

Programming Assignment 1 (100 points)

i. Purpose & Program Description

The purpose of the program was to introduce the use of OOP concepts as well as practice using various features of C++ by designing and implementing a string class container using a dynamic character array in order to construct, store, allocate/deallocate, modify, append, input/output strings and/or characters, and providing a resizing functionality for the array. The `my_string` program includes several method operations including accessors, constructors, copy constructors, copy assignment, as well as operators used to concatenate and insert strings and characters to existing strings and to access specific indices of each string with bounds checking in place. In this programming assignment, a better understanding was developed as to how classes and OOP concepts are implemented and utilized by writing the `my_string` class.

ii. Data Structures

The data structure used in this program was an array, which is a collection or sequence of elements identified by their corresponding indices. In this program, an array of characters is constructed, allocated, and pointed to on the heap as either an empty array of characters, an array with `n`-capacity allocated bytes, an array of characters constructed from user-input c-string, or is constructed via the copy constructor, which copies over the contents of a `my_string` class object to the newly allocated array. The array data structure in this program provides several method operations for constructing, allocating, appending, and manipulating strings:

- **size()**: returns the size of the string
- **capacity()**: returns the length in bytes of the string's allocated memory
- **empty()**: returns true/false if the string is empty in size or not
- **operator[](i)**: returns the character at index `i` of the string, without performing arrays bounds checking
- **at(i)**: returns the character at index `i` of the string, with performing arrays bounds checking.
- **insert(i, s)**: inserts string `s` into our string at a specified index
- **operator+=(q)**: concatenates a specified string `q` to this string
- **operator+=(c)**: appends the character `c` to the string.
- **default constructor**: creates an empty string without any memory allocation.
- **constructor**: constructor for allocating memory of a string with `n`-capacity
- **C-string constructor**: constructs a string with contents taken from a c-string
- **copy constructor**: constructs a new string and copies the contents of a pre-existing `my_string` object
- **destructor**: deallocates memory and makes an empty string.
- **copy assignment**: copies the contents of a specified `my_string` object to the string
- **resize**: a private helper function that resizes the array by doubling the current capacity if it is less than the new size and copying over the contents of the pre-existing `my_string` object
- **print()**: a public function that allows the user to print the contents of a string as well as the string's size and capacity

The implementation of these class methods are straight-forward, note that the private helper function, `resize()`, was included in order to `resize (double)` the array's capacity while the new size (`current string size + appended char/string size`) is greater than the current array capacity. The `resize` function also handles the "zero-capacity" case which is included in order for empty strings to be appended to by incrementing the capacity by 5 while the capacity is less than the size.

iii. Instructions

The program can be compiled via the command line:

```
g++ -std=c++11 -o my_string *.cpp
```

The program can then be executed by the command line:

```
./my_string
```

About half way down upon executing the program it will ask to enter a string, you can input your string here and the input operator will parse the string, only processing the user input string up to the first space and/or newline.

iv. Logical Exceptions & Bugs

In `my_string.cpp`, the program will provide appropriate error messages if the `my_string` object's size is greater than the capacity and will throw an `out_of_range` exception if an invalid index is attempted to be accessed using the `at(i)` function. In the `n-capacity` constructor, which initializes an empty character array with an allocation of `n-capacity` bytes, an exception is handled in which the input `n < 0`, a negative number of bytes cannot be allocated. Also worth noting, in the private helper function, `resize()`, the zero-case for a `my_string` object of 0-capacity is handled for appending characters/strings to an empty string. This case was added in order to prevent logical errors since the `resize` function doubles the current capacity, but if the capacity is 0, doubling 0 gives 0; therefore, I added a case to just increment the capacity by 5 while the capacity is less than the new size in order to provide empty-string appending functionality if this was ever a possible case. In the `insert` function, there is input checking for the `(int i)` argument of the function in order to validate it as a correct index in the string, such index `(i)` should be greater than or equal to zero and less than or equal to the size of the string, so that strings or single characters can be inserted to the boundaries of the string at the beginning, end, or anywhere in-between. As for bugs, I was unable to find any obvious bugs with the `my_string` program. I tried to test as many aspects of each function of the `my_string` program as I could and stress-test those using erroneous inputs, as well as testing them using the instructor's test cases provided on the instructor's website.

v. C++ object oriented features

The `my_string` program incorporates the most fundamental of OOP features, *the class*. The `my_string` class encapsulates both the public and private data access members and methods within their respected objects alongside a combination of a data structure and the algorithms used in order to provide several operations for constructing, allocating, appending, inserting, and manipulating strings for the user. OOP based features such as encapsulation focus on the components of the program such as the string in order to achieve ease of use for the user. The element of encapsulation is important in OOP because it provides a way to hide information and also helps safeguard faulty procedures such as preventing the user from accessing and manipulating private data. Information hiding, for instance, prevents the user from manually changing the capacity or size of a string; which is not ideal information for the user to be able to access because it could lead to errors if the capacity or size values are changed to values that would cause runtime errors.

vi. Testing Results

Although the main test-cases used for the program are provided in the main.cpp program taken from the instructor's website, I had constructed a few alternate cases prior to testing the my_string program with the instructor's test-cases in order to further test the functionality and reliability of the program. From what I had tested, all of the functions prove to function and are reliable, even with erroneous or invalid input values, such as negative byte memory allocation, accessing out of range indices in the array, etc., which the exceptions handled in all the cases I could think of. That being said, I had tested all of the constructors, specifically for the c-string and copy constructor and I had also tested the string and character concatenation functions using both += operators and the insert function. Throughout the test cases, and after every step/change to the strings, I had printed out each string data regarding size and capacity so I could monitor the underlying processes of the my_string class such as the use of the resize() function to double capacity and the copy assignment function copying over array elements from one array to another. Overall, the testing proved the reliability and functionality of the my_string program.

