TASK # 11

Q1:

```
#include<iostream>

using namespace std;


//It is code for printing tree to check my other functions are
working correctly or not
struct Trunk
{
Trunk *prev;
string str;

Trunk(Trunk *prev, string str)
{
this->prev = prev;
this->str = str;
}
};

// Helper function to print branches of the binary tree
void showTrunks(Trunk *p)
{
if (p == NULL)
return;

showTrunks(p->prev);
std::cout<< p->str;
}

class Node
```

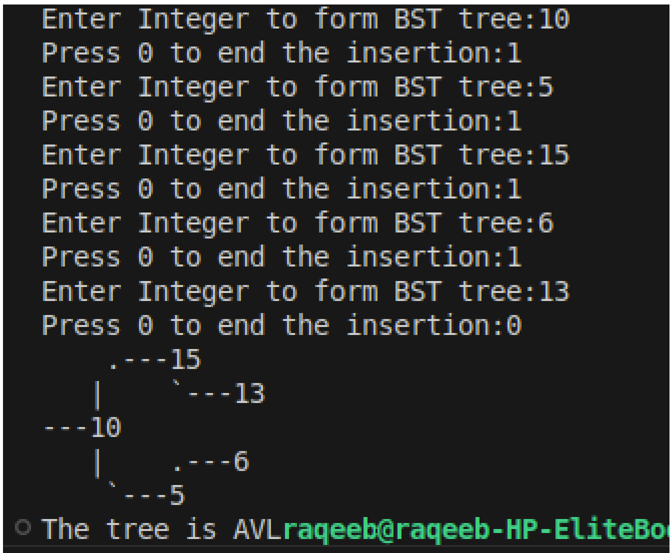
```
{
public:
int info;
Node *left;
Node *right;

Node(int value)
{
info = value;
left=NULL;
right=NULL;
}
};

class BST
{
public:
Node *root;
BST()
{
root = NULL;
}

void Insertion(Node *root2, int value)
{
if(root == NULL)
{
root = new Node(value);
return;
}

if(value < root2->info)
{
if(root2->left == NULL)
{
root2->left = new Node(value);
}
else
{
```

```
Insertion(root2->left, value);
}
}
else
{
if(root2->right == NULL)
{
root2->right = new Node(value);
}
else
{
Insertion(root2->right, value);

}
}
return;;
}

int Check_AVL_or_Not(Node *passed_root)
{
if(root==NULL)
{
cout<<"The tree is empty"<<endl;
return -1;
}

if(passed_root==NULL)
{
return 0;
}
int l=Check_AVL_or_Not(passed_root->left);
int r=Check_AVL_or_Not(passed_root->right);

if(l- r <=1 && r-l<=1)
{
if(l>=r)
return l+1;
else
return r+1;
```

```
}
else
return 2;

}
};

void printTree(Node *root, Trunk *prev, bool isRight)
{
if (root == NULL)
return;
string prev_str = " ";
Trunk *trunk = new Trunk(prev, prev_str);

printTree(root->right, trunk, true);

if (!prev)
trunk->str = "---";
else if (isRight)
{
trunk->str = ".---";
prev_str = " |";
}
else
{
trunk->str = "`---";
prev->str = prev_str;
}

showTrunks(trunk);
cout<< root->info << endl;

if (prev)
prev->str = prev_str;
trunk->str = " |";

printTree(root->left, trunk, false);
}
```

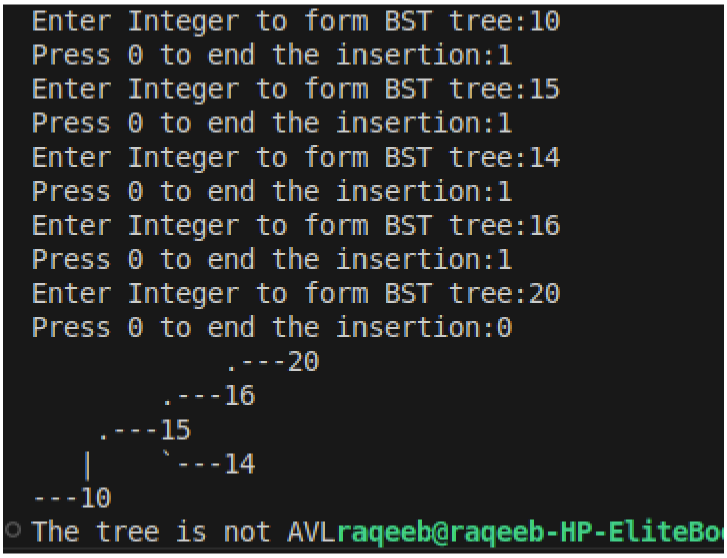
```
int main(void)
{
BST Tree;
while(1)
{
int user_input;
cout<<"Enter Integer to form BST tree:";
cin>>user_input;
Tree.Insertion(Tree.root, user_input);
cout<<"Press 0 to end the insertion:";
cin>>user_input;
if(user_input==0)
break;
}

printTree(Tree.root,NULL,false);
int condition=Tree.Check_AVL_or_Not(Tree.root);
if(condition!=2)
{
cout<<"The tree is AVL";
}
else
{
cout<<"The tree is not AVL";
}


return 0;
}
```

