```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct NODE
{
int info;
struct NODE*link;
};
typedef struct NODE*node;
node getnode()
{
node x;
x=(node)malloc(sizeof(struct NODE));
if(x==NULL)
{
printf("out of mamory\n");
exit(0);
}
return x;
}
node ins_front(node first,int item)
{
node temp;
temp=getnode();
temp->info=item;
temp->link=first;
return temp;
}
node extract(char *s,node head)
{
        int i,n;
        for(i=0;i<strlen(s);i++)
        {
        n=s[i]-'0';
        head=ins_front(head,n);
        }
        return head;
}

node addlong(node head1,node head2,node head3)
{
        int temp,sum,carry=0;
        node cur1,cur2;
        cur1=head1;
        cur2=head2;
        while(cur1!=NULL&&cur2!=NULL)
        {
                temp=cur1->info+cur2->info+carry;
                if(temp>9)
                {
                sum=temp%10;
                carry=temp/10;
                }
                else
                {
                sum=temp;
                carry=0;
                }
                head3=ins_front(head3,sum);
                cur1=cur1->link;
                cur2=cur2->link;
        }
        while(cur1!=NULL)
        {
        temp=cur1->info+carry;
        if(temp>9)
        {
        sum=temp%10;
        carry=temp/10;
        }
        else
        {
        sum=temp;
        carry=0;
        }
        head3=ins_front(head3,sum);
        cur1=cur1->link;
        }
  while(cur2!=NULL)
        {
        temp=cur2->info+carry;
        if(temp>9)
        {
        sum=temp%10;
        carry=temp/10;
        }
        else
        {
        sum=temp;
        carry=0;
        }
        head3=ins_front(head3,sum);
        cur2=cur2->link;
        }

        if(cur1==NULL&&cur2==NULL)
        {
                if(carry==1)
                head3=ins_front(head3,carry);
        }
        return head3;
  }


void display(node first)
{
node cur;
if(first==NULL)
{
printf("Empty\n");
return;
```

```
cur=first;
while(cur!=NULL)
{
printf("%d\t",cur->info);
cur=cur->link;
}
}
void main()
{
        int ch;
        node head1=NULL;
        node head2=NULL;
        node head3=NULL;
        char s1[30],s2[30];
        clrscr();
        printf("\nEnter first integer\n");
        scanf("%s",s1);
        head1=extract(s1,head1);
        display(head1);
        printf("\nEnter second integer\n");
        scanf("%s",s2);
        head2=extract(s2,head2);
        display(head2);
        head3=addlong(head1,head2,head3);
        printf("\nThe result is\n");
        display(head3);
        getch();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct node
{
        float cf;
        float px;
        float py;
        struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
        NODE x;
        x=(NODE)malloc(sizeof(struct node));
        if (x==NULL)
        {
                printf("Memory full\n");
                exit(0);
        }
        return x;
}
NODE insert_rear(float cf,float x,float y,NODE first)
{
        NODE temp,cur;
        temp=getnode();
        temp->cf=cf;
        temp->px=x;
        temp->py=y;
        temp->link=NULL;
        if (first==NULL)
        {
                return temp;
        }
        cur=first;
        while (cur->link!=NULL)
        {
                cur=cur->link;
        }
        cur->link=temp;
        return first;
}
NODE read_poly(NODE first)
{
        int i;
        float cf,px,py;
        printf("Enter -999 to end the polynomial:\n");
        for (i=1;;i++)
        {
                printf("Enter %d term:\n",i);
                printf("Coefficient:\n");
                scanf("%f",&cf);
                if (cf==-999)
                {
                        break;
                }
                printf("Power of x:\n");
                scanf("%f",&px);
                printf("Power of y:\n");
                scanf("%f",&py);
                first=insert_rear(cf,px,py,first);
        }
        return first;
}
float evaluate_polynomial(NODE first)
{
        float x,y,sum=0;
        NODE polynomial;
        printf("Enter the values of x and y:\n");
        scanf("%f%f",&x,&y);
        polynomial=first;
        while (polynomial!=NULL)
        {
                sum=sum+polynomial->cf*pow(x,polynomial-
>px)*pow(y,polynomial->py);
                polynomial=polynomial->link;
        }
        return sum;
}
void display(NODE first)
{
        NODE temp;
        if (first==NULL)
        {
                printf("Polynomial does not exist\n");
        }
        else
        {
                temp=first;
                while (temp->link!=NULL)
                {
                        printf("(%5.2fx^%3.2fy^%3.2f)\t+",temp-
>cf,temp->px,temp->py);
                        temp=temp->link;
                }
                printf("(%5.2fx^%3.2fy^%3.2f)\n",temp->cf,temp-
>px,temp->py);
    }
}
int main(){
    NODE first;
    float res;
    first=NULL;
    printf("Enter the polynomial: \n");
    first = read_poly(first);
    res = evaluate_polynomial(first);
    printf("Polynomial is: \n");
    display(first);
    printf("Result is %f\n",res);
}
```

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>

struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE insert_front(int, NODE);
NODE insert_rear(int, NODE );
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE read_number(NODE head)
{
char c;
while((c=getche())!='\r')
{
 if(isdigit(c))
 head=insert_front(c-'0',head);
 else
 {
 printf("invalid integer\n");
 exit(1);
 }
}
printf("\n");
return head;
}
NODE add_long(NODE h1,NODE h2,NODE h3)
{
 NODE c,c1,c2,h;
 int sum,carry,digit;
 carry=0;
 c1=h1->link;
 c2=h2->link;
 while(c1!=h1 && c2!=h2)
 {
 sum=c1->info+c2->info+carry;
 digit=sum%10;
 carry=sum/10;
```

```
 h3=insert_rear(digit,h3);
 c1=c1->link;
 c2=c2->link;
 }
 if(c1!=h1)
 c=c1,h=h1;
 else
 c=c2,h=h2;
 while(c!=h)
 {
 sum=c->info+carry;
 digit=sum%10;
 carry=sum/10;
 h3=insert_rear(digit,h3);
 c=c->link;
 }
 if(carry==1)
 h3=insert_rear(carry,h3);
 return h3;
}
NODE insert_front(int item, NODE last)
{
NODE temp;
temp=getnode();
temp->info=item;
if(last==NULL)
last=temp;
else
temp->link=last->link;
last->link=temp;
return last;
}
NODE insert_rear(int item, NODE last)
{
NODE temp;
temp=getnode();
temp->info=item;
if(last==NULL)
last=temp;
else
temp->link=last->link;
last->link=temp;
return last;
}
void display_number(NODE head)
{
 int k,n,*a;
 NODE cur;
 if(head->link==head)
 {
 printf("number doesnot exist\n");
 return;
 }
 n=head->info;
 a=(int *)malloc(n*sizeof(int));
 cur=head->link,k=0;
```

```
 {
 a[k++]=cur->info;
 cur=cur->link;
 }
 while(--k!=-1)
 printf("%d",a[k]);
 printf("\n");
}
void main()
{
NODE h1,h2,h3;
clrscr();
h1=getnode();
h2=getnode();
h3=getnode();
h1->link=h1;
h2->link=h2;
h3->link=h3;
h1->info=h2->info=h3->info=0;
printf("enter the first number\n");
h1=read_number(h1);
printf("enter the second number\n");
h2=read_number(h2);
h3=add_long(h1,h2,h3);
printf("num1=");
display_number(h1);
printf("num2=");
display_number(h2);
printf("sum=");
display_number(h3);
getch();
}
```

```c
/*ascending*/
#include<stdio.h>
#include<stdlib.h>
#define QUE_SIZE 5
int item,rear=-1, q[QUE_SIZE],count=0;
void insertrear()
{
    if(rear== QUE_SIZE-1)
    {
        printf("QUEUE OVERFLOW \n");
        return ;
    }

    rear=rear+1;
    q[rear]= item;
    count++;

}
int deleteasc()
{
    int small=99;
    int spos=-1;
    if(count==0)
        {return -1;
    }
    for(int i=0;i<QUE_SIZE;i++)
    {
        if(q[i]<small)
        {
          small=q[i];
          spos=i;
        }

