**Data Structures Lab 4**

**Course:** Data Structures (CL2001) **Semester:** Fall 2023

**Instructor: Taha Ahmed T.A:** N/A

**Note:**

* Maintain discipline during the lab.
* Listen and follow the instructions as they are given.
* Just raise your hand if you have any problem.
* Completing all tasks of each lab is compulsory.
* Get your lab checked at the end of the session.

**Circular Link List**

Circular Linked List

*The****circular linked list****is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.*

class **Node** {

public:

int data;

Node \* next;

Node() {

data = 0;

next = NULL;

}

Node(int d) {

data = d;

next = NULL;

}

**};**

class **CircularLinkedList** {

public:

Node \* head;

CircularLinkedList() {

head = NULL;

}

InsertAtTail();

InsertAtHead();

InsertAfterIndex();

DeleteAtIndex();

UpdateAtIndex();

Display();

};

**Task-1:**

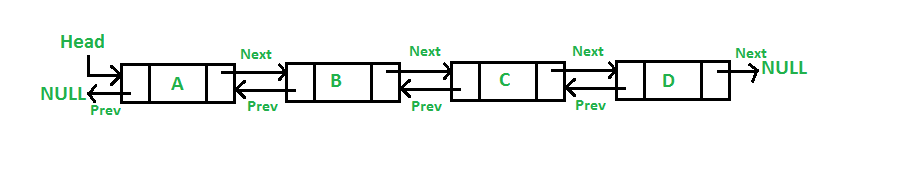
Create a circular link list and perform the mentioned tasks.

1. Insert a new node at the end of the list.
2. Insert a new node at the beginning of list.
3. Insert a new node at given position.
4. Delete any node.
5. Update node at given position.
6. Print the complete circular link list.

Graphical user interface, diagram

Description automatically generated

**Doubly Link List**



Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward or backward easily as compared to Single Linked List.

* **Link** − each link of a linked list can store a data called an element.
* **Next** − each link of a linked list contains a link to the next link called Next.
* **Prev** − each link of a linked list contains a link to the previous link called Prev.

class **Node** {

public:

int data;

Node \* next;

Node \* prev;

Node() {

key = 0;

data = 0;

next = NULL;

prev = NULL;

}

Node(int d) {

data = d;

next = NULL;

prev = NULL;

}

};

class **DoublyLinkedList** {

public:

Node \* head;

DoublyLinkedList() {

head = NULL;

}

DoublyLinkedList(Node \* n) {

head = n;

}

InsertAtTail();

InsertAtHead();

InsertAfterIndex();

DeleteAtIndex();

UpdateAtIndex();

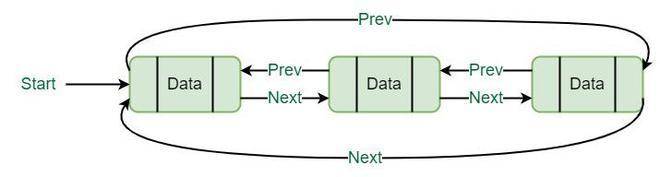
Display();

**};  
Task-2:**

Create a doublylink list and perform the mentioned tasks.

1. Insert a new node at the end of the list.
2. Insert a new node at the beginning of list.
3. Insert a new node at given position.
4. Delete any node.
5. Update node at given position.
6. Print the complete doublylink list.

**Circular Double Link List**



### In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.

* The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.
* The first link's previous points to the last of the list in case of doubly linked list.

class **node**

{

public:

int data;

node \*next;

node \*prev;

};

class **double\_clist**

{

public:

node \*create\_node(int);

insert\_begin();

insert\_last();

insert\_pos();

delete\_pos();

update();

display();

node \*start, \*last;

double\_clist()

{

start = NULL;

last = NULL;

}

};  
**Task-3:**

Create a circular Double link list and perform the mentioned tasks.

1. Insert a new node at the end of the list.
2. Insert a new node at the beginning of list.
3. Insert a new node at given position.
4. Delete any node.
5. Update node at given position.
6. Print the complete circular doublelink list.

Diagram

Description automatically generated

**Task-4:**

Write a function which overloads assignment operator to convert a doubly circular linked list into a singly linked list (not circular).

**Task-5:**

Write a function which reverses a circular singly linked list completely (Make sure to reassign head and tail appropriately).  
  
**Task-6:**

Give an efficient algorithm (Time Complexity: O(1)) for concatenating two doubly linked lists **L** and **M**, with head and tail preserved nodes, into a single list that contains all the nodes of **L** followed by all the nodes of **M**.

**Task-7:**

Create a class named LinkedList2D. This class will contain a **singly** linked list of head pointers (rows), with each head pointer pointing to a separate **doubly circular** linked list of its own (columns). Then create the following functions for the class:

InsertAtTail();

InsertAtHead();

InsertAfterIndex();

DeleteAtIndex();

UpdateAtIndex();

Display();

The functions will be slightly modified from the ones above, since you will have to specify the row number of the linked list on which to perform these operations (function parameter). The display() function should print the entire 2D Linked List.