OutPut:
1)



```
Enter Integer to form BST tree:10
Press 0 to end the insertion:1
Enter Integer to form BST tree:5
Press 0 to end the insertion:1
Enter Integer to form BST tree:15
Press 0 to end the insertion:1
Enter Integer to form BST tree:6
Press 0 to end the insertion:1
Enter Integer to form BST tree:13
Press 0 to end the insertion:0
    .---15
   |    `---13
---10
   |    .---6
    `---5
○ The tree is AVLraqeeb@raqeeb-HP-EliteBo
```

2)



```
Enter Integer to form BST tree:10
Press 0 to end the insertion:1
Enter Integer to form BST tree:15
Press 0 to end the insertion:1
Enter Integer to form BST tree:14
Press 0 to end the insertion:1
Enter Integer to form BST tree:16
Press 0 to end the insertion:1
Enter Integer to form BST tree:20
Press 0 to end the insertion:0
            .---20
        .---16
    .---15
   |    `---14
---10
The tree is not AVLraqeeb@raqeeb-HP-EliteBo
```

Q2:

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

class Node
{
public:
int data;
Node *left;
Node *right;

Node(int value)
{
```

```
data=value;
left=NULL;
right=NULL;
}
};

//For printing Tree only
class Trunk
{
public:
Trunk *prev;
string str;

Trunk(Trunk *prev, string str) : prev(prev), str(str) {}
};

class AVLTree
{
public:
Node *root;

AVLTree()
{
root=NULL;
}

Node *rightRotate(Node *Passed)
{
Node *My_node1 = Passed->left;
Node *My_node2 = My_node1->right;

My_node1->right = Passed;
Passed->left = My_node2;
```

```
return My_node1;
}

Node *leftRotate(Node *Passed)
{
Node *My_node1 = Passed->right;
Node *My_node2 = My_node1->left;

My_node1->left = Passed;
Passed->right = My_node2;

return My_node1;
}

int Check_AVL_or_Not(Node *passed_root)
{
if (passed_root == NULL)
{
return 0;
}

int l = Check_AVL_or_Not(passed_root->left);
int r = Check_AVL_or_Not(passed_root->right);

if (l == -1 || r == -1 || abs(l - r) > 1)
{
return -1;
}

return max(l, r) + 1;
}

Node *Insertion(Node *passed_root, int value)
{
```

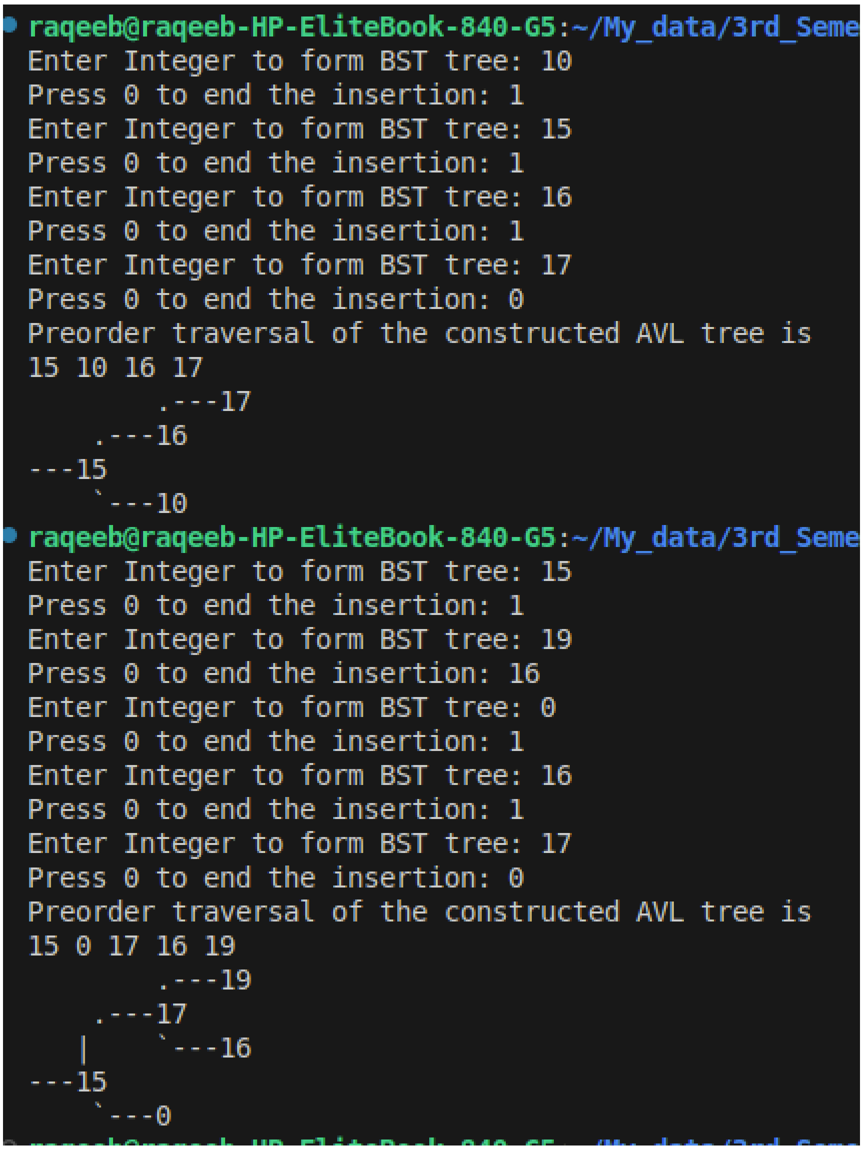
```
if (passed_root == NULL)
{
return new Node(value);
}

