|  |  |
| --- | --- |
| **Array implementation of stack and its application.** | **Ex. No. 1** |
| 15-12-18 |

**YouTube Link:**[**https://youtu.be/MuLtZI3IZ6Y**](https://youtu.be/MuLtZI3IZ6Y)

**Aim :** To demonstrate the working of a stack using an array the elements of the stack may be assumed to be a type of integers

The three operations to be supported are : (a) Push(b) pop (C)isEmpty

(d)isFull (e)display (f)stacktop.

**Description**

A stack is a basic data structure that can be logically thought of as a linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack.There are basically three operations that can be performed on stacks.

**Algorithm**

**PushOperation:**

**Step 1** − Checks if the stack is full.

**Step 2** − If the stack is full, display stack is full.

**Step 3** − If the stack is not full, increments **top** to point next empty space.

**Step 4** − Adds data element to the stack location, where top is pointing.

**Step 5** − Exit.

**Pop Operation:**

**Step 1** − Checks if the stack is empty.

**Step 2** − If the stack is empty, display stack is empty.

**Step 3** − If the stack is not empty, accesses the data element at which **top** is pointing.

**Step 4** − Decreases the value of top by 1.

**Step 5** − Returns the value.

**IsEmpty():**

**Step 1** − if the stack is empty on that time top is.

**Step 2** − If the top == -1.

**Step 3** –display the stack is empty.

**Step 4** –else display stack is not empty.

**Step 5** –Exit.

**IsFull():**

**Step 1** − if the stack is full on that time top is .

**Step 2** –top==max-1.

**Step 3** –display the stack is full.

**Step 4** –else display stack is not full.

**Step 5** − Exit

**Display():**

**Step 1** –In display method display the elements of the stack that user can enter.

**Step 2** –for(i=top to i>=0).

**Step 3** –cout<< stack[i].

**Step 4** − Exit

**Stack\_Top():**

**Step 1** –Display the element of the stack.

**Step 2** –display cout<<s[top];.

**Step 5** − Exit

**Program**

#include <iostream>

using namespace std;

int stack[100], n=100, top=-1;

void push(int b) {

if(top>=n-1)

cout<<"Stack is Full"<<endl;

else {

top++;

stack[top]=b;

}

}

void pop() {

if(top<=-1)

cout<<"Stack empty"<<endl;

else {

cout<<"the popped element is : "<< stack[top] <<endl;

top--;

}

}

void display() {

if(top>=0) {

cout<<"stack elements are:";

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

cout<<stack[i]<<" ";

cout<<endl;

} else

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

}

int main() {

int a, b;

cout<<"1) push in stack"<<endl;

cout<<"2) pop from stack"<<endl;

cout<<"3) display stack"<<endl;

cout<<"4) exit"<<endl;

do {

cout<<"enter choice: "<<endl;

cin>>a;

switch(a) {

case 1: {

cout<<"enter value to be pushed:"<<endl;

cin>>b;

push(b);

break;

}

case 2: {

pop();

break;

}

case 3: {

display();

break;

}

case 4: {

cout<<"exit"<<endl;

break;

}

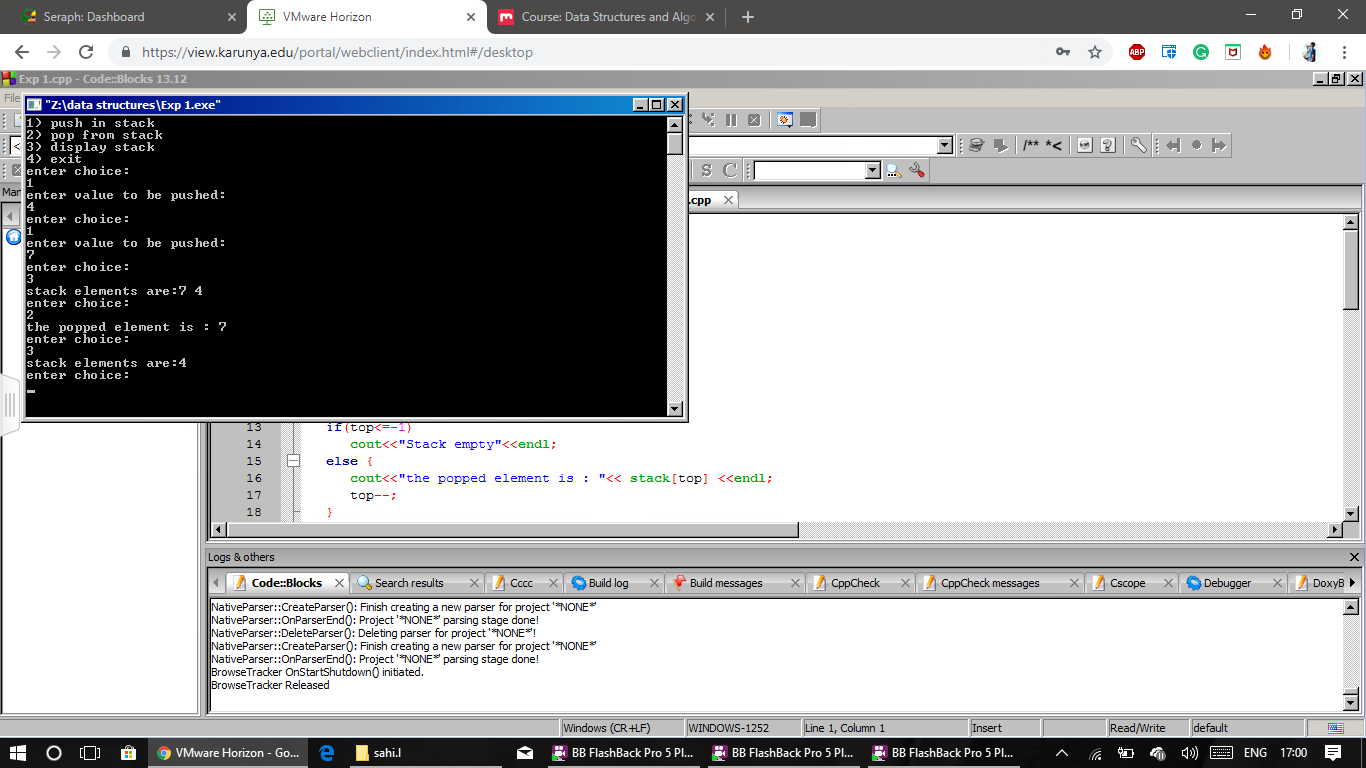
}

}while(a!=4);

return 0;

}

**Input & Output**



**Aim:**

To perform infix to postfix using stack.

**Algorithm:**

**Step 1:** start the program.

**Step 2:** create a class infix and string infix variable and array of some size.

**Step 3:** in main function use get data function to enter infix expression.

**Step 4:** in main function call convert function which converts infix to postfix.

**Step 5:** in convert function use while loop to traverse to the end of infixexpression.

**Step 6:**if the variable if alphanumeric print it.

**Step 7:**if the variable is operator push it into stack.

**Step 8:** while pushing the operator compare the hirachy and according to itperform push or pop operation in stack.

**Step 9:** at last using while loop print all variables in stack.

**Program:**

#include<iostream>

using namespace std;

class infixs

{

string infix,stack;

int top;

public:

infixs()

{

top=-1;

}

void getdata()

{

cout<<"enter the infix expression"<<endl;

cin>>infix;

}

void push(char ch)

{

top++;

stack[top]=ch;

}

char pop()

{

char ch=stack[top];

top--;

return ch;

}

void convert();

int pre(char ch);

};

void infixs::convert()

{

int i=0,j=0;

char ch;

push('#');

while(infix[i]!='\0')

{

ch=infix[i];

if((ch>='a'&&ch<='z')||(ch>='A'&&ch<='Z'))

{

cout<<ch;

}

else if(ch=='+'|| ch=='-'|| ch=='\*'|| ch=='/'|| ch=='^')

{

while(pre(stack[top])>=pre(ch) && stack[top]!='(')

{

cout<<pop();

}

push(ch);

}

else if(ch=='(')

{

push(ch);

}

else if(ch==')')

{

while(stack[top]!='(')

{

cout<<pop();

}

pop();

}

i++;

}

while( stack[top]!='#' )

{

cout<<pop();

}

}

int infixs :: pre( char ch )

{

switch( ch )

{

case '+':

case '-':

return 1;

break;

case '\*':

case '/':

return 2;

break;

case '$':

case '^':

return 3;

break;

/\*case '(':

return 5;

break;

case ')':

return 0;

break;\*/

case '#':

return -1;

break;

default :

return 4;

break;

}

}

int main()

