LINKED LIST

Introduction to List and Linked Lists

- **List** is a term used to refer to a linear collection of data items. A List can be implemented either by using **arrays** or **linked lists**.

- Usually, a large block of memory is occupied by an array which may not be in use and it is difficult to increase the size of an array.

- Another way of storing a list is to have each element in a list contain a field called a **link** or **pointer**, which contains the address of the next element in the list.

- The successive elements in the list need not occupy adjacent space in memory. This type of data structure is called a **linked list**.

Linked List

- It is the most commonly used data structure used to store similar type of data in memory.

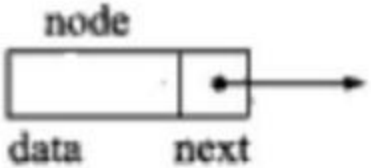
- The elements of a linked list are not stored in adjacent memory locations as in arrays.

- It is a linear collection of data elements, called **nodes**, where the linear order is implemented by means of **pointers**.

Linked List

- In a linear or single-linked list, a node is connected to the next node by a single link.

- A node in this type of linked list contains two types of fields
  - **data**: which holds a list element
  - **next**: which stores a link (i.e. pointer) to the next node in the list.



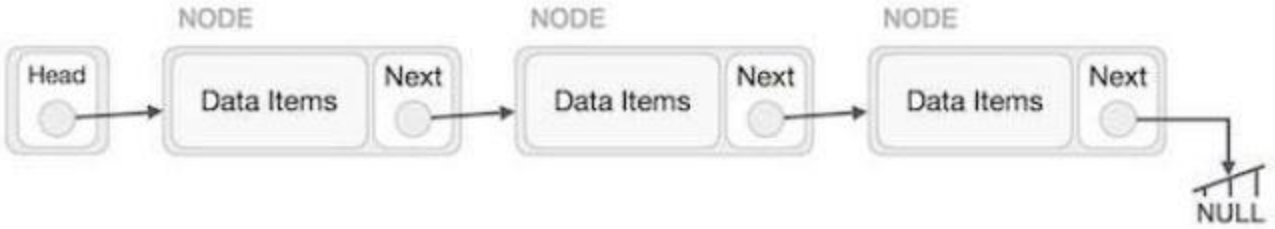
Linked List

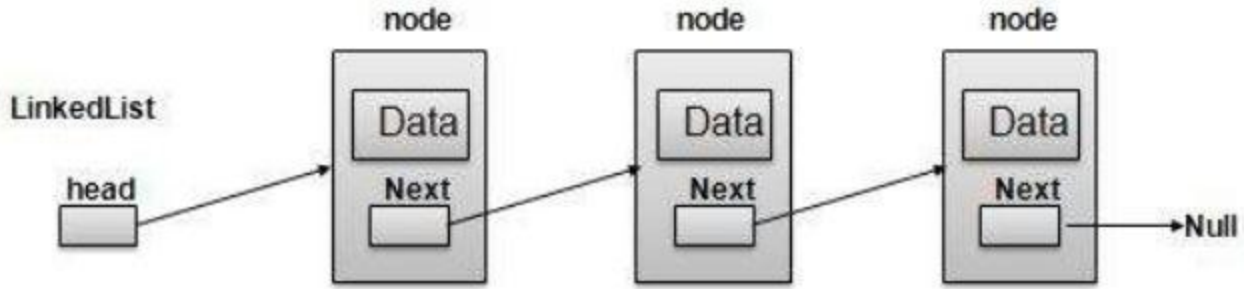
- The structure defined for a single linked list is implemented as follows:

```
struct Node{
        int   info;
        struct   Node   *   next;
}
```

- The structure declared for linear linked list holds two members
  - An integer type variable '**data**' which holds the elements and
  - Another type '**node**', which has next, which stores the address of the next node in the list.

