# Assignment 1 – CMPT 135
**Deadline: Feb. 13, 2019 at 11:55PM**
**You can work in groups of two.**

Read this document in its entirety and carefully before you start anything and understand it. If you have any questions, don't hesitate to email me.

## Problem Statement

In this assignment, you will implement a class named **CMPT135_String** that will mimic the C++ string class. The aim is for you to apply what you learned in class regarding

- Design of classes
- Constructor member functions
- Destructor member functions
- Getter member functions
- Setter member functions
- Other member functions you might need
- Member operator functions, and
- Non-member friend operator functions

For all of you to have the same class design, member function names, and operators; the declaration of the class are provided as shown below. Some of the functions might be difficult to comprehend at CMPT135 level and for that reason; I am also including the actual implementation of some of the functions. The aim is for you to implement all the functions stated in the class declaration **CMPT135_String** class. Also, a test main program is provided hat tests every detail of the **CMPT135_String** class to help you test your class. Your program will be tested against this main program.

## Restriction

*You are allowed to include* ***ONLY*** *#include <iostream> . You are* ***NOT*** *allowed to include anything else. You are* ***NOT*** *allowed to include #include <string> or #include <cstring> or anything else. Moreover, you are NOT allowed to change any function signature. Use the function declarations exactly as they are. If you change any function name, you will lose the marks.*

## Discussion

We would like our **CMPT135_String** class to represent strings as dynamic array of characters terminated with the NULL ('\0') character. This means, our class will have two-member variables:

**int length;**

**char *buffer;**

***The length member variable will be the number of characters in the buffer excluding the NULL character. The buffer will always have a NULL character at the end; although the NULL character is not counted for length.***

# Assignment 1 – CMPT 135
**Deadline: Feb. 13, 2019 at 11:55PM**
**You can work in groups of two.**

```
#include <iostream>
using namespace std;

class CMPT135_String
{
private:
        static const char NULL_TERMINATOR = '\0';  //Instead of writing '\0' many times, it is better to have a constant for it
        int length;          //This will be the length of the string without the NULL character
        char *buffer;        //This buffer will actually have length + 1 characters including the NULL char
private:
        char *getBuffer() const;    //A private member function to return the buffer itself (not a copy of it)
public:
        //Constructors
        CMPT135_String();
        CMPT135_String(const char &c);
        CMPT135_String(const char *buffer);
        CMPT135_String(const CMPT135_String &s);

        //Destructor
        ~CMPT135_String();

        //Getter member functions
        int getLength() const;
        char getCharAt(const int &index) const;

        //Setter member functions
        void setCharAt(const int &index, const char &c);
        void append(const char &c);
        void setBuffer(const char *buffer);

        //Other member functions

        //Operator member functions
        CMPT135_String& operator = (const CMPT135_String &s);
        CMPT135_String& operator = (const char &c);
        CMPT135_String& operator = (const char* buffer);
        CMPT135_String operator + (const CMPT135_String &s) const;
        CMPT135_String operator + (const char *buffer) const;
        CMPT135_String operator + (const char &c) const;
        bool operator == (const CMPT135_String &s) const;
        bool operator == (const char *buffer) const;
        bool operator != (const CMPT135_String &s) const;
        bool operator != (const char *buffer) const;

        //Friend Functions (for operators)
        friend CMPT135_String operator + (const char *buffer, const CMPT135_String &s);
        friend CMPT135_String operator + (const char &c, const CMPT135_String &s);
        friend bool operator == (const char *buffer, const CMPT135_String &s);
        friend bool operator != (const char *buffer, const CMPT135_String &s);
        friend ostream& operator << (ostream &outputStream, const CMPT135_String &s);
        friend istream& operator >> (istream &inputStream, CMPT135_String &s);
};
```

**Assignment 1 – CMPT 135**

**Deadline: Feb. 13, 2019 at 11:55PM**

**You can work in groups of two.**

## Explanation of the functions

**1. CMPT135_String();**

This is a default constructor. This should set the **length** to **zero**. It should create a **buffer** dynamic array of size **1** and put the NULL character at index 0. Here you can take advantage of the declared variable `static const char NULL_TERMINATOR = '\0';` as follows: **this->buffer[0] = this->NULL_TERMINATOR;** Please note that, while the buffer dynamic array has size of 1 where the NULL character is stored at index 0, the **length** member variable is set to zero because the length member variable does not count the NULL character. To have a warm up for the assignment, the actual implementation for this constructor is provided:

```
CMPT135_String :: CMPT135_String() {
	this->length = 0;
	this->buffer = new char[this->getLength() + 1];
	this->buffer[this->getLength()] = this->NULL_TERMINATOR;
}
```

**2. CMPT135_String(const char &c);**

This constructor takes one-character type argument and creates a buffer of size 2, sets the element of the buffer at index 0 to the character argument, while the element at index 1 to the NULL character.

