Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Information Technology

Department of
Artificial Intelligence and Data Science

Name: Siddhesh Dilip Khairnar		
Class: SY-B tech	Division: B	Roll No: 272028
Semester: 3rd	Academic Year:2022-2023	
Subject Name & Code: ES21201AD: Discrete Mathematics		
Title of Assignment: program to calculate no of nodes, depth and height, no of nodes.		
Date of Performance: 28/11/2022	Date of Submission: 05/12/2022	

Problem Statement: Program to calculate no of nodes, depth and height, no of nodes at level n for binary tree.

Introduction to Tree Data Structure: This data structure is a specialized method to organize and store data in the computer to be used more effectively. It consists of a central node, structural nodes, and sub-nodes, which are connected via edges. We can also say that tree data structure has roots, branches, and leaves connected with one another.

The data in a tree are not stored in a sequential manner i.e., they are not stored linearly. Instead, they are arranged on multiple levels or we can say it is a hierarchical structure. For this reason, the tree is considered to be a non-linear data structure.

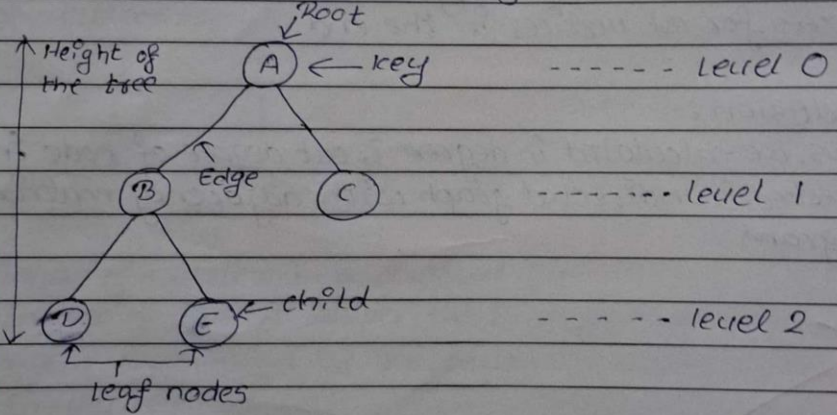
Tutorial No. 8

Problem Statement :-

Write a program to calculate no. of nodes ; depth & height, at level n for binary tree

Introduction to tree data structure :-

A tree is non linear data structure & a hierarchy consisting of a collection of nodes such that each node of the tree stores a values & a list of reference to other nodes. This data structure is a specialized method to organize & shore data in the computer to be used more effectively. It consists of a central node, structural nodes & sub-nodes, which are connected via edges.



Basic terminologies :-

* Parent Node : The node which is a predecessor of a node is called the parent node.

* Child Node : It is the immediate successor of a node.

* Root Node : The topmost node of a tree or the node which does not have any parent node. A non-empty tree must contain exactly one root node & exactly one path from the root to all other nodes of the tree.

* Leaf node : The nodes which do not have any child nodes.

* Ancestor of node : Any predecessor nodes on the path of the root to that node

* Descendant : Any successor node on the path from the leaf node to that node.

* Sibling : Children of the same parent are called siblings

* Level of node : The count of edges on the path from the root node to that node. The root node has level 0.

