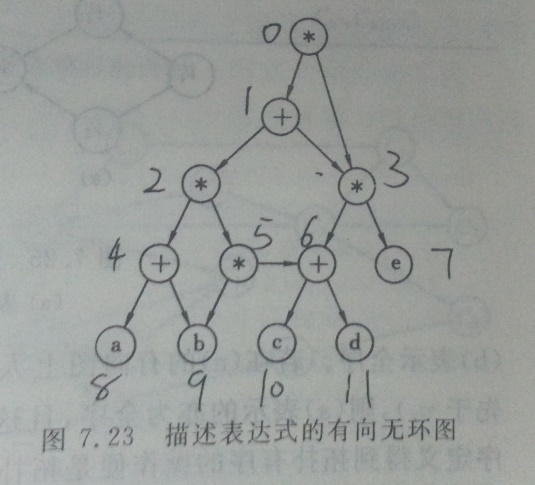
使用课本图7.23作为实例：



#include <iostream>

#include <stack>

#include <cstdlib>

#define MAX\_SIZE 20

using namespace std;

typedef struct ArcNode{ //弧结构体

int adjvex; //弧所指顶点位置

ArcNode\* nextarc;

bool visited;

}ArcNode;

typedef struct VNode{ //顶点结构体

char data;

ArcNode\* firstarc; //第一条依附该顶点的弧指针

}VNode;

typedef struct ALGraph{ //图结构体

int vexnum; //顶点个数

int arcnum; //弧数

VNode\* Ghead; //首个结点指针

}ALGraph;

void setALGraph(ALGraph \*G) //构建课本图7.23,靠近头结点的为左孩子

{

VNode\* p=(VNode\*)malloc(MAX\_SIZE\*sizeof(VNode));

for(int i=0;i<7;++i)

{

p[i].firstarc=(ArcNode\*)malloc(sizeof(ArcNode));

p[i].firstarc->nextarc=(ArcNode\*)malloc(sizeof(ArcNode));

p[i].firstarc->nextarc->nextarc=NULL;

}

p[0].data='\*';

p[0].firstarc->adjvex=1;

p[0].firstarc->nextarc->adjvex=3;

p[1].data='+';

p[1].firstarc->adjvex=2;

p[1].firstarc->nextarc->adjvex=3;

p[2].data='\*';

p[2].firstarc->adjvex=4;

p[2].firstarc->nextarc->adjvex=5;

p[3].data='\*';

p[3].firstarc->adjvex=6;

p[3].firstarc->nextarc->adjvex=7;

p[4].data='+';

p[4].firstarc->adjvex=8;

p[4].firstarc->nextarc->adjvex=9;

p[5].data='\*';

p[5].firstarc->adjvex=9;

p[5].firstarc->nextarc->adjvex=6;

p[6].data='+';

p[6].firstarc->adjvex=10;

p[6].firstarc->nextarc->adjvex=11;

p[7].data='e';

p[7].firstarc=NULL;

p[8].data='a';

p[8].firstarc=NULL;

p[9].data='b';

p[9].firstarc=NULL;

p[10].data='c';

p[10].firstarc=NULL;

p[11].data='d';

p[11].firstarc=NULL;

G->vexnum=12;

G->arcnum=14;

G->Ghead=p;

}

void display(ALGraph \*G) //显示图

{

cout << "顶点个数：" << G->vexnum << endl;

cout << "弧个数：" << G->arcnum << endl;

for(int i=0;i<G->vexnum;++i)

{

cout << i << "\t" << G->Ghead[i].data << ":";

ArcNode\* t=G->Ghead[i].firstarc;

while(t)

{

cout << t->adjvex << "\t";

t=t->nextarc;

}

cout << endl;

}

}

void freeGraph(ALGraph \*G) //释放内存

{

for(int i=0;i<G->vexnum;++i)

{

ArcNode\* arct=G->Ghead[i].firstarc;

while(arct)

{

ArcNode\* temp=arct;

arct=arct->nextarc;

free(temp); //释放弧内存

}

}

free(G->Ghead); //释放顶点内存

free(G); //释放图内存

}

int VNodeindex(ALGraph\* G,VNode\* v) //返回顶点所在下标

{

for(int i=0;i<G->vexnum;++i)

{

bool b=true;

VNode vt=G->Ghead[i];

if(v->data!=vt.data)

b=false;

ArcNode \*v1=v->firstarc;

ArcNode \*v2=vt.firstarc;

while(v1||v2)

{

if(!(v1&&v2)) //弧个数不同

{

b=false; break;

