Experiment No. 1

Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number. Make use of two collision handling techniques and compare them using the number of comparisons required to find a set of telephone numbers

Program:

```
def run():
    hashtable_linear = [None] * 10
    hashtable_quadratic = [None] * 10
    n = int(input("Enter number of Phone Numbers: "))
    for _ in range(n):
        key = int(input("Enter Mobile Number: "))
        position = key % 10
        if hashtable_linear[position] is None:
            hashtable_linear[position] = [key]
        else:
            hashtable_linear = linearProbe(key, hashtable_linear, position)
            print("Linear Probing for empty slot..")
        if hashtable_quadratic[position] is None:
            hashtable_quadratic[position] = [key]
        else:
            hashtable_quadratic = quadraticProbe(key, hashtable_quadratic, position)
            print("Quadratic Probing for empty slot..")
        display(hashtable_linear, "Linear Probing")
        display(hashtable_quadratic, "Quadratic Probing")
    return hashtable_linear, hashtable_quadratic

def linearProbe(key, ht, pos):
    i = 1
    while i < 10:
        probe_position = (pos + i) % 10
        if ht[probe_position] is None:
            ht[probe_position] = [key]
            return ht
        i += 1
    print("Hash table is full")
    return ht

def quadraticProbe(key, ht, pos):
    i = 1
    while i < 10:
        probe_position = (pos + i * i) % 10
        if ht[probe_position] is None:
            ht[probe_position] = [key]
            return ht
        i += 1
    print("Hash table is full")
    return ht

def display(ht, method):
    print(f"\n{method} Hash Table:")
    for pos in range(10):
        val = ht[pos]
        print(f"Position {pos}: {val}")

def search(key, ht, method):
    comparisons = 0
    pos = key % 10
    if ht[pos] is not None and ht[pos][0] == key:
        comparisons += 1
        return pos, comparisons
    i = 1
    while i < 10:
        comparisons += 1
        if method == "linear":
            probe_position = (pos + i) % 10
        else:
            probe_position = (pos + i * i) % 10
        if ht[probe_position] is not None and ht[probe_position][0] == key:
            return probe_position, comparisons
        i += 1
    return -1, comparisons

def main():
    ht_linear, ht_quadratic = None, None
    while True:
        print("\nMenu:")
        print("1. Insert Phone Numbers")
        print("2. Search Phone Number")
        print("3. Exit")
        choice = int(input("Enter your choice: "))

        if choice == 1:
            ht_linear, ht_quadratic = run()
        elif choice == 2:
            if ht_linear is None or ht_quadratic is None:
                print("Please insert phone numbers first.")
            else:
                key = int(input("\nEnter Phone Number to search: "))
                pos_linear, comparisons_linear = search(key, ht_linear, "linear")
                pos_quadratic, comparisons_quadratic = search(key, ht_quadratic,
"quadratic")
                if pos_linear != -1:
                    print(f"Linear Probing: Phone Number found at position
{pos_linear} with {comparisons_linear} comparisons.")
                else:
                    print("Linear Probing: Phone Number not found.")
                if pos_quadratic != -1:
                    print(f"Quadratic Probing: Phone Number found at position
{pos_quadratic} with {comparisons_quadratic} comparisons.")
                else:
                    print("Quadratic Probing: Phone Number not found.")
        elif choice == 3:
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

```
PS G:\My Drive\DSA\DSAL> python3 hash.py


Menu:
1. Insert Phone Numbers
2. Search Phone Number
3. Exit
Enter your choice: 1
Enter number of Phone Numbers: 6
Enter Mobile Number: 12345

Linear Probing Hash Table:
Position 0: None
Position 1: None
Position 2: None
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: None
Position 7: None
Position 8: None
Position 9: None

Quadratic Probing Hash Table:
Position 0: None
Position 1: None
Position 2: None
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: None
Position 7: None
Position 8: None
Position 9: None
Enter Mobile Number: 12346

Linear Probing Hash Table:
Position 0: None
Position 1: None
Position 2: None
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: [12346]
Position 7: None
Position 8: None
Position 9: None

Quadratic Probing Hash Table:
Position 0: None
Position 1: None
Position 2: None
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: [12346]
Position 7: None
Position 8: None
Position 9: None
Enter Mobile Number: 12352

Linear Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: [12346]
Position 7: None
Position 8: None
Position 9: None

Quadratic Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: [12346]
Position 7: None
Position 8: None
Position 9: None
Enter Mobile Number: 11225
Linear Probing for empty slot..
Quadratic Probing for empty slot..

Linear Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: [12346]
Position 7: [11225]
Position 8: None
Position 9: None

Quadratic Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: [12346]
Position 7: None
Position 8: None
Position 9: [11225]
Enter Mobile Number: 35345
Linear Probing for empty slot..
Quadratic Probing for empty slot..

Linear Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: None
Position 5: [12345]
Position 6: [12346]
Position 7: [11225]
Position 8: [35345]
Position 9: None

Quadratic Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: [35345]
Position 5: [12345]
Position 6: [12346]
Position 7: None
Position 8: None
Position 9: [11225]
Enter Mobile Number: 858374
Quadratic Probing for empty slot..

Linear Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: [858374]
Position 5: [12345]
Position 6: [12346]
Position 7: [11225]
Position 8: [35345]
Position 9: None

Quadratic Probing Hash Table:
Position 0: None
Position 1: None
Position 2: [12352]
Position 3: None
Position 4: [35345]
Position 5: [12345]
Position 6: [12346]
Position 7: None
Position 8: [858374]
Position 9: [11225]

Menu:
1. Insert Phone Numbers
2. Search Phone Number
3. Exit
Enter your choice: 2

Enter Phone Number to search: 35345
Linear Probing: Phone Number found at position 8 with 3 comparisons.
Quadratic Probing: Phone Number found at position 4 with 3 comparisons.

Menu:
1. Insert Phone Numbers
2. Search Phone Number
3. Exit
Enter your choice: 2

Enter Phone Number to search: 858374
Linear Probing: Phone Number found at position 4 with 1 comparisons.
Quadratic Probing: Phone Number found at position 8 with 2 comparisons.

Menu:
1. Insert Phone Numbers
2. Search Phone Number
3. Exit
Enter your choice: 3
PS G:\My Drive\DSA\DSAL>
```

Experiment No. 2

To create ADT that implements the “Set” concept

Program:

```
class setBuilder:
    def __init__(self):
         self.set = set()

    def addElements(self):
        n = int(input("No. of elements to insert: "))
        for i in range(n):
            element = input(f"Enter element {i}: ")
            if element not in self.set:
                self.set.add(element)

    def removeElement(self,element):
         self.set.remove(element)

    def setUnion(self, s2):
        unionSet = set()
        unionSet = self.set.copy()
        for element in s2.set:
            if element not in self.set:
                unionSet.add(element)
        return unionSet

    def setIntersection(self, set2):
        intersectionSet = set()
        for element in self.set:
            if element in set2.set:
                intersectionSet.add(element)
        return intersectionSet

    def setDifference(self, set2):
        differenceSet = set()
        for element in self.set:
            if element not in set2.set:
                differenceSet.add(element)
        return differenceSet

    def checkMembership(self, ch):
        element = input("Enter element to check: ")
        if element in self.set:
            print(f"{element} is present in Set{ch}")
        else:
            print(f"{element} is not present in Set{ch}")

def main():
     set1 = setBuilder()
     set2 = setBuilder()
     while True:
        print("**SET OPERATIONS**")
        print("1. Insert elements")
        print("2. Remove element")
        print("3. Display both sets")
        print("4. Find union of set 1 and set 2")
        print("5. Set Intersection")
        print("6. Set Difference")
        print("7. Check membership")
        print("8. Exit")
        choice = input("Enter your choice: ")
        if choice == '1':
            ch = input("Insert elements in Set '1' or Set '2'?")
            if ch == '1':
                print("Enter data for Set1")
                set1.addElements()
            elif ch == '2':
                print("Enter data for Set2")
                set2.addElements()
        elif choice == '2':
            ch = input("Remove element from  Set '1' or Set '2'?")
            if ch == '1':
                ele = input("Enter element to remove: ")
                set1.removeElement(ele)
            elif ch == '2':
                ele = input("Enter element to remove: ")
                set2.removeElement(ele)
        elif choice == '3':
            print("Set 1: ", set1.set)
            print("Set 2: ", set2.set)
        elif choice == '4':
            print("Set 1: ", set1.set)
            print("Set 2: ", set2.set)
            unionSet = set1.setUnion(set2)
            print("Union of Set1 and Set2: " , unionSet)
        elif choice == '5':
            print("Set 1: ", set1.set)
            print("Set 2: ", set2.set)
            iSet = set1.setIntersection(set2)
            print("Intersection of Set1 and Set2: ", iSet)
        elif choice == '6':
            print("Set 1: ", set1.set)
            print("Set 2: ", set2.set)
            dSet = set1.setDifference(set2)
            print("Difference of Set1 and Set2: ", dSet)
        elif choice == '7':
            ch = input("Check membership in Set '1' or Set '2'?")
            if ch == '1':
                set1.checkMembership(ch)
            elif ch == '2':
                set2.checkMembership(ch)
        elif choice == '8':
            break
        else:
            print("Invalid choice. Please try again...")

if __name__ == "__main__":
    main()
```

Output:

```
PS G:\My Drive\DSA\DSAL> python3 set.py
**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 1
Insert elements in Set '1' or Set '2'?1
Enter data for Set1
No. of elements to insert: 3
Enter element 0: A
Enter element 1: B
Enter element 2: C

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 1
Insert elements in Set '1' or Set '2'?2
Enter data for Set2
No. of elements to insert: 4
Enter element 0: C
Enter element 1: D
Enter element 2: E
Enter element 3: F

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 3
Set 1:  {'B', 'A', 'C'}
Set 2:  {'D', 'F', 'E', 'C'}

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 2
Remove element from  Set '1' or Set '2'?1
Enter element to remove: B

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 3
Set 1:  {'A', 'C'}
Set 2:  {'D', 'F', 'E', 'C'}

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 4
Set 1:  {'A', 'C'}
Set 2:  {'D', 'F', 'E', 'C'}
Union of Set1 and Set2:  {'D', 'C', 'A', 'F', 'E'}

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 5
Set 1:  {'A', 'C'}
Set 2:  {'D', 'F', 'E', 'C'}
Intersection of Set1 and Set2:  {'C'}
```

```
**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 6
Set 1:  {'A', 'C'}
Set 2:  {'D', 'F', 'E', 'C'}
Difference of Set1 and Set2:  {'A'}

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 7
Check membership in Set '1' or Set '2'?2
Enter element to check: E
E is present in Set2

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 7
Check membership in Set '1' or Set '2'?1
Enter element to check: B
B is not present in Set1

**SET OPERATIONS**
1. Insert elements
2. Remove element
3. Display both sets
4. Find union of set 1 and set 2
5. Set Intersection
6. Set Difference
7. Check membership
8. Exit
Enter your choice: 8
PS G:\My Drive\DSA\DSAL>
```

Experiment No. 3

Implementation of Binary Search Tree

Program:

```
#include <iostream>

using namespace std;

class BST
{
    struct Node
    {
        int data;
        Node *left, *right;

        Node(int val)
        {
            data = val;
            left = right = nullptr;
        }
    };
    Node *root;
public:
    BST()
    {
        root = nullptr;
    }

