Prog 1: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

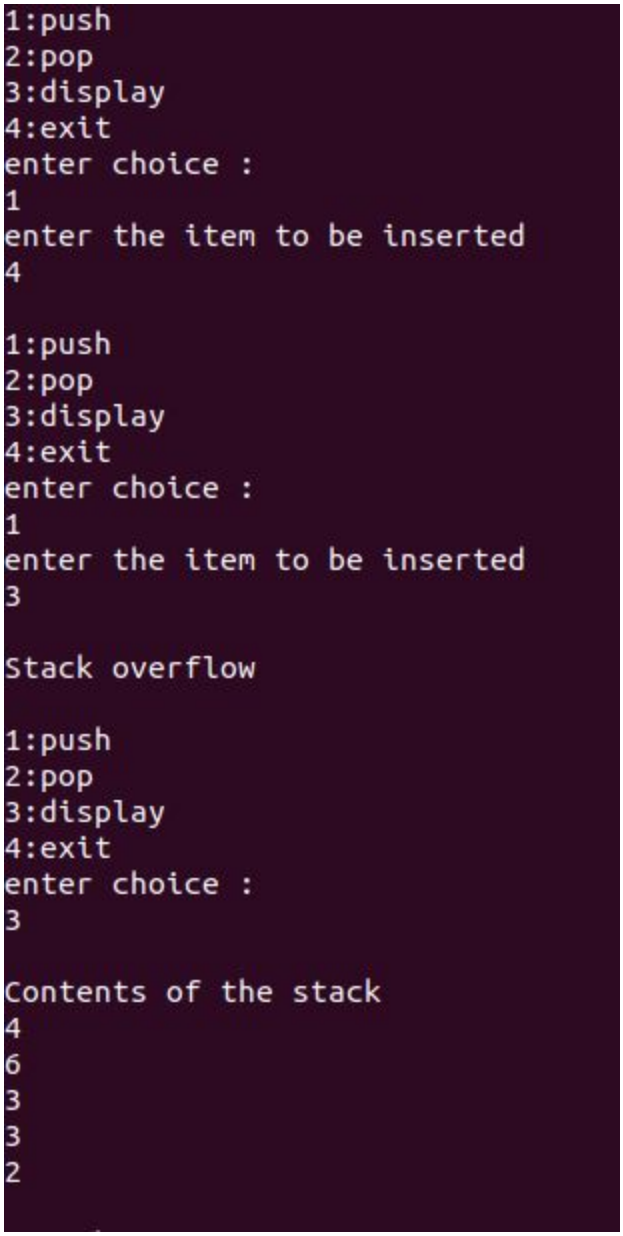
```
#include <stdio.h>
#define STACK_SIZE 5
int push(int item,int s[],int top);
int pop(int s[],int top);
void display(int top,int s[]);

int main()
{
   int top = -1;
   int item,s[10],item_deleted,choice;
   while(1)
   {

      printf("\n1:push\n2:pop\n3:display\n4:exit\nenter choice :\n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:printf("enter the item to be inserted\n");
         scanf("%d",&item);
         top =push(item,s,top);
         break;
         case 2:item_deleted=pop(s,top);
         if(item_deleted!=-1)
         {
         printf("\nitem deleted is %d \n",item_deleted);
         top--;
                        }
         break;
         case 3 : display(top,s);
         break;
         default : return 0;
      }

   }
   return 0;
}

int push(int item,int s[],int top)
{
   if(top==STACK_SIZE-1)
```

```
    {
        printf("\nStack overflow\n");
        return top;
    }
    top++;
    s[top]=item;
    return top;
}

int pop(int s[],int top)
{
    int item_deleted;
    if(top==-1)
    {
        printf("\nStack underflow \n");
        return -1;
    }

    item_deleted = s[top];
    return item_deleted;

}
void display(int top,int s[])
{
    if(top==-1)
    {
        printf("\nstack is empty\n");
        return;

    }
    printf("\nContents of the stack \n");
    for(int i=top;i>=0;i--)
    {
        printf("%d\n",s[i]);
    }
}
```

OUTPUT:

```
1:push
2:pop
3:display
4:exit
enter choice :
1
enter the item to be inserted
5

1:push
2:pop
3:display
4:exit
enter choice :
1
enter the item to be inserted
6

1:push
2:pop
3:display
4:exit
enter choice :
3

Contents of the stack
6
5

1:push
2:pop
3:display
4:exit
enter choice :
2

item deleted is 6
```

```
1:push
2:pop
3:display
4:exit
enter choice :
2

item deleted is 5

1:push
2:pop
3:display
4:exit
enter choice :
3

stack is empty

1:push
2:pop
3:display
4:exit
enter choice :
1
enter the item to be inserted
2

1:push
2:pop
3:display
4:exit
enter choice :
1
enter the item to be inserted
3
```



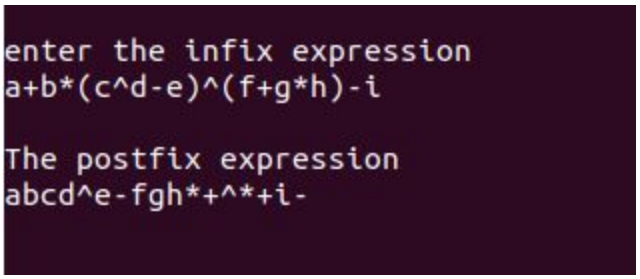
PROGRAM 2 : WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <string.h>
int F(char symbol)
{
        switch(symbol){
        case '+' :
        case '-' : return 2;
```

