**广义表的创建与遍历\_长度\_深度算法实现**

**注意递归的运用**

#include <stdio.h>

#include<string.h>

#include <stdlib.h>

#include <malloc.h>

#include <assert.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#define AtomType int

enum ElemTag{HEAD,ATOM,CHILDLIST}; //新的语句 其中enum是定义枚举型（一个集合，

//集合中的元素是一些命名的整型常量，默认第一个为0，但是可以自定义，

//后续枚举成员在前一个成员上加1.）的关键字

//ElemTag为标识符

//用的书上拓展性的结点构建方法

typedef struct GLNode {

ElemTag tag;

union

{

AtomType atom;

struct GLNode\* hp; //表头指针

};

struct GLNode\* tp; //相当于next指针

}GLNode;

typedef GLNode\* GenList; //定义广义表：定义广义表结点指针类型。即用指向（广义表头结点）的一个指针，来代表整个广义表

void InitGenList(GenList &gl) {

gl = NULL;

}

bool sever(char\* sub, char\* hsub);

//const char\* ga = "(1,2,3)";

//const char\* gb = "(1,(2,3))";

//括号不算在广义表中

//广义表除了表头就是表尾，每次针对表头进行创建

void CreatGenList(GenList& gl, const char\* str) {

int n = strlen(str); //先求出字符串整个的长度

char\* sub = (char\*)malloc(sizeof(char) \* (n - 2)); //创建字符指针并开辟空间，用指向字符串首地址的指针代表整个字符串

char\* hsub = (char\*)malloc(sizeof(char) \* (n - 2));

assert(sub != NULL && hsub != NULL);

//"1,2,3"

strncpy(sub, str + 1, n - 2); //strncpy函数，拷贝字符串里的n个数据

sub[n - 2] = '\0';

if (gl == NULL) {

gl = (GLNode\*)malloc(sizeof(GLNode)); //创建第一个结点,即头结点

assert(gl != NULL);

gl->tag = HEAD;

gl->hp = gl->tp = NULL;

}

GLNode\* p = gl;

while (strlen(sub) != 0) {

p = p->tp = (GLNode\*)malloc(sizeof(GLNode)); //循环一次开辟一个新的空间

assert(p != NULL);

p->hp = p->tp = NULL;

//"1,2,3" ==>"1" hsub="1" sub="2,3"

//"(1,2),3,4"==>"(1,2)" hsub="(1,2)" sub="3,4"

if (sever(sub, hsub)) {

if (hsub[0] == '(') {

p->tag = CHILDLIST; //此时表的表头hp指向新建立的一个头结点

CreatGenList(p->hp, hsub); //递归

}

else {

p->tag = ATOM;

p->atom = atoi(hsub); //字符串转成整型 //由动画演示可以得出 当atom更改时，hp是自动更改的

}

}

}

}

//广义表表头分割函数

//以逗号为分割标志，但千万是不能是括号里的逗号

//"1,2,3" ==>"1" hsub="1" sub="2,3"

//"(1,2),3,4"==>"(1,2)" hsub="(1,2)" sub="3,4"

bool sever(char\* sub, char\* hsub) {

if (\*sub == '\0' || strcmp(sub, "()") == 0) {

hsub[0] = '\0';

return true;

}

int n = strlen(sub);

int i = 0;

char ch = sub[i]; //另外一种写法 char ch=\*sub; while(ch!=',' && \*sub!='\0');

int k = 0; //来描述括号，k!=0说明没有匹配到一对括号

while (i < n && (ch != ',' || k != 0)) { //即要括号里的同时不满足（即不是括号里的逗号）才能跳出循环

if (ch == '(') {

k++;

}

else if (ch == ')') {

k--;

}

i++;

ch = sub[i];

}

//下面进行分割

if (i < n) {

sub[i] = '\0';

strcpy(hsub, sub);

strcpy(sub, sub + i + 1);

}

else if (k != 0) { //k!=0则括号里的永远为真，即是因为i不再小于n才退出的循环,此时内部括号不匹配

return false;

}

else { //"(1,2)"==> hsub="(1,2)",sub="" 该种情况为整体就是一个表头的情况

strcpy(hsub, sub);

sub[0] = '\0';

}

return true;

}

//遍历

//是原子就直接访问，是子表则下滑到子表的链表进行操作

void ShowGenList(GenList& gl) {

GLNode\* p = gl->tp; //指向结构体的指针用->来访问其所指向的结构体的成员 a->b==(\*a).b //该语句是p指向广义表的第一个真实结点

printf("(");

while (p != NULL) {

if (p->tag == ATOM) {

printf("%d", p->atom);

if (p->tp != NULL) {

printf(",");

}

}

else if (p->tag == CHILDLIST) {

ShowGenList(p->hp); //注意这里传入的参数是子表表头（也是结点类型的指针，来代表整个子表）

}

p = p->tp;

}

printf("),");

}

//判空

bool GenListEmpty(GenList& gl) {

return gl->tp = NULL;

}

//长度

int GenListLength(GenList& gl) {

int length = 0;

GLNode\* p = gl->tp;

while (p != NULL) {

length++;

p = p->tp;

}

return length;

}

//深度

int GenListDepth(GenList& gl) {

if (gl->tp == NULL) {

return 1; //规定空表的深度为1

}

GLNode\* p = gl->tp;

int maxdepth = 0;

int dep;

while (p != NULL) {

if (p->tag == CHILDLIST) {

dep=GenListDepth(p->hp);

if (dep > maxdepth) {

maxdepth = dep;

}

}

p = p->tp;

}

return maxdepth + 1;

}

void main() {

GenList gl;

InitGenList(gl);

const char\* ga = "(1,2,3)"; //用指针指向字符串的首地址来代表整个字符串

const char\* gb = "(1,(2,3))";

const char\* gc = "(1,(2),3)";

const char\* gd = "((1,2),3)";

const char\* ge = "((1,2,3))";

const char\* gf = "(1,(2,(3,4)),5)";

const char\* gh = "()";

CreatGenList(gl, ga);

ShowGenList(gl); //不用引用也行

printf("\n");

}