VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

DATA STRUCTURE LAB RECORD

Submitted by

SANNIDHI KASTURI (1BM19CS143)

Under the Guidance of

Prof. LOHITH J.J.
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

Sep-2020 to Jan-2021

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the LAB RECORD carried out by **SANNIDHI KASTURI (1BM19CS143)** who is the bona fide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraiah Technological University, Belgaum during the year 2020-2021. The lab report has been approved as it satisfies the academic requirements in respect of **DATA STRUCTURE LAB RECORD (19CS3PCDST)** work prescribed for the said degree.

Signature of the Guide
Prof. Prof. Lohith J.J.
Assistant Professor
BMSCE, Bengaluru

Signature of the HOD
Dr. Umadevi V
Associate Prof.& Head, Dept. of CSE
BMSCE, Bengaluru

External Viva

Name of the Examiner — Signature with date

1._____ _____

2. _____ _____

Lab Program - 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 and stack empty.

```
#include<stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
int top=-1;
int s[10];
int item;
void push()
{
if(top == STACK_SIZE - 1)
{
printf("Stack Overflow \n");
return;
}
top = top +1;
s[top] = item;
}
int pop()
{
if(top == - 1)
return - 1;
return s[top--];
}
void display()
{
int i;
if(top == - 1)
```

```
{
printf("Stack is empty \n");
return;
}
printf("Contents of the stack are: \n");
for(i=top;i>=0;i--)
{
printf("%d \n", s[i]);
}
}
int main()
{
int item_deleted, choice;
for(;;)
{
printf("\n1:Push \n2:Pop \n3:Display \n4:Exit \n");
printf("Enter the choice : \n");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("Enter the item to be inserted \n");
scanf("%d", &item);
push();
break;
case 2: item_deleted = pop() ;
if(item_deleted == - 1)
printf("Stack is empty \n");
else
printf("Item deleted is %d \n", item_deleted);  break;
case 3: display();
break;
default:exit(0);
}
}
return 0;
}
```

```
clang-7 -pthread -lm -o main main.c
./main

1:Push
2:Pop
3:Display
4:Exit
Enter the choice :
1
Enter the item to be inserted
10

1:Push
2:Pop
3:Display
4:Exit
Enter the choice :
1
Enter the item to be inserted
20

1:Push
2:Pop
3:Display
4:Exit
Enter the choice :
2
Item deleted is 20

1:Push
2:Pop
3:Display
4:Exit
Enter the choice :
3
```

```
Contents of the stack are:
10

1:Push
2:Pop
3:Display
4:Exit
Enter the choice :
4
```

Lab Program - 2

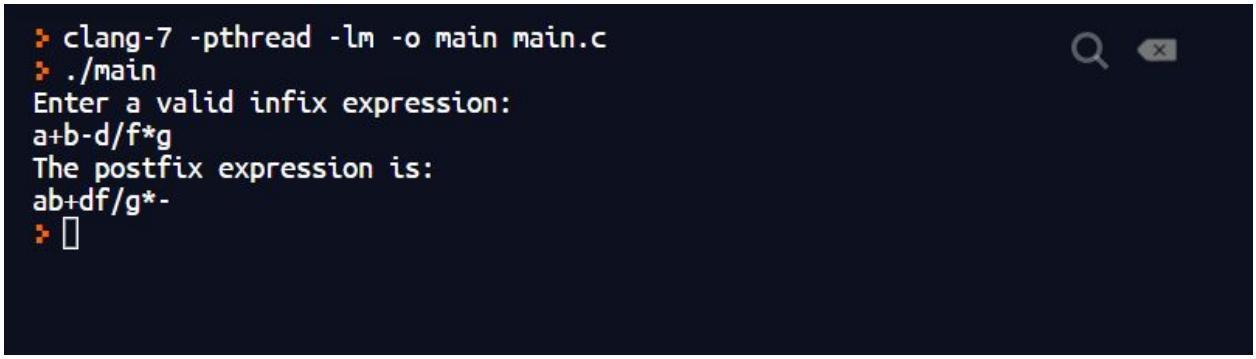
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + , - , * and /

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int fun1(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 fun2(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,j,i;
char s[30];
char symbol;
top=-1;
s[++top]='#';
j=0;
for(i=0;i<strlen(infix);i++)
{
symbol=infix[i];
while(fun1(s[top])>fun2(symbol)) {
postfix[j]=s[top--];
j++;
}
```

```
if(fun1(s[top])!=fun2(symbol))
{
s[++top]=symbol;
}
else
top--;
}
while(s[top]!='#')
{
postfix[j++]=s[top--];
}
postfix[j]='\0';
}
int main()
{
char infix[20],postfix[20];
int a=0,b=0,k;
printf("Enter a valid infix expression:\n"); scanf("%s",infix);
for(k=0;k<strlen(infix);k++)
{
if(infix[k]=='(')
a++;
else if(infix[k]==')')
b++;
else
continue;
```

