Data Structure	Lab 7
Spring 2021	Wednesday, May 26

Lab Goals:

- Project Screens – Register and Login
- Understanding and implementation of Linklist
- Implementation of Stack and Queues

Project Rubrics:

	Implementation
5	Properly formatted and visually pleasant UI/Template integrated + Rubric IV requirements.
4	Integrated database with Registration Screen + Rubric III requirements.
3	Integrated database with Login Screen + Rubric II requirements**.**
2	Registration Screen with user interface + Rubric I requirements
1	Login Screen with user interface created
0	No screen created

Lab Rubrics:

No plagiarism text is present in all rubrics.

	Implementation
10	Queue Implemented with all given functions with linklist + Rubric IX requirements**.**
9	Queue Implemented with all given functions with arrays + Rubric VIII requirements**.**
8	Stack Implemented with all given functions with linklist + Rubric VII requirements**.**
7	Stack Implemented with all given functions with arrays + Rubric VI requirements**.**
6	Linklist Implemented with all its functions + **All functions tested in main (code written/commented)** + Rubric V requirements.

5	Linklist Implemented upto reverse linklist functions **+ All functions tested in main (code written/commented) +** Rubric II requirements.
2	Linklist with FindNode and DisplayList functions **+ All functions tested in main (code written/commented) +** Rubric I requirements**.**
1	Only Node class and linklist manually created in main. No separate class for linklist.
0	Lab missed or solved none of the problems

Project Screens:

Students must create their Registration and Login screens of their project and get themselves evaluated in next week.

Homework Question:

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

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.
- **ISEMPTY:** Returns true if stack is empty, else false.

Reference: https://www.geeksforgeeks.org/stack-data-structure-introduction-program/

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.

Reference:
https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm#:~:text=Queue%20is%20an%20abstract%20data,first%20will%20be%20accessed%20first.