LATE BHAUSAHEB HIRAY S.S. TRUST'S INSTITUTE OF COMPUTER APPLICATION

ISO 9001-2008 CERTIFIED

S.N. 341, Next to New English School, Govt. Colony, Bandra (East), Mumbai – 400051, Tel: 91-22-26570892/3181

Date:09/04/2022

CERTIFICATE

This is to certify that

Mr./Ms. RITIK SHEKHAR KAMERKAR Roll No: 202187 is a student of MCA of 1st year Semester-I has completed successfully full-semester practical/assignments of subject Data Structure for the academic year 2021 – 22.

Subject In-Charge **Director**

External Examiner

LBHSS’s

Hiray Institute of Computer Application

DATA STRUCTURE

PRACTICAL – JOURNAL

F.Y.MCA

Index

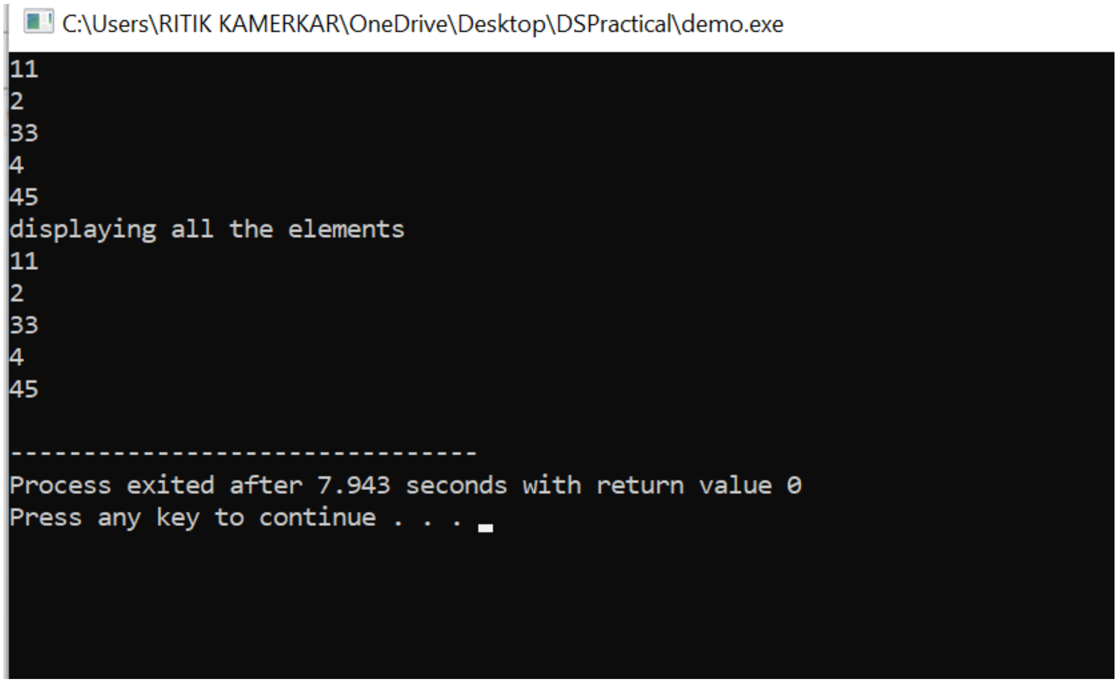
9	**Create and perform various operations on BST** a) Inserting node in BST b) Deleting the node from BST c) To find height of Tree d) To perform Inorder e) To perform Preorder f) To perform Postorder g) To find Maximum value of tree		
10	**Create a minimum spanning tree using any method Kruskal’s algorithm or Prim’s algorithm.**		
11	**Perform various hashing techniques with Linear Probe as collision resolution scheme.**		
12	**Implementing Heap with different operations performed** a) To perform insertion operation b) To create Heap using Heapify method c) To perform Heap sort d) To delete the value in heap		
13	**Implementation of Graph Traversal** a) Implement Depth First Search (DFS) b) Implement Breath First Search (BFS)		

Practical No. 1: Implementation of sorting algorithms.

Aim: Write a program in c++ to assing 5 elements in array. And Print all the elements.

```
#include<iostream>
using namespace std;
int main()
{
//int list[5]={1,2,3,4,5};//static array
//int size;
//cout<<"enter size of the array"<<endl;
//cin>>size;
int list[100];//declaration of array
//at the compile time the memory will be allocated to the array
cout<<"enter array elements"<<endl;
for(int i=0;i<5;i++)
cin>>list[i];
cout<<"displaying all the elements"<<endl;
for(int i=0;i<5;i++)
{
cout<<list[i]<<endl;
}
return 0;
}
```

OutPut:

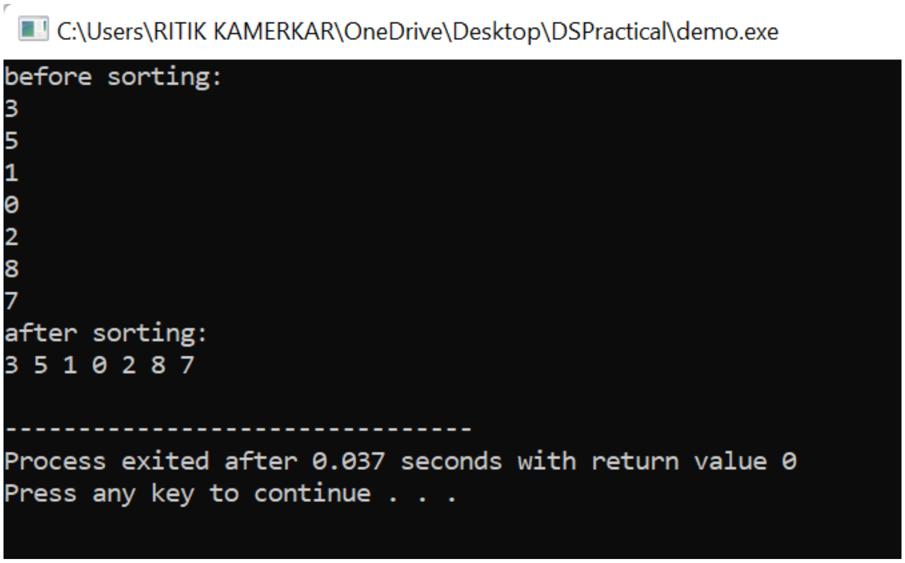

C:\Users\RITIK KAMERKAR\OneDrive\Desktop\DSPractical\demo.exe

```
11
2
33
4
45
displaying all the elements
11
2
33
4
45

--------------------------------
Process exited after 7.943 seconds with return value 0
Press any key to continue . . .
```

Aim: Write a program in c++ to implement selection sort.

```
#include<iostream>
using namespace std;
void SelectionSort(int array[],int arraySize)
{
for(int i=0;i<arraySize;i++)//3,5,1,0,2,8,7
{
//find the minimum element in the array[0...... arraySize]
int currentMin = array[i];
int currentMinIndex = i;
for(int j=i+1;j<arraySize;j++)
{
if(currentMin>array[j])
{
currentMin = array[j];//0 as the min
currentMinIndex = j;
}
}
if(currentMinIndex != i)
{
array[currentMinIndex]=array[i];
array[i]=currentMin;
}
}
}
int main()
{
int array[]={3,5,1,0,2,8,7};
double array1[]={1.1,2.2,3.2,4.2};
string array2[]={"mumbai","delhi","chicago","goa"};
cout<<"before sorting:"<<endl;
for(int i=0;i<7;i++)
{
cout<<array[i]<<" "<<endl;
}
SelectionSort(array,7);
cout<<"after sorting:"<<endl;
for(int i=0;i<7;i++)
{
cout<<array[i]<<" ";

}
cout<<endl;
return 0;
}
```

Output:



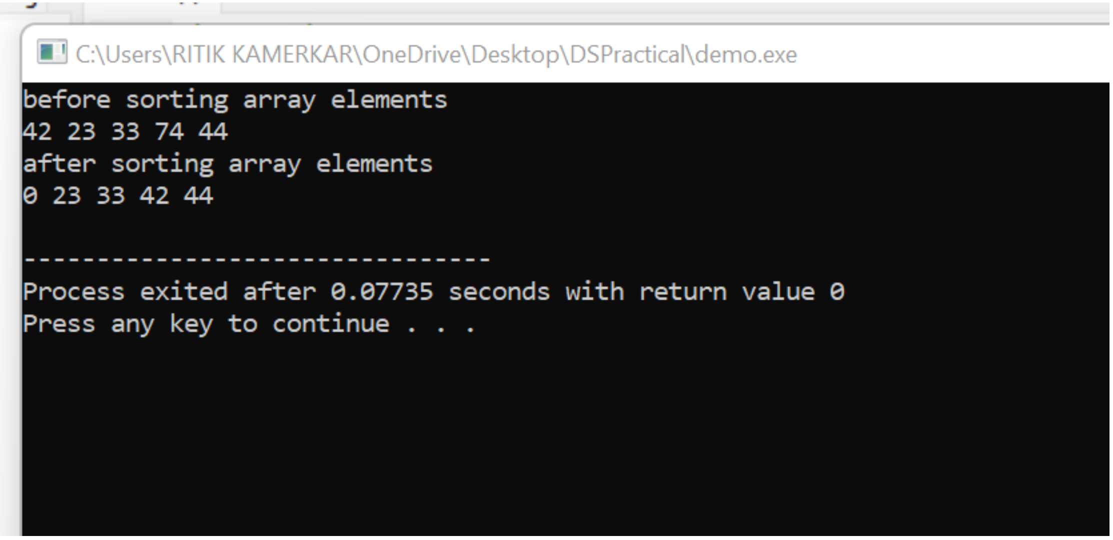
C:\Users\RITIK KAMERKAR\OneDrive\Desktop\DSPractical\demo.exe

```
before sorting:
3
5
1
0
2
8
7
after sorting:
3 5 1 0 2 8 7

--------------------------------
Process exited after 0.037 seconds with return value 0
Press any key to continue . . .
```

Aim: Write a program in c++ to implement bubble sort.

```
#include<iostream>
using namespace std;
void bubbleSort(int array[],int arraySize)
{
for(int pass=0;pass<arraySize;pass++)
{
for(int i=0;i<arraySize;i++)
{
if(array[i]>array[i+1])
{
int temp = array[i];
array[i]=array[i+1];
array[i+1]=temp;
}
}
}
}
void printArray(int array[],int arraySize)
{
for(int i=0;i<arraySize;i++)
{
cout<<array[i]<<" ";
}
cout<<endl;
}
int main()
{
int array[]={42,23,33,74,44};
cout<<"before sorting array elements"<<endl;
printArray(array,5);
cout<<"after sorting array elements"<<endl;
bubbleSort(array,5);
printArray(array,5);
}
```

Output:



C:\Users\RITIK KAMERKAR\OneDrive\Desktop\DSPractical\demo.exe

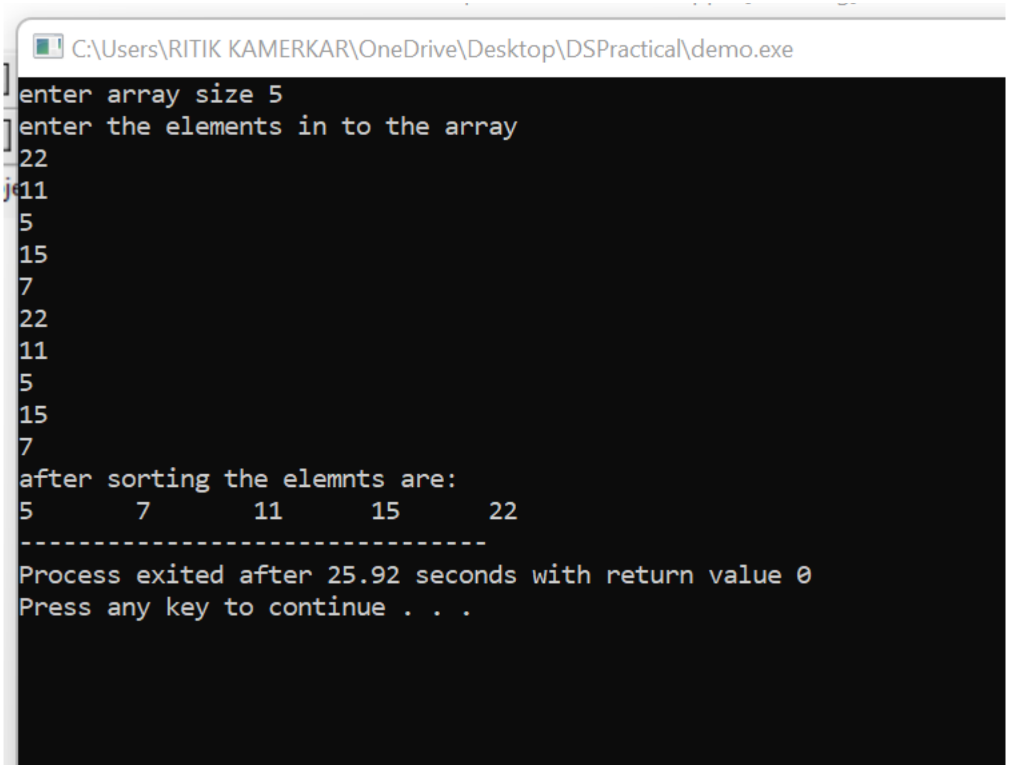
```
before sorting array elements
42 23 33 74 44
after sorting array elements
0 23 33 42 44

--------------------------------
Process exited after 0.07735 seconds with return value 0
Press any key to continue . . .
```

Aim: Write a program in c++ to implement insertion sort.

```
#include<iostream>
using namespace std;
//the array elemnts are 80,77,33,44,11,88,22
void insertionSort(int array[],int arraySize)
{
for(int i=1;i<arraySize;i++)//i=1 we are stating with 77
{
// i=2
//sorted array sublist is array[0,....i-1]
//for i=1 sorted array is array[0]
int currentElement = array[i];//currentElemnt=array[1]
for(int k=i-1;k>=0 && array[k]>currentElement;k--)
//k=0;k>=0 && array[0]>array[1](80>77)
//k=2-1=1;1>=0&&
//k -,k=0
{
array[k+1]=array[k];//array[0]=80
//we are inserting the current element in correct position
array[k]=currentElement;
//array[0+1]=77

}
}
}
int main()
{
int arraySize;
int array[arraySize];
```

```
 cout<<"enter array size";
 cin>>arraySize;
 cout<<"enter the elements in to the array"<<endl;
 for(int i=0;i<arraySize;i++)
 cin>>array[i];

 for(int i=0;i<arraySize;i++)
 cout<<array[i]<<endl;
 insertionSort(array,arraySize);
 cout<<"after sorting the elemnts are:"<<endl;
 for(int i=0;i<arraySize;i++)
 cout<<array[i]<<"\t";
 return 0;
 }
```

Output:



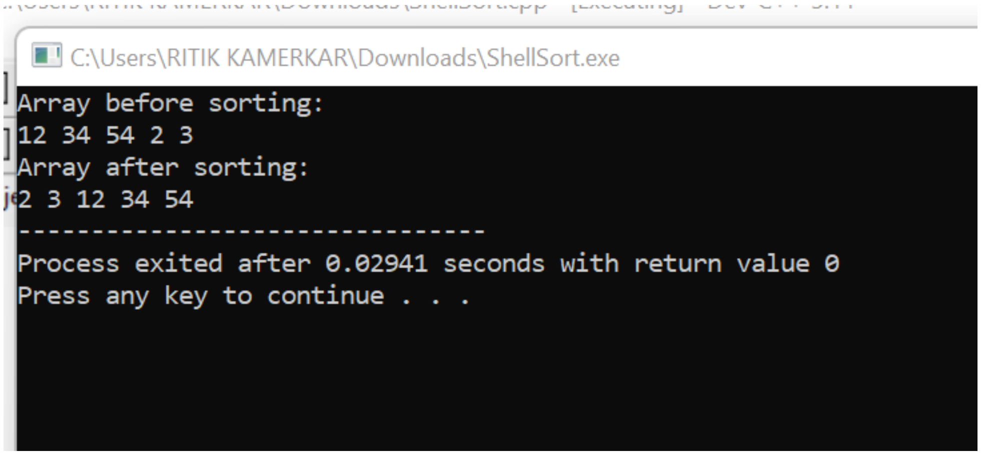
C:\Users\RITIK KAMERKAR\OneDrive\Desktop\DSPractical\demo.exe

```
enter array size 5
enter the elements in to the array
22
11
5
15
7
22
11
5
15
7
after sorting the elemnts are:
5       7       11      15      22
--------------------------------
Process exited after 25.92 seconds with return value 0
Press any key to continue . . .
```

Aim: Write a program in c++ to implement Shell sort.

```
// C++ implementation of Shell Sort
#include  <iostream>
using namespace std;

/* function to sort arr using shellSort */
int shellSort(int arr[], int n)
{
    // Start with a big gap, then reduce the gap
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        // Do a gapped insertion sort for this gap size.
        // The first gap elements a[0..gap-1] are already in gapped order
        // keep adding one more element until the entire array is
        // gap sorted
        for (int i = gap; i < n; i += 1)
        {
            // add a[i] to the elements that have been gap sorted
            // save a[i] in temp and make a hole at position i
            int temp = arr[i];

            // shift earlier gap-sorted elements up until the correct
            // location for a[i] is found
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];

            //  put temp (the original a[i]) in its correct location
            arr[j] = temp;
        }
    }
    return 0;
}

void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Array before sorting: \n";
    printArray(arr, n);

    shellSort(arr, n);

    cout << "\nArray after sorting: \n";
    printArray(arr, n);

    return 0;
}
```

Output:



```
Array before sorting:
12 34 54 2 3
Array after sorting:
2 3 12 34 54
--------------------------------
Process exited after 0.02941 seconds with return value 0
Press any key to continue . . .
```

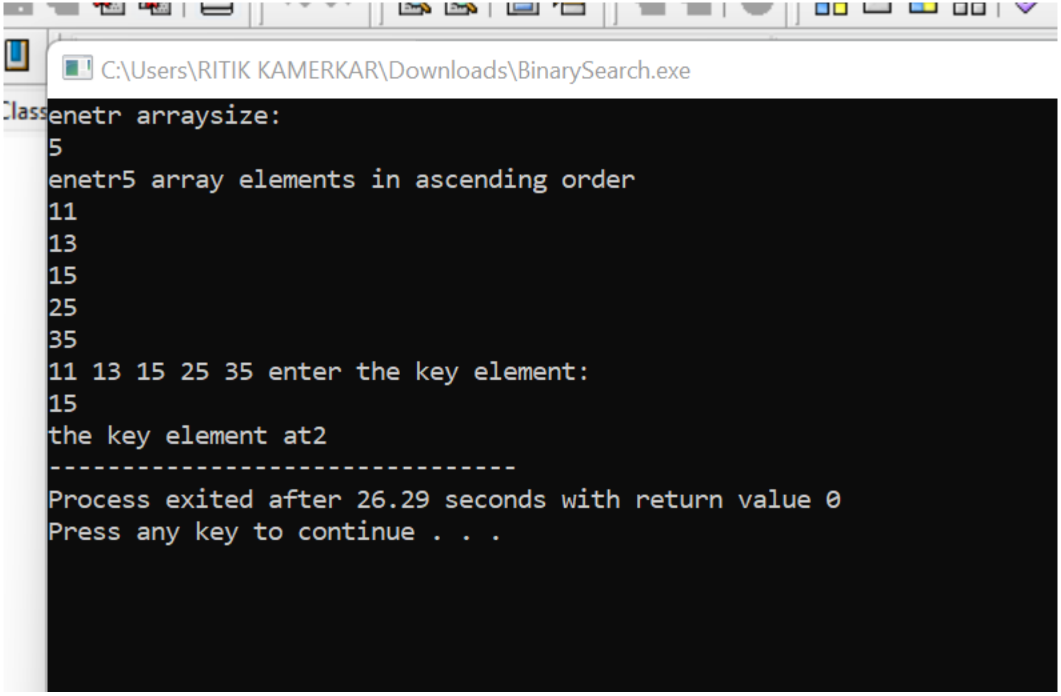
Practical No. 2: Implementation of different searching techniques

Aim: Write a program in c++ to implement Binary Search Tree.

