02/01/2021

**Experiment No:19**

**BINARY SEARCH TREE**

**AIM:**

Create a binary search tree with the following operations:

1. Insert a new node.

2. Inorder traversal.

3. Preorder traversal.

4. Postorder traversal.

5. Delete a node.

6. Count the number of leaf nodes

**DATA STRUCTURES USED:**

Tree using Linked List

**ALGORITHM:**

Algorithm Insert()

ptr=root flag = False

1. While ptr != NULL

2. If ITEM <= ptr->DATA

3. ptr1 = ptr

4. ptr = ptr->LC

5. Else if ITEM > ptr->DATA

6. ptr1 = ptr

7. ptr = ptr->RC

8. Else

9. Flag=True

10. print “Item already exists”

11. Endwhile

12. If ptr = NULL

13. new= GetNode(NODE)

14. new->LC = NULL

15. new->RC = NULL

16. new->DATA = ITEM

17. If ptr1->DATA < ITEM

18. ptr1->RC = new

19. if ptr1->DATA>ITEM

20. ptr1->LC = new

21. Endif

22. EndIf

Algorithm inorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. inorder\_traversal(ptr->LC)

4. print ptr->DATA

5. inorder\_traversal(ptr->RC)

6. Endif

Algorithm preorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. print ptr->DATA

4. preorder\_traversal(ptr->LC)

5. preorder\_traversal(ptr->RC)

6. Endif

Algorithm postorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. postorder\_traversal(ptr->LC)

4. postorder\_traversal(ptr->RC)

5. print ptr->DATA

6. Endif

Algorithm successor(ptr)

1. ptr1 = ptr->RC

2. If ptr1 != NULL

3. While ptr1->LC != NULL

4. ptr1 = ptr1->LC

5. Endwhile

6. Endif

7. Return(ptr1)

Algorithm Delete()

1. ptr = ROOT

2. flag = false

3. While ptr != NULL and flag = false

4. If ITEM < ptr->DATA

5. parent = ptr

6. ptr = ptr->LC

7. Else if ITEM > ptr->DATA

8. parent = ptr

9. ptr = ptr->RC

10. Else

11. flag = true

12. Endif

13. Endwhile

14. If flag = false

15. print "ITEM doesn’t exist"

16. Exit

17. Endif

18. If ptr->LC = NULL and ptr->RC = NULL

19. CASE = 1

20. Else If ptr->LC != NULL and ptr->RC != NULL

21. CASE = 3

22. Else

23. CASE = 2

24. Endif

25. Endif

26. If CASE = 1

27. If parent->LC = ptr

28. parent->LC = NULL

29. Else

30. parent->RC = NULL

31. Endif

32. ReturnNode(ptr)

33.EndIf

34. if CASE = 2

35. If parent->LC = ptr

36. If ptr->LC = NULL

37. parent->LC = ptr->RC

38. Else

39. parent->LC = ptr->LC

40. Endif

41. Else

42. If ptr->LC = NULL

43. parent->RC = ptr->RC

44. Else

45. parent->RC = ptr->LC

46. Endif

47. Endif

48. ReturnNode(ptr)

49. If CASE=3

50. ptr1 = successor(ptr)

