**EXPERIMENT NO. 1**

**IMPLEMENTATION OF DIFFERENT SORTING TECHNIQUES.**

**Experiment No. 1**

**Implementation of different sorting techniques.**

**Q.1 Write program to sort a given number using selection sort.**

**Program :**

#include<stdio.h>

#include<conio.h>

#include<iostream.h>

int main()

{

int a[100],i,j,count,temp;

clrscr();

cout<<"Kartik Chopade Roll No:6\n";

cout<<"how many elements:\n";

cin>>count;

cout<<"enter element of array:\n";

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

{

cin>>a[i];

}

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

{

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

{

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

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

cout<<"the sorted array in ascending order: ";

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

{

cout<<" "<<a[i];

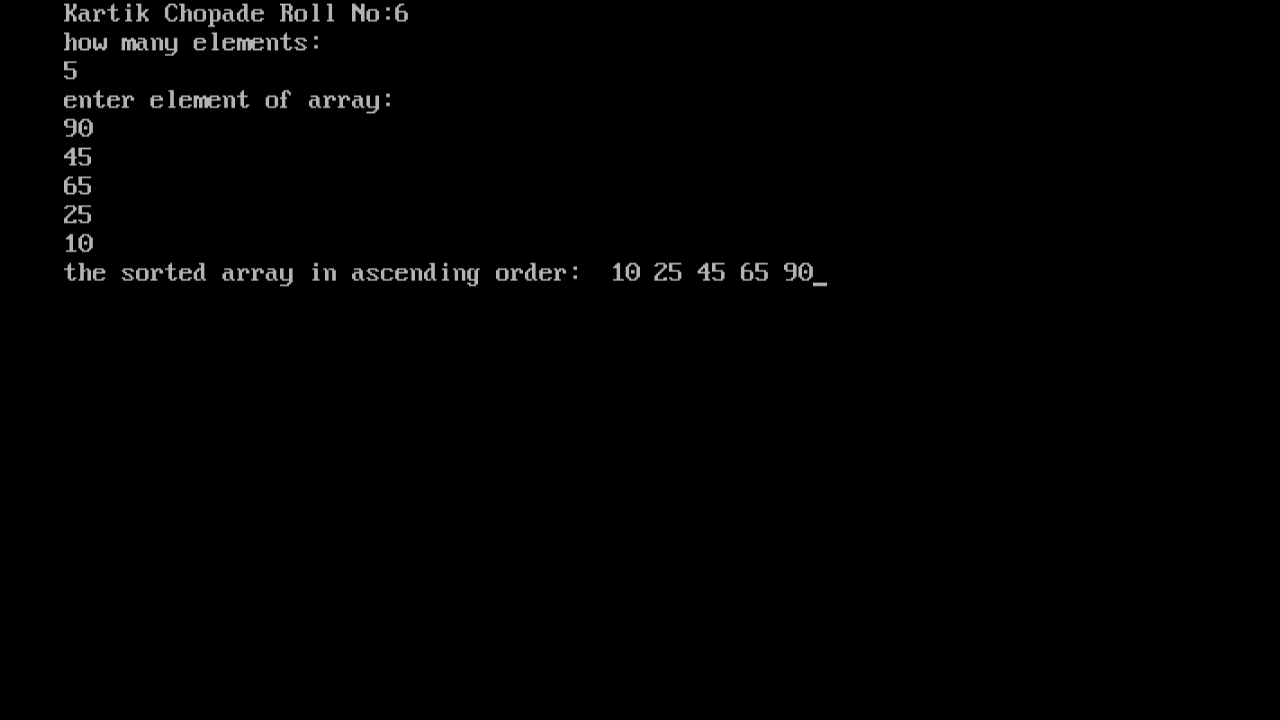
}

getch();

return 0;

}

**Output :**



**Q.2 Write program to sort a given number using bubble sort.**

**Program :**

#include<stdio.h>

#include<conio.h>

#include<iostream.h>

int main()

{

int a[100],i,j,n,temp;

clrscr();

cout<<"Kartik Chopade Roll No:6\n";

cout<<"how many elements:\n";

cin>>n;

cout<<"enter elements in array:\n";

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

{

cin>>a[i];

}

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;

}

}

}

cout<<"the sorted array is: ";

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

{

cout<<" "<<a[i];

}

getch();

return 0;

}

**Output :**



**Q.3 Write program to sort a given number using Insertion sort.**

**Program :**

#include<stdio.h>

#include<conio.h>

#include<iostream.h>

int main()

{

int i,n,j,k,temp,a[100];

clrscr();

cout<<"Kartik Chopade Roll No:6\n";

cout<<"how many elements:\n";

cin>>n;

cout<<"enter elements of array:\n";

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

{

cin>>a[i];

}

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

{

temp=a[k];

j=k-1;

while(j>=0 && temp<=a[j])

{

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

j=j-1;

}

a[j+1]=temp;

}

cout<<"the sorted array is: ";

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

{

cout<<" "<<a[i];

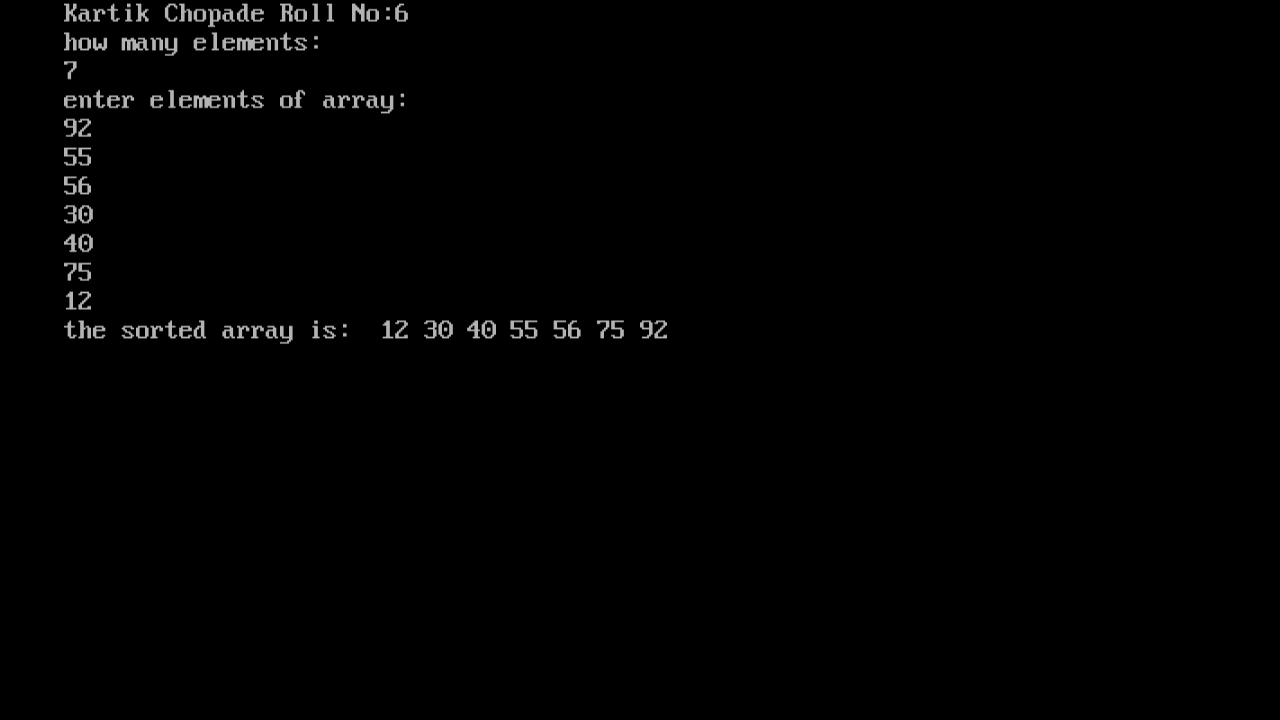
}

getch();

return 0;

}

**Output :**



**Q.4 Write program to sort a given number using Shell sort.**

**Program :**

#include<stdio.h>

#include<conio.h>

#include<iostream.h>

int main()

{

int a[100],i,j,n,gap,temp;

clrscr();

cout<<"Kartik Chopade Roll No:6\n";

cout<<"how many elements:\n";

cin>>n;

cout<<"enter elements in array:\n";

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

{

cin>>a[i];

}

for(gap=n/2;gap>0;gap=gap/2)

{

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

{

temp=a[i];

for(j=i;j>=gap && a[j-gap] > temp;j=j-gap)

a[j]=a[j-gap];

a[j]=temp;

}

}

cout<<"the sorted array is: ";

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

{

cout<<" "<<a[i];

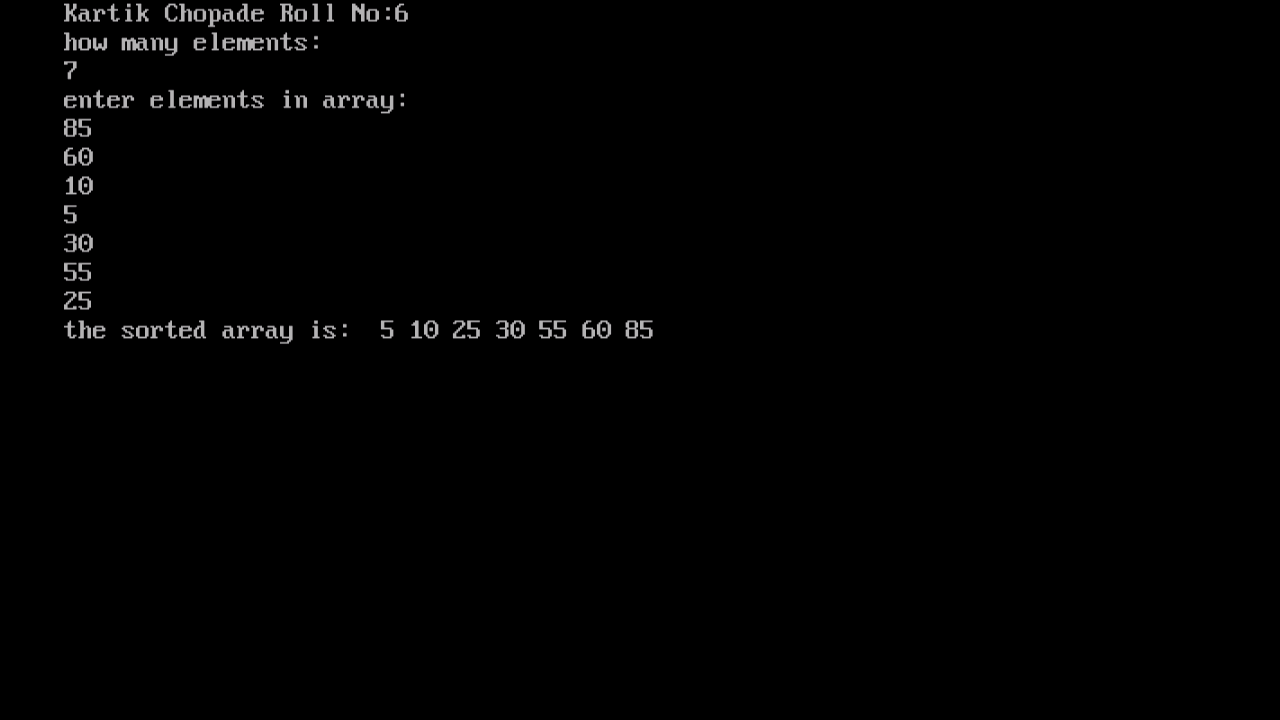
}

getch();

return 0;

