**Batch: B3 Roll No.: 121**

**Experiment / assignment / tutorial No. 7**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of BST & Binary tree traversal techniques. |

**Objective:** To Understand and Implement Binary Search Tree, Preorder, Postorder and Inorder Traversal Techniques.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 1 | Explain the different data structures used in problem solving |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. <https://www.geeksforgeeks.org/binary-tree-data-structure/>
3. <https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html>
4. <https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/>

**Abstract**:

**A tree** is a non- linear data structure used to represent hierarchical relationship existing among several data items. It is a finite set of one or more data items such that, there is a special data item called the root of the tree. Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.

**A binary tree** is a finite set of nodes. It is either empty or It consists a node called root with two disjoint binary trees-Left subtree, Right subtree. The Maximum degree of any node is 2

**A Binary Search Tree** is a node-based binary tree data structure in which 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.

**Related Theory: -**

**Preorder Traversal of BST**

Steps: 1. Visit the root

2. Traverse the left subtree

3. Traverse the right subtree

Preorder traversal is used to create a copy of the tree. It is also used to get the prefix expression of an expression tree. The time complexity is O(N).

**Postorder Traversal of BST**

Steps: 1. Traverse the left subtree

2. Traverse the right subtree

3. Visit the root

Postorder traversal is used to delete the tree. It is also useful to get the postfix expression of an expression tree. The time complexity is O(N).

