Data Structure and Algorithm Analysis---COP3530

Module 6

Total Points: 25

NO LATE ASSIGNMENTS WILL BE ACCEPTED!!

Objectives:

This assignment will access your C++ language skills and understanding of linked lists. After completing this assignment you will be able to do the following:

(1) Manage a singly-linked list with pointers to the front and back of the singly-linked list, (2) allocate memory dynamically (3) implement a default, explicit-value and copy constructors, (4) a destructor, (5) add a node to a singly-linked list, and (6) delete a node anywhere in a singly-linked list.

In this assignment you will implement the SENTENCE ADT (abstract data type) using a singly-linked list of strings. The SENTENCE ADT is defined below. **Call the class you implement "sentence".** Remember, a singly-linked list is composed of nodes. You will also implement a class called "**word**" that has two fields: a string field called "**term**"; and a pointer field called "**next**". Essentially, each word is really a **node** in the linked list. And sentence is the singly-linked list. Each node (word) in the linked list (sentence) will contain one string of a sentence. Note that a space is considered a string. Consider the following declaration of a word. A "sentence is composed of words:

```
class word
{
   public:
      string term;
      word *next;
};
```

The state (private data members) of your "**sentence**" class should contain a pointer to the front of the list of words called "**front**" and a pointer to the back of the list called "**back**". You may add more members to the state and more member functions to the behavior of the class if you determine necessary. Store the definition of your class in a file called "**sentence.cpp**." and the declaration of your class in a file called "**sentence.h** ." You will implement all the code necessary to maintain a sentence. See the ADT below. Test the complete functionality of the class "**sentence**". **You must use the file "sentence_driver.cpp" to help you understand and test all the required functionality of the class.** If you discover that you need more tests while implementing the functionality of the **sentence** class, then add more tests to your driver, **but your final code must be submitted with the given driver .**

ADT---sentence

Data:

A set of **terms**

Operations:

1. **Default constructor:** The default constructor will initialize your state variables. The front and back of the linked list is initially set to NULL or 0; this implies a non-header node implementation of the link list.

2. **Explicit-value constructor:** This constructor will have one argument; a C-style string or a C++ string representing the sentence to be created;

3. **Copy Constructor:** Used during **a call-by-value, return, or initialization/declaration** of a sentence object;

4. **Destructor:** The destructor will de-allocate all memory allocated for the sentence. Put the message "destructor called\n" inside the body of the destructor.

5. **isEmpty:** Check to see if the sentence A is empty; A is the current object; Note if either front = 0 or back = 0 then the list is empty.

6. **length:** Determines the length of the sentence A; remember A is the current object and space should be counted in the length of the sentence;

7. **add_back:** Add a word (node) to the back of the sentence (link list). Note that if the list contains only one node then front and back will point to the same node.

8. **operator<<:** Overload the insertion operator as a friend function with chaining to print sentence A; Remember, to implement the function without the modifier friend to the left of the header. Friend should only appear in the prototype inside the class declaration.

9. **operator= :** Overload the assignment operator as a member function to take a string, with spaces (C-style or C++ string, just be consistent in your implementation) as an argument and assigns its value to A, the current object. Remember that the current object, A, is passed implicitly through **this**, and the **string** argument is passed explicitly through the formal parameter.

10. **operator= :** Overload the assignment operator as a member function **with chaining** to take a sentence object as an argument and assigns its value to A, the current object. Remember that the current object, A, is passed implicitly through **this**, and the **sentence** argument is passed explicitly through the formal parameter.

11. **operator+:** Overload the ‘+” operator as a member function without chaining to add sentence B (adds the set of terms that makeup B's linked list to the back of A's linked list) to the back of sentence A; remember A is the current object;

12. **isEqual:** Returns true if two sentence objects are equal; otherwise false; remember A is the current.

13. **remove:** Deletes the first occurrence of string B (without spaces between characters) from sentence A if it is there; remember A is the current object. Note that B can also be the empty string, or a space.

Submit sentence.h, sentence.cpp, and sentence_driver.cpp to Canvas before the due date and time.

Note: Remember the following:

1. **include a program header in your driver;**
2. **include function headers for all functions in your class implementation file;**
3. **include comments in your code when necessary to improve understanding.**