Name: Avani Bhartiya
Division: SE-AIDS-B (S1 Batch)
Roll Number: 26507

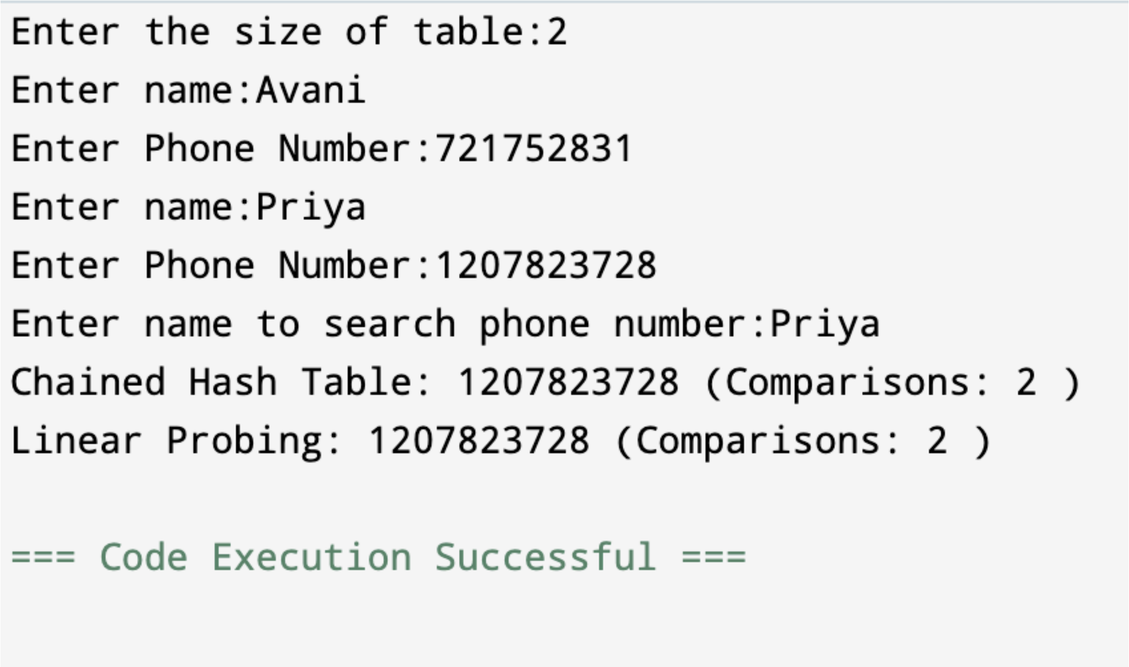
ASSIGNMENT 1:

CODE:

```
class PhoneEntry:
    def __init__(self, name, number):
        self.name = name
        self.number = number

class ChainedHashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(size)]
        self.comparisons = 0

    def hash_function(self, name):
        return sum(ord(c) for c in name) % self.size

    def insert(self, name, number):
        index = self.hash_function(name)
        self.table[index].append(PhoneEntry(name, number))

    def search(self, name):
        self.comparisons = 0
        index = self.hash_function(name)

        for entry in self.table[index]:
            self.comparisons += 1
            if entry.name == name:
                return entry.number
        return None

class LinearProbingHashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size
        self.comparisons = 0

    def hash_function(self, name):
        return sum(ord(c) for c in name) % self.size

    def insert(self, name, number):
        index = self.hash_function(name)

        while self.table[index] is not None:
            index = (index + 1) % self.size

        self.table[index] = PhoneEntry(name, number)

    def search(self, name):
        self.comparisons = 0
        index = self.hash_function(name)

        while self.table[index] is not None:
            self.comparisons += 1
            if self.table[index].name == name:
                return self.table[index].number
            index = (index + 1) % self.size

            if index == self.hash_function(name):
                break

        return None

size = int(input("Enter the size of table:"))
chained_table = ChainedHashTable(size)
linear_table = LinearProbingHashTable(size)

for _ in range(size):
    name = input("Enter name:")
    phone_no = input("Enter Phone Number:")
    chained_table.insert(name , phone_no)
    linear_table.insert(name , phone_no)

search = input("Enter name to search phone number:")
chained_result = chained_table.search(search)
```

```
linear_result = linear_table.search(search)

if chained_result:
    print("Chained Hash Table:", chained_result, "(Comparisons:", chained_table.comparisons, ")")
else:
    print("Chained Hash Table: Not Found")

if linear_result:
    print("Linear Probing:", linear_result, "(Comparisons:", linear_table.comparisons, ")")
else:
    print("Linear Probing: Not Found")
```

OUTPUT:



```
Enter the size of table:2
Enter name:Avani
Enter Phone Number:721752831
Enter name:Priya
Enter Phone Number:1207823728
Enter name to search phone number:Priya
Chained Hash Table: 1207823728 (Comparisons: 2 )
Linear Probing: 1207823728 (Comparisons: 2 )

=== Code Execution Successful ===
```

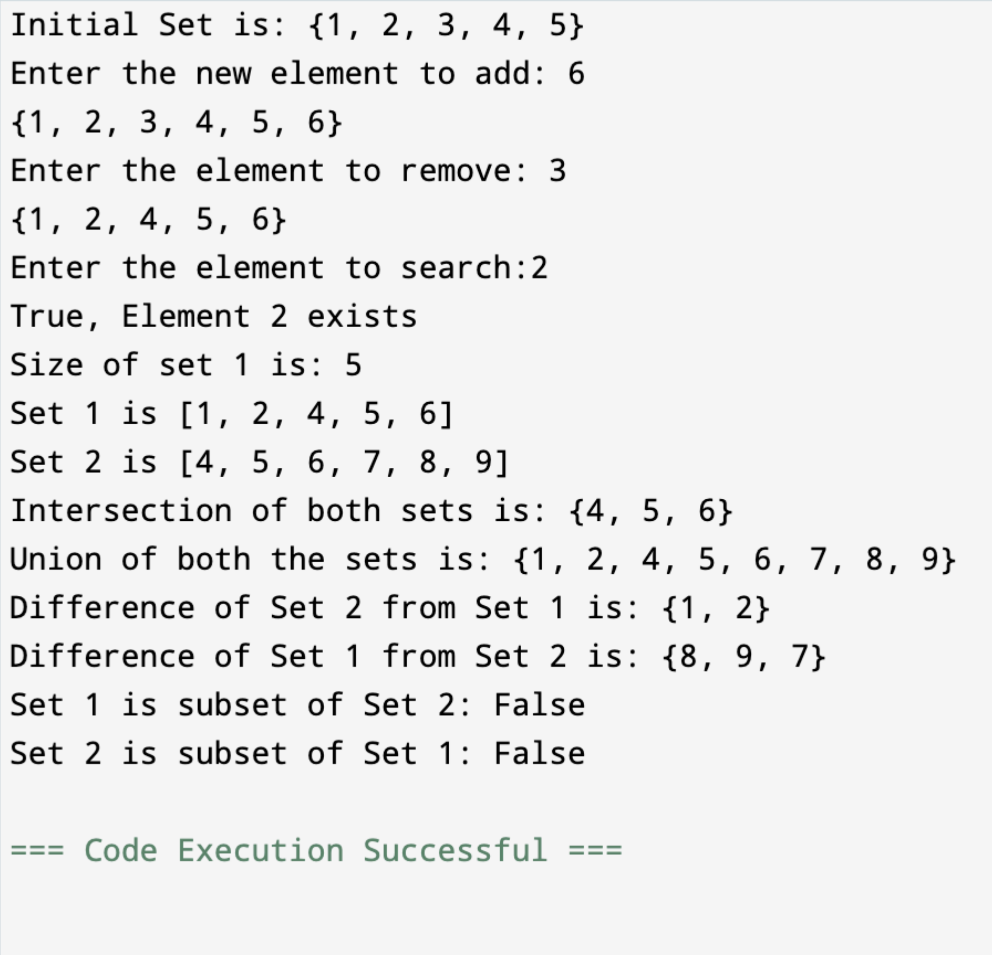
ASSIGNMENT 2:

CODE:

```
set1={1,2,3,4,5}
set2={4,5,6,7,8,9}

print("Initial Set is:",set1)

def add():
    n=int(input("Enter the new element to add: "))
    set1.add(n)
    print(set1)
add()

def remove():
    n=int(input("Enter the element to remove: "))
    if n not in set1:
        print("Element not present in set")
    else:
        set1.remove(n)
    print(set1)
remove()

def present():
    n=int(input("Enter the element to search:"))
    if n in set1:
        print("True, Element",n,"exists")
    else:
        print("False, Element",n,"doesnt exists")
present()

def size():
    print("Size of set 1 is:",len(set1))
size()

def iterator():
    print("Set 1 is",list(iter(set1)))
    print("Set 2 is",list(iter(set2)))
iterator()
```

```
def intersection():
    inter=set1.intersection(set2)
    print("Intersection of both sets is:",inter)
intersection()

def union():
    uni=set1.union(set2)
    print("Union of both the sets is:",uni)
union()

def difference():
    diff=set1.difference(set2)
    print("Difference of Set 2 from Set 1 is:",diff)
    diff1=set2.difference(set1)
    print("Difference of Set 1 from Set 2 is:",diff1)
difference()

def subset():
    ss=set1.issubset(set2)
    print("Set 1 is subset of Set 2:", ss)
    ss1=set2.issubset(set1)
    print("Set 2 is subset of Set 1:", ss1)
subset()
```