}

else if(v1->adjvex!=v2->adjvex) //弧指向不同

{

b=false; break;

}

else{

v1=v1->nextarc;

v2=v2->nextarc;

}

}

if(b) //匹配成功

return i;

}

return -1; //匹配失败

}

//7.16

void InsertVex(ALGraph \*G,VNode \*v)

{

G->Ghead[G->vexnum]=\*v;

G->vexnum++;

int arccount=0; //记录该顶点弧个数

ArcNode\* p=v->firstarc;

while(p)

{

++arccount;

p=p->nextarc;

}

G->arcnum+=arccount;

}

void InsertArc(ALGraph \*G,VNode \*v,VNode \*w) //v指向w

{

int vindex=VNodeindex(G,v);

int windex=VNodeindex(G,w);

if(vindex==-1 || windex==-1) return; //顶点不存在

ArcNode \*t=(ArcNode\*)malloc(sizeof(ArcNode));

t->adjvex=windex;

ArcNode \*p=G->Ghead[vindex].firstarc;

bool b=false; //判断弧是否已插入

if(!p) //出度为0

{

t->nextarc=NULL;

G->Ghead[vindex].firstarc=t;

b=true;

}

else if(windex>p->adjvex)

{

t->nextarc=p;

G->Ghead[vindex].firstarc=t;

b=true;

}

else{

while(p->nextarc)

{

if(windex>p->nextarc->adjvex)

{

t->nextarc=p->nextarc;

p->nextarc=t;

b=true;

}

else

p=p->nextarc;

}

}

if(!b) //插入位置在末尾

{

p->nextarc=t;

t->nextarc=NULL;

}

G->arcnum++;

}

void DeleteVex(ALGraph \*G,VNode \*v)

{

int vindex=VNodeindex(G,v);

if(vindex==-1) return; //顶点不存在

int arccount=0; //记录要删除的弧个数

ArcNode\* p=G->Ghead[vindex].firstarc;

while(p)

{

++arccount;

ArcNode \*temp=p;

p=p->nextarc;

free(temp);

}

G->vexnum--;

for(int i=vindex;i<G->vexnum;++i)

G->Ghead[i]=G->Ghead[i+1];

for(int i=0;i<G->vexnum;++i) //删除其他顶点与其的弧

{

ArcNode\* p=G->Ghead[i].firstarc;

if(!p) continue; //没有弧

if(p->adjvex==vindex) //删除的是第一条弧

{

G->Ghead[i].firstarc=p->nextarc;

free(p);

++arccount;

continue;

}

while(p->nextarc)

{

ArcNode\* temp=p->nextarc;

if(temp->adjvex==vindex)

{

if(temp->nextarc)

p->nextarc=p->nextarc->nextarc;

else

p->nextarc=NULL;

free(temp);

++arccount;

break;

}

p=p->nextarc;

}

}

G->arcnum-=arccount;

}

void DeleteArc(ALGraph \*G,VNode \*v,VNode \*w) //删除v指向w的弧

{

int vindex=VNodeindex(G,v);

int windex=VNodeindex(G,w);

if(vindex==-1 || windex==-1) return; //顶点不存在

ArcNode\* p=G->Ghead[vindex].firstarc;

bool b=false; //判断是否删除成功

if(!p)

{

cout << "要删除的弧不存在！" << endl;

return;

}

if(p->adjvex==windex)

{

if(p->nextarc)

G->Ghead[vindex].firstarc=p->nextarc;

else

G->Ghead[vindex].firstarc=NULL;

b=true;

}

else

{

while(p->nextarc)

{

ArcNode\* temp=p->nextarc;

if(temp->adjvex==vindex)

{

if(temp->nextarc)

p->nextarc=p->nextarc->nextarc;

else

p->nextarc=NULL;

free(temp);

b=true;

break;

}

p=p->nextarc;

}

}

if(b) G->arcnum--;

else

cout << "要删除的弧不存在！" << endl;

}

//7.22

bool b; //记录连通状态

void DFS(ALGraph \*G,int vindex,int windex) //递归算法

{

if(b) return;

ArcNode\* p=G->Ghead[vindex].firstarc;

while(p)

{

if(p->adjvex==windex)

{

b=true; return;

}

DFS(G,p->adjvex,windex);

p=p->nextarc;

}

}

void q22(ALGraph \*G,VNode\* v,VNode\* w) //从V到w是否连通

{

int vindex=VNodeindex(G,v);

int windex=VNodeindex(G,w);

if(vindex==-1 || windex==-1)

