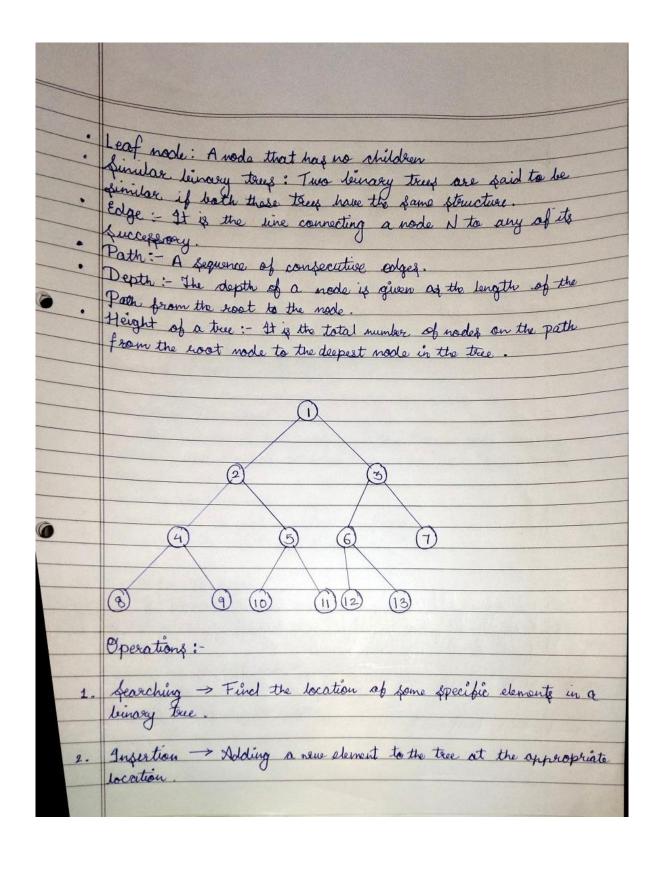
	A: Experiment. No. 04
	Asmoi mul
	real-world Binary Tree and its Town 18
	Aim i Implementation of Binary Tree and its Teaureal for Regenties:
	Objectives:
	1. To learn fundamentals and implementation of Burary Tree.  2. To develop an ability to dozign and analyze algorithms now Theory:
	2. To develop and implementation of Burasy Tree
	tree data it to dasign and analyse algorithms is
	purcure.
_	There's
_	A Binary Tree is a data structure that is defined as a collection of elements called nodes. In a binary tree, the topmos at the most 2 children. A node that has oper children is
_	collection of election of all a data structure that is defined as a
_	element is called to called nodes. In a binary tree, the topino
_	at the most ? alilder and each mode has o, I a
_	called a leaf node or a tryinger and each mode has o, I a
_	data element, a left pointer which son to the the sixth will so
	data element, a left pointer which points to the right child. I
	James James.
	Torminology:
	00
•	Parent: If Nix any node in T that has left successor S, an
	right successor S2, then N is called the present of S, & S2.
	Level number: Every node in the binary tree is assigned to
	Degree of a Node: It is equal to the number of children to
	1000-0-000
	filling: All nodes that are at the same level and shore the same parent are called sublings.
	the same parent are called sublings.



	Deletion -> Deleting some specific node from a leinary tree.
5.	Deletion -> Deleting some specific node from a remary
4.	Teaverstung -> Process of visiting
	Tree transversal and its types:
	Traversing a lunger tree is the Process of visiting each mode
	in the two exactly once in a sustanatic way unlike linear
•	Traversing a binary tree is the Process of visiting each mode in the true exactly once in a systematic way unlike linear data structures in which the elements are traversed fequentially of
	tere is a non-linear data structure in which the elements can
	traises soal in many dishorout way.
	traversed in many different way.
1.	Pre-order Traversal.
	To transport a man - empty binger true in Pro-order , the
	To traverse a non-empty binous true in pro-order of the following apprations are performed recursively at each nade the
	Alva the 1000
	algorithm ways.
	Visiting the root note
0 .	Traversing the left sub-tree and finally. Traversing the right sub-tree.
•	Thater fing the right fine sail.
2.	In order Traversal.
	To traverse a non-empty kingry tree in in order the following operation are performed escursively at each mode. The algorithm
	operation the performed recursively at their mode, or significant
	work by
	Traversing the left sub-tree
	Visiting the root mode and fundy.
	Traversing the right subtree.
	V

-	Past - order traverdal:
	following operations are performed recurringly at each made. The
	following operations are perporned growswell at each made. The
	Pearsering the right but tree of finally.
