CIS 200 - Project 01

1. Problem Statement

Create an original version of the *string* class named *myString*. Each instance of *myString* will be able to handle 25 characters and each method will need to contain a reference “status variable”.

2. Requirements

2.1 Assumptions

- Default *string* class may be used for screen input
- Default *string* class may be used for *setString* method
- User will not input strings that include spaces

2.2 Specifications

- Functions to replicate
    - *size*()
    - *addStart*()
    - *addEnd*()
    - *partString*()
    - *replPartString*()
    - *replWholeString*()
    - *compareString*()
    - *initString*()
    - *setString*()
    - *getString*()
    - *printStringScreen*()
    - *numericString*()
    - *alphabeticString*()
- Program will test each function systematically
    - Some results will be shown to the user
    - Some results will be printed to a file
    - May allow for methods to be tested out of order/individually

3. Decomposition Diagram

- *myString* Class Program
  - Input
    - User inputs string under 25 characters
    - User inputs via command line
  - Process
    - Count the number of characters in a string
    - Iterate character array to append beginning
    - Iterate through array to append to end
    - Replace part of array from given position
    - Replace entire array of characters
    - Compare given array with saved array
    - Initialize/Wipe string to *null*
    - Insert *string* into *myString*
    - Determine if string is integer or real number
    - Determine if string is alphabetic exclusively
  - Output
    - Iterate and return portion of string from any start point
    - Return and print current array of characters

4. Test Strategy

- Valid Data
- Invalid Data

5. Test Plan Version 1

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	*size*()				
Valid Data	2	*addStart*()				
Valid Data	3	*addEnd*()				
Valid Data	4	*partString*()				
Valid Data	5	*replPartString*()				
Valid Data	6	*compareString*()				
Valid Data	7	*initString*()				

Valid Data	8	*setString*()				
Valid Data	9	*getString*()				
Valid Data	10	*printStringScreen*()				
Valid Data	11	*numericString*()				
Valid Data	12	*alphabeticString*()				
Invalid Data	1	*addStart*()				
Invalid Data	2	*addEnd*()				
Invalid Data	3	*partString*()				
Invalid Data	4	*replPartString*()				
Invalid Data	5	*replWholeString*()				
Invalid Data	6	*compareString*()				
Invalid Data	7	*setString*()				
Invalid Data	8	*getString*()				
Invalid Data	9	*printStringScreen*()				

6. Initial Algorithm

1. Create Class *myString*
    - *a.* Create Function *size*()
        - i. Return number of characters in string
        - ii. Count all non-null characters and add to *total*
    - *b.* Create Function *addStart*(myString)
        - i. Add input to beginning of string
        - ii. Move current characters forward and insert
    - *c.* Create Function *addEnd*(myString)
        - i. Append string with input
        - ii. Insert characters to end up to 25
    - *d.* Create Function *partString*(startPos, length)
        - i. Return portion of string from inputted index
        - ii. Take in index integer and iterate from there
    - *e.* Create Function *replPartString*(myString, startPos)
        - i. Replace portion of string from inputted index
        - ii. Take in index of string and replace

f. Create Function *replWholeString*(myString)
    i. Replace entire string with user input
    ii. Wipe string and insert characters
g. Create Function *compareString*(myString)
    i. Compare current string to user input
    ii. Iterate both and check each index
h. Create Function *initString*()
    i. Wipe/Initialize string to null
    ii. Iterate and clear each index
i. Create Function *setString*(string)
    i. Set string to current input
    ii. Iterate and fill each index with corresponding index
j. Create Function *getString*()
    i. Return current string
    ii. Iterate and return each index into string
k. Create Function *printStringScreen*()
    i. Print current string to screen
    ii. Iterate and print each index
l. Create Function *numericString*()
    i. Determine if string is integer or real number
    ii. Iterate and check for numeric or real number characters in correct order
m. Create Function *alphabeticString*()
    i. Determine if string is all alphabetic
    ii. Iterate and check for alpha characters

2. Create Function *outputFile*()
    a. Will print results of each function in class *myString* to file
3. *MAIN* FUNCTION
    a. Test each function of class *myString*
    b. Print results of each function to file with *outputFile*().

7. Test Plan Version 2

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	*size*()	“string”	6		
Valid Data	2	*addStart*()	“my“	“mystring”		
Valid Data	3	*addEnd*()	“s”	“mystrings”		
Valid Data	4	*partString*()	0, 5	“mystr”		
Valid Data	5	*replPartString*()	3, “Test”	“myTest”		

Valid Data	6	*compareString*()	“mystring” “test”	FALSE		
Valid Data	7	*initString*()	null	null string		
Valid Data	8	*setString*()	“mystr” *string*	“mystr” *myString*		
Valid Data	9	*getString*()	“mystr” *myString*	“mystr” *string*		
Valid Data	10	*printStringScreen*()	“mystr”	“mystr” Print		
Valid Data	11	*numericString*()	“6.502”	TRUE		
Valid Data	12	*alphabeticString*()	“abc”	TRUE		
Invalid Data	1	*addStart*()	“abcd...z”	overflow		
Invalid Data	2	*addEnd*()	“zyxw...a”	overflow		
Invalid Data	3	*partString*()	“-1, 5”	Invalid Index		
Invalid Data	4	*replPartString*()	“my“, -1	Invalid Index		
Invalid Data	5	*replWholeString*()	“abc...z”	overflow		
Invalid Data	6	*compareString*()	“”	null		
Invalid Data	7	*setString*()	“”	null		
Invalid Data	8	*getString*()	“”	null - empty		
Invalid Data	9	*printStringScreen*()	“”	null - empty		