OUTPUT:



```
Initial Set is: {1, 2, 3, 4, 5}
Enter the new element to add: 6
{1, 2, 3, 4, 5, 6}
Enter the element to remove: 3
{1, 2, 4, 5, 6}
Enter the element to search:2
True, Element 2 exists
Size of set 1 is: 5
Set 1 is [1, 2, 4, 5, 6]
Set 2 is [4, 5, 6, 7, 8, 9]
Intersection of both sets is: {4, 5, 6}
Union of both the sets is: {1, 2, 4, 5, 6, 7, 8, 9}
Difference of Set 2 from Set 1 is: {1, 2}
Difference of Set 1 from Set 2 is: {8, 9, 7}
Set 1 is subset of Set 2: False
Set 2 is subset of Set 1: False

=== Code Execution Successful ===
```

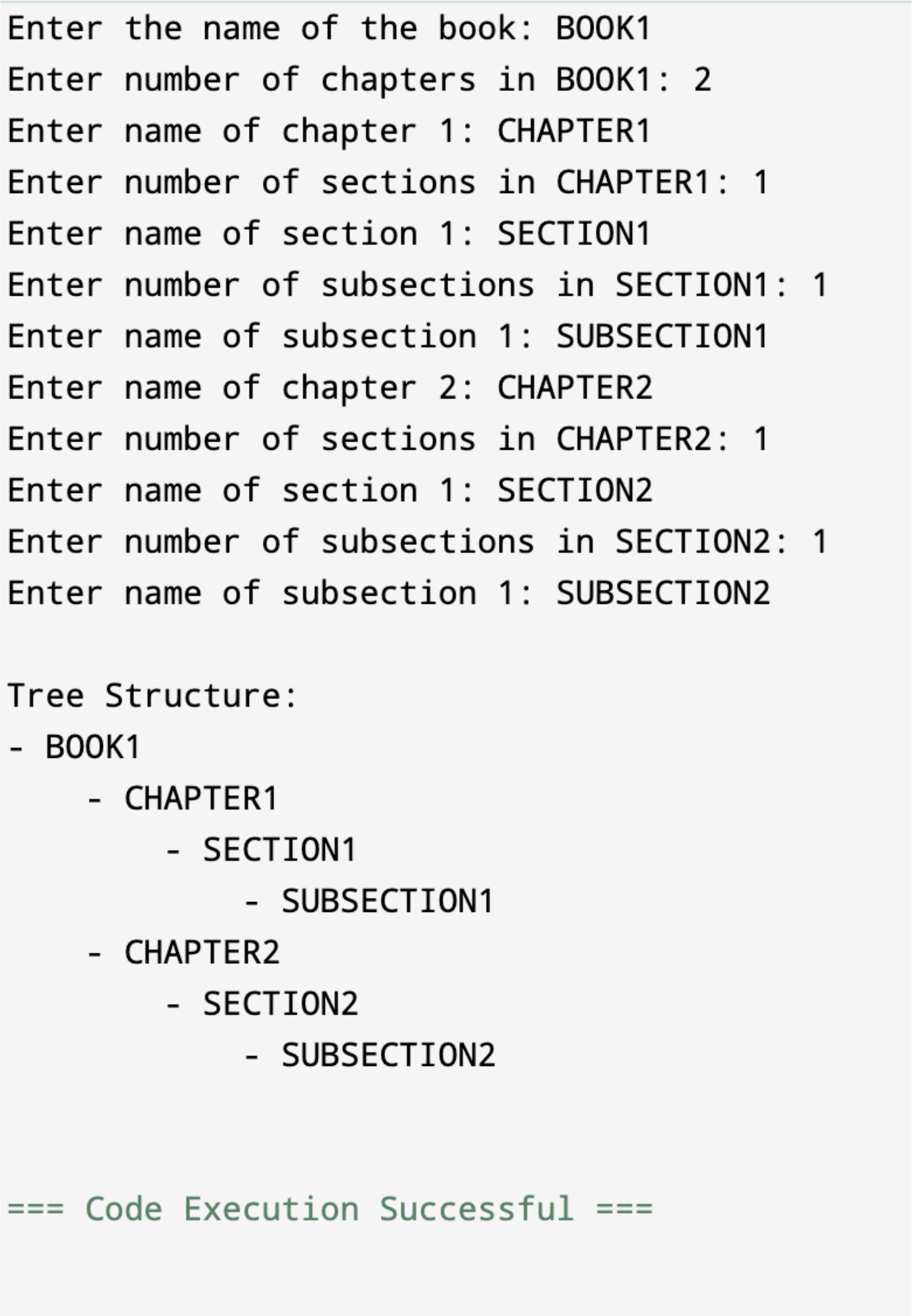
ASSIGNMENT 3:

CODE:

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

class TreeNode {
public:
    string name;
    vector<TreeNode*> children;

   TreeNode(string nodeName)
   {
    name = nodeName;
   }


    ~TreeNode()
    {
        for (TreeNode* child : children)
        {
            delete child;
        }
    }

    void addChild(TreeNode* child)
    {
        children.push_back(child);
    }

    void printTree(int level = 0)
    {
        cout << string(level * 4, ' ') << "- " << name << endl;
        for (TreeNode* child : children) {
            child->printTree(level + 1);
        }
    }
};
```

```
void createChildren(TreeNode* parent, string childType) {
    cout << "Enter number of " << childType << "s in " << parent->name << ": ";
    int count;
    cin >> count;
    cin.ignore();

    for (int i = 1; i <= count; i++)
    {
        cout << "Enter name of " << childType << " " << i << ": ";
        string name;
        getline(cin, name);
        TreeNode* child = new TreeNode(name);
        parent->addChild(child);

        if (childType == "chapter")
        {
            createChildren(child, "section");
        }
        if (childType == "section")
        {
            createChildren(child, "subsection");
        }
    }
}

int main() {
    cout << "Enter the name of the book: ";
    string bookName;
    getline(cin, bookName);
    TreeNode* book = new TreeNode(bookName);

    createChildren(book, "chapter");

    cout << "\nTree Structure:" << endl;
    book->printTree();

