**PRACTICAL 1**

**Aim : Implementation of Searching Algorithms.**

**1)Linear search :**

**CODE:**

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

int a[10],n,i,x;

int flag=0;

cout<<"Enter 10 elements:\n";

for(i=0;i<10;i++)

{

cin>>a[i];

}

cout<<"\nWhich element you want to search -\n";

cin>>x;

for(i=0;i<10;i++)

{

if(x==a[i])

{

flag=1;

break;

}

}

if(flag==1)

{

cout<<"Number is found.\n";

}

else

{

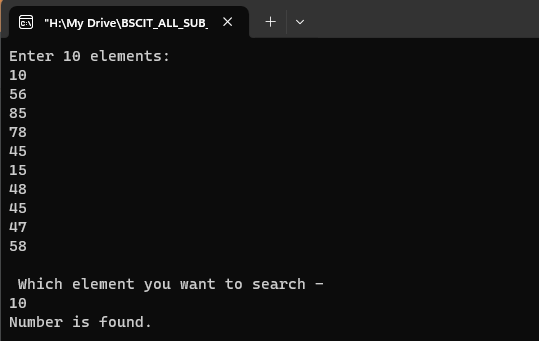
cout<<"The number is not in this array.\n";

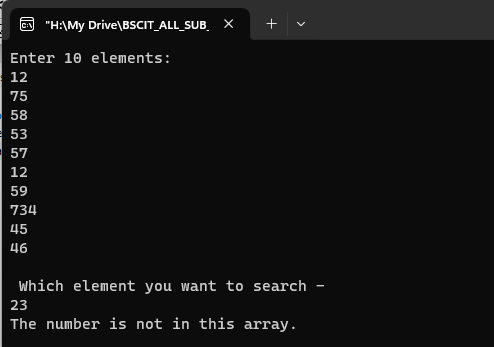
}

return 0;

}

**OUTPUT:**





**2)Binary search :**

**CODE:**

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

int i,n,a[10],st=0,ed=9,mid;

cout<<"Enter 10 elements:\n";

for(i=0;i<10;i++)

{

cin>>a[i];

}

cout<<"\nEnter the number you want to search :\n";

cin>>n;

mid=(st+ed)/2;

while(n!=a[mid]&&st<=ed)

{

if(n>a[mid])

st=mid+1;

else

ed=mid-1;

mid=(st+ed)/2;

}

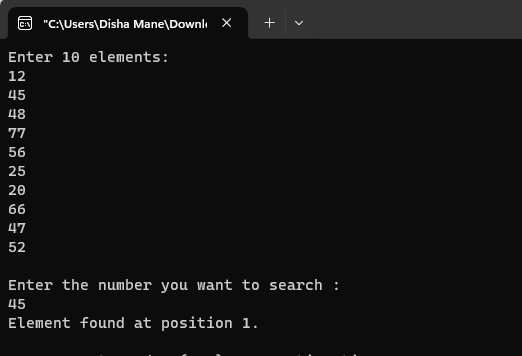
if(n==a[mid])

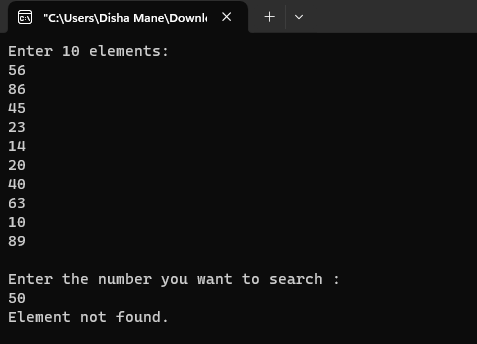
cout<<"Element found at position "<<mid<<".\n";

if(st>ed)

cout<<"Element not found.\n";}

**OUTPUT:**

****

****

**Q.6 Write a program in C++ in a code block to create a binary search tree and perform the following operation: 1. insert 2. find max 3. Display**

Code:-

#include<iostream>

using namespace std;

// Node structure for the binary search tree

struct Node {

int data;

Node\* left;

Node\* right;

};

// Class for binary search tree operations

class BinarySearchTree {

private:

Node\* root;

// Function to insert a new node into the binary search tree

Node\* insertRecursive(Node\* current, int value) {

if (current == nullptr) {

Node\* newNode = new Node;

newNode->data = value;

newNode->left = nullptr;

newNode->right = nullptr;

return newNode;

}

if (value < current->data) {

current->left = insertRecursive(current->left, value);

} else if (value > current->data) {

current->right = insertRecursive(current->right, value);

}

return current;

}

// Function to find the maximum value in the binary search tree

int findMaxRecursive(Node\* current) {

if (current->right == nullptr) {

return current->data;

}

return findMaxRecursive(current->right);

}

// Function to display the elements of the binary search tree (in-order)

void displayInOrder(Node\* current) {

if (current != nullptr) {

displayInOrder(current->left);

cout << current->data << " ";

displayInOrder(current->right);

}

}

public:

// Constructor

BinarySearchTree() {

root = nullptr;

}

// Wrapper function for inserting into the binary search tree

void insert(int value) {

root = insertRecursive(root, value);

cout << "Node with value " << value << " inserted successfully!\n";

}

// Wrapper function for finding the maximum value in the binary search tree

int findMax() {

if (root == nullptr) {

cout << "Tree is empty. Cannot find maximum.\n";

return -1; // Assuming -1 represents an invalid value in this context

}

int maxVal = findMaxRecursive(root);

cout << "Maximum value in the tree: " << maxVal << endl;

return maxVal;

}

// Wrapper function for displaying the binary search tree

void display() {

if (root == nullptr) {

cout << "Tree is empty.\n";

} else {

cout << "Binary Search Tree (In-order traversal): ";

displayInOrder(root);

cout << endl;

}

}

};

int main() {

BinarySearchTree myBST;

int choice, value;

do {

// Display menu

cout << "\nBinary Search Tree Operations:\n";

cout << "1. Insert\n";

cout << "2. Find Max\n";

cout << "3. Display\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

// Insert operation

cout << "Enter the value to insert: ";

cin >> value;

myBST.insert(value);

break;

case 2:

// Find Max operation

myBST.findMax();

break;

case 3:

// Display operation

myBST.display();

break;

case 0:

// Exit the program

cout << "Exiting the program. Goodbye!\n";

break;

default:

// Invalid choice

cout << "Invalid choice. Please try again.\n";

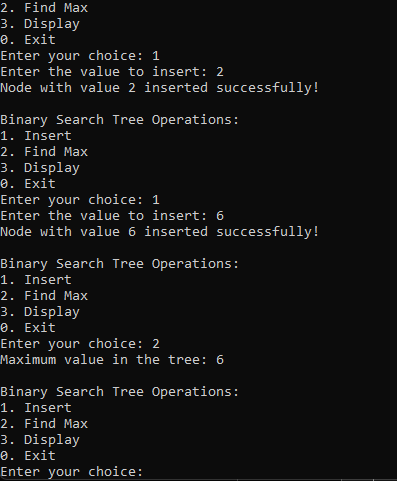
}

} while (choice != 0);

return 0;

}

O/p:-



**PRACTICAL 2**

**Aim : Implementation of different Sorting Techniques.**

**1)Selection Sort**

**2)Bubble Sort**

**3)Insertion Sort**

**1)Selection Sort:**

**CODE:**

#include<iostream>

using namespace std;

int main()

{

int a[10], i, j, n, temp;

cout<<"Selection Sort\n\n";

cout<<"Enter 10 values :\n";

for(n=0;n<10;n++)

{

cin>>a[n];

}

cout<<"\nBefore Selection sort values :\n";

for(n=0;n<10;n++)

{

cout<<"Iteration "<<n<<"\t"<<a[n]<<"\n";

}

cout<<"\nAfter Selection sort values :\n";

for(i=0;i<=n-1;i++)

{

for(j=i+1;j<=n;j++)

{

if(a[i]>a[j])

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

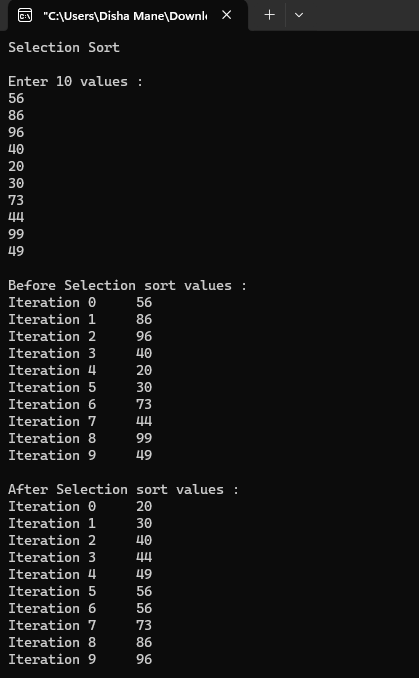
cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**OUTPUT:**



**2)Bubble Sort:**

**CODE:**

#include<iostream>

using namespace std;

int main()

{

int a[10], i, j, n, temp;

cout<<"Bubble Sort:\n\n";

cout<<"Enter 10 values :\n";

for(n=0;n<10;n++)

{

cin >> a[n];

}

cout<<"\nBefore Bubble sort values :\n";

for(n=0;n<10;n++)

{

cout<<"Iteration "<<n<<"\t"<<a[n]<<"\n";

}

cout<<"\nAfter Bubble sort values :\n";

for(i=0;i<=n-1;i++)

{

for(j=0;j<=n-i-1;j++)

{

if(a[j]>a[j+1])

{

temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

for(i=0; i<n;i++)

{

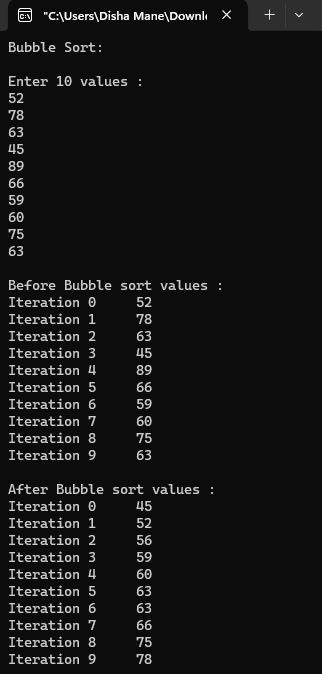
cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**OUTPUT :**



**3)Insertion Sort:**

**CODE:**

#include<iostream>

using namespace std;

int main()

{

int a[10], i, j, x;

cout<<"Insertion Sort\n\n";

cout<<"Enter 10 values :\n";

for(i=0;i<10;i++)

{

cin>>a[i];

}

cout<<"\nBefore Insertion sort values :\n";

for(i=0;i<10;i++)

{

cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

cout<<"\nAfter Insertion sort values :\n";

for(j=1;j<10;j++)

{

x=a[j];

for(i=j-1;i>=0&&x<a[i];i--)

{

a[i+1]=a[i];

}

a[i+1]=x;

}

for(i=0;i<10;i++)

{

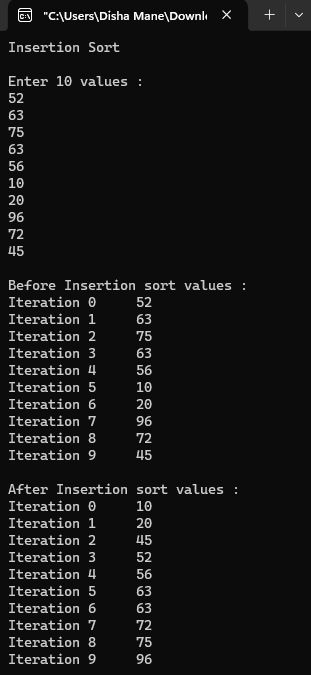
cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**OUTPUT :**

****

**PRACTICAL 3**

**Aim : Implementation of different Sorting Techniques.**

**1)Radix Sort**

**2)Shell Sort**

**1)Radix Sort:**

**CODE:**

#include<iostream>

#include<conio.h>

using namespace std;

class radix

{

public:

void sort()

{

int arr[6],i,j,k,large,noofpasses=0,temp[10][10],divisor=1,arrcount[10],n;

cout<<"Radix Sort\n\n";

cout<<"Enter the size of Array :\n";

cin>>n;

cout<<"\nEnter values :\n";

for(i=0;i<n;i++)

{

cin>>arr[i];

}

cout<<"\nBefore Radix sort values :\n";

for(i=0;i<n;i++)

{

cout<<"Iteration "<<i<<"\t"<<arr[i]<<"\n";

}

large=arr[0];

for(i=0;i<n;i++)

{

if(arr[i]>large)

large=arr[i];

}

while(large>0)

{

noofpasses++;

large/=10;

}

for(i=0;i<noofpasses;i++)

{

for(j=0;j<10;j++)

arrcount[j]=0;

for(j=0;j<n;j++)

{

k=(arr[j]/divisor)%10;

temp[k][arrcount[k]++]=arr[j];

}

int u=0;

for(int p=0;p<10;p++)

{

for(j=0;j<arrcount[p];j++)

{

arr[u++]=temp[p][j];

divisor\*=10;

}

}

}

cout<<"\nAfter Radix sort values :\n";

for(i=0;i<n;i++)

{

cout<<"Iteration "<<i<<"\t"<<arr[i]<<"\n";

}

}

};

int main()

{

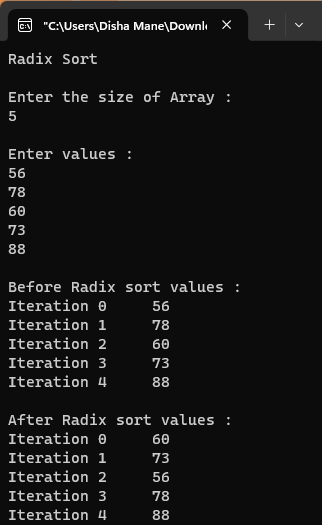
radix r;

r.sort();

return 0;

}

**OUTPUT:**



**2)Shell Sort:**

**CODE:**

#include<iostream>

using namespace std;

int main() {

int a[10], i, j, n,temp;

cout<<"Enter values :\n";

for(i=0;i<10;i++)

{

cin>>a[i];

}

cout<<"\nBefore Shell sort values:\n";

for(i=0; i<10; i++)

{

cout<<"Iteration "<<i<<"\t"<<a[i]<<"\n";

}

cout<<"\nAfter Shell sort values:\n";

for(i=10/2;i>0;i/=2)

{

int flag=1;

while(flag==1)

{

flag=0;

for(j=0;j<10-i;j++)

{

if(a[j]>a[j+i])

{

temp=a[j];

a[j]=a[j+i];

a[j+i]=temp;

flag=1;

}

}

}

}

for(i=0;i<10;i++)

{

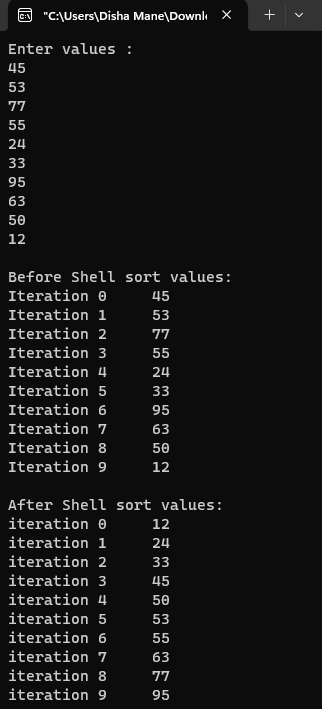
cout<<"iteration "<<i<<"\t"<<a[i]<<"\n";

}

return 0;

}

**OUTPUT:**

****

**PRACTICAL 4**

**Aim : Implementation of Stack Operation.**

**1)Stack Operation Using Array.**

**2)Stack Operation Using Linked List.**

**1)Stack Operation using Array:**

**CODE:**

#include<iostream>

#define MAX 5

using namespace std;

class stack

{

public:

int top=-1;

int x, stk[5],i;

int push(int x)

