Assignment 9

Problem Statement:

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any 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 Height balance tree and find the complexity for finding a keyword

Source Code:

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

struct Node {
    string keyword;
    string meaning;
    Node* left;
    Node* right;
    int height;

    Node(string k, string m) {
        keyword = k;
        meaning = m;
        left = right = nullptr;
        height = 1;
    }
};
```

```
int height(Node* n) {
    return n ? n->height : 0;
}

int getBalance(Node* n) {
    return n ? height(n->left) - height(n->right) : 0;
}

Node* rotateRight(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

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

    y->height = 1 + max(height(y->left), height(y->right));
    x->height = 1 + max(height(x->left), height(x->right));

    return x;
}

Node* rotateLeft(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

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

    x->height = 1 + max(height(x->left), height(x->right));
```

```
    y->height = 1 + max(height(y->left), height(y->right));

    return y;
}

// Insert Node
Node* insert(Node* root, string key, string meaning) {
    if (!root)
        return new Node(key, meaning);

    if (key < root->keyword)
        root->left = insert(root->left, key, meaning);
    else if (key > root->keyword)
        root->right = insert(root->right, key, meaning);
    else {
        cout << "Keyword already exists. Updating meaning.\n";
        root->meaning = meaning;
        return root;
    }

    root->height = 1 + max(height(root->left), height(root->right));
    int balance = getBalance(root);

    // Balancing
    if (balance > 1 && key < root->left->keyword)
        return rotateRight(root);
    if (balance < -1 && key > root->right->keyword)
        return rotateLeft(root);
    if (balance > 1 && key > root->left->keyword) {
```

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

    return root;
}

// Find Minimum
Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left)
        current = current->left;
    return current;
}

// Delete Node
Node* deleteNode(Node* root, string key) {
    if (!root)
        return root;

    if (key < root->keyword)
        root->left = deleteNode(root->left, key);
    else if (key > root->keyword)
        root->right = deleteNode(root->right, key);
    else {
```

```
        if (!root->left || !root->right) {
            Node* temp = root->left ? root->left : root->right;
            delete root;
            return temp;
        }
        Node* temp = minValueNode(root->right);
        root->keyword = temp->keyword;
        root->meaning = temp->meaning;
        root->right = deleteNode(root->right, temp->keyword);
    }

    root->height = 1 + max(height(root->left), height(root->right));
    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)
        return rotateRight(root);
    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = rotateLeft(root->left);
        return rotateRight(root);
    }
    if (balance < -1 && getBalance(root->right) <= 0)
        return rotateLeft(root);
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rotateRight(root->right);
        return rotateLeft(root);
    }

    return root;
}
```

