WEEK - 4

Kavin.J.S
CH.SC.U4CSE24119

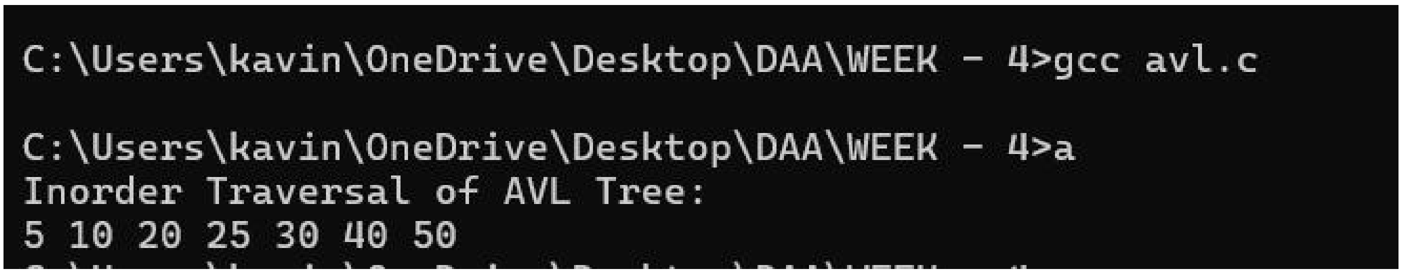
AVL Tree:

Code:

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

struct Node {
   int data;
   struct Node *left, *right;
   int height;
};

int max(int a, int b) {
   return (a > b) ? a : b;
}

int height(struct Node *n) {
   return (n == NULL) ? 0 : n->height;
}

struct Node* createNode(int value) {
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = value;
   node->left = node->right = NULL;
   node->height = 1;
   return node;
}

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;
```

```c
    return y;
}

int getBalance(struct Node *n) {
    return (n == NULL) ? 0 : height(n->left) - height(n->right);
}

struct Node* insert(struct Node* node, int value) {

    if (node == NULL)
        return createNode(value);

    if (value < node->data)
        node->left = insert(node->left, value);
    else if (value > node->data)
        node->right = insert(node->right, value);
    else
        return node;

    node->height = 1 + max(height(node->left), height(node->right));

    int balance = getBalance(node);

    if (balance > 1 && value < node->left->data)
        return rightRotate(node);

    if (balance < -1 && value > node->right->data)
        return leftRotate(node);

    if (balance > 1 && value > node->left->data) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && value < node->right->data) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

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 arr[] = {30, 20, 40, 10, 25, 50, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    for (int i = 0; i < n; i++)
        root = insert(root, arr[i]);

    printf("Inorder Traversal of AVL Tree:\n");
    inorder(root);
    return 0;
}
```

Output:



```
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 4>gcc avl.c

C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 4>a
Inorder Traversal of AVL Tree:
5 10 20 25 30 40 50
```

Red Black Tree:

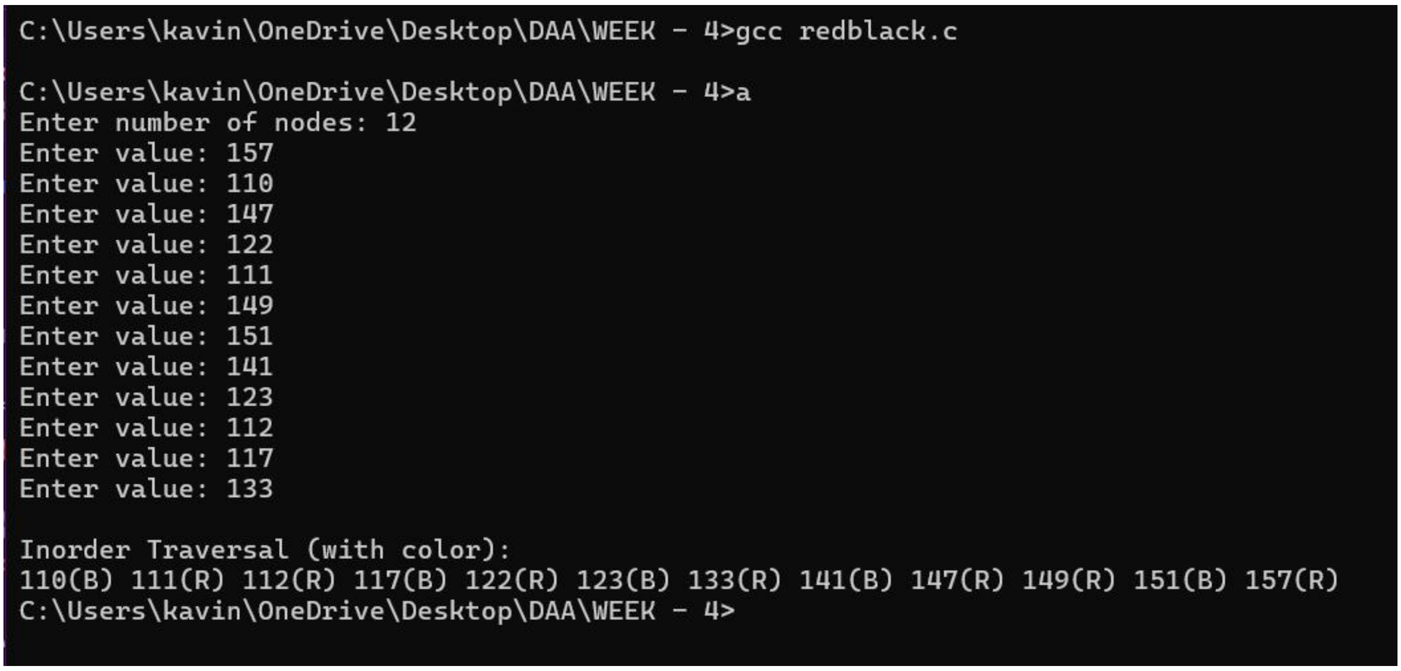
Code:

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

typedef enum { RED, BLACK } Color;

struct Node {
    int data;
    Color color;
    struct Node *left, *right, *parent;
};

struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->color = RED;
    node->left = node->right = NULL;
    node->parent = NULL;
    return node;
}

void leftRotate(struct Node **root, struct Node *x) {
    struct Node *y = x->right;
    x->right = y->left;

    if (y->left)
        y->left->parent = x;

    y->parent = x->parent;

    if (!x->parent)
        *root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;

    y->left = x;
    x->parent = y;
```

```
}

void rightRotate(struct Node **root, struct Node *y) {
  struct Node *x = y->left;
  y->left = x->right;

  if (x->right)
    x->right->parent = y;

  x->parent = y->parent;

  if (!y->parent)
    *root = x;
  else if (y == y->parent->left)
    y->parent->left = x;
  else
    y->parent->right = x;

  x->right = y;
  y->parent = x;
}

void fixInsert(struct Node **root, struct Node *z) {
  while (z != *root && z->parent->color == RED) {

    if (z->parent == z->parent->parent->left) {
      struct Node *y = z->parent->parent->right;

      if (y && y->color == RED) {
        z->parent->color = BLACK;
        y->color = BLACK;
        z->parent->parent->color = RED;
        z = z->parent->parent;
      }
      else {
        if (z == z->parent->right) {
          z = z->parent;
          leftRotate(root, z);
        }

        z->parent->color = BLACK;
        z->parent->parent->color = RED;
        rightRotate(root, z->parent->parent);
      }

    } else {
      struct Node *y = z->parent->parent->left;

      if (y && y->color == RED) {
        z->parent->color = BLACK;
        y->color = BLACK;
        z->parent->parent->color = RED;
        z = z->parent->parent;
      }
      else {
        if (z == z->parent->left) {
          z = z->parent;
          rightRotate(root, z);
        }

        z->parent->color = BLACK;
        z->parent->parent->color = RED;
        leftRotate(root, z->parent->parent);
```

```
            }
        }
    }

    (*root)->color = BLACK;
}

void insert(struct Node **root, int data) {
    struct Node *z = createNode(data);
    struct Node *y = NULL;
    struct Node *x = *root;

    while (x) {
        y = x;
        if (z->data < x->data)
            x = x->left;
        else
            x = x->right;
    }

    z->parent = y;

    if (!y)
        *root = z;
    else if (z->data < y->data)
        y->left = z;
    else
        y->right = z;

    fixInsert(root, z);
}

void inorder(struct Node *root) {
    if (root) {
        inorder(root->left);
        printf("%d(%s) ", root->data,
               root->color == RED ? "R" : "B");
        inorder(root->right);
    }
}

int main() {
    struct Node *root = NULL;
    int n, val;

    printf("Enter number of nodes: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter value: ");
        scanf("%d", &val);
        insert(&root, val);
    }

    printf("\nInorder Traversal (with color):\n");
    inorder(root);

    return 0;
}
```

Output:



```
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 4>gcc redblack.c

C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 4>a
Enter number of nodes: 12
Enter value: 157
Enter value: 110
Enter value: 147
Enter value: 122
Enter value: 111
Enter value: 149
Enter value: 151
Enter value: 141
Enter value: 123
Enter value: 112
Enter value: 117
Enter value: 133

Inorder Traversal (with color):
110(B) 111(R) 112(R) 117(B) 122(R) 123(B) 133(R) 141(B) 147(R) 149(R) 151(B) 157(R)
C:\Users\kavin\OneDrive\Desktop\DAA\WEEK - 4>
```