Object-Oriented Programming *COURSE* – Fall 2021

(BS-CS-F20 Morning & Afternoon)

Assignment # 1

*Assigned on: **Thursday, December 23, 2021***

*Submission Deadline: **Thursday, December 30, 2021 (till 5:00 PM)***

Instructions:

- **This is an individual assignment. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an "F" grade in this course.**
- This assignment will ALSO be counted towards your marks in **OOP-Lab**.
- You are required to create a **multi-file project** for this assignment.
- Do **NOT** copy even a single line of code from any other person or book or Internet or any other source.
- Late submissions will NOT be accepted, whatever your excuse may be.
- This assignment needs to be submitted in **Soft** form. The **Submission Procedure** is given below.

Submission Procedure:

i. Create an empty folder. Put all of the ***.CPP* and .H file(s)** of your assignment in this folder. Do NOT include any other files in your submission, otherwise your submission will NOT be graded. Don't forget to mention your Name, Roll Number and Section in comments at the top of each CPP file.

ii. Now, compress the folder (that you created in previous step) in **.RAR** or **.ZIP** format. The name of the RAR or ZIP file should be ***exactly*** according to the format:

`Mor-A1-BCSF20M123` (for Morning section) OR

`Aft-A1-BCSF20A456` (for Afternoon section),

where the text shown in **BLUE** should be your **complete Roll Number**.

iii. Finally, submit the (single) **.RAR** or **.ZIP** file through **Google Classroom**. Next week I will share the link of **Google Classroom.**

Note: *If you do not follow the above submission procedure, your Assignment will NOT be graded and you will be awarded ZERO marks.*

These *good programming practices* will also have their (significant) weightage in the marking of this assignment:

- Comment your code intelligently. **Uncommented code will NOT be given any credit.**
- Indent your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for input/output.

If your program **does NOT compile correctly** or it **generates any kind of run-time error** (dangling pointer, memory leak etc.) you will get **ZERO marks** in the whole assignment.

Create a C++ class **MyString** which is described below:

```
class MyString
{
  private:
      char * str;        // Pointer to the dynamically allocated null-terminated char array
      int length;        // Variable to store the length of the string (excluding '\0')
  public:
      // Public member functions of the MyString class are described below
};
```

1. Implement the **Default Constructor** for **MyString** class which should make **str** point to an **empty c-string** (on heap) and it should initialize **length** to 0.

2. Implement an **Overloaded Constructor** for **MyString** class that takes a c-string (passed as **const char ***) as its only parameter and allocates/initializes the member variables **str** and **length** appropriately. The prototype of the overloaded constructor should be:

```
MyString (const char *);
```

Test the working of the constructors by creating various objects of **MyString** type. For example:

```
MyString s1;                                      // length of s1 should be 0
MyString s2 ("OOP is Fun!!");                     // length of s2 should be 12

char name[15] = "Pakistan";
MyString s3 (name);                               // length of s3 should be 8
```

3. Implement the **Destructor** for **MyString** class which should properly deallocate all the dynamically allocated memory.

4. Implement a ***constant*** member function **display()** of the **MyString** class which can be used to display the string on screen.

5. Implement a ***constant*** member function **getLength()** of the **MyString** class which returns the length (excluding the null terminator) of the string.

6. Implement the **Copy Constructor** for **MyString** class which should make a ***deep copy*** of the **MyString** object whose reference it will receive as a parameter.

7. Overload the **Assignment Operator** for **MyString** class which can be used to assign one **MyString** object to another. Make sure that your implementation makes a ***deep copy***. The prototype of the overloaded assignment operator should be:

```
const MyString& operator = (const MyString&);
```

8. Overload the **Stream insertion operator `<<`** (as a friend function) to display a **`MyString`** on screen. Make sure that this Stream insertion operator can be used in a ***cascaded way***.

9. Overload the **`+`** operator to concatenate two **`MyString`** objects. This function should return the newly created **`MyString`**. In this function, you are **NOT** allowed to use any library function from the **`<cstring>`** or **`<string>`** header files. The prototype of this function should be:

```
MyString operator + (const MyString&) const;
```

Test the working of the **Copy constructor**, **Assignment operator**, **Stream insertion operator (`<<`)** and **operator +** by executing the following code:

```
MyString ms1("One");
MyString ms2("Two");
MyString ms3("Three");
MyString ms4;

ms4 = ms1 + ms2 + ms3;

cout << ms1;                  // should display: One
cout << ms2;                  // should display: Two
cout << ms3;                  // should display: Three
cout << ms4;                  // should display: OneTwoThree
```

10. Overload the **`==`** operator to compare two **`MyString`** objects for equivalence. In this function, you are **NOT** allowed to use any library function from the **`<cstring>`** or **`<string>`** header files. You MUST implement the logic of checking equality completely by yourself. The prototype of this function should be:

```
bool operator == (const MyString&) const;
```

11. Overload the **`<`** operator to compare two **`MyString`** objects to determine whether one **`MyString`** object (left operand) is **alphabetically smaller** than the other **`MyString`** object (right operand). In this function, you are **NOT** allowed to use any library function from the **`<cstring>`** or **`<string>`** header files. You MUST implement the logic completely by yourself. The prototype of this function should be:

```
bool operator < (const MyString&) const;
```

