Data Structures (Lab-4)

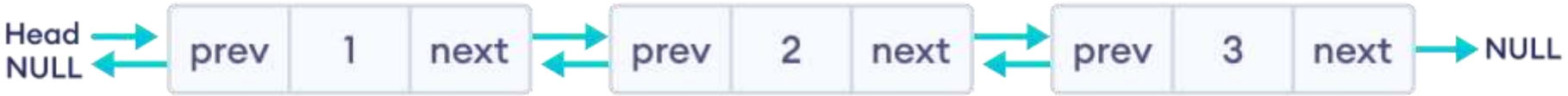
Linked Lists Part II

Agenda for Today

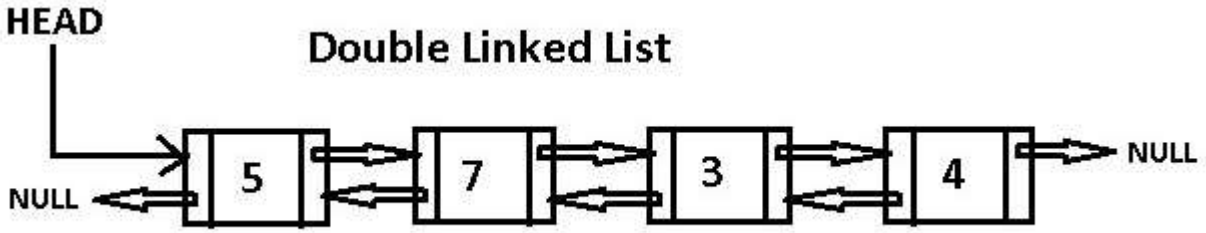
- What is Doubly Linked List
- DLL Creation
- Traversing - DLL
- Insertion at the tail
- Deletion from the tail
- circular singly linked list
- Why Circular Linked List?
- circular singly linked list implementation (Add to tail & print)
- Task

Doubly Linked List (DLL)

- Linked list in which every node has a next pointer and a back pointer
- Every node contains address of next node except last node
- Every node contains address of previous node except the first node



DLL Creation

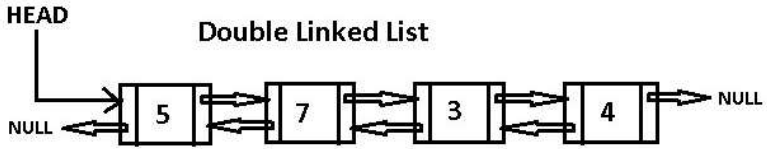


Node Class

```
class Node {
    private:
        int value;
        Node* next;
        Node* prev;
    public:
        Node () { next = prev = NULL; }       // default constructor
        Node (Node* prv, int v, Node* nxt)    // parametarized constructor
        {
            value = v;
            next = nxt;
            prev = prv;
        }
        int getValue () { return value; }
        void setValue (int v) { value=v; }

        void  setNext (Node* n) { next = n; }
        Node* getNext () { return next; }

        void  setPrev (Node* p) { prev = p; }
        Node* getPrev () { return prev; }
```

DLL Creation

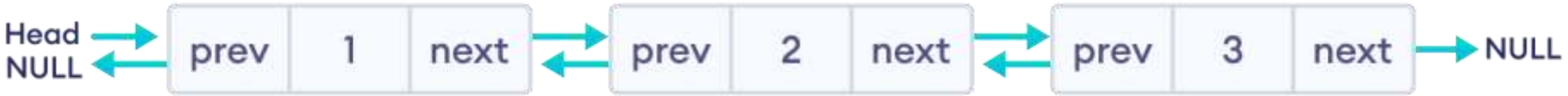
Main Function




```
int main() {
  //create an empty LinkedList
  DoublyLinkedList DLL;
  //Add first node.
  Node* one = new Node(NULL, 5, NULL);
  //linking with head node
  DLL.setHead(one);
  //Add second node.
  Node* two = new Node(one, 7, NULL);
  //linking with first node
  one -> setNext(two);
  //Add third node.
  //Add second node.
  Node* three = new Node(two, 3, NULL);
  //linking with second node
  two -> setNext(three);
  //Add second node.
  Node* four = new Node(three, 4, NULL);
  //linking with first node
  three -> setNext(four);
  //print the content of list
  while (DLL.getHead() != NULL){
    cout << DLL.getHead() -> getValue() << '\t';
    DLL.setHead(DLL.getHead() -> getNext());
  }
  return 0;
}
```

