DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DATA STRUCTURES LAB MANUAL

SUBJECT CODE : A0512

II B.Tech I Semester (MR-20)

ACADEMIC YEAR : 2021-22

MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)

(An Autonomous institution, Autonomy granted by UGC and affiliated to JNTUH, Accredited by NAAC with 'A' Grade, Accredited by NBA (2008-11) & Recipient of World Bank Assistance under TEQIP phase – II S.C.1.1for the period (2011-14))
Maisammaguda, Dhulapally (Post. Via. Kompally), Secunderabad – 500 100.
Website: www.mrec.ac.in

OBJECTIVES:

1. Develop skills to use and analyse simple linear and non-linear datastructures.
2. Strengthen the ability to identify and apply the suitable data structure for the given real world problem.
3. Derive and express space and time complexities.
4. Develop skills in debugging a program.

LIST OF EXPERIMENTS:

1. Write a recursive program to solve Towers of Hanoi problem - N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg.
2. Write a program to create a single linked list, with the following operations:
a) Insertion b) Deletion c) Display the elements d) Count no of elements.
3. Write a program to create a circular linked list, with the following operations:
a) Insertion b) Deletion c) Display the elements d) Count number of elements.
4. Write a program to create a double linked list, with the following operations:
a) Insertion b) Deletion c) Display the elements d) Count number of elements.
5. Write a program to implements stack operations using:
a) Arrays b) Linked list
6. Write a program to: a) Evaluate Postfix expression.
b) Convert infix expression into postfix expression
7. Write a program to implements Linear Queue operations using:
a) Arrays b) Linked list
8. Write a program to implements Circular Queue operations using Arrays
9. Write a program to implements Double-ended Queue operations using:
a) Arrays b) Double Linked List
10. Write a recursive program to create a Binary Tree of integers, traverse the tree in preorder, in order and post order and also print the number of leaf nodes and height of the tree
11. Write a program to create a Binary Search Tree (BST) and perform insert and search Operations on it.
12. Write a program for implementing the following graph traversal algorithms:
a) Breadth First Search (BFS) b) Depth First Search (DFS)

INDEX

1)Write a recursive program to solve Towers of Hanoi problem - N disks are to be transferred from peg S to peg D with Peg I as the intermediate peg.

/*It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

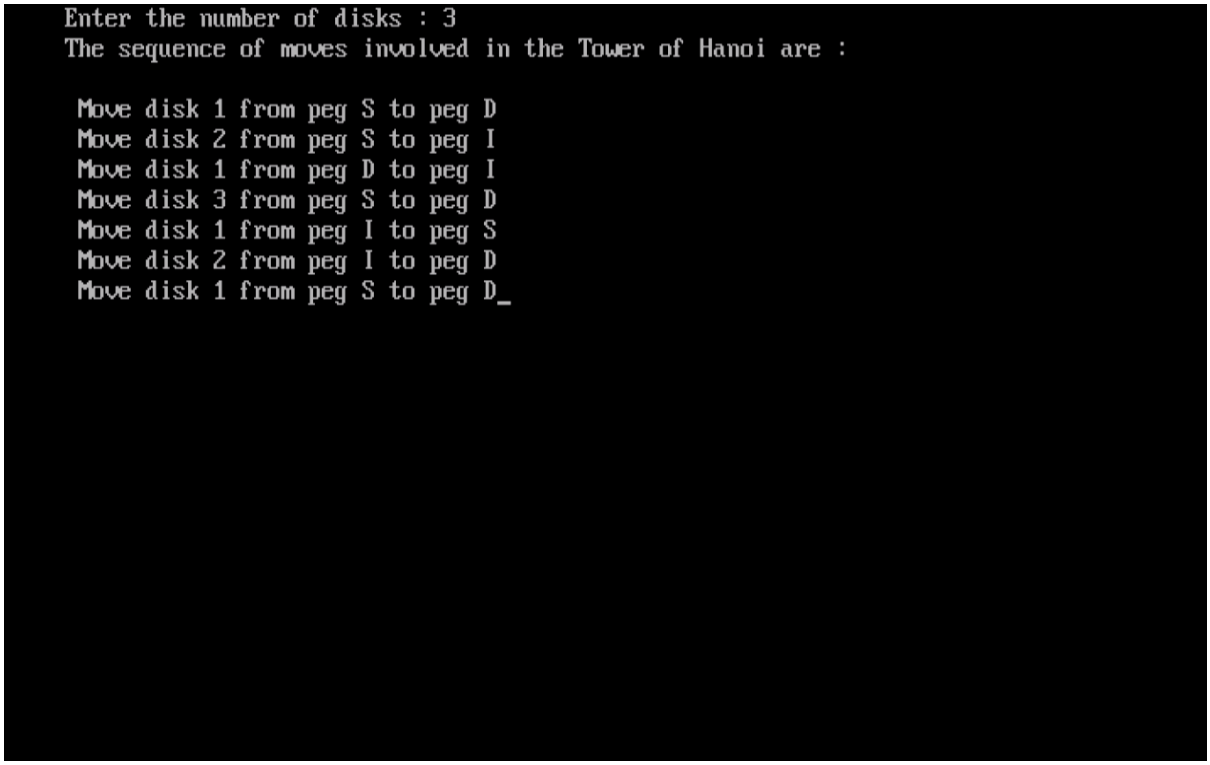
The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.*/

```
#include<conio.h>
#include <stdio.h>
void towers(int, char, char, char);
void main()
{
    int num;
    clrscr();
    printf("Enter the number of disks : ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi
are :\n");
    towers(num, 'S', 'D', 'I');
 getch();
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
    if (num == 1)
    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg,
topeg);
        return;
    }
```

```
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num,
frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```

output:



2. Write a program that uses functions to perform the following:
a) Create a single linked list of integers.
b) Insert an element in all locations of the single linked list.

c) Implement all delete operations on single linked list.
d) Display the contents of the single linked list after insertion/deletion.

/***Singly Linked List** is a linear data structure. In a singly linked list each node in the list stores the contents of the node and a pointer or reference to the next node in the list. It does not store any pointer or reference to the previous node.One NULL pointer will be present in the last node*/

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
struct node
{
int data;
struct node *next;
};
struct node *new_node=NULL ,*ptr=NULL,*preptr=NULL,*temp=NULL;
struct node *start=NULL;
struct node *create_LL(struct node *);
void traverse(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_before(struct node *);
struct node *insert_end(struct node *);
struct node *insert_after(struct node *);
struct node*del_begin(struct node*);
struct node*del_end(struct node*);
struct node*del_node(struct node*);
struct node*del_after(struct node*);
```

```
void main()
{
int option;
clrscr();
do
{
printf("\n********* main menu**************");
printf("\n 1.create a list");
printf("\n 2.display \n3.add node at beg \n 4.add node at end
\n");
printf("\n5.add node before a given node");
printf("\n 6.add node after a given node");
printf("\n 7.delete at the beginning");
printf("\n 8.delete at the end");
printf("\n 9.delete a particular node");
printf("\n 10.delete a node after a given value");
printf("\n 11.exit \n");
printf("\n enter option");
scanf("%d",&option);
switch(option)
{
case 1:
start=create_LL(start);
printf("\n linked list created");
break;
case 2:
traverse(start);
break;
case 3:
start=insert_beg(start);
```

```
break;
case 4:
start=insert_end(start);
break;
case 5:
start=insert_before(start);
break;
case 6:
start=insert_after(start);
break;
case 7:start=del_begin(start);
break;
case 8:start=del_end(start);
break;
case 9:start=del_node(start);
break;
case 10:start=del_after(start);
break;
case 11:
exit(0);
}
}
while(option<=11);
getch();
}

/*single linked list  creation operation*/
struct node *create_LL(struct node* start)
{
int val,n,i=0;
```

```
printf("enter the number of elements required \n");
scanf("%d",&n);
for( i=0;i<n;i++)
{
    printf("\nenter the data");
    scanf("%d",&val);
    new_node=(struct node *)malloc(sizeof(struct node));
 new_node->data=val;

       if(start==NULL)
         {
           new_node->next=NULL;
start=new_node;

          }
else
{
ptr=start;
          while(ptr->next!=NULL)
            {
ptr=ptr->next;
            }
          ptr->next=new_node;
          new_node->next=NULL;
}
}
return start;
}

/*single linked list display operation*/
```

```
void traverse(struct node *start)
{
ptr=start;
if(start==NULL)
{
printf("list is empty");
}
else
{
while(ptr!=NULL)
{
printf("\t %d",ptr->data);
ptr=ptr->next;
}
}
}
/*single linked list  insertion operation*/
struct node *insert_beg(struct node *start)
{
int num;
printf("\nenter data:");
scanf("%d",&num);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;
new_node->next=start;
start=new_node;
return start;
}
struct node *insert_end(struct node *start)
{
```

```
int num,val;
printf("\n enter data:");
scanf("%d",&num);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;
new_node->next=NULL;
ptr=start;
while(ptr->next!=NULL)
ptr=ptr->next;
ptr->next=new_node;
return start;
}
struct node *insert_before(struct node *start)
{
int num,val;
printf("\n enter data:");
scanf("%d",&num);
printf("\n enter value before which data to be inserted:");
scanf("%d",&val);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;
ptr=start;
/*insert before first node*/
if(ptr->data==val)
{
new_node->next=start;
start=new_node;
}
else
{
```

