***ASSIGNMENT-9***

1. ***WAP to implement Binary Search Tree (BST).  
   a. Insert a node into the BST.  
   b. Delete a node from the BST.  
   c. Display the preorder traversal of the BST.  
   d. Display the inorder traversal of the BST.  
   e. Display the postorder traversal of the BST.***

#include<stdio.h>

#include<stdlib.h>

struct btnode{

int info;

int count;

struct btnode \*lchild,\*rchild;

};

typedef struct btnode BTNODE;

BTNODE \*insert(BTNODE \*root,int x);

BTNODE \*deleten(BTNODE \*root,int x);

BTNODE \*minval(BTNODE \*root);

void predisplay(BTNODE \*root);

void indisplay(BTNODE \*root);

void postdisplay(BTNODE \*root);

int main()

{

BTNODE \*root=NULL;

int ch;

printf("1.Insert in Tree\n");

printf("2.Delete an element\n");

printf("3.Display Preorder Traversal\n");

printf("4.Display Inorder Traversal\n");

printf("5.Display Postorder Traversal\n");

printf("6.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter number you want to insert : ");

scanf("%d",&ch);

root = insert(root,ch);

break;

case 2:

printf("enter number you want to delete : ");

scanf("%d",&ch);

root = deleten(root,ch);

break;

case 3:

printf("Preorder Traversal of Tree: ");

predisplay(root);

printf("\n");

break;

case 4:

printf("Inorder Traversal of Tree : ");

indisplay(root);

printf("\n");

break;

case 5:

printf("PostOrder Traversal of Tree : ");

postdisplay(root);

printf("\n");

break;

case 6:

exit;

}

}

return 0;

}

BTNODE \*insert(BTNODE \*root,int x)

{

if(root==NULL)

{

root = (BTNODE \*)malloc(sizeof(BTNODE));

root->info = x;

(root->count)++;

root->lchild = root->rchild = NULL;

return root;

}

else if(root->info > x)

{

root->lchild = insert(root->lchild,x);

}

else if(root->info < x)

{

root->rchild = insert(root->rchild,x);

}

else

(root->count)++;

return root;

}

BTNODE \*deleten(BTNODE \*root,int x)

{

if(root==NULL)

return NULL;

else if(root->info > x)

root->lchild = deleten(root->lchild,x);

else if(root->info < x)

root->rchild = deleten(root->rchild,x);

else{

if(root->count > 1)

(root->count)--;

else if(root->lchild==NULL)

{

BTNODE \*temp = root->rchild;

free(root);

return temp;

}

else if(root->rchild==NULL)

{

BTNODE \*temp = root->lchild;

free(root);

return temp;

}

else

{

BTNODE \*temp = minval(root->rchild);

root->info = temp->info;

root->rchild = deleten(root->rchild,temp->info);

}

}

return root;

}

BTNODE \*minval(BTNODE \*root)

{

if(root->lchild==NULL)

return root;

return minval(root->lchild);

}

void predisplay(BTNODE \*root)

{

if(root!=NULL)

{

int i=0;

while(i<(root->count)){

printf("%d ",root->info);

i++;

}

predisplay(root->lchild);

predisplay(root->rchild);

}

}

void indisplay(BTNODE \*root)

{

if(root!=NULL)

{

int i=0;

indisplay(root->lchild);

while(i<(root->count)){

printf("%d ",root->info);

i++;

}

indisplay(root->rchild);

}

}

void postdisplay(BTNODE \*root)

{

if(root!=NULL)

{

int i=0;

postdisplay(root->lchild);

postdisplay(root->rchild);

while(i<(root->count))

{

printf("%d ",root->info);

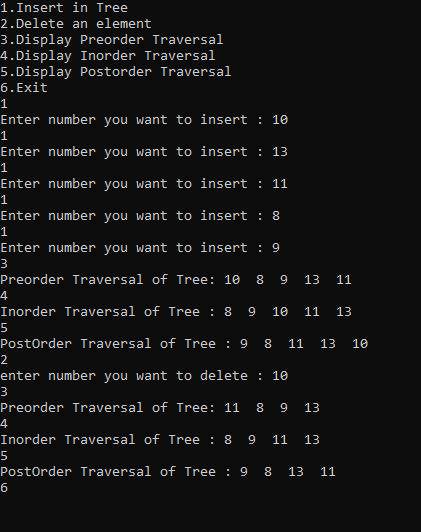
i++;

}

}

}

***OUTPUT :***



1. ***WAP to construct a binary tree given  
   a. Inorder and Postorder traversals.***

