CMPT 135 - FIC 202301 - Assignment 1

Due Date: Tuesday 21 February 2023 11:55PM

Instructor: Dr. Yonas T. Weldeselassie (Ph.D.)

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 have learned in the course regarding

- Design of classes,
- Constructor member functions,
- Destructor member function,
- Assignment operators,
- Getter member functions,
- Setter member functions,
- Member operator functions,
- Non-member friend operator functions, and
- Other non-member and non-friend functions.

In order for all of you to have the same class design, that is to say the same member, friend, and non-member and non-friend functions signatures; I am providing the declaration of the class and description of the non-member and non-friend functions as shown below. The aim is for you to implement the functions listed in the class declaration and the non-member and non-friend functions. Also, I am providing a test program and its output that minimally tests the functionality of the **CMPT135_String** class and the non-member and non-friend functions in order to help you test your work.

Discussion

We would like our **CMPT135_String** class to represent strings as null terminated character arrays. This means our class will have **ONLY ONE** member variable which is a pointer to char data type. Most importantly, in order to avoid any memory leak, an empty **CMPT135_String** object will be represented by assigning the pointer member variable a special C++ pointer value named **`nullptr`**. That is an empty **CMPT135_String** object will have NO memory allocated to it whatsoever (not even for a terminating null character). Of course any non-empty **CMPT135_String** object will have memory allocated for all of its printable characters and a null terminating character.

In addition, in order to help us compute the length of **CMPT135_String** objects, we will provide a static member function that counts the number of printable characters in a null terminated character array. This function will be designed taking into account a pointer with a nullptr value represents an empty **CMPT135_String** object; hence its length is zero. Moreover a null terminated character array with only one byte allocated memory that holds a null character will also have length zero.

Finally, the destructor member function will be designed in such a way that after destruction a **CMPT135_String** object will have its pointer member variable assigned a nullptr value. This will avoid any memory leak when objects go out of scope.

Starting Your Work

In order for you to easily copy and paste the class declaration, partial definition, and the test program code, you are provided with a text file named **StarterCode.txt** that contains the class declaration, partial definition and description of each member and friend function, a test program, and a sample run output of the test program. Please copy and paste the given code from this text file into your project so that you can start your work right away.

Restriction

- *You are allowed to include **ONLY** #include <iostream>. You are **NOT** allowed to include anything else,*
- You are NOT allowed to use any cstring built-in functions such as strcpy, strcat or strlen, and
- You are not allowed to add or remove any member or friend function to the given class declaration.

In addition, you are NOT allowed to change any given function signature. Use the function declarations exactly as they are given below including the spelling and capitalization. If you change any function signature, you will lose all your marks.

Requirement

You are required to define all the missing

- Member and friend functions of the **`CMPT135_String`** class, and
- The non-member and non-friend functions described below.

You should implement your functions in such a way that the given test program runs without any syntax, linking, runtime, or semantic errors and provide the same output as the provided sample run output.

You are also required to submit an original work. Submitting any portion of work that is not original is academic misconduct and will be penalized according to the policies set out in the course outline.

Submission Format

You will find a submission button for Assignment 1 on Moodle under Week 5 and you are required to upload the source code of your program (that is .cpp file) containing the class declaration, class definition, the non-member and non-friend function definitions, and the test main program.

No email submission is allowed for whatever reason.

Submission Due Date and Time

The deadline to upload your program online is **Tuesday 21 February 2023 at 11:55 PM**. Moodle will not allow you to upload after this date and time.

Marking

Your program will be tested under Microsoft Visual C++ 2010 Express and you are advised to test your program on the same IDE before submitting your work. A program with any syntax or linking errors will automatically get zero mark. A program with runtime or semantic errors will lose marks depending on how severe its shortcoming is when it is executed.

