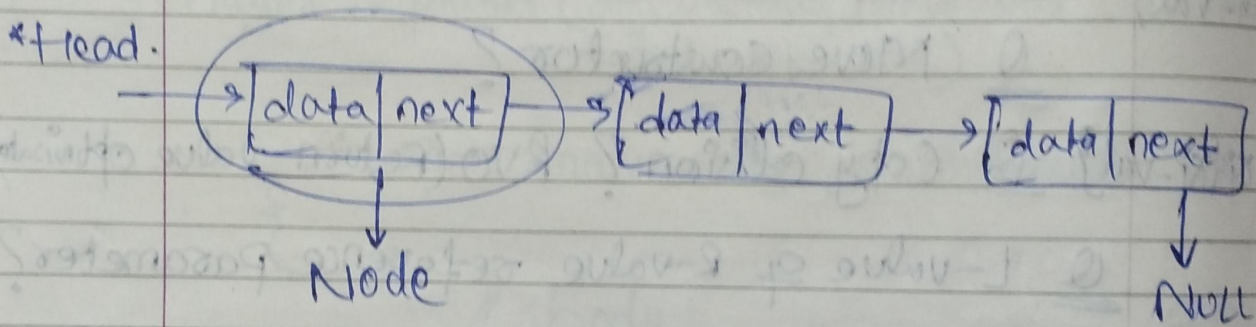
→ collection of Node

11 Linked List (L.L)

A linked list is a linear data structure in which the element are not stored at contigous memory location. The element in a linked list are linked using pointer



⇒ Linked list → Dynamic DS
↓ ↳ grow/shrink at run time
Better than vector

(*) Code :

```
/* Adding the Node at beginning, end, Pos.
 * Implementation of the Node
 */

#include <iostream>
using namespace std;
```

```
class Node {
    Public:
        int data;  // contain data
        Node *next;  // pointer pointing to
                        next node

    // constructor
        Node (int data) {

            this -> data = data;
            this -> next = NULL;
            }
};
    int main() {

    // Pointer to the instance of the class

        Node *node1 = new Node (18);

        cout << node1 -> data << endl; // 18
        cout << node1 -> next << endl; // 0

        return 0;

    }
```

(*) // function to Add new node at beginning

```
void InsertAtHead (Node *&head , int d)
{ // new node create
    Node * temp = new Node(d);
    temp->next = head ;
    head = temp;
}
```

(*) // function to print the linked list

```
void print ( node* &head){
    Node* temp = head;
    while ( temp != NULL){
    cout<< temp -> data << " ";
    temp= temp->next;
    }
    cout<<endl;
}
```

```
if(temp->next == NULL){
InsertAtEnd(tail, d);
return;
}
```

(*) // function to Add new node at the end.

```
Void InsertAtTail (Node* &tail, int d){
    // new node creation
    Node *temp = new Node(d);
    tail->next = temp;
    tail = temp;
}
```

(*) // function to Add new node at a position

```
void InsertAtPosition( Node* &head, Node* &tail, int position, int d)
{
    Node *temp = head;
    int cnt = 1;
    if(position==1){
        InsertAtHead(head,d);
        return; }
    while( cnt < position-1){
        temp = temp->next;
        cnt++;
    } // Inserting at the End.
    // Creating a node for d
    Node * nodeToInsert = new Node(d);
```

```
    nodeToInsert->next = temp->next;
    temp->next = nodeToInsert;
}
```

(*) // destructor in the class to free the memory.

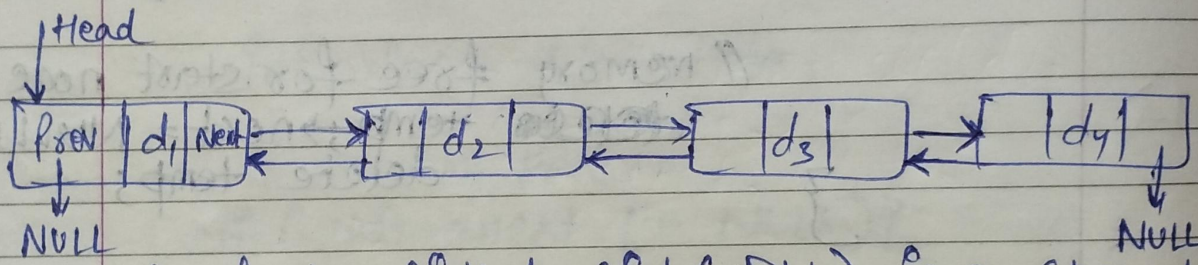
```
~Node(){
    int value = this->data;
    // memory free
    if(this->next != NULL){
        delete next;
        this->next = NULL;
    }
    cout << "memory is free for node with the data value" << value;
}
```

