Experiment topic: **Solving the maze problem**

1. Experimental content:

Project 1 Maze Problem Solving (8 credit hours)

1. Description of the problem: A maze is represented by an m*n rectangular matrix, and 0 and 1 represent paths and obstacles in the maze, respectively. Design a program to find a path from the entrance to the exit for any maze set, or draw the conclusion that there is no path.

2. basic requirements

(1) First implement a stack type with a linked list as the storage structure, and then write a non-recursive program to solve the maze. The obtained paths are output in the form of triplets (i, j, d). Among them: (i, j) indicates a coordinate in the maze, and d indicates the direction to go to the next coordinate. For example, for the maze shown in Figure 3.4 on page 50 of the textbook, the output path is: (1, 1, 1), (1, 2, 2), (2, 2, 2), (3, 2, 3) , (3, 1, 2), . . .

(2) Write a recursive algorithm to find all possible paths in the maze.

(3) Output the maze and its paths in the form of a square matrix.

(4) Design independently according to the requirements of the topic, and write a design report as required after the design is completed .

2. Design ideas

1. Prepare

（1）This experiment needs to use a linked list, and prepare related linked list functions

（2）Prepare a txt text to store the labyrinth required for the experiment, and then read and use it directly in the program.

（3）Design an input program part of the user terminal, and the user inputs the coordinates of the start point and end point of the maze.

（4）Output the maze in txt, add "wall" to the maze.

2、non-recursive

（1）Non-recursive thinking: take the east, west, and northwest (bottom right, top left) as the basic order. Take the starting point of the maze as the starting point. Detect around the point, first detect whether there is a road on the right, if there is, go to the right; if not, detect the bottom, left, and top in turn, if there is no path, return to no path, and mark the path traveled .

（2）Non-recursion needs to take into account the possibility of going back. The idea of the algorithm is that when the point reaches a position where there is no untraveled road (the surrounding points are either walls or roads), take a step back and give priority to Point to the next direction of the direction you are going (for example, go right just now, now start from down first, if the following does not work, then detect left and up), continue to detect, if there are still no untraveled points in the remaining direction, continue to go back, and keep going to the correct drop-in route.

（3）Through the above steps, find the path to the end.

（4）Algorithm implementation: mark the path traveled with 2. Set two structure variables, one stores the current point, and the other stores a detection point and changes with the detection situation. If the detection is successful, the one with the current point will be pushed into the stack, and it will be replaced with the detection success point. The original detection point is changed to the next attempt point of the successful detection point to continue detection. The up, down, left, and right directions of the detection point are realized through i++;i--;j++;j--.

（5）After finding the path, build another stack, pour the existing stack into another stack, and pour out another stack, so as to sequentially output the routes.

3. recursion

（1）Simultaneously detect the surroundings from the current point. If there is a road at a certain point around, it will return success and mark the current point as the road that has been traveled.

（2）When the point is surrounded by walls or roads that have been traveled, the return fails. If a certain point around is the end point, return success.

（3）Through the above algorithm to record the points that have passed, the exit path of the maze can be obtained and output.

4. experimental code

Two versions are given below: (divided into single-file version and multi-file version)