    Node *return_root() { return root; }

    void insert();
    void display(Node *);
    int search();
    int height(Node *);
    int findMin(Node *);
    void mirror(Node *);
    void mirrorTree();
};

void BST::insert()
{
    Node *temp = root;
    int val;
    cout << "\nEnter value of node: ";
    cin >> val;
    Node *newnode = new Node(val);

    if (root == NULL)
    {
        root = newnode;
        cout << "\nRoot node created"<<endl;
    }
    else
    {
        while (true)
        {
            if (val < temp->data)
            {
                if (temp->left == NULL)
                {
                    temp->left = newnode;
                    cout << "\nLeft node Inserted";
                    break;
                }
                else
                {
                    temp = temp->left;
                }
            }
            else if (val > temp->data)
            {
                if (temp->right == NULL)
                {
                    temp->right = newnode;
                    cout << "\nRight node inserted";
                    break;
                }
                else
                {
                    temp = temp->right;
                }
            }
            else
            {
                cout << "\nEntry repeated";
                break;
            }
        }
    }
}

void BST::display(Node *temp)
{
    if (temp != NULL)
    {
        display(temp->left);
        cout << "\t" << temp->data << " ";
        display(temp->right);
    }
}

int BST::search()
{
    Node *temp = root;
    int key;
    cout << "\nEnter the value to search: ";
    cin >> key;
    while (temp != NULL)
    {
        if (key < temp->data)
        {
            temp = temp->left;
        }
        else if (key > temp->data)
        {
            temp = temp->right;
        }
        else
        {
            cout << "\nValue " << key << " is present";
            return 1;
        }
    }
    return 0;
}

int BST::height(Node *node)
{
    if (node == NULL)
        return 0;
    else
    {
        int leftHeight = height(node->left);
        int rightHeight = height(node->right);
        return max(leftHeight, rightHeight) + 1;
    }
}

int BST::findMin(Node *node)
{
    if (node == NULL)
    {
        cout << "\n\tTree is empty";
        return -1;
    }
    while (node->left != NULL)
    {
        node = node->left;
    }
    return node->data;
}

void BST::mirror(Node *node)
{
    if (node == NULL)
        return;

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

    mirror(node->left);
    mirror(node->right);
}

void BST::mirrorTree()
{
    mirror(root);
}

int main()
{
    BST tree;
    int ch, i, val;
    do
    {
        cout << "\n====BST MENU====";
        cout << "\n1.Insert";
        cout << "\n2.Display";
        cout << "\n3.Search the node";
        cout << "\n4.Height of the tree";
        cout << "\n5.Minimum value in the tree";
        cout << "\n6.Mirror the tree";
        cout << "\n7.Exit";
        cout << "\nEnter your choice: ";
        cin >> ch;

        switch (ch)
        {
        case 1:
            tree.insert();
            cout << "\n";
            break;
        case 2:
            cout << "\nTREE:\n";
            if (tree.return_root() == NULL)
                cout << "\nTree is empty";
            else
                tree.display(tree.return_root());
            cout << "\n";
            break;
        case 3:
            i = tree.search();
            if (i == 0)
            {
                cout << "\nValue is absent";
            }
            cout << "\n";
            break;
        case 4:
            cout << "\nHeight of the tree: " << tree.height(tree.return_root());
            cout << "\n";
            break;
        case 5:
            val = tree.findMin(tree.return_root());
            if (val != -1)
                cout << "\nMinimum value in the tree: " << val;
            cout << "\n";
            break;
        case 6:
            tree.mirrorTree();
            cout << "\nTree mirrored successfully";
            cout << "\n";
            break;
        case 7:
            break;
        default:
            cout << "\nINVALID INPUT";
            cout << "\n";
            break;
        }
    } while (ch < 7);
}
```

Output:

```
PS G:\My Drive\DSA\DSAL> g++.exe .\bst.cpp -o build/bst.exe
PS G:\My Drive\DSA\DSAL> .\build\bst.exe

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 1

Enter value of node: 200

Root node created


====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 1

Enter value of node: 80

Left node Inserted

====BST MENU====
1.Insert
2.Display
```

```
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 1

Enter value of node: 340

Right node inserted

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 1

Enter value of node: 20

Left node Inserted

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 1

Enter value of node: 10

Left node Inserted

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 1

Enter value of node: 300

Left node Inserted

====BST MENU====
1.Insert
```

```
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 1

Enter value of node: 500

Right node inserted

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 2

TREE:
        10      20      80      200     300     340     500

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 3

Enter the value to search: 340

Value 340 is present

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 3

Enter the value to search: 2

Value is absent

====BST MENU====
1.Insert
```

```
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 4

Height of the tree: 4

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 5

Minimum value in the tree: 10

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 6

Tree mirrored successfully

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 2

TREE:
        500     340     300     200     80      20      10

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
```

```
Enter your choice: 5

Minimum value in the tree: 500

====BST MENU====
1.Insert
2.Display
3.Search the node
4.Height of the tree
5.Minimum value in the tree
6.Mirror the tree
7.Exit
Enter your choice: 7
PS G:\My Drive\DSA\DSAL>
```

Experiment No. 4

Construct an expression tree from the given prefix expression eg. +--a*bc/def and traverse it using post order traversal (non recursive) and then delete the entire tree.

Program:

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAX_SIZE = 100;

struct Node{
    char data;
    Node* left;
    Node* right;
    Node(char ch){
        data = ch;
        left = right = nullptr;
    }
};

class Stack {
private:
    Node* arr[MAX_SIZE];
    int top;
public:
    Stack() {
        top = -1;
    }
    void push(Node* newNode) {
        if (top >= MAX_SIZE - 1) {
            cout << "Stack Overflow" << endl;
        }
        arr[++top] = newNode;
    }
    Node* pop() {
        if (top < 0) {
            cout << "Stack Underflow" << endl;
            return nullptr;
        }
        return arr[top--];
    }

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

    Node* peek() {
        if (top < 0) {
            cout << "Stack is Empty" << endl;
            return nullptr;
        }
        return arr[top];
    }
};

class ExpressionTree {
private:
    Node* root;
public:
    ExpressionTree() {
        root = nullptr;
    }

    Node* constructTree(string prefix)
    {
        Stack stack;
        for (int i = prefix.size() -1; i >= 0; i--) {
            if (isalnum(prefix[i])) {
                Node* newNode = new Node(prefix[i]);
                stack.push(newNode);
            } else {
                Node* newNode = new Node(prefix[i]);
                newNode->left = stack.pop();
                newNode->right = stack.pop();
                stack.push(newNode);
            }
        }
        return stack.pop();
    }

    // postorder traversal (non recursive) using string reversal
    void postOrder(Node* root)
    {
        if (root == nullptr) {
            return;
        }
        Stack stack;
        string postOrder = "";
        stack.push(root);
        while (!stack.isEmpty()) {
            Node* temp = stack.peek();
            stack.pop();
            postOrder += temp->data;
            if (temp->left) {
                stack.push(temp->left);
            }
            if (temp->right) {
                stack.push(temp->right);
            }
        }

        //string reversal
        reverse(postOrder.begin(), postOrder.end());
        cout << "Postorder traversal: " << postOrder << endl;
    }

    //delete tree (non recursive)
    void deleteTree(Node* root)
    {
        if (root == nullptr) {
            return;
        }
        Stack stack;
        stack.push(root);
        while (!stack.isEmpty()) {
            Node* temp = stack.peek();
            stack.pop();
            if (temp->left) {
                stack.push(temp->left);
            }
            if (temp->right) {
                stack.push(temp->right);
            }
            delete temp;
        }
    }

};

int main() {
    ExpressionTree tree;
    string prefix;
    int choice;
    Node* root = nullptr;

    do {
        cout << "\nMenu:\n";
        cout << "1. Construct Expression Tree\n";
        cout << "2. Postorder Traversal\n";
        cout << "3. Delete Tree\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter prefix expression: ";
                cin >> prefix;
                if(root != nullptr)
                {
                    tree.deleteTree(root);
                }
                root = tree.constructTree(prefix);
                if(root != nullptr){
                    cout << "Expression tree constructed.\n";
                }
                break;
            case 2:
                if (root == nullptr) {
                    cout << "Tree is empty. Construct tree first.\n";
                } else {
                    tree.postOrder(root);
                }
                break;
            case 3:
                if (root == nullptr) {
                    cout << "Tree is empty.\n";
                } else {
                    tree.deleteTree(root);
                    root = nullptr;
                    cout << "Tree deleted.\n";
                }
                break;
            case 4:
                if(root != nullptr){
                    tree.deleteTree(root);
                }
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 4);
}
```

Output:

```
PS G:\My Drive\DSA\DSAL> g++.exe .\exptree.cpp -o build/exptree.exe
PS G:\My Drive\DSA\DSAL> .\build\exptree.exe

Menu:
1. Construct Expression Tree
2. Postorder Traversal
3. Delete Tree
4. Exit
Enter your choice: 1
Enter prefix expression: +--a*bc/def
Expression tree constructed.

Menu:
1. Construct Expression Tree
2. Postorder Traversal
3. Delete Tree
4. Exit
Enter your choice: 2
Postorder traversal: abc*-de/-f+

Menu:
1. Construct Expression Tree
2. Postorder Traversal
3. Delete Tree
4. Exit
Enter your choice: 3
Tree deleted.
```

```
Menu:
1. Construct Expression Tree
2. Postorder Traversal
3. Delete Tree
4. Exit
Enter your choice: 4
Exiting program.
PS G:\My Drive\DSA\DSAL>
```

Experiment No. 5

A Dictionary stores keywords and its meanings. Provide facility for adding new keyword, deleting keywords, updating values of entry. Provide facility to display whole data sorted in ascending /Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.

