Write a program to simulate the working of stack using an array with the following :
a) Push b) Pop c) Display
The program should print appropriate messages for stack overflow, stack underflow

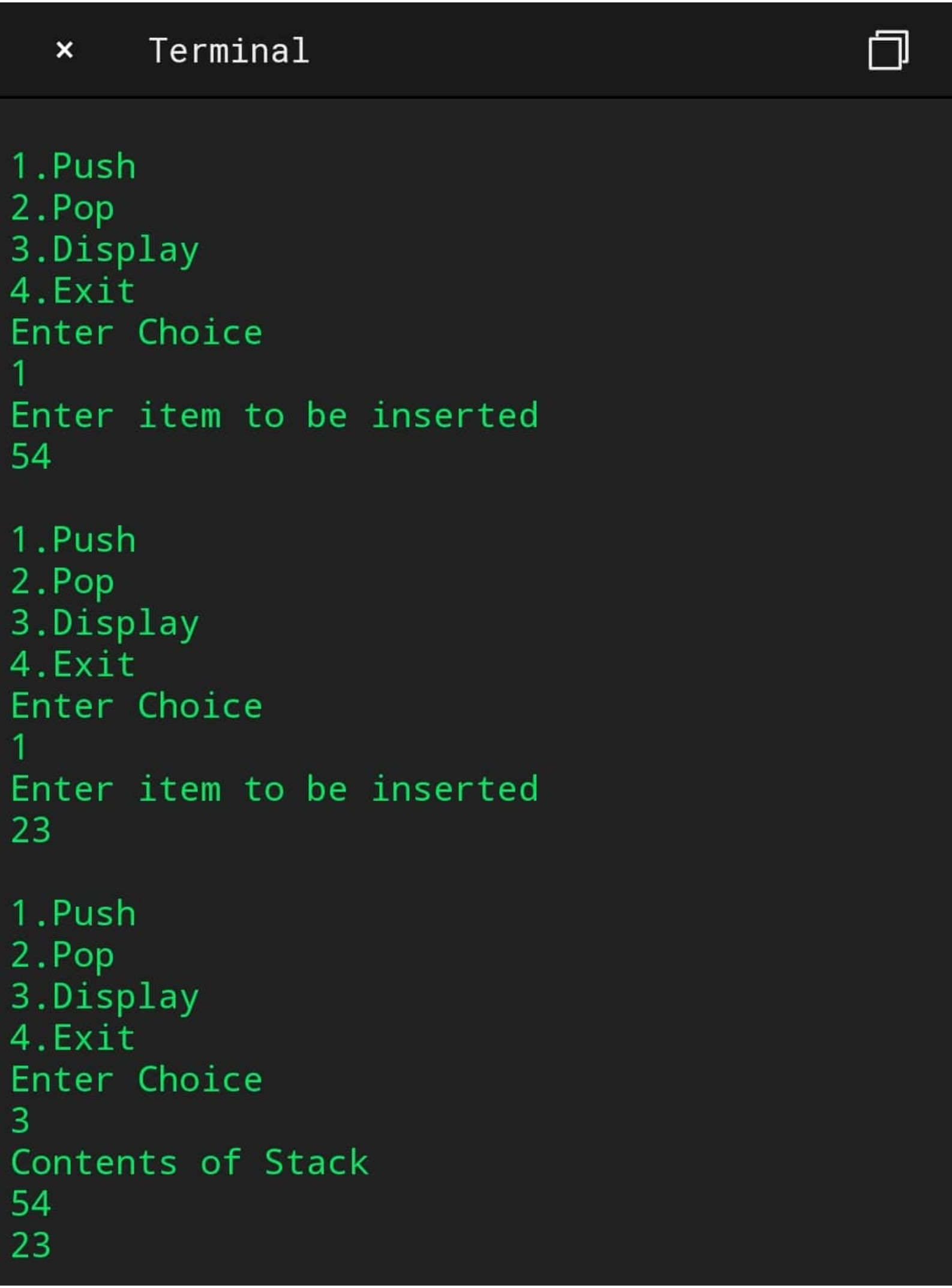
StackUsinArrRef.c

Saved

```c
#include<stdio.h>
#include<conio.h>
#define StackSize 5
int top=-1;
void push(int item,int s[],int *top)
{
  if(*top==StackSize-1)
  {
    printf("Stack Overflow\n");
    return;
  }
  *top=*top+1;
  s[*top]=item;
}
int pop(int s[],int *top)
{
  int ItmDel;
  if(*top==-1)
  {
    printf("Empty Stack. Canmot Delete Items\n");
    return 0;
  }
  ItmDel=s[*top];
  *top=*top-1;
  return ItmDel;
}
void display(int top,int s[])
{
  int i;
  if(top==-1)
  {
    printf("Empty Stack\n");
    return;
  }
  printf("Contents of Stack\n");
  for(i=0;i<=top;i++)
    printf("%d\n",s[i]);
}
void main()
{
  int item,s[10];
  int ItmDel;
  int choice;
  for(;;)
  {
    printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
    printf("Enter Choice\n");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("Enter item to be inserted\n");
        scanf("%d",&item);
        push(item,s,&top);