Class Declaration

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

class CMPT135_String
{
private:
        char *buffer;  //This will be the dynamic array to hold the characters
public:
        //static member function to compute the length of null terminated char arrays
        1.  static int cstrlen(const char *cStr);

        //Default Constructor
        2.  CMPT135_String();

        //Non-default Constructors
        3.  CMPT135_String(const char &c);
        4.  CMPT135_String(const char *cStr); //*cStr is a null terminated char array

        //Copy Constructor
        5.  CMPT135_String(const CMPT135_String &s);

        //Destructor
        6.  ~CMPT135_String();

        //Assignment operators
        7.  CMPT135_String& operator = (const CMPT135_String &s);
        8.  CMPT135_String& operator = (const char &c);
        9.  CMPT135_String& operator = (const char *cStr); //*cStr is a null terminated char array

        //Getter member functions
        10. int length() const;
        11. char& operator[](const int &index) const;

        //Setter member functions
        12. void append(const CMPT135_String &s);
        13. void append(const char &c);
        14. void append(const char *cStr); //*cStr is a null terminated char array

        //Operator member functions
        15. CMPT135_String operator + (const CMPT135_String &s) const;
        16. CMPT135_String operator + (const char &c) const;
        17. CMPT135_String operator + (const char *cStr) const; //*cStr is a null terminated char array
        18. CMPT135_String& operator += (const CMPT135_String &s);
        19. CMPT135_String& operator += (const char &c);
        20. CMPT135_String& operator += (const char *cStr); //*cStr is a null terminated char array
        21. bool operator == (const CMPT135_String &s) const;
        22. bool operator == (const char *cStr) const; //*cStr is a null terminated char array
        23. bool operator != (const CMPT135_String &s) const;
        24. bool operator != (const char *cStr) const; //*cStr is a null terminated char array
        25. bool operator < (const CMPT135_String &s) const;
        26. bool operator < (const char *cStr) const; //*cStr is a null terminated char array
        27. bool operator > (const CMPT135_String &s) const;
        28. bool operator > (const char *cStr) const; //*cStr is a null terminated char array
        29. bool operator <= (const CMPT135_String &s) const;
        30. bool operator <= (const char *cStr) const; //*cStr is a null terminated char array
        31. bool operator >= (const CMPT135_String &s) const;
        32. bool operator >= (const char *cStr) const; //*cStr is a null terminated char array

        //Friend Functions (for operators)
        33. friend CMPT135_String operator + (const char &c, const CMPT135_String &s);
        34. friend CMPT135_String operator + (constchar *cStr, const CMPT135_String &s); //*cStr is a null
            terminated char array
        35. friend bool operator == (constchar *cStr, const CMPT135_String &s); //*cStr a null terminated char
            array
        36. friend bool operator != (const char *cStr, const CMPT135_String &s); //*cStr is a null terminated
            char array
        37. friend bool operator < (const char *cStr, const CMPT135_String &s); //*cStr is a null terminated
            char array
        38. friend bool operator > (constchar *cStr, const CMPT135_String &s); //*cStr is a null terminated
            char array
        39. friend bool operator <= (const char *cStr, const CMPT135_String &s); //*cStr is a null terminated
            char array
        40. friend bool operator >= (const char *cStr, const CMPT135_String &s); //*cStr is a null terminated
            char array
        41. friend ostream& operator << (ostream& outputStream, const CMPT135_String &s);
        42. friend istream& operator >> (istream& inputStream, CMPT135_String &s);
};
```

Explanation of the member and friend functions

```
1.  int CMPT135_String::cstrlen(const char *cStr)
{
        /*
        This function returns the number of printable characters in the null terminated
        character array cStr. That is, it returns the number of characters in the
        array excluding the terminating null character.

        If the pointer cStr does not have any allocated memory but instead it is
        a nullptr, then this function returns 0.

        This means there are two different cases for which this function returns 0.
        The first case is if the cStr pointer has a nullptr value and the second case
        is when cStr is a dynamic array of size 1 with cStr[0] = '\0'

        This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
        */
        if (cStr == nullptr)
                return 0;
        else
        {
                int len = 0;
                while (cStr[len] != '\0')
                        len++;
                return len;
        }
}
2.  CMPT135_String::CMPT135_String()
{
        /*
        This function constructs a default CMPT135_String object whose pointer member
        variable is initialized to nullptr.

