Dequeue

```
# include <stdio.h>
# include <stdlib.h>
# define max 5
int queue[max];
int front=-1,rear=-1;
void enqueuefront(int data)
{
    if((front==0 && rear==max-1)||(front==rear+1))
    {
        printf("\n queue is full");
    }
    else if(front==-1 && rear==-1)
    {    front=rear=0;
        queue[front]=data; }
      else if(front==0)  {
        front=max-1;
        queue[front]=data;}
    else {
        front--;   queue[front]=data;}}
void enqueuerear(int data)
{if((front==0 && rear==max-1)||(front==rear+1))
    {
printf("\n queue is full");
    else if(front==-1 && rear==-1) {
        front=rear=0;
        queue[rear]=data; }
    else if(rear==max-1)
    { rear=0;queue[rear]=data;
    }  else
    {  rear++;
       queue[rear]=data; }
void dequeuefront(){
  if((front==-1 && rear==-1)) {
printf("\n queue is empty");
    }   else if(front==rear)
    {   printf("\n %d is deleted",queue[front]);
        front=rear=-1; }
    else if(front==max-1)
    {  printf("\n %d is deleted",queue[front]);
       front=0; }
    else
    { printf("\n %d is deleted",queue[front]);
      front++;}}
```

```
void dequeuerear()
{ if((front==-1 && rear==-1)){
 printf("\n queue is empty");
     }   else if(rear==front)
     {printf("\n %d is deleted",queue[rear]);
      rear=-1;  }
     else if(rear==0)
     { rear=max-1;
      printf("\n %d is deleted",queue[rear]);
     } else {
        printf("\n %d is deleted",queue[rear]);
        rear--;}}
void display()
{  int i=front;
    if(rear==-1 && front==-1) {
        printf("\n queue is empty"); }
    else
    {while(i!=rear) {
            printf("\n %d",queue[i]);
            i=(i+1)%max; } printf("\n %d",queue[rear]} }}
int main()
{
    int choice,data;
    printf("\n enter 1->insertion from front");
    printf("\n enter 2->insertion from rear");
    printf("\n enter 3->deletion from front");
   printf("\n enter 4->deletion from rear");
    printf("\n enter 5->display");
    printf("\n  enter 6->exit");
    while(1)
    {  printf("\n enter your choice as per instructions");
       scanf("\n %d",&choice);
       switch(choice)
       {  case 1:
          printf("\n enter the data:");
          scanf("%d",&data);
          enqueuefront(data);
          display();
          break;
         case 2:
          printf("\n enter the data");
          scanf("%d",&data);
          enqueuerear(data);
          display();
          break;
          case 3: dequeuefront();
         break; case 4:
         dequeuerear();
         break; case 5:display();
         break;
         case 6:
         exit(1); }}}
```

<u>OUTPUT</u>

```
enter 1->insertion from front
 enter 2->insertion from rear
 enter 3->deletion from front
 enter 4->deletion from rear
 enter 5->display
  enter 6->exit
 enter your choice as per instructions 1
 enter the data: 2
 enter your choice as per instructions 1
enter the data: 3
 3 2enter your choice as per instructions 1
 enter the data: 4 4 3 2
 enter your choice as per instructions 2
 enter the data 3 4 32 3
 enter your choice as per instructions 2
 enter the data 4
 4 3 2    3 4 enter your choice as per instructions 3
 4 is deleted
 enter your choice as per instructions 3
3 is deleted
 enter your choice as per instructions 4
 4 is deleted
 enter your choice as per instructions
```

CREATING A LINKED LIST

```
# include <stdio.h>
# include <stdlib.h>
struct linked{
                int data;
                struct linked *next;
};
void printlinkedlist(struct linked *head)
{
    while(head!=NULL)
    {
        printf("\n %d",head->data);
```

```
      head=head->next;
   }
}

int main()
{
   struct linked *first,*second,*third,*fourth,*five;
   first=(struct linked *)malloc(sizeof(struct linked));
   second=(struct linked *)malloc(sizeof(struct linked));
   third=(struct linked *)malloc(sizeof(struct linked));
   fourth=(struct linked *)malloc(sizeof(struct linked));
   five=(struct linked *)malloc(sizeof(struct linked));
   first->data=1;
   first->next=second;
   second->data=2;
   second->next=third;
   third->data=3;
   third->next=fourth;
   fourth->data=4;
   fourth->next=five;
   five->data=5;
   five->next=NULL;
   printf("\n linked list before deletion");
   printlinkedlist(first);
}
```