Figurative Representation





Properties of Linked list

- The nodes in a linked list are not stored contiguously in the memory
- You don’t have to shift any element in the list
- Memory for each node can be allocated dynamically whenever the need arises.
- The size of a linked list can grow or shrink dynamically

Operations on Linked List

- **Creation:**
  - This operation is used to create a linked list
- **Insertion / Deletion**
  - At/From the beginning of the linked list
  - At/From the end of the linked list
  - At/From the specified position in a linked list
- **Traversing:**
  - Traversing may be either forward or backward
- **Searching:**
  - Finding an element in a linked list
- **Concatenation:**
  - The process of appending second list to the end of the first list

Types of Linked List

- Singly Linked List
- Doubly linked list
- Circular linked list
- Circular doubly linked list

Singly Linked List

- A singly linked list is a dynamic data structure which may grow or shrink, and growing and shrinking depends on the operation made.

- In this type of linked list each node contains two fields one is **data** field which is used to store the data items and another is **next** field that is used to point the next node in the list.

Creating a Linked List

- The **head** pointer is used to create and access unnamed nodes.

```
struct Node{
                    int info;
                    struct Node* next;
};
typedef struct Node NodeType;
NodeType* head;
head=(NodeType *) malloc (sizeof( NodeType ) );
```

- The above statement obtains memory to store a node and assigns its address to head which is a pointer variable.

Creating a Node

- To create a new node, we use the malloc function to dynamically allocate memory for the new node.

- After creating the node, we can store the new item in the node using a pointer to that node.

```
Nodetype *p;
p=(NodeType *) malloc (sizeof( NodeType ) );
p->info=50;
p->next = NULL;
```

- Note that p is not a node; instead it is a pointer to a node.

Creating an empty list

```
void createEmptyList(NodeType *head)
{
        head=NULL;
}
```

OR SIMPLY

```
NodeType *head =Null;
```

Inserting an Element

- While inserting an element or a node in a linked list, we have to do following things:
  - Allocate a node
  - Assign a data to info field of the node.
  - Adjust a pointer

- We can insert an element in following places
  - At the beginning of the linked list
  - At the end of the linked list
  - At the specified position in a linked list

An algorithm to insert a node at the beginning of the singly linked list

Let *head be the pointer to first node in the current list

1. Create a new node using malloc function

   NewNode=(NodeType)malloc(sizeof(NodeType));*

2. Assign data to the info field of new node

   NewNode->info=newItem;

3. Set next of new node to head

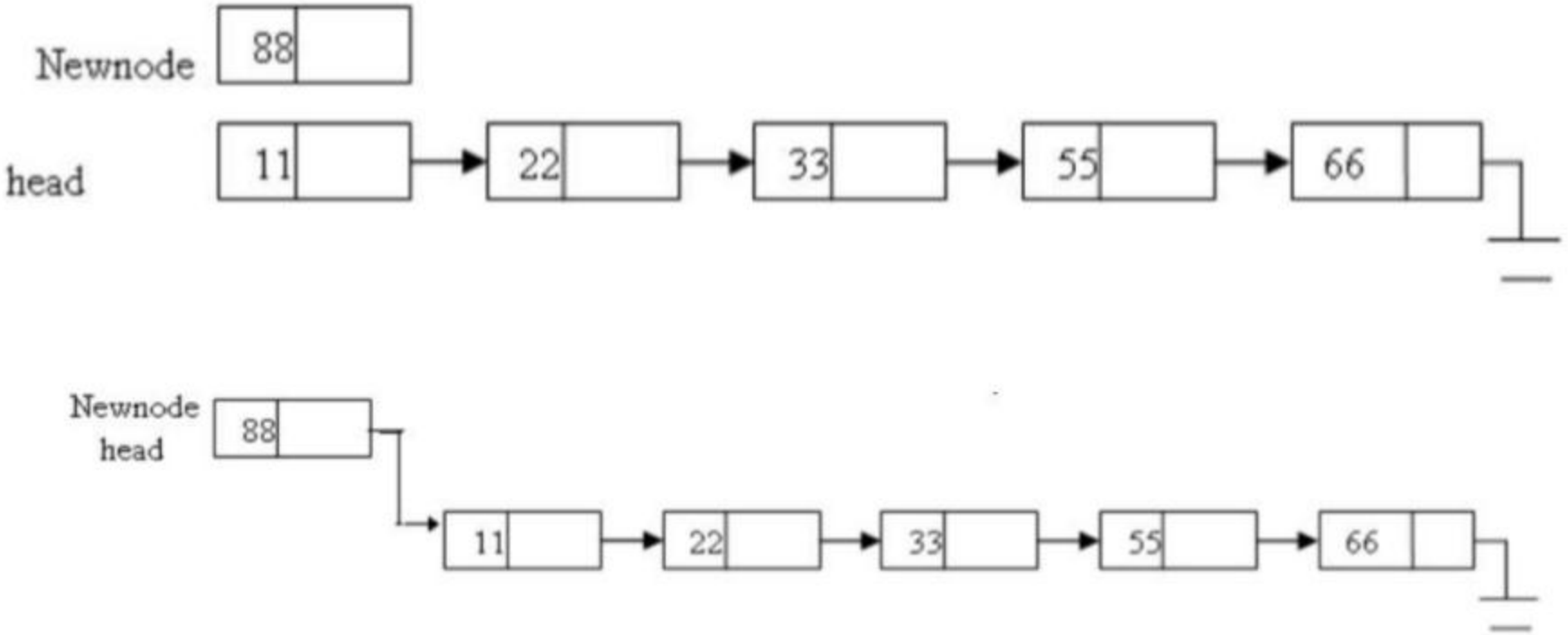
   NewNode->next=head;