**Inorder Traversal of BST**

Steps: 1. Traverse the left subtree

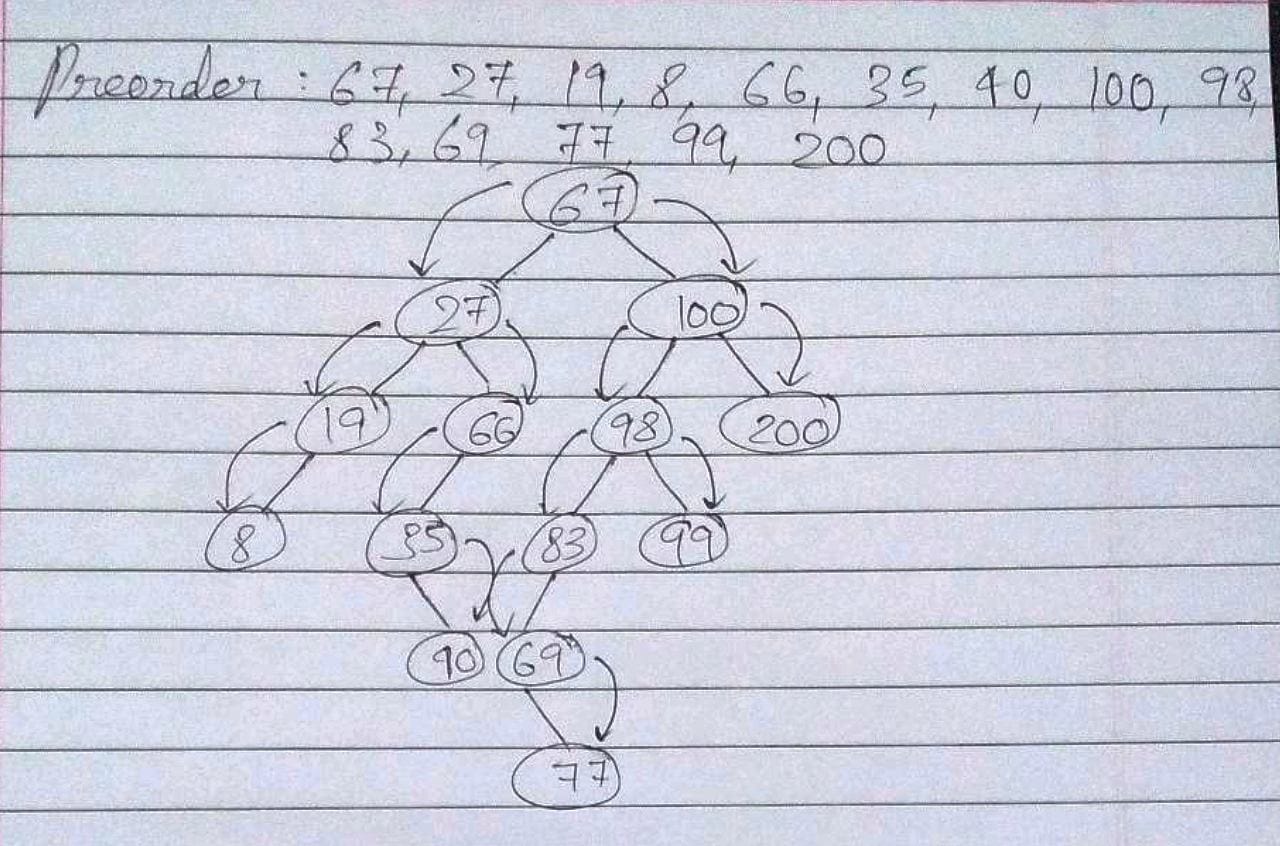
2. Visit the root

3. Traverse the right subtree

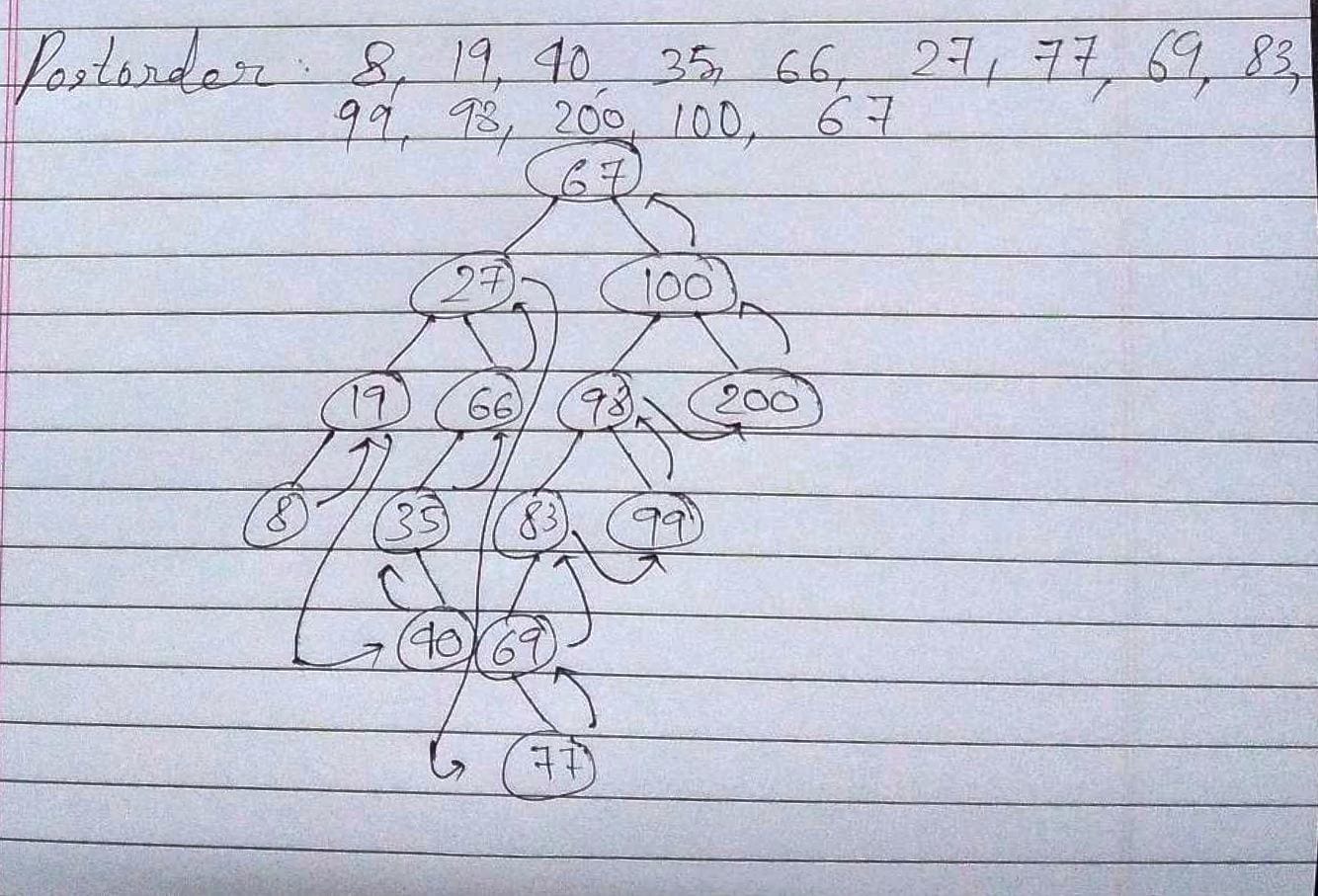
Inorder traversal gives nodes in non-decreasing order. The time complexity is O(N).