// function to delete the node at any position.

```
void deleteNode(int position, Node *&head)
{
    // deleting first or start Node
    if(position == 1){

        Node *temp= head;
        head = head->next;

        // memory free for start node
        temp->next= NULL;
        delete temp;
    }
    else{
        // deleting for any node at middle
        or at last

        Node *curr= head;
        Node *prev= NULL;

        int cnt=1;
        while(cnt< position){
            prev= curr;
            curr= curr->next;
            cnt++;
        }
```

```
Prev->next = Curr->next;
Curr->next = NULL;
delete Curr;
}
}
```

Doubly linked list



A doubly linked list (DLL) is a special type of linked list in which each node contains a pointer to the previous node as well as the next node of the linked list.

```
/*
* Doubly linked list implementation
* Insertion of node at any position
* Deletion of node at any position
* printing the length and linked list
*/
```

```
#include <iostream>
using namespace std;

class Node {

    public:
    int data;
    Node *next;
    Node *prev;

    // constructor
    Node(int d) {

        this->data = d;
        this->next = NULL;
        this->prev = NULL;

    }

    // destructor
    ~Node() {

        int value = this->data;

        // memory released
        if (this->next != NULL) {

            delete next;
            this->next = NULL;

        }

        cout << " Memory is released for the data: " << value << endl;
```

```
    }
};
```

(o) // Method to insert the node at the beginning

```
void InsertAtHead(Node* &head, Node* &tail, int d) {

    // empty list
    if(head == NULL) {
        Node* temp = new Node(d);
        head = temp;
        tail = temp;
    }

    // Already some nodes present
    else {
        // creating a new Node
        Node* temp = new Node(d);
        temp->next = head;
        head->prev = temp;
        head = temp;
    }
}
```

(o)
```
// Method to Insert the Node at the last
void InsertAtTail(Node* &head, Node* &tail,
                  int d) {

    // empty list
    if(tail == NULL){

        Node * temp = new Node(d);
        head = temp;
        tail = temp;
    }
    else {
        Node * temp = new Node(d);
        tail -> next = temp;
        temp -> prev = tail;
        tail = temp;
    }
}
```

(o)
```
// Method to add the node at any position
void InsertAtPosition(Node* &head, Node* &tail,
                      int position, int d) {

    // temp is a pointer used to traverse
    Node * temp = head;
    int cnt = 1;
```

```
// Inserting at the first position
if(position == 1){
    InsertAtHead(head, tail, d);
    return;
}

// traversing to the position at which
node should be inserted
while(cnt < position - 1){
    temp = temp->next;
    cnt++;
}

// Inserting at the last position
if(temp->next == NULL){
    InsertAtTail(head, tail, d);
    return;
}

// Creating a node to hold the d,
nodeToInsert is pointer holding the data
```

```
Node *nodeToInsert = new Node(d);

    nodeToInsert->next = temp->next;
    (temp->next)->prev = nodeToInsert;
     temp->next = nodeToInsert;
     nodeToInsert->prev = temp;

}
```

(0) // Method to get the length of the linked list

```
int getLength( Node * head){
        int len=0;
        Node *temp = head;
        while ( temp != NULL){
            len++;
            temp = temp->next;
        }
        return len;
}
```

(0) // Traversing and Method to print the data

```
void print( Node * head){
        Node *temp = head;
```

```
while (temp != NULL){
    cout<< temp -> data << " ";
    temp = temp -> data;
}
cout<<endl;
}
```

(6) // Method to delete any node at any position.

```
void DeleteNode (Node*& head, Node& data
                 int Position)
{
// creating deleting the first node
if (position == 1){
// creating a temporary pointer and pointing
   to the head
Node *temp = head;
(temp -> next) -> prev = NULL;
head = temp -> next;
temp -> next = NULL;
```

```
        delete temp;
    }

    // deleting for any node at middle or at
       the last
    else {
        Node * curr = head;
        Node * prev = NULL;

        int cnt = 1;

        // traversing to the position of the
           delete Node
        while ( cnt < position) {
            prev = curr;
            curr = curr->next;
            cnt++;
        }
        curr->prev = NULL;
        prev->next = curr->next;

        // Releasing the memory
        curr->next = NULL;
        delete curr;
    }
}
```

```
int main(){
// creation of an object to the instance of
   the class
// Node *node1 = new Node(10);
// creating the head pointer
    Node *head = NULL;
// creating the tail pointer
    Node *tail = NULL;
    print(Head);
  InsertAtHead(head, tail, 65);
    print(Head);
  InsertAtTail(head, tail, 97);
    print(Head);
  InsertAtTail(head, tail, 105);
    print(Head);
  Insert At Position(head, tail, 4, 12);
    print(Head);
    return 0;
}
```