#include<stdio.h>

#include<stdlib.h>

#include<assert.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define NPOS -1

typedef int ElemType;

typedef struct

{

int m;

int n;

ElemType elm;

}TTuple;

typedef struct

{

TTuple \*tup;

int row;

int col;

int unul;

}TMatrix;

TMatrix \*CreateEmptyTMatrix(int sizeM,int sizeN);

TMatrix \*CreateTMatrixFrom2DArray(void \*pArr2D,int sizeM,int sizeN);

void DestroyTMatrix(TMatrix \*pMat);

int ElemLocate(const TMatrix \*const pMat,int m,int n);

void DisplayTMatrix(const TMatrix \*const pMat);

int GetTMatrixSize(const TMatrix \*const pMat);

int AppendElem(TMatrix \*const pMat,ElemType elm,int m,int n);

int DeleteElem(TMatrix \*const pMat,int m,int n);

int TMatrixCopy(TMatrix \*const pMatDest,TMatrix \*const pMatSrc);

int Value(const TMatrix \*const pMat,int m,int n,ElemType \*pElm);

void ForEach(const TMatrix \*const pMat,void (\*func)(ElemType \*pElm));

TMatrix \*CreatEmptyTMatrix(int sizeM,int sizeN)

{

assert(sizeM > 0 && sizeN > 0);

TMatrix \*pMat = (TMatrix \*)malloc(sizeof(TMatrix));

pMat->tup = NULL;

pMat->row = sizeM;

pMat->col = sizeN;

pMat->unul = 0;

return pMat;

}

TMatrix \*CreateTMatrixFrom2DArray(void \*pArr2D,int sizeM,int sizeN)

{

assert(sizeM>0 && sizeN >0);

TMatrix \*pMat = (TMatrix \*)malloc(sizeof(TMatrix));

pMat->row = sizeM;

pMat->col = sizeN;

int m=0,n=0;

for(m=0;m < sizeM;++m)

for(n=0;n<sizeN;++n)

if(((ElemType\*)pArr2D)[sizeM\*m+n]!=0)

++pMat->unul;

pMat->tup = (TTuple\*)calloc(pMat->unul,sizeof(TTuple));

int nPos = 0;

for(m=0;m<sizeM;++m)

for(n=0;n<sizeN;++n)

if(((ElemType\*)pArr2D)[sizeM\*m+n]!=0)

{

pMat->tup[nPos].m = m;

pMat->tup[nPos].n = n;

pMat->tup[nPos].elm = ((ElemType \*)pArr2D)[sizeM\*m + n];

++nPos;

}

return pMat;

}

void DestroyTMatrix(TMatrix \*pMat)

{

free (pMat->tup);

free (pMat);

pMat = NULL;

}

int ElemLocate(const TMatrix \*const pMat,int m,int n)

{

int i=0;

for(i=0;i<pMat->unul;++i)

{if(pMat->tup[i].m == m && pMat->tup[i].n == n)

return i;

}

return NPOS;

}

void DisplayTMatrix(const TMatrix \*const pMat)

{

int m=0,n=0,pos =0;

for(m=0;m<pMat->row;++m)

{

for(n=0;n<pMat->col;++n)

{

pos = ElemLocate(pMat,m,n);

if(pos!= NPOS)

printf("%d",pMat->tup[pos].elm);

else

printf("%d ",0);

}

putchar('\n');

}

}

int GetTMatrixSize(const TMatrix \*const pMat)

{

return pMat->unul \*sizeof(TTuple);

}

int AppendElem(TMatrix \*const pMat,ElemType elm,int m,int n)

{assert(m>=0 && m < pMat->row && n>=0 && n<pMat->col);

assert(elm !=0);

int i=0,pos=0;

pos = ElemLocate(pMat,m,n);

if(pos!=NPOS)

{

pMat->tup[pos].elm = elm;

return pMat->unul;

}

pMat->tup = (TTuple\*)realloc(pMat->tup,sizeof(TTuple)\*(pMat->unul+1));

pMat->tup[pMat->unul].m = m;

pMat->tup[pMat->unul].n = n;

pMat->tup[pMat->unul].elm = elm;

return ++pMat->unul;