**Diagram for :**

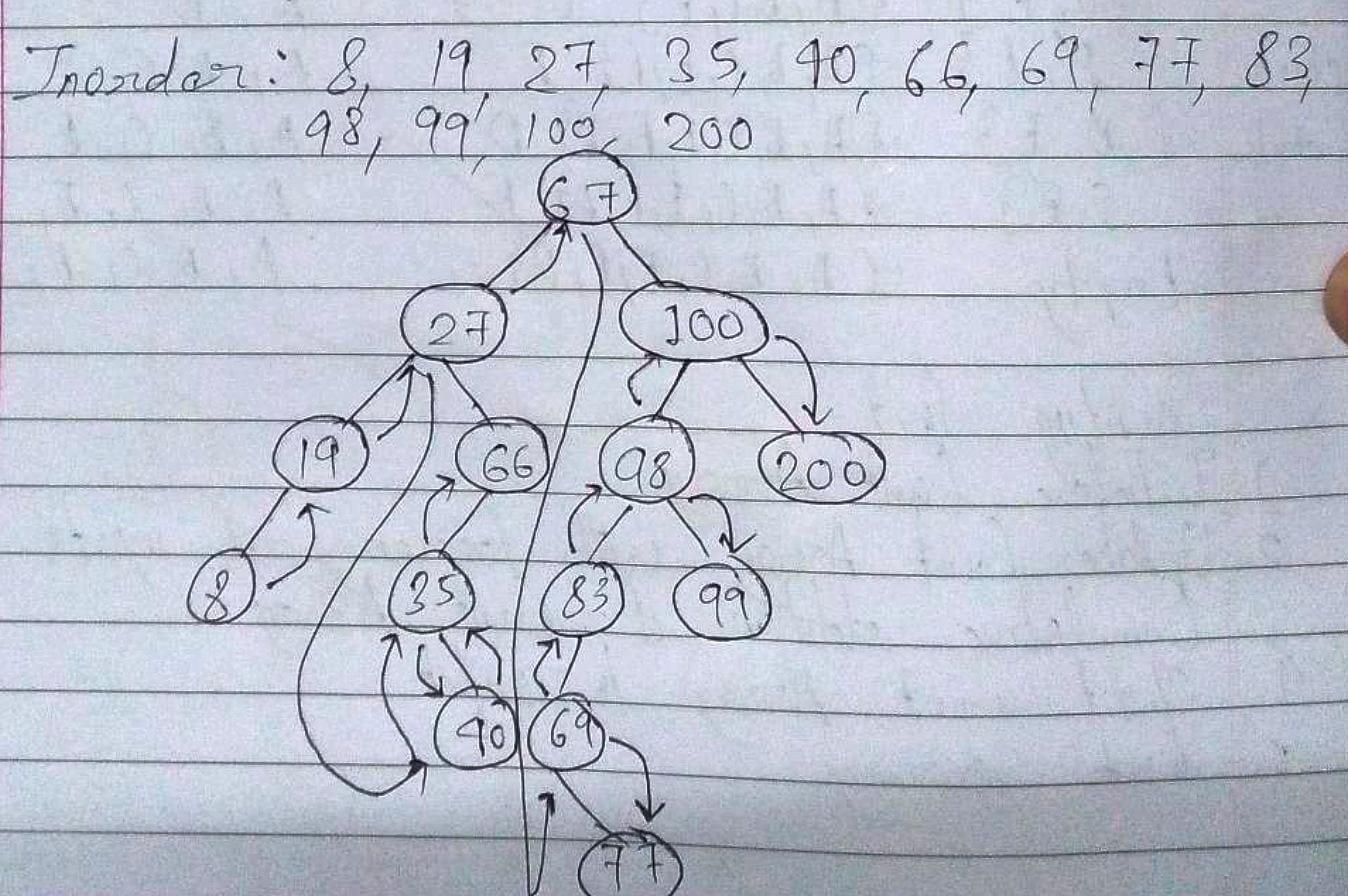
**Preorder Traversal of BST**

****

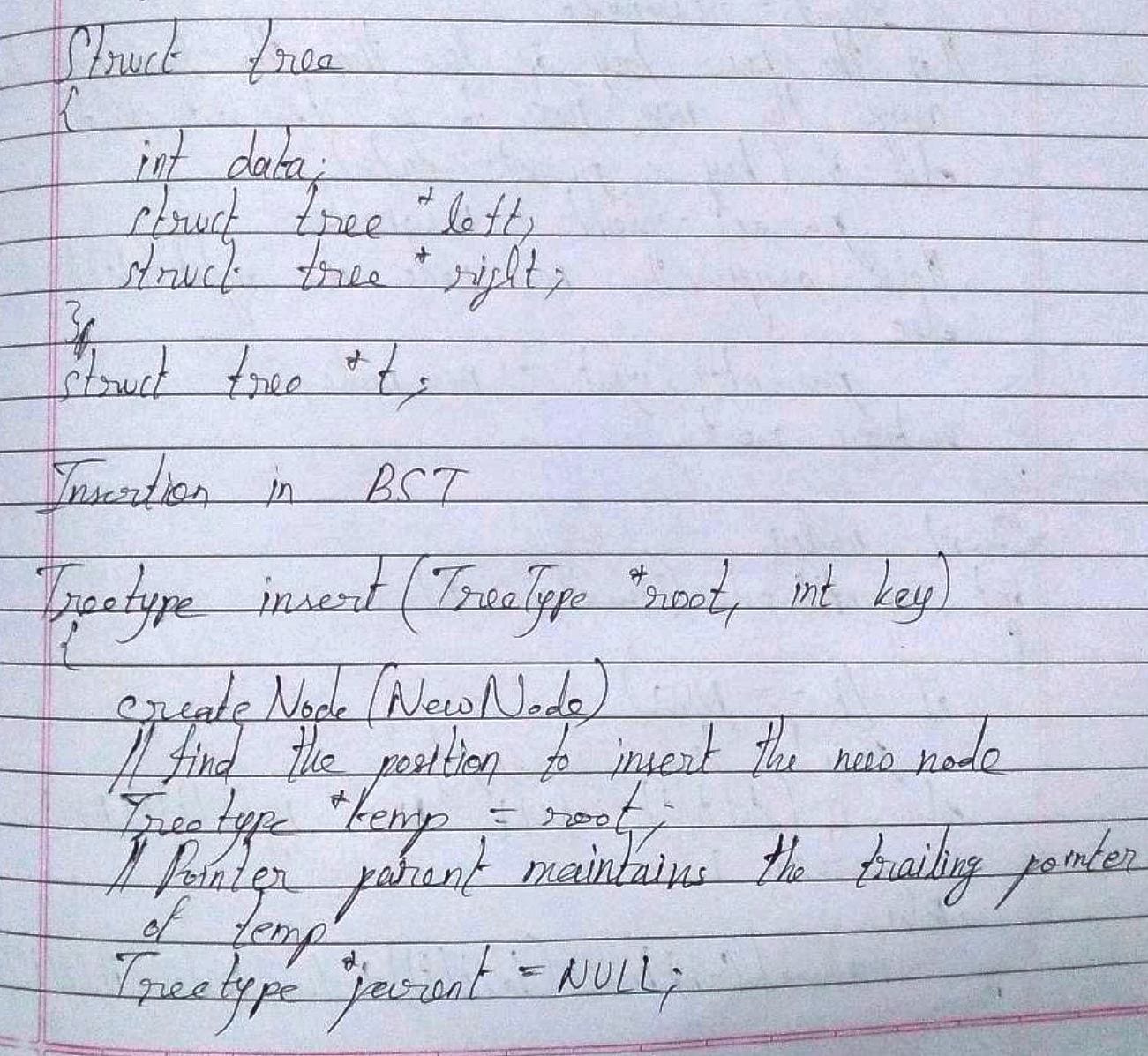
**Postorder Traversal of BST**

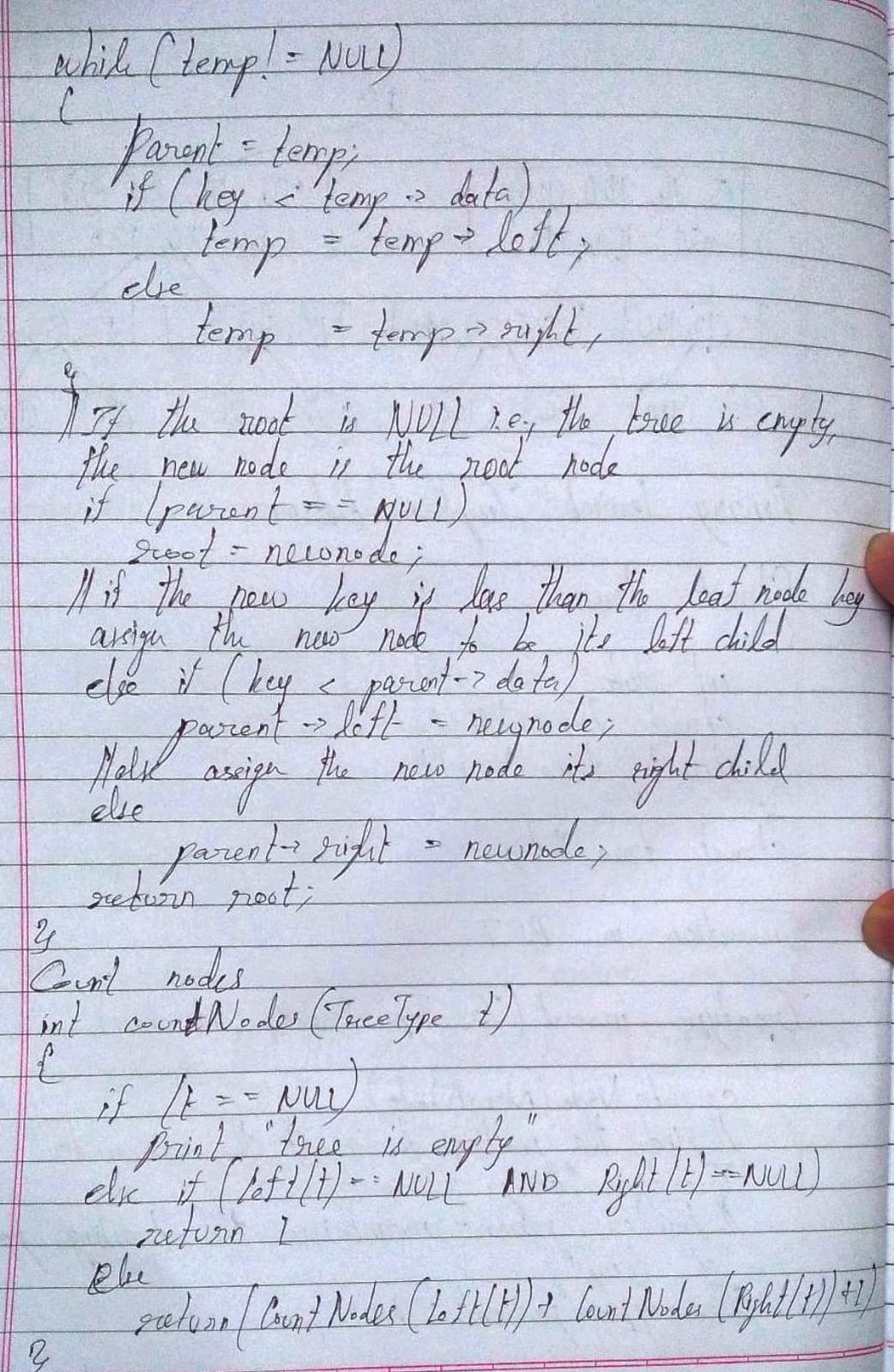
****

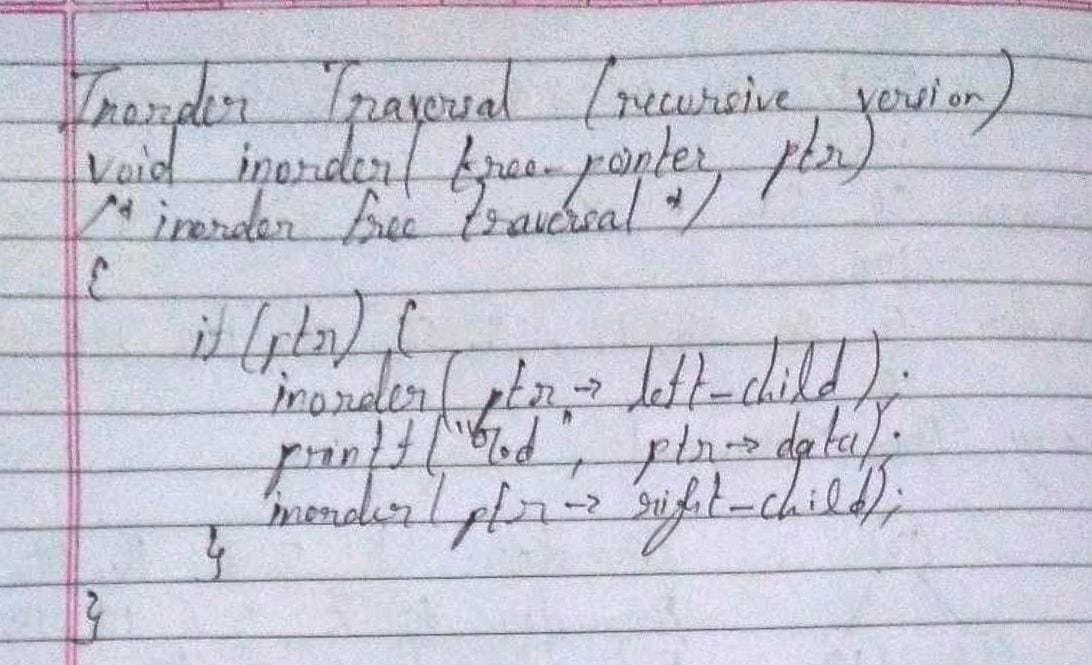
**Inorder Traversal of BST**

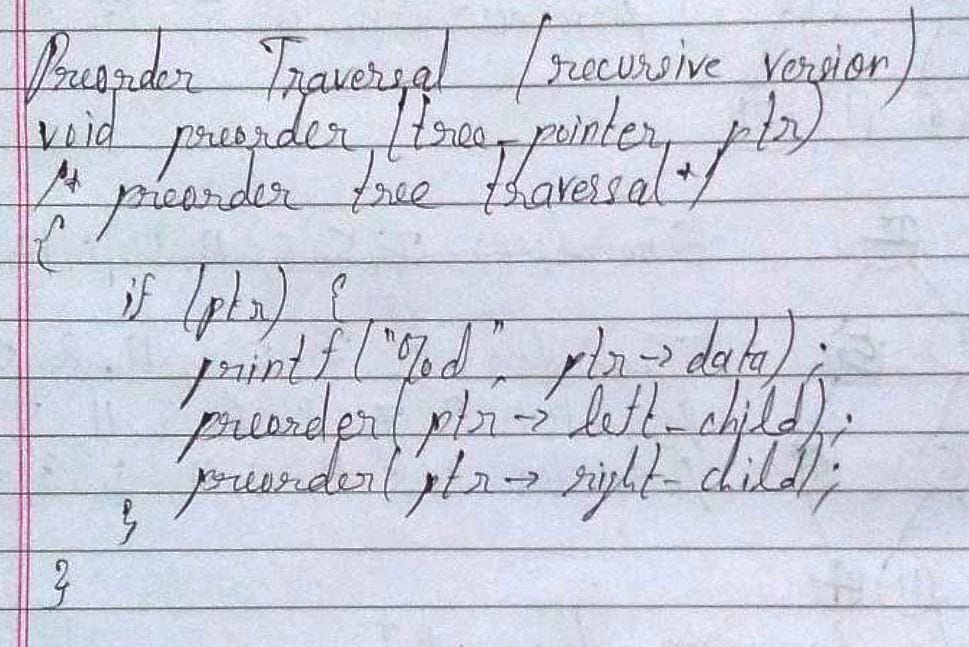
****

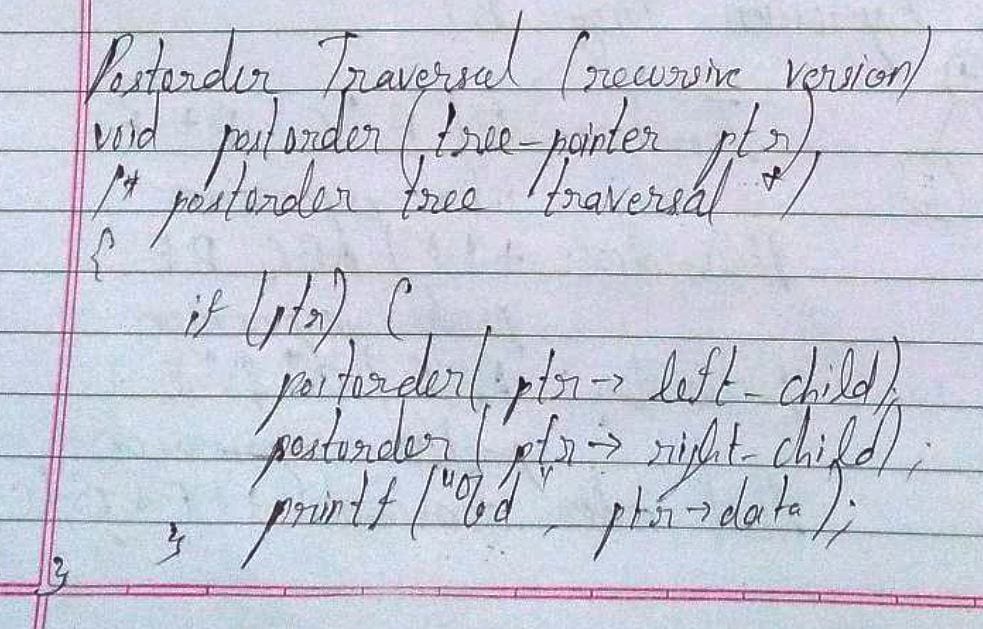
**Algorithm for Implementation of BST & Binary tree traversal techniques:**

****

****

****

****

****

**Implementation Details:**

1. **Enlist all the Steps followed and various options explored.**

**Ans.**

* 1. For code implementation, first the required C libraries were called.
  2. Then, a structure with three fields – one data field, one left subtree address field and one right subtree address field, is declared.
  3. Then, the methods are defined
  4. Then, the main method is defined and the code is written
  5. Then, the code is written for methods for creation of node, insertion of node, inorder traversal, preorder traversal and postorder traversal.

