**ABHISHEK KUMAR**

**DATA STRUCTURE VIVA#3**

**Dated: 2nd December 2022**

**QUESTIONS:**

1. Check if the given Binary tree is BST or not.
2. For a given binary search tree, print the in-order traversal.
3. For a given Binary tree, print the leaf nodes of the tree.
4. For the given BST, print the sum of all the nodes.
5. For a given BST, check if the tree is a complete binary tree or not, and use the balance factor to determine it.
6. In a BST Insert element at a specified location.
7. Delete the root node from BST for a given input.

**CODE**

*// Abhishek Kumar*

*// SAP ID 500106851*

*// MCA Batch 1 VIVA#3 Dated 2nd November 2022*

#include <stdio.h>

#include <stdlib.h>

struct node *// Structure Of Node*

{

    int data;

    struct node \*left;

    struct node \*right;

};

int sum = 0;

char ch, ch1;

struct node \*create(struct node \*, int);

void display(struct node \*, int);

void inorder(struct node \*);

void checkbin(struct node \*);

int sumele(struct node \*);

struct node \*deleteroot(struct node \*, int);

void leafnode(struct node \*);

struct node \*minval(struct node \*);

int balancefactor(struct node \*);

int lheight(struct node \*);

int height(struct node \*);

int main() *// Main Function*

{

    struct node \*p = NULL;

    int choice, item, item1, flag = 0, ele1, ele2;

    while (1)

    {

        printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_BINARY TREE MENU\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

        printf("\n1. Create Binary Tree");

        printf("\n2. Display Binary Tree");

        printf("\n3. Print Inorder Traversal.");

        printf("\n4. Add All Elements Of The Tree.");

        printf("\n5. Check If Complete Binary Tree Or Not.");

        printf("\n6. Insert Element At A Specified Location.");

        printf("\n7. Delete Nodes.");

        printf("\n8. Print Leaf Nodes Of The Binary Tree.");

        printf("\n9. Exit");

        printf("\nEnter Your Choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            do

            {

                printf("\nEnter The Element: ");

                scanf("%d", &item);

                if (flag == 0)

                {

                    p = create(p, item);

                    flag = 1;

                }

                else

                {

                    create(p, item);

                }

                printf("\nDo You Want To Add Another Element in the Binary Search Tree?(Y/N): ");

                scanf(" %c", &ch);

            } while ((ch == 'y') || (ch == 'Y'));

            break;

        case 2:

            printf("\nNodes Of The Binary Search Tree: \n\n");

            display(p, 1);

            break;

        case 3:

            printf("\nInorder Traversal");

            inorder(p);

            break;

        case 4:

            printf("\nAdding All Elements Of The Tree.\n\n\n");

            int addi = sumele(p);

            printf("\nSum Of The Elements Of All Nodes Of The Tree Is: %d", addi);

            sum = 0;

            break;

        case 5:

            printf("\nChecking If Complete Binary Tree Or Not: ");

            int bf = balancefactor(p);

            printf("\nBalance Factor: %d", bf);

            if (bf == 0)

            {

                printf("\n\nThis Is A Complete Binary Tree.");

            }

            else

            {

                printf("\n\nThis Is Not A Complete Binary Tree.");

            }

            break;

        case 6:

            printf("\nEnter The Element To Be Inserted In The Given Binary Tree: ");

            scanf("%d", &item1);

            create(p, item1);

            break;

        case 7:

            deleteroot(p, p->data);

            display(p, 1);

            break;

        case 8:

            printf("\nPrinting Leaf Nodes Of The Binary Tree: ");

            leafnode(p);

            break;

        case 9:

            exit(1);

        default:

            printf("\nInvalid Choice!!!");

            break;

        }

    }

    return 0;

}

struct node \*create(struct node \*ptr, int ele) *// Creation and Insertion Function*

{

    if (ptr == NULL)

    {

        printf("\nNode Created\n");

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

        ptr->left = NULL;

        ptr->right = NULL;

        ptr->data = ele;

        return ptr;