OUTPUT

```
1
2
3
4
5
```

Program of deletion in linked list

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

struct node{
      int data;
      struct node *next;

};
void printlinkedlist(struct node *ptr)
{  while(ptr!=NULL)
   { printf("\n %d",ptr->data);
      ptr=ptr->next;//increment
  }}struct node *delfirstnode(struct node *head){
   struct node *ptr=head;//it points where the head points
   head=head->next;
   free(ptr);
   return(head);

};struct node *delinbetween(struct node *head,int idx)
{
   struct node *p=head;  struct node *q=head->next;//q stores the address of that struct node that is
pointed by head next.
   int i=0; for(int i=0;i<idx-1;i++)
   { p=p->next;
     q=q->next;}
   p->next=q->next;
   free(q);
   return (head)};
  struct node *delatend(struct node *head)
  { struct node *get=head;
   struct node *ptr=get;
   while(ptr->next=NULL)   {  ptr=ptr->next;  }
   get->next=ptr->next;
   free(ptr);
   return (head);};
int main()
{    struct node *head;
   struct node *second;
   struct node *third;
   struct node *fourth;
   head=(struct node *)malloc(sizeof(struct node));
   second=(struct node *)malloc(sizeof(struct node));
   third=(struct node *)malloc(sizeof(struct node));
   fourth=(struct node *)malloc(sizeof(struct node));
   head->data=1;
   head->next=second;
   second->data=2;
   second->next=third;
   third->data=3;
   third->next=fourth;
   fourth->data=4;
   fourth->next=NULL;
   printf("\n Linked list before deletion\t\t");
   printlinkedlist(head);
   printf("\n Linked list after deletion at starting of linked list\t\t");
   head=delfirstnode(head);
   printlinkedlist(head);
   printf("\n linked list after deletion in between the linked list");
   head=delinbetween(head,1);
   printlinkedlist(head);
   printf("\n linked list after deletion from end");
   head=delatend(head);
   printlinkedlist(head);}
```

OUTPUT

```
Linked list before deletion
 1
 2
 3
 4
 Linked list after deletion at starting of linked list
 2
 3
 4
 linked list after deletion in between the linked list
 2
 4
 linked list after deletion from end
 2
```

Program of insertion in linked list

```
# include <stdio.h>
# include <stdlib.h>
struct node{
     int data;
     struct node *next;};
struct node *head=NULL;
void printlinkedlist(struct node*ptr)
{     while(ptr!=NULL)
    {   printf("%d\n ",ptr->data);  ptr=ptr->next;
    }}
void searched(struct node *ptr,int data)
{     int loc=0; while(ptr!=NULL) {
           if(data==ptr->data)
           {  printf("\n %d is sucessfully searched in linked list",data);
             loc++; }
           ptr=ptr->next;  }
            if(loc==0)
            {  printf("\n %d is not sucessfully searched in linked list",data);

            }}struct node *insertatfirst(struct node *first,int data)
{
      head=first;
```

```
   struct node *start=(struct node *)malloc(sizeof(struct node));
   start->data=data;
   start->next=head;
   head=start;
   return head;
   };
struct node *insertinbetween(struct node *head,int data,int idx)
{
   struct node *ptr=(struct node *)malloc(sizeof(struct node));
   struct node *p=head;
   int i=0;
   while(i!=(idx-1))
   {  p=p->next;
      i++; }
   ptr->next=p->next;
   ptr->data=data;
   p->next=ptr;
   return head;};
struct node *insertatend(struct node *head,int data)
{ struct node *kt=(struct node *)malloc(sizeof(struct node));
   kt->data=data;
   struct node *p=head;
   while(p->next!=NULL)  { p=p->next; }
   kt->next=p->next;
   p->next=kt;
   return head;};
struct node *insertafter(struct node *previous,struct node *head,int data)
{ struct node *ptr=(struct node *)malloc(sizeof(struct node));
   ptr->next=previous->next;
   previous->next=ptr;
   ptr->data=data;
   return head;
};int main(){
   struct node *ptr;
   struct node *second;
   struct node *third;
   ptr=(struct node*)malloc(sizeof(struct node));
   second=(struct node*)malloc(sizeof(struct node));
   third=(struct node*)malloc(sizeof(struct node)); ptr->data=1;
   ptr->next=second;
   second->data=2;
   second->next=third;
   third->data=3;
   third->next=NULL;
   printf("\n linked list before insertion");
```

