**Assignment No. 6**

**Name: Bhavin Ratansing Patil**

**Roll No.: 26 SEDA**

**Q.1 Create Binary Tree and Find height of the tree and print leaf nodes. Find mirror image and print original and mirror image.**

**Binary Tree**

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



**Algorithm:**

1. **Mirror of the tree Recursive:**

**Step 1:** If treet! = NULL

**Step 2:** temp =tree->Right

tree->Right= tree->Left

tree->Left=temp

**Step 3:** Mirror\_BST(tree->Left)

**Step 4:** Mirror\_BST(tree->Right)

1. **Height of the tree recursive:**

**Step 1:** If (root=NULL)

Display “tree is empty”

stop

**Step 2:** Else

**Step 3:** return 1 + max(height(root->left), height(root->right))

1. **Leaf Node of tree:**

**Step 1:** if (r == NULL)

return 0;

**Step 2:** if (r->left == NULL && r->right == NULL)

printf("\t%d", r->data);

**Step 3:** return (leaf(r->left) + leaf(r->right))

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct node \*st[100];

int top = -1;

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

} \* root;

// ---------------------------------------------

struct node \*create()

{

    int ch;

    struct node \*temp;

    printf("\nDo you want to create a Tree ?\n(1 for yes 0 for No): ");

    scanf("%d", &ch);

    if (ch == 0)

        return NULL;

    temp = malloc(sizeof(struct node));

    printf("\nEnter the data: ");

    scanf("%d", &temp->data);

    printf("\nEnter data for left child of %d", temp->data);

    temp->left = create();

    printf("\nEnter data for right child of %d", temp->data);

    temp->right = create();

    return temp;

}

// ---------------------------------------------

struct node \*search(struct node \*temp, int data)

{

    if (temp == NULL)

    {

        printf("\n\tData is not present");

        return NULL;

    }

    if (temp->data == data)

    {

        printf("\n\tData is present\n");

        return temp;

    }

    if (temp->data > data)

    {

        return search(temp->left, data);

    }

    else

    {

        return search(temp->right, data);

    }

}

// ---------------------------------------------

void inorder(struct node \*temp)

{

    if (temp != NULL)

    {

        inorder(temp->left);

        printf("\t%d", temp->data);

        inorder(temp->right);

    }

}

void preorder(struct node \*temp)

{

    if (temp != NULL)

    {

        printf("\t%d", temp->data);

        preorder(temp->left);

        preorder(temp->right);

    }

}

void postorder(struct node \*temp)

{

    if (temp != NULL)

    {

        postorder(temp->left);

        postorder(temp->right);

        printf("\t%d", temp->data);

    }

}

// ---------------------------------------------

void push(struct node \*temp)

{

    st[++top] = temp;

}

struct node \*pop()

{

    return st[top--];

}

void inordernr(struct node \*temp)

{

    struct node \*r;

    while (temp != NULL)

    {

        push(temp);

        temp = temp->left;

    }

    while (top != -1)

    {

        r = pop();

        printf("\t%d", r->data);

        r = r->right;

        while (r != NULL)

        {

            push(r);

            r = r->left;

        }

    }

}

void preordernr(struct node \*temp)

{

    struct node \*r;

    while (temp != NULL)

    {

        printf("\t%d", temp->data);

        push(temp);

        temp = temp->left;

    }

    while (top != -1)

    {

        r = pop();

        r = r->right;

        while (r != NULL)

        {

            printf("\t%d", r->data);

            push(r);

            r = r->left;

        }

    }

}

// ---------------------------------------------

int leaf(struct node \*r)

{

    if (r == NULL)

        return 0;

    if (r->left == NULL && r->right == NULL)

    {

        printf("\t%d", r->data);

        return 1;

    }

    return (leaf(r->left) + leaf(r->right));

}

// ---------------------------------------------

struct node \*mirror(struct node \*T)

{

    struct node \*temp;

    if (T == NULL)

    {

        return NULL;

    }

    else

    {

        temp = T->left;

        T->left = mirror(T->right);

        T->right = mirror(temp);

        return T;

    }

}

// ---------------------------------------------

int max(int value1, int value2)

{

    return ((value1 > value2) ? value1 : value2);

}

int height(struct node \*temp)

{

    if (temp == NULL)

    {

        return 0;

    }

    else if (temp->left == NULL && temp->right == NULL)

    {

        return 0;

    }

    return (max(height(temp->left), height(temp->right)) + 1);

}

void main()