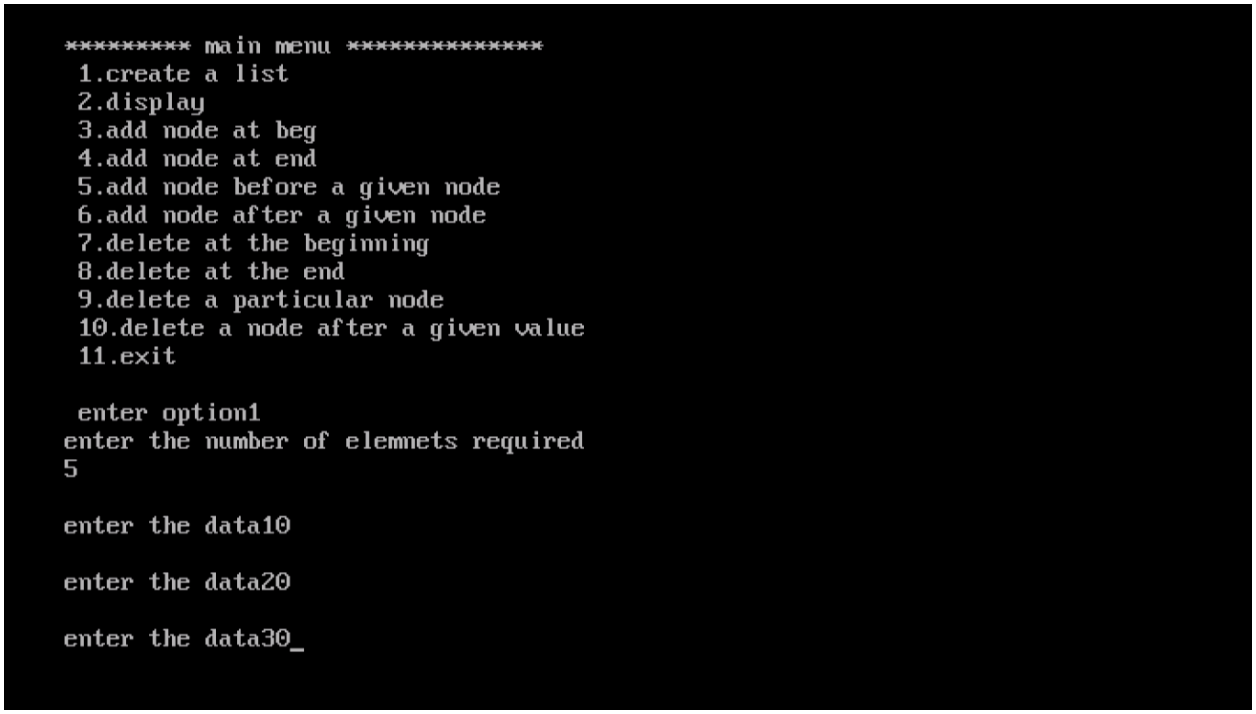
```
while(ptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=new_node;
new_node->next=ptr;
}
return start;
}
struct node *insert_after(struct node *start)
{
int num,val;
printf("\n enter data:");
scanf("%d",&num);
printf("\n enter value after which data has to be inserted");
scanf("%d",&val);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;
ptr=start;
while(ptr->data!=val)
{
ptr=ptr->next;
}
temp=ptr->next;
ptr->next=new_node;
new_node->next=temp;
return start;
}
/*linked list deletion operatios*/
```

```
struct node*del_begin(struct node*start)
{
ptr=start;
start=start->next;
free(ptr);
return start;
}
struct node*del_end(struct node*start)
{
ptr=start;
while(ptr->next!=NULL)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=NULL;
free(ptr);
return start;
}
struct node*del_node(struct node*start)
{
int val;
printf("\n enter the number to be deleted");
scanf("%d",&val);
ptr=start;
/*deleting first node*/
if(ptr->data==val)
{start=del_begin(start);
return start;
```

```
}
else
{

while(ptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=ptr->next;
free(ptr);
}
return start;
}
struct node*del_after(struct node*start)
{
int val;
printf("\n enter the value after which the data has to be
deleted");
scanf("%d",&val);
ptr=start;
while(ptr->data!=val)
{
ptr=ptr->next;
}
if(ptr==NULL)
printf("no element to delete");
else
{
temp=ptr->next;
```

```
ptr->next=temp->next;
free(temp);
}
return start;
}
```

Output:



```
4.add node at end
5.add node before a given node
6.add node after a given node
7.delete at the beginning
8.delete at the end
9.delete a particular node
10.delete a node after a given value
11.exit

enter option2
        10      20      30      40      50
********** main menu ***************
1.create a list
2.display
3.add node at beg
4.add node at end
5.add node before a given node
6.add node after a given node
7.delete at the beginning
8.delete at the end
9.delete a particular node
10.delete a node after a given value
11.exit

enter option
```

```
8.delete at the end
9.delete a particular node
10.delete a node after a given value
11.exit

enter option5

enter data:100

enter value before which data to be inserted:20

********** main menu ***************
1.create a list
2.display
3.add node at beg
4.add node at end
5.add node before a given node
6.add node after a given node
7.delete at the beginning
8.delete at the end
9.delete a particular node
10.delete a node after a given value
11.exit

enter option_
```

```
 4.add node at end
 5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option2
         10      100     20      30      40      50
********** main menu ***************
 1.create a list
 2.display
 3.add node at beg
 4.add node at end
 5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option
```

```
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option9

 enter the number to be deleted20

********** main menu ***************
 1.create a list
 2.display
 3.add node at beg
 4.add node at end
 5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option
```

```
4.add node at end
5.add node before a given node
6.add node after a given node
7.delete at the beginning
8.delete at the end
9.delete a particular node
10.delete a node after a given value
11.exit

enter option2
        10      100     30      40      50
********** main menu ***************
1.create a list
2.display
3.add node at beg
4.add node at end
5.add node before a given node
6.add node after a given node
7.delete at the beginning
8.delete at the end
9.delete a particular node
10.delete a node after a given value
11.exit

enter option
```

3. Write a program that uses functions to perform the following:
a) Create a circular linked list of integers.
b) Insert an element in all locations of the circular linked list.
c) Implement all delete operations on circular linked list.
d) Display the contents of the circular link list after insertion/deletion.

```
/*Circular Linked List is a variation of Linked list in which the  last element points to first
element.There is no NULL pointer present in the list  */

#include<stdio.h>
#include<conio.h>
#include<malloc.h>
struct node
{
int data;
struct node *next;
};
struct node *new_node=NULL ,*ptr=NULL,*preptr=NULL,*temp=NULL;
```

```
struct node *start=NULL;
struct node *create_LL(struct node *);
void traverse(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_before(struct node *);
struct node *insert_end(struct node *);
struct node *insert_after(struct node *);
struct node*del_begin(struct node*);
struct node*del_end(struct node*);
struct node*del_node(struct node*);
struct node*del_after(struct node*);
void main()
{
int option;
clrscr();
do
{
printf("\n********* main menu**************");
printf("\n 1.create a list");
printf("\n 2.display \n3.add node at beg \n 4.add node at end
\n");
printf("\n5.add node before a given node");
printf("\n 6.add node after a given node");
printf("\n 7.delete at the beginning");
printf("\n 8.delete at the end");
printf("\n 9.delete a particular node");
printf("\n 10.delete a node after a given value");
printf("\n 11.exit \n");
printf("\n enter option");
scanf("%d",&option);
switch(option)
{
case 1:
start=create_LL(start);
printf("\n linked list created");
break;
case 2:
traverse(start);
break;
case 3:
start=insert_beg(start);
break;
case 4:
start=insert_end(start);
break;
case 5:
start=insert_before(start);
break;
case 6:
```

```
start=insert_after(start);
break;
case 7:start=del_begin(start);
break;
case 8:start=del_end(start);
break;
case 9:start=del_node(start);
break;
case 10:start=del_after(start);
break;
case 11:
exit(0);
}
}
while(option<=11);
getch();
}

/*circular linked list  creation operation*/
struct node *create_LL(struct node* start)
{
int val,n,i=0;
printf("enter the number of elements required \n");
scanf("%d",&n);
for( i=0;i<n;i++)
{
    printf("\nenter the data");
    scanf("%d",&val);

     new_node=(struct node *)malloc(sizeof(struct node));

   new_node->data=val;

       if(start==NULL)

      {


        start=new_node;
         new_node->next=start;
       }

     else
      {

       ptr=start;

         while(ptr->next!=start)
```

```
                {

               ptr=ptr->next;
                 }

        ptr->next=new_node;

        new_node->next=start;


      }
}
return start;

}

/*circular linked list display operation*/
void traverse(struct node *start)
{

ptr=start;
if(start==NULL)
printf("linked list is empty");
else
{
while(ptr->next!=start)
{
printf("\t %d",ptr->data);
ptr=ptr->next;
}
printf("\t %d",ptr->data);
}
}
/*circular linked list  insertion operation*/
struct node *insert_beg(struct node *start)
{
int num;
ptr=start;
printf("\nenter data:");
scanf("%d",&num);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;

new_node->next=start;
while(ptr->next!=start)
ptr=ptr->next;
ptr->next=new_node;
start=new_node;
return start;
```

