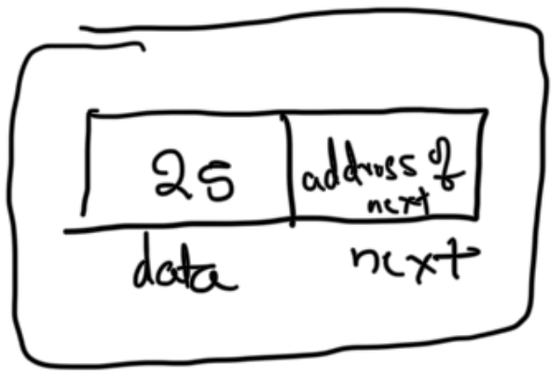
C# Linked Lists

Linked - List (linear data structure)

* Elements are stored in the form of a node, Each node contains 2 sub elements. A data part stores the values of the element and next part that stores the link to the next node.



* The first node is known as HEAD, is always used as reference to traverse the list. The last node points to Null. The LL can be visualized as a chain of nodes, where every node points to the next node.

Types of Linked List

Types of

* Singly LL * Doubly LL * Circular Singly LL * Circular Doubly LL

Singly Linked List

Implementation of Single LL

In C#, Singly LL can be represented as a class and a Node as a seperate class. The LinkedList class contains a reference of Node class type.

```
//node structure
class Node {
   Public int data;
   Public Node next;
};

class LinkedList {
   Public Node head;
// Constructor to create Empty LL
   Public LinkedList() {
```

```
        head = null ;
    }
};
```

Create Singly Linked list

```
// node structure
class Node {

    public int data;

    Public Node next;
};

class LinkedList {

    public Node head;
    // constructor to create an Empty LinkedList
    Public LinkedList() {

        head = null;
    }
```

```
class Implementation {

    Public void Main (String[] args) {
        // create Empty list
        LinkedList myList = new LinkedList();
        // Add first Node
        Node first = new node();
        first.data = 10;
        first.next = null;
        // Linking with head node
        myList.head = first;
        //Add second node
        Node Second = new Node();
        Second.data = 20;
```

```
    }
};
```

```
second.next = null
// linking with first node
first.next = second;
```

Traversing a Node

```
Public void printnode() {
    Node temp = new node();
    temp = head;
    if (temp != null) {
        while (temp != null) {
            Console.WriteLine(temp.data);
            temp = temp.next;
        }
    }
}
```

Adding new Element to the end of the List

```
Public void push_back(int newElement) {
    Node newNode = new Node();
    newNode.data = newElement;
    newNode.next = null;
    if(head == null) {
        head = newNode;
    }
    else {
        Node temp = new Node();
        temp = head;
        while(temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }
```

}

Insert a new Node at the Start of the LL

```
Public void push_Front (int newElement) {

    Node newNode = new Node();
    newNode.data = newElement;
    newNode.Next = null;

    if (head == null)
        head = newNode;
    else {
        newNode.next = head;
        head = newNode;
    }
}
```

Insert a new node at a given position

```
Public void push_at (int newElement, int position) {
    Node newNode = new Node();
    newNode.data = newElement;
    newNode.next = null;
    if (position == 1) {
        newnode.next = head;
        head = newNode
    }
    else {
        Node temp = head;
        for(int i=1; i< position-1 ; i++){
            if (temp != null) {
                temp = temp.next;
            }
        if (temp != null) {
            newNode.next = temp.Next;
            temp.Next = newNode;
        }
```

Delete the first Node of LL

```
Public void pop-Front () {
    if ( head != null) {
```

```
        Node temp = head;
        head = head.next;
        temp = null;
    }
}
```

Delete the Last Node

```
Public void pop_back() {
    if (head.next == null) {
        head = null;
    }
    Node temp = head;
    while (temp.next.next != null) {
        temp = temp.next;
    }
```

```
Node lastNode = temp.next;

temp.next = null;
lastNode = null;
}
```

Delete a node at the given position

```
Public void Pop_atPosition (int position) {
if ( position == 1 && head != null) {
        Node temp= head;
        head = head.Next;
        temp= null;
    }
else {
    Node temp = head;
```

```
for (int i = 1; i < position - 1 ; i++) {
        if (temp != null) {
            temp = temp.next ;
        }
}
if (temp != null && temp.next.next != null) {
    node nodeToDelete = temp.next ;
    temp.next = temp.next.next ;
    nodeToDelete = null ;
}
}
```

