**使用C++完成广义表的创建、输出、复制、求长度、求深度、求表头、求表尾以及遍历**

**一、广义表的概念和数据结构**

广义表是线性表的推广，但线性表的元素仅限于原子项，原子作为结构上不可分割的成分，它可以是一个数或一个结构，若放松对表元素的这种限制，允许它们具有自身独立类型结构，就产生了广义表的概念。

广义表是n(n>=0)个数据元素a1,a2,......ai,......,an的有序序列，一般记作：

ls = (a1,a2 ,........,ai, ...an)

其中：ls是广义表的名称，n是它的长度。

A = ()

B = (e)

C = (a,(b,(c,d)))

D = ((),B,C)

E = (a,E)

F = (())

**二、存储结构**

typedef enum{ATOM,LIST}elemtag;//ATOM==0:原子，LIST==1：子表

typedef char ElemType;

typedef struct GLNode

{

int tag;//公共部分，用于区分原子结点和表结点

int mark;

union {

ElemType data; //data是原子结点的值域

GLNode \*hp;

}val;

GLNode \*tp;

}GLNode ,\*GList;

//val是表结点的指针域，val.hp和val.tp分别指向表头和表尾

**三、广义表的创建**

void CreateGList(GList &L)//广义表的创建

{

char ch;

cin>>ch;

if(ch == ')')

{ //若输入为右括号，则置L的后继指针域为空

L = 0;

return ;

}

else if(ch >='a'&&ch <='z')

{

L = new GLNode;

L->tag = 0;

L->val.data =ch;

L->tp = 0;

}

else if(ch == '(')

{ //若输入为左括号则建立由L所指向的子表结点并递归构造字表

L = new GLNode;

L->tag = 1;

CreateGList(L->val.hp);

if(L->val.hp == 0)

{

L->tp = 0;

return ;

}

GLNode \*p;

p = L;

cin>>ch;

while(ch == ',')

{ //若输入为逗号则递归构造后继表

p->tp = new GLNode;

p = p->tp;

p->tag = 1;

CreateGList(p->val.hp);

cin>>ch;

}

p->tp = 0;

}

}

**四、广义表的输出**

void DisplayGList(GList &L)//广义表的输出

{

if(L == 0)

return ;

else if(L->tag == 0)

{ //对于单元素结点，输出该结点的值

cout<<L->val.data;

}

else

{

cout<<"("; //对于表结点，则先输出左括号，作为开始符号

DisplayGList(L->val.hp);

GLNode \*p;

p = L->tp;

while(p)

{ //若为非空字表，则递归输出此表

cout<<",";

DisplayGList(p->val.hp);

p = p->tp;

}

cout<<")";

}

}

**五、广义表的复制**

GList GListCopy(GList L)//复制广义表

{

GList M;

if(L == 0)

return 0;

else

{

M = new GLNode;

M->tag = L->tag;

if(L->tag == 0)

{ //是原子的话，复制原子

M->val.data = L->val.data;

M->tp = 0;

}

else

{ //分别复制表头和表尾

M->val.hp = GListCopy(L->val.hp);

M->tp = GListCopy(L->tp);

}

return M;

}

}

**六、求广义表的长度**

int LenthGList(GList L) //求L长度

{

if(L!=NULL){

return 1+LenthGList(L->tp);

}

else

{

return 0;

}

}

**七、求广义表的深度**

int DepthGList(GList L)//求广义表L的深度算法

{

if(L == 0)

{

return 0;

}

if(L->tag == 0)

{

return 0;

}

else if(L->val.hp == 0)

{

return 1;

}

else

{

int m1 ,m2;

m1 = DepthGList(L->val.hp)+1;

m2 = DepthGList(L->tp);

if(m1>=m2)

return m1;

else

return m2;

}

}

**八、递归算法遍历广义表**

void GListTraverse(GList L) //遍历广义表L

{

if(!L)

{

cout<<"()";

}

else

{

if(L->tag==ATOM)

{

cout<<L->val.data;

}

else

{

cout<<"(";

GLNode \*p=L;

while(p)

{

GListTraverse(p->val.hp);

p=p->tp;

if(p)

{

cout<<",";

}

}

cout<<")";

}

}

}

**九、非递归算法遍历广义表**

void MarkList(GList M) //非递归算法遍历

{

GLNode \*t=NULL, \*p=M, \*q;

//t指向p的母表结点， p指向当前

Status finished;

finished=false;

while(!finished)

{

while(p->mark==0)

{

//处理表头一路遍历，当遇到原子，退出循环

p->mark=1;//此结点已经遍历

q=p->val.hp;//q指向p的表头或者表尾

if(q&&q->mark==0)

{

if(q->tag==0)

{

//q为原子，则输出

q->mark=1;

cout<<q->val.data<<",";

}

else

{

//否则继续向下遍历，

p->val.hp=t;

p->tag=ATOM;

t=p;

p=q;

//表头标志完，记录路径的方法p的hp域指向t，p的tag记为0

}

}

}

q=p->tp;//q指向此时p的表尾

if(q&&q->mark==0)//如果表尾非空，且未被遍历过，则继续遍历表尾，回到上面while循环

{

p->tp=t;

t=p;

p=q;

}

else

{ //q为空或已经遍历，则回溯 分表头回溯，和表尾回溯

while(t&&t->tag==1)

{//表尾回溯,如果此结点的表头已经遍历，则会不断表尾回溯

q=t;

t=q->tp;

q->tp=p;

p=q;

}

if(!t)

{

finished=true;

}

else

{

//表头回溯，从表头遍历到底，然后向上回溯，并修改此时p的tag以防在表尾回溯时不彻底，

q=t;

t=q->val.hp;

q->val.hp=p;

p=q;

p->tag=LIST;

}

}

}

}

**十、求广义表表头**

GList Head(GList L)//求表头的算法

{

GList H;

if(L == 0)

{ //如果为空，则无表头

cout<<"空表无表头"<<endl;

return 0;

}

else if(L->tag == 0)

{ //如果为原子，无表头

cout<<"原子无表头"<<endl;

return 0;

}

else

{ //调用函数赋值表头

H = GListCopy(L->val.hp);

if(H == 0)

cout<<"空表无表头"<<endl;

return H;

}

}

**十一、求广义表表尾**

GList Tail(GList L)//求表尾的算法

{

GList T;

if(L == 0)

{

cout<<"空表无表尾"<<endl;

return 0;

}

else if(L->tag == 0)

{

cout<<"原子无表尾"<<endl;

return 0;

}

else

{ //调用函数赋值表尾

T = GListCopy(L->tp);

if(T == 0)

{ //如果T为0，则表尾为0

T =new GLNode;

T->tag = 1;

T->tp = 0;

T->val.hp = 0;

}

return T;

}

}