**3. CMPT135_String(const char *buffer);**

This constructor takes a NULL terminated character array which is a C-String we learned in CMPT 130 and then puts the characters in the argument to the buffer member variable by first creating suitable size buffer member variable. It also sets the last character of the buffer member variable to NULL. Moreover, the length member variable is set to number of characters copied.

**4. CMPT135_String(const CMPT135_String &s);**

Here you should first create a suitable size buffer member variable, copy the characters of the argument to the buffer member variable, add a NULL character at the end of the buffer member variable and finally set the length member variable appropriately.

**5. ~CMPT135_String();**

This function should delete the buffer member variable and initialize the length to 0.

**6. int getLength() const;**

This function should return the length member variable.

**7. char getCharAt(const int &index) const;**

This function should return the character at the given index in the buffer **only** if index is greater than or equal to zero and less than length. Otherwise, it must return NULL character.

**8. char *getBuffer() const;**

This is a private member variable. It returns the buffer (not a copy of it). As a warm up for the assignment, I am providing the actual implementation of this function.

```
char* CMPT135_String :: getBuffer() const {
	//This function is designed mainly for internal use.
	//It returns a pointer to the buffer of the class.
	//It doesn't make a new copy of the buffer.
	return this->buffer;
}
```

**9. void setCharAt(const int &index, const char &c);**

This function should modify the character at the given index in the buffer **only** if index is greater than or equal to zero and less than length. Otherwise it should do nothing.

**10. void append(const char &c);**

This function must make the buffer one more in size and then append the argument character to the characters in the buffer, making sure the NULL character is always at the end of the buffer and the length of member variable is adjusted correctly.

**11. void setBuffer(const char *buffer);**

This function changes the buffer of a **CMPT135_String** object. This means you must also adjust the length accordingly. The argument buffer always has a NULL character the end. Most importantly, this function must be implemented with caution with regards to memory leak. When a **CMPT135_String** object calls this function, remember that the calling object may already have a dynamic array allocated to its buffer member variable. Therefore, you need to clean the existing buffer before copying the argument buffer to it.

**12. CMPT135_String& operator = (const CMPT135_String &s);**

This operator has **CMPT135_String** objects on both sides. It assigns the right-hand side operand (that is the argument in this function) to the left-hand side operand (which is the calling object). You must return ***this** for the class to have chain assignment capability.

**13. CMPT135_String& operator = (const char &c);**

This operator has a **CMPT135_String** object on the left-hand side and a character on the right-hand side. It assigns the right-hand side operand (that is the argument in this function) to the left-hand side operand (which is the calling object). You must return ***this** for the class to have chain assignment capability.

**14. CMPT135_String& operator = (const char* buffer);**

This operator has a **CMPT135_String** object on the left-hand side and a NULL terminated character array on the right-hand side. It assigns the right-hand side operand (that is the argument in this function) to the left-hand side operand (which is the calling object). You must return ***this** for the class to have chain assignment capability.

**15. CMPT135_String operator + (const CMPT135_String &s) const;**

This operator has **CMPT135_String** objects on both sides. It creates a new **CMPT135_String** object and assigns the new object buffer all the characters in the buffers of the left-hand side operand (which is the calling object) and the right-hand side operand (that is the argument in this function). You must return newly created **CMPT135_String** object.

**16. CMPT135_String operator + (const char *buffer) const;**

This operator has **CMPT135_String** object on the left-hand side and a NULL terminated character array on the right-hand side. It creates a new **CMPT135_String** object and assigns the new object buffer all the characters in the buffer of the left-hand side operand (which is the calling object) and the right-hand side operand (that is the argument in this function). You must return newly created **CMPT135_String** object.