51. ITEM1 = ptr1->DATA

52. DeleteBST(ITEM1)

53. ptr->DATA = ITEM1

54. Endif

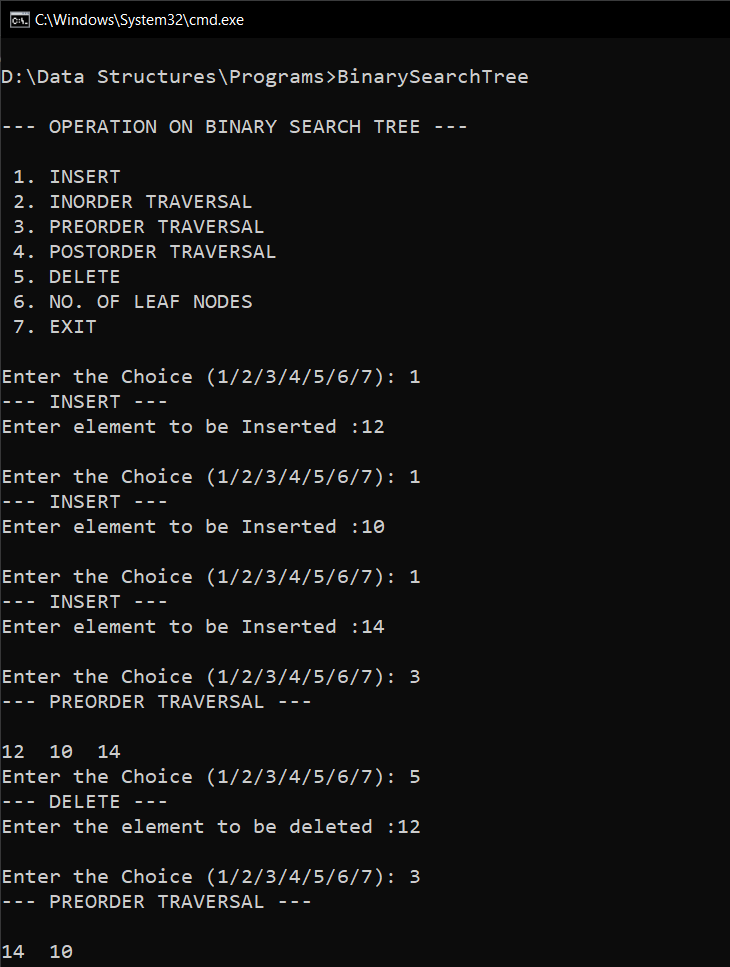
Algorithm no\_of\_leaf\_nodes(root)

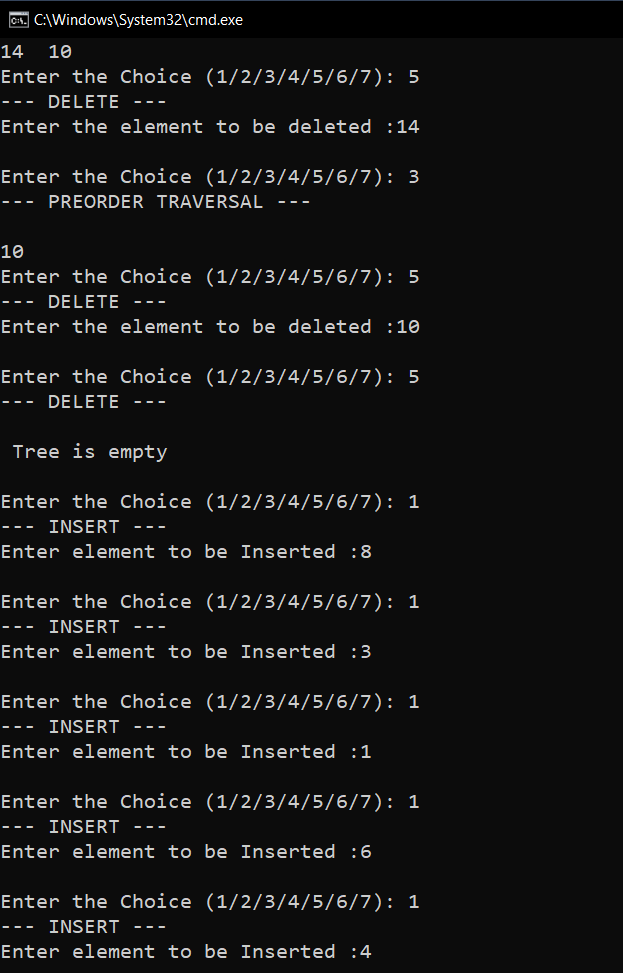
1. ptr=root
2. if ptr =NULL
3. return 0;
4. else if ptr->lchild == NULL && ptr->rchild == NULL
5. return 1;
6. Else
7. return no\_of\_leaf\_nodes(ptr->LC)+ no\_of\_leaf\_nodes(ptr->RC)

