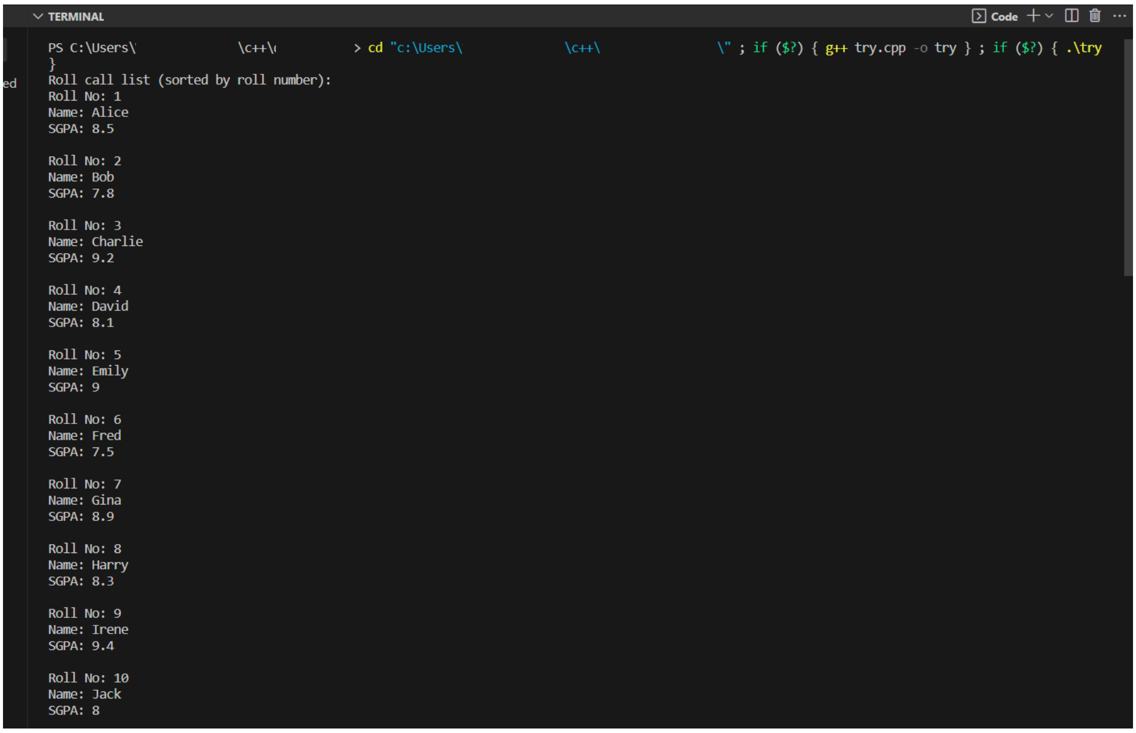
Assignment no 1

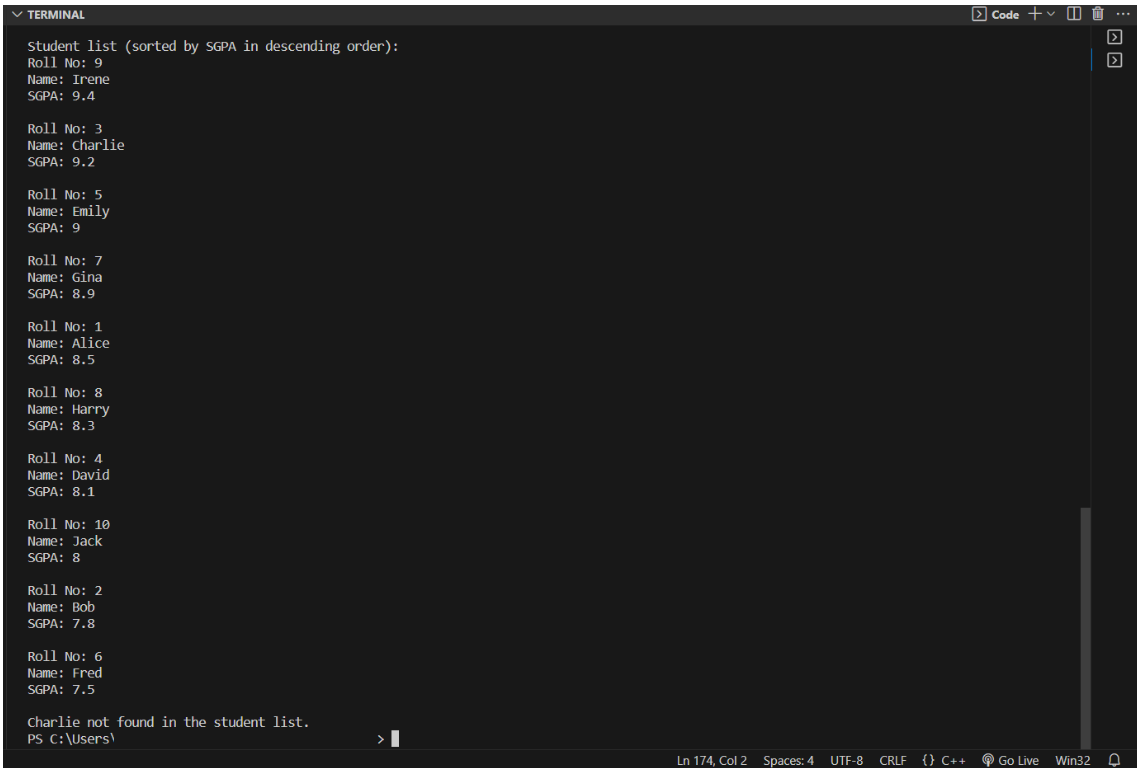
```
C++
#include
<iostream>
#include <vector>
using namespace std;

class Student {
public:
int rollNo;
     string name;
float SGPA;

     Student(int rollNo, string name, float SGPA) {
     this->rollNo = rollNo;
     this->name = name;
     this->SGPA = SGPA;
     }

     void prints() const {
               cout << "Roll No: " << rollNo << endl;
     cout << "Name: " << name << endl;
     cout << "SGPA: " << SGPA << endl;
     }
};

void bubbleSortByRollNo(vector<Student>& students) {
int n = students.size();
for(inti=0;i<n-1;i++){
for(intj=0;j<n-i-1;j++){
                          if (students[j].rollNo > students[j + 1].rollNo) {
swap(students[j], students[j + 1]);
                 }
           }
     }
}

void insertionSortByName(vector<Student>& students) {
for (int i = 1; i < students.size(); i++) {
Student key = students[i];
intj=i-1;

           while (j >= 0 && students[j].name > key.name) {
           students[j + 1] = students[j];
           j--;
           }

           students[j + 1] = key;
     }
```

```
}

int partition(vector<Student>& students, int low, int high) {
Student pivot = students[high];
int i = (low - 1);

    for (int j = low; j < high; j++) {
          if (students[j].SGPA > pivot.SGPA) {
    i++;
    swap(students[i], students[j]);
        }
    }

    swap(students[i  +  1],  students[high]);
    return (i + 1);
}

void quickSortBySGPA(vector<Student>& students, int low, int high) {
if (low < high) {
int pivot = partition(students, low, high);

        quickSortBySGPA(students, low, pivot - 1);
        quickSortBySGPA(students, pivot + 1, high);
    }
}

bool binarySearchByName(vector<Student>& students, string name) {
int low = 0;
int high = students.size() - 1;

    while (low <= high) {
           int mid = low + (high - low) / 2;

        if (students[mid].name == name) {
        return true;
        } else if (students[mid].name < name) {
        low = mid + 1;
        } else {
        high = mid - 1;
        }
    }

    return false;
}

int main() {
vector<Student> students = {
Student(1, "Alice", 8.5),
Student(2, "Bob", 7.8),
           Student(3, "Charlie", 9.2),
Student(4, "David", 8.1),
            Student(5, "Emily", 9.0),
            Student(6, "Fred", 7.5),
            Student(7, "Gina", 8.9),
            Student(8, "Harry", 8.3),
            Student(9, "Irene", 9.4),
            Student(10, "Jack", 8.0)
      };

      // Task a: Design a roll call list using Bubble Sort
      cout  <<  "Roll  call  list  (sorted  by  roll  number):"  <<  endl;
      bubbleSortByRollNo(students);
      for (const Student& student : students) {
      student.prints();
      cout << endl;
      }

// Task a: Design a roll call list using Bubble Sort
cout << "Roll call list (sorted by roll number):" << endl;
bubbleSortByRollNo(students); // Sort students by roll number

for (const Student& student : students) {
           student.prints(); // Use 'student' here
cout << endl;
}

// Task b: Sort students alphabetically by name using Insertion Sort
cout << "Student list (sorted alphabetically by name):" << endl;
insertionSortByName(students); // Sort students by name

for (const Student& student : students) {
           student.prints(); // Use 'student' here
cout << endl;
}

// Task c: Sort students by SGPA (descending order) using Quick Sort
cout << "Student list (sorted by SGPA in descending order):" << endl;
quickSortBySGPA(students, 0, students.size() - 1); // Sort students by
SGPA

for (const Student &student : students) {
           student.prints(); // Use 'student' here
cout << endl;
}

// Task d: Search for a student by name
string searchName = "Charlie";
bool found = binarySearchByName(students, searchName);

if (found) {
          cout << searchName << " found in the student list." << endl;
} else {
    cout << searchName << " not found in the student list." << endl;
}

return 0;

}
```