12. Overload the **`<=`** operator to compare two **`MyString`** objects to determine whether one **`MyString`** object (left operand) is **alphabetically smaller or equal** to the other **`MyString`** object (right operand). In this function, you are **NOT** allowed to use any library function from the **`<cstring>`** or **`<string>`** header files. You MUST implement the logic completely by yourself. The prototype of this function should be:

```
bool operator <= (const MyString&) const;
```

Test the working of the relational operators **`==`**, **`<`**, and **`<=`** by executing the following code:

```
MyString s1 ("abc");
MyString s2 ("ABC");
MyString s3 ("az");
MyString s4 ("abc");

cout << (s1 == s2);         // should display: 0 (false)
cout << (s4 == s1);         // should display: 1 (true)
cout << (s4 == s4);         // should display: 1 (true)

cout << (s1 < s2);          // should display: 0 (false)
cout << (s2 < s1);          // should display: 1 (true)

cout << (s1 < s3);          // should display: 1 (true)
cout << (s1 <= s3);         // should display: 1 (true)
cout << (s1 <= s4);         // should display: 1 (true)
cout << (s2 <= s2);         // should display: 1 (true)

cout << (s3 < s1);          // should display: 0 (false)
cout << (s3 <= s1);         // should display: 0 (false)
cout << (s3 <= s3);         // should display: 1 (true)
```

13. Overload the **`+=`** operator to concatenate another **`MyString`** object with the current **`MyString`** object. If there is not enough space in the current object, then your function should allocate a new char array (of an appropriate size) and should still successfully perform the concatenation. In this function, you are **NOT** allowed to use any library function from the **`<cstring>`** or **`<string>`** header files. The prototype of this function should be:

```
const MyString& operator += (const MyString&);
```

Test the working of **operator +=** by executing the following code:

```
MyString ms1("One");
MyString ms2("Two");
MyString ms3("Three");

ms1 += ms2 += ms3;

ms1.display();              // should display: OneTwoThree
ms2.display();              // should display: TwoThree
ms3.display();              // should display: Three
```

14. Implement the following member function of the **`MyString`** class, which takes a c-string (passed as a **`const char *`**) as its only parameter and changes the contents of the member variables **`str`** and **`length`** appropriately. You may also need to deallocate and reallocate memory in this function. The prototype of this function should be:

```
void setStr (const char *);
```

Test the working of the **`setStr`** function by executing the following code:

```
MyString ms1 ("Hello");
cout << ms1;                       // should display: Hello

ms1.setStr ("Welcome to OOP");
cout << ms1;                       // should display: Welcome to OOP
```

15. Implement a **conversion operator `int`** for **`MyString`** class which can be used to convert a **`MyString`** object into an **integer**. If the MyString object contains any non-digit characters, then this conversion operator should simply return **`0`**. See the following examples:

```
MyString s1 ("7504");
MyString s2 ("56 abc def");

int i1 = s1;                       // using the conversion operator int
int i2 = s2;                       // using the conversion operator int

cout << i1;                        // should display: 7504
cout << i2;                        // should display: 0
```

16. Implement a **conversion operator `char*`** for **`MyString`** class which can be used to convert a **`MyString`** object into a **c-string**. See the following example:

```
MyString s1 ("Hello World");

cout << s1;                        // should display: Hello World

char * cp = s1;                    // using the conversion operator char*
cout << cp;                        // should display: Hello World
delete [] cp;                      // Deallocating the dynamically allocated memory
cp = nullptr;

cout << s1;                        // should still display: Hello World
```

17. Overload the **pre-increment operator `++`** to convert the **`MyString`** to uppercase i.e. all **lowercase letters** in the calling **`MyString`** object will be converted into **uppercase**. All other characters should remain unchanged. See the following examples:

```
MyString s1 ("One !#^; 123 two");
MyString s2;

s2 = ++s1;

s1.display();                  // should display: ONE !#^; 123 TWO
s2.display();                  // should display: ONE !#^; 123 TWO
```

18. Overload the **stream extraction operator `>>`** (as a friend function) to take input of a **`MyString`** from the user. In the implementation of this function, you will need to deallocate and reallocate the array pointed to by **`str`** according to the # of characters entered by the user. You can assume that the string entered by the user contains **at most 100 characters**. Also make sure that this stream extraction operator can be used in a ***cascaded way***.

```
MyString s1 ("Hello");
cout << s1;                    // should display: Hello

cout << "Enter a MyString: ";
cin >> s1;          // Suppose the user enters: Pakistan Zindabad!!

cout << s1;                    // should display: Pakistan Zindabad!!
```

Important Note:

Finally, write a driver program (**`main`** function) to test the working of ALL of the above ***18*** member functions of the **`MyString`** class. In the main function, you MUST clearly mention the corresponding ***Function #*** (***1*** to ***18***) and ***Function name*** in ***Comments*** along with each piece of code which you have written to test a particular member function of the **`MyString`** class.

☺ GOOD LUCK! ☺

Remember: Honesty always gives fruit (no matter how frightening is the consequence); and Dishonesty is always harmful (no matter how helping it may seem in a certain situation)!