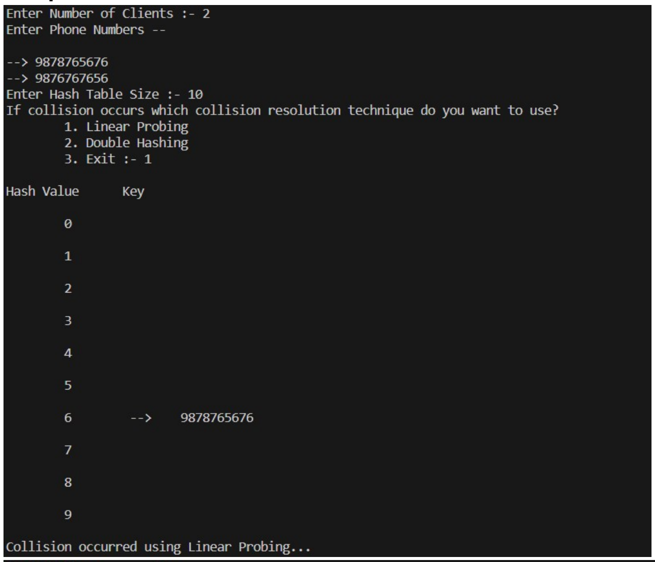
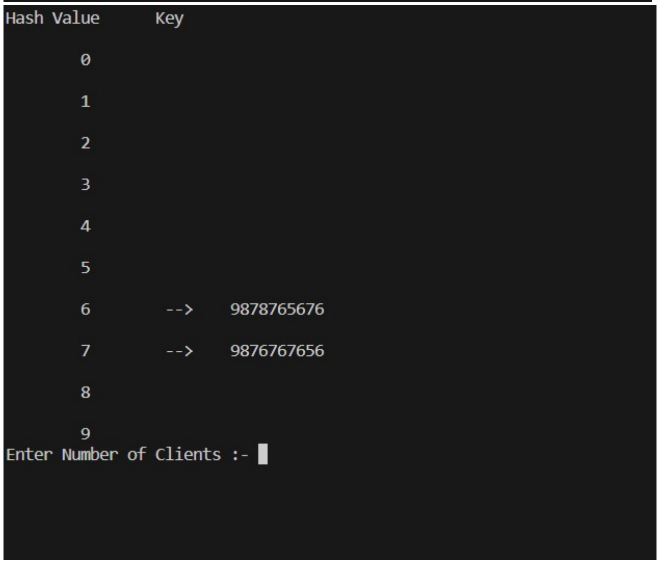
Experiment : 1

```
def tele_database():
    phone_data = []
    n = int(input("Enter Number of Clients :- "))
    print("Enter Phone Numbers --\n")
    for _ in range(n):
        x = int(input("--> "))
        phone_data.append(x)
    return phone_data

def hash_function_1(key_ele, m_size):
    h1 = key_ele % m_size
    return h1

def hash_function_2(key_ele):
    h2 = 7 - (key_ele % 7)
    return h2

def hashtable(ht):
    print(f"\nHash Value \tKey")
    for ele in range(len(ht)):
        if ht[ele] != -1:
            print(f"\n\t{ele} \t --> \t{ht[ele]}")
        else:
            print(f"\n\t{ele}")

# Main function
print("\n\n")

while (1):
    phone_database = tele_database()
    m = int(input("Enter Hash Table Size :- "))
    hash_table = [-1] * m #creates a list with size m, all elements as -1
    opt = int(input("If collision occurs which collision resolution technique do you want
to use?\n\t1. Linear Probing\n\t2. Double Hashing\n\t3. Exit :- "))
    for k in phone_database:
        h_1 = hash_function_1(k, m)
        h_2 = hash_function_2(k)

        if hash_table[h_1] == -1:
            hash_table[h_1] = k
            hashtable(hash_table) # Display the hash table after each insertion
        else:
            if opt == 1:
                print("\nCollision occurred using Linear Probing...")
                temp_h1 = h_1
                while hash_table[temp_h1] != -1:
                    temp_h1 = (temp_h1 + 1) % m # Wrap around if the end of the table is
reached
                    if temp_h1 == h_1: # If we have wrapped around the entire table
                        print("\nHash table is full. Cannot insert more elements.")
                        break
                if hash_table[temp_h1] == -1:
                    hash_table[temp_h1] = k
                hashtable(hash_table)

            elif opt == 2:
                print("\nCollision occurred using Double Hashing...")
                i = 1
                temp_h1 = h_1
                while hash_table[temp_h1] != -1:
                    temp_h1 = (h_1 + (i * h_2)) % m # Wrap around if the end of the table
is reached
                    if temp_h1 == h_1: # If we have wrapped around the entire table
                        print("\nHash table is full. Cannot insert more elements.")
                        break
                    i += 1
                if hash_table[temp_h1] == -1:
                    hash_table[temp_h1] = k
                    hashtable(hash_table)

            elif opt == 3:
                print("\n*** TERMINATED SUCCESSFULLY ***")
                exit(0)
            else:
                print("\nInvalid option. Please choose 1, 2, or 3.")
                break # Break out of the inner loop to re-enter the choice of collision
resolution
```

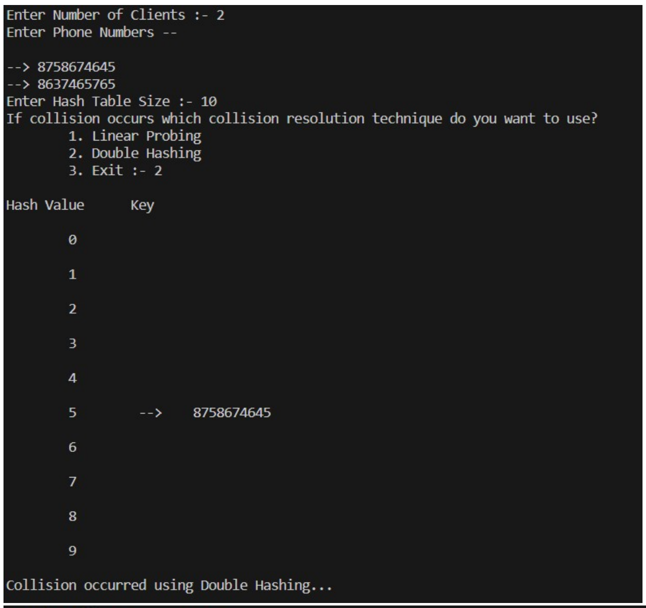
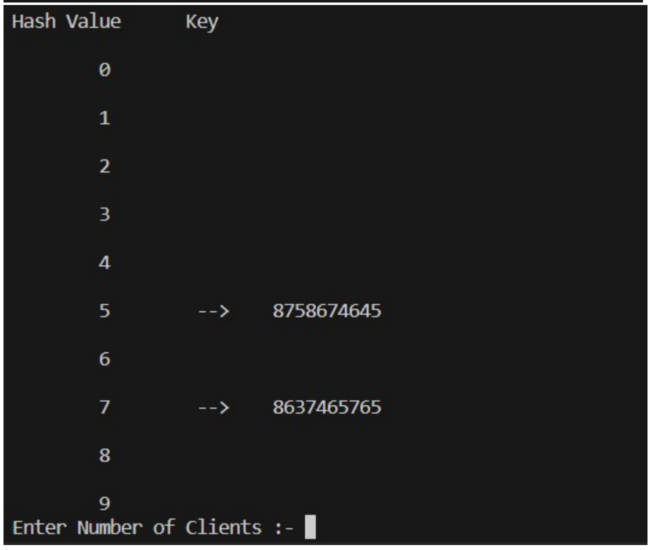
Output :



```
Enter Number of Clients :- 2
Enter Phone Numbers --

--> 9878765676
--> 9876767656
Enter Hash Table Size :- 10
If collision occurs which collision resolution technique do you want to use?
        1. Linear Probing
        2. Double Hashing
        3. Exit :- 1

Hash Value      Key

        0

        1

        2

        3

        4

        5

        6       -->     9878765676

        7

        8

        9

Collision occurred using Linear Probing...
```



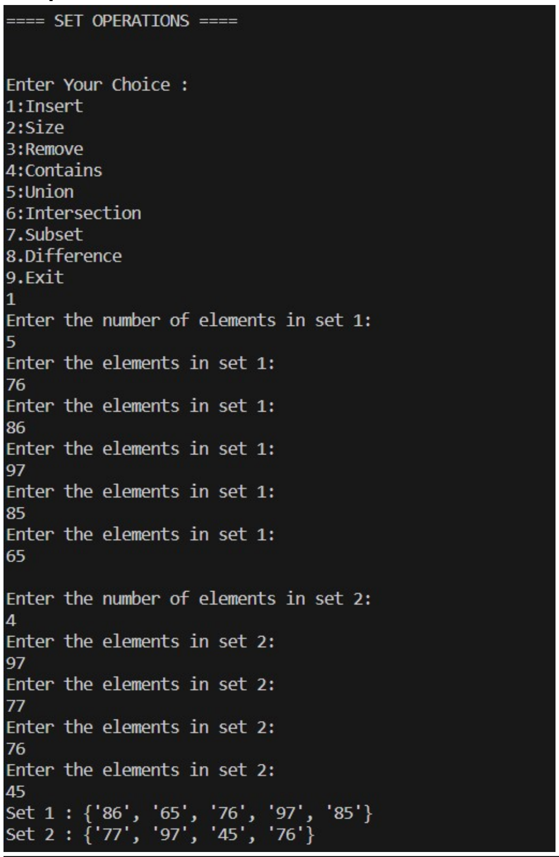
```
Hash Value      Key

        0

        1

        2

        3

        4

        5

        6       -->     9878765676

        7       -->     9876767656

        8

        9
Enter Number of Clients :- 
```


```
Enter Number of Clients :- 2
Enter Phone Numbers --

--> 8758674645
--> 8637465765
Enter Hash Table Size :- 10
If collision occurs which collision resolution technique do you want to use?
        1. Linear Probing
        2. Double Hashing
        3. Exit :- 2

Hash Value      Key

        0

        1

        2

        3

        4

        5        -->    8758674645

        6

        7

        8

        9

Collision occurred using Double Hashing...
```

```
Hash Value      Key

        0

        1

        2

        3

        4

        5        -->    8758674645

        6

        7        -->    8637465765

        8

        9
Enter Number of Clients :-
```

