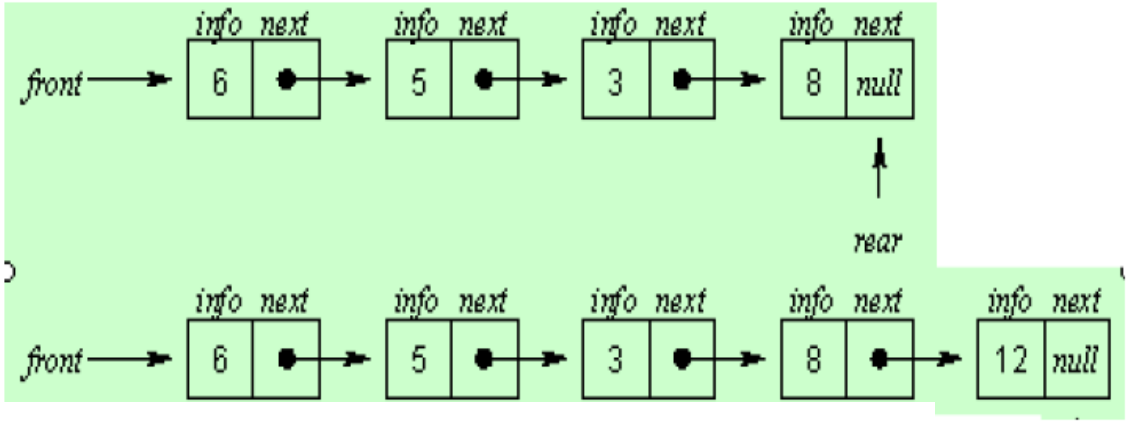
DSA(Lecture#6)

Prepared By
Bal Krishna Subedi

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.




```
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;
        clrscr();
        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;
                }
        }
```

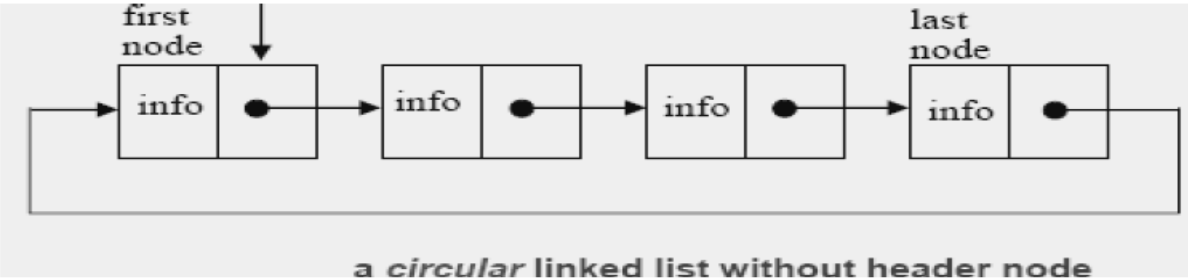
```
void delet()
{
        NodeType *temp;
        if(front==0)
        {
                printf("Queue contain no elements:\n");
                return;
        }
        else if(front->next==NULL)
        {
                temp=front;
                rear=front=NULL;
                printf("\nDeleted item is %d\n",temp->info);
                free(temp);
        }
        else
        {
                temp=front;
                front=front->next;
                printf("\nDeleted item is %d\n",temp->info);
                 free(temp);
        }
}
```

```
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.

