Government College of Engineering, Karad Programming for Problem Solving Lab

Nanekar Saurabh Rajesh

20141212

I1

**Experiment No. 12**

**Title**: Implement of binary search tree and perform recursive and non-recursive in order, pre order and post order traversals.

**Outcome:** Students can perform tree traversals and its related applications.

**Theory:**

**Binary Search Tree** is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than the node’s key.
* The right subtree of a node contains only nodes with keys greater than the node’s key.
* The left and right subtree each must also be a binary search tree.

200px-Binary_search_tree.svg

**Tree Traversals (Inorder, Preorder and Postorder)**

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



**Depth First Traversals:**

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

**Inorder Traversal ([Practice](https://practice.geeksforgeeks.org/problems/inorder-traversal/1)):**

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(right-subtree)

**Uses of Inorder**   
In the case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal s reversed can be used.

Example: In order traversal for the above-given figure is 4 2 5 1 3.

**Preorder Traversal:**

Algorithm Preorder (tree)

1. Visit the root.

2. Traverse the left subtree, i.e., call Preorder(left-subtree)

3. Traverse the right subtree, i.e., call Preorder(right-subtree)

**Uses of Preorder**:   
Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on an expression tree.   
Example: Preorder traversal for the above-given figure is 1 2 4 5 3.

**Postorder Traversal:**

Algorithm Postorder (tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)

2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.

**Uses of Postorder**:  
Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree.

Example: Postorder traversal for the above-given figure is 4 5 2 3 1.

**Analysis:**



**List of similar programs: Solve any one.**

1. Write a C program for [Binary Tree to Binary Search Tree Conversion](https://www.geeksforgeeks.org/binary-tree-to-binary-search-tree-conversion/)
2. Write a C program to [Merge Two Balanced Binary Search Trees](https://www.geeksforgeeks.org/merge-two-balanced-binary-search-trees/)
3. Write a C program to [Convert BST to Min Heap](https://www.geeksforgeeks.org/convert-bst-min-heap/).

Refer <https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/> for more practice.

**Title Program:**Implement of binary search tree and perform recursive and non-recursive in order, pre order and post order traversals.

**Source code of Implemented Programs:**

//Nanekar Saurabh Rajesh\_20141212\_I1

#include <stdio.h>

#include <stdlib.h>

struct node {

        int data;

        struct node\* left;

        struct node\* right;

};

struct node\* newNode(int data)

{

        struct node\* node

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

        node->data = data;

        node->left = NULL;

        node->right = NULL;

        return (node);

}

void printPostorder(struct node\* node)

{

        if (node == NULL)

                return;

        printPostorder(node->left);

        printPostorder(node->right);

        printf("%d ", node->data);

}

void printInorder(struct node\* node)

{

        if (node == NULL)

                return;

        printInorder(node->left);

        printf("%d ", node->data);

        printInorder(node->right);

}

void printPreorder(struct node\* node)

{

        if (node == NULL)

                return;

        printf("%d ", node->data);

        printPreorder(node->left);

        printPreorder(node->right);

}

int main()

{

        struct node\* root = newNode(1);

        root->left = newNode(2);

        root->right = newNode(3);

        root->left->left = newNode(4);

        root->left->right = newNode(5);

        printf("\nPreorder traversal of binary tree is \n");

        printPreorder(root);

        printf("\nInorder traversal of binary tree is \n");

        printInorder(root);

        printf("\nPostorder traversal of binary tree is \n");