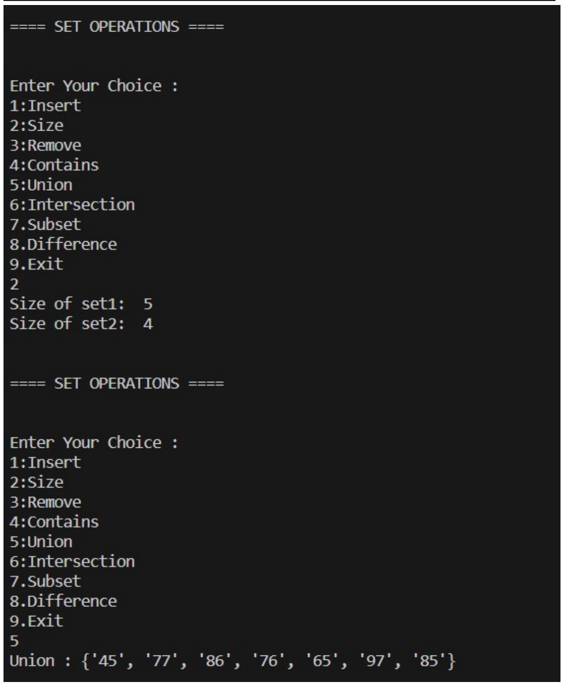
Experiment : 2

```
def main():
    set1 = set()
    set2 = set()
    print("\n\n")
    while(True):
        print("\n\n==== SET OPERATIONS ====\n")
        print("\nEnter Your Choice :")
        choice = int(input("1:Insert \n2:Size \n3:Remove \n4:Contains \n5:Union 
\n6:Intersection \n7.Subset \n8.Difference \n9.Exit\n"))

        if (choice == 1):
            n1 = int(input("Enter the number of elements in set 1: \n"))
            for i in range(n1):
                data_name = input("Enter the elements in set 1: \n")

                set1.add(data_name)
            n2 = int(input("\nEnter the number of elements in set 2: \n"))
            for i in range(n2):
                data_name = input("Enter the elements in set 2: \n")

                set2.add(data_name)

            print("Set 1 :",set1,'\nSet 2 :',set2)

        elif (choice == 2):
            print("Size of set1: ", len(set1))
            print("Size of set2: ", len(set2))

        elif (choice == 3):
            print('Remove element from set 1 and set 2 (1 for set 1 & 2 for set2)\n')
            inp = int(input())
            if inp==1:
                set1.pop()
                print(set1)
            if inp==2:
                set2.pop()
                print(set2)


        elif (choice == 4):
            ip = input("Enter element you want to check: ")
            if ip in set1:
                print("set 1 contains the element ",ip)
            if ip in set2:
                print("set 2 contains the element ",ip)

        elif (choice == 5):
            print("Union :", set1.union(set2))

        elif (choice == 6):
            print("Intersection: ", set1.intersection(set2))

        elif (choice == 7):
            if(set1.issubset(set2)):

                print("Set 1 is Subset of set 2")
            else:
                print("Set 1 is Not a Subset of Set 2")
        elif (choice == 8):
            differ = set2.difference(set1)
            print(differ)

        else:
            exit()
main()
```

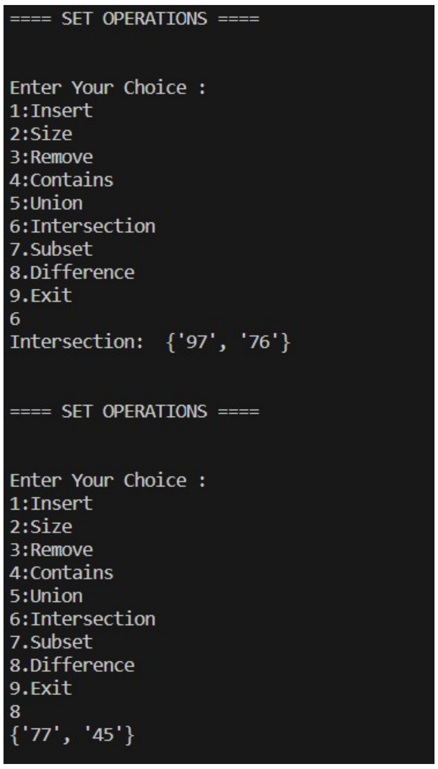
Output:



```
==== SET OPERATIONS ====


Enter Your Choice :
1:Insert
2:Size
3:Remove
4:Contains
5:Union
6:Intersection
7.Subset
8.Difference
9.Exit
1
Enter the number of elements in set 1:
5
Enter the elements in set 1:
76
Enter the elements in set 1:
86
Enter the elements in set 1:
97
Enter the elements in set 1:
85
Enter the elements in set 1:
65

Enter the number of elements in set 2:
4
Enter the elements in set 2:
97
Enter the elements in set 2:
77
Enter the elements in set 2:
76
Enter the elements in set 2:
45
Set 1 : {'86', '65', '76', '97', '85'}
Set 2 : {'77', '97', '45', '76'}
```



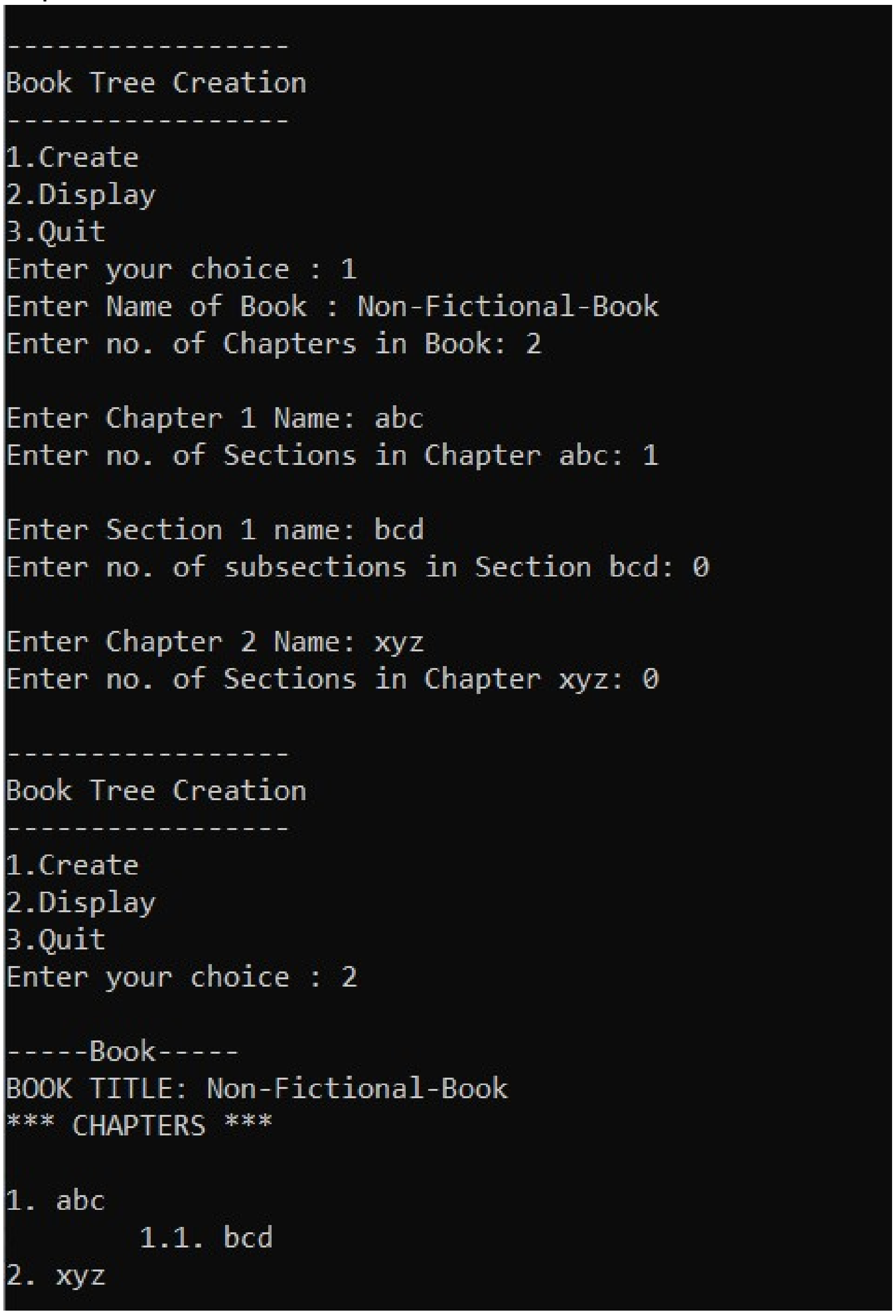
```
==== SET OPERATIONS ====


Enter Your Choice :
1:Insert
2:Size
3:Remove
4:Contains
5:Union
6:Intersection
7.Subset
8.Difference
9.Exit
2
Size of set1:  5
Size of set2:  4


==== SET OPERATIONS ====


Enter Your Choice :
1:Insert
2:Size
3:Remove
4:Contains
5:Union
6:Intersection
7.Subset
8.Difference
9.Exit
5
Union : {'45', '77', '86', '76', '65', '97', '85'}
```



```
==== SET OPERATIONS ====


Enter Your Choice :
1:Insert
2:Size
3:Remove
4:Contains
5:Union
6:Intersection
7.Subset
8.Difference
9.Exit
6
Intersection:  {'97', '76'}


==== SET OPERATIONS ====


Enter Your Choice :
1:Insert
2:Size
3:Remove
4:Contains
5:Union
6:Intersection
7.Subset
8.Difference
9.Exit
8
{'77', '45'}
```

Experiment : 3

```
#include <iostream>
#include <cstdlib>
#include <string.h>
using namespace std;

struct node
{
    char label[30];
    int ch_count;
    int sb_count;
    struct node *child[30];
} *root;

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 << "\nEnter Chapter " << i + 1 << " Name: ";
        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 << "\nEnter Section " << j + 1 << " name: ";
            cin >> root->child[i]->child[j]->label;
            cout << "Enter no. of subsections in Section " << root->child[i]->child[j]-
>label << ": ";
            cin >> root->child[i]->child[j]->sb_count;

            for (k = 0; k < root->child[i]->child[j]->sb_count; k++)
            {
                root->child[i]->child[j]->child[k] = new node;
                cout << "\nEnter Subsection " << k + 1 << " name: ";
                cin >> root->child[i]->child[j]->child[k]->label;
            }
        }
    }
}

void BST::display(node *r1)
{
    int i, j, k, tchapters;
    if (r1 != NULL)
    {
        cout << "\n-----Book-----\n";
        cout << "BOOK TITLE: " << r1->label;
        cout << "\n*** CHAPTERS ***" << endl;
        tchapters = r1->ch_count;

        for (i = 0; i < tchapters; i++)
        {
            cout << "\n"
                 << i + 1 << ". " << r1->child[i]->label;

            for (j = 0; j < r1->child[i]->ch_count; j++)
            {
                cout << "\n\t" << i + 1 << "." << j + 1 << ". " << r1->child[i]->child[j]-
>label;

                for (k = 0; k < r1->child[i]->child[j]->sb_count; k++)
                {
                    cout << "\n\t\t" << i + 1 << "." << j + 1 << "." << k + 1 << ". " <<
r1->child[i]->child[j]->child[k]->label;
                }
            }
        }
    }
}

int main()
{
    int choice;
    BST bst;
    cout << "\n\n" << endl;
    while (1)
    {
        cout << "\n-----------------" << 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();
            break;
        case 2:
            bst.display(root);
            break;
        case 3:
            exit(1);
        default:
            cout << "\nWrong Choice!" << endl;
        }
    }
}
```
Output:

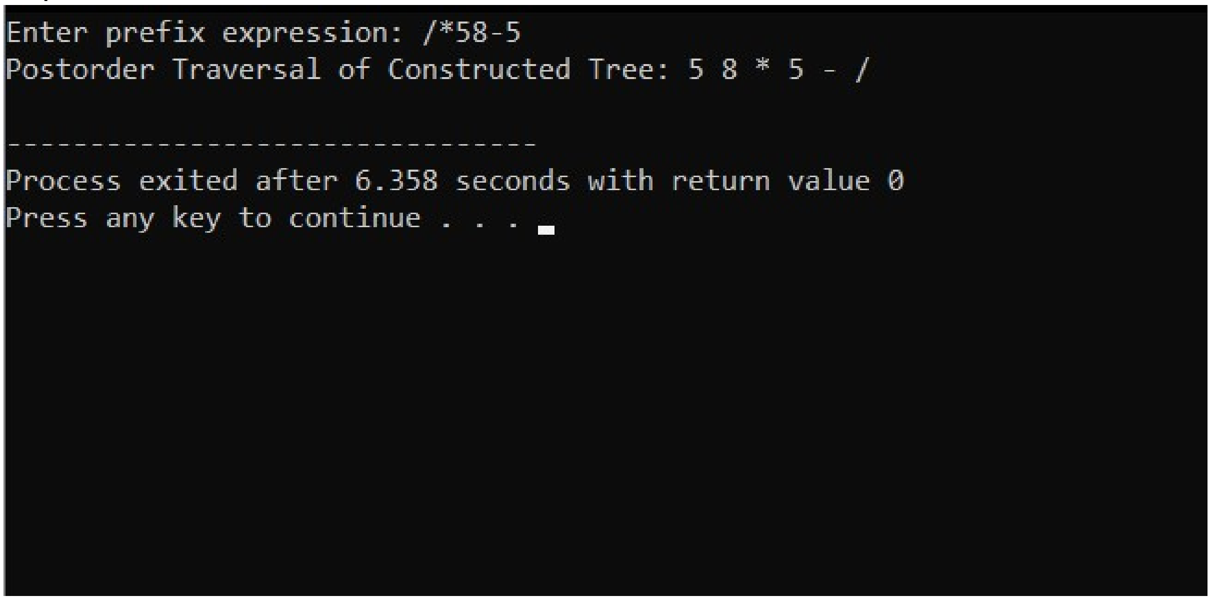


```
-----------------
Book Tree Creation
-----------------
1.Create
2.Display
3.Quit
Enter your choice : 1
Enter Name of Book : Non-Fictional-Book
Enter no. of Chapters in Book: 2

Enter Chapter 1 Name: abc
Enter no. of Sections in Chapter abc: 1

Enter Section 1 name: bcd
Enter no. of subsections in Section bcd: 0

Enter Chapter 2 Name: xyz
Enter no. of Sections in Chapter xyz: 0

-----------------
Book Tree Creation
-----------------
1.Create
2.Display
3.Quit
Enter your choice : 2

-----Book-----
BOOK TITLE: Non-Fictional-Book
*** CHAPTERS ***

1. abc
        1.1. bcd
2. xyz
```

Experiment : 4

```
#include <iostream>
#include <stack>
using namespace std;
// Structure for a Node
struct Node
{
 char data;
 Node *left, *right;

 Node(char val) {
 data = val;
 left = right = NULL;
 }
};
// Function to traverse the tree in postorder
void postOrder(Node* root)
{
 if (root == NULL)
 { return;
 }
 postOrder(root->left);
 postOrder(root->right);
 cout << root->data << " ";
}
// Function to construct a binary tree from a given prefix expression
Node* constructTreeFromPrefix(string prefix)
{
 stack<Node*> st;
 int p= prefix.length();

 for (int i = p- 1; i >= 0; i--)
 {
 Node* newNode = new Node(prefix[i]);

 // If operator, pop two nodes and attach them as children
 if (prefix[i] == '+'|| prefix[i] == '-'|| prefix[i] == '*' || prefix[i]== '/')
 {
 if (!st.empty())
 { newNode->left = st.top();
 st.pop();
 }
 if (!st.empty())
 { newNode->right = st.top();
 st.pop();
 }
 }

 st.push(newNode);
 }
 return st.empty() ? NULL : st.top();
}
// Function to insert a node in the binary tree
Node* insert(Node* root, char data)
{
 if (root == NULL)
 return new Node(data);

 if (data < root->data)
 root->left = insert(root->left, data);

 else
 root->right = insert(root->right, data);
 return root;
}
int main()
{
 string prefix;
 cout << "Enter prefix expression: ";
 cin >> prefix;

 Node* root = constructTreeFromPrefix(prefix);

 cout << "Postorder Traversal of Constructed Tree: ";
 postOrder(root);
 cout << endl;
 return 0;
}
```

Output:



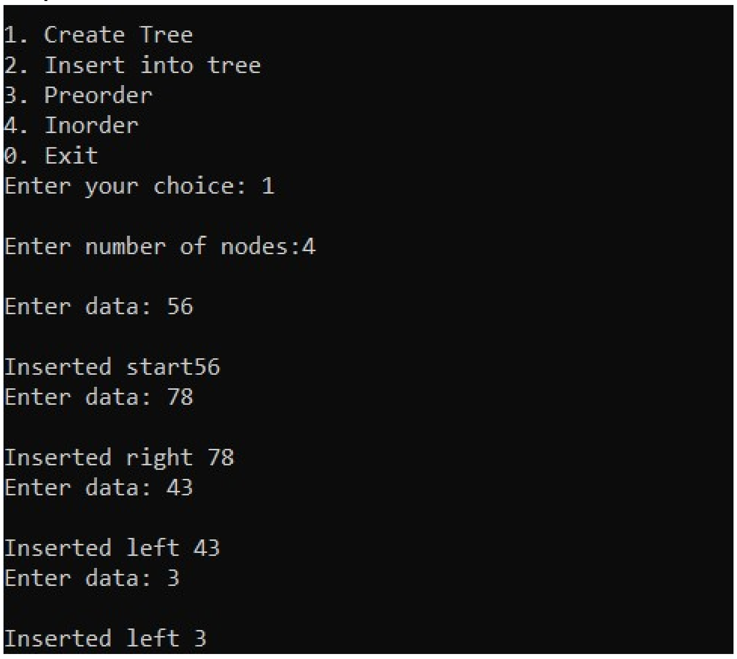
```
Enter prefix expression: /*58-5
Postorder Traversal of Constructed Tree: 5 8 * 5 - /

--------------------------------
Process exited after 6.358 seconds with return value 0
Press any key to continue . . .
```

Experiment : 5

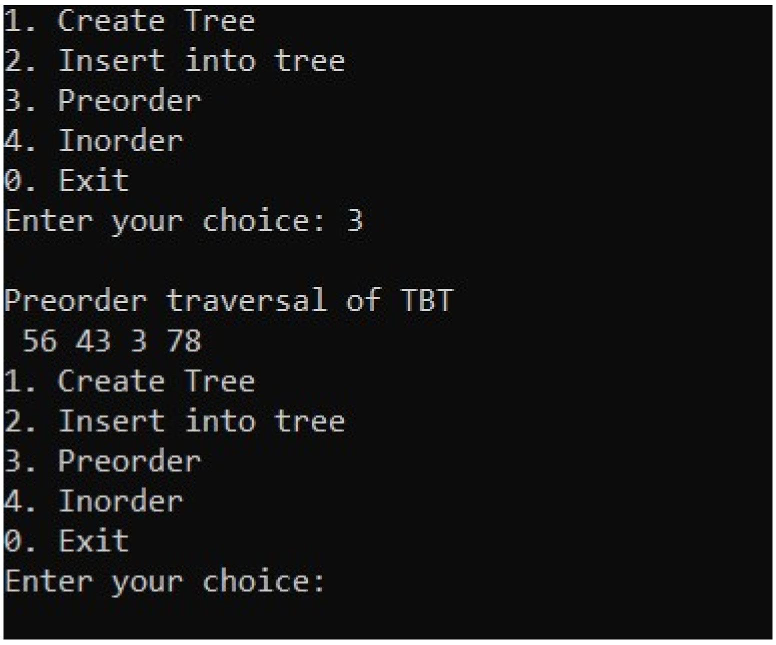
```
#include <iostream>
using namespace std;
class TBT;
class node
{node *left, *right;
    int data;
    bool rbit, lbit;
public:
    node()
    {left = NULL;
        right = NULL;
        rbit = lbit = 0;
    }node(int d)
    {left = NULL;
        right = NULL;
        rbit = lbit = 0;
        data = d;
    }friend class TBT;
};
class TBT
{node *root; // acts as a dummy node
public:
    TBT()
    { // dummy node initialization
        root = new node(9999);
        root->left = root;
        root->rbit = 1;
        root->lbit = 0;
        root->right = root;
    }void create();
    void insert(int data);
    node *inorder_suc(node *);
    void inorder_traversal();
    node *preorder_suc(node *);
    void preorder_traversal();
};
void TBT::preorder_traversal()
{    node *c = root->left;
    while (c != root)
    {        cout << " " << c->data;
        c = preorder_suc(c);
    }}
void TBT::inorder_traversal()
{    node *c = root->left;
    while (c->lbit == 1)
        c = c->left;
    while (c != root)
    {        cout << " " << c->data;
        c = inorder_suc(c);
    }}
node *TBT::inorder_suc(node *c)
{    if (c->rbit == 0)
        return c->right;
    else
        c = c->right;
    while (c->lbit == 1)
    {c = c->left;
    }    return c;}
```

```
node *TBT::preorder_suc(node *c)
{    if (c->lbit == 1)
    {        return c->left;
    }
    while (c->rbit == 0)
    {        c = c->right;
    }    return c->right;}
void TBT::create()
{    int n;
    if (root->left == root && root->right == root)
    {        cout << "\nEnter number of nodes:";
        cin >> n;
        for (int i = 0; i < n; i++)
        {            int info;
            cout << "\nEnter data: ";
            cin >> info;
            this->insert(info);
        }    }
    else    {
        cout << "\nTree is Already created.\n";
    }}
void TBT::insert(int data)
{    if (root->left == root && root->right == root)
    {         node *p = new node(data);
        p->left = root->left;
        p->lbit = root->lbit; // 0
        p->rbit = 0;
        p->right = root->right;
        root->left = p;
        root->lbit = 1;
        cout << "\nInserted start" << data;
        return;    }
    node *cur = root->left;
    while (1)
    {        if (cur->data < data)
        { // insert right
            node *p = new node(data);
            if (cur->rbit == 0)
            {                p->right = cur->right;
                p->rbit = cur->rbit;
                p->lbit = 0;
                p->left = cur;
                cur->rbit = 1;
                cur->right = p;
                cout << "\nInserted right " << data;
                return;
            }
            else
                cur = cur->right;}        if (cur->data > data)
        { // insert left
            node *p = new node(data);
            if (cur->lbit == 0)
            {                p->left = cur->left;
                p->lbit = cur->lbit;
                p->rbit = 0;
                p->right = cur; // successor
                cur->lbit = 1;
                cur->left = p;
                cout << "\nInserted left " << data;
                return;
            }
            else    cur = cur->left;
        }    }}

int main()
{TBT t1;
    int value;
    int choice;
    do
    {        cout << "\n1. Create Tree\n2. Insert into tree\n3. Preorder\n4. Inorder\n0.
Exit\nEnter your choice: ";
        cin >> choice;
        switch (choice)
        {        case 1:
            t1.create();
            break;
        case 2:
            cout << "\nEnter Number(data): ";
            cin >> value;
            t1.insert(value);
            break;
        case 3:
            cout << "\nPreorder traversal of TBT\n";
            t1.preorder_traversal();
            break;
        case 4:
            cout << "\nInorder Traversal of TBT\n";
            t1.inorder_traversal();
            break;
        case 0:
            cout << "\nExiting program.\n";
            break;
        default:
            cout << "\nWrong choice";
        }
    } while (choice != 0);
    return 0;}
```