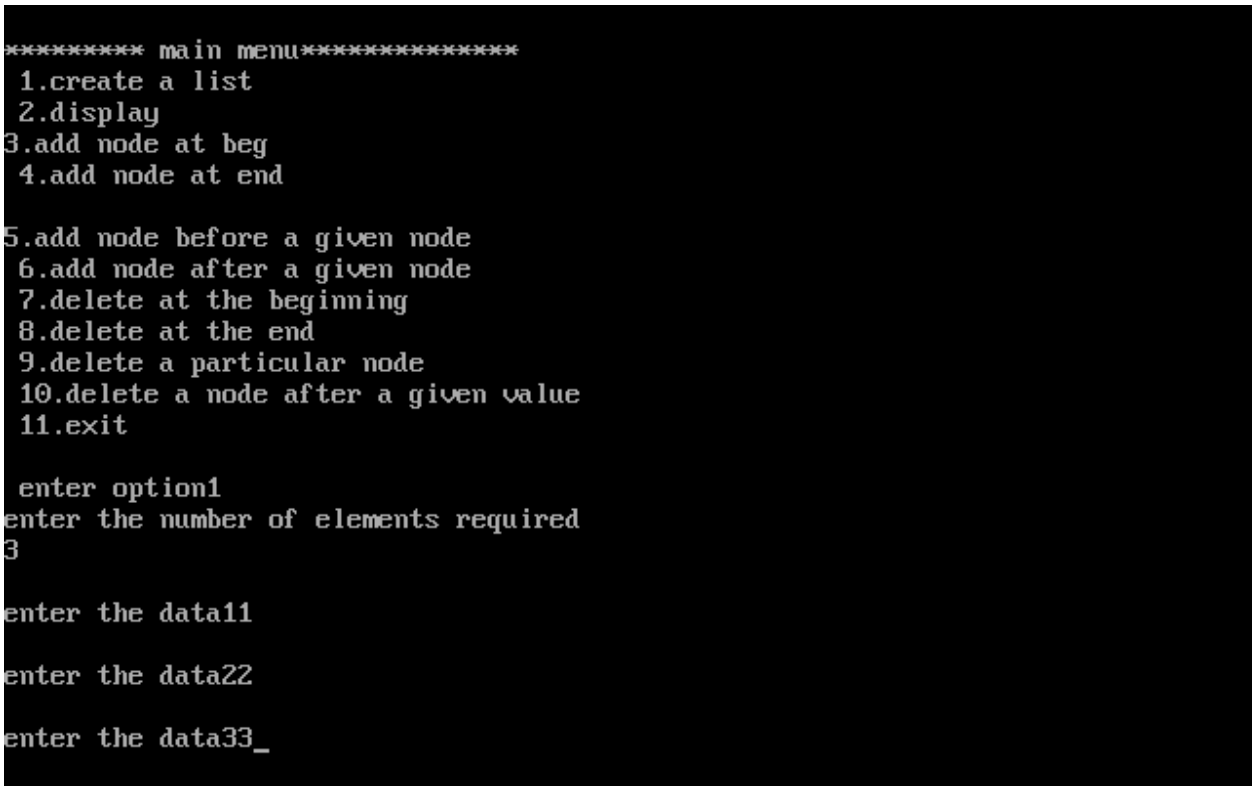
```
}
struct node *insert_end(struct node *start)
{
int num,val;
ptr=start;
printf("\n enter data:");
scanf("%d",&num);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;
while(ptr->next!=start)
ptr=ptr->next;
ptr->next=new_node;
new_node->next=start;
return start;
}
struct node *insert_before(struct node *start)
{
int num,val;
printf("\n enter data:");
scanf("%d",&num);
printf("\n enter value before which data to be inserted:");
scanf("%d",&val);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;
ptr=start;
if(ptr->data==val)
{
  new_node->next=start;
while(ptr->next!=start)
ptr=ptr->next;
ptr->next=new_node;
start=new_node;

}
else
{
while(ptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=new_node;
new_node->next=ptr;
}
return start;
}
struct node *insert_after(struct node *start)
{
int num,val;
```

```
printf("\n enter data:");
scanf("%d",&num);
printf("\n enter value after which data has to be inserted");
scanf("%d",&val);
new_node=(struct node *)malloc(sizeof(struct node));
new_node->data=num;
ptr=start;
while(ptr->data!=val)
{
ptr=ptr->next;
}
temp=ptr->next;

ptr->next=new_node;
new_node->next=temp;


return start;
}
/*linked list deletion operatios*/

struct node*del_begin(struct node*start)
{
ptr=start;
temp=start;


while(ptr->next!=start)
ptr=ptr->next;
/*deleting one existing node*/
if(ptr==start)
{
free(start);
printf("list empty");
}
else
{
start=start->next;
ptr->next=start;
free(temp);
}

return start;
}
struct node*del_end(struct node*start)
{
ptr=start;
while(ptr->next!=start)
{
```

```
preptr=ptr;
ptr=ptr->next;
}
preptr->next=start;
free(ptr);
return start;
}
struct node*del_node(struct node*start)
{
int val;
printf("\n enter the number to be deleted");
scanf("%d",&val);
ptr=start;
if(ptr->data==val)
{
start=del_begin(start);
return start;
}
else
{
while(ptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=ptr->next;
free(ptr);
}
return start;
}
struct node*del_after(struct node*start)
{
int val;
printf("\n enter the value after which the data has to be
deleted");
scanf("%d",&val);
ptr=start;


while(ptr->data!=val)
{
ptr=ptr->next;
}/*deleting a node after last node will delete first node in
circular list*/
if(ptr->next==start)
{
start=del_begin(start);
return start;
}
```

```
else
{
temp=ptr->next;
ptr->next=temp->next;
free(temp);
 }
return start;
}
```

Output:



```
5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option2
         11      22      33
********** main menu***************
 1.create a list
 2.display
3.add node at beg
 4.add node at end

5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option
```

```
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option3

enter data:32

********** main menu***************
 1.create a list
 2.display
3.add node at beg
 4.add node at end

5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option_
```

```
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option9

 enter the number to be deleted22

********** main menu***************
 1.create a list
 2.display
3.add node at beg
 4.add node at end

5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option_
```

```
5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option2
         32      11      33
********** main menu***************
 1.create a list
 2.display
3.add node at beg
 4.add node at end

5.add node before a given node
 6.add node after a given node
 7.delete at the beginning
 8.delete at the end
 9.delete a particular node
 10.delete a node after a given value
 11.exit

 enter option_
```

4. Write a program that uses functions to perform the following:
a) Create a double linked list of integers.
b) Insert an element in all locations of the double linked list.
c) Implement all delete operations on double linked list.
d) Display the contents of the double linked list after insertion/deletion.

/*a **doubly linked list** is a linked data structure that consists of a set of sequentially linked records called nodes Each node contains thee fields.They are actual data part and two pointers that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, typically a sentinel node or NULL, to facilitate traversal of the list. */

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

struct node
{
struct node*prev;
int data;
struct node*next;
}*start,*ptr,*temp,*newnode;
struct node*create(struct node*);
struct node*ins_beg(struct node*);
struct node*ins_end(struct node*);
```

```
struct node*ins_bef(struct node*);
struct node*ins_aft(struct node*);
struct node*del_beg(struct node*);
struct node*del_end(struct node*);
struct node*del_par(struct node*);
struct node*del_bef(struct node*);
struct node*del_aft(struct node*);
void display(struct node*);
void main()
{
int ch;
clrscr();
do
{
printf("\n1.create\n");
printf("2.dispaly\n");
printf("3.insert begin\n");
printf("4.insert end\n");
printf("5.insert before a node\n");
printf("6.insert after a node\n");
printf("7.delete begin\n");
printf("8.delete end\n");
printf("9.delete a node\n");
printf("10.del before\n");
printf("11.del after\n");
printf("12.exit\n");
printf("enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:start=create(start);
break;
case 2:display(start);
break;
case 3:start=ins_beg(start);
break;
case 4:start=ins_end(start);
break;
case 5:start=ins_bef(start);
break;
case 6:start=ins_aft(start);
break;
case 7:start=del_beg(start);
break;
case 8:start=del_end(start);
break;
case 9:start=del_par(start);
break;
case 10:start=del_bef(start);
```

```
break;
case 11:start=del_aft(start);
break;
case 12:exit(0);
}
}while(ch<=12);
getch();
}
struct node*create(struct node*start)
{
int n,i,val;
printf("enter the no");
scanf("%d",&n);
for(i=0;i<n;i++)
{
newnode=(struct node*)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d",&val);
newnode->data=val;
if(start==NULL)
{
newnode->prev=NULL;
newnode->next=NULL;
start=newnode;
}
else
{
ptr=start;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->prev=ptr;
newnode->next=NULL;
}
}
return(start);
}
void display(struct node*start)
{
ptr=start;
if(start==NULL)
{printf("list empty");
}
else
{
while(ptr!=NULL)
{
```

```
printf("\n%d",ptr->data);
ptr=ptr->next;
}
}
}
struct node*ins_beg(struct node* start)
{
int val;
newnode=(struct node*)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d",&val);
newnode->data=val;
newnode->prev=NULL;
newnode->next=start;
start->prev=newnode;
start=newnode;
return start;
}
struct node*ins_end(struct node* start)
{
int val;
newnode=(struct node*)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d",&val);
newnode->data=val;
ptr=start;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->prev=ptr;
newnode->next=NULL;
return(start);
}
struct node*ins_bef(struct node* start)
{
int val, num;
ptr=start;
newnode=(struct node*)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d",&num);
printf("enter the node before which new element has to be
inserted\n");
scanf("%d",&val);
newnode->data=num;
/*insert before first node*/
if(ptr->data==val)
{
```

```
newnode->prev=NULL;
newnode->next=start;
start->prev=newnode;
start=newnode;
}
else
{
while(ptr->data!=val)
{
ptr=ptr->next;
}
temp=ptr->prev;
temp->next=newnode;
newnode->prev=temp;
ptr->prev=newnode;
newnode->next=ptr;

}
return start;
}
struct node*ins_aft(struct node* start)
{
int val, num;
ptr=start;
newnode=(struct node*)malloc(sizeof(struct node));
printf("enter the data");
scanf("%d",&num);
printf("enter the node after which new element has to be
inserted\n");
scanf("%d",&val);
newnode->data=num;
while(ptr->data!=val)
{
ptr=ptr->next;
}
temp=ptr->next;
ptr->next=newnode;
newnode->prev=ptr;
temp->prev=newnode;
newnode->next=temp;

return start;
}
struct node* del_beg(struct node* start)
{
ptr=start;
start=start->next;
start->prev=NULL;
free(ptr);
```

```
return start;
}
struct node* del_end(struct node* start)
{
ptr=start;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
temp=ptr->prev;
temp->next=NULL;
free(ptr);
return start;
}

struct node* del_par(struct node* start)
{
int val;
printf("enter the data");
scanf("%d",&val);
ptr=start;
/*deleting first node*/
if(ptr->data==val)
{
start=start->next;
start->prev=NULL;
free(ptr);
}
else
{
while(ptr->data!=val)
{
ptr=ptr->next;
}
temp=ptr->prev;
temp->next=ptr->next;
ptr->next->prev=temp;
free(ptr);
}
return start;
}
struct node*del_bef(struct node*start)
{
int val;
ptr=start;
printf("enter the data before which element has to delete");
scanf("%d",&val);
/*delete before first node*/
if(ptr->data==val)
```

```
{
printf("\n no element found");
}
else
{

while(ptr->data!=val)
{
ptr=ptr->next;
}
temp=ptr->prev;
/*deleting first node*/
if(temp==start)
{
start=start->next;
start->prev=NULL;

}
else
{
temp->prev->next=ptr;
ptr->prev=temp->prev;
}
}
free(temp);
return start;
}

struct node*del_aft(struct node*start)
{
int val;
ptr=start;
printf("enter the value after which element to be deleted\n");
scanf("%d",&val);
while(ptr->data!=val)
{
ptr=ptr->next;
}
if(ptr->next==NULL)
{
printf("\n no element found");
}
else
{
temp=ptr->next;
ptr->next=temp->next;
temp->next->prev=ptr;
}
free(temp);
```

```
return start;
}
```

Output:

```
9.delete a node
10.del before
11.del after
12.exit
enter your choice
1
enter the no3
enter the data22
enter the data33
enter the data44

1.create
2.dispaly
3.insert begin
4.insert end
5.insert before a node
6.insert after a node
7.delete begin
8.delete end
9.delete a node
10.del before
11.del after
12.exit
enter your choice
```

```
6.insert after a node
7.delete begin
8.delete end
9.delete a node
10.del before
11.del after
12.exit
enter your choice
3
enter the data11