output -



```
PS C:\Users\            \c++\          > cd "c:\Users\            \c++\              \" ; if ($?) { g++ try.cpp -o try } ; if ($?) { .\try
}
Roll call list (sorted by roll number):
Roll No: 1
Name: Alice
SGPA: 8.5

Roll No: 2
Name: Bob
SGPA: 7.8

Roll No: 3
Name: Charlie
SGPA: 9.2

Roll No: 4
Name: David
SGPA: 8.1

Roll No: 5
Name: Emily
SGPA: 9

Roll No: 6
Name: Fred
SGPA: 7.5

Roll No: 7
Name: Gina
SGPA: 8.9

Roll No: 8
Name: Harry
SGPA: 8.3

Roll No: 9
Name: Irene
SGPA: 9.4

Roll No: 10
Name: Jack
SGPA: 8
```



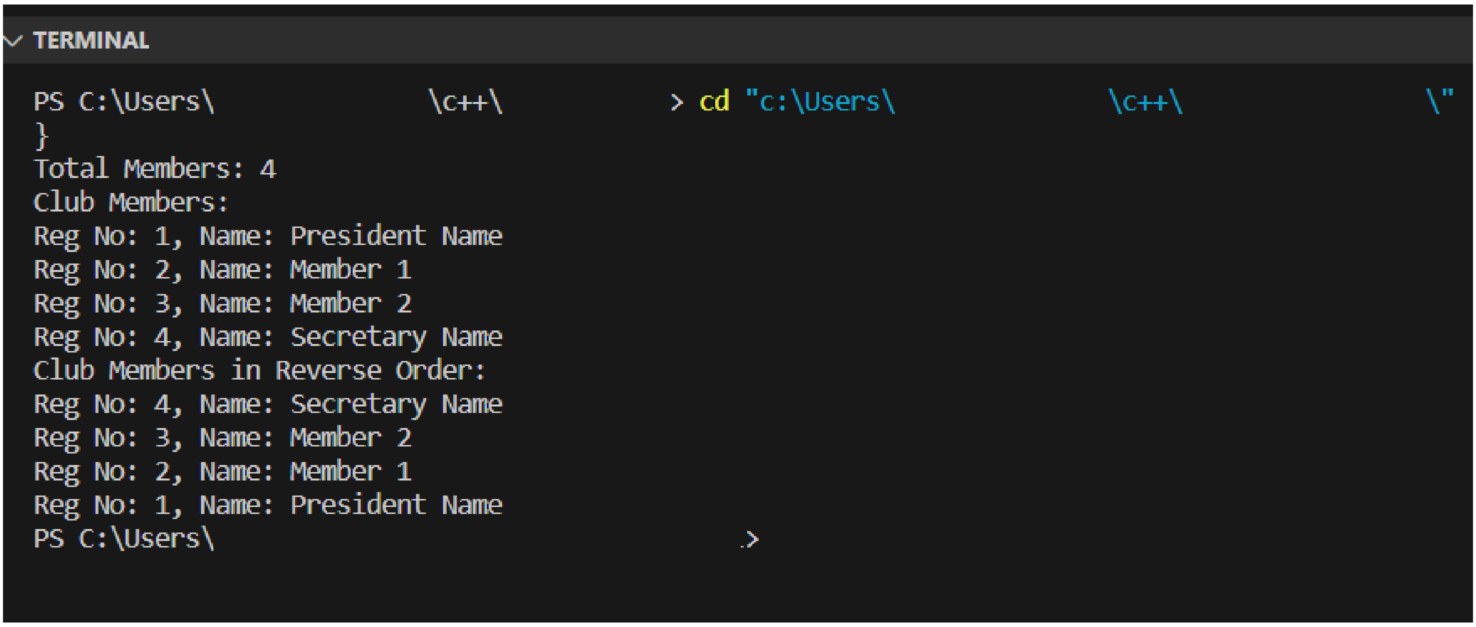
```
Student list (sorted by SGPA in descending order):
Roll No: 9
Name: Irene
SGPA: 9.4

Roll No: 3
Name: Charlie
SGPA: 9.2

Roll No: 5
Name: Emily
SGPA: 9

Roll No: 7
Name: Gina
SGPA: 8.9

Roll No: 1
Name: Alice
SGPA: 8.5

Roll No: 8
Name: Harry
SGPA: 8.3

Roll No: 4
Name: David
SGPA: 8.1

Roll No: 10
Name: Jack
SGPA: 8

Roll No: 2
Name: Bob
SGPA: 7.8

Roll No: 6
Name: Fred
SGPA: 7.5

Charlie not found in the student list.
PS C:\Users\                                   >
```