Code (single-file version):

```
#include <stdio.h>
#include <stdlib.h>

#define M 11
#define N 10

struct dot //Define the coordinate type of the points in the maze
{
	int x;
	int y;
};
struct Element //Define the chain stack element
{
	int x,y; //x row, y column
	int d; //d the direction of the next step
};
typedef struct LStack //chain stack
{
	Element elem;
	struct LStack *next;
}*LinkStack;
/**************stack function****************/
```

```
int InitStack(LinkStack &S)//construct an empty stack
{
        S=NULL;
        return 1;
}
int StackEmpty(LinkStack S)//judging whether the stack is empty
{
        if(S==NULL)
                return 1;
        else
                return 0;
}
int Push(LinkStack &S, Element e)//Push new data elements onto the stack
{
        LinkStack p;
        p=(LinkStack)malloc(sizeof(LStack));
        p->elem=e;
        p->next=S;
        S=p;
        return 1;
}
int Pop(LinkStack &S,Element &e) //The top element of the stack is popped out of the stack
{
        LinkStack p;
        if(!StackEmpty(S))
        {

                e=S->elem;
                p=S;
                S=S->next;
                free(p);
                return 1;
        }
        else
                return 0;
}
/**************** Non-recursive maze path function***********************/
void MazePath(struct dot start,struct dot end,int maze[M][N],int diracswitch[5][2])
{
        int i,j,d;int a,b;
        Element elem,e;
        LinkStack S1, S2;
        InitStack(S1);
        InitStack(S2);
```

```
    maze[start.x][start.y]=2; //mark the entry point
    elem.x=start.x;
    elem.y=start.y;
    elem.d=0; //starts at 0
    Push(S1,elem);
    while(!StackEmpty(S1)) //The stack is not empty and there is a path to go
    {
        Pop(S1,elem);
        i=elem.x;
        j=elem.y;
        d=elem.d+1; //Next direction (skip previous unavailable direction)
        while(d<5) //Explore all directions from bottom right to top left
        {
            a=i+diracswitch[d][0];
            b=j+diracswitch[d][1]; //At the first pass, try to turn right
            if(a==end.x && b==end.y && maze[a][b]==0) //If it reaches the exit
            {
                elem.x=i;
                elem.y=j;
                elem.d=d;
                Push(S1,elem);
                elem.x=a;
                elem.y=b;
                elem.d=666; //direction output is 0 to judge whether it has reached the exit
                Push(S1,elem);
                printf("\n1=Right 2=Down 3=Left 4=Up 666 is to get out of the maze\n\nThe
path is: (row coordinates, column coordinates, direction) (direction 666 represents reaching the
end)\n");
                while(S1) //Reverse the sequence and output the maze path sequence
                {
                    Pop(S1,e);
                    Push(S2,e);
                }
                while(S2)
                {
                    Pop(S2,e);
                    maze[ex][ey]=3;
                    printf("\n(%d,%d,%d)",ex,ey,ed);
                }
                return; //Jump out of the two-layer loop
            }
            if(maze[a][b]==0) //Find a non-exit point that can move forward
            {
                maze[a][b]=2; //Mark walks through this point
```

