CS 2433 C/C++ Programming

Programming assignment 4

10 points

Due date and time: 11:59 PM, September 15, 2016

The objective of this assignment is to familiarize you the use of the struct and typedef keywords/constructs, as well as to continue with familiarization with arrays. The context for this is use of an array of structs that are used as a pool of nodes for managing two lists. One list is a doubly linked list (which allows traversal both forwards and backwards) and a singly linked list (which allows only forward traversal).

Write a C program, in a file named ‘p4.c’, to solve the following:

~~Declare a **typedef** for a **struct** that contains an array of 16 characters, and two **int** values. The **typedef** is to be named **wnode**, the character array **aword** and the ints **last** and **next**. Declare an array of 100 nodes of **typedef wnode** named **listnodes**.~~

~~Your program will accept a series of character strings from the standard input device (**stdin**) and either insert each (up to 15 characters maximum) into a sorted doubly linked list, or delete the specified word (and node) from that list, or terminate processing when prompted, printing the list of words in order, then in reverse order, to the standard output (**stdout**).~~

~~The singly linked list is used to manage the nodes in **listnodes** that are neither the head-node of the doubly linked list nor used in the doubly linked list itself. Nodes for words being inserted into the doubly linked list are obtained from this free-node list, and nodes freed by deletion of words from the doubly linked list are returned to the free-node list.~~

A valid “word” is any printable character string that does not start with a tilde (~). Valid words are to be inserted into the list in ascending order using an insertion sort. Valid words are to be truncated to 15 characters if longer than that. If an input starts with a tilde, the substring to the right of the tilde is to be deleted from the list, and the node freed and placed in the free-node list.

~~Normal program termination (resulting in printing the two lists of words) will occur when the program reads “!stop” from the standard input.~~

For ordering of nodes, the **next** field of the node is to contain the index of the next node in the sorted order of the character strings, while the **last** field of a node is to contain the index of the immediate previous node. (Example: If two adjacent nodes in sorted order contain the words “apple” and “apply”, the **next** field of the “apple” node will contain the index of the node containing “apply”, while the **last** field of the node containing “apply” will contain the index of the node containing “apple”.) You will need to use the **strcmp** function to compare strings during the insertion process. The **strcpy** or **strncpy** function will also be useful. Both of these functions are located in the **string.h** standard header file.

The first node in **listnodes** (index 0) is a special use node, a head-node, that is not properly in the sorted list of words to be constructed. It will contain the string “HEADNODE” in its character string, the **next** field will point to the first node in the ordered doubly linked list, and the **last** field will point to the last node of that list. Both the **next** and **last** fields are initialized to 0, indicating an empty list, since an index of 0 indicates linkage to the **headnode** and thus that the next or previous node does not exist.

The **listnodes** array nodes following node 0 (the headnode) are to be initialized such that the **last** fields all contain 0, and the **next** fields the index of the next node, except the last, in which case the **next** field will contain a -1, indicating end of list. An **int** named **freenodes** will be initialized to a value of 1, indicating the first node following the headnode in the **listnodes** array.

To insert a new node for a new word, copy the word read from **stdin** to the **aword** field of a free node, unlink this node from the free list, then search for a location in the doubly linked list of words to insert the new node/word in its proper place. Unlink a node from the freenodes list as follows, assuming **a** is an **int** used to identify the node being unlinked. Set **a** to the value of **freenodes**. Set **freenodes** to **listnodes[a].next**.

Insertion of a new word/node occurs either immediately before the first word that should follow the word being inserted, or at the end of the list, if there is no word in the list that would follow it.

Insertion between two nodes in the doubly linked list involves the following steps. Find the location where the new node is to be inserted. This location is indicated by the index in **listnodes** of the word/node that is to follow the inserted word/node. (This may be the head-node, if inserting into either an empty list or at the end of the list) Let **a** be the index of the node to be inserted, **b** the index of the node it is to precede. Copy **listnodes[a].next** to **freenodes**. Copy **listnodes[b].last** to **listnodes[a].last**. Copy **b** to **listnodes[a].next**. Copy **a** to **listnodes[b].last**, and to **listnodes[listnodes[a].last].next**.

Deletion of a word/node involves finding the word in the list of nodes. Once found assume **d** is the index of the node to be deleted. Delete the node from the list as follows. Set **listnodes[listnodes[d].last].next** to **listnodes[d].next**. Set **listnodes[listnodes[d].next].last** to **listnodes[d].last**. Set **listnodes[d].next** to **freenodes**. Set **freenodes** to **d**.

If a word to be deleted is not found in the list, print "Not found.\n", and continue processing.

If there are no nodes available for a new node to be inserted (**freenodes** is -1), display an error message stating that there are no free nodes, and terminate execution.

Output on normal termination will list the words, separated by spaces in order first, display a blank line following the in-order list, then the list in reverse order, terminated with a new-line. A very simple example follows.

```
Alar Apple Apply

Apply Apple Alar
```

REQUIREMENTS:

1) Program must compile without errors and start without crashing. (10 points.)

2) Program source must be properly documented, with the standard student and assignment information at the start of the file. Documentation includes a block of comments immediately before each function describing the function inputs and outputs, plus "in-line" documentation of the code. (2 points.)
3) You may assume the list will not be empty at normal termination of processing, but you must print the in-order and reverse-order lists correctly. (8 points.)

CAUTION: This is an individual programming assignment. All work must be done individually. Sharing (or copying from any source) of code segment will be treated as plagiarism and dealt with accordingly.

Submit the solutions using the "handin" command.

Example submission command is "handin cs2433 program4 p4.c", where p4.c is the file being submitted.

See the following Web page for a description of the qsort() function.

http://www.cplusplus.com/reference/cstdlib/qsort/