    delete book;
    return 0;
}
```

OUTPUT:



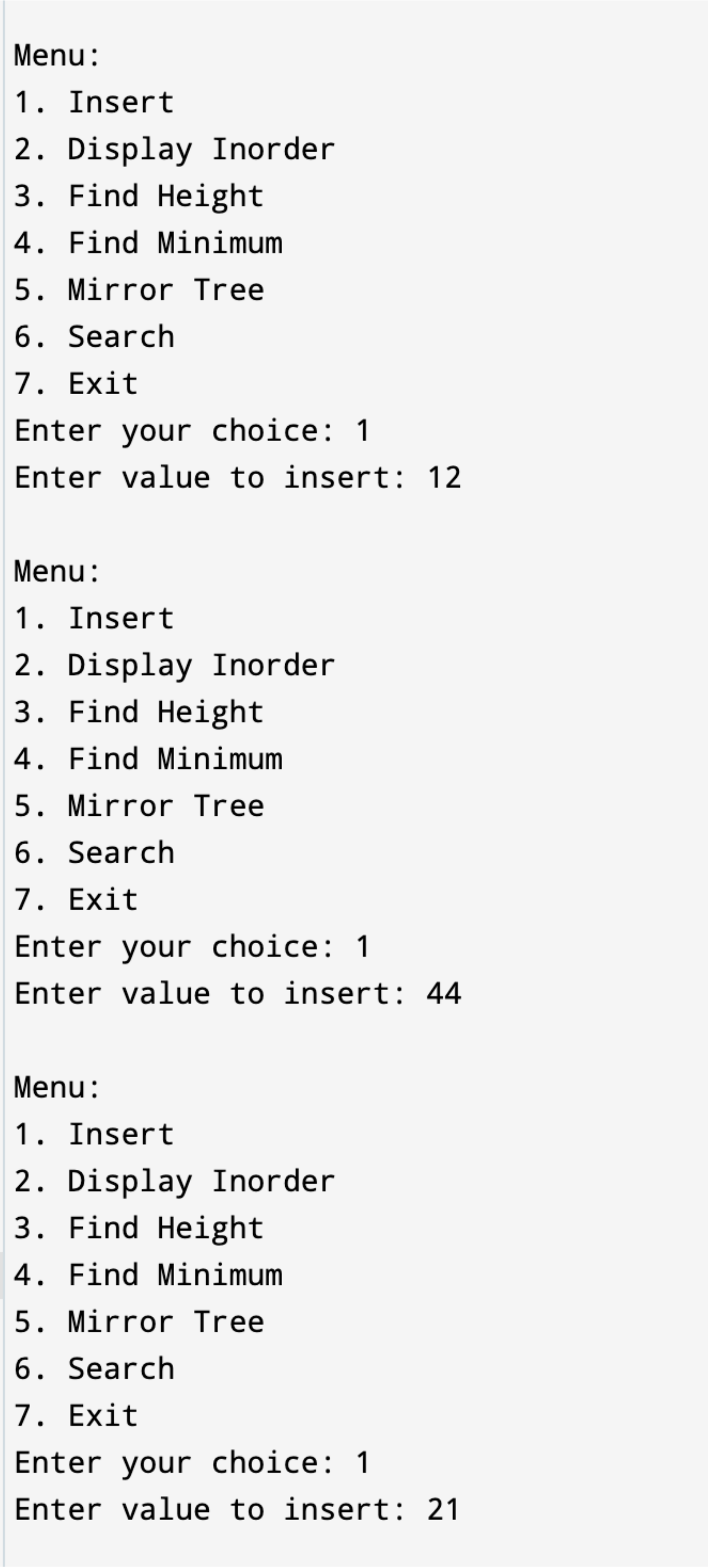
```
Enter the name of the book: BOOK1
Enter number of chapters in BOOK1: 2
Enter name of chapter 1: CHAPTER1
Enter number of sections in CHAPTER1: 1
Enter name of section 1: SECTION1
Enter number of subsections in SECTION1: 1
Enter name of subsection 1: SUBSECTION1
Enter name of chapter 2: CHAPTER2
Enter number of sections in CHAPTER2: 1
Enter name of section 1: SECTION2
Enter number of subsections in SECTION2: 1
Enter name of subsection 1: SUBSECTION2

Tree Structure:
- BOOK1
    - CHAPTER1
        - SECTION1
            - SUBSECTION1
    - CHAPTER2
        - SECTION2
            - SUBSECTION2



=== Code Execution Successful ===
```

ASSIGNMENT 4:

CODE:

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

// Definition of a Node in the Binary Search Tree
struct Node {
    int data;
    Node* left;
    Node* right;

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

// Function to insert a new node in BST
Node* insert(Node* root, int val) {
    if (!root) return new Node(val);
    if (val < root->data)
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);
    return root;
}

// Function to find the number of nodes in the longest path (height of the tree)
int findHeight(Node* root) {
    if (!root) return 0;
    int leftHeight = findHeight(root->left);
    int rightHeight = findHeight(root->right);
    return 1 + max(leftHeight, rightHeight);
}

// Function to find the minimum value in the BST
int findMin(Node* root) {
    if (!root) return -1; // Tree is empty
    while (root->left)
        root = root->left;
    return root->data;
}

// Function to mirror the tree (swap left and right pointers at every node)
void mirror(Node* root) {
    if (!root) return;
    swap(root->left, root->right);
    mirror(root->left);
    mirror(root->right);
}

// Function to search for a value in the BST
bool search(Node* root, int val) {
    if (!root) return false;
    if (root->data == val) return true;
    if (val < root->data)
        return search(root->left, val);
    return search(root->right, val);
}

// Inorder traversal to display the BST
void inorder(Node* root) {
    if (!root) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = nullptr;
    int choice, val;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Insert\n2. Display Inorder\n3. Find Height\n4. Find Minimum\n5. Mirror
Tree\n6. Search\n7. Exit\nEnter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value to insert: ";
                cin >> val;
                root = insert(root, val);
                break;
            case 2:
                cout << "Inorder traversal: ";
                inorder(root);
                cout << endl;
                break;
            case 3:
                cout << "Height of the tree: " << findHeight(root) << endl;
                break;
            case 4:
                cout << "Minimum value in the BST: " << findMin(root) << endl;
                break;
            case 5:
                mirror(root);
                cout << "Tree mirrored.\n";
                break;
            case 6:
                cout << "Enter value to search: ";
                cin >> val;
                cout << (search(root, val) ? "Found" : "Not Found") << endl;
                break;
            case 7:
                return 0;
            default:
                cout << "Invalid choice. Try again.\n";
        }
    }
```

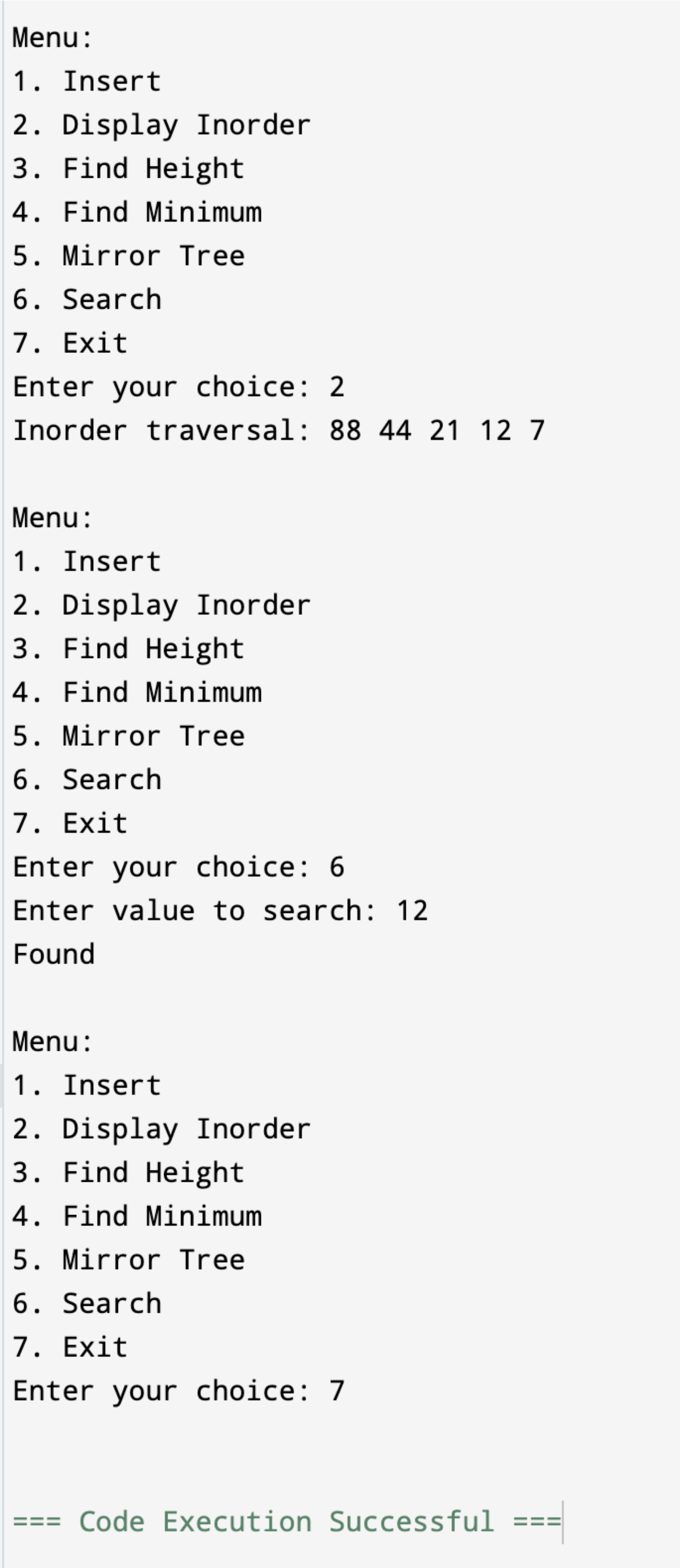
OUTPUT:



```
Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 1
Enter value to insert: 12

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 1
Enter value to insert: 44

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 1
Enter value to insert: 21
```

```
Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 1
Enter value to insert: 88

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 1
Enter value to insert: 7

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 2
Inorder traversal: 7 12 21 44 88
```

```
Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 3
Height of the tree: 3

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 4
Minimum value in the BST: 7

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 5
Tree mirrored.
```



