***ASSIGNMENT-10***

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

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Node

{

struct Node \*left, \*right;

int info;

bool lthread;

bool rthread;

};

typedef struct Node Node;

struct Node \*insert(struct Node \*root, int ikey)

{

Node \*ptr = root;

Node \*par = NULL;

while (ptr != NULL)

{

if (ikey == (ptr->info))

{

printf("Duplicate Key !\n");

return root;

}

par = ptr;

if (ikey < ptr->info)

{

if (ptr -> lthread == false)

ptr = ptr -> left;

else

break;

}

else

{

if (ptr->rthread == false)

ptr = ptr -> right;

else

break;

}

}

Node \*tmp = malloc(sizeof(Node));

tmp -> info = ikey;

tmp -> lthread = true;

tmp -> rthread = true;

if (par == NULL)

{

root = tmp;

tmp -> left = NULL;

tmp -> right = NULL;

}

else if (ikey < (par -> info))

{

tmp -> left = par -> left;

tmp -> right = par;

par -> lthread = false;

par -> left = tmp;

}

else

{

tmp -> left = par;

tmp -> right = par -> right;

par -> rthread = false;

par -> right = tmp;

}

return root;

}

struct Node \*inSucc(struct Node \*ptr)

{

if (ptr->rthread == true)

return ptr->right;

ptr = ptr -> right;

while (ptr->right)

ptr = ptr->left;

return ptr;

}

struct Node \*inorderSuccessor(struct Node \*ptr)

{

if (ptr -> rthread == true)

return ptr->right;

ptr = ptr -> right;

while (ptr -> lthread == false)

ptr = ptr -> left;

return ptr;

}

void inorder(struct Node \*root)

{

if (root == NULL)

printf("Tree is empty");

struct Node \*ptr = root;

while (ptr -> lthread == false)

ptr = ptr -> left;

while (ptr != NULL)

{

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

ptr = inorderSuccessor(ptr);

}

}

struct Node \*inPred(struct Node \*ptr)

{

if (ptr->lthread == true)

return ptr->right;

ptr = ptr->left;

while (ptr->rthread);

ptr = ptr->right;

return ptr;

}

struct Node \*caseA(struct Node \*root, struct Node \*par,

struct Node \*ptr)

{

if (par == NULL)

root = NULL;

else if (ptr == par->left)

{

par->lthread = true;

par->left = ptr->left;

}

else

{

par->rthread = true;

par->right = ptr->right;

}

free(ptr);

return root;

}

struct Node \*caseB(struct Node \*root, struct Node \*par,

struct Node \*ptr)

{

struct Node \*child;

if (ptr->lthread == false)

child = ptr->left;

else

child = ptr->right;

if (par == NULL)

root = child;

else if (ptr == par->left)

par->left = child;

else

par->right = child;

Node \*s = inSucc(ptr);

Node \*p = inPred(ptr);

if (ptr->lthread == false)

p->right = s;

else

{

if (ptr->rthread == false)

s->left = p;

}

free(ptr);

return root;

}

struct Node \*caseC(struct Node \*root, struct Node \*par,

struct Node \*ptr)

{

struct Node \*parsucc = ptr;

struct Node \*succ = ptr -> right;

while (succ->left != NULL)

{

parsucc = succ;

succ = succ -> left;

}

ptr->info = succ->info;

if (succ->lthread == true && succ->rthread == true)

root = caseA(root, parsucc, succ);

else

root = caseB(root, parsucc, succ);

return root;

}

struct Node \*delThreadedBST(struct Node\* root, int dkey)

{

struct Node \*par = NULL, \*ptr = root;

int found = 0;

while (ptr != NULL)

{

if (dkey == ptr->info)

{

found = 1;

break;

}

par = ptr;

if (dkey < ptr->info)

{

if (ptr->lthread == false)

ptr = ptr -> left;

else

break;

}

else

{

if (ptr->rthread == false)

ptr = ptr->right;

else

break;

}

}

if (found == 0)

printf("dkey not present in tree\n");

else if (ptr->lthread == false && ptr->rthread == false)

root = caseC(root, par, ptr);

else if (ptr->lthread == false)

root = caseB(root, par, ptr);

else if (ptr->rthread == false)

root = caseB(root, par, ptr);

else

root = caseA(root, par, ptr);

return root;

}

void preorder(Node \* root){

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

if(!(root->lthread) && !(root->rthread)){

preorder(root->left);

}

if(!(root->lthread) && !(root->rthread)){

preorder(root->right);

}

}

void postorder(Node \* root){

if(!(root->lthread) && !(root->rthread)){

postorder(root->left);

}

if(!(root->lthread) && !(root->rthread)){

postorder(root->right);

}

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

}

int main()

{

struct Node \*root = NULL;

printf("Enter your choice what you want to do\n");

printf("1.Insert an element in TBT\n");

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

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

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

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

printf("6.Exit\n");

while(1)

{

int ch;

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter element : ");

scanf("%d",&ch);

root = insert(root,ch);

break;

case 2:

printf("Enter element you want to delete : ");

scanf("%d",&ch);

root = delThreadedBST(root,ch);

break;

case 3:

preorder(root);

break;

case 4:

inorder(root);

break;

case 5:

postorder(root);

break;

case 6:

exit(1);