4. Set the head pointer to the new node

   head=NewNode;

5. End

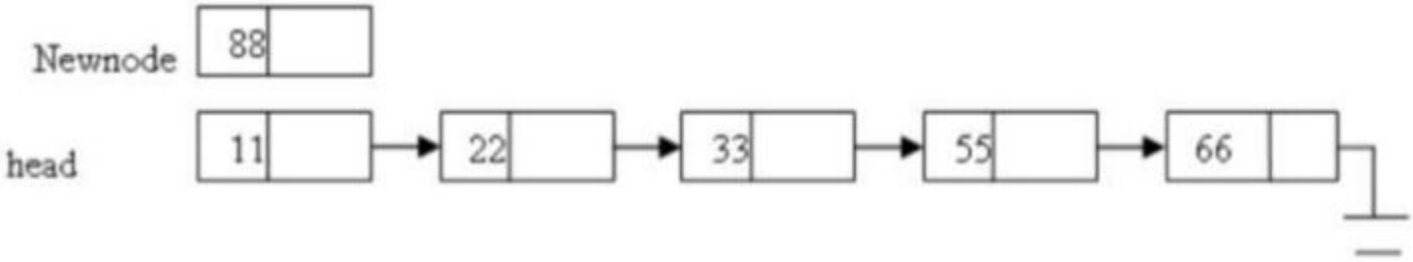
Inserting a node at the beginning of the singly linked list

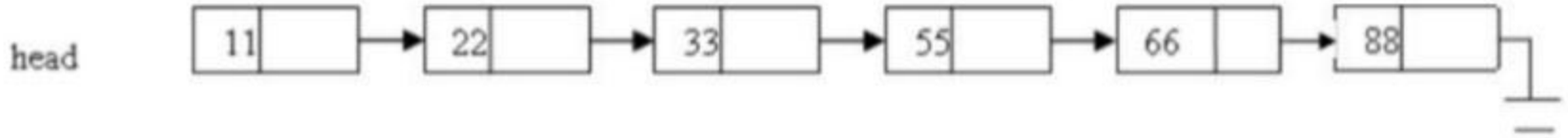


An algorithm to insert a node at the end of the singly linked list

let *head be the pointer to first node in the current list

1. **Create a new node using malloc function**
   NewNode=(NodeType)malloc(sizeof(NodeType));*
2. **Assign data to the info field of new node**
   NewNode->info=newItem;
3. **Set next of new node to NULL**
   NewNode->next=NULL;
4. **if (head ==NULL) then**
   Set head =NewNode.and exit.
5. **Set temp=head;**
6. **while(temp->next!=NULL)**
   temp=temp->next; //increment temp
7. **Set temp->next=NewNode;**
8. **End**