Properties of a tree :-

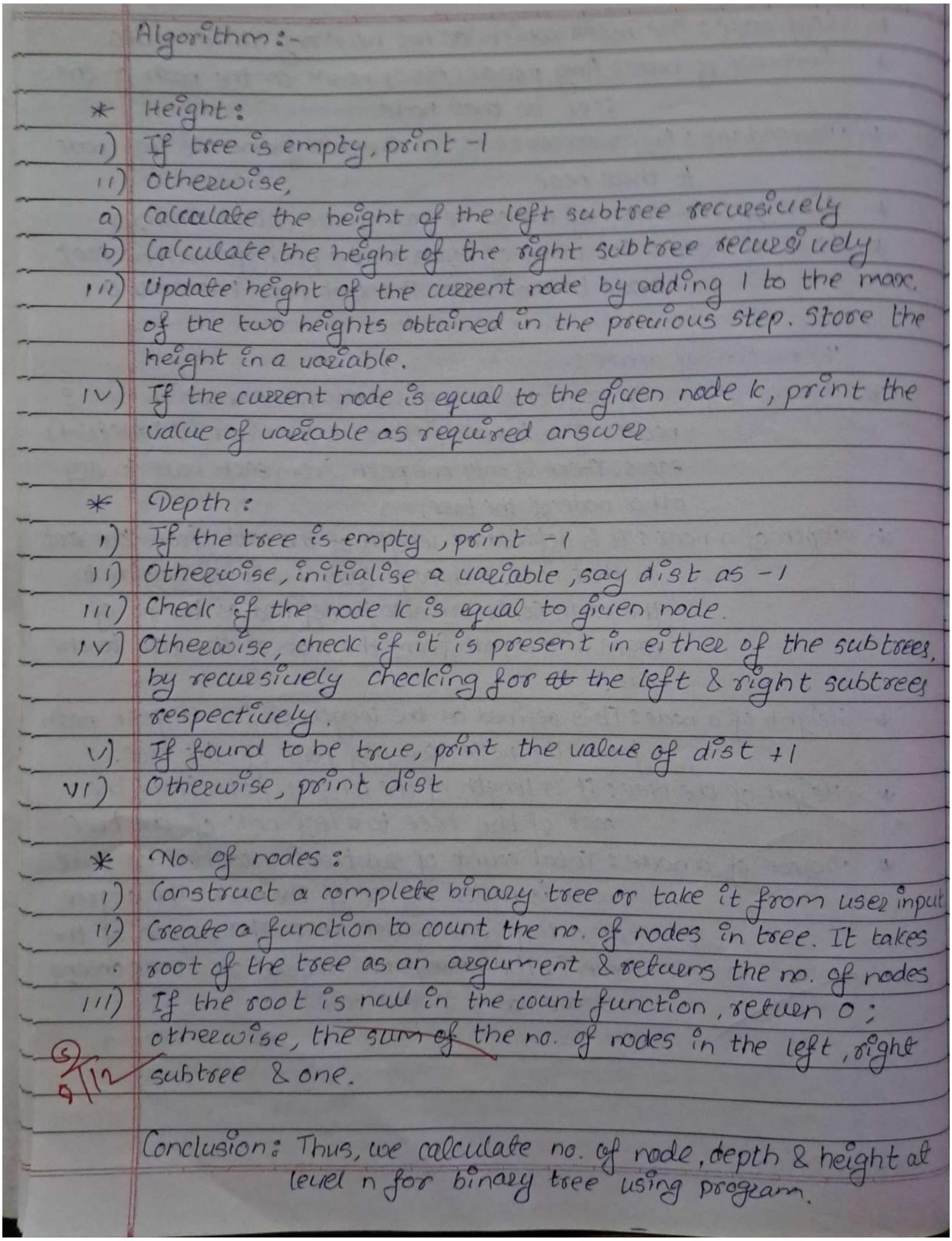
* No. of edges : An edge can be defined as the connection b/w 2 nodes. If a tree has N nodes then it will have (N-1) edges. There is only one path from each node to any other node of the tree.

* Depth of a node : It is defined as length of the path from the root to that node. Each edge adds 1 unit of length to the path. So, it can also be defined as the no. of edges in the path from the root of the tree to the node.

* Height of a node : It is defined as the length of the longest path from the node to a leaf node of the tree.

* Height of the tree : It is length of the longest path from the root of the tree to a leaf node of the tree.

* Degree of a node : Total count of subtrees attached to that node is called degree of the node. The degree of a leaf node must be 0. The degree of the tree is the maximum degree of a node among all the nodes in the tree.