{

cout << "顶点不存在！" << endl;

return;

}

b=false; //默认未连通

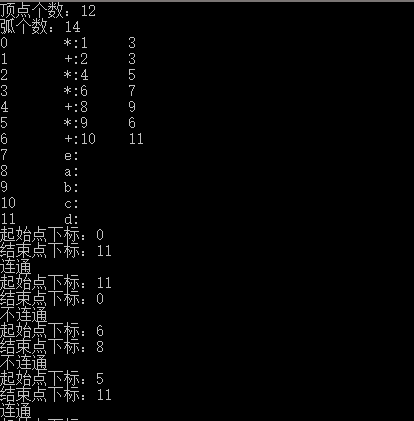
DFS(G,vindex,windex);

if(!b)

cout << "不";

cout << "连通" <<endl;

}



//7.24

void q24(ALGraph \*G)

{

int vnum=G->vexnum;

bool \*visited=(bool\*)malloc(vnum\*sizeof(bool)); //记录顶点的访问状态

for(int i=0;i<vnum;++i) //初始化

visited[i]=false;

stack<VNode> s;

s.push(G->Ghead[0]); //从0下标出开始遍历

visited[0]=true;

cout << "访问次序：" << "0 ";

while(!s.empty())

{

bool b=true; //true表示该顶点所指的点都被访问

VNode vt=s.top();

ArcNode\* p=vt.firstarc;

while(p)

{

if(!visited[p->adjvex]) //未被访问

{

s.push(G->Ghead[p->adjvex]);

visited[p->adjvex]=true;

cout << p->adjvex << " ";

b=false;

break;

}

p=p->nextarc;

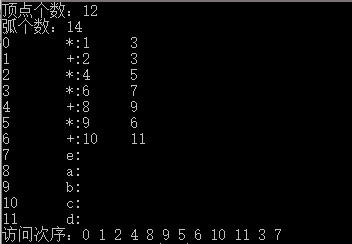
}

if(b) s.pop();

}

free(visited);

}



//7.38

void RPN(ALGraph\* G,int vindex) //vindex为顶点所在下标

{

VNode vt=G->Ghead[vindex];

ArcNode \*p;

p=vt.firstarc;

while(p)

{

if(!p->visited)

{

p->visited=true; //弧变为已访问

ArcNode\* t=G->Ghead[p->adjvex].firstarc;

while(t) //弧所指顶点包含的指出的弧都置成未访问

{

t->visited=false;

t=t->nextarc;

}

RPN(G,p->adjvex);

}

p=p->nextarc;

}

cout << vt.data << " ";

}

void q38(ALGraph \*G)

{

for(int i=0;i<G->vexnum;++i) //所有弧都置为未访问

{

ArcNode\* p=G->Ghead[i].firstarc;

while(p)

{

p->visited=false;

p=p->nextarc;

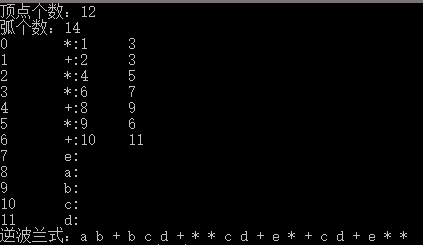
}

}

cout << "逆波兰式：";

RPN(G,0); //递归求解

}



int main()

{

ALGraph \*G=(ALGraph\*)malloc(sizeof(ALGraph));

G->vexnum=0; G->arcnum=0;

G->Ghead=NULL;

setALGraph(G); //建立图

//添加顶点

// VNode\* v=(VNode\*)malloc(sizeof(VNode));

// v->data='x';

// v->firstarc=NULL;

// InsertVex(G,v);

//删除顶点

// DeleteVex(G,v);

//添加弧

// InsertArc(G,v,v);

//删除弧

// VNode\* w=&G->Ghead[11];

// DeleteArc(G,v,w);

display(G); //输出各顶点

//7.22

// while(true)

// {

// int vstart,vend;

// cout << "起始点下标：";

// cin >> vstart;

// cout << "结束点下标：";

// cin >> vend;

// VNode \*v1=&G->Ghead[vstart];

// VNode \*v2=&G->Ghead[vend];

// q22(G,v1,v2);

// }

//7.24

// for(int i=7;i<G->vexnum;++i)

// InsertArc(G,&G->Ghead[i],&G->Ghead[0]);

// q24(G);

//7.38

// q38(G);

freeGraph(G); //释放内存

return 0;

}