Below is screenshots of the alternate testing cases I made to further test the my_string program's functionality.

```
/* ----- ALTERNATE TESTING CASES ----- */

// my_string(-1); // should trigger an out_of_range exception
char c = '!';
my_string v1;
cout << "Constructing an empty string:" << endl;
v1.print();

my_string s1(0); // n-byte allocation constructor
cout << endl;

my_string s2("Whata"); // c-string constructor
cout << "Constructing a string that reads \"Whata\": " << endl;
s2.print();
// cout << s2.at(5) << endl; // should trigger an out_of_range exception
cout << endl;

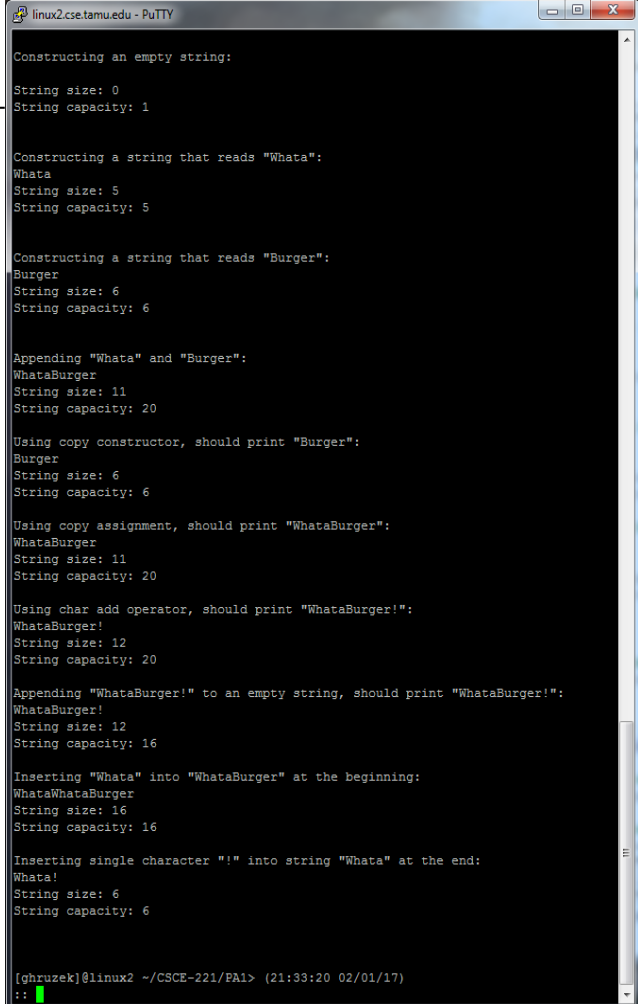
my_string s3("Burger"); // c-string constructor
cout << "Constructing a string that reads \"Burger\": " << endl;
s3.print();
cout << endl;

s2 += s3; // += operator for appending c-string
cout << "Appending \"Whata\" and \"Burger\": " << endl;
s2.print();

my_string s4(s3); // copy constructor
cout << "Using copy constructor, should print \"WhataBurger\": " << endl;
s4.print();
s4 = s2; // copy assignment
cout << "Using copy assignment, should print \"WhataBurger\": " << endl;
s4.print();
s4 += c; // += operator for appending char
cout << "Using char add operator, should print \"WhataBurger!\": " << endl;
s4.print(); // Should print "WhataBurger!"

cout << "Appending \"WhataBurger!\" to an empty string, should print \"WhataBurger!\": " << endl;
v1 += s4;
v1.print();

my_string s5("WhataBurger");
my_string s6("Whata");
my_string s7("!");
// s5.insert(-1, s6); // will throw an out_of_range exception
s6.insert(0, s6); // should produce "WhataWhataBurger"
s6.insert(5, s7); // should produce "Whata!"
// s5.insert(15, s6); // will throw an out_of_range exception
cout << "Inserting \"Whata\" into \"WhataBurger\" at the beginning: " << endl;
s5.print();
cout << "Inserting single character \"!\" into string \"Whata\" at the end: " << endl;
s6.print();
cout << endl;
// system("pause"); // for VS
```



```
linux2.csetamu.edu - PuTTY
Constructing an empty string:
String size: 0
String capacity: 1

Constructing a string that reads "Whata":
Whata
String size: 5
String capacity: 5

Constructing a string that reads "Burger":
Burger
String size: 6
String capacity: 6

Appending "Whata" and "Burger":
WhataBurger
String size: 11
String capacity: 20

Using copy constructor, should print "Burger":
Burger
String size: 6
String capacity: 6

Using copy assignment, should print "WhataBurger":
WhataBurger
String size: 11
String capacity: 20

Using char add operator, should print "WhataBurger!":
WhataBurger!
String size: 12
String capacity: 20

Appending "WhataBurger!" to an empty string, should print "WhataBurger!":
WhataBurger!
String size: 12
String capacity: 16

Inserting "Whata" into "WhataBurger" at the beginning:
WhataWhataBurger
String size: 16
String capacity: 16

Inserting single character "!" into string "Whata" at the end:
Whata!
String size: 6
String capacity: 6

[ghruzek@linux2 ~/CSCE-221/PA1> (21:33:20 02/01/17)
::
```