ICS 46
Homework Assignment 1

Programming Assignment:

- (70 points) Implement class MyString as started below. This class defines a sequence of characters (similar to class string). The implementation must be as an array of characters that are dynamically allocated and resized as necessary. You must use a null character to terminate the string - just like C-strings are null terminated.
- (30 points) Implement a driver to test MyString for correct function. I have started it for you below, but be sure you test all methods implemented in class MyString. YOU MUST NOT ADD ANY DATA MEMBERS. NOTE: if you copy and paste this document, the C++ may have some syntax errors because << and >> are translated to quote characters among other things.
- Write the time T(N) of each function or method in a comment above each one. N is number of characters in the string argument. E.g.,

```
// T(N) = 3 + 3N
int strlen(char *s)
{
        int len = 0;
        for (int i=0; s[i] != '\0'; ++i)
           ++len;
        return len;
}
```

- Submit two things: 1) a single report in .pdf format that shows your program works correctly when run under valgrind. Your report must follow the specified format, then submitted to GradeScope. 2) You must also submit a zip of your program which includes Makefile that builds your program called testMystring by compiling testMystring.cpp with the required warning flags.

Homeworks that are not formatted nicely (better than the example below) will not be accepted. Pay careful attention to memory management. Do not share memory across objects. Obvious code copying, from a textbook, from another student, or within your own program, will be down-graded (write everything once (perhaps by writing auxillary functions), then call the function or method each time that operation is required). Feel free to add additional functions or methods, but please do not add any more data members to class MyString (such as an int to store the logical length of the character string).

Use **for** loops instead of **while** loops where appropriate. You may implement this homework in one file, e.g., MyString.cpp, or you may separate the entire definition of class MyString into a file called MyString.h, then put the testing functions in a file called testMyString.cpp and have that file include MyString.h. Either file structure is acceptable for this homework assignment. If you really want to separate class MyString into two files (.h and .cpp) you may do that also.

```
#include <iostream>
#include <fstream> // for doing file I/O
using namespace std;

// Class MyString defines a class similar to "std::string"

class MyString
{
        char * buf; // points to the array holding the characters in this MyString object.
        // Must be allocated in constructor and, maybe in other functions as well
```

```
    // This C-string must be null termiated and that is how you know the length
    // strlen() will count the number of characters up to the first null character

    // utility method to handle errors, note throws an exception to prevent
    // access to undefined values.  Always call this method when you detect a fatal error from
    // any of your methods.
    void error(const char *s)
    {
        cerr << "Error: " << s << endl;
        throw 0;
    }

public:

    explicit MyString( const char * s = "")
    // constructs this object from a c-string s (s is an array of char terminated by '\0')
    // parameter, s, defaults to the empty string ""
    // write and use strdup() to implment this constructor,
    // it allocates a new array, then uses strcpy() to copy chars from array s to the new array
    {
        // TBD
    }

    // copy constructor for a MyString, must make a deep copy of
    // s for this.  You can use strdup() you wrote
    MyString( const MyString & s )
    {
        // TBD
    }

    // assigns this MyString from MyString s, MUST DO DEEP ASSIGNMENT
    // remember, both this and s have been previously constructed
    // so they each have storage pointed to by buf
    MyString & operator = ( const MyString & s )
    {
            // TBD
    }

    // return a reference to the char at position index, 0 is the first element and so on
    // index must be in bounds (see method above)
    char & operator [] ( const int index )
    {
       // TBD
    }

    int length() const
    {
        // TBD
    }
```

```
// returns the index of the first occurance of c in this MyString
// indices range from 0 to length()-1
// returns -1 if the character c is not in this MyString
int indexOf( char c ) const
{
        // TBD
}

// returns the index of the first occurance of pat in this MyString
// indices range from 0 to length()-1
// returns -1 if the character string pat is not in this MyString
// write and use strstr() to implement this function
int indexOf( const String & pat ) const
{
        // TBD
}

// True if the two MyStrings contain the same chars in same position
// e.g, "abc"=="abc" returns true
// write and use strcmp() to implement this function
bool operator == ( const MyString & s ) const
{
    // TBD
}

// concatenates this and s to make a new MyString
// e.g., "abc"+"def" returns "abcdef"
// write and use str2dup() to implement this function, it should allocate a new array then call strcpy() and
strcat()
MyString operator + ( const MyString & s ) const
{
    // TBD
}

// concatenates s onto end of this
// e.g., s = "abc"; s+="def" now s is "abcdef"
// use str2dup()
MyString & operator += ( const MyString & s )
{
        // TBD
}

// returns another MyString that is the reverse of this MyString
// e.g., s = "abc"; s.reverse() returns "cba"
// write strrev(char *dest, char *src) like strcpy but copies the reverse of src into dest, then use it
MyString reverse() const
{
        // TBD
}
```

```
        // prints out this MyString to the ostream out
        void print( ostream & out ) const
        {
                // TBD
        }

        // reads a word from the istream in and this MyString
        // becomes the same as the characters in that word
        // use getline() to implement read()
        void read( istream & in )
        {
                // TBD
        }

        // destruct a MyString, must free up each node in the head list
        ~MyString()
        {
           // TBD
        }
};

// these two I/O methods are complete as long as you define print and read methods correctly
inline ostream & operator << ( ostream & out, const MyString & str )
{
    str.print(out);
    return out;
}

inline istream & operator >> ( istream & in, MyString & str )
{
    str.read(in);
    return in;
}

// TBD: Write all these testing functions and add more of your own
// follow my example and write a function to test each method.
// Name each of these functions so they clearly indicate what they are testing

MyString copyConstructorTest(MyString l)
{
    return l;
}

void testReverse()
{
    ifstream in("input.txt");
    MyString l;

    while ( in >> l )
    {
```

```
        cout << copyConstructorTest(l)
             << " " << l.length() << " "
             << l.reverse() << endl;
    }
}

// write more test functions here
// ...

int main()
{
    // This try/catch will catch any exception thrown by MyString::error()
    // while calling test functions.
    try {
        testReverse();
       // put more test function calls here written just above
    }
    catch (int i) {
       cout << "Got an exception: " << i << endl;
    }
    cerr << "Net memory allocated at program end: " << NumAllocations << endl;
    cerr << "(should be zero! positive = memory leak, negative = duplicate delete)\n";
    return 0;
}
```