```
        case '*' :
        case '/' : return 4;
        case '^' :
        case '$' : return 5;
        case '(' : return 0;
        case '#' : return -1;
        default : return 8;
        }
}

int G(char symbol)
{
        switch(symbol)
        {
                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],symbol;
        top = -1;
        s[++top] = '#';
        j=0;
        for(i=0;i<strlen(infix);i++)
        {
                symbol = infix[i];
                while(F(s[top])>G(symbol))
                {
                        postfix[j]=s[top--];
                        j++;
                }

                if(F(s[top])!= G(symbol))
                        s[++top]=symbol;
```

```
            else
                        top--;
        }
        while(s[top] !='#')
        {
                postfix[j++] = s[top--];
        }
        postfix[j]='\0';
}

int main()
{
        char infix[20];
        char postfix[20];
        printf("\nenter the infix expression \n");
        scanf("%s",infix);
        infix_postfix(infix,postfix);
        printf("\nThe postfix expression \n");
        printf("%s \n",postfix);
        return 0;
}
```

OUTPUT :



PROGRAM 3 : 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

```
#include <stdio.h>
#define Q_SIZE 3
int item,front=0,rear=-1,q[10];
```

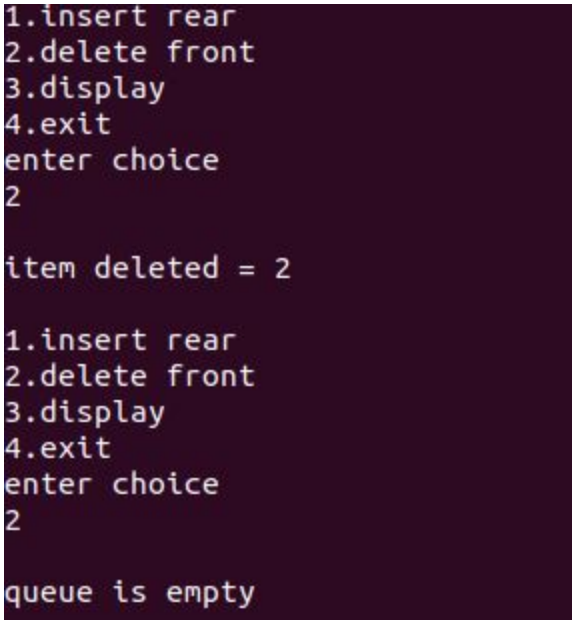
```
void insertrear()
{
	if(rear==Q_SIZE-1)
	{
		printf("\nqueue overflow\n");
		return;
	}
	q[++rear]=item;
}

int deletefront()
{
	if(front>rear)
	{
		front =0;
		rear=-1;
		return -1;
	}
	return q[front++];
}

void displayQ()
{
	if(front>rear)
	{
		printf("\nqueue is empty\n");
		return;
	}
	printf("\ncontents of queue : \n");
	for(int i = front;i<=rear;i++)
	printf("%d\n",q[i]);
}

int main()
{
	int choice;
	for(;;)
	{
		printf("\n1.insert rear\n2.delete front\n3.display\n4.exit\n");
		printf("enter choice \n");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1: printf("\nenter the item to be inserted\n");
```

```
                        scanf("%d",&item);
                        insertrear();
                        break;
                        case 2: item=deletefront();
                        if(item==-1)
                        printf("\nqueue is empty\n");
                        else
                        printf("\nitem deleted = %d\n",item);
                        break;
                        case 3 : displayQ();
                        break;
                        default : return 0;
                }
        }
        return 0;
}
```

```
enter the item to be inserted
3

1.insert rear
2.delete front
3.display
4.exit
enter choice
1

enter the item to be inserted
2
```

```
1.insert rear
2.delete front
3.display
4.exit
enter choice
3

contents of queue :
4
3
2

1.insert rear
2.delete front
3.display
4.exit
enter choice
1

enter the item to be inserted
2

queue overflow
```

```
1.insert rear
2.delete front
3.display
4.exit
enter choice
2

item deleted = 4

1.insert rear
2.delete front
3.display
4.exit
enter choice
2

item deleted = 3

1.insert rear
2.delete front
3.display
4.exit
enter choice
2
```



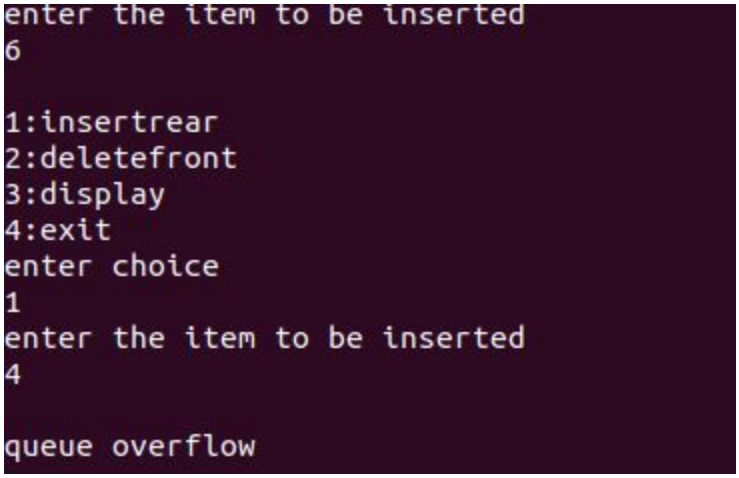
PROGRAM 4 : WAP to simulate the working of a circular queue of integers using an array. Provide the
following operations.
a) Insert b) Delete c) Display
The program should print appropriate messages for queue empty and queue overflow conditions

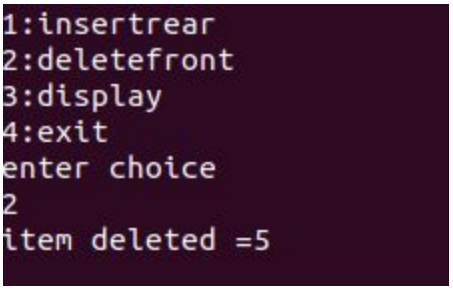
```
#include<stdio.h>

