

Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого

—
Институт компьютерных наук и кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА №1

«Класс MyString»

по дисциплине «Объектно-ориентированное программирование»

Санкт-Петербург

1 ЦЕЛЬ РАБОТЫ

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

2 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.

4 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) за исключением функций работы с Си-строками.
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. Для информирования о недопустимых событиях и значениях аргументов необходимо использовать классы исключений из стандартной библиотеки Си++ (`std::out_of_range` и т.д.).
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.

Для прохождения автоматического тестирования лабораторной необходимо соблюдать следующие условия:

1. Проект должен поддерживать сборку компилятором `clang++`.
2. В корне проекта необходимо разместить текстовый файл «`mystring_h_path.txt`», который содержит относительный путь к заголовочному файлу класса «MyString» в директории проекта (например «`MyString.h`»).
3. Файл с реализацией Python-обертки должен иметь название «`MyString_wrapper.cpp`» и находиться в корне проекта.
4. Проект должен содержать «`.gitignore`» файл, исключающий включение в Git-репозиторий артефактов сборки C++ проекта (объектных и исполняемых файлов).

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

Таблица 1 – Функции, которые необходимо реализовать в рамках класса «MyString» [1]

№	Название функции	Описание	Пример использования	Вывод
Конструкторы и деструктор				
1	MyString()	Конструктор по умолчанию	MyString str; pstr(str);	"" (0, 0)
2-4	MyString(SOURCE_STR)	Конструктор копирования из исходной строки	MyString str("Hello world!"); pstr(str);	"Hello world!" (12, 13)
			std::string s_str = "hello"; MyString str(s_str); pstr(str);	"hello" (5, 6)
			MyString s_str("hello"); MyString str(s_str); pstr(str);	"hello" (5, 6)
5-7	MyString(SOURCE_STR, int count)	Конструктор, инициализирующий класс заданным числом символов из исходной строки	MyString str("hello",4); pstr(str);	"hell" (4, 5)
			MyString str(std::string("hello"),4); pstr(str);	"hell" (4, 5)
			MyString str(MyString("hello"),4); pstr(str);	"hell" (4, 5)
8	MyString(int count, char ch)	Конструктор, инициализирующий класс строкой из заданного числа указанного символа	MyString str(5, '!'); pstr(str);	"!!!!!" (5, 6)
9	~MyString()	Деструктор		
Метод очищения содержимого строки				
10	clear()	Удаление элементов char из строки	MyString str("Hello world!"); pstr(str); str.clear(); pstr(str);	"Hello world!" (12, 13) "" (0, 13)
Метод высвобождения избыточной памяти				
11	shrink_to_fit()	Уменьшает объем (capacity) до длины строки (size)	MyString str("Hello world!"); pstr(str); str.erase(5, 6); pstr(str); str.shrink_to_fit(); pstr(str);	"Hello world!" (12, 13) "Hello!" (6, 13) "Hello!" (6, 7)

№	Название функции	Описание	Пример использования	Вывод
Оператор присваивания				
12	operator=(SOURCE_STR) ;	Оператор присваивания строки	MyString str; str="hello"; pstr(str);	"hello" (5, 6)
13	operator=(char ch)	Оператор присваивания символа	MyString str; str='!'; pstr(str);	"!" (1, 2)
Методы получения свойств объекта (getters)				
14	c_str()	Метод возвращает указатель на массив данных, который обязательно должны завершаться нулем	MyString str("str"); std::cout<<str<<std::endl;	str
15	size()	Метод возвращает число элементов типа char в строке	MyString str("Hello world!"); std::cout<<str.size()<<std::endl;	12
16	capacity()	Метод возвращает объем (capacity) выделенной для хранения строки памяти	MyString str("Hello world!"); std::cout<<str.capacity()<<std::endl;	13
17	empty()	Метод возвращает true если строка пустая	MyString str("Hello world!"); std::cout<<str.empty()<<std::endl; MyString empty; std::cout<<empty.empty()<<std::endl;	0 1
Методы вставки по индексу				
18	insert(int index, int count, char ch)	Вставка count символов ch по индексу	MyString str("aaaaa"); str.insert(0,1,'!'); pstr(str); str.insert(3, 2, '@'); pstr(str);	"!aaaaa" (6, 7) "!aa@aaaa" (8, 9)
19-21	insert(int index, SOURCE_STR)	Вставка строки по индексу	MyString str("aaaaa"); str.insert(1,"@@@@"); pstr(str);	"a@@@@aaaa" (10, 11)
22-24	insert(int index, SOURCE_STR , int count)	Вставка count символов строки по индексу	MyString str("aaaaa"); str.insert(1,"@@@@",2); pstr(str);	"a@@@@aaaa" (7, 8)

