

# **Data Structures and Algorithms**

## **Lab Report**

### **Lab11**



Group Members Name & Reg #:	<b><u>Muhammad Haris Irfan</u></b> <b>(FA18-BCE-090)</b>
Class	Data Structures and Algorithms CSC211 ( <b>BCE-3B</b> )
Instructor's Name	Dilshad Sabir

# In Lab Tasks

## Task:1

Your task is to modify the insert function to incorporate AVL insertion

## Solution:

The code is shown below,

```
struct node* insert(struct node* node, int data)
{
    if (node == NULL)
        return(newNode(data));
    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
    else
        return node;
    node->BalFac = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1 && data < node->left->data)
        return rightRotate(node);
    if (balance < -1 && data > node->right->data)
        return leftRotate(node);
    if (balance > 1 && data > node->left->data)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && data < node->right->data)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
```

Code for right rotate is below:

```
struct node *rightRotate(struct node *y)
{
    struct node *x = y->left;
    struct node *T2 = x->right;

    // Perform rotation
    x->right = y;
    y->left = T2;

    // Update heights
    y->BalFac = max(height(y->left), height(y->right))+1;
    x->BalFac = max(height(x->left), height(x->right))+1;

    // Return new root
    return x;
}
```

### Code for left rotate is below:

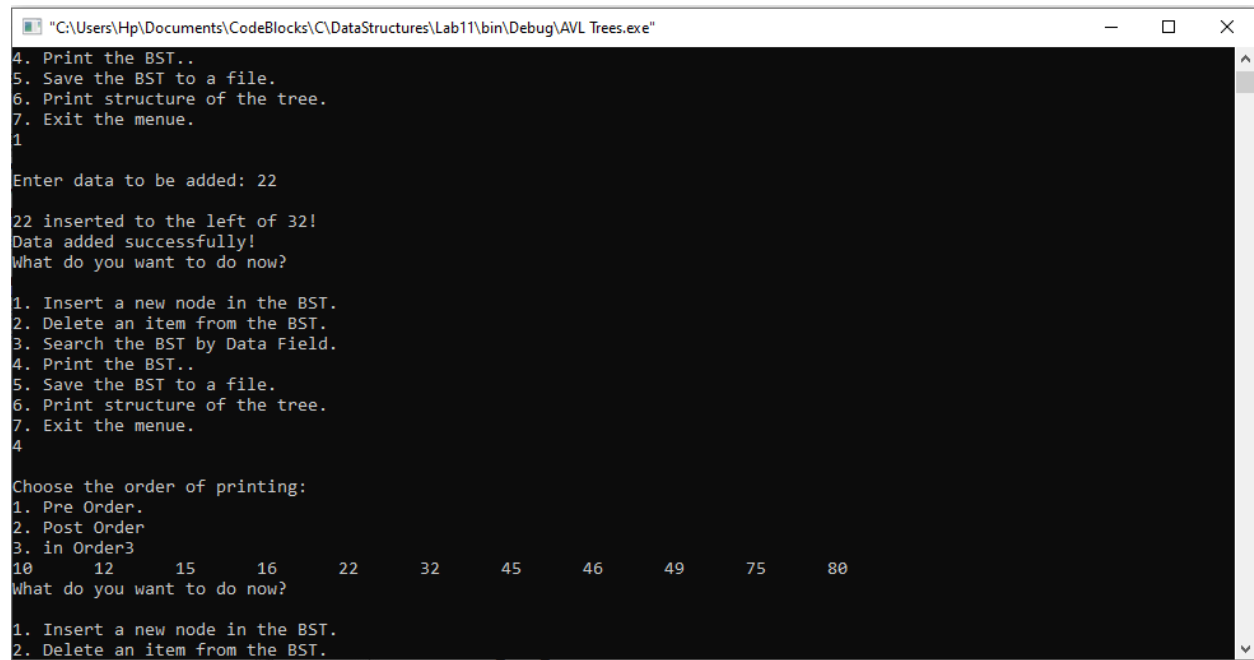
```
struct node *leftRotate(struct node *x)
{
    struct node *y = x->right;
    struct node *T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    // Update heights
    x->BalFac = max(height(x->left), height(x->right))+1;
    y->BalFac = max(height(y->left), height(y->right))+1;

    // Return new root
    return y;
}
```