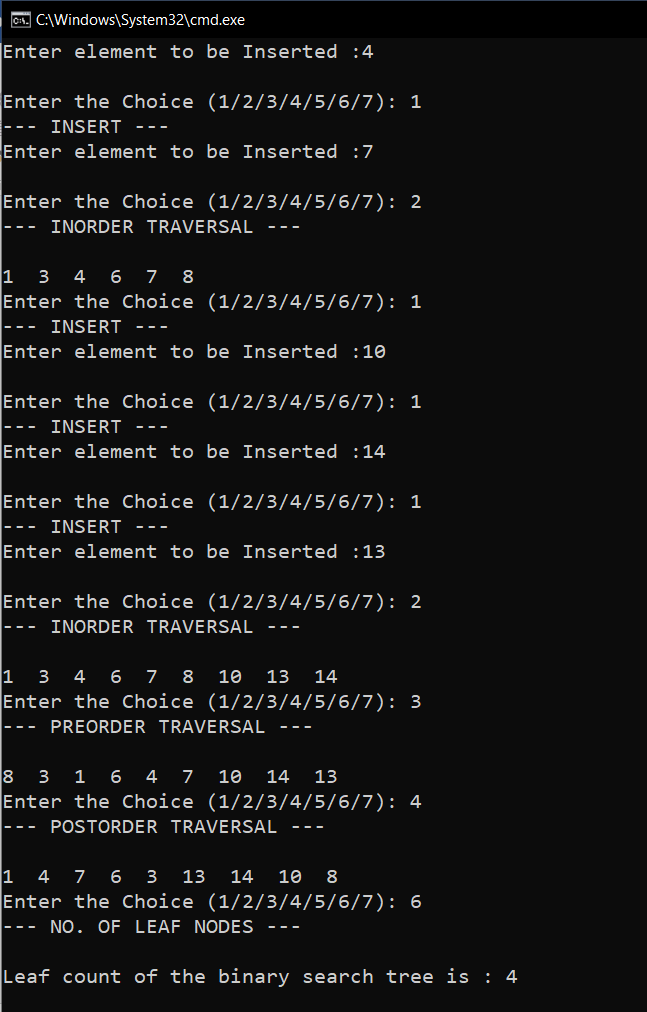
**PROGRAM:**

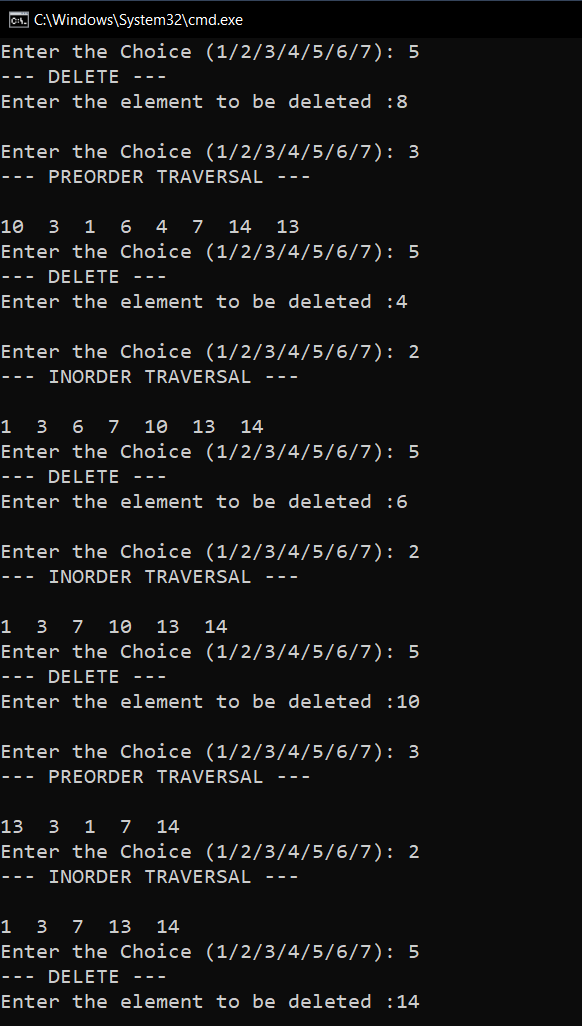
#include<stdio.h>  
#include<stdlib.h>  
  
struct node{  
 int data;  
 struct node \*lchild;  
 struct node \*rchild;  
};  
  
void Insert(struct node\* root,int item){  
 struct node\* ptr=root;  
 struct node\* ptr1;  
 int flag=0;  
 while(ptr!=NULL && flag == 0){  
 if(item<ptr->data){  
 ptr1=ptr;  
 ptr=ptr->lchild;  
 }else if(item>ptr->data){  
 ptr1=ptr;  
 ptr=ptr->rchild;  
 }else{  
 flag=1;  
 printf("\n ITEM already exists \n ");  
 }  
 }  
 if(ptr==NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=item;  
 new->lchild=NULL;  
 new->rchild=NULL;  
 if(ptr1->data<item){  
 ptr1->rchild=new;  
 }  
 if(ptr1->data>item){  
 ptr1->lchild=new;  
 }  
 }  
}  
void inorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 inorder\_traversal(ptr->lchild);  
 printf("%d ",ptr->data);  
 inorder\_traversal(ptr->rchild);  
 }  
}  
void preorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 printf("%d ",ptr->data);  
 preorder\_traversal(ptr->lchild);  
 preorder\_traversal(ptr->rchild);  
 }  
}  
void postorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 postorder\_traversal(ptr->lchild);  
 postorder\_traversal(ptr->rchild);  