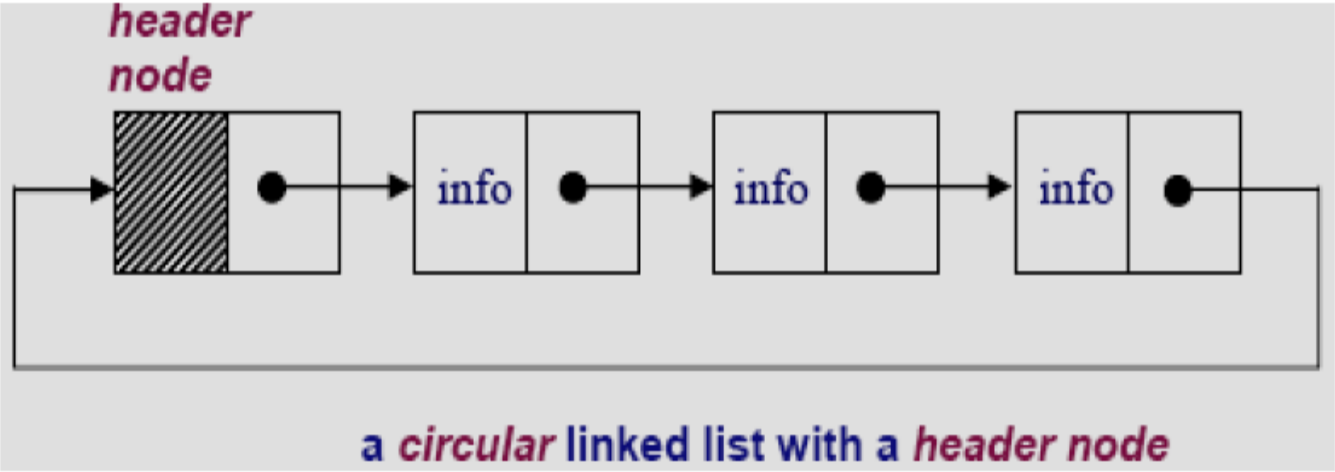


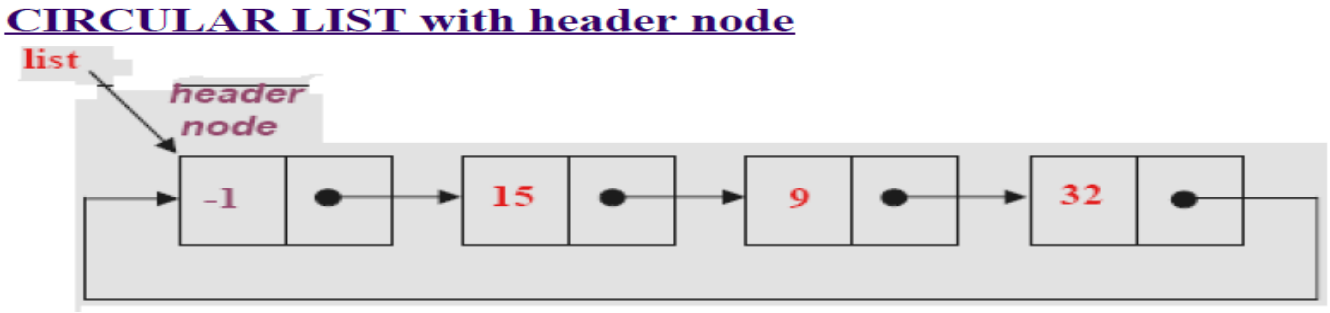
a *circular* linked list without header node

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.





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;
        }
}
```

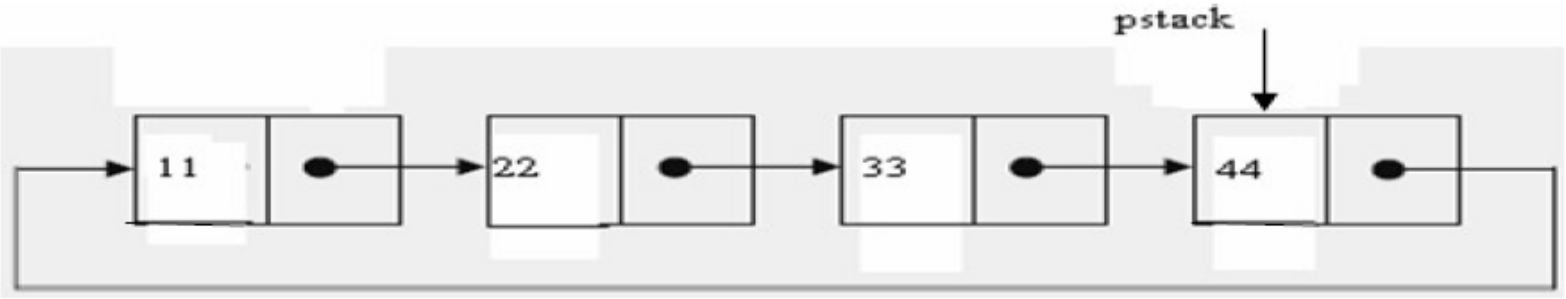
