 \* AVL Tree Program in C

 \*/

**t** node\*

root, **int** key)

{

**if** (root == NULL)

    {  #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\*);

**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. 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:

                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(**struc**

**return** NULL;

    }

**if**(root->data == key)

    {

**return** root;

    }

**if**(key > root->data)

    {

        search(root->right, key);

    }

**else**

    {

        search(root->left, key);

    }

}

return root;

// inorder traversal of the tree

**void** inorder(**struct** node\* root)

{

**if** (root == NULL)

    {

**return**;

    }

    inorder(root->left);

    printf("%d ", root->data);

    inorder(root->right);

}

**Output**