}

**Output :**



**EXPERIMENT NO: 2**

**IMPLEMENTATION OF SEARCHING ALGORITHMS.**

**EXPERIMENT NO: 2**

**Implementation of searching algorithms.**

**Q.1 Write the C++ program for linear search.**

**Program :**

#include<stdio.h>

#include<conio.h>

#include<iostream.h>

int main()

{

int a[20],n,x,i,flag=0;

clrscr();

cout<<"Kartik Chopade Roll No:6\n";

cout<<"how many elements:\n";

cin>>n;

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

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

cin>>a[i];

cout<<"Enter element to search:";

cin>>x;

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

{

if(a[i]==x)

{

flag=1;

break;

}

}

if(flag)

{

cout<<"\nElement is found at position "<<i+1;

}

else

{

cout<<"\nElement not found";

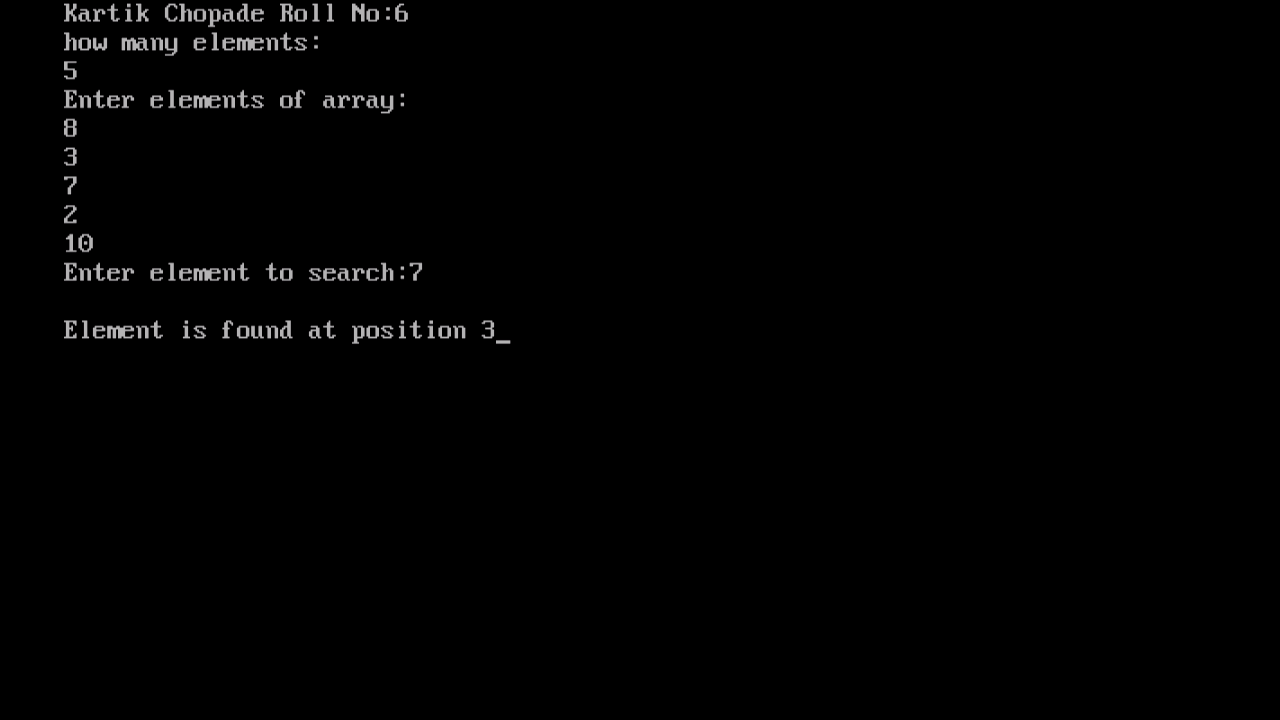
}

getch();

return 0;

}

**Output :**



**Q.2 Write the C++ program for binary search.**

**Program :**

#include<stdio.h>

#include<conio.h>

#include<iostream.h>

int main()

{

int c,first,last,middle,n,search,array[100];

clrscr();

cout<<"Kartik Chopade Roll No:6\n";

cout<<"Enter number of elements\n";

cin>>n;

cout<<"Enter " <<n <<" integers\n";

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

{

cin>>array[c];

}

cout<<"Enter value to find\n";

cin>>search;

first=0;

last=n-1;

middle=(first+last)/2;

while(first<=last)

{

if(array[middle] < search)

{

first=middle+1;

}

else if(array[middle]==search)

{

cout<<search<<" found at location "<<middle+1;

break;

}

else

{

last=middle-1;

}

middle=(first+last)/2;

}

if(first>last)

{

cout<<"Not Found "<<search<<" isen't present in the list\n";

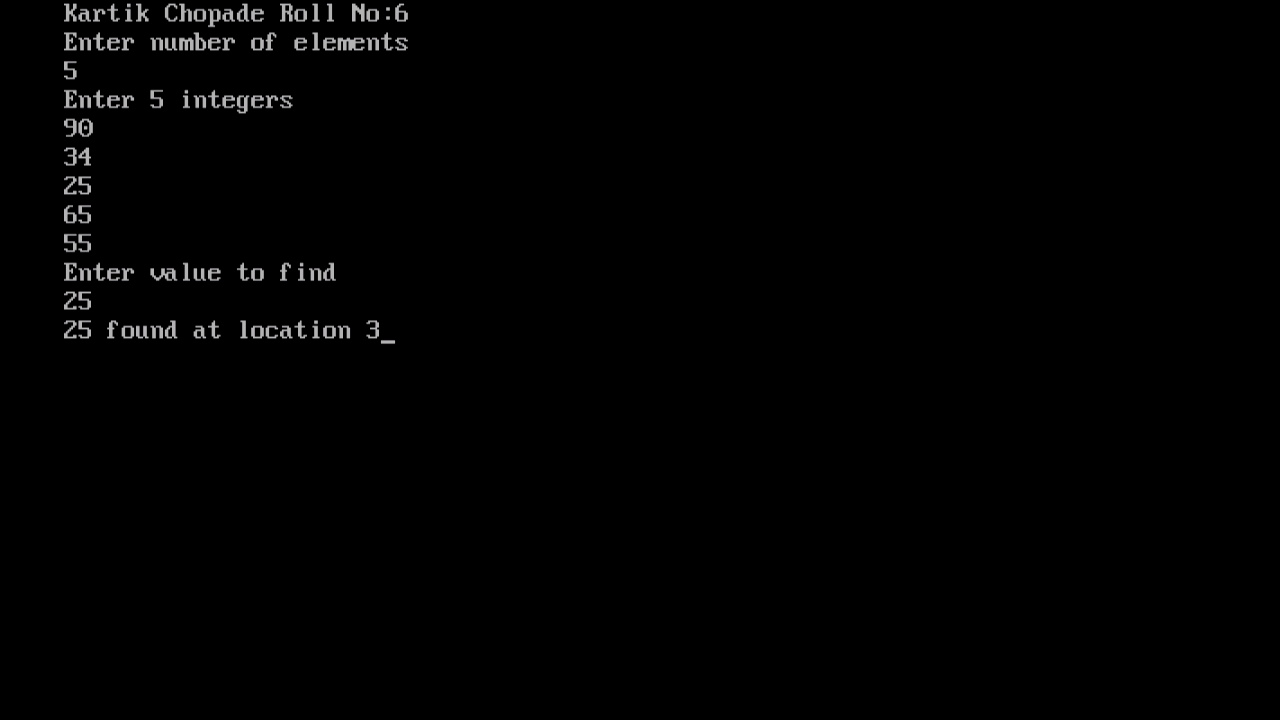
}

getch();

return 0;

}

**Output :**



**EXPERIMENT NO: 3**

**IMPLEMENTATION OF STACKS**

**(USING ARRAYS AND LINKED LIST).**

**EXPERIMENT NO: 3**

**Implementation of stacks(Using arrays and Linked List).**

**A) Stacks using array**

**Program :**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

struct node

{

int info;

struct node \*next;

}\*start, \*temp,\*current;

void circular(int);

void circulardisplay();

int main()

{

start=NULL;

int item,choice,location,element,position;

cout << "Kartik chopade Roll no 6\n";

again:

cout<<"\n1.Insert Item in Circular Linked List\n2.Display circular Linked List \n3.Exit\n";

cout<<"Enter choice : ";

cin>>choice;

switch(choice)

{

case 1:cout<<"Enter item to insert: ";

cin>>item;

circular(item);

goto again;

case 2:cout<<"\nInserted item = ";

circulardisplay();cout<<"\n";

goto again;

case 3:cout<<"\nTHANK YOU";

default:break;

}

}

void circular(int item)

{

temp=(node\*)malloc(sizeof(node));

temp->info=item;

temp->next=start;

node\* current;

if(start==NULL)

{

start=temp;

temp->next=start;

}

else

{

current=start;

while(current->next!=start)

{

current=current->next;

}

current->next=temp;

}

}

void circulardisplay()

{

current= start;

do

{

cout<<current->info<<" ";

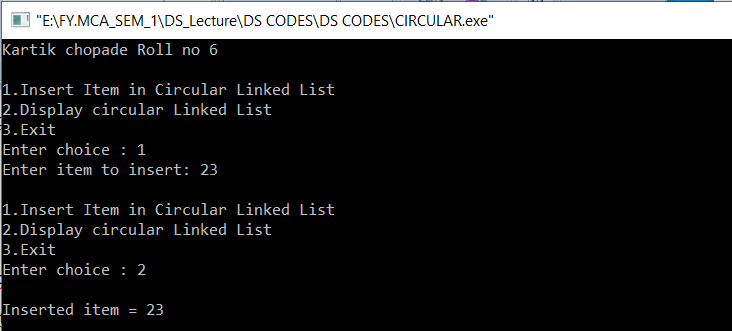
current=current->next;

}

while(current!=start);