        break;
      case 2:
        ItmDel=pop(s,&top);
        if(ItmDel!=0)
        printf("\nItem Deleted: %d\n",ItmDel);
        break;
      case 3:display(top,s);
        break;
      default:exit(0);
    }
  }
}
```

Make public

```
× Terminal

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
1
Enter item to be inserted
12

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
1
Enter item to be inserted
14

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
1
Enter item to be inserted
16

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
1
Enter item to be inserted
18

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
1
Enter item to be inserted
20

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
1
Enter item to be inserted
22
Stack Overflow
```



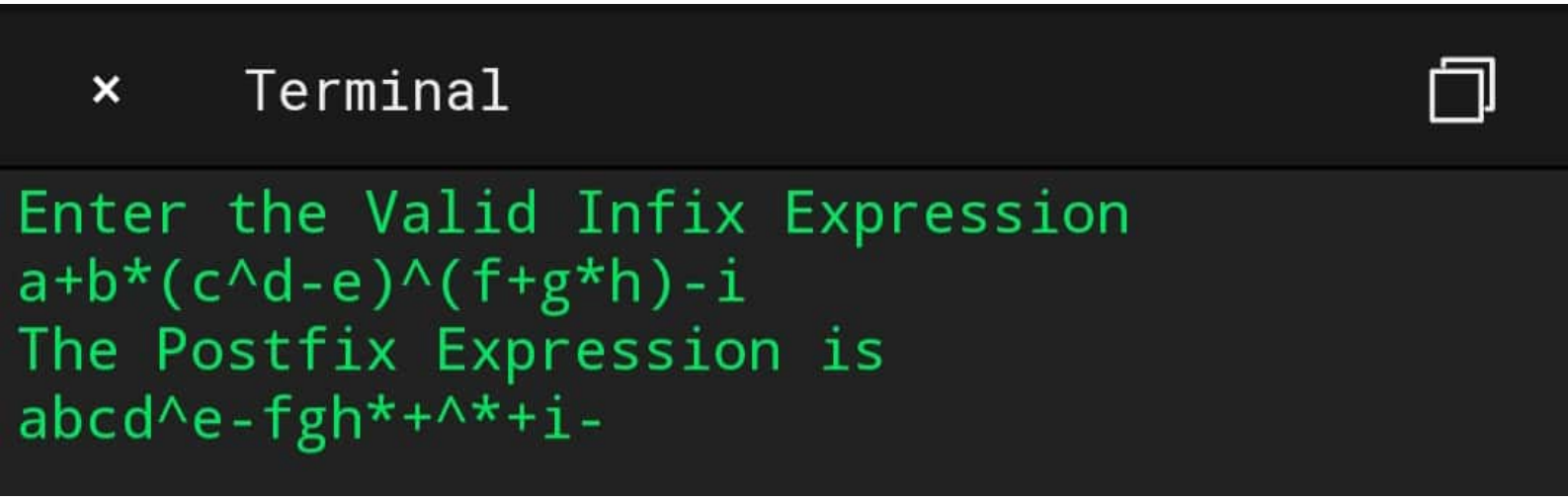
```
Terminal
2.Pop
3.Display
4.Exit
Enter Choice
1
Enter item to be inserted
45

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
2

Item Deleted: 45

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
2

Item Deleted: 64

1.Push
2.Pop
3.Display
4.Exit
Enter Choice
2
Empty Stack. Canmot Delete Items
```