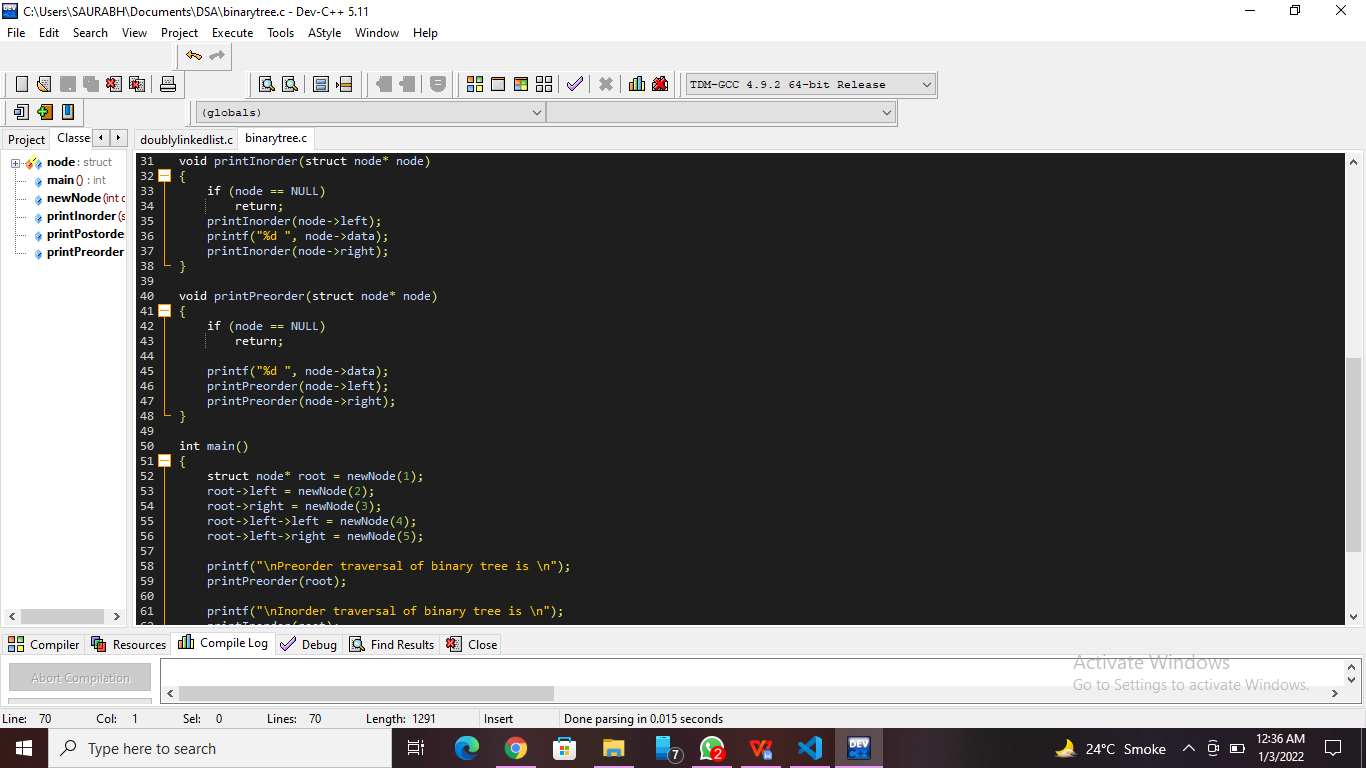
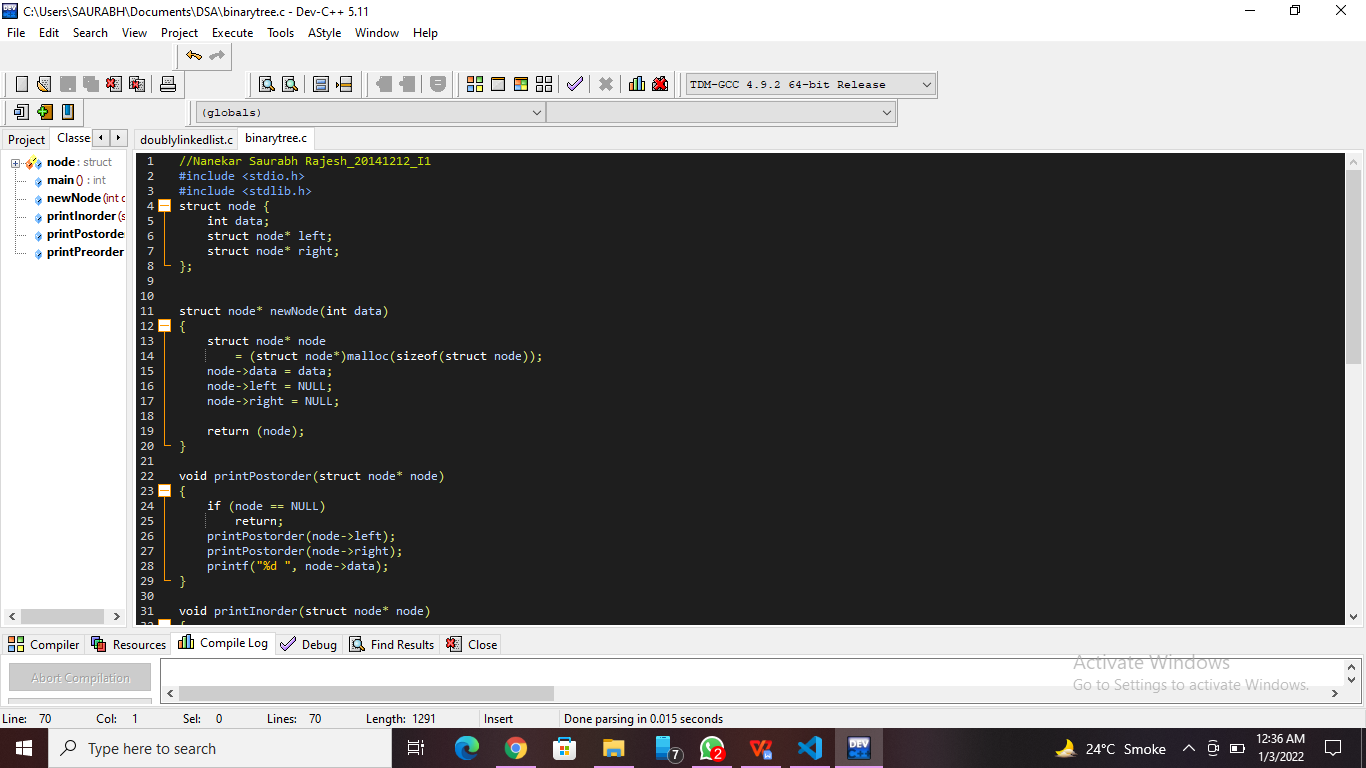
        printPostorder(root);