Output:



```
1. Create Tree
2. Insert into tree
3. Preorder
4. Inorder
0. Exit
Enter your choice: 1

Enter number of nodes:4

Enter data: 56

Inserted start56
Enter data: 78

Inserted right 78
Enter data: 43

Inserted left 43
Enter data: 3

Inserted left 3
```

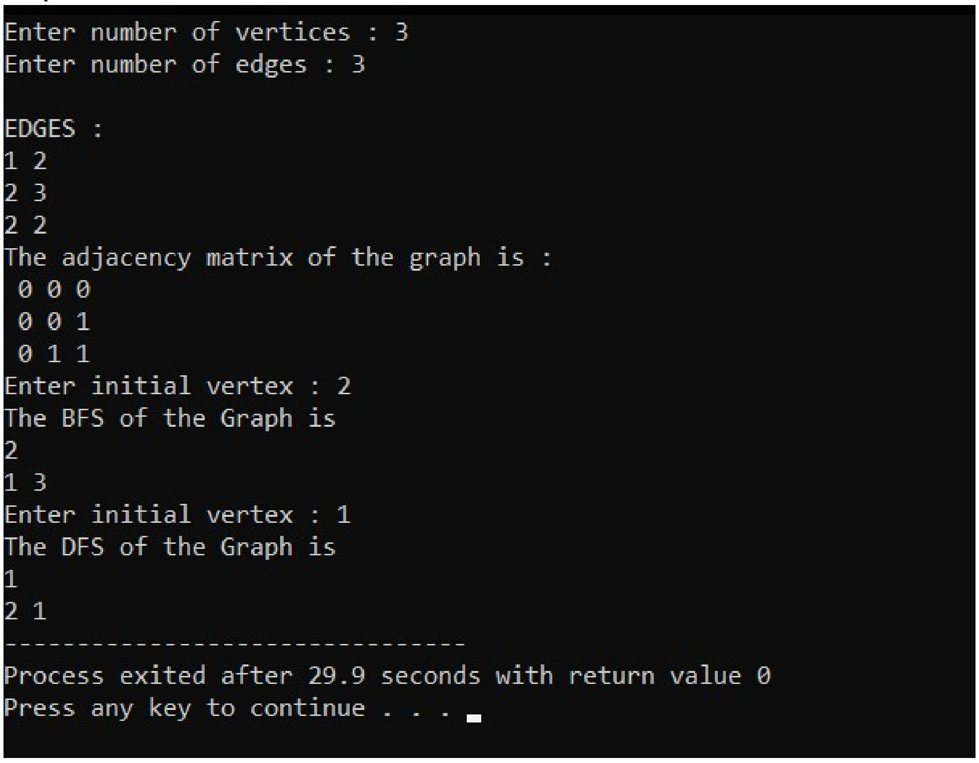


```
1. Create Tree
2. Insert into tree
3. Preorder
4. Inorder
0. Exit
Enter your choice: 3

Preorder traversal of TBT
 56 43 3 78
1. Create Tree
2. Insert into tree
3. Preorder
4. Inorder
0. Exit
Enter your choice:
```

Experiment : 6

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int cost[10][10], i, j, k, n, qu[10], front, rear, v, visit[10], visited[10];
int stk[10], top, visit1[10], visited1[10];
int main()
{
int m;
cout << "Enter number of vertices : ";
cin >> n;
cout << "Enter number of edges : ";
cin >> m;
cout << "\nEDGES :\n";
for (k = 1; k <= m; k++)
{
cin >> i >> j;
cost[i][j] = 1;
cost[j][i] = 1;
}
//display function
cout << "The adjacency matrix of the graph is : " << endl;
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
cout << " " << cost[i][j];
}
cout << endl;
}
cout << "Enter initial vertex : ";
cin >> v;
cout << "The BFS of the Graph is\n";
cout << v<<endl; visited[v] = 1;
k = 1;
while (k < n)
{
for (j = 1; j <= n; j++)
if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1)
{
visit[j] = 1;
qu[rear++] = j;
}
v = qu[front++];
cout << v << " ";
k++;
visit[v] = 0;
visited[v] = 1;
}
cout <<endl<<"Enter initial vertex : ";
cin >> v;
cout << "The DFS of the Graph is\n";
cout << v<<endl;
visited[v] = 1;
k = 1;
while (k < n)
{
for (j = n; j >= 1; j--)
if (cost[v][j] != 0 && visited1[j] != 1 && visit1[j] != 1)
{
visit1[j] = 1;
stk[top] = j;
top++;
}
v = stk[--top];
cout << v << " ";
k++;
visit1[v] = 0;
visited1[v] = 1;
}
return 0;
}
```

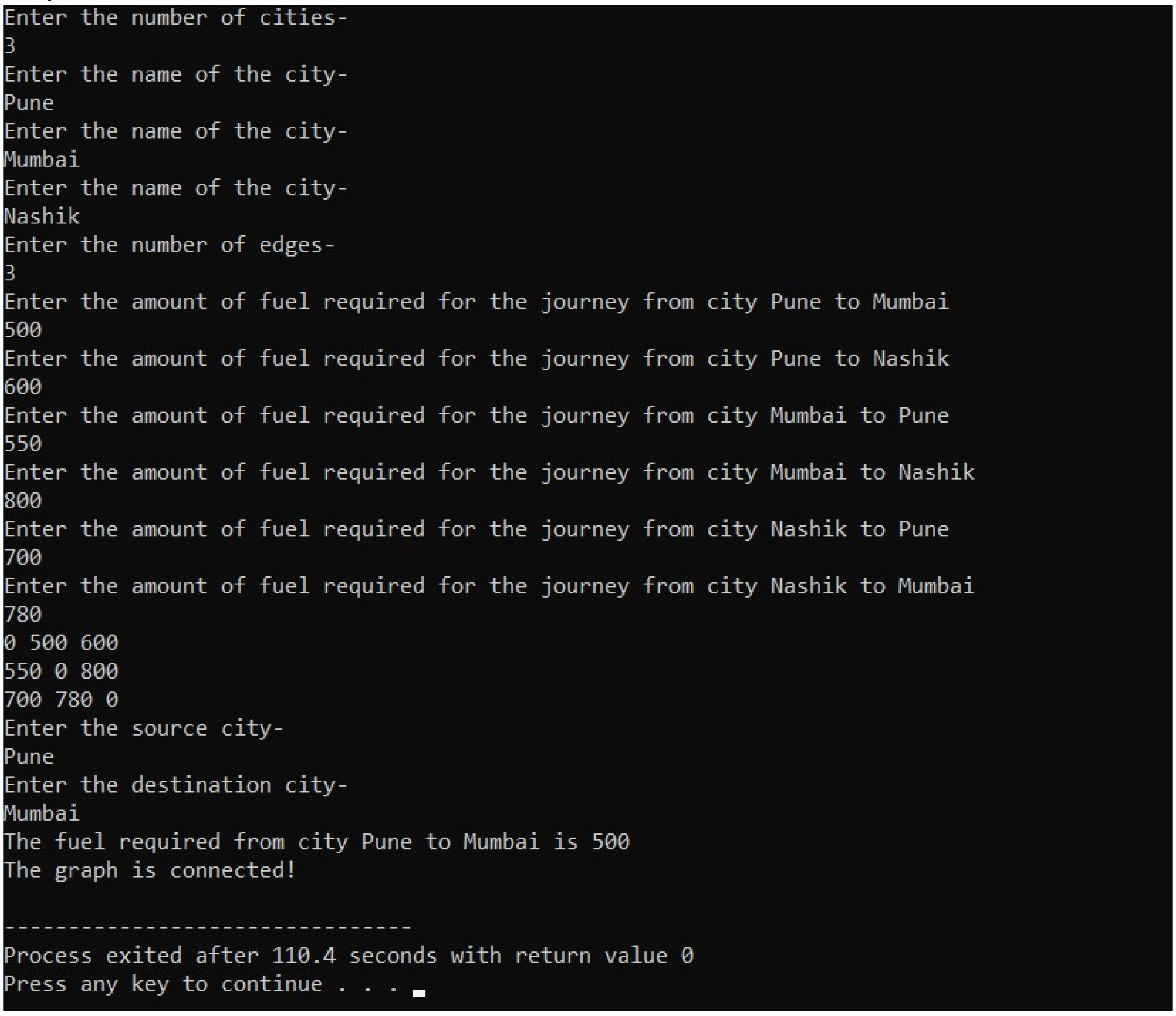
Output:



```
Enter number of vertices : 3
Enter number of edges : 3

EDGES :
1 2
2 3
2 2
The adjacency matrix of the graph is :
 0 0 0
 0 0 1
 0 1 1
Enter initial vertex : 2
The BFS of the Graph is
2
1 3
Enter initial vertex : 1
The DFS of the Graph is
1
2 1
--------------------------------
Process exited after 29.9 seconds with return value 0
Press any key to continue . . .
```

Experiment : 7