```
}
if(a!=b)
{
printf("Invalid infix expression"); exit(0);
}
infix_postfix(infix,postfix);
printf("The postfix expression is: \n");
printf("%s\n",postfix);
return 0;
}
```



Lab Program - 3

Write a program 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>

#include<stdlib.h>

#define queue_size 3

int item,front=0,rear=-1,q[10];

void insertrear()

{

if(rear==queue_size-1)

{

printf("Queue overflow.\n");

return;

}

rear=rear+1;

q[rear]=item;

}

int delfront()

{

if(front>rear)
```

```
{
front=0;
rear=-1;
return -1;
}
return q[front++];
}
void display()
{
int i;
if(front>rear)
{
printf("Queue is empty.\n");
return;
}
printf("Contents of queue are : \n");
for(i=front;i<=rear;i++)
{
printf("%d\n",q[i]);
}
}
int main()
{
int choice;
for(;;)
{
```

```
printf("\n1:Insertrear \n2:Deletefront \n3:Display \n4:Exit\n");
printf("Enter the choice :");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("Enter the item to be inserted:\n");

scanf("%d",&item);

insertrear();

break;

case 2:item=delfront();

if(item==-1)

printf("Queue is empty.\n");  else

printf("Item deleted is %d\n",item);  break;

case 3:display();

break;

default:exit(0);

}

}

return 0;

}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main

1:Insertrear
2:Deletefront
3:Display
4:Exit
Enter the choice :1
Enter the item to be inserted:
10

1:Insertrear
2:Deletefront
3:Display
4:Exit
Enter the choice :1
Enter the item to be inserted:
20

1:Insertrear
2:Deletefront
3:Display
4:Exit
Enter the choice :3
Contents of queue are :
10
20

1:Insertrear
2:Deletefront
3:Display
4:Exit
Enter the choice :2
Item deleted is 10
```

Lab Program - 4

Write a program 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>
#include<stdlib.h>
#define queue_size 3
int item, front = 0, rear = -1, q[queue_size], count = 0;
void insertrear ()
{
if (count == queue_size)
{
printf ("Queue overflow.");
return;
}
rear = (rear + 1) % queue_size;
q[rear] = item;
count++;
}
int deletefront ()
{
if (count == 0)
return -1;
item = q[front];
front = (front + 1) % queue_size;
count = count - 1;
return item;
}
void display ()
{
int i, f;
```

```
if (count == 0)
{
printf ("The queue is empty.");
return;
}
f = front;
printf ("Contents of the queue are : \n");
for (i = 0; i <= count; i++)
{
printf ("%d\n", q[f]);
f = (f + 1) % queue_size;
}
}
int main ()
{
int choice;
for (;;)
{
printf ("\n1.Insert rear \n2.Delete front \n3.Display \n4.Exit \n");
printf ("Enter the choice : ");
scanf ("%d", &choice);
switch (choice)
{
case 1:
printf ("Enter the item to be inserted :");  scanf ("%d", &item);
insertrear ();
break;
case 2:
item = deletefront ();
if (item == -1)
printf ("Queue is empty.\n");
else
printf ("Item deleted is %d.\n", item);  break;
case 3:
display ();
break;
default:
exit (0);
```

```
}
}
return 0;
}
```

```
clang-7 -pthread -lm -o main main.c
./main

1.Insert rear
2.Delete front
3.Display
4.Exit
Enter the choice : 1
Enter the item to be inserted :10

1.Insert rear
2.Delete front
3.Display
4.Exit
Enter the choice : 1
Enter the item to be inserted :20

1.Insert rear
2.Delete front
3.Display
4.Exit
Enter the choice : 3
Contents of the queue are :
10
20
0

