```
/*
Name : Mahesh Gaikwad
 Roll No : SA21
Assignment 9 : AVL TREE
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 a Height
balanced tree and find the complexity for finding a keyword.
*/
#include <iostream>
#include <algorithm> #include <cstdlib>
using namespace std; class node // Node
Structure for AVL tree
{
public: string word,
meaning;
int ht; node *left,
*right;
};
class AVL
{
public: node
*root;
AVL()
{
root = NULL;
}
node *insert(node *, string, string);
node *deleteNode(node *, string);
void preorder(node *); void
inorder(node *); node
*RotateRight(node *); node
*RotateLeft(node *); node
*RR(node *); node *LL(node *);

node *LR(node *); node
*RL(node *); int
height(node *); int BF(node
*); void search(node *,
string); void modify(node *,
string);
};
int AVL::height(node *temp) // Used to calculate Hight of subtree
{
int lh, rh; if (temp == NULL) return 0; lh = (temp-
>left == NULL) ? 0 : 1 + temp->left->ht; rh = (temp-
>right == NULL) ? 0 : 1 + temp->right->ht; return
max(lh, rh);
}
int AVL::BF(node *temp) // Used to calculate Balance factor of each node
{
if (temp == NULL) return 0; return height(temp-
>left) - height(temp->right);
}
node *AVL::RotateRight(node *parent) // Rotate right general function need to
make connection while rotating tree to ward right
{
node *temp; temp = new
node; temp = parent->left;
```

```
parent->left = temp->right;

temp->right = parent;

parent->ht = height(parent);

temp->ht = height(temp);

return temp;

}

node *AVL::RotateLeft(node *parent) // Rotate left general function need to

make connection while rotating tree to ward left

{

node *temp; temp =

new node; temp =

parent->right;


parent->right = temp->left;

temp->left = parent; parent->ht

= height(parent); temp->ht =

height(temp); return temp;

}

node *AVL::RR(node *T) // Rotate towards right

{

T = RotateLeft(T); return

T;

}

node *AVL::LL(node *T) // Rotate towards left

{

T = RotateRight(T); return

T;

}

node *AVL::LR(node *T) // Rotate towards left and then right

{
```

```
T->left = RotateLeft(T->left);
T = RotateRight(T); return
T;
}
node *AVL::RL(node *T) // Rotate towards right and then left
{
T->right = RotateRight(T->right);
T = RotateLeft(T); return T;
}
node *AVL::insert(node *temp, string str_w, string str_m) // Insert each node
into the tree
{
if (temp == NULL)
{
temp = new node; temp->word =
str_w; temp->meaning = str_m;
temp->left = temp->right = NULL;
}
else
{

if (str_w.compare(temp->word) > 0)
{
temp->right = insert(temp->right, str_w, str_m);
if (BF(temp) == -2) temp = (str_w.compare(temp->right-
>word) > 0) ? RR(temp) :

RL(temp);
}
else
{
```

```
temp->left = insert(temp->left, str_w, str_m); if (BF(temp)
== 2) temp = (str_w.compare(temp->left->word) < 0) ?
LL(temp) :

LR(temp);
}
}
temp->ht = height(temp); return
temp;
}
node *AVL::deleteNode(node *temp, string str_w) // Delete a node from avl
tree
{
if (temp == NULL) return
NULL;
else
{
if (str_w.compare(temp->word) > 0)
{
temp->right = deleteNode(temp->right, str_w); if
(BF(temp) == 2) temp = (BF(temp->left) >= 0) ?
LL(temp) : LR(temp);
}
else
{
if (str_w.compare(temp->word) < 0)
{
temp->left = deleteNode(temp->left, str_w); if
(BF(temp) == -2) temp = (BF(temp->right) <= 0) ?