}

**Output:**



**B) Stacks Linked List**

**Program :**

#include <iostream>

using namespace std;

class node

{

public:

int data;

node \*next;

};

class stack

{

public:

node \*top;

};

void display(stack s)

{

node \*p;

p=s.top;

while(p!=NULL)

{

cout<<p->data<<endl;

p=p->next;

}

}

int push(stack \*s, int n)

{

node \*p;

p=new node;

p->data=n;

p->next=s->top;

s->top=p;

}

int pop(stack \*s)

{

int n;

node \*p;

if(s->top==NULL)

{

cout<<"Stack is empty"<<endl;

return -1;

}

else

{

p=s->top;

n=p->data;

s->top=p->next;

delete p;

return n;

}

}

int main()

{

stack s;

int n, c;

s.top=NULL;

do

{

cout << "Kartik chopade Roll no: 6 \n ";

cout<<"1 - PUSH\n";

cout<<"2 - POP\n";

cout<<"3 - DISPLAY\n";

cout<<"4 - EXIT\n";

cout<<"Enter your choice: ";

cin>>c;

switch(c){

case 1:

cout<<"Enter element to be pushed\n";

cin>>n;

push(&s,n);

break;

case 2:

n=pop(&s);

if(n!=-1){

cout<<n<<" has popped out\n";

}

break;

case 3:

display(s);

break;

}

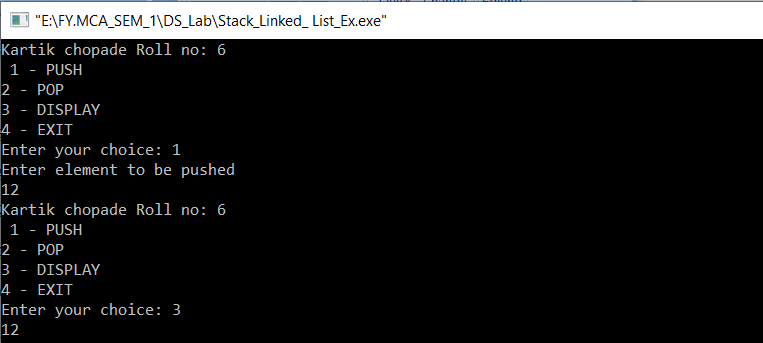
}

while(c!=4);

return 0;

}

**Output:**



**EXPERIMENT NO: 4**

**IMPLEMENTATION OF STACK APPLICATIONS LIKE:**

**A) POSTFIX EVALUATION**

**B) BALANCING OF PARENTHESIS**

**EXPERIMENT NO: 4**

**IMPLEMENTATION OF STACK APPLICATIONS LIKE:**

**A) POSTFIX EVALUATION**

**B) BALANCING OF PARENTHESIS**

**A) Postfix Evaluation**

**Program:**

#include<iostream>

#include<string.h>

#include<math.h>

using namespace std;

class postfix

{

public:

int top;

char p[50];

long int A[50];

postfix();

void input();

void push(long int s);

long int pop();

int full();

int empty();

long int eval\_post();

};

postfix::postfix()

{

top=-1;

}

void postfix::input()

{

cout << "Kartik chopade Roll no: 6 \n ";

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;

}

int main()

{

long int value;

postfix f;

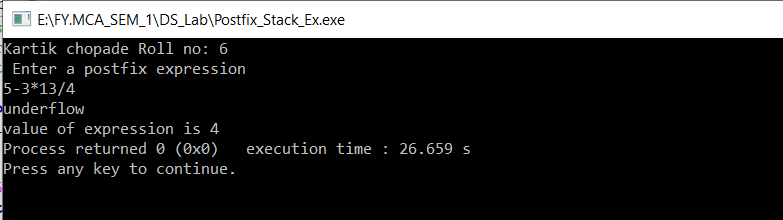
f.input();

value=f.eval\_post();

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

}

**Output:**



**B) Balancing of parenthesis**

**Program :**

#include <iostream>

#include<string.h>

using namespace std;

class paren

{

public:

int top;

char A[50];

paren();

void push(char item);

char pop();

int full();

int empty();

};

paren::paren()

{

top=-1;

}

int paren::full()

{

if(top==49)

{

return 1;

}

else

{

return 0;

}

}

void paren::push(char item)

{

if(full())

{

cout<<"Overfull";

}

else

{

top=top+1;

A[top]=item;

}

}

int paren::empty()

{

if(top==-1)

{

return 1;

}

else

{

return 0;

}

}

char paren::pop()

{

if(empty())

{

cout<<"Underflow\n";

}

else

{

return A[top--];

}

}

int main()

{

paren p;

char exp[50],temp;

int i,valid=1;

cout << "Kartik chopade Roll no: 6 \n ";

cout<<"Enter Expression\n";

cin>>exp;

for(i=0;i<strlen(exp);i++)

{

if(exp[i]=='(')

{

p.push(exp[i]);

}

if(exp[i]==')')

{

if(p.top==-1)

{

valid=0;

}

else

{

temp=p.pop();

}

}

if(exp[i]==')' && temp=='(')

{

valid=1;

}

}

if(p.top>=0)

{

valid=0;

}

if(valid==1)

{

cout<<"Valid Expression\n";

}

else

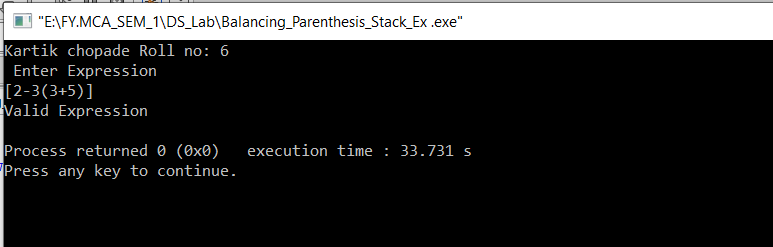
{

cout<<"Invalid Expression\n";

}

}

**Output :**



**EXPERIMENT NO: 5**

**IMPLEMENT ALL DIFFERENT TYPES OF QUEUES.**

**EXPERIMENT NO: 5**

**IMPLEMENT ALL DIFFERENT TYPES OF QUEUES.**

**A) Circular Queue**

**Program :**

#include <iostream>

using namespace std;

int cqueue[5];

int front = -1, rear = -1, n=5;

void insertCQ(int val)

{

if ((front == 0 && rear == n-1) || (front == rear+1))

{

cout<<"Queue Overflow \n";

return;

}

if (front == -1)

{

front = 0;

rear = 0;

}

else

{

if (rear == n - 1)

rear = 0;

else

rear = rear + 1;

}

cqueue[rear] = val ;

}

void deleteCQ()

{

if (front == -1)

{

cout<<"Queue Underflow\n";

return ;

}

cout<<"Element deleted from queue is : "<<cqueue[front]<<endl;

if (front == rear)

{

front = -1;

rear = -1;

}

else

{

if (front == n - 1)

front = 0;

else

front = front + 1;

}

}

void displayCQ()

{

int f = front, r = rear;

if (front == -1)

{

cout<<"Queue is empty"<<endl;

return;

}

cout<<"Queue elements are :\n";

if (f <= r)

{

while (f <= r)

{

cout<<cqueue[f]<<" ";

f++;

}

}

else

{

while (f <= n - 1)

{

cout<<cqueue[f]<<" ";

f++;

}

f = 0;

while (f <= r)

{

cout<<cqueue[f]<<" ";

f++;

}

}

cout<<endl;

}

int main()

{

cout << "Kartik chopade Roll no: 6 \n ";

int ch, val;

cout<<"1)Insert\n";

cout<<"2)Delete\n";

cout<<"3)Display\n";

cout<<"4)Exit\n";

do

{

cout<<"Enter choice : "<<endl;

cin>>ch;

switch(ch)

{

case 1:

cout<<"Input for insertion: "<<endl;

cin>>val;

insertCQ(val);

break;

case 2:

deleteCQ();

break;

case 3:

displayCQ();

break;

case 4:

cout<<"Exit\n";

break;

default: cout<<"Incorrect!\n";

}

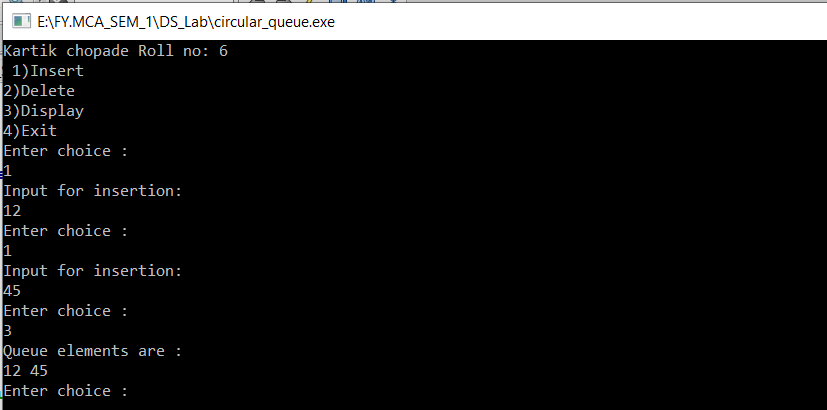
}

while(ch != 4);

return 0;

}

**Output:**



**B) Double Ended Queue**

**Program :**

#include<iostream>

using namespace std;

/\*declaring a structure to create a node\*/

struct node

{

int data;

struct node \*prev, \*next;

};

struct node \*head = NULL, \*tail = NULL;

struct node \* createNode(int data)

{

/\*allocating implicit memory to the node\*/

struct node \*newnode = (struct node \*)malloc(sizeof (struct node));

newnode->data = data;

newnode->next = newnode->prev = NULL;

return (newnode);

}

/\* create sentinel(dummy head & tail) that helps us to do insertion and deletion operation at front and rear so easily. And these dummy head and tail wont get deleted till the end of execution of this program \*/

void createSentinels() /\*creating a head and tail\*/

{

head = createNode(0);

tail = createNode(0);

head->next = tail;

tail->prev = head;

}

/\* insertion at the front of the queue \*/

void enqueueAtFront(int data)

{

struct node \*newnode, \*temp;

newnode = createNode(data);

temp = head->next;

head->next = newnode;

newnode->prev = head;

newnode->next = temp;

temp->prev = newnode;

}

/\*insertion at the rear of the queue \*/

void enqueueAtRear(int data)

{

struct node \*newnode, \*temp;

newnode = createNode(data);

temp = tail->prev;

tail->prev = newnode;

newnode->next = tail;

newnode->prev = temp;

temp->next = newnode;

}

/\* deletion at the front of the queue \*/

void dequeueAtFront()

{

struct node \*temp;

if (head->next == tail)

{

cout << "Queue is empty\n";

}

else

{

temp = head->next;

head->next = temp->next;

temp->next->prev = head;

free(temp);

