызывающую строку в обратном направлении на предмет поиска первого вхождения строки, заданной параметром *strob* (т.е. она находит в вызывающей строке последнее вхождение строки, заданной параметром *strob*). Если поиск прошел удачно, функция *rfind()* возвращает индекс, по которому в вызывающей строке было обнаружено совпадение. Если совпадения не обнаружено, возвращается значение *npos*.

Рассмотрим короткий пример использования функции *find()*.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{int i;
string s1 ="Класс string облегчает обработку строк.";
string s2;
i = s1.find("string");
if(i != string::npos) {
cout << "Совпадение обнаружено в позиции " << i<< endl;
cout << "Остаток строки таков: ";
s2.assign (s1, i, s1.size());
cout << s2;}
return 0;}
```

Программа генерирует такие результаты.

```
Совпадение обнаружено в позиции 6
Остаток строки таков: string облегчает обработку строк.
```

Сравнение строк

Чтобы сравнить полное содержимое одного *string*-объекта с другим, обычно используются описанные выше перегруженные операторы отношений. Но если нужно сравнить часть одной строки с другой, придется использовать функцию-член *compare()*.

```
int compare(size_type start, size_type num, const string &strob) const;
```

Функция *compare()* сравнивает с вызывающей строкой *num* символов строки, заданной параметром *strob*, начиная с индекса *start*. Если вызывающая строка меньше строки *strob*, функция *compare()* возвратит отрицательное значение. Если вызывающая строка больше строки *strob*, она возвратит положительное значение. Если строка *strob* равна вызывающей строке, функция *compare()* возвратит нуль.

Получение строки с завершающим нулем

Несмотря на неоспоримую полезность объектов типа *string*, возможны ситуации, когда вам придется получать из такого объекта символьный массив с завершающим нулем, т.е. его версию С-строки. Например, использовать *string*-объект для создания имени файла. Но, открывая файл, вам нужно задать указатель на стандартную строку с завершающим нулем. Для решения этой проблемы и используется функция-член *c_str()*. Вот как выглядит ее прототип:

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

Эта функция возвращает указатель на *С-версию* строки (т.е. на строку с завершающим нулевым символом), которая содержится в вызывающем объекте типа *string*. Полученная строка с завершающим нулем изменению не подлежит. Кроме того, после выполнения других операций над этим *string*-объектом допустимость применения полученной *С-строки* не гарантируется.