Binary Search Tree:

```
#include<iostream>
using namespace std;
void bubbleSort(int Array[],int arraySize)
{
        for(int pass=1;pass<arraySize;pass++)
        {
                for(int i=0;i<arraySize-pass;i++)
                {
                        if(Array[i]>Array[i+1])
                        {
                                int temp = Array[i];
                                Array[i]=Array[i+1];
                                Array[i+1]=temp;
                        }
                }
        }
}
int binarySearch(int Array[],int arraySize,int key)
{
        int start = 0;
        int end = arraySize -1;
        int mid =int((start+end)/2);//start mid=3
        int pos;
        while(start<=end&&Array[mid]!=key)//array[3]==mid
        {
                if(key<Array[mid])//true proceed to left
                end=mid-1;
                else
                start = mid+1;
                mid =int((start+end)/2);
        }
        if(Array[mid]==key)
        pos=mid;
        else
        pos=-1;
        return pos;
}
int main()
{
        int arraySize,key,i;
        cout<<"enetr arraysize:"<<endl;
        cin>>arraySize;
        int Array[arraySize];
        cout<<"enetr"<<arraySize<<" array elements in ascending order";
        for(i=0;i<arraySize;i++)
        cin>>Array[i];
        bubbleSort(Array,arraySize);
        for(i=0;i<arraySize;i++)
        cout<<Array[i]<<" ";
        cout<<"enter the key element:"<<endl;
        cin>>key;
        int pos=binarySearch(Array,arraySize,key);//pos=mid
        //pos=-1
        if(pos==-1)
        cout<<"key element not found";
        else
        cout<<"the key element at"<<pos;


        return 0;
}
```

Output:



C:\Users\RITIK KAMERKAR\Downloads\BinarySearch.exe

```
enetr arraysize:
5
enetr5 array elements in ascending order
11
13
15
25
35
11 13 15 25 35 enter the key element:
15
the key element at2
--------------------------------
Process exited after 26.29 seconds with return value 0
Press any key to continue . . .
```

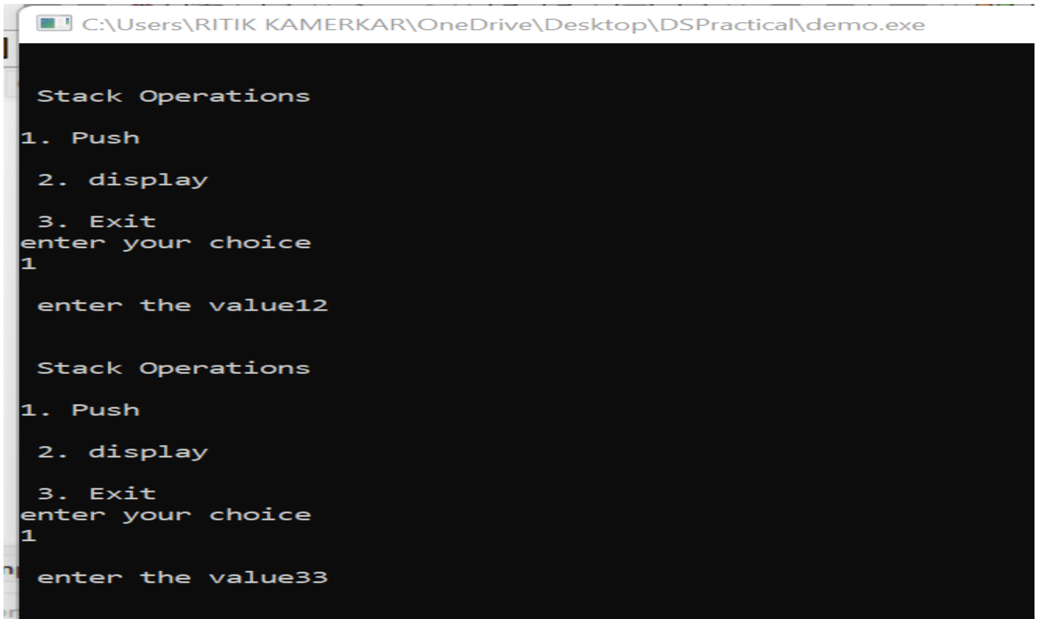
Practical No. 3: Implementation of Stacks (Using arrays and Linked List)

Aim: Write a program in c++ to implement stack with linked list.

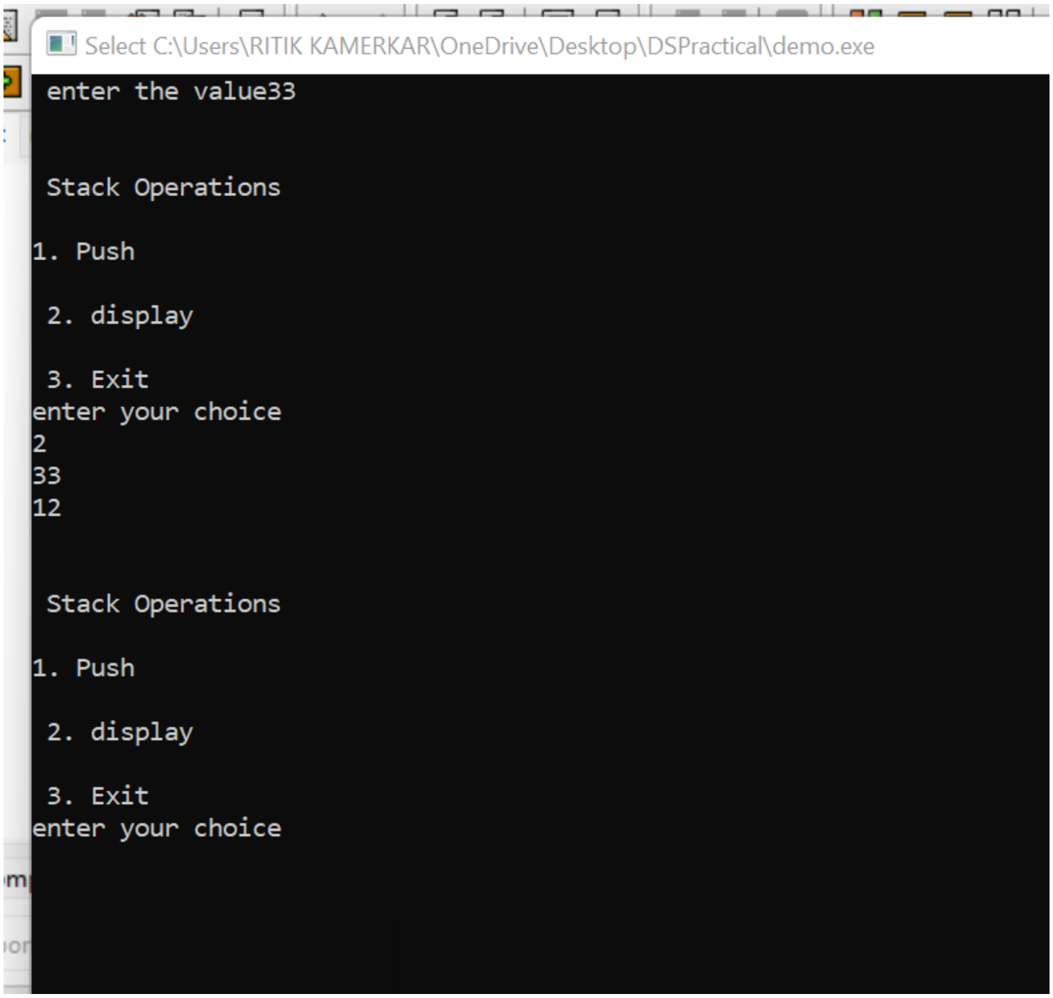
```
#include<iostream>
using namespace std;
struct Node
{
int data;
struct Node* next;
};
struct Node *TOP=NULL;
void push(int x)
{
//checking for overflow
//create a node
struct Node *newNode;//newNode has two parts
//1. data
//2. next
newNode = new Node;//creation of structude
newNode->data=x;//with the help of -> symbol we can
//assign or access from a structure
newNode->next=NULL;
if(TOP==NULL)//checking is there any alement in the stack
TOP=newNode;
else
newNode->next=TOP;
TOP=newNode;
}
void display()
{
struct Node* T;
//cout<<TOP->data;
if(TOP==NULL)
cout<<"the stack is empty"<<endl;

else
{

for(T=TOP ; T!=NULL ; T=T->next)
{

cout<<T->data<<endl;
}

}
}

int main()
```

```
{
int element, c;
while(1)//1 will treated as true
{
cout<<"\n\n Stack Operations\n\n";
cout<<"1. Push\n\n 2. display\n\n ";
cout<<"3. Exit\n";
cout<<"enter your choice"<<endl;
cin>>c;
switch(c)
{
case 1:
cout<<"\n enter the value";
cin>>element;
push(element);
break;
// case 2:
//pop();
// break;
case 2:
display();
break;
case 3:
exit(0);
break;
default:
cout<<"wrong choice";
break;

}
}

return 0;
}
```

Output:



```
C:\Users\RITIK KAMERKAR\OneDrive\Desktop\DSPractical\demo.exe


 Stack Operations

1. Push

 2. display

 3. Exit
enter your choice
1

 enter the value12


 Stack Operations

1. Push

 2. display

 3. Exit
enter your choice
1

 enter the value33
```



Select C:\Users\RITIK KAMERKAR\OneDrive\Desktop\DSPractical\demo.exe

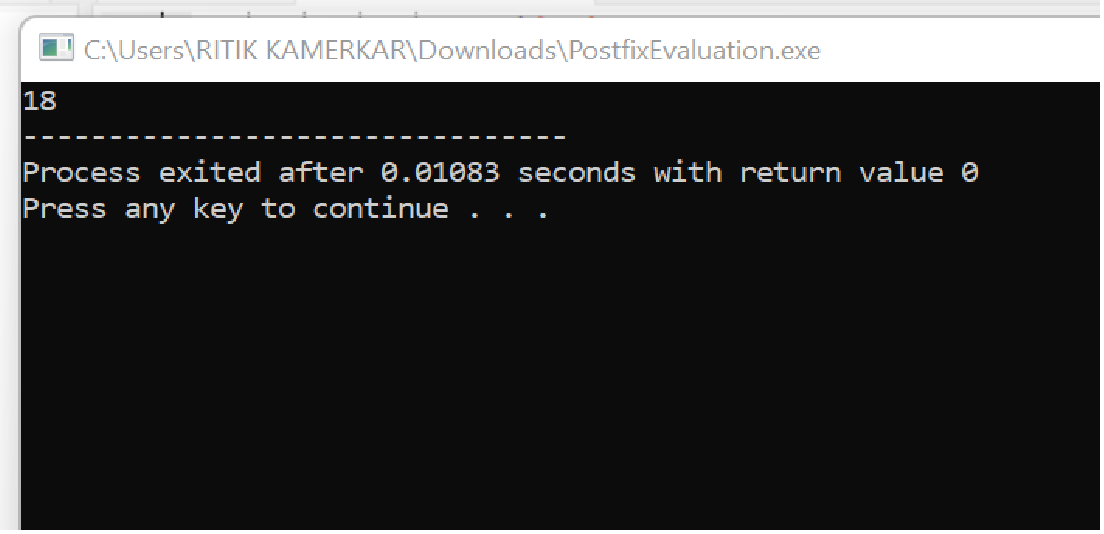
```
 enter the value33


 Stack Operations

1. Push

 2. display

 3. Exit
enter your choice
2
33
12


 Stack Operations

1. Push

 2. display

 3. Exit
enter your choice
```

Practical No. 4: Implementation of Stack Application

Aim: Write a program in c++ to implement Postfix evaluation.

```
#include<iostream>
using namespace std;
#define MAXSIZE 100
int stack[MAXSIZE];//stack can contain 100 elements and stored from 0th index to 99th index
int top = -1;//initially the stack is empty
void push(int item)
{
    //check overflow
    if(top==MAXSIZE-1)
    cout<<"the stack is full\n";
    else
    {
    top++;
    stack[top]=item;
    }
    //stack[++top]=item;
}
int pop()
{
    int item;
    //check for underflow
    if(top==-1)
    cout<<"the stack is empty\n";
    else
    {
        item = stack[top];
        top--;
        //item = stack[top --];
    }
    return item;
}
void evaluatePostfix(char expr[])
{
    int i;
    for(i=0;expr[i];++i)//to scan the expression
    {
        // 5 2 + 3 9 3 / -
        if(expr[i]==' ')//for space between character
        continue;
        else if(isdigit(expr[i]))
        push(expr[i]-'0');
        else
        {
            int A = pop();
            int B = pop();
            switch(expr[i])
            {
                case '+':
                    push(B+A);
                    break;
                case '-':
                    push(B-A);
                    break;
                case '/':
                    push(B/A);
                    break;
                case '*':
                    push(B*A);
                    break;
            }
        }
    }
    cout<<stack[top];
}
int main()
{
    char expr[]="5 2 + 3 * 9 3 / -";
    //52 3 * 3 -
    //156 3 -
    //153
    evaluatePostfix(expr);
    return 0;
}
```

Output:



C:\Users\RITIK KAMERKAR\Downloads\PostfixEvaluation.exe

```
18
--------------------------------
Process exited after 0.01083 seconds with return value 0
Press any key to continue . . .
```

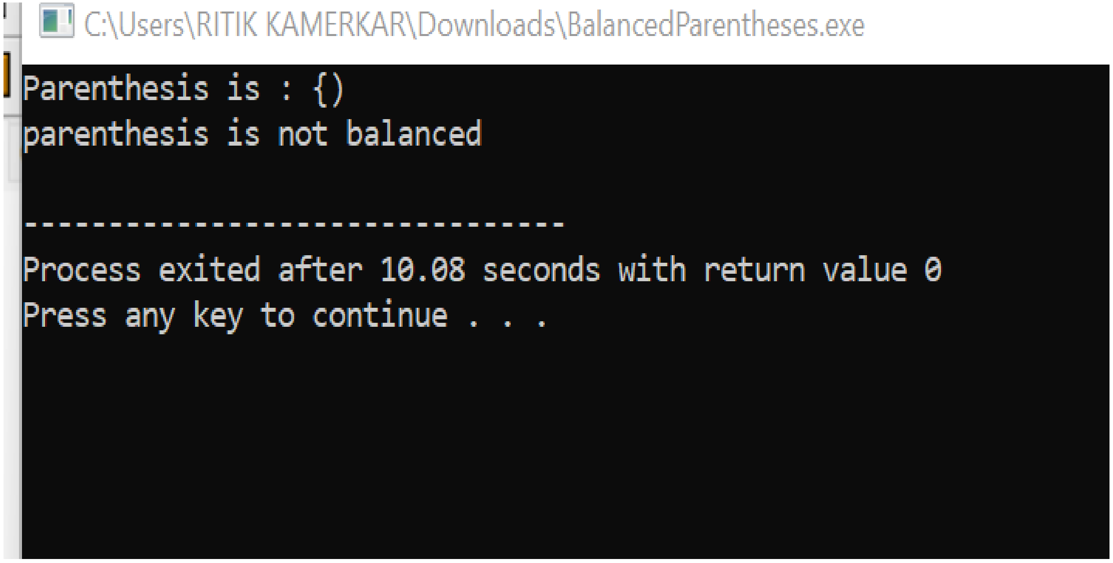
Aim: Write a program in c++ to implement Balancing of Parenthesis.

```
#include<iostream>
#include<string.h>
using namespace std;
#define MAX 20
class Stack //creating Stack
{
	public:
		char stk[MAX];
		int top;
};
Stack s;//stk and top
//to access stk and top we have write s.stk[],s.top
void push(char item)
{//check overflow condition
	if(s.top==(MAX - 1))
	cout<<"stack is full\n";
	else
	{
		s.top++;//0,1
		s.stk[s.top]=item;
	}
}
void pop()
{
	//check underflow condition
	if(s.top==-1)
	{
		cout<<"stack is empty\n";
		//cout<<"expected declaration before closing token\n";
		//exit(0);
	}
	else
	{
		s.top=s.top - 1;
	}
}
//bool balancedParentheses(string expr)
void balancedParentheses(string expr)
{
	s.top=-1;
	char x;//to store the top value

	int i=0;
	for(i=0;i<expr.length();i++)
	{
		if(expr[i]=='('||expr[i]=='{'||expr[i]=='[')
		{
			push(expr[i]);
			continue;
		}
```

```
        else
        {
        switch(expr[i])
        {
            case ')'://{)}[]
                x = s.stk[s.top];//save the top value
                pop();//remove from stack
                if(x=='{'|| x=='[')
                {
                    cout<<"parenthesis is not balanced"<<endl;
                //return false;
                exit(0);
                }
                break;

            case '}':
                x = s.stk[s.top];
                pop();
                if(x=='('|| x=='[')
                {
                cout<<"stack is not balanced"<<endl;
                //return false;
                exit(0);
                }
                break;
            case ']':
                 x = s.stk[s.top];
                pop();
                if(x=='{'|| x=='(')
                //return false;
                exit(0);
                break;
        }
}
    }
    //to conclude parenthesis is balanced
    //we have to check two conditions:
    //1.stack should be enmpty and
    //2. we have to reach at the expression
    //expr.length()
    if(i==expr.length()&& s.top==-1)
    cout<<"parenthesis is balanced\n";
    else
    cout<<"parenthesis is not balanced\n";

}
int main()
{
    cout<<"Parenthesis is : {)"<<endl;
    string expr="{)";
    //cout<<"enter the expression\n";
    //cin>>expr;
```

```
        balancedParentheses(expr);
        return 0;
}
```

Output:



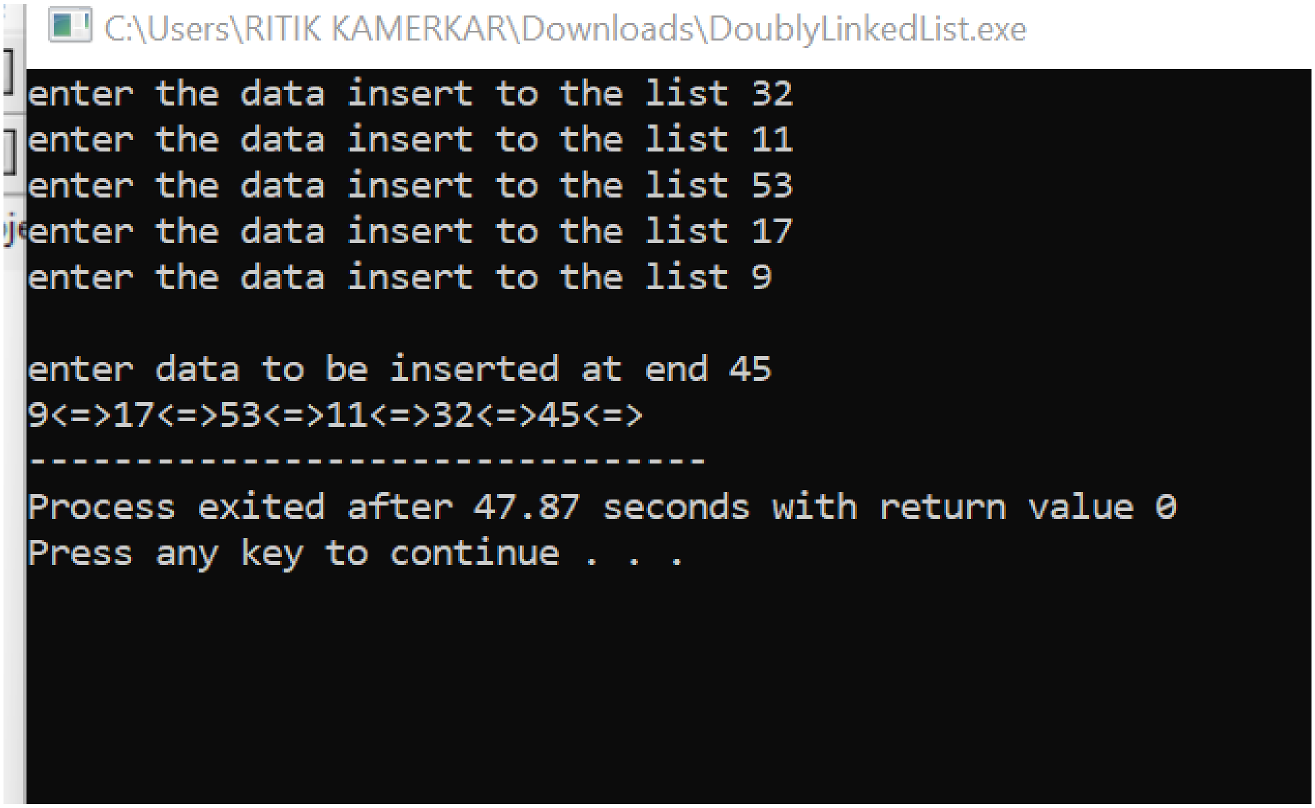
```
Parenthesis is : {)
parenthesis is not balanced

--------------------------------
Process exited after 10.08 seconds with return value 0
Press any key to continue . . .
```