Algorithm:-

* Height:

i) If tree is empty, print -1

ii) Otherwise,

a) Calculate the height of the left subtree recursively

b) Calculate the height of the right subtree recursively

iii) Update height of the current node by adding 1 to the max. of the two heights obtained in the previous step. Store the height in a variable.

iv) If the current node is equal to the given node k, print the value of variable as required answer

* Depth:

i) If the tree is empty, print -1

ii) Otherwise, initialise a variable, say dist as -1

iii) Check if the node k is equal to given node.

iv) Otherwise, check if it is present in either of the subtrees, by recursively checking for ~~at~~ the left & right subtrees respectively.

v) If found to be true, print the value of dist +1

vi) Otherwise, print dist

* No. of nodes:

i) Construct a complete binary tree or take it from user input

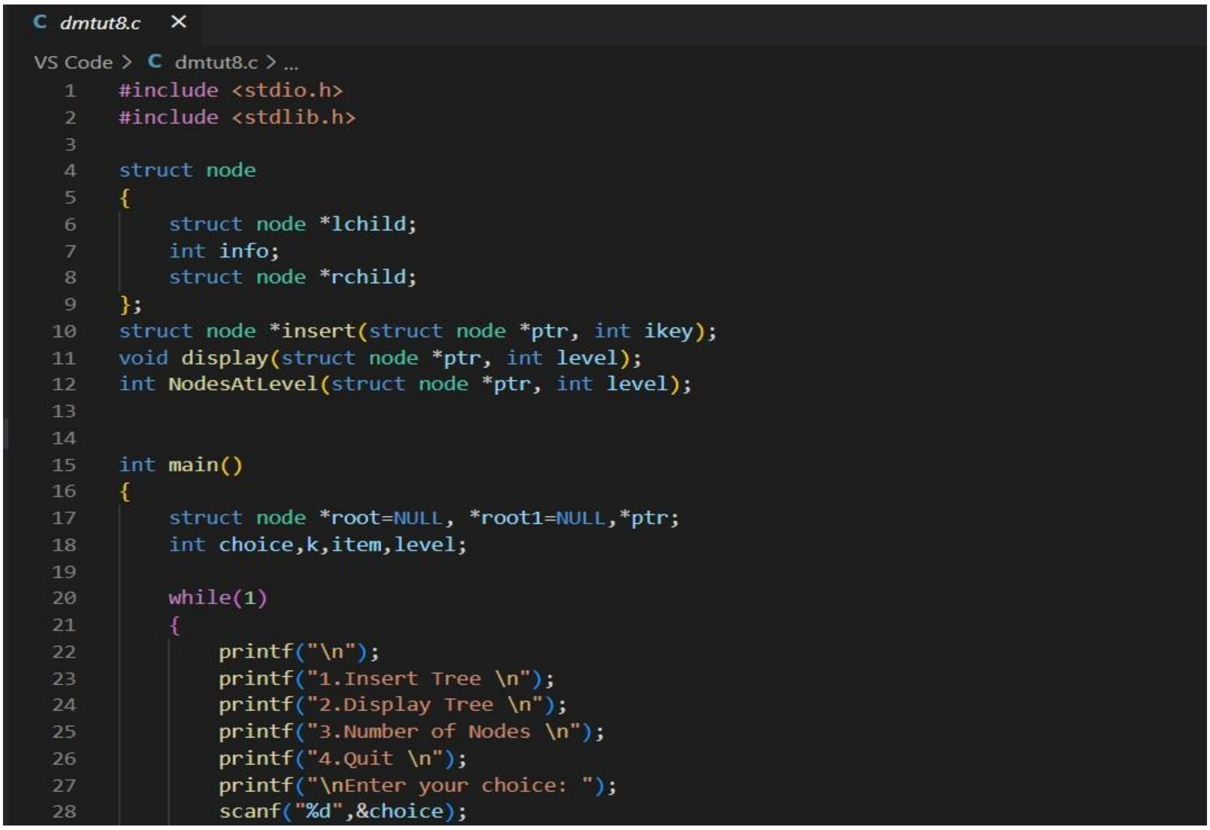
ii) Create a function to count the no. of nodes in tree. It takes root of the tree as an argument & returns the no. of nodes

iii) If the root is null in the count function, return 0; otherwise, the ~~sum of~~ the no. of nodes in the left, right subtree & one.