#define QUE_SIZE 3
int item,front=0,rear=-1,q[QUE_SIZE],count=0;
void insertrear()
{
if(count==QUE_SIZE)
{
printf("\nqueue 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;
```

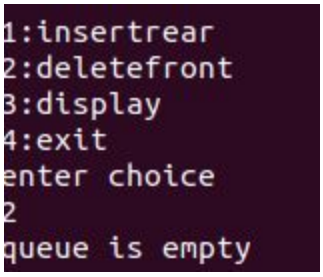
```
count--;
return item;
}
void displayQ()
{
int i,f;
if(count==0)
{
printf("\nqueue is empty\n");
return;
}
f=front;
printf("\nContents of queue \n");
for(i=1;i<=count;i++)
{
printf("%d\n",q[f]);
f=(f+1)%QUE_SIZE;
}
}
int main()
{
 int choice;


 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter 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;
```

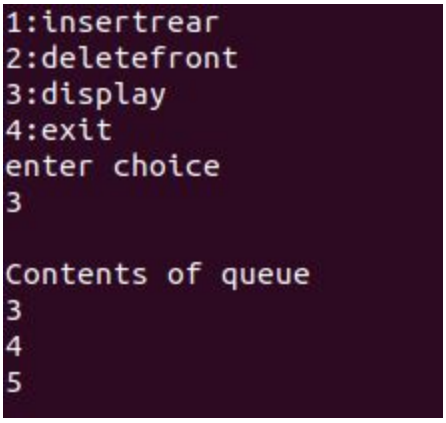
```
 default:return 0;
}
}

}
```

OUTPUT :









PROGRAM 5 : WAP to Implement Singly Linked List with following operations
a) 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.

```
#include <stdio.h>
#include <stdlib.h>
struct node{
        int info;
        struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
```

```
	NODE x;
	x=(NODE)malloc(sizeof(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 = getnode();
	temp->info = item;
	temp->link = NULL;
	if(first == NULL)
	return temp;
	temp->link=first;
	return temp;
}

NODE delete_front(NODE first)
{
	NODE temp;
	if(first == NULL)
	{
		printf("List is empty\n");
		return first;
	}
	printf("Item deleted %d",(first->info));
	temp = first;
	temp=temp->link;
	free(first);
	return temp;
}

NODE insert_rear(NODE first,int item)
{
	NODE temp = getnode(),cur;
```

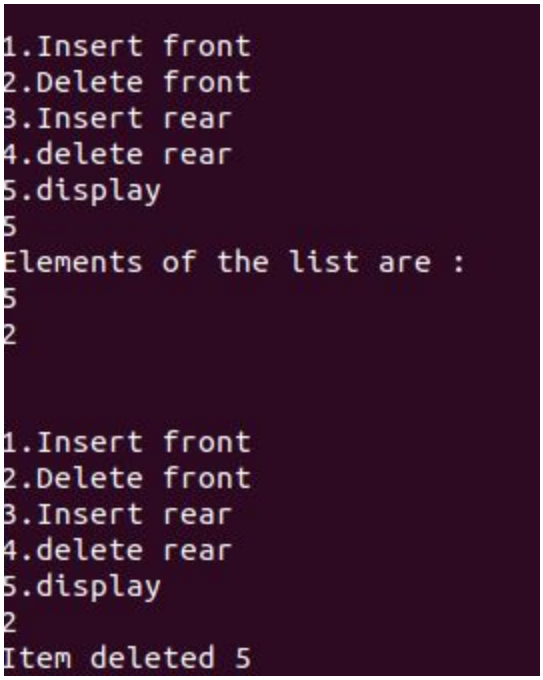
```
        temp->info=item;
        temp->link = NULL;
        if(first==NULL)
        return temp;
        cur=first;
        while(cur->link!=NULL)
        cur=cur->link;
        cur->link=temp;
        return first;
}

NODE delete_rear(NODE first)
{
        NODE cur=first,prev=NULL;
        if(first==NULL)
        {
                printf("List empty\n");
                return NULL;
        }
        if(first->link==NULL)
        {
                printf("item deleted is %d\n",first->info);
                free(first);
                return NULL;
        }
        while(cur->link!=NULL)
        {
                prev=cur;
                cur=cur->link;
        }
        printf("Item deleted is %d\n",(cur->info));
        free(cur);
        prev->link=NULL;
        return first;
}

void display(NODE first)
{
        if(first==NULL)
        {
                printf("List is empty\n");
                return;
        }
        printf("Elements of the list are : \n");
```

```
        for(NODE i=first;i!=NULL;i=i->link)
        printf("%d\n",i->info);
}
int main()
{
        int item,ch;
        NODE first=NULL;
        for(;;)
        {
                printf("\n\n1.Insert front\n2.Delete front\n3.Insert rear\n4.delete
rear\n5.display\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                printf("Enter element to be inserted\n");
                                scanf("%d",&item);
                                first = insert_front(first,item);
                                break;
                        case 2:
                                first = delete_front(first);
                                break;
                        case 3:
                                printf("Enter element to be inserted\n");
                                scanf("%d",&item);
                                first = insert_rear(first,item);
                                break;
                        case 4:
                                first = delete_rear(first);
                                break;
                        case 5:
                                display(first);
                                break;
                        default:return 0;
                }
        }
}
```

```
1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
2
List is empty


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
4
List empty


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
5
List is empty


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
1
Enter element to be inserted
2
```

```
Enter element to be inserted
2


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
1
Enter element to be inserted
5


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
5
Elements of the list are :
5
2
```

```
Elements of the list are :
5
2


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
3
Enter element to be inserted
7


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
5
Elements of the list are :
5
2
7


1.Insert front
2.Delete front
3.Insert rear
4.delete rear
5.display
4
Item deleted is 7
```



PROGRAM 6 : WAP to Implement Singly Linked List with following operations
a) 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.