**17. CMPT135_String operator + (const char &c) const;**

This operator has **CMPT135_String** object on the left-hand side and a single character on the right-hand side. It creates a new **CMPT135_String** object and assigns the new object buffer all the characters in the buffer of the left-hand side operand (which is the calling object) and the character on the right-hand side of the operand (that is the argument in this function). You must return newly created **CMPT135_String** object.

**18. bool operator == (const CMPT135_String &s) const;**

This operator has **CMPT135_String** objects on both sides. It compares the characters in the buffer of the left-hand side operand (which is the calling object) to the characters in the buffer of the right-hand side operand (that is the argument in this function). It returns true if both objects have the same length and their corresponding characters are equal; otherwise it returns false.

**19. bool operator == (const char *buffer) const;**

This operator has **CMPT135_String** object on the left-hand side and a NULL terminated character array on the right-hand side. It compares the characters in the buffer of the left-hand side operand (which is the calling object) to the characters in the buffer on the right-hand side of the operand (which is the argument in this function). It returns true if the calling object has the same number of characters as the argument buffer on the right-hand side of the operand and their corresponding characters are equal; otherwise it returns false.

**20. bool operator != (const CMPT135_String &s) const;**

This operator has **CMPT135_String** objects on both sides. It does the negation of the member operator function **bool operator == (const CMPT135_String &s) const**.

**21. bool operator != (const char *buffer) const;**

This operator has **CMPT135_String** object on the left-hand side and a NULL terminated character array on the right-hand side. It does the negation of the member operator **bool operator == (const char *buffer) const** function.

**22. friend CMPT135_String operator + (const char *buffer, const CMPT135_String &s);**

This operator has a NULL terminated character array on the left-hand side (which is the first parameter in this function) and a **CMPT135_String** object on the right-hand side (which is the second parameter in this function). It creates a new **CMPT135_String** object, sets its buffer to all the characters in the NULL terminated character array and the characters in the **CMPT135_String** object. It returns the newly created object.

**23. friend CMPT135_String operator + (const char &c, const CMPT135_String &s);**

This operator has a single character on the left-hand side (which is the first parameter in this function) and a **CMPT135_String** object on the right-hand side (which is the second parameter in this function). It creates a new **CMPT135_String** object, sets its buffer to the character and the characters in the **CMPT135_String** object. It returns the newly created object.

**24. friend bool operator == (const char *buffer, const CMPT135_String &s);**

This operator has a NULL terminated character array on the left-hand side and a **CMPT135_String** object on the right-hand side. It compares the characters in the NULL terminated character array on the left-hand side operand (which is the first parameter in this function) to the characters in the buffer of the right-hand side object operand (which is the second argument in this function). It returns true if the NULL terminated character array has the same number of characters as the **CMPT135_String** object on the right-hand side of the operand and their corresponding characters are equal; otherwise it returns false.

**25. friend bool operator != (const char *buffer, const CMPT135_String &s);**

This operator has a NULL terminated character array on the left-hand side and a **CMPT135_String** object on the right-hand side. It does the negation of **friend bool operator == (const char *buffer, const CMPT135_String &s)**

**26. friend ostream& operator << (ostream &outputStream, const CMPT135_String &s);**

The output stream might be better provided so that you can start testing your class right after you implement some constructors. The implementation of this function is given below:

```
ostream& operator << (ostream &outputStream, const CMPT135_String &s) {
	outputStream << s.getBuffer();
	return outputStream;
```