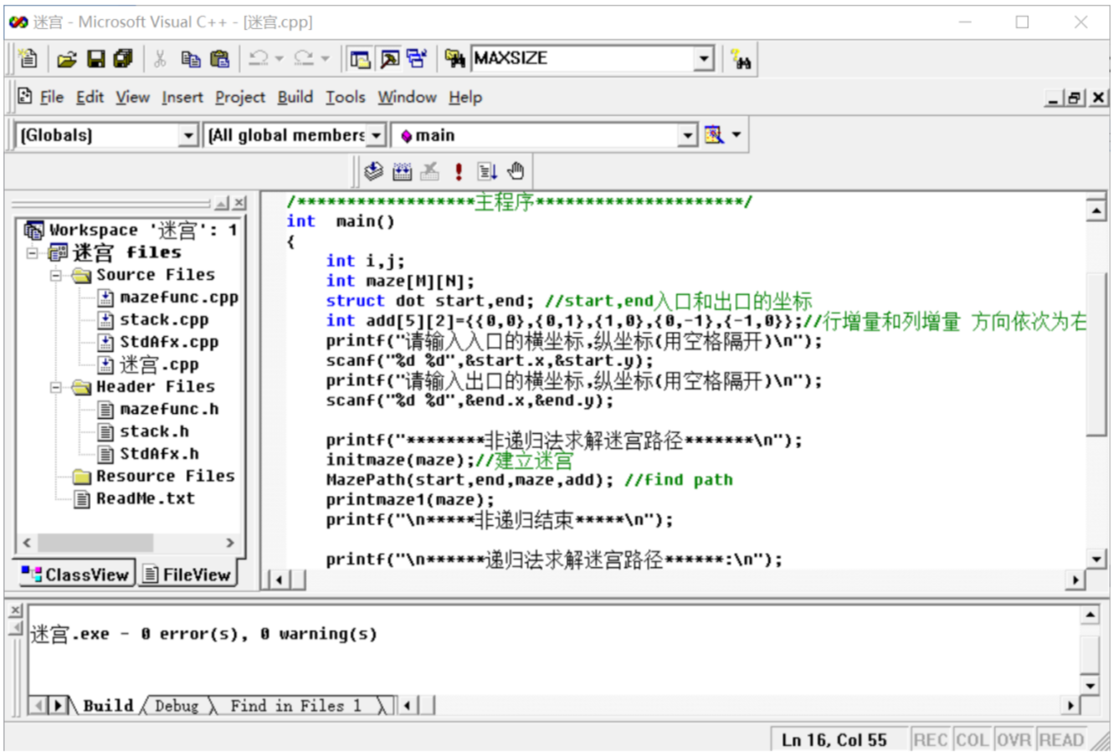
```
                    elem.x=i;
                    elem.y=j;
                    elem.d=d;
                    Push(S1,elem); //The current position is pushed into the stack
                    i=a; //The next point is converted to the current point
                    j=b;
                    d=0;
                }
                d++;
            }
        }
        printf("There is no way to get out of this maze\n");
    }
    /************Build Maze*****************/
    void initmaze(int maze[M][N])
    {
        int m=9,n=8;
        int i,j;
        FILE *fp;
        if((fp=fopen("maze.txt","r"))==NULL)
        {
            printf("Open the file failure...\n");
            exit(0);
        }
        printf("Read file: maze.txt\n");
        for(i=1;i<=m;i++)
            for(j=1;j<=n;j++)
    fscanf(fp,"%d%*[^0-9]",&maze[i][j]);
        fclose(fp);
        printf("The maze you built is (the outermost circle is the wall)...\n");
        for(i=0;i<=m+1;i++) //Add a circle of walls
        {
            maze[i][0]=1;
            maze[i][n+1]=1;
        }
        for(j=0;j<=n+1;j++)
        {
            maze[0][j]=1;
            maze[m+1][j]=1;
        }
        for(i=0;i<=m+1;i++) // output maze
        {
            for(j=0;j<=n+1;j++)
                printf("%d ",maze[i][j]);
```

```
			printf("\n");
		}
	printf("\nOutput the original maze in the form of a square matrix:");
		printf("\n");
		for(i=0; i<11; i++)
		{
			for(j=0;j<10;j++)
			{
				if(maze[i][j]==1)
					printf("■");
				else
					printf(" ");
			}
			printf("\n");
		}
	}

/****************Recursively find the maze path function****************/
int VistMaze(int maze[M][N], int i, int j, struct dot start, struct dot end)
{
	int way = 0;

	maze[i][j] = 2; //Assume the point can go through

	//If you reach the key point, set way to 1 to indicate that the maze has ended
	if (i == 9 && j == 8)
	{
		way = 1;
	}
	/**If the maze is not finished, it will search whether the four directions of the right, bottom,
left, and top of the location can be passed**/
	if (way != 1 && j + 1 <= end.y && maze[i][j + 1] == 0)

	{		//right
		if (VistMaze(maze,i,j+1,start,end) == 1)
			return 1;
	}
	if (way!=1&&i+1<=end.x&&maze[i+1][j]==0)
	{		//down
		if (VistMaze(maze,i+1,j,start,end) == 1)
			return 1;
	}
	if (way!=1&&j-1>=start.y&&maze[i][j-1]==0)
	{	// left
```

```
		if (VistMaze(maze, i, j - 1, start, end) == 1)
			return 1;
	}
	if (way!=1&&i-1>=start.x&&maze[i-1][j]==0)
	{	//up
		if (VistMaze(maze, i - 1, j, start, end) == 1)
			return 1;
	}
		//Set the point back to 0 when there is no connection around
	if(way!=1)
		maze[i][j]=0;

	return way;
}

//print non-recursive maze route
void printmaze1(int maze[M][N])
{	int i,j;
	printf("\n");
	// print out the maze and path
	printf("\nNon-recursive path:\n");
	for(i=0;i<11;i++)
	{
		for(j=0;j<10;j++)
		{
			if (maze[i][j]==0||maze[i][j]==2)
				printf(" ");
			else if(maze[i][j]==1)
				printf("■");
			else if(maze[i][j]==3)
				printf("<>");
		}
		printf("\n");
	}
}

/********Print recursive maze route********/
void printmaze2(int maze[M][N])
{
	int i,j;
	printf("\nRecursively obtained path:\n");/**Print out the maze and path**/
	for(i=0;i<11;i++)
	{
		for(j=0;j<10;j++)
```

```
            {
                if (maze[i][j]==0)
                    printf("  ");
                else if(maze[i][j]==1)
                    printf("■");
                else if(maze[i][j]==2)
                    printf("<>");
            }
            printf("\n");
        }
}
/******************The main program*******************/
int main()
{
        int i,j;
        int maze[M][N];
        struct dot start, end; //start, end coordinates of entrance and exit
        int   add[5][2]={{0,0},{0,1},{1,0},{0,-1},{-1,0}};//row   increment   and   column   Incremental
direction is bottom right, top left
        printf("Please enter the abscissa and ordinate of the entrance (separated by spaces)\n");
scanf("%d %d",&start.x,&start.y);
printf("Please enter the abscissa and ordinate of the exit (separated by spaces)\n");
scanf("%d %d",&end.x,&end.y);

printf("********Non-recursive method to solve the maze path********\n");
        initmaze(maze);//Build maze
        MazePath(start,end,maze,add); //find path
        printmaze1(maze);
        printf("\n*****Non-recursive end*****\n");

printf("\n******Recursive method to solve the maze path******:\n");
        printf("\nReread maze from file maze.txt\n");
        initmaze(maze);//Build maze
        if (VistMaze(maze,1,1,start,end)==0)
        {
            printf("There is no path to go\n");
            exit(0);
        }
        printmaze2(maze);
        printf("*******recursive end******\n");

        printf("*****Program End*****\n");
        return 0;
}
```