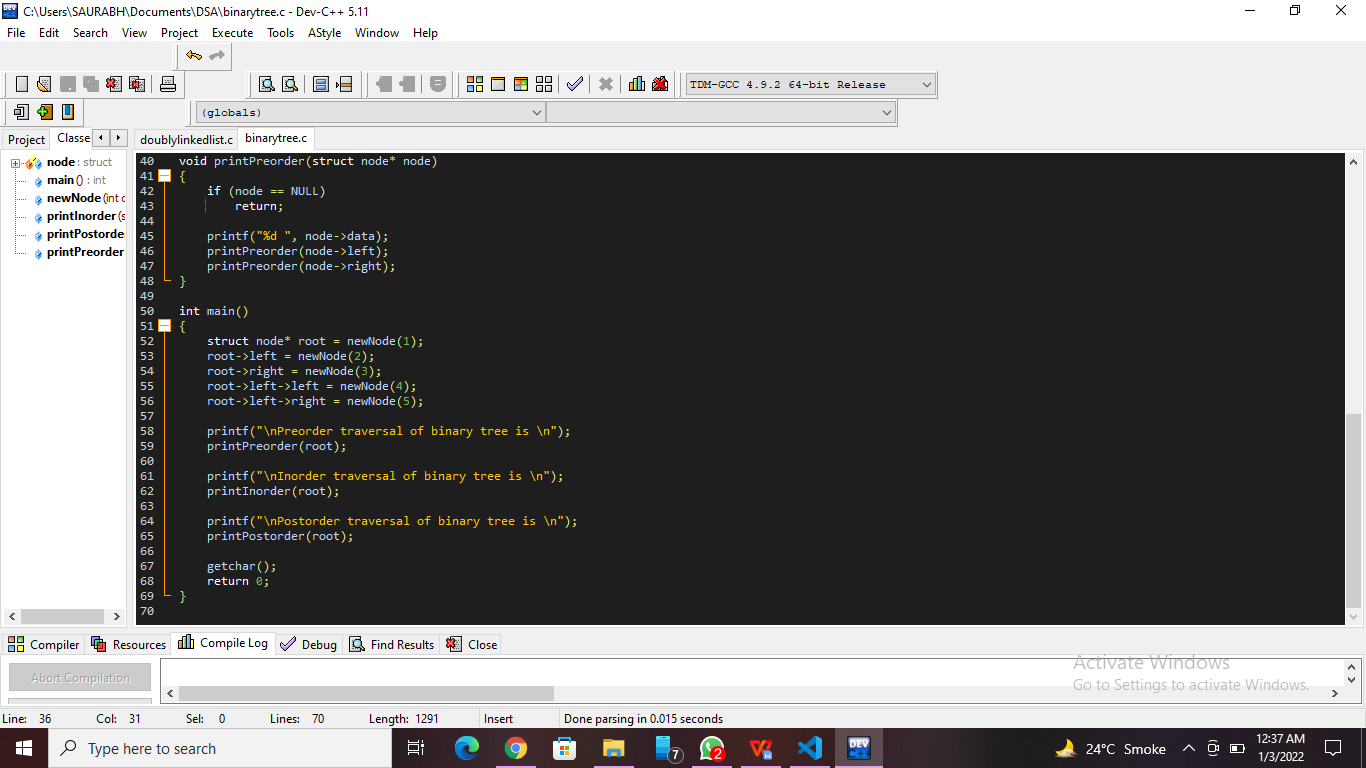
        getchar();