Program:

```
#include <iostream>
#include <string>

using namespace std;

struct Node {
public:
    string keyword;
    string meaning;
    Node* left;
    Node* right;
    Node(string key, string m) {
        keyword = key;
        meaning = m;
        left = nullptr;
        right = nullptr;
    }
};

Node* findMinNode(Node* node) {
    while (node->left != nullptr) {
        node = node->left;
    }
    return node;
}

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

void printInOrder(Node* root) {
    if (root != nullptr) {
        printInOrder(root->left);
        cout << root->keyword << " " << root->meaning << endl;
        printInOrder(root->right);
    }
}
```

```cpp
void printReverseInOrder(Node* root) {
    if (root != nullptr) {
        printReverseInOrder(root->right);
        cout << root->keyword << " " << root->meaning << endl;
        printReverseInOrder(root->left);
    }
}

Node* deleteKeyword(Node* root, string keyword) {
    if (root == nullptr)
        return root;
    if (keyword < root->keyword)
        root->left = deleteKeyword(root->left, keyword);
    else if (keyword > root->keyword)
        root->right = deleteKeyword(root->right, keyword);
    else {
        if (root->left == nullptr && root->right == nullptr)
            return nullptr;
        else if (root->left == nullptr)
            return root->right;
        else if (root->right == nullptr)
            return root->left;
        else {
            Node* minNode = findMinNode(root->right);
            root->keyword = minNode->keyword;
            root->meaning = minNode->meaning;
            root->right = deleteKeyword(root->right, minNode->keyword);
        }
    }
    return root;
}

string search(Node* root, string keyword, int& comparisons) {
    Node* current = root;
    comparisons = 0;
    while (current != nullptr) {
        comparisons++;
        if (keyword == current->keyword) {
            return current->meaning;
        } else if (keyword < current->keyword) {
            current = current->left;
        } else {
            current = current->right;
        }
    }
    return "Keyword not found.";
}

void updateMeaning(Node* root, string keyword, string newMeaning) {
    Node* current = root;
    while (current != nullptr) {
        if (keyword == current->keyword) {
            current->meaning = newMeaning;
            cout << "Meaning updated." << endl;
            return;
        } else if (keyword < current->keyword) {
            current = current->left;
        } else {
            current = current->right;
        }
    }
    cout << "Keyword not found." << endl;
}

int getHeight(Node* root) {
    if (root == nullptr) {
        return 0;
    }
    return 1 + max(getHeight(root->left), getHeight(root->right));
}

int main() {
    Node* root = nullptr;
    string keyword, meaning, newMeaning;
    int comparisons;
    char addMore = 'y';

    while (true) {
        cout << "\n==== Dictionary Menu ====\n";
        cout << "\n1. Add new keyword and meaning\n";
        cout << "2. Print all entries in ascending order\n";
        cout << "3. Print all entries in descending order\n";
        cout << "4. Delete a keyword from dictionary\n";
        cout << "5. Search a keyword\n";
        cout << "6. Update meaning of a keyword\n";
        cout << "7. Get maximum comparisons needed\n";
        cout << "8. Exit program\n";
        cout << "Enter your choice: ";

        int choice;
        cin >> choice;

        switch (choice) {
            case 1:
                addMore = 'y';
                while (addMore == 'y' || addMore == 'Y') {
                    cout << "Enter keyword: ";
                    cin >> keyword;
                    cout << "Enter meaning: ";
                    cin.ignore();
                    getline(cin, meaning);
                    root = insert(root, keyword, meaning);
                    cout << "Add more keywords? (y/n): ";
                    cin >> addMore;
                }
                break;
            case 2:
                if (root == nullptr)
                    cout << "No entries to display." << endl;
                else
                    printInOrder(root);
                break;
            case 3:
                if (root == nullptr)
                    cout << "No entries to display." << endl;
                else
                    printReverseInOrder(root);
                break;
            case 4:
                if (root == nullptr)
                    cout << "No entries to delete." << endl;
                else {
                    cout << "Enter keyword to delete: ";
                    cin >> keyword;
                    root = deleteKeyword(root, keyword);
                }
                break;
            case 5:
                if (root == nullptr)
                    cout << "Dictionary is empty." << endl;
                else {
                    cout << "Enter keyword to search: ";
                    cin >> keyword;
                    cout << search(root, keyword, comparisons) << " (Comparisons: "
<< comparisons << ")" << endl;
                }
                break;
            case 6:
                if (root == nullptr)
                    cout << "Dictionary is empty." << endl;
                else {
                    cout << "Enter keyword to update: ";
                    cin >> keyword;
                    cout << "Enter new meaning: ";
                    cin.ignore();
                    getline(cin, newMeaning);
                    updateMeaning(root, keyword, newMeaning);
                }
                break;
            case 7:
                cout << "Maximum comparisons needed: " << getHeight(root) -1 << endl;
                break;
            case 8:
                return 0;
            default:
                cout << "Invalid choice. Please choose a valid option." << endl;
        }
    }
}
```
Output:

```
PS G:\My Drive\DSA\DSAL> g++.exe .\bstDict.cpp -o build/bstDict.exe
PS G:\My Drive\DSA\DSAL> build/bstDict.exe

==== Dictionary Menu ====
1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 1
Enter keyword: apple
Enter meaning: fruit
Add more keywords? (y/n): y
Enter keyword: elephant
Enter meaning: large mamal
Add more keywords? (y/n): y
Enter keyword: zebra
Enter meaning: animal
Add more keywords? (y/n): y
Enter keyword: dog
Enter meaning: pet
Add more keywords? (y/n): y
Enter keyword: banana
Enter meaning: yellow fruit
Add more keywords? (y/n): y
Enter keyword: cat
Enter meaning: domestic animal
Add more keywords? (y/n): n

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 2
apple fruit
banana yellow fruit
cat domestic animal
dog pet
elephant large mamal
zebra animal

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
```

```
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 3
zebra animal
elephant large mamal
dog pet
cat domestic animal
banana yellow fruit
apple fruit

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 4
Enter keyword to delete: zebra

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 2
apple fruit
banana yellow fruit
cat domestic animal
dog pet
elephant large mamal

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 5
Enter keyword to search: elephant
large mamal (Comparisons: 3)

==== Dictionary Menu ====
1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 6
Enter keyword to update: cat
Enter new meaning: pet
Meaning updated.

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 7
Maximum comparisons needed: 4

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 2
apple fruit
banana yellow fruit
cat pet
dog pet
elephant large mamal

==== Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get maximum comparisons needed
8. Exit program
Enter your choice: 8
```

```
PS G:\My Drive\DSA\DSAL>
```

Experiment No. 6

Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent landmarks as nodes and perform DFS and BFS on that.

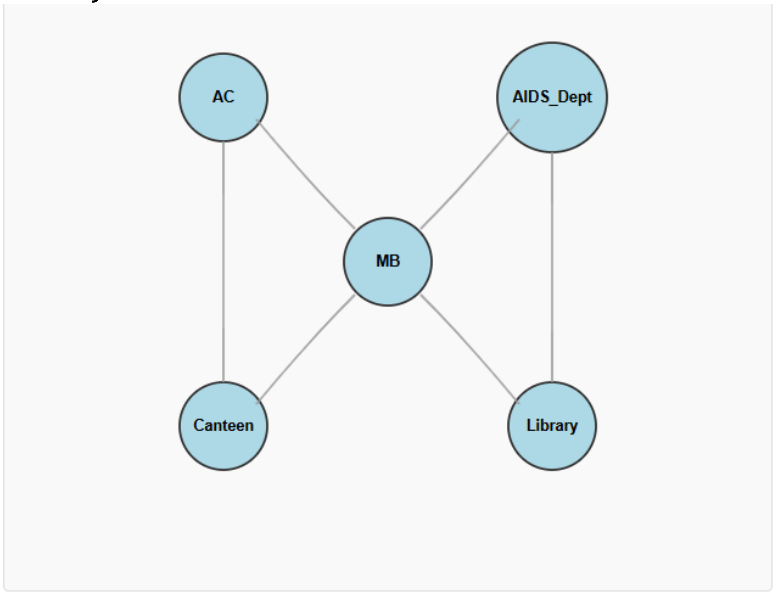
Program:

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

using namespace std;

class Graph {
private:
    int vertices;
    map<string, int> landmarkIndex;
    map<int, string> indexLandmark;
    vector<vector<int>> adjMatrix;
    map<int, vector<int>> adjList;

public:
    Graph(int v) : vertices(v) {
        adjMatrix.resize(vertices, vector<int>(vertices, 0));
    }

    void addLandmark(string name, int index) {
        landmarkIndex[name] = index;
        indexLandmark[index] = name;
    }

    void addEdge(string src, string dest) {
        if (landmarkIndex.find(src) == landmarkIndex.end() ||
landmarkIndex.find(dest) == landmarkIndex.end()) {
            cout << "Invalid landmarks entered!\n";
            return;
        }
        int u = landmarkIndex[src];
        int v = landmarkIndex[dest];
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
        adjList[u].push_back(v);
        adjList[v].push_back(u);
    }

    void displayGraph() {
        int maxWidth = 0;
        for (int i = 0; i < vertices; i++) {
            int nameLength = indexLandmark[i].length();
            if (nameLength > maxWidth) {
                maxWidth = nameLength;
            }
        }

        maxWidth += 2;

        cout << "\nAdjacency Matrix:\n" << setw(maxWidth) << "";
        for (int i = 0; i < vertices; i++) {
            cout << setw(maxWidth) << left << indexLandmark[i];
        }
        cout << "\n";

        for (int i = 0; i < vertices; i++) {
            cout << setw(maxWidth) << left << indexLandmark[i];
            for (int j = 0; j < vertices; j++) {
                cout << setw(maxWidth) << left << adjMatrix[i][j];
            }
            cout << endl;
        }

        cout << "\nAdjacency List:\n";
        for (auto &pair : adjList) {
            cout << setw(maxWidth) << left << indexLandmark[pair.first] << "-> ";
            for (int neighbor : pair.second) {
                cout << indexLandmark[neighbor] << " ";
            }
            cout << endl;
        }
    }

    void dfs() {
        vector<bool> visited(vertices, false);
        cout << "DFS Traversal: ";

        for (int i = 0; i < vertices; i++) {
            if (!visited[i]) {
                stack<int> s;
                s.push(i);

                while (!s.empty()) {
                    int node = s.top();
                    s.pop();
                    if (!visited[node]) {
                        cout << indexLandmark[node] << " ";
                        visited[node] = true;

                        for (int j = vertices - 1; j >= 0; j--) {
                            if (adjMatrix[node][j] != 0 && !visited[j]) {
                                s.push(j);
                            }
                        }
                    }
                }
            }
        }
        cout << endl;
    }

    void bfs() {
        vector<bool> visited(vertices, false);
        cout << "BFS Traversal: ";

        for (int i = 0; i < vertices; i++) {
            if (!visited[i]) {
                queue<int> q;
                visited[i] = true;
                q.push(i);

                while (!q.empty()) {
                    int node = q.front();
                    q.pop();
                    cout << indexLandmark[node] << " ";

                    for (int neighbor : adjList[node]) {
                        if (!visited[neighbor]) {
                            visited[neighbor] = true;
                            q.push(neighbor);
                        }
                    }
                }
            }
        }
        cout << endl;
    }

    void dfsRecursiveUtil(int node, vector<bool> &visited) {
        cout << indexLandmark[node] << " ";
        visited[node] = true;

        for (int neighbor : adjList[node]) {
            if (!visited[neighbor]) {
                dfsRecursiveUtil(neighbor, visited);
            }
        }
    }

    void dfsRecursive() {
        vector<bool> visited(vertices, false);
        cout << "DFS Recursive Traversal: ";

        for (int i = 0; i < vertices; i++) {
            if (!visited[i]) {
                dfsRecursiveUtil(i, visited);
            }
        }
        cout << endl;
    }

    void dfsNonRecursive() {
        vector<bool> visited(vertices, false);
        cout << "DFS Non-Recursive Traversal: ";

        for (int i = 0; i < vertices; i++) {
            if (!visited[i]) {
                stack<int> s;
                s.push(i);

                while (!s.empty()) {
                    int node = s.top();
                    s.pop();

                    if (!visited[node]) {
                        cout << indexLandmark[node] << " ";
                        visited[node] = true;

                        for (int j = vertices - 1; j >= 0; j--) {
                            if (adjMatrix[node][j] != 0 && !visited[j]) {
                                s.push(j);
                            }
                        }
                    }
                }
            }
        }
        cout << endl;
    }
};

int main() {
    int landmarks_no;
    cout << "Enter the number of landmarks: ";
    cin >> landmarks_no;
    cin.ignore();
```

```
Graph g(landmarks_no);
cout << "Enter landmark names:\n";
for (int i = 0; i < landmarks_no; i++) {
    string name;
    getline(cin, name);
    g.addLandmark(name, i);
}

while (true) {
    cout << "\nMenu:\n";
    cout << "1. Add Edge\n";
    cout << "2. Display Graph\n";
    cout << "3. Perform Recursive DFS\n";
    cout << "4. Perform Non-Recursive DFS\n";
    cout << "5. Perform BFS\n";
    cout << "6. Exit\n";
    cout << "Enter choice: ";

    int choice;
    cin >> choice;
    cin.ignore();

    switch (choice) {
        case 1: {
            string src, dest;
            cout << "Enter source landmark: ";
            getline(cin, src);
            cout << "Enter destination landmark: ";
            getline(cin, dest);
            g.addEdge(src, dest);
            break;
        }
        case 2:
            g.displayGraph();
            break;
        case 3:
            g.dfsRecursive();
            break;
        case 4:
            g.dfsNonRecursive();
            break;
        case 5:
            g.bfs();
            break;
        case 6:
            cout << "Exiting program.\n";
            return 0;
        default:
            cout << "Invalid choice! Try again.\n";
    }
    }
    return 0;
}
```
Output:
```
PS G:\My Drive\DSA\DSAL> g++ .\bfsdfsgraph.cpp -o build/bfsdfsgraph
PS G:\My Drive\DSA\DSAL> build/bfsdfsgraph
Enter the number of landmarks: 5
Enter landmark names:
MB
AC
AIDS_Dept
Canteen
Library

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 1
Enter source landmark: MB
Enter destination landmark: AC

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 1
Enter source landmark: MB
Enter destination landmark: AIDS_Dept

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 1
Enter source landmark: MB
Enter destination landmark: Library


Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 1
Enter source landmark: MB
Enter destination landmark: Canteen

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 1
Enter source landmark: AC
Enter destination landmark: Canteen

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 1
Enter source landmark: AIDS_Dept
Enter destination landmark: Library

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 2
```

```
Adjacency Matrix:
           MB         AC         AIDS_Dept  Canteen    Library
MB         0          1          1          1          1
AC         1          0          0          1          0
AIDS_Dept  1          0          0          0          1
Canteen    1          1          0          0          0
Library    1          0          1          0          0

Adjacency List:
MB         -> AC AIDS_Dept Library Canteen
AC         -> MB Canteen
AIDS_Dept  -> MB Library
Canteen    -> MB AC
Library    -> MB AIDS_Dept

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 3
DFS Recursive Traversal: MB AC Canteen AIDS_Dept Library

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 4
DFS Non-Recursive Traversal: MB AC Canteen AIDS_Dept Library

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
```

```
Enter choice: 5
BFS Traversal: MB AC AIDS_Dept Library Canteen

Menu:
1. Add Edge
2. Display Graph
3. Perform Recursive DFS
4. Perform Non-Recursive DFS
5. Perform BFS
6. Exit
Enter choice: 6
Exiting program.
PS G:\My Drive\DSA\DSAL>
```



Adjacency Matrix:

	MB	AC	AIDS_Dept	Canteen	Library
MB	0	1	1	1	1
AC	1	0	0	1	0
AIDS_Dept	1	0	0	0	1
Canteen	1	1	0	0	0
Library	1	0	1	0	0

Adjacency List:

```
MB → AC AIDS_Dept Library Canteen
AC → MB Canteen
AIDS_Dept → MB Library
Canteen → MB AC
Library → MB AIDS_Dept
```

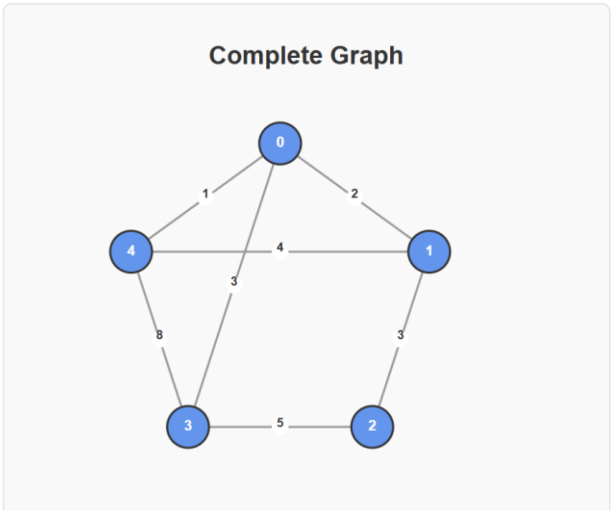
Experiment No. 7

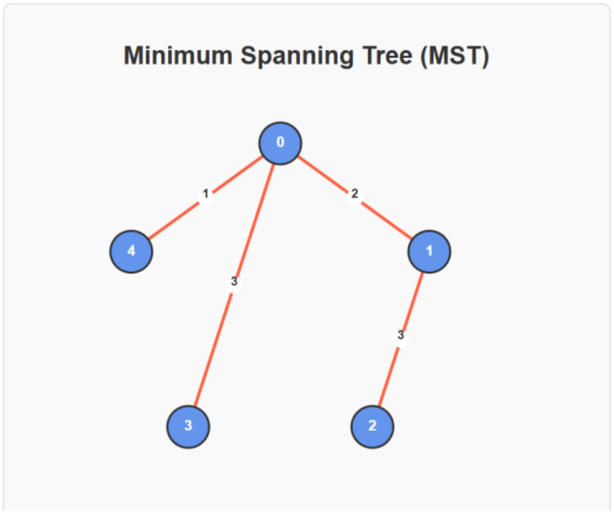
You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

Program:
```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

class Prims {
private:
    int vertices, edges;
    vector<vector<int>> adjMatrix;

public:
    Prims(int v, int e) {
        vertices = v;
        edges = e;
        adjMatrix.resize(vertices, vector<int>(vertices, 0));
    }

    void add_edge() {
        cout << "Enter edges (source destination weight):\n";
        for (int i = 0; i < edges; i++) {
            int source, dest, weight;
            cin >> source >> dest >> weight;

            if (source >= 0 && source < vertices && dest >= 0 && dest < vertices) {
                adjMatrix[source][dest] = weight;
                adjMatrix[dest][source] = weight;
            } else {
                cout << "Invalid edge! Please enter valid vertices.\n";
                i--;
            }
        }
    }

    void display() {
        cout << "\nAdjacency Matrix:\n  ";
        for (int i = 0; i < vertices; i++) {
            cout << i << " ";
        }
        cout << "\n";

        for (int i = 0; i < vertices; i++) {
            cout << i << " ";
            for (int j = 0; j < vertices; j++) {
                cout << adjMatrix[i][j] << " ";
            }
            cout << "\n";
        }
    }

    int minkey(bool visited[], int key[]) {
        int min = INT_MAX, min_index;
        for (int i = 0; i < vertices; i++) {
            if (!visited[i] && key[i] < min) {
                min = key[i];
                min_index = i;
            }
        }
        return min_index;
    }

    void prims() {
        int parent[vertices];
        int key[vertices];
        bool visited[vertices];

        for (int i = 0; i < vertices; i++) {
            key[i] = INT_MAX;
            visited[i] = false;
        }

        key[0] = 0;
        parent[0] = -1;

        for (int count = 0; count < vertices - 1; count++) {
            int u = minkey(visited, key);
            visited[u] = true;

            for (int v = 0; v < vertices; v++) {
                if (adjMatrix[u][v] != 0 && !visited[v] && adjMatrix[u][v] < key[v])
{
                    parent[v] = u;
                    key[v] = adjMatrix[u][v];
                }
            }
        }

        printMST(parent);
    }
    void printMST(int parent[]) {
        cout << "\nMinimum Spanning Tree (Phone Line Connections):" << endl;
        cout << "Edge \tWeight" << endl;
        int total_cost = 0;

        for (int i = 1; i < vertices; i++) {
            cout << parent[i] << " - " << i << "\t" << adjMatrix[parent[i]][i] <<
endl;
            total_cost += adjMatrix[parent[i]][i];
        }

        cout << "\nTotal cost of the minimum spanning tree: " << total_cost << endl;
        cout << "\nThese are the optimal phone line connections between offices." <<
endl;
    }
};

int main() {
    int vertices, edges;
    cout << "Enter the number of offices (vertices): ";
    cin >> vertices;
    cout << "Enter the number of possible connections (edges): ";
    cin >> edges;

    Prims p(vertices, edges);
    p.add_edge();

    int choice;
    do {
        cout << "\nMenu:\n";
        cout << "1. Print Adjacency Matrix\n";
        cout << "2. Find Minimum Spanning Tree using Prim's Algorithm\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                p.display();
                break;
            case 2:
                p.prims();
                break;
            case 3:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 3);
    return 0;
}
```
Output:

```
PS G:\My Drive\DSA\DSAL> g++ .\prims.cpp -o build/prims
PS G:\My Drive\DSA\DSAL> build/prims
Enter the number of offices (vertices): 5
Enter the number of possible connections (edges): 7
Enter edges (source destination weight):
0 1 2
1 2 3
2 3 5
3 4 8
4 0 1
1 4 4
0 3 3

Menu:
1. Print Adjacency Matrix
2. Find Minimum Spanning Tree using Prim's Algorithm
3. Exit
Enter your choice: 1

Adjacency Matrix:
  0 1 2 3 4
0 0 2 0 3 1
1 2 0 3 0 4
2 0 3 0 5 0
3 3 0 5 0 8
4 1 4 0 8 0

Menu:
1. Print Adjacency Matrix
2. Find Minimum Spanning Tree using Prim's Algorithm
3. Exit
Enter your choice: 2

Minimum Spanning Tree (Phone Line Connections):
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   3
0 - 4   1