```
#include<stdio.h>
#include<stdlib.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;
}
NODE insert_rear(NODE first,int item)
{
```

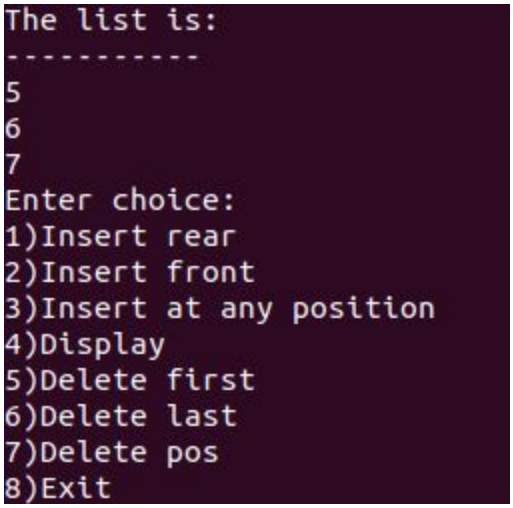
```
NODE temp=getnode();
temp->info=item;
temp->link=NULL;
NODE cur=first;
if(first==NULL)return temp;
while(cur->link!=NULL)
{
cur=cur->link;
}
cur->link=temp;
return first;
}
NODE insert_front(NODE first,int item)
{
NODE temp=getnode();
temp->info=item;
if(first==NULL)return temp;
temp->link=first;
return temp;
}
NODE insert_pos(NODE first,int item,int pos)
{
NODE temp=getnode();
temp->info=item;
temp->link=NULL;
NODE cur=first;
NODE prev=NULL;
if(pos==1)
{
temp->link=first;
return temp;}
int count=0;
while(count<=pos)
{
count=count+1;
if(count==pos)
{
temp->link=cur;
prev->link=temp;
return first;
}
cur=cur->link;
if(count==1)prev=first;
if(count>1)prev=prev->link;
```

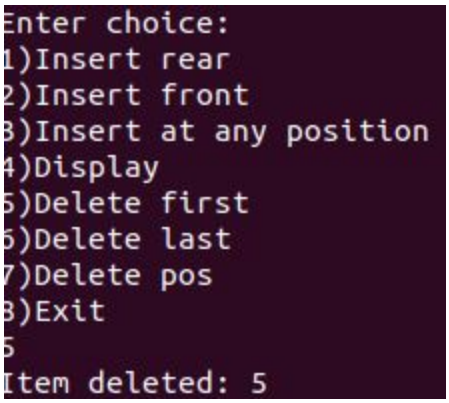
```
}
printf("INVALID POSITION!\n");
free(temp);
return first;
}
NODE delete_first(NODE first)
{
if(first==NULL)
{
printf("LIST EMPTY!\n");
return first;
}
NODE cur=first;
printf("Item deleted: %d\n",first->info);
cur=cur->link;
free(first);
return cur;
}
NODE delete_last(NODE first)
{
if(first==NULL)
{
printf("LIST EMPTY!\n");
return first;
}
NODE cur=first;
NODE prev=NULL;
int c=0;
while(cur->link!=NULL){
cur=cur->link;
c++;
if(c==1)prev=first;
else
{
prev=prev->link;
}
}
printf("Item deleted: %d\n",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE delete_pos(NODE first,int pos)
{
```

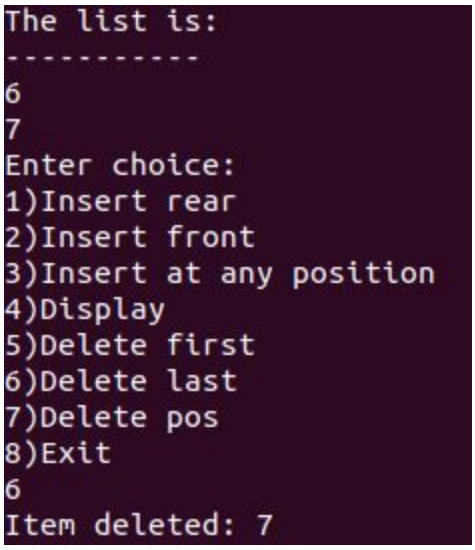
```
NODE cur=first;
NODE prev=first;
int count=1;
if(pos==1)
{
cur=cur->link;
printf("Deleted item: %d\n",first->info);
free(first);
return cur;
}
cur=cur->link;
while(cur!=NULL)
{
count++;
if(pos==count)
{
prev->link=cur->link;
printf("Deleted item: %d\n",cur->info);
free(cur);
return first;
}
cur=cur->link;
prev=prev->link;
}printf("Position not found!\n");
return first;
}
void display(NODE first)
{
NODE cur=first;
printf("The list is:\n----------\n");
while(cur!=NULL)
{
printf("%d\n",cur->info);
cur=cur->link;
}
}
int main()
{
int choice,c=1,item,pos;
NODE first=NULL;
while(c==1)
{
printf("Enter choice:\n1)Insert rear\n2)Insert front\n3)Insert at any position\n4)Display\n5)Delete
first\n6)Delete last\n7)Delete pos\n8)Exit\n");
```

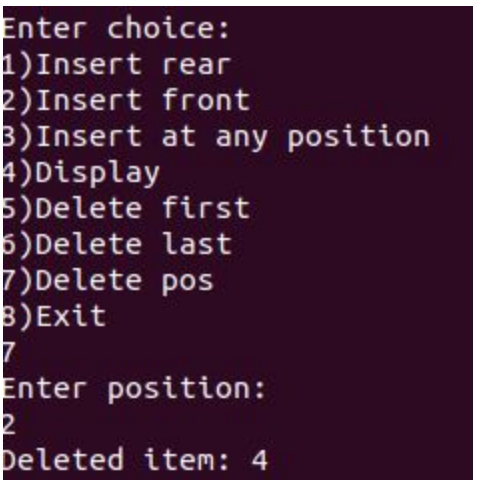
```
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter item :\n");
scanf("%d",&item);
first=insert_rear(first,item);
break;
case 2:
printf("Enter item:\n");
scanf("%d",&item);
first=insert_front(first,item);
break;
case 3:
printf("Enter position\n");
scanf("%d",&pos);
printf("Enter item:\n");
scanf("%d",&item);
first=insert_pos(first,item,pos);
break;case 4:
display(first);
break;
case 5:
first=delete_first(first);
break;
case 6:
first=delete_last(first);
break;
case 7:
printf("Enter position:\n");
scanf("%d",&pos);
first=delete_pos(first,pos);
break;
case 8:
return 0;
default:printf("Invalid choice!\n");
}
}
}
```

OUTPUT :









PROGRAM 7 : WAP Implement Single Link List with following operations
a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