### The Result of the following code is attached below:



```
"C:\Users\Hp\Documents\CodeBlocks\C\DataStructures\Lab11\bin\Debug\AVL Trees.exe"
4. Print the BST..
5. Save the BST to a file.
6. Print structure of the tree.
7. Exit the menu.
1

Enter data to be added: 22
22 inserted to the left of 32!
Data added successfully!
What do you want to do now?

1. Insert a new node in the BST.
2. Delete an item from the BST.
3. Search the BST by Data Field.
4. Print the BST..
5. Save the BST to a file.
6. Print structure of the tree.
7. Exit the menu.
4

Choose the order of printing:
1. Pre Order.
2. Post Order
3. in Order3
10 12 15 16 22 32 45 46 49 75 80
What do you want to do now?

1. Insert a new node in the BST.
2. Delete an item from the BST.
```

# Post Lab Task.

## Task 2:

Save the Tree data to a file (In-Order, Pre-Order and Post-Order)

## Solution

The code is shown below,

### In-Order

```
bool save_in_order(struct node * root, FILE * fptr)
{
    /** Complete this function */

    int data = root->data;
    if(root->left == NULL)    /// Now data of this node will be printed
        fprintf(fptr, " %d ", data);

    if(root->left != NULL)
    {
        save_in_order(root->left, fptr);
        fprintf(fptr, " %d ", data);
    }

    if(root->right != NULL)
    {
        save_in_order(root->right, fptr);
    }

    return;
}
```

### Pre-Order

```
bool save_pre_order(struct node * root, FILE * fptr)
{
    int data = root->data;
    if(root != NULL)
        fprintf(fptr, " %d ", data);
    if(root->left != NULL){
        save_pre_order(root->left, fptr);
    }
    if(root->right != NULL){
        save_pre_order(root->right, fptr);
    }
    return;
}
```

### Post-Order

```
bool save_post_order(struct node * root, FILE * fptr)
{
    int data = root->data;
    if(root->left != NULL){
        print_post_order(root->left);
    }
    if(root->right != NULL){
        print_post_order(root->right);
    }
    fprintf(fptr, " %d ", data);
    return;
}
```

=====

### Task 3:

**Load tree from a file containing numbers.**

### Solution

The code is shown below,

```
int load_tree(struct node * root, FILE * fptr)
{
    /** Complete this function */
    printf("\n\nPrinting tree loaded from a file.\n");
    int i;
    for(int j=0;j<10;j++){
        fscanf(fptr, "%d", &i);
        insert_node(&root, i);
    }
}
```

**The Result of the following code is attached below:**

```
C:\Users\Hp\Documents\CodeBlocks\C\DataStructures\Lab10\bin\Debug\Lab10.exe
Hello! This program lets you manage you an integer BST:
Do you want to load a tree from the file? (y/n): y

Printing tree loaded from a file.

1969897746 as Root node inserted!
1969897746 inserted to the right of 1969897746!
1969897746 inserted to the right of 1969897746!
1969897746 inserted to the right of 1969897746!
1969897746 inserted to the right of 1969897746!
1969897746 inserted to the right of 1969897746!
1969897746 inserted to the right of 1969897746!
1969897746 inserted to the right of 1969897746!
1969897746 inserted to the right of 1969897746!
45 as Root node inserted!
12 inserted to the left of 45!
16 inserted to the right of 12!
32 inserted to the right of 16!
75 inserted to the right of 45!
46 inserted to the left of 75!
15 inserted to the left of 16!
10 inserted to the left of 12!
80 inserted to the right of 75!
49 inserted to the right of 46!

What do you want to do now?
```

## Conclusion:

In this lab, we completed the AVL tree insertion function, to complete an AVL tree after inserting a value we needed to rotate the tree at certain points for which we made two functions, furthermore in Post-lab we implemented saving the our AVL Search Tree as well as loading a tree from a file containing numbers.

-----  
THE END