# 数据结构实验报告——随堂测试

**学号：__20201060330__姓名：___胡诚皓___得分：_____**

## 一、问题描述

将老鼠放到没有盖子的迷宫盒子入口，盒子出口处放上奶酪引导老鼠穿过迷宫。迷宫盒子中存在很多通路，但有的通路路径上被障碍物阻挡无法通行。老鼠可以通过不断尝试来找到出口获得奶酪。请给出一条能让老鼠从迷宫入口到出口获得奶酪的路径。

## 二、实验内容

**1. （必做题）阅读程序，回答下面问题**

**（1）程序使用了什么数据结构，目的是什么？**

答：使用了链栈的结构，目的是使用栈来实现深度优先搜索 DFS 以实现对迷宫的路径搜索。

**（2）迷宫是如何表示的？**

答：使用二维数组进行表示，每一个元素的数字表示了这一格的性质，如代码注释中所说：1 表示墙，2 表示入口，3 表示出口。

**（3）对老鼠来说，迷宫的实际入口位置是哪里？出口是哪里？当前所处的位置是哪里？**

答：实际的入口位置为(1, 1)，出口为(8, 10)，当前所处的位置为(x, y)

**（4）老鼠前进遵循什么规律？**

答：代码中默认迷宫的右下角必为出口，且迷宫一定是以 1 的闭合曲线包围的，即只有右下角可以真正走出迷宫。

每一步都按顺序分别尝试往北、南、西、东四个方向走，只要走得通就一直不断尝试往前冲。

**（5）理论上，老鼠可以前进的方向有几个，分别是如何表示的？实际老鼠可以前进的方向如何判断？**

理论上，老鼠可以前进的方向有四个，分别为往上（北）、往下（南）、往左（西）、往右（东），分别使用 MAZE[x-1][y]、MAZE[x+1][y]、MAZE[x][y-1]、MAZE[x][y+1]（即上一行同一列、下一行同一列、同一行前一列、同一行后一列）

实际上，老鼠前进的方向有几个约束：

- 不能超出地图边界
- 往前进方向要走的下一步不能是墙

- 不能走已经走过的位置

**（6）请描述程序中实现路径探索的过程**

路径探索的整体思路采用深度优先搜索：

①从实际起点(1, 1)开始

②分别尝试走向北、南、西、东四个方向，若都走不通则判断是否已经走到出口，不是出口则标记当前位置为 2（表示已经走过）。通过出栈一个元素回溯到上一步的位置赋给 x、y