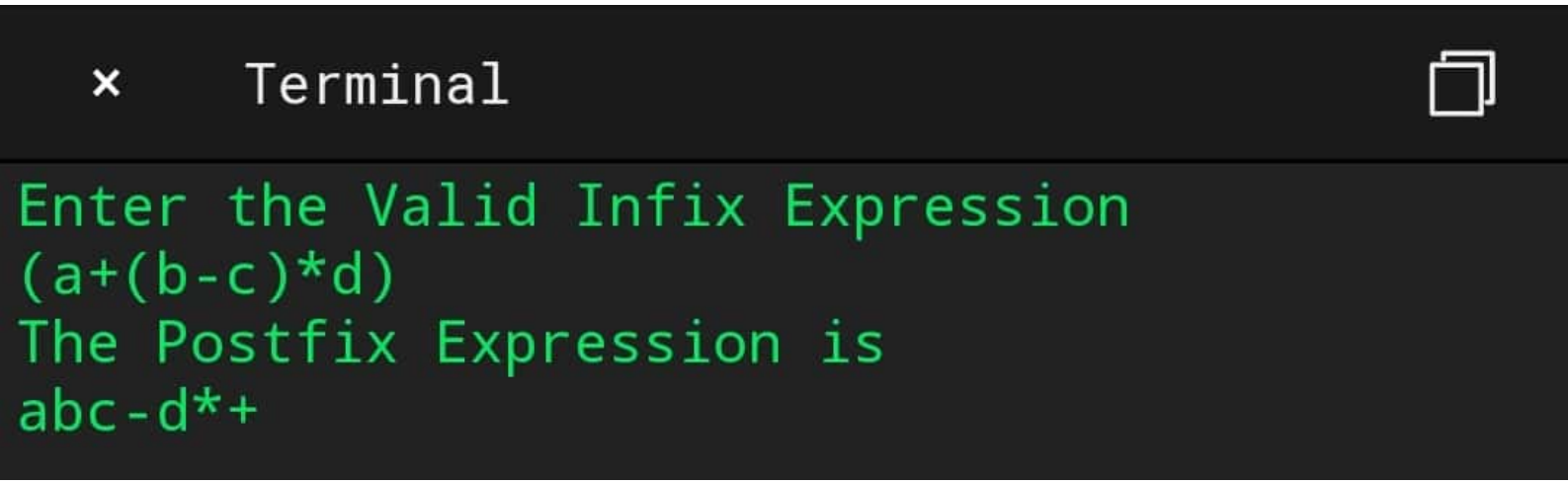
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

