Unit 4 (The List)

a) **Concept and definition**

b) **Inserting and deleting nodes**

c) **Linked implementation of a stack (PUSH / POP)**

d) **Linked implementation of a queue (insert / delete)**

e) **Circular linked list**

- **Stack as a circular list (PUSH / POP)**
- **Queue as a circular list (Insert / delete)**

f) **Doubly linked list (insert / delete)**

Self-referential structure:

It is sometimes desirable to include within a structure one member that is a pointer to the parent structure type. Hence, a structure which contains a reference to itself is called self-referential structure. In general terms, this can be expressed as:

struct node

{

member 1;

member 2;

…….

struct *node *name;*

};

For example, struct node

{

int info;

struct node *next;

};

This is a structure of type node. The structure contains two members: a *info* integer member, and a pointer to a structure of the same type (i.e., a pointer to a structure of type node), called next. Therefore this is a ***self-referential*** structure.

Linked List:

A linked list is a collection of nodes, where each node consists of two parts:

- **info:** the actual element to be stored in the list. It is also called data field.

- **link:** one or two links that points to next and previous node in the list. It is also called next or pointer field.

Illustration:

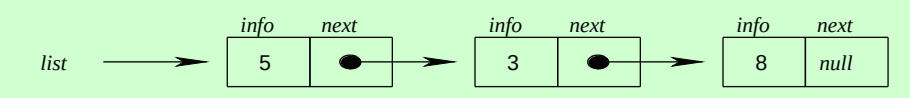


fig:- Singly linked list of integer values

- 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:

The basic operations to be performed on the linked list are as follows:

- **Creation:** This operation is used to create a linked list
- **Insertion:** This operation is used to insert a new node in a linked list in a specified position. A new node may be inserted
  - ✔ At the beginning of the linked list
  - ✔ At the end of the linked list
  - ✔ At the specified position in a linked list
- **Deletion:** The deletion operation is used to delete a node from the linked list. A node may be deleted from
  - ✔ The beginning of the linked list
  - ✔ the end of the linked list
  - ✔ the specified position in the linked list.
- **Traversing:** The list traversing is a process of going through all the nodes of the linked list from on end to the other end. The traversing may be either forward or backward.
- **Searching or find:** This operation is used to find an element in a linked list. In the desired element is found then we say operation is successful otherwise unsuccessful.
- **Concatenation:** It is the process of appending second list to the end of the first list.

Types of Linked List:

basically we can put linked list into the following four types:

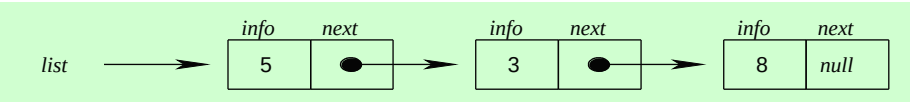
- Singly linked list

- 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 info field which is used to store the data items and another is link field that is used to point the next node in the list. The last node has a NULL pointer.

The following example is a singly linked list that contains three elements 5, 3, 8.



Representation of singly linked list:

We can create a structure for the singly linked list the each node has two members, one is **info** that is used to store the data items and another is **next** field that store the address of next node in the list.

We can define a node as follows:

```
struct Node
{
        int    info;    struct    Node
        *next;
};
typedef struct Node NodeType;
NodeType *head;  //head is a pointer type structure variable
```

This type of structure is called self-referential structure.

- ***The NULL value of the next field of the linked list indicates the last node and we define macro for NULL and set it to 0 as below:***

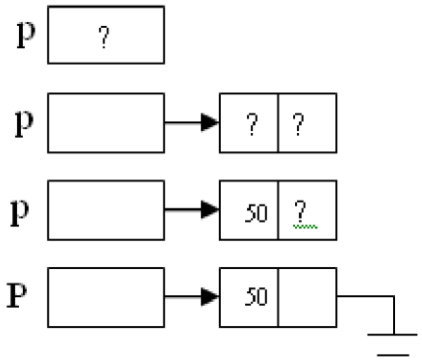
#define NULL 0

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 nose.

The following steps clearly shows the steps required to create a node and storing an item.

```
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.

The getNode function:

we can define a function getNode() to allocate the memory for a node dynamically. It is user-defined function that return a pointer to the newly created node.

