LAB 14

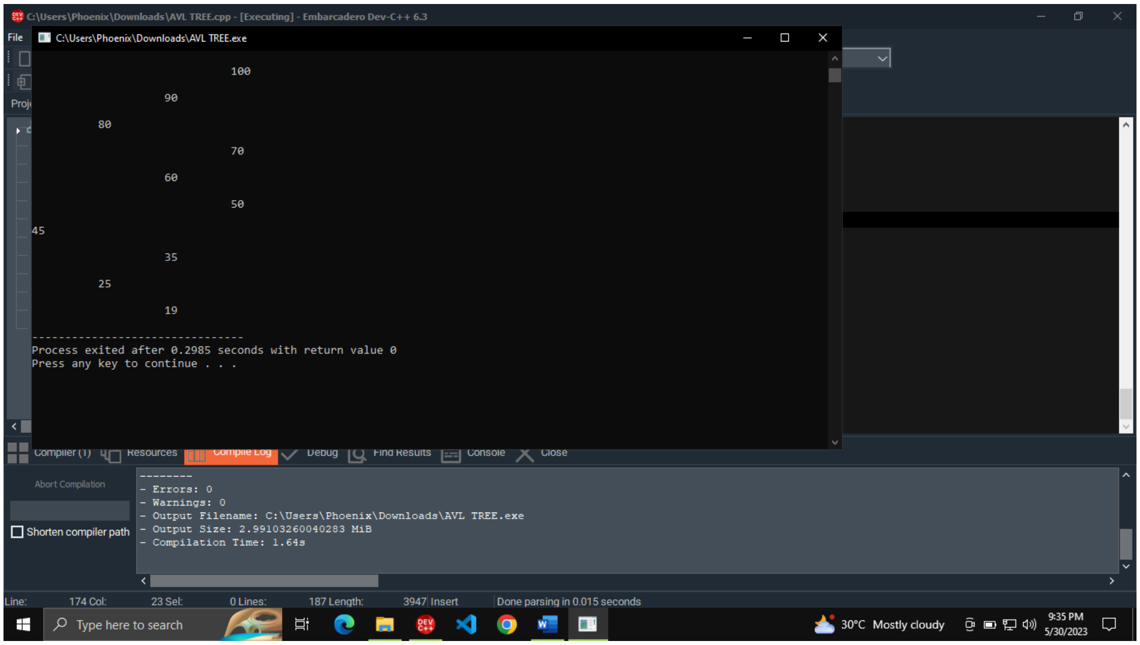
M.HUZAIFA MUSTAFA
CLASS DSA
SECTION AM
SP22_BSCS_0046

1- AVL TREE.
SOURCE CODE:

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

#define COUNT 10

class Node{
        public:
                int key;
                Node *left;
                Node *right;
                int height;
};

//left height VS right height
int max(int a, int b) {
  return (a > b) ? a : b;
}

// calculate height
int height(Node *n) {
  if (n == NULL) {
    return 0;
  } else {
    return n->height;
  }
}

//New node creation
Node* newNode(int key) {
  Node* node = new Node();
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
```

```
    return (node);
}

//right rotate
Node *rightRotate(Node *y) {
    Node *x = y->left;
    Node *T2 = x->right;

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

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

    return x;
}

//left rotate
Node *leftRotate(Node *x) {
    Node *y = x->right;
    Node *T2 = y->left;

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

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

    return y;
}

//get the balance factor
int getBalancefactor(Node *n) {
    if (n == NULL) {
        return 0;
    } else {
        return height(n->left) - height(n->right);
    }
}

Node* insert(Node* node, int key) {
              //find the correct position
    if (node == NULL) {
        return(newNode(key));
    }

    if (key < node->key) {
        node->left = insert(node->left, key);
    } else if (key > node->key) {
        node->right = insert(node->right, key);
    } else {
        return node;
    }

    //update the balance factor
    //balance the tree

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

    int balance = getBalancefactor(node);

    if (balance > 1 && key < node->left->key) {
        return rightRotate(node);
    }

    if (balance < -1 && key > node->right->key) {
        return leftRotate(node);
    }

    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

   return node;
}

// Node with minimum value
Node* minValueNode(Node* node) {
    Node* current = node;

    while (current->left != NULL) {
        current = current->left;
    }
    return current;
}


void print2DUtil(Node* root, int space)
        {
            // Base case
            if (root == NULL)
                return;

            // Increase distance between levels
            space += COUNT;

            // Process right child first
            print2DUtil(root->right, space);

            // Print current node after space
            // count
            cout << endl;
            for (int i = COUNT; i < space; i++)
                cout << " ";
            cout << root->key << "\n";

            // Process left child
            print2DUtil(root->left, space);
        }

        // Wrapper over print2DUtil()
        void print2D(Node* root)
        {
            // Pass initial space count as 0
            print2DUtil(root, 0);
        }


int main(){

    Node* root = NULL;

                            root = insert(root, 45);
                            print2D(root);
                  root = insert(root, 50);
                  print2D(root);
                  root = insert(root, 35);
                  print2D(root);
                  root = insert(root, 25);
                  print2D(root);
                  root = insert(root, 19);
                  print2D(root);
                  root = insert(root, 60);
                  print2D(root);
                  root = insert(root, 70);
                  print2D(root);
                  root = insert(root, 80);
                  print2D(root);
                  root = insert(root, 90);
                  print2D(root);
                  root = insert(root, 100);
                print2D(root);

    return 0;

}
```

PICTURE:


C:\Users\Phoenix\Downloads\AVL TREE.cpp - [Executing] - Embarcadero Dev-C++ 6.3
File
C:\Users\Phoenix\Downloads\AVL TREE.exe
Compiler (1)
Resources
Compile Log
Debug
Find Results
Console
Close
Abort Compilation
□ Shorten compiler path
Line: 174 Col: 23 Sel: 0 Lines: 187 Length: 3947 Insert
Done parsing in 0.015 seconds
Type here to search
30°C Mostly cloudy
9:35 PM
5/30/2023

```
                              100

                    90

          80

                              70

                    60

                              50

45

                    35

          25

                    19

--------------------------------
Process exited after 0.2985 seconds with return value 0
Press any key to continue . . .
```

```
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Phoenix\Downloads\AVL TREE.exe
- Output Size: 2.99103260040283 MiB
- Compilation Time: 1.64s
```