        This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
        */
        buffer = nullptr;
}
3.  CMPT135_String::CMPT135_String(const char &c)
{
        /*
        This function constructs a non-default CMPT135_String that contains one printable
        character (which is the argument) and a null character at the end.

        This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
        */
        buffer = new char[2];
        buffer[0] = c;
        buffer[1] = '\0';
```

```
}
4.  CMPT135_String::CMPT135_String(const char *cStr) //*cStr is a null terminated char array
{
	/*
	This function constructs a non-default CMPT135_String that contains all the printable
	characters of the argument and a null character at the end.

	This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
	*/
	int len = CMPT135_String::cstrlen(cStr);
	if (len == 0)
		buffer = nullptr;
	else
	{
		buffer = new char[len+1];
		for (int i = 0; i < len; i++)
			buffer[i] = cStr[i];
		buffer[len] = '\0';
	}

}
5.  CMPT135_String::CMPT135_String(const CMPT135_String &s)
{
	/*
	This function constructs a CMPT135_String object which is a copy of the argument s
	*/
}
6.  CMPT135_String::~CMPT135_String()
{
	/*
	This function destructs a CMPT135_String object. That is it deletes its buffer and
	assigns the buffer a nullptr value

	This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
	*/
	if (buffer != nullptr)
	{
		delete[] buffer;
		buffer = nullptr;
	}
}
7.  CMPT135_String& CMPT135_String::operator = (const CMPT135_String &s)
{
	/*
	This function assigns a copy of the value of the argument s to the calling object
	*/
}
8.  CMPT135_String& CMPT135_String::operator = (const char &c)
{
	/*
	This function assigns a copy of a CMPT135_string object constructed from the character argument
	to the calling object
	*/
}
9.  CMPT135_String& CMPT135_String::operator = (const char *cStr) //*cStr is a null terminated char array
{
	/*
	This function assigns a copy of a CMPT135_string object constructed from the null terminated
	character array argumentto the calling object
	*/
}
10. int CMPT135_String::length() const
{
	/*
```

```
        This function returns the length of the buffer of the calling objects as computed by the static
        member function CMPT135_String::cstrlen

        This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
        */
        return CMPT135_String::cstrlen(buffer);
}
11. char& CMPT135_String::operator[](const int &index) const
{
        /*
        This function returns the PRINTABLE character at the given index of the calling object.
        If the index is out of the valid bounds, then error message should be printed.
        This function must crash the application using the abort built-in function
        if index out of bound error occurs.
        The valid bounds of the index are in the range [0,this->length()-1]
        */
}
12. void CMPT135_String::append(const CMPT135_String &s)
{
        /*
        This function appends all the printable characters of the argument s to the calling object.
        */
}
13. void CMPT135_String::append(const char &c)
{
        /*
        This function appends the character argument to the calling object
        */
}
14. void CMPT135_String::append(const char *cStr) //*cStr is a null terminated char array
{
        /*
        This function appends all the printable characters of the argument cStr to the calling object.
        */
}
//Operator member functions
15. CMPT135_String CMPT135_String::operator + (const CMPT135_String &s) const
{
        /*
        This function returns a CMPT135_String object constructed from all the characters of the
        calling object followed by the characters of the argument s
        */
}
16. CMPT135_String CMPT135_String::operator + (const char &c) const
{
        /*
        This function returns a CMPT135_String object constructed from all the characters of the
        calling object followed by the character argument c
        */
}
17. CMPT135_String CMPT135_String::operator + (const char *cStr) const//*cStr is a null terminated char array
{
        /*
        This function returns a CMPT135_String object constructed from all the characters of the
        calling object followed by the characters of the null terminated character array argument cStr
        */
}
18. CMPT135_String& CMPT135_String::operator += (const CMPT135_String &s)
{
        /*
        This function appends the characters of s to the calling object and then
        returns the calling object
        */
}
19. CMPT135_String& CMPT135_String::operator += (const char &c)
```