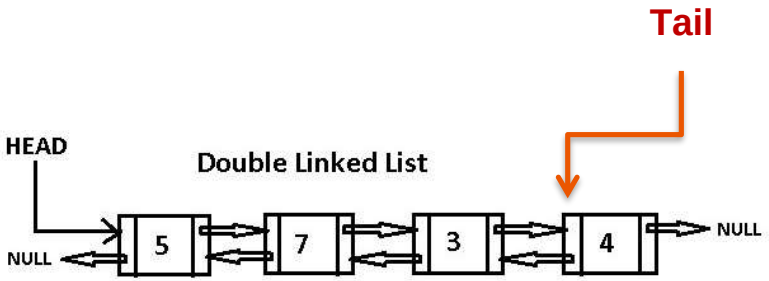
Doubly Linked List (DLL)

- Linked list in which every node has a next pointer and a back pointer
- Every node contains address of next node except last node
- Every node contains address of previous node except the first node



DLL Creation (Better approach)

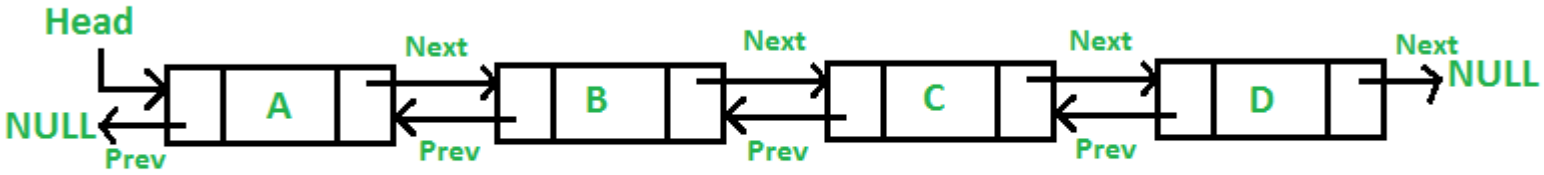
- Add ‘tail’ pointer in your consideration in DLL class to control the interactions with the end of the DLL easily.

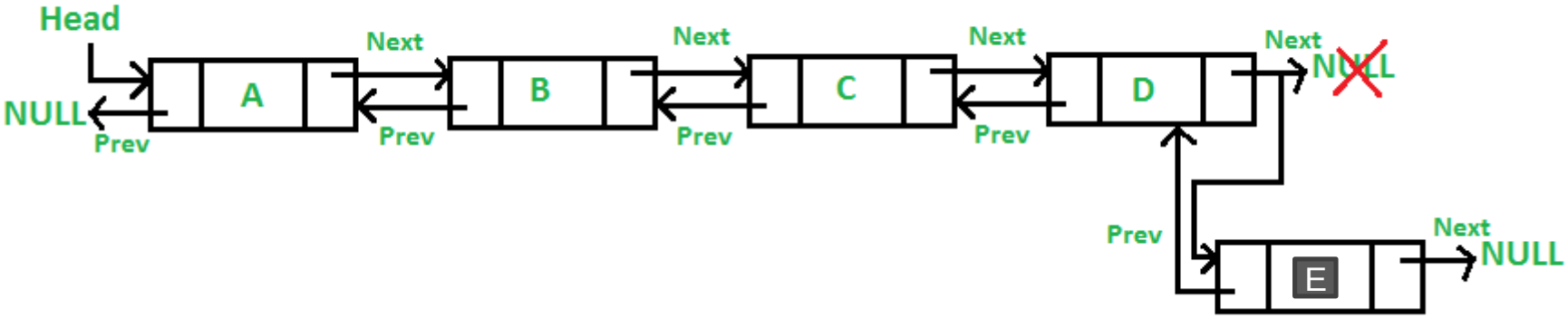


Traversing in DLL

```
class DoublyLinkedList{
    private:
        Node* head;
        Node* tail;
    public:
        DoublyLinkedList(){
          head = NULL;
          tail = NULL;
        }
        void  setHead (Node* h) { head = h; }
        Node* getHead () { return head; }
        void  setTail (Node* t) { tail = t; }
        Node* getTail () { return tail; }
        void print(Node * here){
          while (here != NULL){
            cout << here -> getValue() << '\t';
            here = here -> getNext();
            }
        }
};
```