Code (multi-file version):



```
/******************主程序**********************/
int  main()
{
    int i,j;
    int maze[M][N];
    struct dot start,end; //start,end入口和出口的坐标
    int add[5][2]={{0,0},{0,1},{1,0},{0,-1},{-1,0}};//行增量和列增量 方向依次为右
    printf("请输入入口的横坐标,纵坐标(用空格隔开)\n");
    scanf("%d %d",&start.x,&start.y);
    printf("请输入出口的横坐标,纵坐标(用空格隔开)\n");
    scanf("%d %d",&end.x,&end.y);

    printf("********非递归法求解迷宫路径*******\n");
    initmaze(maze);//建立迷宫
    MazePath(start,end,maze,add); //find path
    printmaze1(maze);
    printf("\n*****非递归结束*****\n");

    printf("\n******递归法求解迷宫路径******:\n");
```

```
迷宫.exe - 0 error(s), 0 warning(s)
```

mazefunc.cpp (storage and reading maze, maze solution related functions)

```
#include "StdAfx.h"
#include "mazefunc.h"
#include "stack.h"
/*************** Non-recursive maze path function***********************/
void MazePath(struct dot start,struct dot end,int maze[M][N],int diracswitch[5][2])
//Non-recursive maze path function
{
    int i,j,d;int a,b;
    Element elem,e;
    LinkStack S1, S2;
    InitStack(S1);
    InitStack(S2);
    maze[start.x][start.y]=2; //mark the entry point
    elem.x=start.x;
    elem.y=start.y;
    elem.d=0; //starts at 0
    Push(S1,elem);
    while(!StackEmpty(S1)) //The stack is not empty and there is a path to go
    {
        Pop(S1,elem);
        i=elem.x;
        j=elem.y;
        d=elem.d+1; //Next direction (skip previous unavailable direction)
        while(d<5) //Explore all directions from bottom right to top left
        {
            a=i+diracswitch[d][0];
            b=j+diracswitch[d][1]; //At the first pass, try to turn right
            if(a==end.x && b==end.y && maze[a][b]==0) //If it reaches the exit
            {
                elem.x=i;
                elem.y=j;
                elem.d=d;
                Push(S1,elem);
                elem.x=a;
                elem.y=b;
                elem.d=666; //direction output is 0 to judge whether it has reached the exit
                Push(S1,elem);
                printf("\n1=Right 2=Down 3=Left 4=Up 666 is to get out of the maze\n\nThe
path is: (row coordinates, column coordinates, direction) (direction 666 represents reaching the
end)\n");
                while(S1) //Reverse the sequence and output the maze path sequence
                {
                    Pop(S1,e);
                    Push(S2,e);
                }
                while(S2)
                {
                    Pop(S2,e);
                    maze[ex][ey]=3;
                    printf("\n(%d,%d,%d)",ex,ey,ed);
                }
                return; //Jump out of the two-layer loop
            }
            if(maze[a][b]==0) //Find a non-exit point that can move forward
            {
                maze[a][b]=2; //Mark walks through this point
                elem.x=i;
                elem.y=j;
                elem.d=d;
                Push(S1,elem); //The current position is pushed into the stack
                i=a; //The next point is converted to the current point
                j=b;
                d=0;
            }
            d++;
        }
    }
```

```
	printf("There is no way to get out of this maze\n");
}
/************Build Maze*****************/
void initmaze(int maze[M][N]) //Build a maze
{
	int m=9,n=8;
	int i,j;
	FILE *fp;
	if((fp=fopen("maze.txt","r"))==NULL)
	{
		printf("Open the file failure...\n");
		exit(0);
	}
	printf("Read file: maze.txt\n");
	for(i=1;i<=m;i++)
		for(j=1;j<=n;j++)
fscanf(fp,"%d%*[^0-9]",&maze[i][j]);
	fclose(fp);
	printf("The maze you built is (the outermost circle is the wall)...\n");
	for(i=0;i<=m+1;i++) //Add a circle of walls
	{
		maze[i][0]=1;
		maze[i][n+1]=1;
	}
	for(j=0;j<=n+1;j++)
	{
		maze[0][j]=1;
		maze[m+1][j]=1;
	}
	for(i=0;i<=m+1;i++) // output maze
	{
		for(j=0;j<=n+1;j++)
			printf("%d ",maze[i][j]);
		printf("\n");
	}
printf("\nOutput the original maze in the form of a square matrix:");
	printf("\n");
	for(i=0; i<11; i++)
	{
		for(j=0;j<10;j++)
		{
			if(maze[i][j]==1)
				printf("■");
			else
```