Practical No. 5: Implementation of all types of linked list

Aim: Write a program in c++ to implement Double Linked list

```
#include<iostream>
using namespace std;
class DoublyLinkedList
{
        struct Node
        {
        int data;
        struct Node* prev;
        struct Node* next;
        };
        struct Node* head;
        int data;
        public:
                DoublyLinkedList();
                void insertAtFront();
                void insertAtEnd();
                void display();
};
DoublyLinkedList::DoublyLinkedList()
{
        head=NULL;
}
void DoublyLinkedList::insertAtFront()
{
        struct Node* temp;//declaration
        cout<<"enter the data insert to the list";
        cin>>data;
        temp=new struct Node;
        temp->prev=NULL;
        temp->data=data;
        temp->next=NULL;
        //check list is empty or not
        if(head==NULL)
        head=temp;//when there is no element in the list
        else
        {
                temp->next=head;
                head->prev=temp;
                head=temp;
        }
}
void DoublyLinkedList::insertAtEnd()
{
        struct Node* temp,*q;
        temp=new struct Node;
        cout<<"\nenter data to be inserted at end";
        cin>>data;
        temp->prev=NULL;
        temp->data=data;
        temp->next=NULL;
        if(head==NULL)
        {
                head=temp;
        }
        else
        {
        q=head;
        while(q->next!=NULL)
        {

        q=q->next;
        }
        q->next=temp;
        temp->prev=q;
        }
}
void DoublyLinkedList::display()
{
        struct Node* q;
        if(head==NULL)
        {
                cout<<"list is empty"<<endl;
        }
        else
        {
                q=head;
                while(q!=NULL)
                {
                        cout<<q->data<<"<=>";
                        q=q->next;
                }
        }
}
int main()
{
        DoublyLinkedList dl;
        dl.insertAtFront();
        dl.insertAtFront();
        dl.insertAtFront();
        dl.insertAtFront();
        dl.insertAtFront();
        dl.insertAtEnd();
        dl.display();
        return 0;
}
```

Output:

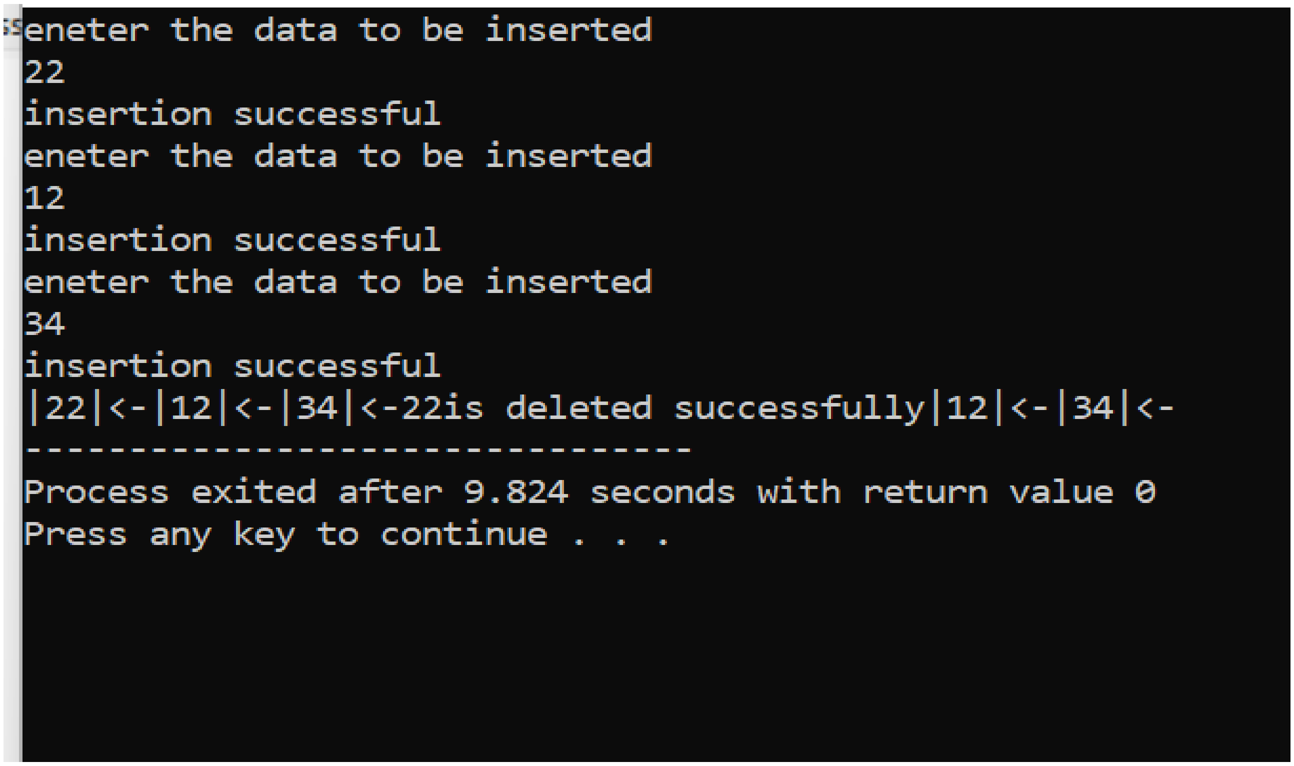


```
enter the data insert to the list 32
enter the data insert to the list 11
enter the data insert to the list 53
enter the data insert to the list 17
enter the data insert to the list 9

enter data to be inserted at end 45
9<=>17<=>53<=>11<=>32<=>45<=>
--------------------------------
Process exited after 47.87 seconds with return value 0
Press any key to continue . . .
```

Practical No. 6: Implement all different types of queues

Aim: Write a program in c++ to implement Queue Array.

```
#include<iostream>
# define max 5
using namespace std;
class Queue
{
        int QArr[max],data;
        int front,rear;
        public:
                Queue();
                void insertToQueue();
                void deleteFromQueue();
                void display();
};
Queue::Queue()
{
        front=rear=-1;
}
void Queue::insertToQueue()
{
        if(rear==max-1)
        {
                cout<<"Queue is Full";
                return;
        }
        else
        {
                if(front==-1)//no element in the queue
                front=0;
                cout<<"eneter the data to be inserted"<<endl;
                cin>>data;
                rear=rear+1;
                QArr[rear]=data;
                cout<<"insertion successful"<<endl;
        }
}
void Queue::deleteFromQueue()
{
        if(front==-1)
        {
        cout<<"queue is empty";
        return;
        }
        else
        {
                data=QArr[front];
                front++;//now front is 1
                cout<<data<<"is deleted successfully";
        }
}
void Queue::display()
{
        if(front==-1 && rear==-1)
        {
                cout<<"queue is empty";
                return;
        }
        else
        {
                for(int i=front;i<=rear;i++)
                cout<<"|"<<QArr[i]<<"|<-";
        }
}
int main()
{
        Queue q;
        q.insertToQueue();
        q.insertToQueue();
        q.insertToQueue();
        q.display();
        q.deleteFromQueue();
        q.display();
        return 0;
}
```



```
eneter the data to be inserted
22
insertion successful
eneter the data to be inserted
12
insertion successful
eneter the data to be inserted
34
insertion successful
|22|<-|12|<-|34|<-22is deleted successfully|12|<-|34|<-
--------------------------------
Process exited after 9.824 seconds with return value 0
Press any key to continue . . .
```

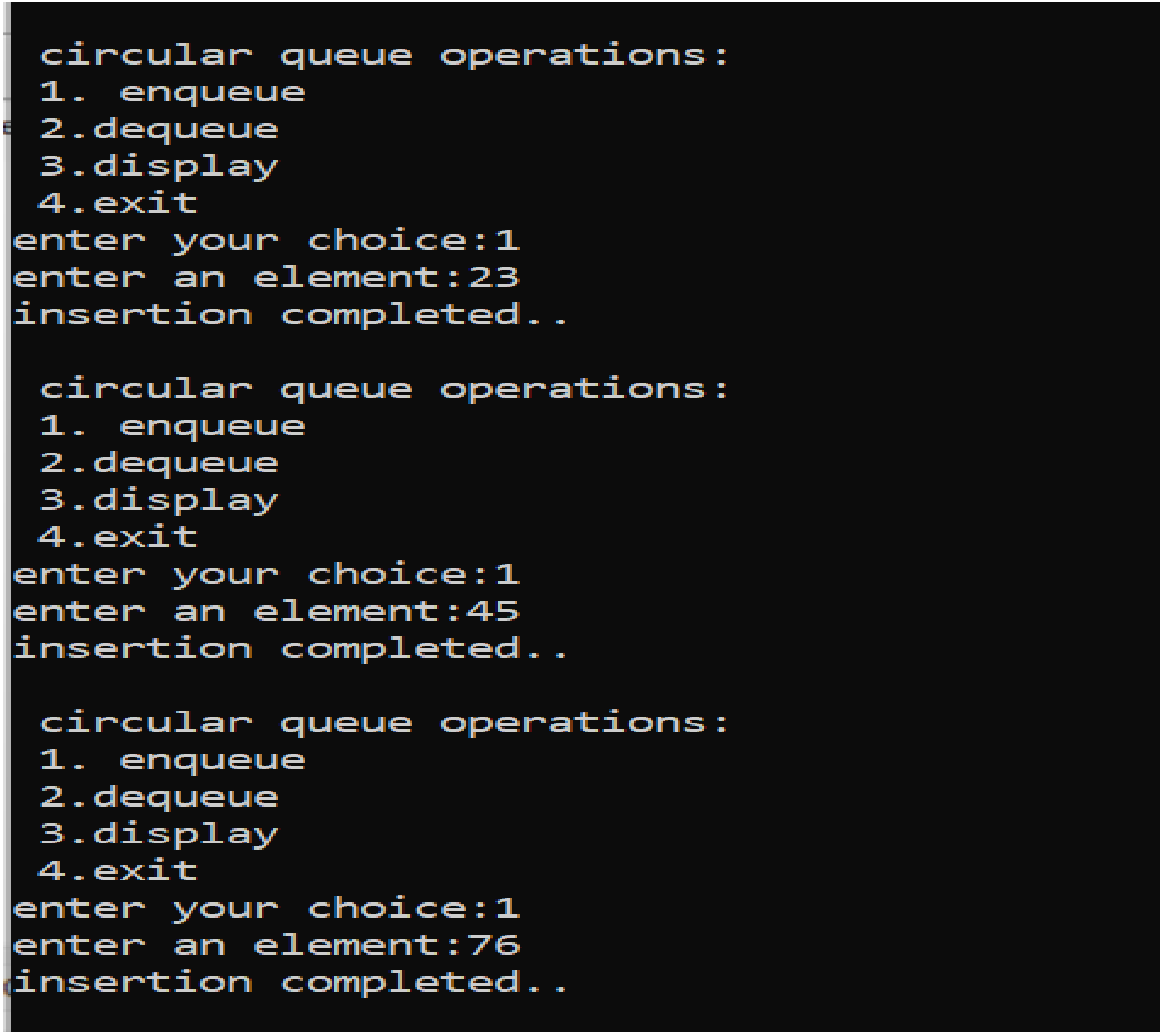
Aim: Write a program in c++ to implement Circular Queue .

```
#include<iostream>
#define size 4
using namespace std;
class CircularQ
{
    public:
        int CQ[size];
        int front,rear;
        CircularQ();
        void enqueue();
        void dequeue();
        void display();
};
CircularQ::CircularQ()
{
    front=rear=-1;
}
void CircularQ::enqueue()
{
    int element;
    //if(rear==size-1)//rear=3,0,1,2,3,
    //cout<<"queue is full";
    if(front==(rear+1)%size)
    //currently rear = 0
    //next element should be inserted at rear+1=1
    //1 index is not empty, then we cann't insert
    //rear=(rear+1)%maxSize==front index
    {
        cout<<"Queue is full\n";
        return;//return from method
    }
    else
    {
        cout<<"enter an element:";
        cin>>element;
        if(front==-1)//queue is empty
        rear=front=0;//initialize rear and front value with 0
        else
        rear=(rear+1)%size;
    }
    CQ[rear]=element;
    cout<<"insertion completed..\n";
}
void CircularQ::dequeue()
{
    int num;
    //underflow, if the queue is empty, no deletion
```

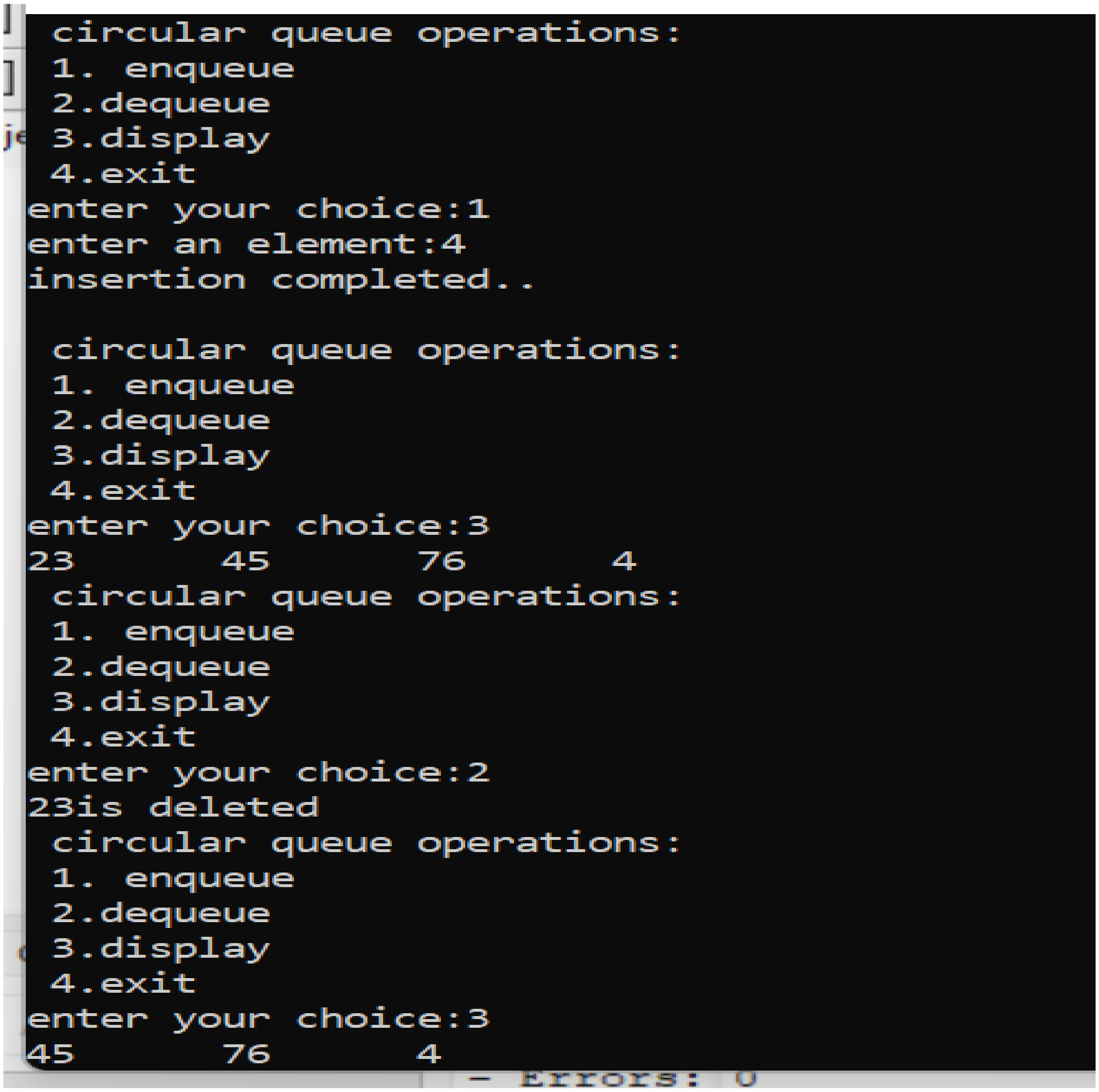
```
        if(front==-1)
        cout<<"queue is empty\n";
        else
        {
                num=CQ[front];
                cout<<num<<"is deleted";
                if(front==rear)
                front=rear=-1;
                else
                front=(front+1)%size;//currently front=3,rear=2
                //front=(3+1)%4=0
        }
}
void CircularQ::display()
{
        int i;
        if(front==-1)
        {

        cout<<"queue is empty";
        return;
        }
        else
        {  //this is applicable only for when front<rear
                for(i=front;i<=rear;i++)//front=3; rear=2
                cout<<CQ[i]<<"\t";
        }
        //when front>rear
        if(front>rear)//front=3, rear=2
        {

        for(i=front;i<size;i++)//front=3;3<4,int second iteration i=4<4
        cout<<CQ[i]<<"\t";//CQ[3]
        //CQ[0],CQ[1],CQ[2]
        for(i=0;i<=rear;i++)
        cout<<CQ[i]<<"\t";
        }

}
int main()
{
        CircularQ cq;
        int choice;
        while(1)
        {
                cout<<"\n circular queue operations:";
                cout<<"\n 1. enqueue \n 2.dequeue \n 3.display \n 4.exit\n";
                cout<<"enter your choice:";
                cin>>choice;
                switch(choice)
                {
                        case 1:cq.enqueue();
```

```
                          break;
                     case 2:cq.dequeue();
                                break;
                     case 3:cq.display();
                                 break;
                     case 4:exit(0);
                     default: cout<<"wrong choice";
             }
      }
}
```

Output:



```
  circular queue operations:
  1. enqueue
  2.dequeue
  3.display
  4.exit
enter your choice:1
enter an element:23
insertion completed..

  circular queue operations:
  1. enqueue
  2.dequeue
  3.display
  4.exit
enter your choice:1
enter an element:45
insertion completed..

  circular queue operations:
  1. enqueue
  2.dequeue
  3.display
  4.exit
enter your choice:1
enter an element:76
insertion completed..
```



```
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter an element:4
insertion completed..

 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:3
23      45      76      4
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:2
23is deleted
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:3
45      76      4
```

