«Знакомство с библиотечным классом string»

Материалы для самостоятельной работы

Постановка задачи

Настоящее занятие ставит своей задачей предварительное ознакомление студентов с классом string стандартной библиотеки C++. Во время занятия студент должен выполнить на компьютере ряд заданий. В процессе выполнения этих заданий следует обращать внимание на отличия в работе со строкой типа string и строками языка Си. Для работы со строкой класса string важно различать два параметра:

- `size (length)`,
- `capacity`.

Первый из параметров (size) определяет размер строки, сколько символов содержится в строке , а второй (capacity) определяет общий объем памяти, выделенный для хранения строки. В общем случае память выделяется с запасом.

Сравнение строк класса string и строк в стиле языка Си

1. Класс string определяет тип данных. Использование строк языка Си построено на принятых в этом языке соглашениях.
2. Класс string инкапсулирует последовательность символов, образующих текст строки. Это позволяет исключить недопустимые операции. К числу таких операций можно отнести попытку получить строку, имеющую длину, превышающую максимально допустимую длину.
3. Класс string определяет операции, обеспечивающие работу со строками.
4. Строки класса string относятся к категории динамических объектов. В случае необходимости они могут расширяться, чтобы выделить дополнительную память для сохранения модифицированной строки.
5. К недостаткам строк класса string можно отнести наличие дополнительных временных расходов, которые связаны с перераспределением памяти во время выполнения ряда операций.

Интерфейс класса string

№	Имя типа или константы. Прототип функции	Пример применения	Пояснение
1	size_type	size_type length();	size_type - беззнаковый тип, объявленный в классе string.В примере используется для представления типа

			значения, возвращаемого функцией length().
2	npos		npos - символическая константа, объявленная в классе string, используемая для представления максимального значения данных типа size_type.
3	string();	string str;	**Конструктор умолчания**. Создаёт пустую строку. В примере создаётся пустая строка str.
4	string(const string& s);	string s_old; //... string s_new(s_old);	**Конструктор копирования**. В примере строка s_new является копией ранее созданной строки s_old.
5	string(const char* s);	string str(“Hello”);	**Конструктор-преобразователь типа**. В примере литерал языка Си преобразуется в строку класса string.
6	string& operator = (const string& s);	string s1, s2; // s1 = s2;	**Перегруженный оператор присваивания.** В последней строке кода, приведённого в примере, будет вызван перегруженный оператор присваивания.
7	string& append(const string& s);	string str1, str2; //... str1.append(str2);	**Объединяет две строки.** В примере это строки str1 и str2. Объединённая строка создаётся путём добавления строки str2 в конец строки str1.
8	string& append(const char* s);	string str; char s[81] = “Hello”; str.append(s);	**Объединяет строку класса string со строкой в стиле Си**.. В примере это str и s. Объединённая строка создаётся путём добавления преобразованной к типу string строки s в конец строки str.
9	string& operator+= (const string& s);	string str1, str2; //... str1 += str2;	**Объединяет две строки.** В примере это строки str1 и str2. Объединённая строка создаётся путём добавления строки str2 в конец строки str1. Для объединения используется перегруженный оператор +=.
10	string operator+ (const string& s);	string str1, str2; // string str3 = str1 + str2;	**Объединяет две строки.** В примере это строки str1 и str2. В процессе объединения создаётся новая строка. Для сохранения новой строки в стеке создаётся временная строка. Объявленная в примере строка str3 служит для сохранения временной строки.

11	string& insert(size_type pos, const string& s);	string str1(“01234567”); string str2(“ABCD”); str1.insert(5, str2);	**Вставка строки**. Строка, являющаяся вторым аргументом (в примере — str2), будет вставлена перед позицией, определяемой первым аргументом (в примере — 5) строки (в примере — str1), для которой вызывается функция insert().
12	string& erase(size_type pos = 0, size_type len = npos);	string str(“01234567”); str.erase(1, 4);	**Удаление символов из строки**. Начиная с позиции pos, удаляет len символов.
13	string substr(size_type pos = 0, size_type len = npos);	string str(“01234567”); string str2 = str.substr(1, 4);	**Получение подстроки.** Возвращает часть строки длиной len, начиная с позиции pos. В примере в строке str2 будет сохранена подстрока, состоящая из следующих символов исходной строки str - 1234
14	size_type find(const string s, size_type pos = 0);		**Выполняет поиск подстроки в строке, для которой вызывается функция.** Поиск начинается с позиции, задаваемой параметром pos. Возвращает позицию первого вхождения подстроки s. В том случае, когда подстрока не будет найдена, функция find() возвращает значение npos.
15	size_type length();	string str(“012345”); cout << str.length();	**Возвращает длину строки.** В примере на экране будет выведено число 6.
16	size_type capacity();	string str(“012345”); cout << str.capacity();	**Возвращает общий объем памяти, выделенный для строки.** В примере на экране должно появиться число 6 (Для Qt Creator). В общем случае объем памяти может превышать длину строки.
17	void reserve(size_type num);	string str; str.reserve(500);	**Резервирует память.** Внутренняя память резервируется, по крайней мере для num символов.
18	void reserve();	string str; //... str.reserve();	**Запрос на сокращение емкости.** Емкость никогда не сокращается до величины, меньшей текущего размера памяти.

