//二叉树的基本运算算法

#include <stdio.h>

#include <malloc.h>

#define MaxSize 100

typedef char ElemType;

typedef struct node

{

ElemType data; //数据元素

struct node \*lchild; //指向左孩子节点

struct node \*rchild; //指向右孩子节点

float num;

} BTNode;

void CreateBTree(BTNode \* &b,char \*str) //创建二叉树

{

BTNode \*St[MaxSize],\*p=NULL;

int top=-1,k,j=0;

char ch;

b=NULL; //建立的二叉树初始时为空

ch=str[j];

while (ch!='\0') //str未扫描完时循环

{

switch(ch)

{

case '(':top++;St[top]=p;k=1; break; //为左孩子节点

case ')':top--;break;

case ',':k=2; break; //为孩子节点右节点

default:p=(BTNode \*)malloc(sizeof(BTNode));

p->data=ch;p->lchild=p->rchild=NULL;

if (b==NULL) //\*p为二叉树的根节点

b=p;

else //已建立二叉树根节点

{

switch(k)

{

case 1:St[top]->lchild=p;break;

case 2:St[top]->rchild=p;break;

}

}

}

j++;

ch=str[j];

}

}

void DestroyBTree(BTNode \*&b)

{ if (b!=NULL)

{ DestroyBTree(b->lchild);

DestroyBTree(b->rchild);

free(b);

}

}

BTNode \*FindNode(BTNode \*b,ElemType x)

{

BTNode \*p;

if (b==NULL)

return NULL;

else if (b->data==x)

return b;

else

{

p=FindNode(b->lchild,x);

if (p!=NULL)

return p;

else

return FindNode(b->rchild,x);

}

}

BTNode \*LchildNode(BTNode \*p)

{

return p->lchild;

}

BTNode \*RchildNode(BTNode \*p)

{

return p->rchild;

}

int BTHeight(BTNode \*b)

{

int lchildh,rchildh;

if (b==NULL) return(0); //空树的高度为0

else

{

lchildh=BTHeight(b->lchild); //求左子树的高度为lchildh

rchildh=BTHeight(b->rchild); //求右子树的高度为rchildh

return (lchildh>rchildh)? (lchildh+1):(rchildh+1);

}

}

void DispBTree(BTNode \*b)

{

if (b!=NULL)

{ printf("%c",b->data);

if (b->lchild!=NULL || b->rchild!=NULL)

{ printf("("); //有孩子节点时才输出(

DispBTree(b->lchild); //递归处理左子树

if (b->rchild!=NULL) printf(","); //有右孩子节点时才输出,

DispBTree(b->rchild); //递归处理右子树

printf(")"); //有孩子节点时才输出)

}

}

}

//二叉树的基本运算算法

#include <stdio.h>

#include <malloc.h>

#define MaxSize 100

typedef char ElemType;

typedef struct node

{

ElemType data; //数据元素

struct node \*lchild; //指向左孩子节点

struct node \*rchild; //指向右孩子节点

} BTNode;

void CreateBTree(BTNode \* &b,char \*str) //创建二叉树

{

BTNode \*St[MaxSize],\*p=NULL;

int top=-1,k,j=0;

char ch;

b=NULL; //建立的二叉树初始时为空

ch=str[j];

while (ch!='\0') //str未扫描完时循环

{

switch(ch)

{

case '(':top++;St[top]=p;k=1; break; //为左孩子节点

case ')':top--;break;

case ',':k=2; break; //为孩子节点右节点

default:p=(BTNode \*)malloc(sizeof(BTNode));

p->data=ch;p->lchild=p->rchild=NULL;

if (b==NULL) //\*p为二叉树的根节点

b=p;

else //已建立二叉树根节点

{

switch(k)

{

case 1:St[top]->lchild=p;break;

case 2:St[top]->rchild=p;break;

}

}

}

j++;

ch=str[j];

}

}

void DestroyBTree(BTNode \*&b)

{ if (b!=NULL)

{ DestroyBTree(b->lchild);

DestroyBTree(b->rchild);

free(b);

}

}

BTNode \*FindNode(BTNode \*b,ElemType x)

{

BTNode \*p;

if (b==NULL)

return NULL;

else if (b->data==x)

return b;

else

{

p=FindNode(b->lchild,x);

if (p!=NULL)

return p;

else

return FindNode(b->rchild,x);

}

}

BTNode \*LchildNode(BTNode \*p)

{

return p->lchild;

}

BTNode \*RchildNode(BTNode \*p)

{

return p->rchild;

}

int BTHeight(BTNode \*b)

