1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт компьютерных наук и кибербезопасности**

**ЛАБОРАТОРНАЯ РАБОТА №1**

1. «Класс MyString»
2. по дисциплине «Объектно-ориентированное программирование»

Санкт-Петербург

2024

# Цель работы

Целью работы – знакомство с базовыми понятиями объектно-ориентированного программирования «класс» и «объект», а также с принципом инкапсуляции.

# Задачи

В рамках выполнения лабораторной работы необходимо решить следующие задачи:

* реализовать класс для представления символьной строки на C++, не используя контейнеры и алгоритмы библиотеки STL;
* расширь функциональность интерпретатора Python с помощью реализованного класса.

# условие

При выполнения лабораторной работы необходимо соблюдать следующие правила:

1. Имя класса – «MyString».
2. Не использовать функции C, когда есть замена в C++ (например, printf, malloc, realloc и другие);
3. Не дублировать один и тот же фрагмент кода в разных функциях.
4. Поведение функции в случае её вызова с неверными значениями аргументов должно быть определено.
5. Емкость (capacity) изменяется в меньшую сторону только в функции shrink\_to\_fit.
6. В процессе работы с объектами класса MyString не должно быть утечек памяти.

В таблице 1 приведен перечень функций, которые необходимо реализовать в рамках класса MyString.