}

return;

}

/\* deletion at the rear of the queue \*/

void dequeueAtRear()

{

struct node \*temp;

if (tail->prev == head)

{

cout << "Queue is empty\n";

}

else

{

temp = tail->prev;

tail->prev = temp->prev;

temp->prev->next = tail;

free(temp);

}

return;

}

/\* display elements present in the queue \*/

void display()

{

struct node \*temp;

if (head->next == tail)

{

cout << "Queue is empty\n";

return;

}

temp = head->next;

while (temp != tail)

{

cout << temp->data;

temp = temp->next;

}

cout << "\n";

}

/\*main program\*/

int main()

{

int data, ch;

cout << "Kartik chopade Roll no: 6 \n ";

createSentinels();

while (1)

{

cout<<"1. Enqueue at front\n2. Enqueue at rear\n";

cout<<"3. Dequeue at front\n4. Dequeue at rear\n";

cout<<"5. Display\n6. Exit\n";

cout<<"Enter your choice:";

cin>>ch;

switch (ch) /\*switch case\*/

{

case 1:

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

cin>>data;

enqueueAtFront(data);

break;

case 2:

cout<<"Enter ur data to insert:";

cin>>data;

enqueueAtRear(data);

break;

case 3:

dequeueAtFront();

break;

case 4:

dequeueAtRear();

break;

case 5:

display();

break;

case 6:

exit(0);

default:

cout<<" enter correct option\n";

break;

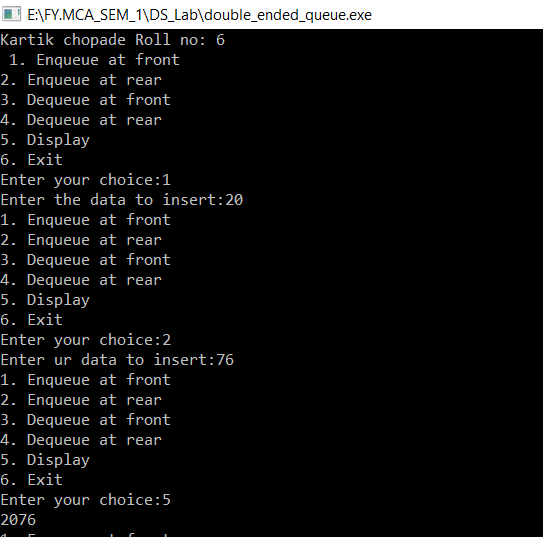
} /\*end of switch case\*/

}

return 0;

}

**Output :**



**EXPERIMENT NO: 6**

**DEMONSTRATE APPLICATION OF QUEUE**

**(EG. PRIORITY QUEUE, BREATH FIRST SEARCH).**

**EXPERIMENT NO: 6**

**DEMONSTRATE APPLICATION OF QUEUE**

**(EG. PRIORITY QUEUE, BREATH FIRST SEARCH).**

**A) Priority Queue**

**Program:**

#include <iostream>

using namespace std;

class priority\_queue

{

public:

struct node

{

int data,pri ;

node \*next;

}; node \*front1,\*q;

public:

priority\_queue33()

{

front1=NULL;

}

void insert1(int x,int pri)

{

node \*temp=new node;

temp->data=x;

temp->pri=pri;

if(front1==NULL || pri <front1->pri)

{

temp->next=front1;

front1=temp;

}

else

{

q=front1;

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

{

q=q->next;

}

temp->next=q->next;

q->next=temp;

}

}

void delete1()

{

node \*temp=new node;

if(front1==NULL)

{

cout<<"\nPriority Queue is empty you cannot perform delete operation on empty queue";

}

else

{

temp=front1;

front1=front1->next;

}

}

void display()

{

node \*temp=new node;

temp=front1;

if(front1==NULL)

{

cout<<"\nQueue is empty";

}

else

{

while(temp!=NULL)

{

cout<<"\n Element: "<<temp->data<<endl;

temp=temp->next;

}

}

}

};

int main()

{

priority\_queue q;

int ch, x,p;

cout << "Kartik chopade Roll no: 6\n";

while(1)

{

cout<<"\n1 - Insert";

cout<<"\n2 - Delete";

cout<<"\n3 - Display";

cout<<"\n4 - Exit";

cout<<"\nEnter your choice: ";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\nEnter element:";

cin>>x;

cout<<"\nEnter priority:";

cin>>p;

q.insert1(x,p);

break;

case 2:

q.delete1();

break;

case 3:

q.display();

break;

case 4:

return 0;

default:

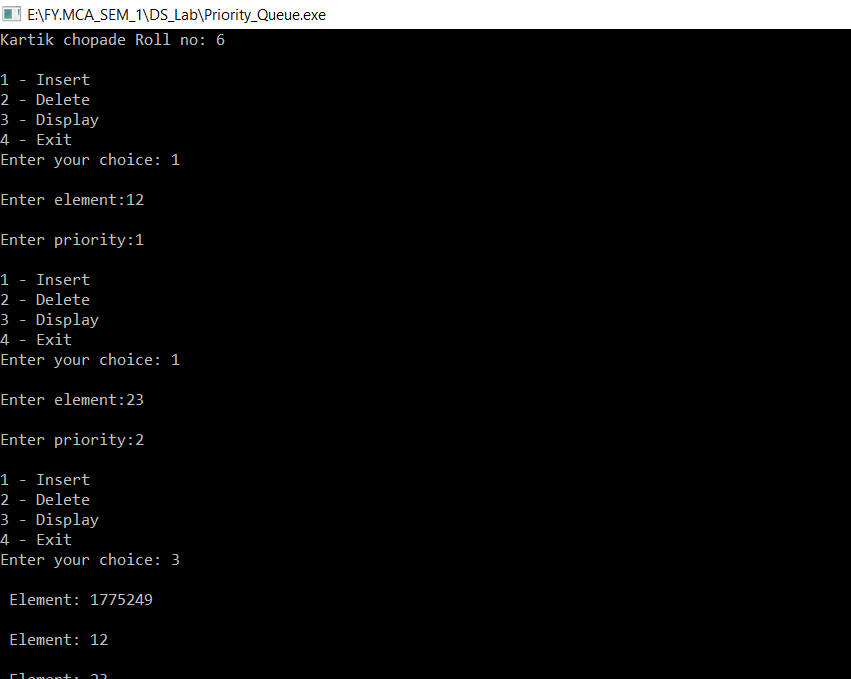
cout<<"wrong choice";

}

}

}

**Output :**



**B) Breath First Search**

**Program :**

#include<iostream>

#include <list>

using namespace std;

class Graph\_BFS

{

int V; // No. of vertices

// Pointer to an array containing adjacency

// lists

list<int> \*adj;

public:

Graph\_BFS(int V); // Constructor

// function to add an edge to graph

void addEdge(int v, int w);

// prints BFS traversal from a given source s

void BFS(int s);

};

Graph\_BFS::Graph\_BFS(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph\_BFS::addEdge(int v, int w)

{

adj[v].push\_back(w); // Add w to v’s list.

}

void Graph\_BFS::BFS(int s)

{

// Mark all the vertices as not visited

bool \*visited = new bool[V];

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

visited[i] = false;

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.push\_back(s);

// 'i' will be used to get all adjacent

// vertices of a vertex

list<int>::iterator i;

while(!queue.empty())

{

// Dequeue a vertex from queue and print it

s = queue.front();

cout << s << " ";

queue.pop\_front();

// Get all adjacent vertices of the dequeued

// vertex s. If a adjacent has not been visited,

// then mark it visited and enqueue it

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

int main()

{ cout<<"Kartik chopade Roll no: 6\n";

// Create a graph given in the above diagram

Graph\_BFS g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

cout << "Following is Breadth First Traversal "

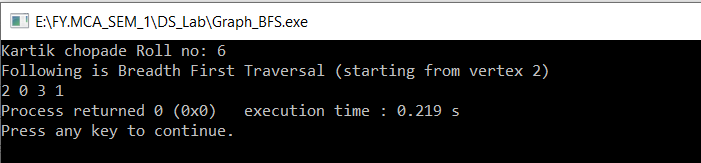
<< "(starting from vertex 2) \n";

g.BFS(2);

return 0;

}

**Output :**



**EXPERIMENT NO: 7**

**IMPLEMENTATION OF ALL TYPES OF LINKED LISTS.**

**EXPERIMENT NO: 7**

**IMPLEMENTATION OF ALL TYPES OF LINKED LISTS.**

**A) Single LinkedList**

**Program :**

#include<iostream>

using namespace std;

struct node

{

int data;

node \*next;

} \*last;

class list

{

public:

list()

{

last==NULL;

}

void insert\_begin(int x)

{

node \*temp=new node;

temp->data=x;

if(last==NULL)

{

last=temp;

temp->next=last;

}

else

{

temp->next=last->next;

last->next=temp;

}

}

void insert\_end(int x)

{

node \*temp=new node;

temp->data=x;

temp->next=last->next;

last->next=temp;

last=temp;

}

after(int x, int pos)

{

node \*temp=new node;

node \*q=last->next;

temp->data=x;

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

{

q=q->next;

}

if(q==last->next)

{

cout<<"less node present";

}

temp->next=q->next;

q->next=temp;

if(q==last)

{

last=temp;

}

}

void display()

{

struct node \*q;

if(last==NULL)

{

cout<<"List is empty"<<endl;

return;

}

q=last->next;

cout<<"Circular link list:"<<endl;

while(q!=last)

{

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

q=q->next;

}

cout<<q->data<<endl;

}

};

int main()

{

cout<<"Kartik chopade Roll no: 6";

list s;

int ch,x;

while(1)

{

cout<<"\n1.Insert AT Begin \n2.Insert At End\n3.Display\n4.Exit\n";

cout<<"Enter choice : "<<endl;

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter data :";

cin>>x;

s.insert\_begin(x);

break;

case 2:

cout<<"Enter data :";

cin>>x;

s.insert\_end(x);

break;

case 3:

s.display();

break;

case 4:

return 0;

default:

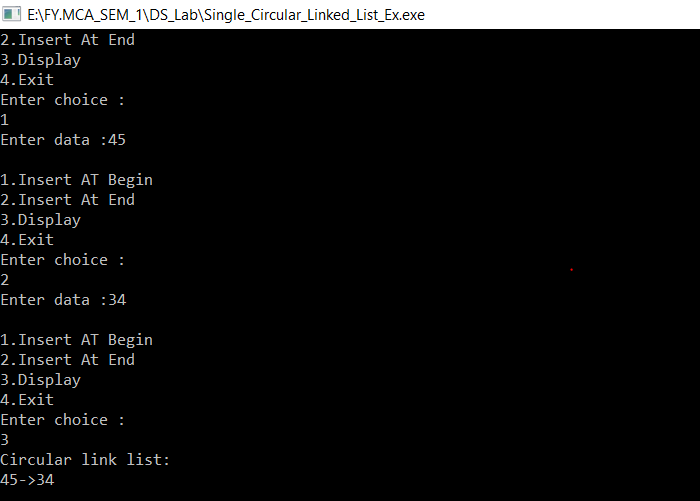
cout<<"Wrong choice";

