**Code:**

# **Name : Rajkumar B L**

# **Reg.No : 2047120**

# **Course : MCS 271 DS (Lab Test 03)**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* Name : Rajkumar B L

 \* Reg  : 2047120

 \* Lab  : Test 03

 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int key;

    struct node \*left, \*right;

};

struct node \*newNode(int item)

{

    struct node \*temp = (struct node \*)malloc(sizeof(struct node));

    temp->key = item;

    temp->left = temp->right = NULL;

    return temp;

}

void traversetree(struct node \*root)

{

    if (root != NULL)

    {

        traversetree(root->left);

        printf("%d -> ", root->key);

        traversetree(root->right);

    }

}

struct node \*search(struct node \*root, int key)

{

    if (root == NULL || root->key == key)

        return root;

    if (root->key < key)

        return search(root->right, key);

    return search(root->left, key);

}

int height\_of\_binary\_tree(struct node \*node)

{

    if (node == NULL)

        return 0;

    else

    {

        int left\_side;

        int right\_side;

        left\_side = height\_of\_binary\_tree(node->left);

        right\_side = height\_of\_binary\_tree(node->right);

        if (left\_side > right\_side)

        {

            return left\_side + 1;

        }

        else

            return right\_side + 1;

    }

}

void lbst\_rbst(struct node \*node, int \*lbst, int \*rbst)

{

    if (node == NULL){

            \*lbst = 0;

            \*rbst = 0;

    }

    else

    {

        \*lbst = height\_of\_binary\_tree(node->left);

        \*rbst = height\_of\_binary\_tree(node->right);

    }

}

int isBstIdentical(struct node \*root1,struct node \*root2)

{

    if (root1 == NULL && root2 == NULL)

        return 1;

    else if (root1 != NULL && root2 == NULL)

        return 0;

    else if (root1 == NULL && root2 != NULL)

        return 0;

    else

    {

        if (root1->key == root2->key && isBstIdentical(root1->left, root2->left) && isBstIdentical(root1->right, root2->right))

            return 1;

        else

            return 0;

    }

}

struct node \*insert(struct node \*node, int key)

{

    if (node == NULL)

        return newNode(key);

    if (key < node->key)

        node->left = insert(node->left, key);

    else if (key > node->key)

        node->right = insert(node->right, key);

    return node;

}

int main(int argc, char const \*argv[])

{

    int ch, i, num;

    struct node \*root = NULL;

    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\*  Name : Rajkumar B L  \*\n\*  Reg  : 2047120       \*\n\*  Lab  : Test 03       \*\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

    // Creating BST 01

    printf("\nLets create BST-01:-");

    do

    {

        printf("\n=========================\n\tMenu\n=========================\n");

        printf("1. Insert a node to BST-01. \n");

        printf("2. Traverse BST-01. \n");

        printf("3. Finish Creating BST-01 \n");

        printf("=========================\n");

        printf("Enter your choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter the number to be inserted: ");

            scanf("%d", &num);

            if (root == NULL)

            {

                root = insert(root, num);

                printf("%d inserted successfully!\n", num);

            }

            else

            {

                insert(root, num);

                printf("%d inserted successfully!\n", num);

            }

            break;

        case 2:

            printf("The tree is :\n");

            //printf("Root -> \t");

            traversetree(root);

            printf("\n");

            break;

        case 3:

            printf("BST-01 Created Successfully!\n");

            printf("Height of BST-01 is: %d \n\n", height\_of\_binary\_tree(root));

            break;

        case 4:

            printf("Bye!\n\n");

            exit(0);

        default:

            printf("Invalid Input!\n");

        }

    } while (ch != 3);

    //Creating BST-02

    int ch2, i2, num2;

    struct node \*root2 = NULL;

    printf("\nLets create BST-02:-");

    do

    {

        printf("\n=========================\n\tMenu\n=========================\n");

        printf("1. Insert a node to BST-02. \n");

        printf("2. Traverse BST-02. \n");

        printf("3. Finish Creating BST-02 \n");

        printf("=========================\n");

        printf("Enter your choice: ");

        scanf("%d", &ch2);

        switch (ch2)

        {

        case 1:

            printf("Enter the number to be inserted: ");

            scanf("%d", &num2);

            if (root2 == NULL)