Assignment No 2

```
C++
#include <iostream>

using namespace std;

struct Member {
        int registrationNo;
string name;
Member* next;
};

class ClubMembers {
private:
     Member* president;
      Member* secretary;

public:
ClubMembers() {
             president = nullptr;
              secretary = nullptr;
     }

     Member* getPresident() {
     return president;
     }

     // Function to add a new member to the club
     void addMember(int regNo, string memberName) {
     Member* newMember = new Member;
     newMember->registrationNo = regNo;
     newMember->name = memberName;
     newMember->next = nullptr;

          if (president == nullptr) {
          president = newMember;
          secretary = newMember;
          } else {
               secretary->next = newMember;
          secretary = newMember;
          }
     }

     // Function to delete a member by registration number
     void deleteMember(int regNo) {
     Member* prev = nullptr;
     Member* curr = president;

          while (curr != nullptr && curr->registrationNo != regNo) {
              prev = curr;
              curr = curr->next;
          }

          if (curr == nullptr) {
          return;
          }

          if (curr == president) {
                president = curr->next;
          } else if (curr == secretary) {
          secretary = prev;
          secretary->next = nullptr;
          } else {
          prev->next = curr->next;
          }

          delete curr;
      }

     // Function to compute the total number of members
     int getTotalMembers() {
     int count = 0;
     Member* current = president;
     while (current != nullptr) {
     count++;
     current = current->next;
     }
     return count;
}

// Function to display the list of members
void displayMembers() {
Member* current = president;
while (current != nullptr) {
                   cout << "Reg No: " << current->registrationNo << ", Name: "
<< current->name << endl;
current = current->next;
          }
}

// Function to display the list in reverse order using recursion
void displayInReverseOrder(Member* current) {
if (current == nullptr)
return;
displayInReverseOrder(current->next);
                cout << "Reg No: " << current->registrationNo << ", Name: " <<
current->name << endl;
}
};
```

```
int main() {
ClubMembers club;
        club.addMember(1, "President Name");
club.addMember(2, "Member 1");
club.addMember(3, "Member 2");
       club.addMember(4, "Secretary Name");

     cout << "Total Members: " << club.getTotalMembers() << endl; cout
     << "Club Members: " << endl;
     club.displayMembers();

     cout << "Club Members in Reverse Order: " << endl;
     club.displayInReverseOrder(club.getPresident());

     return 0;
}
```

output-



```
PS C:\Users\[redacted]\c++\[redacted]> cd "c:\Users\[redacted]\c++\[redacted]\"
}
Total Members: 4
Club Members:
Reg No: 1, Name: President Name
Reg No: 2, Name: Member 1
Reg No: 3, Name: Member 2
Reg No: 4, Name: Secretary Name
Club Members in Reverse Order:
Reg No: 4, Name: Secretary Name
Reg No: 3, Name: Member 2
Reg No: 2, Name: Member 1
Reg No: 1, Name: President Name
PS C:\Users\[redacted]>
```

ASSIGNMENT NO : 3

PROGRAM:

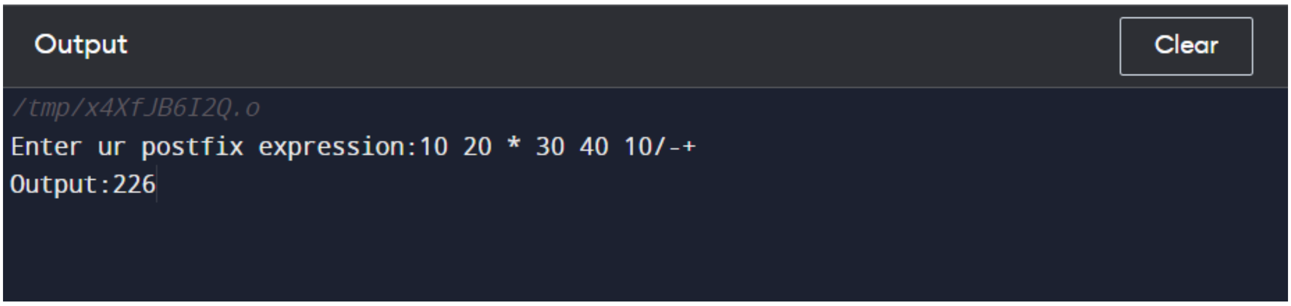
```
#include <iostream>

#include <stdlib.h>

#include <string.h>

#include<ctype.h>

using namespace std;

struct node {

int data;

struct node *next;

};

struct node *top = NULL;

/* create a new node with the given data */

struct node* createNode(int data)

{

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

ptr->data = data;

ptr->next = NULL;

}


void push (int data) {

struct node *ptr= createNode(data);

if (top == NULL) {

top = ptr;
```

```
return;
}
ptr->next = top;
top = ptr;
}
/* pop the top element from the stack */
int pop () {
int data;
struct node *temp;
if (top == NULL)
return -1;
data= top->data;
temp = top;
top = top->next;
free(temp);
return (data);
}
int main() {
char str[100];
int i, data=-1, operand1, operand2, result;
/* i/p postfix expr from the user */
cout <<"Enter ur postfix expression:";
fgets(str, 100, stdin);
for (i= 0;i< strlen(str); i++){
```

```
if (isdigit(str[i])){
/** if the i/p char is digit, parse * character by character to get * complete operand
*/
data= (data ==-1)?0: data;

data = (data * 10) + (str[i]-48);
continue;
}
/* push the operator into the stack */
if (data !=-1){
push(data);
}
if (str[i] == '+' || str[i] =='-'
|| str[i] == '*' || str[i] == '/'){
/*
 * if the i/p character is an operator,
 * then pop two elements from the stack,
 * apply operator and push the result into
 * the stack
*/
operand2= pop();
operand1= pop();
if (operand1 ==-1 || operand2 ==-1)
break;

switch (str[i]) {

case '+':

result= operand1+ operand2;

/* pushing result into the stack */

push(result);

break;

case '-':

result = operand1 - operand2;

push(result);

break;

case '*':

result = operand1 * operand2;

push(result);

break;

case '/':

result = operand1 / operand2;

push(result);


break;

}

}

data = -1;

}
```

```
if (top != NULL && top->next == NULL)
cout<<"Output:"<< top->data;
else
cout<<"u ve entered wrong expression\n";
return 0;
}
```

