Programming Examples

1.

```
#include <stdio.h>
#include <malloc.h>

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

struct node *tree;
void create_tree(struct node *tree);
struct node *insert_element(struct node *tree,int val);
void preorder(struct node *tree);
void inorder(struct node *tree);
struct node *search(struct node *tree);
void postorder(struct node *tree);
struct node *smallest(struct node *tree);
struct node *biggest(struct node *tree);
int totalnodes(struct node *tree);
int totalexternal(struct node *tree);
int totalinternal(struct node *tree);
int height(struct node *tree);
struct node *findmirror(struct node *tree);
struct node *delete_tree(struct node *tree);
struct node *delete_element(struct node *tree, int val);
int main()
{
        int option,val;
        struct node *result;
        do{
        printf("\n\n");
        printf("**********main menu**********\n");
        printf("1.insert element\n");
        printf("2.preorder traversal\n");
        printf("3.inorder traversal\n");
        printf("4.postorder traversal\n");
        printf("5.find the smallest number\n");
        printf("6.find the largest number\n");
        printf("7.delete the element\n");
        printf("8.count the total number of nodes\n");
```

```
printf("9.count the total number of external nodes\n");
printf("10.count the total number of internal nodes\n");
printf("11.determine the height of the tree\n");
printf("12.find the mirror image of the tree\n");
printf("13.delete the tree\n");
printf("14.exit\n");
printf("Enter the option : ");
scanf("%d",&option);
switch(option)
{
case 1:
        while(1)
        {
        printf("Enter the value of the new node : ");
        scanf("%d",&val);
        if(val==-1)
                break;
        else
        tree=insert_element(tree,val);
        }
        break;
case 2:
        printf("preorder traversal : ");
        preorder(tree);
        break;
case 3:
        printf("inorder traversal : ");
        inorder(tree);
        break;
case 4:
        printf("postorder traversal : ");
        postorder(tree);
        break;
case 5:
        result=smallest(tree);
        printf("smallest element is %d",result->data);
        break;
case 6:
        result=biggest(tree);
        printf("biggest element is %d",result->data);
        break;
case 7:
        printf("Enter the value to be deleted");
```

```
                scanf("%d",&val);
                tree=delete_element(tree,val);
                break;
        case 8:
                printf("The total number of nodes is %d",totalnodes(tree));
                break;
        case 9:
                printf("The total number of external nodes is %d",totalexternal(tree));
                break;
        case 10:
                printf("The total number of internal nodes is %d",totalinternal(tree));
                break;
        case 11:
                printf("Height of the tree is %d",height(tree));
                break;
        case 12:
                findmirror(tree);
                printf("complete");
                break;
        case 13:
                tree=delete_tree(tree);
                printf("deleted");
                break;
        }
        }while(option!=14);



}

void create_tree(struct node *tree)
{
        tree=NULL;
}
struct node *insert_element(struct node *tree,int val)
{
        struct node *ptr, *nodeptr ,*parentptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr->data=val;
        ptr->left=NULL;
        ptr->right=NULL;
        if(tree==NULL)
        {
```

```
                tree=ptr;
                tree->left=NULL;
                tree->right=NULL;
        }
        else
        {
                parentptr=NULL;
                nodeptr=tree;
                while(nodeptr!=NULL)
                {
                        parentptr=nodeptr;
                        if(val<nodeptr->data)
                                nodeptr=nodeptr->left;
                        else
                                nodeptr=nodeptr->right;
                }
                if(val< parentptr->data)
                        parentptr->left=ptr;
                else
                        parentptr->right=ptr;
        }
        return tree;
}

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

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

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

struct node *smallest(struct node *tree)
{
        if((tree==NULL) || (tree->left==NULL))
                return tree;
        else
                return smallest(tree->left);
}

struct node *biggest(struct node *tree)
{
        if((tree==NULL) || (tree->right==NULL))
                return tree;
        else
                return biggest(tree->right);
}


int totalnodes(struct node *tree)
{
        if(tree==NULL)
                return 0;
        else
                return (totalnodes(tree->left) + totalnodes(tree->right)+1);
}

int totalexternal(struct node *tree)
{
        if(tree==NULL)
                return 0;
        else if((tree->left==NULL) && (tree->right==NULL))
                return 1;
        else
```

```
                return (totalexternal(tree->left) + totalexternal(tree->right));
}

int totalinternal(struct node *tree)
{
        if(tree==NULL)
                return 0;
        else if((tree->left==NULL) && (tree->right==NULL))
                return 0;
        else
                return (totalinternal(tree->left) + totalinternal(tree->right)+1);
}

int height(struct node *tree)
{
        int left,right;
        if(tree==NULL)
                return 0;
        else
        {
                left=height(tree->left);
                right=height(tree->right);
                if(left>right)
                        return (left+1);
                else
                        return (right+1);
        }

}

struct node *findmirror(struct node *tree)
{
        struct node *temp;
        if(tree!=NULL)
        {
                findmirror(tree->left);
                findmirror(tree->right);
                temp=tree->left;
                tree->left=tree->right;
                tree->right=temp;
        }
}
```

```
struct node *delete_tree(struct node *tree)
{
        if(tree!=NULL)
        {
                delete_tree(tree->left);
                delete_tree(tree->right);
                free(tree);
        }
        return tree;
}

struct node *delete_element(struct node *tree, int val)
{
    struct node *cur, *parent, *suc, *psuc, *ptr;

    if (tree == NULL)
    {
        printf("\n The tree is empty ");
        return tree;
    }

    parent = tree;
    if (val == parent->data)
    {
        cur = parent->right;
        parent = cur;
        parent->left->left = tree->left;
        tree = parent;
        while (parent->right != NULL)
        {
            cur = parent->right;
            parent = cur;
        }
        printf("The node %d has been successfully deleted from the tree!\n", val);
        return tree;
    }

    cur = parent;
    while (cur != NULL && val != cur->data)
    {
        parent = cur;
        if (val < parent->data)
            cur = parent->left;
```

```
        else
            cur = parent->right;
    }
    if (cur == NULL)
    {
        printf("\n The value to be deleted is not present in the tree!\n");
        return (tree);
    }
    if (cur->left == NULL)
        ptr = cur->right;
    else if (cur->right == NULL)
        ptr = cur->left;
    else
    {
        psuc = cur;
        suc = cur->left;
        while (suc->left != NULL)
        {
            psuc = suc;
            suc = suc->left;
        }
        if (cur == psuc)
        {

            suc->left = cur->right;
        }
        else
        {

            suc->left = cur->left;
            psuc->left = suc->right;
            suc->right = cur->right;
        }
        ptr = suc;
    }
    if (parent->left == cur)
        parent->left = ptr;
    else
        parent->right = ptr;
    free(cur);

    printf("The node %d has been successfully deleted from the tree!\n", val);
    return tree;
```