{

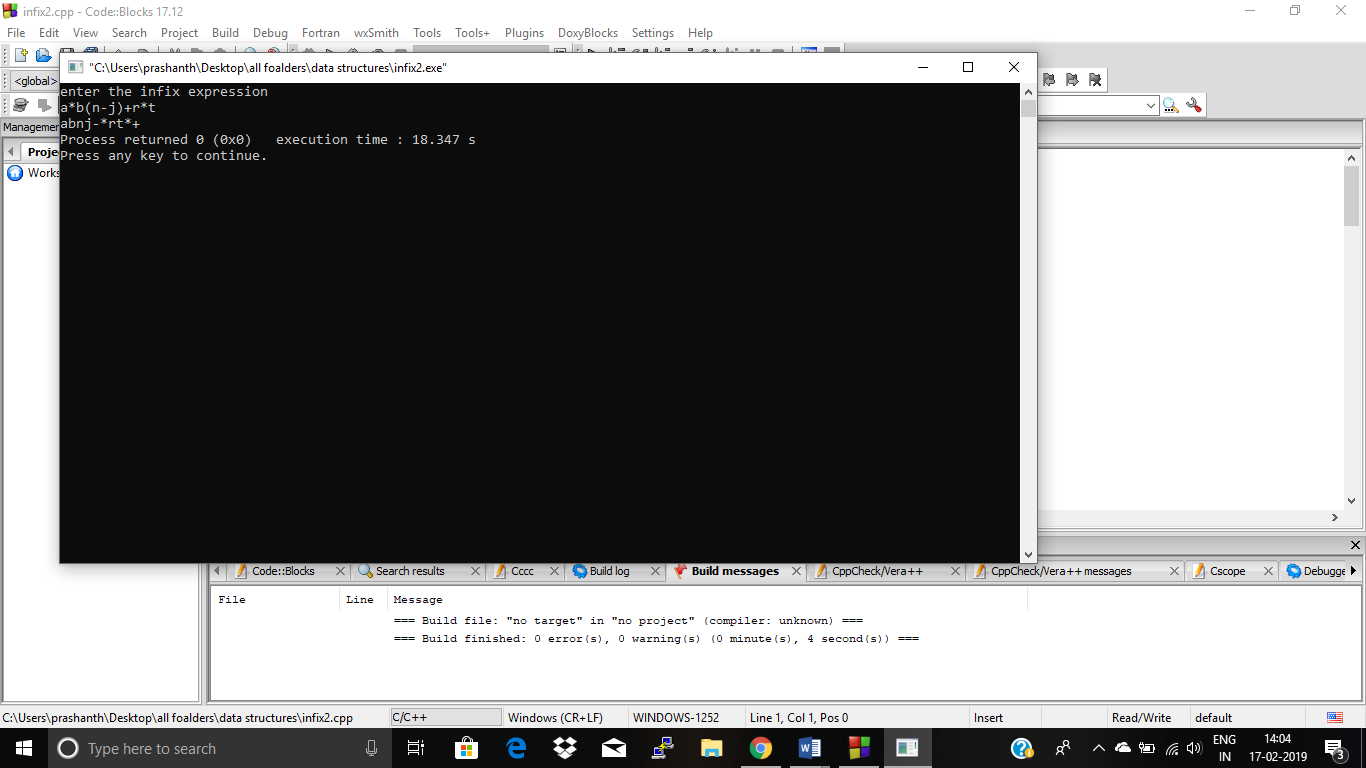
infixs s;

s.getdata();

s.convert();

}

**Output:**



**Result**

The program is executed and output is verified for valid inputs.

|  |  |
| --- | --- |
| **Implementation of Linear queue and circular queue** | **Ex. No. 2** |
| 12-12-18 |

**YouTube link:** [**https://youtu.be/\_R1\_LWvoCks**](https://youtu.be/_R1_LWvoCks)

**Aim**

To implement the functions of enqueue() and dequeue() of an linera queue And circularqueue

**Description**

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data(enqueue) and the other is used to remove data(dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

**Algorithm**

**LinearQueue:**

**EnqueueOperation:**

Queues maintain two data pointers, **front** and **rear**.

**Step 1** − Check if the queue is full.

**Step 2** − If the queue is full, display queue is full.

**Step 3** − If the queue is not full, increment **rear** pointer to point the next empty space.

**Step 4** − Add data element to the queue location, where the rear is pointing.

**Step 5** − return the value.

**DequeueOperation:**

**Step 1**:Check if the queue is empty.

**Step 2**:If the queue is empty, produce underflow error and exit.

**Step 3:**  If the queue is not empty, access the data where **front** is pointing.

**Step 4:**  Increment **front** pointer to point to the next available data element.

**Step 5**: Return the value.

**CircularQueue:**

**Step1:**To insert an element check whether the queue is full or empty

**Step 2:**If (rear+1)%3 is equal to front the queue is full

**Step 3:**The increment the rear position and insert the element

**Step 4:**To delete an element from the queue check whether the queue if full or empty

**Step 5:**In front is equal then queue is empty return back

**Step 6:**Otherwise enter the data in front to a variable

**Step 7:**return the data stored in variable

**Step 8:**stop the program

**Program**

**Circular queue:**

#include<iostream>

using namespace std;

class queue

{

int a[10];

int f=-1;

int r=-1;

int s=4;

public:

void enqueue(int n)

{

if((r+1)%s==f)

{

cout<<"queue is full"<<endl;

}

r=(r+1)%s;

a[r]=n;

if(f==-1)

{

f=0;

}

}

intdequeue()

{

if(f==-1)

{

cout<<"queue is empty"<<endl;

}

int t;

t=a[f];

cout<<t<<endl;

if(r==f)

{

f=r=-1;

}

else

{

f=((f+1)%s);

}

}

void display()

{

for(inti=f;i<=r;i++)

{

cout<<a[i]<<endl;

}

}

};

int main()

{

queue q;

int c;

char ch;

do

{

cout<<"enter the option"<<endl;

cout<<"1.enqueue"<<endl;

cout<<"2.dequeue"<<endl;

cout<<"3.display"<<endl;

cin>>c;

switch(c)

{

case 1:

int d;

cout<<"enter the element"<<endl;

cin>>d;

q.enqueue(d);

break;

case 2:

q.dequeue();

break;

case 3:

q.display();

break;

default:

cout<<"enter the correct option"<<endl;

break;

}

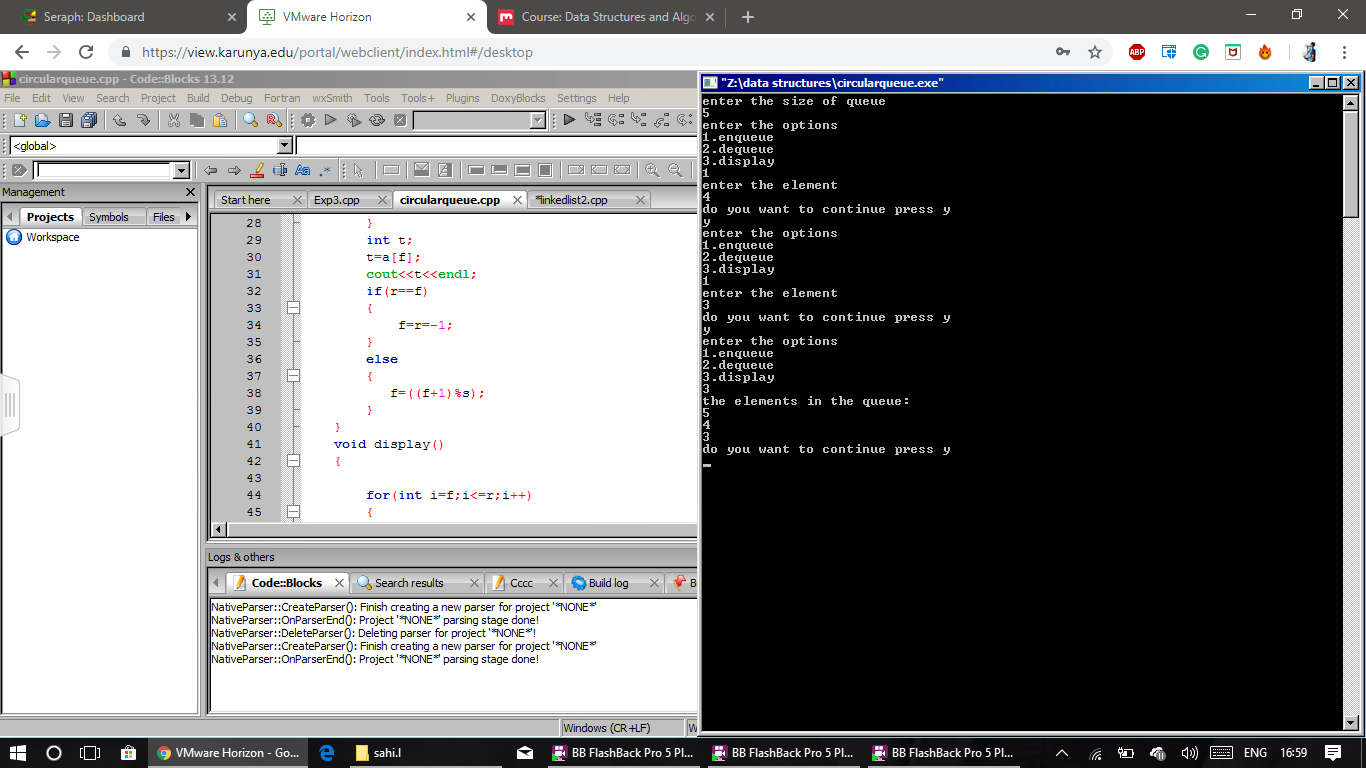
cout<<"do you want to continue press y"<<endl;

cin>>ch;