1.create
2.dispaly
3.insert begin
4.insert end
5.insert before a node
6.insert after a node
7.delete begin
8.delete end
9.delete a node
10.del before
11.del after
12.exit
enter your choice
_
```

```
9.delete a node
10.del before
11.del after
12.exit
enter your choice
2

11
22
33
44
1.create
2.dispaly
3.insert begin
4.insert end
5.insert before a node
6.insert after a node
7.delete begin
8.delete end
9.delete a node
10.del before
11.del after
12.exit
enter your choice
```

```
10.del before
11.del after
12.exit
enter your choice
2

11
22
33
44
1.create
2.dispaly
3.insert begin
4.insert end
5.insert before a node
6.insert after a node
7.delete begin
8.delete end
9.delete a node
10.del before
11.del after
12.exit
enter your choice
9
enter the data33_
```

```
8.delete end
9.delete a node
10.del before
11.del after
12.exit
enter your choice
2

11
22
44
1.create
2.dispaly
3.insert begin
4.insert end
5.insert before a node
6.insert after a node
7.delete begin
8.delete end
9.delete a node
10.del before
11.del after
12.exit
enter your choice
12_
```

5. Write a program that implements stack operations using:
a) Arrays
b) Linked lists

a)Arrays

/*A **stack** is a linear data structure ,in which elements are inserted and removed according to the last-in first-out (LIFO) principle.Only one end present, which is known asTop Of the Stack(TOS). Two main operations in stack are: insert an item into the stack, known as push operation and delete an item out of the stack,called as pop operation*/

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
int st[MAX],top=-1;
void push(int st[],int val);
int pop(int st[]);
int peek(int st[]);
```

```
void display(int st[]);
void main()
{
int v,op;
clrscr();
do
{
printf("\n**** Main Menu****");
printf("\n 1.insert an element ");
printf("\n 2.delete an element");
printf("\n 3.peek value");
printf("\n 4.display");
printf("\n 5.exit");
printf("\n enter your option");
scanf("%d",&op);
switch(op)
{
case 1:printf("\n enter the new value to be inserted");
scanf("%d",&v);
push(st,v);
break;
case 2:v=pop(st);
if(v!=-1)
printf("\n value deleted from stack is %d",v);
break;
case 3:v=peek(st);
if(v!=-1)
printf("\n value at top is %d",v);
break;
case 4:display(st);
```

```
break;
default:exit(0);
     break;
}
}while(op!=5);
getch();
}
void push(int st[],int val)
{
if(top==MAX-1)
printf("stack overflow");
else
{
top=top+1;
st[top]=val;
}
}
int pop(int st[])
{
int a;
if(top==-1)
{
printf("stack underflow");
return -1;
}
else
{
a=st[top];
top=top-1;
return a;
```

```
}
}
int peek(int st[])
{
if(top==-1)
{
printf("stack is empty");
return -1;
}
else
return(st[top]);
}
void display(int st[])
{
int i;
if(top==-1)
printf("stack is empty");
else
{
for(i=top;i>=0;i--)
printf("\n%d",st[i]);
}
}
```

Output:

```
 1.insert an element
 2.delete an element
 3.peek value
 4.display
 5.exit
 enter your option3

 value at top is 20
**** Main Menu****
 1.insert an element
 2.delete an element
 3.peek value
 4.display
 5.exit
 enter your option4

20
10
**** Main Menu****
 1.insert an element
 2.delete an element
 3.peek value
 4.display
 5.exit
 enter your option_
```

```
**** Main Menu****
 1.insert an element
 2.delete an element
 3.peek value
 4.display
 5.exit
 enter your option2

 value deleted from stack is 20
**** Main Menu****
 1.insert an element
 2.delete an element
 3.peek value
 4.display
 5.exit
 enter your option4

10
**** Main Menu****
 1.insert an element
 2.delete an element
 3.peek value
 4.display
 5.exit
 enter your option_
```

b) Linked lists

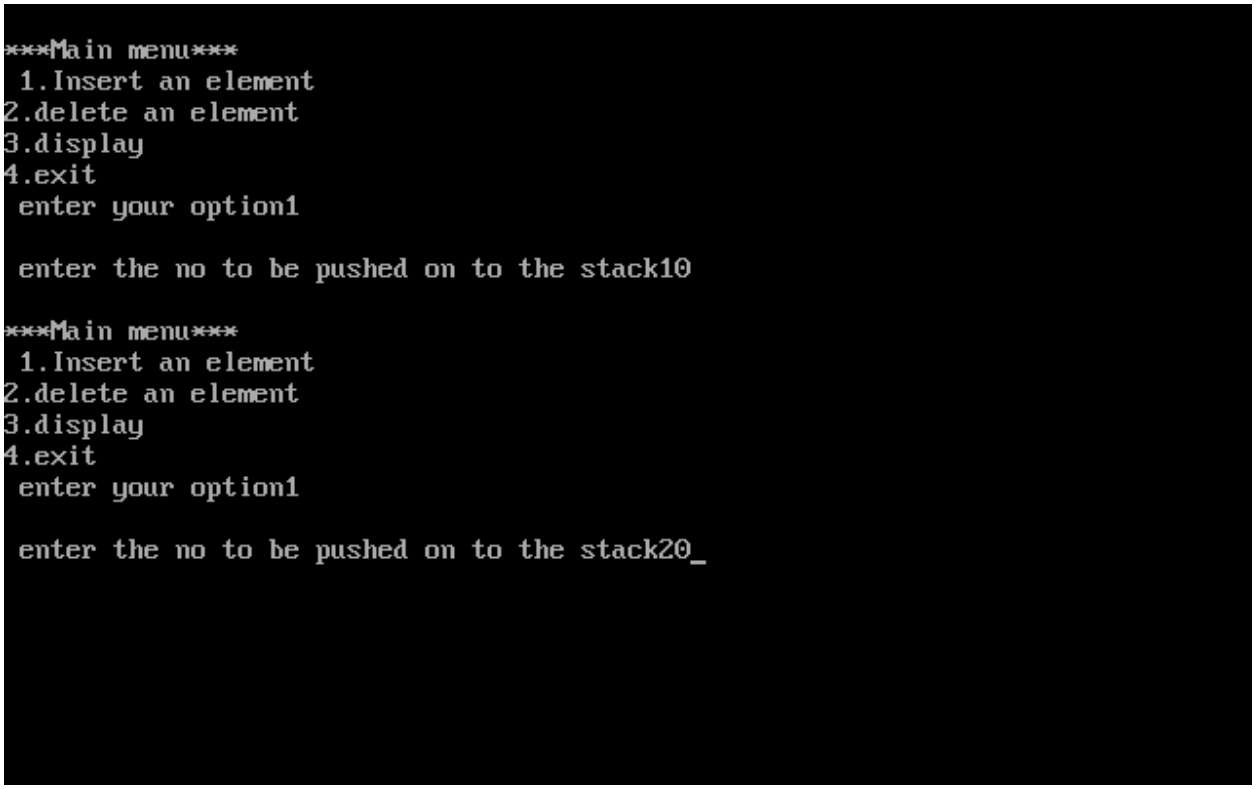
```
/* program that implements stack operations*/
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
struct stack
{
int data;
struct stack *next;
};
struct stack *top=NULL;
struct stack *push(struct stack *,int);
void display(struct stack *);
struct stack *pop(struct stack *);
void main()
```

```
{
int val,option;
clrscr();
do
{
printf("\n***Main menu***");
printf("\n 1.Insert an element \n2.delete an element \n3.display
\n4.exit");
printf("\n enter your option");
scanf("%d",&option);
switch(option)
{
case 1: printf("\n enter the no to be pushed on to the stack");
scanf("%d",&val);
top=push(top,val);
break;
case 2: top=pop(top);
break;
case 3: display(top);
break;
default:exit(0);
}}
while(option<=3);
getch();
}
struct stack *push(struct stack *top,int val)
{
struct stack *ptr;
ptr=(struct stack *)malloc(sizeof(struct stack));
ptr->data=val;
```

```
if(top==NULL)
{
ptr->next=NULL;
top=ptr;
}
else
{
ptr->next=top;
top=ptr;
}
return top;
}
struct stack *pop(struct stack *top)
{
struct stack *ptr;
ptr=top;
if(top==NULL)
printf("\n stack underflow");
else
{
top=top->next;
printf("\n the value being deleted is: %d",ptr->data);
free(ptr);
}
return top;
}
 void display(struct stack *top)
{
struct stack *ptr;
ptr=top;
```

```
if(top==NULL)
printf("Linked list is empty");
else
{
while(ptr!=NULL)
{
printf("\n %d",ptr->data);
ptr=ptr->next;
}}
}
```

Output:



```
***Main menu***
 1.Insert an element
2.delete an element
3.display
4.exit
 enter your option
3

 20
 10
***Main menu***
 1.Insert an element
2.delete an element
3.display
4.exit
 enter your option2

 the value being deleted is: 20
***Main menu***
 1.Insert an element
2.delete an element
3.display
4.exit
 enter your option_
```

```
 20
 10
***Main menu***
 1.Insert an element
2.delete an element
3.display
4.exit
 enter your option2

 the value being deleted is: 20
***Main menu***
 1.Insert an element
2.delete an element
3.display
4.exit
 enter your option3

 10
***Main menu***
 1.Insert an element
2.delete an element
3.display
4.exit
 enter your option4_
```

6. Write a program that uses Stack operations to:
a) Evaluate Postfix expression.

b) Convert infix expression into postfix expression

a)Evaluate Postfix expression.