```
Nodetype *getNode() {
        NodeType *p;
        p==(NodeType*)malloc(sizeof(NodeType));
        return(p);
         }
```

Creating the empty list:

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

Inserting Nodes:

To insert an element or a node in a linked list, the following three things to be done:

- Allocating a node
- Assigning a data to info field of the node
- Adjusting a pointer and a new node may be inserted
- At the beginning of the linked list
- At the end of the linked list
- At the specified position in a linked list

Insertion requires obtaining a new node and changing two links

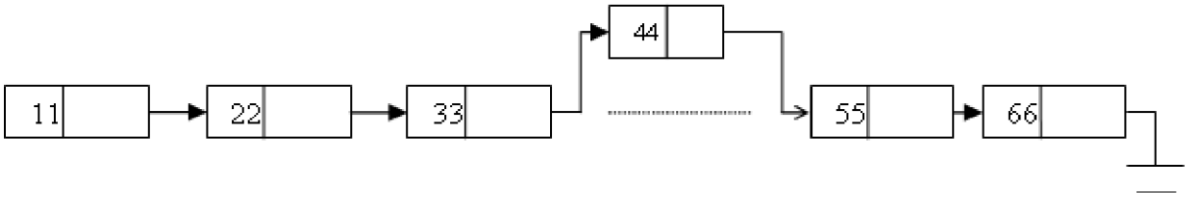


fig:- Inserting the new node with 44 between 33 and 55.

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;

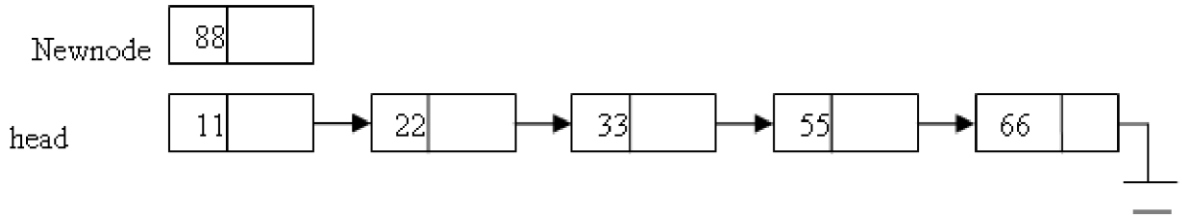
4. Set the head pointer to the new node

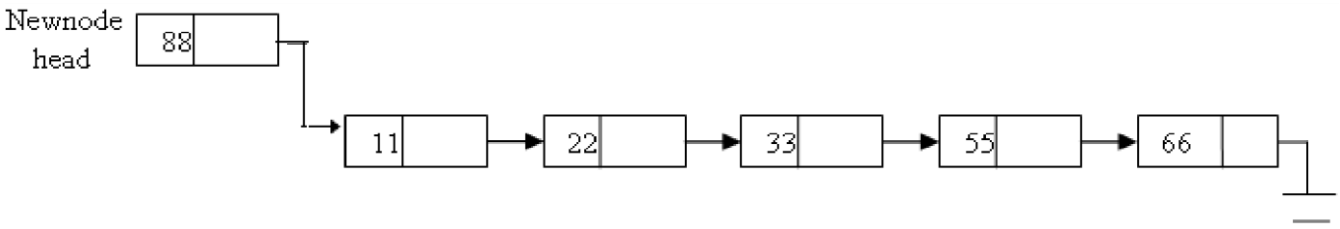
   head=NewNode;

5. End

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

```
void InsertAtBeg(int newItem)
{
        NodeType *NewNode;
        NewNode=getNode();
        NewNode->info=newItem;
        NewNode->next=head;
        head=NewNode;
}
```





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

The C function to insert a node at the end of the linked list:

```
void InsertAtEnd(int newItem)
{
        NodeType *NewNode;
        NewNode=getNode();
        NewNode->info=newItem;
        NewNode->next=NULL;
        if(head==NULL)
        { head=NewNode;
        }
        else
        {
                temp=head;
```

```
                while(temp->next!=NULL)
                {
                        temp=temp->next;
                 }
                temp->next=NewNode;
        }
}
```

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

The C function to insert a node after the given node in singly linked list:

```
void InsertAfterNode(NodeType *p int newItem)
{
        NodeType *NewNode;
        NewNode=getNode();
        NewNode->info=newItem;
        if(p==NULL)
        {
                printf("Void insertion");
                exit(1);
        }
        else
        {
                NewNode->next=p->next;
```

```
            p->next =NewNode..
        }
    }
```

An algorithm to insert a node at the specified position in a 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. 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-1; i++)

   temp=temp->next;

7. Set *NewNode->next=temp->next;*

   set temp->next =NewNode..

8. 8. End

The C function to insert a node at the specified position in a singly linked list:

```
void InsertAtPos(int newItem)
{
        NodeType *NewNode;
        int pos , i ;
        printf(" Enter position of a node at which you want to insert a new node");
        scanf("%d",&pos);
        if(head==NULL)
        {
                printf("void insertion");
                exit(1).
        }
        else
```

```
        {
                temp=head;
                for(i=1; i<pos-1; i++)
                {
                    temp=temp->next;
                }
                NewNode=getNode();
                NewNode->info=newItem;
                NewNode->next=temp->next;
                temp->next =NewNode;
            }
    }
```

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

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

The C function to deleting the first node of the singly linked list:

```
void deleteBeg()
    {
```

```
NodeType *temp; if(head==NULL)
{
        printf("Empty list"); exit(1).
}
else
{ temp=head;
        printf("Deleted item is %d" ,
        head->info);
        head=head->next;
        free(temp);
   }
}
```

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

The C function to deleting the last node of the singly linked list:

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

```
void deleteEnd()
  {
      NodeType *temp; if(head==NULL)
      {
          printf("Empty list");
          return;
      }
      else if(head->next==NULL)
      {
              temp=head;
              head=NULL;
              printf("Deleted item is %d", temp->info);
              free(temp);
      }
      else
      {
               temp=head;
              while(temp->next->next!=NULL)
              {
                      temp=temp->next;
              }
              printf("deleted item is %d'" ,temp->next->info):
              free(temp->next);
              temp->next=NULL;
      }
   }
```

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

The C function to delete a node after the given node in singly linked list:

let *p be the pointer to the node after which we want to delete a new node.

```
void deleteAfterNode(NodeType *p)

{

        NodeType *q;
        if(p==NULL || p->next==NULL )
        {
                printf("Void insertion");
                exit(1);
        }
        else
        {
                q=p->next;
                p->next=q->next;
                free(q);
        }
}
```

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. *S*et temp->next =temp->next->next;
10. free(p);
11. End

The C function to delete a node at the specified position in a singly linked list

```
void deleteAtSpecificPos()
  {
      NodeType *temp *p;
      int pos, i;
      if(head==NULL)
      {
          printf("Empty list");
          return;
      }
      else
      {
              printf("Enter position of a node which you want to delete");
              scanf("%d" , &pos);
              temp=head;
              for(i=1; i<pos-1; i++)
              {
                      temp=temp->next;
              }
              p=temp->next;
      `       printf("Deleted item is %d", p->info);
              temp->next =p->next;
              free(p);
        }
  }
```

Searching an item in a linked list:

To search an item from a given linked list we need to find the node that contain this data item.

If we find such a node then searching is successful otherwise searching unsuccessful.

*let *head be the pointer to first node in the current lis*t

```
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");
       }
      }
```

Complete program:

/***Various operations on singly linked list**********/***

```
#include<stdio.h>
```

```
#include<conio.h>
#include<malloc.h>    //for malloc function
#include<process.h>   //fpr exit function
struct node
{
        int info;
        struct node *next;
};
typedef struct node NodeType;
NodeType *head;
head=NULL;
void insert_atfirst(int);
void insert_givenposition(int);
void insert_atend(int);
void delet_first();
void delet_last();
void delet_nthnode();
void info_sum();
void count_nodes();
void main()
{
        int choice; int item;
        do {
           printf("\n manu for program:\n");
           printf("1. insert first \n2.insert at given position \n3 insert at last \n 4:Delete first
node\n 5:delete last node\n6:delete nth node\n7:count nodes\n8Display items\n10:exit\n");
           printf("enter your choice\n");
           scanf("%d",&choice);
           switch(choice)
                {
                        case 1:
                                printf(“Enter item to be inserted”);
                                scanf(“%d”, &item);
```

```
            insert_atfirst(item);
            break;
        case 2:
            printf("Enter item to be inserted");
            scanf("%d", &item);
             insert_givenposition(item);
            break;
        case 3:
            printf("Enter item to be inserted");
            scanf("%d", &item);
            insert_atend();
            break;
        case 4:
            delet_first();
            break;
        case 5:
            delet_last();
            break;
        case 6:
            delet_nthnode();
            break;
        case 7:
            info_sum();
            break;
        case 8:
            count_nodes();
            break;
        case 9:
            exit(1);
            break;
        default:
            printf("invalid choice\n");
            break;
```

```
                }
            }while(choice<10);
             getch();
        }
/************function definitions**************/
    void insert_atfirst(int item)
    {
        NodeType *nnode;
        nnode=(NodeType*)malloc(sizeof(NodeType));
        nnode->info=item;
        nnode->next=head;
        head=nnode;
    }

    void insert_givenposition(int item)
    {
        NodeType *nnode;
        NodeType *temp;
        temp=head;
        int p,i;
        nnode=(NodeType*)malloc(sizeof(NodeType));
        nnode->info=item;
        if (head==NULL)
        {
            nnode->next=NULL;
            head=nnode;
        }
        else
        {
            printf("Enter Position of a node at which you want to insert an new node\n");
            scanf("%d",&p);
            for(i=1;i<p-1;i++)
            {
```

```
                temp=temp->next;
            }
            nnode->next=temp->next;
            temp->next=nnode;
        }
    }
void insert_atend(int item)
    {
        NodeType *nnode;
        NodeType *temp;
        temp=head;
        nnode=( NodeType *)malloc(sizeof(NodeType));
        nnode->info=item;
        if(head==NULL)
        {
            nnode->next=NULL;
            head=nnode;
        }
        else {
          while(temp->next!=NULL)
          {
             temp=temp->next;
          }
          nnode->next=NULL;
        temp->next=nnode;
        }
    }
 void delet_first()
  {
        NodeType *temp;
        if(head==NULL)
        {
           printf("Void deletion|n");
```