{

    struct node \*temp, \*temp1, \*temp3;

    int choice, ele;

    do

    {

        printf("\n1)Create\n2)Inorder(recursive)\n3)Preorder(recursive)\n4)Postorder(recursive)\n5)Inorder(Non-recursive)\n6)Preorder(Non-recursive)\n7)Search\n8)Print Leaf Node\n9)Height of Tree\n10)Mirror\n0)Quit\n\nEnter Your Choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            root = create();

            temp = root;

            break;

        case 2:

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

            inorder(temp);

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

            break;

        case 3:

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

            preorder(temp);

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

            break;

        case 4:

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

            postorder(temp);

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

            break;

        case 5:

            printf("\n=====Inorder (Non - Recursive)=====\n");

            inordernr(temp);

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

            break;

        case 6:

            printf("\n=====Preorder (Non - Recursive)=====\n");

            preordernr(temp);

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

            break;

        case 7:

            printf("\nEnter the data do you want to search: ");

            scanf("%d", &ele);

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

            search(root, ele);

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

            break;

        case 8:

            printf("\n=====Leafs of Tree=====\n");

            printf("\nNo. of Leaf Nodes = %d", leaf(temp));

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

            break;

        case 9:

            printf("\n=====Height of Tree=====\n");

            int h = height(temp);

            printf("\t%d", h);

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

            break;

        case 10:

            printf("Printing Original and Mirror in Inorder\n");

            temp3 = temp;

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

            inorder(temp3);

            temp1 = mirror(temp3);

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

            inorder(temp1);

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

            break;

        default:

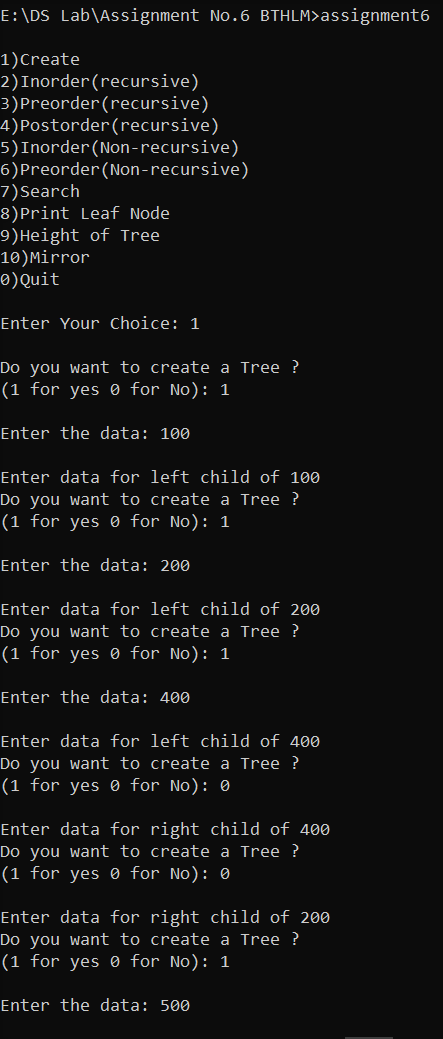
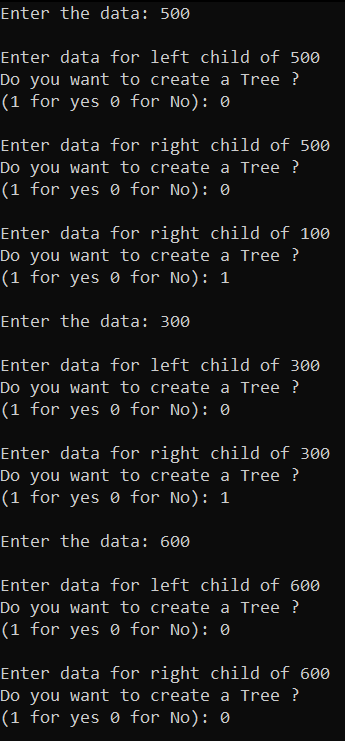
            break;

