**Viva 3**

**EXP. Write a C program**

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. \***

**Coding:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node\* left;

    struct node\* right;

};

struct node\* insert(struct node\* root, int data)

{

    if (root==NULL)

    {

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

       root->data = data;

       root->left = root->right = NULL;

    }

    else if (data < root->data)

       root->left = insert(root->left,data);

    else

       root->right = insert(root->right,data);

    return root;

}

int isBSTUtil(struct node\* root, int min, int max)

{

    if (root==NULL)

        return 1;

     if (root->data < min || root->data > max)

        return 0;

     return

         isBSTUtil(root->left, min, root->data-1) &&

         isBSTUtil(root->right, root->data+1, max);

}

int isBST(struct node\* root)

{

     return(isBSTUtil(root, INT\_MIN, INT\_MAX));

}

void printInorder(struct node\* root)

{

    if (root == NULL)

        return;

    printInorder(root->left);

    printf("%d \n", root->data);

    printInorder(root->right);

}

void deleteNode(struct node\* root,int data)

{

    struct node \*curr, \*parent, \*successor, \*temp ;

    curr = root;

    while(curr->data != data)

    {

        parent = curr;

        curr = (data < curr->data)? curr->left: curr->right;

        if(curr == NULL)

            return;

    }

    // node with no children

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

    {

        if(curr == root)

        {

            root = NULL;

        }

        else if(curr == parent->left)

        {

            parent->left = NULL;

        }

        else

        {

            parent->right = NULL;

        }

        free(curr);

    }

    // node with one child

    else if(curr->left == NULL || curr->right == NULL)

    {

        if(curr == root)

        {

            if(curr->left == NULL)

                root = curr->right;

            else

                root = curr->left;

            free(curr);

        }

        else if(curr == parent->left)

        {

            if(curr->left == NULL)

            {

                parent->left = curr->right;

            }

            else

            {

                parent->left = curr->left;

            }

            free(curr);

        }

        else

        {

            if(curr->left == NULL)

            {

                parent->right = curr->right;

            }

            else

            {

                parent->right = curr->left;

            }

            free(curr);

        }

    }

    // node with two children: get the inorder successor

    else

    {

        temp = curr->right;

        while(temp->left)

            temp = temp->left;

        successor = temp;

        deleteNode(root,successor->data);

        curr->data = successor->data;

    }

}

/\* Function to get the count of leaf nodes in a binary search tree\*/

unsigned int getLeafCount(struct node\* node)

{

  if(node == NULL)

    return 0;

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

    return 1;

  else

    return getLeafCount(node->left)+

           getLeafCount(node->right);

}

int main()

{

    struct node\* root=NULL;

    int data, choice;

    while(1)

    {

        printf("\n\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\n");

        printf("1. Insert\n");

        printf("2. Check BST\n");

        printf("3. Print Inorder\n");

        printf("4. Delete from tree\n");

        printf("5. Exit\n");

        printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

        printf("\nEnter your choice: ");

        scanf("%d", &choice);

        printf("Leaf count of the tree is %d\n", getLeafCount(root));

        switch(choice)

        {

        case 1:

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

            scanf("%d", &data);

            root = insert(root,data);

            break;

        case 2:

            if(isBST(root))

                printf("It is a BST\n");

            else

                printf("It is not a BST\n");

            break;

        case 3:

            printf("In-order traversal :\n");

            printInorder(root);

            break;

        case 4:

            printf("Enter the data to be deleted: ");

            scanf("%d", &data);

            deleteNode(root,data);

            break;

        case 5:

            exit(0);

            break;

        default:

            printf("Wrong choice\n");

        }

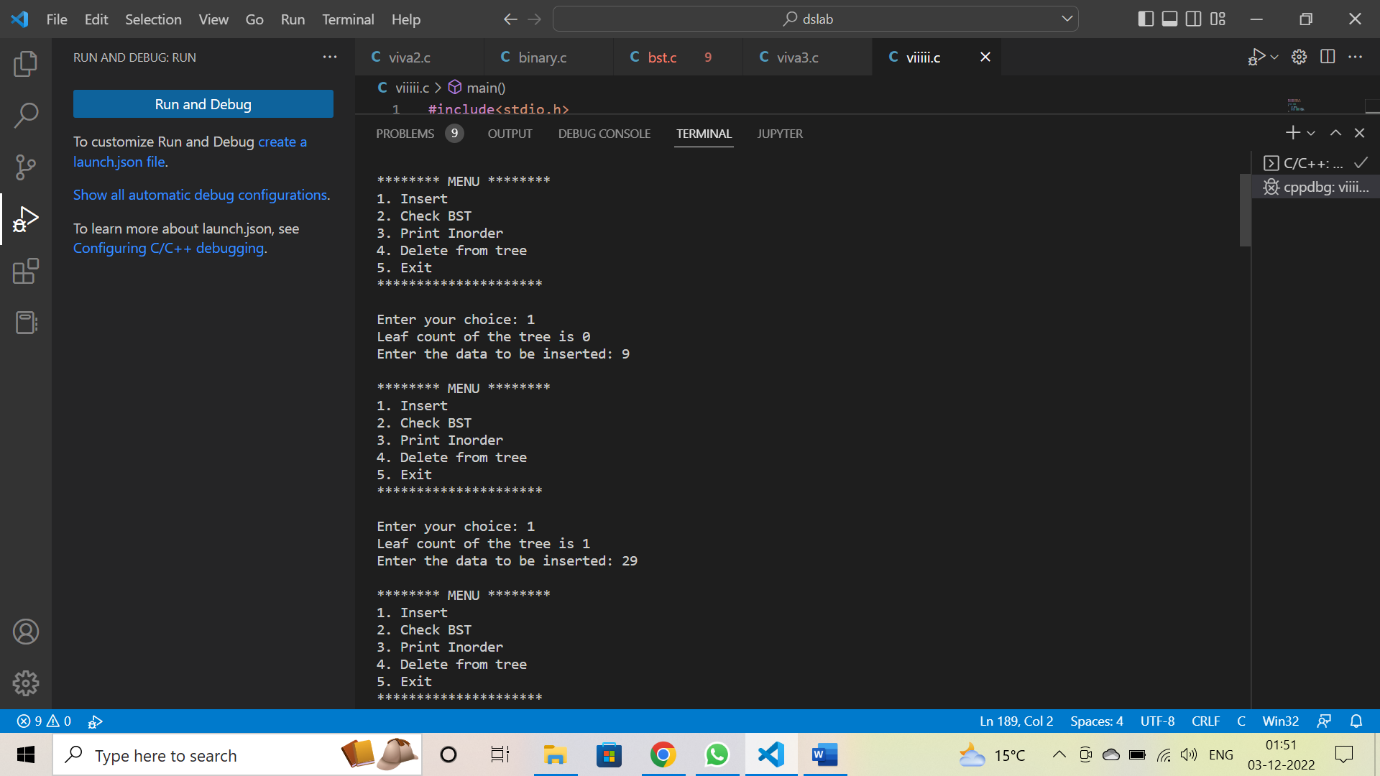
    }

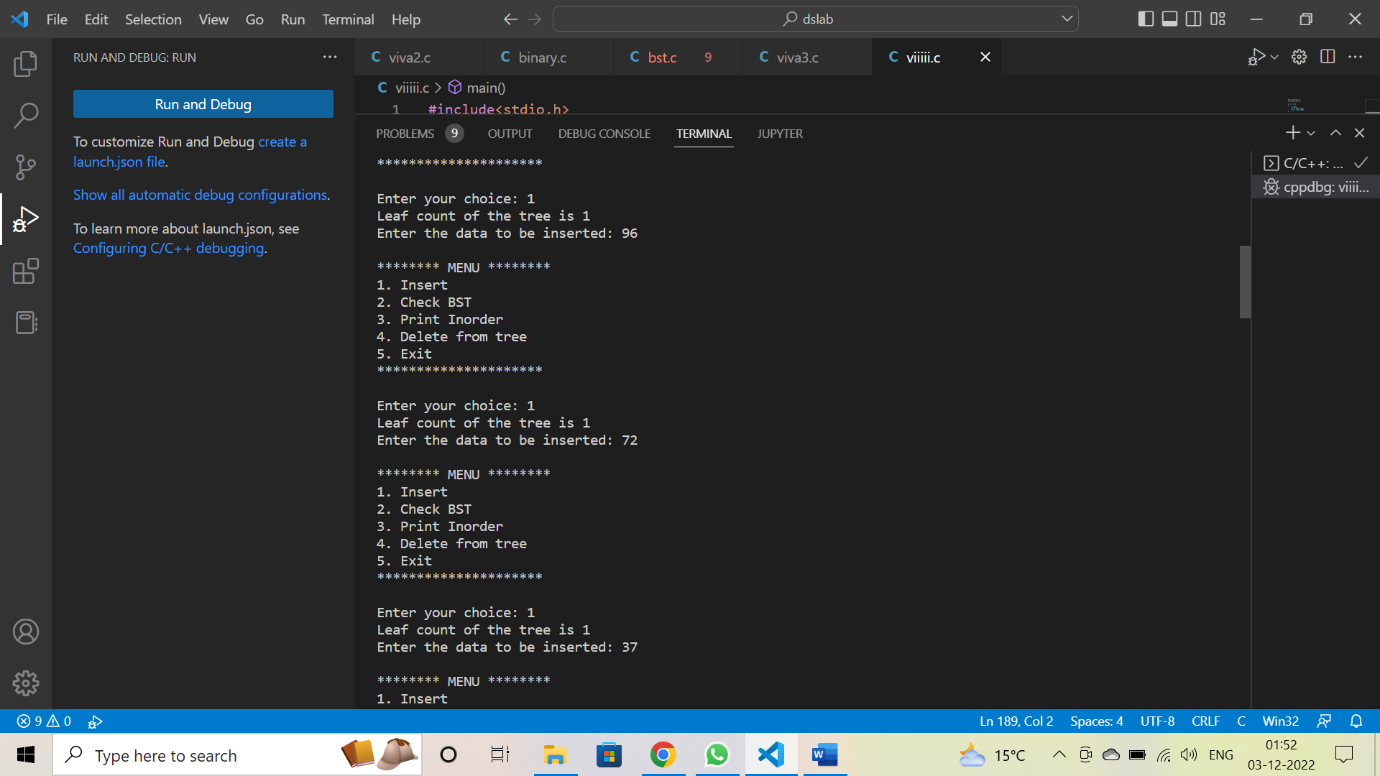
    return 0;

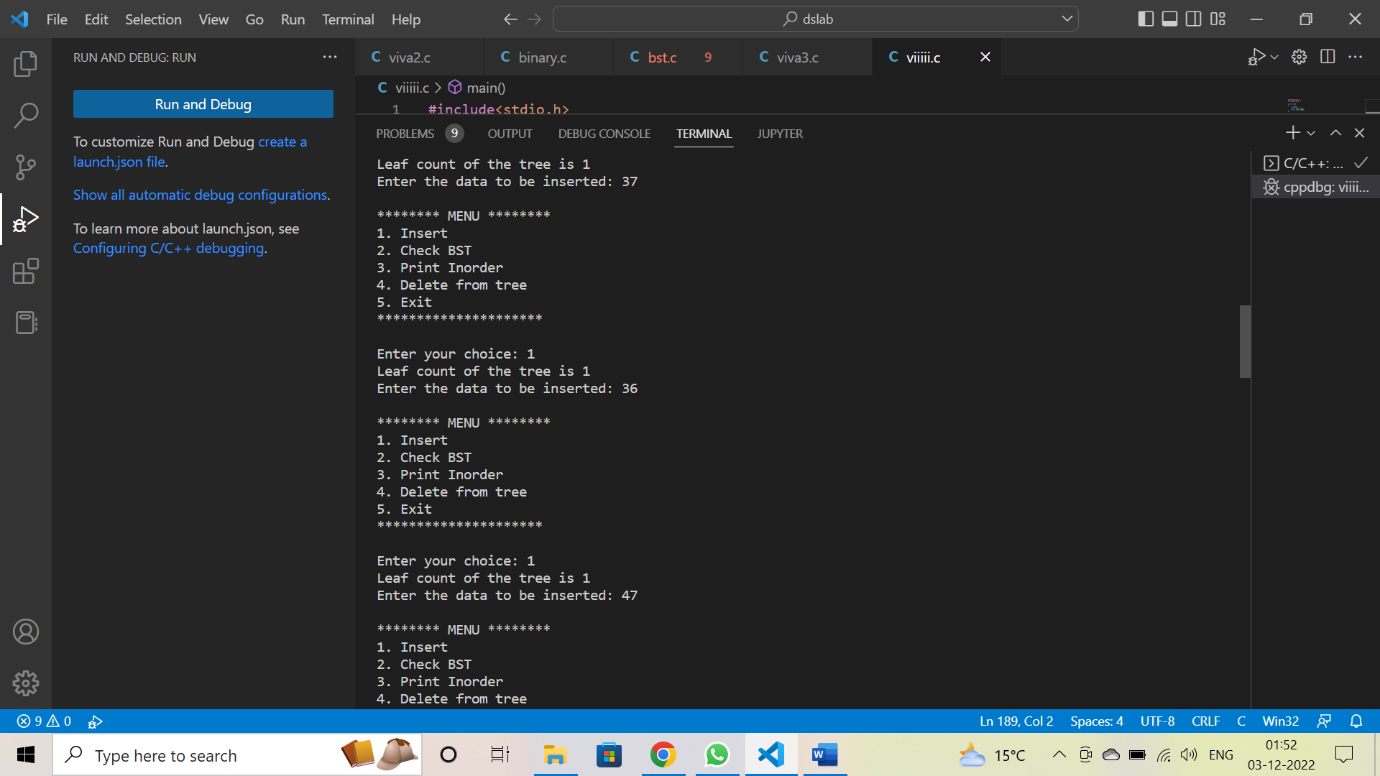
}

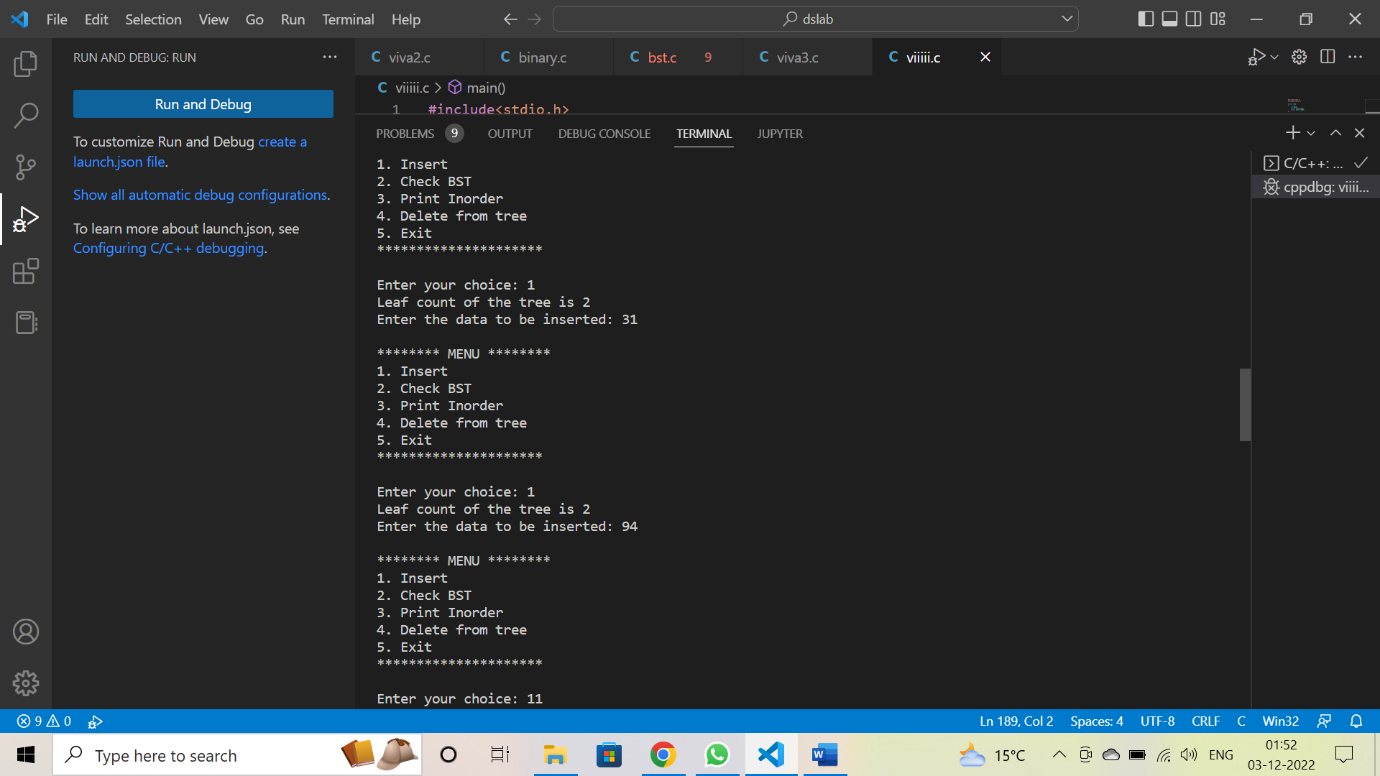
**Output:**

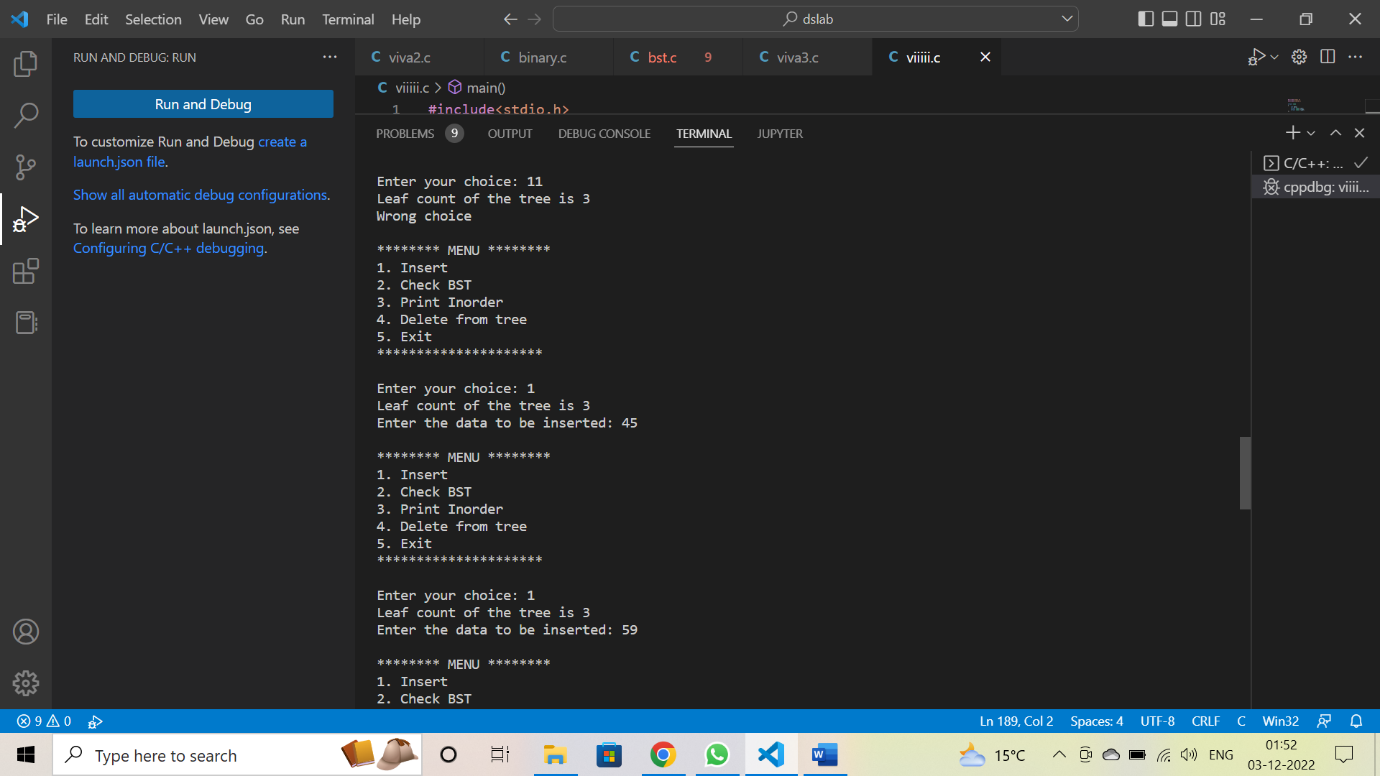
**IF BST**

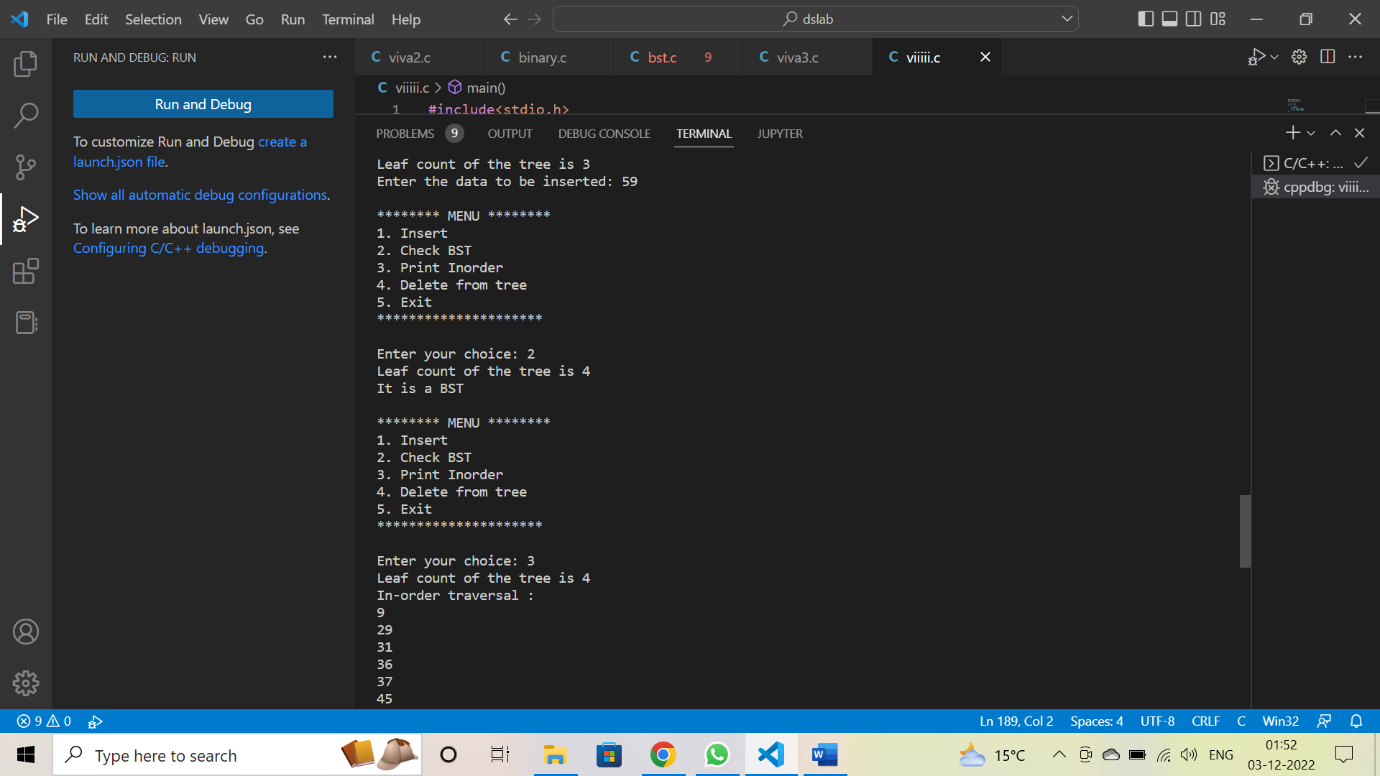


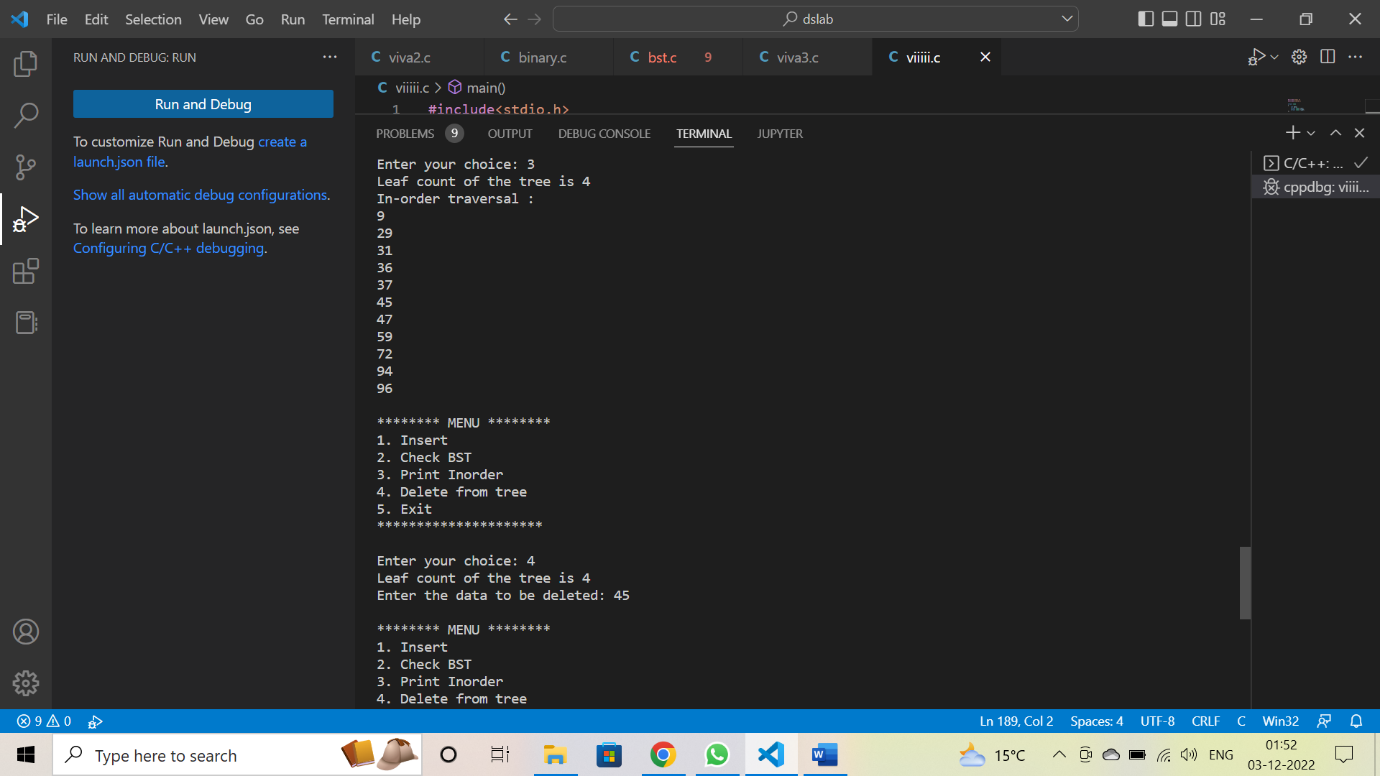


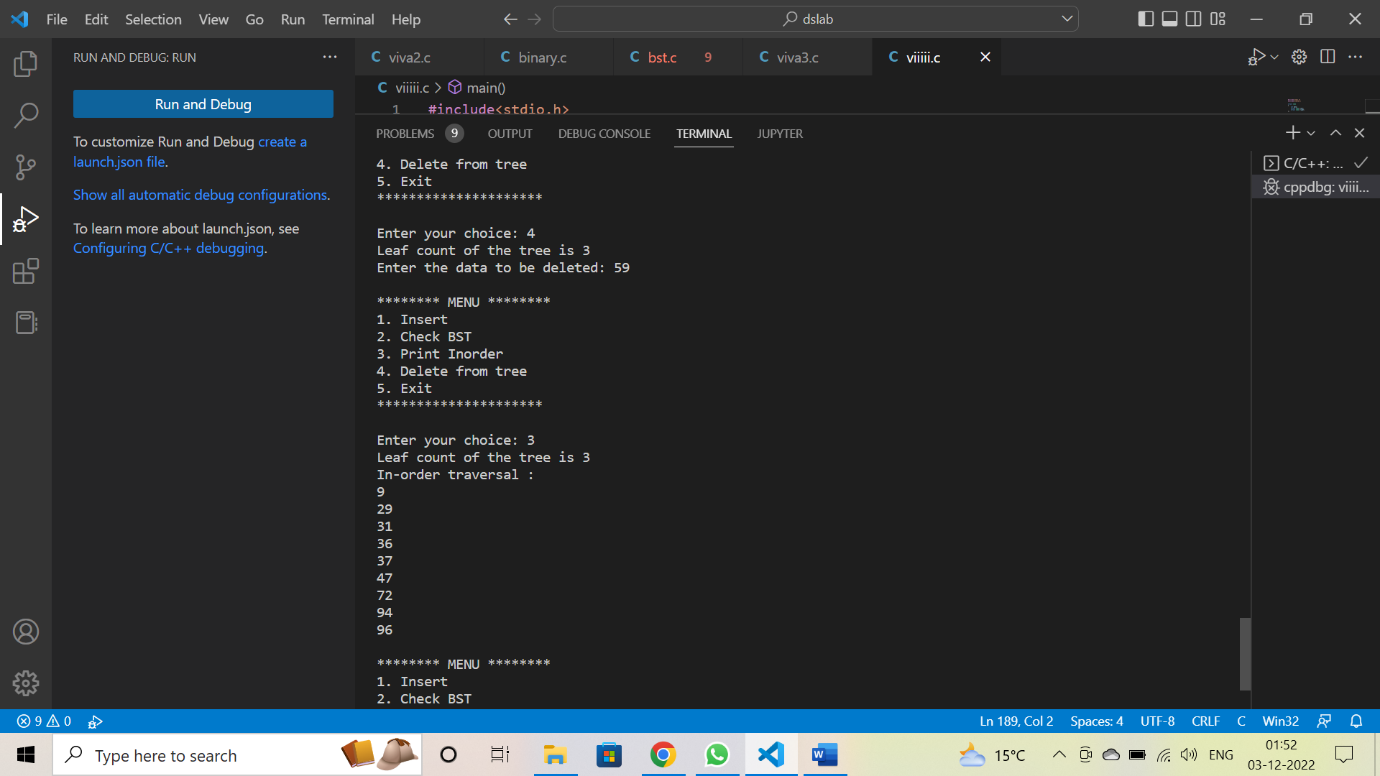


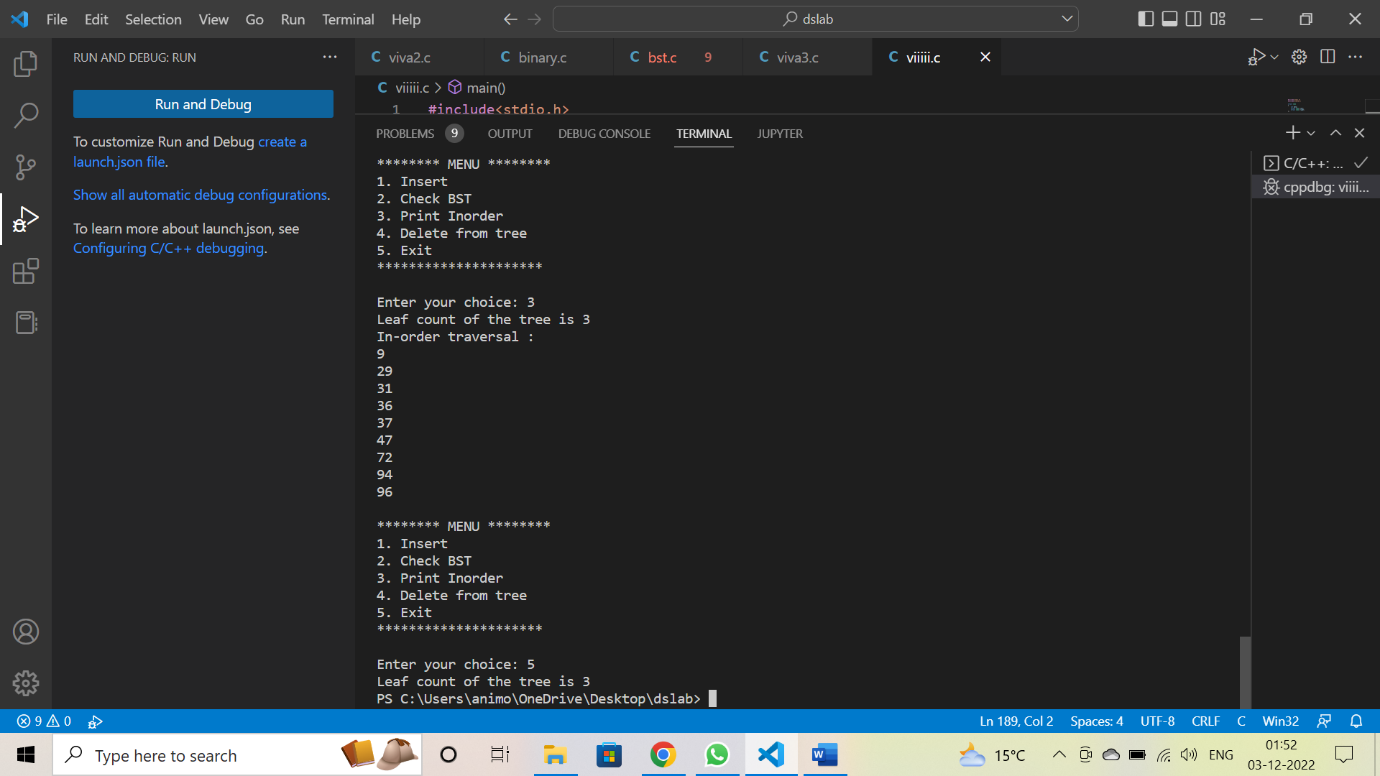












**IF IT is not a BST**