```
{
        /*
        This function appends the character c to the calling object and then
        returns the calling object
        */
}
20. CMPT135_String& CMPT135_String::operator += (const char *cStr) //*cStr is a null terminated char array
{
        /*
        This function appends the characters of the null terminated character array cStr
        to the calling object and then returns the calling object
        */
}
21. bool CMPT135_String::operator == (const CMPT135_String &s) const
{
        /*
        This function tests if the calling object is equal to the argument s
        NOTE:- Two CMPT135_String are equal if they have the same length and characters of the
        calling object are equal to the characters of the argument s at corresponding indexes.

        This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
        */
        int len = this->length();
        if (len != s.length())
                return false;
        else
        {
                for (int i = 0; i<len; i++)
                {
                        if (buffer[i] != s[i])
                                return false;
                }
                return true;
        }
}
22. bool CMPT135_String::operator == (constchar *cStr) const//*cStr is a null terminated char array
{
        /*
        This function tests if the calling object is equal to the null terminated character array cStr.
        In other words, this function tests if the calling object is equal to a CMPT135_String object
        constructed from the null terminated character array argument
        */
}
23. bool CMPT135_String::operator != (const CMPT135_String &s) const
{
        /*
        This function tests if the calling object is not equal to the argument s
        */
}
24. bool CMPT135_String::operator != (const char *cStr) const//*cStr is a null terminated char array
{
        /*
        This function tests if the calling object is not equal to the null terminated character array
        cStr
        */
}
25. bool CMPT135_String::operator < (const CMPT135_String &s) const
{
        /*
        This function tests if the calling object is less than the argument s
        NOTE:- Given two CMPT135_String objects s1 and s2, we compare them as follows:
                Step 1. Find the smallest valid index k such that s1[k] IS NOT EQUAL TO s2[k]
                Step 2. If such an index k is found, then
                        2.1 We say s1 < s2 if s1[k] < s2[k]
                        2.2 Otherwise s1 > s2
                Step 3. If such an index k is not found, then
                        3.1 We say s1 < s2 if the length of s1 is less than the length of s2
                        3.2 We say s1 > s2 if the length of s1 is greater than the length of s2
                        3.3 We say s1 == s2 if the length of s1 is equal to the length of s2.
        */
}
26. bool CMPT135_String::operator < (const char *cStr) const//*cStr is a null terminated char array
{
        /*
        This function tests if the calling object is less than the null terminated character array cStr
        */
}
27. bool CMPT135_String::operator > (const CMPT135_String &s) const
{
        /*
        This function tests if the calling object is greater than the argument s
        */
}
28. bool CMPT135_String::operator > (constchar *cStr) const//*cStr is a null terminated char array
{
        /*
        This function tests if the calling object is greater than the null terminated character array
        cStr
        */
}
29. bool CMPT135_String::operator <= (const CMPT135_String &s) const
{
        /*
        This function tests if the calling object is less than or equal to the argument s
        */
}
30. bool CMPT135_String::operator <= (const char *cStr) const//*cStr is a null terminated char array
{
        /*
        This function tests if the calling object is less than or equal to the null terminated character
        array cStr
        */
}
31. bool CMPT135_String::operator >= (const CMPT135_String &s) const
{
        /*
        This function tests if the calling object is greater than or equal to the argument s
        */
}
32. bool CMPT135_String::operator >= (const char *cStr) const//*cStr is a null terminated char array
{
        /*
        This function tests if the calling object is greater than or equal to the null terminated
        character array cStr
        */
}
33. CMPT135_String operator + (const char &c, const CMPT135_String &s)
{
        /*
        This function returns a CMPT135_String object constructed from thecharacter argument c followed
        by the characters of s
        */
}
34. CMPT135_String operator + (const char *cStr, const CMPT135_String &s) //*cStr is a null terminated char
    array
{
        /*
        This function returns a CMPT135_String object constructed from thecharacters of the null
        terminated character array argument cStr followed by the characters of s
        */
```