}while(ch=='y');

}

**Input and output**



**Linear queue**

#include<iostream>

using namespace std;

class queue

{

int a[10];

int f=-1,r=-1;

public:

void enqueue(intb,int s)

{

if(r==(s-1))

{

cout<<"the queue is full"<<endl;

}

else

{

a[++r]=b;

}

}

void dequeue()

{

if(r==f)

{

cout<<"the queue is empty"<<endl;

}

cout<<a[++f]<<endl;

f++;

}

void display()

{

for(inti=f++;i<=r;i++)

{

cout<<a[i]<<endl;

}

}

};

int main()

{

queue q;

int s;

int c;

char ch;

cout<<"enter the size of queue"<<endl;

cin>>s;

do

{

cout<<"enter the options"<<endl;

cout<<"1.Enqueue"<<endl;

cout<<"2.Dequeue"<<endl;

cout<<"3.Display"<<endl;

cin>>c;

switch(c)

{

case 1:

int d;

cout<<"enter the element"<<endl;

cin>>d;

q.enqueue(d,s);

break;

case 2:

q.dequeue();

break;

case 3:

cout<<"the elements in the queue:"<<endl;

q.display();

break;

default:

cout<<"enter the correct element"<<endl;

break;

}

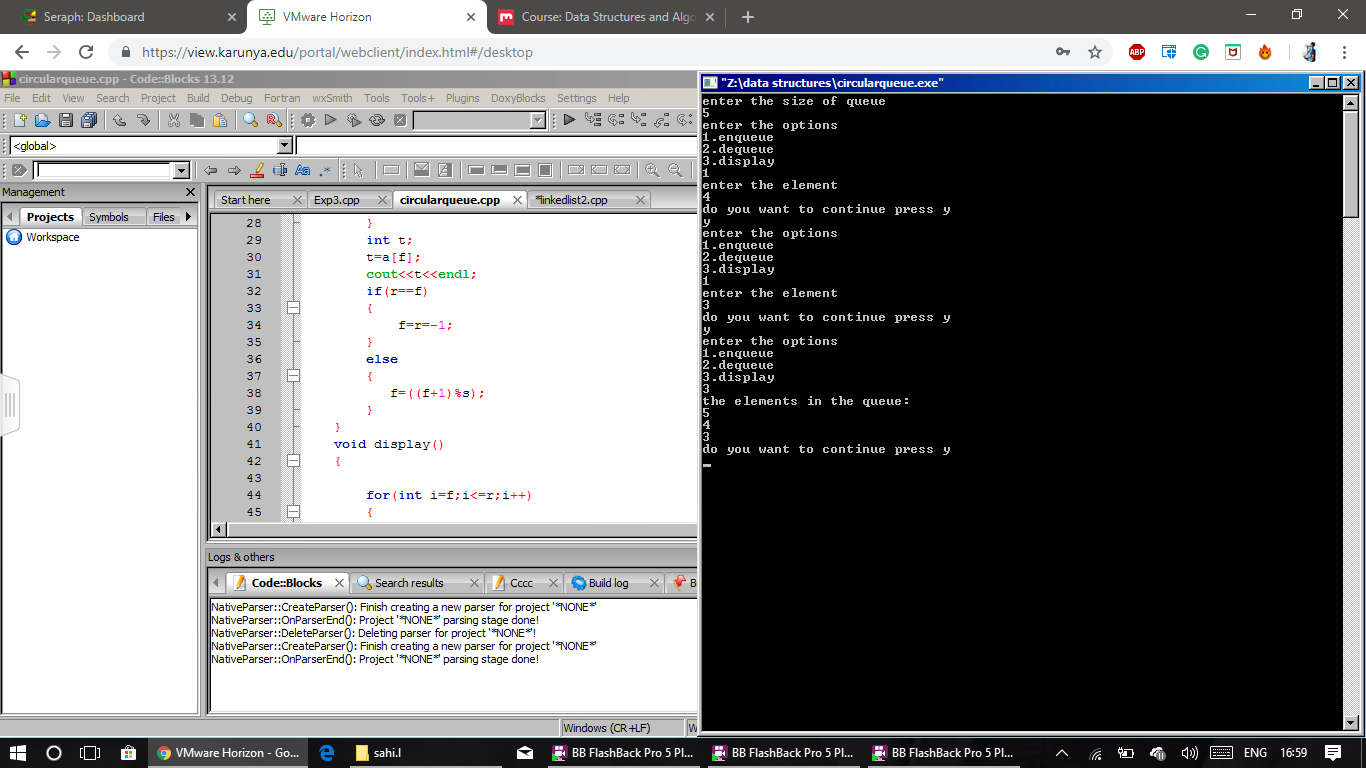
cout<<"do you want to continue press y"<<endl;

cin>>ch;

}while(ch=='y');

}

**Input & Output**



**Result**

The above program has been verified and executed successful.

|  |  |
| --- | --- |
| **Implementation of Singly Linked list** | **Ex. No. 3** |
| 19-12-18 |

**YouTube link:** <https://youtu.be/mzC9s1qMxzw>

**Aim**

Implementation of linked list using arrays.To insert the element and delete the element

**Description**

Data Structures/Singly Linked Lists. Singly Linked Lists are a type of data structure. In asingly linked list, each node in the list stores the contents and a pointer or reference to the next node in the list. It does not store any pointer or reference to the previous node.

**Algorithm**

**Insertion:**

Inserting a new node in the linked list is called insertion.

A new node is created and inserted in the linked list.

There are three cases considered while inserting a node:

1. Insertion at the start
2. Insertion at the end
3. Insertion at a particular position

**Insertion at the Start:**

**Step 1:**New node should be connected to the first node, which means the head. This can be achieved by putting the address of the head in the next field of the new node.

**Step 2:**New node should be considered as a head. It can be achieved by declaring head equals to a new node.

**Insertion at the End:**

**Step 1:**we check the node is empty and create a new node called temp and assign the value int x in its data field

**Step 2:**Link temp-> next to head pointer and head equal to temp .we inserted the newly created node at the end of the linked list..

**Insertion at Particular Position:**

**Step 1:**The insertion of a new node at a particular position

**Step 2:**In this case, we don’t disturb the head and tail nodes.

**Step 3:**Rather, a new node is inserted between two consecutive nodes. So, these two nodes should be accessible by our code.

**Step 4:**We call one node as current and the other as previous, and the new node is placed between them.Pass the address of the new node in the next field of the previous node.Pass the address of the current node in the next field of the new node.We will access these nodes by asking the user at what position he wants to insert the new node. Now, we will start a loop to reach those specific nodes. We initialized our current node by the head and move through the linked list. At the end, we would find two consecutive nodes.

**Deletion:**

**Deletion at the start.**

**Deletion at the end.**

**,Deletion at a particular position.**

**Deletion at the Start:**

**Step 1:**In this case, the first node of the linked list is deleted.

**Step 2:**The first node is called the head. delete the head node.

**Step 3:**Declare a **temp** pointer and pass the address of the first node, head to this pointer.

**Step 4:**Declare the second node of the list as head as it will be the first node of linked list after deletion.

**Step 4:**Delete the temp node.

**Deletion at the End:**

**Step 1:** In the case of the first node, you just need access to the head and you can delete it.

**Step 2:**But in the case of the last node, you also need access to the second to the last node of the linked list as you will delete the last node and make the previous node as the tail of linked list.

**Step 4:** Find a node that comes before the last node. This can be achieved by traversing the linked list.

**Step 5:** We would make two temporary pointers and let them move through the whole linked list.

**Step 6:** At the end, the previous node will point to the second to the last node and the current node will point to the last node. node to be deleted.

**Step 7:**We would delete this node and make the previous node as the tail.

**Deletion at a Particular Position:**In linked list, we can delete a specific node. Here we don’t use the head and tail nodes. We ask the user to input the position of the node to be deleted. After that, we just move two temporary pointers through the linked list until we reach our specific node. Now, we delete our current node and pass the address of the node after it to the previous pointer. This way, the current node is removed from the linked list and the link is established between its previous and next node.

