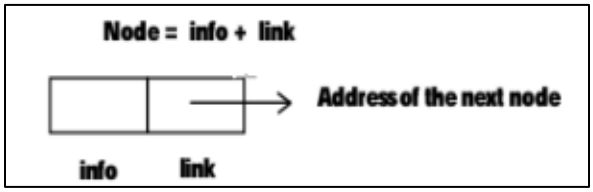
UNIT-1
Linked List

Linked List:

- A Linked List is a linear data structure, in which the elements (data) are not stored at contiguous (Sequential) memory location.
- The elements in a linked list are linked using pointer (address).
- Linked list is a very commonly used data structure which consists of group of nodes.
- Each node has some information. A node contains 2 fields, 1st field is used to store data and 2nd field is used to store address of next node.

The pictorial representation of the node is shown below:



Applications of linked list in real world-

- Image viewer – Previous and next images are linked, hence can be accessed by next and previous button.
- Previous and next page in web browser – We can access previous and next url searched in web browser by pressing back and next button since, they are linked as linked list.
- Music Player – Songs in music player are linked to previous and next song. You can play songs either from starting or ending of the list.

Application of Doubly Linked Lists:

- Redo and undo functionality.
- Use of Back and forward button in a browser.

Advantages of Linked List:

- **Dynamic data structure:** A linked list is a dynamic arrangement so it can grow and shrink at runtime by allocating and deallocating memory. So there is no need to give the initial size of the linked list.
- **No memory wastage:** In the Linked list, efficient memory utilization can be achieved since the size of the linked list increase or decrease at run time so there is no memory wastage and there is no need to pre-allocate the memory.
- **Implementation:** Linear data structures like stack and queues are often easily implemented using a linked list.
- **Insertion and Deletion Operations:** Insertion and deletion operations are quite easier in the linked list. There is no need to shift elements after the insertion or deletion of an element only the address present in the next pointer needs to be updated.

Disadvantages of Linked List:

- **Memory usage:** More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.
- **Traversal:** In a Linked list traversal is more time-consuming as compared to an array. Direct access to an element is not possible in a linked list as in an array by index. For example, for accessing a node at position n, one has to traverse all the nodes before it.
- **Random Access:** Random access is not possible in a linked list due to its dynamic memory allocation.

Types of Linked List:

There are mainly three types of linked lists:

1. Singly Linked List/One Way Linked List
2. Doubly Linked List/Two Way Linked List
3. Circular Linked List

1. Singly Linked List/One Way Linked List:

- In Singly linked list each node has 2 fields one for data and other is for address of next node
- Here, we can only traverse in *one direction* due to the linking of every node to its next node.
- Head node will always store the address of first node.
- Link/Next field of last node will always be NULL because it won't point to any other node.

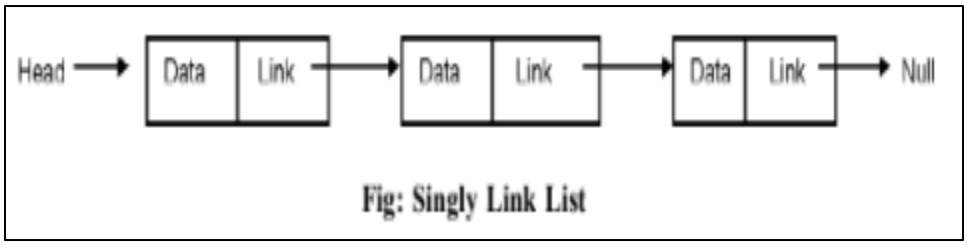


Fig: Singly Link List

The pictorial representation of a singly linked list consisting of the items 50, 20, 45, 10 and 80 is shown in the figure,