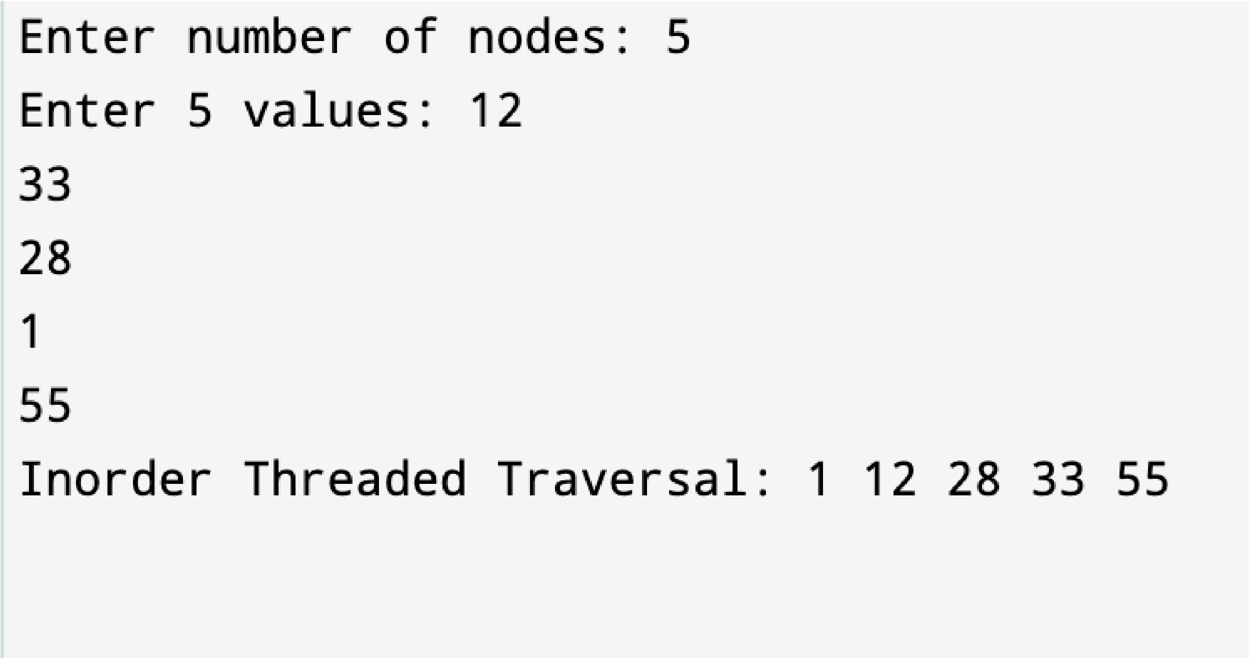
```
Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 2
Inorder traversal: 88 44 21 12 7

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 6
Enter value to search: 12
Found

Menu:
1. Insert
2. Display Inorder
3. Find Height
4. Find Minimum
5. Mirror Tree
6. Search
7. Exit
Enter your choice: 7


=== Code Execution Successful ===
```

ASSIGNMENT 5:

CODE:

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

struct Node {
    int data;
    Node* left = nullptr;
    Node* right = nullptr;
    bool isThreaded = false;

    Node(int val)
    {
        data = val;
    }
};

Node* insert(Node* root, int val)
{
    if (!root)
        return new Node(val);
    if (val < root->data)
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);
    return root;
}

void createThreaded(Node* root, Node*& prev) {
    if (!root)
        return;

    createThreaded(root->left, prev);

    if (!root->left && prev)
   {
        root->left = prev;
        root->isThreaded = true;
    }

    if (prev && !prev->right)
   {
        prev->right = root;
        prev->isThreaded = true;
    }

    prev = root;
    createThreaded(root->right, prev);
}

void inorder(Node* root)
{
    Node* cur = root;
    while (cur && cur->left)
        cur = cur->left;

    while (cur)
 {
        cout << cur->data << " ";

        if (cur->isThreaded)
            cur = cur->right;
        else
       {
            cur = cur->right;
            while (cur && cur->left && !cur->isThreaded)
                cur = cur->left;
        }
    }
}

int main() {
    Node* root = nullptr;
    int n, val;
    cout << "Enter number of nodes: ";
    cin >> n;
    cout << "Enter " << n << " values: ";
    while (n--)
 {
        cin >> val;
        root = insert(root, val);
    }

    Node* prev = nullptr;
    createThreaded(root, prev);

    cout << "Inorder Threaded Traversal: ";
    inorder(root);
    cout << endl;
    return 0;
}
```

OUTPUT:



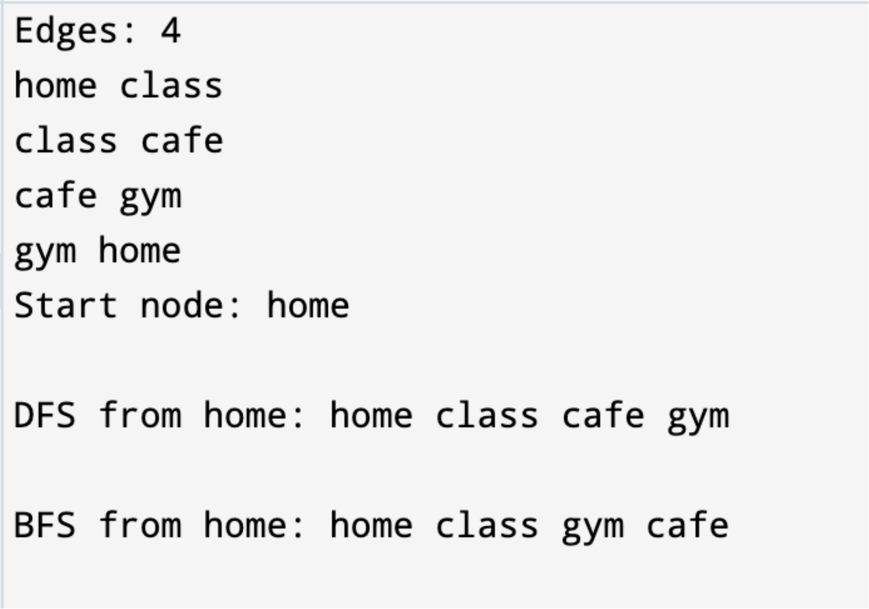
```
Enter number of nodes: 5
Enter 5 values: 12
33
28
1
55
Inorder Threaded Traversal: 1 12 28 33 55
```

ASSIGNMENT 6:

CODE:

```
#include <iostream>
#include <vector>
#include <map>
#include <queue>

using namespace std;

class Graph {
    map<string, vector<string>> adj;
public:
    void addEdge(string u, string v)
    {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void DFS(string start)
    {
        map<string, bool> vis;
        cout << "\nDFS from " << start << ": ";
        DFS_support(start,vis);
        cout << endl;
    }

    void DFS_support(string node, map<string, bool>& vis)
    {
        vis[node] = true;
        cout << node << " ";
        for (auto& nbr : adj[node])
            if (!vis[nbr]) DFS_support(nbr, vis);
    }

    void BFS(string start)
    {
        map<string, bool> vis;
        queue<string> q;
        vis[start] = true;
        q.push(start);

        cout << "\nBFS from " << start << ": ";
        while (!q.empty()) {
            string node = q.front(); q.pop();
            cout << node << " ";
            for (auto& nbr : adj[node])
            {
                if (!vis[nbr])
                {
                    vis[nbr] = true;
                    q.push(nbr);
                }
            }
        }
        cout << endl;
    }
};

int main() {
    Graph g;
    int e;
    cout << "Edges: ";
    cin >> e;
    while (e--)
    {
        string u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }
    string start;
    cout << "Start node: ";
    cin >> start;
    g.DFS(start);
    g.BFS(start);
}
```

OUTPUT:



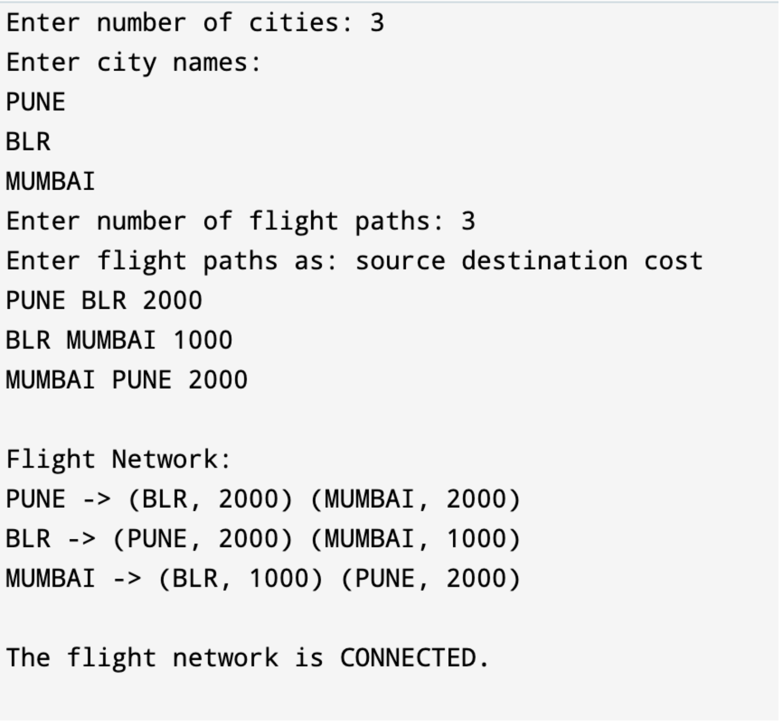
```
Edges: 4
home class
class cafe
cafe gym
gym home
Start node: home