8. Code

Header File - myString.h

```
#pragma once
#ifndef MYSTRING_H
#define MYSTRING_H

#include <cctype>
#include <iostream>
#include <string> //Included to allow use of base string class to fill myString since
recreating that is way above my pay grade...

//ERROR CODE LOOKUP
// -1 = Attempted to fill string with more than 25 characters.
// -2 = Attempted to access index above 25
// -3 = Attempted to fill with whitespace
```

```
class myString
{
private:

	//Define Constant
	const int MAX_SIZE = 25;

	//Define Holder Variables
	int errorCode = 0;
	int length = 0;

	//Define Container for myString
	char charArray[25];

public:
	myString();

	//Set Functions
	void setString(std::string); //PIRMARY FUNCTION
	void setErrorCode(int);
	void setLength(int);

	//Get Functions
	std::string getString();
	int getErrorCode();
	int getLength();

	int size();

	void addStart(myString);
	void addEnd(myString);

	myString partString(int, int);
	myString replPartString(myString, int);
	myString replWholeString(myString);

	bool compareString(myString);

	void initString();
	void printStringScreen();

	bool numericString();
	bool alphabeticString();

	~myString();
};
```

```
#endif // !myString.h
```

Class File - myString.cpp

```
#include "myString.h"

//Default Constructor
myString::myString()
{
	charArray[MAX_SIZE];
	this->initString();
}

//PUBLIC METHODS
///SET COMMANDS

//Description: Sets character array in myString equal to string input
//Pre-Condition: string <= 25 character w/o whitespace
//Post-Condition: Filled character array in myString
void myString::setString(std::string input)
{
	if (input.size() > MAX_SIZE)
	{
		setErrorCode(-1);
	}
	else
	{
		//Update with valid length
		setLength(input.size());

		//Iterate and fill character array with holder string.
		for (int i = 0; i < input.size(); i++)
		{
			charArray[i] = input.at(i);
		}
	}
}

//Description: Access and update private variable errorCode. See list in Header File
//Pre-Condition: An error code needs updating
//Post-Condition: An error code gets updated
void myString::setErrorCode(int input)
{
	errorCode = input;
}

//Description: Access and update private variable length
```

```
//Pre-Condition: Need to keep track of length of character array/myString as it is
changed
//Post-Condition: length that accuratly matched character array in myString
void myString::setLength(int input)
{
	length = input;
}

///GET COMMANDS

//Description: Retrieve character array and present it in a way that can be shown to
the user or used in output log.
//Pre-Condition: NEED MYSTRING
//Post-Condition: YOU GOT MYSTRING
std::string myString::getString()
{
	return charArray;
}

//Description: Retrieve error code for use in output log
//Pre-Condition: Error code needed for output.
//Post-Condition: YOU GOT IT
int myString::getErrorCode()
{
	return errorCode;
}

//Description:  Retrieve private variable length
//Pre-Condition: need
//Post-Condition: got
int myString::getLength()
{
	return length;
}

//Description: Forward the current size of character array
//Pre-Condition: need size
//Post-Condition: YOU GOT IT BOSS
int myString::size()
{
	return getLength();
}

//Description: Appends secondary input to the beginning of original myString
//Pre-Condition: An original myString and an addition
//Post-Condition: addition at the beginning of myString
void myString::addStart(myString input)
{
```

```
	//Declare Holder Variables
	myString holderString;
	int tempLength = this->size();
	int tempLength2 = input.size();

	if ((input.size() + this->size()) > MAX_SIZE)
	{
		setErrorCode(-1);
	}
	else
	{
		//Fill Holder myString
		for (int i = 0; i < length; i++)
		{
			holderString.charArray[i] = this->charArray[i];
		}
		//Wipe myString to fill with new input
		this->initString();

		//Iterate and fill with user input
		for (int i = 0; i < input.size(); i++)
		{
			this->charArray[i] = input.charArray[i];
		}
		//Iterate further and fill with holder input
		for (int i = input.size(); i < (input.size() + tempLength); i++)
		{
			charArray[i] = holderString.charArray[i - input.size()];
		}
		this->setLength(tempLength + tempLength2);
	}
}

//Description: Appends secondary string to the end of a primary string
//Pre-Condition: An original myString an extra myString
//Post-Condition: Lo and behold they are mended together...
void myString::addEnd(myString input)
{
	//Declare Holder Variables
	int tempLength = this->size();
	int tempLength2 = input.size();

	if ((input.size() + this->size()) > MAX_SIZE)
	{
		setErrorCode(-1);
	}
	else
	{
```

```
		//Append character arrays
		int i = tempLength;
		int j = 0;
		do {
			this->charArray[i] = input.charArray[j];
			i++;
			j++;
		} while (i < (tempLength + tempLength2));
		//Update length
		this->setLength(tempLength + tempLength2);
	}
}

//Description: Returns portion of myString from given starting position and length
//Pre-Condition: A string, A spot, and A distance.
//Post-Condition: "He's going the distance"
myString myString::partString(int startPos, int length)
{
	//Declare Holder
	myString holderString;

	if (startPos > MAX_SIZE)
	{
		setErrorCode(-2);
	}
	else if ((startPos + length) > MAX_SIZE)
	{
		setErrorCode(-2);
	}
	else
	{
		int j = 0;
		//Fill Holder myString
		for (int i = (startPos - 1); i < (length + startPos - 1); i++)
		{
			holderString.charArray[j] = this->charArray[i];
			j++;
		}
		holderString.setLength(this->getLength());
	}

	return holderString;
}

//Description: Replaces portion of myString with secondary myString beginning at a
given position
//Pre-Condition: myString to insert, and a position.
//Post-Condition: lime in the coconut.
```

```
myString myString::replPartString(myString input, int startPos)
{
	//Declare Holders
	myString holderString;

	if (startPos > this->length)
	{
		setErrorCode(-3);
	}
	else if ((startPos + input.size()) > MAX_SIZE)
	{
		setErrorCode(-1);
	}
	else
	{
		//Fill Holder myString
		for (int i = 0; i < input.getLength(); i++)
		{
			holderString.charArray[i] = input.charArray[i];
		}
		int j = 0;
		for (int i = (startPos - 1); i < (length + startPos - 1); i++)
		{
			this->charArray[i] = holderString.charArray[j];
			j++;
		}
		this->setLength(startPos + input.size());
	}

	return myString();
}

//Description: Replaces current myString with entirely new myString
//Pre-Condition: bloodlust to kill current myString.
//Post-Condition: Zero evidence it ever happened outside of your mind.
myString myString::replWholeString(myString input)
{
	if (input.size() > MAX_SIZE)
	{
		setErrorCode(-1);
	}
	else
	{
		for (int i = 0; i < MAX_SIZE; i++)
		{
			this->charArray[i] = input.charArray[i];
		}
		this->setLength(input.size());
```

```
	}

	return myString();
}

//Description: Check to see if two given myStrings are identical.
//Pre-Condition: 2 myStrings
//Post-Condition: true or false
bool myString::compareString(myString input)
{
	int i = 0;
	do
	{
		if (this->charArray[i] != input.charArray[i])
		{
			return false;
		}
		i++;
	} while (i < MAX_SIZE);

	return true;
}

//Description: Initializes myString will null characters
//Pre-Condition: desire to erase
//Post-Condition: ... theres nothing here ...
void myString::initString()
{
	//Revert size to empty
	this->setLength(0);

	//Iterate and fill will null characters
	for (int i = 0; i < MAX_SIZE; i++)
	{
		charArray[i] = '\0';
	}
}

//Description: Does what it says
//Pre-Condition: Does it really matter?
//Post-Condition: Does anything really matter?
void myString::printStringScreen()
{
	for (int i = 0; i < size(); i++)
	{
		std::cout << charArray[i];
	}
}
```

```
//Description: Casts character array into double.
//Pre-Condition: If character array contains any non-real int chars, it evaluates to
zero
//Post-Condition: this is a flawed approach as it means you cant perform this
function on a real zero. Didn't have time to build a regular expression from scratch.
bool myString::numericString()
{
	//Cast character array into double.
	double holder = atof(charArray);

	//This solution works for every possible input except zero
	if (holder == 0) //Input was not a real number, or zero
	{
		return false;
	}
	else //Input was a real number
	{
		return true;
	}
}

//Description: Iterate and see if char array contains any non alpha chars
//Pre-Condition: a char array
//Post-Condition: a yes or no answer
bool myString::alphabeticString()
{
	int i = 0;
	while (this->charArray[i] != '\0')
	{
		if (!(isalpha(this->charArray[i])))
		{
			return false;
		}
		i++;
	}
	return true;
}

//Default Detructor
myString::~myString()
{
}
```