1.Insert rear
2.Delete front
3.Display
4.Exit
Enter the choice : 2
Item deleted is 10.
```

Lab Programs - 5 & 6

Write a program to implement a Singly Linked List with following operations:

a) Create a linked list
b) Insertion of a node at first position, at any position and at end of the list
c) Deletion of first element, specified element and last element in the list
d) Display the contents of the linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head;
void beg_insert ()
{
struct node *ptr;
int item;
ptr = (struct node *) malloc (sizeof (struct node));
if (ptr == NULL)
{
printf ("\nList overflow");
}
else
{
printf ("\nEnter the value : ");
scanf ("%d", &item);
ptr->data = item;
ptr->next = head;
head = ptr;
printf ("\nNode is inserted at the front.");  }
}
```

```
void end_insert ()
{
struct node *ptr, *temp;
int item;
ptr = (struct node *) malloc (sizeof (struct node));  if (ptr == NULL)
{
printf ("\nList overflow");
}
else
{
printf ("\nEnter the value : ");
scanf ("%d", &item);
ptr->data = item;
if (head == NULL)
{
ptr->next = NULL;
head = ptr;
printf ("\nNode is inserted.");
}
else
{
temp = head;
while (temp->next != NULL)
{
temp = temp->next;
}
temp->next = ptr;
ptr->next = NULL;
printf ("\nNode is inserted at the end.");
}
}
}
void random_insert ()
{
int i, loc, item;
struct node *ptr, *temp;
ptr = (struct node *) malloc (sizeof (struct node));
if (ptr == NULL)
```

```
{
printf ("\nList overflow");
}
else
{
printf ("\nEnter the value : ");
scanf ("%d", &item);
ptr->data = item;
printf("\nEnter the location of the node after which you want to
insert : ");  scanf ("%d", &loc);
temp = head;
for (i = 0; i < loc; i++)
{
temp = temp->next;
if (temp == NULL)
{
printf ("\nCant insert node.");
return;
}
}
ptr->next = temp->next;
temp->next = ptr;
printf ("\nNode is inserted at the specified position.");  }
}
void beg_delete ()
{
struct node *ptr;
if (head == NULL)
{
printf ("\nList is empty");
}
else
{
ptr = head;
head = ptr->next;
free (ptr);
printf ("\nNode is deleted from the front.");
}
```

```
}
void end_delete ()
{
struct node *ptr, *ptr1;
if (head == NULL)
{
printf ("\nList is empty");
}
else if (head->next == NULL)
{
head = NULL;
free (head);
printf ("\nThe only node of the list is deleted.");  }
else
{
ptr = head;
while (ptr->next != NULL)
{
ptr1 = ptr;
ptr = ptr->next;
}
ptr1->next = NULL;
free (ptr);
printf ("\nNode is deleted from the end.");
}
}
void random_delete ()
{
struct node *ptr, *ptr1;
int loc, i;
printf("\nEnter the location of the node after which you want to
perform deletion : ");
scanf ("%d", &loc);
ptr = head;
for (i = 0; i < loc; i++)
{
ptr1 = ptr;
ptr = ptr->next;
```

```
if (ptr == NULL)
{
printf ("\nCant delete");
return;
}
}
ptr1->next = ptr->next;
free (ptr);
printf ("\nNode is deleted from the specified position %d", loc + 1);
}
void display ()
{
struct node *ptr;
ptr = head;
if (ptr == NULL)
{
printf ("\nNothing to print. List is empty.");
}
else
{
printf ("\nList values are : ");
while (ptr != NULL)
{
printf ("\n%d", ptr->data);
ptr = ptr->next;
}
}
}
int main ()
{
int choice = 0;
while (choice != 8)
{
printf ("\nChoose an option");
printf("\n1.Insert at front \n2.Insert at end \n3.Insert at specified
position \n4.Delete from front \n5.Delete from end \n6.Delete from
specified position \n7.Display list contents \n8.Exit");
printf ("\nEnter your choice : ");
```

```
scanf ("%d", &choice);
switch (choice)
{
case 1:beg_insert ();
break;
case 2:end_insert ();
break;
case 3:random_insert ();
break;
case 4:beg_delete ();  break;
case 5:end_delete ();  break;
case 6:random_delete ();  break;
case 7:display ();  break;
case 8:exit (0);
default:exit (0);
}
}
return 0;
}
```

```
clang-7 -pthread -lm -o main main.c
./main

Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 1

Enter the value : 10

Node is inserted at the front.
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 1

Enter the value : 20

Node is inserted at the front.
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
```

```
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 2

Enter the value : 9

Node is inserted at the end.
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 3

Enter the value : 30

Enter the location of the node after which you want to insert : 2

Node is inserted at the specified position.
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
```

```
Enter your choice : 4

Node is deleted from the front.
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 7

List values are :
10
9
30
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 5

Node is deleted from the end.
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
```

```
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 6

Enter the location of the node after which you want to perform deletion : 1

Node is deleted from the specified position 2
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice : 7

List values are :
10
Choose an option
1.Insert at front
2.Insert at end
3.Insert at specified position
4.Delete from front
5.Delete from end
6.Delete from specified position
7.Display list contents
8.Exit
Enter your choice :
```

Lab Program - 7

Write a program to implement Single Linked 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("\nMemory is full\n");
exit(0);
}
return x;
}
NODE insert_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
{
return temp;
}
temp->link=first;
```

```
first=temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("List is empty. Cannot delete\n");  return first;
}
temp=first;
temp = temp->link;
printf("Item deleted at front end is %d\n",first->info);  free(first);
return temp;
}
NODE IF(NODE second,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(second==NULL)
return temp;
temp->link=second;
second=temp;
return second;
}
NODE IR(NODE second,int item) {
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(second==NULL)
return temp;
cur=second;
while(cur->link!=NULL)
cur=cur->link;
cur->link=temp;
```

```
return second;
}
NODE reverse(NODE first) {
NODE cur,temp;
cur=NULL;
while(first!=NULL)
{
temp=first;
first=first->link;
temp->link=cur;
cur=temp;
}
return cur;
}
NODE ascending(NODE first) {
NODE prev=first;
NODE cur=NULL;
int temp;
if(first== NULL)
{
return 0;
}
else
{
while(prev!= NULL)
{
cur = prev->link;
while(cur!= NULL)
{
if(prev->info > cur->info)  {
temp = prev->info;  prev->info = cur->info;  cur->info = temp;
}
cur = cur->link;
}
prev= prev->link;
}
}
return first;
```