}

**27. friend istream& operator >> (istream &inputStream, CMPT135_String &s);**

The input stream is also provided it might be a little too much for you to figure it out.

```
istream& operator >> (istream &inputStream, CMPT135_String &s) {
        /* This function is designed to read ANY length input string from the user.
        In order to achieve this, we will read character by character until end of
        line character ('\n') is read. The trouble will be where to store unknown
        length input. For this, we will create a char array (named buffer1) of length
        100. The hope is, the input won't exceed 1000 chars. However if it does,
        we will copy the 100 chars to a temp char array, make the buffer1 array of
        twice its current length, copy the characters in the temp array to buffer1
        and continue reading; applying same principle if buffer1 gets filled again.
        Next, we will copy the exact characters read to another buffer (named buffer2)
        and use that to set the buffer of CMPT135_String object parameter. Finally,
        we will do clean up of unnecessary dynamic arrays */
        int bufferCurrentLength = 100;
        char *buffer1 = new char[bufferCurrentLength];
        int length = 0;
        char c;
        while (1) {
                inputStream.get(c);
                if (c == '\n')
                        break;
                buffer1[length++] = c;
                //Now, check in case buffer is filled
                if (length == bufferCurrentLength) {
                        //Copy current buffer to temp array
                        char *temp = new char[bufferCurrentLength];
                        for (int i = 0; i < bufferCurrentLength; i++)
                                temp[i] = buffer1[i];
                        //Delete current buffer
                        delete[] buffer1;
                        //Re-create current buffer with twice size
                        buffer1 = new char[2*bufferCurrentLength];
                        //Copy characters from temp array to current buffer
                        for (int i = 0; i < bufferCurrentLength; i++)
                                buffer1[i] = temp[i];
                        //Adjust the size of the current buffer
                        bufferCurrentLength *= 2;
                        //Delete the temporary array
                        delete[] temp;
                }
        }
        //Create a buffer of exact size for the characters read
        char *buffer2 = new char[length + 1];
        //Copy chars in current buffer to the exact size buffer
        for (int i = 0; i < length; i++)
                buffer2[i] = buffer1[i];
        //Append the a NULL character to the exact size buffer
        buffer2[length] = '\0';
        /* Set the exact size buffer as the buffer of the CMPT135_String
        object being read. The set buffer function will take care of
        adjusting the length of the object. Also the setBuffer function
        will first clean the current buffer for us. */
        s.setBuffer(buffer2);
        //Clean up the current buffer and the exact size buffer
        delete[] buffer1;
        delete[] buffer2;
        //Return the input stream
        return inputStream;
}
```

**Assignment 1 – CMPT 135**

**Deadline: Feb. 13, 2019 at 11:55PM**

**You can work in groups of two.**

## Testing Your class

To have the same testing main program, the main program you need to test your class is provided as well as a sample output.