```
        return;
    }
    else {
            temp=head;
            head=head->next;
            free(temp);
    }
}

void delet_last()
{
    NodeType *hold,*temp;
    if(head==NULL)
    {
       printf("Void deletion|n");
    return;
    }
    else if(head->next==NULL)
    {
        hold=head;      head=NULL;
        free(hold);
    }
    else {
            temp=head;
            while(temp->next->next!=NULL)
      {
        temp=temp->next;
      }
      hold=temp->next;
    temp->next=NULL;
      free(hold);
    }
}
```

```
void delet_nthnode()
 {
        NodeType *hold,*temp;
        int pos, i;
        if(head==NULL)
        {
           printf("Void deletion|n");
           return;
        }
        else
        {
        temp=head;
        printf("Enter position of node which node is to be deleted\n");
        scanf("%d",&pos);
        for(i=1;i<pos-1;i++)
          {
            temp=temp->next;
          }
          hold=temp->next;
          temp->next=hold->next;
          free(hold);
        }
 }
void info_sum()
 {
    NodeType *temp;
   temp=head;
    while(temp!=NULL)
    {
        printf("%d\t",temp->info);
        temp=temp->next;
    }  }
 void count_nodes()
```

```
    {
        int cnt=0;
        NodeType *temp;
        temp=head;
    while(temp!=NULL)
      {
        cnt++;
        temp=temp->next;
      }
      printf("total nodes=%d",cnt);
    }
}
```

Linked list implementation of Stack:

Push function:

let *top be the top of the stack or pointer to the first node of the list.

```
    void push(item)
    {
        NodeType *nnode;
        int data;
        nnode=( NodeType *)malloc(sizeof(NodeType));
        if(top==0)
        {
          nnode->info=item;
          nnode->next=NULL;
          top=nnode;
        }
        else
        {
            nnode->info=item;
            nnode->next=top;
            top=nnode;
        }
```

```
}
```

Pop function:

let *top be the top of the stack or pointer to the first node of the list.

```
void pop()
{
        NodeType *temp;
        if(top==0)
        {
                printf("Stack contain no elements:\n");
                return; }
        else
        {
                temp=top;
                top=top->next;
                printf("\ndeleted item is %d\t",temp->info);
                free(temp);
        }
}
```

A Complete C program for linked list implementation of stack:

/**********Linked list implementation of stack**********/

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<process.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node NodeType;
NodeType *top;
```

```
top=0;
void push(int);
void pop();
void display();
void main()
{
        int choice, item;
        do {
                printf("\n1.Push \n2.Pop \n3.Display\n4:Exit\n");

           printf("enter ur choice\n");
           scanf("%d",&choice);
           switch(choice)
           {
                case 1:
                        printf("\nEnter the data:\n");\
                        scanf("%d",&item);
                        push(item);
                        break;
                case 2:
                        pop();
                        break;
                case 3:
                        display();
                         break;
                case 4:
                        exit(1);
                        break;
                default:
                        printf("invalid choice\n");
                        break;
           }
        }while(choice<5);
```

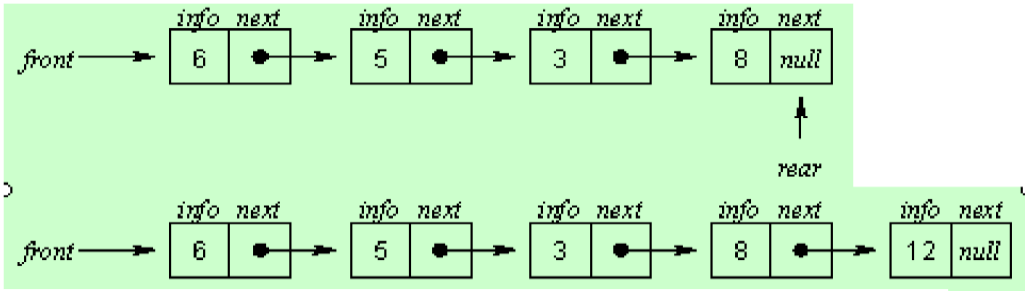
```
            getch();

        }
/**************push function******************/
void push(int item)
        {
            NodeType *nnode;
            int data;
            nnode=( NodeType *)malloc(sizeof(NodeType));
            if(top==0)
            {
                nnode->info=item;
                nnode->next=NULL;
                top=nnode;
            }
            else
            {
                nnode->info=item;
                nnode->next=top;
                top=nnode;
            }
        }
/******************pop function*********************/
void pop()
        {
            NodeType *temp;
            if(top==0)
            {
                printf("Stack contain no elements:\n");
                return;
            }
            else
            {
                temp=top;
```

```
                top=top->next;
                printf("\ndeleted item is %d\t",temp->info);
                free(temp);
        }
    }

/**************display function**********************/
void display()
    {
        NodeType *temp;
         if(top==0)
        {
            printf("Stack is empty\n");
            return;
        }
        else
        {
            temp=top;
            printf("Stack items are:\n");
            while(temp!=0)
            {
                printf("%d\t",temp->info);
                temp=temp->next;
            }
        }
    }
```