    }
    q[spos]=99;
    count=count-1;
    return small;
}

void display()
{
    int i;
    if(count==0)
    {
        printf("QUEUE IS EMPTY \n");
        return;
    }


    printf("CONTENTS OF QUEUE \n");
    for(i=0; i<QUE_SIZE; i++)
    {
       if(q[i]==99)
        continue;
       else
        printf("%d ",q[i]);
    }
}

void main()
{
    int choice ;
    for(;;)
    {
        printf("\nENTER 1.insert rear 2.delete  3.DISPLAY \n");
        printf("ENTER CHOICE \n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("ENTER ITEM TO BE ENTERED \n");
                    scanf("%d", &item);
                    insertrear();
                    break;
            case 2: item= deleteasc();
                    if(item == -1)
                        printf("QUEUE EMPTY \n");
                    else
                        printf("ITEM DELETED IS %d", item);
                    break;
            case 3: display();
                    break;
            default: exit(0);
        }
    }
}
```

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
 {
  int info;
  struct node *rlink;
  struct node *llink;
 };
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert(NODE root,int item)
{
NODE temp,cur,prev;
temp=getnode();
temp->rlink=NULL;
temp->llink=NULL;
temp->info=item;
if(root==NULL)
 return temp;
prev=NULL;
cur=root;
while(cur!=NULL)
{
prev=cur;
cur=(item<cur->info)?cur->llink:cur->rlink;
}
if(item<prev->info)
 prev->llink=temp;
else
 prev->rlink=temp;
return root;
}
void display(NODE root,int i)
{
int j;
if(root!=NULL)
 {
  display(root->rlink,i+1);
  for(j=0;j<i;j++)
          printf("   ");
   printf("%d\n",root->info);
         display(root->llink,i+1);
 }
}
NODE delete(NODE root,int item)
{
NODE cur,parent,q,suc;
if(root==NULL)
{
printf("empty\n");
return root;
}
parent=NULL;
cur=root;
while(cur!=NULL&&item!=cur->info)
{
parent=cur;
cur=(item<cur->info)?cur->llink:cur->rlink;
}
if(cur==NULL)
{
 printf("not found\n");
 return root;
}
if(cur->llink==NULL)
 q=cur->rlink;
else if(cur->rlink==NULL)
 q=cur->llink;
else
 {
 suc=cur->rlink;
 while(suc->llink!=NULL)
  suc=suc->llink;
 suc->llink=cur->llink;
 q=cur->rlink;
 }
 if(parent==NULL)
  return q;
 if(cur==parent->llink)
  parent->llink=q;
 else
  parent->rlink=q;
 freenode(cur);
 return root;
 }

void preorder(NODE root)
{
if(root!=NULL)
 {
  printf("%d\n",root->info);
  preorder(root->llink);
  preorder(root->rlink);
  }
 }
void postorder(NODE root)
```

```
if(root!=NULL)
 {

  postorder(root->llink);
  postorder(root->rlink);
  printf("%d\n",root->info);
  }
 }
void inorder(NODE root)
{
if(root!=NULL)
 {

  inorder(root->llink);
  printf("%d\n",root->info);
  inorder(root->rlink);
  }
 }
void main()
{
int item,choice;
NODE root=NULL;
clrscr();
for(;;)
{
printf("\n1.insert\n2.display\n3.pre\n4.post\n5.in\n6.delete\n7.exit\n
");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item\n");
                scanf("%d",&item);
                root=insert(root,item);
                break;
  case 2:display(root,0);
                break;
  case 3:preorder(root);
                break;
  case 4:postorder(root);
                break;
  case 5:inorder(root);
                break;
  case 6:printf("enter the item\n");
                scanf("%d",&item);
                root=delete(root,item);
                break;
  default:exit(0);
                 break;
         }
       }
 }
```

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<alloc.h>
struct node
{
int info;
struct node*llink;
struct node*rlink;
};
typedef struct node*NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("memory not available");
exit(0);
}
return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert(int item,NODE root)
{
NODE temp,cur,prev;
char direction[10];
int i;
temp=getnode();
temp->info=item;
temp->llink=NULL;
temp->rlink=NULL;
if(root==NULL)
 return temp;
printf("give direction to insert\n");
scanf("%s",direction);
prev=NULL;
cur=root;
for(i=0;i<strlen(direction)&&cur!=NULL;i++)
{
prev=cur;
if(direction[i]=='l')
cur=cur->llink;
else
cur=cur->rlink;
}
if(cur!=NULL||i!=strlen(direction))
{
printf("insertion not possible\n");
freenode(temp);
return(root);
}
if(cur==NULL)
{
if(direction[i-1]=='l')
prev->llink=temp;
else
prev->rlink=temp;
}
return(root);
}
void preorder(NODE root)
{
if(root!=NULL)
{
printf("the item is %d\n",root->info);
preorder(root->llink);
preorder(root->rlink);
}
}
void inorder(NODE root)
{
if(root!=NULL)
{
inorder(root->llink);
printf("the item is%d\n",root->info);
inorder(root->rlink);
}
}
void postorder(NODE root)
{
if (root!=NULL)
{
postorder(root->llink);
postorder(root->rlink);
printf("the item is%d\n",root->info);
}
}
void display(NODE root,int i)
{
int j;
if(root!=NULL)
{
display(root->rlink,i+1);
for (j=1;j<=i;j++)
printf("  ");
printf("%d\n",root->info);
display(root->llink,i+1);
}
}

void main()
{
NODE root=NULL;
int choice,i,item;
clrscr();
for(;;)
{
printf("1.insert\n2.preorder\n3.inorder\n4.postorder\n5.display\n");
```

```
scanf("%d",&choice);
switch(choice)
{
case 1: printf("enter the item\n");
                scanf("%d",&item);
                root=insert(item,root);
                break;
case 2: if(root==NULL)
                {
                 printf("tree is empty");
                }
                else
                {
                 printf("given tree is");
                 display(root,1);
                 printf("the preorder traversal is \n");
                 preorder(root);
                }
                break;
case 3:if(root==NULL)
           {
                printf("tree is empty");
           }
           else
           {
                printf("given tree is");
                display(root,1);
                printf("the inorder traversal is \n");
                inorder(root);
                }
           break;
case 4:if (root==NULL)
                {
                printf("tree is empty");
                 }
           else
           {
                printf("given tree is");
                display(root,1);
                printf("the postorder traversal is \n");
                postorder(root);
           }
          break;
case 5:display(root,1);
           break;
default:exit(0);
}
}
}
```

```c
/*descending*/
#include<stdio.h>
#include<stdlib.h>
#define QUE_SIZE 3
int item,rear=-1, q[QUE_SIZE],count=0;
void insertrear()
{
    if(rear== QUE_SIZE-1)
    {
        printf("QUEUE OVERFLOW \n");
        return ;
    }

    rear=rear+1;
    q[rear]= item;
    count++;

}

int deletedesc()
{
    int largest=-1;
    int spos=-1;
    if(count==0)
        {return -1;
    }
    for(int i=0;i<QUE_SIZE;i++)
    {
        if(q[i]>largest)
        {
          largest=q[i];
          spos=i;
        }

    }

    q[spos]=-1;
    count=count-1;
    return largest;
}

void display()
{
    int i;
    if(count==0)
    {
        printf("QUEUE IS EMPTY \n");
        return;
    }


    printf("CONTENTS OF QUEUE \n");
    for(i=0; i<QUE_SIZE; i++)
    {
       if(q[i]==0)
        continue;
       else
        printf("%d ",q[i]);
    }
}