DFS from home: home class cafe gym

BFS from home: home class gym cafe
```

ASSIGNMENT 7:

CODE:

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

map<string, vector<pair<string, int>>> adj;

bool isConnected(const vector<string>& cities) {
    map<string, bool> visited;
    queue<string> q;

    q.push(cities[0]);
    visited[cities[0]] = true;

    while (!q.empty()) {
        string curr = q.front();
        q.pop();

        for (int i = 0; i < adj[curr].size(); i++) {
            if (!visited[adj[curr][i].first]) {
                visited[adj[curr][i].first] = true;
                q.push(adj[curr][i].first);
            }
        }
    }

    for (int i = 0; i < cities.size(); i++) {
        if (!visited[cities[i]])
            return false;
    }
    return true;
}

int main() {
    int n, e;
    cout << "Enter number of cities: ";
    cin >> n;

    vector<string> cities(n);
    cout << "Enter city names:\n";
    for (int i = 0; i < n; i++) {
        cin >> cities[i];
    }

    cout << "Enter number of flight paths: ";
    cin >> e;
    cout << "Enter flight paths as: source destination cost\n";
    for (int i = 0; i < e; i++) {
        string u, v;
        int cost;
        cin >> u >> v >> cost;
        adj[u].push_back({v, cost});
        adj[v].push_back({u, cost});
    }

    cout << "\nFlight Network:\n";
    for (int i = 0; i < cities.size(); i++) {
        string city = cities[i];
        cout << city << " -> ";
        vector<pair<string, int>>& neighbors = adj[city];
        for (int j = 0; j < neighbors.size(); j++) {
            cout << "(" << neighbors[j].first << ", " << neighbors[j].second << ") ";
        }
        cout << "\n";
    }

    if (isConnected(cities))
        cout << "\nThe flight network is CONNECTED.\n";
    else
        cout << "\nThe flight network is NOT connected.\n";

    return 0;
}
```

OUPUT:



```
Enter number of cities: 3
Enter city names:
PUNE
BLR
MUMBAI
Enter number of flight paths: 3
Enter flight paths as: source destination cost
PUNE BLR 2000
BLR MUMBAI 1000
MUMBAI PUNE 2000

Flight Network:
PUNE -> (BLR, 2000) (MUMBAI, 2000)
BLR -> (PUNE, 2000) (MUMBAI, 1000)
MUMBAI -> (BLR, 1000) (PUNE, 2000)

The flight network is CONNECTED.
```

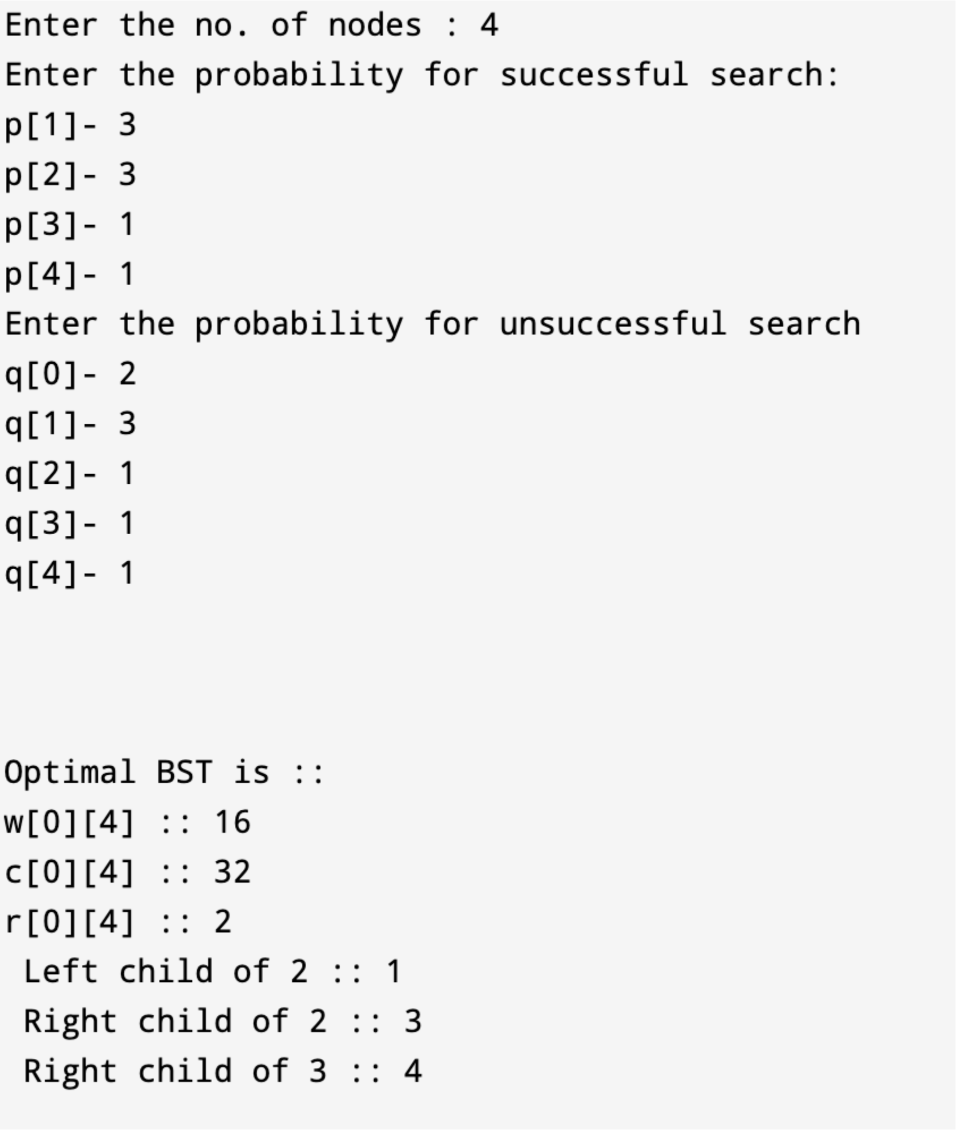
ASSIGNMENT 8:

CODE:

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

void con_obst(void);
void print(int,int);
float a[20],b[20],wt[20][20],c[20][20];
int r[20][20],n;

int main()
  {
int i;
cout<<"\nEnter the no. of nodes : ";
cin>>n;cout<<"\nEnter the probability for successful search :: ";
for(i=1;i<=n;i++)
 {
cout<<"p["<<i<<"]";
cin>>a[i];
 }
cout<<"\nEnter the probability for unsuccessful search :: ";
for(i=0;i<=n;i++)
 {
cout<<"q["<<i<<"]";
cin>>b[i];
 }
con_obst();
print(0,n);
cout<<endl;
}

void con_obst(void)
{
int i,j,k,l,min;
for(i=0;i<n;i++)
 { //Initialisation
c[i][i]=0.0;
r[i][i]=0;
wt[i][i]=b[i];
wt[i][i+1]=b[i]+b[i+1]+a[i+1];
c[i][i+1]=b[i]+b[i+1]+a[i+1];
r[i][i+1]=i+1;
 }

c[n][n]=0.0;
r[n][n]=0;
wt[n][n]=b[n];

for(i=2;i<=n;i++)
 {
for(j=0;j<=n-i;j++)
 {
wt[j][j+i]=b[j+i]+a[j+i]+wt[j][j+i-1];
c[j][j+i]=9999;
for(l=j+1;l<=j+i;l++)
 {
if(c[j][j+i]>(c[j][l-1]+c[l][j+i]))
 {
c[j][j+i]=c[j][l-1]+c[l][j+i];
r[j][j+i]=l;
 }
 }
c[j][j+i]+=wt[j][j+i];
 }
cout<<endl;
 }
cout<<"\n\nOptimal BST is :: ";
cout<<"\nw[0]["<<n<<"] :: "<<wt[0][n];
cout<<"\nc[0]["<<n<<"] :: "<<c[0][n];
cout<<"\nr[0]["<<n<<"] :: "<<r[0][n];
  }
void print(int l1,int r1)
  {
if(l1>=r1)
return;
if(r[l1][r[l1][r1]-1]!=0)
cout<<"\n Left child of "<<r[l1][r1]<<" :: "<<r[l1][r[l1][r1]-1];
if(r[r[l1][r1]][r1]!=0)
```