Practical No. 7: Demonstrate application of queue

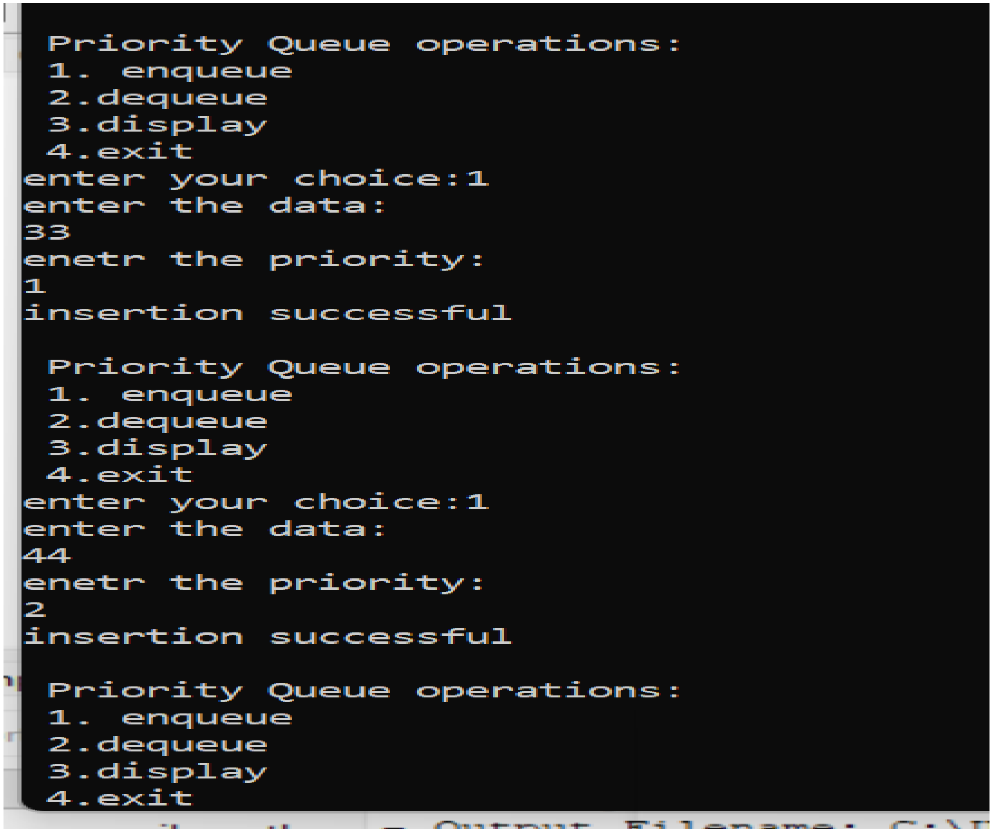
Aim: Write a program in c++ to implement Circular Queue .

```
#include<iostream>
using namespace std;
class PriorityQ
{
	struct Node
	{
	 int data;
	 int prio;
	 Node* next;
	};
	struct Node* head;
	Node *front,*rear;
	public:
		PriorityQ();
	 void enqueue();
	 void dequeue();
	 void display();

};
PriorityQ::PriorityQ()
{
	front=rear=NULL;
}
void PriorityQ::enqueue()
{
	int num,num_prio;
	cout<<"enter the data:"<<endl;
	cin>>num;
	cout<<"enetr the priority:"<<endl;
	cin>>num_prio;
	struct Node* temp,*q;
	temp=new struct Node;//p2
	temp->data=num;
	temp->prio=num_prio;
	temp->next=NULL;
	if(front==NULL||num_prio<front->prio)
	{
		temp->next=front;
		front=temp;//p1
	}
	else
	{
	q=front;
	while(q->next!=NULL&&q->next->prio<=num_prio)
	{
	q=q->next;
```

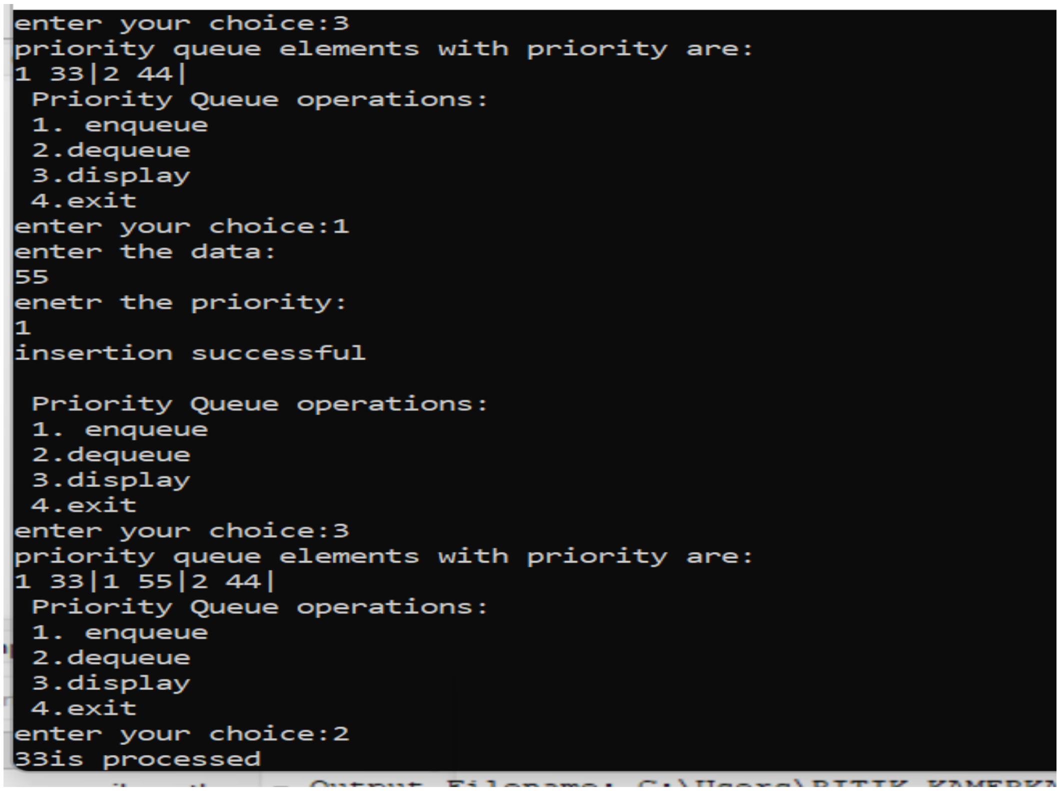
```
    }
    temp->next=q->next;
    q->next=temp;
    }
    cout<<"insertion successful\n";
}
void PriorityQ::dequeue()
{
    Node* temp;
    if(front==NULL)
    cout<<"nothing is there to process\n";
    else
    {
        temp=front;
        cout<<temp->data<<"is processed";
        front=front->next;
        delete(temp);
    }
}
void PriorityQ::display()
{
    Node* q;
    q=front;
    if(front==NULL)
    cout<<"nothing is there to display\n";
    else
    {
        cout<<"priority queue elements with priority are:"<<endl;
        while(q!=NULL)
        {
            cout<<q->prio<<" "<<q->data<<"|";
            q=q->next;
        }
    }
}
int main()
{
    PriorityQ q;
    int choice;
    while(1)
    {
        cout<<"\n Priority Queue operations:";
        cout<<"\n 1. enqueue \n 2.dequeue \n 3.display \n 4.exit\n";
        cout<<"enter your choice:";
        cin>>choice;
        switch(choice)
        {
            case 1:q.enqueue();
                  break;
            case 2:q.dequeue();
                     break;
            case 3:q.display();
```

```
                                    break;
                        case 4:exit(0);
                        default: cout<<"wrong choice";
                }
        }
}
```

Output:

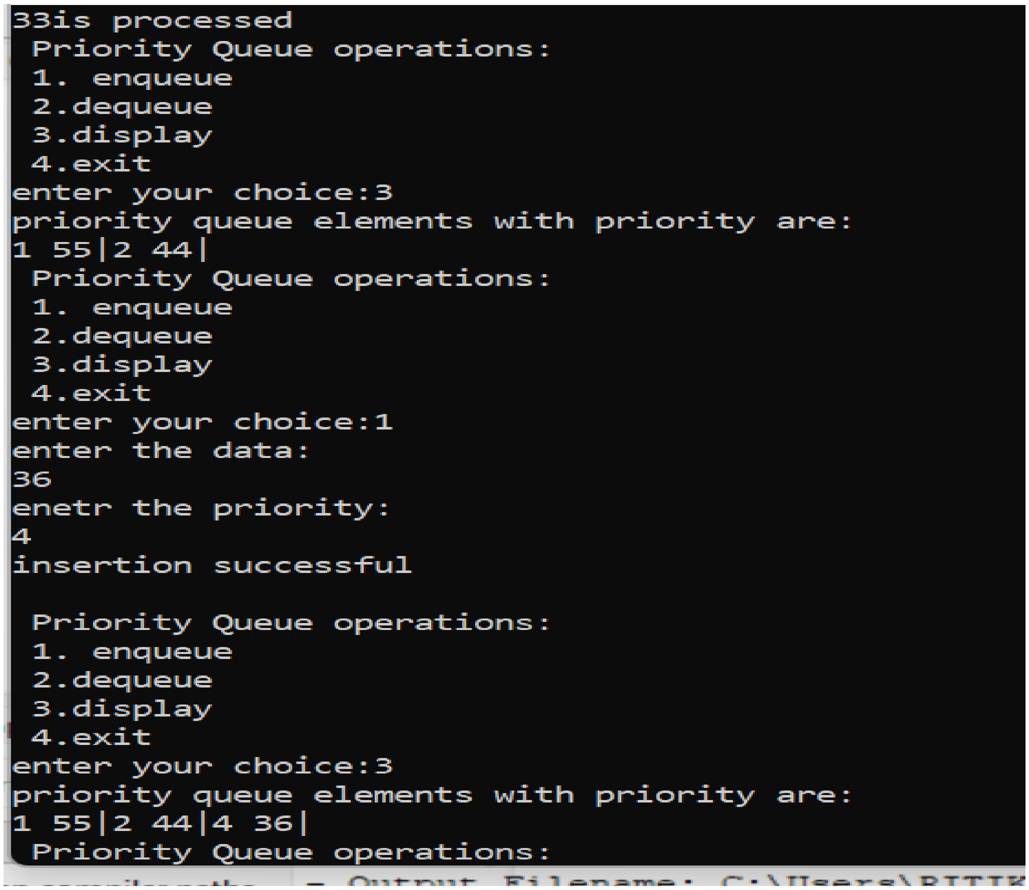


```
 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:
33
enetr the priority:
1
insertion successful

 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:
44
enetr the priority:
2
insertion successful

 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
```



```
enter your choice:3
priority queue elements with priority are:
1 33|2 44|
 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:
55
enetr the priority:
1
insertion successful

 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:3
priority queue elements with priority are:
1 33|1 55|2 44|
 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:2
33is processed
```

- Output Filename: C:\Users\RITIK KAMERKA



```
33is processed
 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:3
priority queue elements with priority are:
1 55|2 44|
 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:
36
enetr the priority:
4
insertion successful

 Priority Queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:3
priority queue elements with priority are:
1 55|2 44|4 36|
 Priority Queue operations:
```

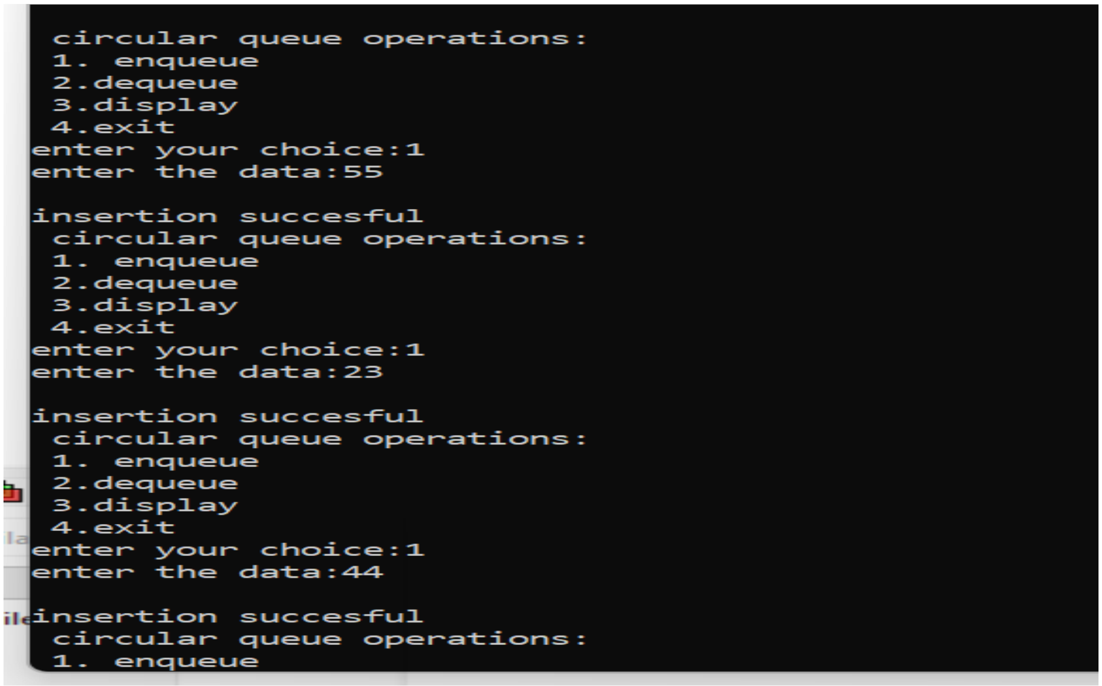
- Output Filename: C:\Users\RITIK

Aim: Write a program in c++ to implement Queue using Linked list .

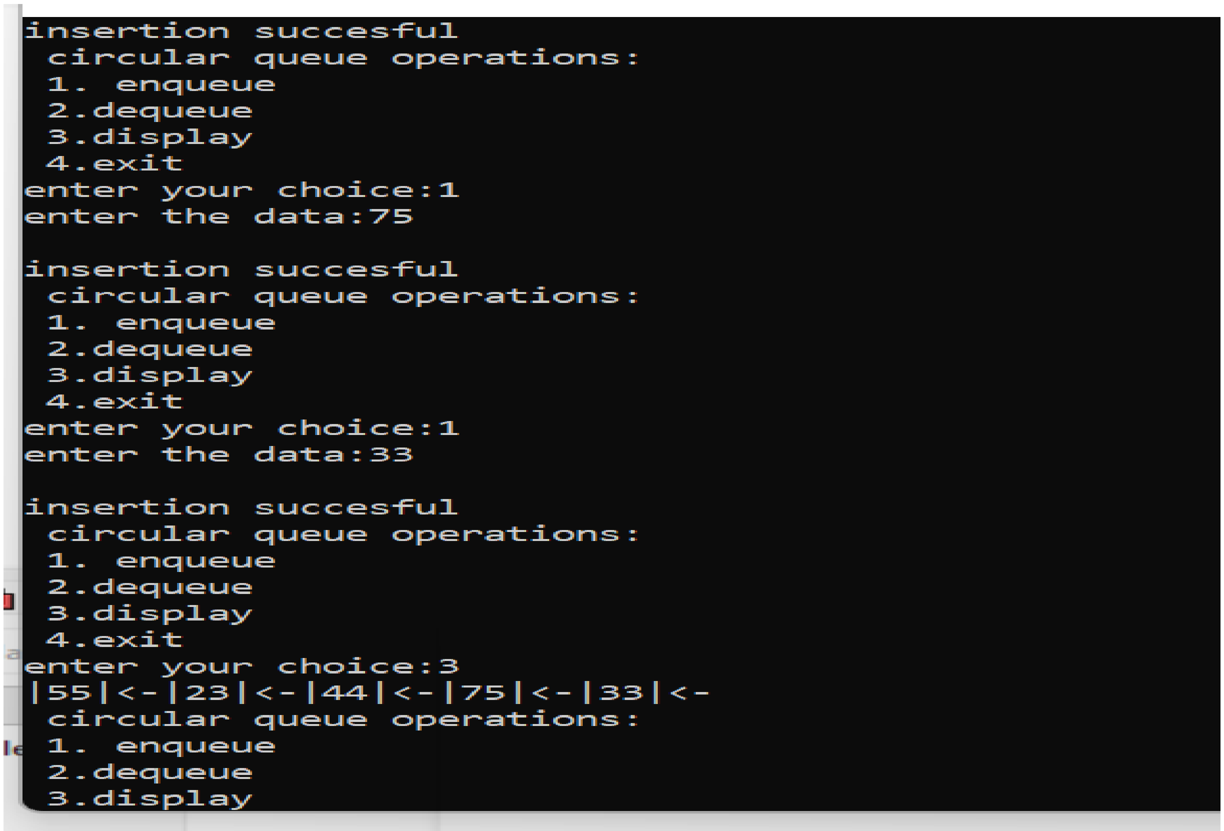
```
//Queue implementation using Linked list
#include<iostream>
using namespace std;
class QueueLink
{
	struct Node
	{
		int data;
		struct Node* next;
	};
	struct Node* front, *rear;
	int num;
	public:
		QueueLink();
		void enqueue();
		void dequeue();
		void display();

};
QueueLink::QueueLink()
{
	front=rear=NULL;
}
void QueueLink::enqueue()
{
	struct Node* NewNode;//declaration of newnode
	cout<<"enter the data:";
	cin>>num;
	NewNode=new struct Node;//creation of newnode
	NewNode->data=num;
	NewNode->next=NULL;
	if(front==NULL)//is queue empty
	{
		front=rear=NewNode;//it is similar to front=rear=0
	}//executed at first time insertion
	else
	{
		rear->next=NewNode;//previous NewNode->next=current NewNode
		rear=NewNode;
	}
	cout<<"\ninsertion succesful";

}
void QueueLink::dequeue()
{
	struct Node *temp;
```

```
    if(front==NULL)//queue is empty
    cout<<"queue is empty";
    else
    {
        num=front->data;//storing the data in num
        //now store the adress of front in temp
        temp=front;
        front=front->next;//new front
        cout<<"deletion successful";
    }
    delete(temp);//free the memory space
}
void QueueLink::display()
{
    struct Node* temp;
    if(front==NULL)
    cout<<"queue is empty, nothing is there to dispaly\n";
    else
    {
    temp=front;
    while(temp!=NULL)
    {
        cout<<"|"<<temp->data<<"|<-";
        temp=temp->next;
    }
    }
}
int main()
{
    QueueLink q;
    int choice;
    while(1)
    {
        cout<<"\n circular queue operations:";
        cout<<"\n 1. enqueue \n 2.dequeue \n 3.display \n 4.exit\n";
        cout<<"enter your choice:";
        cin>>choice;
        switch(choice)
        {
            case 1:q.enqueue();
              break;
            case 2:q.dequeue();
                 break;
            case 3:q.display();
                 break;
            case 4:exit(0);
            default: cout<<"wrong choice";
        }
    }
}
```

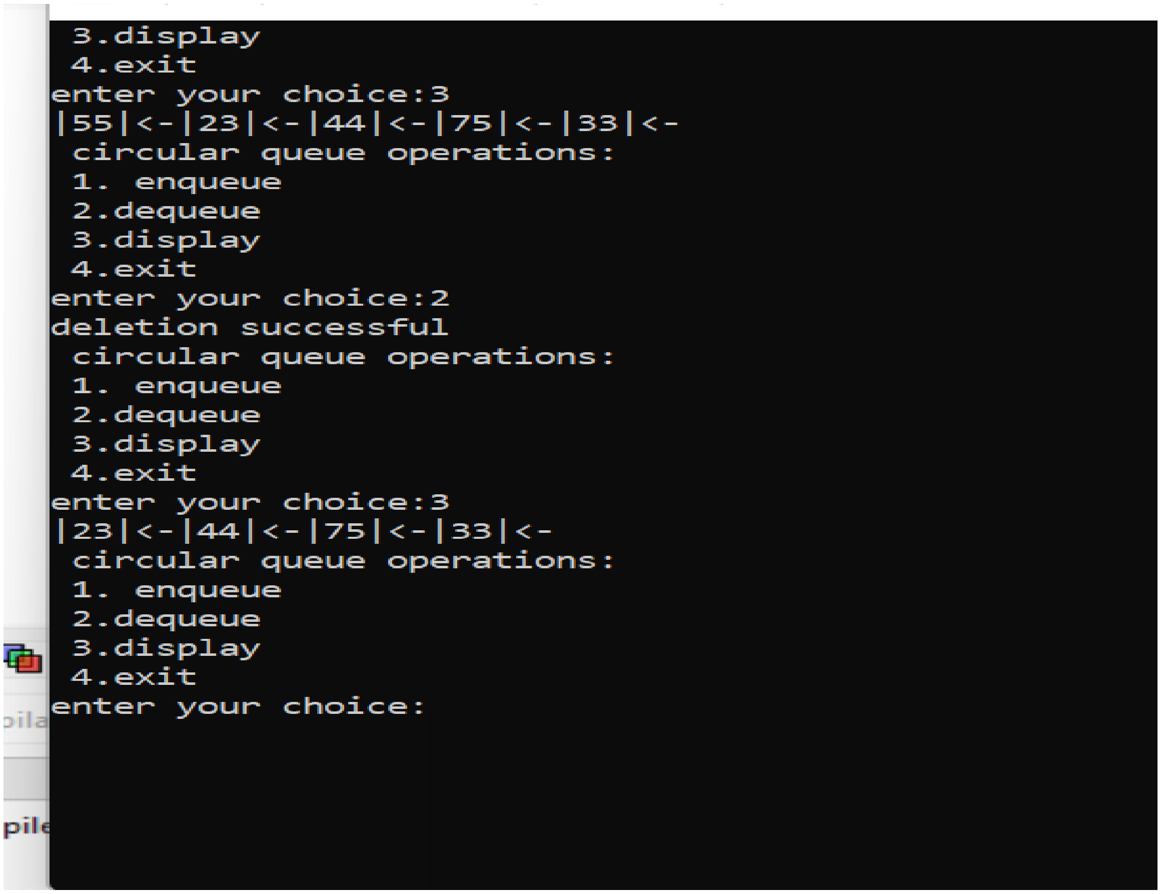
Output:



```
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:55

insertion succesful
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:23

insertion succesful
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:44

insertion succesful
 circular queue operations:
 1. enqueue
```



```
insertion succesful
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:75

insertion succesful
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:1
enter the data:33

insertion succesful
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:3
|55|<-|23|<-|44|<-|75|<-|33|<-
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
```



```
 3.display
 4.exit
enter your choice:3
|55|<-|23|<-|44|<-|75|<-|33|<-
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:2
deletion successful
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:3
|23|<-|44|<-|75|<-|33|<-
 circular queue operations:
 1. enqueue
 2.dequeue
 3.display
 4.exit
enter your choice:
```

