04/01/2021

**Experiment No:18**

**BINARY TREE**

**AIM:**

Create a Binary tree with the following operations:

1. Insert a new node.

2. Inorder traversal.

3. Preorder traversal.

4. Postorder traversal.

5. Delete a node.

**DATA STRUCTURES USED:**

Tree using Linked List

**ALGORITHM:**

Algorithm build\_tree(root)

//ptr=root

1. If ptr != NULL

2. ptr->DATA=item

3. Read option if Node has a left child

4. If opion = yes

5. ptr->LC = GetNode(NODE)

6. build\_tree(ptr->LC)

7. Else

8. ptr->LC = NULL

9. Endif

10. Read option if Node has a right child

11. If option = yes

12. ptr->RC = GetNode(NODE)

13. build\_tree(ptr->RC)

14. Else

15. ptr->RC = NULL

16. Endif

17. Endif

Algorithm search\_link(ptr, KEY)

1. If ptr->DATA != KEY

2. If ptr->LC != NULL

3. ptr1 = SearchLink(ptr->LC, KEY)

4. If ptr1 != NULL

5. Return ptr1

6. Endif

7. Endif

8. If ptr->RC != NULL

9. ptr1 = SearchLink(ptr->RC, KEY)

10. If ptr1 != NULL

11. Return ptr1

12. Endif

13. Endif

14. Return NULL

15. Else

16. Return ptr

17. Endif

Algorithm insert\_tree(ROOT,KEY)

1. ptr = search\_link(ROOT, KEY)

2. If ptr = NULL

3. Print "KEY not found"

4. Exit

5. Else

6. If ptr->LC = NULL or ptr->RC = NULL

7. Read option insert as left child or right child

8. If option = left

9. If ptr->LC = NULL

10. new= GetNode(NODE)

11. new->LC = NULL

12. new->RC = NULL

13. new->DATA=ITEM

14. Else

15. Print "KEY has a left child"

16. Endif

17. Else if option = right

18. If ptr->RC = NULL

19. new= GetNode(NODE)

20. new->LC = NULL

21. new->RC = NULL

22. new->DATA=ITEM

23. Else

24. Print "KEY has a right child"

25. Endif

26. Endif

27. Else

28. Print "KEY has both left child and right child"

29. Endif

30. 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 search\_parent(ptr, parent,KEY)

1. If ptr->DATA != KEY

2. If ptr->LC != NULL

3. parent = SearchParent(ptr->LC, KEY, ptr)

4. If parent != NULL

5. return parent

6. Endif

7. Endif

8. If ptr->RC != NULL

9. parent = SearchParent(ptr->RC, KEY, ptr)

10. If parent != NULL

11. return parent

12. Endif

13. Endif

14. return NULL

15. Else

16. return parent

17. Endif

Algorithm DeleteTree(ROOT,KEY)

1. parent = search\_parent(ROOT, ROOT,KEY)