}

}

}

**Output:**



**B) Circular LinkedList**

**Program:**

#include<iostream>

using namespace std;

struct node

{

int data;

node \*next;

} \*last;

class list

{

public:

list()

{

last==NULL;

}

void insert\_begin(int x)

{

node \*temp=new node;

temp->data=x;

if(last==NULL)

{

last=temp;

temp->next=last;

}

else

{

temp->next=last->next;

last->next=temp;

}

}

void insert\_end(int x)

{

node \*temp=new node;

temp->data=x;

temp->next=last->next;

last->next=temp;

last=temp;

}

after(int x, int pos)

{

node \*temp=new node;

node \*q=last->next;

temp->data=x;

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

{

q=q->next;

}

if(q==last->next)

{

cout<<"less node present";

}

temp->next=q->next;

q->next=temp;

if(q==last)

{

last=temp;

}

}

void display()

{

struct node \*q;

if(last==NULL)

{

cout<<"List is empty"<<endl;

return;

}

q=last->next;

cout<<"Circular link list:"<<endl;

while(q!=last)

{

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

q=q->next;

}

cout<<q->data<<endl;

}

};

int main()

{

cout<<"Kartik chopade Roll no: 6";

list s;

int ch,x;

while(1)

{

cout<<"\n1.Insert AT Begin \n2.Insert At End\n3.Display\n4.Exit\n";

cout<<"Enter choice : "<<endl;

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter data :";

cin>>x;

s.insert\_begin(x);

break;

case 2:

cout<<"Enter data :";

cin>>x;

s.insert\_end(x);

break;

case 3:

s.display();

break;

case 4:

return 0;

default:

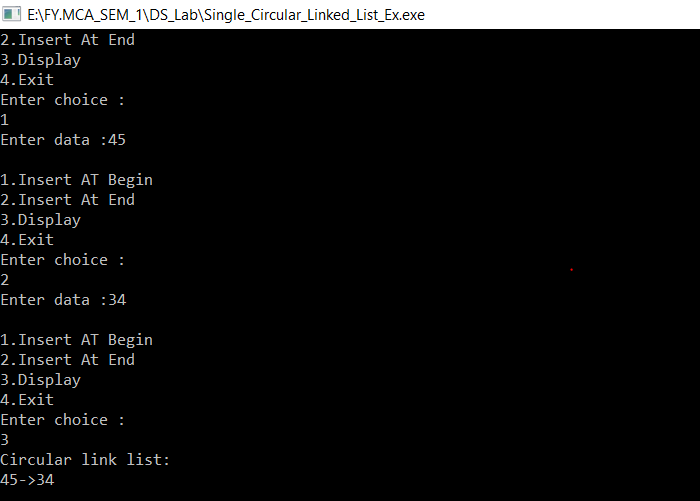
cout<<"Wrong choice";

}

}

}

**Output :**



**C) Doubly LinkedList**

**Program :**

#include <iostream.h>

#include<conio.h>

using namespace std;

class dubblylinkedlist

{

public:

struct node

{

int data;

node \*next,\*prev;

}; node \*start ,\*q ;

dubblylinkedlist()

{

q=NULL;

start =NULL;

}

void insert\_begin(int x)

{

node \*temp=new node;

temp->data=x;

if(start==NULL)

{

temp->next=NULL;

temp->prev=NULL;

start=temp;

}

else

{

temp->next=start;

start->prev=temp;

temp->prev=NULL;

start=temp;

}

}

void insert\_after(int x, int pos)

{

q=start;

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

{

q=q->next;

if(q==NULL)

{

cout<<"\n Less no. of nodes than position exist";

}

}

node \*temp=new node;

temp->data=x;

if(q->next==NULL)

{

q->next=temp;

temp->prev=q;

temp->next=NULL;

}

else

{

temp->next=q->next;

temp->next->prev=temp;

q->next=temp;

temp->prev=q;

}

}

void deletes(int pos)

{ int count=0;

if(pos==1)

{

start=start->next;

}

else if(pos>1 && pos<=count)

{

node \*ptr;

q=start;

while(q!=NULL)

{

q=q->next;

count++;

}

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

{

ptr=q;

q=q->next;

}

ptr->next=q->next;

}

}

void display()

{

node \*temp=new node;

temp=start;

while(temp!=NULL)

{

cout<<temp->data;

cout<<"\n ";

temp=temp->next;

}

if(start==NULL)

{

cout<<"\nDoubly Linked list is empty";

}

}

};

int main()

{

dubblylinkedlist q;

int ch, x,p;

cout << "Kartik chopade Roll no: 6\n";

while(1)

{

cout<<"\n1 - Insert at begining";

cout<<"\n2 - Insert after ";

cout<<"\n3 - Delete";

cout<<"\n4 - DISPLAY";

cout<<"\n5 - EXIT";

cout<<"\nEnter your choice: ";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter data: ";

cin>>x;

q.insert\_begin(x);

break;

case 2:

cout<<"\nEnter Element: ";

cin>>x;

cout<<"\nEnter Position: ";

cin>>p;

q.insert\_after(x,p);

break;

case 3:

cout<<"\nEnter Position: ";

cin>>p;

q.deletes(p);

break;

case 4:

cout<<"Display element : \n ";

q.display();

break;

case 5:

return 0;

default:

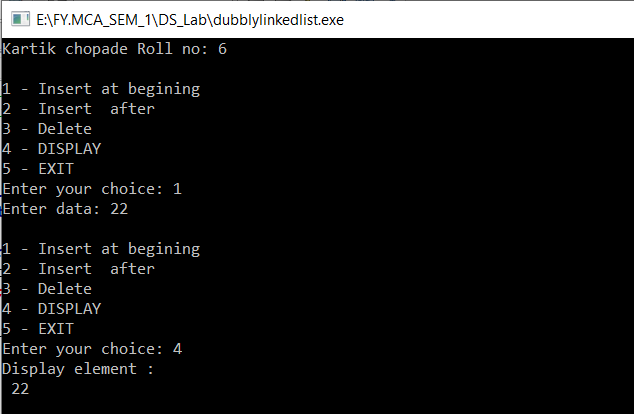
cout<<"wrong choice \n";

}

}

}

**Output :**



**EXPERIMENT NO: 8**

**DEMONSTRATE APPLICATION OF LINKED LIST**

**(EG. POLYNOMIAL ADDITION, SPARSE MATRIX).**

**EXPERIMENT NO: 8**

**DEMONSTRATE APPLICATION OF LINKEDLIST**

**(EG. POLYNOMIAL ADDITION, SPARSE MATRIX).**

**A) Polynomial Addition**

**Program:**

#include <iostream>

#include <iomanip>

using namespace std;

class poly{

public:

int coeff;

int pow\_val;

poly\* next;

};

class add {

poly \*poly1, \*poly2, \*poly3;

public:

add() { poly1 = poly2 = poly3 = NULL; }

void addpoly();

void display();

};

void add::addpoly()

{

int i, p;

poly \*newl = NULL, \*end = NULL;

cout << "Enter highest power for x\n"; cin >> p;

//Read first poly

cout << "\nFirst Polynomial\n";

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

{

newl = new poly;

newl->pow\_val = p;

cout << "Enter Co-efficient for degree" << i << ":: "; cin >> newl->coeff;

newl->next = NULL;

if (poly1 == NULL)

poly1 = newl;

else

end->next = newl;

end = newl;

}

//Read Second poly

cout << "\n\nSecond Polynomial\n"; end = NULL;

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

{

newl = new poly;

newl->pow\_val = p;

cout << "Enter Co-efficient for degree" << i << ":: "; cin >> newl->coeff;

newl->next = NULL;

if (poly2 == NULL)

poly2 = newl;

else

end->next = newl;

end = newl;

}

//Addition Logic

poly \*p1 = poly1, \*p2 = poly2;

end = NULL;

while (p1 != NULL && p2 != NULL) {

if (p1->pow\_val == p2->pow\_val) {

newl = new poly;

newl->pow\_val = p--;

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

newl->next = NULL;

if (poly3 == NULL)

poly3 = newl;

else

end->next = newl;

end = newl;

}

p1 = p1->next;

p2 = p2->next;

}

}

void add::display()

{

poly\* t = poly3;

cout << "\n\nAnswer after addition is : ";

while (t != NULL) {

cout.setf(ios::showpos);

cout << t->coeff;

cout.unsetf(ios::showpos);

cout << "X" << t->pow\_val;

t = t->next;

}

}

int main()

{ cout << "Kartik chopade Roll no: 6\n";

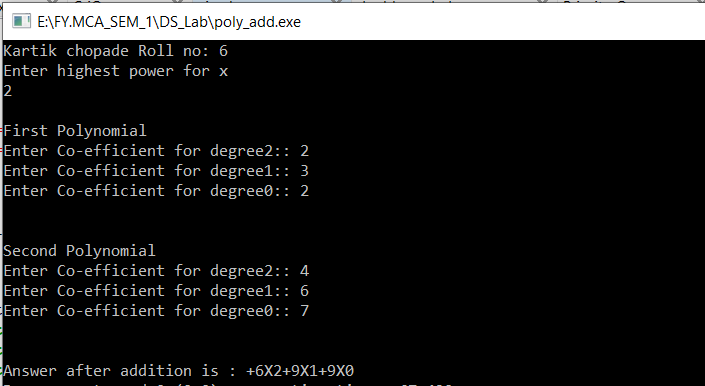
add obj;

obj.addpoly();

obj.display();

}

**Output:**



**B) Sparse Matrix**

**Program:**

#include<iostream>

using namespace std;

int main () {

cout<<"Kartik chopade Roll no: 6\n";

int a[10][10] = { {0, 0, 9} , {5, 0, 8} , {7, 0, 0} };

int i, j, count = 0;

int row = 3, col = 3;

for (i = 0; i < row; ++i) {

for (j = 0; j < col; ++j){

if (a[i][j] == 0)

count++;

}

}

cout<<"The matrix is:"<<endl;

for (i = 0; i < row; ++i) {

for (j = 0; j < col; ++j) {

cout<<a[i][j]<<" ";

}

cout<<endl;

}

cout<<"The number of zeros in the matrix are "<< count <<endl;

if (count > ((row \* col)/ 2))

cout<<"This is a sparse matrix"<<endl;

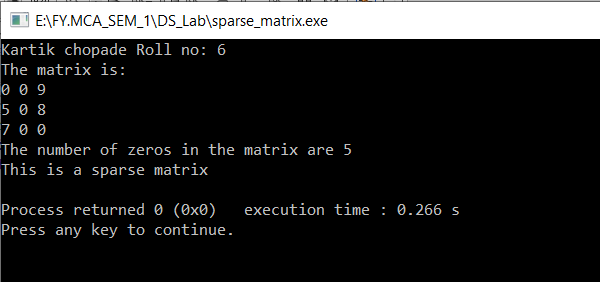
else

cout<<"This is not a sparse matrix"<<endl;

return 0;

}

**Output :**



**EXPERIMENT NO: 9**

**CREATE AND PERFORM VARIOUS OPERATIONS ON BST.**

**EXPERIMENT NO: 9**

**CREATE AND PERFORM VARIOUS OPERATIONS ON BST.**

**Program:**

#include <iostream>

using namespace std;

class BST

{

int data;

BST \*left, \*right;

public:

// Default constructor.