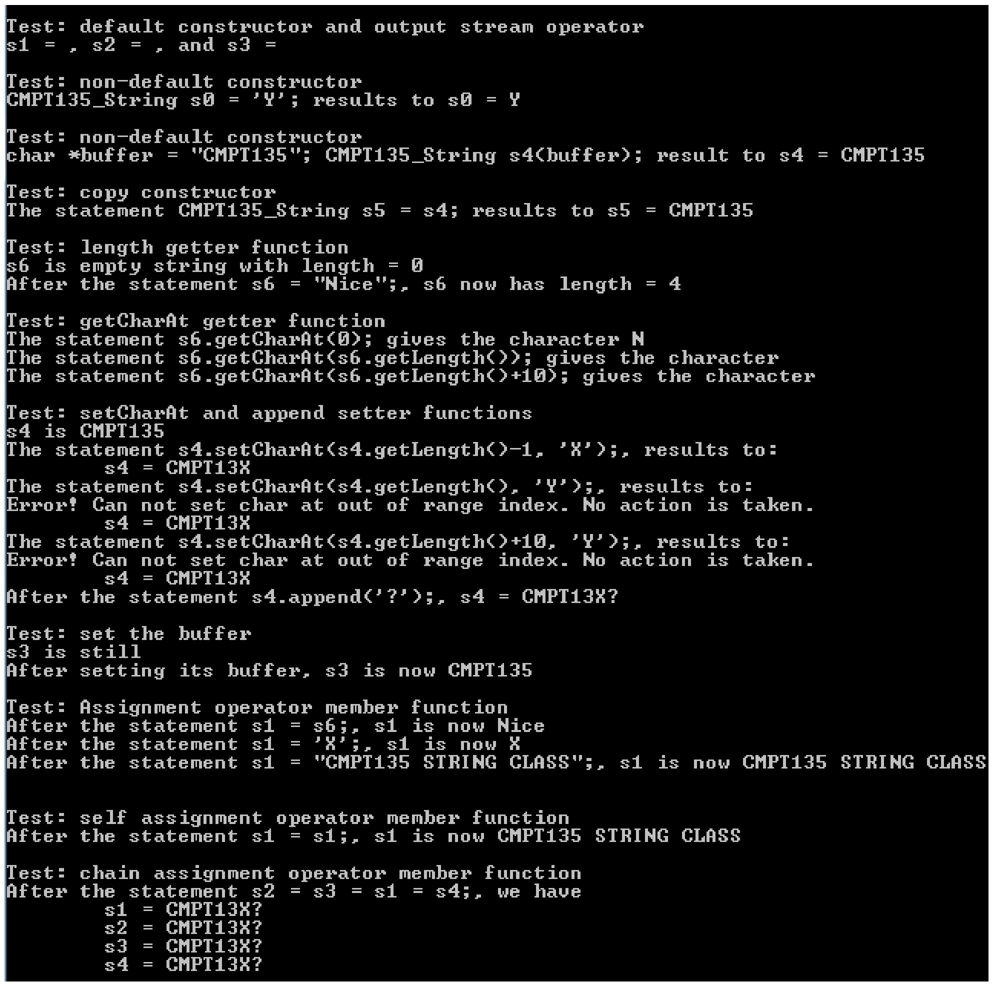
```
#include <iostream>
using namespace std;
int main() {
        cout << "\tFraser International College, CMPT 135 2018-02" << endl;
        cout << "\tAssignment 1 Test Bed" << endl;
        cout << "\t=============================================" << endl;
        cout << endl << endl;
        cout << "Test: default constructor and output stream operator" << endl;
        CMPT135_String s1, s2, s3;
        cout << "s1 = " << s1 << ", s2 = " << s2 << ", and s3 = " << s3 << endl;
        cout << endl;
        cout << "Test: non-default constructor" << endl;
        CMPT135_String s0 = 'Y';
        cout << "CMPT135_String s0 = 'Y'; results to s0 = " << s0 << endl;
        cout << endl;
        cout << "Test: non-default constructor" << endl;
        char *buffer = "CMPT135"; // might need to replace this with const char *buffer = "CMPT135";
        CMPT135_String s4(buffer);
        cout << "char *buffer = \"CMPT135\"; CMPT135_String s4(buffer); result to s4 = " << s4 << endl;
        cout << endl;
        cout << "Test: copy constructor" << endl;
        CMPT135_String s5 = s4;
        cout << "The statement CMPT135_String s5 = s4; results to s5 = " << s5 << endl;
        cout << endl;
        cout << "Test: length getter function" << endl;
        CMPT135_String s6;
        cout << "s6 is empty string with length = " << s6.getLength() << endl;
        s6 = "Nice";
        cout << "After the statement s6 = \"Nice\";, s6 now has length = " << s6.getLength() << endl;
        cout << endl;
        cout << "Test: getCharAt getter function" << endl;
        cout << "The statement s6.getCharAt(0); gives the character " << s6.getCharAt(0) << endl;
        cout << "The statement s6.getCharAt(s6.getLength()); gives the character "
             << s6.getCharAt(s4.getLength()) << endl;
        cout << "The statement s6.getCharAt(s6.getLength()+10); gives the character "
             << s6.getCharAt(s4.getLength()+10) << endl;
        cout << endl;
        cout << "Test: setCharAt and append setter functions" << endl;
        cout << "s4 is " << s4 << endl;
        cout << "The statement s4.setCharAt(s4.getLength()-1, 'X');, results to:" << endl;
        s4.setCharAt(s4.getLength()-1, 'X');
        cout << "\ts4 = " << s4 << endl;
        cout << "The statement s4.setCharAt(s4.getLength(), 'Y');, results to:" << endl;
        s4.setCharAt(s4.getLength(), 'Y');
        cout << "\ts4 = " << s4 << endl;
        cout << "The statement s4.setCharAt(s4.getLength()+10, 'Y');, results to:" << endl;
        s4.setCharAt(s4.getLength()+10, 'Y');
        cout << "\ts4 = " << s4 << endl;
        s4.append('?');
        cout << "After the statement s4.append('?');, s4 = " << s4 << endl;
        cout << endl;
        cout << "Test: set the buffer" << endl;
        cout << "s3 is still " << endl;
        s3.setBuffer(buffer);
        cout << "After setting its buffer, s3 is now " << s3 << endl;
        cout << endl;