Delete all nodes of the LL

```
Public void DeleteALLNodes {
    Node temp = new Node () ;
```

```
while (head != null) {
        temp = head ;
        head = head.next ;
        temp = null ;
    }
}
```

Count nodes

```
public int CountNodes() {
    Node temp = head;
    int i = 0;
    while (temp != null) {
        temp = temp.next;
        i++ ;
    }
return i;
}
```

Delete Even Nodes of the List

```
public void deleteEvenNodes() {

    if (head != null) {

        Node oddNode = head;
        Node EvenNode = head.Next;

        while (oddNode != null && EvenNode != null) {

            oddNode.next = EvenNode.next;
            EvenNode = null;
            oddNode = oddNode.Next;
            if (oddNode != Null) {

                EvenNode = oddNode.next;
            }
        }
    }
}
```

Deleting Even Node of a linked list

```
Public void deleteOddNode() {
    if (head != Null) {
        Node temp = head;
        head = head.Next;
        temp = null;
    if (head != null) {
        Node EvenNode = head;
        Node oddNode = head.next;
        while (EvenNode != null && oddNode != null) {
            EvenNode.next = odd.next;
            oddNode = null;
            EvenNode = EvenNode.Next;
```

```
        if (EvenNode != null) {
            oddNode = EvenNode.next;
        }
    }
}
```

Search an Element

```
Public void SearchElement(int val) {
int i=0; bool = false; // i is for index.
Node temp = head;
if(head != null) {
    while( temp !=null) {
        i++;
        if(temp.data = val) {
            found = true;
```

```
                    break;
                }
            temp= temp.next;
        }
    }
```

Reverse the Linked List;

```
Public void reverseLinkedList () {
    if (head != null) {
        Node prevNode = head;
        Node nextNode = null;
        Node CurrNode = head.Next;
        // make the head next to null
        prevNode.next = null;
        while ( CurrNode != null) {
            nextNode= CurrNode.next;
```

```
        currNode.next = prevNode;
        prevNode = currNode;
        currNode = currNode.next;
    }

    head = prevNode;
  }
}
```

Swap 2 Node's value

```
Public void SwapNodes (int Node1, int Node2) {

  // Get the length of the linked list
  int n = 0;                          | Node Node1 = head;
  Node temp = head;                   | Node Node2 = head;
  while (temp != null) {              | for(int i=0; i< Node1; i++) {
                                      |     Node1 = Node1.next;
                                      |   }
                                      | for( int i=0; i<Node2; i++) {
```

```
        i++; temp=temp.next;
    }

// Check if the parameter values are valid
if( Node1 > n || node1< 1 ||
    Node2 > n || node2< 1) {
    return
}
```

```
Node2 = Node2.next;
}

int val = node1.data;
node1.data = node2.data;
node2.data = val;
}
```

Delete first node by key of LL

```
Public void pop_first(int key){
    Node temp = head;
    if(temp != null){
        if(temp.data == key){
            Node nodeToDelete = head;
```

```
else{
    while(temp.next != null){
        if(temp.next.data == key){
            Node nodeToDelete = temp.next;
            temp.next = temp.next.next;
            nodeToDelete = null;
            break;
```

```
        nodeToDelete = null;
    }
```

```
    {
    temp = temp.next;
    }
}
```

Delete last node by key of the linked list

```
Public void pop_Last(int key) {
    if (head != null) {
        Node lastNode, PrevToLast, temp;
        PrevToLast = null;
        lastnode = null;
    if (head.data == key) {
        Lastnode = head; }
    temp = head;
    while (temp.next != null) {
    if (lastnode != null) {
    if (lastnode == head) {
        head = head.next;
    }
    else {
    node temp = LastNode;
    PrevToLast.next = lastNode.next;
    temp = null;
}
```

```
        if (temp.next.data == key){
            PrevToLast = temp;
            LastNode = temp.next;
        }
        temp = temp.next;
    }
```

Delete all nodes by key of the LL

```
Public void pop_all (int key) {
    Node temp = new Node();
    if (head != null) {
        while (head.data == key){
            head = head.next;
        }
        temp = head;
        while (temp.next != null) {
```

→ `if (temp != null) {`

```
if (temp.next.data == key) {
        node deletenode = temp.next;
        temp.next = temp.next.next;
        deletenode = null;
    }
else {
        temp = temp.next;
    }
}
```