- **Insertion at the tail:**





Insertion at the tail:

In DLL Class

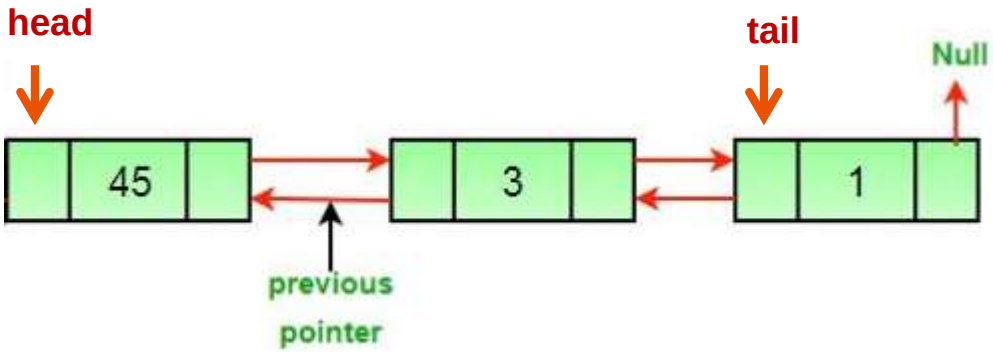
```
void insertAtTail(int entry) {

    if (tail != NULL)
    {
        Node *newNode = new Node();
        newNode -> setValue(entry);

        tail -> setNext(newNode);
        newNode -> setPrev(tail);

        tail = newNode;

    }
    else
    {
        head = tail = new Node(NULL ,entry, NULL);
    }

}
```

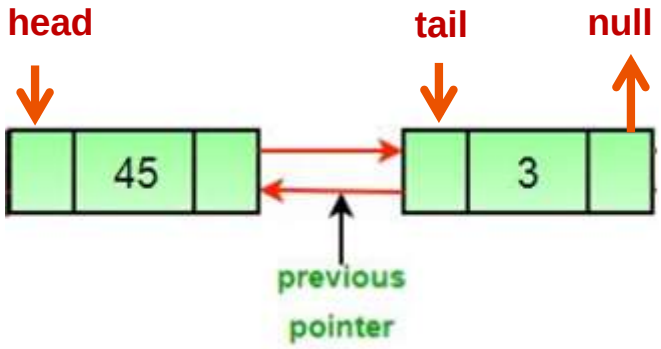
In main function

```
DLL.print(one);

cout << "**************" << endl;

DLL.insertAtTail(40);
DLL.print(one);
```

```
5       7       3       4
**************
5       7       3       4       40
```

Deletion from the tail:





❑ Deletion from the tail:

In DLL Class

```
void DeleteFromTail(){
    if (tail == NULL){
        cout << "empty Linked List";
    }

    tail = tail -> getPrev();

    delete tail -> getNext();

    tail -> setNext(NULL);

}
```

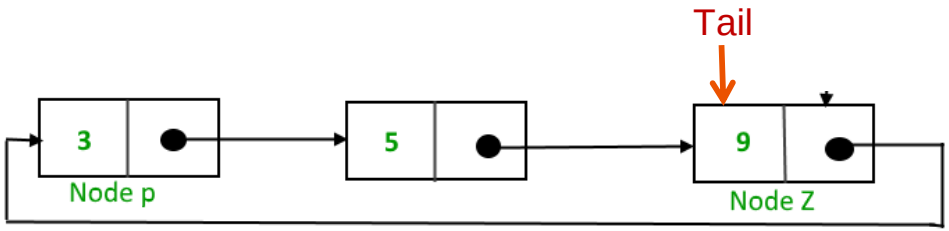
In main function

```
DLL.print(one);

cout << "**************" << endl;
DLL.DeleteFromTail();
DLL.print(one);

cout << "**************" << endl;
DLL.DeleteFromTail();
DLL.print(one);

return 0;
}
```

```
5       7       3       4
**************
5       7       3
**************
5       7
```

circular singly linked list

- A circular linked list is a linked list where all nodes are connected to form a circle.
- Generally, the last node of the linked list has a NULL in the address field.
- But a circular linked list has the address of the head node in the address field of the last node.
- To implement a circular singly linked list, we take an external pointer '**tail**' that points to the last node of the list. If we have a pointer last pointing to the last node, then **tail-> next** will point to the first node.



Why Circular Linked List?

- To be able to insert and delete nodes at the front and at the end of the list without using a loop
- For the insertion of a node at the end, the whole list has to be traversed.
- If instead of the **'head'** pointer, we take a pointer to the last node '**tail**', then there won't be any need to traverse the whole list. So insertion at the beginning or at the end takes constant time, irrespective of the length of the list.

- **circular singly linked list implementation (Add to tail & print)**

Node Class

```
class Node {
    private:
        int value;
        Node* next;
    public:
        Node () { next = NULL; }
        Node (int v, Node* nxt)
        {
            value = v;
            next = nxt;
        }
        int getValue () { return value; }
        void setValue (int v) { value=v; }

        void  setNext (Node* n) { next = n; }
        Node* getNext () { return next; }

};
```

❑ circular singly linked list implementation (Add to tail & print)

CLL Class

```
class CircularLinkedList{
    private:
        Node* tail;

    public:
        CircularLinkedList(){
          tail = NULL;
        }

        void  setTail (Node* t) { tail = t; }
        Node* getTail () { return tail; }

        void print(Node * here){
          Node* temp = tail->getNext();
          if (here == temp && temp == tail){
            cout << temp->getValue();
          }

          while (here->getNext() != temp){
            cout << here -> getValue() << '\t';
            here = here -> getNext();
          }

          cout << here -> getValue() << '\t';
          cout<< endl;
        }
```

```
        void addToTail(int entry) {
            Node *newNode = new Node();
            newNode -> setValue(entry);
            if (tail == NULL)
            {
                tail = newNode;
                tail->setNext(tail);
            }
            else
            {
                newNode -> setNext( tail -> getNext());
                tail->setNext(newNode);
                tail = newNode;
            }
        }
};
```

- **circular singly linked list implementation (Add to tail & print)**

In main function

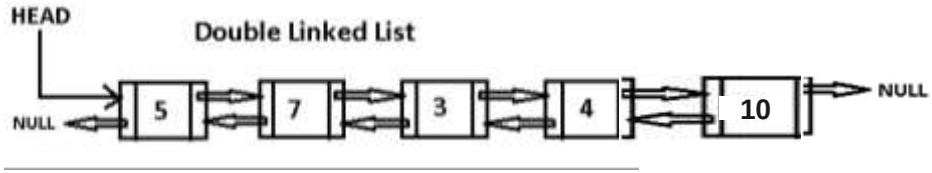
```
int main(){
    CircularLinkedList CLL;
    CLL.addToTail(50);
    CLL.addToTail(500);
    CLL.addToTail(600);
    CLL.addToTail(700);

    CLL.print(CLL.getTail()->getNext());
}
```

```
50      500     600     700

--------------------------------
```

Task of Today

Implement the method to delete specific position from Doubly linked list given the position (int) as a parameter of the method.

- **You should start from the uploaded file on moodle "Lab4_task_beginning_for_students"**
- **You must rename the file with format "yourID_yourName_lab4_task"**
- Function name "remove_at_position(int position)"
- The method will return Boolean variable.
- The method returns **true** if the value deleted.
- The method returns **false** if the value doesn't be deleted.



- See the next slide to get required output

Task of Today

- In main function you should follow below ordering in calling to try all cases of deleting that:
  - delete head “remove_at_position(1)”
  - delete tail “remove_at_position(4)”
  - delete second node “remove_at_position(2)”
  - delete out of range “remove_at_position(6)”

- You should get the below output:

```
********** Original List **********
5       7       3       4       10
********** delete head **********
7       3       4       10
********** delete tail **********
7       3       4
********** delete second node **********
7       4
```

Thank you.