Name - Shantanu Deshpande
Class - SYCSE
Roll no. - 7
Batch - S1
Sub. - D.S.

Practical No. 5

Aim : Implementation of binary tree traversals

Theory :

Binary Tree :

In computer science, a binary tree is a tree data structure in which each node has at most two children. Typically the first node is known as the parent and the child nodes are called left and right.

Def - A binary tree is a finite set of elements that is either empty or its partitioned into three disjoint subset. First contain single element called root. The other subsets are themselves binary trees called as left and right sub trees or it can be empty.

Date : / / 20

Types of binary trees :

1. Strictly binary tree - If every non leaf node in a binary tree has nonempty left and right subtrees then this binary tree is called as strictly binary tree.

2. Complete binary tree - A complete binary tree of depth d is strictly binary tree of depth d all of whose leaves are at level d.

3. Skewed tree :- If every node is having only left and right sub tree. Then it is called as left or right skewed tree.

4. Almost complete binary tree :- A binary tree of depth d is an almost complete binary tree if

i) Any node nd at level less than d-1 has two sons.

ii) For any node nd in the tree with a right descendent at level d, nd must have a left son and every left descendent of nd is either a leaf at level d or has two sons.

Tree traversal:

In computer science, tree-traversal refers to the process of visiting (examining and/or updating) each node in a tree data structure, exactly once, in a systematic way. Such traversals are classified by the order in which the nodes are visited. Tree structures can be traversed in many different ways. Starting at the root of a binary tree, there are three main steps that can be performed and the order in which they are performed defines the traversal type.

To traverse a non-empty binary tree in preorder, perform the following operations recursively at each node, starting with the root node:

1. Visit the node
2. Traverse the left sub tree.
3. Traverse the right sub tree.

(This is also called Depth-first traversal)

To traverse a non-empty binary tree in inorder, perform the following operations recursively at each node:

1. Traverse the left subtree.
2. Visit the node.
3. Traverse the right subtree.

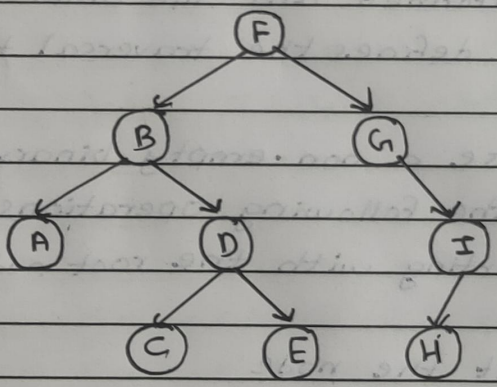
(This is also called symmetric traversal)

Date : / / 20

To traverse a non-empty binary tree in postorder, perform the following operations recursively at each node :

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the node.

Example



In this binary search tree,

- Preorder traversal sequence : F, B, A, D, C, E, G, I, H
  (root, left, right)
- Inorder traversal sequence : A, B, C, D, E, F, G, H, I
  (left, root, right)
- Postorder traversal sequence : A, C, E, D, B, H, I, G, F
  (left, right, root)

Binary search tree=>

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node *left,*right;
}*root=NULL,*p,*q;
void insert(int);
void inorder(struct node *);
struct node * makenode(int a);
int search(int);
void delet(int);
struct node * parent(struct node *pr,struct node *n1,int no);
void main()
{
int ch =1;
while( ch != 5)
{
int ele,result;
printf("1.Insert 2.Delete 3.Display 4.search 5.Exit \nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("Enter the element");
scanf("%d",&ele);
insert(ele);
break;
case 2:
printf("Enter the element to be deleted");
scanf("%d",&ele);
delet(ele);
break;
case 3:
printf("\nIneorder Traversal:");
inorder(root);
break;
case 4:
printf("\n Enter the element for search:");
scanf("%d",&ele);
result=search(ele);
if(result==1)
printf("Element is present");
else
printf("Element is not present");
break;
case 5:
break;
}
}
}
void insert(int x)
{
```

```
if(root==NULL)
root=makenode(x);
else
{
p=root;
while(p!=NULL)
{
q=p;
if(x<p->info)
p=q->left;
else
p=q->right;
}
if(x<=q->info)
q->left=makenode(x);
else
q->right=makenode(x);
}
}
struct node * makenode(int a)
{
struct node *s;
s=(struct node *)malloc(sizeof(struct node));
s->left=NULL;
s->right=NULL;
s->info=a;
return(s);
}
void inorder(struct node *t)
{
if(t==NULL)
return;
inorder(t->left);
printf(" %d",t->info);
inorder(t->right);
}
int search(int a)
{
if (root==NULL)
{
printf("Tree is empty");
return(0);
}
else
{
p=root;
while(p!=NULL)
{
q=p;
if(q->info==a)
{
return(1);
}
if(a<q->info)
p=q->left;
```

```
else
p=q->right;
}
if(p==NULL)
return(0);
}
}
void delet(int x)
{
struct node *rp,*f,*s;
p=root;
q=NULL;
while(p!=NULL && p->info != x)
{
q=p;
p=(x<p->info)?p->left:p->right;
}
if(p==NULL)
{
printf("Element is not present");
return;
}
if(p->left==NULL)
rp=p->right;
else
if(p->right==NULL)
rp=p->left;
else
{
f=p;
rp=p->right;
s=rp->left;
while(s!=NULL)
{
f=rp;
rp=s;
s=rp->left;
}
if(f!=p)
{
f->left=rp->right;
rp->right=p->right;
}
rp->left=p->left;
}
if(q==NULL)
root=rp;
else
if(p==q->left)
q->left=rp;
else q->right=rp;
free(p);
}
```

GDB online Debugger | Compiler

onlinegdb.com

Run | Debug | Stop | Share | Save | { } Beautify | Language C

main.c

input

```
1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:1
Enter the element2
1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:1
Enter the element4
1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:1
Enter the element6
1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:1
Enter the element8
1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:3

Ineorder Traversal: 2 4 6 81.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:2
Enter the element to be deleted2
1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:2
Enter the element to be deleted5
Element is not present1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:2
Enter the element to be deleted6
1.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:3

Ineorder Traversal: 4 81.Insert 2.Delete 3.Display 4.search 5.Exit
Enter your choice:5


...Program finished with exit code 0
Press ENTER to exit console.
```