Practical No. 8: Demonstrate application of linked list

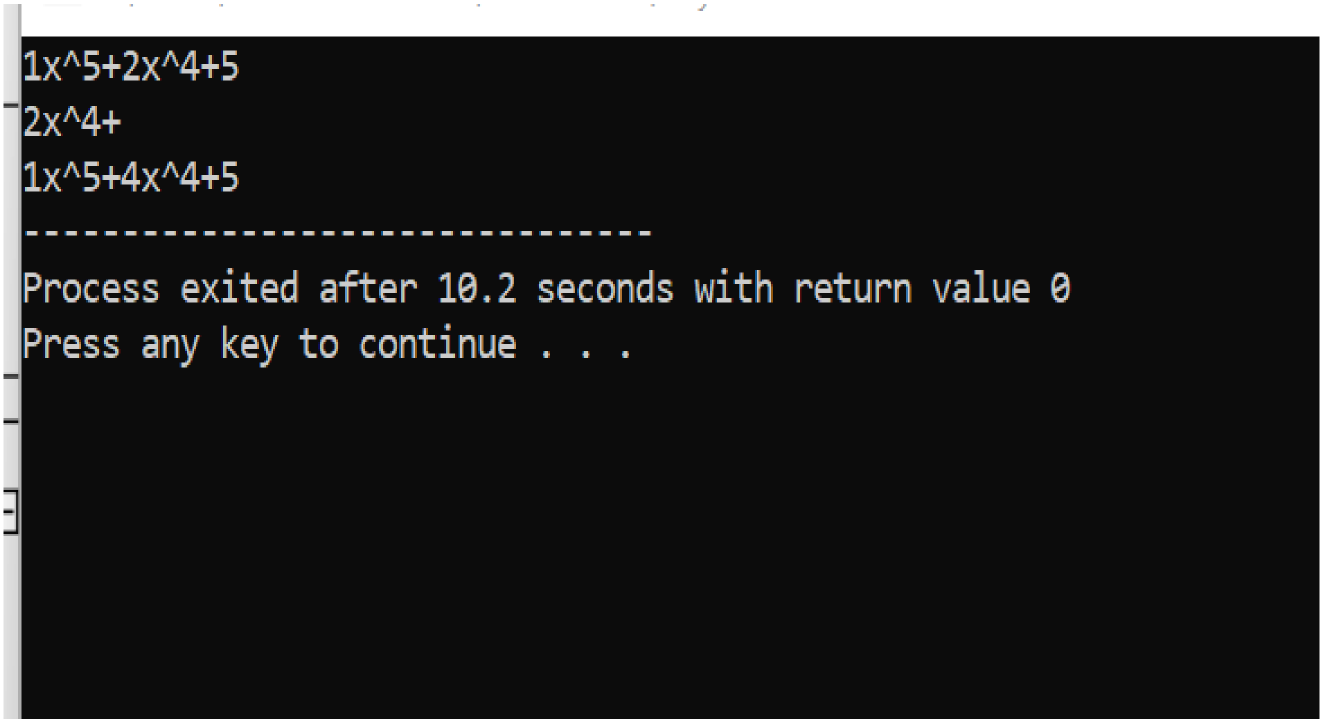
Aim: Write a program in c++ to implement Polynaomial Addition.

```
#include<iostream>
using namespace std;
class PolyAdd
{
struct PolyNode
{
        float coeff;
        int exp;
        PolyNode* link;
};
struct PolyNode* head;
public:
        PolyAdd();
        void createPoly(float c,int e);
        void displayPoly();
        void addPoly(PolyAdd &p1,PolyAdd &p2);

};
PolyAdd::PolyAdd()
{
        head=NULL;
}
void PolyAdd::createPoly(float c,int e)
{
        PolyNode* temp,*q;
        temp=new struct PolyNode;
        temp->coeff=c;
        temp->exp=e;
        temp->link=NULL;
        if(head==NULL||e>head->exp)//there is no element in the list
        {
        temp->link=head;
        head=temp;
        }
        else
        {
                q=head;
                while(q->link!=NULL&&q->link->exp>e)
                {
                        q=q->link;
                }
                        temp->link=q->link;
                q->link=temp;


        }
}
```

```
void PolyAdd::addPoly(PolyAdd &l1,PolyAdd &l2)
{
        struct PolyNode* result;
        //result will be created dynamically after addition
        if(l1.head==NULL&&l2.head==NULL)//both polynomials no value
        return;
        PolyNode* temp1,*temp2;
        temp1=l1.head;
        temp2=l2.head;
        while(temp1!=NULL && temp2!=NULL)
        {//creation of result dynamically
                if(head==NULL)
                {
                head=new PolyNode;
                result=head;
                }
                else
                {
                        result->link=new PolyNode;
                        result=result->link;
                }
                if(temp1->exp < temp2->exp)//temp1->exp=3
                //temp2->exp=4
                {
                        result->coeff=temp2->coeff;
                        result->exp=temp2->exp;
                        temp2=temp2->link;
                }
                else if(temp1->exp >temp2->exp)//temp1-exp=4
                //temp2->exp=3
                {
                result->coeff=temp1->coeff;
                        result->exp=temp1->exp;
                        temp1=temp1->link;
                }
                else
                {
                        if(temp1->exp==temp2->exp)//temp1->exp=3
                        //temp->exp=3, ex. 2x^3 +3x^3=5x^3
                        {
                                result->coeff=temp1->coeff+temp2->coeff;
                                result->exp=temp1->exp;
                                temp1=temp1->link;
                                temp2=temp2->link;
                        }
                }
        }
        while(temp1!=NULL)//for remaining part of the temp1
        {
                if(head==NULL)
                {
                head=new PolyNode;
                result=head;
                }
                else
                {
                        result->link=new PolyNode;
                        result=result->link;
                }
                result->coeff=temp1->coeff;
                result->exp=temp1->exp;
                temp1=temp1->link;
        }
        while(temp2!=NULL)
        {
                if(head==NULL)
                {
                head=new PolyNode;
                result=head;
                }
                else
                {
                        result->link=new PolyNode;
                        result=result->link;
                }
                result->coeff=temp2->coeff;
                result->exp=temp2->exp;
                temp2=temp2->link;

        }
        result->link=NULL;
    }
    void PolyAdd::displayPoly()
    {
        PolyNode* q;
        q=head;
        while(q!=NULL)//q->link!=NULL
        {
                if(q->exp!=0)
                {
                        cout<<q->coeff<<"x^"<<q->exp;
                        cout<<"+";
                }
                else

                cout<<q->coeff;

                q=q->link;
        }
    }
    int main()
    {
        PolyAdd p1,p2,Result;
        p1.createPoly(2,4);//2x^4
```

```
        p1.createPoly(1,5);//x^5
        p1.createPoly(5,0);//5x^0
        p1.displayPoly();
        cout<<endl;
        p2.createPoly(2,4);
        p2.displayPoly();
        cout<<endl;
        Result.addPoly(p1,p2);
        Result.displayPoly();
        return 0;
}
```

Output:



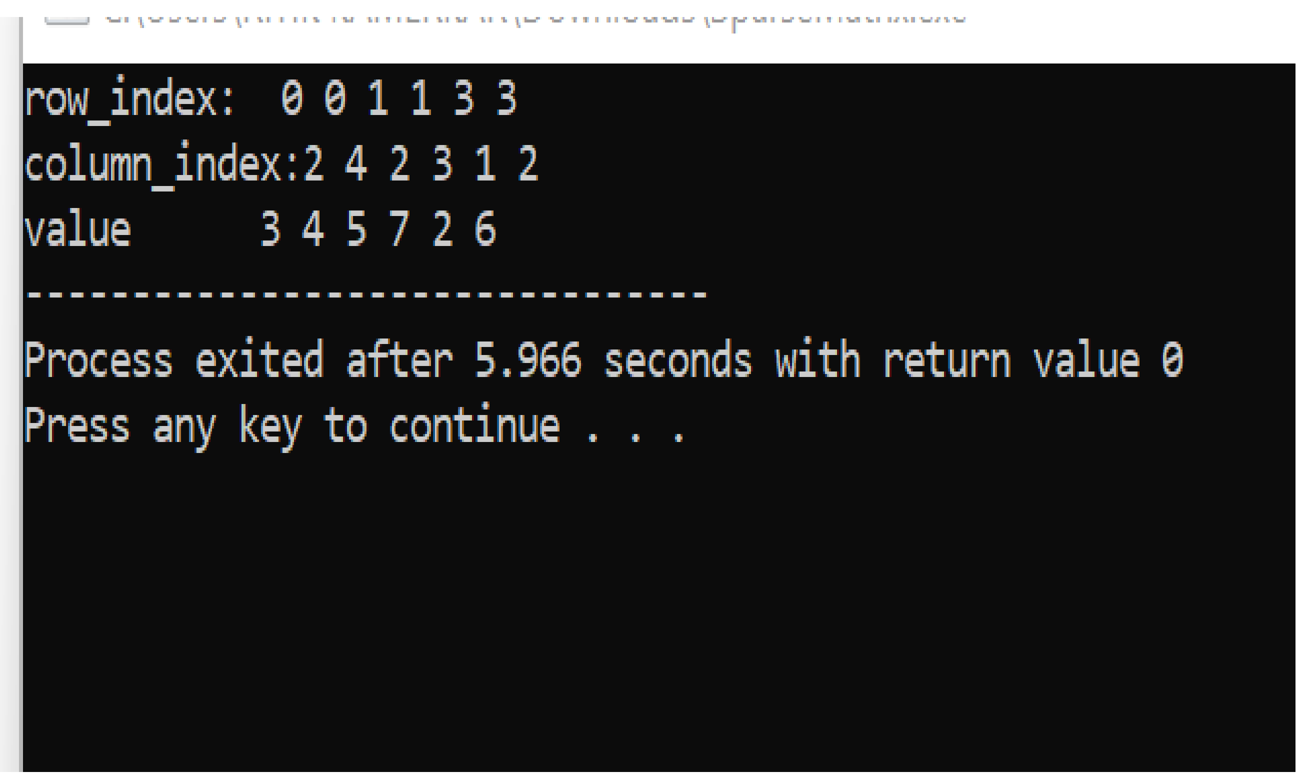
```
1x^5+2x^4+5
2x^4+
1x^5+4x^4+5
--------------------------------
Process exited after 10.2 seconds with return value 0
Press any key to continue . . .
```

Aim: Write a program in c++ to implement Sparse Matrix.

```
//sparse matrix representation in memory using linked list
#include<iostream>
using namespace std;
class SparseMatrix
{
	//create a node to store the values
	//row, col,value, link for the next node
	struct Node
	{
		int row;
		int col;
		int value;
		Node* next;
	};
	struct Node* first;
	public:
		SparseMatrix()
		{
			first=NULL;
		}
		void createNewNode(int row_index,int col_index, int val);
		void printList();

};
void SparseMatrix::createNewNode(int row_index,int col_index,int val)
{
	Node *temp,*q;
	temp=new Node();
	temp->row=row_index;
	temp->col=col_index;
	temp->value=val;
	temp->next=NULL;
	if(first==NULL)//it will excuted only once at first time
	{
	first=temp;
	}
	else
	{
		q=first;
		while(q->next!=NULL)//to reach at the last of the list
		q=q->next;
		q->next=temp;

	}
}
void SparseMatrix::printList()
{

	Node *q;
```

```
    if(first==NULL)
    {
        cout<<"list is empty";
        return;
    }
    q=first;
    cout<<"row_index:"<<"  ";
    //travel from first node to last node, and retrieve only row index value
    while(q!=NULL)
    {
        cout<<q->row<<" ";
        q=q->next;
    }
    cout<<endl;
    cout<<"column_index:";
    q=first;
    while(q!=NULL)
    {
        cout<<q->col<<" ";
        q=q->next;
    }
    cout<<endl;
    cout<<"value"<<"      ";
    q=first;
    while(q!=NULL)
    {
        cout<<q->value<<" ";
        q=q->next;
    }

}
int main()
{
    SparseMatrix s;
    int sparseMatrix[4][5]={{0,0,3,0,4},
                            {0,0,5,7,0},
                            {0,0,0,0,0},
                            {0,2,6,0,0}};
                            //0        0 1       1 3       3
                            //2 4 2 3 1 2
                            //3 4 5 7 2 6
            for(int i=0;i<4;i++)//loop will execute from 0 to 3// loop is for row
            {
                for(int j=0;j<5;j++)
                {
                    //pass only those values which are non zero
                    if(sparseMatrix[i][j]!=0)
                     s.createNewNode(i,j,sparseMatrix[i][j]);
                }
            }
            s.printList();
            return 0;
}
```

Output:



```
row_index:  0 0 1 1 3 3
column_index:2 4 2 3 1 2
value      3 4 5 7 2 6
--------------------------------
Process exited after 5.966 seconds with return value 0
Press any key to continue . . .
```

Practical No. 9: Create and perform various operations on BST

Aim: Write a program in c++ to implement Binary Search Tree perform following operation.

a) **Inserting node in BST**
b) **Deleting the node from BST**
c) **To find height of Tree**
d) **To perform Inorder**
e) **To perform Preorder**
f) **To perform Postorder**

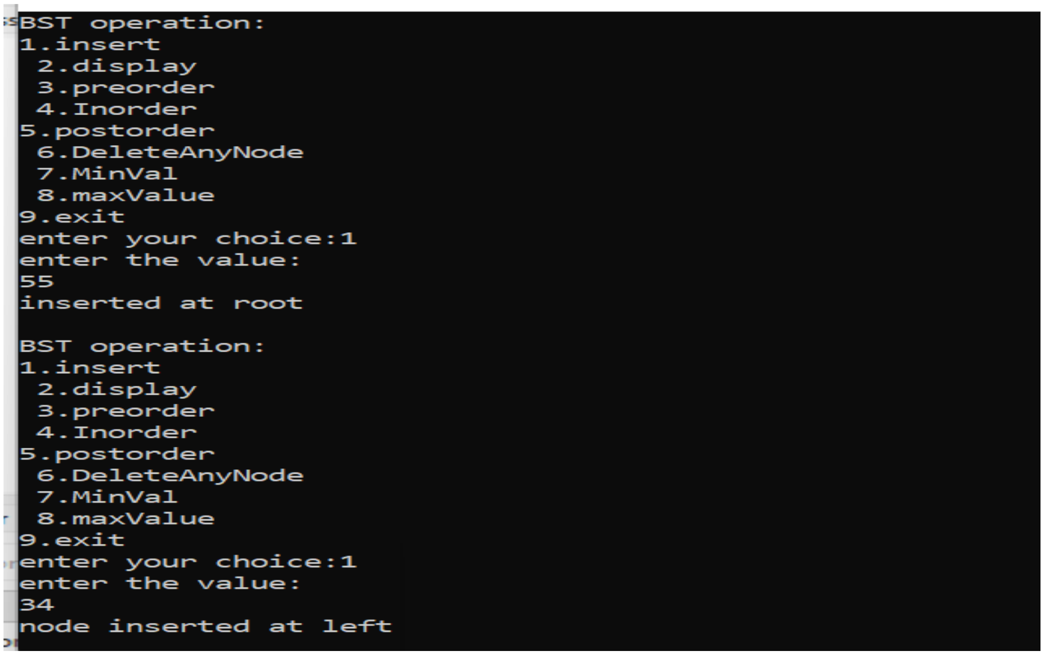
```
#include<iostream>
#define SPACE 5
using namespace std;
class TreeNode
{
        public:
                int val;
                TreeNode* lchild;
                TreeNode* rchild;
                TreeNode()
                {
                        val=0;
                        lchild=NULL;
                        rchild=NULL;
                }

};
class BST
{
public:
TreeNode* root;//one object type pointer of TreeNode
//struct Node* root;
BST()
{
        root=NULL;//tree is empty
}
void insertNode(TreeNode* newNode)//make a tree
{
        //is tree empty or not
        if(root==NULL)//this block will executed only once at start
        {
                root=newNode;
                cout<<"inserted at root\n";
        }
        else//already nodes are there in the tree
        {
                TreeNode* temp = root;
                while(temp!=NULL)
                {
```

```
            if(newNode->val==temp->val)
            {
                cout<<"value is duplicate\n";
                return;
            }
            else if(newNode->val<temp->val&&temp->lchild==NULL)
            {
                temp->lchild=newNode;
                cout<<"node inserted at left\n";
                break;
            }
            else if(newNode->val<temp->val)//this is for finding the last node
            {
                temp=temp->lchild;//50,20,
            }
            else if(newNode->val>temp->val&&temp->rchild==NULL)
            {
                temp->rchild=newNode;
                cout<<"node is inserted at right\n";
                break;
            }
            else

            {
                if(newNode->val>temp->val)
                temp=temp->rchild;
            }
        }

    }

    }
    void display(TreeNode* r, int space)
    {
    if(r==NULL)
    //return -1;
    return;
    space+=SPACE;//5+5=15
    display(r->rchild,space);
    cout<<endl;
    for(int i=SPACE;i<space;i++)//moving the cursor at cmd window
    cout<<" ";

    cout<<r->val<<"\n";

    display(r->lchild,space);
    }
void printPreorder(TreeNode* r)//r will be current root node
{
    if(r==NULL)//r=root
    {
        //cout<<"tree is empty\n";
        return;
    }
    cout<<r->val<<" ";
    printPreorder(r->lchild);
    printPreorder(r->rchild);
}
void printInorder(TreeNode* r)//left,Root,Right
{
    if(r==NULL)
    return;
    printInorder(r->lchild);
    cout<<r->val<<" ";
    printInorder(r->rchild);

}
void printPostorder(TreeNode* r)//Left,Right,Root
{
    if(r==NULL)
    return;
    printPostorder(r->lchild);
    printPostorder(r->rchild);
    cout<<r->val<<" ";

}
TreeNode* minValueNode(TreeNode* node)
{
    //travel the tree in left side
    TreeNode* curr = node;
    while(curr->lchild!=NULL)
    curr=curr->lchild;
    return curr;
}
TreeNode* deleteNode(TreeNode* r,int v)//r=addressof node,v=value of node
{
    bool found =false;

    if(root==NULL)
    {
        cout<<"tree is empty"<<endl;//this is for original tree
        return NULL;
    }
    TreeNode* curr;
    curr=root;
    while(curr!=NULL)
    {
        if(curr->val==v)
        {
            found=true;
            break;
        }
        else
        {
```