/*In normal algebra we use the infix notation like a+b*c. The corresponding postfix notation is abc*+.stack data structure is used for evaluating postfix expression .scan the input and add operands to stack and whenever an operator comes pop first two elements from stack (first poped item is second operand)and perform the operation ,push the rerult back to stack.*/

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#define MAX 25
int st[MAX];
int top=-1;
void push(int st[],int val);
int pop(int st[]);
int post(char exp[]);
void main()
{
int val;
char exp[25];
clrscr();
printf("enter any post fix expression");
gets(exp);
val=post(exp);
printf("result is %d",val);
getch();
}
int post(char exp[])
{
int i=0;
int op1=0,op2=0;
while(exp[i]!='\0')
{
if(isdigit(exp[i]))
push(st,(exp[i]-'0'));
```

```
else
{
op2=pop(st);
op1=pop(st);
switch(exp[i])
{
case '+':push(st,op1+op2);
     break;
case '-':push(st,op1-op2);
     break;
case '*':push(st,op1*op2);
     break;
case '/':push(st,op1/op2);
     break;
case '%':push(st,op1%op2);
     break;
}
}
i++;
}
return(pop(st));
}
void push(int st[],int val)
{
if(top==MAX-1)
printf("stack overflow");
else
{
top=top+1;
st[top]=val;
}
}
int pop(int st[])
{
int a;
if(top==-1)
{
printf("stack underflow");
return -1;
```

```
}else
{
a=st[top];
top--;
}return a;
}
```

Output:

b) Convert infix expression into postfix expression

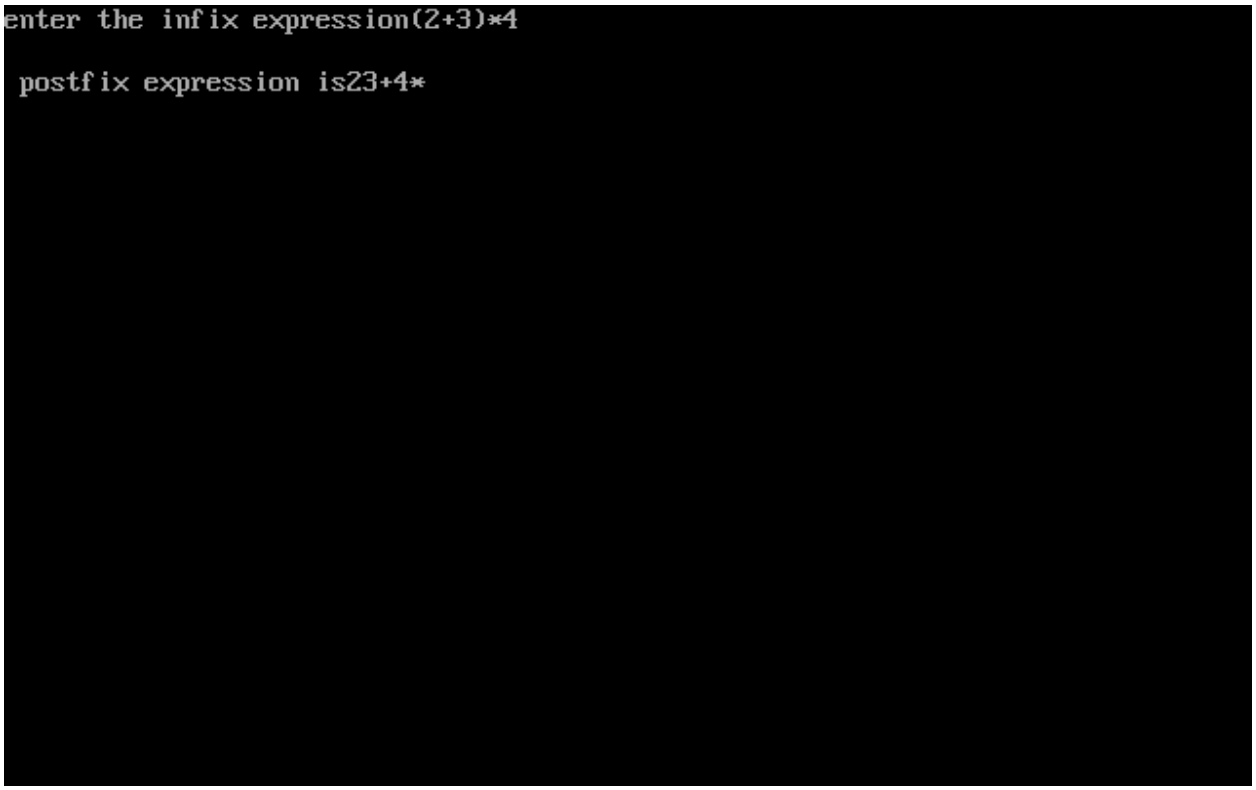
```
/*Scan the input and insert all the operands into an array and
operators in stack according to preceedense.ie if the element in
the stack top has less priority than incoming operator ,then
incoming operator can push to stack,otherwise pop the stack top
element and add it to array  then push the incoming operator to
stack*/

#include<stdio.h>
```

```
#include<conio.h>
#include<string.h>
#include<ctype.h>
#define MAX 25
char st[MAX];
int top=-1;
void push(char st[],char);
char pop(char st[]);
void con(char src[],char tar[]);
int pri(char);
void main()
{
char src[25],tar[25];
clrscr();
printf("enter the infix expression");
gets(src);
strcpy(tar," ");
con(src,tar);
printf("\n postfix expression is");
puts(tar);
getch();
}
void con(char src[],char tar[])
{
int i=0,j=0;
char t;
strcpy(tar," ");
while(src[i]!='\0')
{
if(src[i]=='(')
{
push(st,src[i]);
i++;
}
else if(src[i]==')')
{
while((top!=-1)&&(st[top]!='('))
{
tar[j]=pop(st);
```

```
j++;
}
if(top==-1)
{
printf("\n incorrect expression");
exit(1);
}
t=pop(st);
i++;
}
else if(isdigit(src[i])||isalpha(src[i]))
{
tar[j]=src[i];
j++;
i++;
}
else if(src[i]=='+'||src[i]=='-
'||src[i]=='*'||src[i]=='/'||src[i]=='%')
{
while((top!=-1)&&(st[top]!='(')&&(pri(st[top])>pri(src[i])))
{
tar[j]=pop(st);
j++;
}
push(st,src[i]);
i++;
}
else
{
printf("incorrect element");
exit(1);
}
}
while((top!=-1)&&(st[top]!='('))
{
tar[j]=pop(st);
j++;
}
tar[j]='\0';
}
```

```
int pri(char op)
{
if(op=='/'||op=='*'||op=='%')
return 1;
else if(op=='+'||op=='-')
return 0;
}
void push(char st[],char val)
{
if(top==MAX-1)
printf("stack overflow");
else
{
top=top+1;
st[top]=val;
}
}
char pop(char st[])
{
char a;
if(top==-1)
printf("stack underflow");
else
{
a=st[top];
top=top-1;
}
return a;
}
```

Output:



7. Write a program that implements Linear Queue operations using:

a) Arrays

b) Linked lists

/*Queue is linear data structure, somewhat similar to stack. In contrast to stack, queue is opened at both end called as rear end(for insertion)and front end(for deletion). inserting data into queue is known as enqueue operation and removing data from queue is called as dequeue. Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.*/

A)Arrays

```
/*implementation of linear queue using array*/

#include<stdio.h>
#include<conio.h>
#define MAX 20
int que[MAX];
int front=-1,rear=-1;
void insert();
int delete1();
void display();
void main()
{
int op,val;
clrscr();
do
{
printf("\n*** MAIN MENU***");
printf("\n 1.insert ");
printf("\n 2.delete");
printf("\n 3.display");
printf("\n 4.exit");
printf("\n enter your option:");
```

```
scanf("%d",&op);
switch(op)
{
case 1:insert();
break;
case 2:val=delete1();
if(val!=-1)
printf("The number deleted from the queue is:%d",val);
break;
case 3:display();
break;
case 4:exit(1);
}
}while(op!=4);
getch();
}
void insert()
{
int n;
printf("Enter the number to be inserted in the queue:");
scanf("%d",&n);
if(rear==MAX-1)
{
printf("queue overflow");
}
else if(front==-1&&rear==-1)
{
front=0;
rear=0;
que[rear]=n;
```

```
}
else
{
rear=rear+1;
que[rear]=n;
}
}
int delete1()
{
int val;
if(front==-1||front>rear)
{
printf("underflow condition");
return -1;
}
else
{
val=que[front];
front=front+1;
}
return val;
}
void display()
{
int i;
if(front==-1||front>rear)
{
printf("underflow");
}
else
```

```
{
printf("\n elements in the queue are\n");
for(i=front;i<=rear;i++)
{
printf("%d\n",que[i]);
}
}
}
```

Output:

```
 enter your option:1
Enter the number to be inserted in the queue:22

*** MAIN MENU***
 1.insert
 2.delete
 3.display
 4.exit
 enter your option:1
Enter the number to be inserted in the queue:45

*** MAIN MENU***
 1.insert
 2.delete
 3.display
 4.exit
 enter your option:1
Enter the number to be inserted in the queue:4

*** MAIN MENU***
 1.insert
 2.delete
 3.display
 4.exit
 enter your option:_
```

```

*** MAIN MENU***
 1.insert
 2.delete
 3.display
 4.exit
 enter your option:2
The number deleted from the queue is:22
*** MAIN MENU***
 1.insert
 2.delete
 3.display
 4.exit
 enter your option:3

 elements in the queue are
45
4

*** MAIN MENU***
 1.insert
 2.delete
 3.display
 4.exit
 enter your option:4_
```

b)linked list

```
/*implementation of linear queue using linked list*/

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*next;
}*front=NULL,*rear=NULL,*newnode,*ptr;
struct node*insert();
struct node* del();
void disp(struct node*);
void main()
{
int ch;
clrscr();
do
{
printf("\n1.insert");
printf("\t2.del");
printf("\t 3.disp");
printf("\t 4.exit");
printf("\nenter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:rear=insert();break;
case 2:front=del();break;
case 3:disp(front);break;
```

```
case 4:exit(0);
}
}while(ch<=4);
getch();
}
struct node*insert()
{
int val;
printf("enter the value");
scanf("%d",&val);
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=val;
if(rear==NULL)
{
newnode->next=NULL;
rear=front=newnode;
}
else
{
rear->next=newnode;
newnode->next=NULL;
rear=newnode;
}
return rear;
}
struct node*del()
{

ptr=front;
if(front==NULL)
```

```
printf("del not possible");
else
{
front=front->next;
printf("deleted item %d",ptr->data);
free(ptr);
}
return front;
}
void disp(struct node* front)
{
ptr=front;
if(front==NULL)
printf("empty");
else
{
while(ptr!=NULL)
{
printf("\n%d",ptr->data);
ptr=ptr->next;
}
}
}
```

Output:

```
1.insert        2.del   3.disp  4.exit
enter the choice1
enter the value33

1.insert        2.del   3.disp  4.exit
enter the choice1
enter the value22

1.insert        2.del   3.disp  4.exit
enter the choice1
enter the value44

1.insert        2.del   3.disp  4.exit
enter the choice3

33
22
44
1.insert        2.del   3.disp  4.exit
enter the choice
```

```
enter the choice1
enter the value22

1.insert        2.del   3.disp  4.exit
enter the choice1
enter the value44

1.insert        2.del   3.disp  4.exit
enter the choice3

33
22
44
1.insert        2.del   3.disp  4.exit
enter the choice2
deleted item 33
1.insert        2.del   3.disp  4.exit
enter the choice2
deleted item 22
1.insert        2.del   3.disp  4.exit
enter the choice3