RR(temp) : RL(temp);
}
```

```
else
 {
if (temp->right != NULL)
{
node *temp1; temp1 = temp->right; while (temp1-
>left != NULL) temp1 = temp1->left; temp->word =
temp1->word; temp->right = deleteNode(temp->right,
temp1->word); if (BF(temp) == 2) temp = (BF(temp-
>left) >= 0) ? LL(temp) :

LR(temp);
 }
else return temp-
>left;

}
}
}
temp->ht = height(temp);
return temp;
}
void AVL::preorder(node *root) // Preorder traversal of the tree
{
if (root != NULL)
{
cout << root->word << "(Bf=" << BF(root) << ") ";
preorder(root->left); preorder(root->right);
}
}
void AVL::inorder(node *root) // Inorder traversal of the tree
{
```

```
if (root != NULL)
{
inorder(root->left); cout << root->word <<
"(Bf=" << BF(root) << ") "; inorder(root->right);
}
}
void AVL::search(node *root, string str_w) // Search node in AVL Tree
 {
if (str_w.compare(root->word) < 0)
{
if (root->left == NULL) cout
<< "Word not found";
else search(root->left,
str_w);

}
else if (str_w.compare(root->word) > 0)
{
if (root->right == NULL)
cout << "Word not found";
else search(root->right,
str_w);
 }
else
{
cout << "Word: " << root->word << endl; cout
<< "Meaning: " << root->meaning << endl;
}
}
void AVL::modify(node *root, string str_w) // Modify the meaning into the
AVL tree
```

```
{
if (str_w.compare(root->word) < 0)
{
if (root->left == NULL) cout
<< "Word not found"; else
modify(root->left, str_w);
 }
else if (str_w.compare(root->word) > 0)
{
if (root->right == NULL) cout
<< "Word not found";
else modify(root->right,
str_w);  }
else
{

getline(cin,       root->meaning);
cout << "Enter new meaning: ";
getline(cin, root->meaning);
}
}
int main()
{
AVL Tree;
int ch; string
str1, str2;
cout << "\tOPERATIONS ON AVL TREE\t" << endl; while
(true)
{
cout << "\n1. Create tree" << endl;
cout << "2. Add word" << endl; cout
```

```
<< "3. Display tree" << endl; cout <<
"4. Delete word" << endl; cout << "5.
Search word" << endl; cout << "6.
Modify meaning" << endl; cout << "7.
Exit" << endl; cout << "Enter choice:
"; cin >> ch; switch (ch)
{
case 1: case
2: cout <<
"Enter
word: "; cin
>> str1;
getline(cin,
str2);
cout << "Enter meaning: "; getline(cin,
str2);
Tree.root = Tree.insert(Tree.root, str1, str2);
break; case 3:
cout << "Preorder: ";
Tree.preorder(Tree.root);
cout << endl; cout <<
"Inorder: ";
Tree.inorder(Tree.root);
cout << endl;

break; case 4: cout <<
"Enter word: "; cin >>
str1;
Tree.root = Tree.deleteNode(Tree.root, str1);
break; case 5: cout << "Enter word: "; cin >>
str1; Tree.search(Tree.root, str1); break;
case 6: cout << "Enter word: "; cin >> str1;
Tree.modify(Tree.root, str1); break;
case 7:
exit(1); break;
}
}
return 0;
}
```

OUTPUT

```
OPERATIONS ON AVL TREE
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word

6. Modify meaning
7. Exit
Enter choice: 1
Enter word: dog
Enter meaning: a type of animal
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
```

```
Enter choice: 2

Enter word: book

Enter meaning: something to read with pages
```

```
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 2
Enter word: hello
Enter meaning: greeting
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 2
Enter word: delicious
Enter meaning: tasty
1. Create tree

2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 3
Preorder: dog(Bf=1) book(Bf=0) delicious(Bf=0) hello(Bf=0)
Inorder: book(Bf=0) delicious(Bf=0) dog(Bf=1) hello(Bf=0)
```

```
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 4
Enter word: book
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 3
Preorder: dog(Bf=0) delicious(Bf=0) hello(Bf=0)
Inorder: delicious(Bf=0) dog(Bf=0) hello(Bf=0)
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 5
Enter word: hello

Word: hello
Meaning: greeting
```

```
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 2
Enter word: hazel
Enter meaning: shade
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 3
Preorder: dog(Bf=-1) delicious(Bf=0) hello(Bf=0) hazel(Bf=0)
Inorder: delicious(Bf=0) dog(Bf=-1) hazel(Bf=0) hello(Bf=0)
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 6
Enter word: dog
Enter new meaning: pet animal
1. Create tree
```

```
2. Add word

3. Display tree

4. Delete word


5. Search word

6. Modify meaning

7. Exit

Enter choice: 7
```