```
}
35. bool operator == (const char *cStr, const CMPT135_String &s) //*cStr a null terminated char array
{
        /*
        This function tests if the null terminated character array argument cStr is equal to s
        */
}
36. bool operator != (const char *cStr, const CMPT135_String &s) //*cStr is a null terminated char array
{
        /*
        This function tests if the null terminated character array argument cStr is not equal to s
        */
}
37. bool operator < (const char *cStr, const CMPT135_String &s) //*cStr is a null terminated char array
{
        /*
        This function tests if the null terminated character array argument cStr is less than s
        */
}
38. bool operator > (const char *cStr, const CMPT135_String &s) //*cStr is a null terminated char array
{
        /*
        This function tests if the null terminated character array argument cStr is greater than s
        */
}
39. bool operator <= (const char *cStr, const CMPT135_String &s) //*cStr is a null terminated char array
{
        /*
        This function tests if the null terminated character array argument cStr is less than or equal
        to s
        */
}
40. bool operator >= (const char *cStr, const CMPT135_String &s) //*cStr is a null terminated char array
{
        /*
        This function tests if the null terminated character array argument cStr is greater than or
        equal to s
        */
}
41. ostream& operator << (ostream& out, const CMPT135_String &s)
{
        /*
        This function prints the printable characters in the dynamic array member variable buffer.
        This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
        */
        for (int i = 0; i < s.length(); i++)
                out << s.buffer[i];
        return out;
}
42. istream& operator >> (istream& in, CMPT135_String &s)
{
        /*
        This function is designed to read ANY number of characters from the user.Reading will stop
        when the user presses the Enter Key. In order to achieve this, we will read one character at a
        time until end of line character ('\n') is read.
        This function is implemented by the instructor. MAKING ANY CHANGE TO IT IS NOT ALLOWED.
        */
        s.~CMPT135_String();
        char c;
        while (true)
        {
                in.get(c);
                if (c == '\n')
                        break;
                else
```

```
                    s.append(c);
        }
        return in;
}
```

In addition, you are required to define the following non-member and non-friend functions

```
1.  findCharIndex
        /*
        This function takes a CMPT135_String and a character arguments and returns the smallest valid
        index of the CMPT135_String argument such that the character of the CMPT135_String argument at
        that index is equal to the character argument. If no such an index is found, then this function
        must return -1.
        */
2.  reverseFindCharIndex
        /*
        This function takes a CMPT135_String and a character arguments and returns the largest valid
        index of the CMPT135_String argument such that the character of the CMPT135_String argument at
        that index is equal to the character argument. If no such an index is found, then this function
        must return -1.
        */
3.  countChar
        /*
        This function takes a CMPT135_String and a character arguments and returns the number of times the
        character argument is found in the CMPT135_String argument.
        */
4.  getSubString
        /*
        This function takes a CMPT135_String, a start index, and a length arguments and returns a
        CMPT135_String object with as many as length characters constructed from the characters of the
        CMPT135_String argument starting from the given start index. If there are no enough characters
        to copy as many as length characters from the CMPT135_String argument, then ONLY the available
        characters up to the end of the CMPT135_String argument are copied and a CMPT135_String with a
        shorter length is returned.

        The returned object is known as a substring of the CMPT135_String argument.
        */
5.  isSubString
        /*
        This function takes two CMPT135_String objects s1 and s2 as arguments and returns true if there
        exists a substring of s2 that is equal to s1; otherwise it returns false.
        */
6.  getReversedString
        /*
        This function takes a CMPT135_String argument and returns a CMPT135_String object constructed from
        the characters of the CMPT135_String argument in reverse order.
        */
7.  removeChar
        /*
        This function takes a CMPT135_String and a character arguments and removes each character of the
        CMPT135_String argument that is equal to the character argument from the CMPT135_String argument.
        */
8.  replaceChar
        /*
        This function takes a CMPT135_String and two characters named ch1 and ch2 (in that order) as
        arguments and replaces every character in the CMPT135_String argument that is equal to ch1 by
        the character ch2.
        */
```