Total cost of the minimum spanning tree: 9
```

These are the optimal phone line connections between offices.





Experiment No. 8

Given sequence k = k1<K2…< Kn of n sorted keys, with a search probability pi for each key ki. Build the Binary search tree that has the least search cost given the access probability for each key?

Program:

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

struct TreeNode {
    string key;
    TreeNode* left;
    TreeNode* right;

    TreeNode(string k) : key(k), left(nullptr), right(nullptr) {}
};

double calculateOBSTWeight(const double* P, const double* Q, int n) {
    double W[101][101] = {0};

    for (int i = 0; i <= n; i++) {
        W[i][i] = Q[i];
    }
    for (int i = 0; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            W[i][j] = W[i][j - 1] + P[j - 1] + Q[j];
        }
    }

    cout << "Weights in OBST format by level:" << endl;
    for (int level = 0; level <= n; level++) {
        for (int i = 0; i + level <= n; i++) {
            int j = i + level;
            cout << "W[" << i << "][" << j << "] = " << W[i][j] << " ";
        }
        cout << endl;
    }

    return W[0][n];
}

double calculateOBSTCost(const double* P, const double* Q, int n) {
    double W[101][101] = {0};
    double C[101][101] = {0};

    for (int i = 0; i <= n; i++) {
        W[i][i] = Q[i];
        C[i][i] = 0;
    }
    for (int i = 0; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            W[i][j] = W[i][j - 1] + P[j - 1] + Q[j];
        }
    }
    for (int level = 1; level <= n; level++) {
        for (int i = 0; i + level <= n; i++) {
            int j = i + level;
            C[i][j] = INT_MAX;
            for (int r = i; r < j; r++) {
                double cost = C[i][r] + C[r + 1][j] + W[i][j];
                if (cost < C[i][j]) {
                    C[i][j] = cost;
                }
            }
        }
    }

    cout << "Costs in OBST format by level:" << endl;
    for (int level = 0; level <= n; level++) {
        for (int i = 0; i + level <= n; i++) {
            int j = i + level;
            cout << "C[" << i << "][" << j << "] = " << C[i][j] << " ";
        }
        cout << endl;
    }

    return C[0][n];
}

int calculateOBSTRoots(const double* P, const double* Q, int n, int R[101][101]) {
    double W[101][101] = {0};
    double C[101][101] = {0};

    for (int i = 0; i <= n; i++) {
        W[i][i] = Q[i];
        C[i][i] = 0;
    }

    for (int i = 0; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            W[i][j] = W[i][j - 1] + P[j - 1] + Q[j];
        }
    }
    for (int level = 1; level <= n; level++) {
        for (int i = 0; i + level <= n; i++) {
            int j = i + level;
            C[i][j] = INT_MAX;
            for (int r = i + 1; r <= j; r++) {
                double cost = C[i][r - 1] + C[r][j] + W[i][j];
                if (cost < C[i][j]) {
                    C[i][j] = cost;
                    R[i][j] = r;
                }
            }
        }
    }

    cout << "Roots in OBST format by level:" << endl;
    for (int level = 1; level <= n; level++) {
        for (int i = 0; i + level <= n; i++) {
            int j = i + level;
            cout << "R[" << i << "][" << j << "] = " << R[i][j] << " ";
        }
        cout << endl;
    }

    return R[0][n];
}

// Function to build the OBST using the R table
TreeNode* buildOptimalBST(int i, int j, int R[101][101], const string* keys) {
    if (i > j) return nullptr;

    int r = R[i][j];
    if (r == 0) return nullptr;

    TreeNode* root = new TreeNode(keys[r-1]);
    root->left = buildOptimalBST(i, r-1, R, keys);
    root->right = buildOptimalBST(r, j, R, keys);

    return root;
}

void inorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    inorderTraversal(root->left);
    cout << root->key << " ";
    inorderTraversal(root->right);
}

void preorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    cout << root->key << " ";
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

void postorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    cout << root->key << " ";
}

int main() {
    int n = 0;
    string keys[100];
    double P[100], Q[101];
    double weight = 0, cost = 0;
    int R[101][101] = {0};
    int root = 0;
    TreeNode* rootNode = nullptr;
    bool dataEntered = false;
    int choice;

    cout << "Enter the number of keys: ";
    cin >> n;

    cout << "Enter the keywords: ";
    for (int i = 0; i < n; i++)
        cin >> keys[i];

    cout << "Enter the probabilities of keys (P): ";
    for (int i = 0; i < n; i++)
        cin >> P[i];

    cout << "Enter the probabilities of dummy keys (Q): ";
    for (int i = 0; i <= n; i++)
        cin >> Q[i];

    dataEntered = true;
    cout << "Data entered successfully!\n";

    do {
        cout << "\n\n==== OPTIMAL BINARY SEARCH TREE MENU ====\n";
        cout << "1. Calculate and display OBST weight\n";
        cout << "2. Calculate and display OBST cost\n";
        cout << "3. Calculate and display OBST roots\n";
        cout << "4. Build OBST and display tree traversals\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                weight = calculateOBSTWeight(P, Q, n);
                cout << "The weight of the OBST is: " << weight << endl;
                break;
            case 2:
                cost = calculateOBSTCost(P, Q, n);
                cout << "The cost of the OBST is: " << cost << endl;
                break;
            case 3:
                root = calculateOBSTRoots(P, Q, n, R);
                cout << "The root of the OBST is: " << root << endl;
                break;
            case 4:
                rootNode = buildOptimalBST(0, n, R, keys);

                cout << "\nTree Traversals:" << endl;
                cout << "Inorder: ";
                inorderTraversal(rootNode);
                cout << endl;

                cout << "Preorder: ";
                preorderTraversal(rootNode);
                cout << endl;

                cout << "Postorder: ";
                postorderTraversal(rootNode);
                cout << endl;
                break;
            case 5:
                cout << "Exiting program. Goodbye!\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
                break;
        }
    } while (choice != 5);
    return 0;
}
```
Output:

```
PS G:\My Drive\DSA\DSAL> g++ ./obst.cpp -o build/obst
PS G:\My Drive\DSA\DSAL> build/obst
Enter the number of keys: 4
Enter the keywords: out float if while
Enter the probabilities of keys (P): 1 4 2 1
Enter the probabilities of dummy keys (Q): 4 2 4 1 1
Data entered successfully!


==== OPTIMAL BINARY SEARCH TREE MENU ====
1. Calculate and display OBST weight
2. Calculate and display OBST cost
3. Calculate and display OBST roots
4. Build OBST and display tree traversals
5. Exit
Enter your choice: 1
Weights in OBST format by level:
W[0][0] = 4 W[1][1] = 2 W[2][2] = 4 W[3][3] = 1 W[4][4] = 1
W[0][1] = 7 W[1][2] = 10 W[2][3] = 7 W[3][4] = 3
W[0][2] = 15 W[1][3] = 13 W[2][4] = 9
W[0][3] = 18 W[1][4] = 15
W[0][4] = 20
The weight of the OBST is: 20


==== OPTIMAL BINARY SEARCH TREE MENU ====
1. Calculate and display OBST weight
2. Calculate and display OBST cost
3. Calculate and display OBST roots
4. Build OBST and display tree traversals
5. Exit
Enter your choice: 2
Costs in OBST format by level:
C[0][0] = 0 C[1][1] = 0 C[2][2] = 0 C[3][3] = 0 C[4][4] = 0
C[0][1] = 7 C[1][2] = 10 C[2][3] = 7 C[3][4] = 3
C[0][2] = 22 C[1][3] = 20 C[2][4] = 12
C[0][3] = 32 C[1][4] = 27
C[0][4] = 39
The cost of the OBST is: 39


==== OPTIMAL BINARY SEARCH TREE MENU ====
1. Calculate and display OBST weight
2. Calculate and display OBST cost
3. Calculate and display OBST roots
4. Build OBST and display tree traversals
5. Exit
Enter your choice: 3
Roots in OBST format by level:
R[0][1] = 1 R[1][2] = 2 R[2][3] = 3 R[3][4] = 4
R[0][2] = 2 R[1][3] = 2 R[2][4] = 3
R[0][3] = 2 R[1][4] = 2
R[0][4] = 2
The root of the OBST is: 2
```

```
==== OPTIMAL BINARY SEARCH TREE MENU ====
1. Calculate and display OBST weight
2. Calculate and display OBST cost
3. Calculate and display OBST roots
4. Build OBST and display tree traversals
5. Exit
Enter your choice: 4

Tree Traversals:
Inorder: out float if while
Preorder: float out if while
Postorder: out while if float


==== OPTIMAL BINARY SEARCH TREE MENU ====
1. Calculate and display OBST weight
2. Calculate and display OBST cost
3. Calculate and display OBST roots
4. Build OBST and display tree traversals
5. Exit
Enter your choice: 5
Exiting program. Goodbye!
PS G:\My Drive\DSA\DSAL>
```

Tree Information:

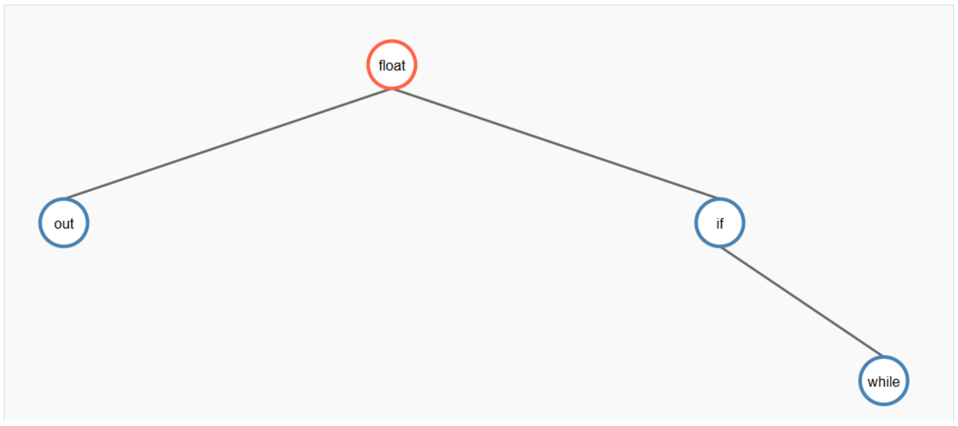
Keywords: out, float, if, while

Inorder: out float if while

Preorder: float out if while

Postorder: out while if float

Root: float



Experiment No. 10

Consider a scenario for hospital to cater services to different kinds of patients as Serious (top priority), b) non-serious (medium priority), c) General Check-up (Least priority). Implement the priority queue to cater services to the patients.

Program:

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

struct Patient {
    string name;
    int priority;
    Patient* next;
};

class PriorityQueue {
private:
    Patient* head;

public:
    PriorityQueue()
    {
        head = nullptr;
    }

    void enqueue(string name, int priority) {
        Patient* newPatient = new Patient{name, priority, nullptr};
        if (!head || priority < head->priority) {
            newPatient->next = head;
            head = newPatient;
        } else {
            Patient* temp = head;
            while (temp->next && temp->next->priority <= priority) {
                temp = temp->next;
            }
            newPatient->next = temp->next;
            temp->next = newPatient;
        }
    }

    void dequeue() {
        if (!head) {
            cout << "Queue is empty. No patients to serve.\n";
            return;
        }
        Patient* temp = head;
        head = head->next;
        cout << "Serving patient: " << temp->name << " (Priority: " << temp->priority
<< ")\n";
        delete temp;
    }

    void display() {
        if (!head) {
            cout << "No patients in the queue.\n";
            return;
        }
        Patient* temp = head;
        cout << "Patients in the queue:\n";
        while (temp) {
            cout << temp->name << " (Priority: " << temp->priority << ")\n";
            temp = temp->next;
        }
    }

    ~PriorityQueue() {
        while (head) {
            Patient* temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    PriorityQueue pq;
    int choice;
    do {
        cout << "\nHospital Priority Queue System\n";
        cout << "1. Add Patient\n";
        cout << "2. Serve Patient\n";
        cout << "3. Display Patients\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore(); // Clear the input buffer

        switch (choice) {
            case 1: {
                string name;
                int priority;
                cout << "Enter patient name: ";
                getline(cin, name); // Use getline to allow spaces in the name
                cout << "Enter priority (1: Serious, 2: Non-serious, 3: General
Check-up): ";
                cin >> priority;
                pq.enqueue(name, priority);
                break;
            }
            case 2:
                pq.dequeue();
                break;
            case 3:
                pq.display();
                break;
            case 4:
                cout << "Exiting the system. Goodbye!\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 4);

    return 0;
}
```

Output:

```
PS G:\My Drive\DSA\DSAL> g++ .\prioqueue.cpp -o build/prio
PS G:\My Drive\DSA\DSAL> build/prio

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name:
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 1

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Someone
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 3

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Random Person
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 2
```

```
Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Demo
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 1

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
   (Priority: 1)
Demo (Priority: 1)
Random Person  (Priority: 2)
Someone (Priority: 3)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 2
Serving patient:    (Priority: 1)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 2
Serving patient: Demo (Priority: 1)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
Random Person  (Priority: 2)
```

```
Someone (Priority: 3)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Middle
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 2

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
Random Person  (Priority: 2)
Middle (Priority: 2)
Someone (Priority: 3)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 2
Serving patient: Random Person  (Priority: 2)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
Middle (Priority: 2)
Someone (Priority: 3)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
```