OUTPUT :



```
/tmp/x4XfJB6I2Q.o
Enter ur postfix expression:10 20 * 30 40 10/-+
Output:226
```

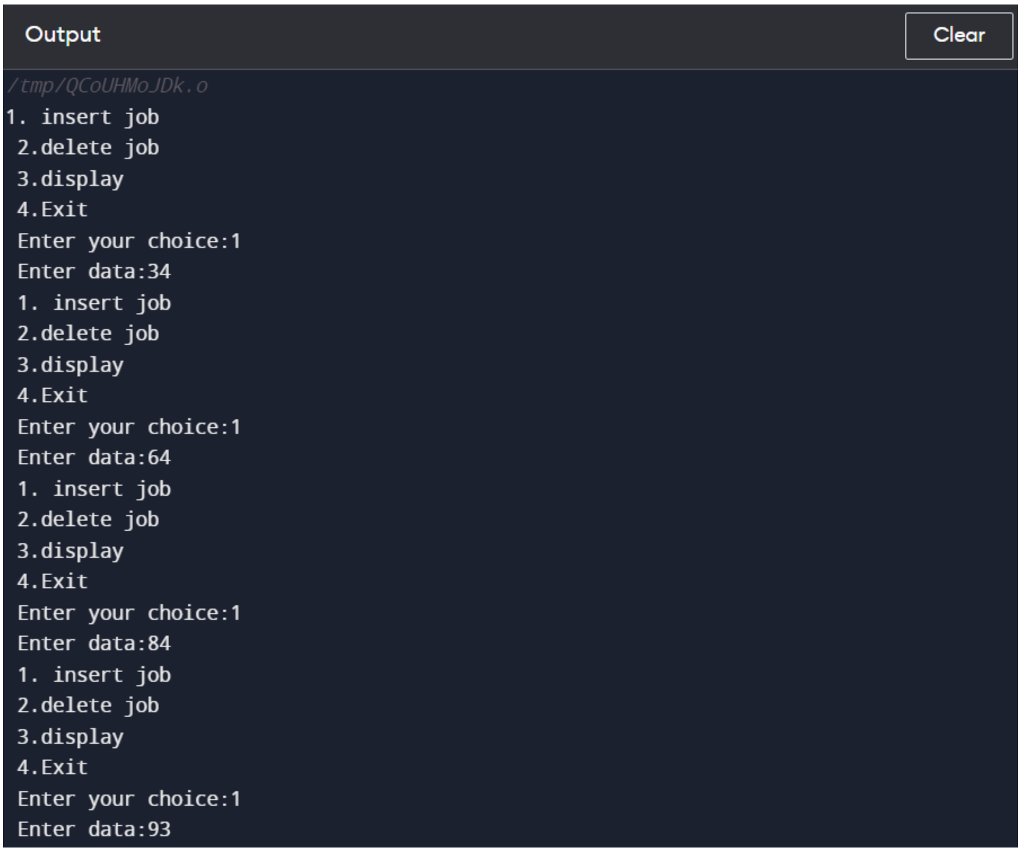
ASSIGNMENT NO : 4

PROGRAM :

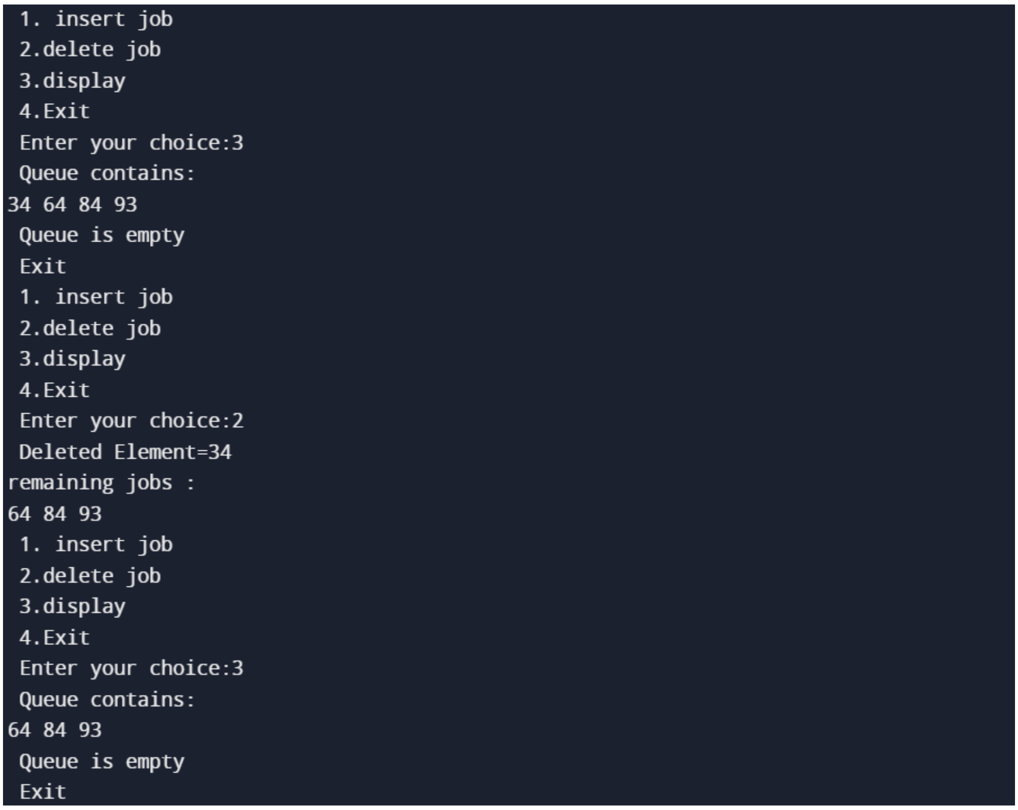
```
#include <iostream>
#define MAX 10
using namespace std;
struct queue
{ int data[MAX];
int front,rear;
};
class Queue
{ struct queue q;
public:
Queue(){q.front=q.rear=-1;}
int isempty();
int isfull();
void enqueue(int);
int delqueue();
void display();
};
int Queue::isempty()
{
return(q.front==q.rear)?1:0;
}
int Queue::isfull()
{ return(q.rear==MAX-1)?1:0;}
void Queue::enqueue(int x)
```

```
{q.data[++q.rear]=x;}
int Queue::delqueue()
{return q.data[++q.front];}
void Queue::display()
{ int i;
cout<<"\n";
for(i=q.front+1;i<=q.rear;i++)
cout<<q.data[i]<<" ";

}
int main()
{ Queue obj;
int ch,x;
do{ cout<<"\n 1. insert job\n 2.delete job\n 3.display\n 4.Exit\n Enter your choice:";
cin>>ch;
switch(ch)
{ case 1: if (!obj.isfull())
{ cout<<"\n Enter data:";
cin>>x;
obj.enqueue(x);
}
else
cout<< "Queue is overflow";
break;
case 2: if(!obj.isempty())
cout<<"\n Deleted Element="<<obj.delqueue();
else
{ cout<<"\n Queue is underflow"; }
cout<<"\nremaining jobs :";
obj.display();
break;
case 3: if (!obj.isempty())
{ cout<<"\n Queue contains:";
obj.display();
}
else
break;
cout<<"\n Queue is empty";
case 4: cout<<"\n Exit";
}
}while(ch!=4);
return 0;
}
```