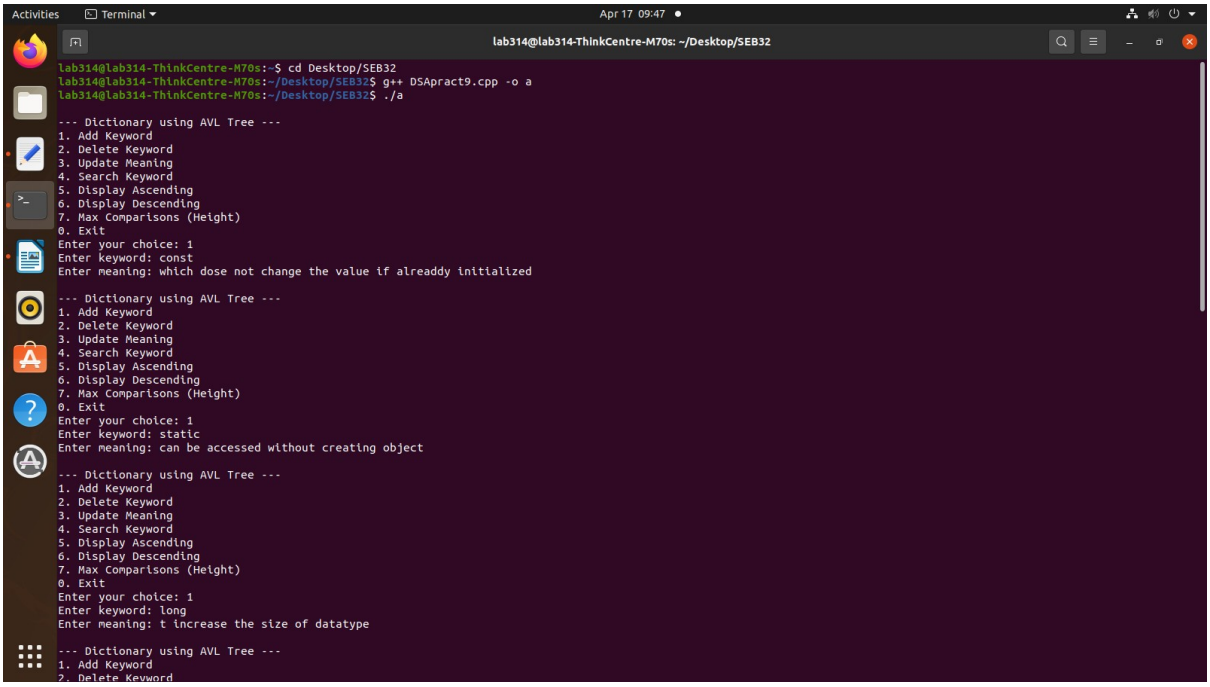
```
// Search keyword
bool search(Node* root, string key, int& comparisons) {
    while (root) {
        comparisons++;
        if (key == root->keyword) {
            cout << "Meaning: " << root->meaning << endl;
            return true;
        }
        if (key < root->keyword)
            root = root->left;
        else
            root = root->right;
    }
    return false;
}

// Display ascending
void displayAscending(Node* root) {
    if (root) {
        displayAscending(root->left);
        cout << root->keyword << ": " << root->meaning << endl;
        displayAscending(root->right);
    }
}

// Display descending
void displayDescending(Node* root) {
    if (root) {
```

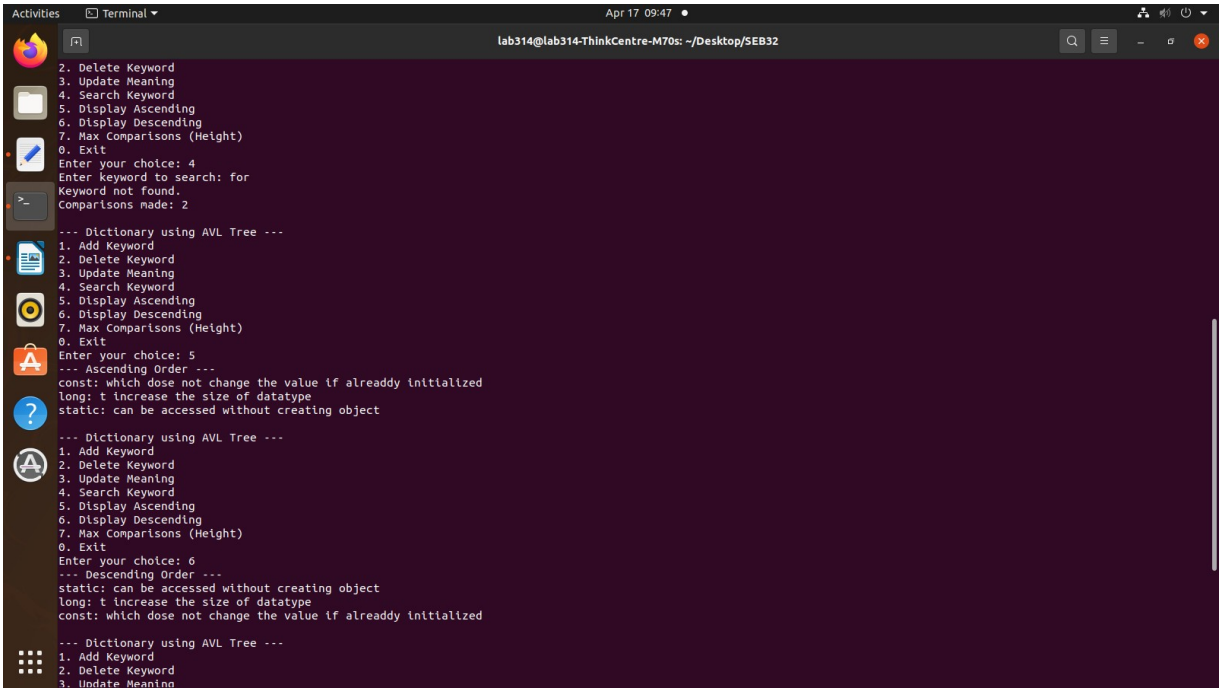
```
        displayDescending(root->right);
        cout << root->keyword << ": " << root->meaning << endl;
        displayDescending(root->left);
    }
}

// Update meaning
bool updateMeaning(Node* root, string key, string newMeaning) {
    while (root) {
        if (key == root->keyword) {
            root->meaning = newMeaning;
            return true;
        }
        if (key < root->keyword)
            root = root->left;
        else
            root = root->right;
    }
    return false;
}

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

    do {
        cout << "\n--- Dictionary using AVL Tree ---\n";
        cout << "1. Add Keyword\n2. Delete Keyword\n3. Update Meaning\n4. Search
Keyword\n";
```

