Assignment 5: MyString COP3330 Fall 2017

Due: Wednesday, November 15, 2017 at 11:59 PM

Objective

This assignment will provide experience in managing dynamic memory allocation inside a class as well as further experience with operator overloading.

Task

Create a class called `MyString`. This will be a string class which allows creation of string objects that have flexible sizes, intuitive operator syntax (through operator overloads), and other useful features. Your class will need to maintain the string internally as an array of characters and dynamic allocation techniques will need to be used since string sizes are not fixed. Your class should maintain any `MyString` object in a valid state at all times and it should not allow any memory leaks.

The class should be written in the files `mystring.h` and `mystring.cpp`. Provided is a starter version of the file `mystring.h` which already has many of the required features declared. Do not change the prototypes of the already-declared functions.

Since the intention of the assignment is for you to write a string class, you may not use the `<string>` library in the creation of this assignment. Use of the standard C++ `<string>` library will result in an automatic 0 grade. The point here is to learn how to build such a library yourself.

A sample driver program is also provided (called `driver.cpp`) to give you a better idea of how some of the functions can be called. This is not a complete and through set of tests so you will still need to write your own test calls to test your class more throughly. You can see the output in `sample.txt`. Note that in the one place that asked for input, the input was “John was here, but not there”.

Program Details and Requirements

1. **Data**
   Your class must allow for storage of a flexibly-sized string of characters. Make sure to declare any appropriate member data variables in the header file. All member data of your class must be private.

2. **Standard Constructors**
   - The default constructor should set the object to respresent an empty string
   - `MyString(const char*)` is a conversion constructor. A c-string will be passed as a parameter and this constructor should set up the string object to store that string inside. This will enable type conversions from c-strings to `MyString` objects.
   - `MyString(int)` is a conversion constructor that should convert an integer value to a string representation. For example if the value 1234 is passed in, the `MyString` object should now store the same string data that would be represented by the c-string “1234”.

Note that these last two constructors will allow automatic type conversions to take place, in this case conversions from `int` to `MyString` and c-style string to `MyString`. This makes our operator overloads move versatile as well. For example, the conversion constructor allows the following statements to work (assuming appropriate definitions of the assignment operator and the + overloads described later):

```
MyString s1 = "Hello, World";
MyString s2 = 12345;
MyString s3 = s1 + 15;          // "Hello, World15"
```

3. **Automatics**
   Since dynamic allocation is necessary, you will need to write appropriate definitions of the special functions: destructor, copy constructor, assignment operator. The destructor should clean up any dynamic memory when a `MyString` object is deallocated. The copy constructor and assignment operator should both be defined to make a deep copy of the object (copying all dynamic data in addition to regular member data) using appropriate techniques. Make sure that none of these functions will ever allow memory leaks in a program.

4. **I/O Functions**
   - `operator<<` should print out the data from the `MyString` object using the standard insertion operator syntax. There should be no extra formatting. Only the string store in the object is printed (no newlines, extra spacing, etc)
   - `operator>>` should read a string from an input stream. This operator should ignore any leading white space before the string data and then read until the next white space is encountered. Prior data in the `MyString` object is discarded. Note: this will only read one word at a time just like the `operator>>` for c-strings.
   - The `getline` function should read a string from an input string. This operator should read everything from the input stream (first parameter) into the `MyString` object (second parameter) until the specified delimiter character is encountered. Notice that if the function is called with 3 parameters, the third parameter is the delimiter. If the function is called with just 2 parameters, the delimiter is the newline by default. Prior data in the `MyString` object is discarded.

5. **Comparison Operators**
   Write overloads for all 6 of the comparison operators (<, >, <=, >=, ==, !=). Each of these operations should test two objects of type `MyString` and return an indication of true or false. You are testing the `MyString` objects for order and/or equality based on the usual meaning of order and equality for c-strings, which is lexicographic ordering. Remember that lexicographic ordering is based on the order of the ascii characters themselves, so it is not exactly the same as pure "alphabetical" ordering.

   Examples:
   ```
   "apple" < "apply"
   "Zebra" < "apple"    // uppercase letters come before lowercase
   "apple" == "apple"   // same strings
   ```

6. **Concatenation Operators**
   - `operator+` should concatenate the two operands together and return a new `MyString` as a result.

- `Operator+=` should concatenate the second operand onto the first one (i.e. changing the first one)

Examples:
```
MyString s1 = "Dog";
MyString s2 = "food";
MyString s3 = s1 + s2; // s3 is "Dogfood" and s1, s2 are not changed
s1 += s2;              // s1 is now "Dogfood"
```

7. Bracket Operators

The bracket operator overloads have these prototypes:
- `char& operator[] (unsigned int index);`
- `const char& operator[] (unsigned int index) const;`

Both of these should return the character at the given index position. Note that the first one returns the character by reference so it allows the slot to be changed. The second returns by `const` reference and is a `const` member function and will run in read-only situations.

Examples:
```
const MyString s = "I love Java";
MyString t = "I love C++";

// these two calls use the const version above
char ch = s[4]; // ch now stores 'v'
ch = s[7]; // ch now stores 'J'

// these calls use the non-const version above
t[0] = 'U'; // s is now "U love C++"
t[3] = 'i'; //s is now "U live C++
```

Note that since the parameter in each is an `unsigned int`, it is not possible to have a negative array index passed in. If the index passed to the function is too big (out of bounds for what's currently stored), then:
- The read-only version should just return the null character
- The L-value version should resize the stored string to accommodate the specified index. All new slots between the end of the prior string and the new index location should be set to spaces.