```
cout<<"\n Right child of "<<r[l1][r1]<<" :: "<<r[r[l1][r1]][r1];
print(l1,r[l1][r1]-1);
print(r[l1][r1],r1);
return;
}
```

OUPUT:



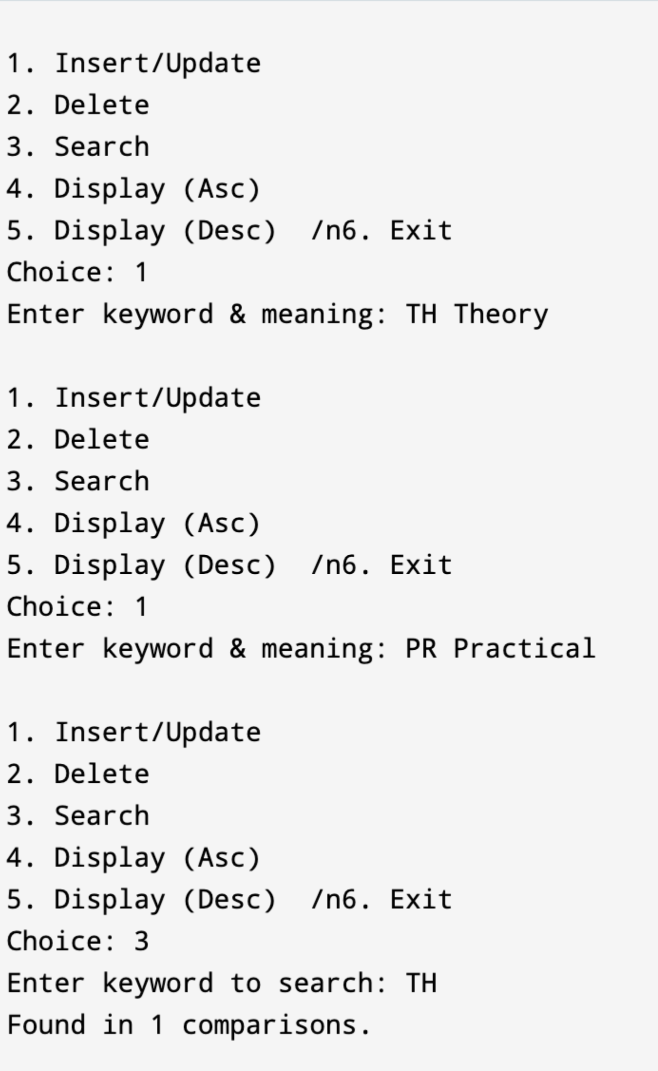
```
Enter the no. of nodes : 4
Enter the probability for successful search:
p[1]- 3
p[2]- 3
p[3]- 1
p[4]- 1
Enter the probability for unsuccessful search
q[0]- 2
q[1]- 3
q[2]- 1
q[3]- 1
q[4]- 1



Optimal BST is ::
w[0][4] :: 16
c[0][4] :: 32
r[0][4] :: 2
 Left child of 2 :: 1
 Right child of 2 :: 3
 Right child of 3 :: 4
```

ASSIGNMENT 9:

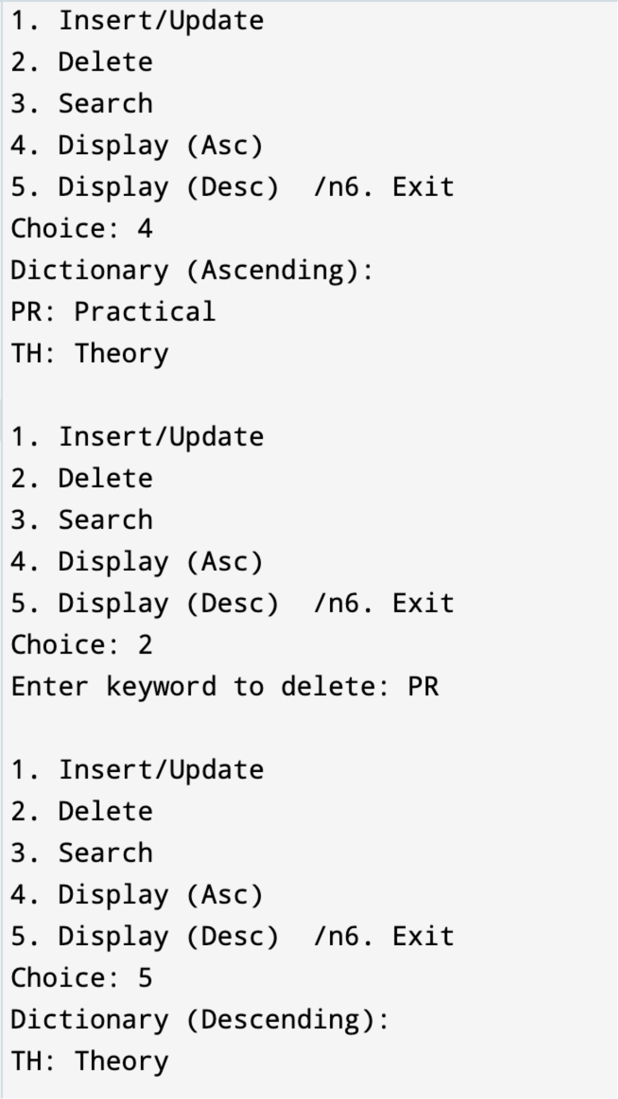
CODE:

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

struct Node {
    string key, meaning;
    int height;
    Node *left, *right;
    Node(string k, string m)
        {
            key = k;
            meaning = m;
            height = 1;
            left = nullptr;
            right = nullptr;
        }
};

int height(Node* n)
{
    if (n == nullptr):
        return 0;
    return n->height;
}
Node* rotateRight(Node* y) {
    Node* x = y->left, *T2 = x->right;
    x->right = y; y->left = T2;
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
    return x;
}

Node* rotateLeft(Node* x) {
    Node* y = x->right, *T2 = y->left;
    y->left = x; x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}

Node* balance(Node* root) {
    if (!root) return root;
    root->height = max(height(root->left), height(root->right)) + 1;
    int bf = height(root->left) - height(root->right);

    if (bf > 1 && root->key > root->left->key) return rotateRight(root);
    if (bf < -1 && root->key < root->right->key) return rotateLeft(root);
    if (bf > 1) { root->left = rotateLeft(root->left); return rotateRight(root); }
    if (bf < -1) { root->right = rotateRight(root->right); return rotateLeft(root); }

    return root;
}

Node* insert(Node* root, string key, string meaning) {
    if (!root) return new Node(key, meaning);
    if (key < root->key) root->left = insert(root->left, key, meaning);
    else if (key > root->key) root->right = insert(root->right, key, meaning);
    else root->meaning = meaning;
    return balance(root);
}

Node* findMin(Node* root) {
    while (root->left) root = root->left;
    return root;
}

Node* remove(Node* root, string key) {
    if (!root) return nullptr;
    if (key < root->key) root->left = remove(root->left, key);
    else if (key > root->key) root->right = remove(root->right, key);
    else {
        if (!root->left) return root->right;
        if (!root->right) return root->left;
        Node* temp = findMin(root->right);
        root->key = temp->key; root->meaning = temp->meaning;
        root->right = remove(root->right, temp->key);
    }
    return balance(root);
}

Node* search(Node* root, string key, int &comparisons) {
    comparisons = 0;
    while (root) {
        comparisons++;
        if (key == root->key) return root;
        root = (key < root->key) ? root->left : root->right;
    }
    return nullptr;
}

void inorder(Node* root) {
    if (!root) return;
    inorder(root->left);
    cout << root->key << ": " << root->meaning << endl;
    inorder(root->right);
}

void reverseInorder(Node* root) {
    if (!root) return;
    reverseInorder(root->right);
    cout << root->key << ": " << root->meaning << endl;
    reverseInorder(root->left);
}

int main() {
    Node* root = nullptr;
    int choice, comparisons;
    string key, meaning;

    while (true) {
        cout << "\n1. Insert/Update  2. Delete  3. Search  4. Display (Asc)  5. Display (Desc)  6. Exit\nChoice: ";
        cin >> choice;
        if (choice == 6) break;
        switch (choice) {
            case 1: cout << "Enter keyword & meaning: "; cin >> key; cin.ignore(); getline(cin,
meaning);
                    root = insert(root, key, meaning); break;
            case 2: cout << "Enter keyword to delete: "; cin >> key;
                    root = remove(root, key); break;
            case 3: cout << "Enter keyword to search: "; cin >> key;
                    if (search(root, key, comparisons))
                        cout << "Found in " << comparisons << " comparisons.\n";
                    else
                        cout << "Not found in " << comparisons << " comparisons.\n";
                    break;
            case 4: cout << "Dictionary (Ascending):\n"; inorder(root); break;
            case 5: cout << "Dictionary (Descending):\n"; reverseInorder(root); break;
        }
    }
    return 0;
}
```