**Assumptions made for Input:**

The assumptions made are that all the input data are integers, and that there is always sufficient memory in the computer so that memory allocation is successful. Further, the input integers are within the range defined for integer data type.

**Built-In Functions Used:**

The built-in functions used include malloc() and free(). The malloc() function can be used by including the stdlib header file. It is used to allocate memory for a new node, according to the size of a node as defined by the structure “struct Node”. The free() function is used for node deletion.

**Program source code for Implementation of BST & Binary tree traversal techniques :**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

};

struct Node \*root = NULL;

struct Node \*create\_node(int);

void insert(int);

int search(int);

void inorder(struct Node\*);

void preorder();

void postorder();

int get\_data();

void main()

{

int ch, data;

do

{

printf("\nEnter:\n'1' to insert a new node.\n'2' to search for a node.\n'3' to get inorder representation.\n'4' to get preorder representation\n'5' to get postorder representation\n'6' to exit\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

data = get\_data();

insert(data);

break;

case 2:

data = get\_data();

if(search(data) == 1)

printf("\nThe data %d has been found.", data);

else

printf("\nThe data was not found.");

break;

case 3:

inorder(root);

break;

case 4:

preorder(root);

break;

case 5:

postorder(root);

break;

case 6:

printf("\nE\tX\tI\tT\tI\tN\tG\t.\t.\t.");

break;

default:

printf("\nPlease enter '1', '2', '3', '4', '5' or '6' only and try again!");

}

}while(ch!=6);

}

struct Node \*create\_node(int data)

{

struct Node \*new\_node;

new\_node = (struct Node\*)malloc(sizeof(struct Node));

if(new\_node == NULL)

{

printf("\nMemory allocation was unsuccessful.");

return NULL;

}

new\_node -> data = data;

new\_node -> left = NULL;

new\_node -> right = NULL;

return new\_node;

}

void insert(int data)

{

struct Node \*new\_node;

new\_node = create\_node(data);

if(new\_node != NULL)

{

if(root == NULL)

{

root = new\_node;

printf("\nThe node having data %d has been inserted.", data);

return;

}

struct Node \*temp = root;

struct Node \*prev = NULL;