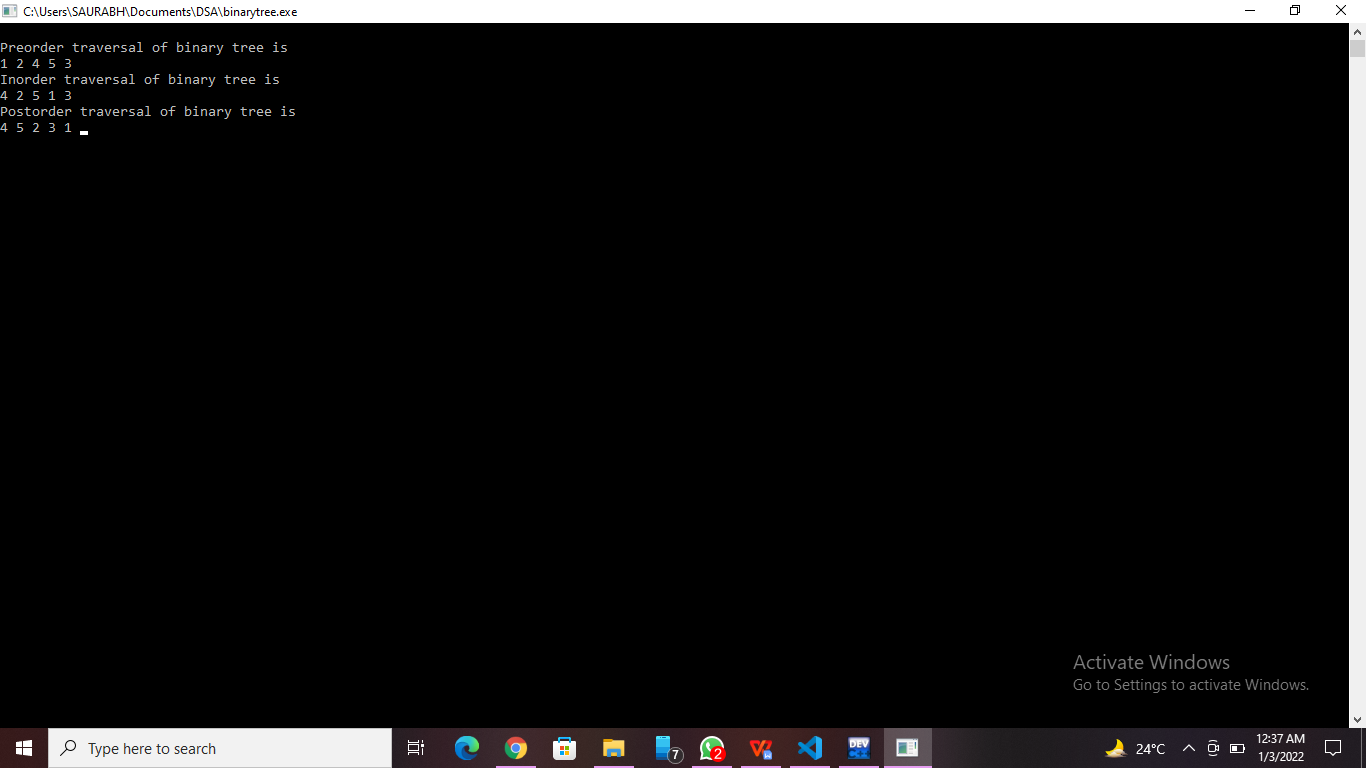
        return 0;