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

/****************Recursively find the maze path function****************/
int VistMaze(int maze[M][N], int i, int j, struct dot start, struct dot end) //recursively find the
maze path function
{
        int way = 0;

        maze[i][j] = 2; //Assume the point can go through

        //If you reach the key point, set way to 1 to indicate that the maze has ended
        if (i == 9 && j == 8)
        {
                way = 1;
        }
        /**If the maze is not finished, it will search whether the four directions of the right, bottom,
left, and top of the location can be passed**/
        if (way != 1 && j + 1 <= end.y && maze[i][j + 1] == 0)

        {               // right
                if (VistMaze(maze,i,j+1,start,end) == 1)
                        return 1;
        }
        if (way!=1&&i+1<=end.x&&maze[i+1][j]==0)
        {               //down
                if (VistMaze(maze,i+1,j,start,end) == 1)
                        return 1;
        }
        if (way!=1&&j-1>=start.y&&maze[i][j-1]==0)
        {       // left
                if (VistMaze(maze, i, j - 1, start, end) == 1)
                        return 1;
        }
        if (way!=1&&i-1>=start.x&&maze[i-1][j]==0)
        {       //up
                if (VistMaze(maze, i - 1, j, start, end) == 1)
                                                        return 1;
        }
                //Set the point back to 0 when there is no connection around
        if(way!=1)
```

```
            maze[i][j]=0;

        return way;
}

/*****print non-recursive maze route*****/
void printmaze1(int maze[M][N]) //print non-recursive maze route
{       int i,j;
        printf("\n");
        // print out the maze and path
        printf("\nNon-recursive path:\n");
        for(i=0;i<11;i++)
        {
                for(j=0;j<10;j++)
                {
                        if (maze[i][j]==0||maze[i][j]==2)
                                printf(" ");
                        else if(maze[i][j]==1)
                                printf("■");
                        else if(maze[i][j]==3)
                                printf("<>");
                }
                printf("\n");
        }
}

/********Print recursive maze route********/
void printmaze2(int maze[M][N]) //print recursive maze route
{
        int i,j;
        printf("\nRecursively obtained path:\n");/**Print out the maze and path**/
        for(i=0;i<11;i++)
        {
                for(j=0;j<10;j++)
                {
                        if (maze[i][j]==0)
                                printf(" ");
                        else if(maze[i][j]==1)
                                printf("■");
                        else if(maze[i][j]==2)
                                printf("<>");
                }
                printf("\n");
        }
```

```
}
```

stack.cpp

```
#include "StdAfx.h"
#include "stack.h"