44
1.insert        2.del   3.disp  4.exit
enter the choice4
```

8. Write a program that implements Circular Queue operations using Arrays

/***Circular queue*** is a linear data structure. It follows FIFO principle. In circular queue the last node is connected back to the first node to make a circle. Both the front and the rear pointers points to the beginning of the array. */

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
int q[MAX];
int front=-1,rear=-1;
void insert();
int delete1();
void display();
void main()
{
int op,val;
clrscr();
do
{
printf("\n*** MAIN MENU***");
printf("\n 1.insert");
printf("\t 2.delete");
printf("\t 3.display");
printf("\t 4.exit");
printf("\n Enter your option:");
scanf("%d",&op);
switch(op)
{
case 1:insert();
break;
case 2:val=delete1();
if(val!=-1)
printf("The value deleted  is: %d",val);
break;
case 3:display();
break;
default:exit(0);
}
}while(op!=4);
getch();
}
void insert()
{
```

```
int n;
printf("\n Enter the number to be inserted :");
scanf("%d",&n);
if((front==0)&&(rear==MAX-1)||(front==rear+1))
{
printf("overflow");
}
else if(front==-1&&rear==-1)
{
front=rear=0;
q[rear]=n;
}
else if((rear==MAX-1)&&(front!=0))
{
rear=0;
q[rear]=n;
}
else
{
rear=rear+1;
q[rear]=n;
}
}
int delete1()
{
int val;
if(front==-1)
{
printf("underflow");
return -1;
}
val=q[front];
if(front==rear)
{
front=-1;
rear=-1;
}
else if(front==MAX-1)
{
front=0;
}
else
{
front=front+1;
}
return val;
}
void display()
{
```

```
int i;
if(front==-1)
{
printf("underflow");
}
else
{
if(front<=rear)
{
for(i=front;i<=rear;i++)
{
printf("\n%d",q[i]);
}
}
else
{
for(i=front;i<=rear;i++)
printf("\n%d",q[i]);
}
}
}
```

Output:

```
 Enter the number to be inserted :22

*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:1

 Enter the number to be inserted :45

*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:1

 Enter the number to be inserted :67

*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:3

22
45
67
*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:
```

```
45
67
*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:2
The value deleted  is: 22
*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:2
The value deleted  is: 45
*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:1

 Enter the number to be inserted :89

*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:3

67
89
*** MAIN MENU***
 1.insert        2.delete        3.display       4.exit
 Enter your option:4
```

9. Write a program that implements Double-ended Queue operations using
a)Arrays

b)linked list

/*a **double-ended queue** (**dequeue**, often abbreviated to **deque**) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail).It is also often called a **head-tail linked list**, */

a) Using ARRAYS

```
#include<stdio.h>
#include<conio.h>
#define max 5
int n,ch,val,ar[20],l=-1,r=-1;
void in_l();
void in_r();
void disp();
void del_l();
void del_r();
void main()
{
clrscr();
do
{
printf("\n 1.input left");
printf("\n 2.input right");
printf("\n 3.disp");
printf("\n 4.exit");
printf("\n5.del_left");
printf("\n6.del_rt");
printf("\nenter choice");
scanf("%d",&ch);
switch(ch)
```

```
{
case 1:in_l();
break;
case 2:in_r();
break;
case 3:disp();
break;
case 4:exit(0);
case 5:del_l();
break;
case 6:del_r();
break;
}
}while(ch<=6);
getch();
}
void in_l()
{
int i=0;
printf("\n enter the value to be inserted");
scanf("%d",&val);
if(l==0 && r==max-1)
printf("\n insertion not possible");
else if(l==-1)
{
l=r=0;
ar[l]=val;
}
else
{
```

```
for(i=r;i>=l;i--)
{
ar[i+1]=ar[i];
}
ar[l]=val;
r=r+1;
}
}
void disp()
{
int i;
if(l==-1)
printf(" queue empty");
else
{
for(i=l;i<=r;i++)
printf("\n%d",ar[i]);
}
}
void in_r()
{
printf("\n enter the value to be inserted");
scanf("%d",&val);
if(r==max-1)
printf("\n insertion not possible");
else if(r==-1)
{
l=r=0;
ar[r]=val;
}
```

```
else
{
r=r+1;
ar[r]=val;
}
}
void del_l()
{
if(l==-1||l>r)
printf("\n deletion not possible");
else
{
val=ar[l];
printf("del elem:%d",val);
l=l+1;
}
}
void del_r()
{
if(l==-1||r<l)
printf("deletion not possible");
else
{
val=ar[r];
printf("\n deleted:%d",val);
r=r-1;
}
}
```

Output:

```
3.del_left
4.del_rt
5.disp
6.exit
enter choice1

 enter the value to be inserted22

1.input left
2.input right
3.del_left
4.del_rt
5.disp
6.exit
enter choice5

22
11
1.input left
2.input right
3.del_left
4.del_rt
5.disp
6.exit
enter choice
```

```
5.disp
6.exit
enter choice2

 enter the value to be inserted7

1.input left
2.input right
3.del_left
4.del_rt
5.disp
6.exit
enter choice5

22
11
6
7
1.input left
2.input right
3.del_left
4.del_rt
5.disp
6.exit
enter choice_
```

b)using linked lists

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
struct node
{
  int data;
  struct node *link;
};
struct dqueue
{
 struct node *front;
 struct node *rear;
}
void initdqueue(struct dqueue *);
void addatend(struct dqueue *,int item);
void addatbeg(struct dqueue *,int item);
void delatend(struct dqueue *);
void delatbeg(struct dqueue *);
void display(struct dqueue *);
int count(struct dqueue);
void delqueue(struct dqueue *);
void main()
{
 struct dqueue dq;
int i,n;
clrscr();
initdqueue(&dq);
addatend(&dq,11);
addatbeg(&dq,10);
```

```
addatend(&dq,12);
addatbeg(&dq,9);
addated(&dq,13);
addatbeg(&dq,8);
addatend(&dq,14);
addatbeg(&dq,7);
display(dq);
n=count(dq);
printf("\ntotal elements:%d",n);
i=delatbeg(&dq);
printf("\nitem extracted:%d",i);
i=delatbeg(&dq);
printf("\nitem extracted:%d",i);
i=delatbeg(&dq);
printf("\nitem extracted:%d",i);
i=delatend(&dq);
printf("\nitem extracted:%d",i);
display(dq);
n=count(dq);
printf("\nelements left:%d",n);
delqueue(&dq);
getch();
}
void initdqueue(struct dqueue *p)
{
p->front=p->rear=NULL:
}
void addatend(struct dqueue *p,int item)
{
struct node *temp;
```

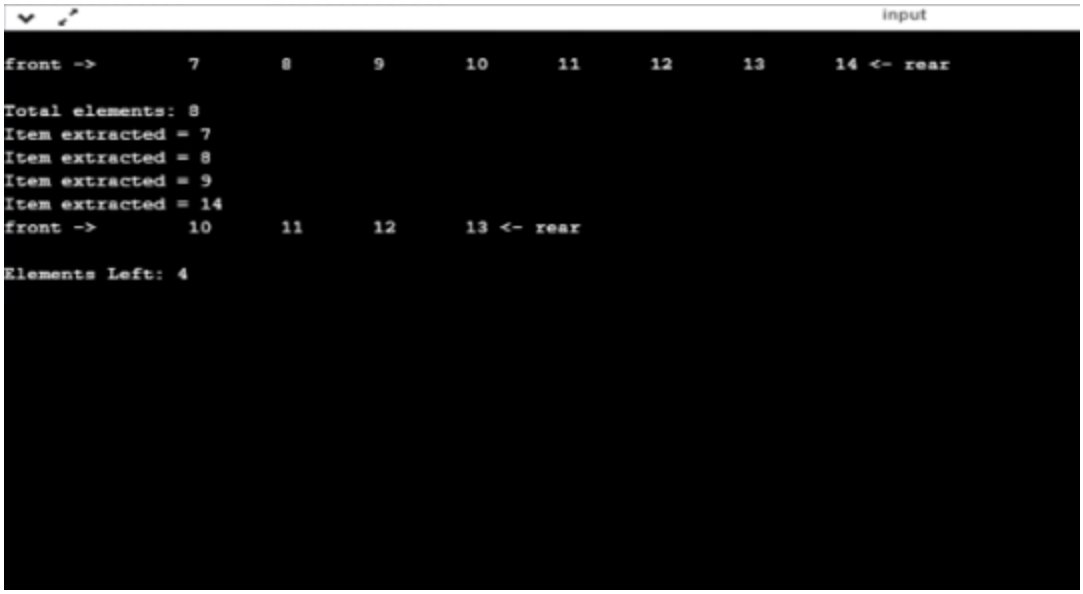
```
temp=(struct node *)malloc(sizeof(struct node));
temp->data=item;
temp->link=NULL;
if(p->front==NULL)
p->front=temp;
else
{
 p->rear->link=temp;
 p->rear=temp;
}
void addatbeg(struct node *p,int item)
{
 struct node *temp;
 int *q;
 temp=(struct node *)malloc(sizeof(struct node));
 temp->data=item;
 temp->link=NULL;
 if(p->front==NULL)
 p->front=p->rear=temp;
 else
 {
temp->link=p->front;
  p->front=temp;
 }
}
int delatbeg(struct dqueue *p)
{
 struct node *temp=p->front;
 int item;
 if(temp==NULL)
```

```
 {
  printf("\nQueue is empty");
  return(0);
 }
else
{
 temp=p->front;
 item=temp->data;
 p->front=temp->link;
 free(temp);
 if(temp==NULL)
 p->rear=NULL:
 return(item);
}
}
int delatend(struct dqueue *p)
{
struct node *temp,*rleft,*q;
int item;
temp=p->front;
if(p->rear==NULL)
{
 printf("\nqueue is empty");
 return 0;
}
else
{
 while(temp!=p=>rear)
 {
  rleft=temp;
```