BST();

// Parameterized constructor.

BST(int);

// Insert function.

BST\* Insert(BST\*, int);

// Inorder traversal.

void Inorder(BST\*);

};

// Default Constructor definition.

BST ::BST()

: data(0)

, left(NULL)

, right(NULL)

{

}

// Parameterized Constructor definition.

BST ::BST(int value)

{

data = value;

left = right = NULL;

}

// Insert function definition.

BST\* BST ::Insert(BST\* root, int value)

{

if (!root)

{

// Insert the first node, if root is NULL.

return new BST(value);

}

// Insert data.

if (value > root->data)

{

// Insert right node data, if the 'value'

// to be inserted is greater than 'root' node data.

// Process right nodes.

root->right = Insert(root->right, value);

}

else

{

// Insert left node data, if the 'value'

// to be inserted is greater than 'root' node data.

// Process left nodes.

root->left = Insert(root->left, value);

}

// Return 'root' node, after insertion.

return root;

}

// Inorder traversal function.

// This gives data in sorted order.

void BST ::Inorder(BST\* root)

{

if (!root) {

return;

}

Inorder(root->left);

cout << root->data << endl;

Inorder(root->right);

}

// Driver code

int main()

{

BST b, \*root = NULL;

// root = b.Insert(root, 50);

// b.Insert(root, 30);

// b.Insert(root, 20);

// b.Insert(root, 40);

// b.Insert(root, 70);

// b.Insert(root, 60);

int n;

char check;

do{ cout << "Kartik chopade Roll no: 6\n ";

cout<<"Add number in binary tree: ";

cin>>n;

root = b.Insert(root, n);

cout<<"Do you want to add more(y/n)? ";

cin>>check;

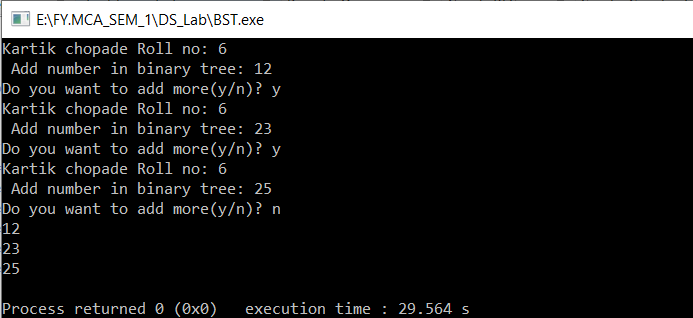
}while(check!='n');

b.Inorder(root);

return 0;

}

**Output :**



**EXPERIMENT NO: 10**

**IMPLEMENTING HEAP WITH DIFFERENT OPERATIONS PERFORMED.**

**EXPERIMENT NO: 10**

**IMPLEMENTING HEAP WITH DIFFERENT OPERATIONS PERFORMED**

**Program :**

#include<iostream>

#include<stdlib.h>

#include<stdio.h>

#define size 100

using namespace std;

class heap1{

public:

int a[size];

int last;

int size1;

};

typedef class heap1 heap;

int n;

void buildheap(heap\* h,int n1);

void reheapup(heap\* h,int nnode);

void reheapdown(heap\* h,int root);

int insert(heap\* h,int s,int data);

int deleteheap(heap \*h);

void print(heap \*h,int n);

int main(){

int i,data,ch;

cout << "Kartik chopade Roll no: 6\n";

heap \*h;

int t;

while(1){

cout<<"\n1 Create heap";

cout<<"\n2 Insert the element in heap";

cout<<"\n3 Delete the element from heap";

cout<<"\n4 exit";

cout<<"\nenter your choice::";

cin>>ch;

switch(ch){

case 1:

cout<<"\nHow many elements to be inserted range [1-9] : ";

cin>>n;

cout<<"\nInsert the elements in an array::";

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

cin>>h->a[i];

}

h->size1=n;

h->last=n-1;

buildheap(h,n);

cout<<"Heap structure is ::\n";

print(h,h->last);

break;

case 2:

cout<<"\ninsert he element to be inserted::";

cin>>data;

t=insert(h,n-1,data);

if(t){

cout<<"\nHeap after insertion::";

print(h,h->last);

}else

cout<<"Heap full";

break;

case 3:

t=deleteheap(h);

if(t){

cout<<"\nHeap after deletion::";

print(h,h->last);

}

else

cout<<"Heap empty";

break;

case 4:

exit(0);

break;

default:

cout<<"\nWrong inputs";

}

}

}

void print(heap \*h,int n){

int i;

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

cout<<"\n"<<h->a[i]<<"\n";

}

}

int deleteheap(heap \*h){

int dataout;

if(h->size1==0){

return 0;

}

dataout=h->a[0];

h->a[0]=h->a[h->last];

h->last--;

n--;

reheapdown(h,0);

cout<<"\nData deleted is:\n"<<dataout;

return 1;

}

int insert(heap\* h,int s,int data){

int i;

h->last=s;

if(h->size1==0)

return 0;

h->last++;

h->a[h->last]=data;

reheapup(h,h->last);

return 1;

}

void buildheap(heap\* h,int n1){

int walker;

h->size1=1;

for(walker=1;walker<n1;walker++)

reheapup(h,walker);

}

void reheapup(heap\* h,int nnode){

int parent,temp;

if(nnode!=0){

parent=(nnode-1)/2;

if(h->a[nnode]>h->a[parent]){

temp=h->a[nnode];

h->a[nnode]=h->a[parent];

h->a[parent]=temp;

reheapup(h,parent);

}

}

}

void reheapdown(heap\* h,int root){

int hold;

int rightdata;

int leftdata;

int largeloc;

int last=h->last;

if((root\*2+1)<=last){

leftdata=h->a[root\*2+1];

if((root\*2+2)<=last)

rightdata=h->a[root\*2+2];

else

rightdata=NULL;

if(!rightdata || leftdata>rightdata)

largeloc=root\*2+1;

else

largeloc=root\*2+2;

if(h->a[root]<h->a[largeloc]){

hold=h->a[root];

h->a[root]=h->a[largeloc];

h->a[largeloc]=hold;

reheapdown(h,largeloc);

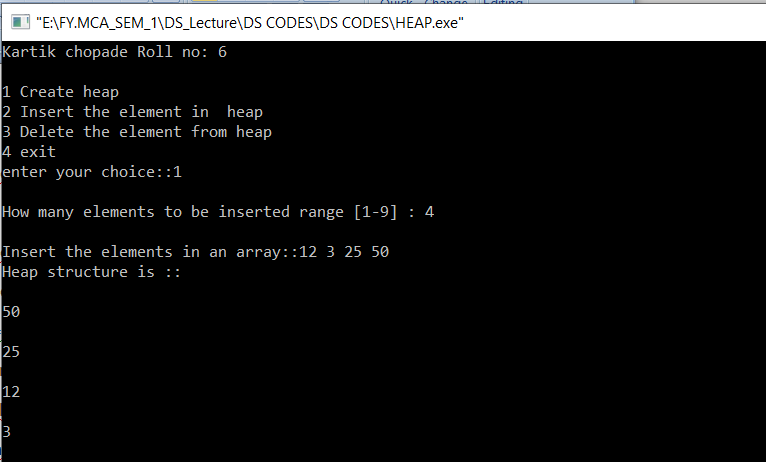
}

}

return;

}

**Output :**



**EXPERIMENT NO: 11**

**CREATE A GRAPH STORAGE STRUCTURE**

**(EG. ADJACENCY MATRIX).**

**EXPERIMENT NO: 11**

**CREATE A GRAPH STORAGE STRUCTURE**

**(EG. ADJACENCY MATRIX).**

**Program :**

#include<iostream>

using namespace std;

class adjmatrix

{

private:

int adj[5][5],n;

int i,j;

int maxedges,orx,ds;

public:

adjmatrix()

{

for(i=0;i<=4;i++)

{

for(j=0;j<=4;j++)

{

adj[i][j]=0;

}

}

}

void graph(void);

void insertvertex(void);

int insertedges(int,int);

void display(void);

};

void adjmatrix::graph(void)

{

cout<<"Enter the no. of vertices"<<endl;

cin>>n;

maxedges=n\*(n-1);

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

{

cout<<"Enter 00 to quit"<<endl;

cin>>orx>>ds;

if((orx==0)||(ds==0))

{

break;

}

if((orx>n)||(ds>n)||(orx<0)||(ds<0))

{

cout<<"Inavlid edges...!!"<<endl;

i--;

}

else

{

adj[orx][ds]=1;

}

}

}

void adjmatrix::insertvertex(void)

{

int i;

n++;

cout<<"Enter vertex"<<endl;

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

{

adj[i][n]=0;

adj[n][i]=0;

}

}

int adjmatrix::insertedges(int u,int v)

{

if(u>n)

{

cout<<"Source does not exits "<<endl;

return n;

}

if(v>n)

{

cout<<"Deatination does not exits"<<endl;

return n;

}

adj[u][v]=1;

}

void adjmatrix::display(void)

{

int i,j;

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

{

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

{

cout<<adj[i][j];

}

cout<<"\n";

}

}

int main()

{

cout << "Kartik chopade Roll no 6\n "<<endl;

int u,v;

adjmatrix a1;

a1.graph();

int opt;

char ch='y';

while(ch=='y')

{

cout<<"1.insert vertex \n 2.insert edges \n 3.display \n"<<endl;

cout<<"Enter option"<<endl;

cin>>opt;

switch(opt)

{

case 1:

{

a1.insertvertex();

cout<<"Do you want to continue...!!!"<<endl;

break;

}

case 2:

{

a1.insertedges(u,v);

cout<<"Do you want to continue...!!!"<<endl;

break;

}

case 3:

{

a1.display();

cout<<"Do you want to continue...!!!"<<endl;

break;