OUTPUT :



```
/tmp/QCoUHMoJDk.o
1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:1
 Enter data:34
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:1
 Enter data:64
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:1
 Enter data:84
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:1
 Enter data:93
```



```
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:3
 Queue contains:
34 64 84 93
 Queue is empty
 Exit
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:2
 Deleted Element=34
remaining jobs :
64 84 93
 1. insert job
 2.delete job
 3.display
 4.Exit
 Enter your choice:3
 Queue contains:
64 84 93
 Queue is empty
 Exit
```

Assignment No. 5

Program:-

```
#include <iostream>

using namespace std;

#define SIZE 5

class dequeue

{

        int a[10], front, rear, count;


public:

        dequeue();

        void add_at_beg(int);

        void add_at_end(int);

        void delete_fr_front();

        void delete_fr_rear();

        void display();

};

dequeue::dequeue()

{

        front =

                -1;

        rear =

                -1;

        count = 0;

}

void dequeue::add_at_beg(int item)

{

        int i;
```

```
    if (front ==
        -1)
    {
        front++;
        rear++;
        a[rear] = item;
        count++;
    }
    else if (rear >= SIZE - 1)
    {
    }
    else
    {
        cout << "\nInsertion is not possible,overflow!!!!";
        for (i = count; i >= 0; i--)
        {
            a[i] = a[i - 1];
        }
        a[i] = item;
        count++;
        rear++;
    }
}
void dequeue::add_at_end(int item)
{
    if (front == -1)
    {
        front++;
        rear++;
        a[rear] = item;
        count++;
    }
    else if (rear >= SIZE - 1)
    {
        cout << "\nInsertion is not possible,overflow!!!";
        return;
    }
    else
    {
        a[++rear] = item;
    }
}
void dequeue::display()
{
    for (int i = front; i <= rear; i++)
    {
        cout << a[i] << " ";
    }
}
void dequeue::delete_fr_front()
{
    if (front == -1)
    {
    }
    else
    {
        cout << "Deletion is not possible:: Dequeue is empty";
        return;
```

```
                if (front == rear)
                {
                        front = rear = -1;
                        return;
                }
                cout << "The deleted element is " << a[front];
                front = front + 1;
        }
}
void dequeue::delete_fr_rear()
{
        if (front == -1)
        {
        }
        else
        {
                cout << "Deletion is not possible:Dequeue is empty";
                return;
                if (front == rear)
                {
                        front = rear = -1;
                }
                cout << "The deleted element is " << a[rear];
                rear = rear - 1;
        }
}
int main()
{
        int c, item;
```

```
dequeue d1;
do
{
        cout << "\n\n****DEQUEUE OPERATION****\n";
        cout << "\n1-Insert at beginning";
        cout << "\n2-Insert at end";
        cout << "\n3_Display";
        cout << "\n4_Deletion from front";
        cout << "\n5-Deletion from rear";
        cout << "\n6_Exit";
        cout << "\nEnter your choice<1-4>:";
        cin >> c;
        switch (c)
        {
        case 1:
                cout << "Enter the element to be inserted:";
                cin >> item;
                d1.add_at_beg(item);
                break;
        case 2:
                cout << "Enter the element to be inserted:";
                cin >> item;
                d1.add_at_end(item);
                break;
        case 3:
                d1.display();
                break;
        case 4:
                d1.delete_fr_front();
```

```
            break;
        case 5:
            d1.delete_fr_rear();
            break;
        case 6:
            exit(1);
            break;
        default:
            cout << "Invalid choice";
            break;
        }
    } while (c != 7);
    return 0;
}
```

Output :-

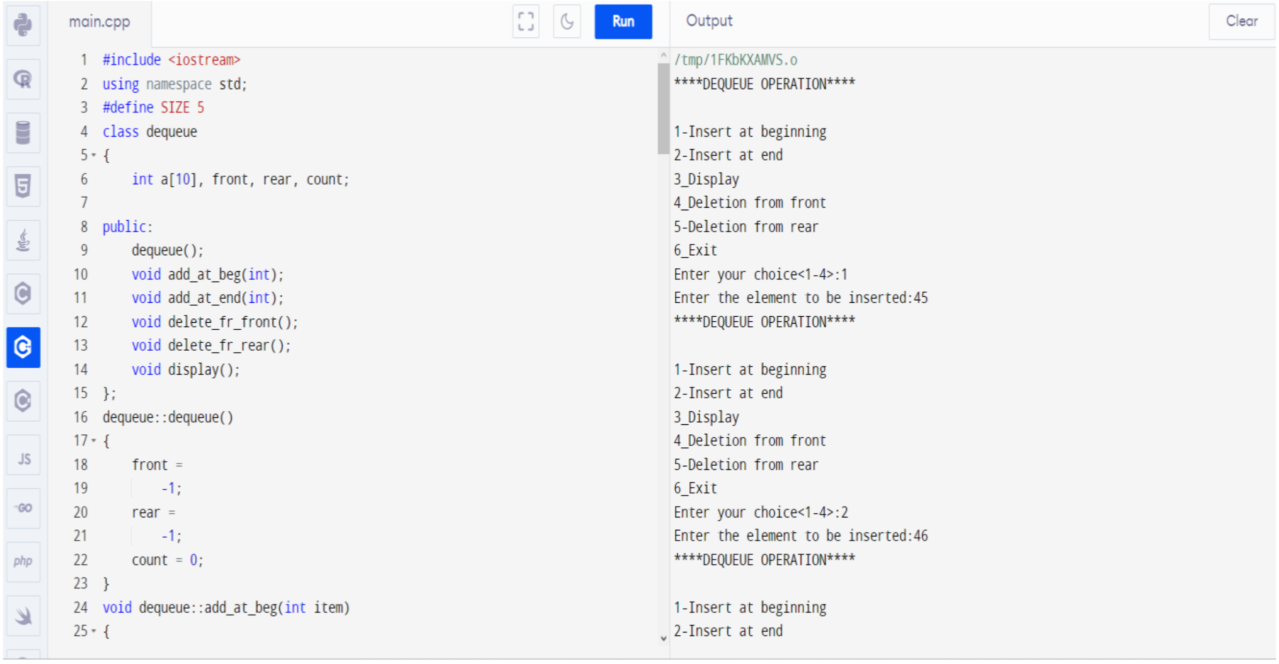
```
****DEQUEUE OPERATION****
1-Insert at beginning
2-Insert at end
3_Display
4_Deletion from front
5-Deletion from rear
6_Exit
Enter your choice<1-4>:1
Enter the element to be inserted:45
****DEQUEUE OPERATION****
1-Insert at beginning
2-Insert at end
3_Display
4_Deletion from front
```