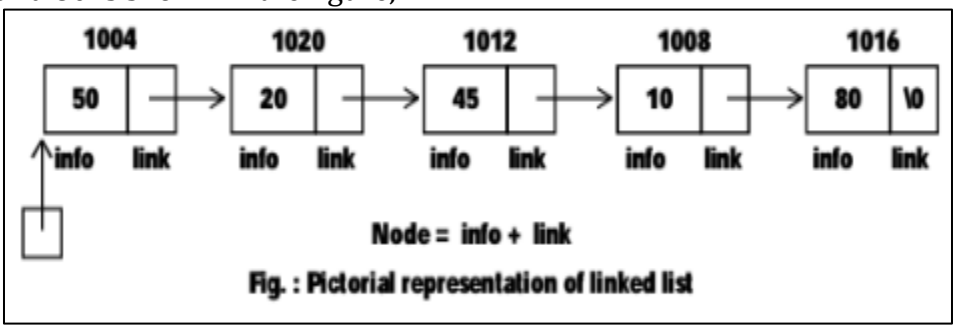


Fig. : Pictorial representation of linked list

2. Doubly Linked List/Two Way Linked List:

- In doubly linked list each node has 3 fields one for data and other two for address of previous and next node.
- Here, we can traverse in both directions.
- Head node will always store the address of first node.
- Previous field of first node and Next field of last node will always be NULL in doubly linked list.

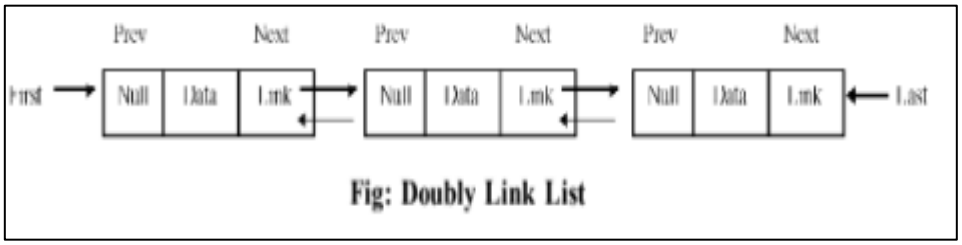


Fig: Doubly Link List

3. Circular Linked List:

- A circular linked list is that in which the last node contains the pointer/address to the first node of the list.

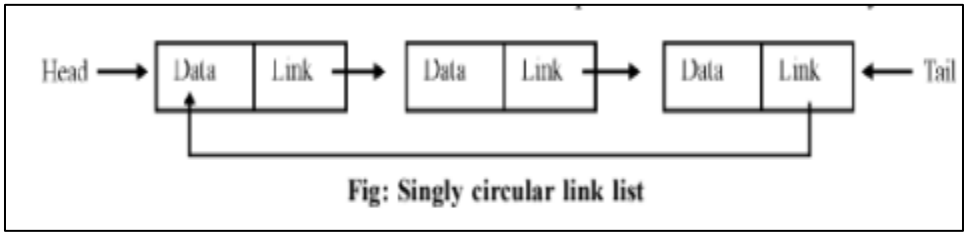
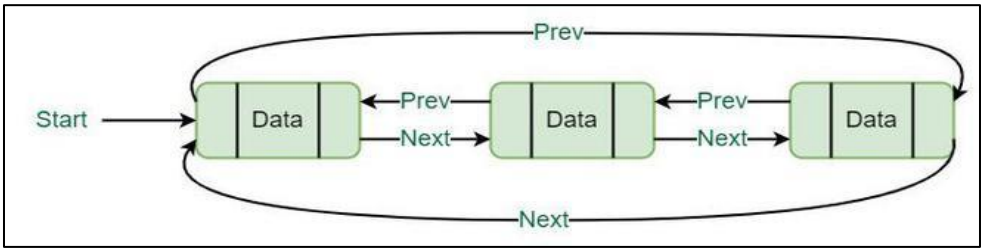


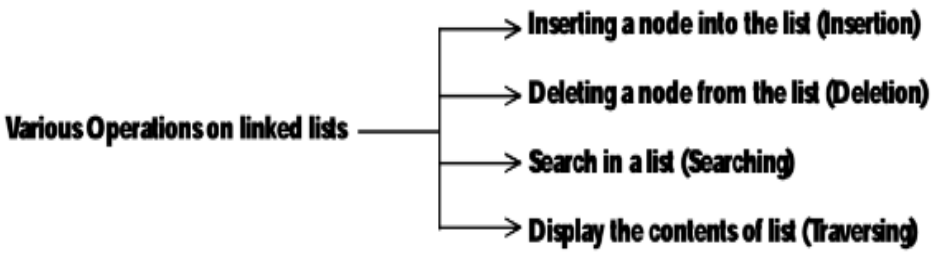
Fig: Singly circular link list



Doubly Circular Linked List

Operations on Linked List:

The basic operations that can be performed on a linked list are shown below:



1. Insert Elements to a Linked List:

You can add elements to either the beginning, middle or end of the linked list.