```
}
NODE descending(NODE first) {
NODE prev=first;
NODE cur=NULL;
int temp;
if(first==NULL)
{
return 0;
}
else
{
while(prev!= NULL)
{
cur = prev->link;
while(cur!= NULL)
{
if(prev->info < cur->info)  {
temp = prev->info;
prev->info = cur->info;
cur->info = temp;
}
cur = cur->link;
}
prev= prev->link;
}
}
return first;
}
NODE concatenate(NODE first,NODE second) {
NODE cur;
if(first==NULL)
return second;
if(second==NULL)
return first;
cur=first;
while(cur->link!=NULL)
{
cur=cur->link;
```

```
}
cur->link=second;
return first;
}
void display(NODE first)
{
NODE temp;
if(first==NULL)
printf("List is empty. Cannot display items.\n");printf("List contents
are : ");
for(temp=first;temp!=NULL;temp=temp->link)
{
printf("\n%d",temp->info);
}
}
int main()
{
int item,choice,pos,element,option,choice2,item1,num;
NODE first=NULL;
NODE second=NULL;
for(;;)
{
printf("\n\nChoose an option");
printf("\n1:Insert_front \n2:Delete_front \n3:Reverse \n4:Sort
\n5.Concatenate \n6:Display  \n7:Exit\n");
printf("Enter the choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("Enter the item at front-end : ");
scanf("%d",&item);
first=insert_front(first,item);
printf("%d inserted at front-end.",first->info);
break;
case 2: first=delete_front(first);
break;
case 3: first=reverse(first);
printf("List is reversed.");
```

```
break;
case 4: printf("Press 1 for Ascending-sort and 2 for Descending-sort :
");  scanf("%d",&option);
if(option==1)
{
first=ascending(first);
printf("List is sorted in ascending order.");
}
if(option==2)
{
first=descending(first);
printf("List is sorted in descending order.");
}
break;
case 5: printf("Create a second list\n");
printf("Enter the number of elements in the second list : ");
scanf("%d",&num);
for(int i=1;i<=num;i++)
{
printf("\nPress 1 to Insert-front and 2 to Insert-rear : ");
scanf("%d",&choice2);
if(choice2==1)
{
printf("Enter the item at front-end : ");
scanf("%d",&item1);
second=IF(second,item1);
}
if(choice2==2)
{
printf("Enter the item at rear-end : ");
scanf("%d",&item1);
second=IR(second,item1);  }
}
first=concatenate(first,second);
printf("\nThe two lists are concatenated.");
break;
case 6: display(first);
break;
default: exit(0);
```

```
break;
}
}
return 0;
}
```

```
clang-7 -pthread -lm -o main main.c
./main


Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 1
Enter the item at front-end : 10
10 inserted at front-end.

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 2
Item deleted at front end is 10


Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
```

```
7:Exit
Enter the choice : 1
Enter the item at front-end : 10
10 inserted at front-end.

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 1
Enter the item at front-end : 20
20 inserted at front-end.

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 1
Enter the item at front-end : 30
30 inserted at front-end.

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
```

```
6:Display
7:Exit
Enter the choice : 2
Item deleted at front end is 30


Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 6
List contents are :
20
10

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 4
Press 1 for Ascending-sort and 2 for Descending-sort : 1
List is sorted in ascending order.

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
```

```
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 6
List contents are :
10
20

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 5
Create a second list
Enter the number of elements in the second list : 2

Press 1 to Insert-front and 2 to Insert-rear : 1
Enter the item at front-end : 10

Press 1 to Insert-front and 2 to Insert-rear : 2
Enter the item at rear-end : 20

The two lists are concatenated.