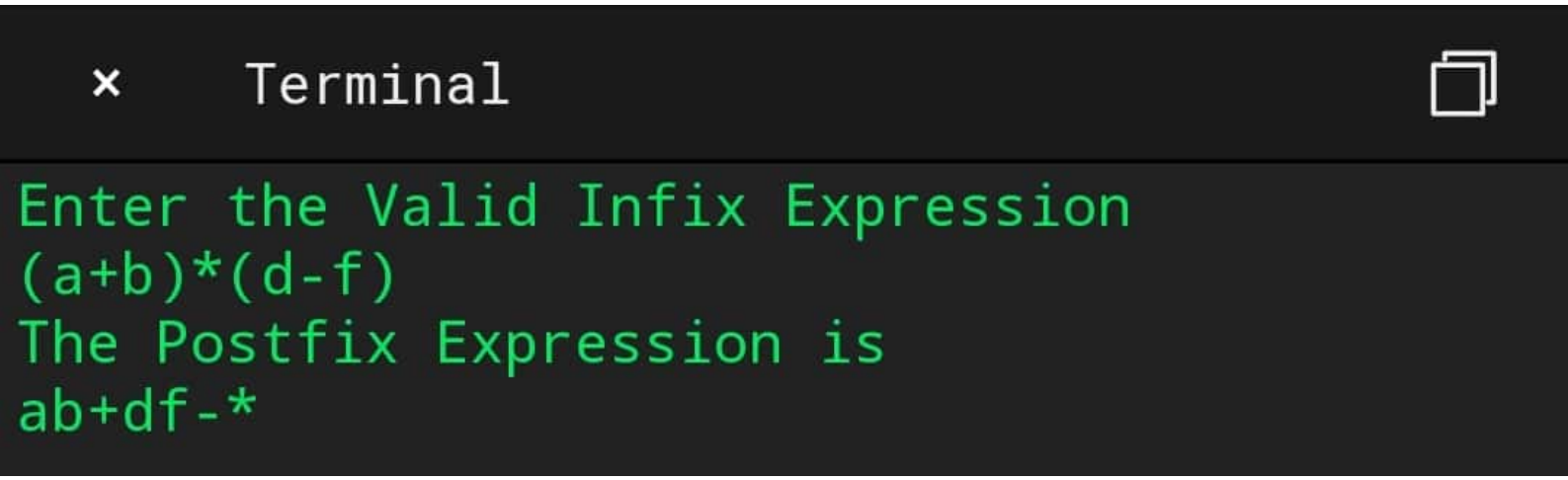
Postfix.c

Saved

```c
#include<stdio.h>
#include<string.h>
int f(char sym)
{
  switch(sym)
  {
    case '+':
    case '-':return 2;
    case '*':
    case '/':return 4;
    case '^':
    case '$':return 5;
    case '(':return 0;
    case '#':return -1;
    default:return 8;
  }
}
int g(char sym)
{
  switch(sym)
  {
    case '+':
    case '-':return 1;
    case '*':
    case '/':return 3;
    case '^':
    case '$':return 6;
    case '(':return 9;
    case ')':return 0;
    default:return 7;
  }
}
void infix_postfix(char infix[],char postfix[])
{
  int top,i,j;
  char s[30],sym;
  top=-1;
  s[++top]='#';
  j=0;
  for(i=0;i<strlen(infix);i++)
  {
    sym=infix[i];
    while(f(s[top])>g(sym))
    {
      postfix[j]=s[top--];
      j++;
    }
    if(f(s[top])!=g(sym))
    s[++top]=sym;
    else
    top--;
  }
  while(s[top]!='#')
  postfix[j++]=s[top--];
  postfix[j]='\0';
}
void main()
{
  char infix[20];
  char postfix[20];
  printf("Enter the Valid Infix Expression\n");
  scanf("%s",infix);
  infix_postfix(infix,postfix);
  printf("The Postfix Expression is ");
  printf("\n%s\n",postfix);
}
```







WAP to simulate the working of a queue of integers using an array. Provide the following operations

a) Insert b) Delete c) Display

The program should print appropriate messages for queue empty and queue overflow conditions

LinearQueue.c

Saved

```c
#include<stdio.h>
#define QUE_SIZE 3
int it,fr=0,rear=-1,q[10];
void insertrear()
{
  if(rear==QUE_SIZE-1)
  {
    printf("Queue Overflow\n");
    return;
  }
  rear+=1;
  q[rear]=it;
}
int deletefront()
{
  if(fr>rear)
  {
    fr=0;
    rear=-1;
    return -1;
  }
  return q[fr++];
}
void display()
{
  if(fr>rear)
  {
    printf("Queue is Empty\n");
    return;
  }
  printf("Contents of Queue:\n");
  for(int i=fr;i<=rear;i++)
    printf("%d\n",q[i]);
}
void main()
{
  int ch;
  for(;;)
  {
    printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
    printf("Enter Choice:");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:
        printf("\nEnter Item to be inserted:");
        scanf("%d",&it);
        insertrear();
        break;
      case 2:
        it=deletefront();
        if(it==-1)
          printf("\nEmpty Queue");
        else
          printf("Item Deleted=%d\n",it);
        break;
      case 3:
        display();
        break;
      default:exit(0);
    }
  }
}
```

```
Terminal

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:2

Empty Queue
1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:1

Enter Item to be inserted:11

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:1

Enter Item to be inserted:22

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:1

Enter Item to be inserted:33

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:1

Enter Item to be inserted:44
Queue Overflow

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:3
Contents of Queue:
11
22
33

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:2
Item Deleted=11

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:2
Item Deleted=22

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:2
Item Deleted=33

1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:2

Empty Queue
1.Insert
2.Delete
3.Display
4.Exit
Enter Choice:4

Process finished.
```

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.

a) Insert b) Delete c) Display

The program should print appropriate messages for queue empty and queue overflow conditions