a. Insert at the beginning

- Allocate memory for new node
- Store data
- Change next of new node to point to head
- Change head to point to recently created node

b. Insert at the End

- Allocate memory for new node
- Store data
- Traverse to last node
- Change next of last node to recently created node

c. Insert at the Middle

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

2. Delete from a Linked List

You can delete either from the beginning, end or from a particular position.

a. Delete from beginning

- Point head to the second node

b. Delete from end

- Traverse to second last element
- Change its next pointer to null

c. Delete from middle

- Traverse to element before the element to be deleted
- Change next pointers to exclude the node from the chain

3. Search an Element on a Linked List

You can search an element on a linked list using a loop using the following steps. We are finding item on a linked list.

- Make head as the current node.
- Run a loop until the current node is NULL because the last element points to NULL.
- In each iteration, check if the key of the node is equal to item. If it the key matches the item, return true otherwise return false.

Singly Linked List Programs:

Program1: Program to add node in the beginning and delete first node:

Solution:

```
#include<iostream>
using namespace std;
struct node
{
	int data;
	node *link;
};
node *start=NULL;
void insert()
{
	node *n=new node;
	cout<<"Enter data:";
	cin>>n->data;
	n->link=NULL;
	if(start==NULL)
	{
		start=n;
		cout<<"\n new node inserted \n";
	}
	else
	{
		n->link=start;
		start=n;
		cout<<"\n new node insreted \n ";
	}
}
void del()
{
	if(start==NULL)
	{
		cout<<"\n no node available \n";
	}
	else
	{
	 node *t;

	  t=start;
	  start=start->link;
	  delete t;
	  cout<<"\n node deleted\n";
	}
}
void display()
{
```

```
        if(start==NULL)
        {
                cout<<"\n no node available \n";
        }
        else
        {       node *r;
          r=start;
                while(r!=NULL)
                {
                        cout<<r->data<<"->";
                        r=r->link;
                }
                cout<<"NULL\n";
                }
}
int main()
{
        int ch;
        while(1)
        {
        cout<<"\n1.insert\n2.delete\n3.display\n4.Exit\n";
        cin>>ch;
        if(ch==1)
        insert();
        else if(ch==2)
        del();
        else if(ch==3)
        display();
        else
        exit(0);
        }
}
```

Program 2: Program to add node at the end and delete first node:

```
#include<iostream>
using namespace std;
struct node
{
        int data;
        node *link;
};
node *start=NULL;
void insert()
{
        node *n=new node;
        cout<<"Enter data:";
        cin>>n->data;
```

```
        n->link=NULL;
        if(start==NULL)
        {
                start=n;
                cout<<"\n new node inserted \n";
        }
        else
        {
         node *r;
         r=start;
         while(r->link!=NULL)
         {
                r=r->link;
         }
         r->link=n;
         cout<<"\n node inserted \n";
        }
}
void del()
{
        if(start==NULL)
        {
                cout<<"\n no node available \n";
        }
        else
        {
         node *t;

          t=start;
          start=start->link;
          delete t;
          cout<<"\n node deleted\n";
        }
}
void display()
{
        if(start==NULL)
        {
                cout<<"\n no node available \n";
        }
        else
        {       node *r;
           r=start;
                while(r!=NULL)
                {
                        cout<<r->data<<"->";
```

```
                r=r->link;
            }
            cout<<"NULL\n";
            }
}
int main()
{
    int ch;
    while(1)
    {
    cout<<"\n1.insert\n2.delete\n3.display\n";
    cin>>ch;
    if(ch==1)
    insert();
    else if(ch==2)
    del();
    else if(ch==3)
    display();
    else
    exit(0);
    }
}
```