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct btnode{

struct btnode \*lchild,\*rchild;

int info;

};

typedef struct btnode BTNODE;

int convertInt(char str[],int arr[]);

BTNODE \*constructTree(int arr1[],int arr2[],int n,int last,int \*p);

void displayIn(BTNODE \*root);

void displayPost(BTNODE \*root);

int main()

{

BTNODE \*root = NULL;

printf("Enter Inorder traversal of Binary Tree : ");

char str1[1000];

gets(str1);

printf("Enter Postorder traversal of Binary Tree : ");

char str2[1000];

gets(str2);

int arr1[100],arr2[100];

int n = convertInt(str1,arr1);

convertInt(str2,arr2);

int i = n-1;

root = constructTree(arr1,arr2,0,n-1,&i);

printf("\nInorder Traversal of Tree : \n");

displayIn(root);

printf("\nPostorder Traversal of Tree : \n");

displayPost(root);

return 0;

}

int convertInt(char str[],int arr[])

{

int l = strlen(str);

int i,j=0;

int temp;

for(i=0;i<l;i++)

{

temp=0;

if(str[i]>='0' && str[i]<='9')

{

while(str[i]!=' ' && i<l){

temp = temp\*10 + (str[i]-'0');

i++;

}

arr[j++]=temp;

}

}

return j;

}

BTNODE \*constructTree(int In[],int Post[],int start,int last,int \*p)

{

if(start>last)

return NULL;

BTNODE \*root = (BTNODE \*)malloc(sizeof(BTNODE));

root->lchild=root->rchild=NULL;

root->info = Post[\*p];

(\*p)--;

if(start==last)

return root;

int i;

for(i=start;i<=last;i++)

if(In[i]==root->info)

break;

root->rchild = constructTree(In,Post,i+1,last,p);

root->lchild = constructTree(In,Post,start,i-1,p);

return root;

}

void displayIn(BTNODE \*root)

{

if(root!=NULL)

{

displayIn(root->lchild);

printf("%d ",root->info);

displayIn(root->rchild);

}

}

void displayPost(BTNODE \*root)

{

if(root!=NULL)

{

displayPost(root->lchild);

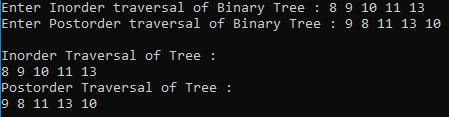
displayPost(root->rchild);

printf("%d ",root->info);

}

}

***OUTPUT :***



***b. Inorder and Preorder traversals.***

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct btnode{

struct btnode \*lchild,\*rchild;

int info;

};

typedef struct btnode BTNODE;

int convertInt(char str[],int arr[]);

BTNODE \*constructTree(int arr1[],int arr2[],int n,int last,int \*p);

void displayIn(BTNODE \*root);

void displayPre(BTNODE \*root);

int main()

{

BTNODE \*root = NULL;

printf("Enter Inorder traversal of Binary Tree : ");

char str1[1000];

gets(str1);

printf("Enter Preorder traversal of Binary Tree : ");

char str2[1000];

gets(str2);

int arr1[100],arr2[100];

int n = convertInt(str1,arr1);

convertInt(str2,arr2);

int i = 0;

root = constructTree(arr1,arr2,0,n-1,&i);

printf("\nInorder Traversal of Tree : \n");

displayIn(root);

printf("\nPreorder Traversal of Tree : \n");

displayPre(root);

return 0;

}

int convertInt(char str[],int arr[])

{

int l = strlen(str);

int i,j=0;

int temp;

for(i=0;i<l;i++)

{

temp=0;

if(str[i]>='0' && str[i]<='9')

{

while(str[i]!=' ' && i<l){

temp = temp\*10 + (str[i]-'0');

i++;

}

arr[j++]=temp;

}

}

return j;

}

BTNODE \*constructTree(int In[],int Pre[],int start,int last,int \*p)

{

if(start>last)

return NULL;

BTNODE \*root = (BTNODE \*)malloc(sizeof(BTNODE));

root->lchild=root->rchild=NULL;

root->info = Pre[\*p];

(\*p)++;

if(start==last)

return root;

int i;

for(i=start;i<=last;i++)

if(In[i]==root->info)

break;

root->lchild = constructTree(In,Pre,start,i-1,p);

root->rchild = constructTree(In,Pre,i+1,last,p);

return root;

}

void displayIn(BTNODE \*root)

{

if(root!=NULL)

{

displayIn(root->lchild);

printf("%d ",root->info);

displayIn(root->rchild);