void main()
{
    int choice ;
    for(;;)
    {
        printf("\nENTER 1.insert rear 2.delete  3.DISPLAY \n");
        printf("ENTER CHOICE \n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("ENTER ITEM TO BE ENTERED \n");
                    scanf("%d", &item);
                    insertrear();
                    break;
            case 2: item= deletedesc();
                    if(item == -1)
                        printf("QUEUE EMPTY \n");
                    else
                        printf("ITEM DELETED IS %d", item);
                    break;
            case 3: display();
                    break;
            default: exit(0);
        }
    }
}
```

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
        int info;
        struct node *rlink;
        struct node *llink;
};
typedef struct node *NODE;
NODE getnode()
{
        NODE x;
        x=(NODE)malloc(sizeof(struct node));
        if (x==NULL)
        {
                printf("Memory full\n");
                exit(0);
        }
        return x;
}
NODE dinsert_front(int item,NODE head)
{
        NODE temp,cur;
        temp=getnode();
        temp->info=item;
        temp->llink=NULL;
        temp->rlink=NULL;
        cur=head->rlink;
        head->rlink=temp;
        temp->llink=head;
        temp->rlink=cur;
        cur->llink=temp;
        return head;
}
NODE dinsert_rear(int item,NODE head)
{
        NODE temp,cur;
        temp=getnode();
        temp->info=item;
        temp->llink=NULL;
        temp->rlink=NULL;
        cur=head->llink;
        head->llink=temp;
        temp->rlink=head;
        cur->rlink=temp;
        temp->llink=cur;
        return head;
}
NODE ddelete_front(NODE head)
{
        NODE cur,next;
        if (head->rlink==head)
        {
                printf("List is empty\n");
                return head;
        }
        cur=head->rlink;
        next=cur->rlink;
        head->rlink=next;
        next->llink=head;
        printf("Item deleted at the front end is:%d\n",cur->info);
        free(cur);
        return head;
}
NODE ddelete_rear(NODE head)
{
        NODE cur,prev;
        if (head->rlink==head)
        {
                printf("List is empty\n");
                return head;
        }
        cur=head->llink;
        prev=cur->llink;
        prev->rlink=head;
        head->llink=prev;
        printf("Item deleted at the rear end is:%d\n",cur->info);
        free(cur);
        return head;
}
void ddisplay(NODE head)
{
        NODE temp;
        if (head->rlink==head)
        {
                printf("List is empty\n");
        }
        printf("The contents of the list are:\n");
        temp=head->rlink;
        while (temp!=head)
        {
                printf("%d\n",temp->info);
                temp=temp->rlink;
        }
}
void dsearch(int key,NODE head)
{
        NODE cur;
        int count;
        if (head->rlink==head)
        {
                printf("List is empty\n");
        }
        cur=head->rlink;
        count=1;
        while (cur!=head && cur->info!=key)
        {
                cur=cur->rlink;
                count++;
        }
        if (cur==head)
        {
```

```
        }
        else
        {
                printf("Key element found at the position
%d\n",count);
        }
}
NODE dinsert_leftpos(int item,NODE head)
{
        NODE cur,prev,temp;
        if (head->rlink==head)
        {
                printf("List is empty\n");
                return head;
        }
        cur=head->rlink;
        while (cur!=head)
        {
                if (cur->info==item)
                {
                        break;
                }
                cur=cur->rlink;
        }
        if (cur==head)
        {
                printf("No such item found in the list\n");
                return head;
        }
        prev=cur->llink;
        temp=getnode();
        temp->llink=NULL;
        temp->rlink=NULL;
        printf("Enter the item to be inserted at the left of the given
item:\n");
        scanf("%d",&temp->info);
        prev->rlink=temp;
        temp->llink=prev;
        temp->rlink=cur;
        cur->llink=temp;
        return head;
}
NODE dinsert_rightpos(int item,NODE head)
{
        NODE temp,cur,next;
        if (head->rlink==head)
        {
                printf("List is empty\n");
                return head;
        }
        cur=head->rlink;
        while (cur!=head)
        {
                if (cur->info==item)
                {
                        break;
                }
                cur=cur->rlink;
        }
        if (cur==head)
        {
                printf("No such item found in the list\n");
                return head;
        }
        next=cur->rlink;
        temp=getnode();
        temp->llink=NULL;
        temp->rlink=NULL;
        printf("Enter the item to be inserted at the right of the
given item:\n");
        scanf("%d",&temp->info);
        cur->rlink=temp;
        temp->llink=cur;
        next->llink=temp;
        temp->rlink=next;
        return head;
}
NODE ddelete_duplicates(int item,NODE head)
{
        NODE prev,cur,next;
        int count=0;
        if (head->rlink==head)
        {
                printf("List is empty\n");
                return head;
        }
        cur=head->rlink;
        while (cur!=head)
        {
                if (cur->info!=item)
                {
                        cur=cur->rlink;
                }
                else
                {
                        count++;
                        if (count==1)
                        {
                                cur=cur->rlink;
                                continue;
                        }
                        else
                        {
                                prev=cur->llink;
                                next=cur->rlink;
                                prev->rlink=next;
                                next->llink=prev;
                                free(cur);
                                cur=next;
                        }
                }
        }
        if (count==0)
```

```
                printf("No such item found in the list\n");
        }
        else
        {
                printf("Removed all the duplicate elements of the
given item successfully\n");
        }
        return head;
}
int main()
{
NODE head;
int item, choice,key;
head=getnode();
head->llink=head;
head->rlink=head;
for(;;)
{
        printf("\n1:dinsert front\n2:dinsert rear\n3:ddelete
front\n4:ddelete rear\n5:ddisplay\n6:dsearch\n7:dinsert
lestpos\n8:dinsert rightpos\n9:ddelete duplicates\n10:exit\n");
        printf("enter the choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1: printf("Enter the item at front end:\n");
                                scanf("%d",&item);
                                head=dinsert_front(item,head);
                                break;
                case 2: printf("Enter the item at rear end:\n");
                                scanf("%d",&item);
                                head=dinsert_rear(item,head);
                                break;
                case 3:head=ddelete_front(head);
                           break;
                case 4:head=ddelete_rear(head);
                           break;
                case 5:ddisplay(head);
                           break;
            case 6:printf("Enter the key element to be searched:\n");
                           scanf("%d",&key);
                           dsearch(key,head);
                           break;
            case 7:printf("Enter the key element:\n");
                           scanf("%d",&key);
                           head=dinsert_leftpos(key,head);
                           break;
                case 8:printf("Enter the key element:\n");
                           scanf("%d",&key);
                           head=dinsert_rightpos(key,head);
                           break;
                case 9:printf("Enter the key element whose duplicates
should be removed:\n");
                           scanf("%d",&key);
                           head=ddelete_duplicates(key,head);
                           break;
                }
        }
        return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct node
{
        float cf;
        float px;
        float py;
        int flag;
        struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
        NODE x;
        x=(NODE)malloc(sizeof(struct node));
        if (x==NULL)
        {
                printf("Memory full\n");
                exit(0);
        }
        return x;
}
NODE insert_rear(float cf,float x,float y,NODE first)
{
        NODE temp,cur;
        temp=getnode();
        temp->cf=cf;
        temp->px=x;
        temp->py=y;
        temp->link=NULL;
        temp->flag=0;
        if (first==NULL)
        {
                return temp;
        }
        cur=first;
        while (cur->link!=NULL)
        {
                cur=cur->link;
        }
        cur->link=temp;
        return first;
}
NODE read_poly(NODE first)
{
        int i;
        float cf,px,py;
        printf("Enter -999 to end the polynomial:\n");
        for (i=1;;i++)
        {
                printf("Enter %d term:\n",i);
                printf("Coefficient:\n");
                scanf("%f",&cf);
                if (cf==-999)
                {
                        break;
                }
                printf("Power of x:\n");
                scanf("%f",&px);
                printf("Power of y:\n");
                scanf("%f",&py);
                first=insert_rear(cf,px,py,first);
        }
        return first;
}
void display(NODE first)
{
        NODE temp;
        if (first==NULL)
        {
                printf("Polynomial does not exist\n");
        }
        else
        {
                temp=first;
                while (temp->link!=NULL)
                {
                        printf("(%5.2fx^%3.2fy^%3.2f)\t+",temp-
>cf,temp->px,temp->py);
                        temp=temp->link;
                }
                printf("(%5.2fx^%3.2fy^%3.2f)\n",temp->cf,temp-
>px,temp->py);
    }
}
NODE add_poly(NODE f1,NODE f2,NODE f3)
{
        NODE p1,p2;
        int x1,x2,y1,y2,cf1,cf2,cof;
        p1=f1;
        while (p1!=NULL)
        {
                x1=p1->px;
                y1=p1->py;
                cf1=p1->cf;
                p2=f2;
                while (p2!=NULL)
                {
                        x2=p2->px;
                        y2=p2->py;
                        cf2=p2->cf;
                        if (x1==x2 && y1==y2)
                        {
                                break;
                        }
                        p2=p2->link;
                }
                if (p2!=NULL)
                {
                        cof=cf1+cf2;
                        p2->flag=1;
```

```
                        {
                                f3=insert_rear(cof,x1,y1,f3);
                        }
                }
                else
                {
                        f3=insert_rear(cf1,x1,y1,f3);
                }
                p1=p1->link;
        }
        p2=f2;
        while (p2!=NULL)
        {
                if (p2->flag==0)
                {
                        f3=insert_rear(p2->cf,p2->px,p2->py,f3);
                }
                p2=p2->link;
        }
        return f3;
}
int main(){
    NODE f1,f2,f3;
    f1=NULL;
    f2=NULL;
    f3=NULL;
    printf("Enter the first polynomial: \n");
    f1 = read_poly(f1);
    printf("Enter the second polynomial: \n");
    f2 = read_poly(f2);
    f3=add_poly(f1,f2,f3);
    printf("The first polynomial is: \n");
    display(f1);
    printf("The second polynomial is: \n");
    display(f2);
    printf("The sum of two polynomial is: \n");
    display(f3);
    return 0;
}
```