        cout << "Test: Assignment operator member function" << endl;
        s1 = s6;
        cout << "After the statement s1 = s6;, s1 is now " << s1 << endl;
        s1 = 'X';
```

```
cout << "After the statement s1 = 'X';, s1 is now " << s1 << endl;
s1 = "CMPT135 STRING CLASS";
cout << "After the statement s1 = \"CMPT135 STRING CLASS\";, s1 is now " << s1 << endl;
cout << endl;
cout << "Test: self assignment operator member function" << endl;
s1 = s1;
cout << "After the statement s1 = s1;, s1 is now " << s1 << endl;
cout << endl;
cout << "Test: chain assignment operator member function" << endl;
s2 = s3 = s1 = s4;
cout << "After the statement s2 = s3 = s1 = s4;, we have" << endl;
cout << "\ts1 = " << s1 << endl;
cout << "\ts2 = " << s2 << endl;
cout << "\ts3 = " << s3 << endl;
cout << "\ts4 = " << s4 << endl;
cout << endl;
cout << "Test: concatenation member operator function" << endl;
s1 = s1 + s3;
cout << "s1 = s1 + s3 results to s1 = " << s1 << endl;
s1 = s1 + "YES";
cout << "s1 = s1 + \"YES\" results to s1 = " << s1 << endl;
s1 = s1 + '6';
cout << endl;
cout << "Test: equality test member operator function" << endl;
if (s2 == s3 && s3 == s4)
        cout << "All s2, s3, and s4 are equal." << endl;
else
        cout << "NOT all s2, s3, and s4 are equal." << endl;
if (s3 == "CMPT13X?")
        cout << "The expression s3 == \"CMPT13X?\" evaluated to TRUE." << endl;
else
        cout << "The expression s3 == \"CMPT13X?\" evaluated to FALSE." << endl;
cout << endl;
cout << "Test: inequality test member operator function" << endl;
if (s1 != s4)
        cout << "The expression s1 != s4 evaluated to TRUE" << endl;
else
        cout << "The expression s1 != s4 evaluated to FALSE" << endl;
if (s3 != "CMPT13X?")
        cout << "The expression s3 != \"CMPT13X?\" evaluated to TRUE." << endl;
else
        cout << "The expression s3 != \"CMPT13X?\" evaluated to FALSE." << endl;
cout << endl;
cout << "Test: concatenation non-member operator friend function" << endl;
s2 = "YES" + s1;
cout << "s2 = \"YES\" + s1 results to s2 = " << s2 << endl;
s2 = 'i' + s2;
cout << "s2 = 'i' + s2 results to s2 = " << s2 << endl;
cout << endl;
cout << "Test: equality test non-member operator friend function" << endl;
if ("CMPT13X?" == s3)
        cout << "The expression \"CMPT13X?\" == s3 evaluated to TRUE." << endl;
else
        cout << "The expression \"CMPT13X?\" == s3 evaluated to FALSE." << endl;
cout << endl;
cout << "Test: inequality test non-member operator friend function" << endl;
if ("CMPT13X?" != s3)
        cout << "The expression \"CMPT13X?\" != s3 evaluated to TRUE." << endl;
else
        cout << "The expression \"CMPT13X?\" != s3 evaluated to FALSE." << endl;
cout << endl;
cout << "Test: input and output stream non-member operator friend functions" << endl;
cout << "Please enter a string s1: ";
cin >> s1;
cout << "After reading s1 from the user, s1 is now " << s1 << endl;
cout << endl;
system("Pause");
```

```
        return 0;
}
```

