Name: - Atish Kumar

Roll No: - 120CS0173

Lab Sheet: - 10

Q1. Write a program to implement an AVL Tree having following functionalities

. • Insert (): This function inserts a new node to an AVL tree. The node contains an integer type of data.

• BF(): This function returns the balance factor of a given node.

• LL(): This function performs LL rotation.

• RR(): This function performs RR rotation.

• LR(): This function performs LR rotation.

• RL(): This function performs RL rotation.

• Display (): This function displays inorder traversal sequence of the AVL tree. After inserting a new node, if the resulting tree is not AVL then insert function calls appropriated rotation function to make the tree an AVL

Program:-

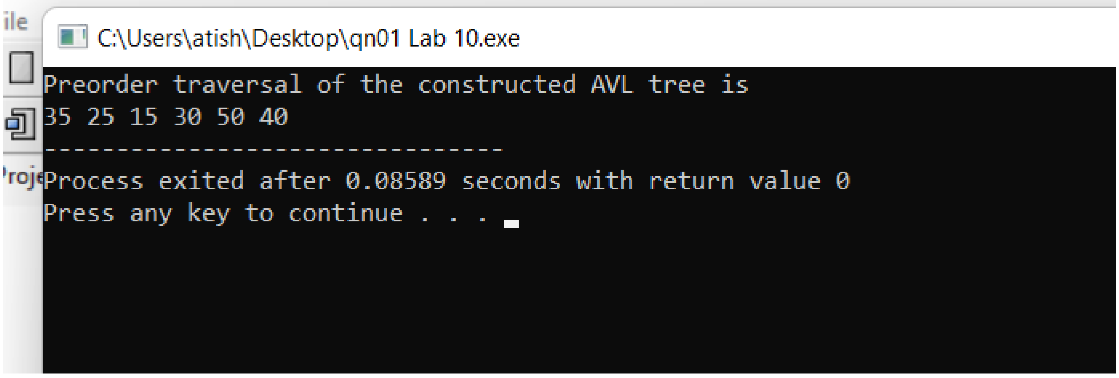
```
#include<stdio.h>
#include<stdlib.h>
 struct Node
{
  int key;
   struct Node *left;
   struct Node *right;
   int height;
};
int max(int a, int b);
 int height(struct Node *N)
{
   if (N == NULL)
     return 0;
   return N->height;
}
int max(int a, int b)
```

```
{

  return (a > b)? a : b;
}
struct Node* newNode(int key)
{
 struct Node* node = (struct Node*)

            malloc(sizeof(struct Node));
  node->key   = key;
  node->left   = NULL;
  node->right  = NULL;
  node->height = 1;  // new node is initially added at leaf
  return(node);
}
 struct Node *rightRotate(struct Node *y)
{
 struct Node *x = y->left;
  struct Node *T2 = x->right;
   // Perform rotation

  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;
}
```

```
struct Node *leftRotate(struct Node *x)
{
  struct Node *y = x->right;

   struct 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;
}
int getBalance(struct Node *N)
{
   if (N == NULL)
      return 0;
   return height(N->left) - height(N->right);
}
struct Node* insert(struct Node* node, int key)
{
 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 // Equal keys are not allowed in BST
      return node;
  node->height = 1 + max(height(node->left),
                    height(node->right));
 int balance = getBalance(node);
   // there are 4 cases
   // Left Left Case
   if (balance > 1 && key < node->left->key)
      return rightRotate(node);
   // Right Right Case
   if (balance < -1 && key > node->right->key)
      return leftRotate(node);

   // Left Right Case
   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;
}
void preOrder(struct Node *root)
{
   if(root != NULL)
   {
      printf("%d ", root->key);
      preOrder(root->left);
      preOrder(root->right);
   }
}
int main()
{
  struct Node *root = NULL;
  root = insert(root, 15);
  root = insert(root, 35);
  root = insert(root, 50);
  root = insert(root, 40);
  root = insert(root, 25);
  root = insert(root, 30);
  printf("Preorder traversal of the constructed AVL"
        " tree is \n");
  preOrder(root);
  return 0;
}
```

OutPut:-



```
Preorder traversal of the constructed AVL tree is
35 25 15 30 50 40
--------------------------------
Process exited after 0.08589 seconds with return value 0
Press any key to continue . . .
```