}

default:

{

cout<<"Do you want to continue...!!!"<<endl;

break;

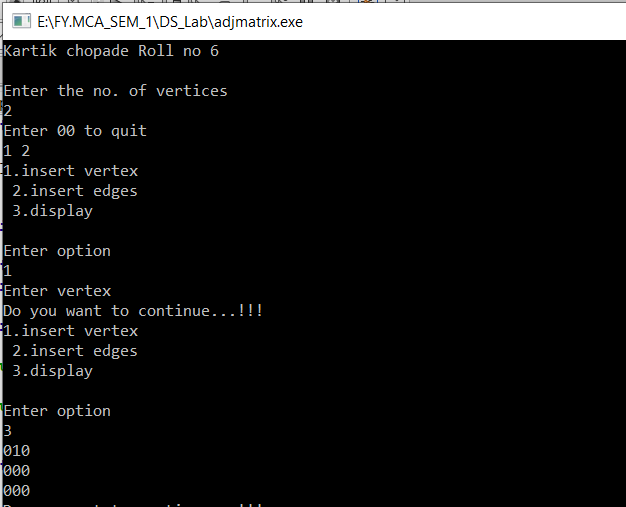
}

}

}

}

**Output :**



**EXPERIMENT NO: 12**

**PERFORM VARIOUS HASHING TECHNIQUES WITH LINEAR PROBE AS COLLISION RESOLUTION SCHEME.**

**EXPERIMENT NO: 12**

**PERFORM VARIOUS HASHING TECHNIQUES WITH LINEAR PROBE AS COLLISION RESOLUTION SCHEME.**

**Program:**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

const int SIZE=15;

static int coll;

class Hash

{

long key;

long index;

long arr[SIZE];

public:

Hash();

int getKey();

void directHash();

void subHash();

void modDivision();

void linProbe();

void digitExHash();

void foldShiftHash();

void foldBoundHash();

void display();

};

Hash::Hash(){

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

arr[i]=-1;

}

}

int Hash::getKey(){

cout<<"Enter the Value for key"<<endl;

cin>>key;

return key;

}

void Hash::linProbe(){

if(arr[index]!=-1){

cout<<"Collision detected at position"<<index<<endl;

coll++;

index=index+1;

if(index>SIZE)

{

index=index%SIZE;

}

linProbe();

}else{

arr[index]=key;

return;

}

}

void Hash::modDivision(){

index=key%SIZE;

if(index==-1){

arr[index]=key;

}else{

linProbe();

}

}

void Hash::directHash(){

index=key;

arr[index]=key;

}

void Hash::subHash(){

index=key-1000;

arr[index]=key;

}

void Hash::digitExHash(){

index=0;

long r,inc=100000,incr=1000;

for(int i=1;i<=6;i++){

if(i==1 || i== 3 || i==4){

incr=incr/10;

r=(key/inc)%10;

index=index+(r\*incr);

}

inc=inc/10;

}

if(index>SIZE){

index=index%SIZE;

}

if(arr[index]==-1){

arr[index]=key;

}else{

linProbe();

}

}

void Hash::display(){

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

cout<<arr[i]<<" | ";

}

cout<<endl;

cout<<"Total Collisions "<<coll<<endl;

}

void Hash::foldShiftHash(){

index=0;

long no=key;

while(no>0){

index=index+(no%1000);

no=no/1000;

}

index=index%100;

if(index>SIZE)

index=index%SIZE;

if(arr[index]==-1){

arr[index]=key;

}else{

linProbe();

}

}

void Hash::foldBoundHash(){

index=0;

long no=key,r,tmp,rem;

for(int i=1;i<=3;i++){

r=no%1000;

tmp=r;

if(i%2!=0){

tmp=0;

while(r>0){

rem=r%10;

tmp=(tmp\*10)+rem;

r=r/10;

}

}

index=index+tmp;

no=no/1000;

}

index=index%100;

if(index>SIZE)

index=index%SIZE;

if(index==-1){

arr[index]=key;

}else{

linProbe();

}

}

int main(){

Hash h;

int op;

cout << "Kartik chopade Roll no 6\n";

cout<<"Only 14 values will be accepted to quit in between insert -1 as key"<<endl;

cout<<"1 for direct hashing\n";

cout<<"2 for Subtraction Hashing\n";

cout<<"3 for Modulo Division Hashing\n";

cout<<"4 for digit extraction hashing\n";

cout<<"5 for shift fold Hashing\n";

cout<<"6 for shift boundry Hashing\n";

cout<<"Enter 7 to exit\n";

cin>>op;

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

switch(op){

case 1:

cout<<"Please Enter values only between 0 to 14\n"<<endl;

if(h.getKey()!=-1){

h.directHash();

}else{goto here;}

case 2:

cout<<"Enter values from 1000 to 1014 and no duplicate values : "<<endl;

if(h.getKey()!=-1)

h.subHash();

else{goto here;}

break;

case 3:

if(h.getKey()!=-1)

h.modDivision();

else{goto here;}

break;

case 4:

cout<<"Enter values only of 6 digits : "<<endl;

if(h.getKey()!=-1)

h.digitExHash();

else{goto here;}

break;

case 5:

cout<<"Enter values only of 9 digits : "<<endl;

if(h.getKey()!=-1)

h.foldShiftHash();

else{goto here;}

break;

case 6:

cout<<"Enter values only of 9 digits : "<<endl;

if(h.getKey()!=-1)

h.foldBoundHash();

else{goto here;}

break;

case 7:

exit(0);

}

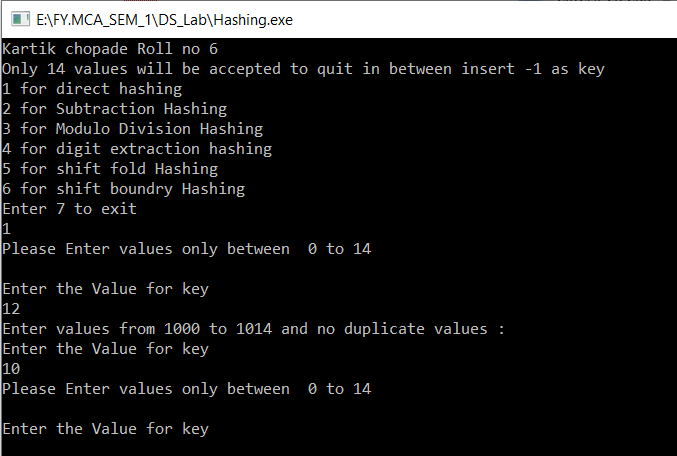
}

here:

h.display();

}

**Output :**



**EXPERIMENT NO: 13**

**CREATE A MINIMUM SPANNING TREE USING ANY METHOD KRUSKAL‟S ALGORITHM OR PRIM‟S ALGORITHM**

**EXPERIMENT NO: 13**

**CREATE A MINIMUM SPANNING TREE USING ANY METHOD KRUSKAL‟S ALGORITHM OR PRIM‟S ALGORITHM**

**Program:**

#include <bits/stdc++.h>

using namespace std;

// a structure to represent a

// weighted edge in graph

class Edge {

public:

int src, dest, weight;

};

// a structure to represent a connected,

// undirected and weighted graph

class Graph {

public:

// V-> Number of vertices, E-> Number of edges

int V, E;

// graph is represented as an array of edges.

// Since the graph is undirected, the edge

// from src to dest is also edge from dest

// to src. Both are counted as 1 edge here.

Edge\* edge;

};

// Creates a graph with V vertices and E edges

Graph\* createGraph(int V, int E)

{

Graph\* graph = new Graph;

graph->V = V;

graph->E = E;

graph->edge = new Edge[E];

return graph;

}

// A structure to represent a subset for union-find

class subset {

public:

int parent;

int rank;

};

// A utility function to find set of an element i

// (uses path compression technique)

int find(subset subsets[], int i)

{

// find root and make root as parent of i

// (path compression)

if (subsets[i].parent != i)

subsets[i].parent

= find(subsets, subsets[i].parent);

return subsets[i].parent;

}

// A function that does union of two sets of x and y

// (uses union by rank)

void Union(subset subsets[], int x, int y)

{

int xroot = find(subsets, x);

int yroot = find(subsets, y);

// Attach smaller rank tree under root of high

// rank tree (Union by Rank)

if (subsets[xroot].rank < subsets[yroot].rank)

subsets[xroot].parent = yroot;

else if (subsets[xroot].rank > subsets[yroot].rank)

subsets[yroot].parent = xroot;

// If ranks are same, then make one as root and

// increment its rank by one

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

// Compare two edges according to their weights.

// Used in qsort() for sorting an array of edges

int myComp(const void\* a, const void\* b)

{

Edge\* a1 = (Edge\*)a;

Edge\* b1 = (Edge\*)b;

return a1->weight > b1->weight;

}

// The main function to construct MST using Kruskal's

// algorithm

void KruskalMST(Graph\* graph)

{

int V = graph->V;

Edge result[V]; // Tnis will store the resultant MST

int e = 0; // An index variable, used for result[]

int i = 0; // An index variable, used for sorted edges

// Step 1: Sort all the edges in non-decreasing

// order of their weight. If we are not allowed to

// change the given graph, we can create a copy of

// array of edges

qsort(graph->edge, graph->E, sizeof(graph->edge[0]),

myComp);

// Allocate memory for creating V ssubsets

subset\* subsets = new subset[(V \* sizeof(subset))];

// Create V subsets with single elements

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

{

subsets[v].parent = v;

subsets[v].rank = 0;

}

// Number of edges to be taken is equal to V-1

while (e < V - 1 && i < graph->E)

{

// Step 2: Pick the smallest edge. And increment

// the index for next iteration

Edge next\_edge = graph->edge[i++];

int x = find(subsets, next\_edge.src);

int y = find(subsets, next\_edge.dest);

// If including this edge does't cause cycle,

// include it in result and increment the index

// of result for next edge

if (x != y) {

result[e++] = next\_edge;

Union(subsets, x, y);

}

// Else discard the next\_edge

}

// print the contents of result[] to display the

// built MST

cout << "Following are the edges in the constructed "

"MST\n";

int minimumCost = 0;

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

{

cout << result[i].src << " -- " << result[i].dest

<< " == " << result[i].weight << endl;

minimumCost = minimumCost + result[i].weight;

}

// return;

cout << "Minimum Cost Spanning Tree: " << minimumCost

<< endl;

}

// Driver code

int main()

{ cout << "Kartik Chopade Roll No:6\n";

/\* Let us create following weighted graph

10

0--------1

| \ |

6| 5\ |15

| \ |

2--------3

4 \*/

int V = 4; // Number of vertices in graph

int E = 5; // Number of edges in graph

Graph\* graph = createGraph(V, E);