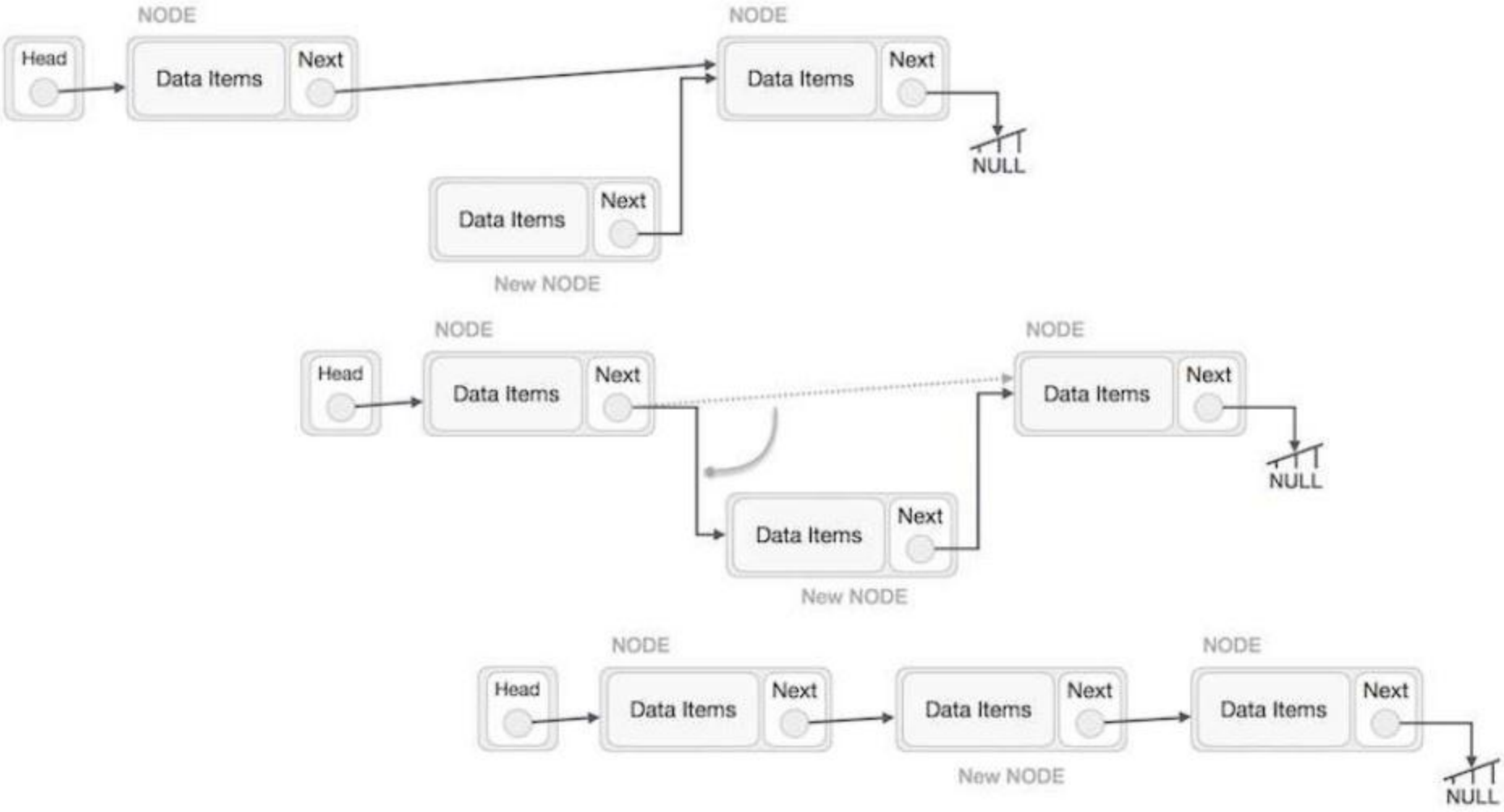

Newnode
88
head
11
22
33
55
66


head
11
22
33
55
66
88

An algorithm to insert a node after the given node in singly linked list

let *head be the pointer to first node in the current list and *p be the pointer to the node after which we want to insert a new node.