Linked list implementation of queue:

Insert function:

let *rear and *front are pointers to the first node of the list initially and insertion of node in linked list done at the rear part and deletion of node from the linked list done from front part.




```
    rear void insert(int item)
{
        NodeType *nnode;

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

        if(rear==0)
        {
            nnode->info=item;
            nnode->next=NULL;
            rear=front=nnode;
        }
        else
        {
            nnode->info=item;
            nnode->next=NULL;
            rear->next=nnode;
            rear=nnode;
        }
}
```

Delete function:

let *rear and *front are pointers to the first node of the list initially and insertion of node in linked list done at the rear part and deletion of node from the linked list done from front part.

```
    void delet()
 {
        NodeType *temp;
        if(front==0)
        {
```

```
                printf("Queue contain no elements:\n");
                return;
        }
        else if(front->next==NULL)
        {
                 temp=front;
                rear=front=NULL;
                printf("\nDeleted item is %d\n",temp->info);
                free(temp);
        }
        else
        {
                temp=front;
                front=front->next;
                printf("\nDeleted item is %d\n",temp->info);
                free(temp);
        }
}
```

A Complete C program for linked list implementation of queue:

/**********Linked list implementation of queue**********/

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<process.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node NodeType;
NodeType *rear,*front;
rear=front=0;
```

```
void insert(int);
void delet();
void display();
void main()
{
        int choice, item;
        do {
                printf("\n1.Insert \n2.Delet \n3.Display\n4:Exit\n");
                printf("enter ur choice\n");
                scanf("%d",&choice);
                switch(choice)
           {
                case 1:
                        printf("\nEnter the data:\n");
                         scanf("%d",&item);
                        insert(item);
                        break;
                case 2:
                        delet();
                        break;
                case 3:
                        display();
                        break;
                case 4:
                        exit(1);
                        break;
                default:
                        printf("invalid choice\n");
    break;
           }
        }while(choice<5);

        getch();
```

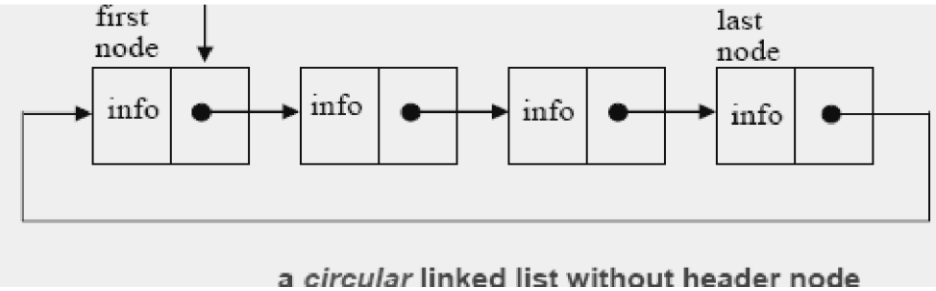
```
	}
/**************insert function******************/
void insert(int item)
	{
		NodeType *nnode;

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

		if(rear==0)
		{
			nnode->info=item;
			nnode->next=NULL;
			rear=front=nnode;
		}
		else
		{
			nnode->info=item;
			nnode->next=NULL;
			rear->next=nnode;
			rear=nnode;
		}
	}
 /******************delet function*******************/
	void delet()
	{
		NodeType *temp;
		if(front==0)
		{
			printf("Queue contain no elements:\n");
			return;
		}
		else if(front->next==NULL)
		{
```