// add edge 0-1

graph->edge[0].src = 0;

graph->edge[0].dest = 1;

graph->edge[0].weight = 10;

// add edge 0-2

graph->edge[1].src = 0;

graph->edge[1].dest = 2;

graph->edge[1].weight = 6;

// add edge 0-3

graph->edge[2].src = 0;

graph->edge[2].dest = 3;

graph->edge[2].weight = 5;

// add edge 1-3

graph->edge[3].src = 1;

graph->edge[3].dest = 3;

graph->edge[3].weight = 15;

// add edge 2-3

graph->edge[4].src = 2;

graph->edge[4].dest = 3;

graph->edge[4].weight = 4;

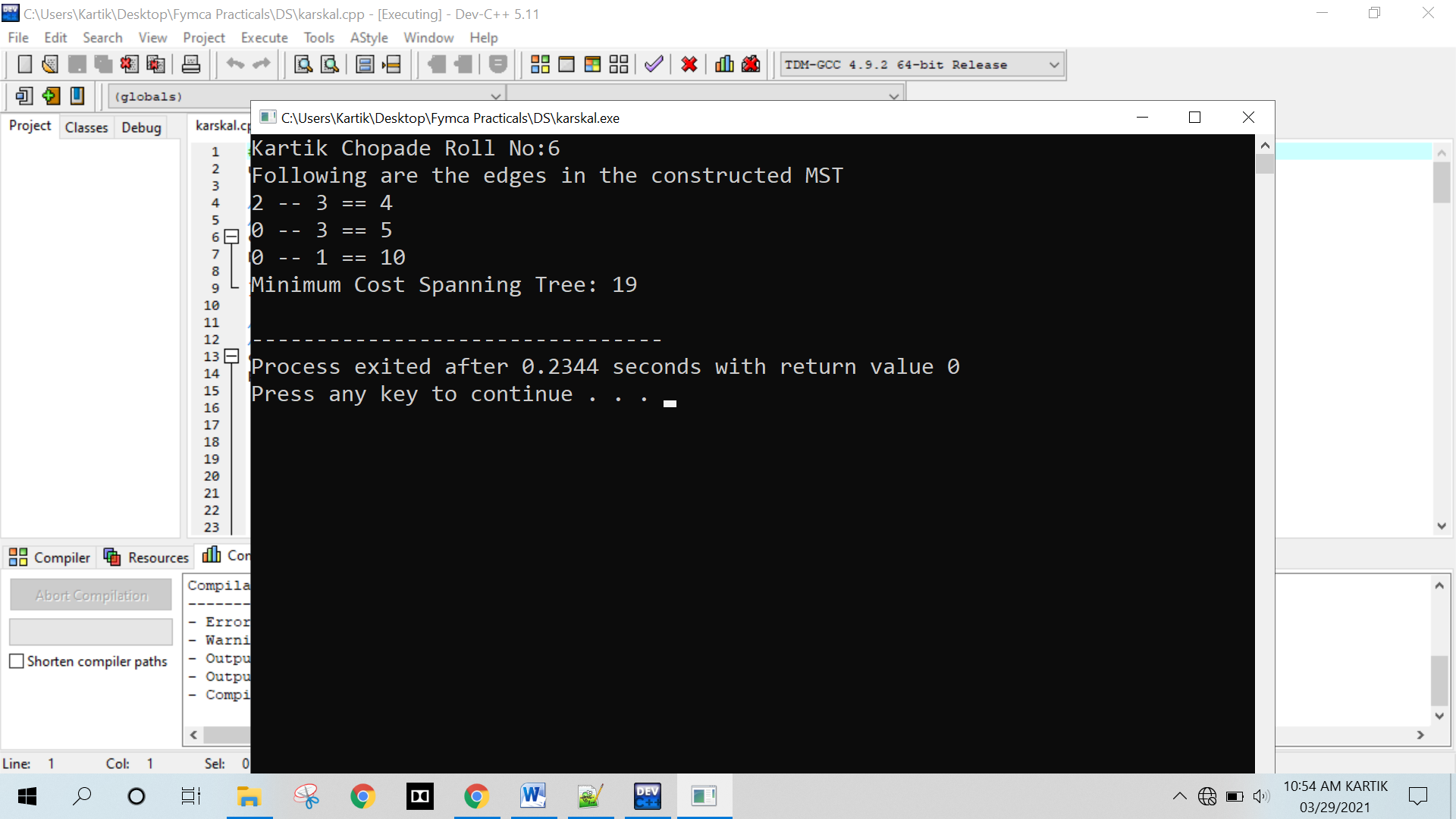
// Function call

KruskalMST(graph);

return 0;

}

**Output :**



**EXPERIMENT NO: 14**

**IMPLEMENTATION OF GRAPH TRAVERSAL. (DFS AND BFS)**

**EXPERIMENT NO: 14**

**IMPLEMENTATION OF GRAPH TRAVERSAL. (DFS AND BFS)**

**Depth First Search (DFS)**

**Program :**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10];

int main()

{

int m;

cout<<"Kartik Chopade Roll No:6\n";

cout <<"Enter no of vertices:";

cin >> n;

cout <<"Enter no of edges:";

cin >> m;

cout <<"\nEDGES \n";

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

{

cin >>i>>j;

cost[i][j]=1;

}

cout <<"Enter initial vertex to traverse from:";

cin >>v;

cout <<"DFS ORDER OF VISITED VERTICES:";

cout << v <<" ";

visited[v]=1;

k=1;

while(k<n)

{

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

if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)

{

visit[j]=1;

stk[top]=j;

top++;

}

v=stk[--top];

cout<<v << " ";

k++;

visit[v]=0;

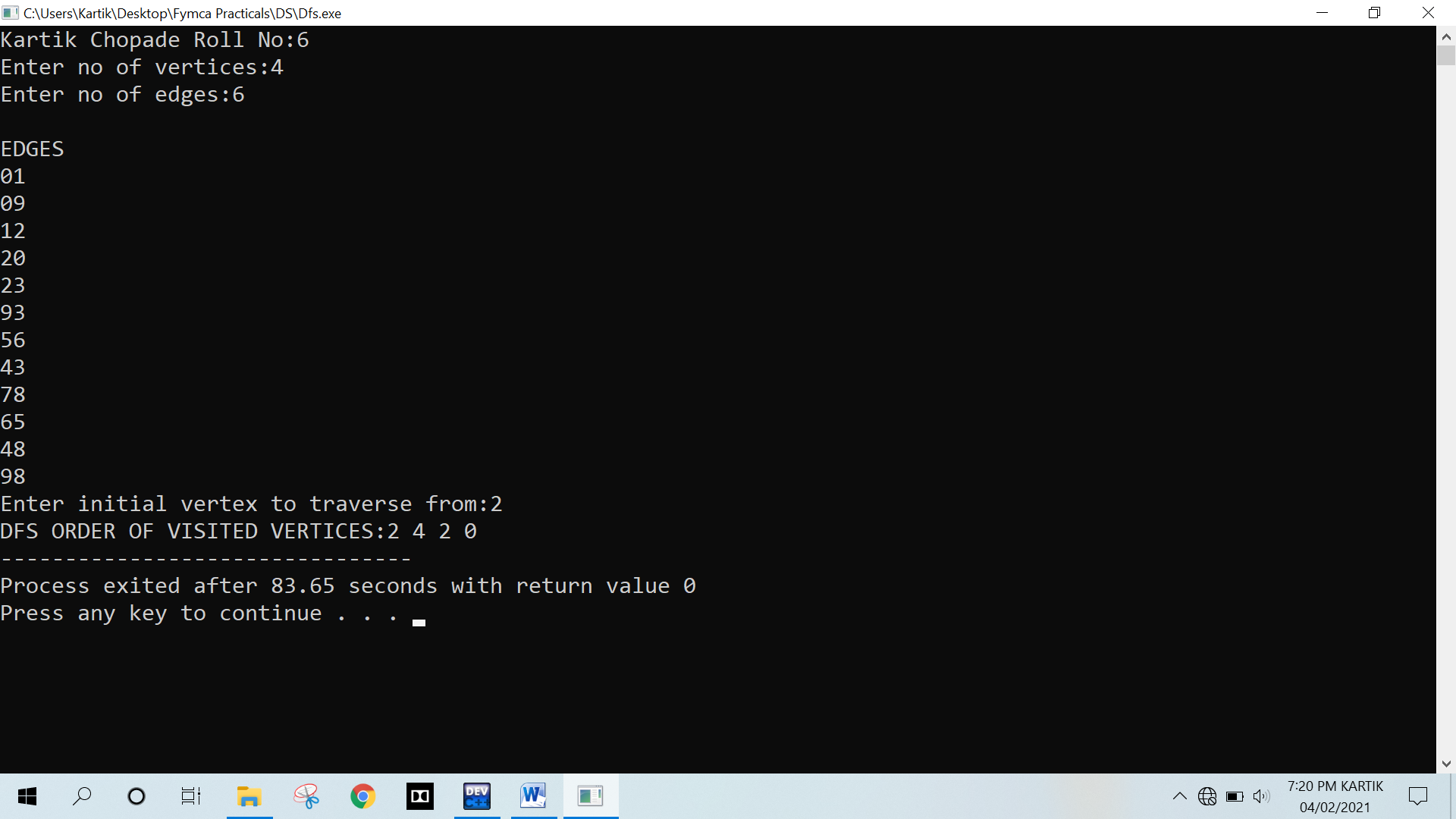
visited[v]=1;

}

return 0;

}

**Output :**



**Breadth First Traversal (BFT)**

**Program :**

#include<iostream>

#include <list>

using namespace std;

// This class represents a directed graph using

// adjacency list representation

class Graph

{

int V; // No. of vertices

// Pointer to an array containing adjacency

// lists

list<int> \*adj;

public:

Graph(int V); // Constructor

// function to add an edge to graph

void addEdge(int v, int w);

// prints BFS traversal from a given source s

void BFS(int s);

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w); // Add w to v's list.

}

void Graph::BFS(int s)

{

// Mark all the vertices as not visited

bool \*visited = new bool[V];

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

visited[i] = false;

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.push\_back(s);

// 'i' will be used to get all adjacent

// vertices of a vertex

list<int>::iterator i;

while(!queue.empty())

{

// Dequeue a vertex from queue and print it

s = queue.front();

cout << s << " ";

queue.pop\_front();

// Get all adjacent vertices of the dequeued

// vertex s. If a adjacent has not been visited,

// then mark it visited and enqueue it

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

// Driver program to test methods of graph class

int main()

{ cout << "kartik chopade Roll No:6\n";

// Create a graph given in the above diagram

Graph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

cout << "Following is Breadth First Traversal "

<< "(starting from vertex 2) \n";

g.BFS(2);

return 0;

}

**Output :**

