

DELHI TECHNOLOGICAL UNIVERSITY

(Department of Applied Mathematics)

Data Structures

(CS251)



SUBMITTED TO:

Dr. Aditya Kaushik

SUBMITTED BY:

Rahul Sharma

(2K18/MC/087)

Sno.	Objective	Date	Signature
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.			
10.			
11.			
12.			
13.			
14.			
15.			
16.			
17.			
18.			
19.			
20.			

DEPARTMENT OF APPLIED **MATHEMATICS**

Vision:

“To be a world class University through education, innovation and research for the service of humanity”

Mission:

- 1.To establish center of excellence in emerging areas of science, engineering, technology, management and allied areas.
- 2.To foster an ecosystem for incubation, product development, transfer of technology and entrepreneurship.
- 3.To create environment of collaboration, experimentation, imagination and creativity.
- 4.To developed human potential and analytical abilities, ethics and integrity.
5. To provide environment friendly, reasonable and sustainable solutions for local and global need.

DEPARTMENT OF APPLIED **MATHEMATICS**

Vision:

To emerge as a center of excellence and eminence by imparting futuristic technical education with solid mathematical background in keeping with global standard, making our students technologically and mathematically competent and ethically strong so that they can readily contribute to the rapid advancement of society and mankind.

Mission:

1. To achieve academic excellence through innovative teaching and learning practices.
2. To improve the research competent to address social needs.
3. To inculcate a culture that supports the reinforces ethical, professional behavior for a harmonious and prosperous society.
4. Strive to make students to understand, appreciate and gain mathematical skills and develop logic, so that they are able to contribute intelligently in decision making which characterizes our scientific and technological age.

PROGRAM 1(a) :- Linear search

CODE :

```
#include<iostream>
using namespace std;
int main()
{
    int array_number[1000];
    int size1,temp=0;
    cout<<"Enter the size of array <1000 : ";
    cin>>size1;
    cout<<"Enter the elements of array : ";
    for(int i=0;i<size1;i++)
    {
        cin>>array_number[i];

    }
    int element;

    char ch='y';
    while(ch=='y' || ch=='Y')
    {
        cout<<"Enter the element to be searched : ";
        cin>>element;
        for(int i=0;i<size1;i++)
        {
            if(array_number[i]==element)
            {
                cout<<"The position of element : "<< i+1<<endl;
                temp=1;
                break;
            }
        }
        if(temp==0)
        {
            cout<<"Entered element is not in the list "<<endl;
        }
        cout<<"If you want to continue enter Y else press N ";
        cin>>ch;
    }
}
```

```
}  
  
return 0;  
  
}
```

Output :-

Enter the size of array <1000 : 6

Enter the elements of array : 1 2 3 4 5 6

Enter the element to be searched : 4

The position of element : 4

If you want to continue enter Y else press N n

Process returned 0 (0x0) execution time : 2205.948 s

Press any key to continue.

PROGRAM 1(b) :- Binary Search

Code :

```
#include<iostream>
using namespace std;
int main()
{
    int array_number[1000];
    int size1,temp=0;
    int start,mid,en;
    cout<<"Enter the size of array <1000 : ";
    cin>>size1;
    cout<<"Enter the elements of array : ";
    for(int i=0;i<size1;i++)
    {
        cin>>array_number[i];

    }
    int element;

    char ch='y';
    while(ch=='y' || ch=='Y')
    {
        cout<<"Enter the element to be searched : ";
        cin>>element;
        start=0;
        en=size1-1;
        mid=(start+en)/2;
        while(start<=en)
        {
            if(array_number[mid]==element)
            {
                cout<<"Position of element : "<<mid+1<<endl;
                temp=1;
                break;
            }
            if(array_number[mid]>element)
            {
                en=mid-1;
            }
        }
    }
}
```

```

        mid=(start+en)/2;
    }
    if(array_number[mid]<element)
    {
        start=mid+1;
        mid=(start+en)/2;
    }
}
if(temp==0)
{
cout<<"Entered element is not in the list "<<endl;
}
cout<<"If you want to continue enter Y else press N ";
cin>>ch;
}

return 0;

}

```

Output :-

Enter the size of array <1000 : 8

Enter the elements of array : 1 2 3 5 7 8 9 10

Enter the element to be searched : 4

Entered element is not in the list

If you want to continue enter Y else press N y

Enter the element to be searched : 3

Position of element : 3

If you want to continue enter Y else press N n

Process returned 0 (0x0) execution time : 68.026 s

Press any key to continue.

PROGRAM 1(C) :- Binary Search Using Recursion.

Code :

```
#include<iostream>
using namespace std;
int array_number[1000];

int binary_search1(int start,inten,int element)
{
    if(start>en)
        return -1;

    int mid=(start+en)/2;
    if(array_number[mid]==element)
    {
        return mid+1;
    }
    if(array_number[mid]>element)
    {
        binary_search1(start,mid-1,element);
    }
    if(array_number[mid]<element)
    {
        binary_search1(mid+1,en,element);
    }
}

int main()
{
    int size1,temp=0;

    cout<<"Enter the size of array <1000 : ";
    cin>>size1;
    cout<<"Enter the elements of array : ";
    for(int i=0;i<size1;i++)
    {
        cin>>array_number[i];
    }
}
```

```

    }
    int element;

    char ch='y';
    while(ch=='y' || ch=='Y')
    {
        cout<<"Enter the element to be searched : ";
        cin>>element;
        temp=binary_search1(0,size1-1,element);
        if(temp==-1)
        {
            cout<<"Entered element is not in the list "<<endl;
        }
        else
        {
            cout<<"Position of the element : "<<temp<<endl;
        }
        cout<<"If you want to continue enter Y else press N ";
        cin>>ch;
    }

    return 0;
}

```

Output :-

Enter the size of array <1000 : 7

Enter the elements of array : 1 2 4 5 6 7 8

Enter the element to be searched : 5

Position of the element : 4

If you want to continue enter Y else press N n

Process returned 0 (0x0) execution time : 14.202 s

Press any key to continue.