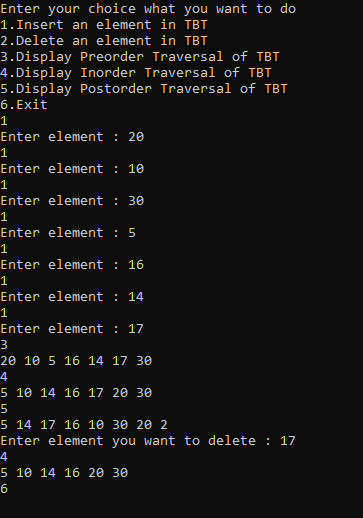
}

}

return 0;

}

***OUTPUT :***



1. ***WAP to construct an Expression Tree (Heterogeneous Tree) from postfix expression.***

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define SIZE 1000

struct btnode{

char info;

struct btnode \*lchild,\*rchild;

};

typedef struct btnode BTNODE;

BTNODE \*create(BTNODE \*root,char str[]);

void postdisplay(BTNODE \*root);

int main()

{

BTNODE \*root = NULL;

printf("Enter Postfix Expression : \n");

char str[SIZE];

scanf("%s",str);

root = create(root,str);

printf("Postorder Traversal of Expression Tree : \n");

postdisplay(root);

return 0;

}

BTNODE \*create(BTNODE \*root,char str[])

{

BTNODE \*\*arr = (BTNODE \*\*)malloc(sizeof(struct btnode\*)\*SIZE);

int top = -1;

int i;

int l = strlen(str);

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

{

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

tmp->info = str[i];

if(str[i]>='A' && str[i]<='Z')

{

tmp->lchild = tmp->rchild = NULL;

arr[++top] = tmp;

}

else

{

tmp->rchild = arr[top--];

tmp->lchild = arr[top--];

arr[++top] = tmp;

}

}

root = arr[top--];

return root;

}

void postdisplay(BTNODE \*root)

{

if(root!=NULL)

{

postdisplay(root->lchild);

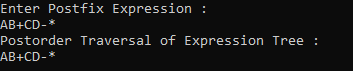
postdisplay(root->rchild);

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

}

}

***OUTPUT :***



1. ***WAP to implement Huffman Tree.***

#include<bits/stdc++.h>

using namespace std;

#define SIZE 10000

struct huffnode{

int freq;

char data;

struct huffnode \*lchild,\*rchild;

};

typedef struct huffnode HNODE;

struct dref{

bool operator()(const HNODE\* a,const HNODE\* b)const{

return a->freq > b->freq;

}

};

typedef priority\_queue<HNODE \*,vector<HNODE\*>,dref> pq;

HNODE \*create(char str[]);

void indisplay(HNODE \*root);

void postdisplay(HNODE \*root);

void decode(HNODE \*root,char str[]);

int main()

{

HNODE \*root = NULL;

char str[SIZE];

cout << "Enter Message : " << endl;

cin >> str;

root = create(str);

//indisplay(root);

cout << "Enter String you want to decode\n";

cin >> str;

decode(root,str);

//postdisplay(root);

return 0;

}

HNODE \*create(char str[])

{

pq p;

int arr[26]={0};

int i=0;

int l = strlen(str);

while(i!=l){

arr[str[i]-65]++;

i++;

}

for(i=0;i<26;i++)

{

if(arr[i]!=0)

{

HNODE \*tmp = (HNODE \*)malloc(sizeof(HNODE));

tmp->lchild = NULL;

tmp->rchild = NULL;

tmp->data = (char)(i+65);

tmp->freq = arr[i];

p.push(tmp);

}

}

HNODE \*a = p.top();

p.pop();

HNODE \*q = (HNODE \*)malloc(sizeof(HNODE));

q->data = a->data;

q->freq = a->freq;

q->lchild = q->rchild = NULL;

a=q;

while(p.size()!=0)

{

HNODE \*b = p.top();

p.pop();

HNODE \*tmp = (HNODE \*)malloc(sizeof(HNODE));

tmp->freq = a->freq + b->freq;

tmp->data = '\0';

if(a->freq < b->freq)

{

tmp->rchild = b;

tmp->lchild = a;

}

else

{

tmp->rchild = a;

tmp->lchild = b;

}

a = tmp;

}

return a;

}

void decode(HNODE \*root,char str[])

{

int i;

int l = strlen(str);

HNODE \*p = root;

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

{

if(str[i]=='1')

{

if(p->rchild==NULL)

{

printf("%c",p->data);

i--;

p = root;

}

else

p = p->rchild;

}

else

{

if(p->lchild==NULL)

{

printf("%c",p->data);

p = root;

i--;

}

else

p = p->lchild;

}

}

printf("%c",p->data);

}

void indisplay(HNODE \*root)

{

if(root!=NULL)

{

indisplay(root->lchild);

printf("%c %d\n",root->data,root->freq);

indisplay(root->rchild);

}

}

void postdisplay(HNODE \*root)

{

if(root!=NULL)

{

postdisplay(root->lchild);

postdisplay(root->rchild);

printf("%c %d\n",root->data,root->freq);

}

}

***OUTPUT :***