	Traversing the right but tree & finally. Visiting the root node.
	2
	Algorithms:
->	
	Decaching for a given value.
	Secreting for a given value. Step 1:- If Tree -> DATA = VAL ORTREE = NULL  Return TREE
	Terror IVOD
	ELSE
	If VAL < TREE -> DATA
	Roturn search Element (TREE -> LEFT VAL)
The state of the s	ELSE
6	Reliver fearch Element (TREE -> RIGHT, VAWE)
•	[END. of AF]
	[END OF JE].
	Step 28- END
->	Angertion :- INSERT (TREE, VAL).
	Step 1 - 4 F TREE = NULL
	Allocate memory for Tree.
	SET TREE → DATA = VAL
	SET TREE -> LEFT = TREE -> RIGHT = NULL.
	ELSE
	IF VAL < TREE → DATA.
	INSERT (TREE -> LEFT, VAL.)

	ELSE
	IF VAL <tree data<="" td="" →=""></tree>
	INSERT (TRED >PIGUE
	Step 2: END OF IF]
1	, † 2. END
6.	DELETION
	Delat ITA
	Step 1: 3f TREE = NULL. Woute "VAL + P
	Woute "VAI + 1
	I MAI FOLLLING " HI A
	Else if VAL STREE -> DATA  Debte (TREE -> LEFT, VAL)
	Else if VAL STREE -> DATA.
	Delete (TREE -> RIGHT , VAL).
	Else if TREE -> LEFT AND TREE -> RIGHT
	SET TEMP = Find largest Node (TREE -> LEFT)
	SET TREE -> DATA = TEMP -> DATA
-	DELETE (TREE -> LEFT, TEMP -> DATA.
	ELSE
	SET TEMP = TREE
	AF TREE → LEFT = NULL AND TREE → RIGHT = NULL.
	SET TREE = NULL
	Elge AF TREE -> LEFT! = NULL
	SET TREE = TREE -> LEFT
	Else
	SET TREE = TREE -> RIGHT.
	(END OF IF)
	FREE TEMP
(0)	LEND OF LP].

Atip 2: END.  Pre-order Traversal.  Atip 1: Report Atip 2 to 4 while TREE! = NULL.  Atip 2: Write TREE > DATA.  Atip 3: PREORDER (TREE > LEFT)  Atip 4: PREORDER (TREE > RIGHT).  END OF LOOP  Atip 5: END  Inorder Traversal:  Atip 1: Report Atips 2 to 4 while TREE! = NULL  Atip 3: Write TREE > RIGHT  Atip 4: INORDER (TREE > RIGHT).  LEND OF LOOP  Atip 5: END  Post-order Traversal  Atip 1: Report Atips 2 to 4 while TREE! = NULL  Atip 2: POSTORDER (TREE > LEFT)  Atip 3: POSTORDER (TREE > RIGHT)  Atip 4: Write TREE > DATA.  [END OF LOOP]  Atip 5: END.		
Pre-order Traversal.  Step 1: Report Step 2 to 4 while TREE! = NULL.  Step 2: Write TREE -> DATA.  Step 3: PREORDER (TREE -> LEFT)  Step 4: PREORDER (TREE -> RIGHT).  END OF LOOP  Step 5: END  > Inorder Traversal.  Step 1: Report steps 2 to 4 while TREE! = NULL  Step 3: Write TREE -> LEFT)  Step 3: Write TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  > Post-order Traversal  Step 1: Report steps 2 to 4 while TREE! = NULL  Step 2: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> RIGHT)  Step 3: Post order (TREE -> RIGHT)  Step 3: Post order (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]		
Pre-order Traversal.  Step 1: Report Step 2 to 4 while TREE! = NULL.  Step 2: Write TREE -> DATA.  Step 3: PREORDER (TREE -> LEFT)  Step 4: PREORDER (TREE -> RIGHT).  END OF LOOP  Step 5: END  > Inorder Traversal.  Step 1: Report steps 2 to 4 while .TREE! = NULL  Step 3: Write TREE -> LEFT)  Step 3: Write TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  > Post-order Traversal  Step 1: Report steps 2 to 4 while .TREE! = NULL  Step 2: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	-	
Pre-order Traversal.  Step 1: Report Step 2 to 4 while TREE! = NULL.  Step 2: Write TREE -> DATA.  Step 3: PREORDER (TREE -> LEFT)  Step 4: PREORDER (TREE -> RIGHT).  END OF LOOP  Step 5: END  > Inorder Traversal.  Step 1: Report steps 2 to 4 while .TREE! = NULL  Step 3: Write TREE -> LEFT)  Step 3: Write TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  > Post-order Traversal  Step 1: Report steps 2 to 4 while .TREE! = NULL  Step 2: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]		
Pre-order Traversal.  Step 1: Report Step 2 to 4 while TREE! = NULL.  Step 2: Write TREE -> DATA.  Step 3: PREORDER (TREE -> LEFT)  Step 4: PREORDER (TREE -> RIGHT).  END OF LOOP  Step 5: END  > Inorder Traversal.  Step 1: Report steps 2 to 4 while .TREE! = NULL  Step 3: Write TREE -> LEFT)  Step 3: Write TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  > Post-order Traversal  Step 1: Report steps 2 to 4 while .TREE! = NULL  Step 2: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> LEFT)  Step 3: Post order (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]		Stena
