CS 344 Homework 5

Due: Tuesday, 3/19/2019 by 12:45 PM

Please STUDY this ENTIRE document BEFORE starting ANY coding!

Make sure that you have a copy of the updated file, doublyLinkedList.c, which was also attached.

NOTE: If you have already modified this file, do NOT start from mine. Start from yours since **some** of the requirements are the same as from the classwork.

Then, edit the file and implement the functions and a main as described below. (Note: I added more functions to the list from the classwork AND I added a description of a main.)

After implementing a function, test it thoroughly by making appropriate calls to it in the main. (See description of main near the end of this document.)

DOCUMENT your code completely! You will be happy that you did.

void * removeFromFront(LINKED_LIST * listPtr)

This removes, and returns, the data from the node at the front of the list or NULL if there is nothing to remove. Remember to free the NODE.

void * removeFromBack(LINKED_LIST * listPtr)

This removes the node at the back of the list and returns its data or NULL if there is nothing to remove. Remember to free the NODE.

int insert(LINKED_LIST * listPtr, void * data, int position)

This inserts the data at the given position in the list as long as the position is bigger than or equal to 0 and less than or equal to the current size of the list. If the position is out of range, the function returns -1. Inserting at position 0 means adding to the front and inserting at the position equal to the size means adding to the back. If all is well, the function returns a 0.

void * remove(LINKED_LIST * listPtr, int position)

This removes the node at the given position in the list as long as the position is bigger than or equal to 0 and less than the current size of the list. If the position is out of range, the function returns NULL. Otherwise, it returns the data of the node at that position. Removing at position 0 means removing the front and removing at the position one less than the size means removing from the back. If all is well, the function returns the data of the node to be removed. Remember to free the NODE.

void deleteList(LINKED_LIST * listPtr)

This deletes the data pointed to by each node of the list as well as the node itself. Remember to set front and back to NULL after all nodes have been deleted.

LINKED_LIST * mergeLists(LINKED_LIST list1, LINKED_LIST list2, void * copyData(void *))

This creates a NEW list comprised of COPY of list2 attached to the end of a COPY of list1. Note that this function does NOT modify EITHER list and that you must make complete COPIES of each node as well as the data that each node points to. That is why you have to pass in the copyData function (see description later on in this document) since it is the only thing that knows what the data looks like. This function should call the following function (that you have to write) to create a copy of the given list:

> **LINKED_LIST * copyList(LINKED_LIST list, void * copyData(void *))**
>
> This function makes a copy of the list parameter and uses the copyData function to create a copy of the data pointed to by each node.

void splitList(LINKED_LIST * originalList, LINKED_LIST * listPart2, int splitPosition)

This function modifies the original list so that it now ends at splitPosition – 1 and the new list starts at splitPosition. Therefore, splitPosition has to be from 1 to (size – 1). If splitPosition is out of range, the function does nothing.

void * copyData(void * data)

This function should be specific to handling integer data that goes on the list. See my display function that prints integer data to see how it is specific to integer data.

int main(int argc, char * argv[])

This function will work with lists of integers. It creates two empty LINKED_LIST variables and then presents the user with a menu of choices for testing all of the primary functions in this doublyLinkedList.c file, including those that were already there: append, prepend, removeFromFront, removeFromBack, insert, remove, mergeLists, and splitList. Based on user response, the appropriate functions are called.

There should be a function for presenting the menu and returning the response as well as a function for handling the response.

The choices that affect one list only should ask the user which list they wish to affect and proceed accordingly on the list chosen.

The user should be able to continuously make selections as long as she does not select whichever option for quitting is in the menu.

Make sure to call the deleteList function to destroy any lists that were created and are no longer in use.

Every time a list is modified or created, the traverseForward function should be called on the list.

Test your program thoroughly please!

Please put your name, the date, and a short description of the assignment at the top of the modified code.

When you are done, zip the program up into a **compressed file named** as follows:

yourUserName_hwk5_mmddyyyy

mmddyyyy refers to the due date. (If you are using your own computer, use your ERAU username.)

Example: bethelmd_hwk5_03192019

Upload the zipped file onto Canvas.