**Program**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node \*next;

};

struct node \*head=NULL;

void insertfro(int x)

{

node\* t=new node;

t->data=x;

t->next=head;

head=t;

}

void insertla(int x)

{

node \*t=new node;

t->data=x;

node \*m=head;

while(m->next!=NULL){

m=m->next;

}m->next=t;

t->next=NULL;

}

void insertmid(int x)

{

int n;

cout<<"enter at wch position the node is to be present"<<endl;

cin>>n;

node \*temp=new node;

temp->data=x;

node \*temp1=head;

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

{

if(temp1==NULL){

cout<<n<<"node doesn't exist"<<endl;

break;

}

temp1=temp1->next;

}

temp->next=temp1->next;

temp1->next=temp;

}

intdeletefro(){

if(head==NULL)

return -1;

node \*temp=head;

head=head->next;

return temp->data;

delete temp;

}

intdella()

{

node \*temp=head;

node \*oldtemp=head;

if(head==NULL)

{

cout<<"empty list"<<endl;

}

while(temp->next!=NULL){

oldtemp=temp;

temp=temp->next;

}

oldtemp->next=NULL;

delete temp;

return temp->data;

}

intdelmid(){

int n;

cout<<"enter wch node u want to delete"<<endl;

cin>>n;

node \*temp=head;

node \*oldtemp;

for(inti=1;i<n-1;n++){

oldtemp=temp;

temp=temp->next;

}

oldtemp->next=temp->next;

cout<<temp->data;

delete temp;

}

void display(){

node \*a;

a=head;

while(a!=NULL)

{

cout<<a->data<<" ";

a=a->next;

}

}

int main()

{

intc;charch;int d;

do{

cout<<"1 for insertfront \n2 forinsertback \n3 for insertmiddle\n4 for deltetfront \n5 for deletelast \n6 for deletemid \n7 for display\n";

cin>>c;

switch(c)

{

case 1:

cout<<"enter the no";

int x;

cin>>x;

insertfro(x);

break;

case 2:

cout<<"enter the no";

cin>>x;

insertla(x);

break;

case 3:

cout<<"enter the no";

cin>>x;

insertmid(x);

break;

case 4:

deletefro();

break;

case 5:

della();

break;

case 6:

delmid();

break;

case 7:

display();

break;

default:

cout<<"the choice is wrong";

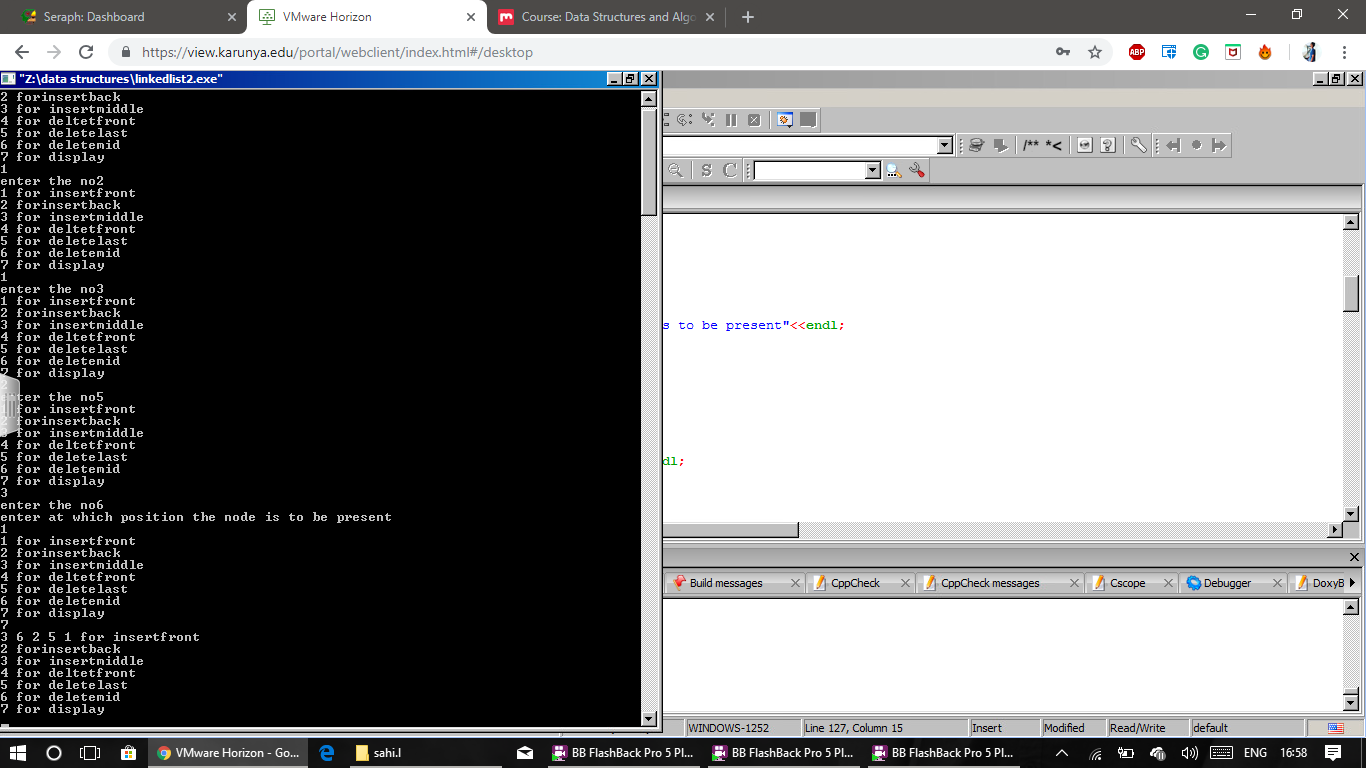
}

cout<<"enter y to continue"<<endl;

cin>>ch;

}while(ch=='y');

}

**Input & Output**

**Result**

The above program has been verified and executed successfully

|  |  |
| --- | --- |
| **Implementation of Stack and Queue using Linked list** | **Ex. No. 4** |
| 09-01-2019 |

**Aim:**

Implementation of the stack using a single linked list.

**Description:**

**Algorithm:**

**Linked Stack:**

**push():**

**Step 1:** Create a new node.

**Step 2:** put value for the node.

**Step 3:** make it's next to point to the 1st node.

**Step 4:** point top to this node.

**Pop():**

**Step 1:** If the stack is not empty

**Step 2:** point the top to top next.

**Step 3:** point the old no of value.

**Step 4:** delete the last node.

**Display():**

**Step 1:** create the temp node.

**Step 2:** Assign top to temp.

**Step 3:** make a while loop to display the elements of the top of the stack to still

Null

**Step 4:** display all elements in the stack.

**Program**

#include <iostream>

using namespace std;

struct Node {

int data;

struct Node \*next;

};

struct Node\* top = NULL;

void push(int val) {

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

newnode->data = val;

newnode->next = top;

top = newnode;

}

void pop() {

if(top==NULL)

cout<<"Stack Underflow"<<endl;

else {

cout<<"The popped element is "<< top->data <<endl;

top = top->next;

}

}

void display() {

struct Node\* ptr;

if(top==NULL)

cout<<"stack is empty";

else {

ptr = top;

cout<<"Stack elements are: ";

while (ptr != NULL) {

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

ptr = ptr->next;

}

}

cout<<endl;

}

int main() {

int ch, val;

cout<<"1) Push in stack"<<endl;

cout<<"2) Pop from stack"<<endl;

cout<<"3) Display stack"<<endl;

cout<<"4) Exit"<<endl;

do {

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

cin>>ch;

switch(ch) {

case 1: {

cout<<"Enter value to be pushed:"<<endl;

cin>>val;

push(val);

break;

}

case 2: {

pop();

break;

}

case 3: {

display();

break;

}

case 4: {

cout<<"Exit"<<endl;

break;

}

default: {

cout<<"Invalid Choice"<<endl;

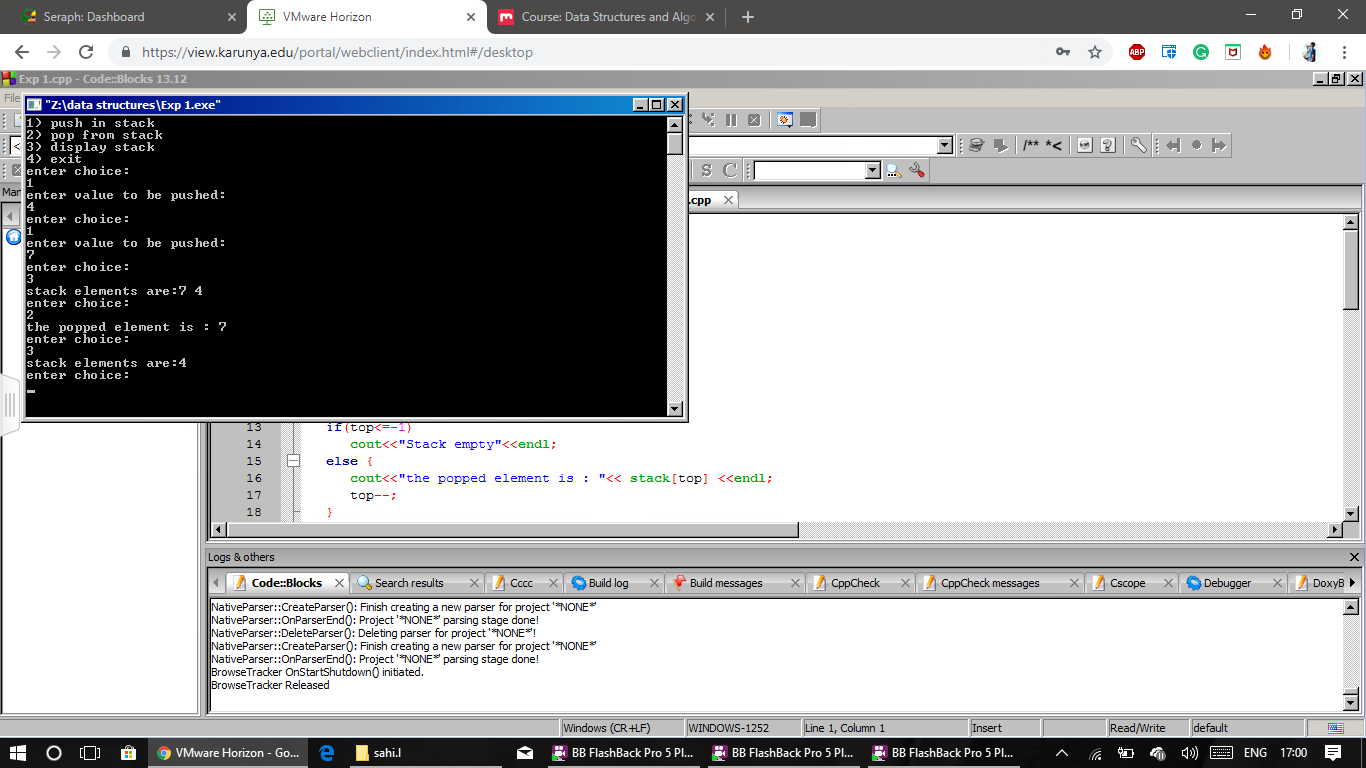
}

}

}while(ch!=4);

return 0;

}

**Input & Output**

**Aim:**

Implementation of the queue using a single linked list.