if (value < passed_root->data)
{
passed_root->left = Insertion(passed_root->left,
value);
}
else if (value > passed_root->data)
{
passed_root->right = Insertion(passed_root->right,
value);
}
else
{
return passed_root;
}

int checker = Check_AVL_or_Not(passed_root);
//This part do the rotation process if needed
according to the checker value
if (checker == -1)
{
if (Check_AVL_or_Not(passed_root->left) >
Check_AVL_or_Not(passed_root->right))
{
if (value < passed_root->left->data)
{
return rightRotate(passed_root);
}
else
{
```

```
passed_root->left = leftRotate(passed_root->left);
return rightRotate(passed_root);
}
}
else
{
if (value > passed_root->right->data)
{
return leftRotate(passed_root);
}
else
{
passed_root->right = rightRotate(passed_root->right);
return leftRotate(passed_root);
}
}
}

return passed_root;
}

void preOrder(Node *root)
{
if (root != NULL)
{
cout << root->data << " ";
preOrder(root->left);
preOrder(root->right);
}
}


//For printing the tree
void showTrunks(Trunk *p)
```

```
{
if (p == NULL)
{
return;
}

showTrunks(p->prev);
cout << p->str;
}

void printTree(Node *root, Trunk *prev, bool isRight)
{
if (root == NULL)
{
return;
}

string prev_str = " ";
Trunk *trunk = new Trunk(prev, prev_str);

printTree(root->right, trunk, true);

if (!prev)
{
trunk->str = "---";
}
else if (isRight)
{
trunk->str = ".---";
prev_str = " |";
}
else
{
trunk->str = "`---";
prev->str = prev_str;
}

showTrunks(trunk);
cout << root->data << endl;

if (prev)
{
prev->str = prev_str;
}
trunk->str = " |";

printTree(root->left, trunk, false);
}
};

int main()
{
//In this code all RR, LL,LR ,RL is implement
//For balancing factor I have used Check_AVL_or_not function
AVLTree My_Tree;
while (1)
{
int user_input;
cout << "Enter Integer to form BST tree: ";
cin >> user_input;
My_Tree.root = My_Tree.Insertion(My_Tree.root, user_input);
cout << "Press 0 to end the insertion: ";
cin >> user_input;
if (user_input == 0)
break;
}
```

```
cout << "Preorder traversal of the constructed AVL
tree is \n";
My_Tree.preOrder(My_Tree.root);
cout << endl;
cout<<"Tree in AVL form"<<endl;
My_Tree.printTree(My_Tree.root, NULL, false);

return 0;
}
```

output:



```
raqeeb@raqeeb-HP-EliteBook-840-G5:~/My_data/3rd_Seme
Enter Integer to form BST tree: 10
Press 0 to end the insertion: 1
Enter Integer to form BST tree: 15
Press 0 to end the insertion: 1
Enter Integer to form BST tree: 16
Press 0 to end the insertion: 1
Enter Integer to form BST tree: 17
Press 0 to end the insertion: 0
Preorder traversal of the constructed AVL tree is
15 10 16 17
        .---17
    .---16
---15
    `---10
raqeeb@raqeeb-HP-EliteBook-840-G5:~/My_data/3rd_Seme
Enter Integer to form BST tree: 15
Press 0 to end the insertion: 1
Enter Integer to form BST tree: 19
Press 0 to end the insertion: 16
Enter Integer to form BST tree: 0
Press 0 to end the insertion: 1
Enter Integer to form BST tree: 16
Press 0 to end the insertion: 1
Enter Integer to form BST tree: 17
Press 0 to end the insertion: 0
Preorder traversal of the constructed AVL tree is
15 0 17 16 19
        .---19
    .---17
   |    `---16
---15
    `---0
```