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define qsize 5
int f=0,r=-1,ch;
int item,q[10];

int isfull()
  {
   return(r==qsize-1)?1:0;
  }
int isempty()
  {
   return(f>r)?1:0;
  }
void insert_rear()
  {
   if(isfull())
         {
          printf("queue overflow\n");
          return;
         }
   r=r+1;
   q[r]=item;
  }
void delete_front()
  {
   if(isempty())
         {
          printf("queue empty\n");
          return;
         }
   printf("item deleted is %d\n",q[(f)++]);
   if(f>r)
         {
          f=0;
          r=-1;
         }
  }
void insert_front()
  {
   if(f!=0)
         {
          f=f-1;
          q[f]=item;
          return;
         }
        else if((f==0)&&(r==-1))
         {
          q[++(r)]=item;
          return;
         }
        else
          printf("insertion not possible\n");
  }
void delete_rear()
  {
   if(isempty())
         {
          printf("queue is empty\n");
          return;
         }
   printf("item deleted is %d\n",q[(r)--]);
   if(f>r)
         {
          f=0;
          r=-1;
         }
  }
void display()
  {
   int i;
   if(isempty())
         {
          printf("queue empty\n");
          return;
         }
   for(i=f;i<=r;i++)
        printf("%d\n",q[i]);
  }
void main()
 {

  clrscr();
  for(;;)
   {
        printf("1.insert_rear\n2.insert_front\n3.delete_rear\n4.delete
_front\n5.display\n6.exit\n");
        printf("enter choice\n");
        scanf("%d",&ch);
        switch(ch)
          {
           case 1:printf("enter the item\n");
                        scanf("%d",&item);
                        insert_rear();
                        break;
           case 2:printf("enter the item\n");
                        scanf("%d",&item);
                        insert_front();
                        break;
           case 3:delete_rear();
                        break;
           case 4:delete_front();
                        break;
           case 5:display();
                        break;
           default:exit(0);
          }
        }
   getch();
  }
```

C-Program to implement CIRCULAR LINKED LIST

```
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
 {
  int info;
  struct node *link;
 };
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE cir_insert_front(NODE last,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
if(last==NULL)
 last=temp;
temp->link=last->link;
last->link=temp;
return last;
}
NODE cir_insert_rear(NODE last,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
if(last==NULL)
last=temp;
else
temp->link=last->link;
last->link=temp;
return temp;
}
```

```
NODE cir_delete_front(NODE last)
{
NODE temp,first;
if(last==NULL)
{
printf("list empty\n");
return NULL;
}
if(last->link==last)
{
printf("item deleted is  %d\n",last->info);
freenode(last);
return NULL;
}
first=last->link;
last->link=first->link;
printf("item deleted at front end is %d\n",first->info);
freenode(first);
return last;
}
NODE cir_delete_rear(NODE last)
{
NODE prev;
if(last==NULL)
{
printf("list empty\n");
return NULL;
}
if(last->link==last)
{
printf("item deleted is %d\n",last->info);
freenode(last);
return NULL;
}
prev=last->link;
while(prev->link!=last)
{
prev=prev->link;
}
prev->link=last->link;
printf("item deleted at rear end is %d\n",last->info);
freenode(last);
return prev;
}
void display(NODE last)
{
NODE temp;
if(last==NULL)
{
 printf("list empty\n");
 return;
```

```
}
printf("contents of circular list are\n");
for(temp=last->link;temp!=last;temp=temp->link)
{
printf("%d\n",temp->info);
}
printf("%d\n",temp->info);
}
void main()
{
int item,choice;
NODE last=NULL;
clrscr();
for(;;)
{
printf("\n1:cir_insert_front\n2:cir_insert_rear\n3:cir_delete_front\n4:cir_delete_rear\n5:display\n
6:exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at front end\n");
                    scanf("%d",&item);
                    last=cir_insert_front(last,item);
                    break;
  case 2:printf("enter the item at rear end\n");
                    scanf("%d",&item);
                    last=cir_insert_rear(last,item);
                    break;
  case 3:last=cir_delete_front(last);
          break;
  case 4:last=cir_delete_rear(last);
          break;
  case 5:display(last);
                    break;
  default:exit(0);
  }
 }
getch();
}
```

C-Program to implement LINKED LIST with functions insertion & deletion with specified position

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
```

```
if(first->link==NULL)
{
printf("item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
printf("item deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE insert_pos(int item,int pos,NODE first)
{
NODE temp,cur,prev;
int count;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL&&pos==1)
{
return temp;
}
if(first==NULL)
{
printf("invalid position\n");
return first;
}
if(pos==1)
{
temp->link=first;
first=temp;
return temp;
}
count=1;
prev=NULL;
cur=first;
while(cur!=NULL&&count!=pos)
{
prev=cur;
cur=cur->link;
count++;
}
if(count==pos)
{
```

```
prev->link=temp;
temp->link=cur;
return first;
}
printf("invalid position\n");
return first;
}
NODE delete_pos(int pos,NODE first)
{
NODE cur;
NODE prev;
int count,flag=0;
if(first==NULL || pos<0)
{
printf("invalid position\n");
return NULL;
}
if(pos==1)
{
cur=first;
first=first->link;
freenode(cur);
return first;
}
prev=NULL;
cur=first;
count=1;
while(cur!=NULL)
{
if(count==pos){flag=1;break;}
count++;
prev=cur;
cur=cur->link;
}
if(flag==0)
{
printf("invalid position\n");
return first;
}
printf("item deleted at given position is %d\n",cur->info);
prev->link=cur->link;
freenode(cur);
return first;
}

void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("list empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
```

```
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,key,pos;
int count=0;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_rear\n 2:Delete_rear\n");
printf(" 3:insert_info_position\n 4:Delete_info_position\n 5:Display_list\n 6:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at rear-end\n");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
  case 2:first=delete_rear(first);
            break;
  case 3:printf("enter the item to be inserted at given position\n");
            scanf("%d",&item);
            printf("enter the position\n");
            scanf("%d",&pos);
            first=insert_pos(item,pos,first);
            break;
  case 4:printf("enter the position\n");
            scanf("%d",&pos);
            first=delete_pos(pos,first);
            break;
  case 5:display(first);
            break;
  default:exit(0);
            break;
 }
}
getch();
}
```

```
#include<stdio.h>
#include<conio.h>
#define N 3
int queue[3][N];
int front[3]={0,0,0};
int rear[3]={-1,-1,-1};
int item,pr;
void main()
{
int ch;
clrscr();
while(1)
{
printf("PRIORITY QUEUE\n");
printf("*****************\n");
printf("\n\t1:PQinsert\n");
printf("\n\t2:PQdelete\n");
printf("\n\t3:PQdisplay\n");
printf("\n\t4:Exit\n");
printf("\nenter the choice\n");
scanf("%d",&ch);
switch(ch)
{
 case 1:printf("\nenter the priority number\n");
                scanf("%d",&pr);
                if(pr>0 && pr<4)
                pqinsert(pr-1);
                else
                printf("\only 3 priority exists 1 2 3\n");
                break;
case 2:pqdelete();
           break;
case 3:display();
           break;
case 4:exit(0);
}
}
getch();
}
pqinsert(int pr)
{
 if(rear[pr]==N-1)
 printf("\n Queue overflow\n");
 else
 {
 printf("\nenter the item\n");
 scanf("%d",&item);
 rear[pr]++;
 queue[pr][rear[pr]]=item;
 }
 return;
}
pqdelete()
{
int i;
for(i=0;i<3;i++)
{
  if(rear[i]==front[i]-1)
  printf("\queue empty\n");
  else
  {
  printf("deleted item is %d of queue %d\n",queue[i][front[i]],i+1);
  front[i]++;
  return;
  }
 }
}
display()
{
int i,j;
for(i=0;i<3;i++)
{
if(rear[i]==front[i]-1)
  printf("\queue empty %d\n",i+1);
else
  {
  printf("\nQUEUE %d:",i+1);
  for(j=front[i];j<=rear[i];j++)
    printf("%d\t",queue[i][j]);
  }
}
  return;
}
```