Source File - source.cpp

```
//Program Name: myString
//Programmer Name: Arthur Aigeltinger IV
```

```
//Description: A custom built version of the built in C++ "string" library.
//Date Created: 10/07/18

#include "myString.h"
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>

//Function Prototypes
void menu();
void printErrorCode(int);
void printLog(std::ofstream&, std::string, std::string, std::string, std::string,
std::string, int);

int main()
{
	//Menu related variables
	int userChoice = 0;
	int error = 0;
	int startPos = 0;
	int length = 0;

	//String declared to interface with new myString class
	std::string holderString;
	std::string holderString2;
	std::string result;

	//Declare all output file related variables
	std::string fileName = "outputLog.txt";
	std::ofstream output;

	//Two seperate instances of myString to allow methods that include parameters
or comparisons of type myString
	myString testString;
	myString testString2;

	menu();

	//Open Log File
	output.open(fileName);

	//Enter Decision Phase
	do
	{
		//Defaults
		holderString = "";
		holderString2 = "";
```

```
            result = "";
            testString.initString();
            testString2.initString();
            int error = 0;
            int startPos = 0;
            int length = 0;

            std::cout << std::endl << "Choice: ";
            std::cin >> userChoice;
            switch (userChoice)
            {
            case 0:
                  //Close Output File
                  output.close();
                  return 0;
                        break;
            case 1:
                  std::cout << "Testing myString::size()" << std::endl;
                  std::cout << "Fill String  : ";
                  //Take in as string::string
                  std::cin >> holderString;
                  //Set as myString
                  testString.setString(holderString);

                  //Process
                  //Print Result
                  std::cout << "Result       : ";
                  std::cout << testString.size();

                  result = std::to_string(testString.size());
                  error = testString.getErrorCode();

                  printErrorCode(error);

                  printLog(output, "size()", holderString, "NULL", "NULL", result,
error);
                  break;

            case 2:
                  std::cout << "Testing myString::addStart(myString)" << std::endl;
                  std::cout << "Fill String 1: ";
                  //Take in as string::string
                  std::cin >> holderString;
                  //Set as myString
                  testString.setString(holderString);

                  std::cout << "Fill String 2: ";
                  //Take in as string::string
```

```
                    std::cin >> holderString2;
                    //Set as myString
                    testString2.setString(holderString2);

                    //Process
                    testString.addStart(testString2);

                    //Print Result
                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "addStart(myString)", holderString,
holderString2, "NULL", result, error);
                    break;

             case 3:
                    std::cout << "Testing myString::addEnd(myString)" << std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    std::cout << "Fill String 2: ";
                    //Take in as string::string
                    std::cin >> holderString2;
                    //Set as myString
                    testString2.setString(holderString2);

                    //Process
                    testString.addEnd(testString2);

                    //Print Result
                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "addEnd(myString)", holderString, holderString2,
"NULL", result, error);
```

```
                    break;

            case 4:
                    std::cout << "Testing myString::partString(startPos, length)" <<
std::endl;
                    std::cout << "Fill String  : ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    std::cout << "startPos     : ";
                    std::cin >> startPos;

                    std::cout << "length       : ";
                    std::cin >> length;

                    //Process
                    testString.replWholeString(testString.partString(startPos,
length));

                    //Print Result
                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "partString(startPos, length)", holderString,
std::to_string(startPos), std::to_string(length), result, error);
                    break;

            case 5:
                    std::cout << "Testing myString::replPartString(myString,
startPos)" << std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    std::cout << "Fill String 2: ";
                    //Take in as string::string
                    std::cin >> holderString2;
                    //Set as myString
                    testString2.setString(holderString2);
```

```
                    std::cout << "startPos     : ";
                    std::cin >> startPos;

                    //Process
                    testString.replPartString(testString2, startPos);

                    //Print Result
                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "replPartString(myString, startPos)",
holderString, holderString2, std::to_string(startPos), result, error);
                    break;

             case 6:
                    std::cout << "Testing myString::replWholeString(myString)" <<
std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    std::cout << "Fill String 2: ";
                    //Take in as string::string
                    std::cin >> holderString2;
                    //Set as myString
                    testString2.setString(holderString2);

                    //Process
                    testString.replWholeString(testString2);

                    //Print Result
                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);
```

```
                    printLog(output, "replWholeString(myString)", holderString,
holderString2, "NULL", result, error);
                    break;

             case 7:
                    std::cout << "Testing myString::compareString(myString)" <<
std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    std::cout << "Fill String 2: ";
                    //Take in as string::string
                    std::cin >> holderString2;
                    //Set as myString
                    testString2.setString(holderString2);

                    //Process
                    //Print Result
                    std::cout << "Result       : ";
                    std::cout << (testString.compareString(testString2));

                    result = testString.compareString(testString2);
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "compareString(myString)", holderString,