```
                temp=front;
                rear=front=NULL;
                printf("\nDeleted item is %d\n",temp->info);
                free(temp);
            }
            else
            {
                temp=front;
                front=front->next;
                printf("\nDeleted item is %d\n",temp->info);
                free(temp);
            }
    }
/***************display function***********************/
void display()
    {
        NodeType *temp;
        temp=front;
        printf("\nqueue items are:\t");
        while(temp!=NULL)
        {
            printf("%d\t",temp->info);
            temp=temp->next;
        }
    }
```

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 .

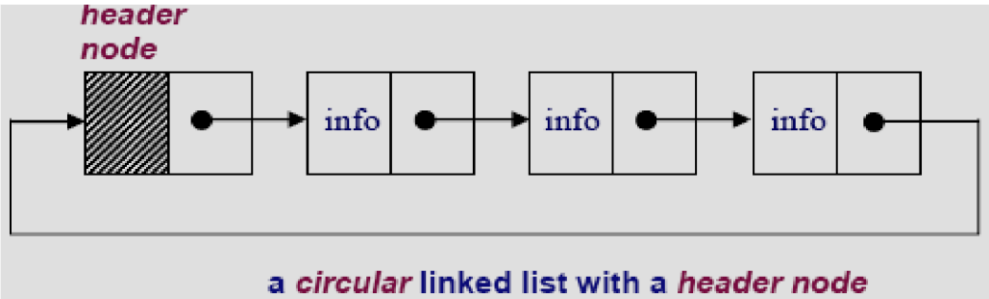
Circular linked lists can be used to help the traverse the same list again and again if needed. A circular list is very similar to the linear list where in the circular list the pointer of the last node points not NULL but the first node.



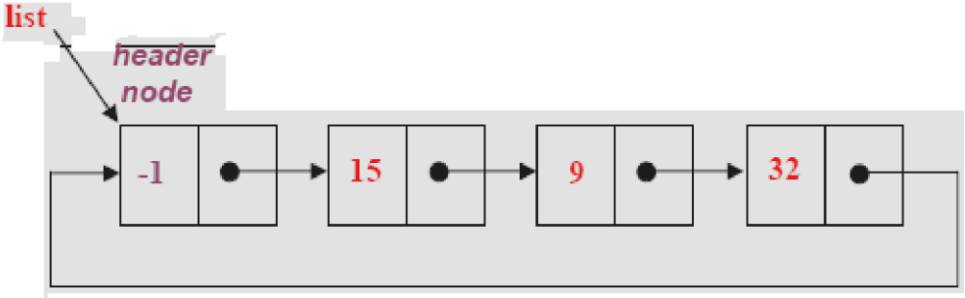
In a circular linked list there are two methods to know if a node is the first node or not.

- Either a external pointer, ***list***, points the first node or
- A ***header node*** is placed as the first node of the circular list.

The header node can be separated from the others by either heaving a ***sentinel value*** as the info part or having a dedicated ***flag*** variable to specify if the node is a header node or not.



CIRCULAR LIST with header node



C representation of circular linked list:

we declare the structure for the circular linked list in the same way as declared it for the 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
```

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

```
void InsertAtBeg(int Item)
{
        NodeType *newnode;
        newnode=(NodeType*)malloc(sizeof(NodeType));
        if(start==NULL)
        {
                newnode->info=item;
                newnode->next=newnode;
                start=newnode;
                last newnode;
        }
        else
        {

                newnode->info=item;

                last->next=newnode;
                last=newnode;
                last->next=start;

        }
}
```

C function to insert a node at the end of a circular linked list:

```
void InsertAtEnd(int Item)
{
        NodeType *newnode;
        newnode=(NodeType*)malloc(sizeof(NodeType));
        if(start==NULL)
        {
```

```
            newnode->info=item;
            newnode->next=newnode;
            start=newnode;
            last newnode;
        }
        else
        {
            newnode->info=item;
            last->next=newnode;
            last=newnode;
            last->next=start;

        }
}
```

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

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

```
void DeleteFirst()
{
    if(start==NULL)
    {
        printf("Empty list");
        exit(1);
    }
    else
    {
        temp=start;
        start=start->next;
        printf(" the deleted element=%d", temp->info);
        last->next=start;
        free(temp)
    }
}
```

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

```
void DeleteLast()
{
```

```
        if(start==NULL)
        {
                printf("Empty list");
                exit(1);
        }
        else if(start==last)  //for only one node
        {
                temp=start;
                printf("deleted element=%d", temp->info);
                free(temp);
                start=last=NULL;
        }
        else
        {
                temp=start;
                while( temp->next!=last)
                        temp=temp->next;
                hold=temp->next;
                last=temp;
                last->next=start;
                printf("the deleted element=%d", hold->info);
                free(hold);
        }
}
```

Stack as a circular List:

To implement a stack in a circular linked list, let pstack be a pointer to the last node of a circular list. Actually there is no any end of a list but for convention let us assume that the first node(rightmost node of a list) is the top of the stack.

An empty stack is represented by a null list.