}

int DeleteElem(TMatrix \*const pMat,int m,int n)

{

assert(m >= 0 && m < pMat->row && n >= 0 && n<pMat->col);

int pos = ElemLocate(pMat,m,n);

if(pos == NPOS)

return NPOS;

for(pos;pos <pMat->unul - 1;++pos)

{

pMat->tup[pos].m = pMat->tup[pos+1].m;

pMat->tup[pos].n = pMat->tup[pos+1].n;

pMat->tup[pos].elm = pMat->tup[pos+1].elm;

}

pMat->tup = (TTuple \*)realloc(pMat->tup,sizeof(TTuple)\*(pMat->unul - 1));

return --pMat->unul;

}

int TMatrixCopy(TMatrix \*const pMatDest,TMatrix \*const pMatSrc)

{

if(pMatDest->tup)

free(pMatDest->tup);

if(pMatSrc->tup)

{

pMatDest->tup =(TTuple\*)calloc(pMatSrc->unul,sizeof(TTuple)\*

pMatSrc->unul);

assert(pMatDest->tup);

memcpy(pMatDest->tup,pMatSrc->tup,sizeof(TTuple)\*pMatSrc->unul);

}

else pMatDest->tup = NULL;

pMatDest->row = pMatSrc->row;

pMatDest->col = pMatSrc->col;

pMatDest->unul = pMatSrc->unul;

return pMatDest->unul;

}

int Value(const TMatrix \*const pMat,int m,int n,ElemType \*pElm)

{

assert(m>=0 && m< pMat->row && n>=0 && n<pMat->col);

int pos = ElemLocate(pMat,m,n);

if(pos != NPOS)

{

\*pElm = pMat->tup[pos].elm;

return pos;

}

else

{

\*pElm = 0;

return NPOS;

}

}

void ForEach(const TMatrix \*const pMat,void(\*func)(ElemType \*pElm))

{

int m = 0,n=0,pos = 0,t=0;

for(m=0;m<pMat->row;++m)

for(n=0;n<pMat->col;++n)

{

pos = ElemLocate(pMat,m,n);

if(pos!=NPOS)

func(&t);

}

}

void display(ElemType \*pElm)

{

if(\*pElm ==0)

putchar('x');

else

printf("%d",\*pElm);

}

int main()

{

ElemType arrMat[15][15] = {

{0,9,0,0,0,0,7,0,0,0,0,0,0,0,0},

{0,6,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,2,0,0,0,0,2,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,2,0,0},

{0,0,0,0,0,7,0,0,0,0,0,0,0,0,0},

{0,8,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,8,0,0,0,0,0,5,0,0,0,8,0,0},

{0,0,0,0,5,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,7,0,0,0,0,0,0,9,0,0,0,0,0,0},

{0,0,0,0,0,0,4,0,0,0,0,0,1,0,0},

{0,0,3,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,8,0,0,0,0,7,0,0,0,0,0},

{0,0,0,8,0,0,0,0,0,0,9,0,0,0,0}

};

TMatrix \*pMat = CreateTMatrixFrom2DArray(arrMat,15,15);

printf("稀疏矩阵占用空间的大小：%d(byte)\n",GetTMatrixSize(pMat));

TMatrix \*pMat2 = CreatEmptyTMatrix(5,5);

TMatrixCopy(pMat2,pMat);

printf("输出稀疏矩阵pMat2:\n");

DisplayTMatrix(pMat2);

printf("将0,0处元素置为1.\n");

AppendElem(pMat2,1,0,0);

printf("删除0,1处的元素.\n");

DeleteElem(pMat2,0,1);

printf("输出稀疏矩阵pMat2:\n");

DisplayTMatrix(pMat2);

int a=-1;

Value(pMat2,10,8,&a);

printf("位置10,8处的元素为：%d\n",a);

printf("将稀疏矩阵中值为0的元素用x代替并全部输出：\n");

ForEach(pMat2,display);

DestroyTMatrix(pMat);

DestroyTMatrix(pMat2);

system("pause");

return 0;

}