1. **Create a new node using malloc function**
   NewNode=(NodeType)malloc(sizeof(NodeType));*
2. **Assign data to the info field of new node**
   NewNode->info=newItem;
3. **Set next of new node to next of p**
   NewNode->next=p->next;
4. **Set next of p to NewNode**
   p->next =NewNode
5. **End**


NODE
Head
Data Items
Next
NODE
Data Items
Next
NULL
Data Items
Next
New NODE
NODE
Head
Data Items
Next
NODE
Data Items
Next
NULL
Data Items
Next
New NODE
NODE
Head
Data Items
Next
Data Items
Next
NODE
Data Items
Next
New NODE
NULL

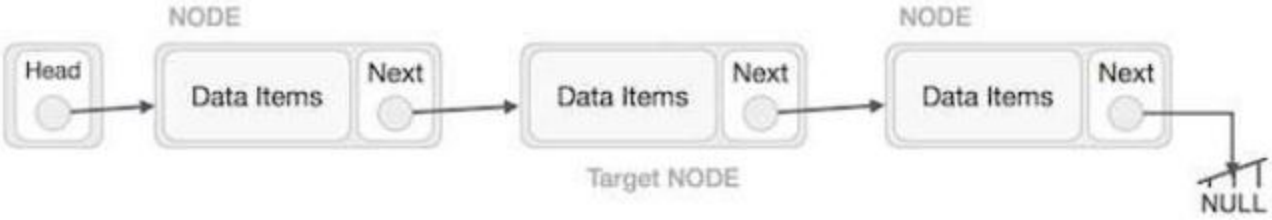
An algorithm to insert a node at the specified position in a linked list

let *head be the pointer to first node in the current list

1. **Create a new node using malloc function**
   NewNode=(NodeType)malloc(sizeof(NodeType));*
2. **Assign data to the info field of new node**
   NewNode->info=newItem;
3. **Enter position of a node at which you want to insert a new node. Let this position is pos.**
4. **Set temp=head;**
5. **if (head ==NULL)then**
   printf("void insertion"); and exit(1).
6. **for(i=1; i<pos; i++)**
   temp=temp->next;
7. **Set *NewNode->next=temp->next;***
   set temp->next =NewNode..
8. **End**

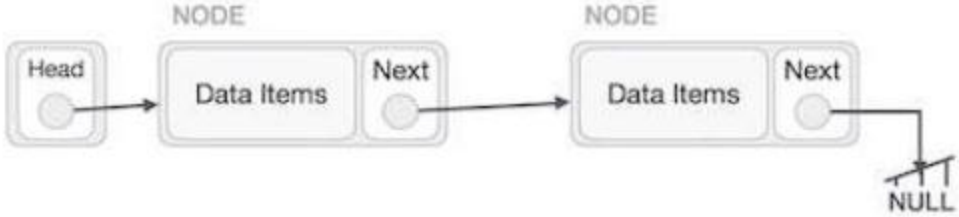
Deleting Nodes

- A node may be deleted:
  - From the beginning of the linked list
  - From the end of the linked list
  - From the specified position in a linked list


NODE
Head
Data Items
Next
Data Items
Next
NODE
Data Items
Next
Target NODE
NULL


NODE
Head
Data Items
Next
Data Items
Next
NODE
Data Items
Next
Target NODE
NULL


NODE
Head
Data Items
Next
Data Items
Next
NODE
Data Items
Next
Target NODE
NULL


NODE
Head
Data Items
Next
NODE
Data Items
Next
NULL

Deleting first node of the linked list

An algorithm to deleting the first node of the singly linked list:

let *head be the pointer to first node in the current list

1. If(head==NULL) then
   print "Void deletion" and exit
2. Store the address of first node in a temporary variable **temp.**
   temp=head;
3. Set head to next of head.
   head=head->next;
4. Free the memory reserved by temp variable.
   free(temp);
5. End

Deleting the last node of the linked list:

An algorithm to deleting the last node of the singly linked list:

let *head be the pointer to first node in the current list

```
1.  If(head==NULL) then            //if list is empty
        print "Void deletion" and exit
2.  else if(head->next==NULL) then      //if list has only one node
      Set temp=head;
      print deleted item as,
      printf("%d" ,head->info);
      head=NULL;
      free(temp);
3.  else
     set temp=head;
     while(temp->next->next!=NULL)
        set temp=temp->next;
      End of while
       free(temp->next);
      Set temp->next=NULL;
4.  End
```

An algorithm to delete a node after the given node in singly linked list:

let *head be the pointer to first node in the current list and *p be the pointer to the node after which we want to delete a new node.