Таблица 1 – Функции, которые необходимо реализовать в рамках класса MyString

| **№** | **Название функции** | **Описание** | **Пример использования** | **Вывод** |
| --- | --- | --- | --- | --- |
| Конструкторы | | | | |
| 1 | Constructor() | Конструктор по умолчанию | MyString a1;  std::cout<<a1<<" "<<a1.size()<<" "<<a1.capacity(); | 0 0 |
| 2 | Constructor(“char string”) | Конструктор по массиву char | MyString a2("Hello world!");  std::cout<<a2<<" "<<a2.size()<<" "<<a2.capacity()<<std::endl; | Hello world! 12 13 |
| 4 | Constructor(std::string) | Конструктор по объекту std::string | std::string s="hello";  MyString a4(s);  std::cout<<a4<<" "<<a4.size()<<" "<<a4.capacity()<<std::endl; | hello 5 6 |
| 5 | Constructor(“char string”, count) | Конструктор, инициализирующий класс заданным числом символов строки «char string» | MyString a5("hello",4);  std::cout<<a5<<" "<< a5.size()<<" "<< a5.capacity()<<std::endl; | hell 4 5 |
| 6 | Constructor(count, char) | Конструктор, инициализирующий класс заданным числом символов | MyString a6(5,'!');  std::cout<<a6<<" "<< a6.size()<<" "<< a6.capacity()<<std::endl; | !!!!! 5 6 |
| 7 | Constructor(MyString) | Конструктор копирования | MyString a7("hello");  MyString a8(a7);  std::cout<<a8<<" "<< a8.size()<<" "<< a8.capacity()<<std::endl; | hello 5 6 |
| 8 | Destructor() | Деструктор |  |  |
| Операторы | | | | |
| 1 | Operator +(Mystring) | Оператор объединения текущего объекта класса с MyString | MyString a1("hel"), a2("lo"), a3;  a3=a1+a2;  std::cout<<a1<<" "<<a2<<" "<<a3 <<std::endl; | hel lo hello |
| 2 | Operator +(“char string”) | Оператор объединения текущего объекта класса с массивом char | MyString a4("hel"), a5;  a5=a4+"lo";  std::cout<<a4<<" "<<a5 <<std::endl; | hel hello |
| 3 | Operator +(std::string) | Оператор объединения текущего объекта класса с std::string | MyString a6("hel"), a7;  std::string s1 = "lo";  a7=a6+s1;  std::cout<<a6<<" "<<a7<<std::endl; | hel hello |
| 4 | Operator +=(“char string”) | Оператор добавления массива char к текущему объекту класса | MyString a1("hel"), a2("lo");  a1+=a2;  std::cout<<a1<<" "<<a2<<std::endl; | hello lo |
| 5 | Operator +=(std::string) | Оператор добавления std::string к текущему объекту класса | MyString a3("hel");  std::string s1="lo";  a3+=s1;  std::cout<<a3<<std::endl; | hello |
| 6 | Operator=(“char string”) | Оператор присваивания массива char | MyString a1;  a1="hello";  std::cout<<a1<<std::endl; | hello |
| 7 | Operator=(std::string) | Оператор присваивания std::string | MyString a2;  std::string s1="hello";  a2=s1;  std::cout<<a2<<std::endl; | hello |
| 8 | Operator=(‘char’) | Оператор присваивания символа | MyString a3;  a3='!';  std::cout<<a3<<std::endl; | ! |
| 9 | Opearator[](int) | Оператор взятия значение по индексу | MyString a4;  a4="hello";  std::cout<<a4[2]<<std::endl; | l |
| 10 | Operator>()  Operator<()  Operator>=()  Operator<=()  Operator!=()  Operator==() | Лексикографическое сравнение | MyString a, b;  a="abcd";  b="abce";  std::cout<<(a==b)<<(a!=b)<<(a>b)<<  (a>=b)<<(a<b)<<(a<=b)<< std::endl; | See how the comparison works in std::string. |
| 11 | c\_str() | Метод возвращает указатель на массив данных, который обязательно должны завершаться нулем |  |  |
| 12 | data() | Метод возвращает указатель на массив данных, который не обязательно должны завершаться нулем | MyString a1("Hello world!");  std::string s1;  s1 = a1.data();  std::cout<<s1<<std::endl; | Hello world! |
| 13 | length() | Аналогичен size |  |  |
| 14 | size() | Метод возвращает число элементов типа char в строке | MyString a1("Hello world!");  std::cout<<s1.size()<<std::endl; | 12 |
| 15 | empty() | Метод возвращает true если строка пустая | MyString a1("Hello world!");  std::cout<<s1.empty()<<std::endl; | 0 |
| 16 | capacity() | Метод возвращает объем (capacity) выделенной для хранения строки памяти | MyString a1("Hello world!");  std::cout<<s1.capacity()<<std::endl; | 13 |
| 17 | shrink\_to\_fit() | Уменьшает объем (capacity) до длины строки (size) | MyString a1("Hello world!");  a1.erase(5, 6);  std::cout<<a1<<" "<<a1.size()<<" "<<a1.capacity()<<std::endl;  a1.shrink\_to\_fit();  std::cout<<a1<<" "<<a1.size()<<" "<<a1.capacity()<<std::endl; | Hello! 6 13  Hello! 6 7 |
| 18 | clear() | Удаление элементов char из строки | MyString a1("Hello world!");  std::cout<<a1<<std::endl;  a1.clear();  std::cout<<a1<<std::endl; | Hello world! |