**Algorithm:**

**LinkedQueue:**

**Step 1:** create a new node.

**Step 2:** Assign the data to the node.

**Step 3:** Assign the rear pointer next to this node.

**Step 4:** Assign the front pointer to the created node.

**DeQueue:**

**Step 1:** if the top is null.

**Step 2:** point the queue is empty.

**Step 3:** else point the top value.

**Step 4:** Assign the top next to its value.

**Step 5:** Delete the top node

**Display:**

**Step 1:** Create a temp node.

**Step 2:** Assign top to temp.

**Step 3:** makes a while loop condition top next to it.

**Step 4:** display the elements in the queue

**Program:**

#include <iostream>

using namespace std;

struct node {

int data;

struct node \*next;

};

struct node\* front = NULL;

struct node\* rear = NULL;

struct node\* temp;

void Insert() {

int val;

cout<<"Insert the element in queue : "<<endl;

cin>>val;

if (rear == NULL) {

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

rear->next = NULL;

rear->data = val;

front = rear;

} else {

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

rear->next = temp;

temp->data = val;

temp->next = NULL;

rear = temp;

}

}

void Delete() {

temp = front;

if (front == NULL) {

cout<<"Underflow"<<endl;

return;

}

else

if (temp->next != NULL) {

temp = temp->next;

cout<<"Element deleted from queue is : "<<front->data<<endl;

free(front);

front = temp;

} else {

cout<<"Element deleted from queue is : "<<front->data<<endl;

free(front);

front = NULL;

rear = NULL;

}

}

void Display() {

temp = front;

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

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

return;

}

cout<<"Queue elements are: ";

while (temp != NULL) {

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

temp = temp->next;

}

cout<<endl;

}

int main() {

int ch;

cout<<"1) Insert element to queue"<<endl;

cout<<"2) Delete element from queue"<<endl;

cout<<"3) Display all the elements of queue"<<endl;

cout<<"4) Exit"<<endl;

do {

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

cin>>ch;

switch (ch) {

case 1: Insert();

break;

case 2: Delete();

break;

case 3: Display();

break;

case 4: cout<<"Exit"<<endl;

break;

default: cout<<"Invalid choice"<<endl;

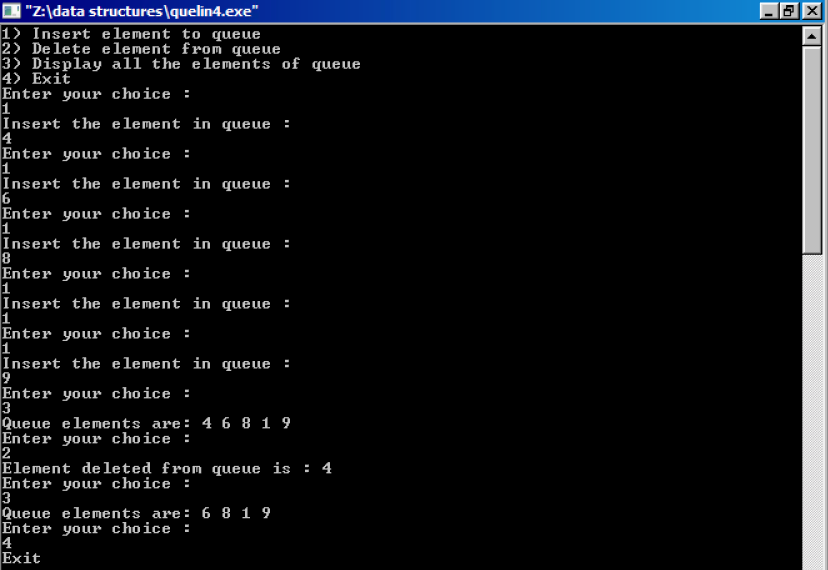
}

} while(ch!=4);

return 0;

}

**Output:**



**Result**

The program is executed and output is verified for valid inputs.

**YouTube link:**

<https://youtu.be/BaP9SQ1QE-8>

|  |  |
| --- | --- |
| **Doubly linked list** | **Ex. No. 5** |
| 19-01-2019 |

**Aim:**

To implement a doubly linked list.

**Algorithm:**

**Step1:** start the program.

**Step2:** select any one of the options to enter the data if want to insert and select node if want to delete.

**Step3:** insert front keep prev and next as null and insert the data if already assign node as head if the node exists keep head next as node address and keep node prev as head and node next as null.

**Step4:** to insert back use while loop to traverse to end and keep the last node next as node address and new node prev as last node address and node next as null.

**Step5:** to insert in middle get the node number and traverse using for loop and perform the operation.

m->prev->next=n;

n->prev=m->prev;

n->next=m;

m->prev=n;

**Step6:** to delete front create a node reference and assign it to head and store head value in a variable and assign head to head next delete node.

**Step7:** to delete at the last using while loop goes to the last node of the list and assign last node prev next to null and delete the node.

**Step8:** to delete in the middle get the node number from the user and traverse using for loop and perform the operation.

p=n->prev->data;

n->prev->prev->next=n;

n->prev=n->prev->prev;

**Step9:** to display use the while loop to reach the last node and while traversing print the nodes data.

**Step 10:** stop the program.

**Program:**

#include <iostream>

using namespace std;

struct node {

int data;

node \*prev;

node \*next;

}\*head=NULL;

void insertFront(int x) {

struct node\* t = new node;

t->data = x;

t->prev = NULL;

t->next = head;

if(head != NULL)

head->prev = t ;

head = t;

}

void insertEnd(int x)

{

if(head==NULL){

insertFront(x);

return;

}

node \*temp=new node;

temp->data=x;

temp->next=NULL;

node \*temp1=head;

while(temp1->next!=NULL)

{

temp1=temp1->next;

}

temp->prev=temp1;

temp1->next=temp;

}

void insertNode(int x,int n)

{

node \*temp=new node;

temp->data=x;

node \*temp1=head;

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

{

if(temp1==NULL)

{

cout<<"Empty";

return;

}

temp1=temp1->next;

}

temp->next=temp1->next;

temp->prev=temp1;

temp1->next->prev=temp;

temp1->next=temp;

}

void deleteFront()

{

node \*temp=head;

if(temp==NULL)

{

cout<<"Empty list"<<endl;

return;

}

head=head->next;

head->prev=NULL;

delete temp;

}

void deleteEnd()

{

node \*temp=head;

if(temp==NULL)

{

cout<<"Empty list"<<endl;

return;

}

while(temp->next!=NULL)

temp=temp->next;

temp->prev->next=NULL;

delete temp;

}

void deleteNode(int n)

{

node \*temp=head;

if(temp==NULL)

{

cout<<"Empty list"<<endl;

return;

}

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

{

if(temp==NULL)

{

cout<<"Empty";

return;

}

temp=temp->next;

}

temp->prev->next=temp->next;

temp->next->prev=temp->prev;

delete temp;

}

void display() {

node\* temp;

temp = head;

while(temp != NULL) {

cout<< temp->data <<endl;

temp = temp->next;

}

}

int main() {

int c;

int num,pos;

char ch='y';

while(ch=='y')

{

cout<<"Enter 1:Insert in the front"<<endl<<"2:Insert at the end"<<endl<<"3:Insert in nth position"<<endl;

cout<<"4:Delete the front node"<<endl<<"5:Delete in the end"<<endl<<"6:Delete from nth position"<<endl<<"7 for printing"<<endl;

cin>>c;

switch(c)

{

case 1:cout<<"Enter element"<<endl;

cin>>num;

insertFront(num);

break;

case 2:cout<<"Enter element"<<endl;

cin>>num;

insertEnd(num);

break;

case 3:cout<<"Enter element"<<endl;

cin>>num;

cout<<"Enter the position"<<endl;

cin>>pos;

insertNode(num,pos);

break;

case 4:deleteFront();

break;

case 5:deleteEnd();

break;

case 6: cout<<"Enter position"<<endl;

cin>>pos;

deleteNode(pos);

break;

case 7:display();

break;

default:cout<<"Invalid choice"<<endl;

break;

