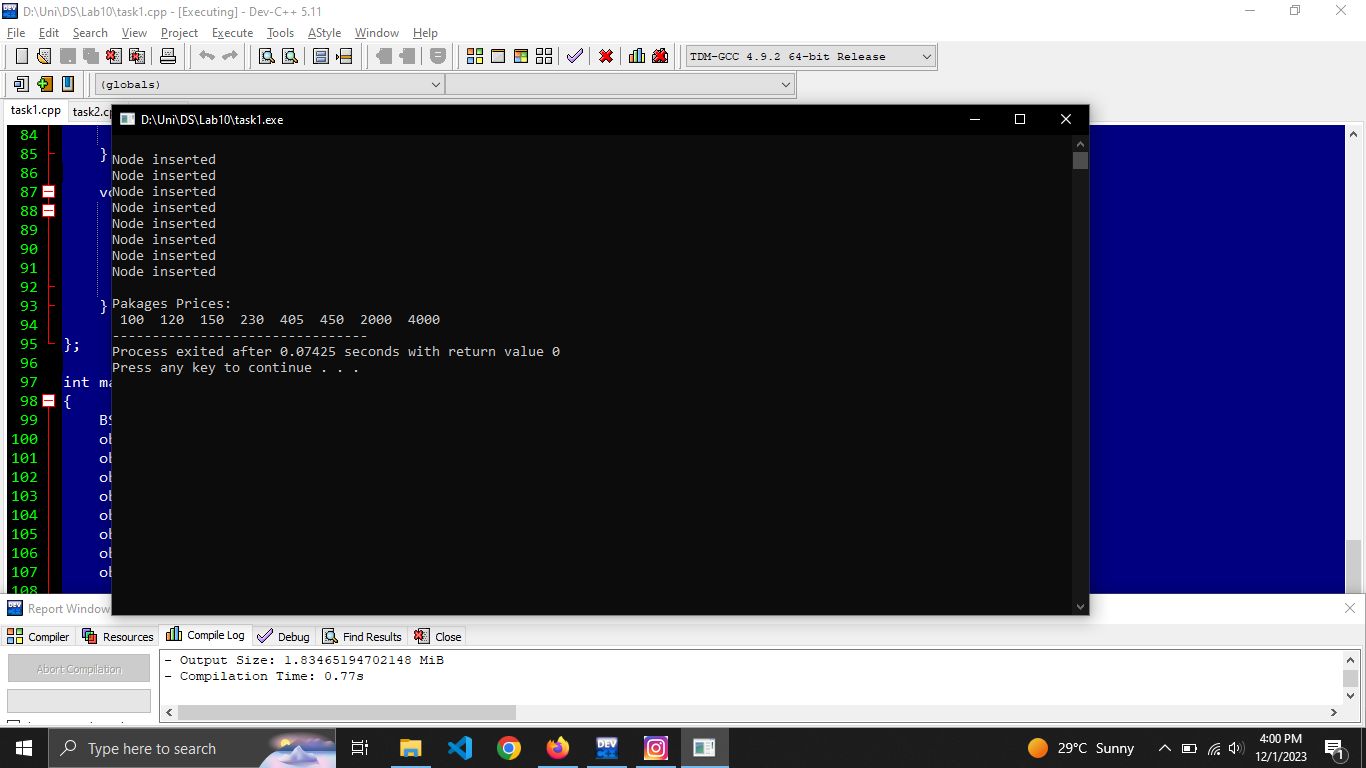
**DS LAB 10**

**K22 6007**

Task 1:



#include <iostream>

using namespace std;

//BST

class Node

{

public:

int data;

Node \*right;

Node \*left;

Node()

{

right = NULL;

left = NULL;

}

};

class BST

{

public:

Node \*root;

BST():root(NULL)

{}

void insert\_node(int data)

{

Node \*new\_node = new Node;

new\_node->data = data;

Node \*tmp = new Node;

if(root == NULL)

{

root = new\_node;

cout<<"\nNode inserted";

return;

}

else

{

tmp = root;

}

while(tmp != NULL)

{

if(tmp->data == new\_node->data)

{

return;

}

//left

if((new\_node->data < tmp->data) && (tmp->left == NULL))

{

tmp->left = new\_node;

cout<<"\nNode inserted";

return;

}

else if(new\_node->data < tmp->data)

{

tmp = tmp->left;

}

//right

if((new\_node->data > tmp->data) && (tmp->right == NULL))

{

tmp->right = new\_node;

cout<<"\nNode inserted";

return;

}

else if(data > tmp->data)

{

tmp = tmp->right;

}

}

return;

}

void display() {

displayInOrder(root);

}

void displayInOrder(Node\* node) {

if (node != NULL) {

displayInOrder(node->left);

cout<< " " << node->data << " ";

displayInOrder(node->right);

}

}

};

int main()

{

BST obj;

obj.insert\_node(2000);

obj.insert\_node(230);

obj.insert\_node(120);

obj.insert\_node(150);

obj.insert\_node(4000);

obj.insert\_node(450);

obj.insert\_node(405);

obj.insert\_node(100);

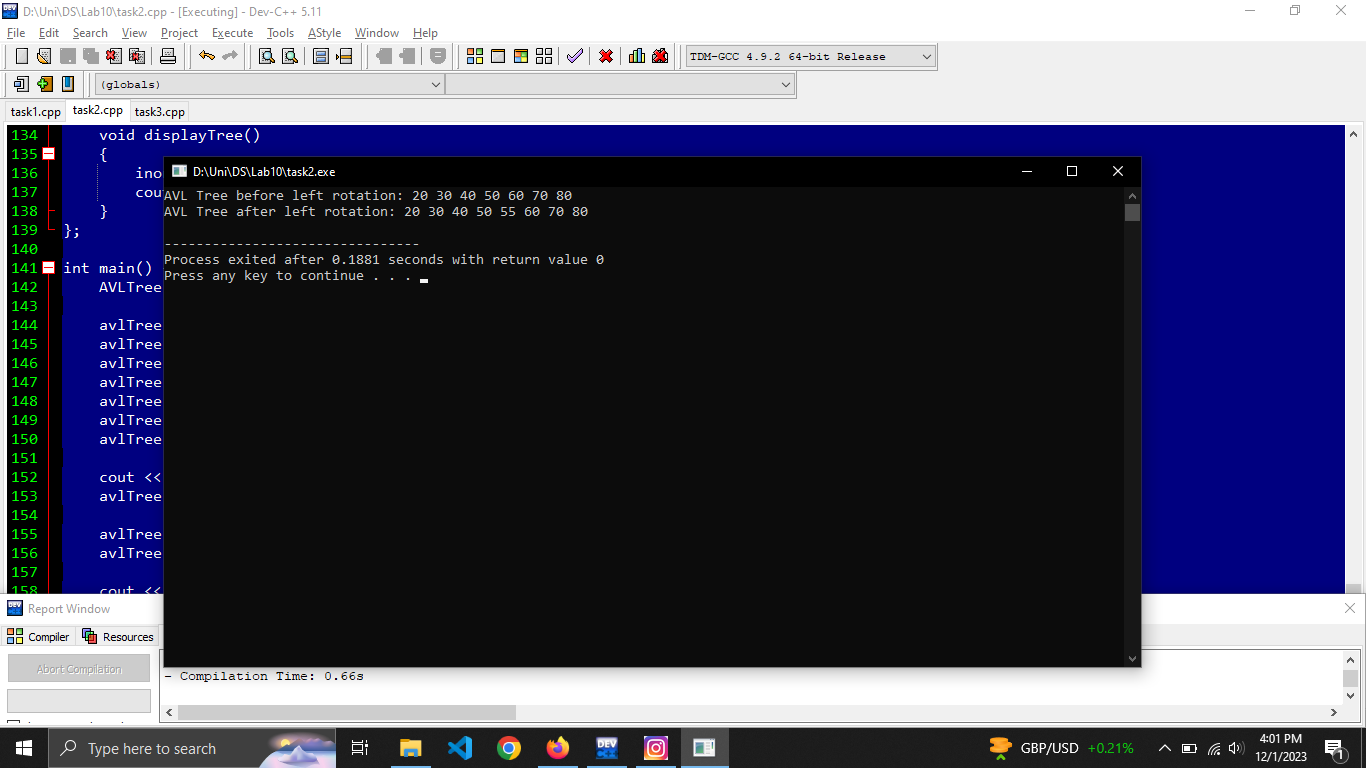
cout<<endl<<"\nPakages Prices: "<<endl;

obj.display();

return 0;

}

Task 2:



#include <iostream>

using namespace std;

class AVLNode

{

public:

int data;

AVLNode\* left;

AVLNode\* right;

int height;

AVLNode(int value) : data(value), left(NULL), right(NULL), height(1) {}

};

class AVLTree

{

private:

AVLNode\* root;

int height(AVLNode\* node)

{

if (node == NULL) return 0;

return node->height;

}

int getBalance(AVLNode\* node)

{

if (node == NULL) return 0;

return height(node->left) - height(node->right);

}

void updateHeight(AVLNode\* node)

{

if (node != NULL) {

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

}

}

AVLNode\* rightRotate(AVLNode\* y)

{

AVLNode\* x = y->left;

AVLNode\* T2 = x->right;

x->right = y;

y->left = T2;

updateHeight(y);

updateHeight(x);

return x;

}

AVLNode\* leftRotate(AVLNode\* x)

{

AVLNode\* y = x->right;

AVLNode\* T2 = y->left;

y->left = x;

x->right = T2;

updateHeight(x);

updateHeight(y);

return y;

}

AVLNode\* insert(AVLNode\* node, int value)

{

if (node == NULL) {

return new AVLNode(value);

}

if (value < node->data) {

node->left = insert(node->left, value);

} else if (value > node->data) {

node->right = insert(node->right, value);

} else {

return node;

}

updateHeight(node);

int balance = getBalance(node);

if (balance > 1 && value < node->left->data)

{

return rightRotate(node);

}

if (balance < -1 && value > node->right->data)

{

return leftRotate(node);

}

if (balance > 1 && value > node->left->data)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && value < node->right->data)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void inorderTraversal(AVLNode\* root)

{

if (root != NULL) {

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right);

}

}

public:

AVLTree() : root(NULL) {}

void insert(int value)

{

root = insert(root, value);

}

void leftRotateRoot() {

if (root != NULL && root->right != NULL)

{

root = leftRotate(root);

}

}

void displayTree()

{

inorderTraversal(root);

cout << endl;

}

};

int main() {

AVLTree avlTree;

avlTree.insert(50);

avlTree.insert(30);

avlTree.insert(70);

avlTree.insert(20);

avlTree.insert(40);

avlTree.insert(60);

avlTree.insert(80);

cout << "AVL Tree before left rotation: ";

avlTree.displayTree();

avlTree.insert(55);

avlTree.leftRotateRoot();

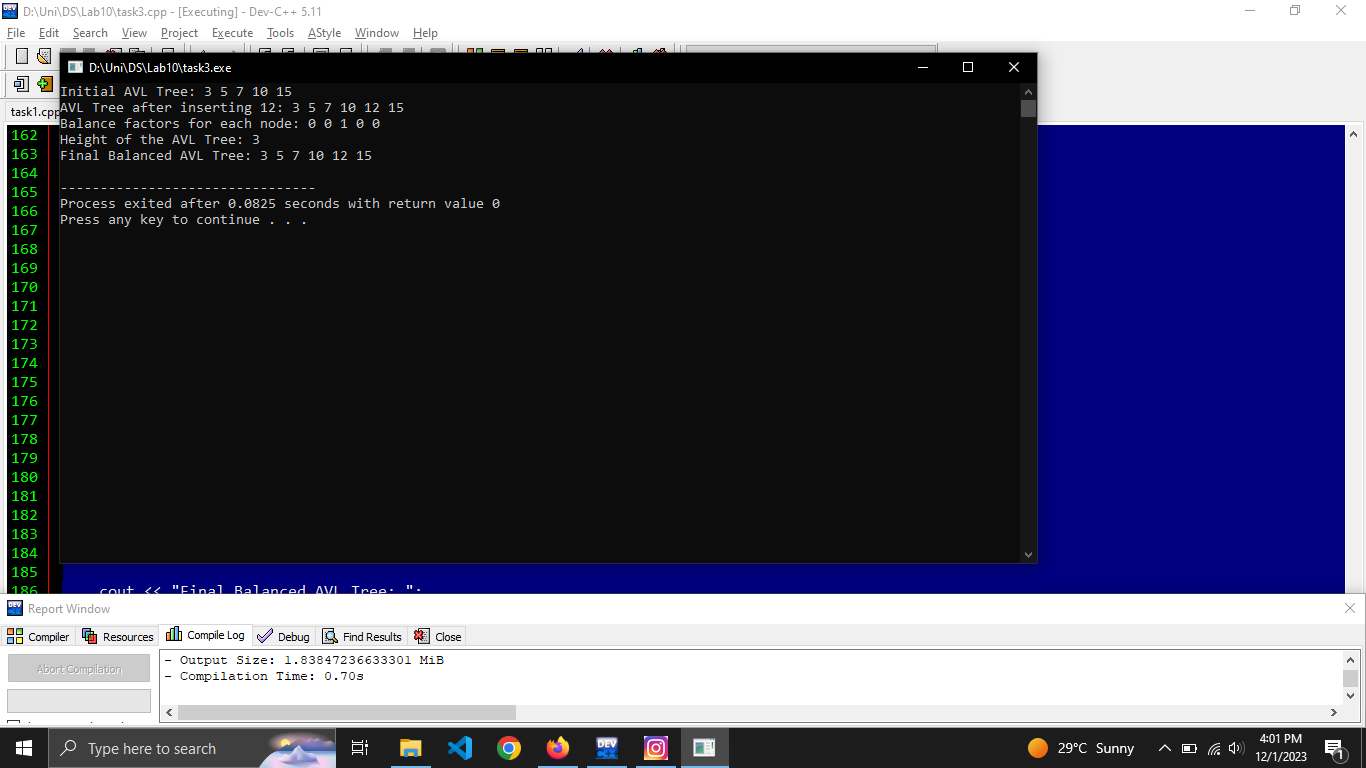
cout << "AVL Tree after left rotation: ";

avlTree.displayTree();

return 0;

}

Task 3:



#include <iostream>

using namespace std;

class AVLNode

{

public:

int data;

AVLNode\* left;

AVLNode\* right;

int height;

AVLNode(int value) : data(value), left(NULL), right(NULL), height(1) {}

};

class AVLTree

{

public:

AVLNode\* root;

int height(AVLNode\* node)

{

if (node == NULL) return 0;

return node->height;

}

int getBalance(AVLNode\* node)

{

if (node == NULL) return 0;

return height(node->left) - height(node->right);

}

void updateHeight(AVLNode\* node)

{

if (node != NULL) {

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

}

}

AVLNode\* rightRotate(AVLNode\* y)

{

AVLNode\* x = y->left;

AVLNode\* T2 = x->right;

x->right = y;

y->left = T2;

updateHeight(y);

updateHeight(x);

return x;

}

AVLNode\* leftRotate(AVLNode\* x)

{

AVLNode\* y = x->right;

AVLNode\* T2 = y->left;

y->left = x;

x->right = T2;

updateHeight(x);

updateHeight(y);

return y;

}

AVLNode\* insert(AVLNode\* node, int value)

{

if (node == NULL)

{

return new AVLNode(value);

}

if (value < node->data)

{

node->left = insert(node->left, value);

} else if (value > node->data)

{

node->right = insert(node->right, value);

} else

{

return node;

}

updateHeight(node);

int balance = getBalance(node);

if (balance > 1 && value < node->left->data)

{

return rightRotate(node);

}

if (balance < -1 && value > node->right->data)

{

return leftRotate(node);

}

if (balance > 1 && value > node->left->data)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && value < node->right->data)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void inorderTraversal(AVLNode\* root)

{

if (root != NULL) {

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right);

}

}

public:

AVLTree() : root(NULL) {}

void insert(int value)

{

root = insert(root, value);

}

void displayTree()

{

inorderTraversal(root);

cout << endl;

}

int getHeight()

{

return height(root);

}

AVLNode\* getRoot()

{

return root;

}

};

int main()

{

AVLTree avlTree;

avlTree.insert(10);

avlTree.insert(5);

avlTree.insert(15);

avlTree.insert(3);

avlTree.insert(7);

cout << "Initial AVL Tree: ";

avlTree.displayTree();

avlTree.insert(12);

cout << "AVL Tree after inserting 12: ";

avlTree.displayTree();

AVLNode\* root = avlTree.getRoot();

cout << "Balance factors for each node: ";

cout << avlTree.getBalance(root) << " ";

AVLNode\* leftChild = root->left;

AVLNode\* rightChild = root->right;

cout << avlTree.getBalance(leftChild) << " ";

cout << avlTree.getBalance(rightChild) << " ";

AVLNode\* leftGrandchild = leftChild->left;

AVLNode\* rightGrandchild = leftChild->right;

cout << avlTree.getBalance(leftGrandchild) << " ";

cout << avlTree.getBalance(rightGrandchild) << " ";

cout << endl;

cout << "Height of the AVL Tree: " << avlTree.getHeight() << endl;

cout << "Final Balanced AVL Tree: ";

avlTree.displayTree();

return 0;

}