C-Program to implement Double Ended Queue(DEQUEUE)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define qsize 5
int isfull(int r)
  {
   return(r==qsize-1)?1:0;
  }
int isempty(int f,int r)
  {
   return(f>r)?1:0;
  }
void insert_rear(int item,int *r,int q[])
  {
   if(isfull(*r))
            {
             printf("queue overflow\n");
             return;
            }
   *r=*r+1;
   q[*r]=item;
  }
void delete_front(int *f,int *r,int q[])
  {
   if(isempty(*f,*r))
            {
             printf("queue empty\n");
             return;
            }
   printf("item deleted is %d\n",q[(*f)++]);
   if(*f>*r)
            {
             *f=0;
             *r=-1;
            }
  }
void insert_front(int *f,int *r,int q[],int item)
  {
   if(*f!=0)
            {
             *f=*f-1;
             q[*f]=item;
             return;
            }
           else if((*f==0)&&(*r==-1))
            {
             q[++(*r)]=item;
             return;
            }
```



```
           else
             printf("insertion not possible\n");
  }
void delete_rear(int *f,int *r,int q[])
  {
   if(isempty(*f,*r))
            {
             printf("queue is empty\n");
             return;
            }
   printf("item deleted is %d\n",q[(*r)--]);
   if(*f>*r)
            {
             *f=0;
             *r=-1;
            }
  }
void display(int f,int r ,int q[])
  {
   int i;
   if(isempty(f,r))
            {
             printf("queue empty\n");
             return;
            }
    for(i=f;i<=r;i++)
           printf("%d\n",q[i]);
  }
void main()
 {
  int f=0,r=-1,item,q[10],ch;
  clrscr();
  for(;;)
   {
           printf("1.insert_rear\n2.insert_front\n3.delete_rear\n4.delete_front\n5.display\n6.exit\n");
           printf("enter choice\n");
           scanf("%d",&ch);
           switch(ch)
             {
              case 1:printf("enter the item\n");
                           scanf("%d",&item);
                           insert_rear(item,&r,q);
                           break;
              case 2:printf("enter the item\n");
                           scanf("%d",&item);
                           insert_front(&f,&r,q,item);
                           break;
              case 3:delete_rear(&f,&r,q);
                           break;
              case 4:delete_front(&f,&r,q);
                           break;
```

```
            case 5:display(f,r,q);
                                break;
            default:exit(0);
            }
          }
  getch();
 }
```

C-Program to implement LINKED LIST(insert-front, delete-front, insert-rear, delete-rear, display)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
NODE delete_front(NODE first)
```



```
{
NODE temp;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
if(first->link==NULL)
{
printf("item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
printf("iten deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
```

```
return first;
}
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("list empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,pos;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n 5:Display_list\n
6:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at front-end\n");
            scanf("%d",&item);
            first=insert_front(first,item);
            break;
  case 2:first=delete_front(first);
            break;
  case 3:printf("enter the item at rear-end\n");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
  case 4:first=delete_rear(first);
            break;
  case 5:display(first);
            break;
 default:exit(0);
            break;
 }
}
getch();
}
```


C-Program to implement QUEUE using LINKED LIST(insert-rear, delete-front ,display)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}

NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
```

```
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("list empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,pos;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_rear\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at rear-end\n");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
  case 2:first=delete_front(first);
            break;
  case 3:display(first);
            break;
 default:exit(0);
            break;
 }
}
getch();
}
```


C-Program to implement STACK using LINKED LIST(insert-rear, delete-rear ,display OR insert-front, delete-fronr ,display)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("stack is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
```

```
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("stack empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,pos;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at front-end\n");
           scanf("%d",&item);
           first=insert_front(first,item);
           break;
  case 2:first=delete_front(first);
           break;
  case 3:display(first);
           break;
 default:exit(0);
           break;
 }
}
```

OR

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
```



```
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("stack is empty cannot delete\n");
return first;
}
if(first->link==NULL)
{
printf("item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
```

```
printf("item deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("stack empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,pos;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_rear\n 2:Delete_rear\n 3:Display_list\n 4:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {

  case 1:printf("enter the item at rear-end\n");
           scanf("%d",&item);
           first=insert_rear(first,item);
           break;
  case 2:first=delete_rear(first);
           break;
  case 3:display(first);
           break;
 default:exit(0);
           break;
 }
}
getch();
}
```


C-Program to implement LINKED LIST(insert-front, delete-front, insert-rear, delete-rear, display,count the items & search the item)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp=first;
```

```
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
if(first->link==NULL)
{
printf("item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
printf("iten deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
int length(NODE first)
{
NODE cur;
int count=0;
if(first==NULL) return 0;
cur=first;
```



```
while(cur!=NULL)
{
count++;
cur=cur->link;
}
return count;
}
void search(int key,NODE first)
{
NODE cur;
if(first==NULL)
{
printf("list is empty\n");
return;
}
cur=first;
while(cur!=NULL)
{
if(key==cur->info)break;
cur=cur->link;
}
if(cur==NULL)
{
printf("search is unsuccessful\n");
return;
}
printf("search successfull\n");
}

void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("list empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,pos,count,key;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n");
printf(" 5:length_of_list\n 6:Search_key\n 7:Display_list\n 8:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
```

```
{
 case 1:printf("enter the item at front-end\n");
        scanf("%d",&item);
        first=insert_front(first,item);
        break;
 case 2:first=delete_front(first);
        break;
 case 3:printf("enter the item at rear-end\n");
        scanf("%d",&item);
        first=insert_rear(first,item);
        break;
 case 4:first=delete_rear(first);
        break;
 case 5:count=length(first);
        printf("length(items) in the list is %d\n",count);
        break;
 case 6:printf("enter the item to be searched\n");
        scanf("%d",& key);
        search(key,first);
        break;
 case 7:display(first);
        break;
default:exit(0);
        break;
 }
}
getch();
}
```


C-Program to implement LINKED LIST(insert-front, delete-front, insert-rear, delete-rear, display,order-list & delete-info)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
```

```
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
if(first->link==NULL)
{
printf("item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
printf("item deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE order_list(int item,NODE first)
{
NODE temp,prev,cur;
temp=getnode();
temp->info=item;
```



```
temp->link=NULL;
if(first==NULL) return temp;
if(item<first->info)
{
temp->link=first;
return temp;
}
prev=NULL;
cur=first;
while(cur!=NULL&&item>cur->info)
{
prev=cur;
cur=cur->link;
}
prev->link=temp;
temp->link=cur;
return first;
}
NODE delete_info(int key,NODE first)
{
NODE prev,cur;
if(first==NULL)
{
printf("list is empty\n");
return NULL;
}
if(key==first->info)
{
cur=first;
first=first->link;
freenode(cur);
return first;
}
prev=NULL;
cur=first;
while(cur!=NULL)
{
if(key==cur->info)break;
prev=cur;
cur=cur->link;
}
if(cur==NULL)
{
printf("search is unsuccessfull\n");
return first;
}
prev->link=cur->link;
printf("key deleted is %d",cur->info);
freenode(cur);
return first;
}
```