{

if(top == MAX - 1)

{

cout<<"Stack Overflow\n\n";

}

else

{

cout<<"Enter the number to push to the stack :\n";

cin>>x;

stk[++top]=x;

stk[top]=x;

}

}

int pop()

{

if(top==-1)

{

cout<<"Stack Underflow\n\n";

}

cout<<"Popped value : ";

x=stk[top];

top--;

cout<<x<<"\n";

}

void display()

{

if(top==-1)

{

cout<<"Stack is empty.\n\n";

}

else

cout<<"Stack :\n";

for(i=top;i>=0;i--)

{

cout<<stk[i]<<"\n";

}

}

};

int main()

{

stack s;

int ch,x;

while(ch!=4)

{

cout<<"1.Push\n2.Pop\n3.Display\n4.Exit\n";

cout<<"Enter the value for operation :\n";

cin>>ch;

switch(ch)

{

case 1:

s.push(x);

break;

case 2:

s.pop();

break;

case 3:

s.display();

break;

case 4:

break;

default:

cout<<"\nWrong choice.\n\n";

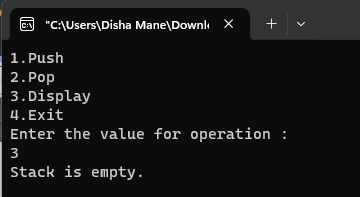
}

}

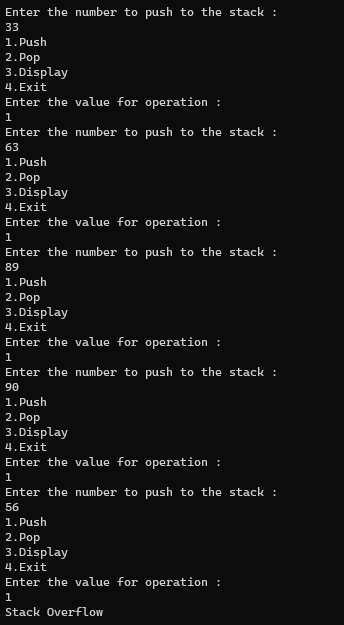
return 0; }

**OUTPUT:**

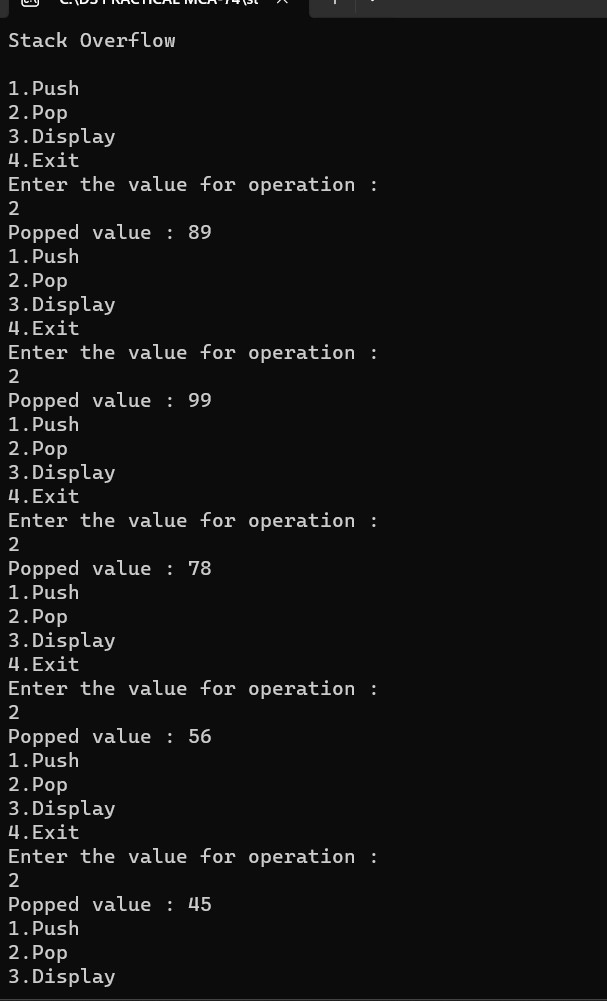
**1)Stack Empty:**

****

**2)Push Operation:**

****

**3)Pop Operation:**



**2)Stack Operation using Linked List:**

**CODE:**

#include<iostream>

using namespace std;

//Creating a NODE Structure

struct node

{

int data;

struct node \*next;

};

//Creating a class STACK

class stack

{

public:

struct node \*top=NULL;

int push(int x) //to insert an element

{

struct node \*ptr;

ptr=new node;

ptr->data=x;

ptr->next=NULL;

if(top!=NULL)

ptr->next=top;

top=ptr;

}

void pop() //to delete an element

{

struct node \*temp;

if(top==NULL)

{

cout<<"\nStack is empty.\n";

}

else

{

temp=top;

top=top->next;

cout<<"Popped value : "<<temp->data<<"\n";

delete temp;

}

}

void display() //to show the stack

{

struct node \*ptr1=top;

if(top==NULL)

{

cout<<"Stack is empty.\n\n";

}

else

{

cout<<"Stack :\n";

while(ptr1!=NULL)

{

cout<<ptr1->data<<"\n";

ptr1=ptr1->next;

}

}

}

};

int main()

{

stack s;

int ch,x;

while(ch!=4)

{

cout<<"1.Push\n2.Pop\n3.Display\n4.Exit\n";

cout<<"Enter the value for operation :\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value :\n";

cin>>x;

s.push(x);

break;

case 2:

s.pop();

break;

case 3:

s.display();

break;

case 4:

break;

default:

cout<<"\nWrong choice.\n\n";

}

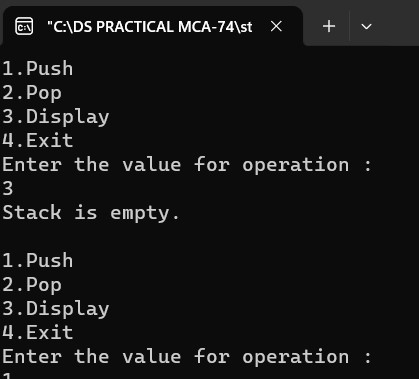
}

return 0;

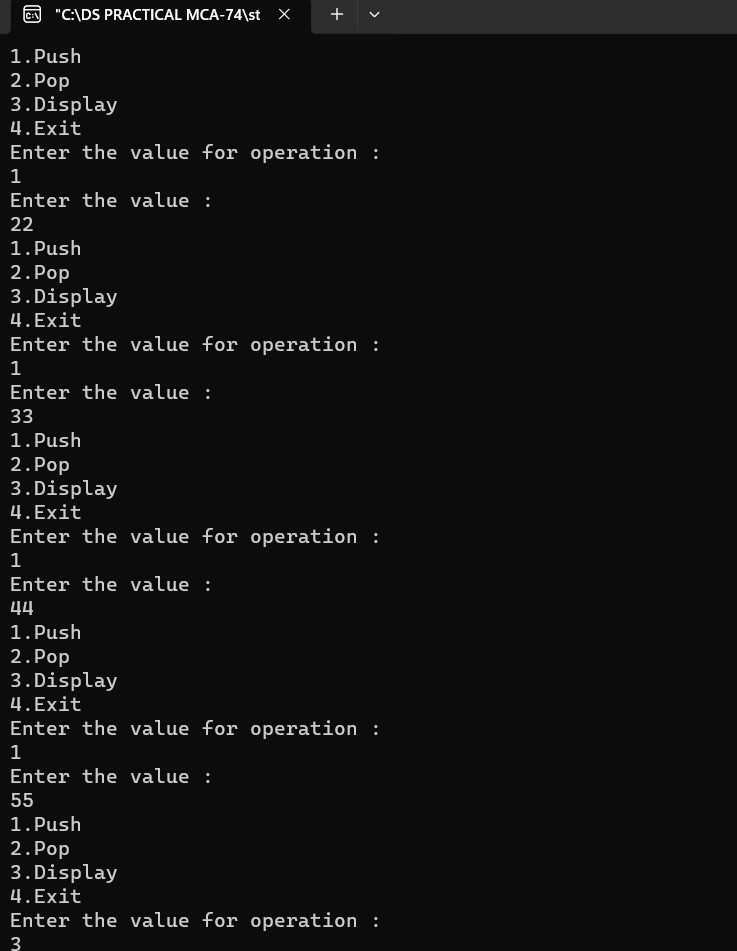
}

**OUTPUT:**

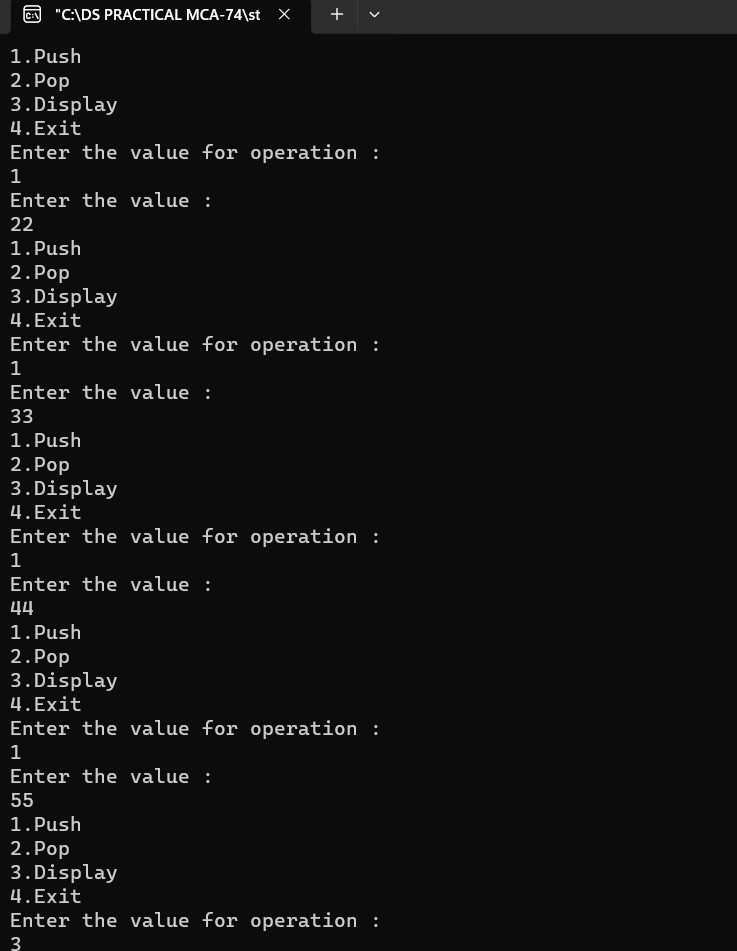
**1)Stack Empty:**

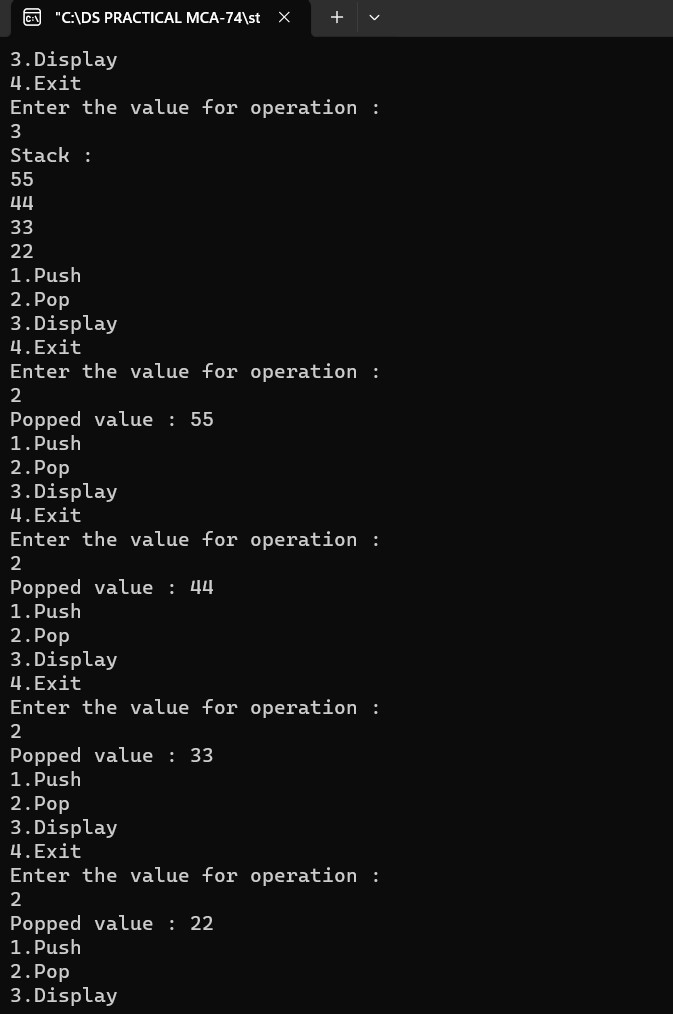


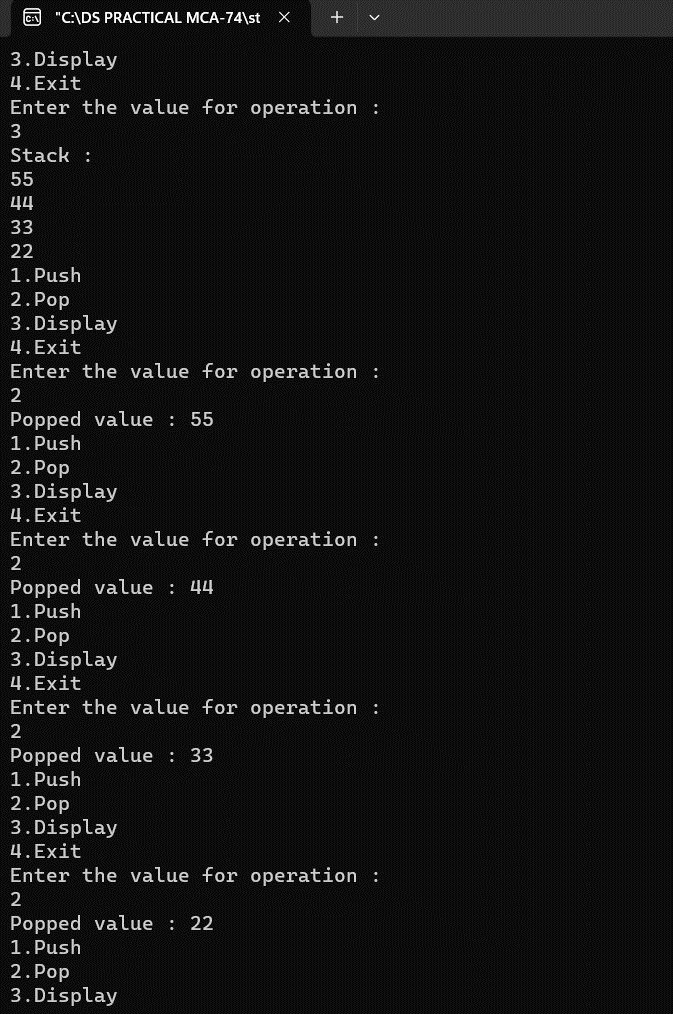
**2)Push Operation:**



**3)Pop Operation:**







**PRACTICAL 5**

**Aim : Implementation of Stack Operation for Balancing of**

**Parenthesis.**

**CODE:**

#include<iostream>

#include<string.h>

#include<conio.h>

using namespace std;

struct node

{

int data;

struct node \*next;

};

struct node \*tmp=NULL;

struct node \*tmp1=NULL;

struct node \*top=NULL;

struct node \*ptr=NULL;

int push(char x)

{

tmp = new node;

tmp->data=x;

tmp->next=NULL;

if(top == NULL)

{

top=tmp;

}

else

{

tmp1=top;

top=tmp;

tmp->next=tmp1;

}

}

char pop()

{

if(top==NULL)

{

cout<<"Stack is empty.\n";

}

else

{

ptr=top;

top=top

->next;

return(ptr

->data);

delete(ptr);

}

}

int main() {

int len,i;

char c,d,e;

char a[30];

cout<<"Enter expression :

\n";

cin>>a;

len=strlen(a);

for(i=0;i<len;i++)

{

if(a[i]=='{' || a[i]=='[' || a[i]=='(')

{

push(a[i]);

}

else

{

switch(a[i])

{

case ')':

c=pop();

if(c=='{' || c=='[')

{

cout<<"Invalid";

getch();

}

break;

case ']':

d=pop();

if(d=='{' || d=='(')

{

cout<<"Invalid";

getch();

}

break;

case '}':

e=pop();

if(e=='(' || e=='[')

{

cout<<"Invalid";

getch();

}

break;

default:

cout<<"Enter the correct choice";

getch();

}

}}

if(top==NULL)

cout<<"Balanced \n";

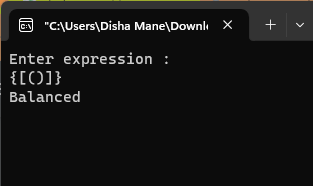
else

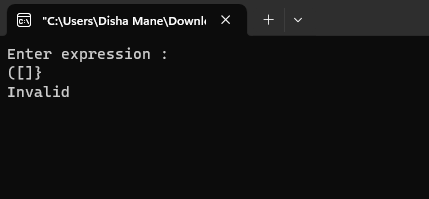
cout<<"Unbalanced \n";

getch();

return 0; }

**OUTPUT:**

****

****

**PRACTICAL 6**

**Aim : Evaluation of Postfix Expression.**

**CODE:**

#include<iostream>

#include<conio.h>

#include<string.h>

#include<math.h>

using namespace std;

class postfix

{

public:

int top;

char p[50];

long int A[50];

postfix()

{

top=-1;

}

void input();

void push(long int s);

long int pop();

int full();

int empty();

long int eval\_post();

};

void postfix::input()

{

cout<<"enter a postfix expression\n";

cin>>p;

}

int postfix::full()

{

if(top==49)

return 1;

else

return 0;

}

void postfix::push(long int s)

{

if(full())

cout<<"overflow\n";

else

{

top=top+1;

A[top]=s;

}}

int postfix::empty() {

if(top==

-1)

return 1;

else

return 0; }

long int postfix::pop() {

if(empty())

cout<<"underflow

\n";

else

return(A[top--]); }

long int postfix::eval\_post() {

long int a,b,temp,result,len;

int i;

len=strlen(p);

p[len]='#';

for(i=0;p[i]!='#';i++) {

if(p[i]<='9'&&p[i]>='0')

push(p[i]

-48);

else {

a=pop();

b=pop();

switch(p[i]) {

case '+':

temp=b+a;

break;

case

'

-':

temp=b

-a;

break;

case '\*':

temp=b\*a;

break;

case '/':

temp=b/1;

break;

case '%':

temp=b%a;

break;

case '^':

temp=pow(b,a); }

push(temp); }}

result=pop();

return result; }

main() {

long int value;

postfix f;

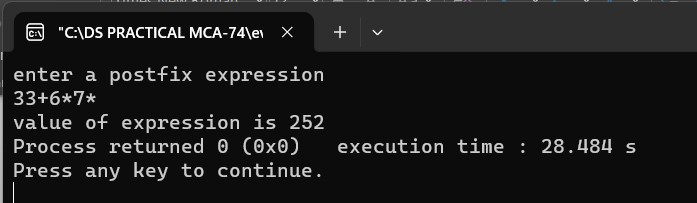
f.input();

value=f.eval\_post();

cout<<"value of expression is "<<value;

getch(); }

**OUTPUT:**



**PRACTICAL 7**

**Aim : Implementation of Queue using Array.**

**CODE:**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class queue

{

public:

int q[4],x,result;

int front=-1;

int rear=-1;

int maxsize=4;

void enqueue(int x)

{

if(rear==maxsize-1)

{

cout<<"Queue full.\n\n";

}

else

{

rear++;

q[rear]=x;

}

}

void dequeue()

{

if(rear==-1)

{

cout<<"Queue empty.\n\n";

}

else

{

cout<<"Deleted.\n";

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

x=q[front];

front++;

}

}

}

void display()

{

if(rear==-1)

{

cout<<"Queue empty.\n\n";

}

else

{

cout<<"Queue :\n";

for(int i=front+1;i<=rear;i++)

{

cout<<q[i]<<"\n";

}

}

}

};

int main()

{

int ch,x;

queue q;

do

{

cout<<"1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n";

cout<<"Enter your choice :\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value :\n";

cin>>x;

q.enqueue(x);

break;

case 2:

q.dequeue();

break;

case 3:

q.display();

break;

case 4:

break;

default:

cout<<"\nInvalid choice!!\n";

}

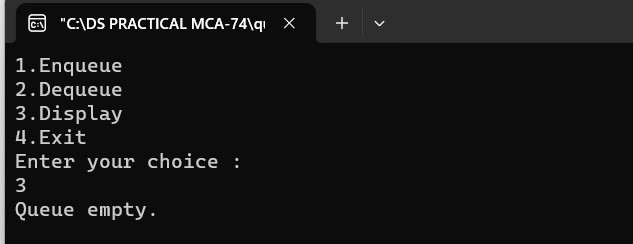
}

while(ch<4);

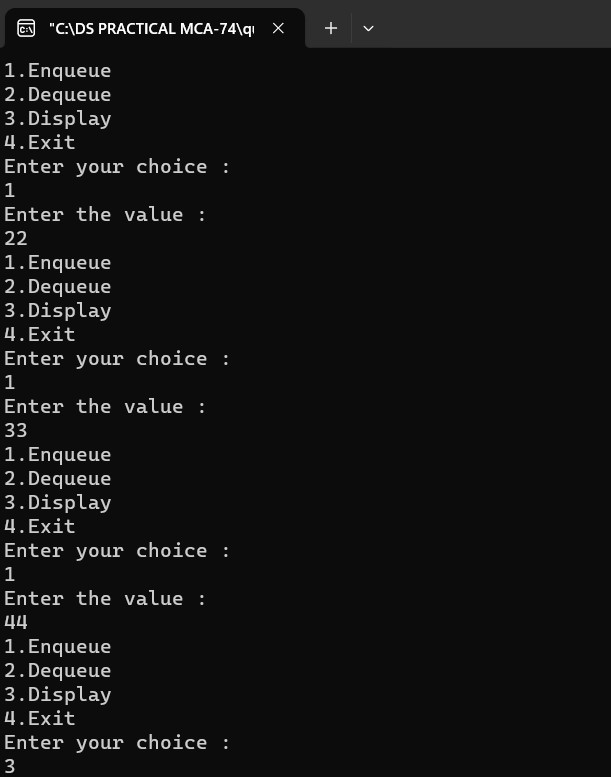
return 0; }

**OUTPUT:**

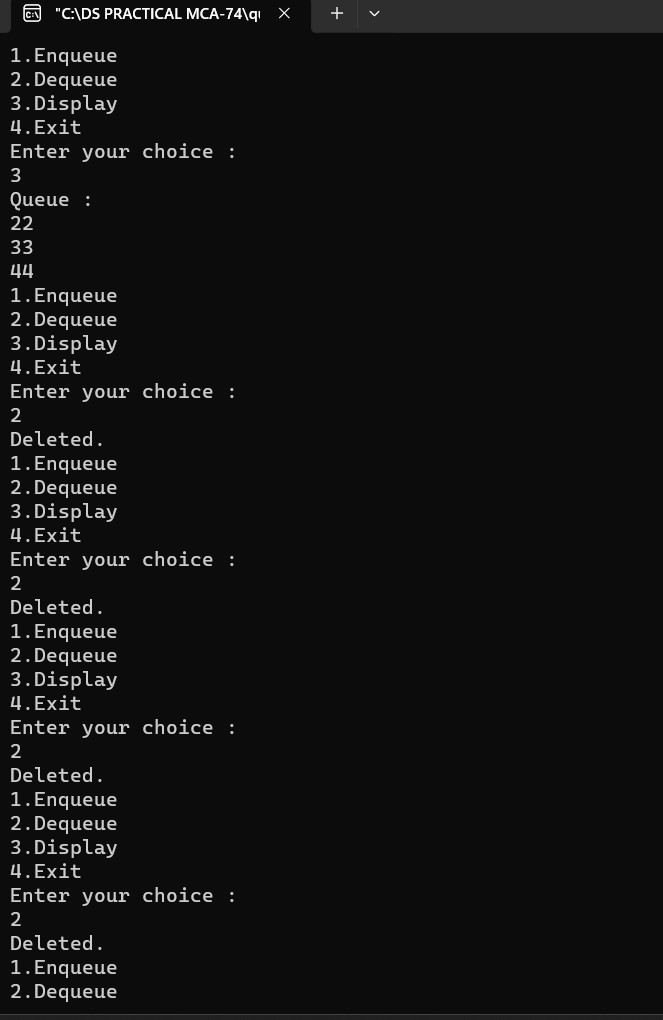
**1)Queue Empty:**

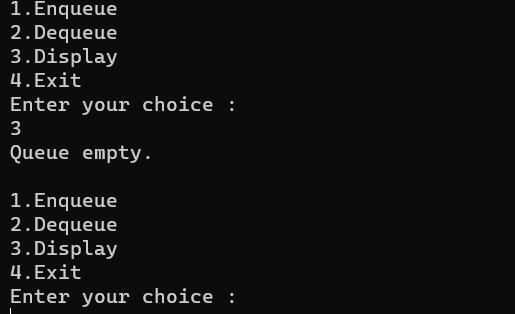


**2)enqueue:**



**3)Dequeue:**





**PRACTICAL 8**

**Aim : Implementation of Queue using Linked List.**

**CODE:**

#include<iostream>

using namespace std;

class linkqueue

{

public:

struct node

{

int data;

struct node \*next;

};

node \*front = NULL;

node \*rear = NULL;

void enqueue(int x)

{

node \*tmp, \*q;

tmp=new node;

tmp->data=x;

tmp->next=NULL;

if(front==NULL && rear==NULL)

{

front=rear=tmp;

}

else

{

rear->next=tmp;

rear=tmp;

}

}

void dequeue()

{

struct node \*tmp=front;

if(front==NULL)

{

cout<<"Queue is empty\n\n";

}

cout<<"Deleted\n";

if(front==rear)

{

front=rear=NULL;

}

else

{

front=front->next;

}

}

void display()

{

if(front==NULL)

{

cout<<"Queue is empty.\n\n";

}

else

{

node \*ptr;

ptr=front;

cout<<"Queue :\n";

while(ptr!=NULL)

{

cout<<ptr->data<<"\t";

cout<<"\n";

ptr=ptr->next;

}

}

}

};

int main()

{

linkqueue q;

int ch,x,n;

do

{

cout<<"1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n";

cout<<"Enter the value for operation :\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value :\n";

cin>>x;

q.enqueue(x);

break;

case 2:

q.dequeue();

break;

case 3:

q.display();

break;

case 4:

break;

default:

cout<<"\n Wrong choice.\n\n";

}

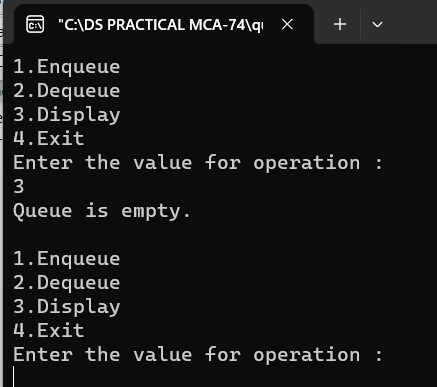
}

while(n!=4);

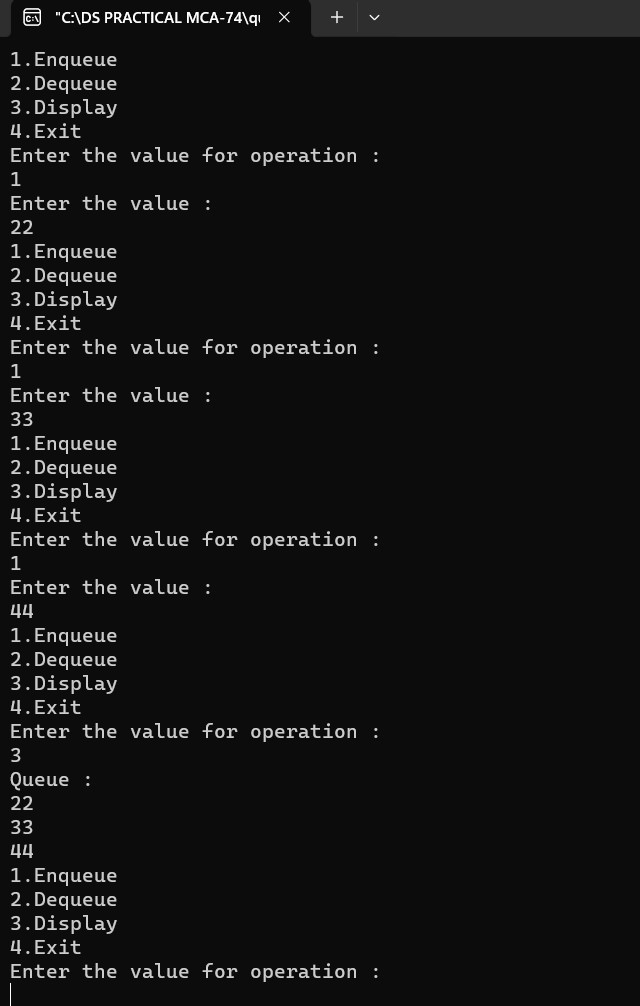
return 0; }

**OUTPUT:**

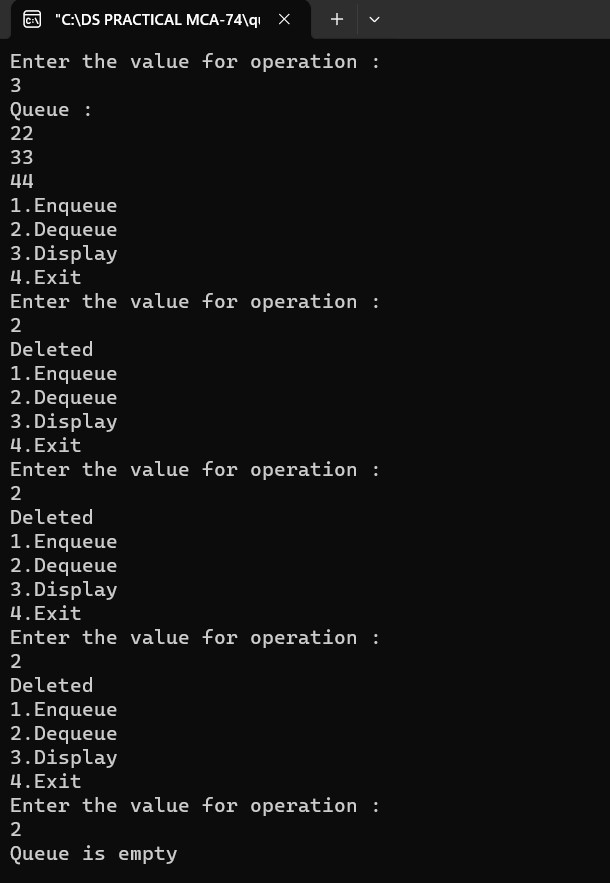
**1)Queue Empty:**



**2)Enqueue:**



**3)Dequeue:**



**PRACTICAL 9**

**Aim : Implementation of Priority Queue.**

**CODE:**

#include<iostream>

using namespace std;

class priority

{

public:

struct node

{

int pr;

int data;

struct node\*next;

};

node \*front=NULL;

void insert(int item, int pr)

{

node \*tmp, \*q;

tmp=new node;

tmp->data=item;

tmp->pr=pr;

if(front==NULL || pr< front->pr)

{

tmp->next=front;

front=tmp;

}

else

{

q=front;

while(q->next!=NULL && q->next->pr<=pr)

q=q->next;

tmp->next=q->next;

q->next=tmp;

}

}

void del()

{

node \*tmp,\*q;

if(front==NULL)

cout<<"Queue is empty";

else

{

tmp=front;

cout<<tmp->data<<" is deleted \n";

front=front->next;

}

}

void display()

{

if(front==NULL)

{

cout<<"Queue is Empty.

\

n

\n";

}

else

{

node \*ptr;

ptr=front;

cout<<"Item\t"<<"Priority\n";

while(ptr!=NULL)

{

cout<<ptr->data<<"\t";57

cout<<ptr->pr<<"\t";

cout<<"\n";

ptr=ptr->next;

}

}

}

};

int main()

{

priority p;

int ch,x,y;

while(ch!=0)

{

cout<<"1.Insert

\n2.Delete

\n3.Display

\n";

cout<<"Enter the choice :

\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the Item :

\n";

cin>>x;

cout<<"Enter the Priority :

\n";

cin>>y;

p.insert(x,y);

break;

case 2:

p.del();

break;

case 3:

p.display();

break;58

default:

cout<<"Enter the correct choice ";

break;

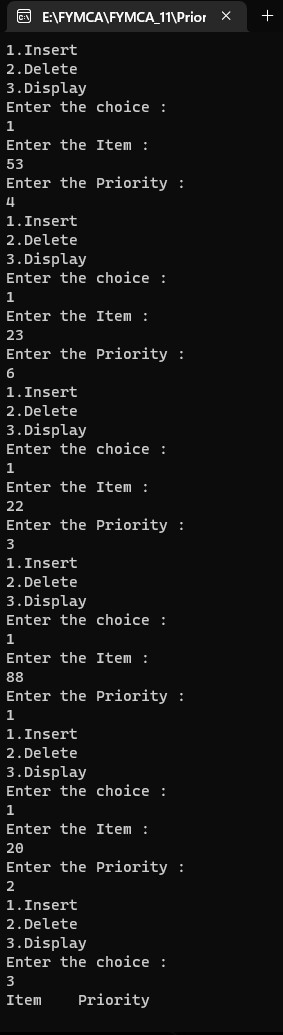
}

}

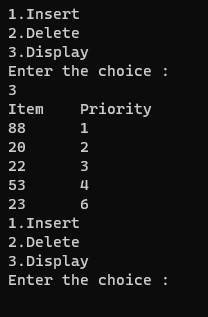
return 0; }

**OUTPUT:**

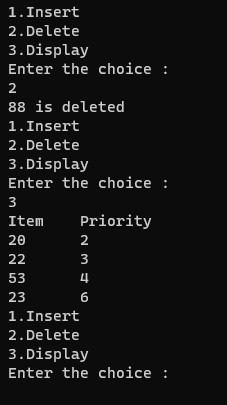
**1)Insert:**



**2)Display:**



**3)Delete:**



**PRACTICAL 10**

**Aim : Implementation of Single Linked List.**

**CODE:**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node

{

Node \*next;

int data;

public:

void add();

void disp(Node \*h);

void merge();

void append();

void intersection();

void unions();

}\*head1=NULL,\*head2=NULL;

void Node:: disp(Node \*h)

{

Node \*tmp;

tmp=h;

while(tmp!=NULL)

{

cout<<tmp->data<<" | ";

tmp=tmp->next;

}

cout<<endl;

}

void Node:: add()

{

int times;

cout<<"Enter the number of elements in First List"<<endl;

cin>>times;

for(int i=0;i<times;i++)

{

cout<<"Enter data"<<endl;

int dat;

cin>>dat;

Node \*tmp,\*q,\*p;

tmp=new Node;

q=head1;

tmp->data=dat;

tmp->next=NULL;

if(head1==NULL)

{

head1=tmp;

}

else

{

while(q!=NULL)

{

if(q->next==NULL)

{

p=q;

}

q=q->next;

}

p->next=tmp;

}

}

cout<<"Elements in List one are : "<<endl;

disp(head1);

cout<<"Enter the number of elements in Second List"<<endl;

cin>>times;

for(int j=0;j<times;j++)

{

cout<<"Enter data"<<endl;

int dat;

cin>>dat;

Node \*tmp,\*q,\*p;

tmp=new Node;

q=head2;

tmp->data=dat;

tmp->next=NULL;

if(head2==NULL)

{

head2=tmp;

}

else

{

while(q!=NULL)

{

if(q->next==NULL)

{

p=q;

}

q=q->next;

}

p->next=tmp;

}

}

cout<<"Elements in List two are : "<<endl;

disp(head2);

}

void Node::intersection()

{

Node \*tmp1,\*tmp2,\*tmp,\*ptr,\*ptr1,\*tn;

tmp1=head1;

tmp2=head2;

tmp=NULL;

tn=NULL;

ptr1=NULL;

while(tmp1!=NULL)

{

tn=tmp2;

while(tn!=NULL)

{

if(tmp1->data==tn->data)

{

tmp=new Node;

tmp->next=ptr1;

tmp->data=tmp1->data;

ptr1=tmp;

}

//ptr1->next=tmp;

tn=tn->next;

}

tmp1=tmp1->next;

}

cout<<"Linked List after insertion is : "<<endl;

disp(ptr1);

}

void Node:: merge()

{

Node \*tmp1,\*tmp2,\*p,\*q;

tmp1=head1;

tmp2=head2;

head2=NULL;

delete head2;

while(tmp1!=NULL && tmp2!=NULL)

{

p=tmp1->next;

tmp1->next=tmp2;

q=tmp2->next;

if(p!=NULL)

tmp2->next=p;

tmp1=p;

tmp2=q;

}

cout<<"Linked List after Merge is : "<<endl;

disp(head1);

}

void Node::unions()

{

Node \*tmp1,\*tmp2,\*tmp,\*q,\*p;

int c=0;

tmp1=head1;

tmp2=head2;

p=NULL;

q=NULL;

tmp=NULL;

while(tmp1!=NULL)

{

tmp=new Node;

tmp->next=NULL;

tmp->data=tmp1->data;

tmp1=tmp1->next;

if(p==NULL)

p=tmp;

else

q->next=tmp;

q=tmp;

}

tmp1=head1;

while(tmp2!=NULL)

{

while(tmp1!=NULL)

{

if(tmp1->data==tmp2->data)

goto a;

tmp1=tmp1->next;

}

tmp=new Node;

tmp->next=NULL;

tmp->data=tmp2->data;

if(p==NULL)

p=tmp;

else

q->next=tmp;

q=tmp;

a:

tmp2=tmp2->next;

}

q->next=NULL;

disp(p);

}

void Node::append()

{

Node\* tmp1,\*p;

tmp1=head1;

while(tmp1!=NULL)

{

p=tmp1;

tmp1=tmp1->next;

if(tmp1==NULL)

{

p->next=head2;

}

}

disp(head1);

delete head2;

}

int main()

{

Node n1;

int choice;

while(1)

{

cout<<"\n1.add\n2.merge\n3.union\n4.intersection\n5.append\n";

cout<<"Enter choice: ";

cin>>choice;

switch(choice)

{

case 1:

n1.add();

break;

case 2:

n1.merge();

break;

case 3:

n1.unions();

break;

case 4:

n1.intersection();

break;

case 5:

n1.append();

break;

default:

cout<<"Please enter correct choice()!!";

}

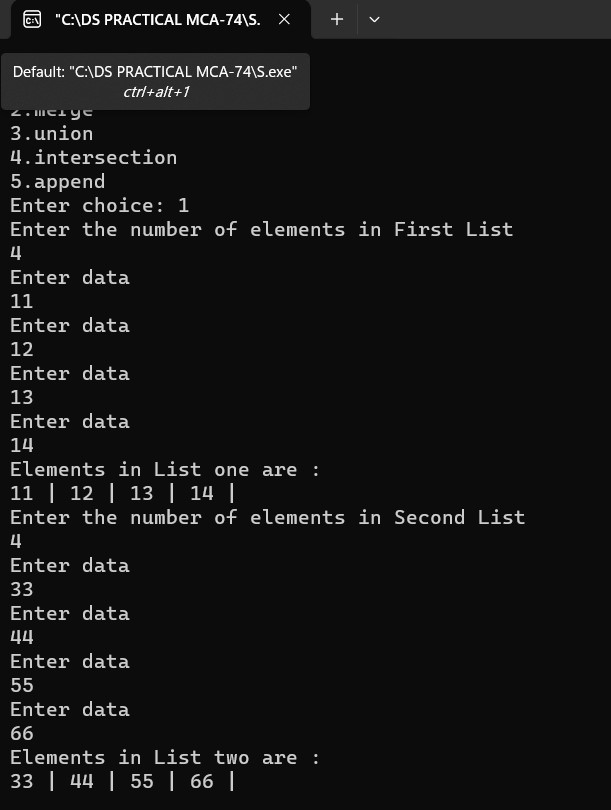
}

return 0;

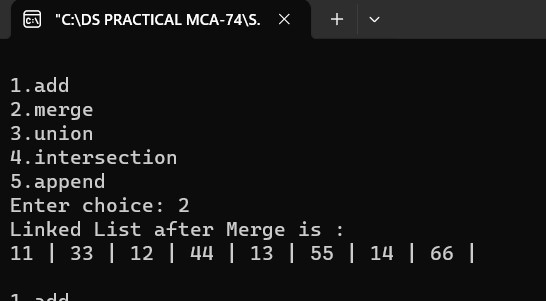
}

**OUTPUT:**

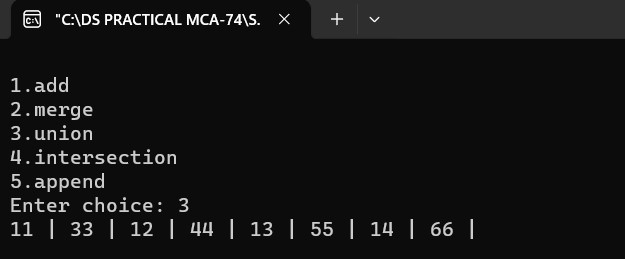
**1)Add:**



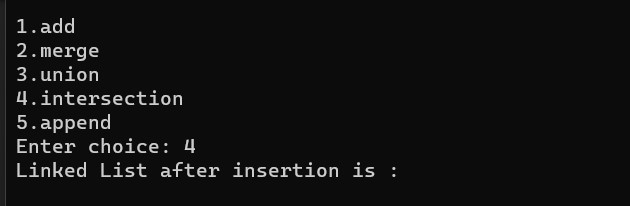
**2)Merge:**



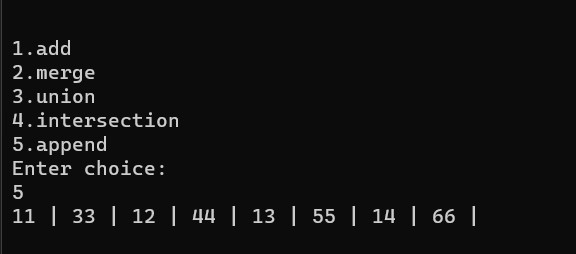
**3)Union:**



**4)Intersection:**



**5)Append:**



**PRACTICAL 11**

**Aim : Implementation of Circular Linked List.**

**CODE:**

#include<iostream>

using namespace std;

class singly\_circular

{

public:

int flag=true;

int pos, i, value, count=0;

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*tmp=NULL;

struct node \*start=NULL;

struct node \*last=NULL;

struct node \*p=NULL;

struct node \*ptr=NULL;

void create(int x)

{

tmp=new node;

tmp->data=x;

if(last==NULL)

{

last=tmp;

tmp->next=last;

}

else

{

tmp->next=last->next;

last->next=tmp;

last=tmp;

}

}

void add\_atbegin(int x)

{

if(last==NULL)

{

cout<<"List is empty.\n";

}

tmp=new node;

tmp->data=x;

tmp->next=last->next;

last->next=tmp;

}

void add\_after(int x,int pos)

{

if(last==NULL)

{

cout<<"List is empty.\n";

}

p=last->next;

for(int i=0;i<pos-1;i++)

{

p=p->next;

if(p==last->next)

{

cout<<"Position does not exist.\n";

//break;

}

}

tmp=new node;

tmp->next=p->next;

tmp->data=x;

p->next=tmp;

if(p==last)

{

last=tmp;

}

}

void del(int x)

{

//p=last->next;

if(last->next==last && last->data==x) // for only one node

{

tmp=last;

last=NULL;

delete(tmp);

return;

}

p=last->next;

if(p->data==x) //first element deleted

{

tmp=p;

last->next=p->next;

delete(tmp);

return;

}

while(p->next!=last)

{

if(p->next->data==x)

{

tmp=p->next;

p->next=tmp->next;

delete(tmp);

//cout<<"Deleted item "<<x;

return;

}//delete element in between

p=p->next;

}

if(p->next->data==x)

{

tmp=p->next;

p->next=last->next;

delete(tmp);

last=p;

return;

}//last element deleted

cout<<"Element not found.\n";

}

void search1(int x)

{

int pos=1;

while(p->next!=last)

{

if(p->data==x)

{

cout<<"Element found at position "<<pos-1<<".\n";

}

p=p->next;

pos++;

}

if(p==NULL)

cout<<"Item not found.\n";

}

void sort()

{

int x;

if(last==NULL)

{

cout<<"List is empty.\n\n";

}

p=last->next;

while(p!=last)

{

ptr=p->next;

while(ptr!=last->next)

{

if(ptr!=last->next)

{

if(p->data>ptr->data)

{

x=p->data;

p->data=ptr->data;

ptr->data=x;

}

}

ptr=ptr->next;

}

p=p->next;

}

}

int count1()

{

if(last==NULL)

{

cout<<"List is empty.\n\n";

}

else

{

p=last->next;

while(p!=last)

{

count++;

p=p->next;

}

count++;

cout<<"Number of element are "<<count<<"\n";

}

}

void display()

{

if(last==NULL)

{

cout<<"List is empty.\n\n";

return;

}

p=last->next;

cout<<"\nSingly Circular Linked List :\n";

while(p!=last)

{

cout<<p->data<<" -> ";

p=p->next;

}

cout<<last->data<<"\n\n";

}

};

int main()

{

singly\_circular d;

int x,ch;

int pos;

while(ch!=9)

{

cout<<"1.Create a list\n2.Add at begin\n3.Add after\n4.Search\n";

cout<<"5.Sort\n6.Count\n7.Display\n8.Delete\n9.Exit\n";

cout<<"Enter the choice:\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value :\n";

cin>>x;

d.create(x);

d.display();

break;

case 2:

cout<<"Enter the value :\n";

cin>>x;

d.add\_atbegin(x);

d.display();

break;

case 3:

cout<<"Enter the position :\n";

cin>>pos;

cout<<"Enter the value :\n";

cin>>x;

d.add\_after(x,pos);

d.display();

break;

case 4:

cout<<"Enter element to be searched:\n";

cin>>x;

d.search1(x);

d.display();

break;

case 5:

cout<<"Before sorting -";

d.display();

d.sort();

cout<<"After sorting -";

d.display();

break;

case 6:

d.count1();

d.display();

break;

case 7:

d.display();

break;

case 8:

cout<<"Enter the element to be delete :\n";

cin>>x;

d.del(x);

d.display();

break;

case 9:

break;

default:

cout<<"Wrong choice.\n";

}

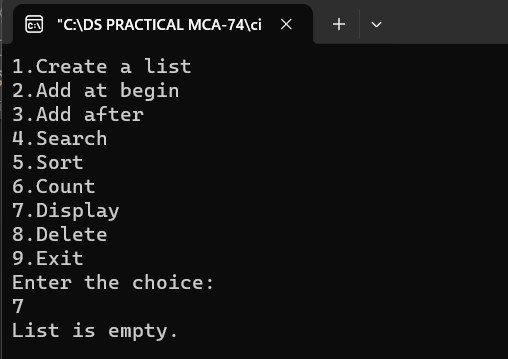
}

return 0;

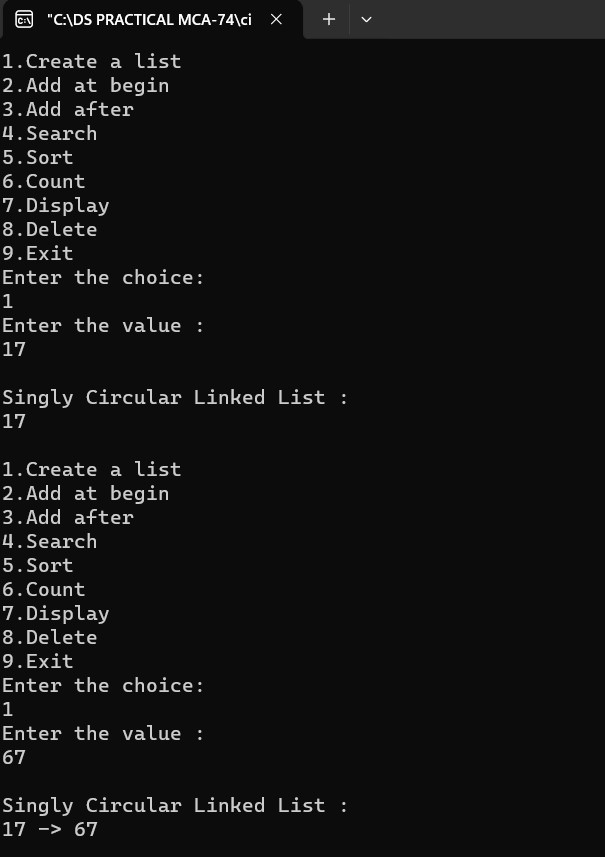
}

**OUTPUT:**

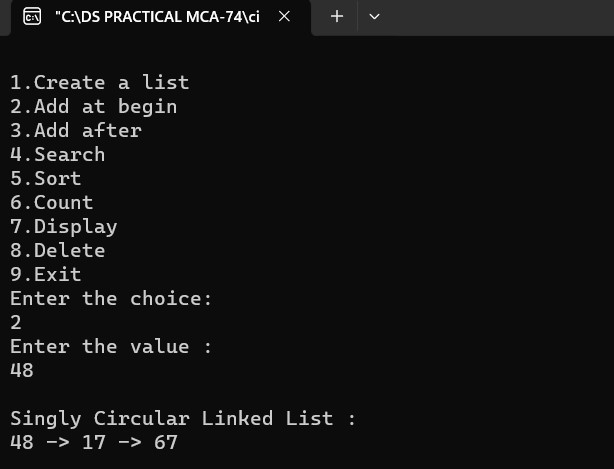
1. **List Empty:**



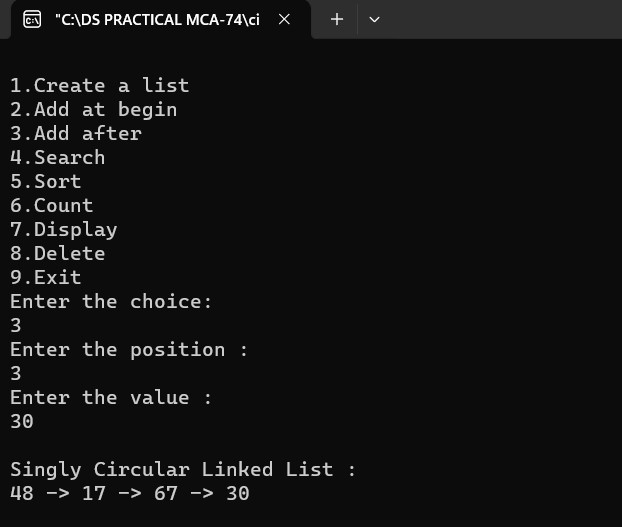
**2) Create List:**



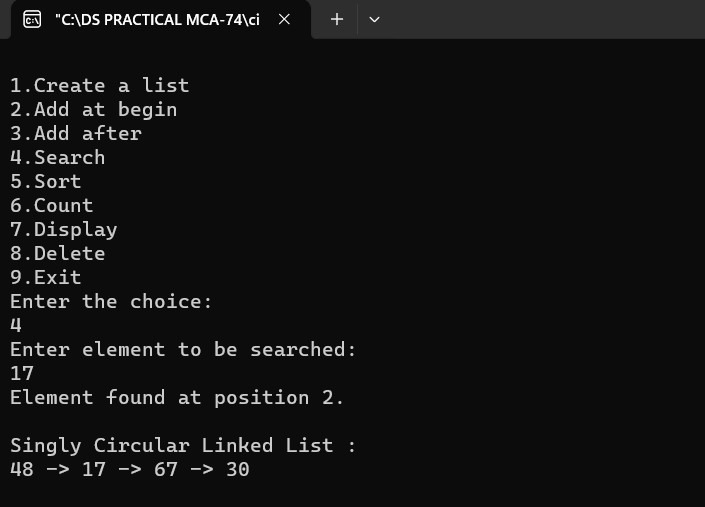
**3)Add At Begin:**



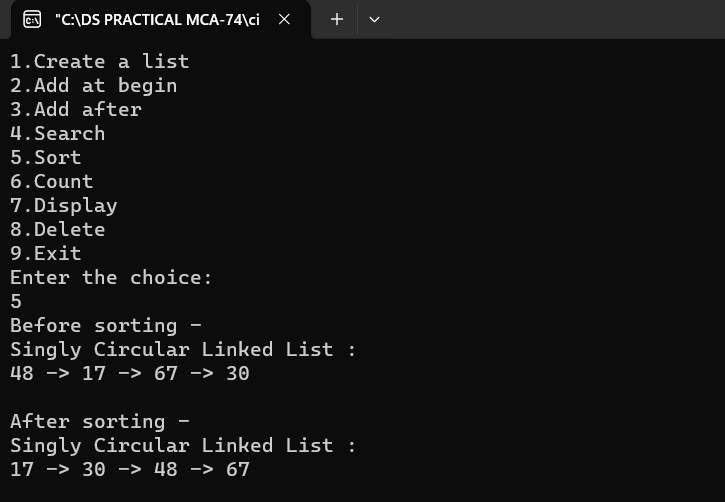
**4)Add After:**



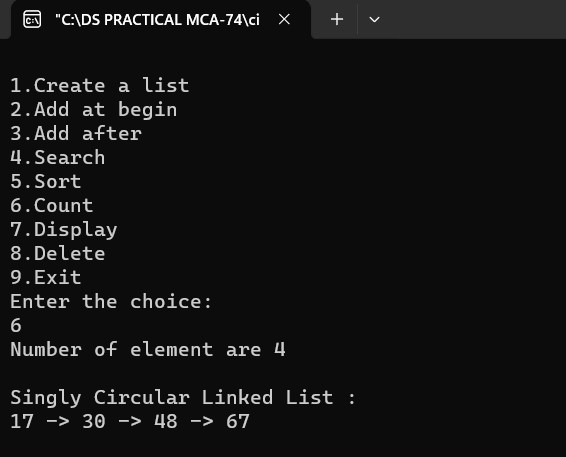
1. **Search:**



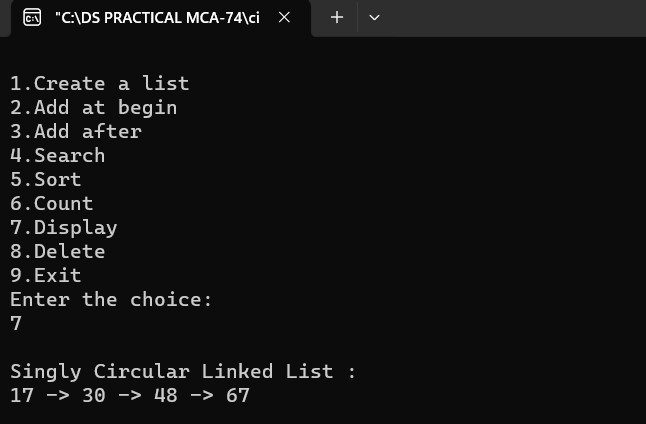
1. **Sort:**



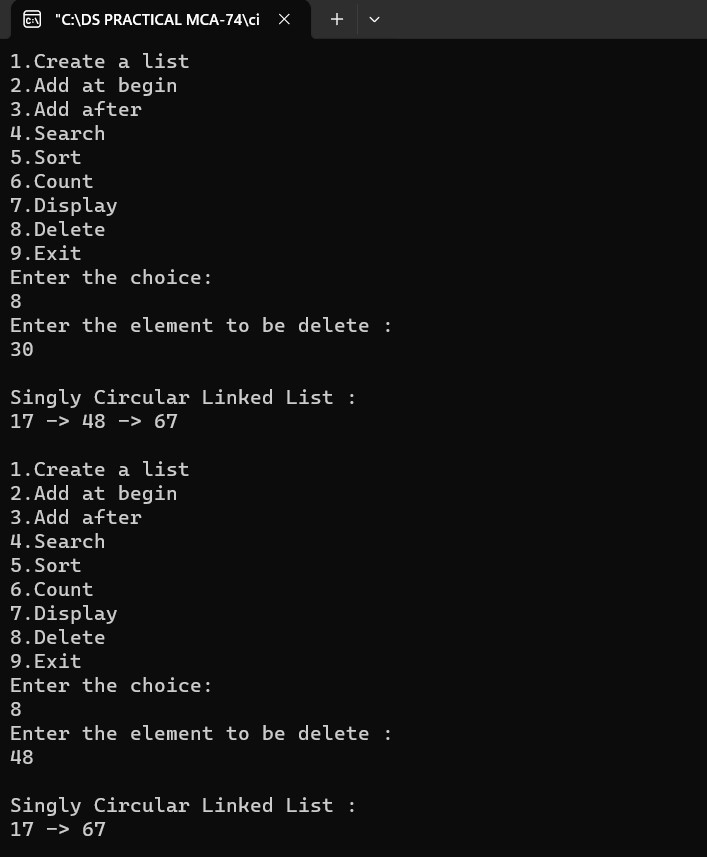
1. **Count:**



1. **Display:**



1. **Delete:**



**PRACTICAL 12**

**Aim : Implementation of Doubly Linked List.**

**CODE:**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node

{

public:

Node \*next;

int info;

}\*front=NULL,\*rear=NULL;

void fpush()

{

int item;

Node \*temp,\*q,\*p;

cout<<"Enter the data :- ";

cin>>item;

temp=new Node;

temp->info=item;

temp->next=NULL;

if(front==NULL)

{

front=temp;

rear=temp;

}

else

{

q=front;

while(q!=NULL)

{

if(q->next==NULL)

{

cout<<"q->data : "<<q->info;

p=q;

}

q=q->next;

}

p->next=temp;

rear=temp;

}

}

void rpush()

{

int data;

Node \*temp,\*q,\*p;

cout<<"Enter the data :- ";

cin>>data;

temp=new Node;

temp->info=data;

temp->next=NULL;

if(front==NULL)

{

front=temp;

rear=temp;

}

else

{

q=front;

while(q!=NULL)

{

if(q->next==NULL)

{

cout<<"q->data : "<<q->info;

p=q;

}

q=q->next;

}

p->next=temp;

rear=temp;

}

}

void fpop()

{

Node\* temp,\*p;

temp=front;

front=temp->next;

delete temp;

/\* while(temp!=NULL)

{

if(temp->next==NULL)

{

p=temp;

}

temp=temp->next;

}\*/

//delete p;

}

void rpop()

{

Node \*temp, \*p;

temp=front;

front=temp->next

delete temp;

}

void display()

{

Node \*tmp;

tmp=front;

cout<<"Elements in queue are : "<<endl;

while(tmp!=NULL)

{

cout<<tmp->info<<" | ";

tmp=tmp->next;

}

cout<<endl;

}

int main()

{

int ch;

while(1)

{

cout<<"\n press 1 for insert from front";

cout<<"\n press 2 for insert from rear";

cout<<"\n press 3 for Delete from front";

cout<<"\n press 4 for Delete from rear";

cout<<"\n press 5 for Display";

cout<<"\n press 6 for exit";

cin>>ch;

switch(ch)

{

case 1:

fpush();

break;

case 2:

rpush();

break;

case 3:

fpop();

break;

case 4:

rpop();

break;

case 5:

display();

break;

case 6:

exit(0);

break;

default:

cout<<"Please enter correct choice :- ";

}

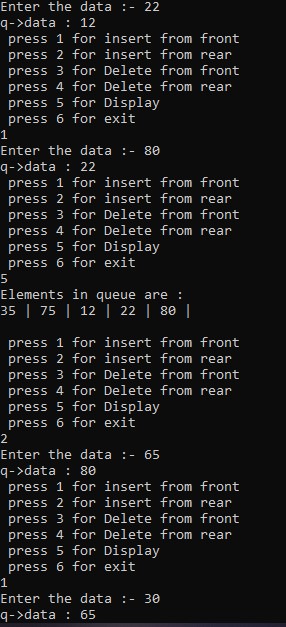
}

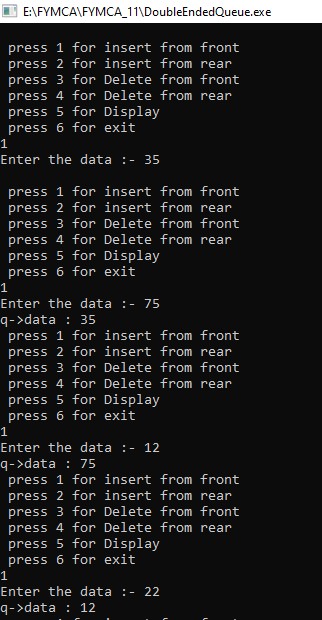
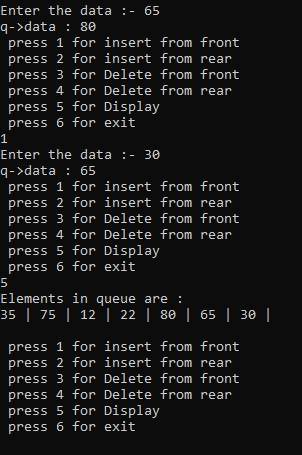
getch();

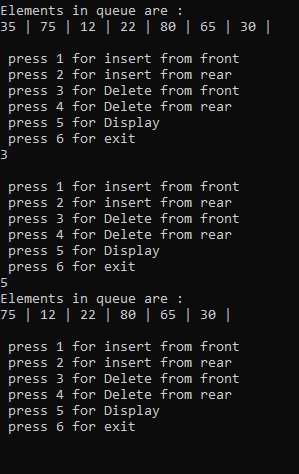
return 0;

}

**OUTPUT:**







**PRACTICAL 13**

**Aim : To Write C++ Program to perform Polynomial Addition**

**CODE:**

#include<iostream>

using namespace std;

struct Node{

int coeff;

int pow;

struct Node \*next;

};

void create\_node(int x, int y, struct Node \*\*temp){

struct Node \*r, \*z;

z = \*temp;

if(z == NULL){

r =(struct Node\*)malloc(sizeof(struct Node));

r->coeff = x;

r->pow = y;

\*temp = r;

r->next = (struct Node\*)malloc(sizeof(struct Node));

r = r->next;

r->next = NULL;

} else {

r->coeff = x;

r->pow = y;

r->next = (struct Node\*)malloc(sizeof(struct Node));

r = r->next;

r->next = NULL;

}

}

void polyadd(struct Node \*p1, struct Node \*p2, struct Node \*result){100

Roll No: 11

NCRD Sterling Institute of Management Studies

while(p1->next && p2->next){

if(p1->pow > p2->pow){

result->pow = p1->pow;

result->coeff = p1->coeff;

p1 = p1->next;

}

else if(p1->pow < p2->pow){

result->pow = p2->pow;

result->coeff = p2->coeff;

p2 = p2->next;

} else {

result->pow = p1->pow;

result->coeff = p1->coeff+p2->coeff;

p1 = p1->next;

p2 = p2->next;

}

result->next = (struct Node \*)malloc(sizeof(struct Node));

result = result->next;

result->next = NULL;

}

while(p1->next || p2->next){

if(p1->next){

result->pow = p1->pow;

result->coeff = p1->coeff;

p1 = p1->next;

}

if(p2->next){

result->pow = p2->pow;

result->coeff = p2->coeff;

p2 = p2->next;

}101

Roll No: 11

NCRD Sterling Institute of Management Studies

result->next = (struct Node \*)malloc(sizeof(struct Node));

result = result->next;

result->next = NULL;

}

}

void printpoly(struct Node \*node){

while(node->next != NULL){

printf("%dx^%d", node->coeff, node->pow);

node = node->next;

if(node->next != NULL)

printf(" + ");

}

}

int main(){

struct Node \*p1 = NULL, \*p2 = NULL, \*result = NULL;

create\_node(51,7,&p1);

create\_node(21,5,&p1);

create\_node(56,0,&p1);

create\_node(11,5,&p2);

create\_node(16,2,&p2);

printf("polynomial 1: ");

printpoly(p1);

printf("\npolynomial 2: ");

printpoly(p2);

result = (struct Node \*)malloc(sizeof(struct Node));

polyadd(p1, p2, result);

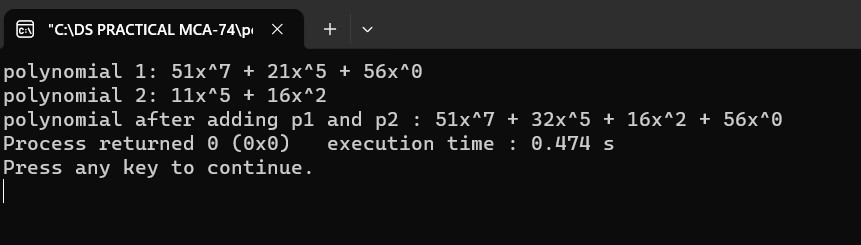
printf("\npolynomial after adding p1 and p2 : ");

printpoly(result);

return 0;

}

**OUTPUT:**



**PRACTCAL 14**

**Aim : Create and perform various operations on Binary Search**

**Tree.**

**CODE:**

#include<iostream>

#include<process.h>

#include<conio.h>

using namespace std;

struct node{

int data;

struct node \*left;

struct node \*right;};

class BST{

public:

node \*tree;

BST() {

tree=NULL;

}

void createTree(node\*\*,int item);

void preorder(node \*);

void inorder(node \*);

void postorder(node \*);

int totalNodes(node \*);

void removeTree(node \*\*);

void findsmallestNode(node \*);

void findLargestNode(node \*);

void deleteNode(int);};

void BST::createTree(node \*\*tree,int item){

if (\*tree==NULL) {

\*tree=new node;

(\*tree)->data=item;

(\*tree)->left=NULL;

(\*tree)->right=NULL; }

else {if((\*tree)->data>item)

createTree(&((\*tree)->left),item);

else

createTree(&((\*tree)->right),item);}}

void BST::preorder(node\*tree){

if(tree!=NULL){

cout<<" "<<tree->data;

preorder(tree->left);

preorder(tree->right);}}

void BST::inorder(node\*tree){

if(tree!=NULL){

inorder(tree->left);

cout<<" "<<tree->data;

inorder(tree->right);}}

void BST::postorder(node\*tree){

if(tree!=NULL){

postorder(tree->left);

postorder(tree->right);

cout<<" "<<tree->data;}}

int BST::totalNodes(node \*tree){

if(tree==NULL)

return 0;

else

return(totalNodes(tree->left)+totalNodes(tree->right)+1);}

void BST::removeTree(node \*\*tree){

if((\*tree)!=NULL){

removeTree(&(\*tree)->left);

removeTree(&(\*tree)->right);

delete(\*tree);}}

void BST::findsmallestNode(node \*tree){

if(tree==NULL||tree->left==NULL)

cout<<tree->data;

else

findsmallestNode(tree->left);}

node \*find\_Insucc(node \*curr){

node \*succ=curr->right;

if(succ!=NULL){

while(succ->left!=NULL)

succ=succ->left;}return(succ);}

void BST::findLargestNode(node \*tree){

if(tree==NULL||tree->right==NULL)

cout<<tree->data;

else

findLargestNode(tree->right);}

void BST::deleteNode(int item){

node \*curr=tree,\*succ,\*pred;

int flag=0,delcase;

while(curr!=NULL && flag!=1){

if(item<curr->data){

pred=curr;

curr=curr->left;}

else if(item>curr->data){

pred=curr;

curr=curr->right;}

else{flag=1;}}

if(flag==0){

cout<<"\n item does not exist:no deletion\n";

getch();}

if(curr->left==NULL && curr->right==NULL)

delcase=1;

else if(curr->left!=NULL && curr->right!=NULL)

delcase=3;

else

delcase=2;

if(delcase==1){

if(pred->left==curr)

pred->left=NULL;

else

pred->right=NULL;

delete(curr);

pred->right;}

if(delcase==2){

if(pred->left==curr){

if(curr->left==NULL)

pred->left=curr->right;

else

pred->left=curr->left;}

else{

if(curr->left==NULL)

pred->right=curr->right;

else

pred->right=curr->left;

delete(curr);}

if(delcase==3){

succ=find\_Insucc(curr);

int item1=succ->data;

deleteNode(item1);

curr->data=item1;}}}

int main(){

BST obj;

int choice;

int height=0,total=0,n,item;

node \*\*tmp;

while(1){

cout<<"\n Binary search tree common operation\n";

cout<<"1)Create Tree \n";

cout<<"2)Traversal \n";

cout<<"3)Total Nodes\n";

cout<<"4)Remove Tree\n";

cout<<"5)Insert Nodes\n";

cout<<"6)Find Smallest Nodes \n";

cout<<"7)Find Largest Node \n";

cout<<"8)Delete Node\n";

cout<<"9)Exit\n";

cout<<"Enter your choice :";

cin>>choice;

switch(choice){

case 1:

cout<<"\n Creating Tree----";

cout<<"How many nodes u want to enter :";

cin>>n;

for(int i=0;i<n;i++){

cout<<"Enter Values :";

cin>>item;

obj.createTree(&obj.tree,item);}

break;

case 2:

cout<<"\n Inorder Traversal :";

obj.inorder(obj.tree);

cout<<"\n preorder Traversal :";

obj.preorder(obj.tree);

cout<<"\n Postorder Traversal :";

obj.postorder(obj.tree);

getch();break;

case 3:

total=obj.totalNodes(obj.tree);

cout<<"Total nodes :"<<total;

getch();break;

case 4:

obj.removeTree(&obj.tree);

cout<<"\n Tree is removed from memory";

getch();break;

case 5:

cout<<"\n Insert node in a tree \n";

cout<<"Enter value :";

cin>>item;

obj.createTree(&obj.tree,item);

cout<<"\nItem is inserted\n";

getch();break;

case 6:

cout<<"\n\nSmallest node is:\n";

obj.findsmallestNode(obj.tree);

getch();break;

case 7:

cout<<"\n\nLargest node is:\n";

obj.findLargestNode(obj.tree);

getch();break;

case 8:

cout<<"\n\n Deleting a node from a tree--\n";

cout<<"Enter value=";

cin>>item;

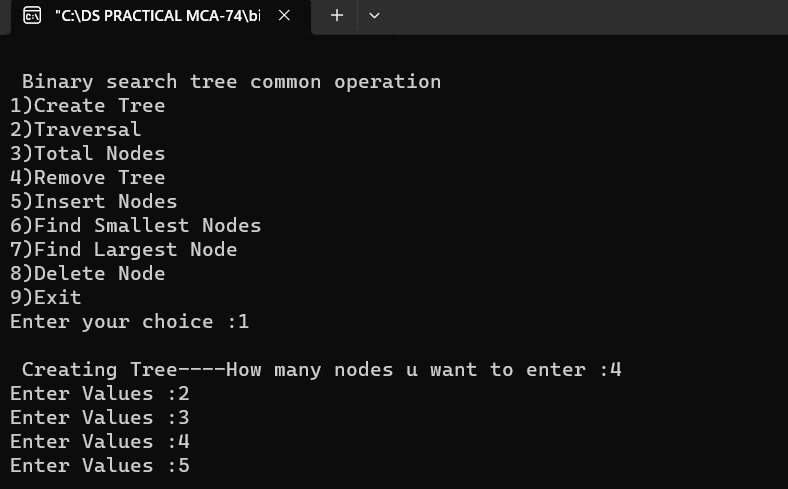
obj.deleteNode(item);

break;

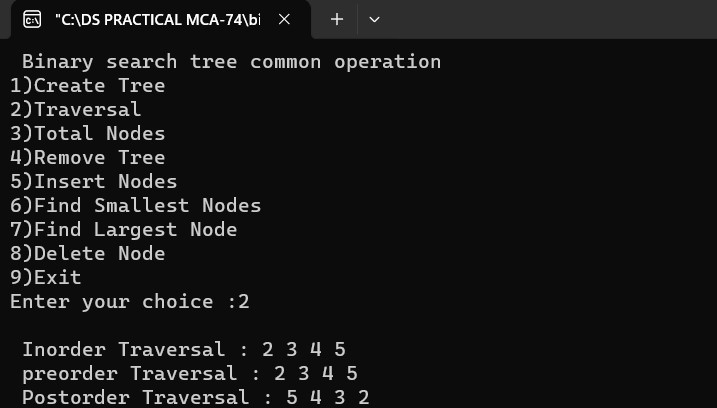
}}}

**OUTPUT:**

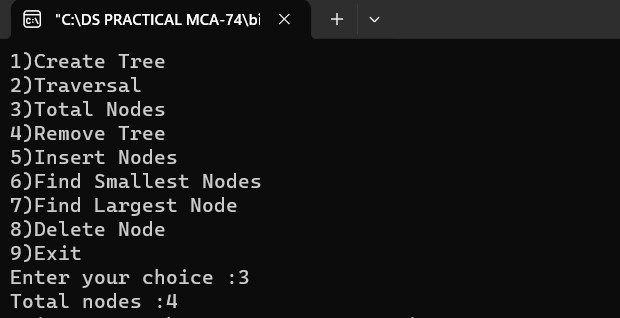
1. **Create Tree:**



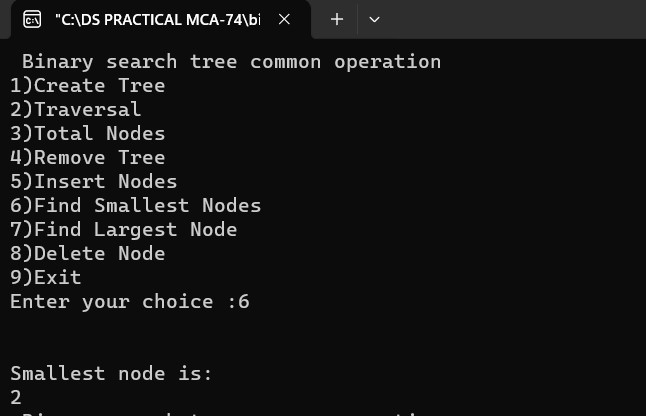
**2)Transversal:**



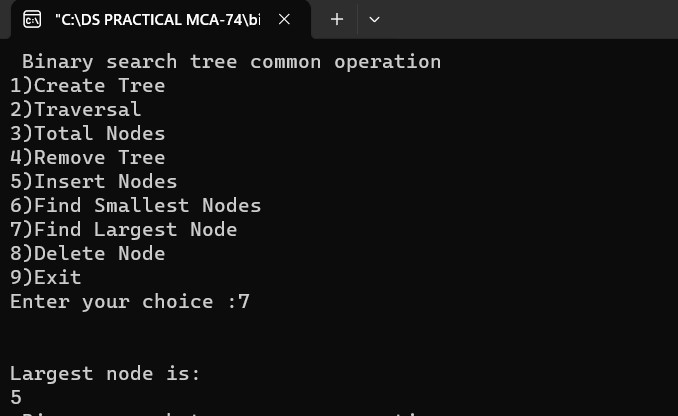
**3)Total Node:**



**4)Smallest Node:**



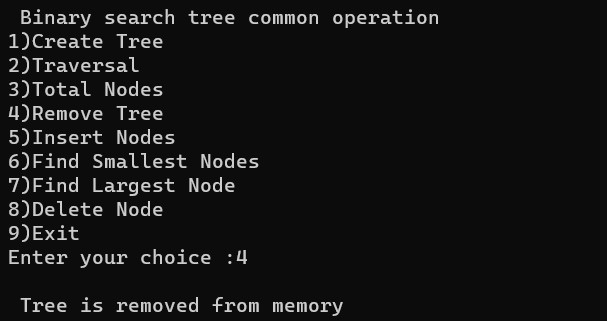
**5)Largest node:**



**6)Insert Node:**



**7)Remove node:**



**PRACTICAL 15**

**Aim : Implementing Heap with different operations performed.**

**1)Min Heap:**

**CODE:**

#include <iostream>

#include <conio.h>

using namespace std;

void min\_heapify(int \*a,int i,int n)

{

int j, temp;

temp = a[i];

j = 2 \* i;

while (j <= n)

{

if (j < n && a[j+1] < a[j])

j = j + 1;

if (temp < a[j])

break;

else if (temp >= a[j])

{

a[j/2] = a[j];

j = 2 \* j;

}

}

a[j/2] = temp;

return;

}

void build\_minheap(int \*a, int n)

{

int i;

for(i = n/2; i >= 1; i--)

{

min\_heapify(a,i,n);

}

}

int main()

{

int n, i, x;

cout<<"Enter the number of elements of array:\n";

cin>>n;

int a[20];

for (i = 1; i <= n; i++)

{

cout<<"Enter element "<<(i)<<endl;

cin>>a[i];

}

build\_minheap(a, n);

cout<<"Min Heap\n";

for (i = 1; i <= n; i++)

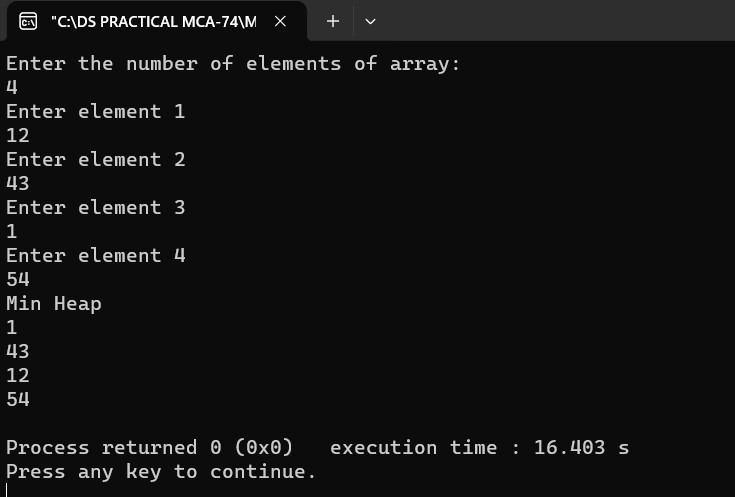
{

cout<<a[i]<<endl;