while(temp != NULL)

{

prev = temp;

if(data > temp -> data)

temp = temp -> right;

else

temp = temp -> left;

}

if(data > prev -> data)

prev -> right = new\_node;

else

prev -> left = new\_node;

printf("\nThe node having data %d has been inserted.", data);

}

}

int search(int key)

{

struct Node \*temp = root;

while(temp != NULL)

{

if(temp -> data == key)

return 1;

else if(key > temp -> data)

temp = temp -> right;

else

temp = temp -> left;

}

return 0;

}

void inorder(struct Node \*root)

{

if(root == NULL)

return;

inorder(root -> left);

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

inorder(root -> right);

}

void preorder(struct Node \*root)

{

if(root == NULL)

return;

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

preorder(root -> left);

preorder(root -> right);

}

void postorder(struct Node \*root)

{

if(root == NULL)

return;

postorder(root -> left);

postorder(root -> right);

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

}

int get\_data()

{

int data;

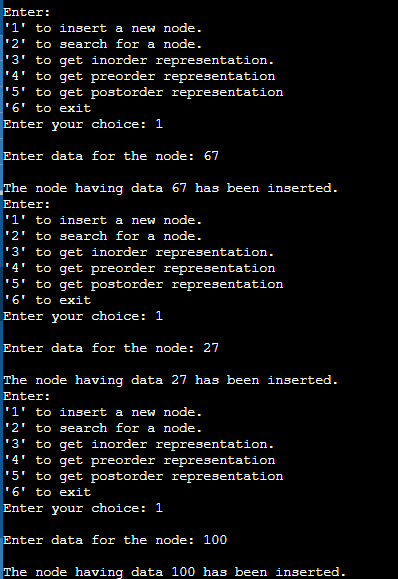
printf("\nEnter data for the node: ");