```
#include <stdio.h>
#include <stdlib.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;
}
NODE insert_rear(NODE first, int item)
{
NODE temp;
temp = getnode();
temp->info = item;
temp->link = NULL;
if (first == NULL)
return temp;
NODE cur = first;
while (cur->link != NULL)
{
cur = cur->link;
}
cur->link = temp;
return first;
}
NODE sort(NODE first)
{
int n = 0, i, j;
NODE cur = first;
if (first == NULL)
{
printf("Empty list!\n");
return first;
}
while (cur != NULL)
{
cur = cur->link;
n++;
}NODE next = NULL;
int t;
for (i = 0; i < n - 1; i++)
{
cur = first;
next = cur->link;
for (j = 0; j < n - 1 - i; j++)
{
if (cur->info > next->info)
{
```

```
t = cur->info;
cur->info = next->info;
next->info = t;
}
next = next->link;
cur = cur->link;
}
}
return first;
}
NODE reverse(NODE first)
{
NODE cur = first;
NODE next = NULL, prev = NULL;
while (cur != NULL)
{
next = cur->link;
cur->link = prev;
prev = cur;
cur = next;
}
return prev;
}
NODE concat(NODE first1, NODE first2)
{
NODE cur = first1;
if (first1 == NULL)
{
return first2;
}
while (cur->link != NULL)
{
cur = cur->link;}
cur->link = first2;
return first1;
}
void display(NODE first)
{
NODE cur = first;
while (cur != NULL)
{
printf("%d\n", cur->info);
cur = cur->link;
}
```

```
}
int main()
{
int choice, c = 1, item;
NODE first1 = NULL, first2 = NULL;
while (c == 1)
{
printf("Enter your choice:\n");
printf("1)Insert in list1\n2)Insert in list2\n3)Sort list1\n4)Sort list2\n5)Reverse list1\n6)Reverse
list2\n7)Concatenate\n8)Display list1\n9)Display list2\n10)Exit\n");
scanf("%d", &choice);
switch (choice)
{
case 1:
printf("Enter item:\n");
scanf("%d", &item);
first1 = insert_rear(first1, item);
break;
case 2:
printf("Enter item:\n");
scanf("%d", &item);
first2 = insert_rear(first2, item);
break;
case 3:
first1 = sort(first1);
printf("After sorting:\n");
display(first1);
break;
case 4:
first2 = sort(first2);
printf("After sorting:\n");display(first2);
break;
case 5:
first1 = reverse(first1);
printf("After reversing:\n");
display(first1);
break;
case 6:
first2 = reverse(first2);
printf("After reversing:\n");
display(first2);
break;
case 7:
first1 = concat(first1, first2);
```

```
printf("After concatenating:\n");
display(first1);
break;
case 8:
printf("LIST1:\n-------\n");
display(first1);
break;
case 9:
printf("LIST2:\n-------\n");
display(first2);
break;
case 10:
return 0;
default:
printf("Invalid choice!\n");
}
}
}
```

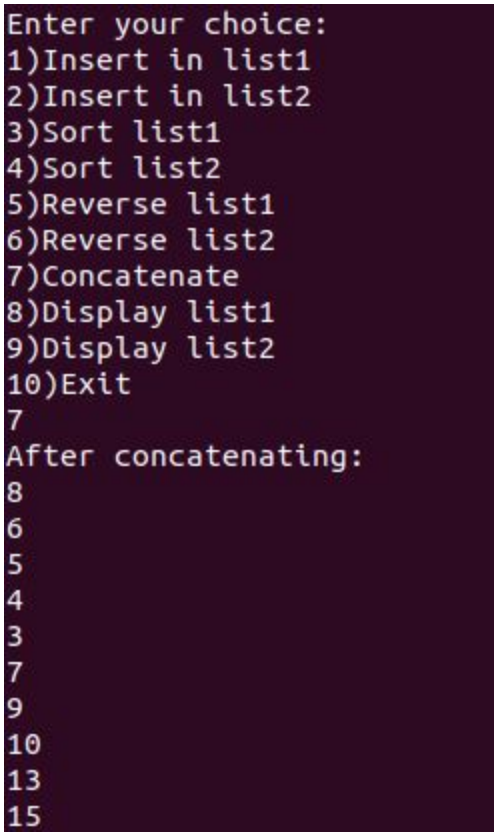
OUTPUT :

