**CS 200: Introduction to Programming**

Dr. Naveed Arshad

Muhammad Kamran Nishat

Assignment 4  
(Due Date: Thursday, November 20, 2014)



This assignment is due on Thursday, 20th November. The usual late submission policy of 10% per day deduction for up to 3 days applies. The assignment needs to be submitted on LMS under the ‘Assignments’ tab.

The course policy about plagiarism is as follows:

1. Students must not share actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students cannot copy code from the Internet.
4. Students must indicate with their submission any assistance they received.
5. All submissions are subject to automated plagiarism detection.

Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee.

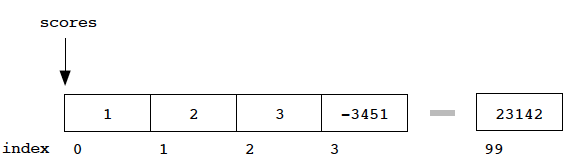
**1. LINK LIST**

Why Linked Lists?

Linked lists and arrays are similar since they both store collections of data. The terminology is that arrays and linked lists store "elements" on behalf of "client" code. The specific type of element is not important since essentially the same structure works to store elements of any type. One way to think about linked lists is to look at how arrays work and think about alternate approaches.

Array Review

Arrays are probably the most common data structure used to store collections of elements. In most languages, arrays are convenient to declare and the provide the handy[ ] syntax to access any element by its index number. Here is a drawing of how the scores array might look like in memory. The key point is that the entire array is allocated as one block of memory.



The disadvantages of arrays are...

1. The size of the array is fixed — 100 elements in this case. Most often this size is specified at compile time with a simple declaration such as in the example above . With a little extra effort, the size of the array can be deferred until the array is created at runtime, but after that it remains fixed
2. the most convenient thing for programmers to do is to allocate arrays which seem "large enough”. So you have to know the maximum size at the moment you allocate array.
3. Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.

Linked lists have their own strengths and weaknesses, but they happen to be strong where arrays are weak. The array's features all follow from its strategy of allocating the memory for all its elements in one block of memory. Linked lists use an entirely different strategy. As we will see, linked lists allocate memory for each element separately and only when necessary.

**Linked List Types: Node and List**

Before writing the code to build the above list, we need two data types…

**Node** The type for the nodes, which will make up the body of the list. These are allocated in the heap. Each node contains a single client data element and a pointer to the next node in the list. Type: class node

class node {

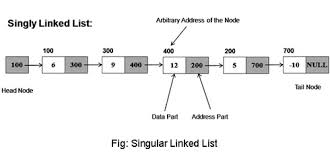
int data;

node\* next;

};

**List:** This will have of the head pointer which points to the first node in the list. We will find next element of the field with next pointer in the head node. We can move to next nodes until we find next point set to NULL.

Sometimes list type also have a pointer named tail that points to the last node of the list.



**TASK 1 (Linked List): [40 Points]**

For this part of the assignment, you need to implement a linked list class that contains various member functions. The basic layout of this linked list class is provided to you in *LinkedList.h.*

The Template ListItem in the LinkedList.h file represents a node in the linked list. The class List (see Linkedlist.h) implements the linked list, which contains a pointer to the head of the linked list and function declarations.

NOTE: *When implementing the member functions, make sure you handle error conditions such as deletion from an empty linked list*.

*Member functions:*

This section defines the purpose of the member functions in the List class for which you have to write the code

List()  
  
This is simply the default constructor of the List class

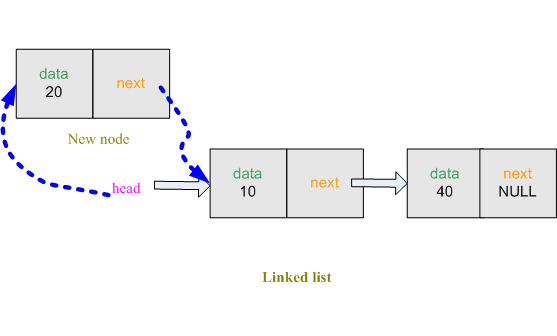
List(const List& otherList)  
  
This is the copy constructor of the List class which when provided with a pointer to another list otherList constructs a linked list with the same elements as otherList.

~List()

Destructor for the List class. Deletes all the nodes in the list and frees the memory allocated to the linked list

void insertAtHead(int item)

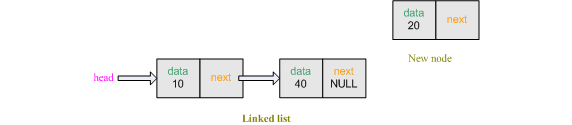
Function which inserts an item of type T (remember the basic unit is a template) at the head of the list.



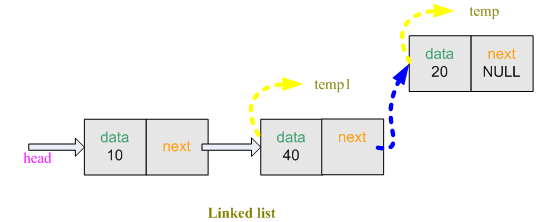
void insertAtTail(int item)

This function inserts an item of type T at the tail of the list.

**BEFORE:**



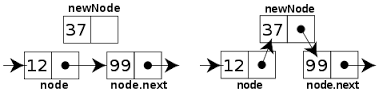
**AFTER:**



void insertAfter(int toInsert, int afterWhat)

This function first goes through the linked list to find the item afterWhat. When this is found it then inserts toInsert after the afterWhat linked list node.

e.g: insertAfter(37,12); should do the following.



Node \*getHead()

Returns the head pointer of the linked list, returns NULL if empty.

Node \*getTail()

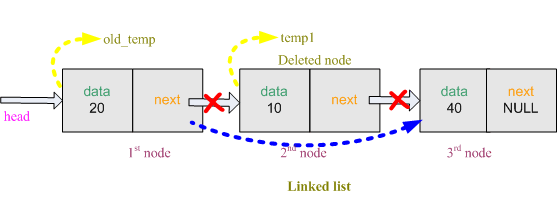
Returns the tail pointer of the linked list, returns NULL if empty.

Node \*searchFor(int item)

Returns the pointer to node contain the element item. The function returns NULL if the list is empty or the item is not found.

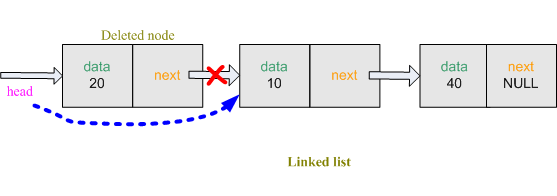
void deleteElement(int item)

This function deletes the first node containing the element item.



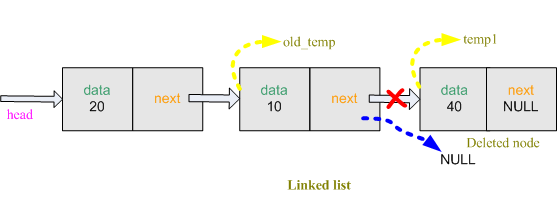
void deleteHead()

Deletes the first node (i.e., the head) of the linked list.



void deleteTail()

Deletes the tail node of the linked list.



int length()

This function returns the length of the linked list i.e., the number of nodes currently in the linked list.

**TASK 2 (Stacks and Queues):**

For this task, you will use your linked list class to implement the stack and queue data structures.

***Stack: [20 Points]***

The Stack class contains a List type object. This means that all the member functions defined in the Linked List class would be accessible. (The same is the case for the Queue class)

*Member functions:*

This section defines member functions of the Stack class for which you have to write the code given in the header file stack.h.

Stack()

This is the default constructor of the Stack class.

Stack(const Stack& otherStack)

This is the copy constructor for the Stack class which when provided with a pointer to another Stack, contructs a new Stack with the same elements as the existing otherStack object.

~Stack()

Destructor for the Stack class. Be sure to free any allocated memory.

void push(int item)

Function which inserts elements in the Stack object (which follows the Last-In First-Out (LIFO) policy for insertions and deletions).

int top()

Function which returns the value on top of the Stack i.e., the element that was last inserted. This function only returns the element without removing it from the Stack.

int pop()

Function which returns the value on top of the Stack (the element last inserted into the Stack) and in addition, removes the element from the Stack.

int length()

This function returns the number of elements in the Stack.

bool isEmpty()

This function returns true if the stack is empty and returns false otherwise.

***Queue: [20 Points]***

The Queue also contains a List type object, just like the Stack.

*Member functions:*

Queue()

This is the default constructor of the Queue class.

Queue(const Queue& otherQueue)

Copy Constructor for the Queue class which when provided with a pointer to another Queue contructs a new Queue object with the same elements as the existing otherQueue object.

~Queue()

Destructor for the Queue class. Be sure to free any allocated memory.

void enqueue(int item)

Function which is used to add elements to the Queue object (which follows the First-In First-Out (FIFO) policy for insertions and deletions).

int front()

Function which returns the element at the front of the Queue i,e., the element in the queue which was inserted first. This function only returns the element without actually removing it from the Queue.

int dequeue()

Function which returns and removes the element at the front of the Queue.

int length()

This function returns the number of elements in the Queue.

bool isEmpty()

This function returns true if the queue is empty and false otherwise.

**TASK 3 (Recursion): [20 Points]**

In this task you don’t have to write code. Yeah! ☺

You have to search for recursive code on Internet. Try to find the most unique and interesting recursive code. But you have to understand it. We will take viva to determine your understanding. You will get no points if you would not be able to explain it.

We will give **20 Bonus points** to 3 students with best among all.