****

**《程序设计综合实践》设计文档**

# 第2章 第2次作业

**学号： 21052115**

**姓名： 姜曜楠**

完成时间 2022 年 3 月

## 1、题目

## 2、程序设计思路

（1）数据结构：

1.定义别名为 SGrid 的结构体，用来存储迷宫格子的状态；

2.定义别名为SPoint的结构体，用来表示位置概念；

3. 定义别名为SMaze的结构体，用来存储迷宫问题的一系列属性值（迷宫的总长和总宽，迷宫的所有格子，现在处于的位置，以及是否发现有效路径）；

4.为方便操作，使用enum关键字来罗列迷宫格子可能存在的四种状态（空地|墙体|足迹|入口|出口）；

（2）功能模块：

1.定义loadMaze函数，用于从文件中导入迷宫，同时具有更新迷宫的作用；

2.定义getRowsAndCols静态函数，获取即将设置的迷宫所对应的总长和总宽，并存储在SMaze结构体对应的成员中；

3.定义allocMaze静态函数，用于初始化SMaze结构体；

4.定义setImportantPoints函数，用来设置迷宫入口和出口，其内部由setEntrance和setGoal两个静态函数构成；

5.定义isValid静态函数，用于检查设置的入口点和出口点在当前迷宫是否有效；

6.定义outOfBound静态函数，用于检查设置的点是否处于当前迷宫的内部及边界上

7.定义tryMazeSolution函数，采用深度优先搜索算法寻找迷宫的一条有效路径；

8.定义checkMazeSolution静态函数，用于检查当前路径是否有效，并在到达终点时调用printMaze函数

9. 定义printMaze函数，用于输出迷宫所有格子现在的状态；

## 3、程序源码

/\* Maze.c -- 迷宫问题的接口实现 \*/

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <assert.h>

#include "Maze.h"

/\* 数据类型定义 \*/

typedef struct SGrid

{

    int status;

} SGrid;

typedef struct SPoint

{

    int row;

    int col;

} SPoint;

typedef struct SMaze

{

    int rows, cols;

    SPoint current;

    SGrid \*\*grids;

    int hasFound;

} SMaze;

/\* 局部函数原型 \*/

static void getRowsAndCols(SMaze \*pProblem, const char \*fileName);

static void allocMaze(SMaze \*pProblem);

static void setEntrance(SMaze \*pProblem);

static void setGoal(SMaze \*pProblem);

static bool outOfBound(SPoint temp, const SMaze \*pProblem);

static bool isValid(SPoint point, const SMaze \*pProblem);

static bool checkMazeSolution(SMaze \*pProblem);

/\* 接口函数 \*/

void loadMaze(SMaze \*pProblem, const char \*fileName)

{

    getRowsAndCols(pProblem, fileName);

    allocMaze(pProblem);

    FILE \*fp;

    fp = fopen(fileName, "r");

    assert(fp != NULL);

    for (int i = 0; i < pProblem->rows; i++)

    {

        for (int j = 0; j < pProblem->cols; j++)

        {

            int temp;

            fscanf(fp, "%d", &temp);

            pProblem->grids[i][j].status = temp;

            if (temp > goal)

            {

                printf("Error digit!");

                system("pause");

                exit(1);

            }

        }

    }

    fclose(fp);

    pProblem->hasFound = 0;

}

void setImportantPoints(SMaze \*pProblem)

{

    setEntrance(pProblem);

    setGoal(pProblem);

}

void printMaze(const SMaze \*pProblem)

{

    char iPrint;

    for (int i = 0; i < pProblem->rows; ++i)

    {

        printf("\n\t");

        for (int j = 0; j < pProblem->cols; ++j)

        {

            switch (pProblem->grids[i][j].status)

            {

            case empty:

                iPrint = ' ';

                break;

            case wall:

                iPrint = '+';

                break;

            case crumb:

                iPrint = '.';

                break;

            case entrance:

                iPrint = 'S';

                break;

            case goal:

                iPrint = 'P';

                break;

            }

            printf("%c", iPrint);

        }

    }

    printf("\n");

}

void tryMazeSolution(SMaze \*pProblem)

{

    if (pProblem->grids[pProblem->current.row][pProblem->current.col].status != wall)

    {

        pProblem->grids[pProblem->current.row][pProblem->current.col].status = crumb;

        pProblem->current.row += 1;

        if (checkMazeSolution(pProblem) == true)

        {

            tryMazeSolution(pProblem);

        }

        pProblem->current.row -= 1;

        pProblem->current.col += 1;

        if (checkMazeSolution(pProblem) == true)

        {

            tryMazeSolution(pProblem);

        }

        pProblem->current.col -= 1;

        pProblem->current.col -= 1;

        if (checkMazeSolution(pProblem) == true)

        {

            tryMazeSolution(pProblem);

        }

        pProblem->current.col += 1;

        pProblem->current.row -= 1;

        if (checkMazeSolution(pProblem) == true)

        {

            tryMazeSolution(pProblem);

        }

        pProblem->current.row += 1;

        pProblem->grids[pProblem->current.row][pProblem->current.col].status = empty;

    }

}

/\* 局部函数 \*/

static void getRowsAndCols(SMaze \*pProblem, const char \*fileName)

{

    assert(pProblem != NULL);

    FILE \*fp = fopen(fileName, "r");

    if (fp == NULL)

    {

        perror("Error");