```cpp
#include<bits/stdc++.h>
using namespace std;
#define MAX 100

struct Graph
{
    int data[MAX][MAX];
    int vertices;

    map<pair<string,string>, int> hash;

    void read()
    {
        for (int i = 0; i < MAX; i++)
        {
            for (int j = 0; j < MAX; j++)
            {
                data[i][j] = 0;
            }
        }
        cout << "Enter the number of cities-\n";
        cin >> vertices;
        string cities[vertices];
        for (int i = 0; i < vertices; i++)
        {
            string city;
            cout << "Enter the name of the city-\n";
            cin >> cities[i];
        }
        int edges;
        cout << "Enter the number of edges-\n";
        cin >> edges;

        for (int i = 0; i < vertices; i++)
        {
            int fuel;
            for (int j = 0; j < vertices; j++)
            {
                if (i != j)
                {
                    cout << "Enter the amount of fuel required for the journey from city " 
+ cities[i] + " to " << cities[j] << endl;
                    cin >> fuel;
                    data[i][j] = fuel;
                    pair<string,string> p = make_pair(cities[i], cities[j]);
                    hash[p] = fuel;
                }
                else
                {
                    pair<string,string> p = make_pair(cities[i], cities[j]);
                    hash[p] = 0;
                }
            }
        }
    }

    void display()
    {
        for (int i = 0; i < vertices; i++)
        {
            for (int j = 0; j < vertices; j++)
            {
                cout << data[i][j] << " ";
            }
            cout << endl;
        }
    }

    void check_path(pair<string,string> p)
    {
        if (hash[p] != 0)
        {
            cout << "The fuel required from city " << p.first << " to " << p.second << " is
" << hash[p] << endl;
        }
        else
        {
            cout << "There is no path from city " << p.first << " to " << p.second << endl;
        }
    }

    void is_connected(int source, vector<bool> &visited)
    {
        visited[source] = true;
        for (int i = 0; i < vertices; i++)
        {
            if (data[source][i] != 0 and visited[i] != true)
            {
                is_connected(i, visited);
            }
        }
    }
};

int main()
{
    Graph g;
    g.read();
    g.display();
    string source, destination;
    cout << "Enter the source city-\n";
    cin >> source;
    cout << "Enter the destination city-\n";
    cin >> destination;
    pair<string,string> p = make_pair(source, destination);
    g.check_path(p);
    vector<bool> visited(MAX, false);
    g.is_connected(0, visited);
    bool flag = true;
    for (int i = 0; i < g.vertices; i++)
    {
        if (visited[i] == false)
        {
            flag = false;
            break;
        }
    }
    if (flag == true)
    {
        cout << "The graph is connected!" << endl;
    }
    else
    {
        cout << "The graph is not connected!" << endl;
    }
    return 0;
}
```
Output:



```
Enter the number of cities-
3
Enter the name of the city-
Pune
Enter the name of the city-
Mumbai
Enter the name of the city-
Nashik
Enter the number of edges-
3
Enter the amount of fuel required for the journey from city Pune to Mumbai
500
Enter the amount of fuel required for the journey from city Pune to Nashik
600
Enter the amount of fuel required for the journey from city Mumbai to Pune
550
Enter the amount of fuel required for the journey from city Mumbai to Nashik
800
Enter the amount of fuel required for the journey from city Nashik to Pune
700
Enter the amount of fuel required for the journey from city Nashik to Mumbai
780
0 500 600
550 0 800
700 780 0
Enter the source city-
Pune
Enter the destination city-
Mumbai
The fuel required from city Pune to Mumbai is 500
The graph is connected!

--------------------------------
Process exited after 110.4 seconds with return value 0
Press any key to continue . . .
```

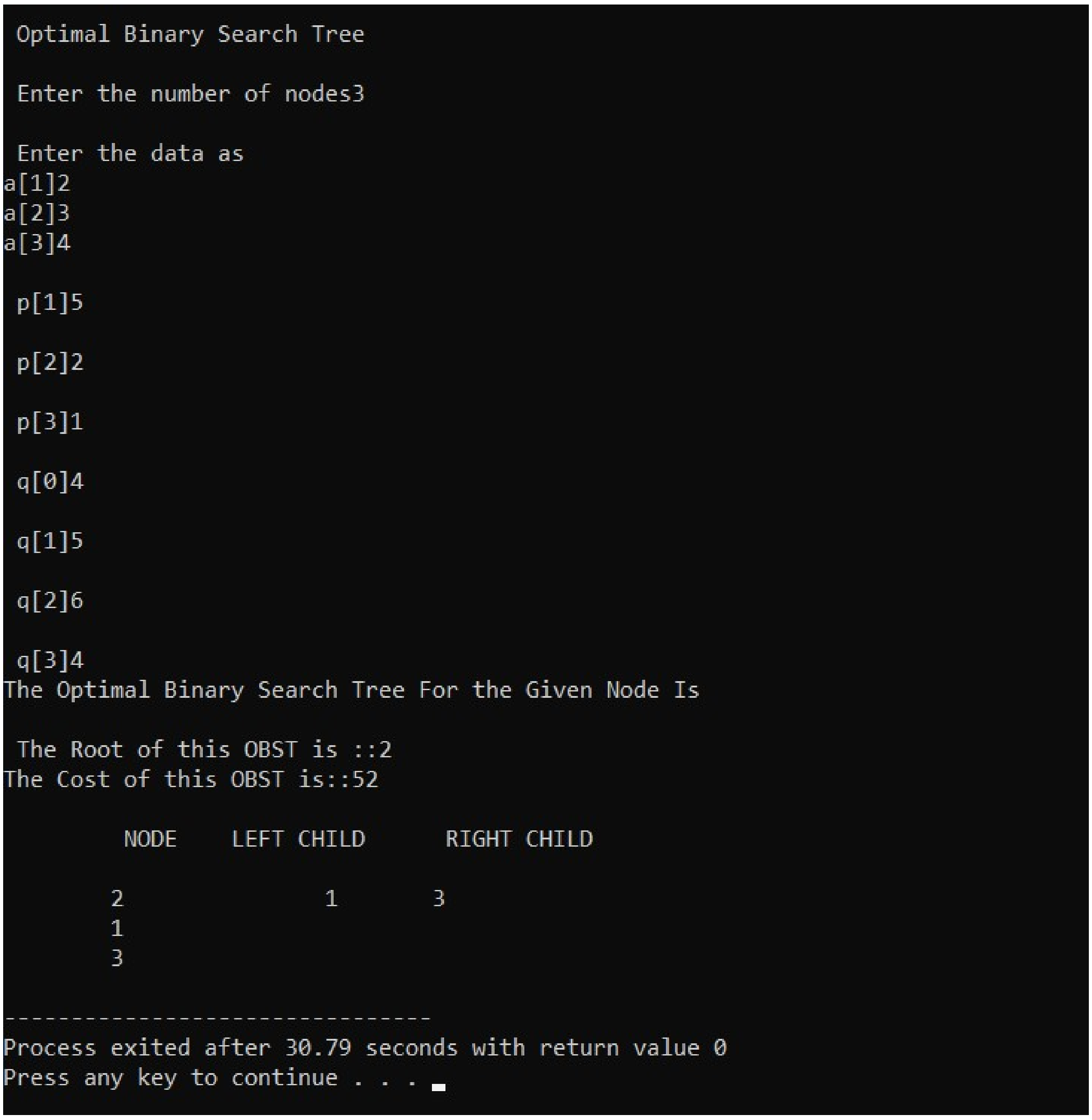
Experiment : 8

```
#include<iostream>
using namespace std;
#define SIZE 10
class OBST
{
int p[SIZE]; // Probabilities with which we search for an element
int q[SIZE];//Probabilities that an element is not found
int a[SIZE];//Elements from which OBST is to be built
int w[SIZE][SIZE];//Weight 'w[i][j]' of a tree having root
int c[SIZE][SIZE];//Cost 'c[i][j] of a tree having root 'r[i][j]
int r[SIZE][SIZE];//represents root
int n; // number of nodes
public:
/* This function accepts the input data */
void get_data()
{
int i;
cout<<"\n Optimal Binary Search Tree \n";
cout<<"\n Enter the number of nodes";
cin>>n;
cout<<"\n Enter the data as\n";
for(i=1;i<=n;i++)
{
cout<<"a["<<i<<"]";
cin>>a[i];
}
for(i=1;i<=n;i++)
{
cout<<"\n p["<<i<<"]";
cin>>p[i];
}
for(i=0;i<=n;i++)
{
cout<<"\n q["<<i<<"]";
cin>>q[i];
}
}
/* This function returns a value in the range 'r[i][j-1]' to 'r[i+1][j]'so
that the cost 'c[i][k-1]+c[k][j]'is minimum */
int Min_Value(int i,int j)
{
int m,k;
int minimum=32000;
for(m=r[i][j-1];m<=r[i+1][j];m++)
{
if((c[i][m-1]+c[m][j])<minimum)
{
minimum=c[i][m-1]+c[m][j];
k=m;
}
}
return k;
}
/* This function builds the table from all the given probabilities It
basically computes C,r,W values */
void build_OBST()
{
int i,j,k,l,m;
```

```
for(i=0;i<n;i++)
{
//initialize
w[i][i]=q[i];
r[i][i]=c[i][i]=0;
//Optimal trees with one node
w[i][i+1]=q[i]+q[i+1]+p[i+1];
r[i][i+1]=i+1;
c[i][i+1]=q[i]+q[i+1]+p[i+1];
}
w[n][n]=q[n];
r[n][n]=c[n][n]=0;
//Find optimal trees with 'm' nodes
for(m=2;m<=n;m++)
{
for(i=0;i<=n-m;i++)
{
j=i+m;
w[i][j]=w[i][j-1]+p[j]+q[j];
k=Min_Value(i,j);
c[i][j]=w[i][j]+c[i][k-1]+c[k][j];
r[i][j]=k;
}
}
}
/* This function builds the tree from the tables made by the OBST function */
void build_tree()
{
int i,j,k;
int queue[20],front=-1,rear=-1;
cout<<"The Optimal Binary Search Tree For the Given Node Is\n";
cout<<"\n The Root of this OBST is ::"<<r[0][n];
cout<<"\nThe Cost of this OBST is::"<<c[0][n];
cout<<"\n\n\t NODE \t LEFT CHILD \t RIGHT CHILD ";
cout<<"\n";
queue[++rear]=0;
queue[++rear]=n;
while(front!=rear)
{
i=queue[++front];
j=queue[++front];
k=r[i][j];
cout<<"\n\t"<<k;
if(r[i][k-1]!=0)
{
cout<<"\t\t"<<r[i][k-1];
queue[++rear]=i;
queue[++rear]=k-1;
}
else
cout<<"\t\t";
if(r[k][j]!=0)
{
cout<<"\t"<<r[k][j];
queue[++rear]=k;
queue[++rear]=j;
}
else
cout<<"\t";
}//end of while
cout<<"\n";
```