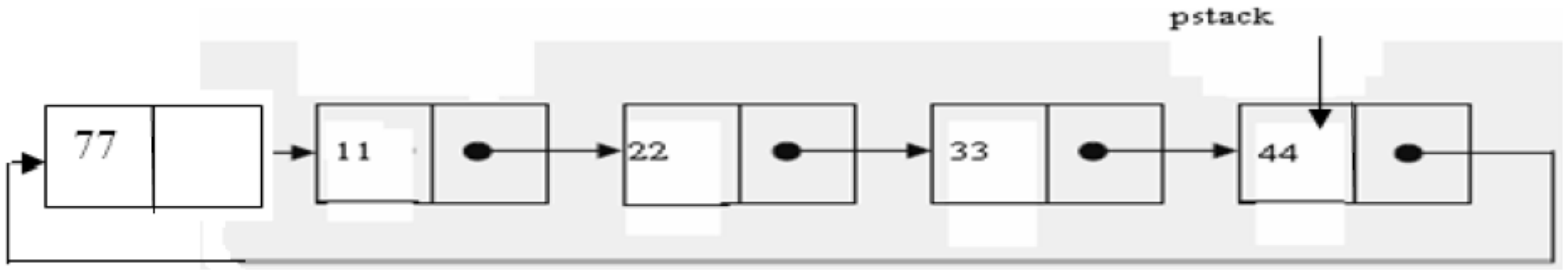


fig: circular linked list



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);

                  }
        }
```

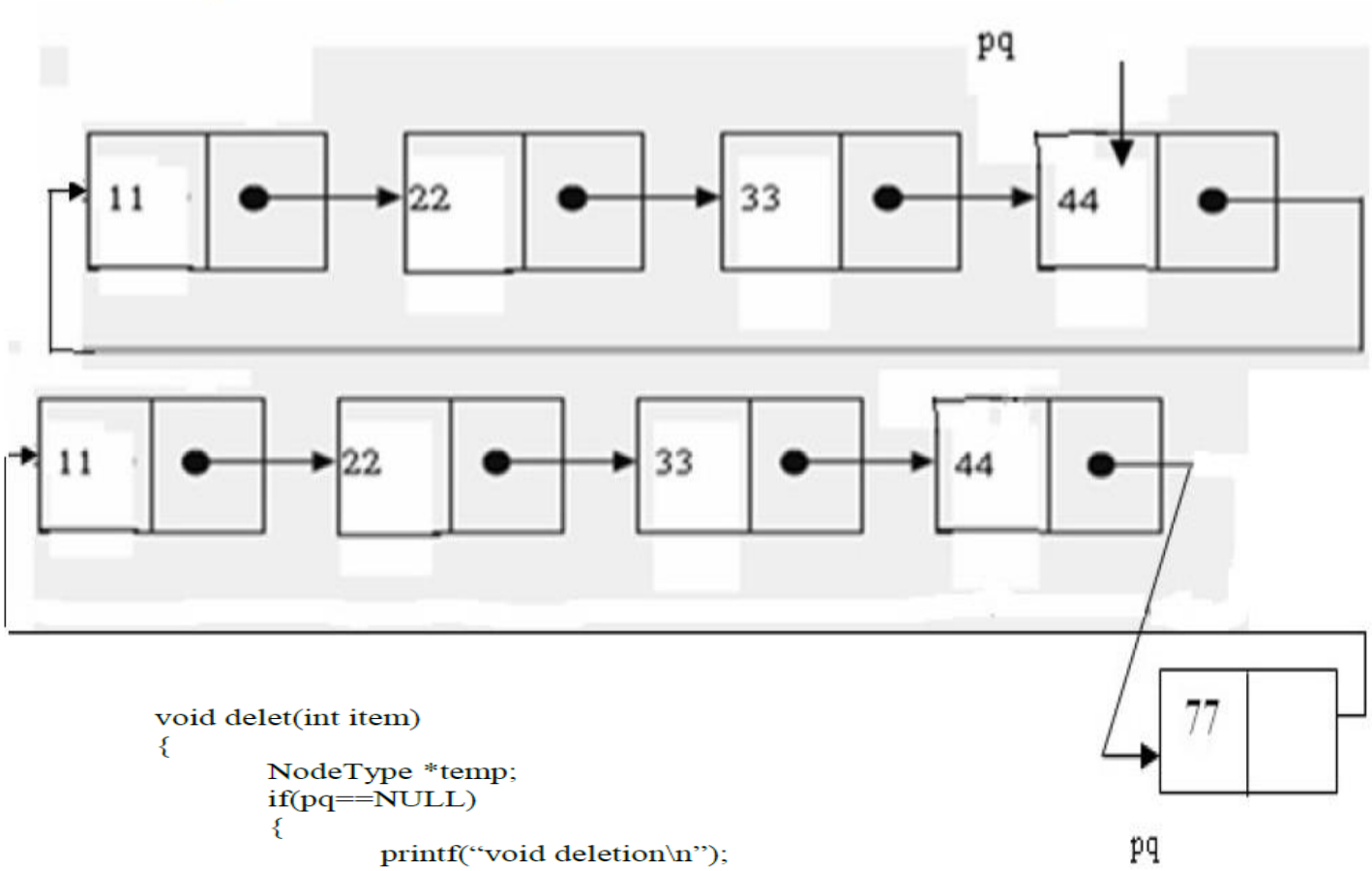
Queue as a circular List:

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.

