**class Node** defines a node that will contain the data, also known as int key and the left and right pointers to its children. It contains an integer height that stores the height of the tree as well.

**int max()** function returns the maximum of two integers.

**int height(Node\*)** takes in a pointer as parameter and if the pointer is NULL, then returns 0. Otherwise it recursively calls the function using the pointer to the height.

**Node\* newNode(int)** takes an integer argument which is the data to be stored inside each node of the AVL tree and. The left and right pointers of each newly created node is NULL and height is 1. This function is of pointer type and returns the pointer to a newly created node.

**Node\* rightRotate(Node \*y)** takes in a pointer to a node y as argument. It assigns the left pointer of y to a new pointer to a node x and the right pointer of x to node T2. It then performs rotation by assigning y to the right pointer of x and T2 to the left pointer of y. Then the heights are updated by assigning height of y to be the maximum of height of left subtree and maximum of right subtree plus 1. Similarly, the height of x is also updated. The new root x is then returned.

**Node\* leftRotate(Node \*x)** takes in a pointer to a node x as argument. It assigns the right pointer of x to a new pointer to a node y and the left pointer of y to node T2. It then performs rotation by assigning y to the right pointer of x and T2 to the left pointer of y. Then the heights are updated by assigning height of x to be the maximum of height of left subtree and maximum of right subtree plus 1. Similarly, the height of y is also updated. The new root y is then returned.

**Int getBalance(Node \*N)** is used to find the difference in heights of the left and right subtree. If the argument node is NULL, then it returns 0, as bt default, leaves have a 0 imbalance. If the node has further pointers, the the difference in heights of the left and right subtree is returned.

**Node\* insert(Node\* node, int key)** takes a node and the value to be inserted as arguments. If the node equals NULL, then the value is inserted into the node. If the value is less than the current node’s value, then insert it left and if value is greater, insert it to its right, just like normal BST insertion. The height of the ancestor node is updated every time BST insertion takes place and the balance factor of every node is stored as an integer balance. Following 4 cases arise-

1.If balance is greater than 1 and its value is lesser than its left child’s value, it has to be right rotated.

2.If balance is lesser than -1 and its value is greater than its right child’s value, it has to be left rotated.

3.If balance is greater than 1 and its value is greater than its eft child’s value, it has to be left right rotated. This is achieved in two steps. First, the node’s left pointer is left rotated. Then the node itself is right rotated and that is returned.

4.If balance is lesser than -1 and its value is lesser than its right child’s value, it has to be right left rotated. This is achieved in two steps. First, the node’s right pointer is right rotated. Then the node itself is left rotated and that is returned.

Finally, the node pointer is returned.

**void preOrder(Node \*root, int height)** takes the root pointer and integer height as an argument. If height equals 0, it prints the value of the root. If its left pointer is not NULL, then it recursively calls the same preorder function till it reaches a left leaf node. Similarly, if the node’s right pointer is not NULL, then it recursively calls the same preorder function till it reaches a right leaf node.

If the height isn’t 0, then it prints the value of the root and then recursively calls the left and right pointers of the root with height updated to height plus 1. This prints out a nice indented tree.

**void printGraphviz(Node \*rr)** essentially takes the root pointer and prints it in a format that can be copied and pasted to the Webgraphviz website so that a neat looking tree can be generated.

The **main()** program consists of creating a Node pointer called root and assigning it to NULL initially. Then the user is asked whether they want a custom input addition or a random sequence of nodes. A switch statement helps achieve this. If it is custom inputs, then the values are inserted. If random is chosen, 1000 random integers generated by the rand() function are inserted into the graph using a for loop. Height of tree is displayed in both cases and a digraph is generated for manual input.