CODE 2{A}

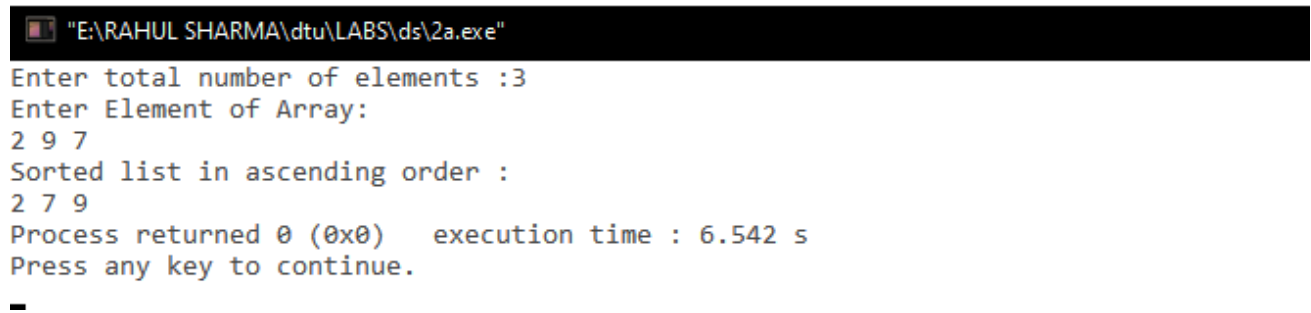
AIM: Write a program for Bubble Sort.

CODE:

```
#include<iostream>
using namespace std;

int main()
{
    int arr[50],n,i,j,temp;
    cout<<"Enter total number of elements :";
    cin>>n;
    cout<<"Enter Element of Array:\n";
    for(i=0; i<n; i++)
    {
        cin>>arr[i];
    }
    for(i=0; i<(n-1); i++)
    {
        for(j=0; j<(n-i-1); j++)
        {
            if(arr[j]>arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
    cout<<"Sorted list in ascending order :\n";
    for(i=0; i<n; i++)
    {
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

OUTPUT:



```
"E:\RAHUL SHARMA\dtu\LABS\ds\2a.exe"
Enter total number of elements :3
Enter Element of Array:
2 9 7
Sorted list in ascending order :
2 7 9
Process returned 0 (0x0)   execution time : 6.542 s
Press any key to continue.
-
```

CODE 2{B}

AIM: Write a program for Bubble Sort using Recursion.

CODE:

```
#include <iostream>
using namespace std;

void bubbleSort(int arr[], int n)
{
    if (n == 1)
        return;
    for (int i=0; i<n-1; i++)
        if (arr[i] > arr[i+1])
            swap(arr[i], arr[i+1]);
    bubbleSort(arr, n-1);
}

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

int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    cout<<"Sorted Array is... ";
    printArray(arr, n);
    return 0;
}
```

OUTPUT:



"E:\RAHUL SHARMA\dtu\LABS\ds\2a.exe"

```
Sorted Array is... 11 12 22 25 34 64 90
Process returned 0 (0x0)    execution time : 0.188 s
Press any key to continue.
```

■

CODE 3{A}

AIM: Write a code for Implementing Stack using Array.

CODE:

```
#include<iostream>

using namespace std;

#define MAX 1000

class Stack {
    int top;

public:
    int a[MAX];

    Stack() { top = -1; }
    bool push(int x);
    int pop();
    int peek();
    bool isEmpty();
};

bool Stack::push(int x)
{
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    }
    else {
        a[++top] = x;
        cout << x << " pushed into stack\n";
        return true;
    }
}

int Stack::pop()
{
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    }
    else {
        int x = a[top--];
        return x;
    }
}
```

```

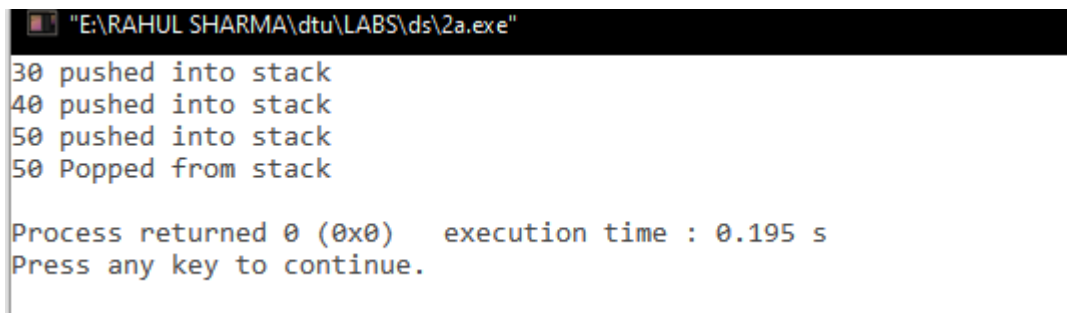
int Stack::peek()
{
    if (top < 0) {
        cout << "Stack is Empty";
        return 0;
    }
    else {
        int x = a[top];
        return x;
    }
}

bool Stack::isEmpty()
{
    return (top < 0);
}

int main()
{
    class Stack s;
    s.push(30);
    s.push(40);
    s.push(50);
    cout << s.pop() << " Popped from stack\n";
    return 0;
}

```

OUTPUT:



```

"E:\RAHUL SHARMA\dtu\LABS\ds\2a.exe"
30 pushed into stack
40 pushed into stack
50 pushed into stack
50 Popped from stack

Process returned 0 (0x0)   execution time : 0.195 s
Press any key to continue.

```

CODE 3{B}

AIM: Write a code for evaluating arithmetic expression by converting it from infix to postfix expression.

CODE:

```
#include<iostream>
#include<stack>
using namespace std;

int prec(char c)
{
    if(c == '^')
        return 3;
    else if(c == '*' || c == '/')
        return 2;
    else if(c == '+' || c == '-')
        return 1;
    else
        return -1;
}

void infixToPostfix(string s)
{
    stack<char> st;
    st.push('N');
    int l = s.length();
    string ns;
    for(int i = 0; i < l; i++)
    {
        if((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <=
'Z'))
            ns+=s[i];
        else if(s[i] == '(')
            st.push('(');
        else if(s[i] == ')')
        {
            while(st.top() != 'N' && st.top() != '(')
            {
                char c = st.top();
                st.pop();
                ns += c;
            }
            if(st.top() == '(')
            {
                char c = st.top();
                st.pop();
            }
        }
    }
    ns += st.top();
    st.pop();
    cout << ns << endl;
}
```

```

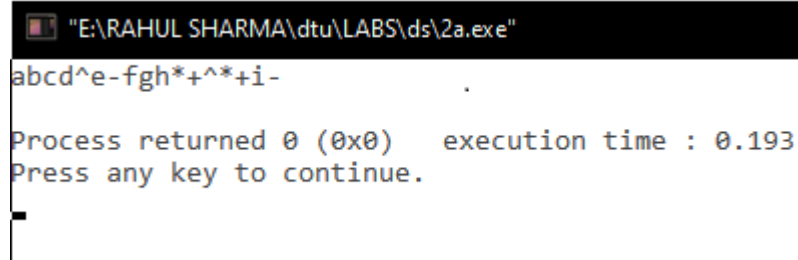
        }
        else{
            while(st.top() != 'N' && prec(s[i]) <=
prec(st.top()))
            {
                char c = st.top();
                st.pop();
                ns += c;
            }
            st.push(s[i]);
        }

    }
    while(st.top() != 'N')
    {
        char c = st.top();
        st.pop();
        ns += c;
    }

    cout << ns << endl;
}
int main()
{
    string exp = "a+b*(c^d-e)^(f+g*h)-i";
    infixToPostfix(exp);
    return 0;
}

```

OUTPUT:



```

"E:\RAHUL SHARMA\dtu\LABS\ds\2a.exe"
abcd^e-fgh*+^*+i-
.
Process returned 0 (0x0)   execution time : 0.193
Press any key to continue.

```


CODE 3{C}

AIM: Write a code to check for balanced parenthesis in an expression.

CODE:

```
#include<iostream>
#include<stack>
using namespace std;

bool areParanthesisBalanced(string expr)
{
    stack<char> s;
    char x;
    for (int i=0; i<expr.length(); i++)
    {
        if (expr[i]=='('||expr[i]=='['||expr[i]=='{')
        {
            s.push(expr[i]);
            continue;
        }
        if (s.empty())
            return false;

        switch (expr[i])
        {
            case ')':
                x = s.top();
                s.pop();
                if (x=='{' || x=='[')
                    return false;
                break;

            case '}':
                x = s.top();
                s.pop();
                if (x=='(' || x=='[')
                    return false;
                break;


            case ']':
                x = s.top();
                s.pop();
                if (x=='(' || x=='{')
                    return false;
                break;
        }
    }
    return (s.empty());
}
```

```
}

int main()
{
    string expr = "{()}";

    if (areParanthesisBalanced(expr))
        cout << "Balanced";
    else
        cout << "Not Balanced";
    return 0;
}
```

OUTPUT:

 "E:\RAHUL SHARMA\dtu\LABS\ds\2a.exe"

Balanced

Process returned 0 (0x0) execution time : 0.185 s

Press any key to continue.

CODE 4{A}

AIM: Write a code for Implementation of circular queue.

CODE:

```
#include<iostream>
using namespace std;

struct Queue
{
    int rear, front;
    int size;
    int *arr;
    Queue(int s)
    {
        front = rear = -1;
        size = s;
        arr = new int[s];
    }
    void enqueue(int value);
    int dequeue();
    void displayQueue();
};

void Queue::enqueue(int value)
{
    if ((front == 0 && rear == size-1) ||
        (rear == (front-1)%(size-1)))
    {
        printf("\nQueue is Full");
        return;
    }

    else if (front == -1)
    {
        front = rear = 0;
        arr[rear] = value;
    }

    else if (rear == size-1 && front != 0)
    {
        rear = 0;
        arr[rear] = value;
    }

    else
    {
        rear++;
    }
}
```

```

        arr[rear] = value;
    }
}

int Queue::deQueue()
{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return 1;
    }

    int data = arr[front];
    arr[front] = -1;
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else if (front == size-1)
        front = 0;
    else
        front++;

    return data;
}

void Queue::displayQueue()
{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return;
    }
    printf("\nElements in Circular Queue are: ");
    if (rear >= front)
    {
        for (int i = front; i <= rear; i++)
            printf("%d ", arr[i]);
    }
    else
    {
        for (int i = front; i < size; i++)
            printf("%d ", arr[i]);

        for (int i = 0; i <= rear; i++)
            printf("%d ", arr[i]);
    }
}

int main()
{

```

```
Queue q(5);
q.enqueue(14);
q.enqueue(22);
q.enqueue(13);
q.enqueue(-6);
q.displayQueue();
printf("\nDeleted value = %d", q.dequeue());
printf("\nDeleted value = %d", q.dequeue());

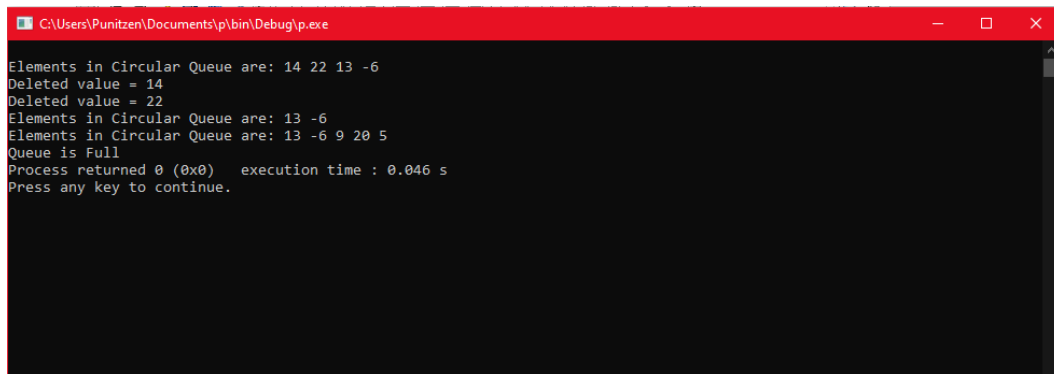
q.displayQueue();

q.enqueue(9);
q.enqueue(20);
q.enqueue(5);

q.displayQueue();

q.enqueue(20);
return 0;
}
```

OUTPUT:



```
C:\Users\Punitzen\Documents\p\bin\Debug\p.exe
Elements in Circular Queue are: 14 22 13 -6
Deleted value = 14
Deleted value = 22
Elements in Circular Queue are: 13 -6
Elements in Circular Queue are: 13 -6 9 20 5
Queue is Full
Process returned 0 (0x0) execution time : 0.046 s
Press any key to continue.
```

CODE 4{B}

AIM: Write a code for Implementation of priority queue.

CODE:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int data;
    int priority;

    struct node* next;
} Node;

Node* newNode(int d, int p)
{
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = d;
    temp->priority = p;
    temp->next = NULL;

    return temp;
}

int peek(Node** head)
{
    return (*head)->data;
}

void pop(Node** head)
{
    Node* temp = *head;
    (*head) = (*head)->next;
    free(temp);
}

void push(Node** head, int d, int p)
{
    Node* start = (*head);
    Node* temp = newNode(d, p);
    if ((*head)->priority > p)
    {
        temp->next = *head;
        (*head) = temp;
    }
    else {
        while (start->next != NULL &&
            start->next->priority < p) {
```

```

        start = start->next;
    }
    temp->next = start->next;
    start->next = temp;
}

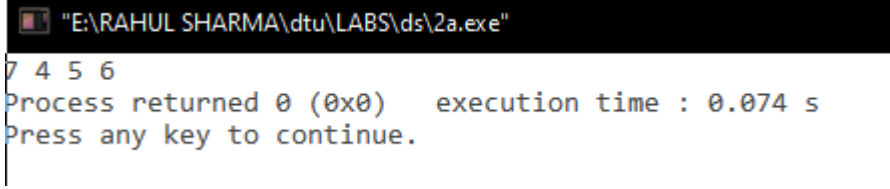
int isEmpty(Node** head)
{
    return (*head) == NULL;
}

int main()
{
    Node* pq = newNode(4, 1);
    push(&pq, 5, 2);
    push(&pq, 6, 3);
    push(&pq, 7, 0);

    while (!isEmpty(&pq)) {
        printf("%d ", peek(&pq));
        pop(&pq);
    }
    return 0;
}

```

OUTPUT:



```

7 4 5 6
Process returned 0 (0x0)   execution time : 0.074 s
Press any key to continue.

```

CODE 5(b)

AIM: Write a code for Implementation of Doubly Linked List.

Code:

```
#include<iostream>
using namespace std;
class Node
{
public:
int data;
Node* next;
Node* prev;
};
void deleteNode(Node** head_ref, Node* del)
{
if (*head_ref == NULL || del == NULL)
return;
if (*head_ref == del)
*head_ref = del->next;
if (del->next != NULL)
del->next->prev = del->prev;
if (del->prev != NULL)
del->prev->next = del->next;
free(del);
return;
}
bool search(Node* head, int x)
{
Node* current = head;
while (current != NULL)
{
if (current->data == x)
return true;
current = current->next;
}
return false;
}
void push(Node** head_ref, int new_data)
{
Node* new_node = new Node();
new_node->data = new_data;
new_node->prev = NULL;
new_node->next = (*head_ref);
if ((*head_ref) != NULL)
(*head_ref)->prev = new_node;
(*head_ref) = new_node;
}
```



```

void printList(Node* node)
{
while (node != NULL)
{
    cout << node->data << " ";
    node = node->next;
}
}
int main()
{
Node* head = NULL;
push(&head, 2);
push(&head, 4);
push(&head, 8);
push(&head, 10);
cout << "Original Linked list ";
printList(head);
deleteNode(&head, head);
deleteNode(&head, head->next);
deleteNode(&head, head->next);
cout << "\nModified Linked list ";
printList(head);
    search(head, 2)? cout<<"Yes" : cout<<"No";
return 0;
}

```

Output:

Output

```

Original Linked list 10 8 4 2
Modified Linked list 8 No

```

CODE 5(c)

AIM: Write a code for Implementation of Circular Linked List.

Code:

```
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
int data;
Node* next;
};
bool search(struct Node* head, int x)
{ if (head == NULL)
return false;
if (head->data == x)
return true;
return search(head->next, x);
}
void push(Node** head_ref, int data)
{ Node* ptr1 = new Node();
ptr1->data = data;
ptr1->next = *head_ref;
if (*head_ref != NULL) {
Node* temp = *head_ref;
while (temp->next != *head_ref)
temp = temp->next;
temp->next = ptr1;
}
else
ptr1->next = ptr1;
*head_ref = ptr1;
}
void printList(Node* head)
{
Node* temp = head;
if (head != NULL) {
do { cout << temp->data << " ";
temp = temp->next;
} while (temp != head);
}
cout << endl;
}
void deleteNode(Node** head, int key)
{ if (*head == NULL)
return;
if ((*head)->data==key && (*head)->next==*head)
{
```

```

        free(*head);
        *head=NULL;
    }
    Node *last=*head,*d;
    if((*head)->data==key) {
        while(last->next!=*head)
            last=last->next;
        last->next=(*head)->next;
        free(*head);
        *head=last->next;
    }
    while(last->next!=*head&&last->next->data!=key) {
        last=last->next;
    }
    if(last->next->data==key) {
        d=last->next;
        last->next=d->next;
        free(d);
    }
    else
        cout<<"no such key found";
}
int main()
{ Node* head = NULL;
  push(&head, 2);
  push(&head, 5);
  push(&head, 7);
  push(&head, 8);
  push(&head, 10);
  printList(head);
  deleteNode(&head, 7);
  printList(head);
  search(head,8)? cout<<"Yes" : cout<<"No";
  return 0;
}