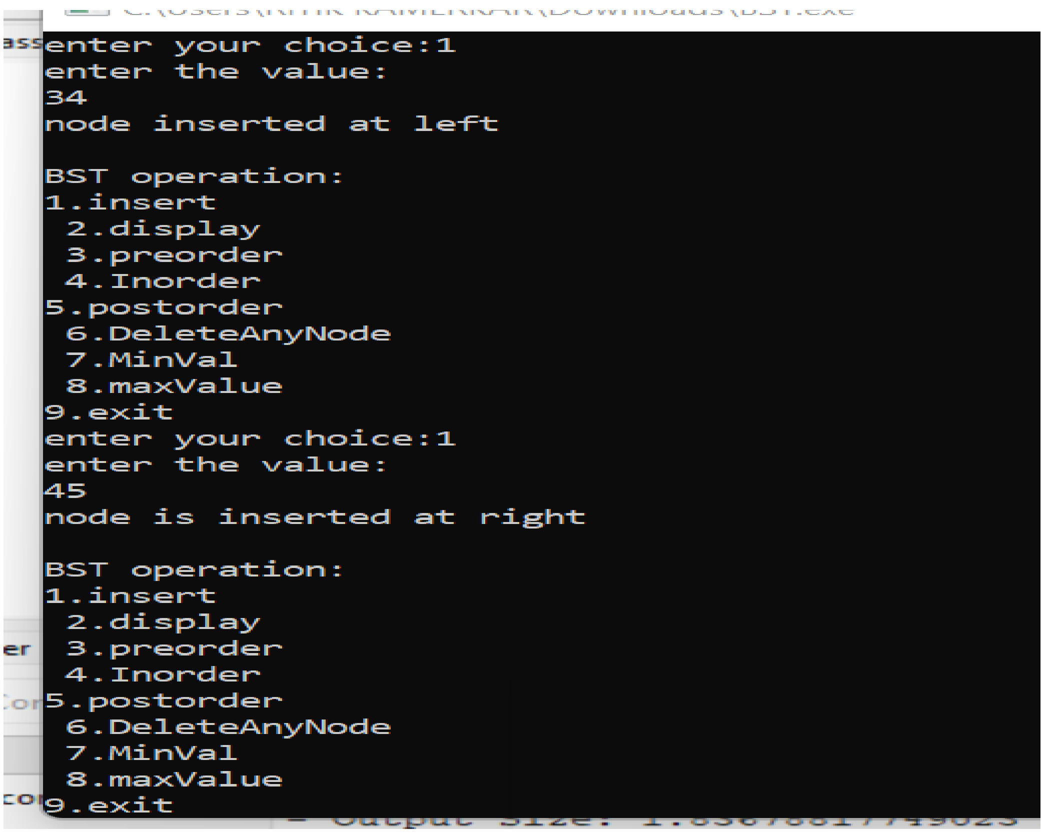
```
        if(v>curr->val)
        curr=curr->rchild;
        else
        curr=curr->lchild;
    }
}
if(found==false)
{
    cout<<"the value is not present"<<endl;
    return NULL;
}
//the tree is existing
if(r==NULL)
{
    return NULL;
}
//r has some address value,e.g r=0x5ff3008
else if(v<r->val)
{
r->lchild=deleteNode(r->lchild,v);
}
else if(v>r->val)
{
    r->rchild=deleteNode(r->rchild,v);
}
else//you found the node proceeding for delete
{
    //case-1
    if(r->lchild==NULL && r->rchild==NULL)
    {
        delete r;
        r=NULL;
        //return r;
    }
    else if(r->lchild==NULL)//only right child(leaf node) exists
    {
        TreeNode* temp=r;
        r=r->rchild;
        delete temp;
        //return r;
    }
    else if(r->rchild==NULL)//only left child (leaf node) exist
    {
        TreeNode* temp =r;
        r=r->lchild;
        delete temp;
        //return r;
    }
    //case -3
    else
    {//we have to select randomly left sub tree or right subtree
        TreeNode* temp = minValueNode(r->rchild);
```

```
            r->val=temp->val;
            r->rchild=deleteNode(r->rchild,temp->val);
            //TreeNode* temp=maxValNode(r->lchild)
            //r->val=temp->val;
            //r->lchild=deleteNode(r->lchild,temp->val)
        //  return r;
        }
    }
    return r;

}
};
int main()
{
    BST obj;
    int value,choice;
    while(1)
    {
        cout<<"BST operation:"<<endl;
        cout<<"1.insert\n 2.display\n 3.preorder\n 4.Inorder\n";
        cout<<"5.postorder\n 6.DeleteAnyNode\n 7.MinVal\n 8.maxValue\n";
        cout<<"9.exit\n";
        cout<<"enter your choice:";
        cin>>choice;
        TreeNode* newNode=new TreeNode();
        switch(choice)
        {
            case 1:
                cout<<"enter the value:"<<endl;
                cin>>value;

                newNode->val=value;
                obj.insertNode(newNode);
                cout<<endl;
                break;
            case 2:
                    obj.display(obj.root,5);
                    break;
            case 3: obj.printPreorder(obj.root);
                    break;
            case 4: obj.printInorder(obj.root);
                    break;
            case 5: obj.printPostorder(obj.root);
                    break;
            case 6: cout<<"enter the value to be deleted:"<<endl;
                    cin>>value;
                    obj.deleteNode(obj.root,value);
                    break;
            case 9:  exit(0);
            default: cout<<"wrong choice";

        }
        }


        return 0;
}
```

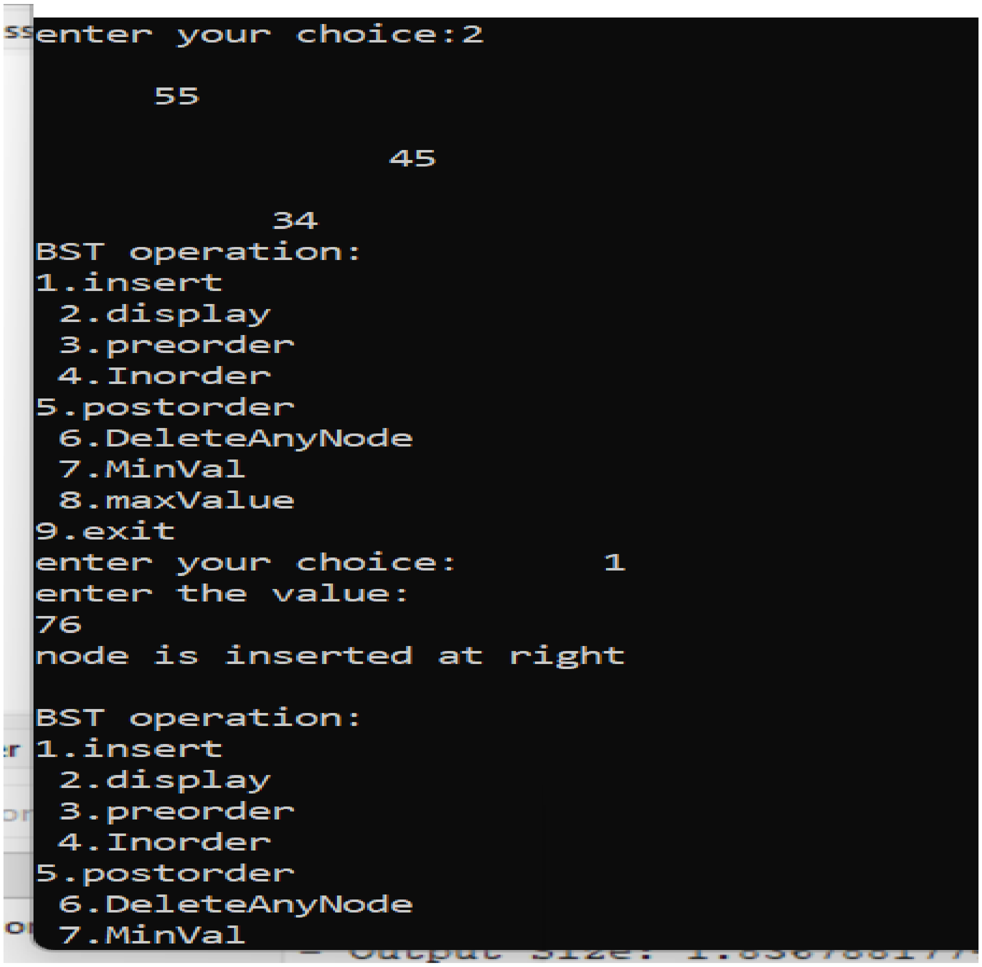
Output:



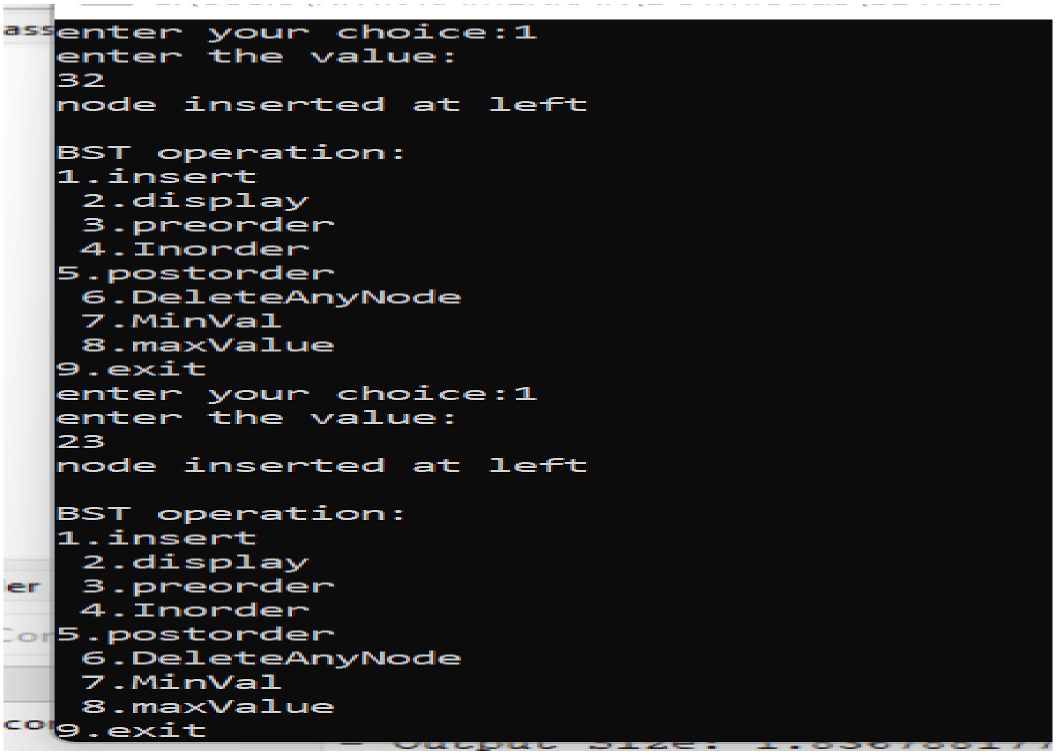
```
BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:1
enter the value:
55
inserted at root

BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:1
enter the value:
34
node inserted at left
```



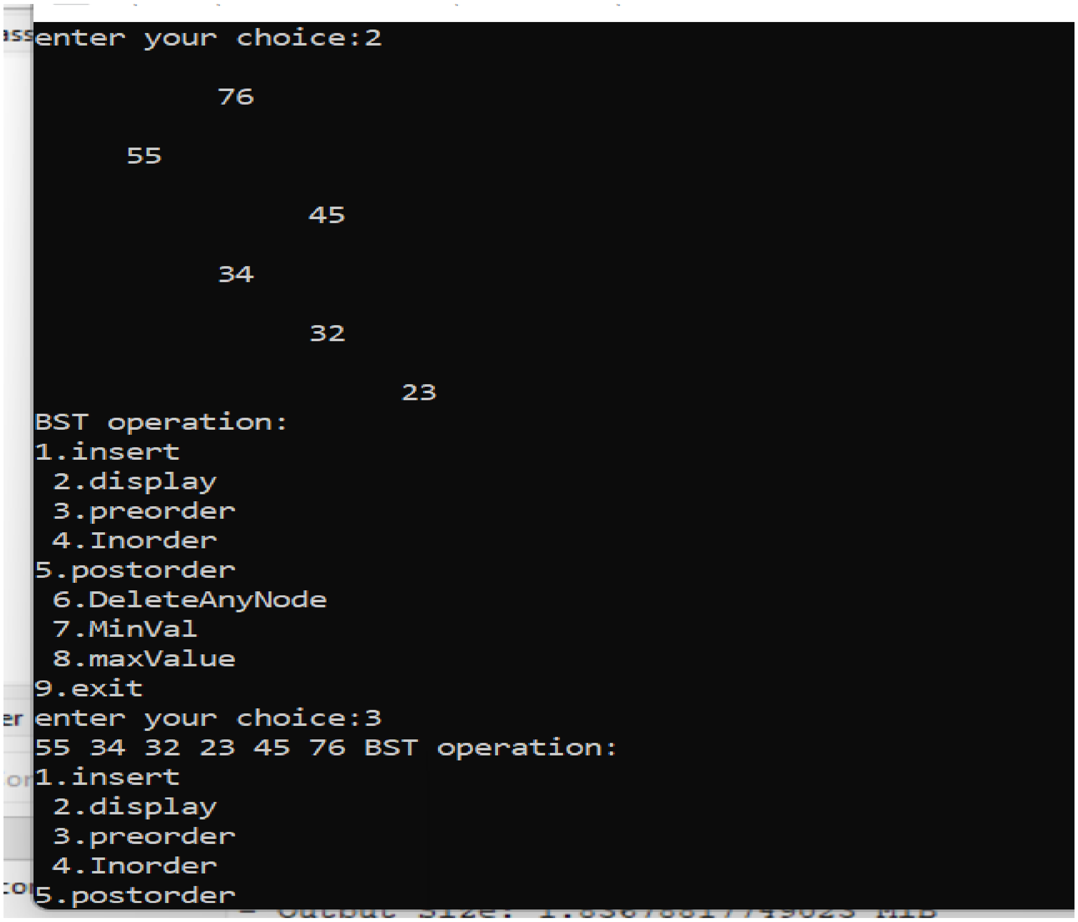
```
enter your choice:1
enter the value:
34
node inserted at left

BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:1
enter the value:
45
node is inserted at right

BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
```



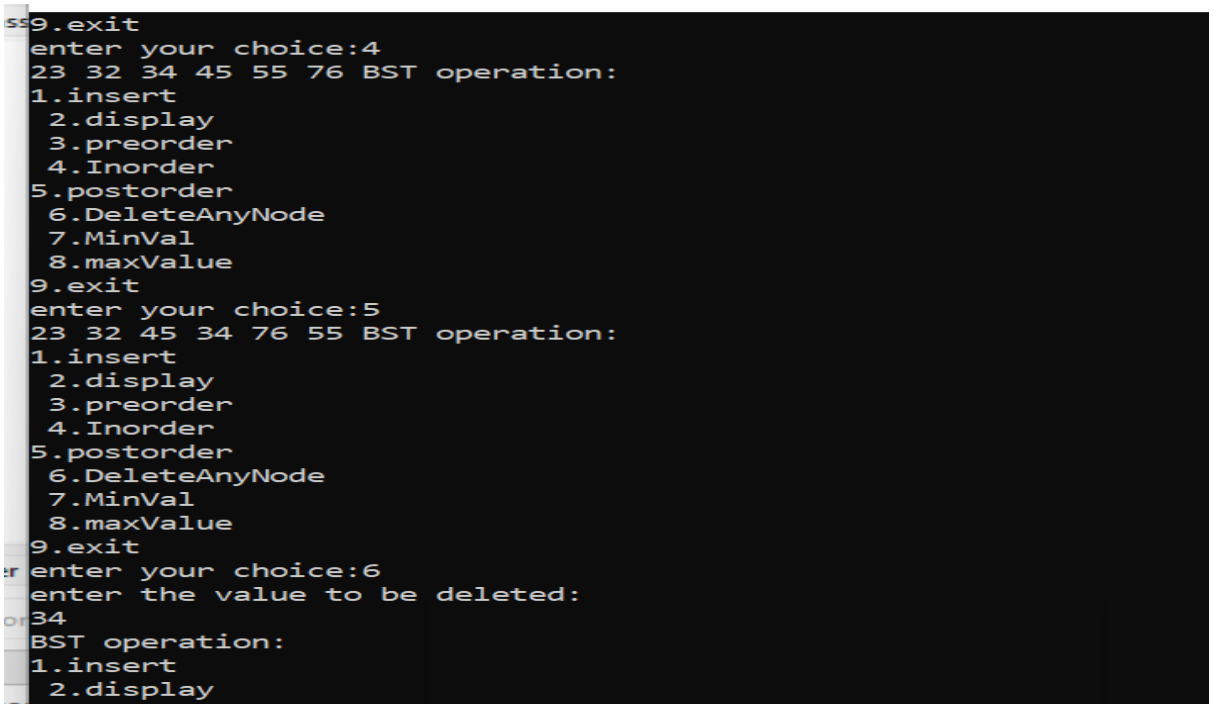
```
enter your choice:2

     55

               45

          34
BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:      1
enter the value:
76
node is inserted at right

BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
```



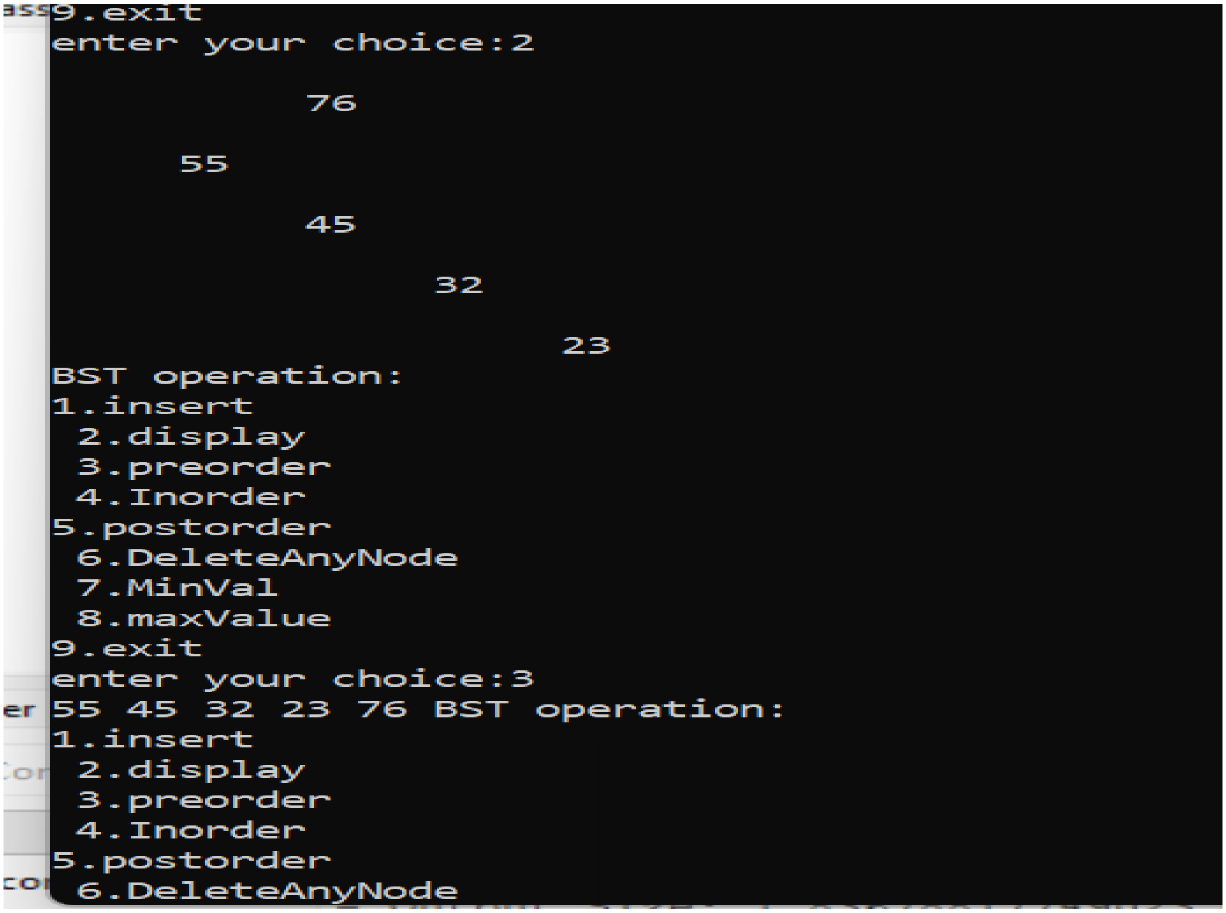
```
enter your choice:1
enter the value:
32
node inserted at left

BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:1
enter the value:
23
node inserted at left

BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
```



```
enter your choice:2

          76

     55

               45

          34

               32

                    23
BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:3
55 34 32 23 45 76 BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
```



```
9.exit
enter your choice:4
23 32 34 45 55 76 BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:5
23 32 45 34 76 55 BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:6
enter the value to be deleted:
34
BST operation:
1.insert
 2.display
```



```
9.exit
enter your choice:2

          76

     55

          45

               32

                    23
BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
 7.MinVal
 8.maxValue
9.exit
enter your choice:3
55 45 32 23 76 BST operation:
1.insert
 2.display
 3.preorder
 4.Inorder
5.postorder
 6.DeleteAnyNode
```

Practical No. 10: Create a minimum spanning tree using any method Kruskal’s algorithm or Prim’s algorithm