2. If parent! = NULL

3. Ptr1=parent->LC

4. Ptr2=parent->RC

5. If ptr1 != NULL and ptr1->DATA = KEY

6. If ptr1->LC = NULL and ptr1->RC = NULL

7. parent->LC = NULL

8. Else

9. Print "KEY is not a leaf node"

10. Endif

11. Else

12. If ptr2->LC = NULL and ptr2->RC = NULL

13. parent->RC = NULL

14. Else

15. Print "KEY is not a leaf node"

16. Endif

17. Endif

18. Else

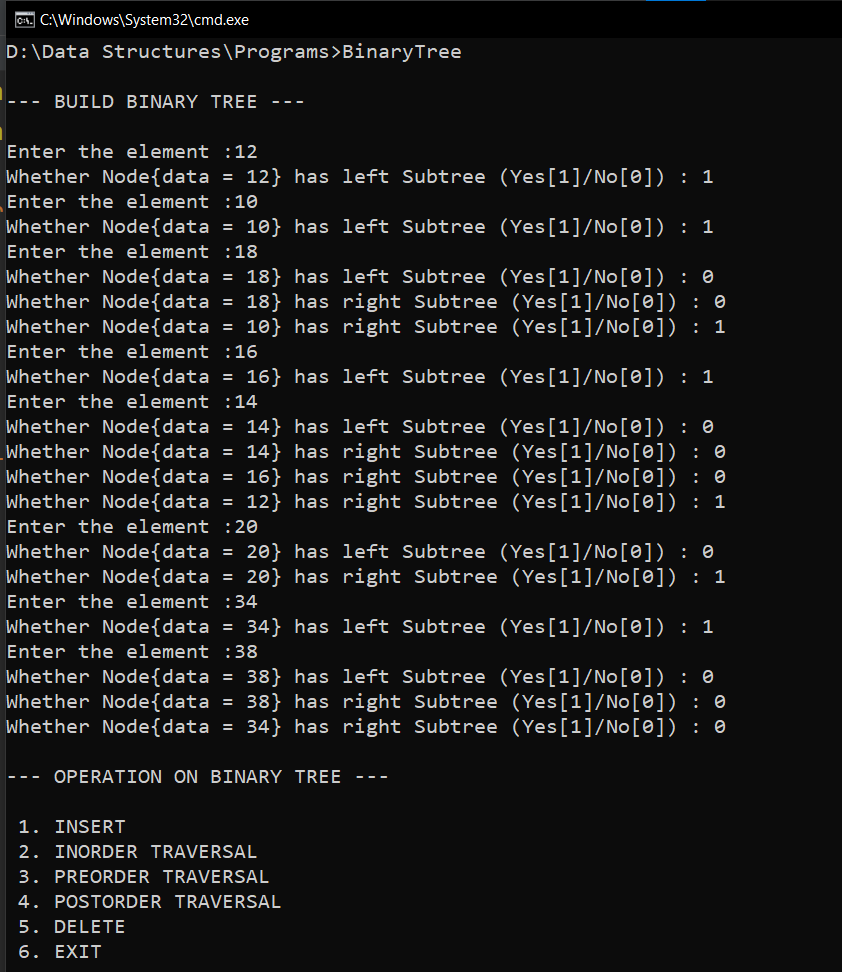
19. Print "KEY not found"