}

cout<<"Do you want to continue?"<<endl;

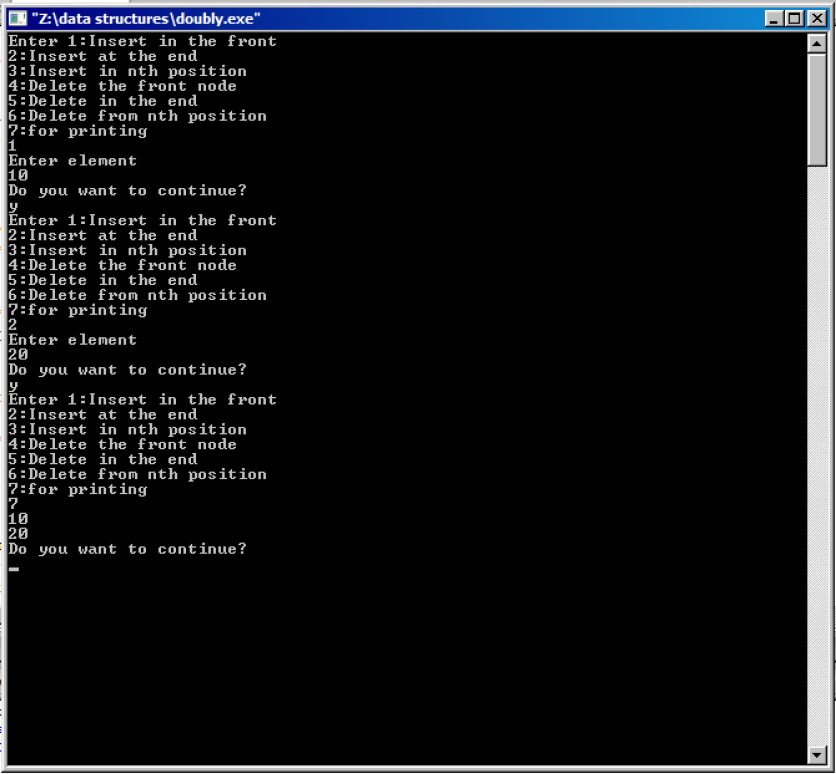
cin>>ch;

}

return 0;

}

**Output:**



**Result**

The program is executed and output is verified for valid inputs.

**Link:**

[**https://youtu.be/b4UWOkB-l1A**](https://youtu.be/b4UWOkB-l1A)

|  |  |
| --- | --- |
| **Implementation of binary search and linear search** | **Ex. No. 6** |
| **23-01-2019** |

**Aim:**

Implementation of linear search.

**Algorithm :**

**Linear search:**

Step 1: let the element to be searched in an array a[] be k.

Step 2: we traverse the array from its first position and compare each element with k

Step 3: if it matches, we return true.

**Program:**

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

int arr[10],i,num,n,c=0,pos;

cout<<"Enter the array size:";

cin>>n;

cout<<"Enter Array Elements "<<endl;

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

{

cin>>arr[i];

}

cout<<"Enter the number to be searched:";

cin>>num;

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

{

if(arr[i]==num)

{

c=1;

pos=i+1;

break;

}

}

if(c==0)

{

cout<<"Number not found......";

}

else

{

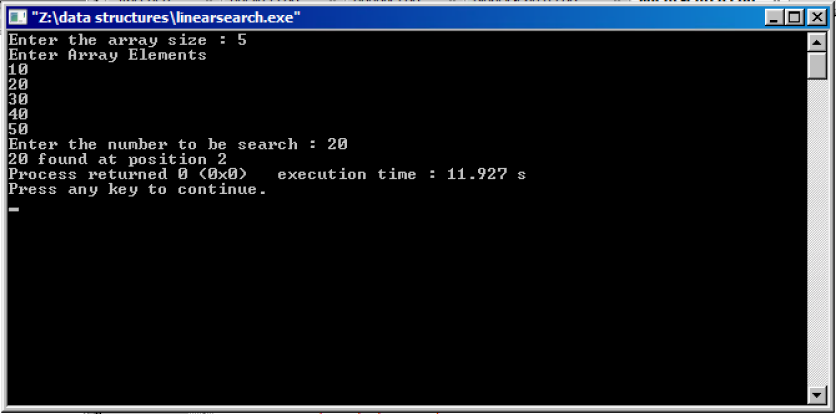
cout<<num<<"found at position"<<pos;

}

getch();

}

**Output:**



**Aim :**

Implimentation binary search.

**Algorithm :**

**Binary Search:**

**Step 1:** let the element to be searched in the array a[ ] be k.

**Step 2:** we look at the middle element of the array if its equal to k we return true

and print the elements

**Step 3:** else if it is less than k, we bring the right pointer to mid-1.

**Step 4:** else we bring the left pointer mid+1.

**Program:**

#include <iostream>

using namespace std;

int main()

{

int count, i, arr[30], num, first, last, middle;

cout<<"how many elements would you like to enter?:";

cin>>count;

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

{

cout<<"Enter number "<<(i+1)<<": ";

cin>>arr[i];

}

cout<<"Enter the number that you want to search:";

cin>>num;

first = 0;

last = count-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < num)

{

first = middle + 1;

}

else if(arr[middle] == num)

{

cout<<num<<" found in the array at the location "<<middle+1<<"\n";

break;

}

else {

last = middle - 1;

}

middle = (first + last)/2;

}

if(first > last)

{

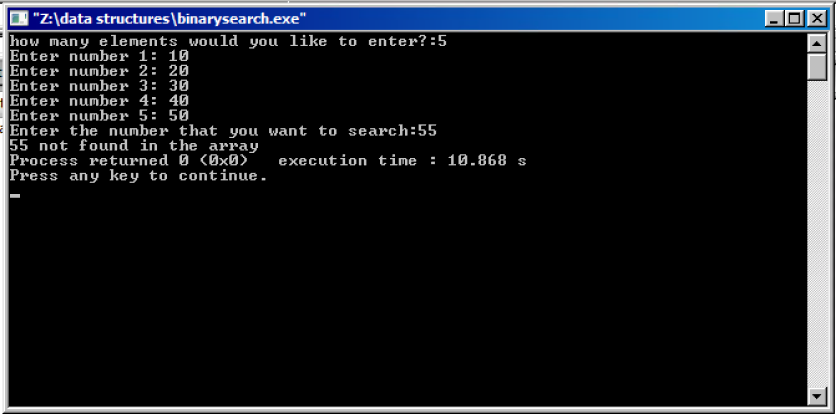
cout<<num<<" not found in the array";

}

return 0;

}

**Output:**



**Result:**

The program is executed and output is verified for valid inputs.

**Link:**

<https://youtu.be/EvZDulbxrhU>

|  |  |
| --- | --- |
| **Implementation of Quick and merge sort** | **Ex. No. 7** |
| **06-02-2019** |

**Aim :**

To implement quick sort.

**Algorithm :**

**Step1:** start the program.

**Step2:** in main function call quick sort function with array size and array and p value.

**Step3:** in quick sort function call partion function which used for array partion.

**Step4:** repeat step 3 until the array is sorted.

**Step5:** in partion function if(a[i]<=a[j]) then swap them at last swap j and i+1.

**Step6:** after this print the sorted array.

**Step7:** stop the program.

**Program:**

#include <iostream>

using namespace std;

void quick\_sort(int[],int,int);

int partition(int[],int,int);

int main()

{

int a[50],n=8,i;

cout<<"\nEnter array elements:"<<endl;

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

cin>>a[i];

quick\_sort(a,0,n-1);

cout<<"\nArray after sorting:";

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

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

return 0;

}

void quick\_sort(int a[],int l,int u)

{

int j;

if(l<u)

{

j=partition(a,l,u);

cout<<endl<<"After partition"<<endl;

for(int i=0;i<8;i++)

{

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

}

quick\_sort(a,l,j-1);

quick\_sort(a,j+1,u);

}

}

int partition(int a[],int l,int u)

{

int v,i,j,temp;

v=a[l];

i=l;

j=u+1;

do

{

do

i++;

while(a[i]<v&&i<=u);

do

j--;

while(v<a[j]);

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}while(i<j);

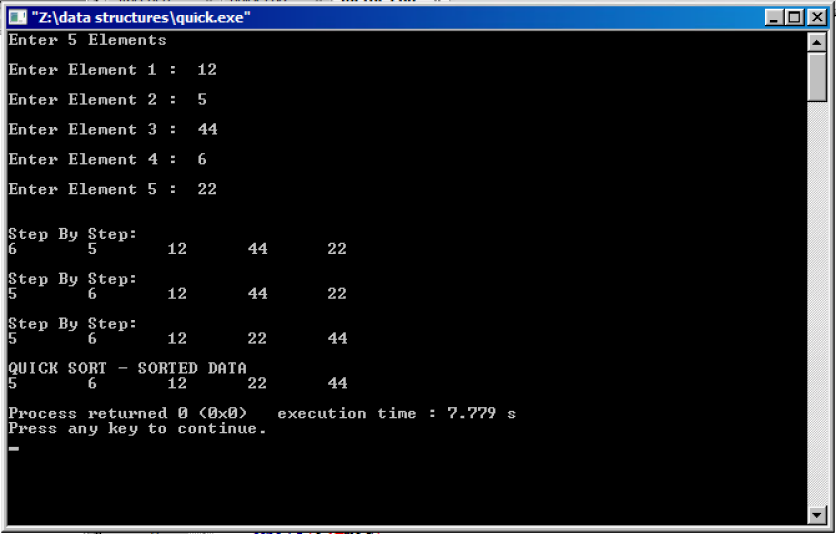
a[l]=a[j];

a[j]=v;

return(j);

}

**Output:**



**Aim:**

To implement merge sort.