```
Enter your choice: 4
Exiting the system. Goodbye!
PS G:\My Drive\DSA\DSAL>
```

Experiment No. 11

The department maintains student information. The file contains roll number, name, division and address. Allow users to add, delete information about students. Display information of a particular employee. If the record of the student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use a sequential file to maintain the data.

Program:

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <iomanip>

using namespace std;

struct Student {
    string rollNumber;
    string name;
    string division;
    string address;
};

void addStudent() {
    ofstream outFile("students.txt", ios::app);
    if (!outFile) {
        cerr << "Error opening file for writing." << endl;
        return;
    }

    Student student;
    cout << "Enter Roll Number: ";
    getline(cin >> ws, student.rollNumber);
    cout << "Enter Name: ";
    getline(cin, student.name);
    cout << "Enter Division: ";
    getline(cin, student.division);
    cout << "Enter Address: ";
    getline(cin, student.address);

    outFile << student.rollNumber << "," << student.name << "," << student.division
<< "," << student.address << endl;
    outFile.close();
    cout << "Student added successfully." << endl;
}
```

```
void searchStudent(const string& roll) {
    ifstream inFile("students.txt");
    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return;
    }

    string line;
    bool found = false;
    while (getline(inFile, line)) {
        stringstream ss(line);
        string tokens[4];
        for (int i = 0; i < 4; ++i) {
            getline(ss, tokens[i], ',');
        }

        if (tokens[0] == roll) {
            cout << setw(15) << left << "Roll Number" << setw(20) << left << "Name"
<< setw(15) << left << "Division" << setw(30) << left << "Address" << endl;
            cout << setw(15) << left << tokens[0] << setw(20) << left << tokens[1] <<
setw(15) << left << tokens[2] << setw(30) << left << tokens[3] << endl;
            found = true;
            break;
        }
    }

    inFile.close();
    if (!found) {
        cout << "Student with roll number " << roll << " not found." << endl;
    }
}

void deleteStudent(const string& roll) {
    ifstream inFile("students.txt");
    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return;
    }

    ofstream tempFile("temp.txt");
    if (!tempFile) {
        cerr << "Error creating temporary file." << endl;
        inFile.close();
        return;
    }

    string line;
    bool found = false;
    while (getline(inFile, line)) {
        stringstream ss(line);
        string tokens[4];
        for (int i = 0; i < 4; ++i) {
            getline(ss, tokens[i], ',');
        }

        if (tokens[0] == roll) {
            found = true;
        } else {
            tempFile << line << endl;
        }
    }

    inFile.close();
    tempFile.close();

    if (found) {
        remove("students.txt");
        rename("temp.txt", "students.txt");
        cout << "Student with roll number " << roll << " deleted successfully." <<
endl;
    } else {
        remove("temp.txt");
        cout << "Student with roll number " << roll << " not found." << endl;
    }
}

void displayAllStudents() {
    ifstream inFile("students.txt");
    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return;
    }
    string line;
    cout << setw(15) << left << "Roll Number" << setw(20) << left << "Name" <<
setw(15) << left << "Division" << setw(30) << left << "Address" << endl;
    while (getline(inFile, line)) {
        stringstream ss(line);
        string tokens[4];
        for (int i = 0; i < 4; ++i) {
            getline(ss, tokens[i], ',');
        }
        cout << setw(15) << left << tokens[0] << setw(20) << left << tokens[1] <<
setw(15) << left << tokens[2] << setw(30) << left << tokens[3] << endl;
    }
    inFile.close();
}

int main() {
    int choice;
    string rollNumber;
    fstream file("students.txt", ios::in | ios::out | ios::trunc);
    file.close();

    do {
        cout << "\nStudent Information System" << endl;
        cout << "1. Add Student" << endl;
        cout << "2. Search Student" << endl;
        cout << "3. Delete Student" << endl;
        cout << "4. Display all Students" << endl;
        cout << "0. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                addStudent();
                break;
            case 2:
                cout << "Enter Roll Number to search: ";
                cin >> rollNumber;
                searchStudent(rollNumber);
                break;
            case 3:
                cout << "Enter Roll Number to delete: ";
                cin >> rollNumber;
                deleteStudent(rollNumber);
                break;
            case 4:
                displayAllStudents();
                break;
            case 0:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 0);

    return 0;
}
```
Output:

```
PS G:\My Drive\DSA\DSAL> ./build/studentfile

Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
```

```
Enter your choice: 1
Enter Roll Number: 78
Enter Name:
Enter Division: A
Enter Address: Pune
Student added successfully.

Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
Enter your choice: 1
Enter Roll Number: 55
Enter Name: Someone
Enter Division: C
Enter Address: Mumbai
Student added successfully.

Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
Enter your choice: 1
Enter Roll Number: 2
Enter Name: Random
Enter Division: B
Enter Address: Pune
Student added successfully.

Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
Enter your choice: 1
Enter Roll Number: 67
Enter Name: Anonymous
Enter Division: A
Enter Address: Mumbai
Student added successfully.
Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
Enter your choice: 4
Roll Number    Name                Division       Address
78                      A              Pune
55             Someone             C              Mumbai
2              Random              B              Pune
67             Anonymous           A              Mumbai

Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
Enter your choice: 2
Enter Roll Number to search: 2
Roll Number    Name                Division       Address
2              Random              B              Pune

Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
Enter your choice: 3
Enter Roll Number to delete: 55
Student with roll number 55 deleted successfully.

Student Information System
1. Add Student
2. Search Student
3. Delete Student
4. Display all Students
0. Exit
Enter your choice: 4
Roll Number    Name                Division       Address
78                      A              Pune
2              Random              B              Pune
```

```
67              Anonymous           A               Mumbai
```

Experiment No. 12

Implementation of a direct access file -Insertion and deletion of a record from a direct access file

Program:

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>

using namespace std;

struct Person {
    int id;
    char name[50];
    int age;
};

long calculateOffset(int id, long recordSize) {
    return (id - 1) * recordSize;
}

void insertPerson() {
    fstream outFile("persons.dat", ios::binary | ios::in | ios::out);

    if (!outFile) {
        ofstream createFile("persons.dat", ios::binary);
        createFile.close();
        outFile.open("persons.dat", ios::binary | ios::in | ios::out);
    }

    Person person;
    cout << "Enter ID: ";
    cin >> person.id;
    cin.ignore();
    cout << "Enter Name: ";
    cin.getline(person.name, 50);
    cout << "Enter Age: ";
    cin >> person.age;

    long recordSize = sizeof(Person);
    long offset = calculateOffset(person.id, recordSize);

    outFile.seekp(0, ios::end);
    long fileSize = outFile.tellp();
    if (fileSize < offset + recordSize) {
        Person dummy;
        dummy.id = -1;
        outFile.seekp(0, ios::end);
        while (outFile.tellp() < offset) {
            outFile.write((char*)(&dummy), sizeof(Person));
        }
    }

    outFile.seekp(offset);
    outFile.write((char*)(&person), sizeof(Person));

    outFile.close();
    cout << "Person record inserted successfully." << endl;
}

void searchPerson(int id) {
    ifstream inFile("persons.dat", ios::binary);
    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return;
    }

    Person person;
    long recordSize = sizeof(Person);
    long offset = calculateOffset(id, recordSize);

    inFile.seekg(0, ios::end);
    long fileSize = inFile.tellg();

    if (offset >= fileSize) {
        cout << "Person with ID " << id << " not found." << endl;
        inFile.close();
        return;
    }

    inFile.seekg(offset);
    inFile.read((char*)(&person), sizeof(Person));

    if (inFile.gcount() == sizeof(Person) && person.id == id) {
        cout << "ID: " << person.id << endl;
        cout << "Name: " << person.name << endl;
        cout << "Age: " << person.age << endl;
    } else {
        cout << "Person with ID " << id << " not found." << endl;
    }

    inFile.close();
}
```

```
void deletePerson(int id) {
    fstream file("persons.dat", ios::binary | ios::in | ios::out);
    if (!file) {
        cerr << "Error opening file." << endl;
        return;
    }

    Person person;
    long recordSize = sizeof(Person);
    long offset = calculateOffset(id, recordSize);

    // Check if this offset is beyond the end of the file
    file.seekg(0, ios::end);
    long fileSize = file.tellg();

    if (offset >= fileSize) {
        cout << "Person with ID " << id << " not found." << endl;
        file.close();
        return;
    }

    file.seekg(offset);
    file.read(reinterpret_cast<char*>(&person), sizeof(Person));

    if (file.gcount() == sizeof(Person) && person.id == id) {
        // Logically delete the record by setting ID to -1
        person.id = -1;
        file.seekp(offset);
        file.write(reinterpret_cast<char*>(&person), sizeof(Person));
        cout << "Person with ID " << id << " deleted." << endl;
    } else {
        cout << "Person with ID " << id << " not found." << endl;
    }

    file.close();
}

void displayAllPersons() {
    ifstream inFile("persons.dat", ios::binary);
    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return;
    }

    Person person;
    cout << left << setw(10) << "ID" << setw(50) << "Name" << setw(10) << "Age" <<
endl;
    cout << string(70, '-') << endl;

    while (inFile.read(reinterpret_cast<char*>(&person), sizeof(Person))) {
        if (person.id != -1) { // Skip logically deleted records
```

```
            cout << left << setw(10) << person.id
                 << setw(50) << person.name
                 << setw(10) << person.age << endl;
        }
    }

    inFile.close();
}

void displayAllPersonsWithEmptySlots() {
    ifstream inFile("persons.dat", ios::binary);
    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return;
    }

    Person person;
    cout << left << setw(10) << "ID" << setw(50) << "Name" << setw(10) << "Age" <<
endl;
    cout << string(70, '-') << endl;

    int recordNumber = 1;
    while (inFile.read(reinterpret_cast<char*>(&person), sizeof(Person))) {
        cout << left << setw(10) << (person.id == -1 ? "Empty" :
to_string(person.id))
             << setw(50) << (person.id == -1 ? "Empty Slot" : person.name)
             << setw(10) << (person.id == -1 ? "-" : to_string(person.age)) << endl;
        recordNumber++;
    }

    inFile.close();
}

int main() {
    int choice, id;
    fstream file("persons.dat", ios::binary | ios::in | ios::out | ios::trunc);
    file.close();

    do {
        cout << "\nDirect Access File - Person Records" << endl;
        cout << "1. Insert Person" << endl;
        cout << "2. Search Person" << endl;
        cout << "3. Delete Person" << endl;
        cout << "4. Display All Persons" << endl;
        cout << "5. Display All Persons (With Empty Slots)" << endl;
        cout << "0. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                insertPerson();
                break;
            case 2:
                cout << "Enter ID to search: ";
                cin >> id;
                searchPerson(id);
                break;
            case 3:
                cout << "Enter ID to delete: ";
                cin >> id;
                deletePerson(id);
                break;
            case 4:
                displayAllPersons();
                break;
            case 5:
                displayAllPersonsWithEmptySlots();
                break;
            case 0:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 0);

    return 0;
}
```
Output:

```
PS G:\My Drive\DSA\DSAL> g++ .\directAccess.cpp -o build/directAccess
PS G:\My Drive\DSA\DSAL> ./build/directAccess

Direct Access File - Person Records
1. Insert Person
2. Search Person
3. Delete Person
4. Display All Persons
5. Display All Persons (With Empty Slots)
0. Exit
Enter your choice: 1
Enter ID: 1
Enter Name:
Enter Age: 55
Person record inserted successfully.

Direct Access File - Person Records
1. Insert Person
```

```
2. Search Person
3. Delete Person
4. Display All Persons
5. Display All Persons (With Empty Slots)
0. Exit
Enter your choice: 1
Enter ID: 15
Enter Name: Random Person
Enter Age: 12
Person record inserted successfully.

Direct Access File - Person Records
1. Insert Person
2. Search Person
3. Delete Person
4. Display All Persons
5. Display All Persons (With Empty Slots)
0. Exit
Enter your choice: 1
Enter ID: 5
Enter Name: Someone
Enter Age: 23
Person record inserted successfully.

Direct Access File - Person Records
1. Insert Person
2. Search Person
3. Delete Person
4. Display All Persons
5. Display All Persons (With Empty Slots)
0. Exit
Enter your choice: 4
ID        Name                                              Age
----------------------------------------------------------------------
1                                                55
5         Someone                                           23
15        Random Person                                     12

Direct Access File - Person Records
1. Insert Person
2. Search Person
3. Delete Person
4. Display All Persons
5. Display All Persons (With Empty Slots)
0. Exit
```

```
Enter your choice: 5
ID        Name                                                  Age
----------------------------------------------------------------------
1                                                55
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
5         Someone                                               23
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
Empty     Empty Slot                                            -
15        Random Person                                         12

Direct Access File - Person Records
1. Insert Person
2. Search Person
3. Delete Person
4. Display All Persons
5. Display All Persons (With Empty Slots)
0. Exit
Enter your choice: 2
Enter ID to search: 5
ID: 5
Name: Someone
Age: 23

Direct Access File - Person Records
1. Insert Person
2. Search Person
3. Delete Person
4. Display All Persons
5. Display All Persons (With Empty Slots)
0. Exit
Enter your choice: 0
Exiting program.
```

Experiment No. 9

A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide a facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword

Program

```
#include <iostream>
#include <string>
#include <algorithm>
#include <queue>
#include <atomic>

using namespace std;

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

void printLevelOrder(Node* root);
Node* rightRotate(Node* y);
Node* leftRotate(Node* x);
int getHeight(Node* node);
void updateHeight(Node* node);
int calculateBalanceFactor(Node* node);
Node* findMinNode(Node* node);
Node* balanceTree(Node* node);

// Utility functions
int getHeight(Node* node) {
    if (node == nullptr)
        return 0;
    return node->height;
}
```

```
void updateHeight(Node* node) {
    if (node != nullptr) {
        node->height = 1 + max(getHeight(node->left), getHeight(node->right));
    }
}

int calculateBalanceFactor(Node* node) {
    if (node == nullptr)
        return 0;
    return getHeight(node->left) - getHeight(node->right);
}

// Global flag to track if a rotation occurred during an operation
atomic<bool> rotation_occurred(false);