Step 2: Write TREE -> DATA  Step 3: PREORDER (TREE -> LEPT)  Step 4: PREORDER (TREE -> RIGHT).  END OF LOOP  Step 5: END   Inorder Tecaversal.  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 3: Write TREE -> RIGHT).  [END OF LOOP]  Step 5: END  Post - order Traversal  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 2: Post order (TREE -> LEFT)  Step 3: Postorder (TREE -> LEFT)  Step 3: Postorder (TREE -> LEFT)  Step 3: Postorder (TREE -> RIGHT)  Step 3: Postorder (TREE -> RIGHT)  Step 3: Postorder (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]		TOPE END.
Step 2: Write TREE -> DATA  Step 3: PREORDER (TREE -> LEPT)  Step 4: PREORDER (TREE -> RIGHT).  END OF LOOP  Step 5: END   Inorder Tecaversal.  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 3: Write TREE -> RIGHT).  [END OF LOOP]  Step 5: END  Post - order Traversal  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 2: Post order (TREE -> LEFT)  Step 3: Postorder (TREE -> LEFT)  Step 3: Postorder (TREE -> LEFT)  Step 3: Postorder (TREE -> RIGHT)  Step 3: Postorder (TREE -> RIGHT)  Step 3: Postorder (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	~	Per- sed T
Step 2: Write TREE -> DATA  Step 3: PREORDER (TREE -> LEPT)  Step 4: PREORDER (TREE -> RIGHT).  END OF LOOP  Step 5: END   Inorder Tecaversal.  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 3: Write TREE -> RIGHT).  [END OF LOOP]  Step 5: END  Post - order Traversal  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 2: Post order (TREE -> LEFT)  Step 3: Postorder (TREE -> LEFT)  Step 3: Postorder (TREE -> LEFT)  Step 3: Postorder (TREE -> RIGHT)  Step 3: Postorder (TREE -> RIGHT)  Step 3: Postorder (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	_	Step 1: Parent the att of the state of the
SEP 4: PREORDER (TREE -> LEFT)  END OF LOOP  Step 5: END  Increder Traversal.  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 2: INORDER (TREE -> LEFT)  Step 3: Write TREE -> RIGHT  [END OF LOOP]  Step 5: END  Post-order Traversal  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 2: Past order (TREE -> LEFT)  Step 3: Postorder (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]		
END OF LOOP  Step 5: END  Inorder Tecanessal.  Step 1: Repeat steps 2 to 4 while .TREE! = NULL  Step 2: INORDER (TREE -> LEFT)  Step 3: White TREE -> RIGHT).  [END OF LOOP]  Step 5: END  Post-order Teanessal.  Step 1: Repeat steps 2 to 4 while .TREE! = NULL.  Step 2: Post or Der (TREE -> LEFT)  Step 3: Post or Der (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	_	PREMOTED (TOPE ) IEET
Atop 5: END  Inorder Teawerfal.  Atop 1: Repeat stops 2 to 4 while TREE!= NULL  Atop 2: INORDER (TREE -> LEFT)  Atop 4: INORDER (TREE -> RIGHT).  LEND OF LOOP!  Atop 5: END  Post-corder Teawafal.  Atop 1: Repeat stops 2 to 4 while TREE!= NULL.  Atop 2: Past ORDER (TREE -> RIGHT)  Atop 3: POSTORDER (TREE -> RIGHT)  Atop 4: Write TREE -> DATA.  [END OF LOOP]	-	Step 4: PREORNEZ (TREE > RIGHT)
Inorder Tecoversal.  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 2: INORDER (TREE -> LEFT)  Step 3: Write TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT)  LEND OF LOOP!  Step 5: END  Post-order Teaverful  Step 1: Repeat steps 2 to 4 while TREE! = NULL  Step 2: Post ORDER (TREE -> LEFT)  Step 3: POSTORDER (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	-	END OF LOOP
Juander Tecanocesal.  Step 1: Repeat steps 2 to 4 while TREE!= NULL  Step 3: Write TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 8: END  Post-order Travesal  Step 1: Repeat steps 2 to 4 while TREE!= NULL  Step 2: Post or DER (TREE -> RIGHT)  Step 3: Post or DER (TREE -> RIGHT)  Step 3: Post or DER (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	-	\$top 5: END
Step 2: INORDER (TREE -> LEFT)  Step 3: White TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  > Post-order Travegal.  Step 1: Repeat steps 2 to 4 while TREE! = NULL.  Step 2: Post ORDER (TREE -> RIGHT)  Step 3: POSTORDER (TREE -> RIGHT)  Step 4: White TREE -> DATA.  [END OF LOOP]	-	
Step 2: INORDER (TREE -> LEFT)  Step 3: Write TREE -> RIGHT  Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  > Post-order Travegal  Step 1: Repeat steps 2 to 4 while TREE! = NULL.  Step 2: Post ORDER (TREE -> RIGHT)  Step 3: POSTORDER (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]		Inorder Teconocesal.