Program 3: Program to add node at the end and delete last node:

```
#include<iostream>
using namespace std;
struct node
{
    int data;
    node *link;
};
node *start=NULL;
void insert()
{
    node *n=new node;
    cout<<"Enter data:";
    cin>>n->data;
    n->link=NULL;
    if(start==NULL)
    {
        start=n;
        cout<<"\n new node inserted \n";
    }
    else
    {
     node *r;
     r=start;
```

```
        while(r->link!=NULL)
        {
                r=r->link;
        }
        r->link=n;
        cout<<"\n node inserted \n";
      }
}
void del()
{
      if(start==NULL)
      {
                cout<<"\n no node available \n";
      }
      else
      {
        node *t,*r;
        if(start->link==NULL) //if only one node exists
        {
                delete start;
                start=NULL;
         cout<<"\n node deleted \n";
        }
        else
        {
        t=start;
        while(t->link!=NULL) //to reach till last node
        {
                 t=t->link;
        }
        r=start;
        while(r->link!=t)
        {
                r=r->link;
        }
        r->link=NULL;
        delete t;
        cout<<"\n node deleted \n";
        }

      }
}
void display()
{
      if(start==NULL)
      {
                cout<<"\n no node available \n";
      }
      else
      {         node *r;
```

```
        r=start;
                while(r!=NULL)
                {
                        cout<<r->data<<"->";
                        r=r->link;
                }
                cout<<"NULL\n";
                }
}
int main()
{
      int ch;
      while(1)
      {
      cout<<"\n1.insert\n2.delete\n3.display\n";
      cin>>ch;
      if(ch==1)
      insert();
      else if(ch==2)
      del();
      else if(ch==3)
      display();
      else
      exit(0);
      }
}
```

Program 4: INSERTION & DELETION At Specified Position In SINGLY LINKED LIST

```
#include<iostream>
using namespace std;
struct node
{
      int data;
      node *link;
};
node *start=NULL;
int p=0,count=0;
void insert()
{
   node *n=new node;
   cout<<"\nEnter data:\n";
   cin>>n->data;
   n->link=NULL;
      cout<<"enter position:";
      cin>>p;//1
      if(p==1 && start==NULL) //if list is empty
      {
   start=n;
   count++;
```

```
	cout<<"\n node inserted\n";
	}
	else if(p==1 && start!=NULL) //to insert data at 1st position if we have more nodes available
	{	node *u;
		u=start;
		start=n;
		n->link=u;
		count++;
		cout<<"\n node inserted\n";
	}
	else if(p>count || p<=0)
	{
	  cout<<"\n can not insert node\n";
	}

	else //to insert node at any position except 1st position
	{
		node *r,*t;
		int i=1;
		r=start;
		while(i<p-1)
		{
			r=r->link;
			i++;
		}
		t=r->link;
		r->link=n;
		n->link=t;
		count++;
		cout<<"\n node inserted\n";
	}
}
//=======================================
void del()
{
	cout<<"\nEnter position\n";
	cin>>p;
	if(p>count || p<=0 || start==NULL)
	{
		cout<<"\n no node available at this position\n";
	}
	else if(p==1) //to delete first node
	{
		node *t;
		t=start;
		start=t->link;
		delete t;
		count--;
		cout<<"\n node deleted \n";
	}
```

```
        else
        {
                int i;
                node *r,*t,*u;
                r=start;
                while(i<p-1)
                {
                        r=r->link;
                        i++;
                }
                t=r->link;
                u=t->link;
                r->link=u;
                delete t;
                count--;
                cout<<"\n node deleted \n";
                cout<<"total node="<<count;
        }

}
//=========================================
void display()
{
        if(start==NULL)
        {
                cout<<"\n no node available \n";
        }
        else
        {       node *r;
           r=start;
                while(r!=NULL)
                {
                        cout<<r->data<<"->";
                        r=r->link;
                }
                cout<<"NULL\n";
                }
                cout<<"\ntotal node="<<count;
}
int main()
{
        int ch;
        while(1)
        {
        cout<<"\n1.insert\n2.delete\n3.display\n";
        cin>>ch;
        if(ch==1)
        insert();
        else if(ch==2)
        del();
```

```
        else if(ch==3)
        display();
        else
        exit(0);
        }
}
```

Program to search an element from linked list:

```
void search()
{
        int item;
        node *r;
        if(start==NULL)
        {
                Cout<<"\n**no node available**\n";
        }
        else
        {
                cout<<Enter item to search\n";
                cin<<item;
                r=start;
                while(r!=NULL)
                {
                        if(item==r->data)
                        {
                        Cout<< item<<" found in the list\n";
                         break;
                        }
                        else
                   r=r->link;
                }
                if(r==NULL)
                 cout<<item<< " Not found in the list\n";

        }
}
```

DOUBLY LINKED LIST:

Program 1: Program to insert node at the end, in the beginning and deletion of first node