// Right rotation
Node* rightRotate(Node* y) {
    cout << "Performing Right Rotation at node: " << y->keyword << endl;
    Node* x = y->left;
    Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    updateHeight(y);
    updateHeight(x);

    rotation_occurred = true;
    return x;
}

// Left rotation
Node* leftRotate(Node* x) {
    cout << "Performing Left Rotation at node: " << x->keyword << endl;
    Node* y = x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;

    updateHeight(x);
    updateHeight(y);

    rotation_occurred = true;
    return y;
}

Node* findMinNode(Node* node) {
    while (node->left != nullptr) {
        node = node->left;
    }
    return node;
}
```

```
// Helper function to balance the tree at a given node
Node* balanceTree(Node* node) {
    updateHeight(node);
    int balance = calculateBalanceFactor(node);
    Node* original_node = node;

    // Left Heavy
    if (balance > 1) {
        // Left-Right Case
        if (calculateBalanceFactor(node->left) < 0) {
            cout << "Tree before LR rotation (left part) at " << node->left->keyword
<< ":" << endl;
            printLevelOrder(original_node);
            node->left = leftRotate(node->left);
            updateHeight(node);
            cout << "Tree after LR rotation (left part done), before right rotation at
" << node->keyword << ":" << endl;
            printLevelOrder(node);
            node = rightRotate(node);
            cout << "Tree after LR rotation (final):" << endl;
            printLevelOrder(node);
        }
        // Left-Left Case
        else {
            cout << "Tree before LL rotation at " << node->keyword << ":" << endl;
            printLevelOrder(original_node);
            node = rightRotate(node);
            cout << "Tree after LL rotation:" << endl;
            printLevelOrder(node);
        }
    }
    // Right Heavy
    else if (balance < -1) {
        // Right-Left Case
        if (calculateBalanceFactor(node->right) > 0) {
            cout << "Tree before RL rotation (right part) at " << node->right->keyword
<< ":" << endl;
            printLevelOrder(original_node);
            node->right = rightRotate(node->right);
            updateHeight(node);
            cout << "Tree after RL rotation (right part done), before left rotation at
" << node->keyword << ":" << endl;
            printLevelOrder(node);
            node = leftRotate(node);
            cout << "Tree after RL rotation (final):" << endl;
            printLevelOrder(node);
        }
        // Right-Right Case
        else {
            cout << "Tree before RR rotation at " << node->keyword << ":" << endl;
            printLevelOrder(original_node);
            node = leftRotate(node);
            cout << "Tree after RR rotation:" << endl;
            printLevelOrder(node);
        }
    }

    return node;
}

Node* insert(Node* root, string keyword, string meaning) {
    if (root == nullptr)
        return new Node(keyword, meaning);

    if (keyword < root->keyword)
        root->left = insert(root->left, keyword, meaning);
    else if (keyword > root->keyword)
        root->right = insert(root->right, keyword, meaning);
    else
        return root;

    return balanceTree(root);
}

void printInOrder(Node* root) {
    if (root != nullptr) {
        printInOrder(root->left);
        cout << root->keyword << " (" << calculateBalanceFactor(root) << ") " << root-
>meaning << endl;
        printInOrder(root->right);
    }
}

void printReverseInOrder(Node* root) {
    if (root != nullptr) {
        printReverseInOrder(root->right);
        cout << root->keyword << " (" << calculateBalanceFactor(root) << ") " << root-
>meaning << endl;
        printReverseInOrder(root->left);
    }
}

// Function to print the tree level by level
void printLevelOrder(Node* root) {
    if (root == nullptr) {
        cout << "Tree is empty." << endl;
        return;
    }

    queue<Node*> q;
    q.push(root);

    cout << "Level Order Traversal:" << endl;
    while (!q.empty()) {
        int levelSize = q.size();
        for (int i = 0; i < levelSize; ++i) {
            Node* current = q.front();
            q.pop();

            if (current != nullptr) {
                cout << current->keyword << "(" << calculateBalanceFactor(current) <<
") ";
                q.push(current->left);
                q.push(current->right);
            } else {
                cout << "null ";
            }
        }
        cout << endl;
    }
    cout << "------------------------" << endl;
}

Node* deleteKeyword(Node* root, string keyword) {
    if (root == nullptr)
        return root;

    if (keyword < root->keyword)
        root->left = deleteKeyword(root->left, keyword);
    else if (keyword > root->keyword)
        root->right = deleteKeyword(root->right, keyword);
    else {
        if (root->left == nullptr || root->right == nullptr) {
            Node* temp = root->left ? root->left : root->right;
            if (temp == nullptr) {
                temp = root;
                root = nullptr;
            } else {
                *root = *temp;
            }
            delete temp;
        } else {
            Node* temp = findMinNode(root->right);
            root->keyword = temp->keyword;
            root->meaning = temp->meaning;
            root->right = deleteKeyword(root->right, temp->keyword);
        }
    }

    if (root == nullptr)
        return root;

    return balanceTree(root);
}

string search(Node* root, string keyword, int& comparisons) {
    Node* current = root;
    comparisons = 0;
    while (current != nullptr) {
        comparisons++; // Count this comparison
        if (keyword == current->keyword) {
            return current->meaning;
        } else if (keyword < current->keyword) {
            current = current->left;
        } else {
            current = current->right;
        }
    }
    return "Keyword not found.";
}

void updateMeaning(Node* root, string keyword, string newMeaning) {
    Node* current = root;
    while (current != nullptr) {
        if (keyword == current->keyword) {
            current->meaning = newMeaning;
            cout << "Meaning updated." << endl;
            return;
        } else if (keyword < current->keyword) {
            current = current->left;
        } else {
            current = current->right;
        }
    }
    cout << "Keyword not found." << endl;
}

int main() {
    Node* root = nullptr;
    string keyword, meaning, newMeaning;
    int comparisons = 0; // Initialize here
    char addMore = 'y';

    while (true) {
        cout << "\n==== AVL Dictionary Menu ====\n";
        cout << "\n1. Add new keyword and meaning\n";
        cout << "2. Print all entries in ascending order\n";
        cout << "3. Print all entries in descending order\n";
        cout << "4. Delete a keyword from dictionary\n";
        cout << "5. Search a keyword\n";
        cout << "6. Update meaning of a keyword\n";
        cout << "7. Get Tree Height\n";
        cout << "8. Display Tree Level Order\n";
        cout << "9. Exit program\n";
        cout << "Enter your choice: ";

        int choice;
        cin >> choice;

        switch (choice) {
            case 1:
                addMore = 'y';
                while (addMore == 'y' || addMore == 'Y') {
                    cout << "Enter keyword: ";
                    cin >> keyword;
                    cout << "Enter meaning: ";
                    cin.ignore();
                    getline(cin, meaning);

                    rotation_occurred = false;
                    Node* old_root = root; // Keep track if it's the very first node
                    int dummy_comps_before;
                    string meaning_before = search(root, keyword, dummy_comps_before);
// Check if exists before

                    root = insert(root, keyword, meaning);
                    printLevelOrder(root);

                    // Provide feedback based on rotation or if it was a new node vs
duplicate
                    if (rotation_occurred) {
                        cout << "Keyword '" << keyword << "' inserted/processed
(rotation occurred)." << endl;
                    } else {
                        // Check if it was actually inserted or was a duplicate
                        if (meaning_before == "Keyword not found.") {
                             cout << "Keyword '" << keyword << "' inserted. Tree
remains balanced." << endl;
                        } else {
                             cout << "Keyword '" << keyword << "' already exists. No
changes made." << endl;
                        }
                    }
                    cout << "Add more keywords? (y/n): ";
                    cin >> addMore;
                }
                break;
            case 2:
                if (root == nullptr)
                    cout << "No entries to display." << endl;
                else
                    printInOrder(root);
                break;
            case 3:
                if (root == nullptr)
                    cout << "No entries to display." << endl;
                else
                    printReverseInOrder(root);
                break;
            case 4:
                if (root == nullptr)
                    cout << "No entries to delete." << endl;
                else {
                    cout << "Enter keyword to delete: ";
                    cin >> keyword;
                    int dummy_comparisons;
                    string search_result = search(root, keyword, dummy_comparisons);

                    if (search_result != "Keyword not found.") {
                        rotation_occurred = false;
                        root = deleteKeyword(root, keyword);
                        cout << "Keyword '" << keyword << "' deleted." << endl;
                        if (!rotation_occurred) {
                            cout << "Tree remains balanced." << endl;
                        }
                    } else {
                        cout << "Keyword not found, cannot delete." << endl;
                    }
                }
                break;
            case 5:
                if (root == nullptr)
                    cout << "Dictionary is empty." << endl;
                else {
                    cout << "Enter keyword to search: ";
                    cin >> keyword;
                    comparisons = 0;
                    string result = search(root, keyword, comparisons);
                    cout << result << " (Comparisons: " << comparisons << ")" << endl;
                }
                break;
            case 6:
                if (root == nullptr)
                    cout << "Dictionary is empty." << endl;
                else {
                    cout << "Enter keyword to update: ";
                    cin >> keyword;
                    cout << "Enter new meaning: ";
                    cin.ignore();
                    getline(cin, newMeaning);
                    updateMeaning(root, keyword, newMeaning);
                }
                break;
            case 7:
                cout << "Tree Height: " << getHeight(root) << endl;
                break;
            case 8:
                cout << "Current Tree State:" << endl;
                printLevelOrder(root);
                break;
            case 9:
                return 0;
            default:
                cout << "Invalid choice. Please choose a valid option." << endl;
        }
    }
}
```

Output

```
PS G:\My Drive\DSA\DSAL> g++ ./avltree.cpp -o build/avltree
PS G:\My Drive\DSA\DSAL> build/avltree

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
```

```
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 1
Enter keyword: stack
Enter meaning: lifo
Level Order Traversal:
stack(0)
null null
------------------------
Keyword 'stack' inserted. Tree remains balanced.
Add more keywords? (y/n): y
Enter keyword: queue
Enter meaning: fifo
Level Order Traversal:
stack(1)
queue(0) null
null null
------------------------
Keyword 'queue' inserted. Tree remains balanced.
Add more keywords? (y/n): y
Enter keyword: array
Enter meaning: linear ds
Tree before LL rotation at stack:
Level Order Traversal:
stack(2)
queue(1) null
array(0) null
null null
------------------------
Performing Right Rotation at node: stack
Tree after LL rotation:
Level Order Traversal:
queue(0)
array(0) stack(0)
null null null null
------------------------
Level Order Traversal:
queue(0)
array(0) stack(0)
null null null null
------------------------
```

```
Keyword 'array' inserted/processed (rotation occurred).
Add more keywords? (y/n): y
Enter keyword: tree
Enter meaning: hierarchical ds
Level Order Traversal:
queue(-1)
array(0) stack(-1)
null null null tree(0)
null null
------------------------
Keyword 'tree' inserted. Tree remains balanced.
Add more keywords? (y/n): y
Enter keyword: avl
Enter meaning: self balancing
Level Order Traversal:
queue(0)
array(-1) stack(-1)
null avl(0) null tree(0)
null null null null
------------------------
Keyword 'avl' inserted. Tree remains balanced.
Add more keywords? (y/n): y
Enter keyword: algorithm
Enter meaning: step by step instructions
Level Order Traversal:
queue(0)
array(0) stack(-1)
algorithm(0) avl(0) null tree(0)
null null null null null null
------------------------
Keyword 'algorithm' inserted. Tree remains balanced.
Add more keywords? (y/n): n

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 1
Enter keyword: apple
Enter meaning: fruit
Level Order Traversal:
queue(1)
array(1) stack(-1)
algorithm(-1) avl(0) null tree(0)
null apple(0) null null null null
null null
------------------------
Keyword 'apple' inserted. Tree remains balanced.
Add more keywords? (y/n): n

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 2
algorithm (-1) step by step instructions
apple (0) fruit
array (1) linear ds
avl (0) self balancing
queue (1) fifo
stack (-1) lifo
tree (0) hierarchical ds

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 3
tree (0) hierarchical ds
```

```
stack (-1) lifo
queue (1) fifo
avl (0) self balancing
array (1) linear ds
apple (0) fruit
algorithm (-1) step by step instructions

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 4
Enter keyword to delete: stack
Tree before LL rotation at queue:
Level Order Traversal:
queue(2)
array(1) tree(0)
algorithm(-1) avl(0) null null
null apple(0) null null
null null
------------------------
Performing Right Rotation at node: queue
Tree after LL rotation:
Level Order Traversal:
array(0)
algorithm(-1) queue(0)
null apple(0) avl(0) tree(0)
null null null null null null
------------------------
Keyword 'stack' deleted.