}

}

**OUTPUT:**



**2)Max Heap:**

**CODE:**

#include <iostream>

#include <conio.h>

using namespace std;

void max\_heapify(int \*a, int i, int n)

{

int j, temp;

temp = a[i];

j = 2 \* i;

while (j <= n)

{

if (j < n && a[j+1] > a[j])

j = j + 1;

if (temp > a[j])

break;

else if (temp <= a[j])

{

a[j / 2] = a[j];

j = 2 \* j;}

}

a[j/2] = temp;

return;

}

void build\_maxheap(int \*a,int n)

{

int i;

for(i = n/2; i >= 1; i--)

{

max\_heapify(a,i,n);

}

}

int main()

{

int n, i, x;

cout<<"Enter the number of elements of array : \n";

cin>>n;

int a[20];

for (i = 1; i <= n; i++)

{

cout<<"Enter the element "<<(i)<<endl; cin>>a[i];

}

build\_maxheap(a,n);

cout<<"Max Heap\n";

for (i = 1; i <= n; i++)

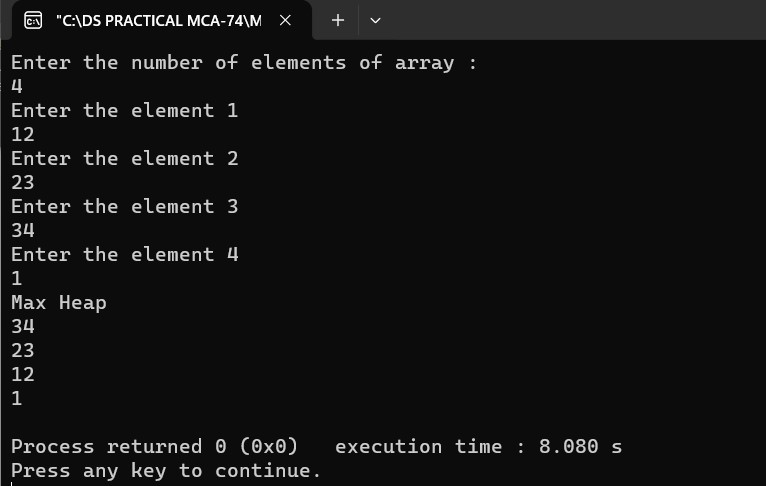
{

cout<<a[i]<<endl;

}

}

**OUTPUT:**



**PRACTICAL 16**

**Aim : Implementation of Depth First Search and Breadth First Search.**

**1)Depth First Search:**

**CODE:**

#include<iostream>

#include<stdio.h>

#define max 20

using namespace std;

int adj[max][max];

bool visited[max];

int n;

int frnt;

void create\_graph()

{

int i, max\_edges,origin,destin;

cout<<"Enter no. of nodes: ";

cin>>n;

max\_edges=n\*(n-1);

for(i=1;i<=max\_edges;i++)

{

cout<<"Enter edge (0 0 to quit) : "<<i<<"\n";

cin>>origin>>destin;

if(origin==0||destin==0)

break;

if(origin>n||destin>n||origin<=0||destin<=0)

{

cout<<"Invalid edge \n";

i--;

}

else

{

adj[origin][destin]=1;

}

}

}

void display()

{

int i, j;

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

cout<<adj[i][j]<<"\t";

}

cout<<"\n";

}

}

void dfs(int v)

{

int i, stack[max], top=-1,pop\_v, j, t;

int c;

top++;

stack[top]=v;

while(top>=0)

{

pop\_v=stack[top];

top--;

if(visited[pop\_v]==false)

{

cout<<pop\_v;

visited[pop\_v]=true;

}

else

continue;

for(i=n;i>=1;i--)

{

if(adj[pop\_v][i]==1 && visited[i]==false)

{

top++;

stack[top]=i;

}

}

}

}

void bfs(int v)

{

int i, frnt, rear;

int que[20];

frnt=rear=-1;

cout<<v;

visited[v]=true;

rear++;

frnt++;

que[rear]=v;

while(frnt<=rear)

{

v=que[frnt];

frnt++;

for(i=1;i<=n;i++)

{

if(adj[v][i]==1&&visited[i]==false)

{

cout<<i<<"\t";

visited[i]=true;

rear++;

que[rear]=i;

}

}

}

}

void adj\_nodes(int v)

{

int i;

for(i=1;i<=n;i++)

{

int i;

for(i=1;i<=n;i++)

{

if(adj[v][i]==1)

cout<<i;

cout<<"\n";

}

}

}

int main()

{

int i, v, ch;

create\_graph();

while(1)

{

cout<<"\n";

cout<<"1. Adjacency Matrix \n";

cout<<"2. Depth first search using stack\n";

cout<<"3. Breadth first search\n";

cout<<"4. exit \n";

cout<<"Enter your choice\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Adjacency Matrix \n";

display();

break;

case 2:

cout<<"Enter starting node for Depth First Search: \n";

cin>>v;

for(i=1;i<=n;i++)

visited[i]=false;

dfs(v);

break;

case 3:

cout<<"Enter starting node for Breadth First Search: \n";

cin>>v;

for(i=1;i<=n;i++)

visited[i]=false;

bfs(v);

break;

case 4:

break;

default:

cout<<"Wrong Choice";

break;

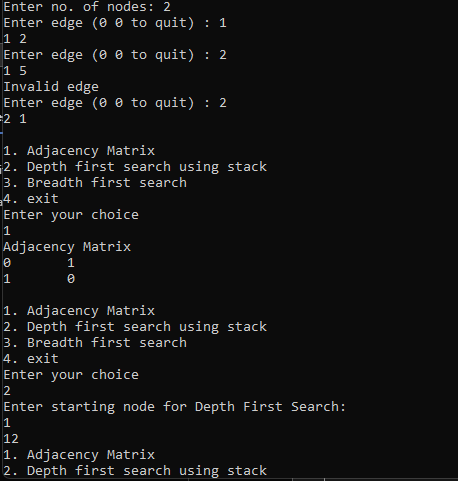
}

}

return 0;

}

Output:-



**2)Breadth First Search:**

**CODE:**

#include<iostream>

#include<conio.h>

#include<iomanip>

using namespace std;

int adj[10][10],n,visited[10];

void bfs(int v)

{

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

visited[v]=1;

q[++rear]=v;

cout<<"Visiting order...\n";

while(front!=rear)

{

v=q[++front];

cout<<v;

for(i=0;i<n;i++)

{

if(!visited[i] && adj[v][i])

{

visited[i]=1;

q[++rear]=i;

}

}

} }

int main()

{

int i,j,m,a,b,v;

char c;

cout<<"\nEnter the no of nodes And no of edges:";

cin>>n>>m;

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

adj[i][j]=0;

}

}

for(i=1;i<=m;i++)

{

cout<<"enter an edge\n";

cin>>a>>b;

adj[a][b]=1;

adj[b][a]=1;

}

do

{

cout<<"Adjancency matrix\n";

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

cout<<setw(4)<<adj[i][j];

}

cout<<"\n";

}

cout<<"enter initial value\n";

cin>>v;

for(i=0;i<n;i++)

{

visited[i]=0;

}

bfs(v);

cout<<"\nDo u wish to continue(y/n)????";

cin>>c;

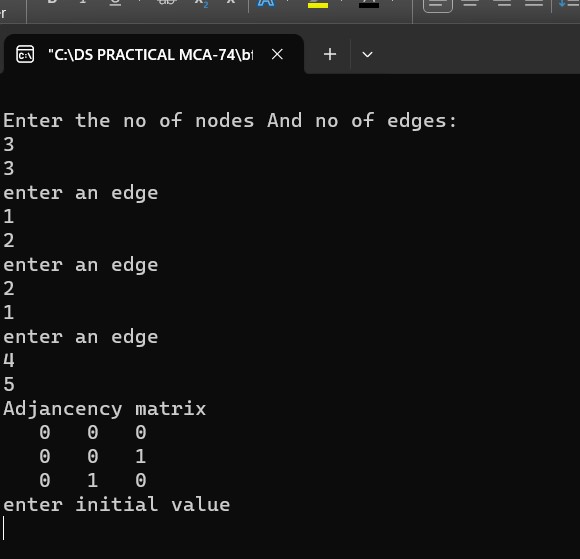
}

while(c!='n');

getch();

}

**OUTPUT:**



**PRACTICAL17**

**Aim : Write a program to create a graph storage structure (e.g.**

**Adjacency matrix).**

**CODE:**

#include<iostream>

using namespace std;

class adjMatrix

{

int \*\*adj;

bool \*visited;

int n,i,j;

public:

adjMatrix(int n)

{

this->n=n;

visited=new bool[n];

adj=new int \*[n];

for(i=1;i<=n;i++)

{

adj[i]=new int [n];

for(j=1;j<=n;j++)

{

adj[i][j]=0;

}

}

}

int add\_edge(int origin, int dest)

{

if(origin>n||dest>n||origin<0||dest<0)

{

cout<<"Wrong nodes";

}

else

{

adj[origin][dest]=1;

}

}

int display()

{

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

cout<<adj[i][j]<<"\t";

}

cout<<"\n";

}

}

};

int main()

{

int nodes, Max\_edges,i, origin, dest;

cout<<"Enter maximum node: ";

cin>>nodes;

adjMatrix am(nodes);

Max\_edges=nodes\*(nodes-1);

cout<<"Enter -1 -1 to exit";

for(i=0;i<Max\_edges;i++)

{

cout<<"\nEnter edges: ";

cin>>origin>>dest;

if((origin==-1)&&(dest==-1))

break;

else

am.add\_edge(origin,dest);

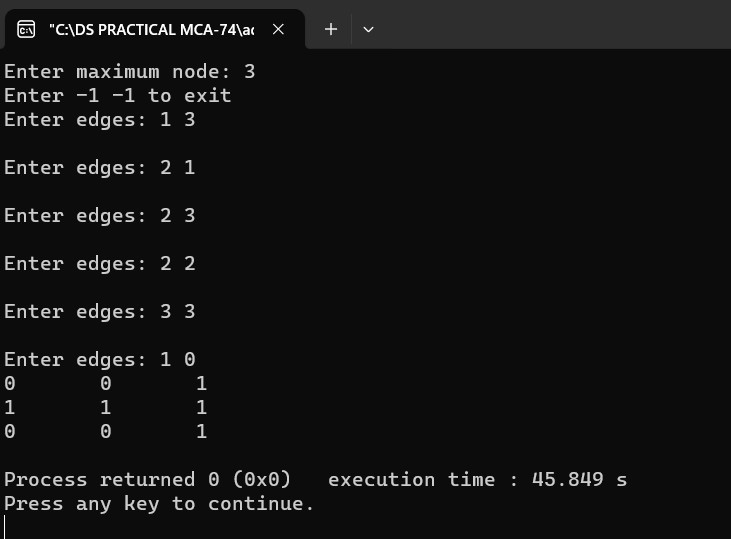
}

am.display();

return 0;

}

**OUTPUT:**



**PRACTICAL 18**

**Aim : Perform various hashing techniques with Linear Probe as**

**collision resolution scheme.**

**CODE:**

#include<iostream>

#include<iomanip>

using namespace std;

int hashsearch(int a[],int x,int n)

{

int index,start;

index=x%n;

if(a[index]==x)

return index;

else

if(a[index]==-1)

return -1;

else

{

start=index;

do

{

index=(index+1)%n;

if(a[index]==x)

return index;

else

if(a[index]==-1)

break;

}

while(index !=start);

return -1;

}

}

int main()

{

int hsh[10],i,x,index,k;

cout<<"Hashing using linear probing\n\n";

cout<<"Hash table creation\n\n";

for(i=0;i<10;i++)

hsh[i]=-1;

for(i=1;i<=7;i++)

{

cout<<"\nEnter a number: ";

cin>>x;

index=x%10;

while(hsh[index]!=-1)

index=(index+1)%10;

hsh[index]=x;

}

cout<<"\nHASH TABLE\n";

for(i=0;i<10;i++)

cout<<setw(4)<<i;

cout<<"\n";

for(i=0;i<10;i++)

cout<<setw(4)<<hsh[i];

do

{

cout<<"\nelement to be searched to stop enter -1\n";

cin>>x;

if(x>=0)

{

k=hashsearch(hsh,x,10);

if(k>=0)

cout<<x<<"is present at hsh["<<k<<"]\n";

else

cout<<x<<"is not present\n";

}

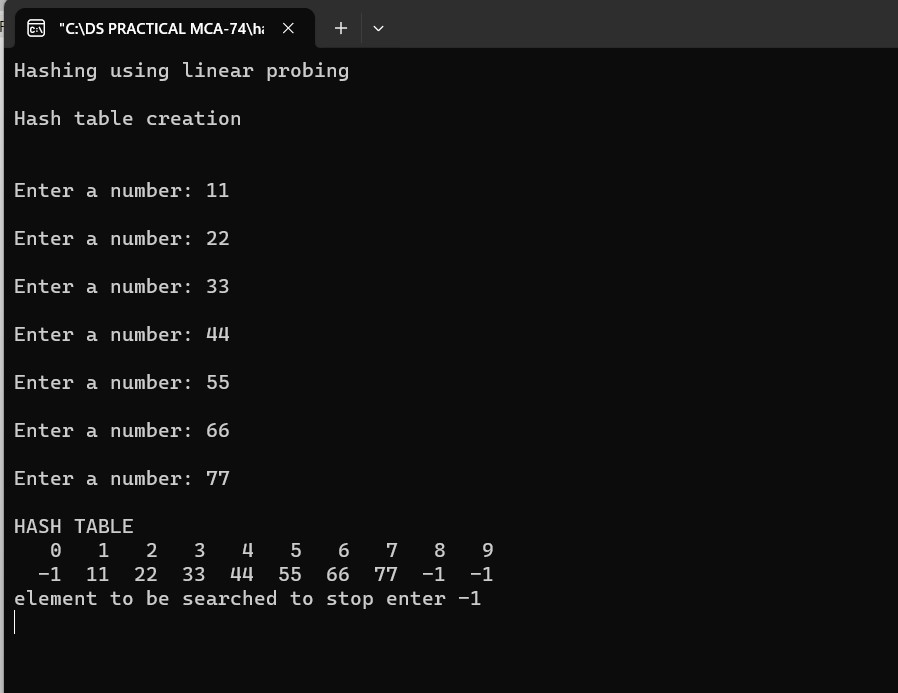
}

while(x>=0);

return 0;

}

**OUTPUT:**



**PRACTICAL 19**

**Aim : Write a program to create a minimum spanning tree using**

**Kruskal’s Algorithm.**

**CODE:**

#include<bits/stdc++.h>

#include<iostream>

using namespace std;

typedef pair<int, int> iPair;

struct Graph

{

int V, E;

vector< pair<int, iPair> > edges;

Graph(int V, int E)

{

this->V = V;

this->E = E;

}

void addEdge(int u, int v, int w)

{

edges.push\_back({w, {u, v}});

}

int kruskalMST();

};

struct DisjointSets

{

int \*parent, \*rnk;

int n;

DisjointSets(int n)

{

this->n = n;

parent = new int[n+1];

rnk = new int[n+1];

for (int i = 0; i <= n; i++)

{

rnk[i] = 0;

parent[i] = i;

}

}

int find(int u)

{

if (u != parent[u])

parent[u] = find(parent[u]);

return parent[u];

}

void merge(int x, int y)

{

x = find(x), y = find(y);

if (rnk[x] > rnk[y])

parent[y] = x;

else

parent[x] = y;

if (rnk[x] == rnk[y])

rnk[y]++;

}

};

int Graph::kruskalMST()

{

int mst\_wt = 0;

sort(edges.begin(), edges.end());

DisjointSets ds(V);

vector< pair<int, iPair> >::iterator it;

for (it=edges.begin(); it!=edges.end(); it++)