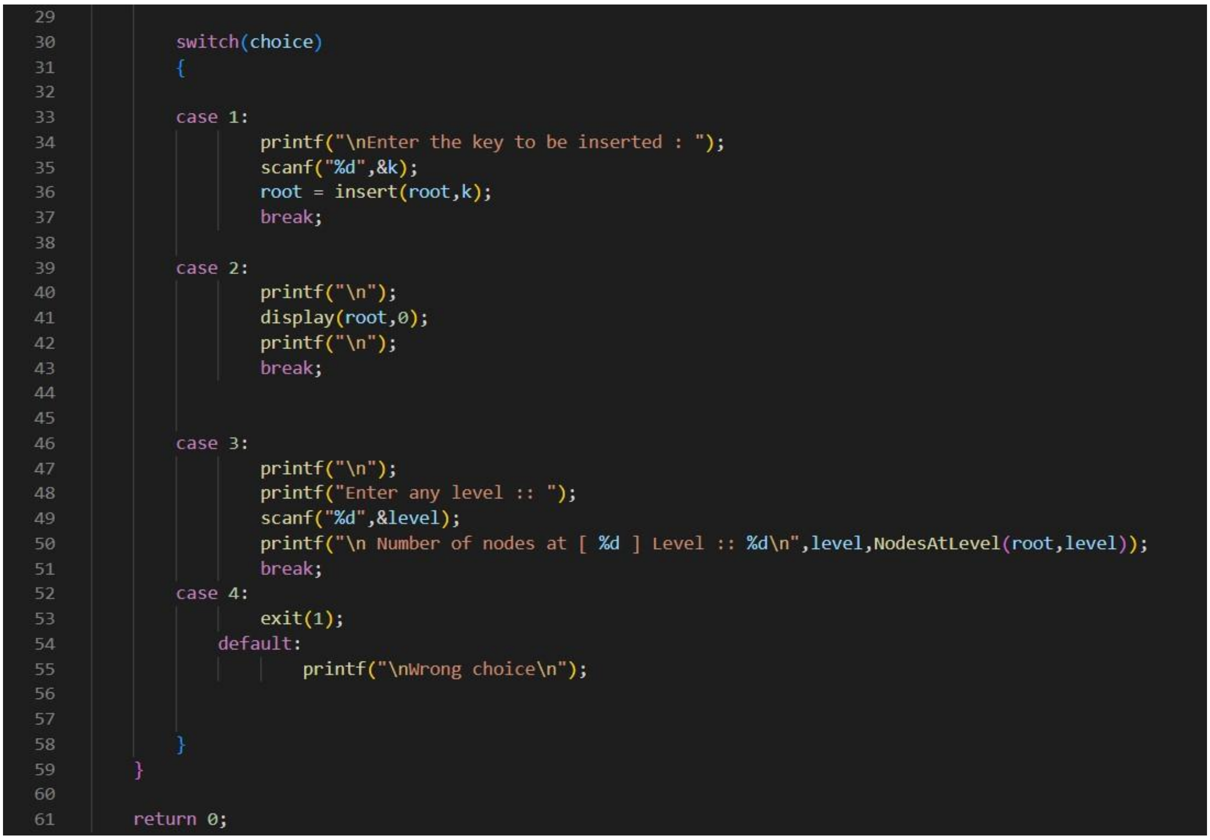
5
9/12

Conclusion: Thus, we calculate no. of node, depth & height at level n for binary tree using program.