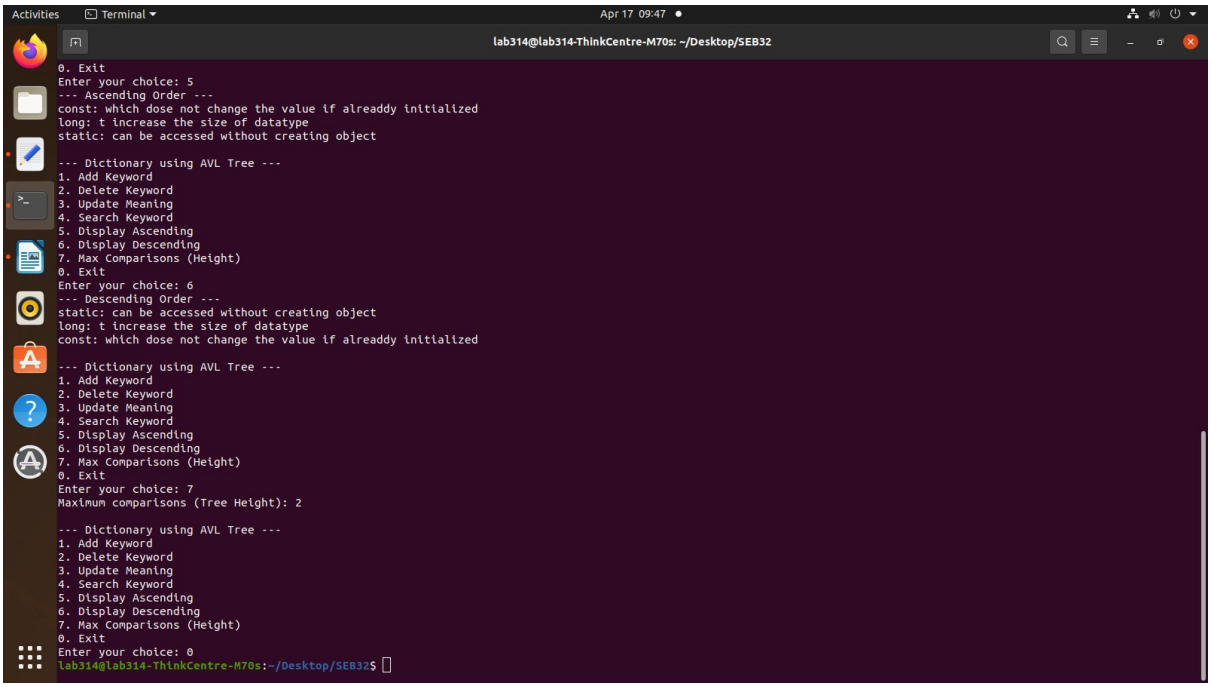
```
    cout << "5. Display Ascending\n6. Display Descending\n7. Max Comparisons (Height)\
n0. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch(choice) {
      case 1:
        cout << "Enter keyword: "; cin >> key;
        cout << "Enter meaning: "; cin.ignore(); getline(cin, meaning);
        root = insert(root, key, meaning);
        break;
      case 2:
        cout << "Enter keyword to delete: "; cin >> key;
        root = deleteNode(root, key);
        break;
      case 3:
        cout << "Enter keyword to update: "; cin >> key;
        cout << "Enter new meaning: "; cin.ignore(); getline(cin, meaning);
        if (updateMeaning(root, key, meaning))
          cout << "Meaning updated.\n";
        else
          cout << "Keyword not found.\n";
        break;
      case 4: {
        int comparisons = 0;
        cout << "Enter keyword to search: "; cin >> key;
        if (!search(root, key, comparisons))
          cout << "Keyword not found.\n";
        cout << "Comparisons made: " << comparisons << endl;
        break;
```

```
            }
            case 5:
                cout << "--- Ascending Order ---\n";
                displayAscending(root);
                break;
            case 6:
                cout << "--- Descending Order ---\n";
                displayDescending(root);
                break;
            case 7:
                cout << "Maximum comparisons (Tree Height): " << height(root) << endl;
                break;
        }
    } while(choice != 0);

    return 0;
}
```

