#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define stack\_init\_size 200

#define stack\_increment 10

#define ERROE 0

#define TRUE 1

#define FALSE 0

#define OVERFLOW 0

#define OK 1

typedef int Status;

typedef struct {

int x;

int y;

}PosType;

typedef struct {

int ord; // 通道块在路径上的 “序号 ”

PosType seat; //通道块在迷宫中的 “坐标位置 ”

int di; //从此通道块走向下一通道块的 “方向 ”

}SElemType;

typedef struct {

SElemType\* base;

SElemType\* top;

int stacksize;

}SqStack;

int mg[20][20];

/\* 随机生成迷宫的函数

/\* 为了能够让尽量能通过，将能通过的块和不能通过的块数量比大致为 2:1\*/

void Random() {

int i, j, k;

srand(time(NULL));

mg[1][0] = mg[1][1] = mg[18][19] = 0; //将入口、出口设置为 “ 0即可通过 ”

for (j = 0; j < 20; j++)

mg[0][j] = mg[19][j] = 1; /\* 设置迷宫外围 “不可走 ”，保证只有一个出口和入口 \*/

for (i = 2; i < 19; i++)

mg[i][0] = mg[i - 1][19] = 1; /\* 设置迷宫外围 “不可走 ”， 保证只有一个出口和入

口\*/

for (i = 1; i < 19; i++)

for (j = 1; j < 19; j++) {

k = rand() % 3; //随机生成 0、 1、 2 三个数

if (k)

mg[i][j] = 0;

else {

if ((i == 1 && j == 1) || (i == 18 && j == 18)) /\* 因为距入口或出口一步的路是必

经之路，故设该通道块为 “ 0” 加大迷宫能通行的概率 \*/

mg[i][j] = 0;

else

mg[i][j] = 1;

}

}

}

//构造一个空栈

Status InitStack(SqStack& s) {

s.base = (SElemType\*)malloc(stack\_init\_size \* sizeof(SElemType));

if (!s.base) return OVERFLOW;

s.top = s.base;

s.stacksize = stack\_init\_size;

return OK;

}

//当前块可否通过

Status Pass(PosType e) {

if (mg[e.x][e.y] == 0) //0 时可以通过

return OK; // 如果当前位置是可以通过，返回 1

return OVERFLOW; // 其它情况返回 0

}

//留下通过的足迹

Status FootPrint(PosType e) {

mg[e.x][e.y] = 7;

return OK;

}

//出栈

Status Pop(SqStack& s, SElemType& e) {

if (s.top == s.base)

return ERROE;

e = \*--s.top;

return OK;

}

//压入栈

Status Push(SqStack& s, SElemType e) {

if (s.top - s.base >= s.stacksize) {

s.base = (SElemType\*)realloc(s.base,(s.stacksize + stack\_increment)\* sizeof(SElemType));

if (!s.base)exit(OVERFLOW);

s.top = s.base + s.stacksize;

s.stacksize += stack\_increment;

}

\*s.top++ = e;

return OK;

}

//是否空栈

Status StackEmpty(SqStack s) {

if (s.top == s.base)

return OK;

return OVERFLOW;

}

//留下不能通过的足迹

Status MarkPrint(PosType e) {

mg[e.x][e.y] = 3;

return OK;

}//下一步

PosType NextPos(PosType& e, int dir) {

PosType E;

switch (dir) {

case 1:E.x = e.x; //向下

E.y = e.y + 1;

break;

case 2:E.x = e.x + 1; //向右

E.y = e.y;

break;

case 3:E.x = e.x; //向上

E.y = e.y - 1;

break;

case 4:E.x = e.x - 1; //向左

E.y = e.y;

break;

}

return E;

}

// 若迷宫 maze 中从入口 start 到出口 end 的通道，则求得一条存放在栈中

// （从栈底到栈顶） ，并返回 TRUE ；否则返回 FALSE

//迷宫函数

Status MazePath(int mg, PosType start, PosType end, SqStack & s) {

PosType curpos;

InitStack(s);

SElemType e;

int curstep;

curpos = start; // 设定 "当前位置 " 为"入口位置 "

curstep = 1; // 探索第一步

do {

if (Pass(curpos)) { // 当前位置可通过，即是未曾走到过的通道块

FootPrint(curpos); // 留下足迹

e.di = 1;

e.ord = curstep;

e.seat = curpos;

Push(s, e);// 加入路径

if (curpos.x == end.x && curpos.y == end.y) {

printf("\n\n0∩ \_∩0 能到达终点 !\n 路径如下： \n");

return TRUE;

}

curpos = NextPos(curpos, 1); // 下一位置是当前位置的东邻curstep++; // 探索下一步

}

else { // 当前位置不能通过

if (!StackEmpty(s)) {

Pop(s, e);

while (e.di == 4 && !StackEmpty(s)) {

MarkPrint(e.seat);

Pop(s, e);

}

if (e.di < 4) {

e.di++;

Push(s, e);// 留下不能通过的标记，并退回一步

curpos = NextPos(e.seat, e.di); /\* 当前位置设为新方向的相邻块 \*/

}//if

}//if

}//else

} while (!StackEmpty(s));

printf("\n\n 此迷宫不能走通！ \n");

return FALSE;

}

//打印迷宫

void PrintMaze() {

int i, j;

printf(" 运行路径 :\n\n");

for (i = 0; i < 20; i++) {

for (j = 0; j < 20; j++) {

if (mg[i][j] == 0)printf(" ");

else if (mg[i][j] == 1) printf("■"); //迷宫的“墙”

else if (mg[i][j] == 3) printf(" "); //不通的路

else if (mg[i][j] == 7) printf("○");//通过的路径

}

printf("\n");

}

printf("\n");

}

void main() {

SqStack S;

PosType start, end;

start.x = 1; start.y = 0; //起点坐标

end.x = 18; end.y = 19; //终点坐标

Random(); printf("\n\nTest 1:");

MazePath(mg[20][20], start, end, S);

PrintMaze();

system("pause");

Random();

printf("\nTest 2:");

MazePath(mg[20][20], start, end, S);

PrintMaze();

system("pause");

Random();

printf("\nTest 3:");

MazePath(mg[20][20], start, end, S);

PrintMaze();

}