Задание № 1 Создание пустой строки

Для создания пустой строки следует предусмотреть вызов конструктора без параметров (см. программный код, приведённый ниже). Вызов конструкторов происходит автоматически при компиляции определения переменной классового (пользовательского) типа.

Объявления пустой строки имеет следующий формат

```
string <имя переменной>;
```

Например:

```
string str;
```

При компиляции определения переменной str будет автоматически вызван конструктор умолчания, прототип которого содержится в строке 3 таблицы, приведённой выше.

Обратите внимание на то обстоятельство, что здесь объявляется переменная, а не массив, как принято делать при работе со строкой в стиле языка Си. Вам не надо обдумывать, какой размер памяти надо выбрать для хранения строки str. В процессе выполнения этого задания следует убедиться в том, что действительно будет создана пустая строка. Для этого достаточно вывести содержимое этой строки на экран. Это можно сделать с помощью предопределённого объекта cout и перегруженного оператора <<. Можно дополнительно убедиться в том, что строка str не будет содержать текст. Для этого можно воспользоваться вызовом функции length(), входящей в интерфейс класса string (строка 15 таблицы). Эта функция для пустой строки должна возвращать нулевое значение. Кроме того, можно экспериментально оценить общего объёма памяти, который будет выделен для хранения.

```
#include <iostream>// Для работы с предопределенным объектом cout
#include <string>     //  Для работы с классом string
using std::cout;   // Для использования имени cout без уточнения namespace std,
          // где это имя объявлено; в противном случае надо использовать std :: cout
using std :: endl; // Для использования имени endl без уточнения namespace std,
          // где это имя объявлено; в противном случае надо использовать std :: endl
using std :: string;
int main()
{
    string str;      // Контролируемая операция — конструктор умолчания
    cout << "str-> " << str << endl;
    cout << "----------------------------------------" << endl;
    cout << "str.length() = " << str.length() << endl;// Вызов
                              // функции для вычисления длины строки
    cout << "----------------------------------------" << endl;
    cout << "str.capacity() = " << str.capacity() << endl;
          //  Вызов функции для определения общего размера памяти,  выделенной
                                                    //  для строки str
    cout << "----------------------------------------" << endl;
    return 0;
}
```

Задание №2 Преобразование строки в стиле языка Си в объект типа string

Для преобразования можно воспользоваться конструктором преобразователем типа (см. строку 5 таблицы, приведённой выше). В порядке знакомства с этим видом конструктора выполните программный код, приведённый ниже. Убедитесь в правильности преобразования, обратите внимание на объем памяти, который будет выделен для хранения объекта str.

```
#include <iostream>
#include <string>
using std :: cout;
using std :: cin;
using std :: endl;
using std :: string;
int main()
{
    string str(“1234”); // Контролируемая операция – конструктор -
                                                    // преобразователь типа
    cout << "str-> " << str << endl;
    cout << "----------------------------------------" << endl;
    cout << "str.length() = " << str.length() << endl;
    cout << "----------------------------------------" << endl;
    cout << "str.capacity() = " << str.capacity() << endl;
    cout << "----------------------------------------" << endl;
    return 0;
}
```

Задание №3 Использование конструктора копирования

В программном коде, приведенном ниже, для создания копии существующего объекта str1 используется конструктор копии (см. строку 15 таблицы).Убедитесь в том, что копия создается. Обратите внимание на объем памяти, который будет занимать строка str2, которая является копией строки str1(объем для выделяемой памяти для копии и для оригинала должен совпадать). Вопрос для обдумывания: «Какие соображения заставили разработчиков класса string создать собственную версию конструктора копирования».

```
#include <iostream>
#include <string>
using std :: cout;
using std :: cin;
using std :: endl;
using std :: string;
int main()
{
    string str1(“1234”);
    string str2(str1); // Контролируемая операция – конструктор копирования
```

```
    cout << "str2-> " << str2 << endl;
    cout << "----------------------------------------" << endl;
    cout << "str2.length() = " << str2.length() << endl;
    cout << "----------------------------------------" << endl;
    cout << "str2.capacity() = " << str2.capacity() << endl;
    cout << "----------------------------------------" << endl;
    return 0;
}
```