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
```

```
6:Display
7:Exit
Enter the choice : 6
List contents are :
10
20
10
20

Choose an option
1:Insert_front
2:Delete_front
3:Reverse
4:Sort
5.Concatenate
6:Display
7:Exit
Enter the choice : 7
```

Lab Program - 8

Write a program to implement Stacks using Linked List representation.

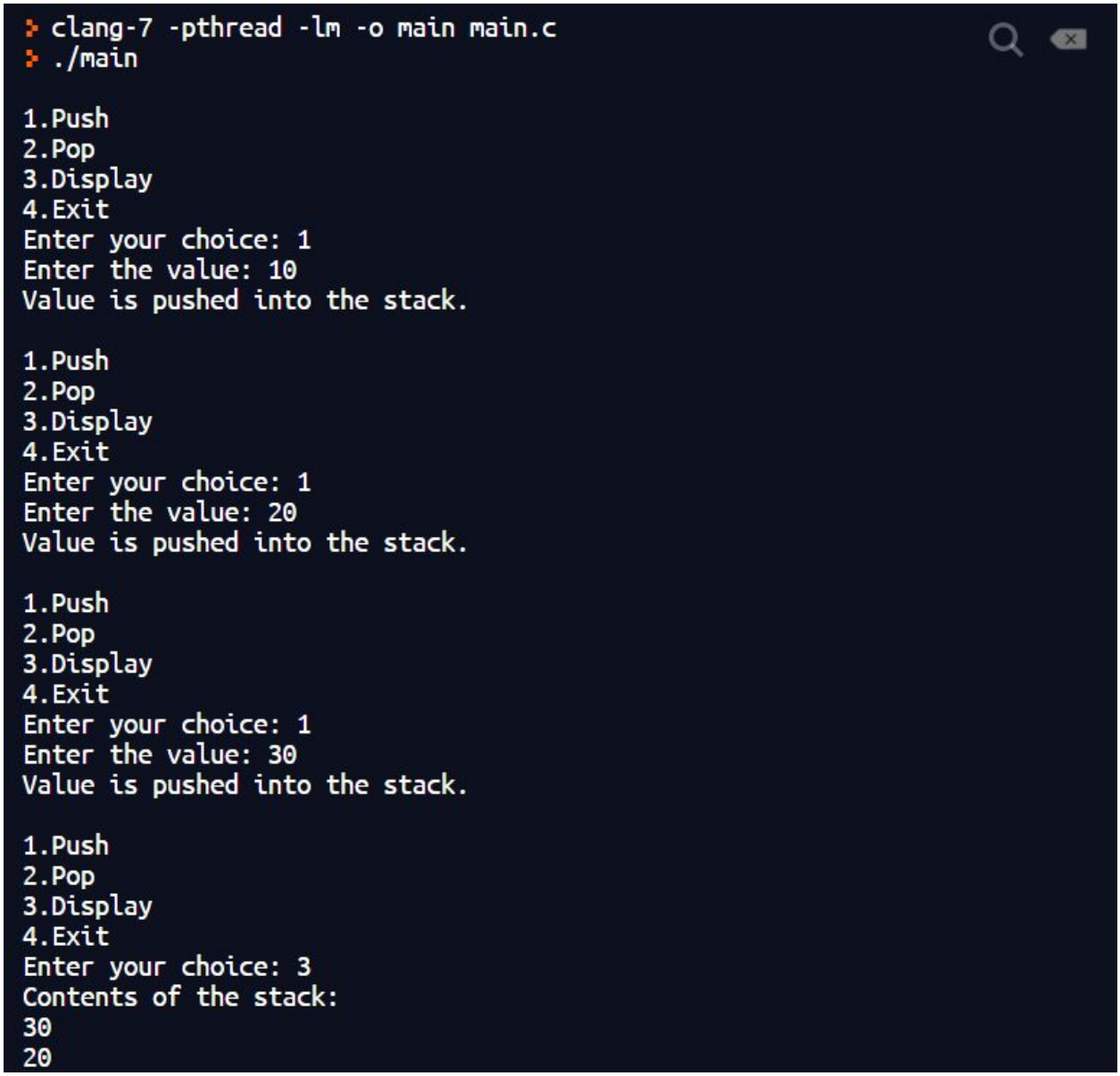
```
#include <stdio.h>
#include <stdlib.h>
void push();
void pop();
void display();
struct node
{
int val;
struct node *next;
};
struct node *head;
int main ()
{
int choice=0;
while(choice != 4)
{
printf("\n1.Push\n2.Pop\n3.Display\n4.Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice)
{
```

```
case 1:
{
push();
break;
}
case 2:
{
pop();
break;
}
case 3:
{
display();
break;
}
case 4:
{
printf("Exit");
break;
}
default:
{
printf("Please Enter valid choice.\n");
}
}
}
```

```
return 0;

}

void push ()

{

int val;

struct node *ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)

{

printf("Memory is full.");

}

else

{

printf("Enter the value: ");

scanf("%d",&val);

if(head==NULL)

{

ptr->val = val;

ptr -> next = NULL;

head=ptr;

}

else

{

ptr->val = val;

ptr->next = head;

head=ptr;

}
```

```
printf("Value is pushed into the stack.\n");  }
}
void pop()
{
int item;
struct node *ptr;
if (head == NULL)
{
printf("Stack Underflow\n");
}
else
{
item = head->val;
ptr = head;
head = head->next;
printf("%d is popped from the stack.\n",item);  free(ptr);
}
}
void display()
{
int i;
struct node *ptr;
ptr=head;
if(ptr == NULL)
{
printf("Stack is empty.\n");
```

```
}
else
{
printf("Contents of the stack: \n");
while(ptr!=NULL)
{
printf("%d\n",ptr->val);
ptr = ptr->next;
}
}
}
```


clang-7 -pthread -lm -o main main.c
./main
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the value: 10
Value is pushed into the stack.
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the value: 20
Value is pushed into the stack.
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the value: 30
Value is pushed into the stack.
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
Contents of the stack:
30
20

```
10

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2
30 is popped from the stack.