```
}
};//end of the class
/*This is the main function */
int main()
{
OBST obj;
obj.get_data();
obj.build_OBST();
obj.build_tree();
return 0;
}
```

Output:



```
 Optimal Binary Search Tree

 Enter the number of nodes3

 Enter the data as
a[1]2
a[2]3
a[3]4

 p[1]5

 p[2]2

 p[3]1

 q[0]4

 q[1]5

 q[2]6

 q[3]4
The Optimal Binary Search Tree For the Given Node Is

 The Root of this OBST is ::2
The Cost of this OBST is::52

         NODE    LEFT CHILD      RIGHT CHILD

        2                1       3
        1
        3

--------------------------------
Process exited after 30.79 seconds with return value 0
Press any key to continue . . .
```

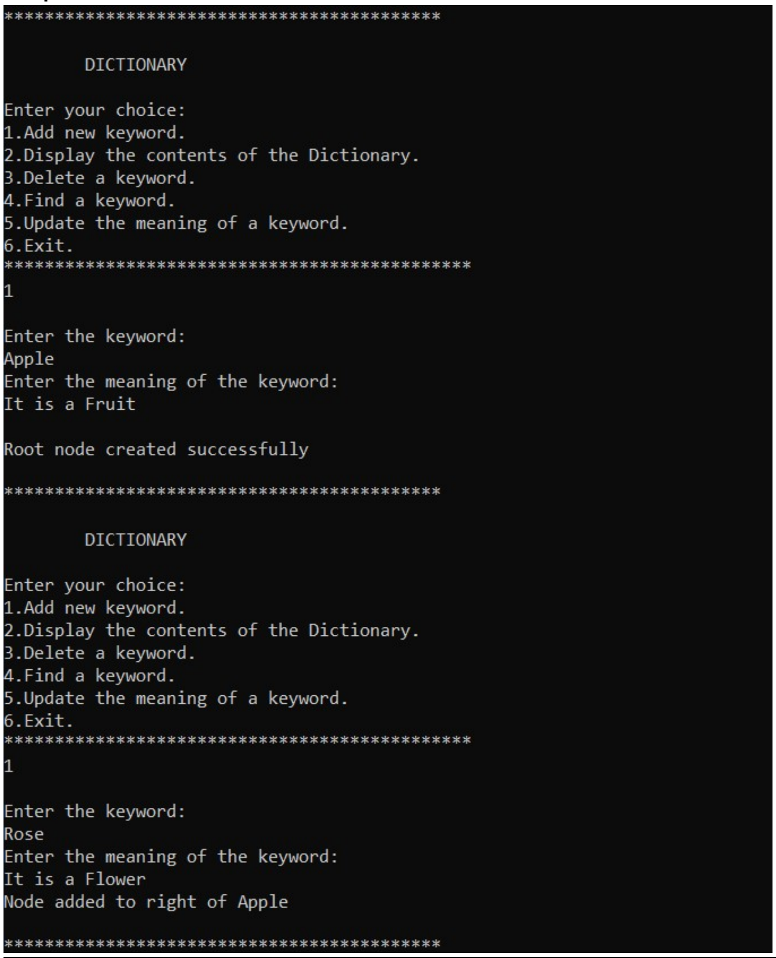
Experiment : 9

```
#include <iostream>
#include <string.h>
using namespace std;
class dict
{
    dict *root, *node, *left, *right, *tree1;
    string s1, s2;
    int flag, flag1, flag2, flag3, cmp;

public:
    dict()
    {
        flag = 0, flag1 = 0, flag2 = 0, flag3 = 0, cmp = 0;
        root = NULL;
    }
    void input();
    void create_root(dict *, dict *);
    void check_same(dict *, dict *);
    void input_display();
    void display(dict *);
    void input_remove();
    dict *remove(dict *, string);
    dict *findmin(dict *);
    void input_find();
    dict *find(dict *, string);
    void input_update();
    dict *update(dict *, string);
};

void dict::input()
{
    node = new dict;
    cout << "\nEnter the keyword:\n";
    cin >> node->s1;
    cout << "Enter the meaning of the keyword:\n";
    cin.ignore();
    getline(cin, node->s2);
    create_root(root, node);
}

void dict::create_root(dict *tree, dict *node1)
{
    int i = 0, result;
    char a[20], b[20];
    if (root == NULL)
    {
        root = new dict;
        root = node1;
        root->left = NULL;
        root->right = NULL;
        cout << "\nRoot node created successfully" << endl;
        return;
    }
    for (i = 0; node1->s1[i] != '\0'; i++)
    {
        a[i] = node1->s1[i];
    }
    for (i = 0; tree->s1[i] != '\0'; i++)
    {
        b[i] = tree->s1[i];
    }
    result = strcmp(b, a);
    check_same(tree, node1);
    if (flag == 1)
    {
        cout << "The word you entered already exists.\n";
        flag = 0;
    }
    else
    {
        if (result > 0)
        {
            if (tree->left != NULL)
            {
                create_root(tree->left, node1);
            }
            else
            {
                tree->left = node1;
                (tree->left)->left = NULL;
                (tree->left)->right = NULL;
                cout << "Node added to left of " << tree->s1 << "\n";
                return;
            }
        }
        else if (result < 0)
        {
            if (tree->right != NULL)
            {
                create_root(tree->right, node1);
            }
            else
            {
                tree->right = node1;
                (tree->right)->left = NULL;
                (tree->right)->right = NULL;
                cout << "Node added to right of " << tree->s1 << "\n";
                return;
            }
        }
    }
}

void dict::check_same(dict *tree, dict *node1)
{
    if (tree->s1 == node1->s1)
    {
        flag = 1;
        return;
    }
    else if (tree->s1 > node1->s1)
    {
        if (tree->left != NULL)
        {
            check_same(tree->left, node1);
        }
    }
    else if (tree->s1 < node1->s1)
    {
        if (tree->right != NULL)
        {
            check_same(tree->right, node1);
        }
    }
}

void dict::input_display()
{
    if (root != NULL)
    {
        cout << "The words entered in the dictionary are:\n\n";
        display(root);
    }
    else
    {
        cout << "\nThere are no words in the dictionary.\n";
    }
}

void dict::display(dict *tree)
{
    if (tree->left == NULL && tree->right == NULL)
    {
        cout << tree->s1 << " = " << tree->s2 << "\n\n";
    }
    else
    {
        if (tree->left != NULL)
        {
            display(tree->left);
        }
        cout << tree->s1 << " = " << tree->s2 << "\n\n";
        if (tree->right != NULL)
        {
            display(tree->right);
        }
    }
}

void dict::input_remove()
{
    char t;
    if (root != NULL)
    {
        cout << "\nEnter a keyword to be deleted:\n";
        cin >> s1;
        remove(root, s1);
        if (flag1 == 0)
        {
            cout << "\nThe word '" << s1 << "' has been deleted.\n";
        }
        flag1 = 0;
    }
    else
    {
        cout << "\nThere are no words in the dictionary.\n";
    }
}
```

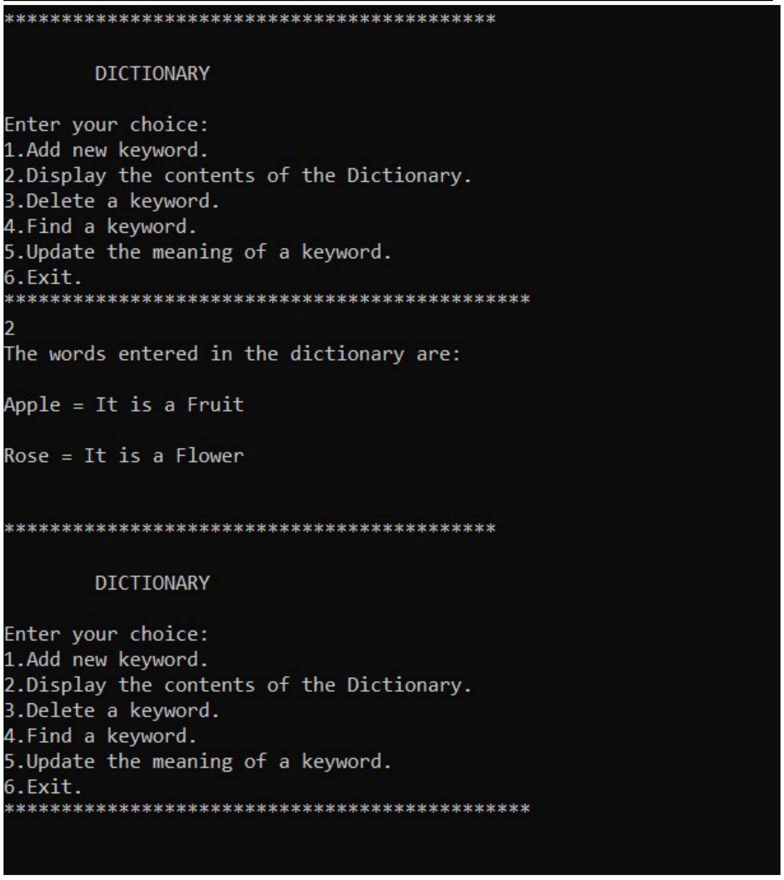
```
dict *dict::remove(dict *tree, string s3)
{
    dict *temp;
    if (tree == NULL)
    {
        cout << "\nWord not found.\n";
        flag1 = 1;
        return tree;
    }
    else if (tree->s1 > s3)
    {
        tree->left = remove(tree->left, s3);
        return tree;
    }
    else if (tree->s1 < s3)
    {
        tree->right = remove(tree->right, s3);
        return tree;
    }
    else
    {
        if (tree->left == NULL && tree->right == NULL)
        {
            delete tree;
            tree = NULL;
        }
        else if (tree->left == NULL)
        {
            temp = tree;
            tree = tree->right;
            delete temp;
        }
        else if (tree->right == NULL)
        {
            temp = tree;
            tree = tree->left;
            delete temp;
        }
        else
        {
            temp = findmin(tree->right);
            tree = temp;
            tree->right = remove(tree->right, temp->s1);
        }
    }
    return tree;
}

dict *dict::findmin(dict *tree)
{
    while (tree->left != NULL)
    {
        tree = tree->left;
    }
    return tree;
}

void dict::input_find()
{
    flag2 = 0, cmp = 0;
    if (root != NULL)
    {
        cout << "\nEnter the keyword to be searched:\n";
        cin >> s1;
        find(root, s1);
        if (flag2 == 0)
        {
            cout << "Number of comparisons needed: " << cmp << "\n";
            cmp = 0;
        }
    }
    else
    {
        cout << "\nThere are no words in the dictionary.\n";
    }
}

dict *dict::find(dict *tree, string s3)
{
    if (tree == NULL)
    {
        cout << "\nWord not found.\n";
        flag2 = 1;
        flag3 = 1;
        cmp = 0;
    }
    else
    {
        if (tree->s1 == s3)
        {
            cmp++;
            cout << "\nWord found.\n";
            cout << tree->s1 << ": " << tree->s2 << "\n";
            tree1 = tree;
            return tree;
        }
        else if (tree->s1 > s3)
        {
            cmp++;
            find(tree->left, s3);
        }
        else if (tree->s1 < s3)
        {
            cmp++;
            find(tree->right, s3);
        }
    }
    return tree;
}

void dict::input_update()
{
    if (root != NULL)
    {
        cout << "\nEnter the keyword to be updated:\n";
        cin >> s1;
        update(root, s1);
    }
    else
    {
        cout << "\nThere are no words in the dictionary.\n";
    }
```