\$\$\frac{2}{\text{inorder (TREE \rightarrow LEFT)}}\$\$\text{inite TREE \rightarrow RIGHT)}\$\$\text{Linorder (TREE \rightarrow RIGHT)}\$\$\text{Lend of Loop]}\$\$\$\text{dep 5}: END\$\$\$\text{END}\$\$ \text{Post-order Traverfal}\$\$\$\text{dep 1}: Repeat fleps 2 to 4 while TREE! = NULL.\$\$\text{dep 2}: Post order (TREE \rightarrow LEFT)\$\$\$\text{dep 3}: Post order (TREE \rightarrow LEFT)\$\$\$\text{dep 3}: Post order (TREE \rightarrow DATA.\$\$\text{END of Loop}\$\$\text{lend}\$\$		step 1: Repeat steps 2 to 4 while TREE 1= NULL
Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  Post-order Travegal.  Step 1: Repeat steps 2 to 4 while TREE! = NULL.  Step 2: Post ORDER (TREE -> LE FT)  Step 3: POSTORDER (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]		PED 2: INORDER CIRBE -> LEFT)
Step 4: INORDER (TREE -> RIGHT).  [END OF LOOP]  Step 5: END  Post-order Travesful.  Step 1: Repeat steps 2 to 4 while TREE! = NULL.  Step 2: POSTORDER (TREE -> LEFT)  Step 3: POSTORDER (TREE -> RIGHT)  Step 4: Waite TREE -> DATA.  [END OF LOOP]		\$\$\$ 3: Write TREE → RIGHT
\$top 8: END  → Post-cooler Travegal.  \$top 1: Repeat \$tops 2 to 4 while TREE! = NULL.  \$top 2: Postorder (TREE → LEFT)  \$top 3: Postorder (TREE → RIGHT)  \$top 4: Write TREE → DATA.  [END OF LOOP]		Step 4: INORDER (TREE -> RIGHT).
→ Post-order Traverful  Step 1: Repeat steps 2 to 4 while TREE! = NULL.  Step 2: Past ORDER (TREE → LEFT)  Step 3: POSTORDER (TREE → RIGHT)  Step 4: Write TREE → DATA.  [END OF LOOP]		
Step 1: Repeat steps 2 to 4 while TREE! = NULL.  Step 2: POSTORDER (TREE -> LEFT)  Step 3: POSTORDER (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	-	φαρ 5: END
Step 1: Repeat steps 2 to 4 while TREE! = NULL.  Step 2: POSTORDER (TREE -> LEFT)  Step 3: POSTORDER (TREE -> RIGHT)  Step 4: Write TREE -> DATA.  [END OF LOOP]	>	Pact made Tananal
Step 2: POSTORDER (TREE -> LEFT) Step 3: POSTORDER (TREE -> RIGHT) Step 4: Write TREE -> DATA.  [END OF LOOP]		
Step 3: POSTORDER (TREE -> RIGHT)  Step 4: Waite TREE -> DATA.  [END OF LOOP]	- 11	14 2' Pact MODED (TORE - 1 ECT)
(END OF LOOP)		+ 2 PACTABRER (TREE -> PIGHT)
[END OF LOOP]	- 13	to 4: West TORE -> DOTA
	1	[FATD OF LOOP]
фф 3. БМД.	0	
	4	top 5: END.
	-	
	-	
	+	
	+	
	-	

	P.