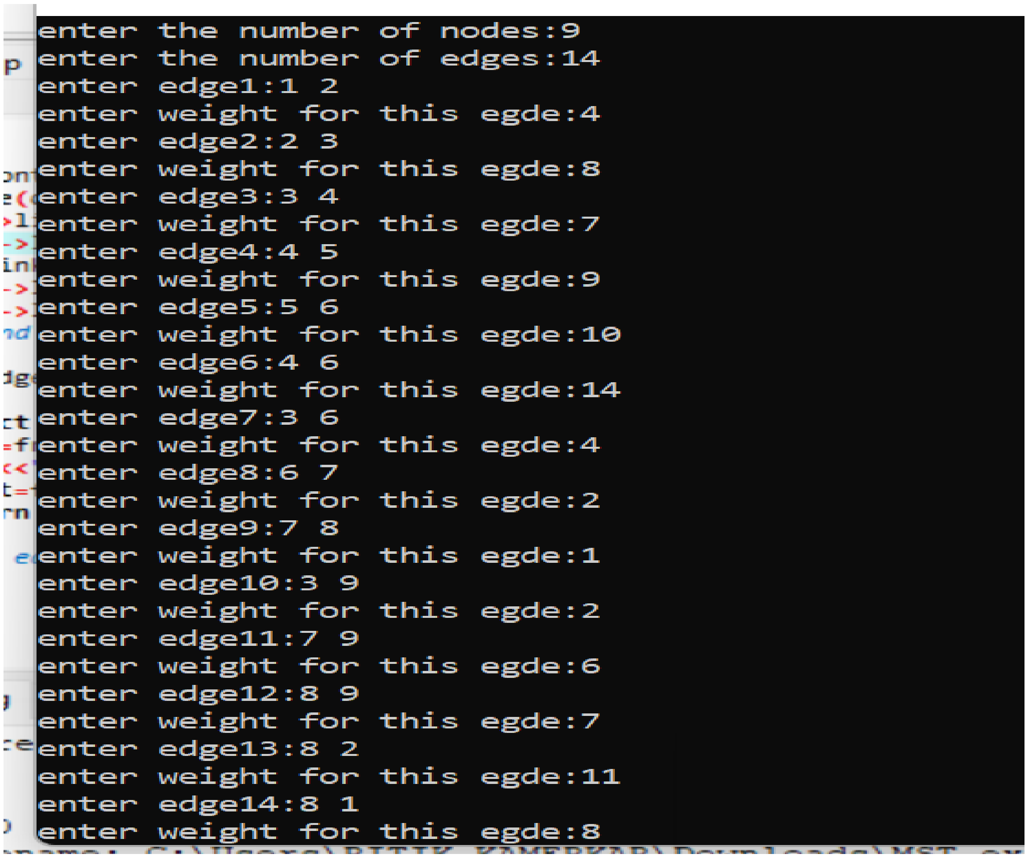
Aim: Write a program in c++ to implement minimum spanning tree.

```
//step-1 create a weightd graph
//max- edges=n*(n-1)/2
//step-2 enter the edges in the priority queue
//step-3 make spanning tree by retrieving the edges from priority queue
#include<iostream>
#define max 20
using namespace std;
struct edge//required for queue
{
	int u;//origin
	int v;//destination
	int weight;
	edge* link;
};
edge* front=NULL;
int father[max];//hold father of each node
struct edge tree[max];//this array will contain edges of spanning tree
int count=0;//count the no of edges that should be equal to n-1
int wt_tree=0;//hold the totla weight of spanning tree
int n;//total number of nodes in the graph
int e;//total number of edges in the graph
void makeTree();
void insertTree(int i,int j,int wt);
void insertPqueue(int i,int j,int wt);
struct edge* deletePqueue();

void createGraph()
{
	int i;//for loop
	int wt;//weight of the edge
	int origin;//start node
	int destin;//end node
	cout<<"enter the number of nodes:";
	cin>>n;//n=9
	cout<<"enter the number of edges:";
	cin>>e;//e=16
	for(i=1;i<=e;i++)//i=1 to 16
	{
		cout<<"enter edge"<<i<<":";//edge 1: 1 2
		cin>>origin>>destin;
		if((origin==0)&&(destin==0))
		break;
		cout<<"enter weight for this egde:";//1
```

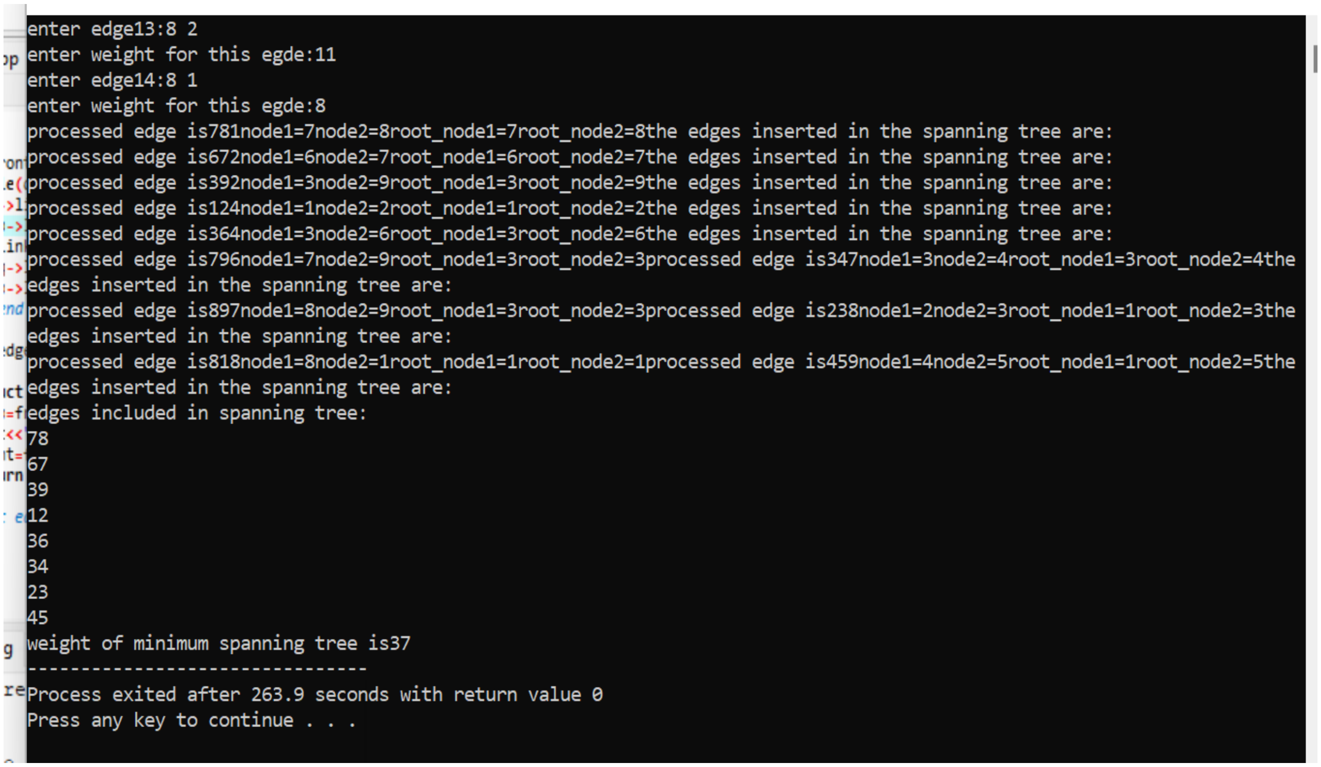
```
        cin>>wt;
        if(origin>n||destin>n||origin<=0||destin<=0)//1 10
        {
            cout<<"invalid edge"<<endl;
            i--;
        }
        else
        insertPqueue(origin,destin,wt);
    }
    if(i<n-1)
    {
        cout<<"spanning tree is not possible";
        exit(1);
    }
}
int main()
{
    int i;
    createGraph();
    makeTree();
    cout<<"edges included in spanning tree:"<<endl;
    for(i=1;i<=count;i++)
    {
        cout<<tree[i].u;
        cout<<tree[i].v;
        cout<<endl;
    }
    cout<<"weight of minimum spanning tree is"<<wt_tree;
    return 0;
}
void makeTree()
{//take edges from the priority queue
struct edge *temp;
int node1,node2;//for origin and destination
//declare one array which will store parent of each node, so that we can avoid the loop
int root_node1,root_node2;
while(count<n-1)//a tree need n-1 edges
{
    temp=deletePqueue();//1-2,3
    node1=temp->u;//1
    node2=temp->v;//2
    cout<<"node1="<<node1;
    cout<<"node2="<<node2;
    while(node1>0)
    {
        root_node1=node1;//1
        node1=father[node1];//0
    }
    while(node2>0)
    {
        root_node2=node2;//2
        node2=father[node2];//0
    }
    cout<<"root_node1="<<root_node1;//1
    cout<<"root_node2="<<root_node2;//2
    //condititon for checking loop
    if(root_node1!=root_node2)
    {
        insertTree(temp->u,temp->v,temp->weight);
        wt_tree=wt_tree+temp->weight;
        father[root_node2]=root_node1;//father of 2 is 1
    }

}
}
void insertTree(int i,int j,int wt)
{
    cout<<"the edges inserted in the spanning tree are:"<<endl;
    count++;
    tree[count].u=i;
    tree[count].v=j;
    tree[count].weight=wt;
}
void insertPqueue(int i,int j,int wt)
{
struct edge* temp,*q;
temp = new edge();//3-9=1
temp->u=i;
temp->v=j;
temp->weight=wt;
if(front==NULL||temp->weight<front->weight)//1-3=5
{
    temp->link=front;
    front=temp;
}
else
{
    q=front;
    while(q->link!=NULL&&q->link->weight<=temp->weight)
    q=q->link;
    temp->link=q->link;
    q->link=temp;
    if(q->link==NULL)
    temp->link=NULL;
    }//end of else
}
struct edge* deletePqueue()
{
    struct edge* temp;
    temp=front;
    cout<<"processed edge is"<<temp->u<<temp->v<<temp->weight;
    front=front->link;
    return temp;
}
```

//insert edge in the tree

Output:



```
enter the number of nodes:9
enter the number of edges:14
enter edge1:1 2
enter weight for this egde:4
enter edge2:2 3
enter weight for this egde:8
enter edge3:3 4
enter weight for this egde:7
enter edge4:4 5
enter weight for this egde:9
enter edge5:5 6
enter weight for this egde:10
enter edge6:4 6
enter weight for this egde:14
enter edge7:3 6
enter weight for this egde:4
enter edge8:6 7
enter weight for this egde:2
enter edge9:7 8
enter weight for this egde:1
enter edge10:3 9
enter weight for this egde:2
enter edge11:7 9
enter weight for this egde:6
enter edge12:8 9
enter weight for this egde:7
enter edge13:8 2
enter weight for this egde:11
enter edge14:8 1
enter weight for this egde:8
```



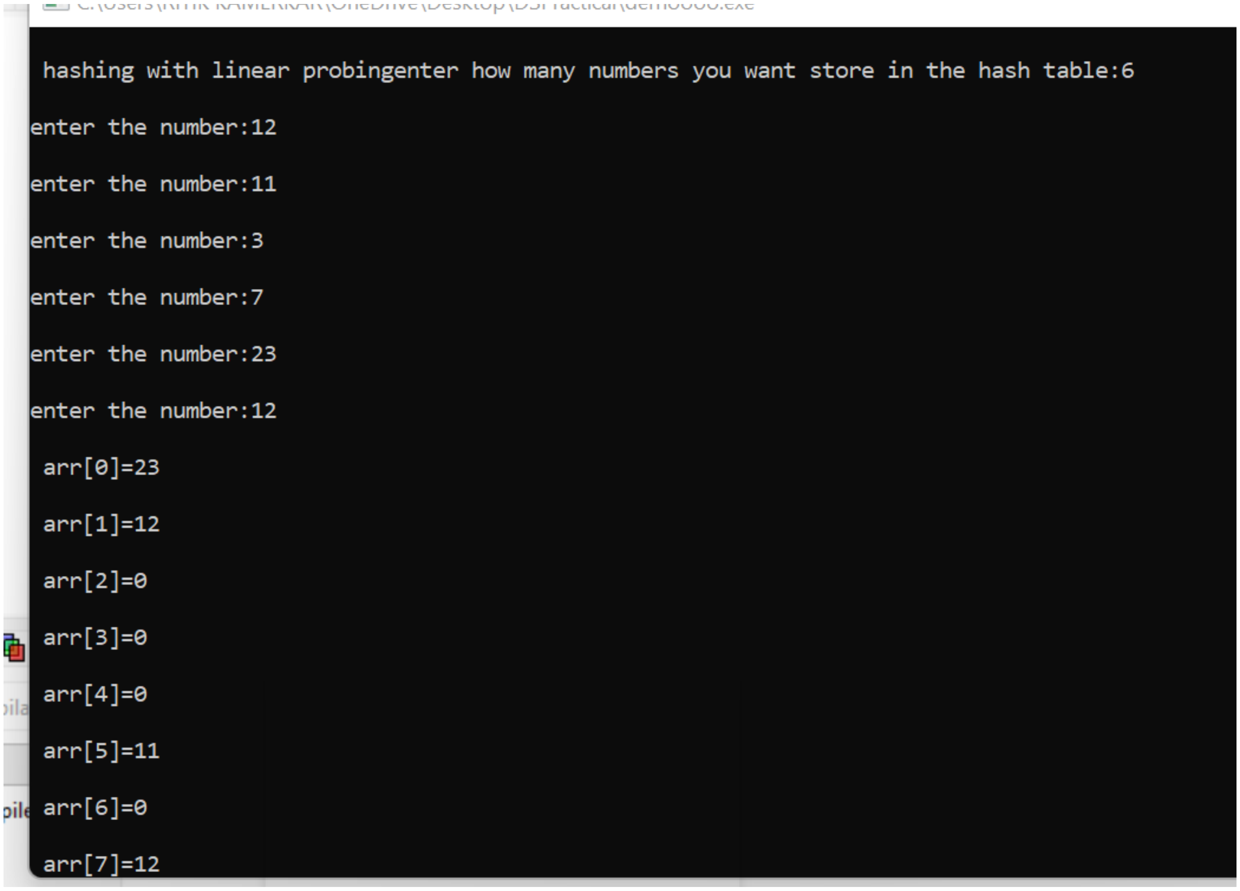
```
enter edge13:8 2
enter weight for this egde:11
enter edge14:8 1
enter weight for this egde:8
processed edge is781node1=7node2=8root_node1=7root_node2=8the edges inserted in the spanning tree are:
processed edge is672node1=6node2=7root_node1=6root_node2=7the edges inserted in the spanning tree are:
processed edge is392node1=3node2=9root_node1=3root_node2=9the edges inserted in the spanning tree are:
processed edge is124node1=1node2=2root_node1=1root_node2=2the edges inserted in the spanning tree are:
processed edge is364node1=3node2=6root_node1=3root_node2=6the edges inserted in the spanning tree are:
processed edge is796node1=7node2=9root_node1=3root_node2=3processed edge is347node1=3node2=4root_node1=3root_node2=4the
edges inserted in the spanning tree are:
processed edge is897node1=8node2=9root_node1=3root_node2=3processed edge is238node1=2node2=3root_node1=1root_node2=3the
edges inserted in the spanning tree are:
processed edge is818node1=8node2=1root_node1=1root_node2=1processed edge is459node1=4node2=5root_node1=1root_node2=5the
edges inserted in the spanning tree are:
edges included in spanning tree:
78
67
39
12
36
34
23
45
weight of minimum spanning tree is37
--------------------------------
Process exited after 263.9 seconds with return value 0
Press any key to continue . . .
```

Practical No. 11: Perform various hashing techniques with Linear Probe as collision resolution scheme.

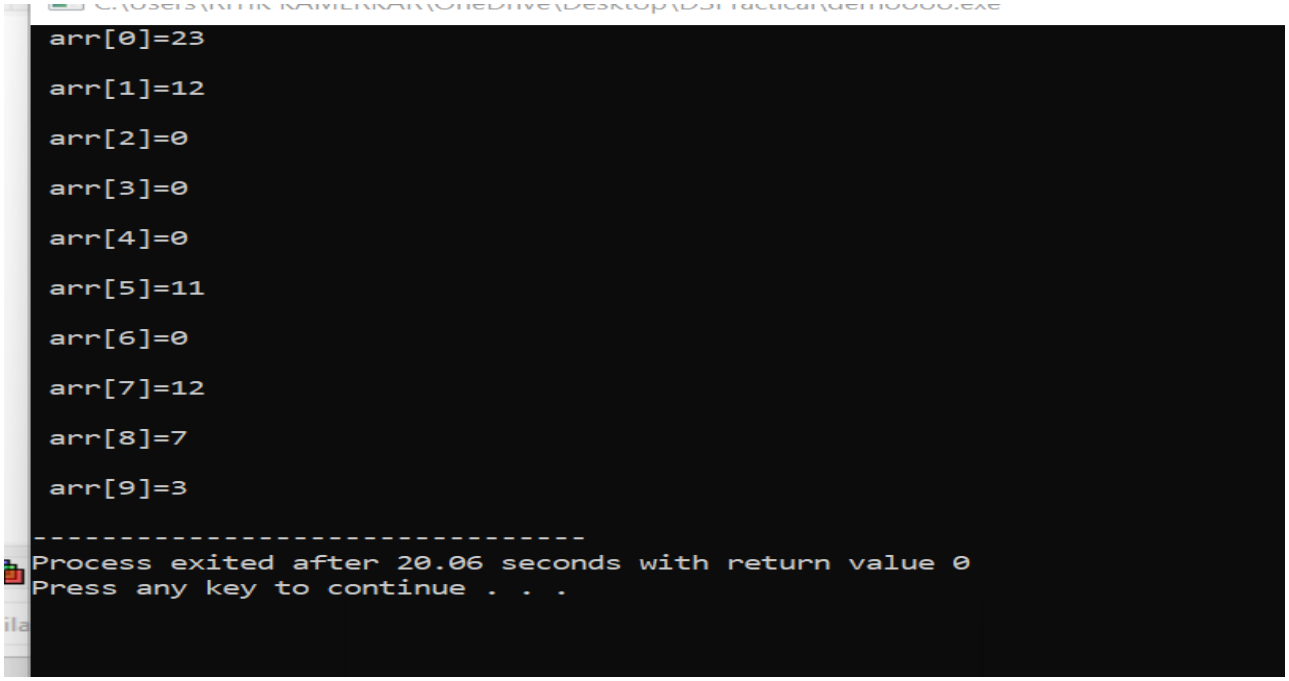
Aim: Write a program in c++ to implement Linear Probing.

```
#include<iostream>
using namespace std;
class LinearProbe
{
long arr[10];//hash table with size 10
public:
void hash()
{
long n,num,pos;
for(int i=0;i<10;i++)
arr[i]=0;//initial assigning 0 to each index
cout<<"\n hashing with linear probing";
cout<<"enter how many numbers you want store in the hash table:";
cin>>n;
for(int i=1;i<=n;i++)
{
cout<<"\nenter the number:";
cin>>num;//1
pos=((2*num)+3)%10;//modulo hash function//pos=5
//how to place the elements
for(int j=0;j<10;j++)//placing the numbers in arr
{
if(arr[pos]==0)
{
arr[pos]=num;
break;
}
if(pos==9)
pos=0;
else
{
pos++;
}
}
}
for(int i=0;i<=9;i++)
cout<<"\n arr["<<i<<"]="<<arr[i]<<"\n";
}
};
int main()
{
LinearProbe l;
l.hash();
}
```

Output:



```
 hashing with linear probingenter how many numbers you want store in the hash table:6
enter the number:12
enter the number:11
enter the number:3
enter the number:7
enter the number:23
enter the number:12
 arr[0]=23
 arr[1]=12
 arr[2]=0
 arr[3]=0
 arr[4]=0
 arr[5]=11
 arr[6]=0
 arr[7]=12
```



```
 arr[0]=23
 arr[1]=12
 arr[2]=0
 arr[3]=0
 arr[4]=0
 arr[5]=11
 arr[6]=0
 arr[7]=12
 arr[8]=7
 arr[9]=3
--------------------------------
Process exited after 20.06 seconds with return value 0
Press any key to continue . . .
```