}

```
**********main menu***********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 1
Enter the value of the new node : 2
Enter the value of the new node : 1
Enter the value of the new node : 3
Enter the value of the new node : 4
Enter the value of the new node : -1
```

```
**********main menu***********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 2
preorder traversal : 2134
```

```
**********main menu***********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 3
inorder traversal : 1234
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 4
postorder traversal : 1432
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 5
smallest element is 1
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 6
biggest element is 4
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 7
Enter the value to be deleted3
The node 3 has been successfully deleted from the tree!
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 8
The total number of nodes is 4
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 9
The total number of external nodes is 2
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 10
The total number of internal nodes is 2
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 11
Height of the tree is 3
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 12
complete
```

```
**********main menu**********
1.insert element
2.preorder traversal
3.inorder traversal
4.postorder traversal
5.find the smallest number
6.find the largest number
7.delete the element
8.count the total number of nodes
9.count the total number of external nodes
10.count the total number of internal nodes
11.determine the height of the tree
12.find the mirror image of the tree
13.delete the tree
14.exit
Enter the option : 13
deleted
```

Programming Exercises

2.

```
#include <stdio.h>
#include <malloc.h>

struct node
{
        int data;
        struct node *left;
        struct node *right;
};
struct node *tree;
void create_tree(struct node *tree);
int totalnodes(struct node *tree);
struct node *insert(struct node *tree, int val);
int main()
{
        int val;
        create_tree(tree);
        while(1)
        {
                printf("Enter the element of node : ");
                scanf("%d",&val);
                if(val==-1)
                        break;
                tree=insert(tree,val);
        }

        printf("The number of total node is %d\n",totalnodes(tree));
}
```

```
void create_tree(struct node *tree)
{
        tree=NULL;
}

struct node *insert(struct node *tree, int val)
{
        struct node *ptr,*parentptr,*nodeptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr->data=val;
        ptr->left=NULL;
        ptr->right=NULL;
        if(tree==NULL)
        {
                tree=ptr;
                tree->right=NULL;
                tree->left=NULL;
        }
        else
        {
                parentptr=NULL;
                nodeptr=tree;
                while(nodeptr!=NULL)
                {
                        parentptr=nodeptr;
                        if(val<nodeptr->data)
                                nodeptr=nodeptr->left;
                        else
                                nodeptr=nodeptr->right;
                }
                if(val<parentptr->data)
                        parentptr->left=ptr;
                else
                        parentptr->right=ptr;
        }
        return tree;
}

int totalnodes(struct node *tree)
{
        if(tree==NULL)
```

```
        return 0;
    else
        return (totalnodes(tree->left)+totalnodes(tree->right)+1);
}
```