```

Output:

Output

10 8 7 5 2

10 8 5 2

Yes

PROGRAM 6:- IMPLEMENT STACKS USING LINKED LIST

CODE:-

```
#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;
}

```

OUTPUT:-

```

1) Push in stack
2) Pop from stack
3) Display stack
4) Exit

Enter choice: 1
Enter value to be pushed: 2
Enter choice: 1
Enter value to be pushed: 6
Enter choice: 1
Enter value to be pushed: 8
Enter choice: 1
Enter value to be pushed: 7
Enter choice: 2
The popped element is 7
Enter choice: 3
Stack elements are:8 6 2
Enter choice: 5
Invalid Choice
Enter choice: 4
Exit

```

PROGRAM 7:- IMPLEMENT QUEUE USING LINKED LIST

CODE:-

```
#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:-

```

1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice : 1
Insert the element in queue : 4
Enter your choice : 1
Insert the element in queue : 3
Enter your choice : 1
Insert the element in queue : 5
Enter your choice : 2
Element deleted from queue is : 4
Enter your choice : 3
Queue elements are : 3 5
Enter your choice : 7
Invalid choice
Enter your choice : 4
Exit

```

PROGRAM 8-A:-INSERTION, DELETION AND TRAVERSAL IN BST

CODE:-

```
# include <stdio.h>
```

```
# include <malloc.h>
```

```
struct node
```

```
{
```

```
    int info;
```

```
    struct node *lchild;
```

```
    struct node *rchild;
```

```
}*root;
```

```
void find(int item,struct node **par,struct node **loc)
```

```
{
```

```
    struct node *ptr,*ptrsave;
```

```
    if(root==NULL) /*tree empty*/
```

```
    {
```

```
        *loc=NULL;
```

```
        *par=NULL;
```

```
        return;
```

```
    }
```

```
    if(item==root->info) /*item is at root*/
```

```
    {
```

```
        *loc=root;
```

```
        *par=NULL;
```

```
        return;
```

```
    }
```

```

/*Initialize ptr and ptrsave*/
if(item<root->info)
    ptr=root->lchild;
else
    ptr=root->rchild;
ptrsave=root;

while(ptr!=NULL)
{
    if(item==ptr->info)
    {
        *loc=ptr;
        *par=ptrsave;
        return;
    }
    ptrsave=ptr;
    if(item<ptr->info)
        ptr=ptr->lchild;
    else
        ptr=ptr->rchild;
}/*End of while */
*loc=NULL; /*item not found*/
*par=ptrsave;
}/*End of find()*/

```

```

void insert(int item)
{
    struct node *tmp,*parent,*location;
    find(item,&parent,&location);
    if(location!=NULL)
    {
        printf("Item already present");
        return;
    }
}

```

```

tmp=(struct node *)malloc(sizeof(struct node));

tmp->info=item;

tmp->lchild=NULL;

tmp->rchild=NULL;


if(parent==NULL)

    root=tmp;

else

    if(item<parent->info)

        parent->lchild=tmp;

    else

        parent->rchild=tmp;

}/*End of insert()*/

```

```

void case_a(struct node *par,struct node *loc )
{

    if(par==NULL) /*item to be deleted is root node*/

        root=NULL;

    else

        if(loc==par->lchild)

            par->lchild=NULL;

        else

            par->rchild=NULL;

}/*End of case_a()*/

```

```

void case_b(struct node *par,struct node *loc)
{

    struct node *child;


    /*Initialize child*/

    if(loc->lchild!=NULL) /*item to be deleted has lchild */

        child=loc->lchild;

```

```

else          /*item to be deleted has rchild */
    child=loc->rchild;

if(par==NULL ) /*Item to be deleted is root node*/
    root=child;
else
    if( loc==par->lchild) /*item is lchild of its parent*/
        par->lchild=child;
    else          /*item is rchild of its parent*/
        par->rchild=child;

}/*End of case_b()*/

```

```

void case_c(struct node *par,struct node *loc)
{
    struct node *ptr,*ptrsave,*suc,*parsuc;

    /*Find inorder successor and its parent*/
    ptrsave=loc;
    ptr=loc->rchild;
    while(ptr->lchild!=NULL)
    {
        ptrsave=ptr;
        ptr=ptr->lchild;
    }
    suc=ptr;
    parsuc=ptrsave;

    if(suc->lchild==NULL && suc->rchild==NULL)
        case_a(parsuc,suc);
    else
        case_b(parsuc,suc);

    if(par==NULL) /*if item to be deleted is root node */

```

```

        root=suc;
else
    if(loc==par->lchild)
        par->lchild=suc;
    else
        par->rchild=suc;

    suc->lchild=loc->lchild;
    suc->rchild=loc->rchild;
}/*End of case_c()*/
int del(int item)
{
    struct node *parent,*location;
    if(root==NULL)
    {
        printf("Tree empty");
        return 0;
    }

    find(item,&parent,&location);
    if(location==NULL)
    {
        printf("Item not present in tree");
        return 0;
    }

    if(location->lchild==NULL && location->rchild==NULL)
        case_a(parent,location);
    if(location->lchild!=NULL && location->rchild==NULL)
        case_b(parent,location);
    if(location->lchild==NULL && location->rchild!=NULL)
        case_b(parent,location);
    if(location->lchild!=NULL && location->rchild!=NULL)

```

```
        case_c(parent,location);

    free(location);

}/*End of del()*/
```

```
int preorder(struct node *ptr)
{
    if(root==NULL)
    {
        printf("Tree is empty");
        return 0;
    }
    if(ptr!=NULL)
    {
        printf("%d ",ptr->info);
        preorder(ptr->lchild);
        preorder(ptr->rchild);
    }
}/*End of preorder()*/
```

```
void inorder(struct node *ptr)
{
    if(root==NULL)
    {
        printf("Tree is empty");
        return;
    }
    if(ptr!=NULL)
    {
        inorder(ptr->lchild);
        printf("%d ",ptr->info);
        inorder(ptr->rchild);
    }
}/*End of inorder()*/
```