Practical No. 12: Implementing Heap with different operations performed

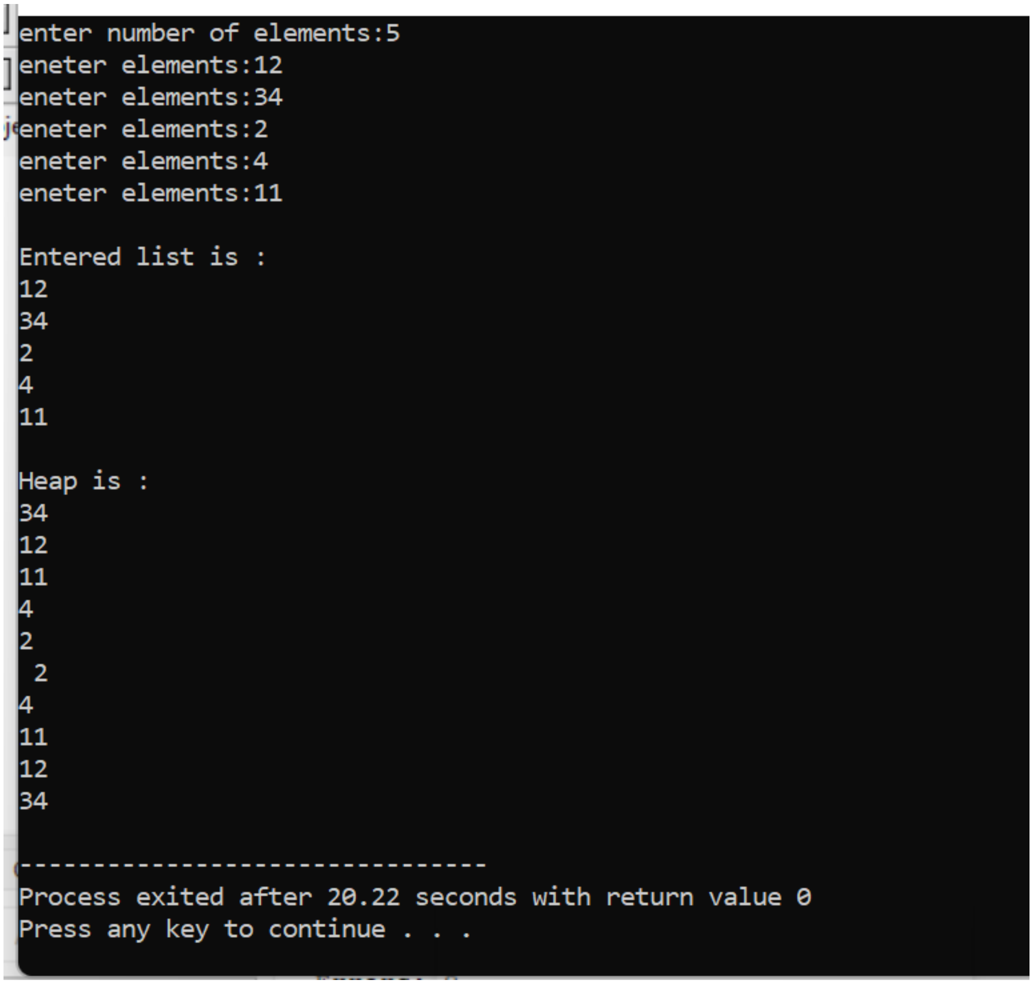
Aim: Write a program in c++ to implement heap .

```
#include<iostream>
using namespace std;
#define height 10
int arr[20],n;
//Function to display the elements in the array
void insert(int num,int loc)//35 4
{
	int par;
while(loc>0)//0
{
par = (loc-1)/2;//1st element,0th
if (num<=arr[par])//[0]=15,[1]=35 [4]=20
{
arr[loc]=num;
return;
}
arr[loc]=arr[par];//
loc=par;//recursive,loc=4,loc=1,loc=0
}/*End of while*/
arr[0]=num;
}/*End of insert()*/
//This function to create a heap
void create_heap()
{
int i;
for(i=0;i<n;i++)
//maxHeapify( arr, n,largest);
insert(arr[i],i);
}/*End of create_heap()*/
//Function to insert an element to the heap
void display()
{
  int i;
for(i=0;i<n;i++)
cout<<arr[i]<<endl;
cout<<" ";
}/*End of display()*/
void maxHeapify(int arr[],int n, int i)
{
	int largest = i;//i=3
	int l=2*i;//6,
	int r=(2*i)+1;//7
	//comparing the root with its left and right child
	while(l<= n && arr[l]>arr[largest])
	{
		largest=l;
```

```
        }
        while(r<=n && arr[r]>arr[largest])
        {
                largest=r;
        }
        if(largest!=i)
        {
                int temp=arr[i];
                arr[i]=arr[largest];
                arr[largest]=temp;
                maxHeapify( arr, n,largest);
        }
}
void build(int a[],int n)//create heap
{
        int i;
        for(i=n/2;i>=0;i--)
        maxHeapify(a,n,i);
}
void del_root(int last)
{
int left,right,i,temp;
i=0; /*Since every time we have to replace root with last*/
/*Exchange last element with the root */
temp=arr[i];
arr[i]=arr[last];
arr[last]=temp;
left=2*i+1; /*left child of root*/
right=2*i+2;/*right child of root*/
while( right < last)
{
if ( arr[i]>=arr[left] && arr[i]>=arr[right] )
return;
if ( arr[right]<=arr[left] )
{
        temp=arr[i];
arr[i]=arr[left];
arr[left]=temp;
i=left;
}
else
{
temp=arr[i];
arr[i]=arr[right];
arr[right]=temp;
i=right;
}
left=2*i+1;
right=2*i+2;
}/*End of while*/
if(left==last-1 && arr[i] < arr[left])
//if (left==last–1 && arr[i] < arr[left] )/*right==last*/
```

```
{
temp=arr[i];
arr[i]=arr[left];
arr[left]=temp;
}
}/*End of del_root*/
//Function to sort an element in the heap
void deleteRoot(int arr[],int n)
{
        int lastElement = arr[n-1];
        arr[0]=lastElement;
        n=n-1;
        maxHeapify(arr,n-2,0);
}
void heap_sort()
{
int last;
for(last = n-1; last>=0;last--)

del_root(last);
}/*End of del_root*/
int main()
{
int i;
cout<<"enter number of elements:";
cin>>n;
for(i=0;i<n;i++)
{
cout<<"eneter elements:";
cin>>arr[i];
}
cout<<"\nEntered list is :\n";
display();
//create_heap();
//maxHeapify(arr,n,i);
build(arr,n);
cout<<"\nHeap is :\n";
//del_root(n-1);
//display();
display();
heap_sort();
//deleteRoot(arr,n);
display();
return 0;
}
```

Output:



```
enter number of elements:5
eneter elements:12
eneter elements:34
eneter elements:2
eneter elements:4
eneter elements:11

Entered list is :
12
34
2
4
11

Heap is :
34
12
11
4
2
 2
4
11
12
34

--------------------------------
Process exited after 20.22 seconds with return value 0
Press any key to continue . . .
```

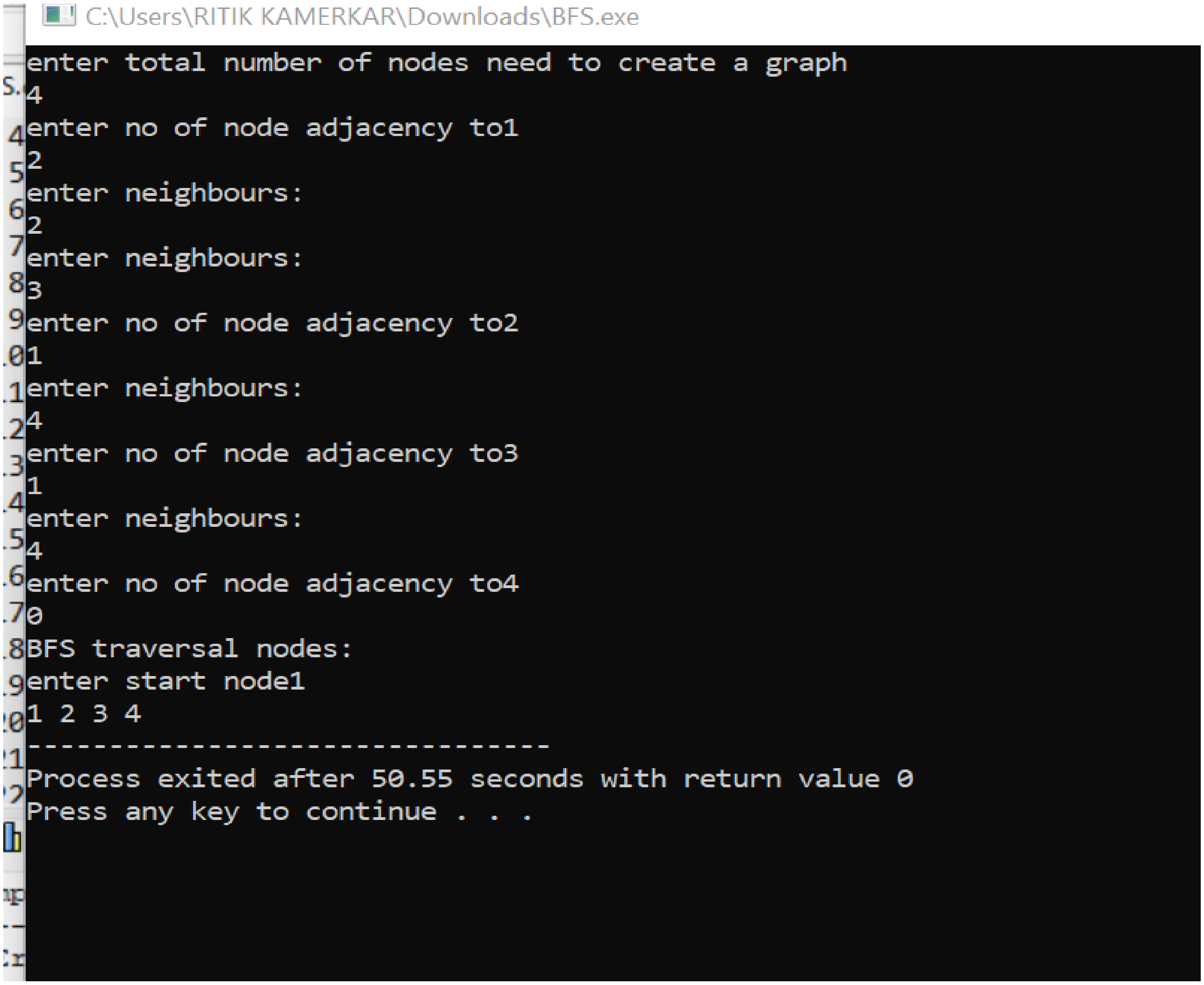
Practical No. 13: Implementation of Graph Traversal

Aim: Write a program in c++ to implement Breadth-First Search.

```
//BFS
//queue, ajacency link representation of nodes
#include<iostream>
#define max 50
using namespace std;
struct node//graph node
{
        int vertex;//node 1
        node* next;//next are 2 and 3
};
node* adj[max];
int totalNodes;
int queue[max],front=-1,rear=-1;
void enqueue(int item)
{
        rear=rear+1;
        queue[rear]=item;
        if(front==-1)
        front=0;
}
int dequeue()
{
        int proItem=queue[front];
        if(front==rear)
        front=rear=-1;
        else
        front=front+1;
        return (proItem);
}
int isQueueEmpty()
{
        if(front==-1)
        return 1;
        else
        return 0;
}
void createGraph()
{
        node* newNode,*last;
        int neighbours,neighbour_node;
        cout<<"enter total number of nodes need to create a graph"<<endl;//4
        cin>>totalNodes;
        for(int i=1;i<=totalNodes;i++)//start from 1 to 4
        {
                cout<<"enter no of node adjacency to"<<i<<endl;//2 neighbours i.e 2 and 3
                cin>>neighbours;//2
                for(int j=1;j<=neighbours;j++)//accepting the neighbours
                {
                        cout<<"enter neighbours:"<<endl;//2,3
                        cin>>neighbour_node;
```

```
            newNode=new node;
            newNode->vertex=neighbour_node;
            newNode->next=NULL;
            if(adj[i]==NULL)
            adj[i]=last=newNode;//excuted at starting
            else
            {
            last->next=newNode;//executed at the time of 3
            last=newNode;
            }
        }
    }

}
//BFS
void BFSTraversal()
{
    node* temp;
    int startNode,v,proNode;
    int status[max];
    const int ready=1,wait=2,processed=3;
    cout<<"enter start node";//1
    cin>>startNode;
    for(int i=1;i<=totalNodes;i++)
    status[i]=ready;//status of 1=1,2=1,3=1,4=1
    //call enqueue method to insert the nodes
    enqueue(startNode);
    status[startNode]=wait;//this will not enter again in the queue
    while(isQueueEmpty()!=1)
    {
        //in queue only the start node is there
    proNode=dequeue();  //start node,2
    cout<<proNode<<" ";//display of start node
    status[proNode]=processed;
    temp=adj[proNode];//neighbour of second node
    while(temp!=NULL)//for finding neighbour nodes
    {
        v=temp->vertex;//2,3
        if(status[v]==ready)
        {
            enqueue(v);//enter 2 in the queue,enter 3
            status[v]=wait;
        }
        temp=temp->next;//3
    }
    }
}
int main()
{
    createGraph();
    cout<<"BFS traversal nodes:"<<endl;
    BFSTraversal();
}
```

Output:



C:\Users\RITIK KAMERKAR\Downloads\BFS.exe

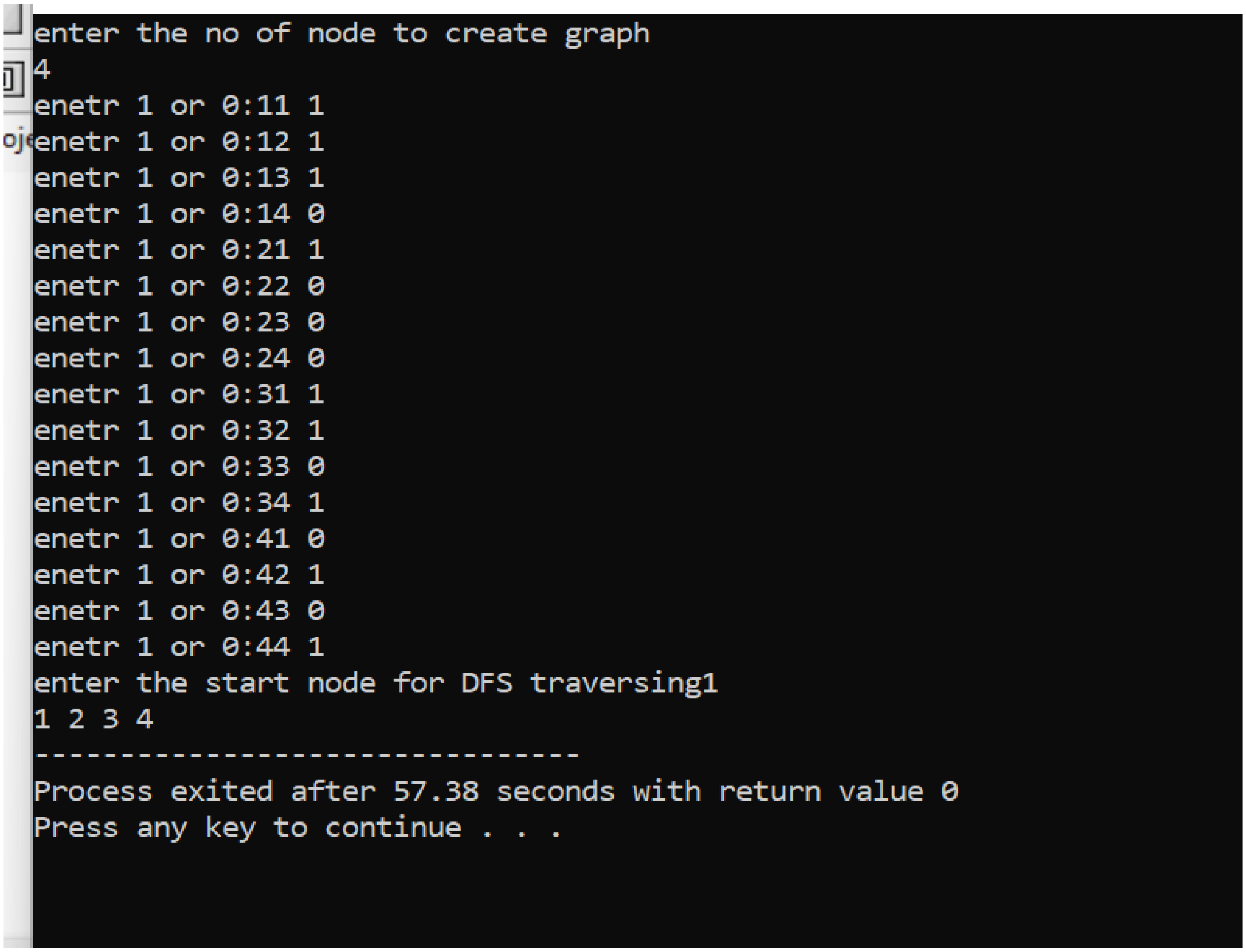
```
enter total number of nodes need to create a graph
4
enter no of node adjacency to1
2
enter neighbours:
2
enter neighbours:
3
enter no of node adjacency to2
1
enter neighbours:
4
enter no of node adjacency to3
1
enter neighbours:
4
enter no of node adjacency to4
0
BFS traversal nodes:
enter start node1
1 2 3 4
--------------------------------
Process exited after 50.55 seconds with return value 0
Press any key to continue . . .
```

Aim: Write a program in c++ to implement Depth-First Search.

```
#include<iostream>
#define max 10
using namespace std;
void buildAdjMatrix(int adj[max][max],int n)//n repesents no nodes
{//int adj[max][max] means 10*10 space of matrix created in memory
        int i,j;
        for(i=1;i<=n;i++)//for row=1,2
        for(j=1;j<=n;j++)//1 1, 1 2,1 3, 1 4, 1 5,n=5
        //2 1, 2 2, 2 3, 2 4, 2 5
        {
                cout<<"enetr 1 or 0:"<<i<<j<<" ";
                cin>>adj[i][j];
        }
}
void DFS(int startNode,int visited[],int adj[][max],int node)
{
        int j;
        visited[startNode]=1;//visited[1]=1,visited[2]=1,visited[4]=1,visited[5]=1
        //visited[3]=1
        cout<<startNode<<" ";//start node is 1->2->4->5->3
        for(j=1;j<=node;j++)//j=1,j=2,
        {
                if(adj[startNode][j]==1&&visited[j]==0)//adj[1][1],adj[1][2],
                //after this, adj[2][1],adj[2][2],adj[2][3],adj[2][4]
                //adj[4][1],adj[4][2],adj[4][3],adj[4][4],adj[4][5]
                //adj[5][]
                //adj[2][5]
                //adj[1][3]

                DFS(j,visited,adj,node);//DFS is for node 2,DFS call for 4,dfs will
call for 5
        }
}
int main()
{
        int adj[max][max],node,startNode;
        int visited[max];
        cout<<"enter the no of node to create graph"<<endl;
        cin>>node;
```

```
	buildAdjMatrix(adj,node);//grapg creation cpmpleted
	for(int i=1;i<=node;i++)
	visited[i]=0;//all the are not visited
	cout<<"enter the start node for DFS traversing";
	cin>>startNode;
	if(visited[startNode]==0)
	DFS(startNode,visited,adj,node);
	return 0;

}
```

Output:



```
enter the no of node to create graph
4
enetr 1 or 0:11 1
enetr 1 or 0:12 1
enetr 1 or 0:13 1
enetr 1 or 0:14 0
enetr 1 or 0:21 1
enetr 1 or 0:22 0
enetr 1 or 0:23 0
enetr 1 or 0:24 0
enetr 1 or 0:31 1
enetr 1 or 0:32 1
enetr 1 or 0:33 0
enetr 1 or 0:34 1
enetr 1 or 0:41 0
enetr 1 or 0:42 1
enetr 1 or 0:43 0
enetr 1 or 0:44 1
enter the start node for DFS traversing1
1 2 3 4
--------------------------------
Process exited after 57.38 seconds with return value 0
Press any key to continue . . .
```