holderString2, "NULL", result, error);
                    break;

             case 8:
                    std::cout << "Testing myString::initString()" << std::endl;
                    std::cout << "Fill String  : ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    //Process
                    testString.initString();

                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
```

```
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "initString()", holderString, "NULL", "NULL",
result, error);
                    break;

            case 9:
                    std::cout << "Testing myString::setString(std::string)" <<
std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    //Processs
                    testString.setString(holderString);

                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "setString(string)", holderString, holderString,
"NULL", result, error);
                    break;

            case 10:
                    std::cout << "Testing myString::getString()" << std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    //Processs
                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);
```

```
                    printLog(output, "getString()", holderString, "NULL", "NULL",
result, error);
                    break;

            case 11:
                    std::cout << "Testing myString::printStringScreen()" <<
std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    //Process
                    std::cout << "Result       : ";
                    testString.printStringScreen();

                    result = testString.getString();
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "printStringScreen()", holderString, "NULL",
"NULL", result, error);
                    break;

            case 12:
                    std::cout << "Testing myString::numericString()" << std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    if (testString.numericString())
                    {
                            //Process
                            std::cout << "Result       : ";
                            std::cout << "Is a Real Number";
                            result = "Is Real Number";
                    }
                    else
                    {
                            //Process
                            std::cout << "Result       : ";
                            std::cout << "Not a Real Number";
                            result = "Not Real Number";
                    }
```

```
                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "numericString()", holderString, "NULL", "NULL",
result, error);
                    break;

             case 13:
                    std::cout << "Testing myString::alphabeticString()" << std::endl;
                    std::cout << "Fill String 1: ";
                    //Take in as string::string
                    std::cin >> holderString;
                    //Set as myString
                    testString.setString(holderString);

                    if (testString.alphabeticString())
                    {
                           //Process
                           std::cout << "Result       : ";
                           std::cout << "Is Alphabetic";
                           result = "Is Alphabetic";
                    }
                    else
                    {
                           //Process
                           std::cout << "Result       : ";
                           std::cout << "Not Alphabetic";
                           result = "Not Alphabetic";
                    }

                    error = testString.getErrorCode();

                    printErrorCode(error);

                    printLog(output, "alphabeticString()", holderString, "NULL",
"NULL", result, error);
                    break;

             case 14:
                    menu();
                    break;

                    default:
                           std::cout << "Invalid Choice, Try Again" << std::endl;
                           break;
                    }
```

```
            }
      while (userChoice != 0);

      //system("pause");
      return 0;
}


void menu()
{
      //Prompt
      std::cout << "Please select a function to test by entering it's number: ";
      std::cout << std::endl;
      std::cout << "Functions of myString
      " << std::endl;
      std::cout << "(1)  size()
      " << std::endl;
      std::cout << "(2)  addStart(myString)
" << std::endl;
      std::cout << "(3)  addEnd(myString)
      " << std::endl;
      std::cout << "(4)  partString(startPos, length)
" << std::endl;
      std::cout << "(5)  replPartString(myString, startPos)                  " <<
std::endl;
      std::cout << "(6)  replWholeString(myString)                           " <<
std::endl;
      std::cout << "(7)  compareString(myString)
" << std::endl;
      std::cout << "(8)  initString()
      " << std::endl;
      std::cout << "(9)  setString(string)
" << std::endl;
      std::cout << "(10) getString()
      " << std::endl;
      std::cout << "(11) printStringScreen()
" << std::endl;
      std::cout << "(12) numericString()
      " << std::endl;
      std::cout << "(13) alphabeticString()
" << std::endl;
      std::cout << std::endl;
      std::cout << "Other Function
      " << std::endl;
      std::cout << "(14) menu() - Draw this menu again                       " <<
std::endl;
      std::cout << "(0)  EXIT
            " << std::endl;
```

```
}

void printErrorCode(int error)
{
	std::cout << std::endl;
	switch (abs(error))
	{
	case 0:
		std::cout << "Error Code   : 0  - No Errors Occured" << std::endl;
		break;
	case 1:
		std::cout << "Error Code   : -1  *Attempted to fill string with more
than 25 character*" << std::endl;
		break;
	case 2:
		std::cout << "Error Code   : -2  *Attempted to access index above 25*"
<< std::endl;
		break;
	case 3:
		std::cout << "Error Code   : -3  *Attempted to fill with whitespace*" <<
std::endl;
		break;
	}
}

void printLog(std::ofstream& out, std::string function, std::string initialString,
std::string paramOne, std::string paramTwo, std::string result, int error)
{
	out << "Function Tested : " << function << std::endl;
	out << "Initial String  : " << initialString << std::endl;
	out << "Parameter 1     : " << paramOne << std::endl;
	out << "Parameter 2     : " << paramTwo << std::endl;
	out << "Result          : " << result << std::endl;

	switch (abs(error))
	{
	case 0:
		out << "Error Code      : 0  - No Errors Occured" << std::endl;
		break;
	case 1:
		out << "Error Code      : -1  *Attempted to fill string with more than
25 character*" << std::endl;
		break;
	case 2:
		out << "Error Code      : -2  *Attempted to access index above 25*" <<
std::endl;
		break;
	case 3:
```

```
            out << "Error Code      : -3  *Attempted to fill with whitespace*" <<
std::endl;
            break;
        }
        out << std::endl;
}
```

