Implementation of AVL Tree

```
#include<stdio.h>
#include<stdlib.h>

// structure of the tree node
struct node
{
   int data;
   struct node* left;
   struct node* right;
   int ht;
};

// global initialization of root node
struct node* root = NULL;

// function prototyping
struct node* create(int);
struct node* insert(struct node*, int);
struct node* delete(struct node*, int);
struct node* search(struct node*, int);
struct node* rotate_left(struct node*);
struct node* rotate_right(struct node*);
int balance_factor(struct node*);
int height(struct node*);
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);

int main()
{
   int user_choice, data;
   char user_continue = 'y';
   struct node* result = NULL;

   while (user_continue == 'y' || user_continue == 'Y')
   {
      printf("\n\n------- AVL TREE --------\n");
      printf("\n1. Insert");
      printf("\n2. Delete");
      printf("\n3. Search");
```

```
printf("\n4. Inorder");
printf("\n5. Preorder");
printf("\n6. Postorder");
printf("\n7. EXIT");

printf("\n\nEnter Your Choice: ");
scanf("%d", &user_choice);

switch(user_choice)
{
   case 1:
      printf("\nEnter data: ");
      scanf("%d", &data);
      root = insert(root, data);
      break;

   case 2:
      printf("\nEnter data: ");
      scanf("%d", &data);
      root = delete(root, data);
      break;

   case 3:
      printf("\nEnter data: ");
      scanf("%d", &data);
      result = search(root, data);
      if (result == NULL)
      {
         printf("\nNode not found!");
      }
      else
      {
         printf("\n Node found");
      }
      break;
   case 4:
      inorder(root);
      break;

   case 5:
```

```
            preorder(root);
            break;

        case 6:
            postorder(root);
            break;

        case 7:
            printf("\n\tProgram Terminated\n");
            return 1;

        default:
            printf("\n\tInvalid Choice\n");
      }

      printf("\n\nDo you want to continue? ");
      scanf(" %c", &user_continue);
   }

   return 0;
}

// creates a new tree node
struct node* create(int data)
{
   struct node* new_node = (struct node*) malloc (sizeof(struct node));

   // if a memory error has occurred
   if (new_node == NULL)
   {
      printf("\nMemory can't be allocated\n");
      return NULL;
   }
   new_node->data = data;
   new_node->left = NULL;
   new_node->right = NULL;
   return new_node;
}

// rotates to the left
```

```
struct node* rotate_left(struct node* root)
{
    struct node* right_child = root->right;
    root->right = right_child->left;
    right_child->left = root;

    // update the heights of the nodes
    root->ht = height(root);
    right_child->ht = height(right_child);

    // return the new node after rotation
    return right_child;
}

// rotates to the right
struct node* rotate_right(struct node* root)
{
    struct node* left_child = root->left;
    root->left = left_child->right;
    left_child->right = root;

    // update the heights of the nodes
    root->ht = height(root);
    left_child->ht = height(left_child);

    // return the new node after rotation
    return left_child;
}

// calculates the balance factor of a node
int balance_factor(struct node* root)
{
    int lh, rh;
    if (root == NULL)
        return 0;
    if (root->left == NULL)
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
```

```
        rh = 0;
    else
        rh = 1 + root->right->ht;
    return lh - rh;
}

// calculate the height of the node
int height(struct node* root)
{
    int lh, rh;
    if (root == NULL)
    {
        return 0;
    }
    if (root->left == NULL)
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
        rh = 0;
    else
        rh = 1 + root->right->ht;

    if (lh > rh)
        return (lh);
    return (rh);
}

// inserts a new node in the AVL tree
struct node* insert(struct node* root, int data)
{
    if (root == NULL)
    {
        struct node* new_node = create(data);
        if (new_node == NULL)
        {
            return NULL;
        }
        root = new_node;
    }
```

```
else if (data > root->data)
{
   // insert the new node to the right
   root->right = insert(root->right, data);

   // tree is unbalanced, then rotate it
   if (balance_factor(root) == -2)
   {
      if (data > root->right->data)
      {
         root = rotate_left(root);
      }
      else
      {
         root->right = rotate_right(root->right);
         root = rotate_left(root);
      }
   }
}
else
{
   // insert the new node to the left
   root->left = insert(root->left, data);

   // tree is unbalanced, then rotate it
   if (balance_factor(root) == 2)
   {
      if (data < root->left->data)
      {
         root = rotate_right(root);
      }
      else
      {
         root->left = rotate_left(root->left);
         root = rotate_right(root);
      }
   }
}
// update the heights of the nodes
root->ht = height(root);
```

```
    return root;
}

// deletes a node from the AVL tree
struct node * delete(struct node *root, int x)
{
    struct node * temp = NULL;

    if (root == NULL)
    {
        return NULL;
    }

    if (x > root->data)
    {
        root->right = delete(root->right, x);
        if (balance_factor(root) == 2)
        {
            if (balance_factor(root->left) >= 0)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
                root = rotate_right(root);
            }
        }
    }
    else if (x < root->data)
    {
        root->left = delete(root->left, x);
        if (balance_factor(root) == -2)
        {
            if (balance_factor(root->right) <= 0)
            {
                root = rotate_left(root);
            }
            else
            {
```

```
            root->right = rotate_right(root->right);
            root = rotate_left(root);
        }
    }
}
else
{
    if (root->right != NULL)
    {
        temp = root->right;
        while (temp->left != NULL)
            temp = temp->left;

        root->data = temp->data;
        root->right = delete(root->right, temp->data);
        if (balance_factor(root) == 2)
        {
            if (balance_factor(root->left) >= 0)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
                root = rotate_right(root);
            }
        }
    }
    else
    {
        return (root->left);
    }
}
root->ht = height(root);
return (root);
}

// search a node in the AVL tree
struct node* search(struct node* root, int key)
{
```

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

    if(root->data == key)
    {
        return root;
    }

    if(key > root->data)
    {
        search(root->right, key);
    }
    else
    {
        search(root->left, key);
    }
}

// inorder traversal of the tree
void inorder(struct node* root)
{
    if (root == NULL)
    {
        return;
    }

    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

// preorder traversal of the tree
void preorder(struct node* root)
{
    if (root == NULL)
    {
        return;
    }
```

```
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

// postorder traversal of the tree
void postorder(struct node* root)
{
    if (root == NULL)
    {
        return;
    }

    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
```

```
.Insert Beg
2.Insert Middle
3.Insert End
4.Delete Beg
5.Delete Middle
6.Delete End
7.Find
8.Traverse
9.Exit
Enter your choice : 1
Enter the element : 1
Enter your choice : 2
Enter the position element : 2
Enter the element : 2


...Program finished with exit code 0
Press ENTER to exit console.
```

```
.Delete Beg
.Delete Middle
.Delete End
.Find
.Traverse
.Exit
nter your choice : 3
nter the element : 5
nter your choice : 6
he deleted item is 5
nter your choice : 1
nter the element : 1
nter your choice : 3
nter the element : 5
nter your choice : 7
nter the element : 5
lement found...!
nter your choice : 8
       5
nter your choice : 9
```