**COP 3530 Data Structures Fall 2015- - - - - - -Assignment#4----WORD**

**Total Points: 100 points**

**Due Date and Time: 9/23/15, at 11:00PM**

**NO Late Assignments!!!**

In this assignment you will implement the WORD ADT (abstract data type) using a singly-linked list of characters. The WORD ADT is defined below. **Call the class you implement "WORD".**  Remember, a singly-linked list is composed of nodes. You will also implement a class called "**character**" that has two fields:  a char field called "symbol"; and a pointer field called "next". Essentially, each character is really a **node** in the list.  Each node (character) in the list (word) will contain one alpha/numeric character of a word. Consider the following declaration of a character. A WORD is composed of characters:

**class character**

**{**

**public:**

**char symbol;**

**character \*next;**

**};**

The state (private data members) of your WORD class should contain a pointer to the front of the list of characters 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 "**word**.cpp." and the declaration of your class in a file called "**word**.h ."  You will implement all the code necessary to maintain a word.  See the ADT below. Test the complete functionality of the class WORD. **Use the file "word\_driver.cpp" as a guide to help you understand and test all the functionality of the class.** If you discover that you need more tests while implementing the functionality of WORD, then add more tests to your driver.

**ADT---WORD**

**Data:**

A set of characters

**Operations:**

**1. IsEmpty:** Check to see if the word A is empty; A is the current object;

**2. Length:** Determines the length of the word A; remember A is the current object;

**3. Insert:** **Inserts a copy of word B (adds alphanumeric characters that makeup B's linked list into A's linked list) into word A before position p; remember A is the current object; If p is 0 or 1, B is inserted at the front of A. If p is 2 or greater but less than or equal to the length of A, then B is inserted just before the node in position p. If p is greater to the length of A, then B is inserted at the end of A. Note that the first node is in position 1; the second is in position 2; the third node is in position 3, and the nth node is in position n. YOU MUST INSERT A SEPARATE COPY OF B INTO A.**

**4. Remove:** Deletes the first occurrence of word B (removes the first set of characters that makeup B's linked list from A's linked list) from word A if it is there; remember A is the current object;

**5. RemoveAll:** Removes all occurrences of word B (removes each set of characters that makeup B's linked list from A's linked list) from word A if it is there; remember A is the current object. Note that the search for a word to be removed always starts at the beginning;

6. operator<<**:**  Overload the insertion operator as a friend function with chaining to print a word A;

7. operator= : Overload the assignment operator as a member function to take a string (C-style or C++ string, just be consistent in your implementation) as an argument and assigns its value to A, the current object;

8. operator= : Overload the assignment operator as a member function **with chaining** to take a word object as an argument and assigns its value to A, the current object;

9. operator+: Overload the ‘+” operator as a member function without chaining to add word B (adds the set of symbols that makep B's linked list to the back of A's linked list) to the back of word A; remember A is the current object;

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

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

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

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

**14. IsEqual:** Returns true if two word objects are equal; otherwise false; remember A is the current

Submit word.h, word.cpp, and word\_driver.cpp in a ONE zip file to Blackboard in the Assignments Area before the due date and time. Only assignments submitted through Blackboard will be graded.

Please get started and ask questions ASAP!