**（7）请说明程序中各函数的功能，并对你认为的关键语句进行注释**

① push

尾插法向链栈中插入一个坐标 x、y 的位置，即向 stack 栈压入位置点元素

② pop

从 stack 栈中弹出一个位置元素，并把弹出后栈顶元素的数据域赋值给 x、y

③ chkExit

判断位置(x, y)是否为出口(ex, ey)位置，并且判断该出口是否位于迷宫的墙边，即判断出口是否真正可以走出迷宫。

**添加注释后的代码：**

```
#include <stdio.h>
#include <stdlib.h>
#define EAST  MAZE[x][y+1]  /*定义东方的相对位置*/
#define WEST  MAZE[x][y-1]  /*定义西方的相对位置*/
#define SOUTH MAZE[x+1][y]    /*定义南方的相对位置*/
#define NORTH MAZE[x-1][y]    /*定义北方的相对位置*/
#define ExitX 8           /*定义出口的 X 坐标在第 8 行*/
#define ExitY 10          /*定义出口的 Y 坐标在第 10 列*/
struct list
{
    int x,y;
    struct list* next;
};
typedef struct list node;
typedef node* link;
//以二维数组表示迷宫地图，1 表示墙、2 表示入口或死路（即回溯回来的路径）、3 表示出口
int MAZE[10][12] = {2,1,1,1,1,0,0,0,1,1,1,1,   /*声明迷宫数组*/
                    1,0,0,0,1,1,1,1,1,1,1,1,
                     1,1,1,0,1,1,0,0,0,0,1,1,
                      1,1,1,0,1,1,0,1,1,0,1,1,
                      1,1,1,0,0,0,0,1,1,0,1,1,
```

```
                    1,1,1,0,1,1,0,1,1,0,1,1,
                    1,1,1,0,1,1,0,1,1,0,1,1,
                    1,1,1,0,1,1,0,0,1,0,1,1,
                    1,1,0,0,0,0,0,0,1,0,0,1,
                    1,1,1,1,1,1,1,1,1,1,1,3};
link push(link stack,int x,int y)
{
    link newnode;
    newnode = (link)malloc(sizeof(node));
    if(!newnode)
    {
        printf("Error!内存分配失败!\n");
        return NULL;
    }
    newnode->x=x;
    newnode->y=y;
    newnode->next=stack;
    stack=newnode;
   return stack;
}
//弹出 stack 栈中的元素，并且将出栈之后栈顶元素的数据域赋值给 x 和 y
link pop(link stack,int* x,int* y)
{
    link top;
    if(stack!=NULL)
    {
        top=stack;
        stack=stack->next;
        *x=top->x;
        *y=top->y;
        free(top);
        return stack;
    }
    else
        *x=-1;
    return stack;
}
//判断以 ex、ey 为终点，当前位置为 x、y 时是否能走出迷宫
int chkExit(int x,int y,int ex,int ey)
{
    if(x==ex&&y==ey)
    {
        if(NORTH==1||SOUTH==1||WEST==1||EAST==2)
            return 1;
```

```
        if(NORTH==1||SOUTH==1||WEST==2||EAST==1)
            return 1;
        if(NORTH==1||SOUTH==2||WEST==1||EAST==1)
            return 1;
        if(NORTH==2||SOUTH==1||WEST==1||EAST==1)
            return 1;
    }
    return 0;
}

int main()
{
    int i,j,x,y;
    link path = NULL;
    x=1;     /*入口的 X 坐标*/
    y=1;    /*入口的 Y 坐标*/
    printf("[迷宫的地模拟图(1 表示墙,2 表示入口,3 表示出口]\n"); /*打印出迷宫的路径图*/
    for(i=0;i<10;i++)
    {
        for(j=0;j<12;j++)
            printf("%2d",MAZE[i][j]);
        printf("\n");
    }
    while(x<=ExitX&&y<=ExitY)
    {
        //将当前走到的位置标记为 6，并且按顺序分别尝试往北、南、西、东四个方向走
        MAZE[x][y]=6;
        if(NORTH==0)
        {
            x -= 1;
            path=push(path,x,y);
        }
        else if(SOUTH==0)
        {
            x+=1;
            path=push(path,x,y);
        }
        else if(WEST==0)
        {
            y-=1;
            path=push(path,x,y);
        }
        else if(EAST==0)
        {
```

```
            y+=1;
            path=push(path,x,y);
        }
        else if(chkExit(x,y,ExitX,ExitY)==1) /*检查是否走到出口了*/
            break;
        else
        {
            MAZE[x][y]=2;
            path=pop(path,&x,&y);
        }
    }
    printf("----------------------------\n");
    printf("[6 表示老鼠走过的路线]\n"); /*打印出老鼠走完迷宫后的路径图*/
    printf("----------------------------\n");
    for(i=0;i<10;i++)
    {
        for(j=0;j<12;j++)
            printf("%2d",MAZE[i][j]);
        printf("\n");
    }
    system("pause");
    return 0;
}
```

**2. (选做题)改进算法（请在编写的程序中注明改进点在哪里。比如，使老鼠闯迷宫游戏可根据获得的不同迷宫地图给出正确通关路径）**

**用户手册：**

改进了地图存储的规则，地图指定规则如下：

- 迷宫必须被一条闭合的 1 的路径包围（不一定是矩形）
- 迷宫地图中的 2 即为老鼠开始走的位置，3 为老鼠的目标终点
- 在地图中只能有一个 2 与一个 3
- 上一条中的 2、3 必须在 1 的闭合路径中