№	Название функции	Описание	Пример использования	Вывод
25-27	insert(int index, SOURCE_STR , int s_index, int count)	Вставка count символов исходной строки начиная с индекса s_index по индексу index	MyString str("aaaaa"); str.insert(1, "abcde", 1, 2); pstr(str);	"abcaaaa" (7, 8)
Методы вставка в конец строки				
28	append(int count, char ch)	Добавление count символов ch	MyString str; str.append(3, '!'); pstr(str); str.append(3, '@'); pstr(str);	"!!!" (3, 4) "!!!@@" (6, 7)
29-31	append(SOURCE_STR);	Добавление строки	MyString str; str.append("Hello "); pstr(str); str.append("world"); pstr(str);	"Hello " (6, 7) "Hello world" (11, 12)
32-34	append(SOURCE_STR , int count)	Добавление count символов строки	MyString str; str.append("Hello world", 6); pstr(str); str.append("world"); pstr(str);	"Hello " (6, 7) "Hello world" (11, 12)
35-37	append(SOURCE_STR , int s_index, int count)	Добавление count символов строки начиная с заданного индекса	MyString str; str.append("Hello world", 0, 6); pstr(str); str.append("Hello world", 6, 5); pstr(str);	"Hello " (6, 7) "Hello world" (11, 12)
Метод удаления подстроки				
38	erase(int index, int count)	Удаление count символов по индексу	MyString str("Hello world!"); str.erase(5, 6); pstr(str);	"Hello!" (6, 13)
Методы замены подстроки				
39-41	replace(int index, int count, SOURCE_STR)	Замена count символов по заданному индексу на строку	MyString str("hello amazing world"); str.replace(6, 7, "wonderful"); pstr(str);	"hello wonderful world" (21, 22)
42-44	replace(int index, int count, SOURCE_STR , int s_count)	Замена count символов по заданному индексу на s_count символов строки	MyString str("hello amazing world"); str.replace(6, 7, "wonderful", 6); pstr(str);	"hello wonder world" (18, 20)

№	Название функции	Описание	Пример использования	Вывод
45-47	replace(int index, int count, SOURCE_STR , int s_index, int count)	Замена count символов по заданному индексу на s_count символов строки, начиная с индекса s_index	MyString str("hello amazing world"); str.replace(6,7,"wonderful", 1, 2); pstr(str);	"hello on world" (14, 20)
Методы извлечения подстроки				
48	substr(int index)	Метод возвращает подстроку с началом по заданному индексу	MyString str("hello amazing world"), substr; substr=str.substr(6); pstr(substr);	"amazing world" (13, 14)
49	substr(int index, int count)	Метод возвращает подстроку длины count с началом по заданному индексу	MyString str("hello amazing world"), substr; substr=str.substr(6,7); pstr(substr);	"amazing" (7, 8)
Операторы сложения (конкатенации) и расширения строк				
50-52	operator+ (SOURCE_STR)	Оператор объединения текущего объекта класса с заданной строкой	MyString left("hel"), right("lo"), r; r=left+right; pstr(left); pstr(right); pstr(r);	"hel" (3, 4) "lo" (2, 3) "hello" (5, 6)
53-55	operator+= (SOURCE_STR)	Оператор добавления заданной строки к текущему объекту класса	MyString str("hel"), toadd("lo"); str+=toadd; pstr(str); pstr(toadd);	"hello" (5, 6) "lo" (2, 3)
Оператор индексации				
56	opearator[] (int index)	Оператор взятия значение по индексу	MyString str("hello"); std::cout<<str[2]<<std::endl; str[2]='L'; pstr(str);	l "heLlo" (5, 6)
Метод и операторы сравнения строк				
57	compare(const MyString&)	Лексикографическое сравнение строк (возвращает -1, 0, 1)	MyString a("abcd"),b("abce"); std::cout<<a.compare(b) <<b.compare(a) <<std::endl;	Сравнить с реализацией операторов сравнения класса std::string
58-60	operator>()	Лексикографическое сравнение строк (возвращает bool)	MyString a("abcd"),b("abce"); std::cout<<(a==b)<<(a!=b)<<(a>b)<<(a>=b)<<(a<b)<<(a<=b)<< std::endl;	
	operator<()			
	operator>=()			