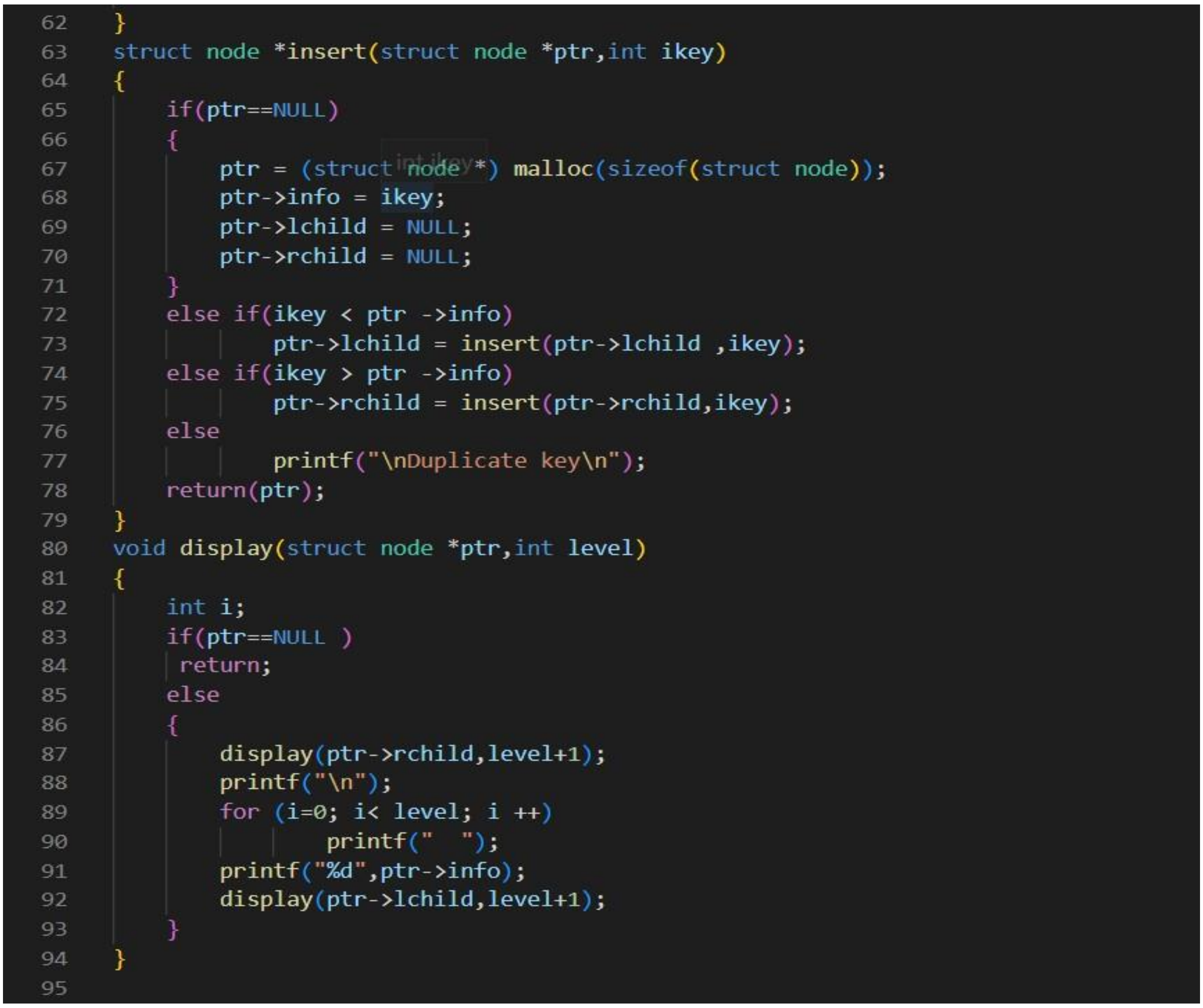
Program Input:



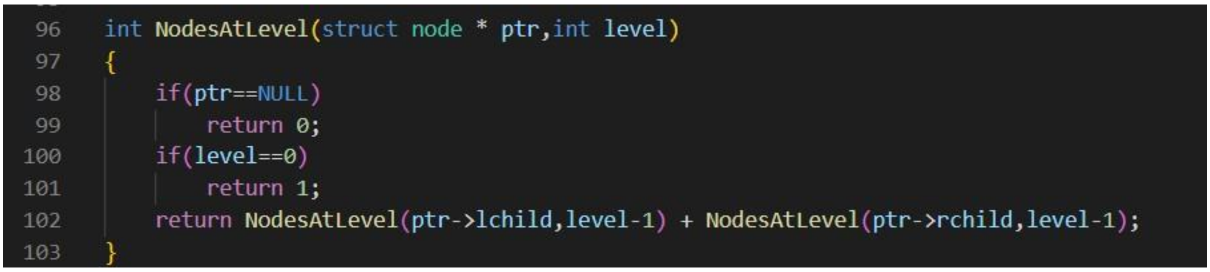
```c
#include <stdio.h>
#include <stdlib.h>

struct node
{
    struct node *lchild;
    int info;
    struct node *rchild;
};
struct node *insert(struct node *ptr, int ikey);
void display(struct node *ptr, int level);
int NodesAtLevel(struct node *ptr, int level);


int main()
{
    struct node *root=NULL, *root1=NULL,*ptr;
    int choice,k,item,level;

    while(1)
    {
        printf("\n");
        printf("1.Insert Tree \n");
        printf("2.Display Tree \n");
        printf("3.Number of Nodes \n");
        printf("4.Quit \n");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
```



```c

        switch(choice)
        {

        case 1:
                printf("\nEnter the key to be inserted : ");
                scanf("%d",&k);
                root = insert(root,k);
                break;

        case 2:
                printf("\n");
                display(root,0);
                printf("\n");
                break;


        case 3:
                printf("\n");
                printf("Enter any level :: ");
                scanf("%d",&level);
                printf("\n Number of nodes at [ %d ] Level :: %d\n",level,NodesAtLevel(root,level));
                break;
        case 4:
                exit(1);
            default:
                    printf("\nWrong choice\n");


        }
    }

    return 0;
```