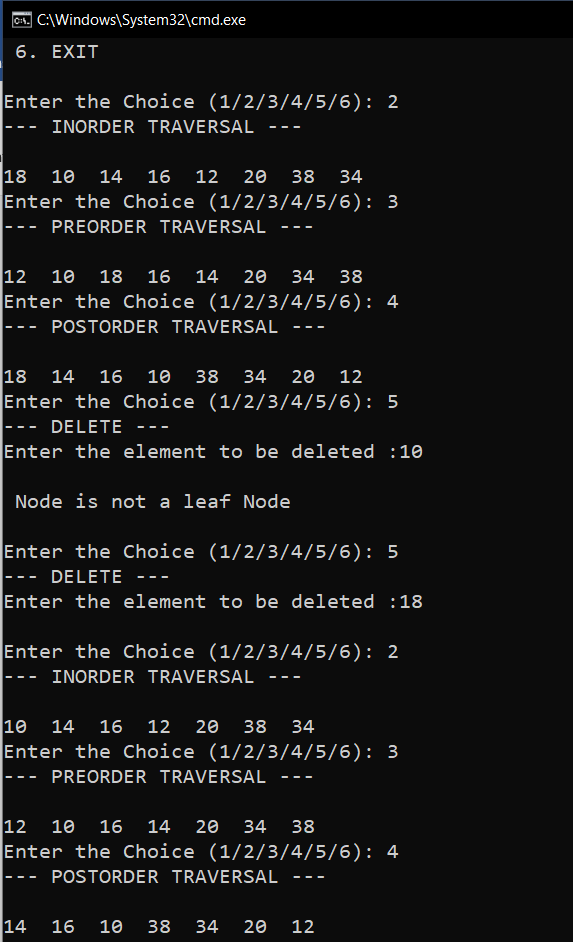
20. Endif

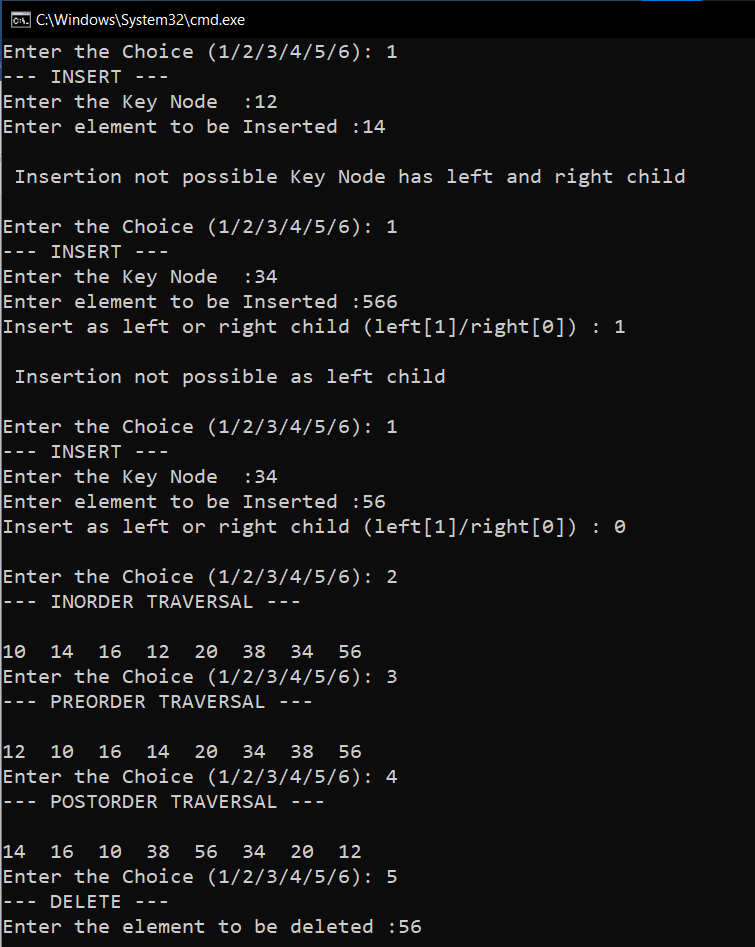
**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
  
struct node{  
 int data;  
 struct node \*lchild;  
 struct node \*rchild;  
};  
  
void build\_tree(struct node\* ptr){  
 int item ,ans;  
 if(ptr!=NULL){  
 printf("Enter the element :");  
 scanf("%d", &item);  
 ptr->data=item;  
 printf("Whether Node{data = %d} has left Subtree (Yes[1]/No[0]) : ",ptr->data);  
 scanf("%d", &ans);  
 struct node\* lcptr = (struct node\*)malloc(sizeof(struct node));  
 if(ans==1){  
 ptr->lchild=lcptr;  
 build\_tree(lcptr);  
 }else{  
 lcptr=NULL;  
 ptr->lchild=NULL;  
 build\_tree(lcptr);  
 }  
 printf("Whether Node{data = %d} has right Subtree (Yes[1]/No[0]) : ",ptr->data);  
 scanf("%d", &ans);  
 struct node\* rcptr = (struct node\*)malloc(sizeof(struct node));  
 if(ans==1){  
 ptr->rchild=rcptr;  
 build\_tree(rcptr);  
 }else{  
 rcptr=NULL;  
 ptr->rchild=NULL;  
 build\_tree(rcptr);  
 }  
 }  
}  
  
struct node \* search\_link(struct node\* root ,int key){  
 struct node\* ptr = root;  
 struct node\* ptr1;  
 if(ptr->data != key){  
 if(ptr->lchild!=NULL){  
 ptr1 = search\_link(ptr->lchild, key);  
 if(ptr1 != NULL){  
 return ptr1;  
 }  
 }  
 if(ptr->rchild!=NULL){  
 ptr1 = search\_link(ptr->rchild, key);  
 if(ptr1 != NULL){  
 return ptr1;  
 }  
 }  
 return NULL;  
 }  
 else{  
 return ptr;  
 }  
}  
  