OUTPUT:



```

1. Insert/Update
2. Delete
3. Search
4. Display (Asc)
5. Display (Desc)  /n6. Exit
Choice: 1
Enter keyword & meaning: TH Theory

1. Insert/Update
2. Delete
3. Search
4. Display (Asc)
5. Display (Desc)  /n6. Exit
Choice: 1
Enter keyword & meaning: PR Practical

1. Insert/Update
2. Delete
3. Search
4. Display (Asc)
5. Display (Desc)  /n6. Exit
Choice: 3
Enter keyword to search: TH
Found in 1 comparisons.
```



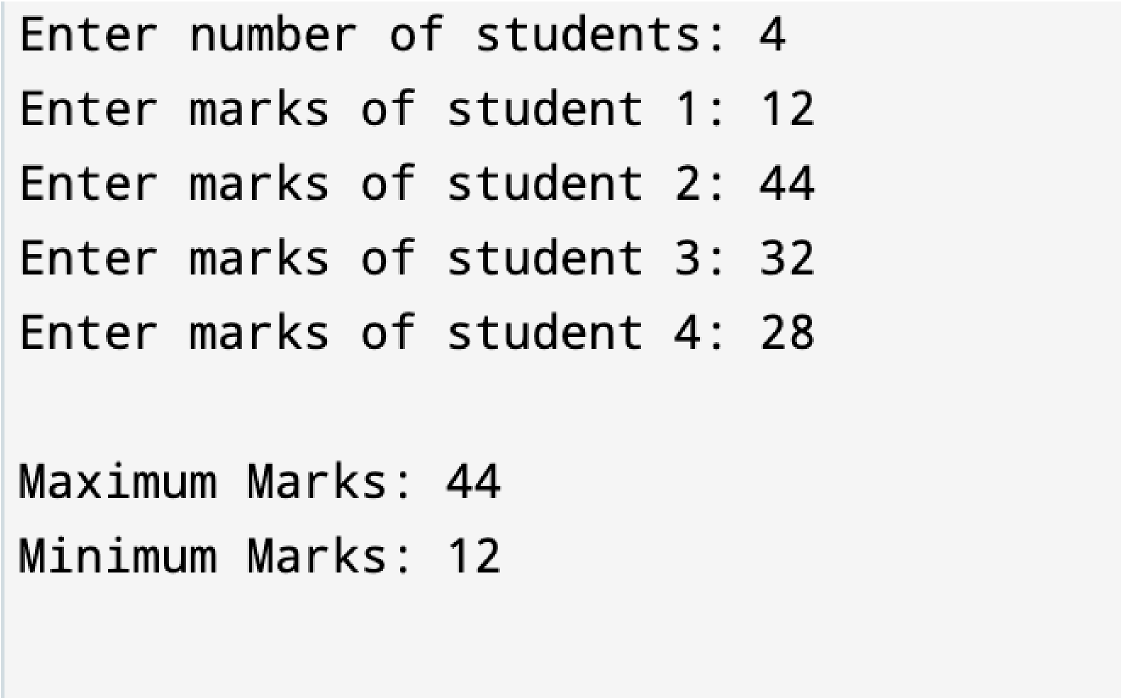
```
1. Insert/Update
2. Delete
3. Search
4. Display (Asc)
5. Display (Desc)  /n6. Exit
Choice: 4
Dictionary (Ascending):
PR: Practical
TH: Theory

1. Insert/Update
2. Delete
3. Search
4. Display (Asc)
5. Display (Desc)  /n6. Exit
Choice: 2
Enter keyword to delete: PR

1. Insert/Update
2. Delete
3. Search
4. Display (Asc)
5. Display (Desc)  /n6. Exit
Choice: 5
Dictionary (Descending):
TH: Theory
```

ASSIGNMENT 10:

CODE:

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

void buildMaxHeap(vector<int>& arr)
{
   make_heap(arr.begin(), arr.end());

}
void buildMinHeap(vector<int>& arr)
{
   make_heap(arr.begin(), arr.end(), greater<int>());
}
int main()
{
   int n;
   cout << "Enter number of students: ";
   cin >> n;
   vector<int> marks(n);
   for (int i = 0; i < n; i++)
   {
      cout << "Enter marks of student " << i + 1 << ": ";
      cin >> marks[i];
   }

   vector<int> maxHeap = marks;
   buildMaxHeap(maxHeap);
   cout << "\nMaximum Marks: " << maxHeap.front() << endl;

   vector<int> minHeap = marks;
   buildMinHeap(minHeap);
   cout << "Minimum Marks: " << minHeap.front() << endl;

   return 0;
}
```

OUTPUT:



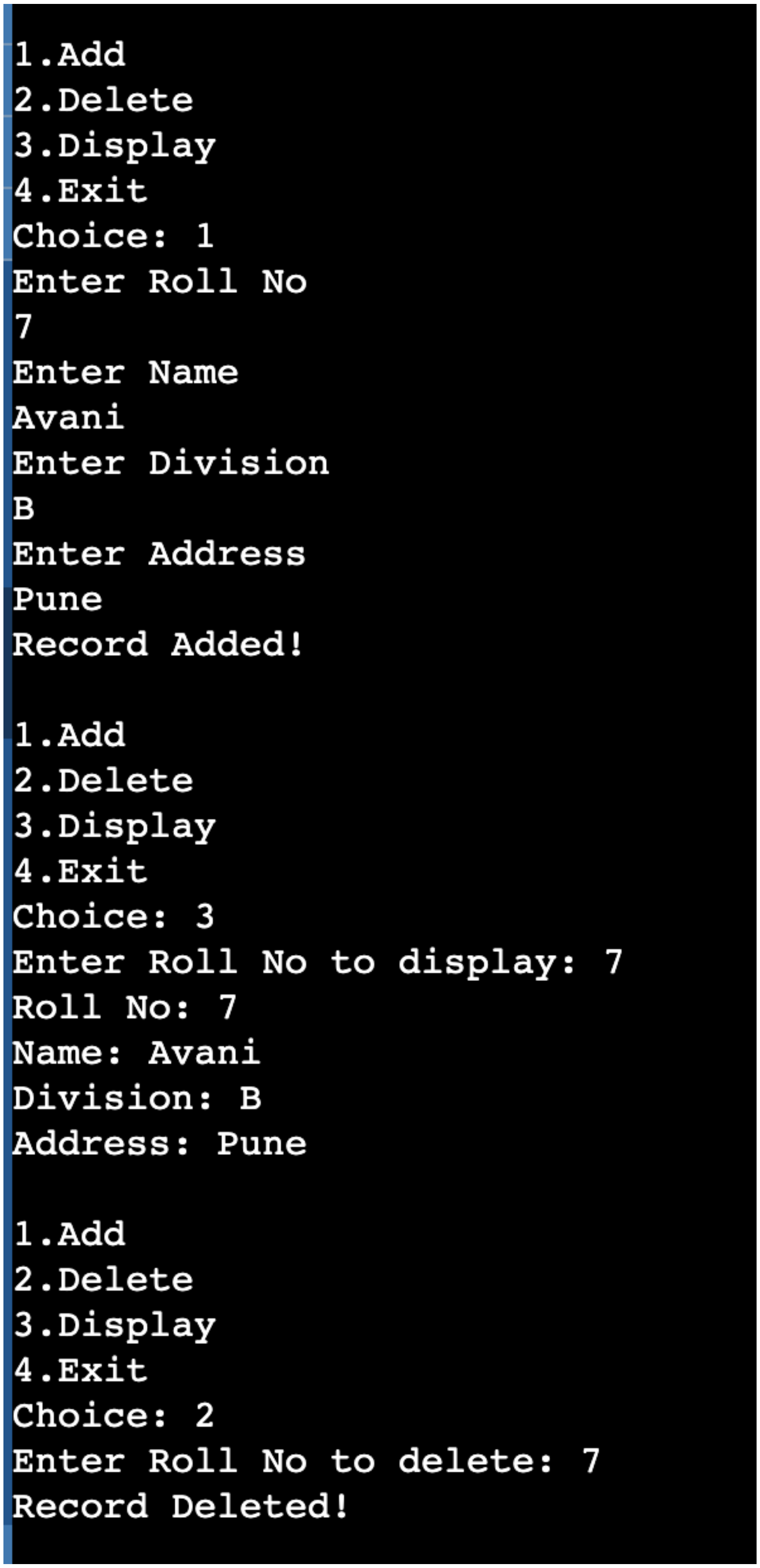
```
Enter number of students: 4
Enter marks of student 1: 12
Enter marks of student 2: 44
Enter marks of student 3: 32
Enter marks of student 4: 28