 printf("%d ",ptr->data);  
 }  
}  
struct node\* successor(struct node\* ptr){  
 struct node\* ptr1;  
 ptr1=ptr->rchild;  
 if(ptr1!=NULL){  
 while(ptr1->lchild!=NULL){  
 ptr1=ptr1->lchild;  
 }  
 }  
 return(ptr1);  
}  
  
void Delete(struct node\* root,int item){  
 struct node\* ptr=root;  
 struct node\* ptr1;  
 struct node\* parent=NULL;  
 int flag=0,temp;  
 while(ptr!=NULL && flag == 0){  
 if(item<ptr->data){  
 parent=ptr;  
 ptr=ptr->lchild;  
 }else if(item>ptr->data){  
 parent=ptr;  
 ptr=ptr->rchild;  
 }else{  
 flag=1;  
 }  
 }  
 if(flag==0){  
 printf(" \nITEM doesn't exists\n");  
 }else{  
 if(ptr->lchild==NULL && ptr->rchild==NULL){  
 if(parent->lchild==ptr){  
 parent->lchild=NULL;  
 }  
 if(parent->rchild==ptr){  
 parent->rchild=NULL;  
 }  
 free(ptr);  
 }else if(ptr->lchild!=NULL && ptr->rchild!=NULL){  
 ptr1 = successor(ptr);  
 temp =ptr1->data;  
 Delete(root,temp);  
 ptr->data=temp;  
 free(ptr1);  
 }else{  
 if(parent->lchild==ptr){  
 if(ptr->lchild==NULL){  
 parent->lchild=ptr->rchild;  
 }else{  
 parent->lchild=ptr->lchild;  
 }  
 }else if(parent->rchild==ptr){  
 if(ptr->lchild==NULL){  
 parent->rchild=ptr->rchild;  
 }else{  
 parent->rchild=ptr->lchild;  
 }  
 }  
 free(ptr);  
 }  
 }  
}  
  