9. Updated Algorithm

1. Create Class *myString*

    a. Define Privates
        i. Const int MAX_SIZE = 25
        ii. Integer errorCode = 0
        iii. Integer length = 0;
        iv. Character array charArray[25]
    b. Define Publics
    c. Setter Functions
    d. Create Function *setString(string)*
        i. If input size > MAX_SIZE
            1. *setErrorCode(-1)*
        ii. Else
            1. Take in string
            2. Iterate and fill charArray[]
    e. Create Function *setErrorCode(int input)*
        i. errorCode = input
    f. Create Function *setLength(int input)*
        i. Length = input
    g. Getter Functions
    h. Create *getString()*
        i. Returns charArray as string
    i. Create Function *getErrorCode()*
        i. Returns errorCode
    j. Create Function *getLength()*
        i. Returns length
    k. Create Function *size*()
        i. Return number of characters in myString
        ~~ii. Count all non-null characters and add to *total*~~
    l. Create Function *addStart*(myString input)
        ~~i. Add input to beginning of string~~
        ~~ii. Move current characters forward and insert~~
        iii. Add together old size and new size for setErrorCode()
        iv. Move original myString into holder

v. Push new myString to front
vi. Re-add old myString

m. Create Function *addEnd*(myString)
i. Add together old size and new size for setErrorCode()
ii. If two sizes <= 25
1. Append string with input
~~iii. Insert characters to end up to 25~~

n. Create Function *partString*(startPos, length)
i. Add together old size and new size for setErrorCode()
ii. If two sizes <= 25
1. Take in index integer and iterate from there
2. Return portion of string from inputted index

o. Create Function *replPartString*(myString, startPos)
i. Add together old size and new size for setErrorCode()
ii. If two sizes <= 25
1. Replace portion of string from inputted index
2. Take in index of string and replace

p. Create Function *replWholeString*(myString)
i. initString()
ii. ~~Replace~~ Fill entire string with user input
~~iii. Wipe string and insert characters~~

q. Create Function *compareString*(myString)
i. Compare current string to user input
ii. Iterate both and check each index

r. Create Function *initString*()
i. Wipe/Initialize string to null
ii. Iterate and clear each index

~~*s.* Create Function *setString*(string)~~
~~i. Set string to current input~~
~~ii. Iterate and fill each index with corresponding index~~

~~*t.* Create Function *getString*()~~
~~i. Return current string~~
~~ii. Iterate and return each index into string~~

u. Create Function *printStringScreen*()
i. Print current string to screen
ii. Iterate and print each index

v. Create Function *numericString*()
~~i. Determine if string is integer or real number~~
~~ii. Iterate and check for numeric or real number characters in correct order~~
iii. Use flawed method of casting to double since you're incompetant or don't have enough time to learn regex to do this faster.

w. Create Function *alphabeticString*()
i. Determine if string is all alphabetic

        ii. Iterate and check for alpha characters

2. ~~Create Function *outputFile*()~~
    a. ~~Will print results of each function in class *myString* to file~~
3. *MAIN* FUNCTION
    a. ~~Test each function of class *myString*~~
    b. ~~Print results of each function to file with *outputFile*().~~
    c. Create Function menu()
        i. Prints out list of all functions to test with respective number
    d. Create printErrorCode()
        i. Simplifies showing error codes in console
    e. Create printLog()
        i. Take in output stream, string, string, string, string, string, int
        ii. Format that data by file, function, original string, parameter one, parameter two, result, error.
        iii. Format
    f. Create main()
        i. Create holder variables
            1. Int userChoice
            2. Int error
            3. Int startPos
            4. Int Length
            5. Holderstring
            6. Holderstring2
            7. Result
            8. Filename
            9. Output stream
            10. myString
            11. myString2
        ii. Run menu()
        iii. Open file
        iv. Do
        v. Load defaults for all variables above
        vi. Ask for userchoice
            1. Case 1
                a. Run size()
                b. Output log for size()
            2. Case 2
                a. Run addStart()
                b. Output log for above
            3. Case 3
                a. Run addEnd()
                b. Output log for above
            4. Case 4

a. Run partString()
b. Output log for above

5. Case 5
    a. Run replPartString()
    b. Output log for above
6. Case 6
    a. Run replWhokeString()
    b. Output log for above
7. Case 7
    a. Run compareString()
    b. Output log for above
8. Case 8
    a. Run initString()
    b. Output log for above
9. Case 9
    a. Run setString()
    b. Output log for above
10. Case 10
    a. Run getString()
    b. Output log for above
11. Case 11
    a. Run printStringScreen()
    b. Output log for above
12. Case 12
    a. Run numericString()
    b. Output log for above
13. Case 13
    a. Run alphabeticString()
    b. Output log for above
14. Case 14
    a. Run menu() again
15. Default
    a. “Invalid choice”

vii. While (userChoice != 0)

viii. Return 0 / EXIT

10. Test Plan Version 3

*Linux tests all ended in the same fashion. Those duplicates are withheld.