## Sample Output

For the main program given above, the output is:

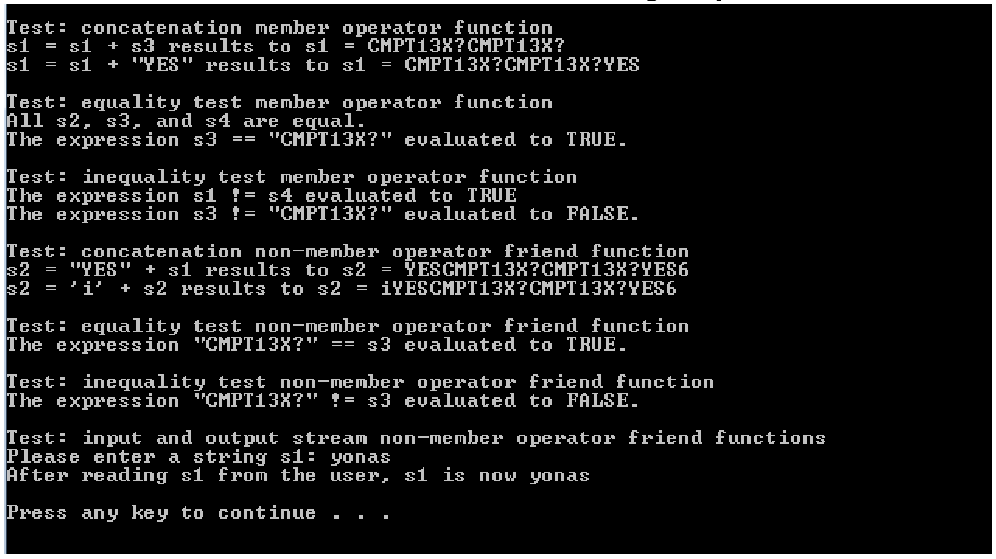


```
Test: default constructor and output stream operator
s1 = , s2 = , and s3 =

Test: non-default constructor
CMPT135_String s0 = 'Y'; results to s0 = Y

Test: non-default constructor
char *buffer = "CMPT135"; CMPT135_String s4(buffer); result to s4 = CMPT135

Test: copy constructor
The statement CMPT135_String s5 = s4; results to s5 = CMPT135

Test: length getter function
s6 is empty string with length = 0
After the statement s6 = "Nice";, s6 now has length = 4

Test: getCharAt getter function
The statement s6.getCharAt(0); gives the character N
The statement s6.getCharAt(s6.getLength()); gives the character
The statement s6.getCharAt(s6.getLength()+10); gives the character

Test: setCharAt and append setter functions
s4 is CMPT135
The statement s4.setCharAt(s4.getLength()-1, 'X');, results to:
        s4 = CMPT13X
The statement s4.setCharAt(s4.getLength(), 'Y');, results to:
Error! Can not set char at out of range index. No action is taken.
        s4 = CMPT13X
The statement s4.setCharAt(s4.getLength()+10, 'Y');, results to:
Error! Can not set char at out of range index. No action is taken.
        s4 = CMPT13X
After the statement s4.append('?');, s4 = CMPT13X?

Test: set the buffer
s3 is still
After setting its buffer, s3 is now CMPT135

Test: Assignment operator member function
After the statement s1 = s6;, s1 is now Nice
After the statement s1 = 'X';, s1 is now X
After the statement s1 = "CMPT135 STRING CLASS";, s1 is now CMPT135 STRING CLASS


Test: self assignment operator member function
After the statement s1 = s1;, s1 is now CMPT135 STRING CLASS

Test: chain assignment operator member function
After the statement s2 = s3 = s1 = s4;, we have
        s1 = CMPT13X?
        s2 = CMPT13X?
        s3 = CMPT13X?
        s4 = CMPT13X?
```

**Assignment 1 – CMPT 135**

**Deadline: Feb. 13, 2019 at 11:55PM**

**You can work in groups of two.**



```
Test: concatenation member operator function
s1 = s1 + s3 results to s1 = CMPT13X?CMPT13X?
s1 = s1 + "YES" results to s1 = CMPT13X?CMPT13X?YES

Test: equality test member operator function
All s2, s3, and s4 are equal.
The expression s3 == "CMPT13X?" evaluated to TRUE.

Test: inequality test member operator function
The expression s1 != s4 evaluated to TRUE
The expression s3 != "CMPT13X?" evaluated to FALSE.

Test: concatenation non-member operator friend function
s2 = "YES" + s1 results to s2 = YESCMPT13X?CMPT13X?YES6
s2 = 'i' + s2 results to s2 = iYESCMPT13X?CMPT13X?YES6

Test: equality test non-member operator friend function
The expression "CMPT13X?" == s3 evaluated to TRUE.

Test: inequality test non-member operator friend function
The expression "CMPT13X?" != s3 evaluated to FALSE.

Test: input and output stream non-member operator friend functions
Please enter a string s1: yonas
After reading s1 from the user, s1 is now yonas

Press any key to continue . . .
```

## Submission Format

You are required to submit your program (only the C++ file) online through Moodle. You will find a submission button for Assignment 1 on Moodle on Week 4 and you are required to upload your program written as C++ class declaration, member function definitions, friend functions definitions and main program.

## Submission Deadline

The deadline to upload your program online is **Feb. 13, 2019 at 11:55PM**. Moodle will not allow you to upload after this date and time.

## Marking

A non-working program will automatically get zero. A program that works but doesn't give right output or gives partial right output will lose marks depending how severe its shortcoming is. This assignment carries 4% of the total course marks.

Good Luck!

- Credit for designing this assignment goes to Dr. Yonas T. Weldeselassie.