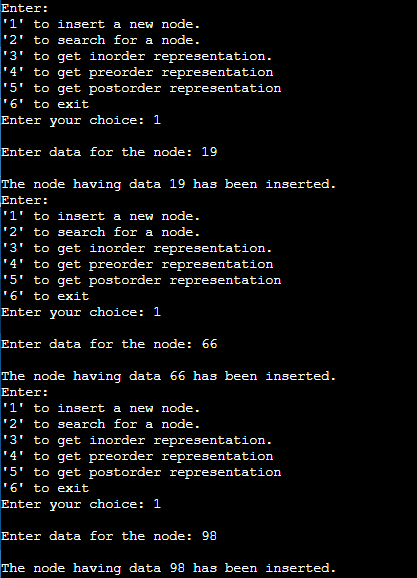
scanf("%d", &data);

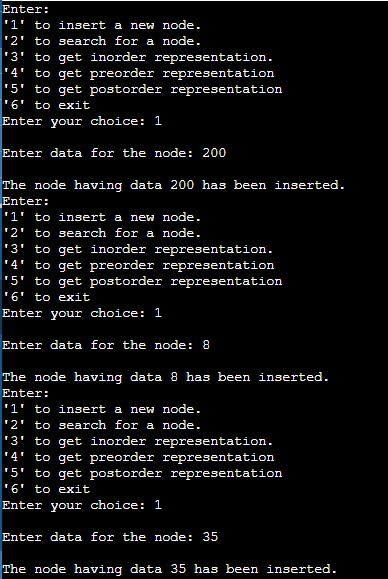
return data;

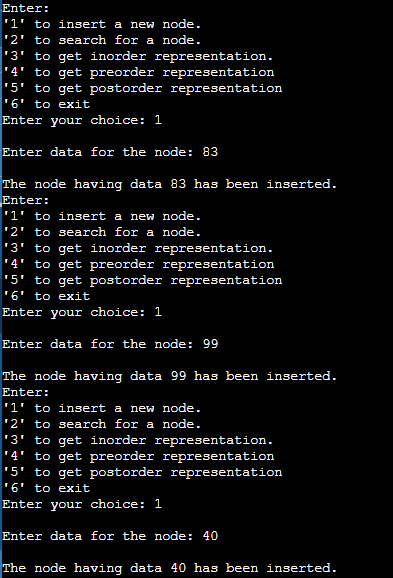
}

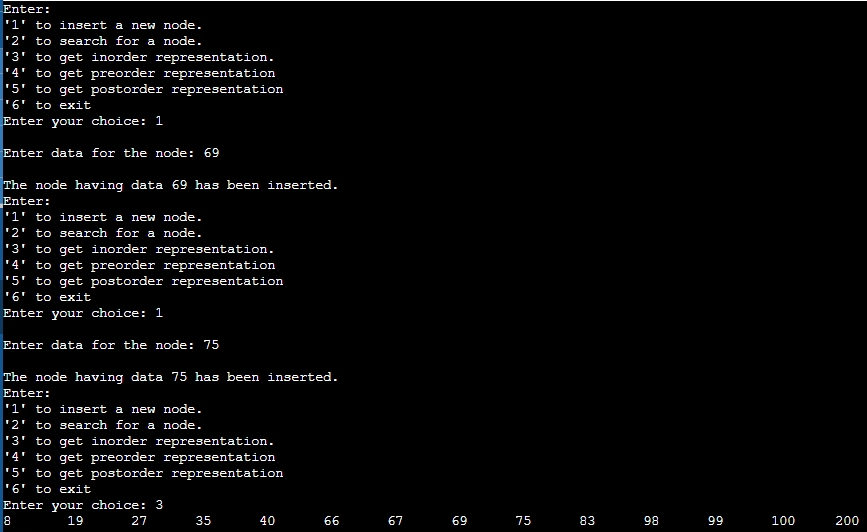
**Output Screenshots for Each Operation:**