```

void postorder(struct node *ptr)
{
    if(root==NULL)
    {
        printf("Tree is empty");
        return;
    }
    if(ptr!=NULL)
    {
        postorder(ptr->lchild);
        postorder(ptr->rchild);
        printf("%d ",ptr->info);
    }
}
/*End of postorder()*/

```

```

void display(struct node *ptr,int level)
{
    int i;
    if ( ptr!=NULL )
    {
        display(ptr->rchild, level+1);
        printf("\n");
        for (i = 0; i < level; i++)
            printf("  ");
        printf("%d", ptr->info);
        display(ptr->lchild, level+1);
    }
}
/*End of display()*/

```

```

main()
{
    int choice,num;
    root=NULL;

```



```
while(1)
{
    printf("\n");
    printf("1.Insert\n");
    printf("2.Delete\n");
    printf("3.Inorder Traversal\n");
    printf("4.Preorder Traversal\n");
    printf("5.Postorder Traversal\n");
    printf("6.Display\n");
    printf("7.Quit\n");
    printf("Enter your choice : ");
    scanf("%d",&choice);

    switch(choice)
    {
        case 1:
            printf("Enter the number to be inserted : ");
            scanf("%d",&num);
            insert(num);
            break;

        case 2:
            printf("Enter the number to be deleted : ");
            scanf("%d",&num);
            del(num);
            break;

        case 3:
            inorder(root);
            break;

        case 4:
            preorder(root);
            break;

        case 5:
            postorder(root);
```

```

                break;

            case 6:

                display(root,1);

                break;

            case 7:

                break;

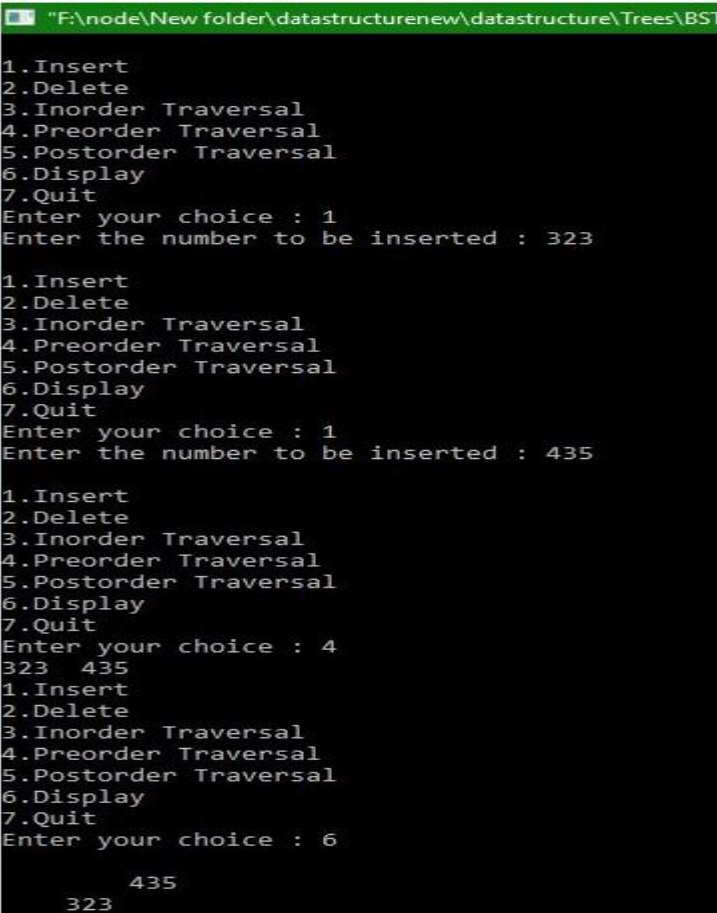
            default:

                printf("Wrong choice\n");

        }

```

OUTPUT:-



```

F:\node\New folder\datastructurenew\datastructure\Trees\BST
1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 323

1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 435

1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 4
323 435

1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6

435
323

```

PROGRAM 9:-SORTING-INSERTION SORT, MERGESORT, QUICKSORT

CODE:-1 (INSERTION SORT)

```
#include<iostream>

using namespace std;

void display(int *array, int size) {

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

        cout << array[i] << " ";

    cout << endl;

}

void insertionSort(int *array, int size) {

    int key, j;

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

        key = array[i]; //take value

        j = i;

        while(j > 0 && array[j-1]>key) {

            array[j] = array[j-1];

            j--;

        }

        array[j] = key; //insert in right place

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n]; //create an array with given number of elements

    cout << "Enter elements:" << endl;

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

        cin >> arr[i];

    }

}
```

```

cout << "Array before Sorting: ";

display(arr, n);

insertionSort(arr, n);

cout << "Array after Sorting: ";

display(arr, n);
}

```

CODE:-2 (MERGE SORT)

```

#include<iostream>

using namespace std;

void swapping(int &a, int &b) {    //swap the content of a and b

    int temp;

    temp = a;

    a = b;

    b = temp;
}

void display(int *array, int size) {

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

        cout << array[i] << " ";

    cout << endl;
}

void merge(int *array, int l, int m, int r) {

    int i, j, k, nl, nr;

    //size of left and right sub-arrays

    nl = m-l+1; nr = r-m;

    int larr[nl], rarr[nr];

    //fill left and right sub-arrays

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

        larr[i] = array[l+i];

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

        rarr[j] = array[m+1+j];

    i = 0; j = 0; k = l;

    //marge temp arrays to real array

```

```

while(i < nl && j<nr) {
    if(larr[i] <= rarr[j]) {
        array[k] = larr[i];
        i++;
    }else{
        array[k] = rarr[j];
        j++;
    }
    k++;
}

while(i<nl) {    //extra element in left array
    array[k] = larr[i];
    i++; k++;
}

while(j<nr) {    //extra element in right array
    array[k] = rarr[j];
    j++; k++;
}
}

void mergeSort(int *array, int l, int r) {
    int m;
    if(l < r) {
        int m = l+(r-l)/2;
        // Sort first and second arrays
        mergeSort(array, l, m);
        mergeSort(array, m+1, r);
        merge(array, l, m, r);
    }
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;

```