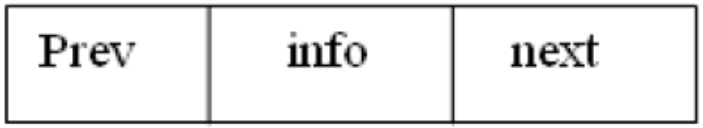


Fig: A node in doubly linked list

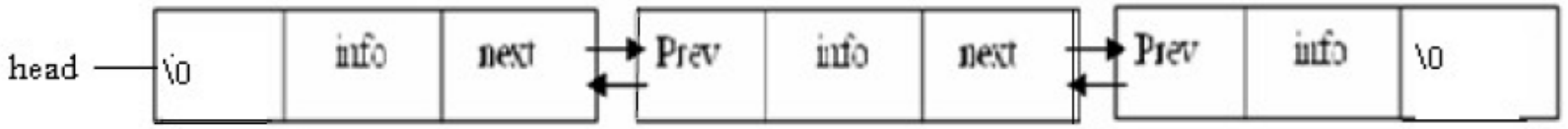


fig: A doubly linked list with three nodes

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:

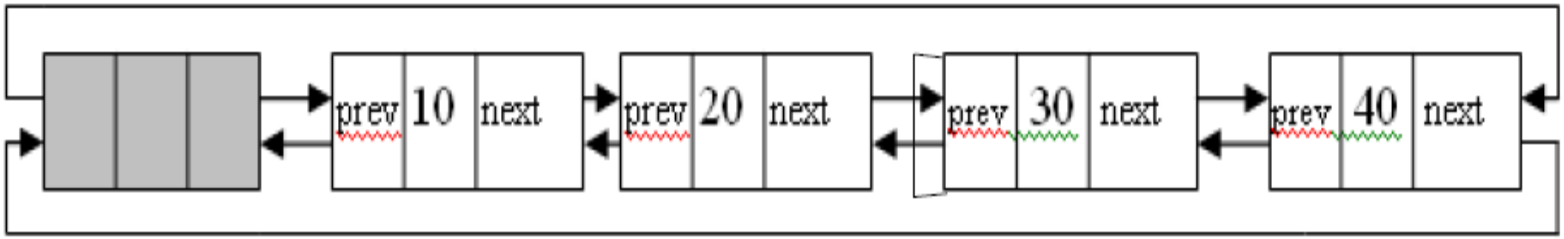
1. if head==NULL then
   print "empty list" and exit
2. else if(head->next==NULL) then
   set hold=head
   set head=NULL
   free(hold)
3. else
   set temp=head;
   while(temp->next->next !=NULL)
   temp=temp->next
   end while