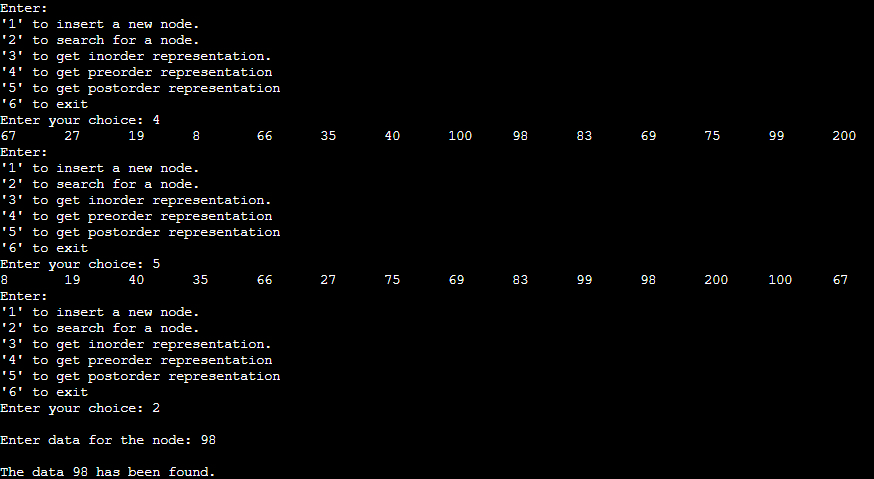
****

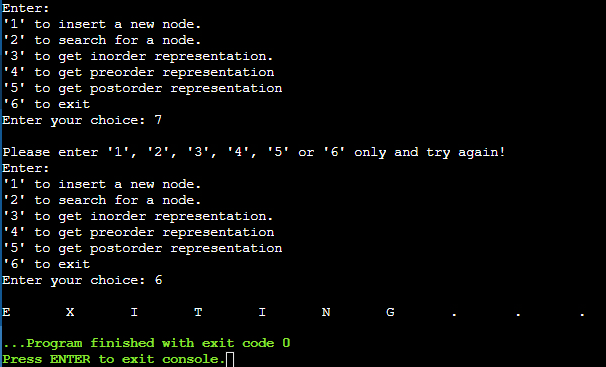
****

****

****

****

****

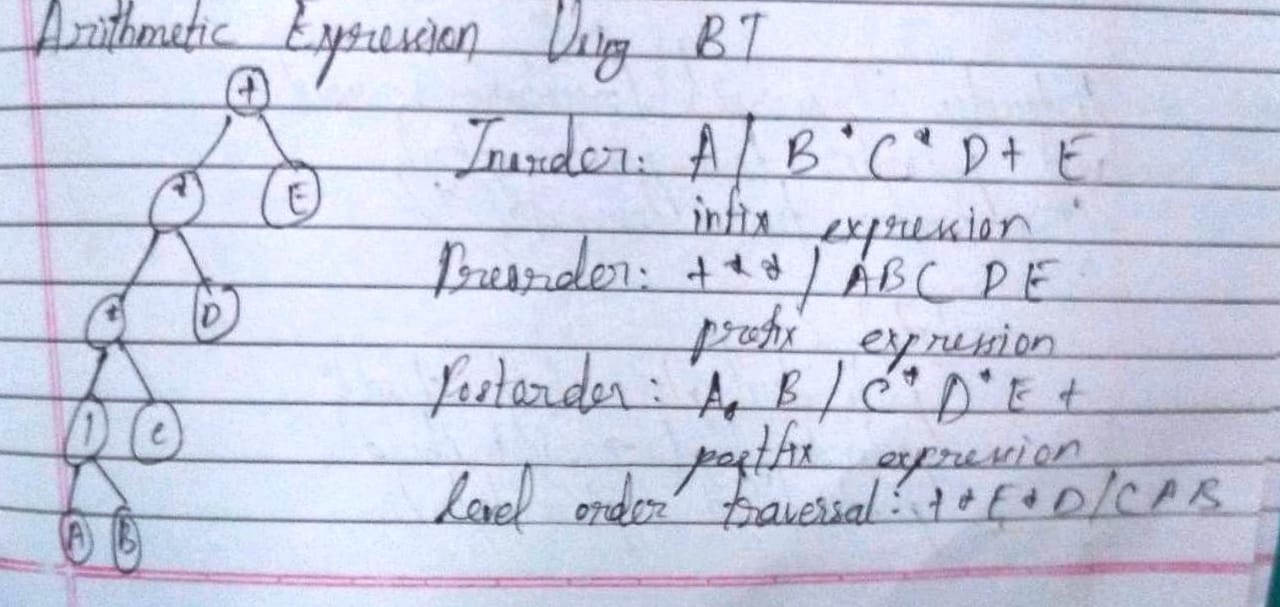
****

**Conclusion:-**

Thus, in this experiment, the concept of Binary Search Trees and their traversals has been learnt and implemented in C programming. Trees are non-linear data structure. They make for faster insertion and deletion of nodes as well as faster searching or traversing of nodes. Thus, the advantages of linked list (faster insertion and deletion) and array (faster searching) are combined in one data structure.

**PostLab Questions:**

1. **Illustrate 2 Applications of Trees.**
   1. Trees are used to store hierarchical data, like folder structure, organization structure, XML/HTML data.
   2. Trees are used to evaluate an expression. Preorder traversal of an expression tree gives the prefix expression. Postorder traversal of an expression tree gives the postfix expression.



1. **Compare and Contrast between B Tree and B+ Tree?**

**Ans.**

|  |  |  |
| --- | --- | --- |
| Sr. No. | B Tree | B+ Tree |
| 1. | All the keys and records are stored in both internal as well as leaf nodes. | The keys are the indexes stored in the internal nodes and records are stored in the leaf nodes. |
| 2. | Keys cannot be repeatedly stored. Hence, there is no duplication of keys or records. | Redundant keys can be present in the internal nodes. |
| 3. | Leaf nodes are not linked to each other. | Leaf nodes are linked to each other to provide sequential access. |
| 4. | Searching is not very efficient because the records are either stored in leaf or internal nodes. | Searching is very efficient because all the records are stored in the leaf nodes. |