1. ***if****(p==NULL or p->next==NULL) then*
   print "deletion not possible and exit
2. set q=p->next
3. *Set p->next=q->next;*
4. **free**(q)
5. End

An algorithm to delete a node at the specified position in a singly linked list:

let *head be the pointer to first node in the current list

1. *Read position of a node which to be deleted, let it be pos.*
2. if head==NULL
   print "void deletion" and exit
3. Enter position of a node at which you want to delete a new node. Let this position is pos.
4. Set temp=head
   declare a pointer of a structure let it be *p
5. if (head ==NULL)then
   print "void ideletion" and exit
   otherwise;.
6. for(i=1; i<pos-1; i++)
   temp=temp->next;
7. *print deleted item is temp->next->info*
8. *Set p=temp->next;*
9. Set temp->next =temp->next->next;
10. free(p);
11. End

Searching an item in a linked list

- Let *head be the pointer to first node in the current list

1. **If head==Null**

   Print "Empty List"

2. **Else, enter an item to be searched as key**
3. **Set temp==head**
4. **While temp!=Null**

   If (temp->info == key)

   Print "search success"

   temp=temp->next;

5. **If temp==Null**

   Print "Unsuccessful search"

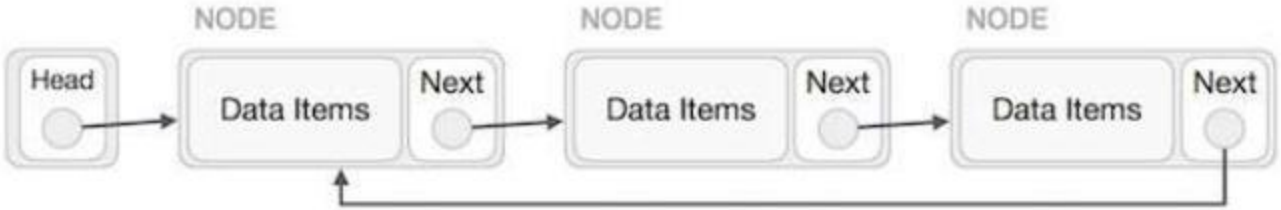
```
void searchItem()
{
    NodeType *temp;
    int key;
    if(head==NULL)
    {
            printf("empty list");
            exit(1);
    }
    else
    {
            printf("Enter searched item");
            scanf("%d" ,&key);
            temp=head;
            while(temp!=NULL)
            {
                    if(temp->info==key)
                    {
                            printf("Search successful");
                            break;
                    }
                    temp=temp->next;
            }
            if(temp==NULL)
            printf("Unsuccessful search");
    }
```

Circular Linked List

- A circular linked list is a list where the link field of last node points to the very first node of the list.

- Complicated linked data structure.

- A circular list is very similar to the linear list where in the circular list pointer of the last node points not Null but the first node.



C representation of circular linked list

- We declare structure for the circular linked list in the same way as linear linked list.

```
struct node
{
        int info;
        struct node *next;
};
typedef struct node NodeType;
NodeType *start=NULL:
NodeType *last=NULL:
```

Algorithms to insert a node in a circular linked list

Algorithm to insert a node at the beginning of a circular linked list:

1. Create a new node as
   newnode=(NodeType*)malloc(sizeof(NodeType));
2. if start==NULL then
   set newnode->info=item
   set newnode->next=newnode
   set start=newnode
   set last = newnode
   end if
3. else
   set newnode->info=item
   set newnode->next=start
   set start=newnode
   set last->next=newnode

   end else
4. End

Algorithm to insert a node at the end of a circular linked list

1. Create a new node as
   newnode=(NodeType*)malloc(sizeof(NodeType));
2. if start==NULL then
   set newnode->info=item
   set newnode->next=newnode
   set start=newnode
   set last newnode
   end if
3. else
   set newnode->info=item
   set last->next=newnode
   set last=newnode
   set last->next=start
   end else
4. End

Algorithms to delete a node from a circular linked list

Algorithm to delete a node from the beginning of a circular linked list:

1. if start==NULL then
   "empty list" and exit
2. else
   set temp=start
   set start=start->next
   print the deleted element=temp->info
   set last->next=start;
   free(temp)
   end else
3. End

Algorithm to delete a node from the end of a circular linked list:

1. if start==NULL then
   "empty list" and exit
2. else if start==last
   set temp=start
   print deleted element=temp->info
   free(temp)
   start=last=NULL
3. else
   set temp=start
   while(temp->next!=last)
   set temp=temp->next
   end while
   set hold=temp->next
   set last=temp
   set last->next=start
   print the deleted element=hold->info
   free(hold)
   end else
4. End

Doubly Linked List

- A linked list in which all nodes are linked together by multiple number of links i.e. each node contains three fields (two pointer fields and one data field) rather than two fields is called doubly linked list.