void insert\_tree(struct node\* root,int key){  
 struct node\* ptr;  
 int item,ans;  
 printf("Enter element to be Inserted :");  
 scanf("%d", &item);  
 ptr=search\_link(root,key);  
 if(ptr==NULL){  
 printf("\n Search Unsucessful \n");  
 }else{  
 if((ptr->lchild==NULL)||(ptr->rchild==NULL)){  
 printf("Insert as left or right child (left[1]/right[0]) : ");  
 scanf("%d", &ans);  
 if(ans==1){  
 if(ptr->lchild==NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=item;  
 new->lchild=NULL;  
 new->rchild=NULL;  
 ptr->lchild=new;  
 }else{  
 printf("\n Insertion not possible as left child \n");  
 }  
 }  
 if(ans==0){  
 if(ptr->rchild==NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=item;  
 new->lchild=NULL;  
 new->rchild=NULL;  
 ptr->rchild=new;  
 }else{  
 printf("\n Insertion not possible as right child \n");  
 }  
  
 }  
 }  
 else{  
 printf("\n Insertion not possible Key Node has left and right child \n");  
 }  
 }  
}  
  
  
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\* search\_parent(struct node\* ptr ,struct node\* parent,int item){  
 if(ptr->data != item){  
 if(ptr->lchild != NULL){  
 parent = search\_parent(ptr->lchild,ptr,item);  
 if(parent != NULL)  
 return parent;  
 }  
 if(ptr->rchild != NULL)  
 {  
 parent = search\_parent(ptr->rchild,ptr,item);  
 if(parent != NULL)  
 return parent;  
 }  
 return NULL;  
 }  
 else  
 return parent;  
}  
  
  
void delete\_tree(struct node\* root,int item){  
 struct node\* parent;  
 struct node\* ptr;  
 struct node\* ptr1;  
 struct node\* ptr2;  
 ptr=root;  
 if(ptr==NULL){  
 printf("\n Tree is Empty\n");  
 }else if(root->data == item ){  
 if(root->lchild==NULL && root->rchild == NULL){  
 root=NULL;  
 }  
 else{  
 printf("\n Node is not a leaf Node\n");  
 }  
 }else{  
 parent = search\_parent(root,root,item);  
 if(parent!=NULL){  
 ptr1 = parent->lchild;  
 ptr2 = parent->rchild;  
 if(ptr1->data==item){  
 if((ptr1->lchild==NULL)&&(ptr1->rchild==NULL)){  
 parent->lchild=NULL;  
 }else{  
 printf("\n Node is not a leaf Node\n");  
 }  
 }  
 if(ptr2->data==item){  
 if((ptr2->lchild==NULL)&&(ptr2->rchild==NULL)){  
 parent->rchild=NULL;  
 }else{  
 printf("\n Node is not a leaf Node\n");  
 }  
 }  
 }else{  
 printf("\n Node with data item doesn't exists\n");  
 }  
 }  
}  
  
void main(){  
 int n,item,key;  
 char ans='y';  
 struct node\* root = (struct node\*)malloc(sizeof(struct node));  
 root->lchild=NULL;  
 root->rchild=NULL;  
 printf("\n--- BUILD BINARY TREE --- \n\n");  
 build\_tree(root);  
 printf("\n--- OPERATION ON BINARY 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. EXIT \n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT ---\n");  
 if(root == NULL){  
 printf("\n Tree is empty \n\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &item);  
 root = (struct node\*)malloc(sizeof(struct node));  
 root->lchild=NULL;  
 root->rchild=NULL;  
 root->data=item;  
 }else{  
 printf("Enter the Key Node :");  
 scanf("%d", &key);  
 insert\_tree(root,key);  
 }  
 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 if(root->data==item){  
 printf("\n Node is not a leaf Node\n");  
 }else{  
 delete\_tree(root,item);  
 }  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 6:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**







**RESULT:**

The given operations are performed on a binary tree.