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

struct node
{  node *prev;
        int data;
        node *next;
};
```

```
node *start=NULL;
//=========================
void insert_beg()
{
   node *n=new node;
   cout<<"\nEnter data:\n";
   cin>>n->data;
   n->prev=NULL;
   n->next=NULL;

   if(start==NULL)
   {
      start=n;
      cout<<"\n node inserted \n";
      }
      else
      {
               node *r;
               r=start;
               start=n;
               n->next=r;
               cout<<"\n node inserted \n";
      }
}
//=======================
void insert_end()
{
   node *n=new node;
   cout<<"\nEnter data:\n";
   cin>>n->data;
   n->prev=NULL;
   n->next=NULL;

      if(start==NULL)
   {
      start=n;
      cout<<"\n node inserted \n";
      }
      else
      {
               node *r;
               r=start;
               while(r->next!=NULL)
               {
                 r=r->next;
               }
```

```
                r->next=n;
                cout<<"\n node inserted \n";
        }

}
//=======================
void display()
{
        if(start==NULL)
        {
                cout<<"\n no node available \n";
        }
        else
        {       node *r;
                r=start;
                while(r!=NULL)
                {
                        cout<<r->data<<"->";
                        r=r->next;
                }
                cout<<"NULL\n";
        }
}
//=======================
void del_beg()
{
 if(start==NULL)
 {
  cout<<"\n can not delete\n";
 }
 else if(start->next==NULL) //if only one node exists
 {
        node *p;
        p=start;
        start=NULL;
        delete p;
        cout<<"\n node deleted \n";
 }
 else //if more than one node exists
 {
        node *t;
        t=start;
        start=start->next;
        start->prev=NULL;
        delete t;
        cout<<"\n node deleted\n";
```

```
 }
}

//=========================
int main()
{
      int ch;
      while(1)
      {
      cout<<"\n1.insert begining\n2.insert end\n3.delete begining\n4.display\n";
      cin>>ch;
      if(ch==1)
      insert_beg();
      else if(ch==2)
      insert_end();
   else if(ch==3)
      del_beg();
      else if(ch==4)
      display();
      else
      exit(0);
      }
}
```

Circular linked list program:

```
#include<iostream>
using namespace std;
struct node
{
      int data;
      node *link;
};
node *start=NULL;
int p=0,c=0;
//=========
void insert_beg()
{
 node *n=new node;
 cout<<"enter data\n";
 cin>>n->data;
 n->link=NULL;
 if(start==NULL)
 {
      start=n;
```

```
        n->link=start;
        cout<<"Node inserted\n";
        c++;
    }
    else
    {
        node *r,*t;
        r=t=start;
        start=n;
        n->link=t;
        while(r->link!=t)
        {
          r=r->link;
        }
          r->link=start;
         cout<<"Node inserted\n";
                c++;
    }
}
//===========
void insert_end()
{
  node *n=new node;
   cout<<"enter data\n";
  cin>>n->data;
  n->link=NULL;
  if(start==NULL)
  {
        start=n;
        n->link=start;
        cout<<"Node inserted\n";
                c++;
  }
  else
  {
        node *r;
        r=start;
         while(r->link!=start)
         {
                r=r->link;
         }
           r->link=n;
           n->link=start;
         cout<<"Node inserted\n";
                c++;
  }
```

```
}
//===========
void insert_mid()
{
node *n=new node;
cout<<"enter data\n";
cin>>n->data;
 n->link=NULL;

 cout<<"\nEnter position\n";
 cin>>p;
 if(p==1 && start==NULL)
 {
     start=n;
     n->link=start;
     c++;
     cout<<"\n node insreted \n";
 }
 else if(p==1 && start!=NULL)
 {
     node *r;
     r=start;
     while(r->link!=start)
     {
      r=r->link;
     }
     r->link=n;
     n->link=start;
     start=n;
     c++;
     cout<<"\n node insreted \n";
 }
 else if(p>c || p<=0)
 {
     cout<<"\n can not delete\n";
 }
 else
 {
     int i=1;
     node *r=start;
     while(i<p-1)
     {
              r=r->link;
              i++;
      }
      n->link=r->link;
```