```
LIST1:
-------
4
5
6
8
3
Enter your choice:
1)Insert in list1
2)Insert in list2
3)Sort list1
4)Sort list2
5)Reverse list1
6)Reverse list2
7)Concatenate
8)Display list1
9)Display list2
10)Exit
```

```
LIST2:
-------
7
9
10
15
13
Enter your choice:
1)Insert in list1
2)Insert in list2
3)Sort list1
4)Sort list2
5)Reverse list1
6)Reverse list2
7)Concatenate
8)Display list1
9)Display list2
10)Exit
```

```
Enter your choice:
1)Insert in list1
2)Insert in list2
3)Sort list1
4)Sort list2
5)Reverse list1
6)Reverse list2
7)Concatenate
8)Display list1
9)Display list2
10)Exit
3
After sorting:
3
4
5
6
8
```

```
Enter your choice:
1)Insert in list1
2)Insert in list2
3)Sort list1
4)Sort list2
5)Reverse list1
6)Reverse list2
7)Concatenate
8)Display list1
9)Display list2
10)Exit
5
After reversing:
8
6
5
4
3
```



PROGRAM 8 : WAP to implement Stack & Queues using Linked Representation

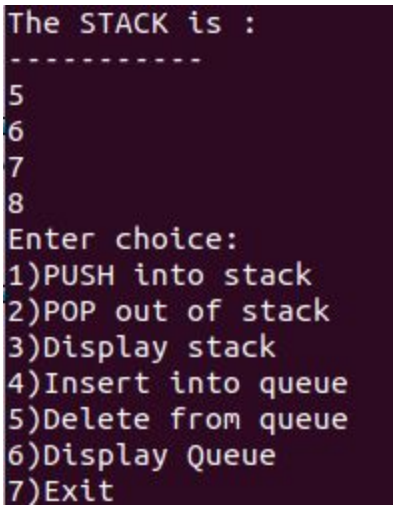
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("MEMORY FULL!\n");
exit(0);
}
```

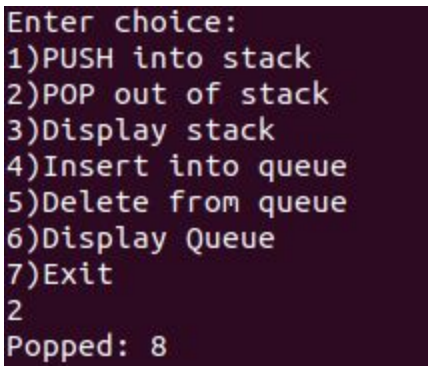
```
return x;
}
NODE insert_rear(NODE first,int item)
{
NODE temp=getnode();
temp->info=item;
temp->link=NULL;
NODE cur=first;
if(first==NULL)return temp;
while(cur->link!=NULL)
{cur=cur->link;
}
cur->link=temp;
return first;
}
NODE delete_first(NODE first)
{
if(first==NULL)
{
printf("EMPTY!\n");
return first;
}
NODE cur=first;
printf("Item deleted: %d\n",first->info);
cur=cur->link;
free(first);
return cur;
}
NODE delete_last(NODE first)
{
if(first==NULL)
{
printf("Underflow\n");
return first;
}
NODE cur=first;
NODE prev=NULL;
int c=0;
while(cur->link!=NULL)
{
cur=cur->link;
c++;
if(c==1)prev=first;
else
```

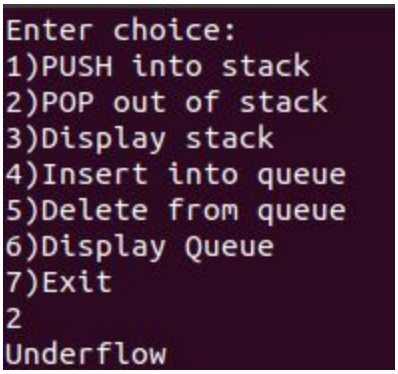
```
{
prev=prev->link;
}
}
printf("Popped: %d\n",cur->info);
free(cur);prev->link=NULL;
return first;
}
void display(NODE first)
{
NODE cur=first;
while(cur!=NULL)
{
printf("%d\n",cur->info);
cur=cur->link;
}
}
int main()
{
int c=1,choice,item;
NODE qf=NULL,sf=NULL;
while(c==1)
{
printf("Enter choice:\n");
printf("1)PUSH into stack\n2)POP out of stack\n3)Display stack\n4)Insert into queue\n5)Delete
from queue\n6)Display Queue\n7)Exit\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter item:\n");
scanf("%d",&item);
sf=insert_rear(sf,item);
break;
case 2:
sf=delete_last(sf);
break;
case 3:
printf("The STACK is :\n----------\n");
display(sf);
break;
case 4:
printf("Enter item:\n");
scanf("%d",&item);
```

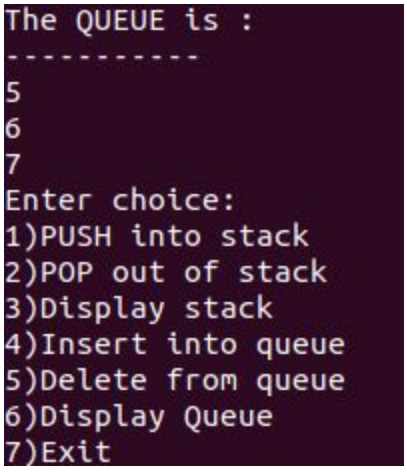
```
qf=insert_rear(qf,item);
break;
case 5:qf=delete_first(qf);
break;
case 6:
printf("The QUEUE is :\n----------\n");
display(qf);
break;
case 7:
return 0;
break;
default:
printf("Invalid choice!\n");
}
}
}
```