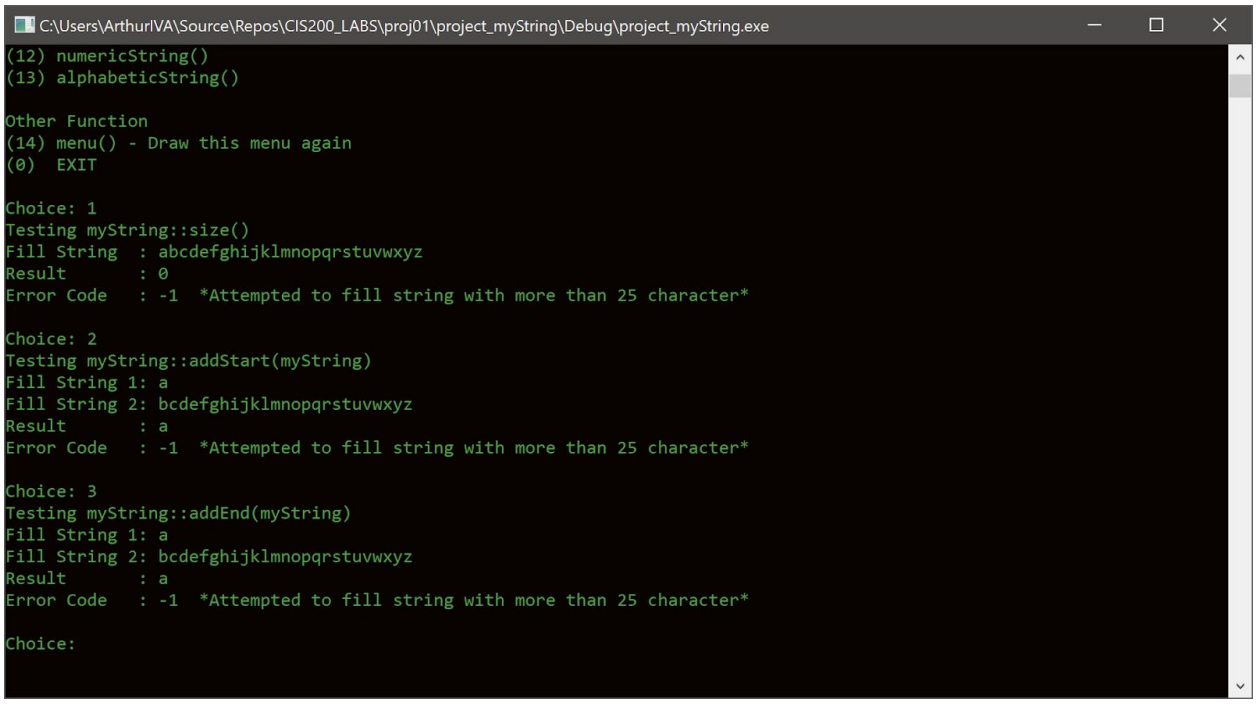
Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail

Valid Data	1	*size*()	“string”	6	6	Pass
Valid Data	2	*addStart*()	“string” “my“	“mystring”	“mystring”	Pass
Valid Data	3	*addEnd*()	“mystring” “s”	“mystrings”	“mystrings”	Pass
Valid Data	4	*partString*()	1, 5 “mystring”	“mystr”	“mystr”	Pass
Valid Data	5	*replPartString*()	3, “mystring” “Test”	“myTest”	“myTest”	Pass
Valid Data	6	*replWholeString*()	“mystring” “testing”	“testing”	“testing”	Pass
Valid Data	7	*compareString*()	“mystring” “test”	FALSE	0 / False	Pass
Valid Data	8	*initString*()	null	null string	null string	Pass
Valid Data	9	*setString*()	“mystr” *string*	“mystr” *myString*	“mystr” *myString*	Pass
Valid Data	10	*getString*()	“mystr” *myString*	“mystr” *string*	“mystr” *string*	Pass
Valid Data	11	*printStringScreen*()	“mystr”	“mystr” Print	“mystr” Print	Pass
Valid Data	12	*numericString*()	“6.502”	TRUE	TRUE	Pass
Valid Data	13	*alphabeticString*()	“abc”	TRUE	TRUE	Pass
Valid Data	14	*printLog*()	Valid Test 4 - Invalid Test 4	Good Output	outputLog.txt	Pass
Invalid Data	1	*size*()	“abcd...z”	Error -1	Error -1	Pass
Invalid Data	2	*addStart*()	“a” “bcd...z”	Error -1	Error -1	Pass
Invalid Data	3	*addEnd*()	“a” “bcd...z”	Error -1	Error -1	Pass
Invalid Data	4	*partString*()	“26, 5”	Invalid Index	Error	Pass
Invalid Data	5	*replPartString*()	“my“, 26	Invalid Index	Error	Pass
Invalid Data	6	*replWholeString*()	“abc...z”	overflow	Error	Pass
~~Invalid Data~~	~~7~~	~~*compareString*()~~	~~“”~~	~~null~~	Turns out these arent testable	
~~Invalid Data~~	~~8~~	~~*setString*()~~	~~“”~~	~~null~~		
~~Invalid Data~~	~~9~~	~~*getString*()~~	~~“”~~	~~null - empty~~		