```
    printlinkedlist(ptr);
    printf("\n linked list after insertion at first");
    ptr=insertatfirst(ptr,11);
    printlinkedlist(ptr);
    searched(ptr,2);
    ptr=insertinbetween(ptr,7,1);
    printf("\n linked list in between");
    printlinkedlist(ptr);
    printf("\n linked list at end");
    ptr=insertatend(head,9);
    printlinkedlist(head);
    printf("n linked list after a node");
    head=insertafter(second,ptr,56);
    printlinkedlist(ptr);}
```

OUTPUT

```
linked list before insertion1
 2
 3 linked list after insertion at first11
 1
 2
 3
2 is sucessfully searched in linked list
 linked list in between11
 7
 1
 2
 3
 linked list at end11
 7
 1
 2
 3
 9
 n linked list after a node11
 7
 1
 2
 56
 3
 9
```

DELETION IN DOUBLY LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int num;
    struct node * preptr;
    struct node * nextptr;
}*stnode, *ennode;
 void DlListcreation(int n);
void DlListDeleteFirstNode();
void DlListDeleteLastNode();
void DlListDeleteAnyNode(int pos);
void displayDlList(int a);
int main()
{
    int n,num1,a,insPlc;
    stnode = NULL;
    ennode = NULL;
              printf("\n\n Doubly Linked List : Delete node from any position of a doubly linked list :\n");
              printf("----------------------------------------------------------------------------------\n");
    printf(" Input the number of nodes (3 or more ): ");
    scanf("%d", &n);
    DlListcreation(n);
    a=1;
    displayDlList(a);
    printf(" Input the position ( 1 to %d ) to delete a node : ",n);
    scanf("%d", &insPlc);
   if(insPlc<1 || insPlc>n)
    {
     printf("\n Invalid position. Try again.\n ");
    }
                    if(insPlc>=1 && insPlc<=n)
 {
     DlListDeleteAnyNode(insPlc);
        a=2;
    displayDlList(a);
      }
    return 0;}
 void DlListcreation(int n)
{
    int i, num;
    struct node *fnNode;
  if(n >= 1)
    {
        stnode = (struct node *)malloc(sizeof(struct node));

        if(stnode != NULL)
        {
            printf(" Input data for node 1 : "); // assigning data in the first node
            scanf("%d", &num);
           stnode->num = num;
            stnode->preptr = NULL;
            stnode->nextptr = NULL;
            ennode = stnode;
            for(i=2; i<=n; i++)
```

```
          {
              fnNode = (struct node *)malloc(sizeof(struct node));
              if(fnNode != NULL)
              {
                  printf(" Input data for node %d : ", i);
                  scanf("%d", &num);
                  fnNode->num = num;
                  fnNode->preptr = ennode;    // new node is linking with the                 previous node
                  fnNode->nextptr = NULL;     // set next address of fnnode is NULL
                  ennode->nextptr = fnNode;   // previous node is linking with the new node
                  ennode = fnNode;            // assign new node as last node  }
              else
              { printf(" Memory can not be allocated."); break;}
          }   }
      else
      {
          printf(" Memory can not be allocated.");
      } }}void DlListDeleteAnyNode(int pos)
{
    struct node *curNode;
    int i;

    curNode = stnode;
    for(i=1; i<pos && curNode!=NULL; i++){
        curNode = curNode->nextptr;
   }  if(pos == 1)
    {
        DlListDeleteFirstNode();
    }
    else if(curNode == ennode)
    {
        DlListDeleteLastNode();
    }
    else if(curNode != NULL)
    { curNode->preptr->nextptr = curNode->nextptr;
        curNode->nextptr->preptr = curNode->preptr;

        free(curNode); //Delete the n node
    }
    else
    {
        printf(" The given position is invalid!\n");
    }
}

void DlListDeleteFirstNode()
{
    struct node * NodeToDel;
    if(stnode == NULL)
    {
        printf(" Delete is not possible. No data in the list.\n");
    }
    else
    {
        NodeToDel = stnode;
        stnode = stnode->nextptr;   // move the next address of starting node to 2 node
        stnode->preptr = NULL;      // set previous address of staring node is NULL
        free(NodeToDel);            // delete the first node from memory
    }
}

void DlListDeleteLastNode()
{
    struct node * NodeToDel;

    if(ennode == NULL)
    {
        printf(" Delete is not possible. No data in tin the list.\n");
    }
    else
    {
        NodeToDel = ennode;
        ennode = ennode->preptr;    // move the previous address of the last node to 2nd last node
        ennode->nextptr = NULL;     // set the next address of last node to NULL
        free(NodeToDel);            // delete the last node
    }
}
void displayDlList(int m)
{
    struct node * tmp;
    int n = 1;
    if(stnode == NULL)
    {
        printf(" No data found in the List yet.");
    }
    else
    {
        tmp = stnode;
        if (m==1)
        {
        printf("\n Data entered in the list are :\n");
        }
        else
        {
         printf("\n After deletion the new list are :\n");
        }
        while(tmp != NULL)
        {
            printf(" node %d : %d\n", n, tmp->num);
            n++; tmp = tmp->nextptr; // current pointer moves to the next node
        }
    }
```

```
}
```

