**《程序设计专题》实验报告1**

**大程序开发基础：函数与程序结构、递归**

# 1. 实验目的

通过实验：

1）掌握大程序中的函数与程序结构设计思想方法；

2）掌握递归的应用；

3）走迷宫综合程序设计方法。

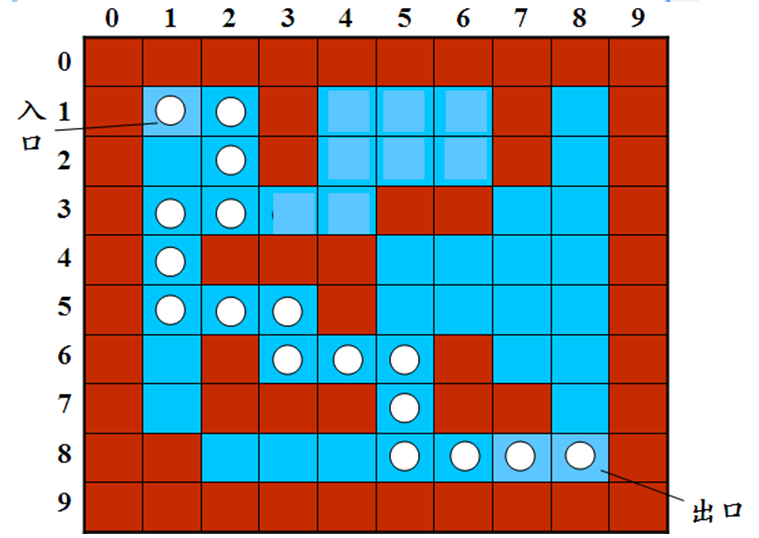
# 2. 提交说明

打包以下内容，压缩为.zip格式，上传至学在浙大。上传的压缩文件命名为学号\_姓名\_HW1.zip。

* 本实验报告(转为pdf)；
* 工程文件，删除其中的可执行文件和中间文件；
* 其它附件（如果有，例如使用说明等）

# 3. 实验内容

**用递归求解迷宫问题**

****

**求迷宫路径算法的基本思想：**

* 若当前位置“可通”，则纳入路径，继续前进；
* 若当前位置“不可通”，则后退，换方向（按东南西北的顺序）继续探索；
* 若四周“均无通路”，则将当前位置从路径中删除出去

**参考求解思路：**

设定当前位置的初值为入口位置；

do{

　 if (当前位置可通)

　　 {

　　　　　将当前位置插入栈顶； 　　　　　　// 纳入路径

　　　　　若该位置是出口位置，则算法结束；

　　　　　　// 此时栈中存放的是一条从入口位置到出口位置的路径

　　　　　否则切换当前位置的东邻方块为新的当前位置；

　　 }

　　 else

　　 {

　　　　 若栈不空且栈顶位置尚有其他方向未被探索，则设定新的当前位置为: 沿顺时针方向旋转找到的栈顶位置的下一相邻块；

　　　　 若栈不空但栈顶位置的四周均不可通，

　　　　 则{ 删去栈顶位置； 　　　　　// 从路径中删去该通道块

　　　　　　 若栈不空，则重新测试新的栈顶位置，

　　　　　　 直至找到一个可通的相邻块或出栈至栈空；

　　　　　 }

　　 }

} while (栈不空)；

## 3.1 实验要求：

1）程序应当包含基本的输入输出功能，例如迷宫的创建（最好采用程序自动生成）、求解路径的显示等；

2）采用多文件的设计方法实现，给出详细的文件模块设计与实现过程；

文件模块设计内容包括介绍每个模块完成的功能、维护的数据结构，实现过程描述各个模块对数据的操作（算法），算法可以用伪代码形式（如上利用栈的代码段）来体现，或用语言来描述。

3）源代码中应当有注释，以方便它人阅读和理解。注意代码规范，例如缩进、变量命名、函数命名等。参考《C编码规范》、《本学期作业代码自检规范》文档。

## 3.2 文件模块设计

### 3.2.1 main.c

在这个模块中包含了该程序的所有步骤，包括变量定义、函数声明。

### 3.2.2 create.c

create.c文件中定义了create和create1函数，该函数的作用是生成迷宫。

Create中我使用的创建迷宫的方法是先将整个迷宫做成均无法通行（全为-1），然后随机出互不相同的起点（op）、终点（ed），从起点开始向任意方向随机探索，直到到达终点，其间的所有经过的点全部打通（数值为0）。

如果打通的点不到地图的一半，就再随机打通一些点。

Create1中我用了DFS的方法生成迷宫，该方法用递归的形式寻找出一条迷宫路线，分别寻找每个到达的点的四周是否还有未走过的路线。缺点是主路线较明显且迷宫行数只能为奇数，但是生成的迷宫要远远优于create函数中生成的。

### 3.2.3 search.c

search.c文件中定义了search函数，该函数的作用是寻找从起点到终点的路线，我写了两种算法，其中search函数是老师提供的思路，search1函数是BFS算法，该算法解决迷宫问题相对简单。

Search函数的具体方法是先判断当前位置是否为0，如果是就将当前位置纳入记录路径的数列，再判断是否到达终点，若是则结束函数。如果不是终点，向右探索一个单位，再进行一遍search。

如果当前位置不是0，那么先退回到上一步，再探索其他三个方向，如果三个方向都探索完毕后，就将此处路径记为-1，并回到上一步。

Search1函数的思想是从最初的单元格开始直接逐步搜索，在某一步到达终点后停止函数并记录路径。

### 3.2.4 print.c

此为输出文件，包含输出函数print。

print函数会将在search函数中的路径数组依次输出。

print1函数为search1函数的配套输出函数，采用了递归输出的模式。

## 3.3 实现过程

创建迷宫的主要步骤如下：

while(!(p.x==ed.x&&p.y==ed.y)){

i=rand()%4;

tmp=fx[i].x+p.x;

tmp1=fx[i].y+p.y;

while(tmp<1||tmp>N||tmp1<1||tmp1>N){

i=rand()%4;

tmp=fx[i].x+p.x;

tmp1=fx[i].y+p.y;

}p.x=tmp;

p.y=tmp1;

map0[p.x][p.y]=0;

}//保证从起点能顺利到达终点

Search的具体代码如下：

if(map0[p.x][p.y]==0){

i=0;

map0[p.x][p.y]=-1;

cnt++;

roadx[cnt]=p.x;

roady[cnt]=p.y;

if(p.x==ed.x&&p.y==ed.y){

return;

}

else{

p.x=p.x+fx[i].x;

p.y=p.y+fx[i].y;

}search();

}

else{

p.x=p.x-fx[i].x;

p.y=p.y-fx[i].y;

if(i<3){

i++;

p.x=p.x+fx[i].x;

p.y=p.y+fx[i].y;

search();

}

else if(i==3){

map0[p.x][p.y]=-1;

roadx[cnt]=-1;

roady[cnt]=-1;

cnt--;

p.x=roadx[cnt];

p.y=roady[cnt];

map0[p.x][p.y]=0;

cnt--;

search();

}

}

## 3.4 运行结果展示