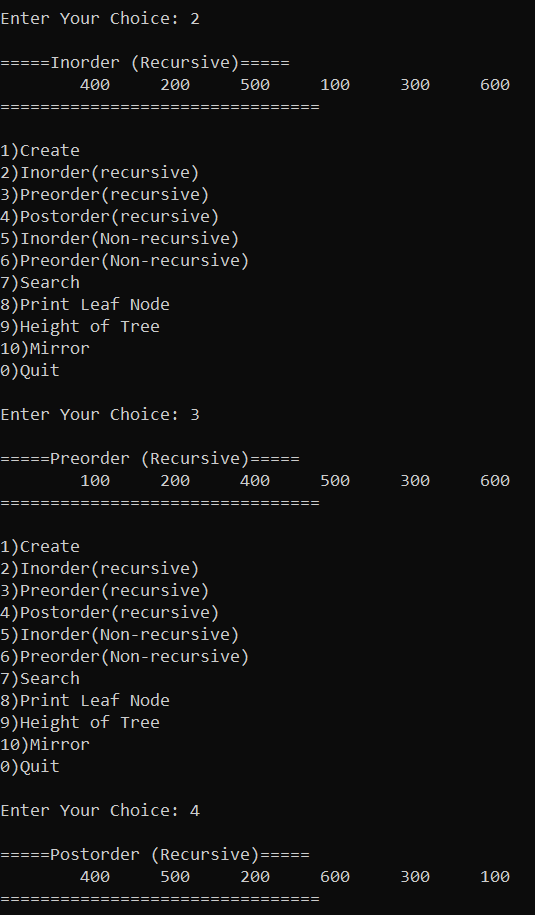
        }

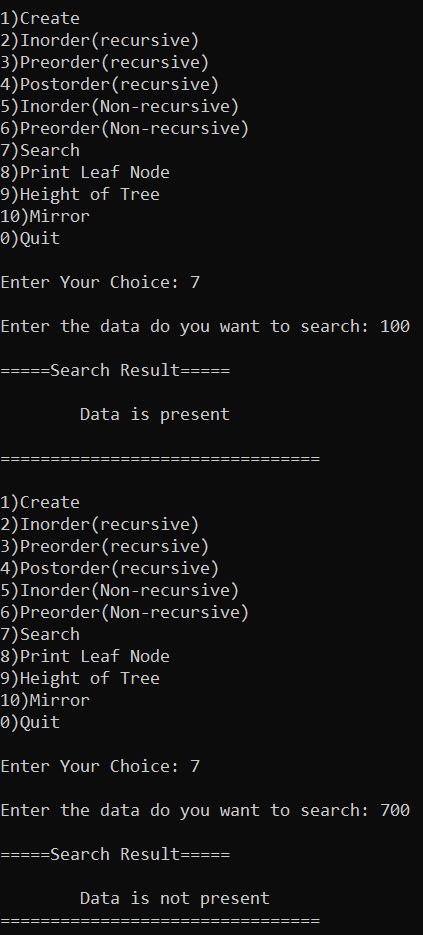
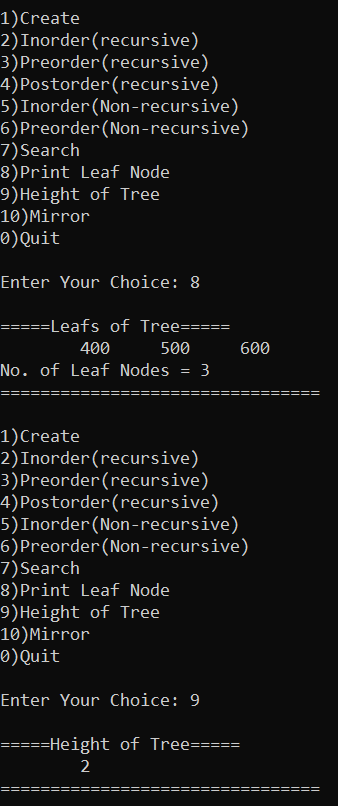
    } while (choice != 0);

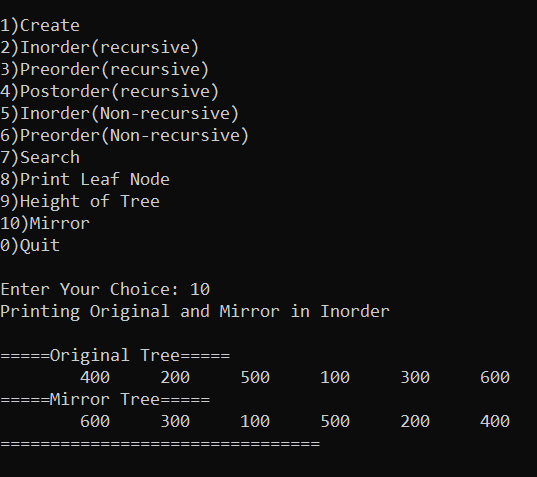
}

**Output:**

 ****

****

** **

****