5-Deletion from rear

6_Exit

Enter your choice<1-4>:2

Enter the element to be inserted:46



```
#include <iostream>
using namespace std;
#define SIZE 5
class dequeue
{
    int a[10], front, rear, count;

public:
    dequeue();
    void add_at_beg(int);
    void add_at_end(int);
    void delete_fr_front();
    void delete_fr_rear();
    void display();
};
dequeue::dequeue()
{
    front =
        -1;
    rear =
        -1;
    count = 0;
}
void dequeue::add_at_beg(int item)
{
```

```
/tmp/1FKbKXAMVS.o
****DEQUEUE OPERATION****

1-Insert at beginning
2-Insert at end
3_Display
4_Deletion from front
5-Deletion from rear
6_Exit
Enter your choice<1-4>:1
Enter the element to be inserted:45
****DEQUEUE OPERATION****

1-Insert at beginning
2-Insert at end
3_Display
4_Deletion from front
5-Deletion from rear
6_Exit
Enter your choice<1-4>:2
Enter the element to be inserted:46
****DEQUEUE OPERATION****

1-Insert at beginning
2-Insert at end
```

DSA Assignment No.6

Program Code :

```
#include <iostream>
# include <cstdlib>
# include <string.h>
using namespace std;
/*
* Node Declaration
*/
struct node
{
char label[10];
int ch_count;
struct node *child[10];
}*root;
/*
* Class Declaration
*/
class BST
{
public:
void create_tree();
void display(node * r1);
BST()
{
root = NULL;
}
};
void BST::create_tree()
{
int tbooks,tchapters,i,j,k;
root = new node();
cout<<"Enter name of book";
cin>>root->label;
cout<<"Enter no. of chapters in book";
cin>>tchapters;
root->ch_count = tchapters;
for(i=0;i<tchapters;i++)
{
root->child[i] = new node;
```

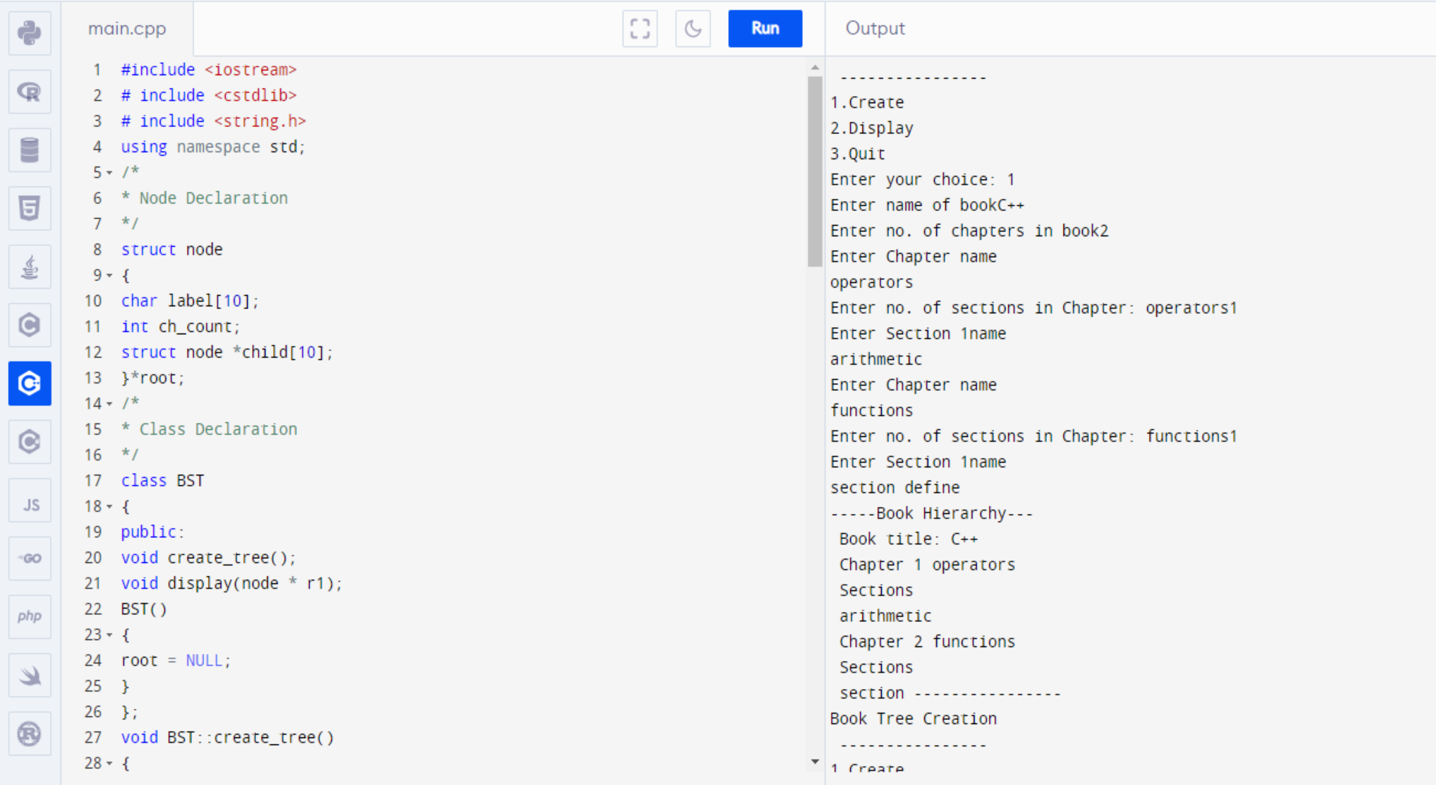
```
cout<<"Enter Chapter name\n";
cin>>root->child[i]->label;
cout<<"Enter no. of sections in Chapter: "<<root->child[i]->label;
cin>>root->child[i]->ch_count;
for(j=0;j<root->child[i]->ch_count;j++)
{
root->child[i]->child[j] = new node;
cout<<"Enter Section "<<j+1<<"name\n";
cin>>root->child[i]->child[j]->label;
//cout<<"Enter no. of subsections in "<<r1->child[i]->child[j]->label;
//cin>>r1->child[i]->ch_count;
}
}
}
void BST::display(node * r1)
{
int i,j,k,tchapters;
if(r1 != NULL)
{
cout<<"\n-----Book Hierarchy---";
cout<<"\n Book title: "<<r1->label;
tchapters= r1->ch_count;
for(i=0;i<tchapters;i++) {
cout<<"\n Chapter "<<i+1;
cout<<" "<<r1->child[i]->label;
cout<<"\n Sections";
for(j=0;j<r1
->child[i]
->ch_count;j++)
{
//cin>>r1->child[i]->child[j]->label;
cout<<"\n "<<r1->child[i]->child[j]->label;
}}}}
/** Main Contains Menu
*/
int main() {
int choice;
BST bst;
while (1) {
cout<<" ----------------"<<endl;
cout<<"Book Tree Creation"<<endl;
cout<<" ----------------"<<endl;
cout<<"1.Create"<<endl;
cout<<"2.Display"<<endl;
```

```
cout<<"3.Quit"<<endl;
cout<<"Enter your choice: ";
cin>>choice;
switch(choice) {
case 1:
bst.create_tree();
case 2:
bst.display(root);
break;
case 3:
exit(1);
default:
cout<<"Wrong choice"<<endl;
}
}
}
```