CircularQ.c

Saved

```c
#include<stdio.h>
#define size 3
int item,front=0,rear=-1,q[size],count=0;
void insertrear()
{
    if(count==size)
    {
        printf("Queue Overflow\n");
        return;
    }
    rear=(rear+1)%size;
    q[rear]=item;
    count++;
}
int deletefront()
{
    if(count==0)
    return -1;
    item=q[front];
    front=(front+1)%size;
    count=count-1;
    return item;
}
void display()
{
    int i,f;
    if(count==0)
    {
        printf("Queue Empty\n");
        return;
    }
    f=front;
    for(i=1;i<=count;i++)
    {
        printf("%d\n",q[f]);
        f=(f+1)%size;
    }
}
void main()
{
    int ch;
    for(;;)
    {
        printf("\n1.Insert Rear\n2.Delete Front\n3.Display\n4.Exit\n");
        printf("Enter Choice:");
        scanf("%d",&ch);
    switch(ch)
    {
    case 1:printf("Enter Item:");
         scanf("%d",&item);
         insertrear();
         break;
    case 2:item=deletefront();
         if(item==-1)
         printf("Queue Empty\n");
         else
         printf("Item Deleted =%d\n",item);
         break;
    case 3:display();
         break;
    default:exit(0);
    }
    }
}
```

```
Terminal

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:1
Enter Item:10

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:1
Enter Item:20

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:1
Enter Item:30

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:1
Enter Item:40
Queue Overflow

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:3
10
20
30

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:2
Item Deleted =10

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:2
Item Deleted =20

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:2
Item Deleted =30

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:2
Queue Empty

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:3
Queue Empty

1.Insert Rear
2.Delete Front
3.Display
4.Exit
Enter Choice:4

Process finished.
```

WAP to Implement Singly Linked List with following operations

a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.

Singly_LL.c

```c
#include<stdio.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("Memory 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 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 = %d\n",first->info);
    free(first);
    return temp;
}
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 at Rear-end = %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 = %d\n",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 = %d\n",cur->info);
prev->link=cur->link;
freenode(cur);
return first;
}
void display(NODE first)
{
NODE temp;
if(first==NULL)
printf("Empty List. 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;
for(;;)
{
printf("\n1.Insert_Front\n2.Insert_Rear\n3.Delete_Front\n4.Delete_Rear\n");
printf("5.Insert_Info_Position\n6.Delete_Info_Position\n7.Display List\n8.Exit\n");
printf("Enter Choice:");
scanf("%d",&choice);
switch(choice)
{
  case 1:
    printf("Enter the Item at Front-end: ");
    scanf("%d",&item);
    first=insert_front(first,item);
    break;
  case 2:
    printf("Enter the item at Rear-end:");
    scanf("%d",&item);
    first=insert_rear(first,item);
    break;
  case 3:
    first=delete_front(first);
    break;
  case 4:
    first=delete_rear(first);
    break;
  case 5:
    printf("Enter Item to be Inserted at Given Position:");
    scanf("%d",&item);
    printf("Enter Position:");
    scanf("%d",&pos);
    first=insert_pos(item,pos,first);
    break;
  case 6:
    printf("Enter Position:");
    scanf("%d",&pos);
    first=delete_pos(pos,first);
    break;
  case 7:
    display(first);
    break;
  default:
    exit(0);
    break;
  }
 }
}
```

```
Terminal

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 20

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
20
10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:5
Enter Item to be Inserted at Given Position:25
Enter Position:2

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:6
Enter Position:4
Item Deleted at Given Position = 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
25
20
```

WAP to Implement Singly Linked List with following operations
a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

Singly_LL.c

```c
#include<stdio.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("Memory 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 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 = %d\n",first->info);
    free(first);
    return temp;
}
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 at Rear-end = %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 = %d\n",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 = %d\n",cur->info);
prev->link=cur->link;
freenode(cur);
return first;
}
void display(NODE first)
{
NODE temp;
if(first==NULL)
printf("Empty List. 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;
for(;;)
{
printf("\n1.Insert_Front\n2.Insert_Rear\n3.Delete_Front\n4.Delete_Rear\n");
printf("5.Insert_Info_Position\n6.Delete_Info_Position\n7.Display List\n8.Exit\n");
printf("Enter Choice:");
scanf("%d",&choice);
switch(choice)
{
  case 1:
    printf("Enter the Item at Front-end: ");
    scanf("%d",&item);
    first=insert_front(first,item);
    break;
  case 2:
    printf("Enter the item at Rear-end:");
    scanf("%d",&item);
    first=insert_rear(first,item);
    break;
  case 3:
    first=delete_front(first);
    break;
  case 4:
    first=delete_rear(first);
    break;
  case 5:
    printf("Enter Item to be Inserted at Given Position:");
    scanf("%d",&item);
    printf("Enter Position:");
    scanf("%d",&pos);
    first=insert_pos(item,pos,first);
    break;
  case 6:
    printf("Enter Position:");
    scanf("%d",&pos);
    first=delete_pos(pos,first);
    break;
  case 7:
    display(first);
    break;
  default:
    exit(0);
    break;
  }
 }
}
```

```
Terminal

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 20

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
20
10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:5
Enter Item to be Inserted at Given Position:25
Enter Position:2

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:6
Enter Position:4
Item Deleted at Given Position = 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
25
20
```