}

**Screenshots of Output:**

****

****

****

**Practice Program:**Write a C program for [Binary Tree to Binary Search Tree Conversion](https://www.geeksforgeeks.org/binary-tree-to-binary-search-tree-conversion/)

//Nanekar Saurabh Rajesh\_20141212\_I1

#include <stdio.h>

#include <stdlib.h>

struct node{

    int data;

    struct node \*left;

    struct node \*right;

};

struct node \*root = NULL;

int treeArray[100];

int ind = 0;

struct node\* createNode(int data)

{

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

    newNode->data= data;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

int calculateSize(struct node \*node)

{

    int size = 0;

    if (node == NULL)

     return 0;

    else {

        size = calculateSize (node->left) + calculateSize (node->right) + 1;

        return size;

    }

}

void convertBTtoArray(struct node \*node)

{

    if(root == NULL){

        printf("Tree is empty\n");

        return;

    }

    else

    {

        if(node->left != NULL)

            convertBTtoArray(node->left);

        treeArray[ind] = node->data;

        ind++;

        if(node->right!= NULL)

            convertBTtoArray(node->right);

    }

}

struct node\* createBST(int start, int end)

{

    if (start > end) {

        return NULL;

    }

    int mid = (start + end) / 2;

    struct node \*temp = createNode(treeArray[mid]);

    temp->left = createBST(start, mid - 1);

    temp->right = createBST(mid + 1, end);

    return temp;

}

struct node\* convertBTBST(struct node \*node)

{

    int treeSize = calculateSize(node);

    convertBTtoArray(node);

    int compare (const void \* a, const void \* b) {

        return ( \*(int\*)a - \*(int\*)b );

    }

    qsort(treeArray, treeSize, sizeof(int), compare);

    struct node \*d = createBST(0, treeSize - 1);

    return d;

}

void inorderTraversal(struct node \*node)

{

    if(root == NULL){

        printf("Tree is empty\n");

        return;

       }

    else {

        if(node->left!= NULL)

            inorderTraversal(node->left);

        printf("%d ", node->data);

        if(node->right!= NULL)

            inorderTraversal(node->right);

    }

}

int main()

{

    root = createNode(1);

    root->left = createNode(2);

    root->right = createNode(3);

    root->left->left = createNode(4);

    root->left->right = createNode(5);

    root->right->left = createNode(6);

    root->right->right = createNode(7);

    printf("Inorder representation of binary tree: \n");

    inorderTraversal(root);