**Algorithm:**

**Step1:** start the program.

**Step2:** enter the elements in the array.

**Step3:** call merge sort function from main with array size, lower and higherlimits as arguments.

**Step4:** in merge sort function call the function recursively until l<r.

**Step5:** from merge sort function call merge function to merge the array divided.

**Step6:** in merge sort function divide array into two parts and sort the two arrays and compare the elements in two arrays and put them in new array.

**Step7:** put the remaining elements after comparing in the array.

**Step8:** put the elements into the original array using for loop.

**Step9:** in main call display function to display sorted array.

**Step 10:** stop the program.

**Program:**

#include <iostream>

using namespace std;

void Merge(int \*a, int low, int high, int mid)

{

int i, j, k, temp[high-low+1];

i = low;

k = 0;

j = mid + 1;

while (i <= mid && j <= high)

{

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

{

temp[k] = a[i];

k++;

i++;

}

else

{

temp[k] = a[j];

k++;

j++;

}

}

while (i <= mid)

{

temp[k] = a[i];

k++;

i++;

}

while (j <= high)

{

temp[k] = a[j];

k++;

j++;

}

for (i = low; i <= high; i++)

{

a[i] = temp[i-low];

}

}

void MergeSort(int \*a, int low, int high)

{

int mid;

if (low < high)

{

mid=(low+high)/2;

MergeSort(a, low, mid);

MergeSort(a, mid+1, high);

Merge(a, low, high, mid);

}

}

int main()

{

int n, i;

cout<<"\nEnter the number of data element to be sorted: ";

cin>>n;

int arr[n];

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

{

cout<<"Enter element "<<i+1<<": ";

cin>>arr[i];

}

MergeSort(arr, 0, n-1);

cout<<"\nSorted Data ";

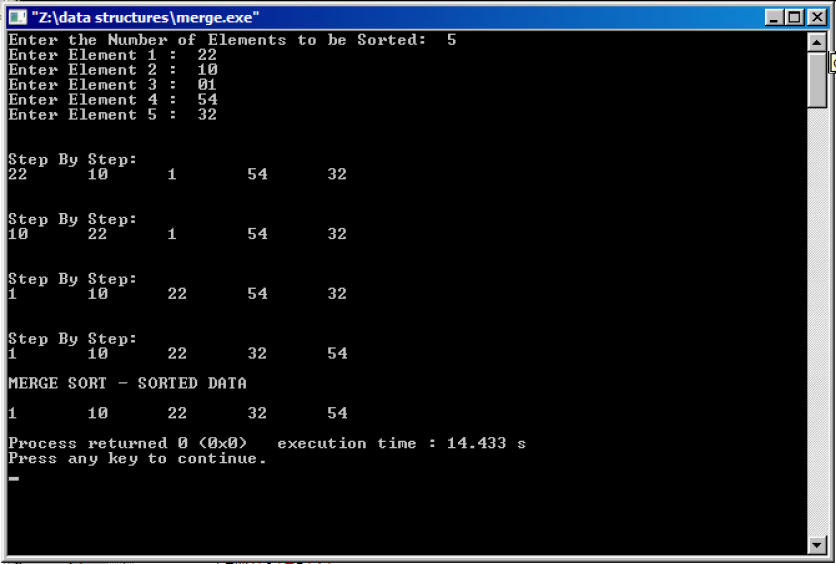
for (i = 0; i < n; i++)

cout<<"->"<<arr[i];

return 0;

}

**Output:**



**Result:**

The program is executed and output is verified for valid inputs.

**Link:** [**https://youtu.be/1n4fumFVZTs**](https://youtu.be/1n4fumFVZTs)

|  |  |
| --- | --- |
| **Hashing** | **Ex. No. 8** |
| **23-01-2019** |

**Aim :**

Implementation of Hashing.

**Algorithm:**

Step 1: Start the program

Step 2: Create an array of size n.

Step 3: Initially fill the full array with -1.

Step 4: In

1.addition

2.deletion

3.search

4.display.

Step 5: Enter 1 to Insert element and respective option to perform the operation.

Step 6: If the index of a number is already filled then insert element entered in next

free location.

Step 7: If the position is not free then Hashing table is full.

Step 8: If any element is deleted then replace index value with -1.

Step 9: Stop the program.

**Program:**

#include<iostream>

using namespace std;

bool full(int a[]){

int u=0;

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

if(a[i]==-1)

u++;

}

if(u==0){

return true;

}

else

return false;

}

bool empty(int a[]){

int k=0;

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

if(a[i]!=-1||a[i]!=-2)

k++;

}

if(k==0)

return true;

else

return false;

}

int hashFunction(int x){

return x%10;

}

void insert(int x,int a[]){

if(full(a)==true)

cout<<"The array is full\n";

else{

int p=hashFunction(x);

kbc:

if(a[p]==-1||a[p]==-2){

a[p]=x;

cout<<"The element inserted at position "<<p<<endl;

}

else{

if(p==9){

p=-1;

}

p+=1;

goto kbc;

}

}

}

int search(int x,int a[]){

int k=0;

if(empty(a)==true)

return -10;

else if(full(a)==true)

return -11;

else{

int p=hashFunction(x);

trs:

if(x==a[p])

return p;

else{

if(p==9){

p=0;

k++;}

else{

p++;

k++;

}

if(k!=10)

goto trs;

}

}

return -100;

}

void delet(int x,int a[]){

if(empty(a)==true){

cout<<"The array is empty\n";

}

else{

int p=search(x,a);

a[p]=-2;

cout<<"element deleted\n";

}

}

void display(int a[]){

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

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

}

cout<<endl;

}

int main()

{

int a[10];

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

a[i]=-1;

}

lab:

cout<<"Enter the choice\n";

cout<<"1.Insert\n2.Search\n3.Delete\n4.Display\n5.Exit\n";

int c;cin>>c;

if(c==1){

cout<<"Enter the number you want to insert\n";

int x;cin>>x;

insert(x,a);

goto lab;

}

else if(c==2){

cout<<"Enter the number to be searched\n";

int x;cin>>x;

int p=search(x,a);

if(p==-100)

{

cout<<"Element is not found";

}

else{

cout<<"Found at node: "<<p<<endl;

}

goto lab;

}

else if(c==3){

cout<<"Enter the number you want to delete\n";

int x;cin>>x;

delet(x,a);

goto lab;

}

else if(c==4){

display(a);

goto lab;

}

else if(c==5){}

else{

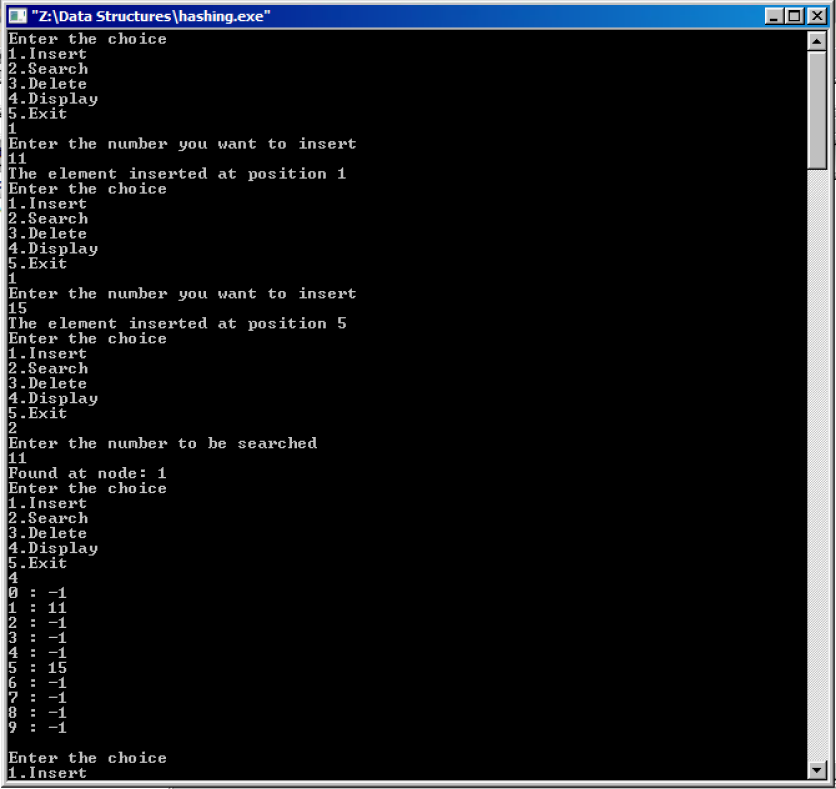
cout<<"Enter the correct choice\n";

goto lab;

}

}

**Output:**



**Result:**

The program is executed and output is verified for valid inputs.