WAP Implement Single Link List with following operations

a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

SortRevCon.c

Saved

```c
#include<stdio.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("Memory 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 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 = %d\n",first->info);
    free(first);
    return temp;
}
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 at Rear-end = %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 = %d\n",cur->info);
    free(cur);
    prev->link=NULL;
    return first;
}
void display(NODE first)
{
    NODE temp;
    if(first==NULL)
        printf("Empty List. Cannot Display Items.\n");
    for(temp=first;temp!=NULL;temp=temp->link)
    {
        printf("%d\n",temp->info);
    }
}
NODE sort(NODE first)
{
    NODE temp1;
    NODE temp2;
    for(temp1=first;temp1!=NULL;temp1=temp1->link)
    {
        for(temp2=temp1->link;temp2!=NULL;temp2=temp2->link)
        {
            first=temp1->info;
            temp1->info=temp2->info;
            temp2->info=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;
}
NODE concat(NODE first,NODE sec)
{
    NODE cur;
    if(first==NULL)
        return sec;
    if(sec==NULL)
        return first;
    cur=first;
    while(cur->link!=NULL)
        cur=cur->link;
    cur->link=sec;
    return first;
}
void main()
{
    int item,choice,key,pos,n;
    int count=0;
    NODE first=NULL;
    for(;;)
    {
        printf("\n1.Insert_Front\n2.Insert_Rear\n3.Delete_Front\n4.Delete_Rear\n");
        printf("5.Display List\n6.Sort List\n7.Reverse\n8.Concatenate Two Lists\n9.Exit\n");
        printf("Enter Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            printf("Enter the Item at Front-end: ");
            scanf("%d",&item);
            first=insert_front(first,item);
            break;
        case 2:
            printf("Enter the item at Rear-end:");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
        case 3:
            first=delete_front(first);
            break;
        case 4:
            first=delete_rear(first);
            break;
        case 5:
            display(first);
            break;
        case 6:
            sort(first);
            display(first);
            break;
        case 7:
            first=reverse(first);
            display(first);
            break;
        case 8:
            printf("Enter No. of Nodes in List 1 :");
            scanf("%d",&n);
            NODE a=NULL;
            for(int i=0;i<n;i++)
            {
                printf("Enter Item:");
                scanf("%d",&item);
                a=insert_rear(a,item);
            }
            printf("Enter No. of Nodes in List 2 :");
            scanf("%d",&n);
            NODE b=NULL;
            for(int i=0;i<n;i++)
            {
                printf("Enter Item:");
                scanf("%d",&item);
                b=insert_rear(b,item);
            }
            a=concat(a,b);
            display(a);
            break;
        default:
            exit(0);
            break;
        }
    }
}
```

```
Terminal

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 31

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 32

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:5
32
31
30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:6
30
31
32

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:7
32
31
30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:8
Enter No. of Nodes in List 1 :2
Enter Item:50
Enter Item:60
Enter No. of Nodes in List 2 :3
Enter Item:70
Enter Item:80
Enter Item:90
50
60
70
80
90
```

WAP to implement Stack & Queues using Linked Representation

Stack_Implementation.c

Saved

```c
#include<stdio.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\n",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;
for(;;)
{
printf("\n1.Insert Rear\n2.Delete Rear\n3.Display List\n4.Exit\n");
printf("Enter Choice:");
scanf("%d",&choice);
switch(choice)
 {

  case 1:printf("Enter Item at Rear-end:");
   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;
 }
}
}
```

```
× Terminal

1.Insert Rear
2.Delete Rear
3.Display List
4.Exit
Enter Choice:1
Enter Item at Rear-end:20

1.Insert Rear
2.Delete Rear
3.Display List
4.Exit
Enter Choice:1
Enter Item at Rear-end:30

1.Insert Rear
2.Delete Rear
3.Display List
4.Exit
Enter Choice:3
10
20
30

1.Insert Rear
2.Delete Rear
3.Display List
4.Exit
Enter Choice:2
item deleted at rear-end is 30

1.Insert Rear
2.Delete Rear
3.Display List
4.Exit
Enter Choice:3
10
20

1.Insert Rear
2.Delete Rear
3.Display List
4.Exit
Enter Choice:4

Process finished.
```