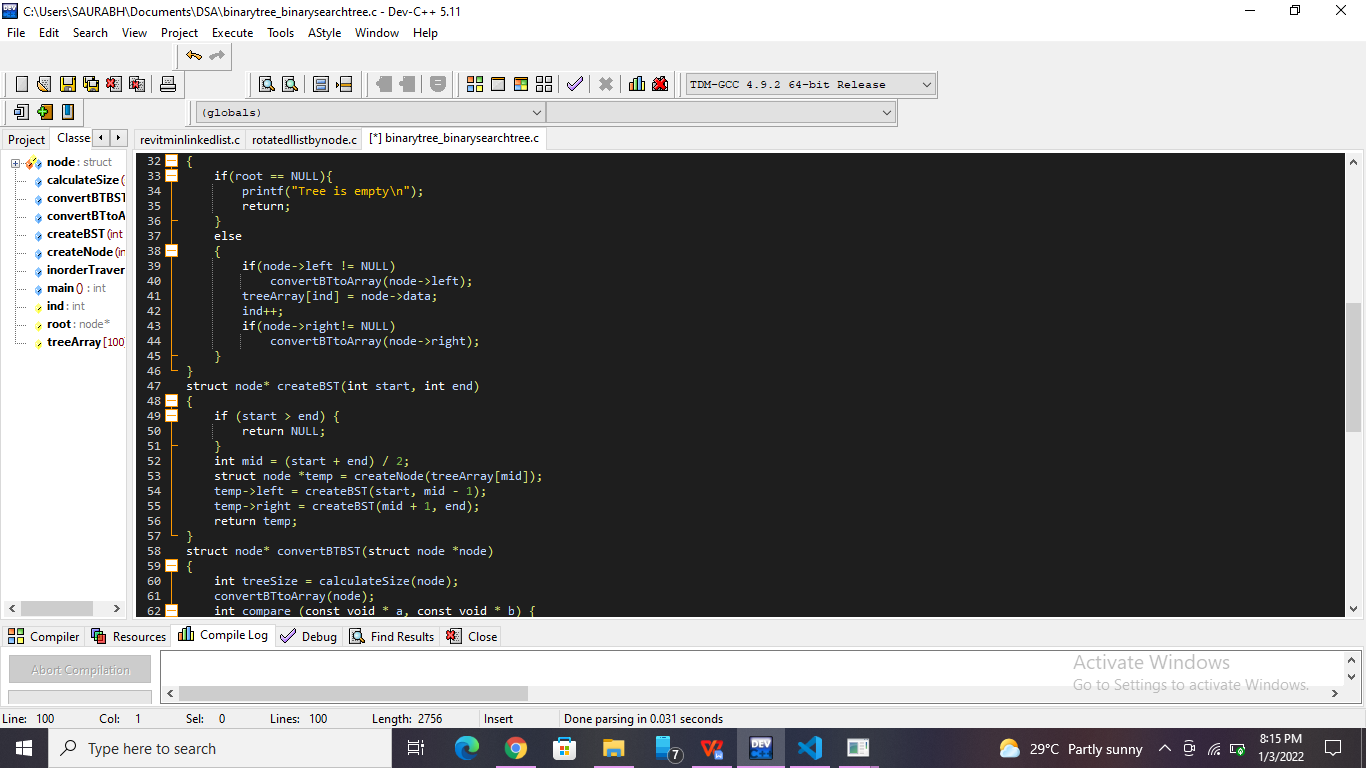
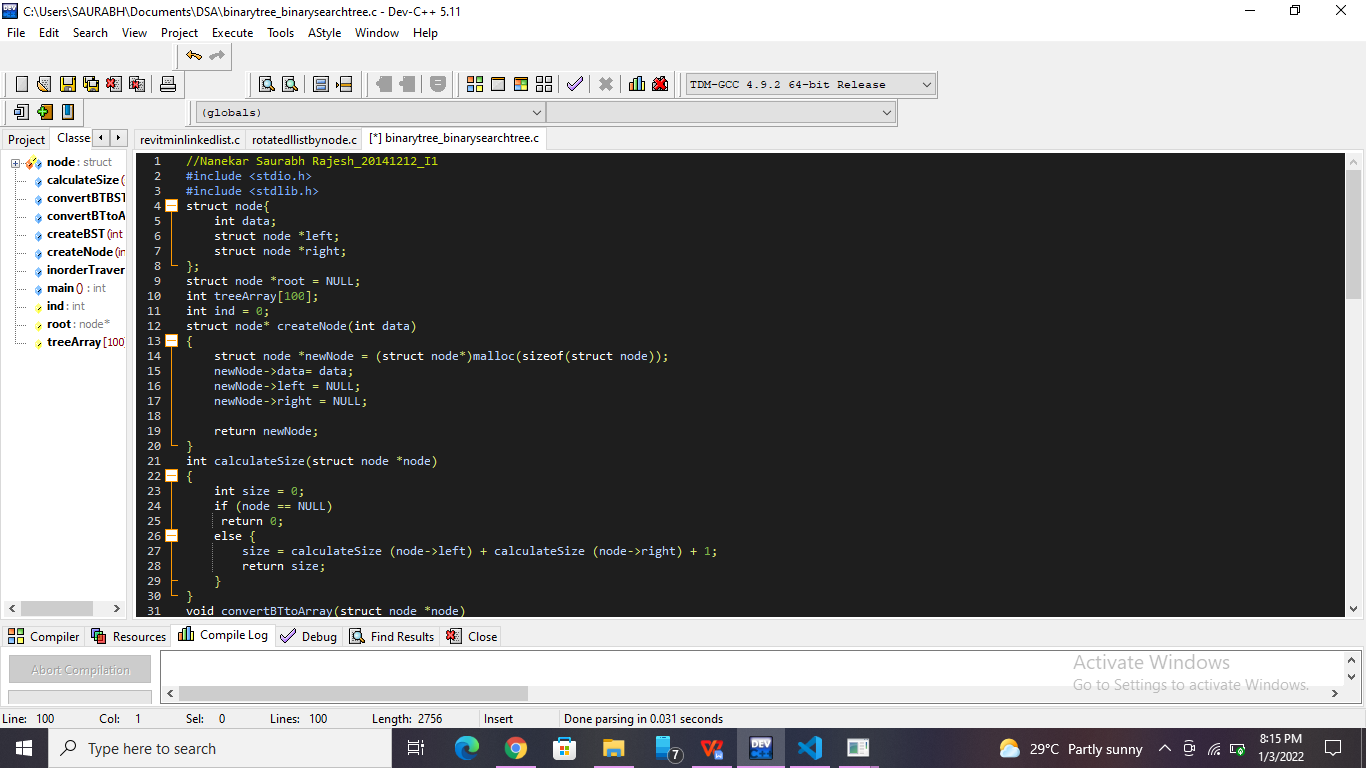
    struct node \*bst = convertBTBST(root);

