In this week in the lab, we will implement stack and queue using linked list.You have got a breif idea about them in the class room and the steps are also available in the Linked List slide. Now, we will write the code in the lab. We will try to complete the queue implementation in the lab and you will need to complete the stack at home. After completing the queue code, you will need to upload it in Webcourses.

♦ Problem1: Implement a queue using linked list :

- The main concept: anything added to the linked list will be inserted at the end and anything deleted should be deleted from the front of the linked list.
- But, to add anyting at the end, traversing to the end everytime is expensive/time consuming. Imagine havig a queue of size 1000000. For inserting next item, you will need to visit all 1000000 to find the last node in the queue.

So, to solve the problem we will maintain two pointers:

- One for front of the list
- One for the back

- Our struct to store the queue, would actually store two pointers to linked list structs:

- The first one would point to the head of the list.
- The second one would always point to the last node in that list

See the node structure and queue structure bellow:

```
// Stores one node of the linked list.
struct node {
    int data;
    struct node* next;
};
```

```
// Stores our queue.
struct queue {
    struct node* front;
    struct node* back;
};
```

Functions needed:

I. **init**: This function should make front and rear of the queue as NULL.

II. **enqueue**:
   a) Create a new node and store the inserted value into it.
   b) Link the back node's next pointer to this new node.
   c) Move the back node to point to the newly added node.

III. **dequeue**:
   a) Store a temporary pointer to the beginning of the list.
   b) Move the front pointer to the next node in the list.
   c) Free the memory pointed to by the temporary pointer.

IV. **front**:
Directly access the data stored in the first node through the front pointer to the list.

V. **empty**:
Check if both pointers (front, back) are null.

♦ Problem 2: Implement a stack from linked list (no need to submit it, but a very good practice!)

I. Where would you push?
In the beginning of the linked list (it means you are the node with new item as head of the list)

II. Where the items should be deleted from in case of POP?
From the head of the linked list.

Very easy, right? You don't have to traverse the linked list at all. Just insert in the head and also delete from the head!

So,we just change the name of our node structure:

```
 // Struct used to form a stack of integers.
struct stack {
  int data;
  struct stack *next;
};
```

The functions needed:

I. **init:** This function initialize the stack to be empty.

II. **push**: Inserts an item into the front of the list. Return 1 if the push succeeded, 0 if it failed.

III. **pop**: removes and return the first node of the stack and returns a pointer to the node.

IV. **top**: Returns the integer at the top of the stack, without poping it. Returns -1 if the stack is empty.