            {

                root2 = insert(root2, num2);

                printf("%d inserted successfully!\n", num2);

            }

            else

            {

                insert(root2, num2);

                printf("%d inserted successfully!\n", num2);

            }

            break;

        case 2:

            printf("The tree is :\n");

            //printf("Root -> \t");

            traversetree(root2);

            printf("\n");

            break;

        case 3:

            printf("BST-02 Created Successfully!\n");

            printf("Height of BST-02 is: %d \n\n", height\_of\_binary\_tree(root2));

            break;

        default:

            printf("Invalid Input!\n");

        }

    } while (ch2 != 3);

    //Final Output

    int ch3;

    int lbst, rbst;

    printf("\nLabTest 03 Output:-");

    do

    {

        printf("\n=========================\n\tMenu\n=========================\n");

        printf("1. Print BST-01. \n");

        printf("2. Print BST-02. \n");

        printf("3. Check Identical \n");

        printf("4. Exit \n");

        printf("=========================\n");

        printf("Enter your choice: ");

        scanf("%d", &ch2);

        switch (ch2)

        {

        case 1:

            printf("The Bst-01 tree is : ");

            traversetree(root);

            printf(" Null");

            lbst\_rbst(root, &lbst, &rbst);

            if (lbst > rbst)

                printf("\nBST-02 Tree is uneven LBST:%d RSBT:%d\n", lbst, rbst);

            else if (lbst < rbst)

                printf("\nBST-02 Tree is uneven LBST:%d RSBT:%d\n", lbst, rbst);

            else if (lbst == rbst)

                printf("\nBST-02 Tree is even LBST:%d RSBT:%d\n", lbst, rbst);

            printf("Height of BST-01 is: %d \n\n", height\_of\_binary\_tree(root));

            break;

        case 2:

            printf("The BST-02 tree is : ");

            traversetree(root2);

            printf(" Null");

            lbst\_rbst(root2,&lbst, &rbst);

            if (lbst > rbst)

                printf("\nBST-02 Tree is uneven LBST:%d RSBT:%d\n",lbst,rbst);

            else if (lbst < rbst)

                printf("\nBST-02 Tree is uneven LBST:%d RSBT:%d\n", lbst, rbst);

            else if(lbst==rbst)

                printf("\nBST-02 Tree is even LBST:%d RSBT:%d\n", lbst, rbst);

            printf("Height of BST-02 is: %d \n\n", height\_of\_binary\_tree(root2));

            break;

        case 3:

            if (isBstIdentical(root, root2))

                printf("BST-01 AND BST-02 are identical!\n");

            else

                printf("BST-01 AND BST-02 are not identical!\n");

            break;

        default:

            printf("Invalid Input!\n");

        }

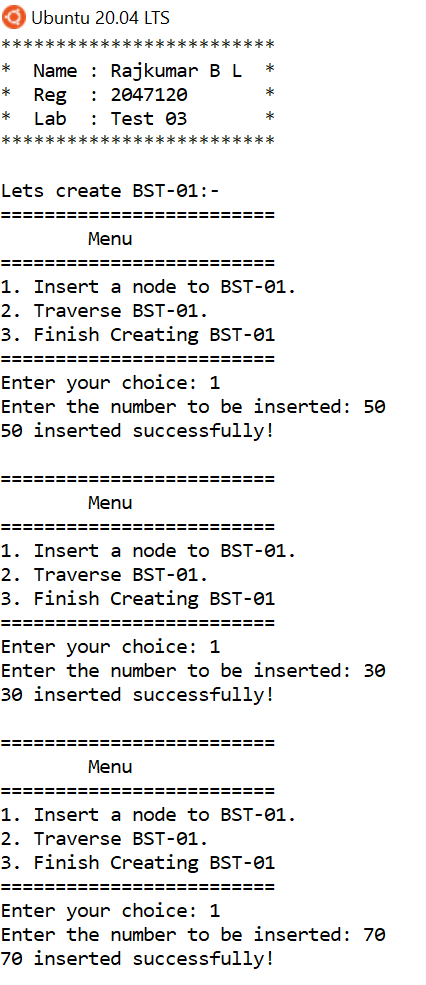
    } while (ch2 != 4);