Examples:
```
const MyString s = "Howdy"; // length of s is 5 characters
char ch = s[10];            // ch now stores '\0'
MyString t = "Hello";       // length of t is 5 characters
t[7] = 'b';                 // t is now "Hello  b" (length is now 8)
```

8. Standard Accessors
- `getLength` should return the length of the stored string (i.e. number of characters). For example, "Hello" has 5 characters.
- `getCString` should return the actual stored data as a c-string (i.e. null-terminated char array).

9. **Substring Functions**
   There are two versions of `substring`. Both should return a `MyString` object that consists of a portion of the original string. Neither should change the calling object. The first parameter represents the starting index of the substring. The second parameter is optional and gives the length of the substring to return. If the length is too long or is not provided, default to the rest of the string.

   Examples:
```
MyString s = "Greetings, Earthling!";
MyString x, y, z;
x = s.substring(4);       // x is now "tings, Earthling!"
y = s.substring(3, 5);    // y is now "eting"
z = s.substring(16, 10); // z is now "ling!"
```

10. **`insert` Function**
    This function should change the calling object by inserting the data from the second parameter at the index given by the first parameter. If the index is out of bounds (longer than the length of the string), then just insert at the end. This function should also return the calling object.

    Examples:
```
MyString s = "Hello world";
s.insert(6, "cruel "); // s is now "Hello cruel world"
s.insert(20, "!!!");   // s is now "Hello cruel world!!!"
```

11. **`indexOf` Function**
    This function should search through the `MyString` to find the first occurrence of the pattern or substring given in the parameter. The function should return the first index where it was found. If the pattern was not found, it should return -1.

    Examples:
```
MyString s = "The bobcat likes to concatenate";
int x = s.indexOf("cat"); //x is now 7
x = s.indexOf("dog");     //x is now -1
```

General Requirements

- As usual, no global variables
- All member data must be private
- Use appropriate good programming practices as denoted on previous assignments
- Since the only output involved with your class will be in the << overload, your output must match mine exactly
- You may not use classes from the Standard Template Library. This includes `<string>` as the whole point of this assignment is for you to learn how to manage dynamic memory issues inside of a class yourself, not rely on STL classes to do it for you
- You may use standard I/O libraries like `<iostream>` and `<iomanip>` as well as the common C libraries `<cstring>` and `<cctype>`
- Do not use language or library features that are C++11 only

Extra Credit

Create an overloaded version of the – operator to "subtract" two `MyString` objects. The meaning of – is that it should return a `MyString` object that is the result of taking the first string and removing all instances of the second string from it.

Examples:

```
MyString s = "The bobcat concatenated the catapult with the catamaran";
MyString t = "cat";
MyString result = s – t; // stores "The bob conenated the apult with the amaran";
```

Hints and Tips

- **Converting between `int`s and characters**
  Be aware that 1 and '1' are not the same thing. '1' stores the ascii value of the character representing 1, which happens to be 49. The easiest way to convert between single digit integers and the character code representation is to add or subtract the ascii value of '0'.

  Example:

  ```
  int x = 5;
  char ch = x + '0'; //ch is '5'
  ```

- **Input**
  For the >> operator overload as well as the `getline` function, there are some issues to be careful about. You will not be able to just use the normal version of >> (or `getline`) for c-strings because it attempts to read consecutive characters until a delimiter is encountered. The problem here is that we will be entering an unknown number of characters, so you won't be able to allocate all of the space in advance. These operations may need to dynamically resize the array as you read characters in.

Submitting

Archive your `mystring.h`, `mystring.cpp`, and `README` files into a simple tar ball (no compression). Submit to the assignment 5 link on blackboard. Make sure the submitted files are named as specified by the syllabus and this writeup, and that you do not include any extra files.

General Advice

- Make sure to double check your blackboard submission to make sure everything works when downloaded.
- Email a copy of your finished homework files to your own FSU account. This email will have a time stamp that shows when they were sent and will also serve as a backup. Useful in case something happens to blackboard.
- Periodically (e.g. nightly) make a backup of your assignment to another machine (e.g. personal computer, linprog, email). Computers die and accidents happen, having a backup prevents you

from having to start from scratch. This is also a good feature to have in your makefile as you can abstract the details behind a single call.

- Make sure to include the `README` file as specified in the assignment syllabus http://ww2.cs.fsu.edu/~dennis/teaching/2017_fall_cop3330/docs/syllabus.pdf

COP 3330 Fall 2017 Assignment 5 Due: 2017-11-15 11:59 PM

Student Name: **Grader:**

Grade: **Date Submitted:** **Date Graded:**

Description	Earned	Possible	Comments
1. Class Structure		5	
2. Constructors		10	
3. Automatics		10	
4. I/O		10	
5. Comparisons		5	
6. Concatenation		10	
7. Bracket Operators		10	
8. Accessors		5	
9. Substring		5	
10. Insert		5	
11. IndexOf		5	
12. Difference		+5	
13. README		10	
14. Submission		10	

Notes: