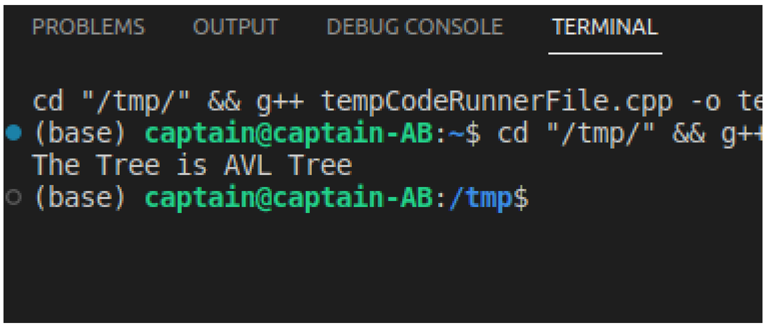
LAB TASK # 10

QUESTION # 1

CODE:

```
#include <iostream>
using namespace std;
class node {
public:
int info;
node* left;
node* right;
};
node* Node(int d) {
node* Node = new node();
Node->info = d;
Node->left = NULL;
Node->right = NULL;
return(Node);
}
int height(node* node) {
if(node == NULL)
return 0;
return 1 + max(height(node->left), height(node->right));
}
int AVL(node *root) {
int lh;
int rh;
if(root == NULL)
return 1;
lh = height(root->left);
rh = height(root->right);
if(abs(lh-rh) <= 1 || (lh-rh) == -1) ;
return 1;
}
int main() {
node *root = Node(7);
root->left = Node(6);
root->right = Node(12);
root->left->left = Node(4);
root->left->right = Node(5);
root->right->right = Node(13);
if(AVL(root))
cout << "The Tree is AVL Tree"<<endl;
else
cout << "The Tree is not AVL Tree "<<endl;
return 0;
}
```

OUTPUT:



```
cd "/tmp/" && g++ tempCodeRunnerFile.cpp -o te
(base) captain@captain-AB:~$ cd "/tmp/" && g++
The Tree is AVL Tree
(base) captain@captain-AB:/tmp$
```

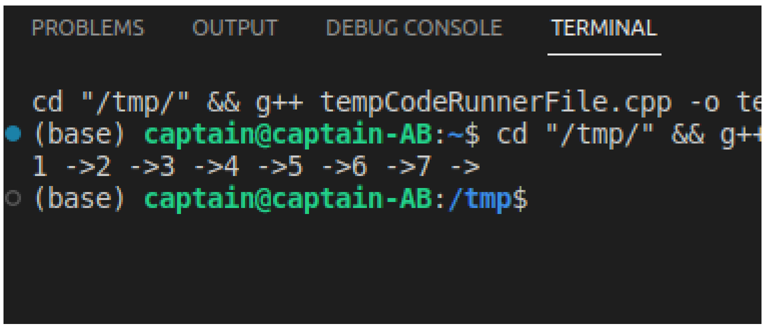
QUESTION # 2

CODE:

```
#include<iostream>
using namespace std;
class Node{
public:
Node *left;
Node*right;
int info;
int height;

Node(int info)
{
this->info=info;
height=0;
left=right=NULL;
}
};

class AVLtree{
private:

Node*root;
void makeEmpty(Node* t);
Node* insert(int x,Node*t);
Node* leftleftrotate(Node* &C);
Node* rightrightrotate(Node*&C);
Node* leftrightrotate(Node* &C);
Node* rightleftrotate(Node* &C);
int height(Node*t);
int getBalance(Node*t);
void inorder(Node *t);
public:
AVLtree()
{
root=NULL;
}
void insert(int x){
root=insert(x,root);
}
void display()
{
inorder(root);
cout<<endl;
}

};



Node* AVLtree::leftleftrotate(Node* &A)
{
Node* newNode = A->right;
A->right = newNode->left;
newNode->left = A;
A->height = max(height(A ->left), height(A ->right)) + 1;
newNode ->height = max(height(newNode->right), A->height) + 1;
return newNode;
}

Node* AVLtree::rightrightrotate(Node* &C)
{
Node* newNode = C->left;
C->left = newNode->right;
newNode->right = C;
C->height = max(height(C ->left), height(C ->right)) + 1;
newNode ->height = max(height(newNode->left), C->height) + 1;
return newNode;
}

Node* AVLtree::leftrightrotate(Node*& t)
{
t->left = leftleftrotate(t->left);
return rightrightrotate(t);
}
```

```
Node* AVLtree::rightleftrotate(Node*& t)
{
t->right = rightrightrotate(t->right);
return leftleftrotate(t);
}

void AVLtree::inorder(Node *t)
{
if(t==NULL)
return;
inorder(t->left);
cout<<t->info<<" ->";
inorder(t->right);
}
int AVLtree::height(Node* t)
{
return(t==NULL ? -1 : t->height);
}
int AVLtree::getBalance(Node*t)
{
if(t==NULL)
return 0;
else
return height(t->left) - height(t->right);
}
void AVLtree::makeEmpty(Node* t) {
if(t == NULL)
return;
makeEmpty(t->left);
makeEmpty(t->right);
delete t;
}
Node* AVLtree:: insert(int x, Node* t)
{
if(t == NULL)
{
t = new Node (x);
}
else if(x < t->info)
{
t->left = insert(x, t->left);
if(height(t->left) - height(t->right) == 2)
{
if(x < t->left->info)
t = rightrightrotate(t);
else
t = leftrightrotate(t);
}
}
else if(x > t->info)
{
t->right = insert(x, t->right);
if(height(t->right) - height(t->left) == 2)
{
if(x > t->right->info)
t = leftleftrotate(t);
else
t = rightleftrotate(t);
}
}

t->height = max(height(t->left), height(t->right))+1;
return t;
}

int main()
{
AVLtree tree1;
tree1.insert(1);
tree1.insert(2);
tree1.insert(7);
tree1.insert(4);
tree1.insert(5);
tree1.insert(6);

tree1.insert(3);

tree1.display();
}
```

OUTPUT:



```
cd "/tmp/" && g++ tempCodeRunnerFile.cpp -o te
(base) captain@captain-AB:~$ cd "/tmp/" && g++
1 ->2 ->3 ->4 ->5 ->6 ->7 ->
(base) captain@captain-AB:/tmp$
```