}

}

void displayPre(BTNODE \*root)

{

if(root!=NULL)

{

printf("%d ",root->info);

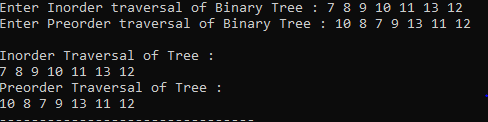
displayPre(root->lchild);

displayPre(root->rchild);

}

}

***OUTPUT :***



1. ***WAP to implement the following:  
    a. Count the number of nodes in a binary tree.  
    b. Count the number of leaf nodes in a binary tree.  
    c. Count the number of non-leaf nodes in a binary tree.  
    d. Return the height of the binary tree.  
    e. Check whether the tree is a strict binary tree or not.  
    f. Check whether the two trees are equal or not.***

#include<stdio.h>

#include<stdlib.h>

struct btnode{

int info;

int count;

struct btnode \*lchild,\*rchild;

};

typedef struct btnode BTNODE;

BTNODE \*insert(BTNODE \*root,int x);

int countnode(BTNODE \*root);

int countleaf(BTNODE \*root);

int countnonleaf(BTNODE \*root);

int checkstrict(BTNODE \*root,int ch);

int height(BTNODE \*root);

int identical(BTNODE \*root,BTNODE \*root2);

int main()

{

BTNODE \*root=NULL;

int ch;

printf("1.Insert in Tree\n");

printf("2.Count number of nodes\n");

printf("3.Number of leaf node\n");

printf("4.Number of non-leaf node\n");

printf("5.Height of Binary Tree\n");

printf("6.Check Strict Binary tree or not\n");

printf("7.Two Trees are equal or not\n");

printf("8.Exit\n");

while(1)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter number you want to insert : ");

scanf("%d",&ch);

root = insert(root,ch);

break;

case 2:

printf("Number of nodes = %d\n",countnode(root));

break;

case 3:

printf("Number of leaf node = %d\n",countleaf(root));

break;

case 4:

printf("Number of non-leaf node = %d\n",countnonleaf(root));

break;

case 5:

printf("Height of binary Tree : %d\n",height(root));

break;

case 6:

ch=1;

if(checkstrict(root,ch))

printf("A Strict Binary Tree\n");

else

printf("Not a Strict Binary Tree\n");

break;

case 7:

printf("Enter total elements of second Tree : ");

scanf("%d",&ch);

int i,j;

BTNODE \*root2=NULL;

for(i=1;i<=ch;i++){

scanf("%d",&j);

root2 = insert(root2,j);

}

if(identical(root,root2))

printf("Trees are equal\n");

else

printf("Not Equal\n");

break;

//default :

// exit(1);

}

}

return 0;

}

BTNODE \*insert(BTNODE \*root,int x)

{

if(root==NULL)

{

root = (BTNODE \*)malloc(sizeof(BTNODE));

root->info = x;

(root->count)++;

root->lchild = root->rchild = NULL;

return root;

}

else if(root->info > x)

{

root->lchild = insert(root->lchild,x);

}

else if(root->info < x)

{

root->rchild = insert(root->rchild,x);

}

else

(root->count)++;

return root;

}

int countnode(BTNODE \*root)

{

if(root==NULL)

return 0;

return (1+countnode(root->lchild)+countnode(root->rchild));

}

int countleaf(BTNODE \*root)

{

if(root==NULL)

return 0;

if(root->lchild==NULL && root->rchild==NULL)

return 1;

return (countleaf(root->lchild)+countleaf(root->rchild));

}

int countnonleaf(BTNODE \*root)

{

if(root==NULL)

return 0;

if(root->lchild==NULL && root->rchild==NULL)

return 0;

return (1+countnonleaf(root->lchild)+countnonleaf(root->rchild));

}

int height(BTNODE \*root) {

if(root==NULL)

return 0;

else if(root->lchild==NULL && root->rchild==NULL)

return 0;

int a = 1+height(root->lchild);

int b = 1+height(root->rchild);

if(a>b)

return a;

return b;

}

int checkstrict(BTNODE \*root,int ch)

{

int a = countnonleaf(root);

int b = countleaf(root);

if(b==a+1)

return 1;

return 0;

}

int identical(BTNODE \*root,BTNODE \*root2)

{

if(root==NULL && root2==NULL)

return 1;

if(root!=NULL && root2!=NULL){

int a = root->info == root2->info;

return (a && (identical(root->lchild,root2->lchild)) && (identical(root->rchild,root2->rchild)));

}

}

***OUTPUT :***