{

int lchildh,rchildh;

if (b==NULL) return(0); //空树的高度为0

else

{

lchildh=BTHeight(b->lchild); //求左子树的高度为lchildh

rchildh=BTHeight(b->rchild); //求右子树的高度为rchildh

return (lchildh>rchildh)? (lchildh+1):(rchildh+1);

}

}

void DispBTree(BTNode \*b)

{

if (b!=NULL)

{ printf("%c",b->data);

if (b->lchild!=NULL || b->rchild!=NULL)

{ printf("("); //有孩子节点时才输出(

DispBTree(b->lchild); //递归处理左子树

if (b->rchild!=NULL) printf(","); //有右孩子节点时才输出,

DispBTree(b->rchild); //递归处理右子树

printf(")"); //有孩子节点时才输出)

}

}

}

//顺序队列（非环形队列）基本运算算法

#define MaxSize 100

typedef struct

{

BTNode \*pt; //存放当前结点指针

int d; //存放相对根节点的左右偏移量

}NodeType; //非环形队列元素类型

typedef struct

{

NodeType data[MaxSize];

int front,rear; //队头和队尾指针

} SqQueue;

void InitQueue(SqQueue \*&q)

{ q=(SqQueue \*)malloc (sizeof(SqQueue));

q->front=q->rear=-1;

}

void DestroyQueue(SqQueue \*&q) //销毁队列

{

free(q);

}

bool QueueEmpty(SqQueue \*q) //判断队列是否为空

{

return(q->front==q->rear);

}

bool enQueue(SqQueue \*&q,NodeType e) //进队

{ if (q->rear==MaxSize-1) //队满上溢出

return false; //返回假

q->rear++; //队尾增1

q->data[q->rear]=e; //rear位置插入元素e

return true; //返回真

}

bool deQueue(SqQueue \*&q,NodeType &e) //出队

{ if (q->front==q->rear) //队空下溢出

return false;

q->front++;

e=q->data[q->front];

return true;

}

#include<stdio.h>

#include "btree\_sqqence.h"

int Nodes(BTNode \*b)//计算一棵给定二叉树的所有节点个数

{ int num1,num2;

if (b==NULL)

return 0;

else

return Nodes(b->lchild)+Nodes(b->rchild)+1;

}

void DispLeaf(BTNode \*b)//输出一棵给定二叉树的所有叶子节点

{

if (b!=NULL)

{

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

printf("%c ",b->data); //访问叶子节点

DispLeaf(b->lchild); //输出左子树中的叶子节点

DispLeaf(b->rchild); //输出右子树中的叶子节点

}

}

int Level(BTNode \*b,char x,int h) //返回结点值对应二叉树层次，h置初值1

{

int l;

if (b==NULL)

return(0);

else if (b->data==x)

return(h);

else

{ l=Level(b->lchild,x,h+1); //在左子树中查找

if (l!=0)

return(l);

else //在左子树中未找到,再在右子树中查找

return(Level(b->rchild,x,h+1));

}

}

int Width(BTNode \*b) //层次遍历求二叉树的宽度

{

BTNode \*p;

SqQueue \*qu;

NodeType qelem;

int n,min=0,max=0;

InitQueue(qu); //初始化非环形队列

qelem.pt=b;qelem.d=0; //创建根节点对应的队列元素

enQueue(qu,qelem); //根节点进队

while(!QueueEmpty(qu)) //队不空循环

{

deQueue(qu,qelem);

p=qelem.pt;

n=qelem.d;

if(n<min)

min=n; //存放最小偏移量

if(n>max)

max=n; //存放最大偏移量

if(p->lchild!=NULL) //创建p的左孩子对应的队列元素并进队（如果有）

{

qelem.pt=p->lchild;

qelem.d=n-1;

enQueue(qu,qelem);

}

if(p->rchild!=NULL) //创建p的右孩子对应的队列元素并进队（如果有）

{

qelem.pt=p->rchild;

qelem.d=n+1;

enQueue(qu,qelem);

}

}

return(max-min+1);

}

int main()

{

char \*str="A(B(D,E(H(J,K(L,M(,N))),)),C(F,G(,I)))";

BTNode \*b;

char x;

int h;

CreateBTree(b,str);

printf("二叉树的结点个数为：%d\n",Nodes(b));

printf("从左到右输出所有叶子节点:");

DispLeaf(b);

printf("\n");

printf("请输入要返回层次的节点值:");

scanf\_s("%c",&x);

h=Level(b,x,1);

if (h==0)

printf("b中不存在%c节点\n",x);

else

printf("在b中%c节点的层次为%d\n",x,h);

printf("二叉树的宽度为:%d\n",Width(b));

DestroyBTree(b);

return 0;

}