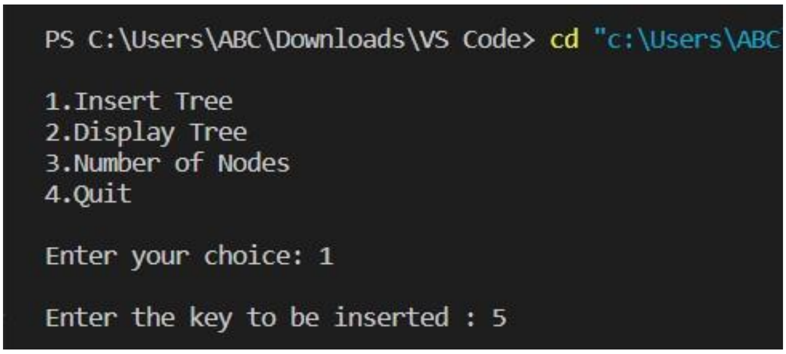


```c
}
struct node *insert(struct node *ptr,int ikey)
{
    if(ptr==NULL)
    {
        ptr = (struct node *) malloc(sizeof(struct node));
        ptr->info = ikey;
        ptr->lchild = NULL;
        ptr->rchild = NULL;
    }
    else if(ikey < ptr ->info)
            ptr->lchild = insert(ptr->lchild ,ikey);
    else if(ikey > ptr ->info)
            ptr->rchild = insert(ptr->rchild,ikey);
    else
            printf("\nDuplicate key\n");
    return(ptr);
}
void display(struct node *ptr,int level)
{
    int i;
    if(ptr==NULL )
     return;
    else
    {
        display(ptr->rchild,level+1);
        printf("\n");
        for (i=0; i< level; i ++)
                printf("  ");
        printf("%d",ptr->info);
        display(ptr->lchild,level+1);
    }
}

```

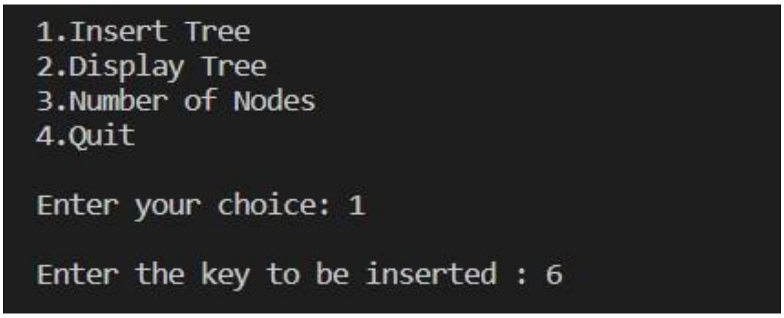


```c
int NodesAtLevel(struct node * ptr,int level)
{
    if(ptr==NULL)
        return 0;
    if(level==0)
        return 1;
    return NodesAtLevel(ptr->lchild,level-1) + NodesAtLevel(ptr->rchild,level-1);
}
```