/*************stack function***************/
int InitStack(LinkStack &S)//construct an empty stack
{
    S=NULL;
    return 1;
}
int StackEmpty(LinkStack S)//judging whether the stack is empty
{
    if(S==NULL)
        return 1;
    else
        return 0;
}
int Push(LinkStack &S, Element e)//Push new data elements onto the stack
{
    LinkStack p;
    p=(LinkStack)malloc(sizeof(LStack));
    p->elem=e;
    p->next=S;
    S=p;
    return 1;
}
int Pop(LinkStack &S,Element &e) //The top element of the stack is popped out of the stack
{
    LinkStack p;
    if(!StackEmpty(S))
    {

        e=S->elem;
        p=S;
        S=S->next;
        free(p);
        return 1;
    }
    else
        return 0;
}
```

StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//     maze.pch will be the pre-compiled header
//     stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

Maze.cpp

```
// Maze.cpp : Defines the entry point for the console application.
//
#include "stdAfx.h"
#include "stack.h"
#include "mazefunc.h"

/******************The main program*******************/
int main()
{
    int i,j;
    int maze[M][N];
    struct dot start, end; //start, end coordinates of entrance and exit
    int  add[5][2]={{0,0},{0,1},{1,0},{0,-1},{-1,0}};//row  increment  and  column  Incremental
direction is bottom right, top left
    printf("Please enter the abscissa and ordinate of the entrance (separated by spaces)\n");
scanf("%d %d",&start.x,&start.y);
printf("Please enter the abscissa and ordinate of the exit (separated by spaces)\n");
scanf("%d %d",&end.x,&end.y);

printf("********Non-recursive method to solve the maze path********\n");
    initmaze(maze);//Build maze
    MazePath(start,end,maze,add); //find path
    printmaze1(maze);
    printf("\n*****Non-recursive end*****\n");

printf("\n******Recursive method to solve the maze path******:\n");
    printf("\nReread maze from file maze.txt\n");
    initmaze(maze);//Build maze
    if (VistMaze(maze,1,1,start,end)==0)
    {
        printf("There is no path to go\n");
        exit(0);
    }
```

```
	printmaze2(maze);
	printf("*******recursive end******\n");

	printf("*****Program End*****\n");
	return 0;
}
```

mazefunc.h

```
struct dot //Define the coordinate type of the points in the maze
{
	int x;
	int y;
};
void MazePath(struct dot start,struct dot end,int maze[M][N],int diracswitch[5][2]);
//Non-recursive maze path function
void initmaze(int maze[M][N]); //Build maze
int VistMaze(int maze[M][N], int i, int j, struct dot start, struct dot end); //recursively find the
maze path function
void printmaze1(int maze[M][N]); //print non-recursive maze route
void printmaze2(int maze[M][N]); //print recursive maze route
```

stack.h

