Program 6 - ADT Lists

Due Date: 5:00 p.m., December 1, 2016
All programs will be tested on the machines in the Q22 lab. It is required that your code run on the computers in the lab (or via ssh).

Any changes made to the assignment after posting will be in red

- *All structs and function declarations should go into a header file*
- *You must use separate compilation for this Program, with a header file and source file for your data types*

Part 1: Linked Lists

- Part A: Doubly Linked List
    - You should break your code into 3 files
        - list.c/h
        - main.c //driver code provided below
    - Write a Data struct that holds an integer called value
    - Create a doubly linked list using a list and node structs. You must create your linked list on the heap (using malloc). Your linked list should have the following operations:
        - createList - initializes the linked list struct with the following attributes:
            - A pointer to a head and tail node
            - returns a pointer to a List struct created on the heap
        - Insert - inserts an element at a specified index in the list.
            - void insertData(List * list, int index, Data value);
                - If the index is out of bounds, add the Node to the end of your list. For example, if you have 3 items in your list, and you attempt to insert an item at position 5, it should just be placed at position 3 (starting from 0).
                - *Do not put 'empty' nodes in your list to pad out the positions.*
        - Delete - deletes an element from a specified index in the list.
            - int removeData(List * list, int index);
                - If the index is out of bounds, you should just return without doing anything.
                - if the index is negative, you should traverse backwards from the tail
        - Read - returns a pointer to the data element stored in the list
            - Data * readData(List * list, int index);

            - If the index is out of bounds, you may handle it however you wish, but your program should not crash.
        - Empty - returns 1 if the list is empty, 0 if it is not

- Part B: Doubly Linked List Search
    - Add the following functionality
        - searchForward - search for the value starting from the front node
        - searchBackward - search for the value starting from the back node.
    - Return the number of steps before the value is found.
        - Do not worry about duplicate values in the list. In this case the number of steps won't equal the size of the list.
        - If the value is not found, return -1
    - Upon exiting, use your delete function should clean up the linked list memory.

Part 2: Stacks and Queues

- Part A: Stack
    - Create a Stack struct that is a wrapper for your linked list
    - You should implement the following functions that take a Stack:
        - void push(Stack * stack, Data value)
        - Data pop(Stack * stack)

- Part B: Queue
    - Create a Queue struct that is a wrapper for your linked list
    - You should implement the following functions that take a Queue:
        - void enqueue(Queue * queue, Data value)
        - Data dequeue(Queue * queue)

Part 3 - Submission

- Required code organization:
    - program6.c - Do not alter the main driver code
    - list.c/h - Your header file should have (at minimum) the following function declarations:
        - Data struct
            - value (int)
        - Node struct
            - data (Data)
            - next (Node *)
            - prev (Node *)

        - //definition of list structure.
        - List struct
            - head (Node *)
            - tail Node *
        - List * createList();
        - void insertData(List * , int , Data);
        - void removeData(List * , int);
        - Data * readData(List * , int);
        - int isEmpty(List *);
            - returns 0 if empty and 1 if not empty
        - void deleteList(List *);
            - deletes the entire list from memory
        - int searchForward(List *, Data);
        - int searchBackward(List *, Data);
        - Stack struct
            - List * list
        - void push(Stack *, Data);
        - Data pop(Stack *);
        - Queue struct
            - List * list;
        - void enqueue(Queue *, Data);
        - Data dequeue(Queue *);
    - makefile
        - *You must have the following labels in your makefile:*
            - all - to compile all your code to an executable called '**program4**' (no extension). **Do not run**.
            - run - to compile if necessary and run
            - checkmem - to compile and run with valgrind
            - clean - to remove all executables and object files
- While inside your Program 6 folder, create a zip archive with the following command
    - zip -r program6 *
        - This creates an archive of all file and folders in the current directory called program6.zip
        - **Do not zip the folder itself, only the files required for the program**
- Upload the archive to Blackboard under Program 6.
- You may demo your program by downloading your archive from Blackboard. Extract your archive, then run your code, show your source, and answer any questions the TA may have.

Grading Guidelines

Total:

- **Part 1:**
    - Passes Test # 1 *(2 points)*
    - Passes Test # 2 *(5 points)*
    - Passes Test # 3 *(3 points)*
    - Passes Test # 4 *(4 points)*
    - Passes Test # 7 (3 points)
- **Part 2:**
    - Passes Test # 5 (4 points)
    - Passes Test # 6 (4 points)
- **Style Guidelines and Memory Leaks**
    - You will lose significant points for the following:
        - Makefile does not have requested format and labels (-5 points)
        - Does not pass Valgrind Tests (-10 points)
        - Does not follow requested program structure and submission format (-10 points)
        - Does not follow formatting guidelines (-5 points)