Queue_Implementation.c

Saved

```c
#include<stdio.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 =%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;
    for(;;)
    {
        printf("\n1.Insert Rear\n2.Delete Front\n3.Display List\n4.Exit\n");
        printf("Enter Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("Enter Item at Rear-end:");
              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;
        }
    }
}
```

```
× Terminal

1.Insert Rear
2.Delete Front
3.Display List
4.Exit
Enter Choice:1
Enter Item at Rear-end:10

1.Insert Rear
2.Delete Front
3.Display List
4.Exit
Enter Choice:1
Enter Item at Rear-end:30

1.Insert Rear
2.Delete Front
3.Display List
4.Exit
Enter Choice:1
Enter Item at Rear-end:50

1.Insert Rear
2.Delete Front
3.Display List
4.Exit
Enter Choice:3
10
30
50

1.Insert Rear
2.Delete Front
3.Display List
4.Exit
Enter Choice:2
Item Deleted at Front-end =10

1.Insert Rear
2.Delete Front
3.Display List
4.Exit
Enter Choice:3
30
50

1.Insert Rear
2.Delete Front
3.Display List
4.Exit
Enter Choice:4

Process finished.
```

WAP Implement doubly link list with primitive operations

a) Create a doubly linked list. b) Insert a new node to the left of the node.

c) Delete the node based on a specific value. c) Display the contents of the list