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
```

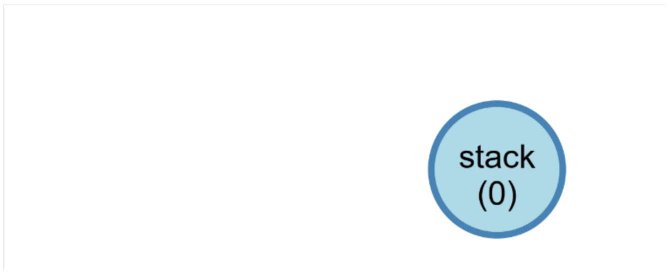
```
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 8
Current Tree State:
Level Order Traversal:
array(0)
algorithm(-1) queue(0)
null apple(0) avl(0) tree(0)
null null null null null null
------------------------

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 5
Enter keyword to search: algorithm
step by step instructions (Comparisons: 2)

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 6
Enter keyword to update: queue
Enter new meaning: linear fifo
Meaning updated.

==== AVL Dictionary Menu ====
1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 7
Tree Height: 3

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 8
Current Tree State:
Level Order Traversal:
array(0)
algorithm(-1) queue(0)
null apple(0) avl(0) tree(0)
null null null null null null
------------------------

==== AVL Dictionary Menu ====

1. Add new keyword and meaning
2. Print all entries in ascending order
3. Print all entries in descending order
4. Delete a keyword from dictionary
5. Search a keyword
6. Update meaning of a keyword
7. Get Tree Height
8. Display Tree Level Order
9. Exit program
Enter your choice: 9
```

```
PS G:\My Drive\DSA\DSAL>
```

AVL Tree Operations Visualization

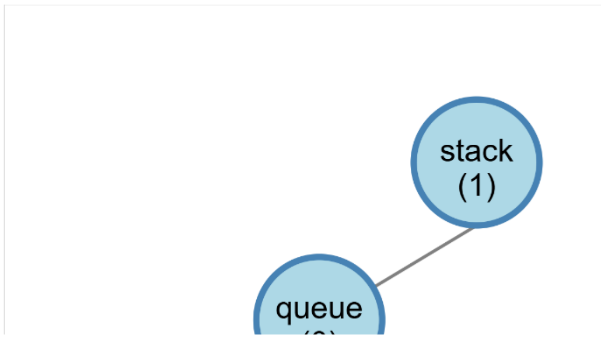
1. After inserting 'stack'

```
Level Order Traversal:
stack(0)
null null
```



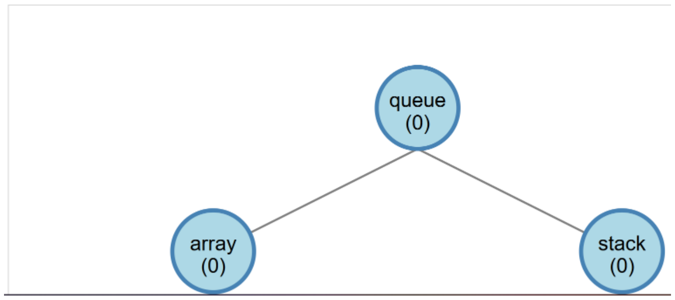
2. After inserting 'queue'

```
Level Order Traversal:
stack(1)
queue(0) null
null null
```



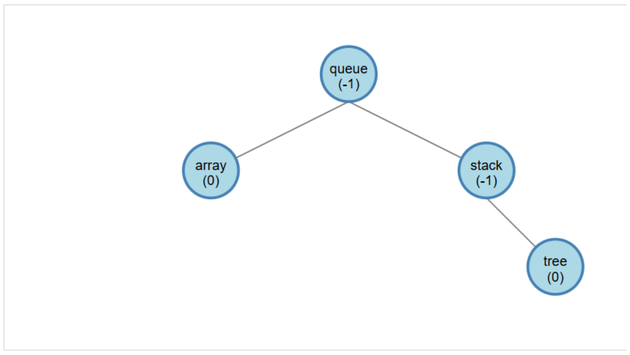
3. After inserting 'array' (LL Rotation)

```
Level Order Traversal:
queue(0)
array(0) stack(0)
null null null null
```



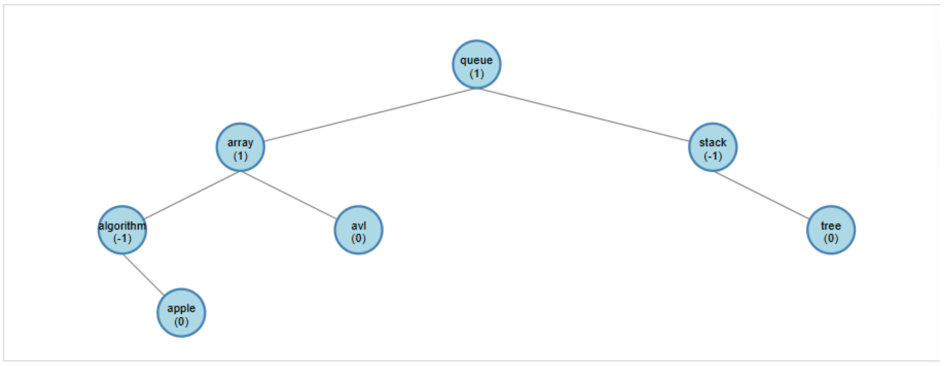
4. After inserting 'tree'

```
Level Order Traversal:
queue(-1)
array(0) stack(-1)
null null null tree(0)
null null
```



5. After inserting 'apple'

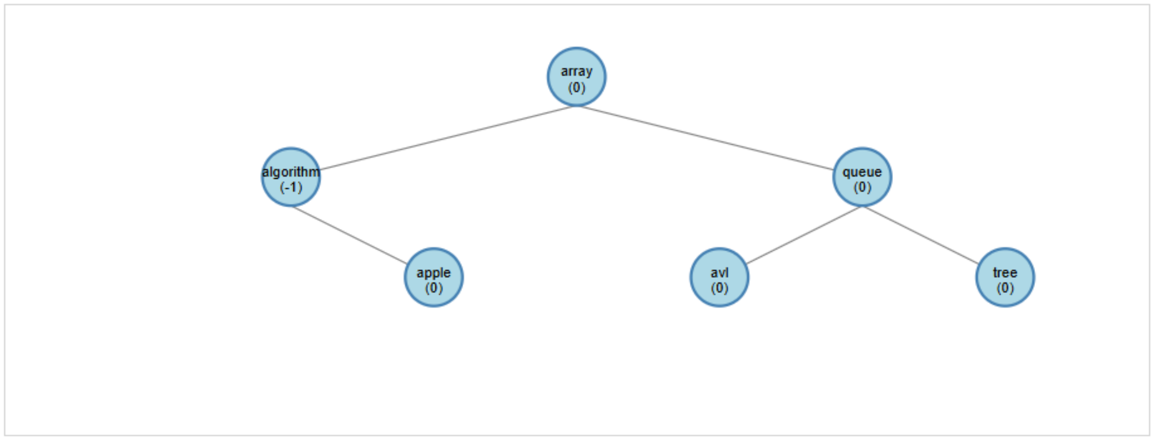
```
Level Order Traversal:
queue(1)
array(1) stack(-1)
algorithm(-1) avl(0) null tree(0)
null apple(0) null null null null
null null
```



6. After deleting 'stack' (LL Rotation)

```
Level Order Traversal:
array(0)
algorithm(-1) queue(0)
null apple(0) avl(0) tree(0)
null null null null null null
```

Final AVL Tree



Experiment No. 10

Consider a scenario for hospital to cater services to different kinds of patients as Serious (top priority), b) non-serious (medium priority), c) General Check-up (Least priority). Implement the priority queue to cater services to the patients.

Program:

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

struct Patient {
    string name;
    int priority;
    Patient* next;
};

class PriorityQueue {
private:
    Patient* head;
public:
    PriorityQueue()
    {
        head = nullptr;
    }

    void enqueue(string name, int priority) {
        Patient* newPatient = new Patient{name, priority, nullptr};
        if (!head || priority < head->priority) {
            newPatient->next = head;
            head = newPatient;
        } else {
            Patient* temp = head;
            while (temp->next && temp->next->priority <= priority) {
                temp = temp->next;
            }
            newPatient->next = temp->next;
            temp->next = newPatient;
        }
    }

    void dequeue() {
        if (!head) {
            cout << "Queue is empty. No patients to serve.\n";
            return;
        }
        Patient* temp = head;
        head = head->next;
        cout << "Serving patient: " << temp->name << " (Priority: " << temp->priority
<< ")\n";
        delete temp;
    }

    void display() {
        if (!head) {
            cout << "No patients in the queue.\n";
            return;
        }
        Patient* temp = head;
        cout << "Patients in the queue:\n";
        while (temp) {
            cout << temp->name << " (Priority: " << temp->priority << ")\n";
            temp = temp->next;
        }
    }

    ~PriorityQueue() {
        while (head) {
            Patient* temp = head;
            head = head->next;
            delete temp;
        }
    }
};
```

```
int main() {
    PriorityQueue pq;
    int choice;
    do {
        cout << "\nHospital Priority Queue System\n";
        cout << "1. Add Patient\n";
        cout << "2. Serve Patient\n";
        cout << "3. Display Patients\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore(); // Clear the input buffer

        switch (choice) {
            case 1: {
                string name;
                int priority;
                cout << "Enter patient name: ";
                getline(cin, name); // Use getline to allow spaces in the name
                cout << "Enter priority (1: Serious, 2: Non-serious, 3: General
Check-up): ";
                cin >> priority;
                pq.enqueue(name, priority);
                break;
            }
            case 2:
                pq.dequeue();
                break;
            case 3:
                pq.display();
                break;
            case 4:
                cout << "Exiting the system. Goodbye!\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 4);

    return 0;
}
```

Output:

```
PS G:\My Drive\DSA\DSAL> g++ .\prioqueue.cpp -o build/prio
PS G:\My Drive\DSA\DSAL> build/prio

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
```

```
4. Exit
Enter your choice: 1
Enter patient name:
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 1

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Someone
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 3

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Random Person
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 2

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Demo
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 1

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
    (Priority: 1)
Demo (Priority: 1)
Random Person  (Priority: 2)
Someone (Priority: 3)

Hospital Priority Queue System
```

```
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 2
Serving patient:     (Priority: 1)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 2
Serving patient: Demo (Priority: 1)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
Random Person  (Priority: 2)
Someone (Priority: 3)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 1
Enter patient name: Middle
Enter priority (1: Serious, 2: Non-serious, 3: General Check-up): 2

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
Random Person  (Priority: 2)
Middle (Priority: 2)
Someone (Priority: 3)
```

```
Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 2
Serving patient: Random Person  (Priority: 2)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 3
Patients in the queue:
Middle (Priority: 2)
Someone (Priority: 3)

Hospital Priority Queue System
1. Add Patient
2. Serve Patient
3. Display Patients
4. Exit
Enter your choice: 4
Exiting the system. Goodbye!
PS G:\My Drive\DSA\DSAL>
```