NAME :AL UMA

REG: 230701368

EX NO 9 IMPLEMENTATION OF BINARY SEARCH TREE

```
#include <stdio.h>
#include<stdlib.h>
struct tree
{
int data;
struct tree *left;
struct tree *right;
}*root=NULL;
// Fucntion declarations
void insert();
void delete(struct tree *,int);
struct tree * inorder_succ(struct tree *);
void inorder(struct tree *);
void search();
int main()
{
int ans=1,key;
struct tree *ptr=NULL;
int choice;
do
{
printf("Enter your choice:-\n1.Insert\n2.Delete\n3.Display\n4.Search\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
insert();
break;
case 2:
printf("\nEnter the value to be deleted\n");
scanf("%d",&key);
ptr=root;
delete(ptr,key);
break;
case 3:
ptr=root;
inorder(ptr);
break;
case 4:
search();
break;
}
printf("\nWant to continue?\nPress 1.YES \t 0.NO\n");
scanf("%d",&ans);
} while(ans==1);
}
void insert()
{
int Flag=0,key;
struct tree *parent,*ptr=root;
printf("Enter the value to be inserted\n");
scanf("%d",&key);
while(ptr!=NULL && Flag==0) // loop to reach the node where the newnode has to be inserted.
{
if(key<ptr->data)
{
parent=ptr;
ptr=ptr->left;
}
else if(key>ptr->data)
{
parent=ptr;
ptr=ptr->right;
}
else if(key==ptr->data)
{
Flag=1;
}
}
//creating newnode using malloc and setting the data and links of new node.
struct tree *newnode=malloc(sizeof(struct tree));
newnode->left=newnode->right=NULL;
newnode->data=key;
if(parent==NULL)
{
root=newnode;
}
else
{
if(key<parent->data)
parent->left=newnode;
```

```
else
parent->right=newnode;
}
}
void inorder(struct tree *ptr) //displaying the data as per inorder traversal
{
if(ptr!=NULL)
{
inorder(ptr->left);
printf("%d->",ptr->data);
inorder(ptr->right);
}
}
void search()//function to search the given key
{
int Flag=0,key;
struct tree *parent,*ptr=root;
printf("Enter the key to be searched\n");
scanf("%d",&key);
while(ptr!=NULL && Flag==0)
{
if(key<ptr->data)
{
parent=ptr;
ptr=ptr->left;
}
else if(key>ptr->data)
{
parent=ptr;
ptr=ptr->right;
}
else if(key==ptr->data)
{
Flag=1;
printf("%d found",ptr->data );
}
}
if(Flag==0)
printf("Required Key not found");
}
void delete(struct tree *ptr,int key)// function to delete the given key.
{
struct tree *parent=NULL;
int Flag=0;
```

```
while(ptr!=NULL && Flag==0)// loop to reach the node to be deleted.
{
if(key<ptr->data)
{
parent=ptr;
ptr=ptr->left;
}
else if(key>ptr->data)
{
parent=ptr;
ptr=ptr->right;
}
else if(key==ptr->data)
{
Flag=1;
}
}
if(Flag==0)
printf("Required Key does not exist");
else
{
if(ptr->left==NULL && ptr->right==NULL ) //if the node to be deleted in the leaf node.
{
if(parent==NULL)//condition for if node to be deleted is the root node.
{
root=NULL; //root will become NULL
}
else if (key<parent->data)
parent->left =NULL;
else
parent->right=NULL;
free(ptr);
}
else if(ptr->left==NULL || ptr->right==NULL )//if the node to e deleted has one child.
{
if(parent==NULL)//if the node to be deleted is the root node.
{
if(ptr->right==NULL)
root=ptr->left; //root will change to ptr->left
else
root=ptr->right;//root will change to ptr->right
}
else if(key<parent->data)
{
```

```
if (ptr->left!=NULL)
parent->left=ptr->left;
else
parent->left=ptr->right;
}
else if(key>parent->data)
{
if (ptr->left!=NULL)
parent->right=ptr->left;
else
parent->right=ptr->right;
}
}
else if(ptr->left!=NULL && ptr->right!=NULL )//if the node to be deleted has two children.
{
struct tree*new_ptr;
new_ptr=inorder_succ(ptr->right);//inorder_succ() function to find the inorder successor of
the node to be deleted.
int save=new_ptr->data; //new_ptr is the inorder_successor.new_ptr->data is preserved in
variable save.
delete(ptr,new_ptr->data);//call the delete function to delete the inorder successor
ptr->data=save;// replace the data of ptr(node to bbe deleted) by data of inorder
successor(save).
}
}
//return root;//return root
}
struct tree * inorder_succ(struct tree *pt)//function to find inorder successor of the given node
{
while(pt->left!=NULL)
{
pt=pt->left;
}
return pt;
}
```