迷宫问题
一．需求设计：以一个m*n 的长方阵表示迷宫，0和1分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口的通道，或得出没有通路的结论。
二．概要设计：
存储结构：
采用了数组以及结构体来存储数据，在探索迷宫的过程中用到的栈，属于顺序存储结构。

```
//八个方向的数组表示形式
int move[8][2]={{0,1},{1,1},{1,0},{1,-1},{0,-1},{-1,-1},{-1, 0},{-1, 1}};
//用结构体表示位置
struct position
{
        int x,y;
};
position stack[m*m+1];
```

基本算法：
走迷宫的过程可以模拟为一个搜索的过程：每到一处，总让它按东、东南、南、西南、西、西北、北、东北8个方向顺序试探下一个位置；如果某方向可以通过，并且不曾到达，则前进一步，在新位置上继续进行搜索；如果8个方向都走不通或曾经到达过，则退回一步，在原来的位置上继续试探下一位置。
每前进或后退一步，都要进行判断：若前进到了出口处，则说明找到了一条通路；若退回到了入口处，则说明不存在通路。
用一个字符类型的二维数组表示迷宫，数组中每个元素取值"0"（表示通路）或"1"（表示墙壁）。迷宫的入口点在位置（1，1）处，出口点在位置（m,m）处。设计一个模拟走迷宫的算法，为其寻找一条从入口点到出口点的通路。
二维数组的第0行、第m+1行、第0列、第m+1列元素全置成"1"， 表示迷宫的边界；第1行第1列元素和第m行第m列元素置成"0"， 表示迷宫的入口和出口；其余元素值用随机函数产生。
假设当前所在位置是（x,y）。沿某个方向前进一步，它可能到达的位置最多有8个。
如果用二维数组move记录8个方向上行下标增量和列下标增量，则沿第i个方向前进一步，可能到达的新位置坐标可利用move数组确定：

```
x=x+move[i][0]
y=y+move[i][1]
```

从迷宫的入口位置开始，沿图示方向顺序依次进行搜索。
在搜索过程中，每前进一步，在所到位置处做标记"?"
（表示这个位置在通路上），并将该位置的坐标压入栈中。
每次后退的时候，先将当前所在位置处的通路标记"?"改
成死路标记"×"（表示这个位置曾到达过但走不通，以后
不要重复进入），然后将该位置的坐标从栈顶弹出。
搜索到出口位置时，数组中那些值为"?"的元素形成一条
通路。
三．详细设计：
源程序：

```
/*
        迷宫问题
        走迷宫的过程可以模拟为一个搜索的过程：每到一
```

处，总让它按东、东南、南、西南、西、西北、北、东北个方向顺序试探下一个位置；如果某方向可以通过，并且不曾到达，则前进一步，在新位置上继续进行搜索；如果个方向都走不通或曾经到达过，则退回一步，在原来的位置上继续试探下一位置。

每前进或后退一步，都要进行判断：若前进到了出口处，则说明找到了一条通路；若退回到了入口处，则说明不存在通路。

用一个字符类型的二维数组表示迷宫，数组中每个元素取值"0"（表示通路）或"1"（表示墙壁）。迷宫的入口点在位置（1，1）处，出口点在位置（m,m）处。这个算法，为其寻找一条从入口点到出口点的通路。

*/

```
#include<iostream>
#include<stdlib.h>
#include<time.h>

int main()
{
	//设定m=10
	const int m=10;

	//数组的形式表示八个方向
	int move[8][2]={{0,1},{1,1},{1,0},
{1,-1},{0,-1},{-1,-1},{-1, 0},{-1, 1}};

	//用结构体表示位置
	struct position
	{
		int x,y;
	};

	//用于记录和输出迷宫探路中相关符号，包括1 .
	char maze[m+2][m+2];
	//用栈来存储探路过程中的数据
	position stack[m*m+1];
	int top;//栈顶
	int i,x,y,ok;
	position p;
	//二维数组的第0行、第m+1行、第0列、第m+1列元素全
	//置成"1"，表示迷宫的边界；第1行第1列元素和第m行第m列
	//元素置成"0"，表示迷宫的入口和出口；其余元素值用随机
	//函数产生。
	srand(time(0));
	for(x=1;x<=m;x++)
		for(y=1;y<=m;y++)
```

```
                    maze[x][y]=48+rand()%2;
maze[1][1]='0';maze[m][m]='0';
for(x=0;x<=m+1;x++)
{
          maze[x][0]='1';maze[x][m+1]='1';
}
for(y=0;y<=m+1;y++)
{
          maze[0][y]='1';maze[m+1][y]='1';
}
p.x=1;p.y=1;
top=1;stack[top]=p;
maze[1][1]='.';
//开始探路
//走迷宫的过程可以模拟为一个搜索的过程：每到一
//处，总让它按东、东南、南、西南、西、西北、北、东北
//个方向顺序试探下一个位置；如果某方向可以通过，并且不
//曾到达，则前进一步，在新位置上继续进行搜索；如果个
//方向都走不通或曾经到达过，则退回一步，在原来的位置上
//继续试探下一位置。
//    每前进或后退一步，都要进行判断：若前进到了出
//口处，则说明找到了一条通路；若退回到了入口处，则说明
//不存在通路。
do{
          p=stack[top];
          ok=0;i=0;
          while((ok==0)&&(i<8))
          {
                    x=p.x+move[i][0];
                    y=p.y+move[i][1];
                    if(maze[x][y]=='0')
                    {
                              p.x=x;p.y=y;
                              stack[++top]=p;
                              maze[x][y]='.';
                              ok=1;
                    }
                    i++;
          }
          if(i==8)
          {
                    maze[p.x][p.y]='*';
                    top--;
          }
}while((top>0)&&((p.x!=m)||(p.y!=m)));
//输出探路结果
if(top==0) printf("没有路径\n");
```

```
        else printf("有路径\n");
        //输出探路迷宫留下的踪迹
        for(x=1;x<=m;x++)
        {
                printf("\n");
                for(y=1;y<=m;y++)
                        printf("%c ",maze[x][y]);

        }
        printf("\n");
    system("pause");
}
```

四. 调试分析：

测试数据和结果：

当设定迷宫为10*10：

当设定迷宫为5*5：

算法时间复杂度：

O(m2)

对相关问题的思考：

这个迷宫问题的算法中，要在开始设置迷宫的大小。在探索迷宫路线的过程中，是通过不断的尝试来得到最终的结果，由于不能对已经设定为可走路径进行返回，所以在这个算法中有时候可能不能得到走出迷宫的路径。如下：