- It provides bidirectional traversal.

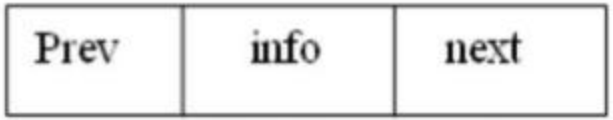


Fig: A node in doubly linked list

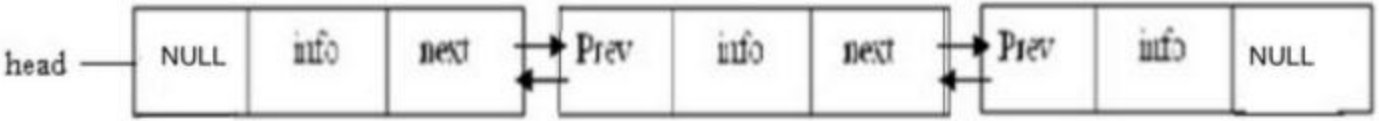
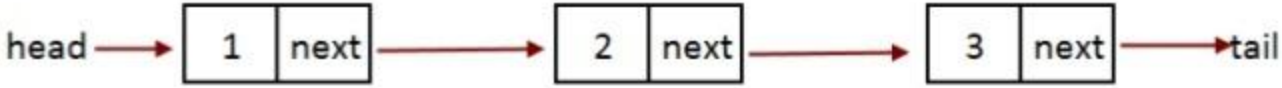
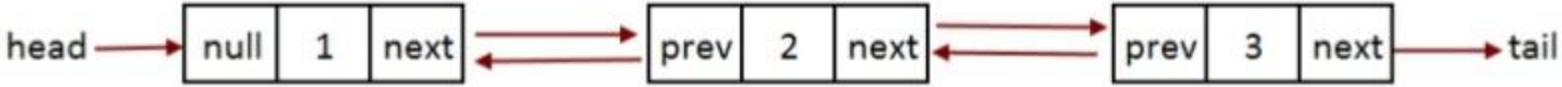


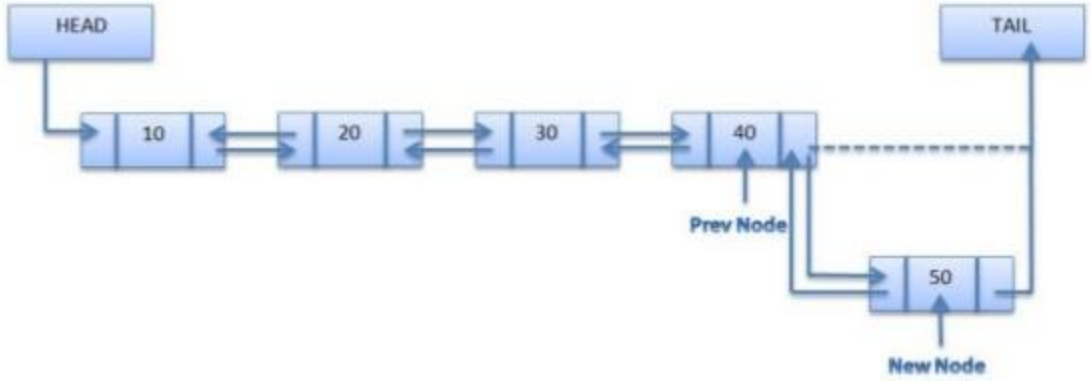
fig: A doubly linked list with three nodes



Singly Linked List



Doubly Linked List



C representation of doubly linked list:

```
struct node
{
        int info;
        struct node *prev;
        struct node *next;
};
typedef struct node NodeType;
NodeType *head=NULL:
```