```
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("list empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
void main()
{
int item,choice,key;
NODE first=NULL;
clrscr();
for(;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n");
printf(" 5:Order_list\n 6:Delete_info\n 7:Display_list\n 8:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at front-end\n");
            scanf("%d",&item);
            first=insert_front(first,item);
            break;
  case 2:first=delete_front(first);
            break;
  case 3:printf("enter the item at rear-end\n");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
  case 4:first=delete_rear(first);
            break;
  case 5:printf("enter the item to be inserted in ordered_list\n");
            scanf("%d",&item);
            first=order_list(item,first);
            break;
  case 6:printf("enter the key to be deleted\n");
            scanf("%d",&key);
            first=delete_info(key,first);
            break;
  case 7:display(first);
            break;
  default:exit(0);
            break;
 }
}
getch();
}
```


C-Program to concatenate two lists & reverse the list(LINKED LIST)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
 {
  int info;
  struct node *link;
 };
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
  printf("list empty");

 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
NODE concat(NODE first,NODE second)
```

```
{
 NODE cur;
 if(first==NULL)
  return second;
 if(second==NULL)
  return first;
 cur=first;
 while(cur->link!=NULL)
  cur=cur->link;
 cur->link=second;
 return first;
}
NODE reverse(NODE first)
 {
 NODE cur,temp;
 cur=NULL;
 while(first!=NULL)
  {
   temp=first;
   first=first->link;
   temp->link=cur;
   cur=temp;
  }
 return cur;
}

void main()
{
int item,choice,pos,i,n;
NODE first=NULL,a,b;
clrscr();
for(;;)
{
printf("1.insert_front\n2.concat\n3.reverse\n4.dislay\n5.exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item\n");
                    scanf("%d",&item);
                    first=insert_rear(first,item);
                    break;
  case 2:printf("enter the no of nodes in 1\n");
                    scanf("%d",&n);
                    a=NULL;
                    for(i=0;i<n;i++)
                     {
                      printf("enter the item\n");
                      scanf("%d",&item);
                      a=insert_rear(a,item);
                     }
```



```
                     printf("enter the no of nodes in 2\n");
                    scanf("%d",&n);
                    b=NULL;
                    for(i=0;i<n;i++)
                     {
                      printf("enter the item\n");
                      scanf("%d",&item);
                      b=insert_rear(b,item);
                     }
                     a=concat(a,b);
                     display(a);
                    break;
case 3:first=reverse(first);
                    display(first);
                    break;
case 4:display(first);
                    break;
default:exit(0);
}
}
getch();
}
```

Program : Implementation of Queue for Strings using C

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define QUE_SIZE 3
int front=0,rear=-1,c=0;
char item[30];
struct
{
   char s[30];
}q[10];
void insertrear()
{
   if(rear == QUE_SIZE - 1)
   {
      printf("-------------------\n");
      printf("Queue OVERFLOW!!\n");
      printf("-------------------\n");
      return;
   }
   rear++;

   strcpy(q[rear].s,item);
}
int deletefront()
{
   if(front>rear)
   {
      front = 0;
      rear = -1;
      return -1;
   }
   strcpy(item,q[front++].s);
}
void displayQ()
{
   if(front>rear)
   {
      printf("-------------------\n");
      printf("Queue is empty\n");
      printf("-------------------\n");
      return;
   }
   printf("Contents of Queue\n");
   for(int i = front;i<=rear;i++)
   {
      puts(q[i].s);
   }
}
void main()
{
   int choice;
   for(;;)
   {
      printf("Enter \n1.for insertion\n2.for deletion\n3.for display\n4.exit\n");
      scanf("%d",&choice);
      switch(choice)
```

```
        {
            case 1: printf("Enter the item to be inserted\n");
                    scanf("%s",&item);
                    insertrear();
                    break;
            case 2: c = deletefront();
                    if(c == -1)
                    {
                       printf("-------------------\n");
                       printf("Queue is empty\n");
                       printf("-------------------\n");
                    }
                    else
                       printf("Item deleted = %s\n",item);
                    break;
            case 3: displayQ();
                    break;
            default: exit(0);
        }
    }
}
```

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
 {
  int info;
  struct node *rlink;
  struct node *llink;
 };
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_rear(NODE head,int item)
{
NODE temp,cur;
temp=getnode();
temp->rlink=NULL;
temp->llink=NULL;
temp->info=item;
cur=head->llink;
temp->llink=cur;
cur->rlink=temp;
head->llink=temp;
temp->rlink=head;
head->info=head->info+1;
return head;
}
NODE insert_leftpos(int item,NODE head)
{
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("list empty\n");
return head;
}
cur=head->rlink;
while(cur!=head)
{
if(item==cur->info)break;
cur=cur->rlink;
}
if(cur==head)
{
 printf("key not found\n");
 return head;
 }
 prev=cur->llink;
 printf("enter towards left of %d=",item);
 temp=getnode();
 scanf("%d",&temp->info);
 prev->rlink=temp;
 temp->llink=prev;
 cur->llink=temp;
 temp->rlink=cur;
 return head;
}
void display(NODE head)
{
NODE temp;
if(head->rlink==head)
{
printf("list empty\n");
return;
}
for(temp=head->rlink;temp!=head;temp=temp->rlink)
printf("%d\n",temp->info);
}
void main()
{
int item,choice,key;
NODE head;
head=getnode();
head->rlink=head;
head->llink=head;
clrscr();
for(;;)
{
printf("\n1.insert_rear\n2.insert_key\n3.display\n4.exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item\n");
                scanf("%d",&item);
                head=insert_rear(head,item);
                break;
  case 2:printf("enter the key item\n");
                scanf("%d",&item);
                head=insert_leftpos(item,head);
                break;
  case 3:display(head);
                break;
  default:exit(0);
                 break;
  }
 }
}
```

WAP to delete all nodes whose information field is specified as Key (to remove duplicates)

structure of node
getnode()
free()
insert-rear or insert-front()
display()

```
NODE delete_all_key(int item,NODE head)
{
NODE prev,cur,next;
int count;
   if(head->rlink==head)
    {
     pf("LE");
     return head;
     }
count=0;-----assume key initially not present
cur=head->rlink;-----find the address of first node
while(cur!=head)
{
  if(item!=cur->info)
  cur=cur->rlink;
  else
 {
  count++---- udate the number of occurances
  prev=cur->llink;
  next=cur->rlink;
  prev-->rlink=next;
  next->llink=prev;
  freenode(cur);
  cur=next; ----search continues again
 }
}
if(count==0)
  pf(key not found);
  else
 pf(key found at %d positions and are deleted\n, count);

return head;
}
```

IF,IR,DF,DR,D
simple search
insert left,right of the key node specified
remove duplicates

PROGRAM: FIND THE LARGEST AND SMALLEST ELEMENT IN THE BINARY SEARCH TREE

Code:

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

struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};
typedef struct node *NODE;

NODE getnode()
{
    NODE x;
    x = (NODE)malloc(sizeof(struct node));
    if(x == NULL)
    {
        printf("Mmeory full\n");
        exit(0);
    }
    return x;
}

void freenode(NODE x)
{
    free(x);
}

NODE insert(NODE root,int item)
{
    NODE temp,cur,prev;
    temp = getnode();
    temp->info = item;
    temp->llink = NULL;
    temp->rlink = NULL;
    if(root == NULL)
        return temp;
    cur = root;
    while(cur!=NULL)
    {
        prev = cur;
        cur = (item<cur->info)?cur->llink:cur->rlink;
    }
    if(item<prev->info)
        prev->llink = temp;
    else
        prev->rlink = temp;
    return root;
}

void display(NODE root,int i)
{
    int j;
    if(root!=NULL)
    {
        display(root->rlink,i+1);
        for(j=0;j<i;j++)
                printf("      ");
        printf("%d\n",root->info);
                display(root->llink,i+1);
    }
}


NODE delete(NODE root,int item)
{
    NODE cur,parent,q,suc;
    if(root==NULL)
    {
        printf("================\nTREE EMPTY\n============\n");
        return root;
    }
    parent=NULL;
    cur=root;
    while(cur!=NULL&&item!=cur->info)
    {
        parent=cur;
        cur=(item<cur->info)?cur->llink:cur->rlink;
    }
    if(cur==NULL)
    {
        printf("not found\n");
        return root;
    }
    if(cur->llink==NULL)
        q=cur->rlink;
    else if(cur->rlink==NULL)
        q=cur->llink;
    else
    {
        suc=cur->rlink;
        while(suc->llink!=NULL)
            suc=suc->llink;
        suc->llink=cur->llink;
        q=cur->rlink;
    }
    if(parent==NULL)
        return q;
    if(cur==parent->llink)
        parent->llink=q;
    else
        parent->rlink=q;
    freenode(cur);
    return root;
 }

void preorder(NODE root)
{
    if(root!=NULL)
    {
        printf("%d\n",root->info);
        preorder(root->llink);
        preorder(root->rlink);
    }