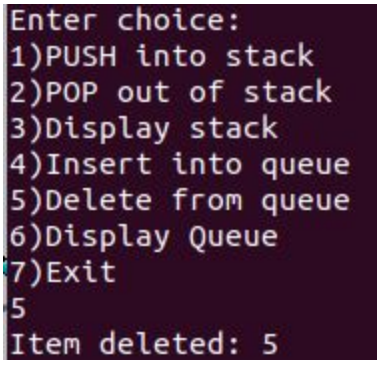
OUTPUT :











PROGRAM 9 : 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. d) Display the contents of the list

```
#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("mem full\n");
  return NULL;
 }
 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 element to be inserted ");
 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;
}
printf("\nList : \n");
for(temp=head->rlink;temp!=head;temp=temp->rlink)
printf("%d\n",temp->info);
```

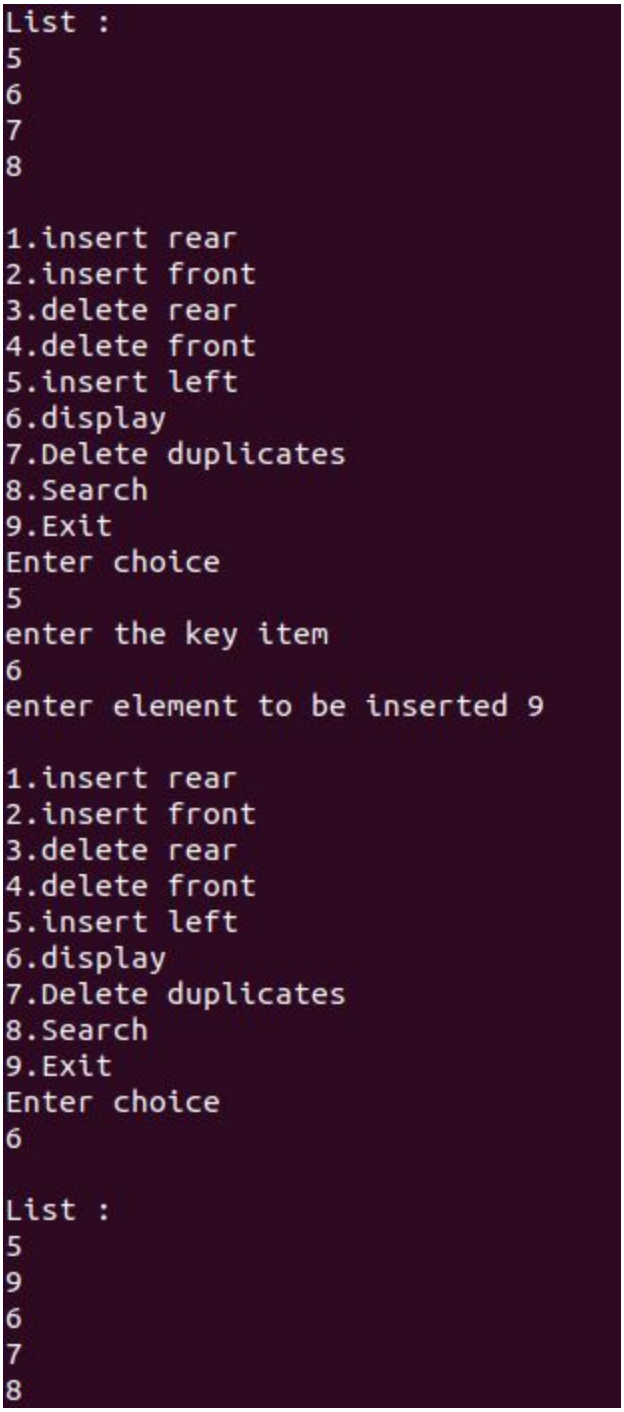
```
}

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 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 delete_all_keys(NODE head,int key)
{
        int count =0;
        if(head->rlink==head)
        {
                printf("List empty \n");
                return head;
        }
        NODE cur = head->rlink;
        NODE prev;
        NODE next;
        while(cur!=head)
        {
                if(cur->info==key)
                {
                        if(count!=0)
                        {
                        prev=cur->llink;
                        next=cur->rlink;
                        prev->rlink=next;
                        next->llink=prev;
                        freenode(cur);
                        count++;
                        cur=next;
                        }
                        else
                        {
                                count++;
                                cur=cur->rlink;
                        }
                }
                else
                cur=cur->rlink;
        }
        if(count==0)
        {
                printf("Element not found\n");
                return head;
        }
```

```
        if(count==1)
        {
                printf("\nno duplicates\n");
        }
        printf("Element deleted in %d positions",(count-1));
        return head;
}

void search(NODE head,int item)
{
        NODE cur;
        if(head->rlink==head)
        {
                printf("List empty\n");
                return;
        }
        cur=head->rlink;
        while(cur!=head)
        {
                if(cur->info==item)
                {
                        printf("Element found\n");
                        return;
                }
                cur = cur->rlink;
        }
        printf("Element not found \n");
        return ;
}


int main()
{
int item,choice;
NODE head;
head=getnode();
head->rlink=head;
head->llink=head;
for(;;)
{
printf("\n1.insert rear\n2.insert front\n3.delete rear\n4.delete front\n5.insert
left\n6.display\n7.Delete duplicates\n8.Search\n9.Exit\nEnter 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 item at front end\n");
                    scanf("%d",&item);
                    head=dinsert_front(item,head);
                    break;
  case 3:head=ddelete_rear(head);
                    break;
  case 4:head=ddelete_front(head);
                    break;
  case 5:printf("enter the key item\n");
                    scanf("%d",&item);
                    head=insert_leftpos(item,head);
                    break;
  case 6:display(head);
                    break;
  case 7:printf("Enter item \n");
                    scanf("%d",&item);
                    head = delete_all_keys(head,item);
                    break;
  case 8:printf("Enter item \n");
                    scanf("%d",&item);
                    search(head,item);
                    break;
  default:exit(0);
                     break;
  }
 }
}
```

OUTPUT :



PROGRAM 10 : 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.