OUTPUT :

```
----------------
Book Tree Creation
 ----------------
1.Create
2.Display
3.Quit
Enter your choice: 1
Enter name of bookC++
Enter no. of chapters in book2
Enter Chapter name
operators
Enter no. of sections in Chapter: operators1
Enter Section 1name
arithmetic
Enter Chapter name
functions
Enter no. of sections in Chapter: functions1
Enter Section 1name
section define
-----Book Hierarchy---
 Book title: C++
 Chapter 1 operators
 Sections
 arithmetic
 Chapter 2 functions
```

Sections
section ----------------



```
#include <iostream>
# include <cstdlib>
# include <string.h>
using namespace std;
/*
* Node Declaration
*/
struct node
{
char label[10];
int ch_count;
struct node *child[10];
}*root;
/*
* Class Declaration
*/
class BST
{
public:
void create_tree();
void display(node * r1);
BST()
{
root = NULL;
}
};
void BST::create_tree()
{
```

```
 ---------------
1.Create
2.Display
3.Quit
Enter your choice: 1
Enter name of bookC++
Enter no. of chapters in book2
Enter Chapter name
operators
Enter no. of sections in Chapter: operators1
Enter Section 1name
arithmetic
Enter Chapter name
functions
Enter no. of sections in Chapter: functions1
Enter Section 1name
section define
-----Book Hierarchy---
 Book title: C++
 Chapter 1 operators
 Sections
 arithmetic
 Chapter 2 functions
 Sections
 section ----------------
Book Tree Creation
 ---------------
1.Create
```

Assignment No : 07

PROGRAM

```
#include <iostream>
using namespace std;
#include <conio.h>
struct tree
{
    tree *l, *r;
    int data;
}*root = NULL, *p = NULL, *np = NULL, *q;

void create()
{
    int value,c = 0;
    while (c < 7)
    {
        if (root == NULL)
        {
            root = new tree;
            cout<<"enter value of root node\n";
            cin>>root->data;
            root->r=NULL;
            root->l=NULL;
        }
        else
        {
            p = root;
            cout<<"enter value of node\n";
            cin>>value;
            while(true)
            {
                if (value < p->data)
                {
                    if (p->l == NULL)
                    {
                        p->l = new tree;
                        p = p->l;
                        p->data = value;
                        p->l = NULL;
                        p->r = NULL;
                        cout<<"value entered in left\n";
                        break;
                    }
                    else if (p->l != NULL)
                    {
                        p = p->l;
                    }
                }
                else if (value > p->data)
                {
                    if (p->r == NULL)
                    {
                        p->r = new tree;
                        p = p->r;
                        p->data = value;
                        p->l = NULL;
                        p->r = NULL;
                        cout<<"value entered in right\n";
                break;
            }
                    else if (p->r != NULL)
                    {
                        p = p->r;
                    }
                 }
             }
        }
        c++;
    }
}
void inorder(tree *p)
{
    if (p != NULL)
    {
        inorder(p->l);
        cout<<p->data<<endl;
        inorder(p->r);
    }
}
void preorder(tree *p)
{
    if (p != NULL)
    {
        cout<<p->data<<endl;
        preorder(p->l);
        preorder(p->r);
    }
}
void postorder(tree *p)
{
    if (p != NULL)
    {
        postorder(p->l);
        postorder(p->r);
        cout<<p->data<<endl;
    }
}
int main()
{
    create();
    cout<<"printing traversal in inorder\n";
    inorder(root);
    cout<<"printing traversal in preorder\n";
    preorder(root);
    cout<<"printing traversal in postorder\n";
    postorder(root);
    getch();
}
```

OUTPUT

```
enter value of root node

10

enter value of node

5

value entered in left

enter value of node

15

value entered in right

enter value of node

4

value entered in left

enter value of node

6

value entered in right

enter value of node

14

value entered in left

enter value of node

16

value entered in right

printing traversal in inorder

4

5

6

10

14

15

16

printing traversal in preorder

10

5

4

6

15

14

16

printing traversal in postorder

4

6

5

14

16

15

10
```

Assignment No : 08

PROGRAM

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

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

class BST {
private:
    Node* root;

    Node* insertUtil(Node* root, int val) {
        if (root == nullptr) {
            return new Node(val);
        }

        if (val < root->data) {
            root->left = insertUtil(root->left, val);
        } else if (val > root->data) {
            root->right = insertUtil(root->right, val);
        }

        return root;
    }

    int longestPathUtil(Node* root) {
        if (root == nullptr) {
            return 0;
        }

        int leftDepth = longestPathUtil(root->left);
        int rightDepth = longestPathUtil(root->right);

        return 1 + max(leftDepth, rightDepth);
    }

    int findMinUtil(Node* root) {
        if (root == nullptr) {
            cout << "Tree is empty." << endl;
            return -1; // Return some default value indicating empty tree
        }

        while (root->left != nullptr) {
            root = root->left;
        }
        return root->data;
    }