The structure for the circular linked list implementation of stack is:

```
struct node
{
        int info;
```

```
    struct node *next;
};
typedef struct node NodeType;
NodeType *pstack=NULL;
```

C function to check whether the list is empty or not as follows:

```
int IsEmpty()
{
    if(pstack==NULL)
        return(1);
    else
        return(0);
}
```

PUSH function:

```
void PUSH(int item)
{
    NodeType newnode;
    newnode=(NodeType*)malloc(sizeof(NodeType));
    newnode->info=item;
    if(pstack==NULL)
    {
        pstack=newnode;
        pstack->next=pstack;
    }
    else
    {
        newnode->next=pstack->next;
        pstack->next=newnode;
    }
}
```

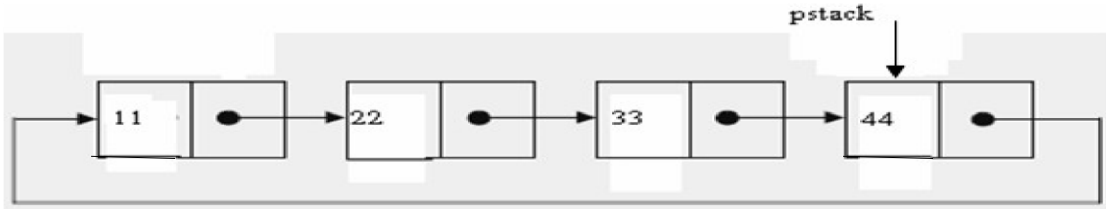
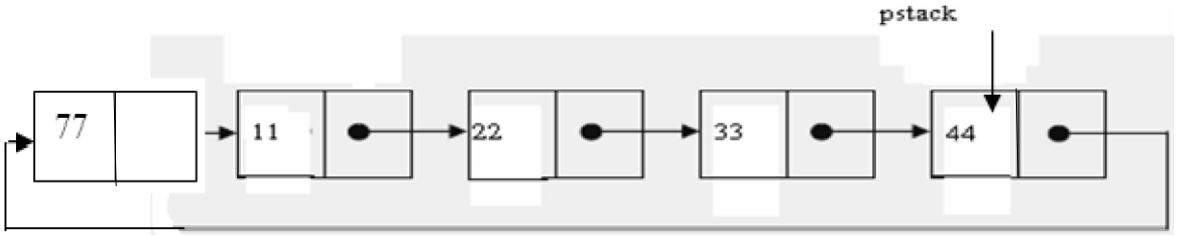


fig: circular linked list



POP function:

```
void POP()
{
        NodeType *temp; if(pstack==NULL)
        {
                printf("Stack underflow\n');
                exit(1);
        }
        else if(pstack->next==pstack)     //for only one node
        {
                printf("poped item=%d", pstack->info);
                pstack=NULL;
        }
        else
        {
                temp=pstack->next;
                pstack->next=temp->next;
                printf("poped item=%d", temp->info);
                free(temp);
        }
}
```

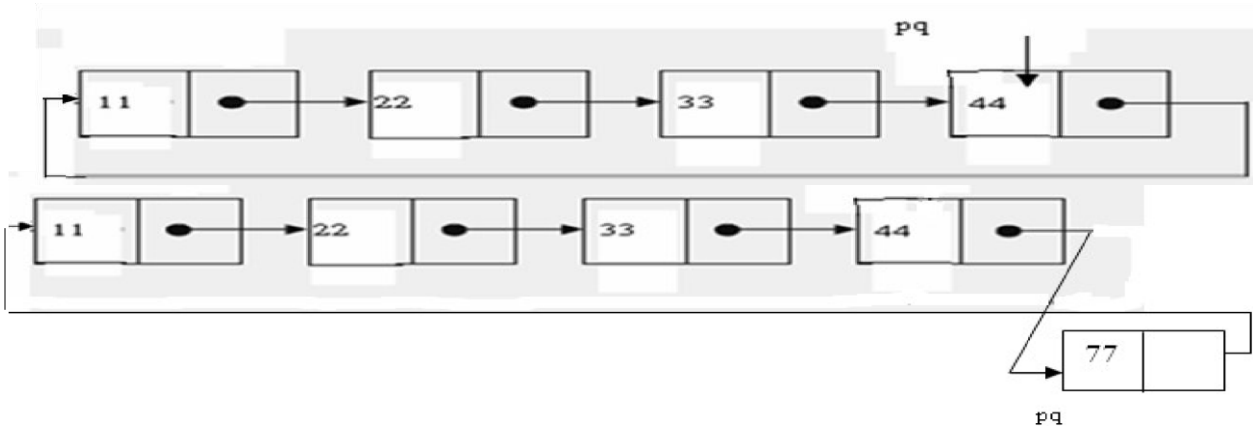
Queue as a circular List:

It is easier to represent a queue as a circular list than as a linear list. As a linear list a queue is specified by two pointers, one to the front of the list and the other to its rear. However, by using a circular list, a queue may be specified by a single pointer q to that list. node(q) is the rear of the queue and the following node is its front.