```
  temp=temp->link;
 }
q=p->rear;
item=q->data;
free(q);
p->rear=rleft;
p->rear->link=NULL;
if(p->rear==NULL)
p->front=NULL;
return(item);
}
}
void display(struct dqueue dq)
{
struct node *temp=dq.front;
 printf("\nfront->");
 while(temp!=NULL)
 {
  if(temp->link===NULL)
  {
   printf("\t%d",temp->data);
   printf("<-rear");
  }
  else
  printf("\t%d",temp->data);
  temp=temp->link;
 }
 printf("\n");
}
int count(struct dqueue dq)
```

```
{
 int c=0;
 struct node *temp=dq.front;
 while(temp!=NULL)
 {
  temp=temp->link;
  c++;
 }
 return c;
}
void delqueue(struct dqueue *p)
{
  struct node *temp;
  if(p->front==NULL)
  return;
  while(p->front!=NULL)
  {
   temp=p->front;
   p->front=p->front->link;
   free(temp);
  }
}
```

OUTPUT


input
front -> 7 8 9 10 11 12 13 14 <- rear
Total elements: 8
Item extracted = 7
Item extracted = 8
Item extracted = 9
Item extracted = 14
front -> 10 11 12 13 <- rear
Elements Left: 4

10) Write a recursive program to create a Binary Tree of integers, traverse the tree in preorder, in order and post order and also print the number of leaf nodes and height of the tree.

/***Binary tree*** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.Displaying nodes or tree can be done using traversals*/

/*recursive implementation*/

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
int count=1;
 struct node
{
 int data;
 struct node*left;
 struct node*right;
}*root;
struct node *insert(struct node*,int);
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);
int LeatCount(struct node *);
int height(struct node*);
int height(struct node* root)
{
int lh,rh;
   if (root==NULL)
       return 0;
```

```
   else
   {
       /* compute the depth of each subtree */
  lh = height(root->left);
  rh = height(root->right);

       /* use the larger one */
       if (lh > rh)
     return(lh+1);
       else return(rh+1);
   }
}
int LeafCount(struct node *root)
{
  if(root==NULL)
    return 0;
  if(root->left == NULL && root->right==NULL)
    return 1;
  else
    return LeafCount(root->left)+
     LeafCount(root->right);
}

struct node *insert(struct node * root,int n)
{
 if(root==NULL)
 {
  root=( struct node*)malloc(sizeof(struct node));
  root->data=n;
  root->left=NULL;
```

```
  root->right=NULL;
  count++;
 }
 else
 {
 if(count%2==0)
 root->left=insert((root->left),n);
 else
 root->right=insert((root->right),n);
 }
 return root;
}

//traverse the tree in inorder
void inorder( struct node*root)
{
 if(root!=NULL)
 {
  inorder(root->left);
  printf("%d\t",root->data);
  inorder(root->right);
 }
}

//traverse the tree in preorder
void preorder(struct node*root)
{
 if(root!=NULL)
 {
  printf("%d\t",root->data);
```

```
  preorder(root->left);
  preorder(root->right);
 }
}

//traverse the tree in postorder
void postorder(struct node*root)
{
 if(root!=NULL)
 {
  postorder(root->left);
  postorder(root->right);
  printf("%d\t",root->data);

 }
}

void main()
{
int n,ch,c,h;
clrscr();
do
{
printf("\n 1.insert \t 2.inorder\t
3.preoreder\t4.postoredr\t5.leafcount\t 6.height \t7.exit\n");
printf("enter choice\n");
scanf("%d",&ch);
switch(ch)
 {
 case 1:printf("enter the number");
```

```
        scanf("%d",&n);
        root=insert(root,n);
        break;
      case 2: inorder(root);
        break;
      case 3:preorder(root);
        break;
      case 4:postorder(root);
        break;
      case 5:c=LeafCount(root);
        printf("\n %d leaf nodes  present",c);
        break;
      case 6:h=height(root);
        printf("\n %d is the height of the tree",h);
        break;

      case 7:exit(0);
      }
      }while(ch<=7);
      getch();
      }
```

OUTPUT

```
 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
1
enter the number22

 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
1
enter the number33

 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
1
enter the number55

 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
1
enter the number66_
```

```
enter choice
1
enter the number66

 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
2
66      33      22      55
 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
5

 2 leaf nodes  present
 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
6

 3 is the height of the tree
 1.insert        2.inorder       3.preoreder     4.postoredr     5.leafcount
 6.height       7.exit
enter choice
7
```

11) Write a program for Binary Search tree operations

/*A binary search tree (BST) is a tree in which all nodes follows the below mentioned properties

- The left sub-tree of a node has key less than or equal to its parent node's key.
- The right sub-tree of a node has key greater than or equal to its parent node's key.*/

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
typedef struct node
{
    int data;
    struct node *left, *right;
}Node;
Node *root=NULL,*ptr,*tpar,*newnode,*temp,*parent;
void inorder(struct node *);
void post(struct node *);
void pre(struct node*);
Node* insert(Node*,int);
Node* del(Node*,int);
Node* mini(Node*);
Node *delb(Node*);
Node* search(Node *,int);
void inorder(Node *root)
{
    if (root != NULL)
    {
       inorder(root->left);
       printf("\t%d",root->data);
       inorder(root->right);
    }
```

```
}
void post(Node *root)
{
    if (root != NULL)
    {
       post(root->left);
       post(root->right);
       printf("\t%d",root->data);
    }
}
 void pre(Node *root)
{
    if (root != NULL)
    {

       printf("\t%d",root->data);
       pre(root->left);
       pre(root->right);
    }
}
Node* insert(Node* root,int val)
{
  newnode=(Node *)malloc (sizeof(Node));
  newnode->data=val;
  newnode->left=newnode->right=NULL;
    if (root == NULL)
    root=newnode;
else
{
```

```
    if (val < root->data)
       root->left= insert(root->left,val);
    else
       root->right= insert(root->right,val);
 }
   return root;
}
Node * mini(Node* ptr)
{      while (ptr->left != NULL)
       {
       tpar=ptr;
       ptr = ptr->left;
       }
   return ptr;
}
Node* del(Node* root, int key)
{
    ptr=search(root,key);
    if(ptr->left==NULL &&ptr->right==NULL)
    {  if(parent==NULL)
       {free(ptr);
       printf("tree empty");
       }
       else
       {
       if(parent==root)
       free(ptr);
       else
       {
       if(parent->left==ptr)
```

```
      parent->left=NULL;
      else
      parent->right=NULL;
      }
      }
      free(ptr);
    }
  else if(ptr->left==NULL && ptr->right!=NULL)
{
if(parent==NULL)
    {root=ptr->right;
    return root;
    }
    else
    {
    if(parent->left==ptr)

         parent->left=ptr->right;
      else
           parent->right=ptr->right;
     }
  free(ptr);
}
  else if (ptr->right==NULL && ptr->left!=NULL)
  { if(parent==NULL)
      {
      root=ptr->left;
      return root;
      }
      else
```

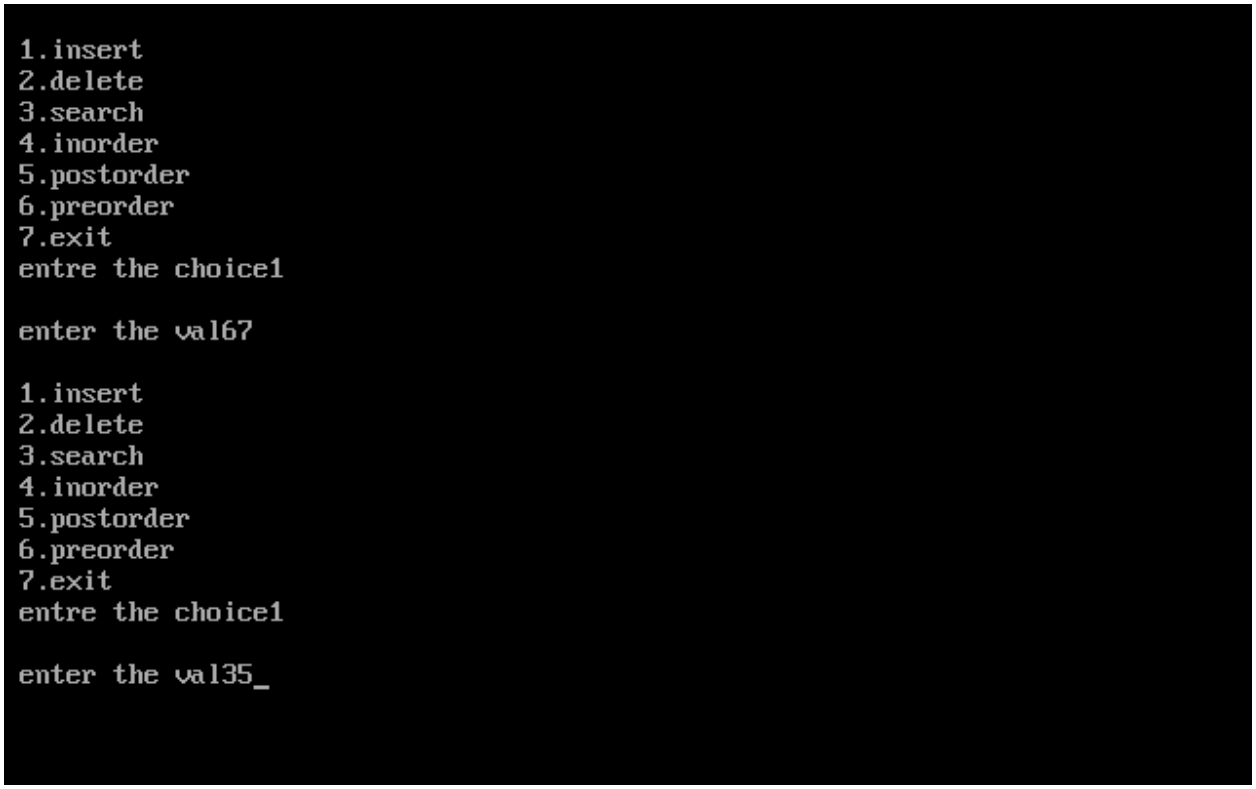
```
        {

      if(parent->left==ptr)
            parent->left=ptr->left;
         else
            parent->right=ptr->left;
        }
     free(ptr);
    }

     else
     {
     root=delb(ptr);
     }
     return root;
  } //end of fun

     Node *delb(Node *ptr)
     {
     temp=mini(ptr->right); //finding inorder successor
      ptr->data=temp->data;
      tpar->left=temp->right;
      free (temp);
      if(parent==NULL)
      return ptr;
      else
      return root;