修改或使用新的迷宫地图，请使用变量名 MAZE 存储，并修改初始化中的 m、n、x、y，m 为 MAZE 的行数，n 为 MAZE 的列数，(x, y)为起点的坐标（即 2 的位置）（0-based）

**程序测试：**

使用的还是原来的地图，但根据上述规则修改了地图的符号定义


**源代码：**

7.c

```
#include <stdio.h>
#include <stdlib.h>
//define the relative location of east, west, south and north
#define EAST  MAZE[x][y+1]
#define WEST  MAZE[x][y-1]
#define SOUTH MAZE[x+1][y]
#define NORTH MAZE[x-1][y]

struct list {
    int x, y;
    struct list *next;
};

typedef struct list node;
typedef node *link;
//2-dimension array to define map of maze
int MAZE[10][12] = {1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                    1, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                    1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
                    1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
                    1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
                    1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
```

```
                    1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
                    1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
                    1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 3, 1,
                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
//int MAZE[6][5] = {1, 1, 1, 0, 1,
//                  1, 0, 2, 1, 1,
//                  1, 0, 0, 0, 1,
//                  1, 0, 1, 3, 1,
//                  1, 0, 0, 0, 1,
//                  1, 1, 1, 1, 1};

link push(link stack, int x, int y) {
    link newnode;
    newnode = (link) malloc(sizeof(node));
    if (!newnode) {
        printf("Memory Error!");
        return NULL;
    }
    newnode->x = x;
    newnode->y = y;
    newnode->next = stack;
    return newnode;
}

//enhance the logic of pop
link pop(link stack, int *x, int *y) {
    link top = stack;
    //back to last step
    stack = stack->next;
    if (stack != NULL) {
        *x = stack->x;
        *y = stack->y;
        free(top);
    } else {
        //back to the beginning, means no way to exit
        *x = -1;
    }
    return stack;
}

//rewrite the logic of chkExit
//able to reach exit from (x,y) directly
int chkExit(int x, int y) {
    return SOUTH == 3 || WEST == 3 || NORTH == 3 || EAST == 3;
```

```
}

int main() {
    int i, j, x, y;
    int m, n;
    //flag used for main loop
    int flag = 0;
    link path = NULL;
    //initial entrance coordinate
    x = 1, y = 1;
    //initial the size of MAZE
    m = 10, n = 12;
    printf("MAZE map (1 for wall, 0 for road, 2 for entrance, 3 for exit):\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            printf("%2d", MAZE[i][j]);
        printf("\n");
    }

    while (flag != 1) {
        //walk to north, south, west and east in sequence
        if (NORTH == 0) {
            x -= 1;
            path = push(path, x, y);
        } else if (SOUTH == 0) {
            x += 1;
            path = push(path, x, y);
        } else if (WEST == 0) {
            y -= 1;
            path = push(path, x, y);
        } else if (EAST == 0) {
            y += 1;
            path = push(path, x, y);
        } else if (chkExit(x, y) != 0) {
            //set flag to 1 if reach the exit
            flag = 1;
        } else {
            //use 2 to mark the road to dead end
            MAZE[x][y] = 2;
            //dead end, get back to previous position
            path = pop(path, &x, &y);
            //unreachable maze, mark to quit the loop
            if (x == -1)
                flag = 1;
```

```
        }
        //use 6 to mark passed road
        MAZE[x][y] = 6;
    }

    //distinguish unreachable maze
    if (path == NULL) {
        printf("No chance to get out of the maze :(");
    } else {
        //display the map with trace 6
        printf("--------------------------\n");
        printf("6 for the passed road.\n");
        printf("--------------------------\n");
        for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++)
                printf("%2d", MAZE[i][j]);
            printf("\n");
        }
    }

    system("pause");
    return 0;
}
```