Insertion function:

```
void insert(int item)
{
```

```
        NodeType *nnode;
        nnode=( NodeType *)malloc(sizeof(NodeType));
         nnode->info=item;
        if(pq==NULL)
                pq=nnode;
        else {
                 nnode->next=pq->next;
                pq->next=nnode;
                pq=nnode;
        }
}
```



Deletion function:

```
Void delet(int item)
        {
                NodeType *temp;
                if(pq==NULL)
                {
                        printf("void deletion\n");
                        exit(1);
                }
                else if(pq->next==pq)     //for only one node
                {
                        printf("poped item=%d", pq->info);
```

```
            pq=NULL;
        }
        else
        {
            temp=pq->next;
            pq->next=temp->next;
            printf("poped item=%d", temp->info);
            free(temp);
        }
}
```

Doubly Linked List:

A linked list in which all nodes are linked together by multiple number of links ie 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.

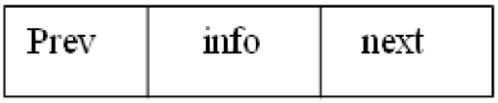


Fig: A node in doubly linked list

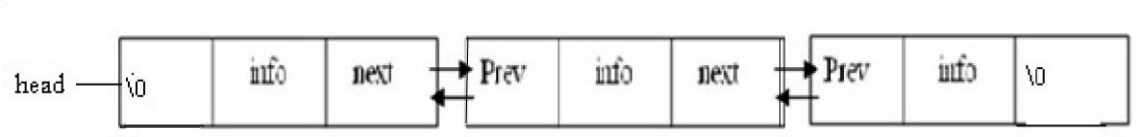


fig: A doubly linked list with three nodes

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)

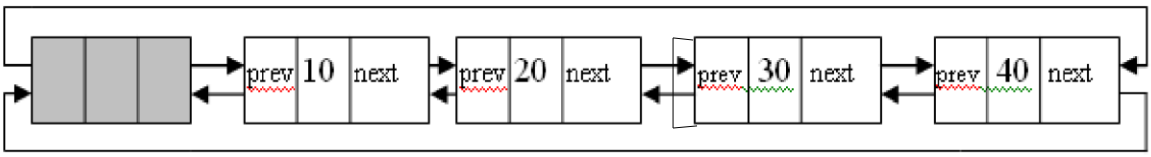
4. End

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.



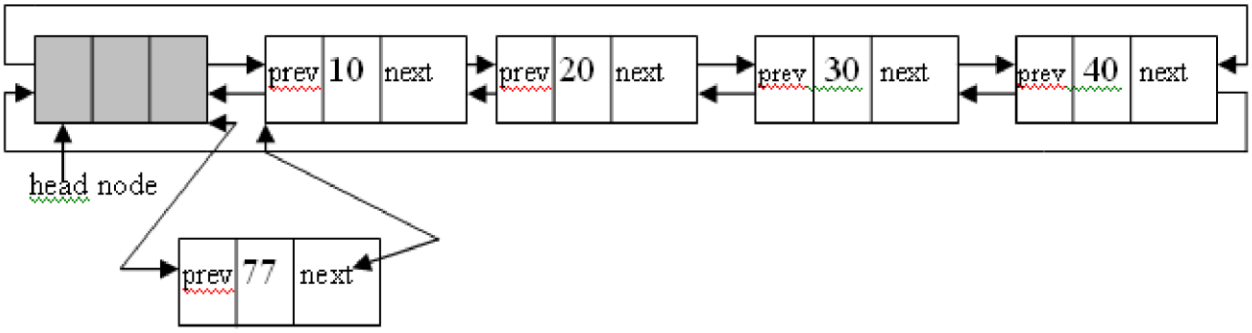
head node Fig: A circular doubly linked list

C representation of doubly circular linked list:

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

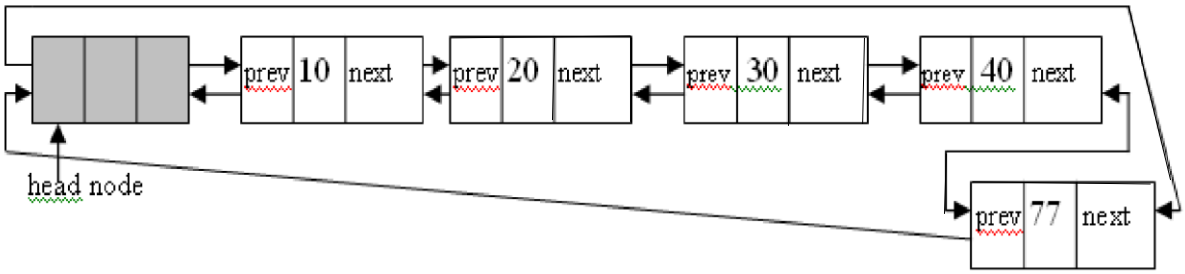
Algorithm to insert a node at the beginning of a circular doubly linked list :

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->next
4. set head->next=newnode
5. set newnode->prev=head
6. set newnode->next=temp
7. set temp->prev=newnode
8. End



Algorithm to insert a node at the end 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 temp->next=newnode
5. set newnode->prev=temp
6. set newnode->next=head
7. set head->prev=newnode
8. End



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