     }
void  main()
```

```
{
int ch,val;
clrscr();
do
{
printf("\n 1.insert");
printf("\n 2.delete");
printf("\n 3.search");
printf("\n 4.inorder");
printf("\n 5.postorder");
printf("\n 6.preorder");
printf("\n 7.exit");
printf("\n entre the choice");
scanf("%d",&ch);
switch(ch)
{

case 1:printf("\n enter the val");
       scanf("%d",&val);
       root=insert(root,val);
       break;
case 2: printf("\n enter the val");
       scanf("%d",&val);
       root=del(root,val);
       break;
case 3:printf("\n enter the val");
       scanf("%d",&val);
       parent=NULL;
       ptr=search(root,val);
       if(ptr!=NULL)
```

```
       printf("\n successful");
       else
       printf("search is unsuccessful");
       break;
case 4:inorder(root);
       break;
case 5:post(root);
       break;
case 6:pre(root);
       break;
case 7:exit(0);
}
}while(ch<=7);
getch();
}
Node* search(Node*ptr,int k)
{
if(ptr==NULL)
return NULL;
else if(ptr->data==k)
return ptr;
else
{
if(k<ptr->data)
{
parent=ptr;
return search(ptr->left,k);
}
else
{
```

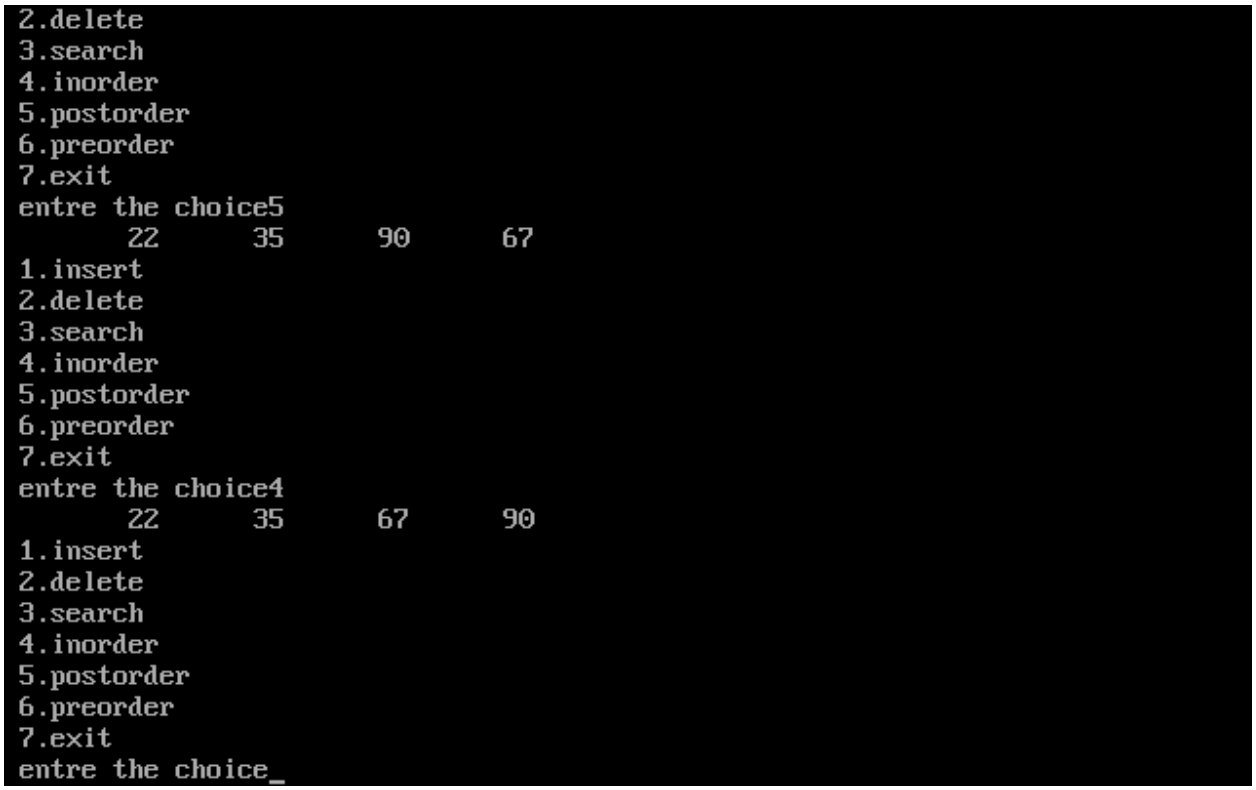
```
parent=ptr;
return search(ptr->right,k);
}}}
```

Output



```
6.preorder
7.exit
entre the choice1

enter the val90

1.insert
2.delete
3.search
4.inorder
5.postorder
6.preorder
7.exit
entre the choice1

enter the val22

1.insert
2.delete
3.search
4.inorder
5.postorder
6.preorder
7.exit
entre the choice_
```

```
7.exit
entre the choice1

enter the val22

1.insert
2.delete
3.search
4.inorder
5.postorder
6.preorder
7.exit
entre the choice3

enter the val90

successful
1.insert
2.delete
3.search
4.inorder
5.postorder
6.preorder
7.exit
entre the choice6
```



12) Write a program for implementing the following graph traversal algorithms:

a) Depth First Traversal (DFT)

b) Breadth First Traversal (BFT)

/*a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices.

Depth First Search algorithm(DFS) traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search when a dead end occurs in any iteration.

Breadth First Search algorithm(BFS) traverses a graph in a breadthwards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.*/

a) Depth First Traversal (DFT)

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

```
#include<conio.h>
void create(int);
void display(int);

void DFS(int,int);
int n,v,a[10][10]={0},visit[10];

void main()
{
  int ch;
  clrscr();
   printf("enter the no. of vertices\n");
            scanf("%d",&n);
do
  {
    printf("1-create\n2-display\n3-DFS\n4-exit\n");
    printf("enter your choice\n");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:
            create(n);
            break;
      case 2:  display(n);
            break;

     case 3:   printf("enter the starting\n");
            scanf("%d",&v);
            DFS(v,n);
            break;
      case 4:  exit(1);
      default: printf("invalid choice\n");
    }
    }while(ch<=4);

    getch();

}

void create(int n)
```

```
{
  int i,c,m,s,d;
  printf("1=directed\t2=undirected\n");
  scanf("%d",&c);
  if(c==1)
    m=n*(n-1);
  else
    m=n*(n-1)/2;
  for(i=1;i<=m;i++)
  {
    printf("enter 2 numbers between which edge should be
present\n");
    scanf("%d%d",&s,&d);
    if(s==99 && d==99)
      break;
    else
    {
      a[s][d]=1;
      if(c==2)
     a[d][s]=1;
    }
  }
}

void display(int n)
{
  int i,j;
  printf("\t");
  for(j=1;j<=n;j++)
    printf("%d\t",j);
  printf("\n");
  for(i=1;i<=n;i++)
  {
    printf("%d\t",i);
    for(j=1;j<=n;j++)
      printf("%d\t",a[i][j]);
    printf("\n");
  }
}
```

```
void DFS(int v,int n)
{
  int i,top=-1,s[10];
  for(i=1;i<=n;i++)
    visit[i]=0;
  top++;
  s[top]=v;
  while(top>=0)
  {
    v=s[top];
    top--;
    if(visit[v]==0)
    {
      printf("%d\t",v);
      visit[v]=1;
    }
    else
      continue;
    for(i=n;i>=1;i--)
    {
      if(a[v][i]==1 && visit[i]==0)
      {
     top++;
     s[top]=i;
      }
    }
  }
}
```

Output:

```
enter the no. of vertices
4
1-create
2-display
3-DFS
4-exit
enter your choice
1
1=directed      2=undirected
2
enter 2 numbers between which edge should be present
1 2
enter 2 numbers between which edge should be present
2 3
enter 2 numbers between which edge should be present
3 4
enter 2 numbers between which edge should be present
99 99
1-create
2-display
3-DFS
4-exit
enter your choice


1-create
2-display
3-DFS
4-exit
enter your choice
2
       1       2       3       4
1      0       1       0       0
2      1       0       1       0
3      0       1       0       1
4      0       0       1       0
1-create
2-display
3-DFS
4-exit
enter your choice
3
enter the starting
1
1       2       3       4       1-create
2-display
3-DFS
4-exit
enter your choice
4
```

b)Breadth First Traversal (BFT)

/*creation of graph using adjacency matrix and perform BFS traversal*/

```
#include<stdio.h>
#include<conio.h>
void adjm(int);
void bfs(int,int);
int ad[10][10]={0,0};
int vis[10];
void main()
{
int v,n,i;
clrscr();
printf("entr the no:of nodes");
scanf("%d",&n);
adjm(n);
printf("\nenter the starting node");
scanf("%d",&v);
for(i=1;i<=n;i++)
vis[i]=0;
bfs(v,n);
getch();
}
void bfs(int v,int n)
{
int f=0,r=0,i,q[10];
printf("\n%d",v);
vis[v]=1;
r++;
f++;
q[r]=v;
while(f<=r)
{

v=q[f];
f++;
for(i=1;i<=n;i++)
  {
 if(ad[v][i]==1&&vis[i]==0)
    {
    printf("\n%d",i);
    vis[i]=1;
    r++;
    q[r]=i;
    }
  }
}
    }


void adjm(int n)
{
```

```
int max,s,d,i,j;
int gt;
clrscr();

//fflush(stdin);
printf("type of graph(d/u)");
scanf("%d",&gt);
if(gt==0)
max=n*(n-1)/2;
else
max=n*(n-1);
for(i=1;i<=max;i++)
{
printf("enter the edjes between source and dest");
scanf("%d%d",&s,&d);
if(s==99 && d==99)
break;
else
{
ad[s][d]=1;
if(gt==0)
ad[d][s]=1;
}
}
printf("\n the adj matrix are");
for(i=1;i<=n;i++)
{
printf("\n");
for(j=1;j<=n;j++)
{
printf("\t%d",ad[i][j]);
}
}
}
```

output:

```
type of graph(d/u)0
enter the edjes between source and dest1 2
enter the edjes between source and dest1 4
enter the edjes between source and dest3 4
enter the edjes between source and dest99 99

 the adj matrix are
        0       1       0       1
        1       0       0       0
        0       0       0       1
        1       0       1       0
enter the starting node1

1
2
4
3_
```