Output:




Activities
Terminal
Apr 17 09:47
lab314@lab314-ThinkCentre-M70s: ~/Desktop/SEB32
2. Delete Keyword
3. Update Meaning
4. Search Keyword
5. Display Ascending
6. Display Descending
7. Max Comparisons (Height)
0. Exit
Enter your choice: 4
Enter keyword to search: for
Keyword not found.
Comparisons made: 2
--- Dictionary using AVL Tree ---
1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Search Keyword
5. Display Ascending
6. Display Descending
7. Max Comparisons (Height)
0. Exit
Enter your choice: 5
--- Ascending Order ---
const: which dose not change the value if alreaddy initialized
long: t increase the size of datatype
static: can be accessed without creating object
--- Dictionary using AVL Tree ---
1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Search Keyword
5. Display Ascending
6. Display Descending
7. Max Comparisons (Height)
0. Exit
Enter your choice: 6
--- Descending Order ---
static: can be accessed without creating object
long: t increase the size of datatype
const: which dose not change the value if alreaddy initialized
--- Dictionary using AVL Tree ---
1. Add Keyword
2. Delete Keyword


Activities
Terminal
Apr 17 09:47
lab314@lab314-ThinkCentre-M70s: ~/Desktop/SEB32
0. Exit
Enter your choice: 5
--- Ascending Order ---
const: which dose not change the value if alreaddy initialized
long: t increase the size of datatype
static: can be accessed without creating object
--- Dictionary using AVL Tree ---
1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Search Keyword
5. Display Ascending
6. Display Descending
7. Max Comparisons (Height)
0. Exit
Enter your choice: 6
--- Descending Order ---
static: can be accessed without creating object
long: t increase the size of datatype
const: which dose not change the value if alreaddy initialized
--- Dictionary using AVL Tree ---
1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Search Keyword
5. Display Ascending
6. Display Descending
7. Max Comparisons (Height)
0. Exit
Enter your choice: 7
Maximum comparisons (Tree Height): 2
--- Dictionary using AVL Tree ---
1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Search Keyword
5. Display Ascending
6. Display Descending
7. Max Comparisons (Height)
0. Exit
Enter your choice: 0
lab314@lab314-ThinkCentre-M70s:~/Desktop/SEB32$