Задание №4 Копирование строк с помощью оператора присваивания

Убедитесь в том, что объекты класса string можно копировать с помощью оператора присваивания. Обратите внимание на то обстоятельство, что в интерфейсе класса имеется перегруженный вариант этого оператора. Перегруженный оператор будет заменять тот оператор, который создается компилятор по умолчанию. Вопрос для обдумывания: «Какие соображения заставили разработчиков класса string реализовать перегрузку оператора присваивания?». Вспомните, что для строк в стиле Си такое копирование не возможно. Выполните следующий код.

```
#include <iostream>
#include <string>
using std :: cout;
using std :: cin;
using std :: endl;
using std :: string;
int main()
{
    string str1(“1234”);
    cout << "str1-> " << str1 << endl;
    cout << "----------------------------------------" << endl;
    cout << "str1.length() = " << str.length() << endl;
    cout << "----------------------------------------" << endl;
    cout << "str1.capacity() = " << str.capacity() << endl;
    cout << "----------------------------------------" << endl;

    string str2;
    str2 = str1;       // Контролируемая операция – оператор присваивания
    out << "str2-> " << str2 << endl;
    cout << "----------------------------------------" << endl;
    cout << "str2.length() = " << str2.length() << endl;
    cout << "----------------------------------------" << endl;
    cout << "str2.capacity() = " << str2.capacity() << endl;
    cout << "----------------------------------------" << endl;
    return 0;
}
```

Задание №5 Сцепление строк

Объединение, или сцепление, или конкатенация может быть выполнена двумя способами:

- путем применения перегруженного оператора «+=».
- вызова функции append().

```
#include <iostream>
#include <string>
using std :: cout;
using std :: cin;
using std :: endl;
using std :: string;
int main()
{
    string str1(“1234”);
    string str2(“5678”);
    str2 + = str1;      // Конролируемая операция перегруженный составной
                        // оператор присваивания:  +=
                        // Альтернативное решение: str2.append(str1);
    cout << "str2-> " << str2 << endl;
    cout << "----------------------------------------" << endl;
    cout << "str2.length() = " << str.length() << endl;
    cout << "----------------------------------------" << endl;
    cout << "str2.capacity() = " << str.capacity() << endl;
    cout << "----------------------------------------" << endl;
    return 0;
}
```

Задание №6 Выделение подстроки

Выделение подстроки выполняется с помощью функции substr(), которая имеет следующий прототип (см. строку 13 таблицы):

```
string substr( size_type pos = 0, size_type len = npos);
```

С помощью этой функции выполняется попытка выделить подстроку из n символов. Выделение начинается с позиции pos.

```
#include <iostream>
#include <string>
using std :: cout;
using std :: cin;
using std :: endl;
using std :: string;
int main()
{
    string str(“012345”);
    string sub = str.substr (1, 3); // Проверяемая операция
    cout << "sub-> " << sub << endl;
    cout << "----------------------------------------" << endl;
```

```
    cout << "sub.length() = " <<sub.length() << endl;
    cout << "----------------------------------------" << endl;
    cout << "sub.capacity() = " << sub.capacity() << endl;
    cout << "----------------------------------------" << endl;
    return 0;
}
```

Задание №7 Напишите программу для определения количества расширений строки

Организуйте арифметический цикл, в котором на каждом шаге длина строки будет увеличивается на 1. Для этого можно воспользоваться, например, вызовом функции push_back, которая добавляет один символ в конец строки и увеличивает её длину на 1. Поддерживайте в программе две переменные cap_old и cap_cur, которые должны отслеживать два значения ёмкости (capacity). Переменная cap_old должна следить за значением ёмкости до расширения, а переменная cap_cur – после расширения. В программе должен действовать счётчик, например, count. Значение счётчика должно увеличиваться на единицу на тех шагах выполнения цикла, где значения переменных cap_old и cap_cur не совпадают.

Задание №8 Напишите программу для анализа динамики расширения строки.

Организуйте арифметический цикл, в котором на каждом шаге длина строки будет увеличивается на 1. Для этого можно воспользоваться, например, вызовом функции push_back. Поддерживайте в программе две переменные cap_old и cap_cur, которые должны отслеживать два значения ёмкости (capacity). Переменная cap_old должна следить за значением ёмкости до расширения, а переменная cap_cur – должна следить за значением ёмкости после расширения. В программе должна быть реализована логика, определяющая момент расширения. Например:

```
if(cap_old != cap_cur)
{
        // Расширение строки имело место. Вывести новое значение ёмкости
        // на экран или записать его в массив.
}
```