| 19 | Operator<<(std::basic\_ostream), Operator>>(std::basic\_istream) | Операторы ввода / вывода | MyString a1("Hello world!");  std::cout<<a1<<std::endl; | Hello world! 12 13  0 13 |
| Вставка по индексу | | | | |
| 1 | insert(index, count, char) | Вставка conut символов char по индексу | MyString a1 = "aaaaa";  a1.insert(0,1,'!');  std::cout<<a1<<std::endl;  a1.insert(3, 2, '@');  std::cout << a1 << std::endl; | !aaaaa  !aa@@aaa |
| 2 | insert(index, “string”) | Вставка строки по индексу | MyString a2 = "aaaaa";  a2.insert(1,"@@@@@");  std::cout<<a2<<std::endl; | a@@@@@aaaa |
| 3 | insert(index, “string”, count) | Вставка count символов строки по индексу | MyString a3 = "aaaaa";  a3.insert(1,"@@@@@",2);  std::cout<<a3<<std::endl; | a@@aaaa |
| 4 | insert(index, std::string) | Вставка строки std::string по индексу | MyString a4 = "aaaaa";  std::string s1 = "@@@@@";  a4.insert(1,s1);  std::cout<<a4<<std::endl; | a@@@@@aaaa |
| 5 | insert(index, std::string, count) | Вставка count символов строки std::string по индексу | MyString a5 = "aaaaa";  std::string s2 = "@@@@@";  a5.insert(1,s2,2);  std::cout<<a5<<std::endl; | a@@aaaa |
| Удаление подстроки | | | | |
| 1 | erase(index, count) | Удаление count символов по индексу | MyString a1("Hello world!");  a1.erase(5, 6);  std::cout<<a1<<std::endl; | Hello! |
| Вставка в конец строки | | | | |
| 1 | append(count, char) | Добавление count символов | MyString a1;  a1.clear();  a1.append(3,'!');  std::cout<<a1<<std::endl;  a1.append(3,'@');  std::cout<<a1<<std::endl; | !!!  !!!@@@ |
| 2 | append(“string”) | Добавление строки | MyString a2;  a2.clear();  a2.append("Hello ");  std::cout<<a2<<std::endl;  a2.append("world");  std::cout<<a2<<std::endl; | Hello  Hello world |
| 3 | append(“string”, index, count) | Добавление count символов строки начиная с заданного индекса | MyString a3;  a3.clear();  a3.append("Hello world",0,6);  std::cout<<a3<<std::endl;  a3.append("Hello world",6,5);  std::cout<<a3<<std::endl; | Hello  Hello world |
| 4 | append(std::string) | Добавление строки std:: string | MyString a4;  std::string s1="Hello ", s2="world";  a4.clear();  a4.append(s1);  std::cout<<a4<<std::endl;  a4.append(s2);  std::cout<<a4<<std::endl; | Hello  Hello world |
| 5 | append(std::string, index, count) | Добавление count символов строки std:: string начиная с заданного индекса | MyString a5;  std::string s3="Hello world";  a5.clear();  a5.append(s3,0,6);  std::cout<<a5<<std::endl;  a5.append(s3,6,5);  std::cout<<a5<<std::endl; | Hello  Hello world |
| Замена подстроки | | | | |
| 1 | replace(index, count, “string”) | Замена count символов по заданному индексу на строку | MyString a1="hello amazing world";  a1.replace(6,7,"wonderful");  std::cout<<a1<<std::endl; | hello wonderful world |
| 2 | replace(index, count, std::string) | Замена count символов по заданному индексу на строку std::string | MyString a2="hello amazing world";  std::string s1="wonderful";  a2.replace(6,7,s1);  std::cout<<a2<<std::endl; | hello wonderful world |
| Извлечение подстроки | | | | |
| 1 | substr(index) | Метод возвращает подстроку с началом по заданному индексу | MyString a1 = "hello amazing world", a2;  a2=a1.substr(6);  std::cout<<a2<<std::endl; | amazing world |
| 1 | substr(index,count) | Метод возвращает подстроку длины count с началом по заданному индексу | MyString a3 = "hello amazing world", a4;  a4=a3.substr(6,7);  std::cout<<a4<<std::endl; | amazing |
| Поиск подстроки | | | | |
| 1 | find(“string”) | Метод возвращает индекс первого вхождения строки | MyString a="hello amazing world amazing";  int i;  i=a.find("amazing");  std::cout<<i<<std::endl; | 6 |
| 2 | find(“string”, index) | Аналогичен find(“string”), но выполняет поиск начиная с заданного индекса | MyString a="hello amazing world amazing";  int i;  i=a.find("amazing", 7);  std::cout<<i<<std::endl; | 20 |
| 3 | find(std::string) | Аналогичен find(“string”) | MyString a="hello amazing world amazing";  int i;  std::string s="amazing";  i=a.find(s);  std::cout<<i<<std::endl; | 6 |
| 4 | find(std::string, index) | Аналогичен find(“string”, index) | MyString a="hello amazing world amazing";  int i;  std::string s="amazing";  i=a.find(s,7);  std::cout<<i<<std::endl; | 20 |