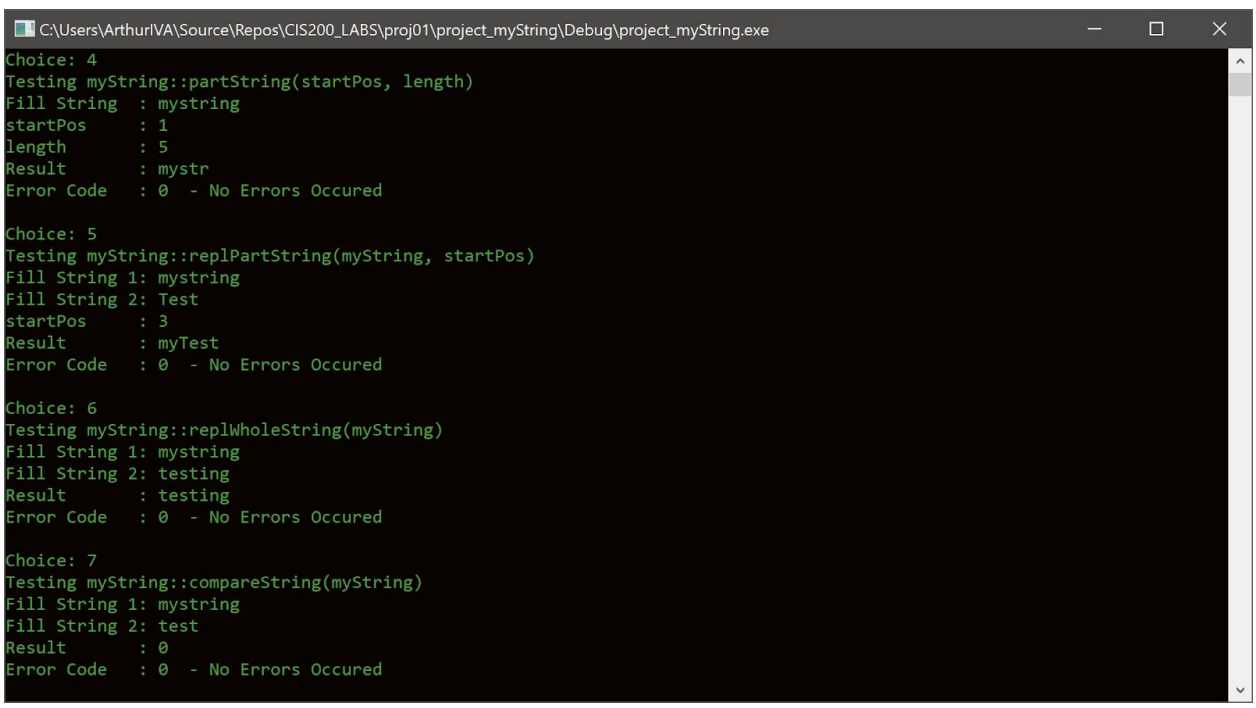
~~Invalid Data~~	~~10~~	~~*printStringScreen*()~~	~~“”~~	~~null - empty~~		

11. Screenshots

Valid Tests 1, 2, 3



Valid Tests 4, 5, 6, 7



Valid Tests 8, 9, 10, 11

```
C:\Users\ArthurIVA\Source\Repos\CIS200_LABS\proj01\project_myString\Debug\project_myString.exe

Choice: 8
Testing myString::initString()
Fill String  : testing
Result       :
Error Code   : 0  - No Errors Occured

Choice: 9
Testing myString::setString(std::string)
Fill String 1: mystr
Result       : mystr
Error Code   : 0  - No Errors Occured

Choice: 10
Testing myString::getString()
Fill String 1: mystr
Result       : mystr
Error Code   : 0  - No Errors Occured

Choice: 11
Testing myString::printStringScreen()
Fill String 1: mystr
Result       : mystr
Error Code   : 0  - No Errors Occured

Choice:
```

Valid Tests 12, 13

```
C:\Users\ArthurIVA\Source\Repos\CIS200_LABS\proj01\project_myString\Debug\project_myString.exe

Choice: 12
Testing myString::numericString()
Fill String 1: 6.502
Result       : Is a Real Number
Error Code   : 0  - No Errors Occured

Choice: 13
Testing myString::alphabeticString()
Fill String 1: abc
Result       : Is Alphabetic
Error Code   : 0  - No Errors Occured

Choice:
```

Valid Test 14

[outputLog.txt]

Function Tested : partString(startPos, length)

