```
#include<stdio.h>
#include<stdlib.h>
struct treenode {
	struct treenode *lchild;
	int info;
	struct treenode *rchild;
};
struct listnode {
	int info;
	struct listnode* next;
};
struct listnode *create_list(struct
listnode*start,int n);
void display(struct listnode *start);
struct listnode *addatbeg(struct listnode*start,
int data);
struct listnode *addatend(struct listnode *start,
int data);
struct treenode *construct(struct listnode *inptr,
struct listnode *postptr, int num);
void inorder(struct treenode *tree);
void postorder(struct treenode *tree);
int height(struct treenode *ptr);
void printLevelOrder(struct treenode* root);
void printCurrentLevel(struct treenode *root,
int level);
void left_to_right(struct treenode* root, int
level);
void right_to_left(struct treenode *root, int
level);
int findDepth(struct treenode* root, int x);
int main() {
	struct treenode *tree;
	struct listnode *inptr, *postptr;
	int n,choice,h;
	printf("Enter the number of elements in
the list: ");
	scanf("%d", &n);
	printf("Enter the inorder
expression:\n");
	inptr=create_list(inptr,n);
	printf("The inorder expression entered
is: ");
	display(inptr);
	printf("Enter the postorder
expression:\n");
	postptr=create_list(postptr,n);
	printf("The inorder expression entered
is: ");
	display(postptr);
	tree=construct(inptr,postptr,n);
	printf("Inorder traversal: ");
	inorder(tree);
	printf("\nPost order traversal: ");
	postorder(tree);
	printf("\n");
	while(choice!=5) {
		printf("1. Find the height if the
tree\n2. Find the depth of the tree\n3. Perform
level order traversal\n4. Perform Spiral
traversal\n5. Exit\n");
		scanf("%d", &choice);
		printf("\n");
		switch(choice) {

			case 1: {

				int h;


h=height(tree);

				printf("The
height of the tree is %d\n", h);

				printf("\n");

				break;

			}

			case 2: {

				int x,d;

				printf("Enter
the element whose depth needs to be found
out: ");

				scanf("%d",
&x);


d=findDepth(tree, x);

				printf("Depth
is %d\n", d);

				printf("\n");

				break;

			}

			case 3: {

				printf("The
level order traversal is: ");


printLevelOrder(tree);

				printf("\n");

				break;
			}
			case 4: {
				int i,flag;
```

```
                                        printf("The
Spiral traversal of the tree is: ");
                                        int
h=height(tree);

                                    for(i = 1; i <= h; i++)
                                    {
                                        if(i%2 != 0)
                                        {
                                            flag = 0;

left_to_right(tree,i);}
                                        if(i%2 == 0)
                                        {
                                            flag = 1;
                                right_to_left(tree,i);
                                        }
                                    }
                                    printf("\n");
                                        break;
                                }
                                case 5: {
                                        exit(1);
                                }
                                default:
                        printf("Re-enter your
choice.\n");
                        }
                }
                return 0;
}
struct listnode * create_list(struct listnode *
start, int n){
int i,data;
start=NULL;
if(n==0)
return start;
printf("Enter the element to be inserted : ");
scanf("%d",&data);
start=addatbeg(start,data );
for(i=2;i<=n;i++) {
printf("Enter the element to be inserted : " );
scanf("%d",&data);
start=addatend(start,data );
}
return start;
}
void display(struct listnode *start) {
struct listnode *p;
if(start==NULL) {

                        printf("Empty List\n");

                }



                p=start;

                while(p!=NULL) {

                        printf("%3d", p->info);

                        p=p->next;

                }

                printf("\n\n");
}


struct listnode *addatbeg(struct listnode *start,
int data) {

                struct listnode *tmp;

                tmp=(struct listnode
*)malloc(sizeof(struct listnode));

                tmp->info = data;

                tmp->next = start;

                start = tmp;

                return start;

}



struct listnode *addatend(struct listnode *start,
int data) {

                struct listnode *tmp, *p;

                tmp= (struct listnode
*)malloc(sizeof(struct listnode));

                tmp->info = data;

                p = start;

                while(p->next!=NULL) {

                        p = p->next;

                }

                p->next=tmp;

                tmp->next = NULL;

                return start;

}
```

```
