DATA STRUCTURES LABORATORY

Course Code: 1BCSL306

VTU 2025 Scheme – Semester 3

EXPERIMENT – 06

Binary Tree

Level-Order Insertion and Tree Traversals

Simple Code • Easy to Remember • Easy to Write in Exam

SEARCH CREATORS

Your VTU Study Partner

Experiment 06

1. Aim

Aim

To develop a menu-driven C program to create a Binary Tree using level-order insertion and perform preorder, inorder and postorder traversals.

2. VTU Prescribed Experiment

VTU Prescribed Experiment

Develop a menu-driven C program for Binary Tree operations.

a. Create a Binary Tree using level-order insertion of integer keys.

b. Implement the function `createTree()` to construct the tree.

c. Implement `preorder()` traversal.

d. Implement `inorder()` traversal.

e. Implement `postorder()` traversal.

f. Provide a menu-driven interface with the following options:

- **i.** Create Tree
- **ii.** Display Preorder Traversal
- **iii.** Display Inorder Traversal
- **iv.** Display Postorder Traversal
- **v.** Exit

3. Simple Theory

Binary Tree:

A Binary Tree is a tree data structure in which each node can have at most two children. They are:

$$\boxed{Left\ Child}$$

and:

$$\boxed{Right\ Child}$$

Each node contains:

$$\boxed{Data + Left\ Pointer + Right\ Pointer}$$

Node structure:

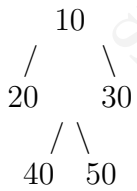
```
struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
};
```

4. Example Binary Tree

Suppose the values are inserted as:

$$10,\ 20,\ 30,\ 40,\ 50$$

Using level-order insertion, the tree becomes:



5. Level-Order Insertion

Level-order insertion fills the tree:

$$\boxed{Level\ by\ Level}$$

from:

$$\boxed{Left\ to\ Right}$$

For example:

$$10,\ 20,\ 30,\ 40,\ 50$$

is inserted as:

$$10 \rightarrow 20, 30 \rightarrow 40, 50$$

6. Tree Traversals

6.1. Preorder

Order:

$$\boxed{Root \rightarrow Left \rightarrow Right}$$

Short form:

$$\boxed{NLR}$$

6.2. Inorder

Order:

$$\boxed{Left \rightarrow Root \rightarrow Right}$$

Short form:

$$\boxed{LNR}$$

6.3. Postorder

Order:

$$\boxed{Left \rightarrow Right \rightarrow Root}$$

Short form:

$$\boxed{LRN}$$

7. Algorithm

7.1. Create Tree

1. Start.
2. Read number of nodes.
3. Create root node using first value.
4. Use a queue to store node addresses.
5. Take one node from queue.
6. Insert new node as left child if left is empty.

7. Otherwise insert new node as right child.
8. Add newly created node to queue.
9. Repeat until all values are inserted.
10. Return root.

7.2. Preorder Algorithm

1. If root is not NULL:
2. Display root data.
3. Traverse left subtree.
4. Traverse right subtree.

$$\boxed{Root \rightarrow Left \rightarrow Right}$$

7.3. Inorder Algorithm

1. If root is not NULL:
2. Traverse left subtree.
3. Display root data.
4. Traverse right subtree.

$$\boxed{Left \rightarrow Root \rightarrow Right}$$

7.4. Postorder Algorithm

1. If root is not NULL:
2. Traverse left subtree.
3. Traverse right subtree.
4. Display root data.

$$\boxed{Left \rightarrow Right \rightarrow Root}$$

Easy Exam Memory

Remember only:

$$\boxed{Preorder = NLR}$$

$$\boxed{Inorder = LNR}$$

$$\boxed{Postorder = LRN}$$

Where:

$$N = Node, \quad L = Left, \quad R = Right$$

8. C Program – Easy Student Version

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

struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* newNode(int x)
{
    struct Node *p;

    p=(struct Node*)malloc(sizeof(struct Node));

    p->data=x;
    p->left=NULL;
    p->right=NULL;

    return p;
}

struct Node* createTree()
{
    struct Node *root,*q[20],*temp,*p;
    int front=0,rear=-1;
    int n,i,x;

    printf("Enter number of nodes: ");
    scanf("%d",&n);

    if(n==0)
        return NULL;

    printf("Enter root value: ");
    scanf("%d",&x);

    root=newNode(x);
    q[++rear]=root;

    for(i=1;i<n;i++)
    {
        temp=q[front];

        printf("Enter value: ");
        scanf("%d",&x);

        p=newNode(x);

```

```
        if(temp->left==NULL)
        {
            temp->left=p;
            q[++rear]=p;
        }
        else
        {
            temp->right=p;
            q[++rear]=p;
            front++;
        }
    }

    return root;
}

void preorder(struct Node *root)
{
    if(root!=NULL)
    {
        printf("%d ",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(struct Node *root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%d ",root->data);
        inorder(root->right);
    }
}

void postorder(struct Node *root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ",root->data);
    }
}

int main()
{
    struct Node *root=NULL;
    int ch;

    do
    {
```

```
        printf("\n1.Create Tree");
        printf("\n2.Preorder");
        printf("\n3.Inorder");
        printf("\n4.Postorder");
        printf("\n5.Exit");

        printf("\nEnter Choice: ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                root=createTree();
                break;

            case 2:
                printf("Preorder: ");
                preorder(root);
                printf("\n");
                break;

            case 3:
                printf("Inorder: ");
                inorder(root);
                printf("\n");
                break;

            case 4:
                printf("Postorder: ");
                postorder(root);
                printf("\n");
                break;

            case 5:
                printf("Program Ended\n");
                break;

            default:
                printf("Invalid Choice\n");
        }

    }while(ch!=5);

    return 0;
}
```


9. Simple Program Explanation

9.1. Node Structure

```
struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
};
```

Each node stores:

$$\boxed{Data + Left + Right}$$

9.2. newNode() Function

A new node is created dynamically using:

```
malloc()
```

Then:

```
p->left=NULL;
p->right=NULL;
```

means the new node initially has no children.

9.3. createTree() Function

A simple queue is used:

```
struct Node *q[20];
```

The queue stores addresses of tree nodes.

First:

$$\boxed{Root}$$

is inserted.

Then new nodes are added:

$$\boxed{Left\ Child}$$

first, followed by:

$$\boxed{Right\ Child}$$

This produces level-order insertion.

9.4. preorder() Function

```
printf("%d ",root->data);
preorder(root->left);
preorder(root->right);
```

Therefore:

$$\boxed{NLR}$$

9.5. inorder() Function

```
inorder(root->left);
printf("%d ",root->data);
inorder(root->right);
```

Therefore:

$$\boxed{LNR}$$

9.6. postorder() Function

```
postorder(root->left);
postorder(root->right);
printf("%d ",root->data);
```

Therefore:

$$\boxed{LRN}$$

Easy Exam Memory

The traversal code is almost the same.
Only the position of:

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

changes.
For Preorder:

$$\boxed{\textit{Print First}}$$

For Inorder:

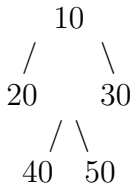
$$\boxed{\textit{Print Middle}}$$

For Postorder:

$$\boxed{\textit{Print Last}}$$

10. Example

Consider the tree:



10.1. Preorder

Rule:

$$Root \rightarrow Left \rightarrow Right$$

Therefore:

$$\boxed{10,\ 20,\ 40,\ 50,\ 30}$$

10.2. Inorder

Rule:

$$Left \rightarrow Root \rightarrow Right$$

Therefore:

$$\boxed{40,\ 20,\ 50,\ 10,\ 30}$$

10.3. Postorder

Rule:

$$Left \rightarrow Right \rightarrow Root$$

Therefore:

$$\boxed{40,\ 50,\ 20,\ 30,\ 10}$$

11. Sample Input and Output

```
1.Create Tree
2.Preorder
3.Inorder
4.Postorder
5.Exit

Enter Choice: 1

Enter number of nodes: 5
Enter root value: 10
Enter value: 20
Enter value: 30
Enter value: 40
Enter value: 50


Enter Choice: 2

Preorder: 10 20 40 50 30


Enter Choice: 3

Inorder: 40 20 50 10 30


Enter Choice: 4

Postorder: 40 50 20 30 10


Enter Choice: 5

Program Ended
```


12. Simple Test Cases

Test	Operation	Expected Result
1	Create a tree with one node	The entered node becomes root.
2	Create tree with values 10,20,30	20 becomes left child and 30 becomes right child.
3	Perform Preorder	Root should be displayed first.
4	Perform Inorder	Root should be displayed between left and right subtrees.
5	Perform Postorder	Root should be displayed last.

13. Result

Result

Thus, the C program to create a Binary Tree using **level-order insertion** was successfully implemented.
The program performs:

- Binary Tree Creation
- Level-Order Insertion
- Preorder Traversal
- Inorder Traversal
- Postorder Traversal

14. Important Viva Questions

Important Viva Questions

1. What is a Binary Tree?
A Binary Tree is a tree in which each node has at most two children.

2. What are the two children called?
Left child and Right child.

3. What is the root node?
The topmost node of a tree is called the root.

4. What is a leaf node?
A node having no children is called a leaf node.

5. What is level-order insertion?
It inserts nodes level by level from left to right.

6. Which data structure is useful for level-order insertion?
Queue.

7. What is Preorder traversal?

$$\boxed{Root \rightarrow Left \rightarrow Right}$$

8. What is Inorder traversal?

$$\boxed{Left \rightarrow Root \rightarrow Right}$$

9. What is Postorder traversal?

$$\boxed{Left \rightarrow Right \rightarrow Root}$$

10. What is another name for Preorder?

$$\boxed{NLR}$$

11. What is another name for Inorder?

$$\boxed{LNR}$$

12. What is another name for Postorder?

$$\boxed{LRN}$$

13. Why is recursion used in tree traversal?
Because every subtree is itself a tree.

14. What does NULL represent in a tree node?
It indicates that the corresponding child does not exist.

15. Which function is used for dynamic memory allocation?

`malloc()`

15. 1-Minute Exam Revision

Easy Exam Memory

Node

$$\boxed{Data + Left + Right}$$

Creation

$$\boxed{Level\ by\ Level}$$

Preorder

$$\boxed{NLR}$$

Inorder

$$\boxed{LNR}$$

Postorder

$$\boxed{LRN}$$

Functions

$$\boxed{newNode()}$$

$$\boxed{createTree()}$$

$$\boxed{preorder()}$$

$$\boxed{inorder()}$$

$$\boxed{postorder()}$$

16. Easy Code Memory Map

Remember

Remember the program in this order:

$$\boxed{Node \rightarrow newNode \rightarrow createTree \rightarrow Traversals \rightarrow Main}$$

The three traversal functions are almost identical.
Only remember:

$$\boxed{Preorder = Print\ First}$$

$$\boxed{Inorder = Print\ Middle}$$

$$\boxed{Postorder = Print\ Last}$$

That is the easiest way to reproduce all three functions in the exam.

SEARCH CREATORS

1BCSL306 – Data Structures Laboratory

EXPERIMENT 06
COMPLETED

Binary Tree

Level-Order Insertion and Tree Traversals

Understand → Remember → Write → Execute → Viva

Your VTU Study Partner