Maximum Marks: 44
Minimum Marks: 12
```

ASSIGNMENT 11:

CODE:

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

void addStudent() {
    ofstream file("students.txt", ios::app);
    string roll, name, div, addr;
    cout << "Enter Roll No"<<endl;
    cin >> roll ;
    cout << "Enter Name"<<endl;
    cin >> name ;
    cout << "Enter Division"<<endl;
    cin>> div;
    cout << "Enter Address"<<endl;
    cin>> addr;
    file << roll << " " << name << " " << div << " " << addr << "\n";
    cout << "Record Added!\n";
}

void deleteStudent() {
    string roll, r, n, d, a;
    bool found = false;
    cout << "Enter Roll No to delete: ";
    cin >> roll;
    ifstream in("students.txt");
    ofstream out("temp.txt");
    while (in >> r >> n >> d >> a) {
        if (r != roll) out << r << " " << n << " " << d << " " << a << "\n";
        else found = true;
    }
    in.close();
    out.close();
    if (found)
    {
        remove("students.txt");
        rename("temp.txt", "students.txt");
        cout << "Record Deleted!\n";
    }
    else
    {
        remove("temp.txt");
        cout << "Record Not Found!\n";
    }
}

void displayStudent() {
    string roll, r, n, d, a;
    bool found = false;
    cout << "Enter Roll No to display: ";
    cin >> roll;
    ifstream in("students.txt");
    while (in >> r >> n >> d >> a) {
        if (r == roll) {
            cout << "Roll No: " << r << "\nName: " << n << "\nDivision: " << d << "\nAddress: " <<
a << "\n";
            found = true; break;
        }
    }
    if (!found)
    cout << "Record Not Found!\n";
}

int main() {
    int choice;
    do {
        cout << "\n1.Add\n2.Delete\n3.Display\n4.Exit\nChoice: ";
        cin >> choice;
        switch (choice) {
            case 1: addStudent(); break;
            case 2: deleteStudent(); break;
            case 3: displayStudent(); break;
            case 4: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while (choice != 4);
    return 0;
}
```

OUTPUT:



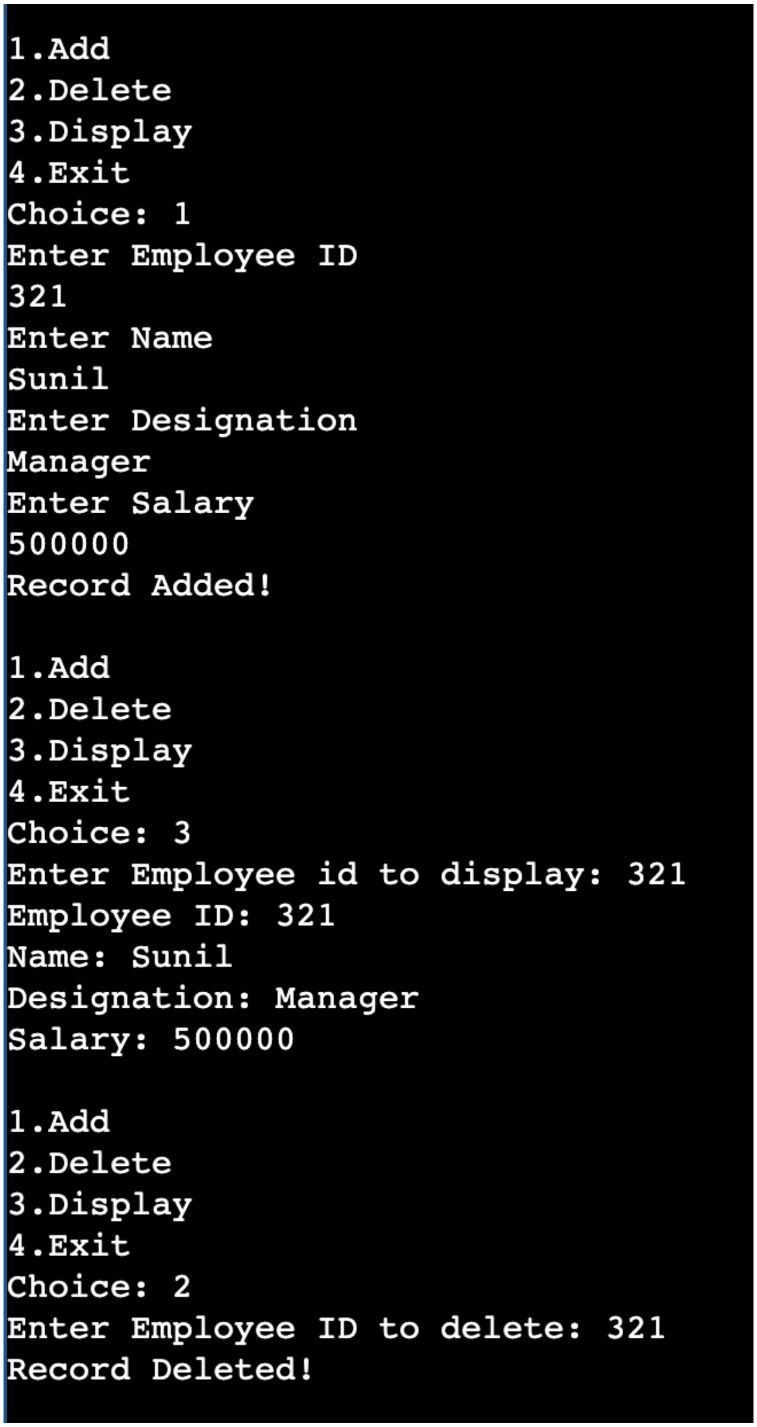
```
1.Add
2.Delete
3.Display
4.Exit
Choice: 1
Enter Roll No
7
Enter Name
Avani
Enter Division
B
Enter Address
Pune
Record Added!

1.Add
2.Delete
3.Display
4.Exit
Choice: 3
Enter Roll No to display: 7
Roll No: 7
Name: Avani
Division: B
Address: Pune

1.Add
2.Delete
3.Display
4.Exit
Choice: 2
Enter Roll No to delete: 7
Record Deleted!
```

ASSIGNMENT 12:

CODE:

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

void addEmployee() {
    ofstream file("employee.txt", ios::app);
    string id, name, des, sal;
    cout << "Enter Employee ID"<<endl;
    cin >> id;
    cout << "Enter Name"<<endl;
    cin >> name ;
    cout << "Enter Designation"<<endl;
    cin>> des;
    cout << "Enter Salary"<<endl;
    cin>> sal;
    file << id << " " << name << " " << des<< " " << sal << "\n";
    cout << "Record Added!\n";
}

void deleteEmployee()
 {
    string id, i, n, d, s;
    bool found = false;
    cout << "Enter Employee ID to delete: ";
    cin >> id;
    ifstream in("employee.txt");
    ofstream out("temp.txt");
    while (in >> i >> n >> d >> s)
 {
        if (r != id)
                out << i << " " << n << " " << d << " " << s << "\n";
        else
                found = true;
    }
    in.close();
    out.close();
    if (found)
    {
        remove("employee.txt");
        rename("temp.txt", "employee.txt");
        cout << "Record Deleted!\n";
    }
    else
    {
        remove("temp.txt");
        cout << "Record Not Found!\n";
    }
}

void displayEmployee() {
    string id, i, n, d, s;
    bool found = false;
    cout << "Enter Employee id to display: ";
    cin >> id;
    ifstream in("employee.txt");
    while (in >> i >> n >> d >> s) {
        if (r == id) {
            cout << "Employee ID: " << r << "\nName: " << n << "\nDesignation: " << d <<
"\nSalary: " << a << "\n";
            found = true; break;
        }
    }
    if (!found)
    cout << "Record Not Found!\n";
}

int main() {
    int choice;
    do {
        cout << "\n1.Add\n2.Delete\n3.Display\n4.Exit\nChoice: ";
        cin >> choice;
        switch (choice) {
            case 1: addEmployee(); break;
            case 2: deleteEmployee(); break;
            case 3: displayEmployee(); break;
            case 4: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while (choice != 4);
    return 0;
}
```

OUTPUT:



```
1.Add
2.Delete
3.Display
4.Exit
Choice: 1
Enter Employee ID
321
Enter Name
Sunil
Enter Designation
Manager
Enter Salary
500000
Record Added!

1.Add
2.Delete
3.Display
4.Exit
Choice: 3
Enter Employee id to display: 321
Employee ID: 321
Name: Sunil
Designation: Manager
Salary: 500000

1.Add
2.Delete
3.Display
4.Exit
Choice: 2
Enter Employee ID to delete: 321
Record Deleted!
```