ЛЕКЦИЯ 13.

Стандартный класс string

С не поддерживает стандартного типа «строка». Вместо этого надо работать с массивами символов, завершающимися символом с кодом тип **char** *). Функции для работы с этим типом данных отличаются высокой производительностью, но весьма неудобны и небезопасны в использовании, поскольку проверка на выход за границы отведенной памяти не производится. Для того чтобы использовать объекты класса string, необходимо включить заголовочный файл:
#include <string>

Тип данных (точнее, класс) **string** лишен этих недостатков, хотя и проигрывает типу char * в производительности.

Основные действия со строками выполняются в нем с помощью операций и методов, а длина строки изменяется динамически в соответствии с потребностями пользователя.

Определение объектов класса string и присваивание значений

```
1) string s1(”осень”);
2) string s2=”зима”;
3) string s3, s4; //пустые строки
   s3=s2; //присваивание строк
   s4=”Ни ”+s1+” ни ”;
   //объединение строк (конкатенация)
   s4+=s2;
   //добавление в конец строки
4) string s5(s4);
// инициализация объекта типа string
// другим объектом того же типа.
```

Операция сложения может конкатенировать объекты класса string не только между
собой, но и со строками встроенного типа. `string s1(”hello”);`

```
const char *pc = ”, ”;
string s2(”world”);
string s3 = s1 + pc + s2 + "\n";
```

Подобные выражения работают потому, что компилятор знает, как автоматически преобразовывать объекты встроенного типа в объекты класса string. Конструкторы и операция присваивания для класса **string** определены

таким образом, что можно работать и с типом **char *** , так что допустимы следующие фрагменты программ:

```
const char * sc1="Hello!";
string s1,   // пустая строка
       s2("Good bye!"),
       s3(sc1);
s1=sc1;
```

С другой стороны, перевод из типа **string** в тип **char** * достаточно сложен. Класс **string** имеет в своем составе метод **c_str**, который возвращает указатель на нуль-терминированную строку, однако дополнительная память при этом не выделяется, метод имеет прототип

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

т.е. он возвращает указатель на константу.

Для полноценной работы с такой строкой необходимо записать следующий код:

```
string s1;
// задаем значение для s1
const char *sc1=s1.c_str();
char *sc2=new char[strlen(sc1)+1];
strcpy(sc2, sc1);
// и далее работаем со строкой sc2
```

Еще один вариант реализации этого механизма:

```
char *sc2= new char[s1.length()+1];
strcpy(sc2, s1.c_str()));
```

С другой стороны, запись

```
char *sc2 = sc1;
```

приведет к ошибке компиляции из-за неверного преобразования типов.

Ввод-вывод объектов класса string

```
string name, address;
cout << "ваше фио?" << endl;
getline(cin, name);
//ввод строки
cout <<
"ваш адрес в несколько строк?\n";
cout<<"Окончание ввода-$\n";
getline(cin, address, $);
cout<<"Ваше имя: "<<name<<endl;
cout<<"Ваш адрес: "<<address<<endl;
```

Основные методы класса string

В описании этих функций тип **size_type** представляет собой беззнаковый

целый тип, достаточный для хранения размера самой большой строки, а поле **npos** класса **string** представляет собой самое большое положительное число типа **size_type**.

Получение характеристик строки

size_type length()
size()
длина строки (количество символов в ней)
max_size()
максимальная длина, которая может быть у строки (4.3 млрд. байт для MSVC 2005)
capacity()
объем памяти, занимаемый строкой (с учетом служебных полей)
bool empty()
истина, если строка пуста

Поиск объектов класса string (подстрок)

size_type find(строка1, позиция=0)
Ищет первое вхождение строки1, начиная с заданной позиции, слева.
size_type find(символ, позиция=0)
То же для символа
size_type rfind(строка1,позиция=npos)
То же для поиска справа строки1
size_type rfind(символ, позиция=npos)
То же для символа
size_type find_first_of(строка1, позиция=0)
Ищет первое вхождение любого символа из строки1, начиная с заданной позиции, слева.
size_type find_last_of(строка1, позиция=npos)
То же для поиска справа.
Все эти функции возвращают -1, если цель не найдена.
(В старом варианте был исходный номер позиции)!!!!!!!!!!

Модификация объектов класса string

insert(позиция, строка1)
Вставляет в строку ***this** значение строки1, начиная с указанной позиции. Строка1 может быть как типа string, так и char *.
erase(позиция=0, число_символов = остаток)
Удаляет указанное число символов, начиная с заданной позиции (или остаток строки).

clear()
Очищает строку
substr(позиция=0,число_символов=остаток)
Выделяет в качестве подстроки указанное число символов, начиная с заданной позиции (или остаток строки).
replace(позиция, число_символов,строка1)
Заменяет указанное число символов, начиная с заданной позиции на строку1.
append(число_символов,символ)
Добавляет указанное число символов в конец строки

Сравнение объектов класса string

compare(позиция, число_символов, строка1, позиция1, число_символов1)
Сравнивает **число_символов** строки ***this**, начиная с **позиции** и **число_символов1** строки1, начиная с указанной **позиции1**.
Функция возвращает 0, если совпадают, -1, если идут до и +1, если идут после.

Доступ к символам в объектах класса string

К отдельным символам объекта типа string можно обращаться с помощью операции взятия индекса. 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] = '_';
```

Объекты класса string не заканчиваются нулевым символом, у них хранится в отдельной переменной длина строки. Поэтому не пытайтесь искать признак конца строки.

```
string s1(”прекрасная королева”), s2, s3=”ворона”;
s2=s1;
s1.erase(0,3);
cout<<s1.erase(12,2)<<endl;
s1.replace(8,6,s3);
```

Более подробно см. в литературе, например,Т.А.Павловская. С/С++. Программирование на языке высокого уровня. Глава 11.
КОНЕЦ ЛЕКЦИИ