    }

    int cols = 0, leftRows = 0;

    char str[1024];

    fgets(str, 1024, fp);

    for (int i = 0; str[i] != '\n'; i++)

    {

        if (str[i] == ',' || str[i] == ' ')

        {

            continue;

        }

        else

        {

            cols++;

        }

    }

    pProblem->cols = cols;

    while (fgets(str, 1024, fp) != NULL)

    {

        leftRows++;

    }

    pProblem->rows = leftRows + 1;

    fclose(fp);

}

static void allocMaze(SMaze \*pProblem)

{

    assert(pProblem != NULL);

    pProblem->grids = (SGrid \*\*)malloc(pProblem->rows \* sizeof(SGrid \*));

    for (int i = 0; i < pProblem->rows; i++)

    {

        pProblem->grids[i] = (SGrid \*)malloc(pProblem->cols \* sizeof(SGrid));

    }

    pProblem->current.col = pProblem->current.row = -1;

}

static bool checkMazeSolution(SMaze \*pProblem)

{

    if (pProblem->hasFound == 1)

    {

        return false;

    }

    switch (pProblem->grids[pProblem->current.row][pProblem->current.col].status)

    {

    case wall:

    case crumb:

        return false;

    case goal:

        printMaze(pProblem);

        pProblem->hasFound = 1;

    case empty:

        return true;

    }

}

static void setEntrance(SMaze \*pProblem)

{

    SPoint temp;

    printf("\nPlease input the row of the entrance (1 - %d): ", pProblem->rows);

    scanf("%d", &(temp.row));

    printf("\nPlease input the col of the entrance (1 - %d): ", pProblem->cols);

    scanf("%d", &(temp.col));

    while (!isValid(temp, pProblem))

    {

        printf("\nError: The point can't be an entrance!");

        printf("\nPlease input a new row (1 - %d): ", pProblem->rows);

        scanf("%d", &(temp.row));

        printf("\nPlease input a new col (1 - %d): ", pProblem->cols);

        scanf("%d", &(temp.col));

    }

    pProblem->grids[temp.row - 1][temp.col - 1].status = entrance;

    /\* 设置current点初始位置 \*/

    pProblem->current.row = temp.row - 1;

    pProblem->current.col = temp.col - 1;

}

static void setGoal(SMaze \*pProblem)

{

    SPoint temp;

    printf("\nPlease input the row of the goal (1 - %d): ", pProblem->rows);

    scanf("%d", &(temp.row));

    printf("\nPlease input the col of the goal (1 - %d): ", pProblem->cols);

    scanf("%d", &(temp.col));

    while (!isValid(temp, pProblem))

    {

        printf("\nError: The point can't be an goal!");

        printf("\nPlease input a new row (1 - %d): ", pProblem->rows);

        scanf("%d", &(temp.row));

        printf("\nPlease input a new col (1 - %d): ", pProblem->cols);

        scanf("%d", &(temp.col));

    }

    pProblem->grids[temp.row - 1][temp.col - 1].status = goal;

}

static bool isValid(SPoint point, const SMaze \*pProblem)

{

    if (!outOfBound(point, pProblem))

    {

        if (point.row == 1 && point.col != 1)

        {

            return true;

        }

        if (point.row != 1 && point.col == 1)

        {

            return true;

        }

        if (point.row == pProblem->rows && point.col != pProblem->cols)

        {

            return true;

        }

        if (point.row != pProblem->rows && point.col == pProblem->cols)

        {

            return true;

        }

    }

    return false;

}

static bool outOfBound(SPoint point, const SMaze \*pProblem)

{

    if (point.row < 1 || point.row > pProblem->rows ||

        point.col < 1 || point.col > pProblem->cols)

    {

        return true;

    }

    return false;

}

/\* 测试函数 \*/

void mazeTest(void)

{

    SMaze problem, \*p = &problem;

    char \*fileName = "C\\Default.txt";

    loadMaze(p, fileName);

    printf("\nThe default maze is: ");

    printMaze(p);

    setImportantPoints(p);

    printMaze(p);

    printf("\n\n\tPress Enter to choose show the road, others to load from maze.txt: ");

    fflush(stdin);

    char ch;

    while ((ch = getchar()) != '\n')

    {

        scanf("%s", &fileName);

        loadMaze(p, fileName);

        printf("\nThe new maze is: ");

        printMaze(p);

        setImportantPoints(p);

        printMaze(p);

        printf("\n\n\tPress Enter to choose show the road, others to load from maze.txt: ");

    }

    tryMazeSolution(p);

    if (p->hasFound == 0)

    {

        printf("There is no road to the goal of this maze.");

    }

}

int main(void)

{

    while (1)

    {

        mazeTest();

        system("pause");

    }

    return 0;

}

/\* Maze.h -- 迷宫问题的接口 \*/

#ifndef MAZE\_H\_INCLUDED

#define MAZE\_H\_INCLUDED

enum gridStatus

{

    empty,

    wall,

    crumb,

    entrance,

    goal

};

/\* 数据类型声明 \*/

typedef struct SGrid SGrid;

typedef struct SPoint SPoint;

typedef struct SMaze SMaze;

/\* 函数原型 \*/

void loadMaze(SMaze \*pProblem, const char \*fileName);

void setImportantPoints(SMaze \*pProblem);

void tryMazeSolution(SMaze \*pProblem);

void printMaze(const SMaze \*pProblem);

#endif

## 4、运行截图