	Exemple:
	Routing Tables: A santing table is used to link routes in a network
	neturally nave of repeat to range sources
	• To an and the second
	· relet are used in file system directories.
	lees are voidely used for information storage and retrieval in
	System lead to letes
	Trees are used in file system directories.  Trees are widely used for information storage and retrieval in sympleal tables.
•	onclusion: Thus, we understand the concept of burary trees there
	Conclusion: Thus, we understand the concept of binary trees their operations including traversal and its various types and also boren its implementation.
	its implementation
	- particular.
	Outcome: Implement tree data sterreture for real-world application.
•	
The state of the s	
THE RESIDENCE OF	
BL. September	

```
File Edit Search Run Compile Debug Project Options
                                                                                              Window Help
 -[0]-
                                               = BINARYTR.C =
                                                                                                        =1=[#]=
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <malloc.h>
struct node
      int data:
     struct node *left;
     struct node *right;
};
struct node *tree;
void create(struct node *);
struct node *insert(struct node *, int);
void inorder(struct node *);
void preorder(struct node *);
void postorder(struct node *);
      int choice, x;
     struct node *ptr;
void main()
     printf ('so - MELCOME TO IMPLEMENTATION OF BIMORY TREE TROVERSALS
     create(tree);
     do
          printf("\n *** --- opertaions available --- ***");
printf("\n 1. Insert a Mode");
printf("\n 2. Bisplay Inorder Traversal");
printf("\n 3. Bisplay Prender Traversal");
printf("\n 4. Bisplay Posterder Traversal");
printf("\n 5. Exit \n");
printf("\n 5. Exit \n");
printf(" Flease enter your choice ; ");
scanf("\n", &choice);
switch (choice)
           switch (choice)
           {
           case 1:
                 printf("\n Enter the data to be inserted : "); scanf("xd", &x);
                 tree = insert(tree, x);
                 break;
```

```
case 2:
            printf("An Elements in the inorder traversals are : "):
            inorder(tree);
            printf("\n");
            break:
        case 3:
            printf("An Elements in the preorder traversals are : ");
            preorder(tree);
            printf("\n");
            break;
        case 4:
            printf("\n Elements in the postorder traversala are : ");
            postorder(tree);
            printf("\n");
            break;
        default:
            printf("In Please enter a valid option 1, 2, 3, 4.");
            break:
    } while (choice != 5);
void create(struct node *tree)
    tree = NULL;
// Function for inserting a new node
struct node *insert(struct node *tree, int x)
    struct node *p, *temp, *root;
    p = (struct node *)malloc(sizeof(struct node));
    p->data = x;
    p->left = NULL;
    p->right = NULL;
    if (tree == NULL)
        tree = p;
        tree->left = NULL:
        tree->right = NULL;
    else
```

```
root = NULL;
        temp = tree;
        while (temp != NULL)
            root = temp;
            if (x < temp->data)
    temp = temp->left;
            else
                temp = temp->right;
        if (x < root->data)
            root->left = p;
        else
            root->right = p;
    return tree;
// Function for Inorder Traversals
void inorder(struct node *tree)
    if (tree != NULL)
        inorder(tree->left);
        printf(" zd \t", tree->data);
        inorder(tree->right);
    }
// Function for Preorder Traversals
void preorder(struct node *tree)
    if (tree != NULL)
        printf(" xd st", tree->data);
        preorder(tree->left);
        preorder(tree->right);
    }
// Function for Postorder Traversals
void postorder(struct node *tree)
```

```
if (tree != NULL)
        inorder(tree->left);
        printf(" xd \t", tree->data);
        inorder(tree->right);
// Function for Preorder Traversals
void preorder(struct node *tree)
    if (tree != NULL)
        printf(" xd \t", tree->data);
preorder(tree->left);
        preorder(tree->right);
// Function for Postorder Traversals
void postorder(struct node *tree)
    if (tree != NULL)
        postorder(tree->left):
        postorder(tree->right);
printf(" >d >t", tree->data);
    F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu
```

```
--- WELCOME TO IMPLEMENTATION OF BINARY TREE TRAVERSALS ---
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 1
Enter the data to be inserted: 77
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice: 1
Enter the data to be inserted: 84_
```

```
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 1
Enter the data to be inserted: 99
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 1
Enter the data to be inserted: 22
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
```

```
Please enter your choice : 1
Enter the data to be inserted: 55
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 2
                                              55
                                                       77
                                                               84
                                                                       99
Elements in the inorder traversala are: 22
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice :
Elements in the preorder traversala are: 77 22
                                                       55
                                                               84
                                                                       99
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice :
Please enter your choice:
Elements in the postorder traversala are: 55
                                                               99
                                                       22
                                                                       84
```