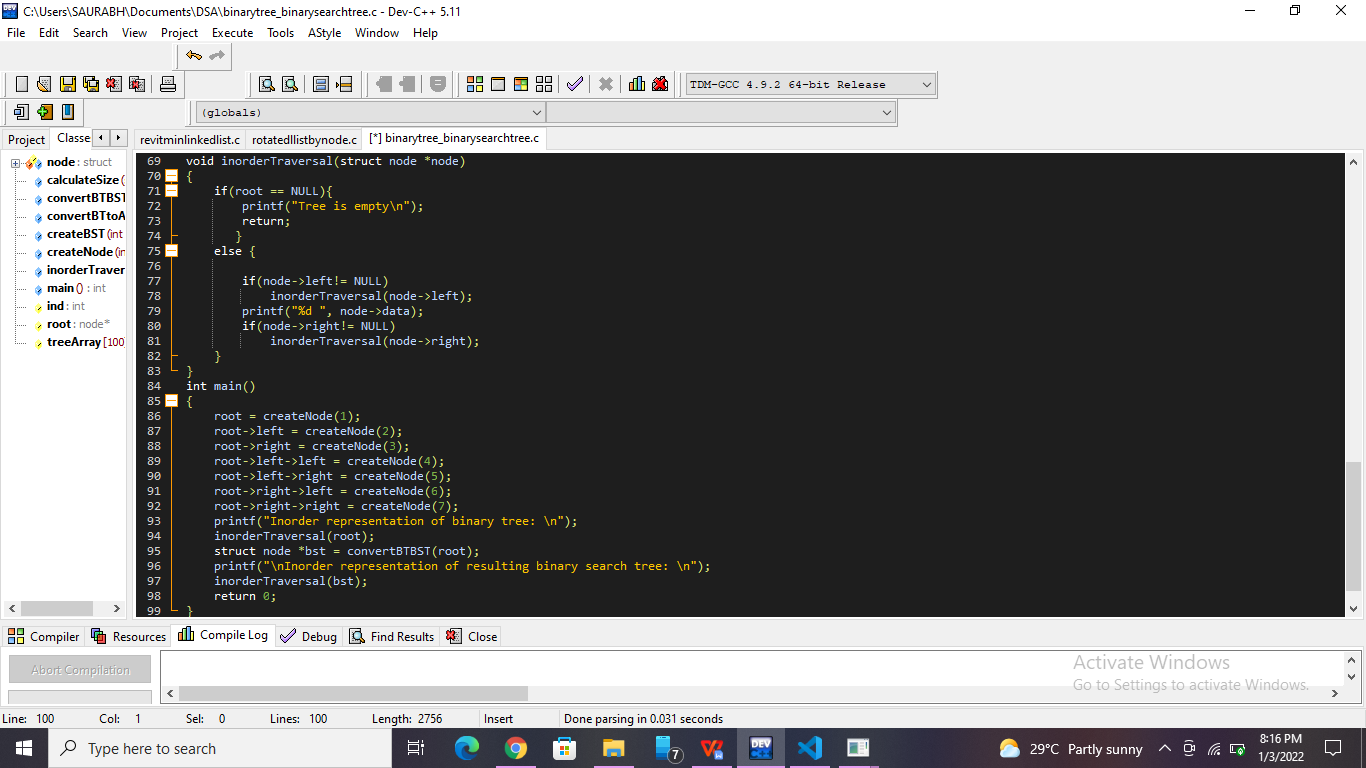
    printf("\nInorder representation of resulting binary search tree: \n");