№	Название функции	Описание	Пример использования	Вывод
	operator<=()			
	operator!=()			
	operator==()			
Методы поиска подстроки				
61-63	find(SOURCE_STR)	Метод возвращает индекс первого вхождения строки	MyString str="hello amazing world amazing"; int i=str.find("amazing"); std::cout<<i<<std::endl;	6
64-66	find(SOURCE_STR , int index)	Аналогичен find("string"), но выполняет поиск начиная с заданного индекса	MyString str="hello amazing world amazing"; int i=a.find("amazing", 7); std::cout<<i<<std::endl;	20

SOURCE_STR – необходимо поддерживать входные строки трех типов: `const char*`, `std::string&`, `MyString&`.

5 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;
- построение графа зависимостей методов класса «MyString» по вызовам (рекомендуется использовать инструмент `llvm` средство `dot_unmangler.py`).

3. Третий этап (*занятие №3*):

- рассмотрение существующих способов создания Python-обертки C++ класса;
- реализация Python-обертки для C++ класса «MyString»;
- тестирование Python-обертки.

6 ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ

6.1 Дополнительный функционал класса `MyString`

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

В рамках выполнения дополнительного задания необходимо:

1. Реализовать функции, которые описаны в таблице 2.
2. Реализовать свой собственный тип исключения для недопустимого преобразования.

Таблица 2 – Дополнительные функции класса «`MyString`»

№	Название функции	Описание
1	<code>MyString(MyString&&)</code>	Конструктор перемещения (move-семантика)
2	<code>MyString(0x1234568)</code>	Конструктор, реализующий конвертацию числа в строку
3	<code>MyString(0.05)</code>	Конструктор, реализующий конвертацию float-числа в строку
4	<code>operator=</code>	Оператор присвоения, реализующий move-семантику
5-6	<code>operator<<(std::basic_ofstream)</code> <code>operator>>(std::basic_ifstream)</code>	Операторы ввода / вывода в файл
7	<code>find()</code>	Метод поиска подстроки, реализующий алгоритм Ахо-Корасика
8	<code>at(index)</code>	Метод возвращает элемент по индексу при его наличии, иначе генерирует исключение
9	<code>to_int()</code>	Метод выполняет конвертацию строки в число
10	<code>to_float()</code>	Метод выполняет конвертацию строки в float-число

В отчет по лабораторной работе необходимо привести:

1. Сравнение move и сору-семантики.
2. Примеры обработки исключений.

6.2 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.

2. 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	<code>iterator begin()</code>	Возвращает <code>iterator</code> , указывающий на первый символ строки
2	<code>iterator end()</code>	Возвращает <code>iterator</code> , указывающий на символ, <i>который следует за последним символом строки</i>
3	<code>const_iterator cbegin()</code>	Возвращает <code>const_iterator</code> , указывающий на первый символ строки
4	<code>const_iterator cend()</code>	Возвращает <code>const_iterator</code> , указывающий на символ, <i>который следует за последним символом строки</i>
5	<code>reverse_iterator rbegin()</code>	Возвращает <code>reverse_iterator</code> , указывающий на первый символ строки
6	<code>reverse_iterator rend()</code>	Возвращает <code>reverse_iterator</code> , указывающий на символ, <i>который следует за последним символом строки</i>

№	Название функции	Описание
7	<code>const_reverse_iterator rcbegin()</code>	Возвращает <code>const_reverse_iterator</code> , указывающий на первый символ строки
8	<code>const_reverse_iterator rcend()</code>	Возвращает <code>const_reverse_iterator</code> , указывающий на символ, <i>который следует за последним символом строки</i>

В отчет по лабораторной работе необходимо привести:

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

6.3 Модификация `private`-полей класса при помощи отладчика

Цель дополнительного задания:

1. Изучение представления перегружаемых методов в скомпилированном коде (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».