```
}

dict *dict::update(dict *tree, string s3)
{
    flag3 = 0;
    find(tree, s3);
    if (flag3 == 0)
    {
        cout << "\nEnter the updated meaning of the keyword:\n";
        cin.ignore();
        getline(cin, tree1->s2);
        cout << "\nThe meaning of '" << s3 << "' has been updated.\n";
    }
    return tree;
}

int main()
{
    cout<<"\n"<<endl;
    int ch;
    dict d;
    do
    {
        cout << "\n*******************************************\n"
                "\n\tDICTIONARY\n"
                "\nEnter your choice:\n"
                "1.Add new keyword.\n"
                "2.Display the contents of the Dictionary.\n"
                "3.Delete a keyword.\n"
                "4.Find a keyword.\n"
                "5.Update the meaning of a keyword.\n"
                "6.Exit.\n"
                "**********************************************\n";
        cin >> ch;
        switch (ch)
        {
        case 1:
            d.input();
            break;
        case 2:
            d.input_display();
            break;
        case 3:
            d.input_remove();
            break;
        case 4:
            d.input_find();
            break;
        case 5:
            d.input_update();
            break;
        default:
            cout << "\nPlease enter a valid option!\n";
            break;
        }
    } while (ch != 6);
    return 0;
}
```

Output:



```
*******************************************


       DICTIONARY

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
**********************************************
1

Enter the keyword:
Apple
Enter the meaning of the keyword:
It is a Fruit

Root node created successfully

*******************************************


       DICTIONARY

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
**********************************************
1

Enter the keyword:
Rose
Enter the meaning of the keyword:
It is a Flower
Node added to right of Apple

*******************************************
```

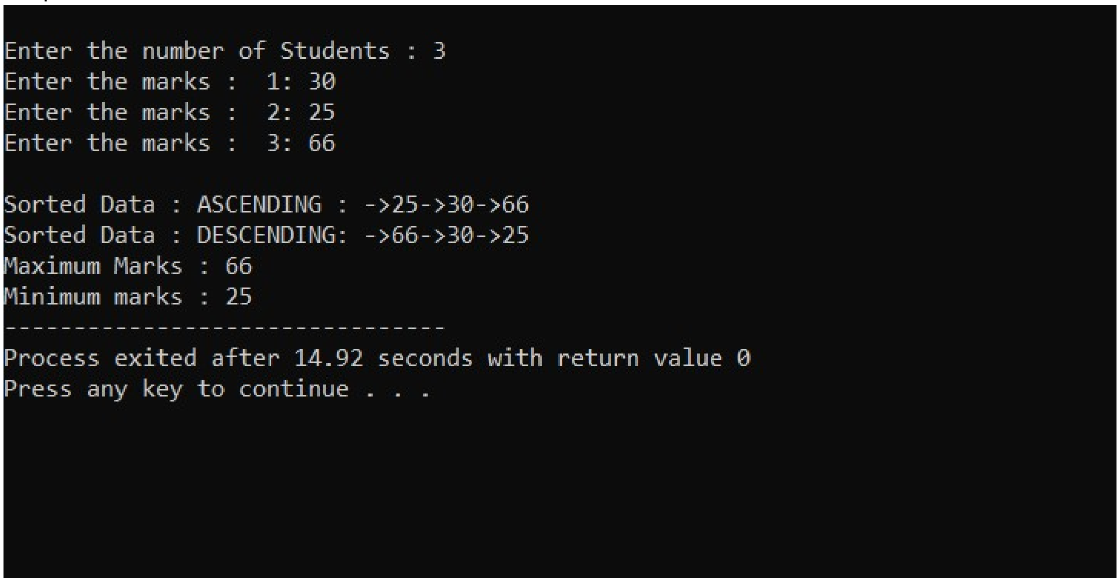


```
*******************************************


       DICTIONARY

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
**********************************************
2
The words entered in the dictionary are:

Apple = It is a Fruit

Rose = It is a Flower


*******************************************


       DICTIONARY

Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
**********************************************
```

Experiment : 10

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

void MaxHeapify(int a[], int i, int n)
{
    int j, temp;
    temp = a[i];
    j = 2 * i;

    while (j <= n)
    {
        if (j < n && a[j + 1] > a[j])
            j = j + 1;

        if (temp > a[j])
            break;

        else if (temp <= a[j])
        {
            a[j / 2] = a[j];
            j = 2 * j;
        }
    }
    a[j / 2] = temp;
    return;
}

void MinHeapify(int a[], int i, int n)
{
    int j, temp;
    temp = a[i];
    j = 2 * i;

    while (j <= n)
    {
        if (j < n && a[j + 1] < a[j])
            j = j + 1;

        if (temp < a[j])
            break;

        else if (temp >= a[j])
        {
            a[j / 2] = a[j];
            j = 2 * j;
        }
    }
    a[j / 2] = temp;
    return;
}

void MaxHeapSort(int a[], int n)
{
    int i, temp;
    for (i = n; i >= 2; i--)
    {
        temp = a[i];
        a[i] = a[1];
        a[1] = temp;

        MaxHeapify(a, 1, i - 1);
    }
}

void MinHeapSort(int a[], int n)
{
    int i, temp;
    for (i = n; i >= 2; i--)
    {

        temp = a[i];
        a[i] = a[1];
        a[1] = temp;

        MinHeapify(a, 1, i - 1);
    }
}

void Build_MaxHeap(int a[], int n)
{
    int i;
    for (i = n / 2; i >= 1; i--)
        MaxHeapify(a, i, n);
}

void Build_MinHeap(int a[], int n)
{
    int i;
    for (i = n / 2; i >= 1; i--)
        MinHeapify(a, i, n);
}

int main()
{
    cout<<"\n"<<endl;
    int n, i;
    cout << "\nEnter the number of Students : ";
    cin >> n;
    n++;
    int arr[n];
    for (i = 1; i < n; i++)
    {
        cout << "Enter the marks :  " << i << ": ";
        cin >> arr[i];
    }

    Build_MaxHeap(arr, n - 1);
    MaxHeapSort(arr, n - 1);

    int max, min;
    cout << "\nSorted Data : ASCENDING : ";

    for (i = 1; i < n; i++)
        cout << "->" << arr[i];
    min = arr[1];
    Build_MinHeap(arr, n - 1);
    MinHeapSort(arr, n - 1);
    cout << "\nSorted Data : DESCENDING: ";
    max = arr[1];
    for (i = 1; i < n; i++)
        cout << "->" << arr[i];
    cout << "\nMaximum Marks : " << max << "\nMinimum marks : " << min;
    return 0;
}
```

Output:



```
Enter the number of Students : 3
Enter the marks :  1: 30
Enter the marks :  2: 25
Enter the marks :  3: 66

Sorted Data : ASCENDING : ->25->30->66
Sorted Data : DESCENDING: ->66->30->25
Maximum Marks : 66
Minimum marks : 25
--------------------------------
Process exited after 14.92 seconds with return value 0
Press any key to continue . . .
```

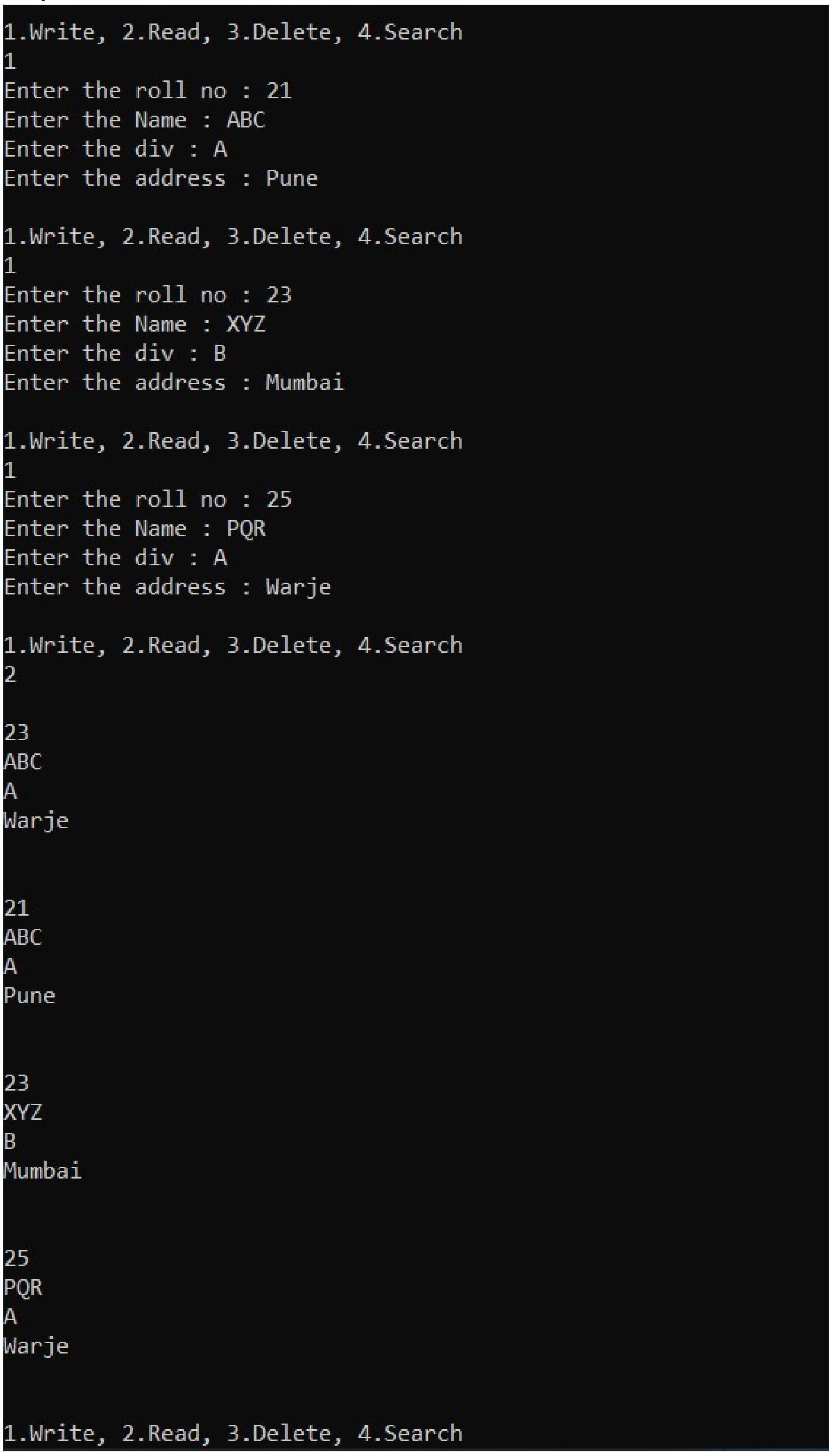
Experiment : 11

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

void writef()
{
    int roll;
    string name, div, address;
    fstream write;

    cout << "Enter the roll no : ";
    cin >> roll;
    cout << "Enter the Name : ";
    cin >> name;
    cout << "Enter the div : ";
    cin >> div;
    cout << "Enter the address : ";
    cin >> address;