Algorithms to insert a node in a doubly linked list:

Algorithm to insert a node at the beginning of a doubly linked list:

1. Allocate memory for the new node as,
   newnode=(NodeType*)malloc(sizeof(NodeType))
2. Assign value to info field of a new node
   set newnode->info=item
3. set newnode->prev=newnode->next=NULL
4. set newnode->next=head
5. set head->prev=newnode
6. set head=newnode
7. End

C function to insert a node at the beginning of a doubly linked list:

```
void InsertAtBeg(int Item)
{
        NodeType *newnode;
        newnode=(NodeType*)malloc(sizeof(NodeType));
        newnode->info=item;
        newnode->prev=newnode->next=NULL;
        newnode->next=head;
        head->prev=newnode;
        head=newnode;
}
```

Algorithm to insert a node at the end of a doubly linked list:

1. Allocate memory for the new node as,
   newnode=(NodeType*)malloc(sizeof(NodeType))
2. Assign value to info field of a new node
   set newnode->info=item
3. set newnode->next=NULL
4. if head==NULL
   set newnode->prev=NULL;
   set head=newnode;
5. if head!=NULL
   set temp=head
   while(temp->next!=NULL)
   temp=temp->next;
   end while
   set temp->next=newnode;
   set newnode->prev=temp
6. End

Algorithm to delete a node from beginning of a doubly linked list:

1. if head==NULL then
   print "empty list" and exit
2. else
   set hold=head
   set head=head->next
   set head->prev=NULL;
   free(hold)
3. End

Algorithm to delete a node from end of a doubly linked list:

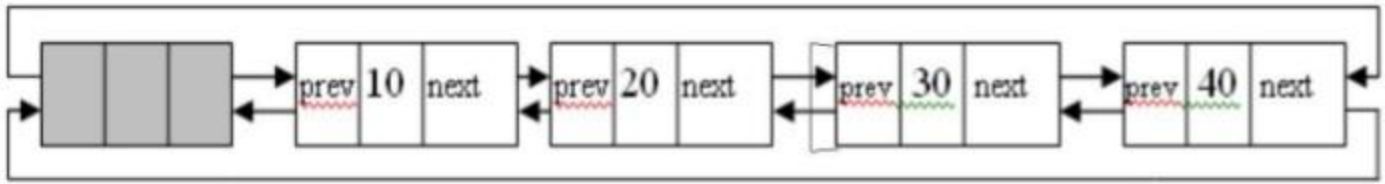
1. if head==NULL then
   print "empty list" and exit
2. else if(head->next==NULL) then
   set hold=head
   set head=NULL
   free(hold)
3. else
   set temp=head;
   while(temp->next->next !=NULL)
   temp=temp->next
   end while
   set hold=temp->next
   set temp->next=NULL
   free(hold)
4. End

Circular Doubly Linked List

- A circular doubly linked list is one which has the successor and predecessor pointer in circular manner.

- It is a doubly linked list where the next link of last node points to the first node and previous link of first node points to last node of the list.

- The main objective of considering circular doubly linked list is to simplify the insertion and deletion operations performed on doubly linked list.



head node Fig: A circular doubly linked list

C representation of doubly circular linked list:

```
struct node
{
        int info;
        struct node *prev;
        struct node *next;
};
typedef struct node NodeType;
NodeType *head=NULL:
```

Algorithm to insert a node at the beginning of a circular doubly linked list:

1. Allocate memory for the new node as,
   newnode=(NodeType*)malloc(sizeof(NodeType))
2. Assign value to info field of a new node
   set newnode->info=item
3. set temp = head -> prev
4. set newnode->prev=temp
5. set newnode->next=head
6. set head->prev=newnode
7. set temp->next=newnode
8. set head=newnode

Algorithm to delete a node from the beginning of a circular doubly linked list:

1. if head->next==NULL then
   print "empty list" and exit
2. else
   set temp=head->next;
   set head->next=temp->next
   set temp->next=head
   free(temp)
3. End

Algorithm to delete a node from the end of a circular doubly linked list:

1. if head->next==NULL then
   print "empty list" and exit
2. else
   set temp=head->prev;
   set head->left=temp->left
   free(temp)
3. End