{

int u = it->second.first;

int v = it->second.second;

int set\_u = ds.find(u);

int set\_v = ds.find(v);

if (set\_u != set\_v)

{

cout << u << " - " << v << endl;

mst\_wt += it->first;

ds.merge(set\_u, set\_v);

}

}

return mst\_wt;

}

int main()

{

int V = 9, E = 14;

Graph g(V, E);

g.addEdge(0, 1, 3);

g.addEdge(0, 7, 8);

g.addEdge(1, 2, 6);

g.addEdge(1, 7, 7);

g.addEdge(2, 3, 7);

g.addEdge(2, 4, 2);

g.addEdge(2, 5, 4);

g.addEdge(3, 7, 9);

g.addEdge(3, 5, 4);

g.addEdge(4, 5, 3);

g.addEdge(5, 7, 10);

g.addEdge(6, 7, 1);

g.addEdge(6, 3, 6);

g.addEdge(7, 8, 7);

cout << "Edges of MST are \n";

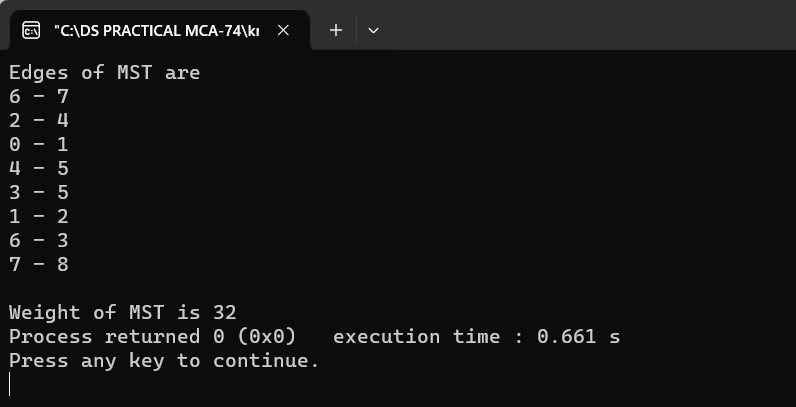
int mst\_wt = g.kruskalMST();

cout << "\nWeight of MST is " << mst\_wt;

return 0;

}

**OUTPUT:**



**PRACTICAL 20**

**Aim : Write a program to create a minimum spanning tree using**

**Prim’s Algorithm.**

**CODE:**

#include <bits/stdc++.h>

using namespace std;

#define V 5

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index; }

void printMST(int parent[], int graph[V][V])

{

cout << "Edge \tWeight\n";

for (int i = 1; i < V; i++)

cout << parent[i] << " - " << i << " \t"

<< graph[i][parent[i]] << " \n"; }

void primMST(int graph[V][V])

{

int parent[V];

int key[V];

bool mstSet[V];

for (int i = 0; i < V; i++)

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v] && mstSet[v] == false

&& graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v]; }

printMST(parent, graph); }

int main() {

int graph[V][V] = { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

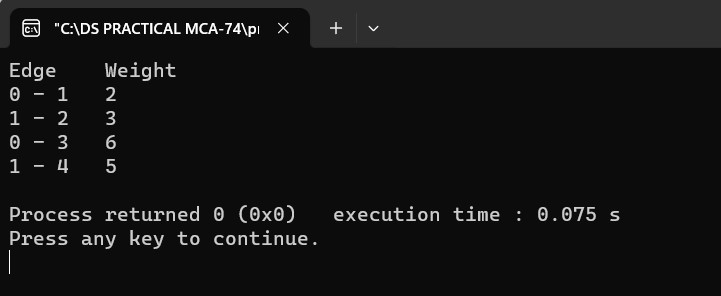
{ 0, 5, 7, 9, 0 } };

primMST(graph);

return 0;

}

**OUTPUT:**



**Practical exam extra questions**

**Q.1 write a program in c++ in code block to represent a doubly linked list and perform the following operations on it: 1. Insert 2. Display 3. Count**

**Code:-**

#include<iostream>

using namespace std;

// Node structure for doubly linked list

struct Node {

int data;

Node\* prev;

Node\* next;

};

// Class for doubly linked list operations

class DoublyLinkedList {

private:

Node\* head;

public:

// Constructor

DoublyLinkedList() {

head = nullptr;

}

// Function to insert a new node at the end of the list

void insert(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->prev = nullptr;

newNode->next = nullptr;

if (head == nullptr) {

// If the list is empty, make the new node as the head

head = newNode;

} else {

// Traverse to the end of the list

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

// Insert the new node at the end

temp->next = newNode;

newNode->prev = temp;

}

cout << "Node inserted successfully!\n";

}

// Function to display the elements of the list

void display() {

if (head == nullptr) {

cout << "List is empty\n";

} else {

Node\* temp = head;

cout << "Doubly Linked List: ";

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

}

// Function to count the number of nodes in the list

int count() {

int count = 0;

Node\* temp = head;

while (temp != nullptr) {

count++;

temp = temp->next;

}

return count;

}

};

int main() {

DoublyLinkedList myList;

int choice, value;

do {

// Display menu

cout << "\nDoubly Linked List Operations:\n";

cout << "1. Insert\n";

cout << "2. Display\n";

cout << "3. Count\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

// Insert operation

cout << "Enter the value to insert: ";

cin >> value;

myList.insert(value);

break;

case 2:

// Display operation

myList.display();

break;

case 3:

// Count operation

cout << "Number of nodes in the list: " << myList.count() << endl;

break;

case 0:

// Exit the program

cout << "Exiting the program. Goodbye!\n";

break;

default:

// Invalid choice

cout << "Invalid choice. Please try again.\n";

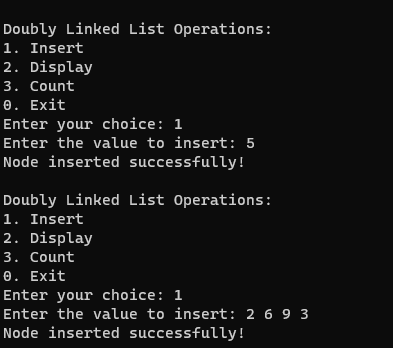
}

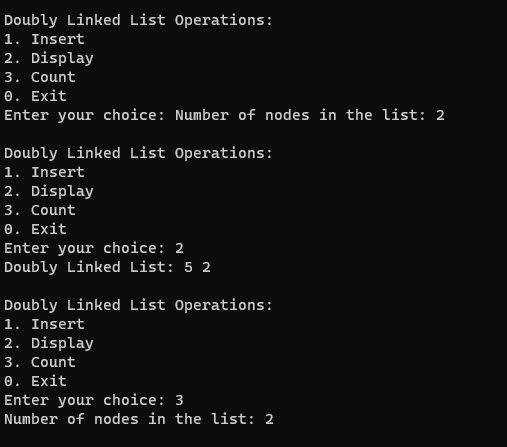
} while (choice != 0);

return 0;

}

Output:-





Q.2Write a program in C++ in a code block to create a double-ended queue and perform the following operation: 1. Enqueue front 2. Dequeue rear 3. Display

Code:-

#include<iostream>

#include<deque>

using namespace std;

int main() {

deque<int> myDeque; // Creating a double-ended queue

int choice, value;

do {

// Display menu

cout << "\nDouble-Ended Queue (Deque) Operations:\n";

cout << "1. Enqueue at Front\n";

cout << "2. Dequeue at Rear\n";

cout << "3. Display\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

// Enqueue at Front

cout << "Enter the value to enqueue at the front: ";

cin >> value;

myDeque.push\_front(value);

cout << "Element enqueued at the front successfully!\n";

break;

case 2:

// Dequeue at Rear

if (!myDeque.empty()) {

int dequeuedValue = myDeque.back();

myDeque.pop\_back();

cout << "Element dequeued from the rear: " << dequeuedValue << endl;

} else {

cout << "Deque is empty. Cannot dequeue from the rear.\n";

}

break;

case 3:

// Display

if (!myDeque.empty()) {

cout << "Double-Ended Queue (Deque): ";

for (const auto& element : myDeque) {

cout << element << " ";

}

cout << endl;

} else {

cout << "Deque is empty.\n";

}

break;

case 0:

// Exit the program

cout << "Exiting the program. Goodbye!\n";

break;

default:

// Invalid choice

cout << "Invalid choice. Please try again.\n";

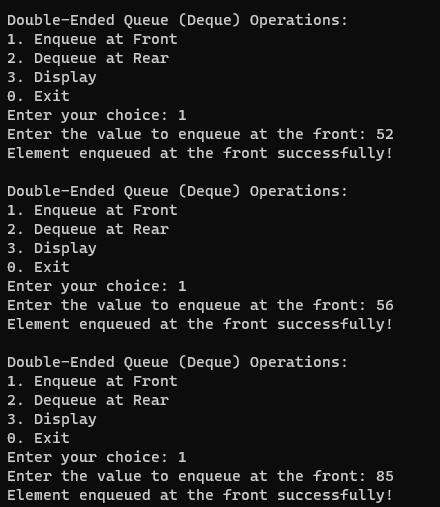
}

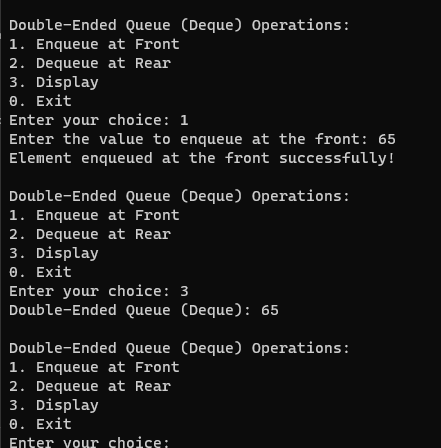
} while (choice != 0);

return 0;

}

O/p:-





Q. 3 Write a program in C++ in a code block to create a priority queue and perform the following operation: 1. Enqueue 2. Dequeue 3. Display

Code:-

#include<iostream>

#include<queue>

using namespace std;

int main() {

priority\_queue<int> myPriorityQueue;

int choice, value;

do {

// Display menu

cout << "\nPriority Queue Operations:\n";

cout << "1. Enqueue\n";

cout << "2. Dequeue\n";

cout << "3. Display\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

// Enqueue operation

cout << "Enter the value to enqueue: ";

cin >> value;

myPriorityQueue.push(value);

cout << "Element enqueued successfully!\n";

break;

case 2:

// Dequeue operation

if (!myPriorityQueue.empty()) {

int dequeuedValue = myPriorityQueue.top();

myPriorityQueue.pop();

cout << "Element dequeued: " << dequeuedValue << endl;

} else {

cout << "Priority Queue is empty. Cannot dequeue.\n";

}

break;

case 3:

// Display operation

if (!myPriorityQueue.empty()) {

cout << "Priority Queue elements (highest priority first): ";

while (!myPriorityQueue.empty()) {

cout << myPriorityQueue.top() << " ";

myPriorityQueue.pop();

}

cout << endl;

} else {

cout << "Priority Queue is empty.\n";

}

break;

case 0:

// Exit the program

cout << "Exiting the program. Goodbye!\n";

break;

default:

// Invalid choice

cout << "Invalid choice. Please try again.\n";

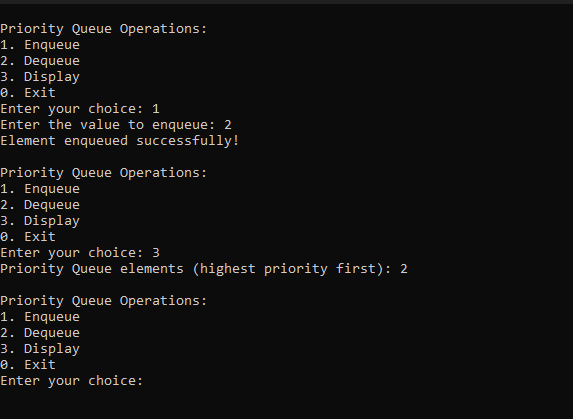
}

} while (choice != 0);

return 0;

}

O/p:-



Q.4 Write a program in C++ in a code block to create a Singly linked list perform the following operation: 1. Insert 2. Search 3. Display

Code:-

#include<iostream>

using namespace std;

// Node structure for singly linked list

struct Node {

int data;

Node\* next;

};

// Class for singly linked list operations

class SinglyLinkedList {

private:

Node\* head;

public:

// Constructor

SinglyLinkedList() {

head = nullptr;

}

// Function to insert a new node at the end of the list

void insert(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = nullptr;

if (head == nullptr) {

// If the list is empty, make the new node as the head

head = newNode;

} else {

// Traverse to the end of the list

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

// Insert the new node at the end

temp->next = newNode;

}

cout << "Node inserted successfully!\n";

}

// Function to search for a value in the list

bool search(int value) {

Node\* temp = head;

while (temp != nullptr) {

if (temp->data == value) {

return true; // Value found

}

temp = temp->next;

}

return false; // Value not found

}

// Function to display the elements of the list

void display() {

if (head == nullptr) {

cout << "List is empty\n";

} else {

Node\* temp = head;

cout << "Singly Linked List: ";

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

}

};

int main() {

SinglyLinkedList myList;

int choice, value;

do {

// Display menu

cout << "\nSingly Linked List Operations:\n";

cout << "1. Insert\n";

cout << "2. Search\n";

cout << "3. Display\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

// Insert operation

cout << "Enter the value to insert: ";

cin >> value;

myList.insert(value);

break;

case 2:

// Search operation

cout << "Enter the value to search: ";

cin >> value;

if (myList.search(value)) {

cout << "Value " << value << " found in the list.\n";

} else {

cout << "Value " << value << " not found in the list.\n";

}

break;

case 3:

// Display operation

myList.display();

break;

case 0:

// Exit the program

cout << "Exiting the program. Goodbye!\n";

break;

default:

// Invalid choice

cout << "Invalid choice. Please try again.\n";

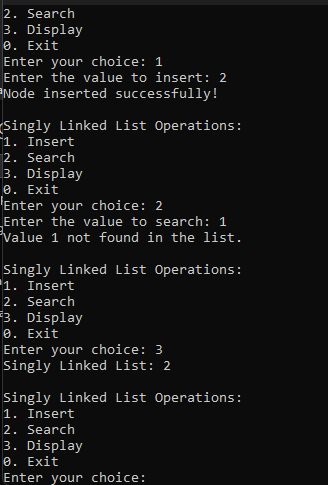
}

} while (choice != 0);

return 0;

}

Output:-



Q.5 Write a program in C++ in a code block to create a ordinary queue and perform the following operation: 1. Enqueue 2. Dequeue 3. Display

Code:-

#include<iostream>

#include<queue>

using namespace std;

int main() {

queue<int> myQueue;

int choice, value;

do {

// Display menu

cout << "\nQueue Operations:\n";

cout << "1. Enqueue\n";

cout << "2. Dequeue\n";

cout << "3. Display\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

// Enqueue operation

cout << "Enter the value to enqueue: ";

cin >> value;

myQueue.push(value);

cout << "Element enqueued successfully!\n";

break;

case 2:

// Dequeue operation

if (!myQueue.empty()) {

int dequeuedValue = myQueue.front();

myQueue.pop();

cout << "Element dequeued: " << dequeuedValue << endl;

} else {

cout << "Queue is empty. Cannot dequeue.\n";

}

break;

case 3:

// Display operation

if (!myQueue.empty()) {

cout << "Queue elements: ";

queue<int> tempQueue = myQueue;

while (!tempQueue.empty()) {

cout << tempQueue.front() << " ";

tempQueue.pop();

}

cout << endl;

} else {

cout << "Queue is empty.\n";

}

break;

case 0:

// Exit the program

cout << "Exiting the program. Goodbye!\n";

break;

default:

// Invalid choice

cout << "Invalid choice. Please try again.\n";

}

} while (choice != 0);

return 0;

}

Output:-