# ЭТАПЫ выполнения

Процесс выполнения лабораторной работы включает в себя следующие этапы:

1. Первый этап(*занятие №1*):
   * выполнение алгоритмической декомпозиции класса MyString для определения функций, на базе которых могут быть реализованы остальные;
   * реализация базовых функций;
   * тестирование написанных функций.
2. Второй этап (*занятие №2*):
   * реализация оставшихся методов MyString;
   * тестирование написанных функций.
3. Третий этап (*занятие №3*):
   * рассмотрение существующих способов создания Python-обертки C++ класса;
   * реализация Python-обертки для C++ класса MyString;
   * тестирование Python-обертки.

# Дополнительные задания

## Дополнительный функционал класса Mystring

Цель дополнительного задания – получение опыта работы с исключениями C++, move-семантикой.

В рамках выполнения дополнительного задания необходимо:

1. Добавить поддержку базовых std::exceptions и реализовать свой собственный тип исключения для недопустимого преобразования. Функции substr(), replace(), erase(), insert() должны генерировать исключение out\_of\_range.
2. Реализовать функции, которые описаны в таблице 2.

Таблица 2 – Дополнительные функции класса MyString

| **№** | **Название функции** | **Описание** |
| --- | --- | --- |
| 1 | Constructor(&&) | Конструктор перемещения (move-семантика) |
| 2 | Constructor(0x1234568) | Конструктор, реализующий конвертацию числа в строку |
| 3 | Constructor(0.05) | Конструктор, реализующий конвертацию float-числа в строку |
| 4 | Operator=&& | Оператор присвоения, реализующий move-семантику |
| 5 | Operator<<(std::basic\_ofstream)  Operator>>(std::basic\_ifstream) | Операторы ввода / вывода в файл |
| 6 | find() | Метод поиска подстроки, реализующий алгоритм Ахо-Корасика |
| 7 | at(index) | Метод возвращает элемент по индексу при его наличии, иначе генерирует исключение |
| 8 | to\_int() | Метод выполняет конвертацию строки в число |
| 9 | to\_float() | Метод выполняет конвертацию строки в float-число |

В отчет по лабораторной работе необходимо привести:

1. Сравнение move и copy-семантики.

2. Примеры обработки исключений.

## Шаблон «Итератор»

Цель дополнительного задания – изучение поведенческого паттерна «Итератор».

В рамках выполнения дополнительного задания необходимо:

1. Реализовать функции, приведенные в таблице 3, для четырех типов итераторов:

* iterator – итератор (forward), который выполняет обход строки от начала до конца и может быть использован для её изменения;
* const\_iterator – константный итератор, который выполняет обход строки от её начала до конца;
* reverse\_iterator – реверсивный итератор (backward), который выполняет обход строки от конца до начала и может быть использован для её изменения;
* const\_reverse\_iterator – аналогичен reverse\_iterator, но не может быть использован для модификации строки.

1. Для всех функций класса MyString, принимающих в качестве аргумента индекс, необходимо реализовать версию с итератором: например, insert(index, “string”) -> insert(iterator, “string”).

Таблица 3 – Дополнительные функции класса MyString

| **№** | **Название функции** | **Описание** |
| --- | --- | --- |
| 1 | iterator begin() | Возвращает iterator, указывающий на первый символ строки |
| 2 | iterator end() | Возвращает iterator, указывающий на символ, *который следует за последним символом строки* |
| 3 | const\_iterator cbegin() | Возвращает const\_iterator, указывающий на первый символ строки |
| 4 | const\_iterator cend() | Возвращает const\_iterator, указывающий на символ, *который следует за последним символом строки* |
| 5 | reverse\_iterator rbegin() | Возвращает reverse\_iterator, указывающий на первый символ строки |
| 6 | reverse\_iterator rend() | Возвращает reverse\_iterator, указывающий на символ, *который следует за последним символом строки* |
| 7 | const\_reverse\_iterator rcbegin() | Возвращает const\_reverse\_iterator, указывающий на первый символ строки |
| 8 | const\_reverse\_iterator rcend() | Возвращает const\_reverse\_iterator, указывающий на символ, *который следует за последним символом строки* |

В отчет по лабораторной работе необходимо привести:

1. Сравнение итератора и индекса.
2. Результаты тестирования итератора.

