C++ Programming
Lab 3

- (8 points) Write a String class which will be a wrapper class to the C style strings. The strings will be of varying lengths and must grow and shrink as necessary. Your String class must implement all the appropriate methods including constructors/destructor, assignment, equality operators, the index operator, reverse, indexOf (find), print, and read. Do not use any of the C *str* functions (e.g. strcmp, or strcpy), however write them yourself, as static methods, then use them.

  NOTE: You will use dynamic allocation (by calling new[]) and you must delete[] the storage you allocate when you no longer need it. strdup() will allocate the right amount of storage and make an exact copy of the string passed in as argument, then return the address of that new storage. It will be useful for implementing many methods including the constructors.
  NOTE: Some of your methods will work find using the implementation from lab 2, but it is up to you to figure out which ones are ok and which must be re-written.
  NOTE: This week, you are also splitting class string into two files with only class and function declarations in String.h file and all method and function definitions in String.cpp file.
- Class String declaration:

```
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);
```

```
    }
    static int strlen(const char *src);
    static char *strdup(const char *src); // notice this new function
    // … and any other auxiliary static methods you need
    char * buf; // base of array for the characters in this string
    // DO NOT add a length data member!! Use the null terminator.
};
ostream & operator << (ostream & out, String str);
istream & operator >> (istream & in, String & str);
```

- (2 points) Write a main function which tests each function defined in your class String. You may reuse part or all of your main from the previous homework if it is good. Write two functions, new_char_array(int n_bytes), and delete_char_array(char *p), which keep track of the number of array allocations in a static counter initialized to zero (see the lecture slides for an example). Call these functions instead of new and delete. Print out the value of your allocation counter at the end of your main function. It must be zero. Use this format: ”Number of new allocations minus number of delete deallocations is 0”

- File Organization: Put your declaration of class String in a file String.h and put all the definitions of the member functions and operator << and operator >> in String.cpp. Put your main function and any testing functions in a file named test_string.cpp. Put the main program below in a file named standard_main.cpp and include your String.h in all the .cpp files. You should be able to comple and run two programs: test_string and standard_main.

Specifics:

For your report, run both programs (test_string and standard_main) using valgrind to show that they function correctly and that they have no memory leaks.

Put all your files in *lab3.zip* and write your report in a file *report3.doc*.
Then, upload these two files to the EEE dropbox for lab 3.

//standard_main.cpp for Lab 3

```
#include "String.h"

int main()
{
        String firstString("First");
        String secondString("Second");
        String thirdString(firstString);
        String fourthString("Fourth");
        String fifthString = String();

        cout << "+: " << firstString + secondString << endl;
        cout << "+=: " << (firstString += secondString) << endl;
        cout << "indexOf(String): " << firstString.indexOf(secondString) << endl;
```

```
	cout << "indexOf(char): " << firstString.indexOf('t') << endl;
	cout << "LT: " << (secondString < firstString) << endl;
	cout << "GT: " << (secondString > firstString) << endl;
	cout << "LE: "<< (secondString <= firstString) << endl;
	cout <<"GE: "<< (secondString >= firstString) << endl;
	cout << "<<: "<< fifthString << endl;
	cout << "<<: "<< fourthString << endl;
	cout << "==: " << (fifthString == fourthString) << endl;
	cout << "indexOf(String): "<< fourthString.indexOf(fifthString) << endl;
	cout << "size(): "<< fifthString.size() << endl;
	cout << "size(): "<< fourthString.size() << endl;
	cout << "[]: "<< thirdString[1] << endl;
	cout << "reverse(): "<< fourthString.reverse() << endl;
	fifthString = thirdString;
	cout << "<<: " << fifthString << " " << thirdString << endl;
	cout << "[]: "<< fifthString[1] << endl;
	cout << "[]: "<< fifthString[10] << endl;
	cout <<"!=: " << (fifthString != thirdString) << endl;

	cout << "Enter a test string: ";
	cin >> firstString;
	cout << firstString << endl;

	cout << (firstString < secondString) << endl;
	cout << (firstString <= thirdString) << endl;
	cout << (firstString > fourthString) << endl;
	cout << (firstString >= fifthString) << endl;

	return 0;
}
```