   set hold=temp->next
   set temp->next=NULL
   free(hold)
4. End

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

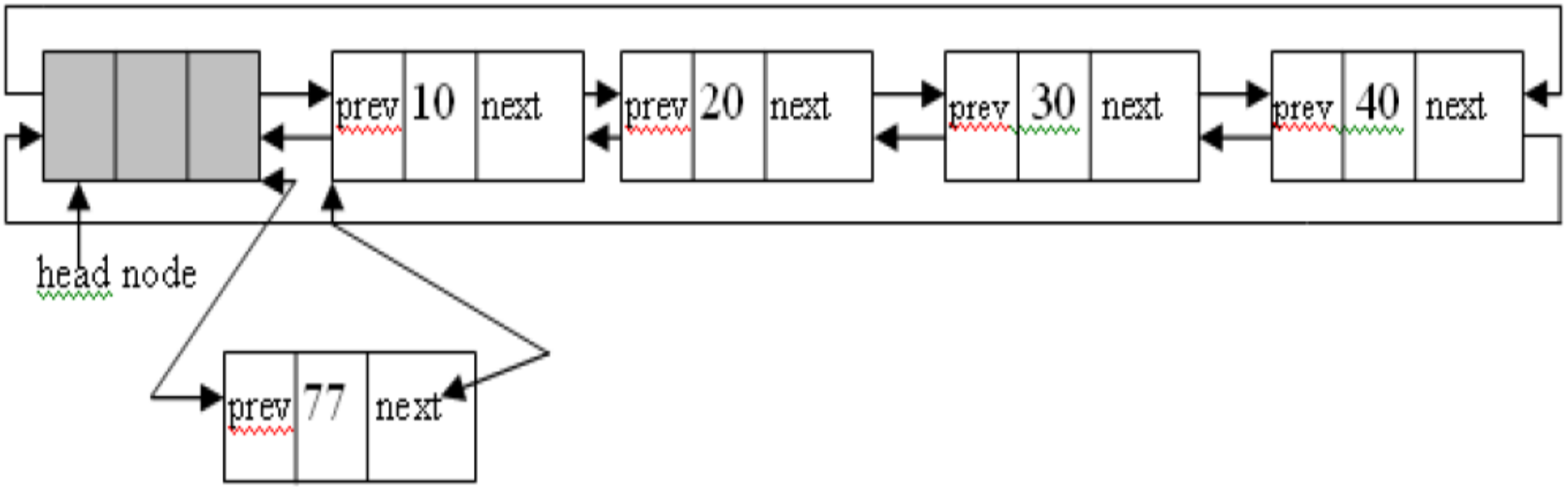
Fig: A circular doubly linked list

C representation of doubly circular linked list:

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

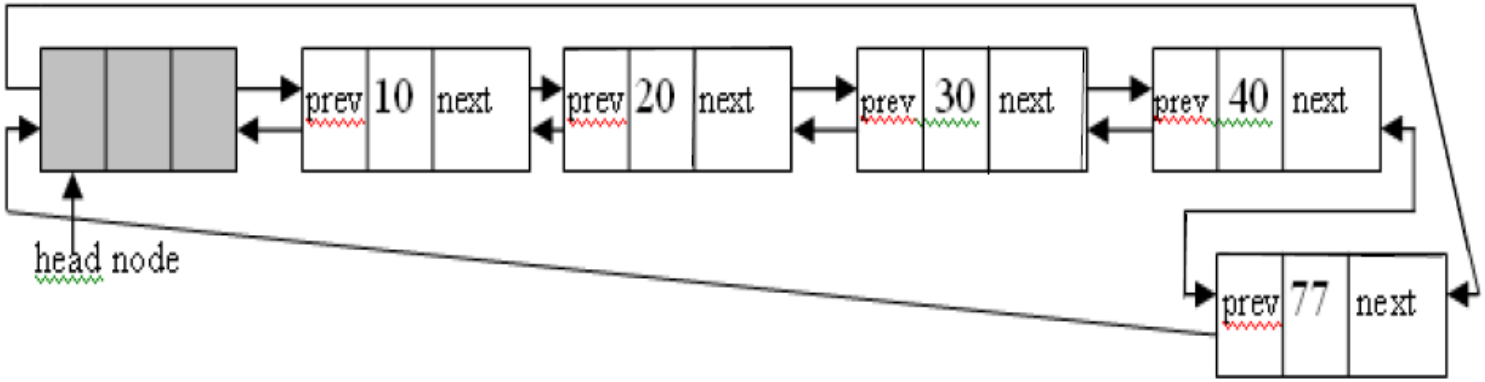
Algorithm to insert a node at the beginning of a circular doubly linked list:

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

Thanks You

Any Queries