 }
void postorder(NODE root)
{
```

```
    if(root!=NULL)
    {
        postorder(root->llink);
        postorder(root->rlink);
        printf("%d\n",root->info);
    }
}
void inorder(NODE root)
{
    if(root!=NULL)
    {
        inorder(root->llink);
        printf("%d\n",root->info);
        inorder(root->rlink);
    }
}

 void findlargest(NODE root)
 {
     NODE cur;
     cur = root;
    if(root==NULL)
    {
        printf("================\nTREE EMPTY\n=============\n");
        return ;
    }
    while(cur->rlink!=NULL)
    {
        cur = cur->rlink;
    }
    printf("=====The largest element in this BST is %d========== ",cur->info);
 }
 void findsmallest(NODE root)
 {
     NODE cur;
     cur = root;
    if(root==NULL)
    {
        printf("================\nTREE EMPTY\n=============\n");
        return ;
    }
    while(cur->llink!=NULL)
    {
        cur = cur->llink;
    }
    printf("=====The smallest element in this BST is %d========== ",cur->info);
 }
int main(void)
{
    int item,choice;
    NODE root=NULL;
    printf("================\nBINARY SEARCH TREE\n=============\n");
    for(;;)
    {
        printf("==================\n");
        printf("\n1.INSERT \n2.DISPLAY
TREE\n3.PREORDER\n4.POSTORDER\n5.INORDER\n6.DELETE\n7.FIND LARGEST
ELEMENT\n8.FIND SMALLEST ELEMENT\n9.EXIT\n");
        printf("ENTER YOUR CHOICE\n");
        scanf("%d",&choice);
        printf("==================\n");
        switch(choice)
        {
            case 1:printf("enter the item\n");
                        scanf("%d",&item);
                        root=insert(root,item);
                        break;
            case 2:display(root,0);
                        break;
            case 3:preorder(root);
                        break;
            case 4:postorder(root);
                        break;
            case 5:inorder(root);
                        break;
            case 6:printf("enter the item\n");
                        scanf("%d",&item);
                        root=delete(root,item);
                        break;
                    case 7:findlargest(root);
                            break;
                    case 8:findsmallest(root);
                            break;
            default:exit(0);

            }
        }
}
```

C-Program to implement STACK using arrays(Using Global Variables)

(1)

```
#include<stdio.h>
#include<process.h>
#include<conio.h>
#define STACK_SIZE 5
int top=-1;
int s[10];
int item;
void push()
{
if(top==STACK_SIZE-1)
{
printf("stack overflow\n");
return;
}
top=top+1;
s[top]=item;
}
int pop()
{
if(top==-1) return -1;
return s[top--];
}
void display()
{
int i;
if(top==-1)
{
printf("stack is empty\n");
return;
}
printf("contents of the stack\n");
for(i=top; i>=0; i--)
{
printf("%d\n",s[i]);
}
}
void main()
{
int item_deleted;
int choice;
clrscr();
for(;;)
{
printf("\n 1:push\n 2:pop\n 3:display\n 4:exit\n");
printf("enter the choice\n");
scanf("%d", &choice);
switch(choice)
{
```



```
case 1:printf("enter the item to be inserted\n");
       scanf("%d",&item);
       push();
       break;
case 2:item_deleted=pop(); (-1)
        if(item_deleted==-1)
       printf("stack is empty\n");
       else
       printf("item deleted is %d\n",item_deleted);
       break;
case 3:display();
       break;
default:exit(0);
}
}
getch();
}
```

C-Program to implement STACK using arrays(By Passing Parameters)

(2)

```
#include<stdio.h>
#include<process.h>
#include<conio.h>
#define STACK_SIZE 5
int top=-1;

void push(int item,int s[],int *top)
{
if(*top==STACK_SIZE-1)
{
printf("stack overflow\n");
return;
}
*top=*top+1;
s[*top]=item;
}
int pop(int s[],int *top)
{
int item_deleted;
if(*top==-1)
{
printf("stack underflow cannot delete\n");
return 0;
}
item_deleted=s[*top];
*top=*top-1;
return item_deleted;
}
void display(int top,int s[])
```

```
{
int i;
if(top==-1)
{
printf("stack is empty\n");
return;
}
printf("contents of the stack\n");
for(i=0;i<=top;i++)
{
printf("%d\n",s[i]);
}
}
void main()
{
int item,s[10];
int item_deleted;
int choice;
clrscr();
for(;;)
{
printf("\n 1:push\n 2:pop\n 3:display\n 4:exit\n");
printf("enter the choice\n");
scanf("%d", &choice);
switch(choice)
{
case 1:printf("enter the item to be inserted\n");
      scanf("%d" ,&item);
      push(item, s,&top);       (12,s[0], 2000)
      break;
case 2:item_deleted=pop(s, &top);
      if(item_deleted!=0)
      printf("\item deleted is %d\n",item_deleted);
      break;
case 3:display(top, s);
      break;
default:exit(0);
}
}
getch();
}
```

(3)

```
#include<stdio.h>
#include<process.h>
#include<conio.h>
#define STACK_SIZE 5
int top=-1;
void push(int item,int s[],int *top);
int pop(int s[],int *top);
```



```
void display(int top,int s[]);
void main()
{
int item,s[10];
int item_deleted;
int choice;
clrscr();
for(;;)
{
printf("\n 1:push\n 2:pop\n 3:display\n 4:exit\n");
printf("enter the choice\n");
scanf("%d", &choice);
switch(choice)
{
case 1:printf("enter the item to be inserted\n");
      scanf("%d",&item);
      push(item,s,&top);
      break;
case 2:item_deleted=pop(s,&top);
      if(item_deleted!=0)
      printf("\item deleted is %d\n",item_deleted);
      break;
case 3:display(top,s);
      break;
default:exit(0);
}
}
getch();
}
void push(int item,int s[],int *top)
{
if(*top==STACK_SIZE-1)
{
printf("stack overflow\n");
return;
}
*top=*top+1;
s[*top]=item;
}
int pop(int s[],int *top)
{
int item_deleted;
if(*top==-1)
{
printf("stack underflow cannot delete\n");
return 0;
}
item_deleted=s[*top];
*top=*top-1;
return item_deleted;
}
```

```
void display(int top,int s[])
{
int i;
if(top==-1)
{
printf("stack is empty\n");
return;
}
printf("contents of the stack\n");
for(i=0;i<=top;i++)
{
printf("%d\n",s[i]);
}
}
```

C-Program to implement LINEAR QUEUE(Ordinary Queue)

(1)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 5
int item,front=0,rear=-1,q[10];
void insertrear()
{
if(rear==QUE_SIZE-1)
{
printf("queue overflow\n");
return;
}
rear=rear+1;
q[rear]=item;
}
int deletefront()
{
if(front>rear) return -1;
return q[front++];
}
void displayQ()
{
int i;
if(front>rear)
{
printf("queue is empty\n");
return;
}
printf("Contents of queue \n");
for(i=front;i<=rear;i++)
{
printf("%d\n",q[i]);
}
```



```
}
void main()
{
 int choice;
 clrscr();
 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter the choice\n");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:printf("enter the item to be inserted\n");
          scanf("%d",&item);
          insertrear();
          break;
 case 2:item=deletefront();
          if(item==-1)
          printf("queue is empty\n");
          else
          printf("item deleted =%d\n",item);
          break;
 case 3:displayQ();
          break;
 default:exit(0);
}
}
getch();
}
```

C-Program to implement LINEAR QUEUE(Ordinary Queue) (By Passing Parameters)

(2)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 5
int item,front=0,rear=-1,q[10];
void insertrear(int item,int *rear,int q[])
{
if(*rear==QUE_SIZE-1)
{
printf("queue overflow\n");
return;
}
*rear=*rear+1;
q[*rear]=item;
}
int deletefront(int *front,int *rear,int q[])
{
if(*front>*rear) return -1;
```