int no\_of\_leaf\_nodes(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr == NULL){  
 return 0;  
 }else if(ptr->lchild == NULL && ptr->rchild == NULL){  
 return 1;  
 }else{  
 return no\_of\_leaf\_nodes(ptr->lchild)+no\_of\_leaf\_nodes(ptr->rchild);  
 }  
}  
  
void main(){  
 int n,item,var=0;  
 char ans='y';  
 struct node\* root = NULL;  
 printf("\n--- OPERATION ON BINARY SEARCH TREE --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. INORDER TRAVERSAL\n");  
 printf(" 3. PREORDER TRAVERSAL\n");  
 printf(" 4. POSTORDER TRAVERSAL\n");  
 printf(" 5. DELETE \n");  
 printf(" 6. NO. OF LEAF NODES \n");  
 printf(" 7. EXIT \n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6/7): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &item);  
 if(root==NULL){  
 root = (struct node\*)malloc(sizeof(struct node));  
 root->lchild=NULL;  
 root->rchild=NULL;  
 root->data=item;  
 }else{  
 Insert(root,item);  
 }  
 var++;  
 break;  
 case 2:printf("--- INORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 inorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 3:printf("--- PREORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 preorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 4:printf("--- POSTORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 postorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 5:printf("--- DELETE ---\n");  
 if(root!=NULL){  
 printf("Enter the element to be deleted :");  
 scanf("%d", &item);  
 if(root->data==item && root->lchild==NULL && root->rchild == NULL){  
 root = NULL;  
 }else if(root->data==item && root->lchild==NULL){  
 root=root->rchild;  
 }else if(root->data==item && root->rchild==NULL){  
 root=root->lchild;  
 }else{  
 Delete(root,item);  
 }  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 6:printf("--- NO. OF LEAF NODES ---\n");  
 printf("\nLeaf count of the binary search tree is : %d\n",no\_of\_leaf\_nodes(root));  
 break;  
 case 7:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

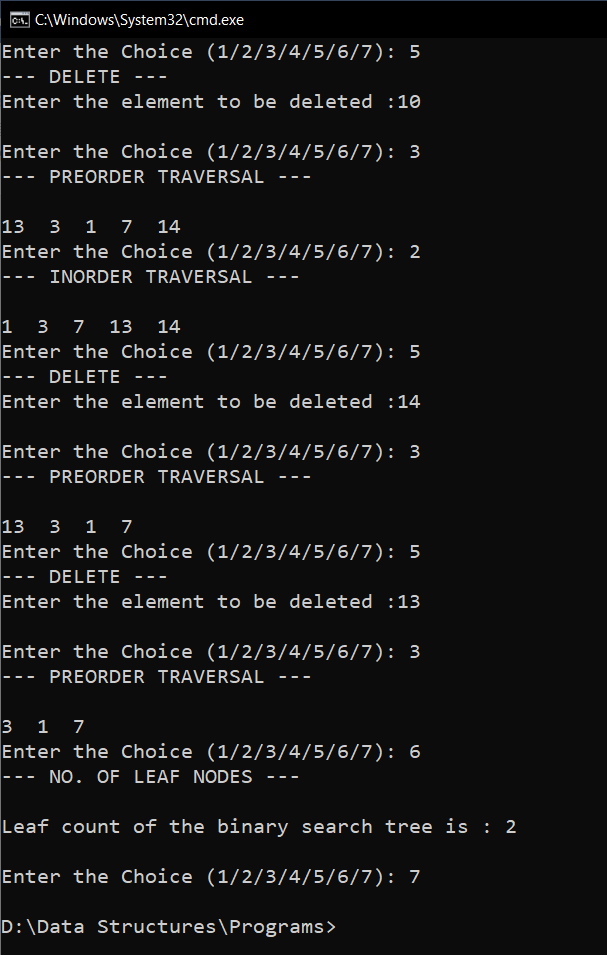
**OUTPUT:**











**RESULT:**

The given operations are performed on a binary search tree.