    }

    else if (ele < ptr->data)

    {

        ptr->left = create(ptr->left, ele);

    }

    else

    {

        ptr->right = create(ptr->right, ele);

    }

    return ptr;

}

void display(struct node \*ptr, int level) *// Display Function*

{

    if (ptr != NULL)

    {

        display(ptr->right, level + 1);

        for (int i = 0; i < level; i++)

            printf("   ");

        printf("%4d\n", ptr->data);

        display(ptr->left, level + 1);

    }

    return;

}

void inorder(struct node \*ptr) *// Inorder Traversal*

{

    if (ptr != NULL)

    {

        inorder(ptr->left);

        printf("->");

        printf("%d", ptr->data);

        inorder(ptr->right);

    }

    return;

}

struct node \*deleteroot(struct node \*ptr, int ele) *// Delete Root Function*

{

    if (ptr == NULL)

        return ptr;

    if (ele < ptr->data)

        ptr->left = deleteroot(ptr->left, ele);

    else if (ele > ptr->data)

        ptr->right = deleteroot(ptr->right, ele);

    else

    {

        if (ptr->left == NULL)

        {

            struct node \*temp = ptr->right;

            free(ptr);

            return temp;

        }

        else if (ptr->right == NULL)

        {

            struct node \*temp = ptr->left;

            free(ptr);

            return temp;

        }

        struct node \*temp = minval(ptr->right);

        ptr->data = temp->data;

        ptr->right = deleteroot(ptr->right, temp->data);

    }

    return ptr;

}

int sumele(struct node \*ptr) *// Adding Elements Function*

{

    if (ptr != NULL)

    {

        sum = sum + ptr->data;

        sumele(ptr->left);

        sumele(ptr->right);

    }

    return sum;

}

void leafnode(struct node \*ptr) *// Printing Leaf Nodes*

{

    if (ptr != NULL)

    {

        leafnode(ptr->left);

        leafnode(ptr->right);

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

        {

            printf("%d ", ptr->data);

        }

    }

    return;

}

struct node \*minval(struct node \*ptr) *// Finding Left Most Node.*

{

    struct node \*temp = ptr;

    while (temp && temp->left != NULL)

        temp = temp->left;

    return temp;

}

int balancefactor(struct node \*ptr)*//Calculating Balance Factor.*

{

    int bfvalue;

    int hleft = lheight(ptr->left);

    int hright = height(ptr->right);

    bfvalue = abs(hleft - hright);

    return bfvalue;

}

int lheight(struct node \*ptr)*//Calculating Left height Of Tree.*

{

    int lefth;

    if (ptr == NULL)

        return 0;

    else

        lefth = lheight(ptr->left);

    return (lefth + 1);

}

int height(struct node \*ptr)*//Calculating Right Height of Tree.*

{

    int righth;

    if (ptr == NULL)

        return 0;