```
struct Element //Define the chain stack element
{
	int x,y; //x row, y column
	int d; //d the direction of the next step
};
typedef struct LStack //chain stack
{
	Element elem;
	struct LStack *next;
}*LinkStack;
int InitStack(LinkStack &S);//Construct an empty stack
int StackEmpty(LinkStack S);//judging whether the stack is empty
int Push(LinkStack &S, Element e);//Push new data elements onto the stack
int Pop(LinkStack &S,Element &e); //The top element of the stack is popped out
```

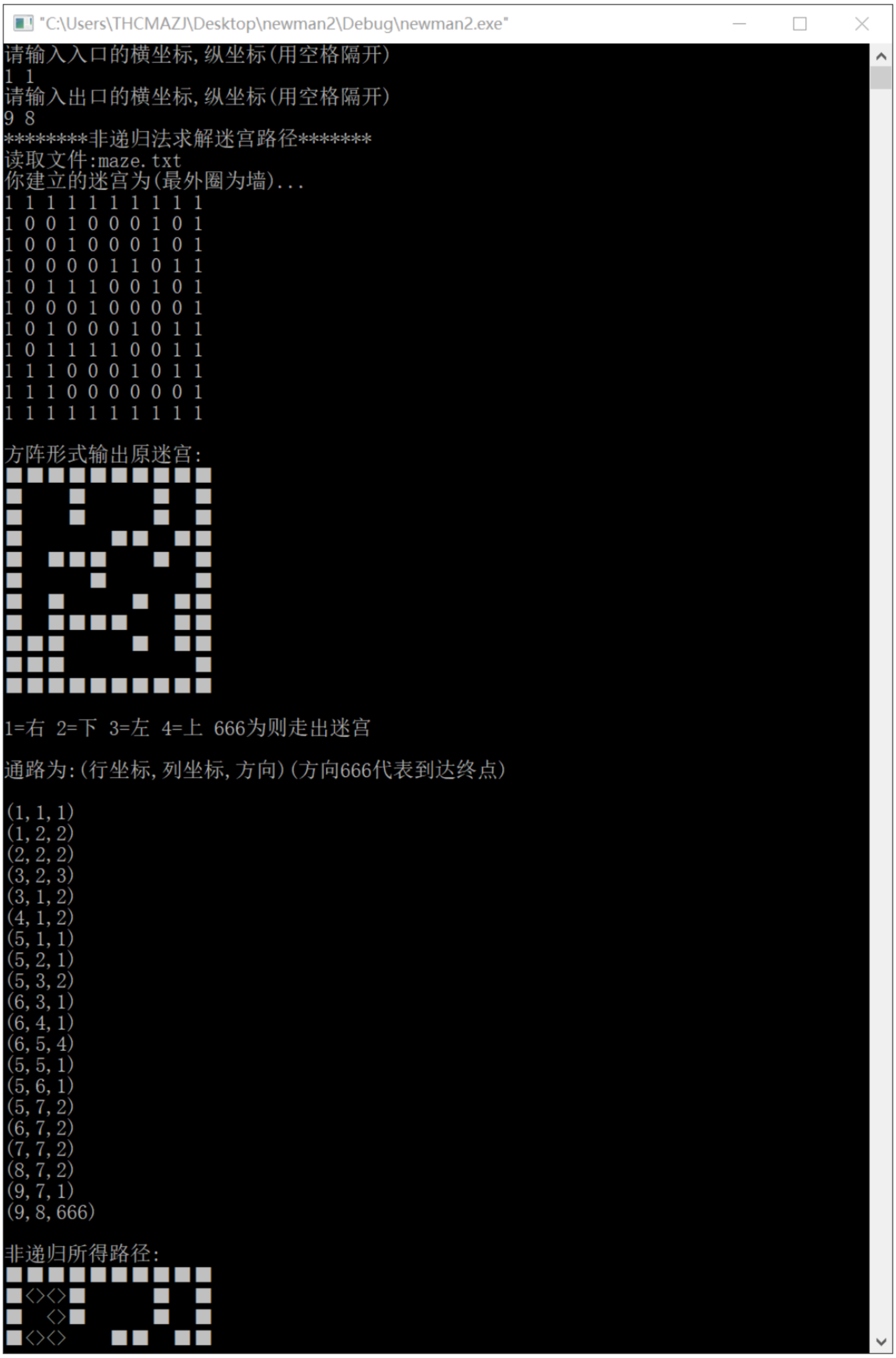
StdAfx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H__0E803D4B_FB41_4F97_8C09_439185F31E24__INCLUDED_)
```