```

int arr[n]; //create an array with given number of elements

cout << "Enter elements:" << endl;

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

    cin >> arr[i];

}

cout << "Array before Sorting: ";

display(arr, n);

mergeSort(arr, 0, n-1); // (n-1) for last index

cout << "Array after Sorting: ";

display(arr, n);

}

```

CODE:- 3 (QUICK SORT)

```

#include<iostream>

#include<cstdlib>

using namespace std;

void swap(int *a, int *b) {

    int temp;

    temp = *a;

    *a = *b;

    *b = temp;

}

int Partition(int a[], int l, int h) {

    int pivot, index, i;

    index = l;

    pivot = h;

    for(i = l; i < h; i++) {

        if(a[i] < a[pivot]) {

            swap(&a[i], &a[index]);

            index++;

        }

    }

    swap(&a[pivot], &a[index]);

    return index;

}

```

```

    }
}

swap(&a[pivot], &a[index]);

return index;
}

int RandomPivotPartition(int a[], int l, int h) {
    int pvt, n, temp;

    n = rand();

    pvt = l + n%(h-l+1);

    swap(&a[h], &a[pvt]);

    return Partition(a, l, h);
}

int QuickSort(int a[], int l, int h) {
    int pindex;

    if(l < h) {
        pindex = RandomPivotPartition(a, l, h);

        QuickSort(a, l, pindex-1);

        QuickSort(a, pindex+1, h);
    }

    return 0;
}

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];
    }

    QuickSort(arr, 0, n-1);

    cout<<"\nSorted Data ";

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

```

```
    cout<<"-"<<arr[i];  
  
    return 0;  
}
```

OUTPUT:-

```
Enter the number of elements: 6  
Enter elements:  
14 20 78 98 20 45  
Array before Sorting: 14 20 78 98 20 45  
Array after Sorting: 14 20 20 45 78 98
```


PROGRAM 10:-DEPTH FIRST TRAVERSAL AND BREADTH FIRST TRAVERSAL

CODE:-1(DEPTH FIRST TRAVERSAL)

```
#include<iostream>

#include<list>

using namespace std;

class Graph
{
    int V;

    list<int> *adj;

    void DFSUtil(int v, bool visited[]);

public:
    Graph(int V);

    void addEdge(int v, int w);

    void DFS(int v);

};

Graph::Graph(int V)
{
    this->V = V;

    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}

void Graph::DFSUtil(int v, bool visited[])
{
    visited[v] = true;
```

```

    cout << v << " ";

    list<int>::iterator i;

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

        if (!visited[*i])

            DFSUtil(*i, visited);
}

void Graph::DFS(int v)
{
    bool *visited = new bool[V];

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

        visited[i] = false;

    DFSUtil(v, visited)
}

void Graph::BFS(int s)
{
    bool *visited = new bool[V];

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

        visited[i] = false;

    list<int> queue;

    visited[s] = true;

    queue.push_back(s);

    list<int>::iterator i;

    while(!queue.empty())
    {
        s = queue.front();

        cout << s << " ";

        queue.pop_front();

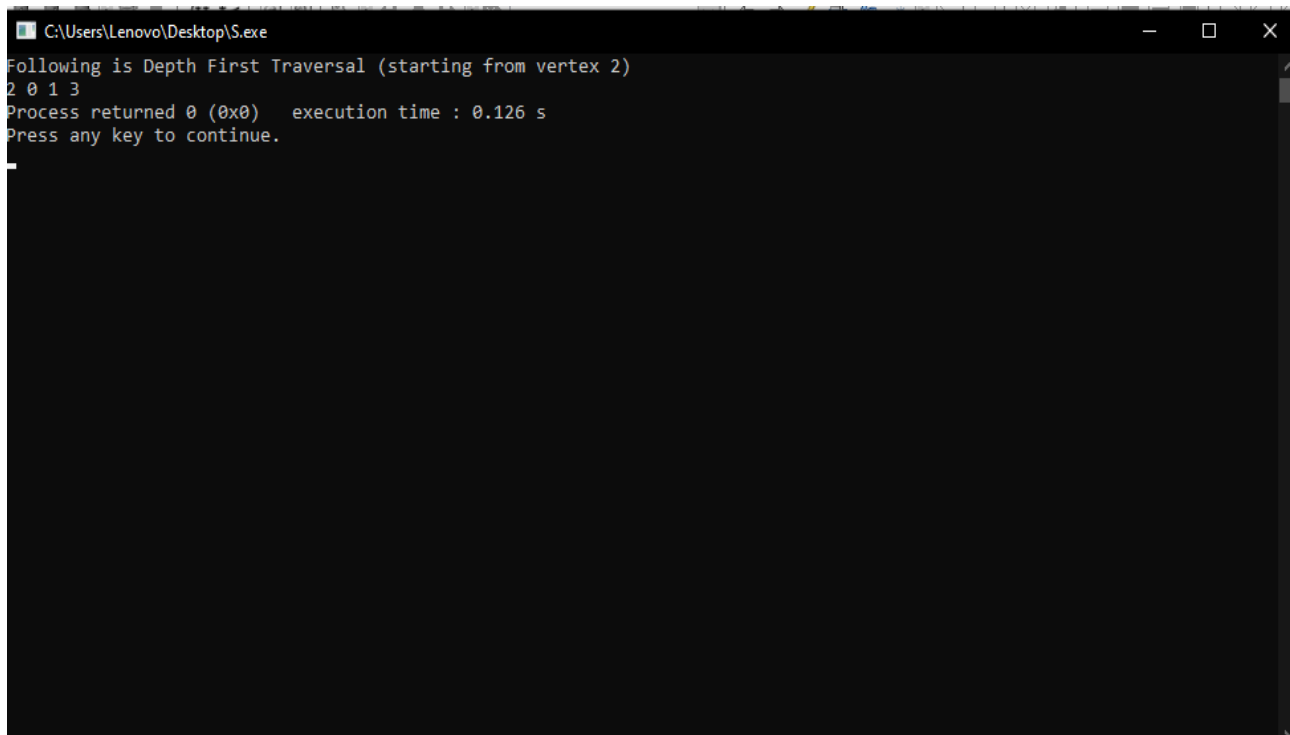
        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;