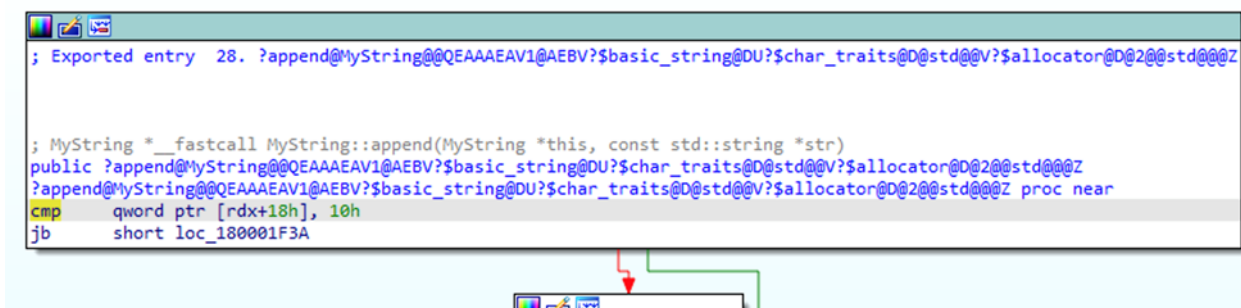


Рисунок 1 – Фрагмент метода append(MyString *this, const std::string *str) класса «MyString»

В отчет по лабораторной работе необходимо привести:

1. Описание процесса создания DLL.
2. Рисунок с фрагментом таблицы 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.

6.4 Реализация фаззинг-тестирования функционала класса MyString

Цель дополнительного задания:

1. Изучение подхода к тестированию программного обеспечения методом генеративного фаззинг-тестирования.

В рамках выполнения дополнительного задания необходимо:

1. Изучить инструмент генеративного фаззинг-тестирования программного обеспечения – AFL++ [2].
2. Разработать вспомогательную программу (harness) для реализации фаззинг-тестирования класса «MyString». Разработанная harness-программа

должна обеспечивать тестирования полного перечня методов класса из основной части задания лабораторной работы.

3. Разработать корпус тестовых данных для проведения фаззинг-тестирования.

4. Провести фаззинг-тестирование. В результате необходимо обеспечить 100% покрытия кода класса «MyString».

5. Реализовать вспомогательную программу для визуализации результатов покрытия кода класса «MyString» сгенерированными инструментом AFL++ fuzz-тестами (например, можно использовать функционал оценки покрытия gcov компилятора GCC [3]).

В отчет по лабораторной работе необходимо привести:

1. Описание процесса подготовки к фаззинг-тестированию (разработка harness-программы и корпуса тестовых данных).

2. Рисунок с результатами исчерпывающего фаззинг-тестирования (статус фаззинга AFL++).

3. Описание процесса визуализации результатов фаззинг-тестирования. Привести примеры снимков экрана с визуализацией покрытия основных фрагментов кода класса «MyString».

7 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:
 - a. краткое описание класса «MyString» (выбранный формат хранения строки);
 - b. результирующий граф алгоритмической декомпозиции;
 - c. результаты тестирования реализованных базовых методов;
 - d. примеры тестирования остальных методов;
 - e. описание реализации Python-обертки для C++ класса «MyString»;
 - f. примеры тестирования Python-обертки.
4. Ход работы по выполнению дополнительных заданий (в случае наличия).
5. Выводы.

8 РЕКОМЕНДУЕМЫЕ ИСТОЧНИКИ

1. Описание сигнатур методов класса `basic_string` стандартной библиотеки Си++. URL: https://en.cppreference.com/w/cpp/string/basic_string.html.
2. Выполнение фаззинг-тестирования с помощью инструмента AFL++. URL: https://aflplus.plus/docs/fuzzing_in_depth/.
3. Анализ покрытия кода с помощью функционала GCOV компилятора GCC. URL: <https://appsec.guide/docs/fuzzing/c-cpp/techniques/coverage-analysis/>.

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;
}
```