Output

```
Doubly Linked List : Delete node from any position of a doubly linked list :
--------------------------------------------------------------------------------                    Input the number of
nodes (3 or more ): 3
 Input data for node 1 : 1
 Input data for node 2 : 2
 Input data for node 3 : 3
  Data entered in the list are :
 node 1 : 1
 node 2 : 2
 node 3 : 3
 Input the position ( 1 to 3 ) to delete a node : 3
 After deletion the new list are :
 node 1 : 1
 node 2 : 2
```

STACK IMPLEMENTATION USING LINKED LIST

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

struct stack{
   int data;
   struct stack *next;

}*top;

void push(int data)
{
   top=0;
   struct stack *newnode=(struct stack *)malloc(sizeof(struct stack));
   newnode->data=data;
   newnode->next=top;
   top=newnode;
}
void pop()
{
   struct stack *ptr=top;
   printf("\n popped data is %d",ptr->data);
   top=ptr->next;
   free(ptr);
}

void main()
{
   push(2);
   push(3);
   push(4);
   push(7);
   pop();
   pop();
}
```

Output

```
popped data is 7
```

TOWER OF HANOI

```
#include <stdio.h>

void toH(int n, char rodA, char rodC, char rodB)
{
        if (n == 1)
        {
                printf("\n Move disk 1 from rod %c to rod %c",rodA ,rodC );
                return;
        }
        toH(n-1, rodA, rodB, rodC);
        printf("\n Move disk %d from rod %c to rod %c", n, rodA, rodC);
        toH(n-1, rodB, rodC,rodA);
}

int main()
{
        int no_of_disks ;
        printf("Enter number of disks: ");
        scanf("%d", &no_of_disks);
        toH(no_of_disks, 'A','C','B');
        return 0;
}
```

Output

```
Enter the number of disks: 4
Move disk 1 from rod A to rod B
 Move disk 2 from rod A to rod C
 Move disk 1 from rod B to rod C
 Move disk 3 from rod A to rod B
 Move disk 1 from rod C to rod A
 Move disk 2 from rod C to rod B
 Move disk 1 from rod A to rod B
 Move disk 4 from rod A to rod C
 Move disk 1 from rod B to rod C
 Move disk 2 from rod B to rod A
 Move disk 1 from rod C to rod A
 Move disk 3 from rod B to rod C
 Move disk 1 from rod A to rod B
 Move disk 2 from rod A to rod C
 Move disk 1 from rod B to rod C
```