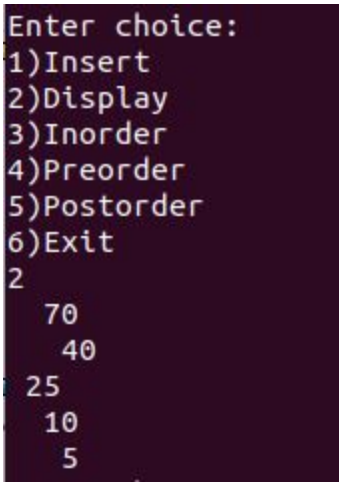
```
#include <stdio.h>
```

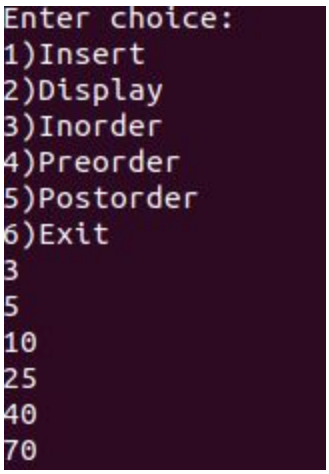
```
#include <stdlib.h>
struct node
{
int info;
struct node *rlink;
struct node *llink;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x = (NODE)malloc(sizeof(struct node));
if (x == NULL)
{
printf("MEMORY FULL\n");
exit(0);
}
return x;
}
NODE insert(NODE root, int item)
{
NODE temp = getnode();
temp->info = item;
temp->llink = NULL;
temp->rlink = NULL;
if (root == NULL)
{
return temp;
}
NODE cur = root;
while (cur != NULL)
{
if (item > cur->info)
{
if (cur->rlink != NULL)
cur = cur->rlink;
else
{
cur->rlink = temp;
break;
}
}
else if (item < cur->info){
if (cur->llink != NULL)
cur = cur->llink;
```

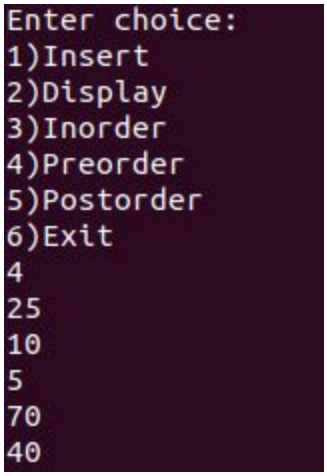
```
else
{
cur->llink = temp;
break;
}
}
else
{
printf("Item exists!\n");
break;
}
}
return root;
}
void inorder(NODE root)
{
if (root != NULL)
{
inorder(root->llink);
printf("%d\n", root->info);
inorder(root->rlink);
}
}
void preorder(NODE root)
{
if (root != NULL)
{
printf("%d\n", root->info);
preorder(root->llink);
preorder(root->rlink);
}
}
void postorder(NODE root)
{
if (root != NULL)
{
postorder(root->llink);
postorder(root->rlink);printf("%d\n", root->info);
}
}
void display(NODE root, int i)
{
int j;
if (root != NULL)
```

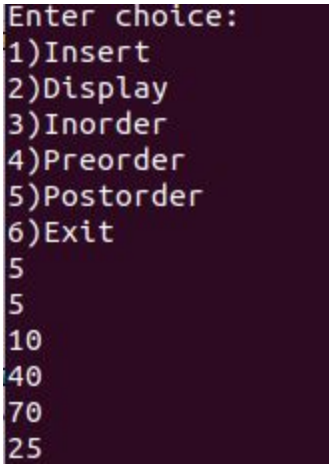
```
{
display(root->rlink, i + 1);
for (j = 0; j < i; j++)
printf(" ");
printf("%d \n", root->info);
display(root->llink, i + 1);
}
}
int main()
{
int choice, c = 1, item;
NODE root = NULL;
while (c == 1)
{
printf("Enter choice:\n1)Insert\n2)Display\n3)Inorder\n4)Preorder\n5)Postorder\n6)Exit\n");
scanf("%d", &choice);
switch (choice)
{
case 1:
printf("Enter item:\n");
scanf("%d", &item);
root = insert(root, item);
break;
case 2:
display(root, 1);
break;
case 3:
inorder(root);
break;
case 4:
preorder(root);
break;
case 5:
postorder(root);
break;case 6:c=0;break;
default:
printf("Invalid choice!\n");
}
}
}
```

OUTPUT :


Enter choice:
1)Insert
2)Display
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
70
40
25
10
5


Enter choice:
1)Insert
2)Display
3)Inorder
4)Preorder
5)Postorder
6)Exit
3
5
10
25
40
70


Enter choice:
1)Insert
2)Display
3)Inorder
4)Preorder
5)Postorder
6)Exit
4
25
10
5
70
40


Enter choice:
1)Insert
2)Display
3)Inorder
4)Preorder
5)Postorder
6)Exit
5
5
10
40
70
25