    Node* swapPointersUtil(Node* root) {
        if (root == nullptr) {
            return nullptr;
        }

        Node* temp = root->left;
        root->left = root->right;
        root->right = temp;

        swapPointersUtil(root->left);
        swapPointersUtil(root->right);

        return root;
    }

    Node* searchUtil(Node* root, int val) {
        if (root == nullptr || root->data == val) {
            return root;
        }

        if (val < root->data) {
            return searchUtil(root->left, val);
        } else {
            return searchUtil(root->right, val);
        }
    }

public:
    BST() {
        root = nullptr;
    }

    void insert(int val) {
        root = insertUtil(root, val);
    }

    int longestPath() {
        return longestPathUtil(root);
    }

    int findMin() {
        return findMinUtil(root);
    }

    void swapPointers() {
        root = swapPointersUtil(root);
    }

    bool search(int val) {
        Node* result = searchUtil(root, val);
        return result != nullptr;
    }
};

int main() {
```

```cpp
    BST tree;

    // Inserting values into the BST
    int values[] = {5, 3, 7, 1, 4, 6, 9};
    int numValues = sizeof(values) / sizeof(values[0]);

    for (int i = 0; i < numValues; i++) {
        tree.insert(values[i]);
    }

    // Example usage of BST functionalities
    cout << "Longest path in the tree: " << tree.longestPath() << endl;
    cout << "Minimum value in the tree: " << tree.findMin() << endl;

    cout << "Swapping left and right pointers at every node..." << endl;
    tree.swapPointers();

    int searchVal = 6;
    cout << "Searching for value " << searchVal << ": ";
    if (tree.search(searchVal)) {
        cout << "Found!" << endl;
    } else {
        cout << "Not Found!" << endl;
    }

    return 0;
}
```

OUTPUT

```
Longest path in the tree: 3

Minimum value in the tree: 1

Swapping left and right pointers at every node...

Searching for value 6: Not Found!
```

Assignment No : 09

PROGRAM

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

class Graph {
private:
    int V;
    vector<vector<int>> adjList;

public:
    Graph(int vertices) {
        V = vertices;
        adjList.resize(V);
    }

    void addEdge(int src, int dest) {
        adjList[src].push_back(dest);
    }

    void DFS(int start) {
        vector<bool> visited(V, false);
        DFSUtil(start, visited);
    }

    void DFSUtil(int vertex, vector<bool>& visited) {
        visited[vertex] = true;
        cout << vertex << " ";

        for (int i = 0; i < adjList[vertex].size(); ++i) {
            int adjVertex = adjList[vertex][i];
            if (!visited[adjVertex]) {
                DFSUtil(adjVertex, visited);
            }
        }
    }

    void BFS(int start) {
        vector<bool> visited(V, false);
        queue<int> queue;

        visited[start] = true;
        queue.push(start);

        while (!queue.empty()) {
            int current = queue.front();
            cout << current << " ";
            queue.pop();

            for (int i = 0; i < adjList[current].size(); ++i) {
                int adjVertex = adjList[current][i];
                if (!visited[adjVertex]) {
                    visited[adjVertex] = true;
                    queue.push(adjVertex);
                }
            }
        }
    }
};

int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;
    cout << "Enter the number of edges: ";
    cin >> E;

    Graph graph(V);

    cout << "Enter " << E << " edges (format: source destination):" << endl;
    for (int i = 0; i < E; ++i) {
        int src, dest;
        cin >> src >> dest;
        graph.addEdge(src, dest);
    }

    int startVertex;
    cout << "Enter the starting vertex for traversal: ";
    cin >> startVertex;

    cout << "DFS traversal starting from vertex " << startVertex << ": ";
    graph.DFS(startVertex);
    cout << endl;

    cout << "BFS traversal starting from vertex " << startVertex << ": ";
    graph.BFS(startVertex);
    cout << endl;

    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 4

Enter the number of edges: 5

Enter 5 edges (format: source destination):

0 1

0 2

1 2

2 3

3 0

Enter the starting vertex for traversal: 0

DFS  traversal starting from vertex 0: 0 1 2 3

BFS traversal starting from vertex 0: 0 1 2 3
```

Assignment No : 10

PROGRAM

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

struct Edge {
    int src, dest, weight;
};

class Graph {
private:
    int V;
    vector<Edge> edges;

public:
    Graph(int vertices) : V(vertices) {}

    void addEdge(int src, int dest, int weight) {
        Edge edge;
        edge.src = src;
        edge.dest = dest;
        edge.weight = weight;
        edges.push_back(edge);
    }

    int find(vector<int>& parent, int i) {
        if (parent[i] == -1)
            return i;
        return find(parent, parent[i]);
    }

    void unionSet(vector<int>& parent, int x, int y) {
        int xroot = find(parent, x);
        int yroot = find(parent, y);
        parent[xroot] = yroot;
    }

    void kruskalMST() {
        vector<Edge> result(V - 1);
        sort(edges.begin(), edges.end(), [](const Edge& a, const Edge& b) {
            return a.weight < b.weight;
        });

        vector<int> parent(V, -1);

        int e = 0;
        int i = 0;
        while (e < V - 1 && i < edges.size()) {
            Edge next_edge = edges[i++];
            int x = find(parent, next_edge.src);
            int y = find(parent, next_edge.dest);

            if (x != y) {
                result[e++] = next_edge;
                unionSet(parent, x, y);
            }
        }

        cout << "Minimum Spanning Tree formed by connecting offices with minimum 
cost:" << endl;
        for (int j = 0; j < V - 1; ++j) {
            cout << result[j].src << " - " << result[j].dest << " : " << 
result[j].weight << endl;
        }
    }
};

int main() {
    int numOffices, numConnections;
    cout << "Enter the number of offices: ";
    cin >> numOffices;
    cout << "Enter the number of connections: ";
    cin >> numConnections;

    Graph graph(numOffices);

    cout << "Enter " << numConnections << " connections in the format: src dest cost" 
<< endl;
    for (int i = 0; i < numConnections; ++i) {
        int src, dest, cost;
        cin >> src >> dest >> cost;
        graph.addEdge(src, dest, cost);
    }

    graph.kruskalMST();

    return 0;
}
```

OUTCOME

```
Enter the number of offices: 4

Enter the number of connections: 5

Enter 5 connections in the format: src dest cost

0 1 4

0 2 1

1 2 3

2 3 2

3 0 5

Minimum Spanning Tree formed by connecting offices with minimum cost:

0 - 2 : 1

2 - 3 : 2

1 - 2 : 3
```