struct treenode *construct(struct listnode *inptr,
struct listnode *postptr, int num) {

        struct treenode *temp;

        struct listnode *q,*ptr;

        int i,j;

        if(num==0)

                return NULL;

        ptr=postptr;

        for(i=1;i<num;i++)

                ptr=ptr->next;



        temp=(struct treenode
*)malloc(sizeof(struct treenode));

        temp->info=ptr->info;

        temp->lchild=NULL;

        temp->rchild=NULL;
        if(num==1)

                return temp;
        q=inptr;
        for(i=0;q->info!=ptr->info;i++)
        q=q->next;
        temp->lchild=construct(inptr,postptr,i);
        for(j=1;j<=i;j++)
                postptr=postptr->next;
temp->rchild=construct(q->next,postptr,num-i-
1);
        return temp;
}
void inorder(struct treenode *ptr) {

        if(ptr==NULL)
        return;
        inorder(ptr->lchild);
        printf("%3d", ptr->info);
        inorder(ptr->rchild);
}
void postorder(struct treenode *ptr) {
        if(ptr==NULL)
                return;
        postorder(ptr->lchild);

        postorder(ptr->rchild);

        printf("%3d", ptr->info);
}
int height(struct treenode *ptr) {

        int h_left,h_right;

        if(ptr==NULL)

                return 0;

        h_left=height(ptr->lchild);

        h_right=height(ptr->rchild);

        if(h_left>h_right)

                return 1+h_left;

        else

                return 1+h_right;

}
/* Function to print level order traversal a tree*/

void printLevelOrder(struct treenode* root)

{

   int h = height(root);

   int i;

   for (i = 1; i <= h; i++)

      printCurrentLevel(root, i);

}
/* Print nodes at a current level */

void printCurrentLevel(struct treenode *root,
int level)

{

   if (root == NULL)

      return;

   if (level == 1)

      printf("%3d", root->info);
   else if (level > 1) {
      printCurrentLevel(root->lchild, level - 1);
      printCurrentLevel(root->rchild, level - 1);

   }
}
void left_to_right(struct treenode* root, int
level) {
```

```
    if (root != NULL) {

        if (level == 1) {

            printf("%3d", root->info);

        }
                    else if (level > 1) {
            left_to_right(root->lchild, level-1);
            left_to_right(root->rchild, level-1);
        }
    }
}

void right_to_left(struct treenode *root, int
level) {
    if(root!=NULL) {
        if(level==1) {
            printf("%3d", root->info);
        }
else {
 right_to_left(root->rchild, level-1);
 right_to_left(root->lchild, level-1);
 }
 }
}
int findDepth(struct treenode* root, int x) {
    // Base case
    if (root == NULL) {
        return -1;
    }
           // Initialize distance as -1
 int dist = -1;
 // Check if x is current node=
 if ((root->info == x) || (dist =
findDepth(root->lchild, x)) >= 0 || (dist =
findDepth(root->rchild, x)) >= 0)
        return dist + 1;
    return dist;
}
```

```
O/P : /tmp/dJnQTFLauf.o
Enter the number of elements in the list: 9
Enter the inorder expression:
Enter the element to be inserted : 1
Enter the element to be inserted : 2
Enter the element to be inserted : 3
Enter the element to be inserted : 4
5
Enter the element to be inserted : 64

5
Enter the element to be inserted : Enter the
element to be inserted :
7
Enter the element to be inserted : 8
Enter the element to be inserted : 9
The inorder expression entered is:   1  2  3  4
4  5  7  8  9
Enter the postorder expression:
Enter the element to be inserted : 9
8Enter the element to be inserted :
7
Enter the element to be inserted : 6
Enter the element to be inserted : 5
Enter the element to be inserted : 4
Enter the element to be inserted : 3
Enter the element to be inserted : 2
Enter the element to be inserted : 1
Enter the element to be inserted : 8
The inorder expression entered is:   9  7  6  5
4  3  2  1  8

Inorder traversal:   9  2  3  4  7  5  6  8  1
Post order traversal:   9  7  6  5  4  3  2  1  8
1. Find the height if the tree
2. Find the depth of the tree
3. Perform level order traversal
4. Perform Spiral traversal
5. Exit
1
The height of the tree is 6

1. Find the height if the tree
2. Find the depth of the tree
3. Perform level order traversal
4. Perform Spiral traversal
5. Exit
2
Enter the element whose depth needs to be
found out: 6
Depth is 5

1. Find the height if the tree
2. Find the depth of the tree
3. Perform level order traversal
4. Perform Spiral traversal
5. Exit
3
The level order traversal is:   8  2  1  9  3  4  5
7  6
1. Find the height if the tree
2. Find the depth of the tree
3. Perform level order traversal
4. Perform Spiral traversal
5. Exit
4
The Spiral traversal of the tree is:   8  1  2  9  3
4  5  6  7
1. Find the height if the tree
2. Find the depth of the tree
3. Perform level order traversal
4. Perform Spiral traversal
5. Exit
```