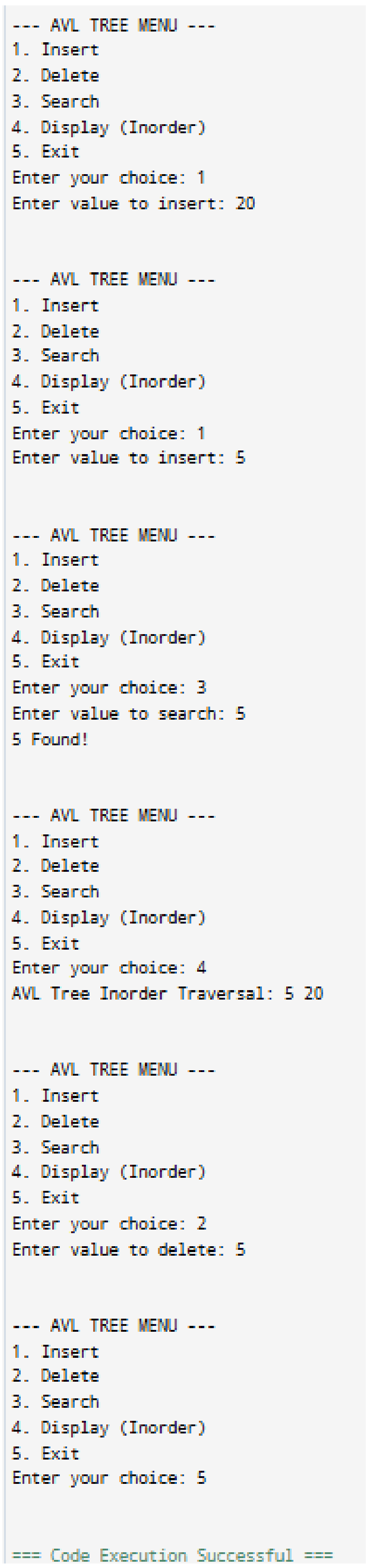
Experiment 21: AVL Tree

Code:

```
#include  <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node *left, *right;

    int height;

};

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* createNode(int value) {

    struct node* newNode = (struct node*)malloc(sizeof(struct node));

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    newNode->height = 1;

    return newNode;

}

struct node* rightRotate(struct node* y) {

    struct node* x = y->left;

    struct node* T2 = x->right;

    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 createNode(key);
    if (key < node->data)
        node->left = insert(node->left, key);
    else if (key > node->data)
        node->right = insert(node->right, key);
    else
        return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1 && key < node->left->data)
        return rightRotate(node);
    if (balance < -1 && key > node->right->data)
        return leftRotate(node);
    if (balance > 1 && key > node->left->data) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->data) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
struct node* minValueNode(struct node* node) {
    struct node* current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}
struct node* deleteNode(struct node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        if ((root->left == NULL) || (root->right == NULL)) {
            struct node* temp = root->left ? root->left : root->right;
            if (temp == NULL) {
                temp = root;
                root = NULL;
            } else {
                *root = *temp;
            }
            free(temp);
        } else {
            struct node* temp = minValueNode(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }
    if (root == NULL) return root;
    root->height = max(height(root->left), height(root->right)) + 1;
    int balance = getBalance(root);
    if (balance > 1 && getBalance(root->left) >= 0)
        return rightRotate(root);
    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
struct node* search(struct node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (key < root->data)
        return search(root->left, key);
    return search(root->right, key);
}
void inorder(struct node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
int main() {
    struct node* root = NULL;
    int choice, value;
    while (1) {
        printf("\n\n--- AVL TREE MENU ---\n");
        printf("1. Insert\n2. Delete\n3. Search\n4. Display (Inorder)\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                root = deleteNode(root, value);
                break;
            case 3:
                printf("Enter value to search: ");
                scanf("%d", &value);
                if (search(root, value))
                    printf("%d Found!\n", value);
                else
                    printf("%d Not Found!\n", value);
                break;
            case 4:
                printf("AVL Tree Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;

            case 5:
                exit(0);
            default:
                printf("Invalid Choice!\n");
        }
    }
    return 0;
}
```

Output:



```
--- AVL TREE MENU ---
1. Insert
2. Delete
3. Search
4. Display (Inorder)
5. Exit
Enter your choice: 1
Enter value to insert: 20


--- AVL TREE MENU ---
1. Insert
2. Delete
3. Search
4. Display (Inorder)
5. Exit
Enter your choice: 1
Enter value to insert: 5


--- AVL TREE MENU ---
1. Insert
2. Delete
3. Search
4. Display (Inorder)
5. Exit
Enter your choice: 3
Enter value to search: 5
5 Found!


--- AVL TREE MENU ---
1. Insert
2. Delete
3. Search
4. Display (Inorder)
5. Exit
Enter your choice: 4
AVL Tree Inorder Traversal: 5 20


--- AVL TREE MENU ---
1. Insert
2. Delete
3. Search
4. Display (Inorder)
5. Exit
Enter your choice: 2
Enter value to delete: 5


--- AVL TREE MENU ---
1. Insert
2. Delete
3. Search
4. Display (Inorder)
5. Exit
Enter your choice: 5


=== Code Execution Successful ===
```