                queue.push_back(*i);
            }
        }
    }
}

```

```
    }  
    }  
}  
  
int main()  
{  
    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 Depth First Traversal"  
        << " (starting from vertex 2) \n";  
    g.DFS(2);  
}
```

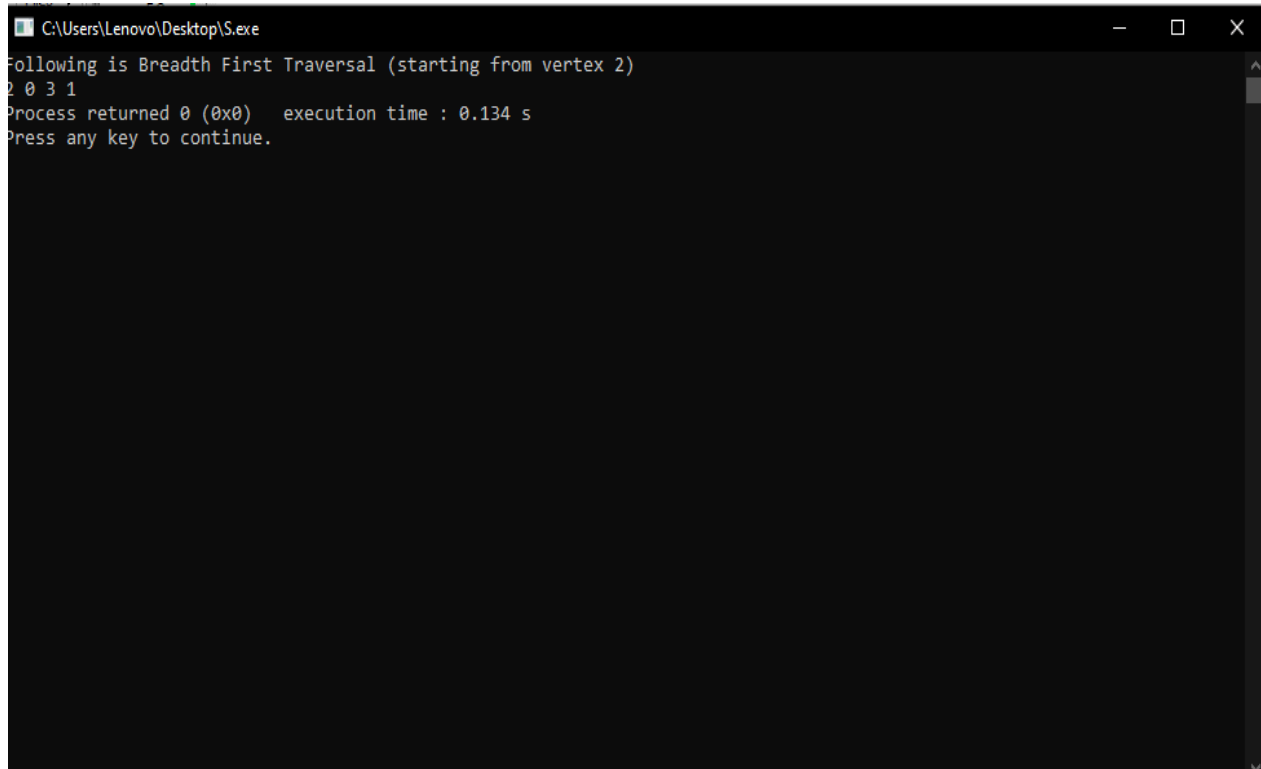
OUTPUT:-(DFS)



The screenshot shows a Windows command prompt window with the title bar "C:\Users\Lenovo\Desktop\S.exe". The output of the program is displayed in the console:

```
Following is Depth First Traversal (starting from vertex 2)  
2 0 1 3  
Process returned 0 (0x0)   execution time : 0.126 s  
Press any key to continue.
```

OUTPUT:-(BFS)



A screenshot of a Windows command prompt window. The title bar at the top shows the file path "C:\Users\Lenovo\Desktop\S.exe" and standard window controls (minimize, maximize, close). The command prompt displays the following text: "Following is Breadth First Traversal (starting from vertex 2)", "2 0 3 1", "Process returned 0 (0x0) execution time : 0.134 s", and "Press any key to continue." The text is in a light blue color on a black background.

```
C:\Users\Lenovo\Desktop\S.exe
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
Process returned 0 (0x0) execution time : 0.134 s
Press any key to continue.
```

CODE 5(a)

AIM: Write a code for Implementation of Singly Linked List.

Code:

```
#include <iostream>
using namespace std;
struct Node{
    int data;
    Node *next;
};
struct Node *head=NULL;
void insertNode(int n){
    struct Node *newNode=new Node;
    newNode->data=n;
    newNode->next=head;
    head=newNode;
}
bool search(Node* head, int x)
{
    Node* current = head;
    while (current != NULL)
    {
        if (current->data == x)
            return true;
        current = current->next;
    }
    return false;
}
void display(){
    if(head==NULL){
        cout<<"List is empty!"<<endl;
        return;
    }
    struct Node *temp=head;
    while(temp!=NULL){
        cout<<temp->data<<" ";
        temp=temp->next;
    }
    cout<<endl;
}
void deleteItem(){
    if(head==NULL){
        cout<<"List is empty!"<<endl;
        return;
    }
    cout<<head->data<<" is removed."<<endl;
    head=head->next;
```

```
}  
int main(){  
    insertNode(10);  
    insertNode(20);  
    insertNode(30);  
    insertNode(40);  
    insertNode(50);  
    display();  
    deleteItem();  
    search(head, 20)? cout<<"Yes" : cout<<"No";  
    display();  
    return 0;  
}
```

Output:

Output

50 40 30 20 10

50 is removed.

Yes40 30 20 10