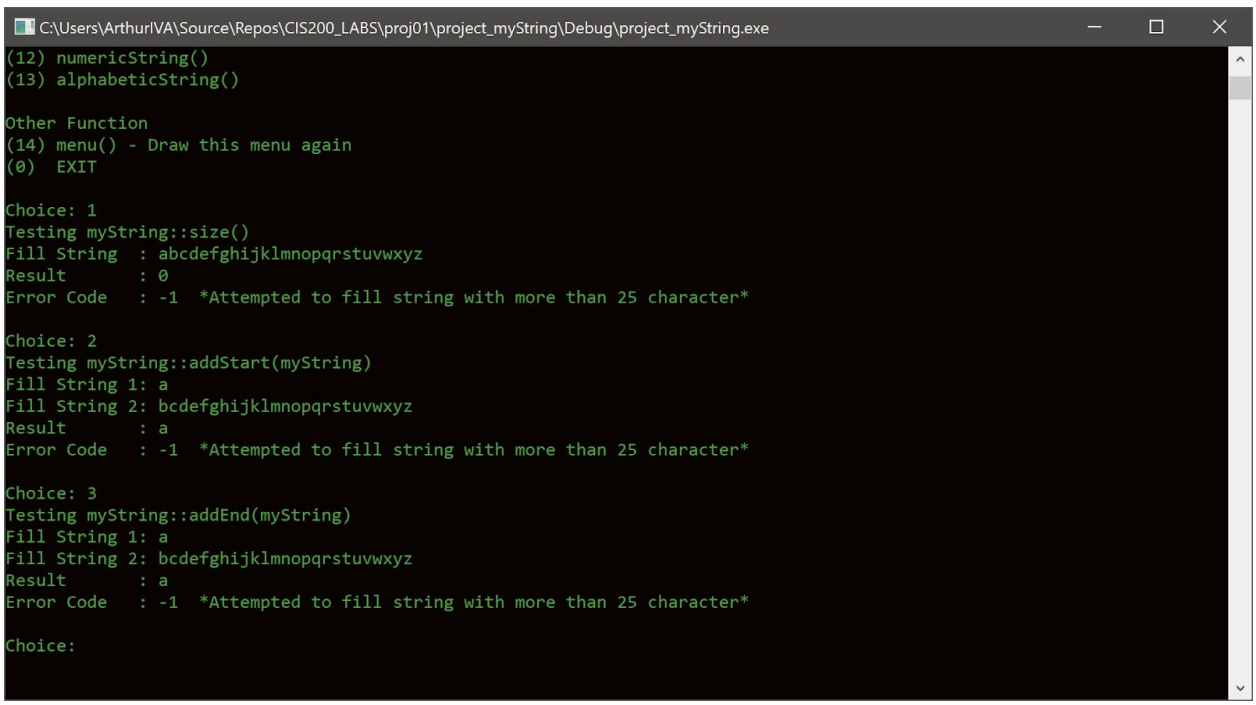
```
Initial String  : mystring
Parameter 1     : 1
Parameter 2     : 5
Result          : mystr
Error Code      : 0  - No Errors Occured

Function Tested : replPartString(myString, startPos)
Initial String  : mystring
Parameter 1     : Test
Parameter 2     : 3
Result          : myTest
Error Code      : 0  - No Errors Occured

Function Tested : replWholeString(myString)
Initial String  : mystring
Parameter 1     : testing
Parameter 2     : NULL
Result          : testing
Error Code      : 0  - No Errors Occured

Function Tested : compareString(myString)
Initial String  : mystring
Parameter 1     : test
Parameter 2     : NULL
Result          :
Error Code      : 0  - No Errors Occured

Function Tested : initString()
Initial String  : testing
Parameter 1     : NULL
Parameter 2     : NULL
Result          :
Error Code      : 0  - No Errors Occured

Function Tested : setString(string)
Initial String  : mystr
Parameter 1     : mystr
Parameter 2     : NULL
Result          : mystr
Error Code      : 0  - No Errors Occured

Function Tested : getString()
Initial String  : mystr
```

```
Parameter 1     : NULL
Parameter 2     : NULL
Result          : mystr
Error Code      : 0  - No Errors Occured

Function Tested : printStringScreen()
Initial String  : mystr
Parameter 1     : NULL
Parameter 2     : NULL
Result          : mystr
Error Code      : 0  - No Errors Occured

Function Tested : numericString()
Initial String  : 6.502
Parameter 1     : NULL
Parameter 2     : NULL
Result          : Is Real Number
Error Code      : 0  - No Errors Occured

Function Tested : alphabeticString()
Initial String  : abc
Parameter 1     : NULL
Parameter 2     : NULL
Result          : Is Alphabetic
Error Code      : 0  - No Errors Occured

Function Tested : size()
Initial String  : abcdefghijklmnopqrstuvwxyz
Parameter 1     : NULL
Parameter 2     : NULL
Result          : 0
Error Code      : -1  *Attempted to fill string with more than 25 character*

Function Tested : addStart(myString)
Initial String  : a
Parameter 1     : bcdefghijklmnopqrstuvwxyz
Parameter 2     : NULL
Result          : a
Error Code      : -1  *Attempted to fill string with more than 25 character*

Function Tested : addEnd(myString)
Initial String  : a
Parameter 1     : bcdefghijklmnopqrstuvwxyz
```

Parameter 2 : NULL
Result : a
Error Code : -1 *Attempted to fill string with more than 25 character*

Function Tested : partString(startPos, length)
Initial String : mystring
Parameter 1 : -1
Parameter 2 : 5
Result :
Error Code : -1 *Attempted to fill string with more than 25 character*

Invalid Tests 1, 2, 3



Invalid Tests 4, 5, 6

```
C:\Users\ArthurIVA\Source\Repos\CIS200_LABS\proj01\project_myString\Debug\project_myString.exe
Other Function
(14) menu() - Draw this menu again
(0)  EXIT

Choice: 4
Testing myString::partString(startPos, length)
Fill String  : mystring
startPos     : 26
length       : 5
Result       :
Error Code   : -2  *Attempted to access index above 25*

Choice: 5
Testing myString::replPartString(myString, startPos)
Fill String 1: mystring
Fill String 2: test
startPos     : 26
Result       : mystring
Error Code   : -3  *Attempted to fill with whitespace*

Choice: 6
Testing myString::replWholeString(myString)
Fill String 1: mystring
Fill String 2: abcdefghijklmnopqrstuvwxyz
Result       :
Error Code   : -3  *Attempted to fill with whitespace*

Choice:
```

12. Error Log

Error Type (Logic/Runtime)	Cause of Error	Solution to Error
Logic	Casting charArray to double within numericString() give correct answer every time except with input = 0	None at the moment.

13. Status

Program is working.