Project 7 Documentation

Project 7 requires students to create their own String class and utilize dynamic memory to allocate/deallocate memory space to store strings. The project also required that students implement numerous functions to read, write, and manipulate these objects. The sample main source file provided will be analyzed with the corresponding output below:

**The output🡪**

Testing Default ctor

Testing Parametrized ctor

Testing Copy ctor

The copy-ctor holds: Allocated Size: 35 String: MyString parametrized constructor!

Testing dtor

Testing size()

21

Testing length()

20

Testing c\_str()

C-String equivalent successfully obtained!

Testing operator==()

Same success

Different success

Testing operator=()

Allocated Size: 18 String: Before assignment

Allocated Size: 28 String: After performing assignment

Testing operator+

Allocated Size: 30 String: The first part and the second

Testing operator[]()

Testing operator<<()

Allocated Size: 24 String: Access successful

**The source file🡪**

//This is the main driver file for project 7, regarding building a MyString class.

#include <iostream>

#include <String/MyString.h>

using namespace std;

int main(){

//(1)

std::cout << "Testing Default ctor" << std::endl;

MyString ms\_default;

//(2)

std::cout << "Testing Parametrized ctor" << std::endl;

MyString ms\_parametrized("MyString parametrized constructor!");

//(3)

std::cout << "Testing Copy ctor" << std::endl;

MyString ms\_copy(ms\_parametrized);

std::cout << "The copy-ctor holds: "<<ms\_copy;

//(4)

std::cout << "Testing dtor" << std::endl;

{

MyString ms\_destroy("MyString to be destroyed...");

}//(5),(6)

MyString ms\_size\_length("Size and length test");

std::cout << "Testing size()" << std::endl;

cout << ms\_size\_length.size() << endl;

std::cout << "Testing length()" << std::endl;

cout << ms\_size\_length.length() << endl;

//(7)

std::cout << "Testing c\_str()" << std::endl;

MyString ms\_toCstring("C-String equivalent successfully obtained!");

cout << ms\_toCstring.c\_str() << endl;

//(8)

std::cout << "Testing operator==()" << std::endl;

MyString ms\_same1("The same"), ms\_same2("The same");

if (ms\_same1==ms\_same2)

cout << "Same success" << endl;

MyString ms\_different("The same (NOT)");

if (!(ms\_same1==ms\_different))

cout << "Different success" << endl;

//(9)

std::cout << "Testing operator=()" << std::endl;

MyString ms\_assign("Before assignment");

std::cout<<ms\_assign;

ms\_assign = MyString("After performing assignment");

std::cout<< ms\_assign;

//(10)

std::cout << "Testing operator+" << std::endl;

MyString ms\_append1("The first part");

MyString ms\_append2(" and the second");

MyString ms\_concat = ms\_append1+ ms\_append2;

std::cout << ms\_concat;

//(11)

std::cout << "Testing operator[]()" << std::endl;

MyString ms\_access("Access successful (NOT)");

ms\_access[17] = 0;

//(12)

std::cout << "Testing operator<<()" << std::endl;

cout << ms\_access << endl;

return 0;

}

**Analysis🡪**

1. This test simply calls default constructor for a MyString object.
2. This test calls the parametrized constructor with string input “MyString parametrized constructor!”.
3. This test copies the above parametrized constructor, then outputs a message stating the length of the above string (34) , then +1 because the null terminator.
4. This test calls the destructor on a test string ms\_destroy.
5. And
6. Use functions size() and length() to output the size of the container (20) +1 or 21 because length of the string itself is (20) then add one for size, because null terminator.
7. This test calls c.str(), which outputs the information pointed to by m\_buffer of the object ms\_toCstring.
8. This test checks the == overload, by first checking if objects ms\_same1 and ms\_same2 both have the same string pointed to by their respective m\_buffer(s) to then output “Same success.” It also checks if it works when m\_buffer(s) are different, with string “The same” and “The same (NOT)” and prints “Different success” if the == overload returns false.
9. This test checks if = overload does in fact set the calling object (ms\_assign) equal to another object. “Before assignment” is printed by cout<<ms\_assign, then ms\_assign is set equal to an object with m\_buffer pointing to “After performing assignment”. m\_assign is cout again to then output that message, showing that = overload operator works.
10. This test checks + overload, which concats two strings pointed to by two object’s m\_buffer(s) then returns a new object by value into ms\_concat. It is cout and gives “The first part and the second”, which shows that this overload concats two objects correctly.
11. This test sets index 17 of ms\_access, which holds “Access successful (NOT)” ,to 0 using the [] overload, with the implications of this shown in test 12.
12. This test outputs using the <<overload, to show “Allocated Size: 24 String: Access successful”, meaning that test 11 worked because it prints out Access successful.

**\*Please note that I included extra cout statements throughout these tests, a bit different from the one provided in the project 7 pdf to show that all of my tests work correctly.**