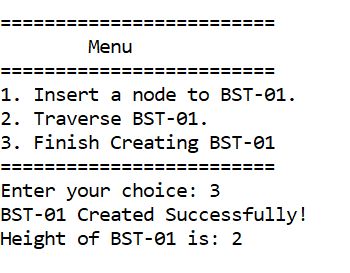
    return 0;

}

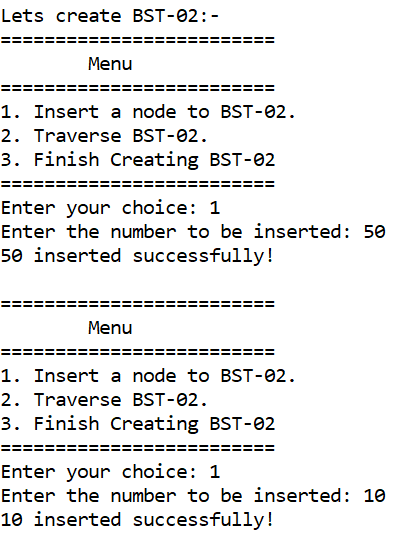
**Output:**

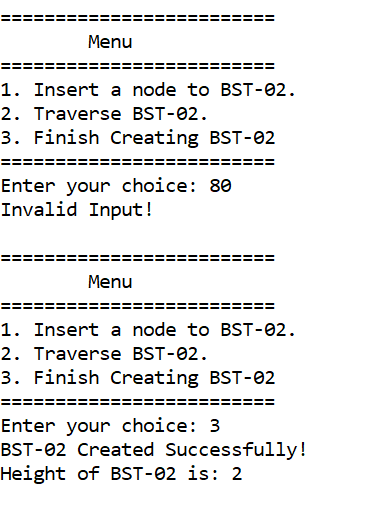
**Creating BST-01**

****

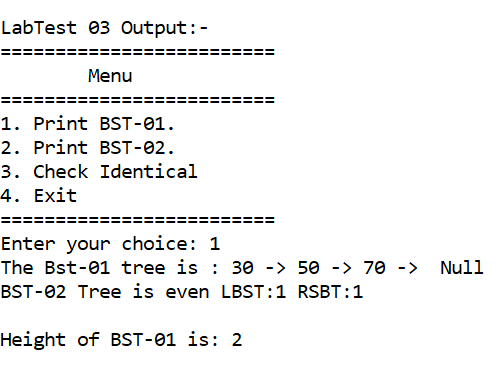
****

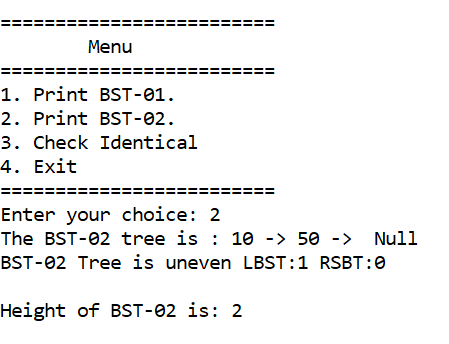
**Creating BST-02**

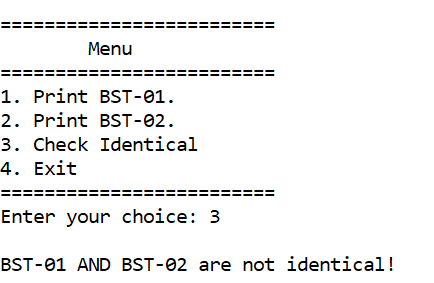
****

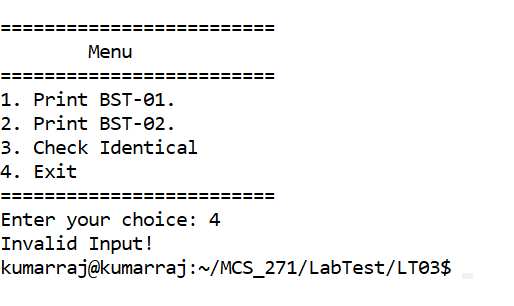
****

**LT03 – Output (Display, Height, Identical)**

****

****

****

****