    else

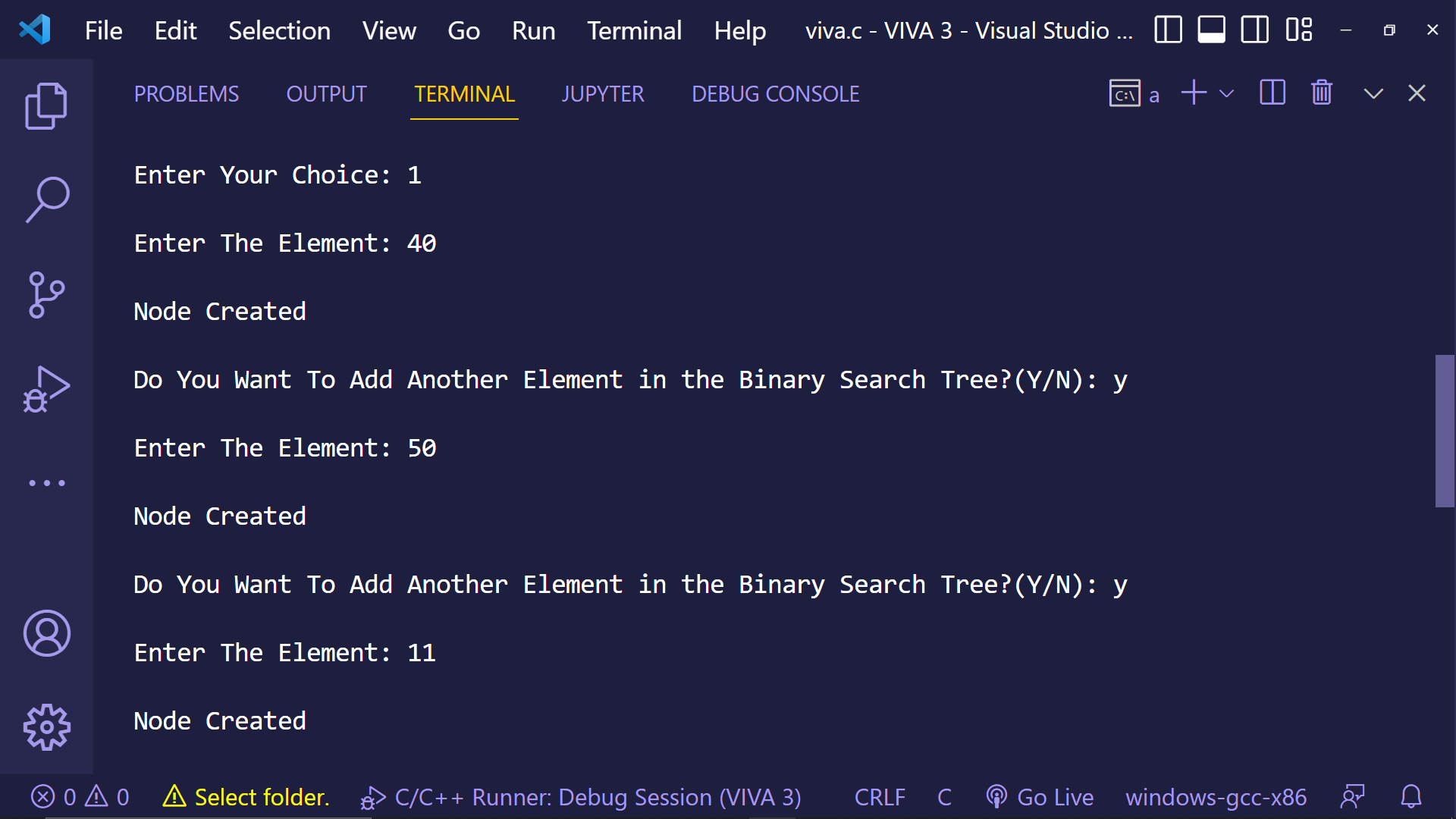
        righth = height(ptr->right);

    return (righth + 1);

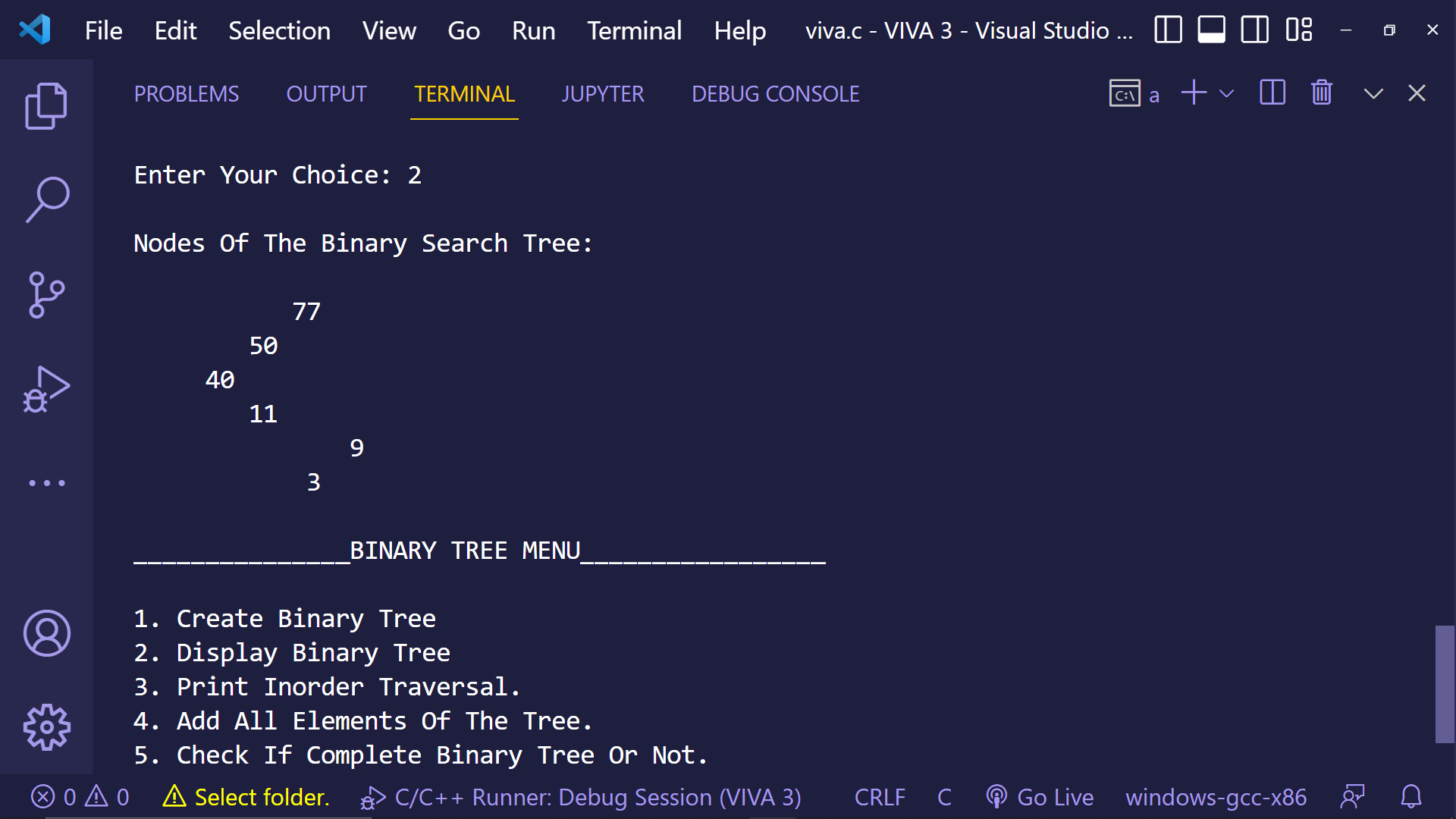
}

**OUTPUT SCREENSHOTS:**

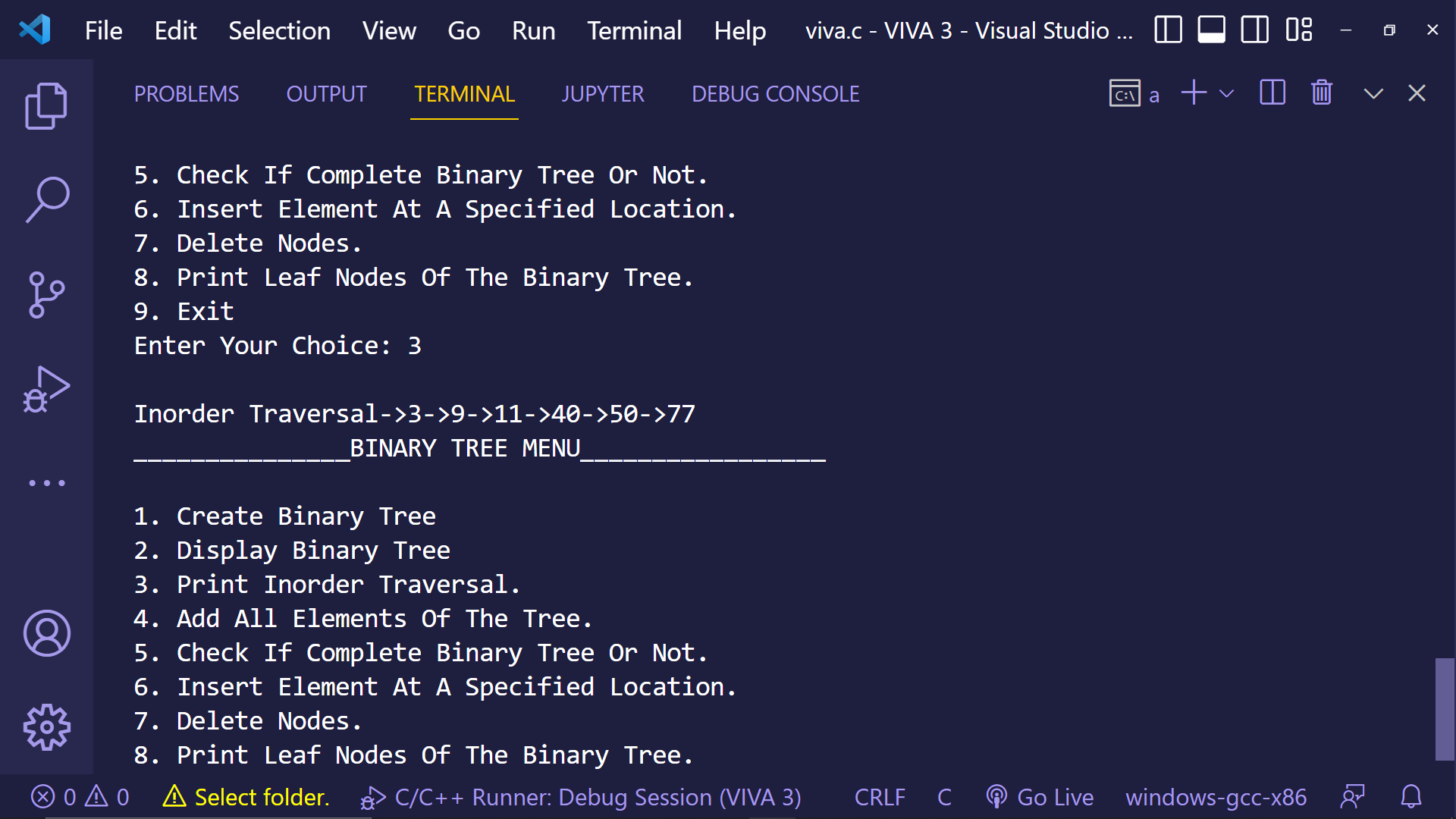
**Creating Tree.**



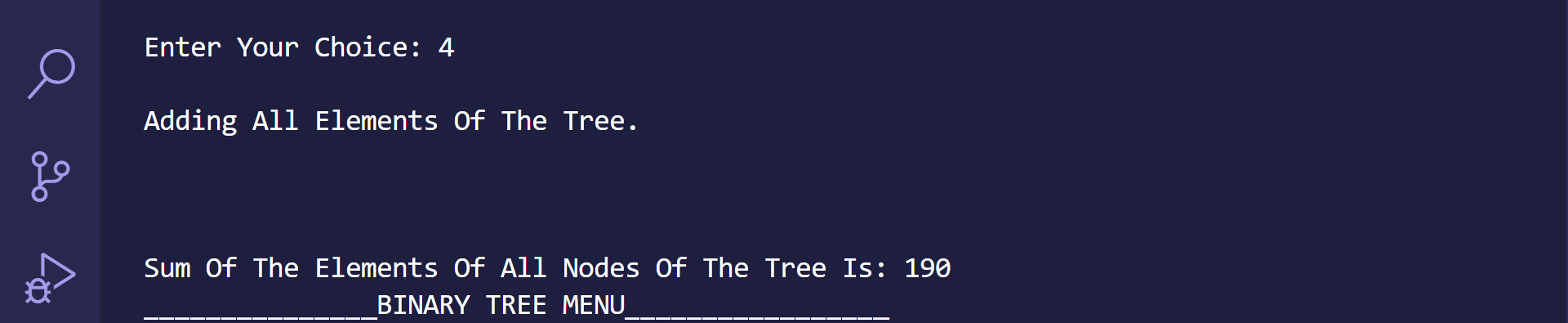
**Display Tree (Left Tilted Tree).**



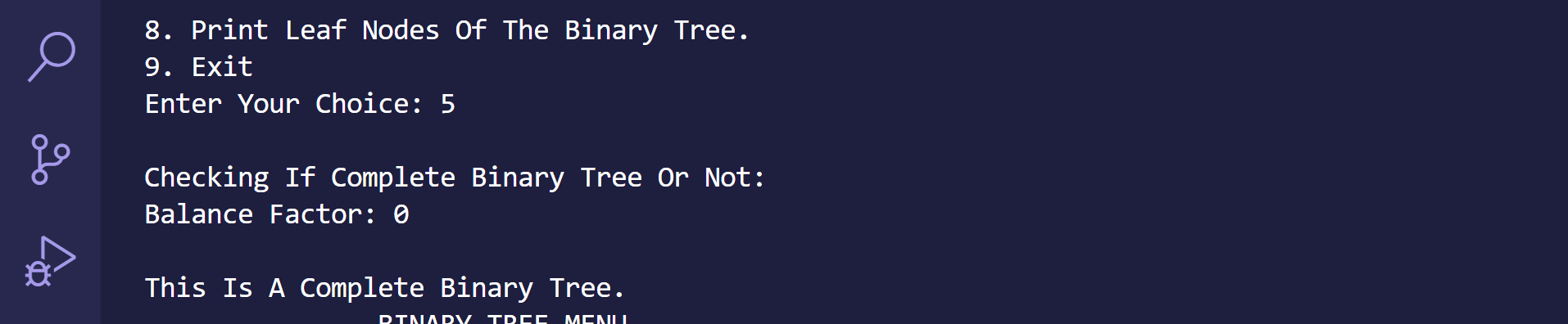
1. **Inorder Traversal Of The BST.**



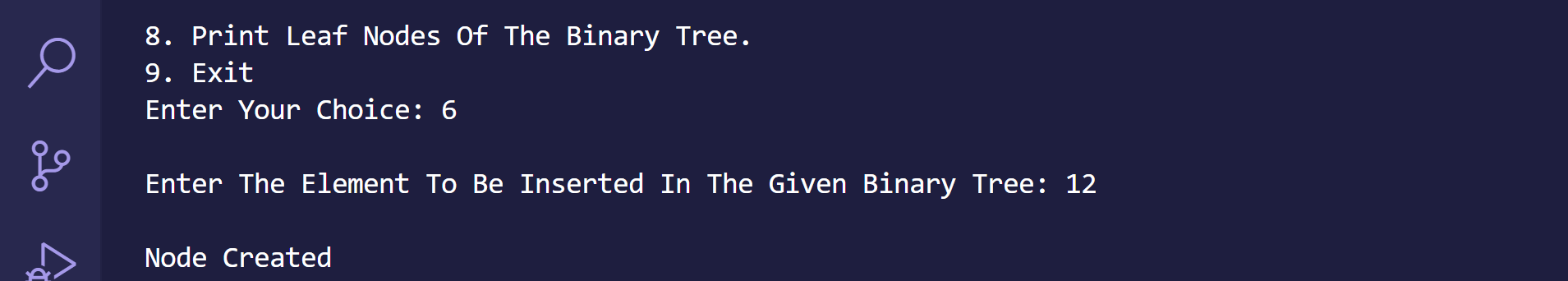
1. **Add All Elements Of The Nodes.**



1. **Check For Complete Binary Or Not.**



1. **Insert Element At Desired Location.**



1. **Delete Root Node.**



1. **Print Leaf Nodes Of The Tree.**

