CS 202 - Computer Science II

Project 5

Due date (FIXED): Friday, 4/7/2017, 11:59 pm

Objectives: The main objectives of this project are to test your ability to create and use dynamic memory, and to review your knowledge to manipulate classes, pointers and iostream to all extents.

Description:
For this project you will create your own **String** class. **Pointer-based** and **Bracket-notation** ([ ]) in order to perform **array manipulation** are both **allowed** from now on. The following header file extract gives the required specifications for the class:

```
//Necessary preprocessor #define(s)
…
//Necessary include(s)
…
//Your C-style string helper functions if necessary
…
//Class specification
class MyString{

  public:
    MyString();                                             //(1)
    MyString(const char* str);                              //(2)
    MyString(const MyString& other_myStr);                  //(3)
    ~MyString();                                            //(4)

    int size() const;                                       //(5)
    int length() const;                                     //(6)
    const char* c_str() const;                              //(7)

    bool operator== ( const MyString& other_myStr) const;   //(8)
    MyString& operator= (const MyString& other_myStr);      //(9)
    MyString& operator+ (const MyString& other_myStr);      //(10)
    char& operator[] (int position);                        //(11)

friend ostream& operator<<(ostream& os, const MyString& myStr); //(12)

  private:
    void buffer_deallocate();                               //(13)
    void buffer_allocate(int size);                         //(14)

    char *m_buffer;
    int m_size;
};
…
```

Specifications explained:

The **MyString** Class will contain the following **private** data members:

- **m_buffer,** a char-type pointer, pointing to the Dynamically Allocated data. Note that is no longer a static array. Dynamic Memory management has to guarantee that it points to a properly allocated memory region, otherwise Segmentation Faults will occur in your program. Also, Dynamic Memory management has to guarantee that it is properly deallocated when appropriate, and deallocated-&-reallocated when its size has to change.
- **m_size**, an int, denoting how many characters are currently allocated for m_buffer. Note that this has to be properly initialized and updated each time the dynamically allocated memory is changed.

, will have the following **private** helper methods**:**

- **(13) buffer_deallocate** – will deallocate the dynamic memory pointed to by m_buffer. Note tht m_size which keeps track of m_buffer has to be updated too.
- **(14) buffer_allocate** – will allocate the required **int size** of char array and point m_buffer to it. It also has to check whether there is an already allocated space for m_buffer, and properly deallocate it prior to reallocating the new memory required. Note that m_size which keeps track of m_buffer has to be updated too. (Hint: you may want to try implementing exception handling for the dynamic memory allocation performed in this method. It is not a requirement, but your method should at least be checking for NULL evaluation of the new expression).

and will have the following **public** member functions:

- **(1) Default Constructor** – will instantiate a new object with no valid data. Hint: which member(s) need to be initialized in this case?
- **(2) Parametrized Constructor** – will instantiate a new object which will be initialized to hold a copy of the C-string **str** passed as a parameter. Hint: has to properly handle allocation, and to do this it will need to "examine" the input C-string to know how many items long the allocated space for m_buffer has to be.
- **(3) Copy Constructor** – will instantiate a new object which will be a separate copy of the data of the **other_myStr** object which is getting copied. Hint: Remember deep and shallow object copies.
- **(4) Destructor** – will destroy the instance of the object. Hint: Any allocated memory pointed-to by m_buffer has to be deallocated in here.

- **(5) size** will return the size of the currently allocated char buffer.
- **(6) length** will return the size of the string without counting the null-terminating character. Hint: The return value of this and size() will of course be different.
- **(7) c_str** will return a pointer to a char array which will represent the C-string equivalent of the calling MyString object's data. Hint: It has to be a null-terminated char array in order to be a valid C-string representation.
- **(8) operator==** will check if (or if not) the calling object represents the same string as another MyString object, and return true (or false) respectively.
- **(9) operator=** will assign a new value to the calling object's string data, based on the data of the **other_mySt**r object passed as a parameter. Returns a reference to the calling object to be used for cascading operator= as per standard practice. Hint: Think what needs to happen before allocating new memory for the new data to be held by the calling object.

- **(10) operator+** will assign a new value to the calling object's string data, which will be the result of appending the **other_myStr** object's data to the original calling object's data (same rationale as string concatenation). Returns a reference to the calling object to be used for cascading operator= as per standard practice. Hint: Think the order in which you will allocate and deallocate memory to hold the new data for the calling object.
- **(11) operator[]** will allow by-reference accessing of a specific character at index **int position** within the allocated **m_buffer** char array. This allows to access the MyString data by reference and read/write at specific m_buffer locations. Note: Should not care if the position requested is more than the m_buffer size.

as well as a friend function:

- **(12) operator<<** will output (to terminal or file depending on the type of ostream& os object passed as a parameter to it) the MyString data (the C-string representation held within m_buffer).

The MyString.h header file should be as per the specifications. The MyString.cpp source file you create will hold the required implementations. You should also create a source file proj5.cpp which will be a test driver for your class.

The test driver has to demonstrate that your MyString class works as specified:

- You may use any strings (sentences, words, etc.) of your liking to demonstrate the use of all the class methods. Go through them one-by-one in code sections tagged by the appropriate number, the following example is considered enough:

```
//(1)
MyString ms_default();
//(2)
MyString ms_parametrized("MyString parametrized constructor!");
//(3)
MyString ms_copy(ms_parametrized);
//(4)
MyString* ms_Pt = new MyString("MyString to be deleted…");
delete ms_Pt;
ms_Pt = NULL;
//(5),(6)
MyString ms_size_length("Size and length test");
cout << ms_size_length.size() << endl;
cout << ms_size_length.length() << endl;
//(7)
MyString ms_toCstring("C-String equivalent successfully obtained!");
cout << ms_toCstring.c_str() << endl;
//(8)
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)
MyString ms_assign("Before assignment");
```

```
ms_assign = MyString("After performing assignment");
//(10)
MyString ms_append1("The first part");
MyString ms_append2(" and the second");
MyString ms_concat = ms_append1+ ms_append2;
//(11)
MyString ms_access("Access successful (NOT)");
ms_access[17] = 0;
//12
cout << ms_access << endl;
```

Do not forget to initialize pointers appropriately where needed.
Do not forget to perform allocation, dellocation, dellocation-&-reallocation of dynamic memory when needed! Memory accesssing without proper allocation will cause Segmentation Faults. Forgetting to deallocate memory will cause Memory Leaks!

The completed project should have the following properties:

- Written, compiled and tested using Linux.
- It must compile successfully on the department machines using Makefile(s), which will be invoking the g++ compiler. Instructions how to remotely connect to department machines are included in the Projects folder in WebCampus.
- The code must be commented and indented properly.
  Header comments are required on all files and recommended for the rest of the program.
  Descriptions of functions commented properly.
- A one page (minimum) typed sheet documenting your code. This should include the overall purpose of the program, your design, problems (if any), and any changes you would make given more time.

Turn in: Compressed Header & Source files, Makefile(s), and project documentation.

Submission Instructions:

- You will submit your work via WebCampus
- Name your code file proj5.cpp
- If you have header file, name it proj5.h
- If you have class header and source files, name them as the respective class (MyString.h MyString.cpp) This source code structure is not mandatory, but advised.
- Compress your:
  1. Source code
  2. Makefile(s)
  3. Documentation

  Do not include executable
- Name the compressed folder:
  PA#_Lastname_Firstname.zip
  Ex: PA5_Smith_John.zip

Late Submission:
A project submission is "late" if any of the submitted files are time-stamped after the due date and time. Projects will be accepted up to 24 hours late, with 20% penalty.