```
        r->link=n;
        c++;
       cout<<"\n node inserted \n";
 }
}
//===========
void del_beg()
{
 if(start==NULL)
 {
       cout<<"node not available\n";
 }
 else if(start->link==start)
 {
       delete start;
       start=NULL;
       cout<<"\n node deleted \n";
       c--;
 }
 else
 {
       node *r,*q;
       q=r=start;
       start=start->link;
       while(r->link!=q)
       {
                r=r->link;
        }
        r->link=start;
        delete q;
        cout<<"\n node deleted \n";
                c--;
}
}
//=========
void del_end()
{
       if(start==NULL)
 {
       cout<<"node not available\n";
 }
 else if(start->link==start)
 {
       delete start;
       start=NULL;
       cout<<"\n node deleted \n";
```

```
                c--;
 }
 else
 {
      node *r,*t;
      r=start;
      while(r->link!=start)
      {
                r=r->link;
       }
       t=start;
       while(t->link!=r)
       {
                t=t->link;
       }
       t->link=start;
       delete r;
      cout<<"\n node deleted \n";
      c--;
 }
}
//==========
void del_mid()
{
   int i=1;
      cout<<"\n Enter position:\n";
   cin>>p;
      if(p>c || start==NULL || p<1)
      {
                        cout<<"node not available\n";
      }
      else if(p==1 && start->link==start)
      {
                        delete start;
      start=NULL;
      cout<<"\n node deleted \n";
      c--;
      }
      else if(p==1 && start!=NULL)
      {
                node *r,*q;
      q=r=start;
      start=start->link;
      while(r->link!=q)
      {
                r=r->link;
```

```
        }
        r->link=start;
        delete q;
        cout<<"\n node deleted \n";
        c--;
       }
       else
       {
        node *r,*q;
        r=start;
         while(i<p-1)
         {
                  r=r->link;
                  i++;
         }
                 q=r->link;
                 r->link=q->link;
                 c--;
                 cout<<"\n node deleted \n";
       }
}
//==========
void display()
{
 if(start==NULL)
 {
       cout<<"node not available\n";
 }
 else
 {
       node *r;
       r=start;
       do
       {
                 cout<<r->data<<" ";
                 r=r->link;
        }while(r!=start);
        }
}
//=========
int main()
{
       int ch;
       while(1)
       {
```