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
Contents of the stack:
20
10

1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 4
Exit
```

Write a program to implement Queues using Linked List representation.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *front;
struct node *rear;
void insert();
void deleteq();
void display();
int main ()
{
int choice;
while(choice != 4)
{
printf("\n1.Insert rear\n2.Delete front\n3.Display\n4.Exit\n");
printf("Enter your choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1:
insertq();
break;
case 2:
deleteq();
break;
case 3:
display();
break;
case 4:
exit(0);
break;
default:
```

```
printf("Enter valid choice.\n");  }
}
}
int insertq()
{
struct node *ptr;
int item;
ptr = (struct node *) malloc (sizeof(struct node));
if(ptr == NULL)
{
printf("Queue Overflow\n");
return 0;
}
else
{
printf("Enter the value: ");
scanf("%d",&item);
ptr -> data = item;
if(front == NULL)
{
front = ptr;
rear = ptr;
front -> next = NULL;
rear -> next = NULL;
}
else
{
rear -> next = ptr;
rear = ptr;
rear->next = NULL;
}
}
}
void deleteq()
{
struct node *ptr;
if(front == NULL)
{
printf("Queue Underflow.\n");
return;
```

```
}
else
{
ptr = front;
front = front -> next;
printf("%d is deleted from the queue.\n",ptr->data);  free(ptr);
}
}
void display()
{
struct node *ptr;
ptr = front;
if(front == NULL)
{
printf("Queue is empty.\n");  }
else
{
 printf("Contents of the queue are:\n");
 while(ptr != NULL)
{
printf("%d\n",ptr -> data);
ptr = ptr -> next;
}
}
}
```

```
1.Insert rear
2.Delete front
3.Display
4.Exit
Enter your choice: 1
Enter the value: 10

1.Insert rear
2.Delete front
3.Display
4.Exit
Enter your choice: 1
Enter the value: 20

1.Insert rear
2.Delete front
3.Display
4.Exit
Enter your choice: 2
10 is deleted from the queue.

1.Insert rear
2.Delete front
3.Display
4.Exit
Enter your choice: 3
Contents of the queue are:
20
```

Lab Program - 9

Write a program to implement doubly linked list with primitive operations: a) Create a doubly linked list b) Insert nodes at both ends c) Delete nodes at both ends d) Insert a new node to the left of the specified node e) Insert a new node to the right of the specified node f) Delete all key elements g) 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("Memory is full.\n");

exit(0);

}

return x;

}
```

```
void freenode(NODE x)
{
free(x);
}
NODE dinsert_front(int item,NODE head) {
NODE temp,cur;
temp=getnode();
temp->info=item;
cur=head->rlink;
head->rlink=temp;
temp->llink=head;
temp->rlink=cur;
cur->llink=temp;
return head;
}
NODE dinsert_rear(int item,NODE head) {
NODE temp,cur;
temp=getnode();
temp->info=item;
cur=head->llink;
head->llink=temp;
temp->rlink=head;
temp->llink=cur;
cur->rlink=temp;
return head;
}
```

```
NODE ddelete_front(NODE head)
{
NODE cur,next;
if(head->rlink==head)
{
printf("List is empty.\n");
return head;
}
cur=head->rlink;
next=cur->rlink;
head->rlink=next;
next->llink=head;
printf("Node deleted is %d",cur->info);
freenode(cur);
return head;
}
NODE ddelete_rear(NODE head) {
NODE cur,prev;
if(head->rlink==head)
{
printf("List is empty.\n");
return head;
}
cur=head->llink;
prev=cur->llink;
head->llink=prev;
```

```
prev->rlink=head;
printf("Node deleted is %d",cur->info);
freenode(cur);
return head;
}
NODE insert_leftpos(int item,NODE head) {
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("List is empty.\n");
return head;
}
cur=head->rlink;
while(cur!=head)
{
if(item==cur->info)
break;
cur=cur->rlink;
}
if(cur==head)
{
printf("Key not found.\n");
return head;
}
prev=cur->llink;
printf("Enter towards left of %d = ",item);  temp=getnode();
```

```
scanf("%d",&temp->info);
prev->rlink=temp;
temp->llink=prev;
cur->llink=temp;
temp->rlink=cur;
return head;
}
NODE insert_rightpos(int item,NODE head) {
NODE temp,cur,prev;
if(head->rlink==head)
{
printf("List is empty.\n");
return head;
}
cur=head->llink;
while(cur!=head)
{
if(item==cur->info)
break;
cur=cur->llink;
}
if(cur==head)
{
printf("Key not found.\n");
return head;
}
```

```
prev=cur->rlink;
printf("Enter towards right of %d = ",item);  temp=getnode();
scanf("%d",&temp->info);
prev->llink=temp;
temp->rlink=prev;
cur->rlink=temp;
temp->llink=cur;
return head;
}
NODE delete_all_key(int item,NODE head) {
NODE prev,cur,next;
int count;
if(head->rlink==head)
{
printf("List is empty.");
return head;
}
count=0;
cur=head->rlink;
while(cur!=head)
{
if(item!=cur->info)
cur=cur->rlink;
else
{
count++;
```