## Модификация private-полей класса при помощи отладчика

Цель дополнительного задания:

1. Изучение представления перегружаемых методов в скомпилированном кода (name mangling).

2. Проверка возможности модификации private-полей класса при помощи отладчика.

В рамках выполнения дополнительного задания необходимо:

1. Реализовать динамически подключаемую библиотеку (DLL), которая экспортирует методы класса MyString, реализованного в рамках основной части лабораторной работы №1.

2. Собрать DLL в конфигурации проекта Release.

3. При помощи IDA Pro получить таблицу Exports собранной DLL и сравнить имена перегруженных методов (например, append с разными аргументами), приведенные в их реализации (рисунок 1).

4. Реализовать программу, которая импортирует необходимые методы из реализованной DLL и использует в качестве функции main, исходный код из приложения 1.

5. Собрать исполняемый файл в конфигурации проекта Release.

6. При помощи IDA Pro выполнить отладку собранного исполняемого файла и модифицировать строку «A picture is worth a thousand words.», хранящуюся в *объекте* класса MyString.

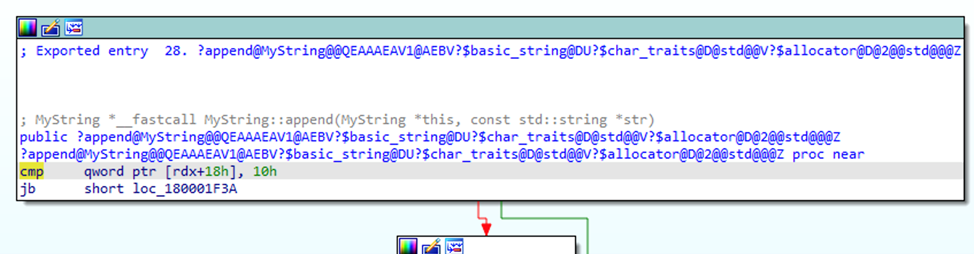


Рисунок 1 – Фрагмент метода append(MyString \*this, const std::string \*str) класса MyString

В отчет по лабораторной работе необходимо привести:

1. Описание процесса создания DLL.

2. Рисунок с фрагментом таблица экспорта DLL.

3. Краткое описание необходимости применения механизма кодирования имен перегружаемых методов и их примеры в виде рисунков.

4. Рисунок, демонстрирующий память процесса с полями объекта класса MyString до и после модификации, а также рисунок с выводом в консоль.

# Требования к отчету

Отчет должен включать следующие пункты:

1. Цель работы.
2. Задачи с указанием выполненных дополнительных заданий.
3. Ход работы – краткое описание этапов выполнения работы:
   1. краткое описание класса MyString (выбранный формат хранения строки);
   2. результирующий граф алгоритмической декомпозиции;
   3. результаты тестирования реализованных базовых методов
   4. примеры тестирования остальных методов;
   5. описание реализации Python-обертки для C++ класса MyString;
   6. примеры тестирование Python-обертки.
4. Выводы

**Приложение** 1

Исходный код программы, необходимый для выполнения дополнительного задания «Модификация private-полей класса при помощи отладчика».

#include "mystring.h"

\_\_declspec(noinline) void modify(MyString& str) {

if (str.length() == 0) {

str = "A picture is worth a thousand words.";

}

}

int main() {

MyString str;

modify(str);

std::cout << "Current string: " << str << "\n" << "Size: " << str.length() << "\n";

return 0;

}