CS261L Data Structures and Algorithms (Pr)
Lab Manual (Week 9)

Instructor:
- Mr. Samyan Qayyum Wahla

Registration No.__________________

Name: _________________________

Guide Lines/Instructions:
You may talk with your fellow CS261-ers about the problems. However:
- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

Today’s Task:
- Design of Data Structures(LinkedList, Stack Queue)
- Application of Custom Data Structures for algorithms

Part 1: Design of Data Structures(LinkedList, Stack Queue)

1. Implement **LinkList** class in C++ which must have following functions.
   class **LinkList** {

```
public:

        List(void) { head = NULL; }  // constructor

        ~List(void);                             // destructor

        bool isEmpty() { return head == NULL; }

        Node* insertNode(int index, int x);    //insert at the given index

        Node* insertAtHead(int x); //insert at start of list

        Node* insertAtEnd(int x); //insert at end of list

        bool findNode(int x);  //search for data value x in the list

        bool deleteNode(int x); //delete all occurrences of x

        bool deleteFromStart();   //deletes starting node of list

        bool deleteFromEnd();   //deletes last node of list

        void displayList(void);

        Node* reverseList(); //reverses the linklist and returns a new list

        Node* sortList(Node *list); //sorts the input-ed list

        Node* removeDuplicates(Node *list); //removes duplicates from list

        Node* mergeLists(Node *list1, Node *list2); //merges two lists

        Node* interestLists(Node *list1, Node *list2); //results contains intersection of two lists
```

```
private:

        Node* head;

};
```

2. Implement **Stack** and **Queues**
    a. With arrays
    b. With Linklists
3. Update Node Class in the above LinkList, Create a new class with Data, Next and Prev pointer, Create DoublyLinkList and rewrite all the operations where required. Compare the time complexity of operations in LinkList and DoublyLinkList.

Stack Functions:

- **PUSH:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **POP:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- **PEEK OR TOP:** Returns top element of stack.

Queue Functions:

- **ENQUEUE()** − add (store) an item to the queue.
- **DEQUEUE()** − remove (access) an item from the queue.
- **PEEK()** − Gets the element at the front of the queue without removing it.
- **ISFULL()** − Checks if the queue is full.
- **ISEMPTY()** − Checks if the queue is empty.

Part 2: Application of Custom Data Structures for algorithms

1. Write an algorithm to reverse words in the sentence.
   Note: Do not tokenize the word, rather traverse the sentence character by character, Use the stack to solve the problem.
   Input: I am from University of Engineering and Technology Lahore
   Output: Lahore Technology and Engineering of University from am I
2. Implement a basic calculator that supports integer operands like 1, 64738, and -42 as well as the (binary) integer operators +, -, *, /, and %. The style of arithmetic expressions our calculator will evaluate is also called a post-fix notation. Stacks are great for doing this job! Your task is to write a program that uses our Stack interface and one of the given implementations to perform these calculations as specified here. Your program should work as follows:
    a. The user enters input through System.in consisting of operands and operators, presumably in post-fix notation. We have also included some extra operators to get information about results and the current state of the stack.
    b. If the user enters a valid integer, you push that integer onto the stack.
    c. If the user enters a valid operator, you pop two integers off the stack, perform the requested operation, and push the result back onto the stack.
    d. If the user enters the symbol ? (that's a question mark), you print the current state of the stack using a method toString(Implement in your stack data structure)

e. If the user enters the symbol ^ (that’s a caret), you pop the top element off the stack and print only that element (not the entire stack) followed by a new line.
f. If the user enters the symbol ! (that’s an exclamation mark or bang), you exit the program.