```
return q[*front++];
}
void displayQ(int front,int rear,int q[])
{
int i;
if(front>rear)
{
printf("queue is empty\n");
return;
}
printf("Contents of queue \n");
for(i=front;i<=rear;i++)
{
printf("%d\n",q[i]);
}
}
void main()
{
 int choice;
 clrscr();
 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter the choice\n");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:printf("enter the item to be inserted\n");
          scanf("%d",&item);
          insertrear(item,&rear,q);
          break;
 case 2:item=deletefront(&front,&rear,q);
          if(item==-1)
          printf("queue is empty\n");
          else
          printf("item deleted =%d\n",item);
          break;
 case 3:displayQ(front,rear,q);
          break;
 default:exit(0);
}
}
getch();
}
```

(3)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 5
int item,front=0,rear=-1,q[10];
```



```
void insertrear(int,int *,int []);
int deletefront(int *,int *,int []);
void displayQ(int ,int ,int []);
void main()
{
 int choice;
 clrscr();
 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter the choice\n");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:printf("enter the item to be inserted\n");
          scanf("%d",&item);
          insertrear(item,&rear,q);
          break;
 case 2:item=deletefront(&front,&rear,q);
          if(item==-1)
          printf("queue is empty\n");
          else
          printf("item deleted =%d\n",item);
          break;
 case 3:displayQ(front,rear,q);
          break;
 default:exit(0);
}
}
getch();
}
void insertrear(int item,int *rear,int q[])
{
if(*rear==QUE_SIZE-1)
{
printf("queue overflow\n");
return;
}
*rear=*rear+1;
q[*rear]=item;
}
int deletefront(int *front,int *rear,int q[])
{
if(*front>*rear) return -1;
return q[(*front)++];
}
void displayQ(int front,int rear,int q[])
{
int i;
if(front>rear)
{
```

```
printf("queue is empty\n");
return;
}
printf("Contents of queue \n");
for(i=front;i<=rear;i++)
{
printf("%d\n",q[i]);
}
}
```

(4)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 5
int item,front=0,rear=-1,q[10];
int IsFull(int rear)
{
return rear==QUE_SIZE-1;
}
int IsEmpty(int front,int rear)
{
return front>rear;
}
void insertrear(int item,int *rear,int q[])
{
if(IsFull(*rear))
{
printf("queue overflow\n");
return;
}
*rear=*rear+1;
q[*rear]=item;
}
int deletefront(int *front,int *rear,int q[])
{
if(IsEmpty(*front,*rear)) return -1;
return q[(*front)++];
}
void displayQ(int front,int rear,int q[])
{
int i;
if(front>rear)
{
printf("queue is empty\n");
return;
}
printf("Contents of queue \n");
for(i=front;i<=rear;i++)
{
printf("%d\n",q[i]);
}
```



```
}
void main()
{
 int choice;
 clrscr();
 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter the choice\n");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:printf("enter the item to be inserted\n");
          scanf("%d",&item);
          insertrear(item,&rear,q);
          break;
 case 2:item=deletefront(&front,&rear,q);
          if(item==-1)
          printf("queue is empty\n");
          else
          printf("item deleted =%d\n",item);
          break;
 case 3:displayQ(front,rear,q);
          break;
 default:exit(0);
}
}
getch();
}
```

C-Program to implement CIRCULAR QUEUE(By using global variables)

(1)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 3
int item,front=0,rear=-1,q[QUE_SIZE],count=0;
void insertrear()
{
if(count==QUE_SIZE)
{
printf("queue overflow\n");
return;
}
rear=(rear+1)%QUE_SIZE;
q[rear]=item;
count++;
}
int deletefront()
```

```
{
if(count==0) return -1;
item=q[front];
front=(front+1)%QUE_SIZE;
count=count-1;
return item;
}
void displayQ()
{
int i,f;
if(count==0)
{
printf("queue is empty\n");
return;
}
f=front;
printf("Contents of queue \n");
for(i=1;i<=count;i++)
{
printf("%d\n",q[f]);
f=(f+1)%QUE_SIZE;
}
}
void main()
{
 int choice;

 clrscr();
 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter the choice\n");
 scanf("%d",&choice);

 switch(choice)
 {
 case 1:printf("enter the item to be inserted\n");
          scanf("%d",&item);
          insertrear();
          break;
 case 2:item=deletefront();
          if(item==-1)
          printf("queue is empty\n");
          else
          printf("item deleted =%d\n",item);
          break;
 case 3:displayQ();
          break;
 default:exit(0);
}
}
```



```
getch();
}
```

C-Program to implement CIRCULAR QUEUE(By passing parameters)

(2)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 5
void insertrear(int item,int *rear,int *q,int *count)
{
if(*count==QUE_SIZE)
{
printf("queue overflow\n");
return;
}
*rear=(*rear+1)%QUE_SIZE;
q[*rear]=item;
(*count)++;
}
void deletefront(int *front,int *q,int *count)
{
if(*count==0)
{
printf("queue empty\n");
return;
}
printf("item deleted is %d\n",q[*front]);
*front=(*front+1)%QUE_SIZE;
(*count)--;
}
void displayQ(int front,int q[],int count)
{
int i,j;
if(count==0)
{
printf("queue is empty\n");
return;
}
i=front;
printf("Contents of queue \n");
for(j=1;j<=count;j++)
{
printf("%d\n",q[i]);
i=(i+1)%QUE_SIZE;
}
}
void main()
{
 int choice;
```

```
int item,front=0,rear=-1,q[10],count=0;
clrscr();
for(;;)
{
printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("enter the item to be inserted\n");
        scanf("%d",&item);
        insertrear(item,&rear,q,&count);
        break;
case 2:deletefront(&front,q,&count);
        break;

case 3:displayQ(front,q,count);
        break;
default:exit(0);
}
}
getch();
}
```

(3)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define qsize 5
int isfull(int count)
  {
  return(count==qsize)?1:0;
  }
int isempty(int count)
  {
  return(count==0)?1:0;
  }
void insert_rear(int *count,int *r,int item,int q[])
  {
  if(isfull(*count))
          {
           printf("queue full\n");
           return;
          }
  *r=(*r+1)%qsize;
  q[*r]=item;
  (*count)++;
  }
void delete_front(int *count,int *f,int q[])
  {
```



```
  if(isempty(*count))
          {
           printf("queue empty\n");
           return;
          }
  printf("item deleted is %d\n",q[*f]);
  *f=(*f+1)%qsize;
  (*count)--;
  }
void display(int f,int count,int q[])
  {
  int i,j;
  if(isempty(count))
          {
           printf("queue empty\n");
           return;
          }
  i=f;
  for(j=1;j<=count;j++)
         {
          printf("%d\n",q[i]);
          i=(i+1)%qsize;
         }
}
void main()
 {
 int f=0,r=-1,item,q[10],ch,count=0;
 clrscr();
 for(;;)
  {
        printf("1.insert_rear\n2.delete_front\n3.display\n4.exit\n");
        printf("enter choice\n");
        scanf("%d",&ch);
        switch(ch)
          {
          case 1:printf("enter the item\n");
                        scanf("%d",&item);
                        insert_rear(&count,&r,item,q);
                        break;
          case 2:delete_front(&count,&f,q);
                        break;
          case 3:display(f,count,q);
                        break;
          default:exit(0);
          }
         }
  }
```

(4)

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 3
void insertrear(int item,int *rear,int *q,int *count)
{
if(*count==QUE_SIZE)
{
printf("queue overflow\n");
return;
}
*rear=(*rear+1)%QUE_SIZE;
q[*rear]=item;
(*count)++;
}
int deletefront(int *front,int q[],int *count)
{
int item;
if(*count==0)
return -1;
item=q[*front];
*front=(*front+1)%QUE_SIZE;
*count-=1;
return item;
}
void displayQ(int front,int q[],int count)
{
int i;
if(count==0)
{
printf("queue is empty\n");
return;
}
printf("Contents of queue \n");
for(i=1;i<=count;i++)
{
printf("%d\n",q[front]);
front=(front+1)%QUE_SIZE;
}
}
void main()
{
 int choice;
 int item,front=0,rear=-1,q[10],count=0;
 clrscr();
 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter the choice\n");
 scanf("%d",&choice);
 switch(choice)
```



```
 {
 case 1:printf("enter the item to be inserted\n");
        scanf("%d",&item);
        insertrear(item,&rear,q,&count);
        break;
 case 2:item=deletefront(&front,q,&count);
        if(item==-1)
        {
        printf("queue is empty\n");
        break;
        }
        printf("item deleted is %d\n",item);
        break;

 case 3:displayQ(front,q,count);
        break;
 default:exit(0);
}
}
getch();
}
```