```
prev=cur->llink;

next=cur->rlink;

prev->rlink=next;

next->llink=prev;

freenode(cur);

cur=next;

}

}

if(count==0)

printf("Key not found.");

else

printf("Key found at %d positions and are deleted.\n", count);  return
head;

}

void display(NODE head)

{

NODE temp;

if(head->rlink==head)

{

printf("List is empty.\n");

return;

}

printf("Contents of the list : \n");

temp=head->rlink;

while(temp!=head)

{
```

```
printf("%d ",temp->info);
temp=temp->rlink;
}
printf("\n");
}
int main()
{
NODE head,last;
int item, choice;
head=getnode();
head->rlink=head;
head->llink=head;
for(;;)
{
printf("\n1:Insert front\n2:Insert rear\n3:Delete front\n4:Delete
rear\n5:Insert left  position\n6:Insert right position\n7:Delete all key
elements\n8:Display\n9:Exit\n");
printf("Enter the choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("Enter the item to be inserted at front end : ");
scanf("%d",&item);
last=dinsert_front(item,head);
break;
case 2: printf("Enter the item to be inserted at rear end : ");
scanf("%d",&item);
last=dinsert_rear(item,head);
```

```
break;
case 3: last=ddelete_front(head);
break;
case 4: last=ddelete_rear(head);
break;
case 5: printf("Enter the key item : ");
scanf("%d",&item);
head=insert_leftpos(item,head);  break;
case 6: printf("Enter the key item : ");  scanf("%d",&item);
head=insert_rightpos(item,head);  break;
case 7: printf("Enter the key item : ");  scanf("%d",&item);
head=delete_all_key(item,head);  break;
case 8: display(head);
break;
default: exit(0);
}
}
return 0;
}
```

```
1:Insert front
2:Insert rear
3:Delete front
4:Delete rear
5:Insert left  position
6:Insert right position
7:Delete all key elements
8:Display
9:Exit
Enter the choice : 1
Enter the item to be inserted at front end : 10

1:Insert front
2:Insert rear
3:Delete front
4:Delete rear
5:Insert left  position
6:Insert right position
7:Delete all key elements
8:Display
9:Exit
Enter the choice : 2
Enter the item to be inserted at rear end : 5

1:Insert front
2:Insert rear
3:Delete front
4:Delete rear
5:Insert left  position
6:Insert right position
7:Delete all key elements
8:Display
9:Exit
Enter the choice : 3
Node deleted is 10
```

```
1:Insert front
2:Insert rear
3:Delete front
4:Delete rear
5:Insert left  position
6:Insert right position
7:Delete all key elements
8:Display
9:Exit
Enter the choice : 5
Enter the key item : 10
Key not found.

1:Insert front
2:Insert rear
3:Delete front
4:Delete rear
5:Insert left  position
6:Insert right position
7:Delete all key elements
8:Display
9:Exit
Enter the choice : 6
Enter the key item : 5
Enter towards right of 5 = 20

1:Insert front
2:Insert rear
3:Delete front
4:Delete rear
5:Insert left  position
6:Insert right position
7:Delete all key elements
8:Display
9:Exit
```

```
Enter the choice : 7
Enter the key item : 5
Key found at 1 positions and are deleted.

1:Insert front
2:Insert rear
3:Delete front
4:Delete rear
5:Insert left  position
6:Insert right position
7:Delete all key elements
8:Display
9:Exit
Enter the choice : 9
```

Lab 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, pre order 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;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("Memory is full.\n");

exit(0);

}

return x;

}
```

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

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

```
suc=suc->llink;

suc->llink=cur->llink;

q=cur->rlink;

}

if(parent==NULL)

return q;

if(cur==parent->llink)  parent->llink=q;

else

parent->rlink=q;

freenode(cur);

return root;

}

void preorder(NODE root) {

if(root!=NULL)

{

printf("%d\n",root->info);  preorder(root->llink);
preorder(root->rlink);  }

}

void postorder(NODE root) {

if(root!=NULL)

{

postorder(root->llink);  postorder(root->rlink);
printf("%d\n",root->info);  }

}

void inorder(NODE root) {

if(root!=NULL)
```