```
        cout<<"\n1.insert beginning\n2.insert end\n3.delete beginning\n4.delete
end\n5.dispaly\n6.insert_mid\n7.delete mid\n";
        cin>>ch;
        if(ch==1)
        insert_beg();
        else if(ch==2)
        insert_end();
     else if(ch==3)
        del_beg();
        else if(ch==4)
        del_end();
        else if(ch==5)
        display();
        else if(ch==6)
        insert_mid();
        else if(ch==7)
        del_mid();
        else
        exit(0);
        }
}
```

Polynomial Manipulation:

- Polynomial manipulations are one of the most important applications of linked lists.
- Polynomials are an important part of mathematics. A polynomial is a collection of different terms, each comprising coefficients, and exponents. It can be represented using a linked list. This representation makes polynomial manipulation efficient.

Polynomial Representation using linked list:

- A polynomial is composed of different terms where each of them holds a coefficient and an exponent.
- An essential characteristic of the polynomial is that each term in the polynomial expression consists of two parts:

1. one is the coefficient
2. other is the exponent

Example:

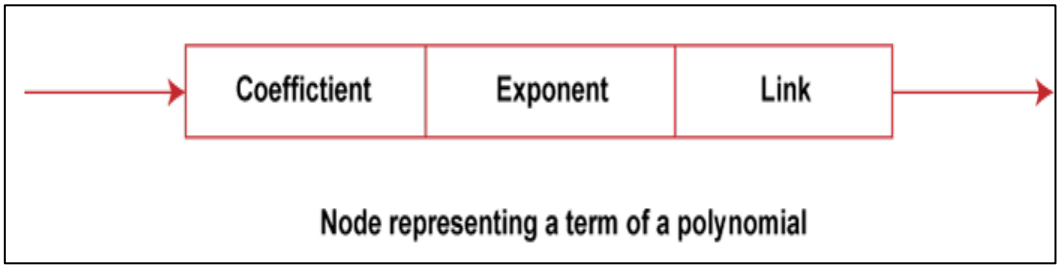
$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 is its exponential value.

- While representing a polynomial using a linked list, each polynomial term represents a node in the linked list.

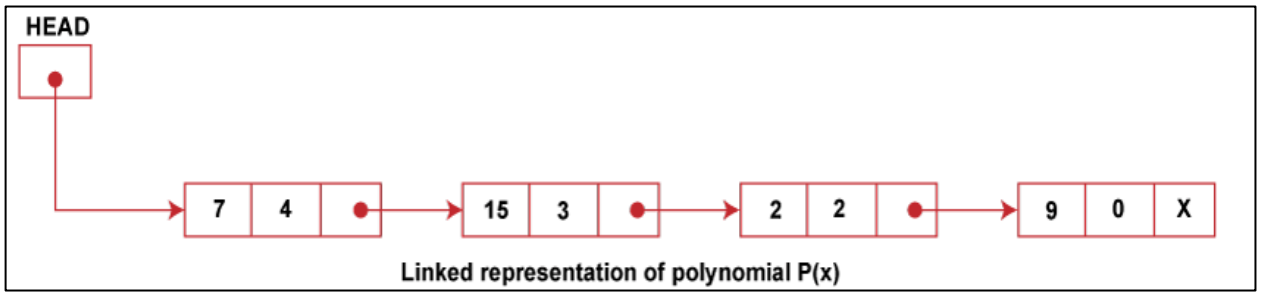
Each node of a linked list representing polynomial constitute three parts:

- The first part contains the value of the coefficient of the term.
- The second part contains the value of the exponent.
- The third part, LINK points to the next term (next node).

The structure of a node of a linked list that represents a polynomial is shown below:



Consider a polynomial $P(x) = 7x^2 + 15x^3 - 2x^2 + 9$. Here 7, 15, -2, and 9 are the coefficients, and 4,3,2,0 are the exponents of the terms in the polynomial. On representing this polynomial using a linked list, we have



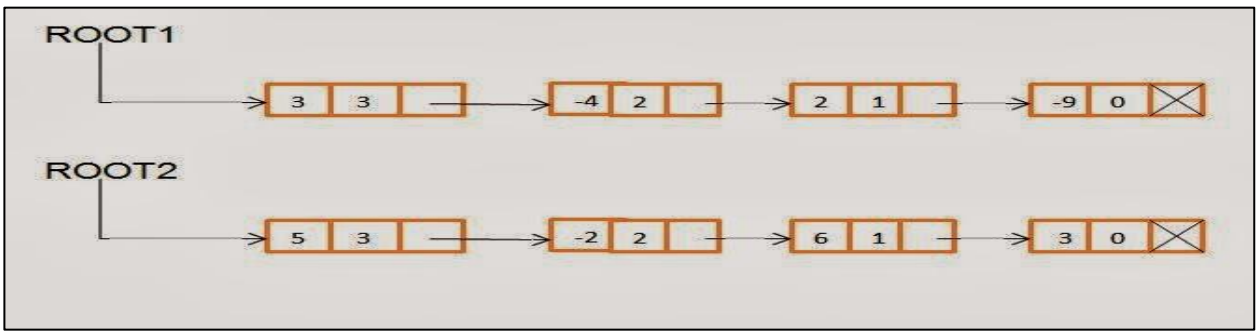
Observe that the number of nodes equals the number of terms in the polynomial. So we have 4 nodes.

Addition of Polynomials:

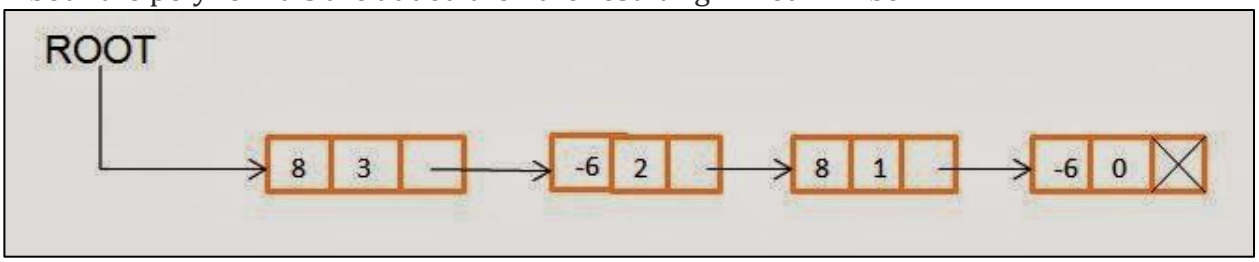
Let us consider an example an example to show how the addition of two polynomials is performed,

Polynomial 1: **$3x^3-4x^2+2x-9$**

Polynomial 2: **$5x^3-2x^2+6x+3$**



If both the polynomials are added then the resulting linked will be:



*****