    // writing to file
    write.open("students.txt", ios::app);
    write << roll << "\n";
    write << name << "\n";
    write << div << "\n";
    write << address << "\n";
    write << "\n";
    write.close();
}

void readf()
{
    int roll;
    string name, div, address;
    fstream read;
    read.open("students.txt", ios::in);
    while (read >> roll >> name >> div >> address)
    {
        cout << endl;
        cout << roll << endl;
        cout << name << endl;
        cout << div << endl;
        cout << address << endl;
        cout << endl;
    }

    read.close();
}

void searchf()
{
    int roll, froll;
    bool flag = false;
    string name, div, address;
    fstream read;
    cout << "Enter Roll no to search" << endl;
    cin >> froll;
    read.open("students.txt", ios::in);
```

```cpp
    while (read >> roll >> name >> div >> address)
    {
        if (roll == froll)
        {
            flag = true;
            cout << "Record Found" << endl;
        }
    }
    if (!flag)
        cout << "Record Not Found" << endl;

    read.close();
}

void deletef()
{
    int roll, froll;
    bool flag = false;
    string name, div, address;
    fstream read, write;
    cout << "Enter Roll no to Delete" << endl;
    cin >> froll;
    read.open("students.txt", ios::in);
    while (read >> roll >> name >> div >> address)
    {
        if (roll == froll)
        {
            flag = true;
        }
        else
        {
            write.open("stud_updated.txt", ios::app);
            write << roll << "\n";
            write << name << "\n";
            write << div << "\n";
            write << address << "\n";
            write << "\n";
            write.close();
        }
    }
    read.close();
    remove("students.txt");
    rename("stud_updated.txt", "students.txt");
}

int main()
{
    int ch;
    while (1)
    {
        cout << endl;
        cout << "1.Write, 2.Read, 3.Delete, 4.Search" << endl;
        cin >> ch;

        if (ch == 1)
            writef();
        else if (ch == 2)
            readf();
        else if (ch == 3)
            deletef();
        else if (ch == 4)
```

```
            searchf();
        else
            exit(0);
    }

    return 0;
}
```

Output:

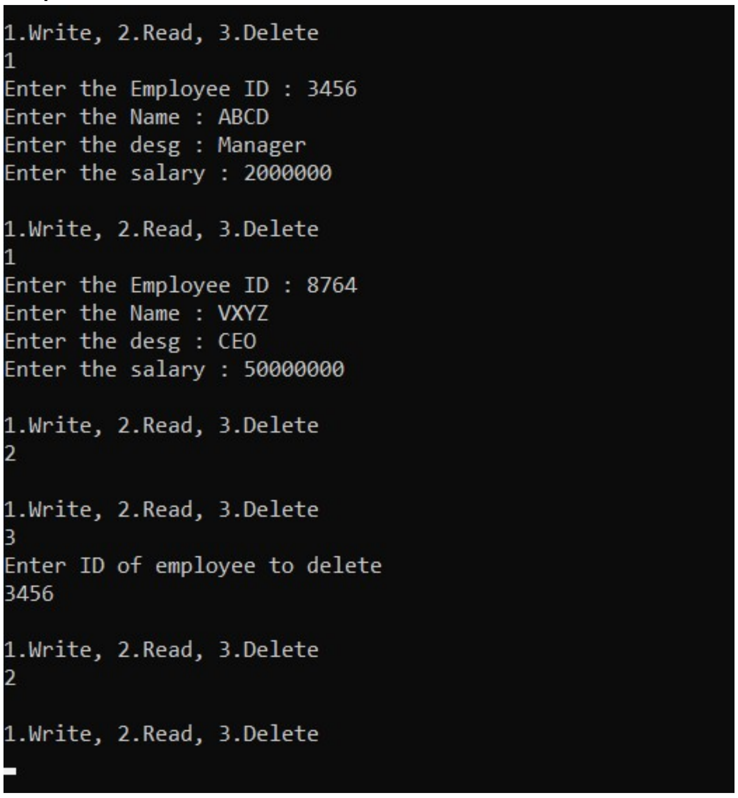


```
1.Write, 2.Read, 3.Delete, 4.Search
1
Enter the roll no : 21
Enter the Name : ABC
Enter the div : A
Enter the address : Pune

1.Write, 2.Read, 3.Delete, 4.Search
1
Enter the roll no : 23
Enter the Name : XYZ
Enter the div : B
Enter the address : Mumbai

1.Write, 2.Read, 3.Delete, 4.Search
1
Enter the roll no : 25
Enter the Name : PQR
Enter the div : A
Enter the address : Warje

1.Write, 2.Read, 3.Delete, 4.Search
2

23
ABC
A
Warje


21
ABC
A
Pune


23
XYZ
B
Mumbai


25
PQR
A
Warje


1.Write, 2.Read, 3.Delete, 4.Search
```

Experiment : 12

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

void writef()
{
    int id;
    string name, desg, sal;
    fstream write;

    cout << "Enter the Employee ID : ";
    cin >> id;
    cout << "Enter the Name : ";
    cin >> name;
    cout << "Enter the desg : ";
    cin >> desg;
    cout << "Enter the salary : ";
    cin >> sal;

    if (id > 0 && id <= 10)
    {
        write.open("1to10.txt", ios::app);
        write << id << "\n";
        write << name << "\n";
        write << desg << "\n";
        write << sal << "\n";
        write << "\n";
        write.close();
    }
    else if (id > 11 && id <= 20)
    {
        write.open("11to20.txt", ios::app);
        write << id << "\n";
        write << name << "\n";
        write << desg << "\n";
        write << sal << "\n";
        write << "\n";
        write.close();
    }
    else if (id > 21 && id <= 30)
    {
        write.open("21to30.txt", ios::app);
        write << id << "\n";
        write << name << "\n";
        write << desg << "\n";
        write << sal << "\n";
        write << "\n";
        write.close();
    }
}

void readf()
{
    int id;
    string name, desg, sal;
    fstream read;
    read.open("1to10.txt", ios::in);
    {
        while (read >> id >> name >> desg >> sal)
        {
            cout << id << endl;
            cout << name << endl;
            cout << desg << endl;
            cout << sal << endl;
            cout << endl;
        }
    }
    read.close();
    read.open("11to20.txt", ios::in);
    {
        while (read >> id >> name >> desg >> sal)
        {
            cout << id << endl;
            cout << name << endl;
            cout << desg << endl;
            cout << sal << endl;
            cout << endl;
        }
    }
    read.close();
    read.open("21to30.txt", ios::in);
    {
        while (read >> id >> name >> desg >> sal)
        {
            cout << id << endl;
            cout << name << endl;
            cout << desg << endl;
            cout << sal << endl;
            cout << endl;
        }
    }
    read.close();
}

void deletef()
{
    fstream read, write;
    int id, did;
    string name, desg, sal;
    cout << "Enter ID of employee to delete" << endl;
    cin >> did;

    if (did > 0 && did <= 10)
    {
        read.open("1to10.txt", ios::in);
        while (read >> id >> name >> desg >> sal)
        {
            if (id == did)
            {
                cout << "Deleting Record.." << endl;
            }
            else
            {
                write.open("1to10up.txt", ios::app);
                write << id << "\n";
                write << name << "\n";
                write << desg << "\n";
                write << sal << "\n";
                write << "\n";
                write.close();
            }
        }
        read.close();
        remove("1to10.txt");
        rename("1to10up.txt","1to10.txt");
    }
    else if (did > 10 && did <= 20)
    {
        read.open("11to20.txt", ios::in);
        while (read >> id >> name >> desg >> sal)
        {
            if (id == did)
            {
                cout << "Deleting Record.." << endl;
            }
            else
            {
                write.open("11to20up.txt", ios::app);
                write << id << "\n";
                write << name << "\n";
                write << desg << "\n";
                write << sal << "\n";
                write << "\n";
                write.close();
            }
        }
        read.close();
        remove("11to20.txt");
        rename("11to20up.txt","11to20.txt");
    }
    else if (did > 20 && did <= 30)
    {
        read.open("21to30.txt", ios::in);
        while (read >> id >> name >> desg >> sal)
        {
            if (id == did)
            {
                cout << "Deleting Record.." << endl;
            }
            else
            {
                write.open("21to30up.txt", ios::app);
                write << id << "\n";
                write << name << "\n";
                write << desg << "\n";
                write << sal << "\n";
                write << "\n";
                write.close();
            }
        }
        read.close();
        remove("21to30.txt");
        rename("21to30up.txt","21to30.txt");
    }
}

int main()
{
    int ch;
    while (1)
    {
        cout << endl;
        cout << "1.Write, 2.Read, 3.Delete" << endl;
        cin >> ch;

        if (ch == 1)
            writef();
        else if (ch == 2)
            readf();
        else if (ch == 3)
            deletef();
        else
            exit(0);
    }

    return 0;
}
```

Output:



```
1.Write, 2.Read, 3.Delete
1
Enter the Employee ID : 3456
Enter the Name : ABCD
Enter the desg : Manager
Enter the salary : 2000000

1.Write, 2.Read, 3.Delete
1
Enter the Employee ID : 8764
Enter the Name : VXYZ
Enter the desg : CEO
Enter the salary : 50000000

1.Write, 2.Read, 3.Delete
2

1.Write, 2.Read, 3.Delete
3
Enter ID of employee to delete
3456

1.Write, 2.Read, 3.Delete
2

1.Write, 2.Read, 3.Delete
```