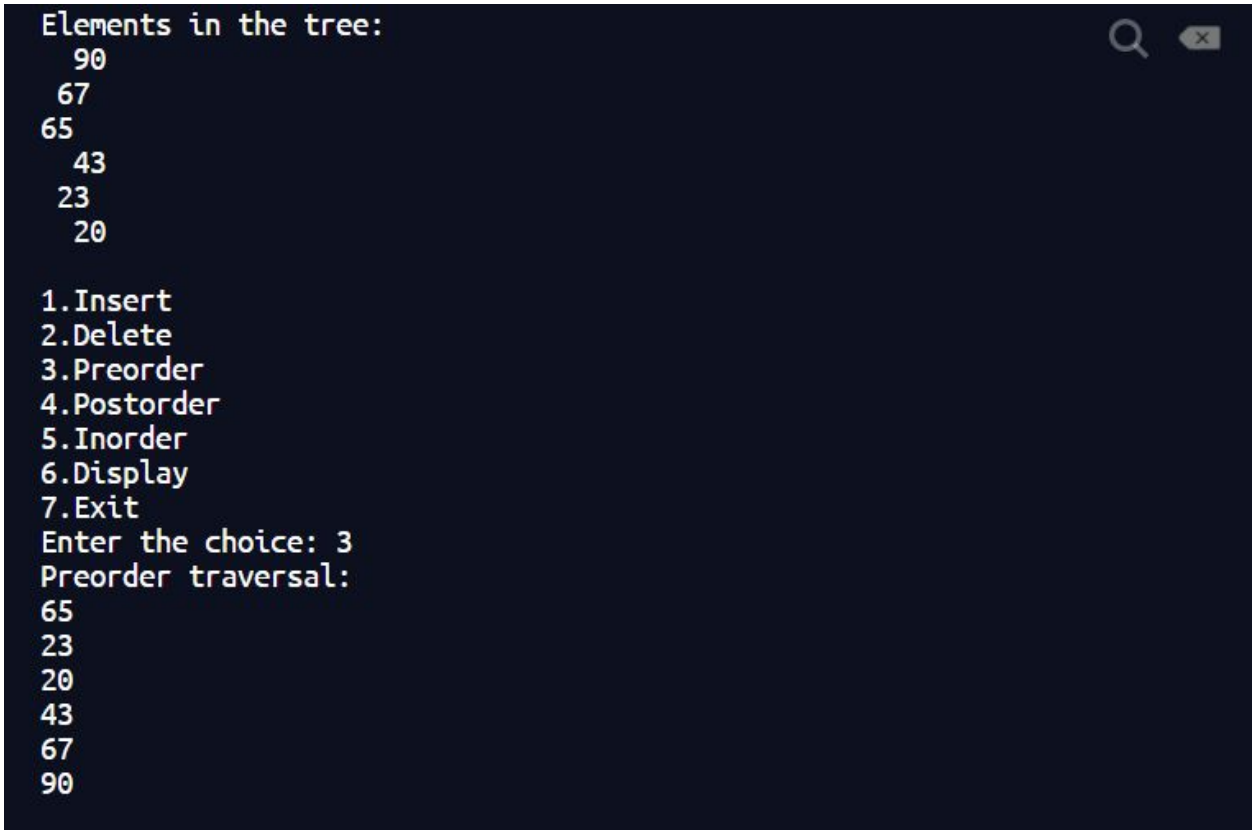
```
{
inorder(root->llink);
printf("%d\n",root->info);
inorder(root->rlink);
}
}
void display(NODE root,int i)
{
int j;
if(root!=NULL)
{
display(root->rlink,i+1);
for(j=0;j<i;j++)
printf(" ");
printf("%d\n",root->info);
display(root->llink,i+1);
}
}
int main()
{
int item,choice;
NODE root=NULL;
for(;;)
{
printf("\n1.Insert\n2.Delete\n3.Preorder\n4.Postorder\n5.Inorder\n6.Disp
lay\n7.Exit\n");  printf("Enter the choice: ");
```

```
scanf("%d",&choice);
switch(choice)
{
case 1: printf("Enter the item: ");  scanf("%d",&item);
root=insert(root,item);
break;
case 2: printf("Enter the item: ");  scanf("%d",&item);
root=delete(root,item);
break;
case 3: printf("Preorder traversal: \n");  preorder(root);
break;
case 4: printf("Postorder traversal: \n");  postorder(root);
break;
case 5: printf("Inorder traversal: \n");  inorder(root);
break;
case 6: printf("Elements in the tree: \n");  display(root,0);
break;
default:exit(0);
break;
}
}
return 0;
}
```

```
1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 1
Enter the item: 45

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 1
Enter the item: 23

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 1
Enter the item: 20

1.Insert
2.Delete
3.Preorder
4.Postorder
```

```
5.Inorder
6.Display
7.Exit
Enter the choice: 1
Enter the item: 43

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 1
Enter the item: 65

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 1
Enter the item: 67

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 1
Enter the item: 90
```

```
1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 6
Elements in the tree:
   90
  67
 65
45
  43
 23
  20

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 2
Enter the item: 45

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 6
```


Elements in the tree:
90
67
65
43
23
20
1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 3
Preorder traversal:
65
23
20
43
67
90

```
1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 4
Postorder traversal:
20
43
23
90
67
65

1.Insert
2.Delete
3.Preorder
4.Postorder
5.Inorder
6.Display
7.Exit
Enter the choice: 5
Inorder traversal:
20
23
43
65
67
90
```