Doubly_LL.c

Saved

```c
#include<stdio.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 Full!!\n");
     exit(0);
     }
   return x;
  }
void freenode(NODE x)
{
  free(x);
}
NODE dinsert_front(int item,NODE head)
{
    NODE temp,cur;
    temp=getnode();
    temp->info=item;
    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;
    cur=head->llink;
    head->llink=temp;
    temp->rlink=head;
    temp->llink=cur;
    cur->rlink=temp;
    return head;
}
NODE ddelete_front(NODE head)
{
    NODE cur,next;
    if(head->rlink==head)
    {
        printf("dq empty\n");
        return head;
    }
    cur=head->rlink;
    next=cur->rlink;
    head->rlink=next;
    next->llink=head;
    printf("the node deleted is %d",cur->info);
    freenode(cur);
    return head;
}
NODE ddelete_rear(NODE head)
{
    NODE cur,prev;
    if(head->rlink==head)
    {
        printf("dq empty\n");
        return head;
    }
    cur=head->llink;
    prev=cur->llink;
    head->llink=prev;
    prev->rlink=head;
    printf("the node deleted is %d",cur->info);
    freenode(cur);
    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 item 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;
}
NODE insert_rightpos(int item,NODE head)
{
NODE temp,cur,prev;
if(head->llink==head)
{
printf("List Empty\n");
return head;
}
cur=head->llink;
while(cur!=head)
{
if(item==cur->info)break;
cur=cur->llink;
}
if(cur==head)
{
 printf("Key Not Found\n");
 return head;
}
prev=cur->rlink;
printf("Enter item towards Right of %d :",item);
temp=getnode();
scanf("%d",&temp->info);
prev->llink=temp;
temp->rlink=prev;
cur->rlink=temp;
temp->llink=cur;
return head;
}
NODE delete_all_key(int item,NODE head)
{
NODE prev,cur,next;
int count;
    if(head->rlink==head)
    {
     printf("List Empty.");
     return head;
    }
count=0;
cur=head->rlink;
while(cur!=head)
{
  if(item!=cur->info)
  cur=cur->rlink;
  else
 {
   count++;
   prev=cur->llink;
   next=cur->rlink;
   prev->rlink=next;
   next->llink=prev;
   freenode(cur);
   cur=next;
 }
}
if(count==0)
    printf("Key not Found");
else
    printf("Key found in %d Position(s) and are Deleted\n", count);
return head;
}
NODE search(int item,NODE head)
{
NODE prev,cur,next;
int count;
    if(head->rlink==head)
    {
     printf("List Empty.");
     return head;
    }
count=0;
cur=head->rlink;
while(cur!=head)
{
  if(item!=cur->info)
  {
    printf ("Item not Found\n");
    break;
  }
  else
  {
   printf ("Search Successful\n");
   break;
  }
}
}
void display(NODE head)
{
    NODE temp;
    if(head->rlink==head)
    {
        printf("List Empty. Cannot Display Items.\n");
        return;
    }
    printf("Contents of List.\n");
    temp=head->rlink;
    while(temp!=head)
    {
        printf("%d ",temp->info);
        temp=temp->rlink;
    }
    printf("\n");
}
void main()
{
    NODE head,last;
    int item, choice;
    head=getnode();
    head->rlink=head;
    head->llink=head;
    for(;;)
    {
      printf("\n1.Insert Front\n2.Insert Rear\n3.Delete Front\n4.Delete Rear\n5.Insert Left of Node");
      printf("\n6.Insert Right of Node\n7.Delete Duplicates\n8.Search\n9.Display\n10.Exit\n");
      printf("Enter choice :");
      scanf("%d",&choice);
      switch(choice)
      {
          case 1:
              printf("Enter the Item at Front end :");
              scanf("%d",&item);
              last=dinsert_front(item,head);
              break;
          case 2:
              printf("Enter the Item at Rear end :");
              scanf("%d",&item);
              last=dinsert_rear(item,head);
              break;
          case 3:
              last=ddelete_front(head);
              break;
          case 4:
              last=ddelete_rear(head);
              break;
          case 5:
              printf("Enter the Key Item :");
          scanf("%d",&item);
          head=insert_leftpos(item,head);
          break;
          case 6:
              printf("Enter the Key Item :");
          scanf("%d",&item);
          head=insert_rightpos(item,head);
          break;
          case 7:
              printf("Enter the Key Item to be Deleted :");
          scanf("%d",&item);
          delete_all_key(item,head);
          break;
          case 8:
            printf("Enter Item to be Searched :");
          scanf("%d",&item);
          search(item,head);
          break;
          case 9:
              display(head);
              break;
          default:exit(0);
      }
    }
}
```

```
Terminal

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :2
Enter the Item at Rear end :10

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :2
Enter the Item at Rear end :20

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :2
Enter the Item at Rear end :30

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :5
Enter the Key Item :20
Enter Item towards Left of 20 :19

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :6
Enter the Key Item :20
Enter Item towards Right of 20 :21

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :9
Contents of List,
10 19 20 21 30

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :1
Enter the Item at Front end :19

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :2
Enter the Item at Rear end :19

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :9
Contents of List,
19 10 19 20 21 30 19

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :7
Enter the Key Item to be Deleted:19
Key found in 3 Position(s) and are Deleted

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :9
Contents of List,
10 20 21 30

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :8
Enter Item to be Searched :10
Search Successful

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert Left of Node
6.Insert Right of Node
7.Delete Duplicates
8.Search
9.Display
10.Exit
Enter Choice :8
Enter Item to be Searched :11
Item not Found
```

Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order

c) To display the elements in the tree.

BinarySearchTree.c

Saved

```c
#include<stdio.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;
}
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);
    }
}
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;
    for(;;)
    {
    printf("\n1.Insert\n2.Display\n3.Pre-Order\n4.In-Order\n5.Post-Order\n6.Exit\n");
    printf("Enter Choice:");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
            printf("Enter Item:");
        scanf("%d",&item);
        root=insert(root,item);
        break;
        case 2:
            display(root,0);
        break;
        case 3:
            preorder(root);
        break;
        case 4:
            inorder(root);
        break;
        case 5:
            postorder(root);
        break;
        default:
            exit(0);
        break;
   }
    }
}
```

```
Terminal

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:1
Enter Item:100

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:1
Enter Item:20

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:1
Enter Item:10

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:1
Enter Item:30

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:1
Enter Item:200

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:1
Enter Item:150

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:1
Enter Item:300

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:3
100
20
10
30
200
150
300

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:4
10
20
30
100
150
200
300

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:5
10
30
20
150
300
200
100

1.Insert
2.Display
3.Pre-Order
4.In-Order
5.Post-Order
6.Exit
Enter Choice:6

Process finished.
```