БлогNot. Лекции по C/C++: строки класса string

Лекции по C/C++: строки класса string

В современном стандарте C++ определен класс с функциями и свойствами (переменными) для организации работы со строками (в классическом языке C строк как таковых нет, есть лишь массивы символов `char`):

```
#include <string>
```

> Не путайте с подключением Си-совместимой библиотеки для работы со строками `char *`:
>
> ```
> #include <string.h>
> ```
>
> или
>
> ```
> #include <cstring>
> ```

Для работы со строками также нужно подключить стандартный `namespace`:

```
using namespace std;
```

В противном случае придётся везде указывать описатель класса `std::string` вместо `string`.

Ниже приводится пример программы, работающей со `string` (в старых си-совместимых компиляторах не работает!):

```
#include <iostream>
#include <string>
#include <malloc.h>
using namespace std;

int main () {
 string s = "Test";
 s.insert (1,"!");
 cout << s.c_str() << endl;
 string *s2 = new string("Hello");
 s2->erase(s2->end());
 cout << s2->c_str();
 cin.get(); return 0;
}
```

Основные возможности, которыми обладает 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 использовать функцию `strcpy()`;

- доступ к отдельным символам строки для чтения и записи. Во встроенном массиве для этого применяется операция взятия индекса или косвенная адресация с помощью указателя;
- сравнение двух строк на равенство. Для встроенного типа используются функции семейства `strcmp()`;
- конкатенация (сцепление) двух строк, дающая 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()`, а также позаботиться о выделении памяти;
- встроенные средства определения длины строки (функции-члены класса `size()` и `length()`). Узнать длину строки встроенного типа можно только вычислением с помощью функции `strlen()`;
- возможность узнать, пуста ли строка.

Рассмотрим эти базовые возможности более подробно.

Инициализация строк при описании и **длина строки** (не включая завершающий нуль-терминатор):

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

Строка может быть задана и пустой:

```
string st2;
```

Для проверки того, **пуста ли строка**, можно сравнить ее длину с 0:

```
if ( ! st.size() ) // пустая
```

или применить метод `empty()`, возвращающий `true` для пустой строки и `false` для непустой:

```
if ( st.empty() ) // пустая
```

Третья форма создания строки инициализирует объект типа `string` другим объектом того же типа:

```
string st3( st );
```

Строка `st3` инициализируется строкой `st`. Как мы можем убедиться, что эти **строки совпадают**? Воспользуемся оператором сравнения (==):

```
if ( st == st3 ) // инициализация сработала
```

Как **скопировать одну строку в другую**? С помощью обычной операции присваивания:

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

Для **сцепления строк** используется операция сложения (+) или операция сложения с присваиванием (+=). Пусть даны две строки:

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

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

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

Если же мы хотим добавить `s2` в конец `s1`, мы должны написать:

```
s1 += s2;
```

Операция сложения может сцеплять 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 `char *`, а значимые слова – объектами класса `string`:

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

Обратное преобразование при этом **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 с названием `c_str()` ("строка Си"):

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

Функция `c_str()` возвращает указатель на символьный массив, содержащий строку объекта `string` в том виде, в каком она находилась бы во встроенном строковом типе. Ключевое слово `const` здесь предотвращает "опасную" в современных визуальных средах возможность непосредственной модификации содержимого объекта через указатель.

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( "www.disney.com" );
int size = str.size();
for ( int i = 0; i < size; i++ )
 if ( str[i] == '.' ) str[ i ] = '_';
cout << str;
```

Но лучше читать документацию по C++ и пользоваться его возможностями. Например, предыдущее действие мы могли бы выполнить вызовом одной-единственной функции `replace()`:

```
replace( str.begin(), str.end(), '.', '_' );
```

Правда, здесь использован не метод `replace` класса `string`, а одноимённый алгоритм:

```
#include <algorithm>
```

Поскольку объект `string` ведет себя как контейнер, к нему могут применяться и другие алгоритмы. Это позволяет решать задачи, не решаемые напрямую функциями класса `string`.

Ниже приводится краткое описание основных операторов и функций класса `string`, ссылки в таблице ведут к русскоязычным описаниям в интернете. Более полный список возможностей класса `string` можно получить, например, в Википедии или на сайте cplusplus.com.

Задание символов в строке	
operator=	присваивает значения строке
assign	назначает символы строке
Доступ к отдельным символам	
at	получение указанного символа с проверкой выхода индекса за границы
operator[]	получение указанного символа
front	получение первого символа
back	получение последнего символа
data	возвращает указатель на первый символ строки
c_str	возвращает немодифицируемый массив символов C, содержащий символы строки
Проверка на вместимость строки	
empty	проверяет, является ли строка пустой
size length	возвращает количество символов в строке
max_size	возвращает максимальное количество символов
reserve	резервирует место под хранение
Операции над строкой	
clear	очищает содержимое строки
insert	вставка символов
erase	удаление символов
push_back	добавление символа в конец строки
pop_back	удаляет последний символ
append	добавляет символы в конец строки
operator+=	добавляет символы в конец строки
compare	сравнивает две строки
replace	заменяет каждое вхождение указанного символа
substr	возвращает подстроку

copy	копирует символы
resize	изменяет количество хранимых символов
swap	обменивает содержимое
Поиск в строке	
find	поиск символов в строке
rfind	поиск последнего вхождения подстроки
find_first_of	поиск первого вхождения символов
find_first_not_of	найти первое вхождение отсутствия символов
find_last_of	найти последнее вхождение символов
find_last_not_of	найти последнее вхождение отсутствия символов

Следует обратить внимание, что у любой функции класса `string` может быть несколько *перегрузок* - разновидностей с одинаковыми именами, отличающихся между собой списками и типами аргументов.

В качестве недостатков класса `string` можно отметить следующее:

- отсутствие в классе встроенных средств для разбора строк по набору разделителей (аналога функции `strtok` для строк `char *`);
- возможное замедление быстродействия по отношению к `char *` при сложной обработке данных.

Ниже приведён код для разбора введённой с клавиатуры строки `string` на слова. Можно доработать этот код, исключив знаки препинания, стоящие последними символами строк `vecstr[i]`, а также слова, не содержащие ни одного алфавитно-цифрового символа.

```
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
using namespace std;

int main() {
 cout << "Enter the string: ";
 string str;
 getline(cin, str);

 vector <string> vecstr;
 string word;
 stringstream s(str);

 while (s >> word) vecstr.push_back(word);

 int vsize = vecstr.size();
 for (int i = 0; i < vsize; i++)
  cout << vecstr[i] << endl;
```

```
 cin.get();
 return 0;
}
```

При написании программы дополнительно использованы возможности стандартных классов vector и и stringstream. Вот здесь пример показан и в "классическом" стиле разбора.

Оглавление серии

теги: textprocessing c++ учебное

Здесь можно оставить комментарий, обязательны только выделенные цветом поля. Не пишите лишнего, и всё будет хорошо

Ваше имя:

Пароль (если желаете запомнить имя):

Любимый URL (если указываете, то вставьте полностью):

Текст сообщения (до 1024 символов):

Введите код сообщения: 48_{17}

Добавить Сброс

05.11.2015, 08:33; рейтинг: 79847