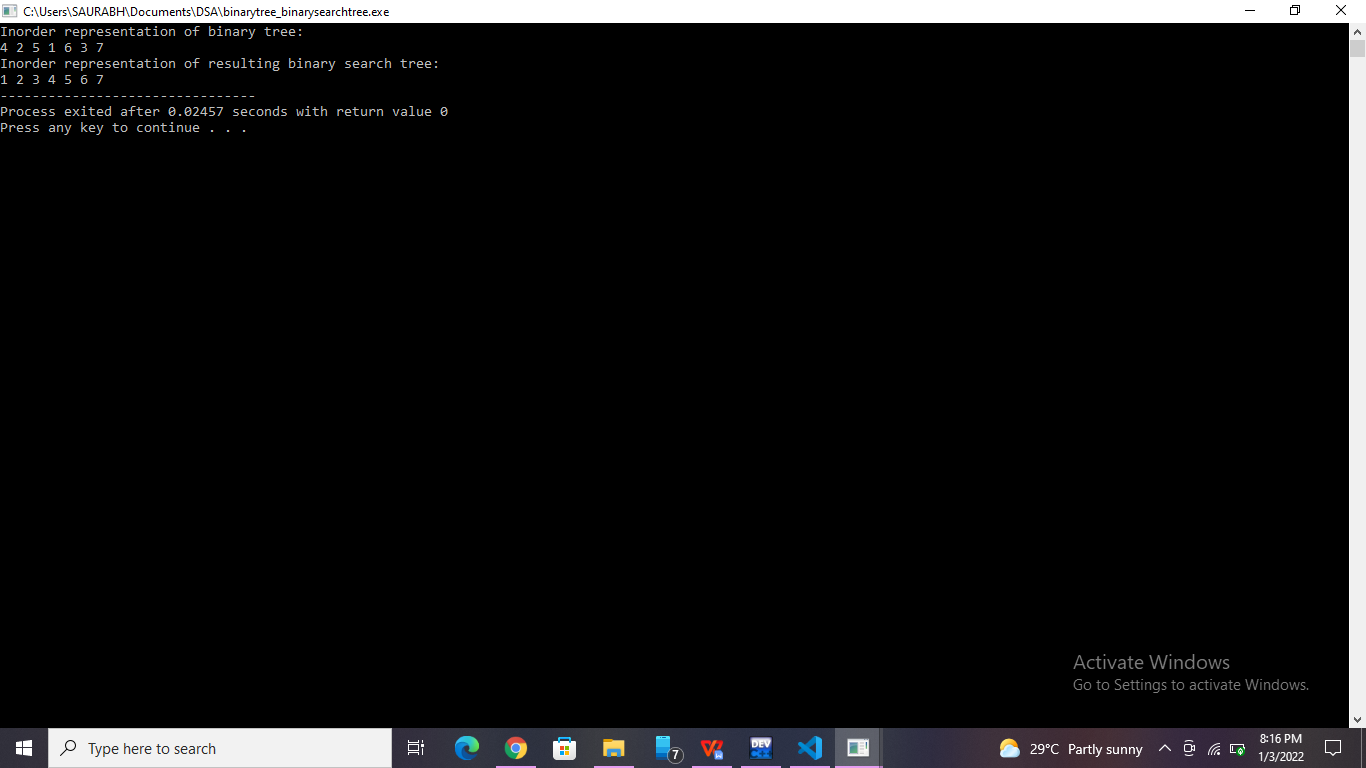
    inorderTraversal(bst);

    return 0;

}

**Screenshots of Practice Program:-**

****

****

**List of sample questions for oral examination:**

1. What is binary search tree?
2. What is a full binary tree?
3. How do you create a binary tree?
4. Is an empty tree a binary search tree?
5. What are differences between binary tree and binary search tree?

**Conclusion:**

Binary search trees are a very powerful data structure to have in your programming tool belt. If done right, handling large amounts of sorted data becomes easier and quicker.