C++ Programming

Lab 2

- (8 points) Write a String class which will be a wrapper class to the C-style strings. The strings will be of varying logical lengths, but have a fixed physical (maximum) length of MAXLEN (defined to be 128 characters). Your String class must implement all the appropriate methods including constructors, assignment, equality operators, the index operator [], reverse, indexOf (find), print, and read. Do not use any of the C *str* functions (e.g. strcmp, strlen, or strcpy). You will be writing them yourself, as static methods. Then, you will use your static methods. REMEMBER: never copy/paste code from any source - you must write everything yourself.
- **File organization**: Put the declaration of class String in a file named String.h. Put the definitions of the String methods in a file called String.cpp. Put your main program and testing functions in a file named string_test.cpp. string_test.cpp and String.cpp must include String.h. You can compile your program with the command as follows: `g++ String.cpp string_test.cpp -o string_test`
  You can run it by calling it as follows: `string_test`
- Class String declaration:

```
#define MAXLEN 128
class String
{
public:
  /// Both constructors should construct
  /// this String from the parameter s
  String(const char * s = "");
  String(const String & s);
  String operator = (const String & s);
  char & operator [] (int index);
  int size();
  String reverse(); // does not modify this String
  int indexOf(const char c);
  int indexOf(const String pattern);
  bool operator == (const String s);
  bool operator != (const String s);
  bool operator > (const String s);
  bool operator < (const String s)
  bool operator <= (const String s);
  bool operator >= (const String s);
  /// concatenates this and s to return result
  String operator + (const String s);
  /// concatenates s onto end of this
  String operator += (const String s);
  void print(ostream & out);
  void read(istream & in);
  ~String();
private:
  bool inBounds(int i)
```

```
    {
      return i >= 0 && i < strlen(buf);
    }// HINT: some C string primitives you should define and use
    static int strlen(const char *s);
    static char * strcpy(char *dest, const char *src);
    static char * strcat(char *dest, const char *src);
    static int strcmp(const char *left, const char *right);
    static int strncmp(const char *left, const char *right, int
n);
    static char * strchr(const char *str, int c);
    static char * strstr(const char *haystack, const char
*needle);
    char buf[MAXLEN]; // array for the characters in this string
    // DO NOT store the 'logical' length of this string
    // use the null '\0' terminator to mark the end
  };
  ostream & operator << (ostream & out, String str);
  istream & operator >> (istream & in, String & str);
```

- (2 points) Write a main function which tests each public method defined in your class String. Give at least two and at most 4 tests for each method. Good organization would be something like the following:

```
void test_constructor_and_print()
{
     String s("Hello World");
     cout << s << endl;
}
int main()
{
     test_constructor_and_print();
     test_assignment();
     // ...
}
```

NOTE: You should think how you are going to get from an empty class definition to one that is fully functional. The foolish approach is to code it all up, then start debugging. In other words, you may think twice if you don't have a working program until it is fully complete. Instead, the best approach is a combination of "test driven development" and "incremental development" where you identify the minimum functionality you need to test. Then, you write a test case for that. Then, implement the functionality until the test performs correctly. Finally, you identify the next feature to add, write the test, implement the feature, test, etc.

For example, if you are writing class string, to write the simplest test program, you need functioning constructors and a functioning print method. First write a simple main function that declares a string and initializes it via the constructor, then print out the string to see if it was constructed correctly. Now go write the constructors and print and operator <<. To write the constructors, you need the utility function, strcpy(), which is defined as a static method (static means it has no this parameter, but is still a member function so it can access private parts of strings). Test your program until it is functioning correctly. Note, you may also need the copy constructor if any strings are passed as parameters by copy or returned by copy (as opposed to by reference). The copy constructor for a class that takes one parameter that is a reference to an instance of the class. It may or may not be const as well. Constructors are responsible for building this string to look just like the C-string or string object they are being constructed from. strcpy() should do the work. Both constructors should call strcpy to copy the characters from the parameter into this buf.

Next, you can think about writing operator = or operator ==, but whatever you decide to write next, first add the test case to your main function, then implement the new function and test it. The relational and equality operators can use strcmp() to do the hard work. You can read about any of these functions via google because they are standard C functions, but we are writing them ourselves to use in implementing our string class.
One more major suggestion, do not call strlen() unless absolutely necessary. It is expensive, O(N), and is not the way to iterate through a C-string with for loops. Instead iterate through C-strings using the test for the null terminating character. That is the correct and efficient way to do it.