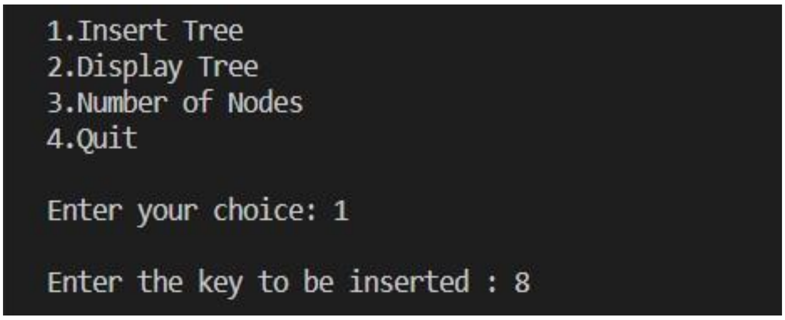
Program Output:



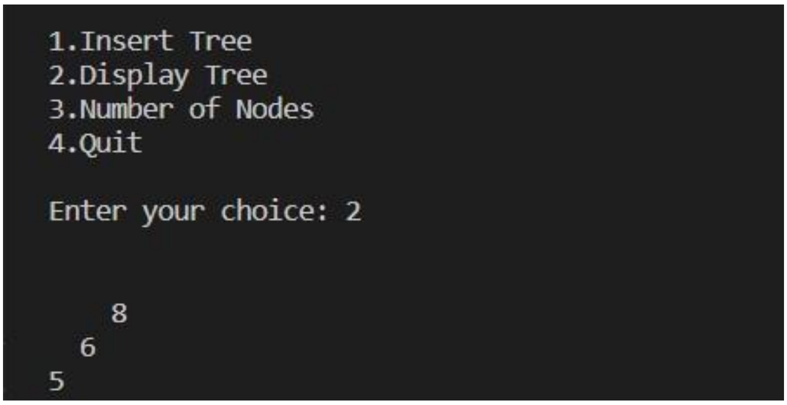
```
PS C:\Users\ABC\Downloads\VS Code> cd "c:\Users\ABC

1.Insert Tree
2.Display Tree
3.Number of Nodes
4.Quit

Enter your choice: 1

Enter the key to be inserted : 5
```



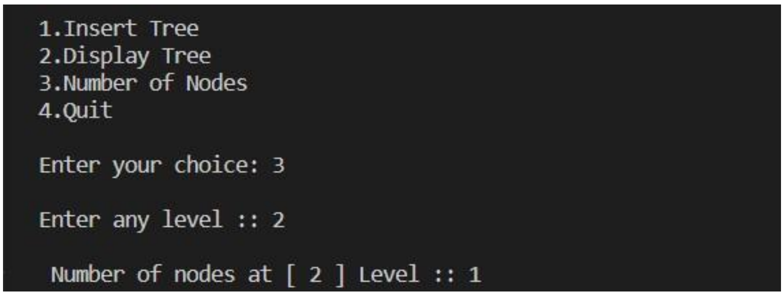
```
1.Insert Tree
2.Display Tree
3.Number of Nodes
4.Quit

Enter your choice: 1

Enter the key to be inserted : 6
```



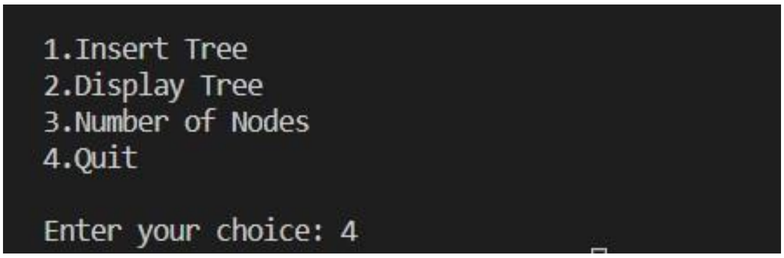
```
1.Insert Tree
2.Display Tree
3.Number of Nodes
4.Quit

Enter your choice: 1

Enter the key to be inserted : 8
```



```
1.Insert Tree
2.Display Tree
3.Number of Nodes
4.Quit

Enter your choice: 2


    8
  6
5
```



```
1.Insert Tree
2.Display Tree
3.Number of Nodes
4.Quit

Enter your choice: 3

Enter any level :: 2

 Number of nodes at [ 2 ] Level :: 1
```



```
1.Insert Tree
2.Display Tree
3.Number of Nodes
4.Quit

Enter your choice: 4
```