**Link:**

[**https://youtu.be/T-IfNzDjIsU**](https://youtu.be/T-IfNzDjIsU)

|  |  |
| --- | --- |
| **Implementation of Binary Search Tree** | **Ex. No. 9** |
| 20-02-2019 |

**Aim:**

To write a program to implement binary search tree using linked lists.

**Description**

Write a brief description about the concepts implemented in the exercise.

**Algorithm:**

* Start the program
* Create a struct node with data left child address and right child address and create a root node
* In search() recursively go to the left child if key less than data or right child if key greater than data. if not found and node null then data not found
* In insert () first search for the data if it is not
* In display key first then children for pre order
* Display left child the key then right child for in order
* Display left child, right child then key for the post order traversal
* Create new and call the function accordingly to user’s input
* End the program

**Program:**

# include <iostream>

using namespace std;

struct node

{

int info;

struct node \*left;

struct node \*right;

}\*root;

class BST

{

public:

void insert(node \*, node \*);

void preorder(node \*);

void inorder(node \*);

void postorder(node \*);

void display(node \*, int);

BST()

{

root = NULL;

}

};

int main()

{

int choice, num;

BST bst;

node \*temp;

while (1)

{

cout<<"1.Insert Element "<<endl;

cout<<"2.Inorder Traversal"<<endl;

cout<<"3.Preorder Traversal"<<endl;

cout<<"4.Postorder Traversal"<<endl;

cout<<"5.Display"<<endl;

cout<<"6.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

temp = new node;

cout<<"Enter the number to be inserted : ";

cin>>temp->info;

bst.insert(root, temp);

break;

case 2:

cout<<"Inorder Traversal of BST:"<<endl;

bst.inorder(root);

cout<<endl;

break;

case 3:

cout<<"Preorder Traversal of BST:"<<endl;

bst.preorder(root);

cout<<endl;

break;

case 4:

cout<<"Postorder Traversal of BST:"<<endl;

bst.postorder(root);

cout<<endl;

break;

case 5:

cout<<"Display BST:"<<endl;

bst.display(root,1);

cout<<endl;

break;

case 6:

exit(1);

default:

cout<<"Wrong choice"<<endl;

}

}

}

void BST::insert(node \*tree, node \*newnode)

{

if (root == NULL)

{

root = new node;

root->info = newnode->info;

root->left = NULL;

root->right = NULL;

cout<<"Root Node is Added"<<endl;

return;

}

if (tree->info == newnode->info)

{

cout<<"Element already in the tree"<<endl;

return;

}

if (tree->info > newnode->info)

{

if (tree->left != NULL)

{

insert(tree->left, newnode);

}

else

{

tree->left = newnode;

(tree->left)->left = NULL;

(tree->left)->right = NULL;

cout<<"Node Added To Left"<<endl;

return;

}

}

else

{

if (tree->right != NULL)

{

insert(tree->right, newnode);

}

else

{

tree->right = newnode;

(tree->right)->left = NULL;

(tree->right)->right = NULL;

cout<<"Node Added To Right"<<endl;

return;

}

}

}

void BST::preorder(node \*ptr)

{

if (root == NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if (ptr != NULL)

{

cout<<ptr->info<<" ";

preorder(ptr->left);

preorder(ptr->right);

}

}

void BST::inorder(node \*ptr)

{

if (root == NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if (ptr != NULL)

{

inorder(ptr->left);

cout<<ptr->info<<" ";

inorder(ptr->right);

}

}

void BST::postorder(node \*ptr)

{

if (root == NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if (ptr != NULL)

{

postorder(ptr->left);

postorder(ptr->right);

cout<<ptr->info<<" ";

}

}

void BST::display(node \*ptr, int level)

{

int i;

if (ptr != NULL)

{

display(ptr->right, level+1);

cout<<endl;

if (ptr == root)

cout<<"Root->: ";

else

{

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

cout<<" ";

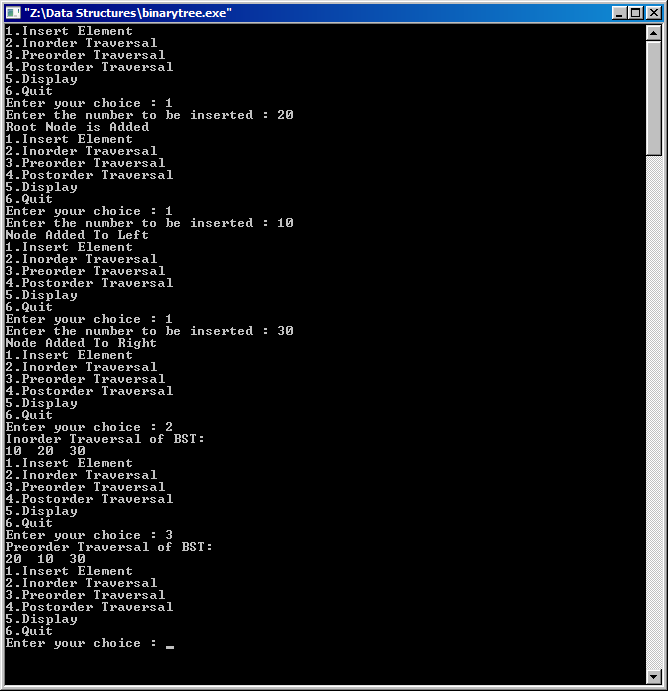
}

cout<<ptr->info;

display(ptr->left, level+1);

}}

**Input & Output:**



**Result:**

The given programs are implemented and the outputs are verified

**YouTube link:**[**https://youtu.be/lj6QxbqmQwg**](https://youtu.be/lj6QxbqmQwg)

|  |  |
| --- | --- |
| **Implementation of Graph Traversals** | **Ex. No. 10** |
| 13-03-2019 |

**Aim:**

To Implement the depth-first search using a stack data structure

**Algorithm**

* Start the program
* Declare the required variables such cost[10][10], I, j, k, n, stk [10], top ,v, visit[10],visited[10];
* In the main function get the data from the user which implements the graph.
* Now sort the graph from the initial node which is given by the user
* Mark the node as visited when it is adjacent to the parent node and then move to stack
* When any node is un-visited from array list then repeat the operation to go near that node
* End the program.

**Program:**

#include<iostream>

using namespace std;

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

int main()

{

int m;

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;

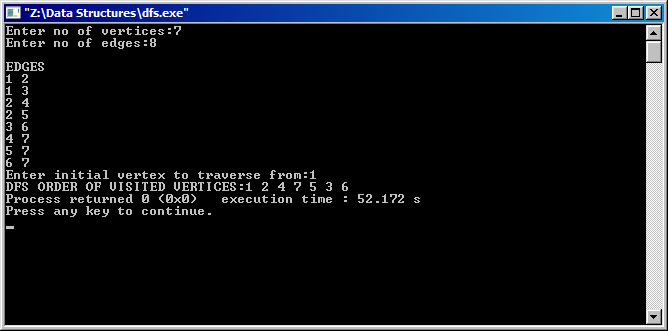
visited[v]=1;

}

return 0;

}

**Input & Output**



**Aim:**

To Implement the breadth-first search using a Queue data structure.

**Algorithm:**

* Start the program
* Declare the required variables cost[10][10], j, k, n, qu[10], front, rare, v, visit[10], visited[10];
* In the main function get the data from the user which implements the graph.
* Now sort the graph from the initial node which is given by the user
* Mark the nodes as visited when if the root node has a direct edge with any node and then move to queue
* When any node is un-visited from array list then repeat the operation to go near that node
* End the program.

**Program:**

#include<iostream>

using namespace std;

int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];

int main()

{

int m;

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 <<"Visitied vertices:";

cout <<v<<" ";

visited[v]=1;

k=1;

while(k<n)

{

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

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

{

visit[j]=1;

qu[rare++]=j;

}

v=qu[front++];

cout<<v <<" ";

k++;

visit[v]=0;

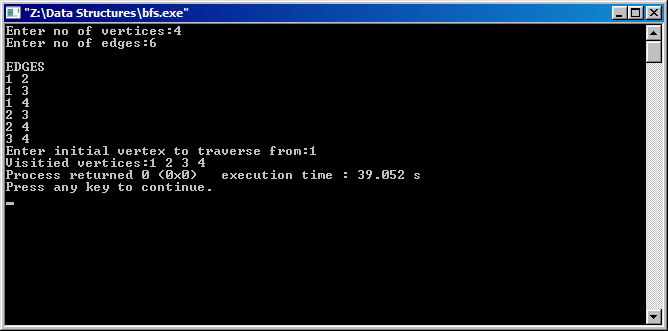
visited[v]=1;

}

return 0;

}

**Input & Output:**



**Result:**

The program is executed and output is verified for valid inputs.

**Link**: <https://youtu.be/t4n7qaW1TA0>