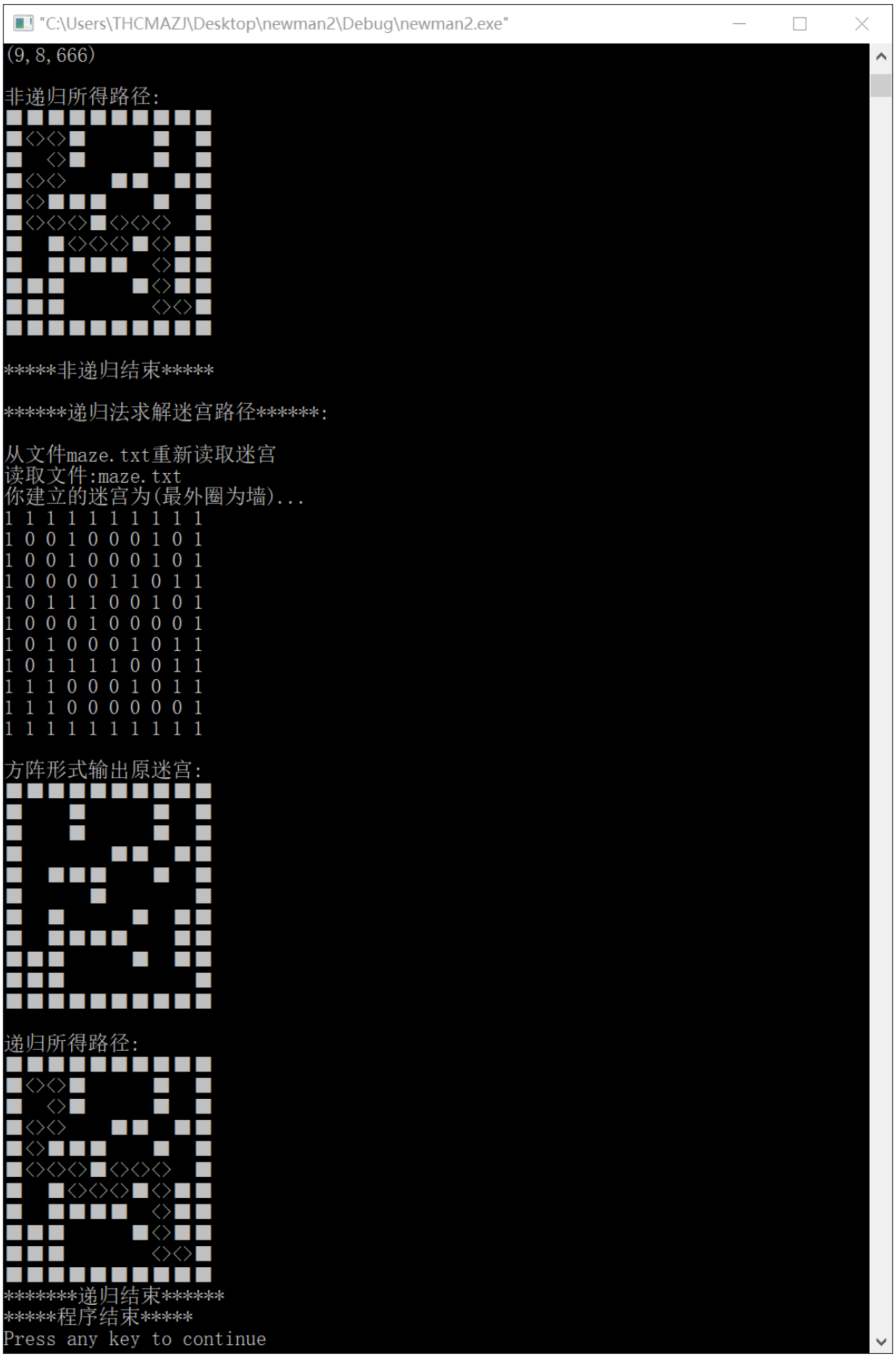
```
#define AFX_STDAFX_H__0E803D4B_FB41_4F97_8C09_439185F31E24__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN		// Exclude rarely-used stuff from Windows headers

#include <stdio.h>
#include <stdlib.h>

#define M 11
#define N 10

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__0E803D4B_FB41_4F97_8C09_439185F31E24__INCLUDED_)
```

二、Experimental test

The screenshot of the experiment is as follows:


"C:\Users\THCMAZJ\Desktop\newman2\Debug\newman2.exe"

```
请输入入口的横坐标,纵坐标(用空格隔开)
1 1
请输入出口的横坐标,纵坐标(用空格隔开)
9 8
********非递归法求解迷宫路径*******
读取文件:maze.txt
你建立的迷宫为(最外圈为墙)...
1 1 1 1 1 1 1 1 1 1
1 0 0 1 0 0 0 1 0 1
1 0 0 1 0 0 0 1 0 1
1 0 0 0 0 1 1 0 1 1
1 0 1 1 1 0 0 1 0 1
1 0 0 0 1 0 0 0 0 1
1 0 1 0 0 0 1 0 1 1
1 0 1 1 1 1 0 0 1 1
1 1 1 0 0 0 1 0 1 1
1 1 1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1

方阵形式输出原迷宫:
■■■■■■■■■■
■    ■      ■  ■
■    ■      ■  ■
■        ■■  ■■
■  ■■■    ■  ■
■      ■        ■
■  ■      ■  ■■
■  ■■■■    ■■
■■■      ■  ■■
■■■            ■
■■■■■■■■■■

1=右 2=下 3=左 4=上 666为则走出迷宫

通路为:(行坐标,列坐标,方向)(方向666代表到达终点)

(1,1,1)
(1,2,2)
(2,2,2)
(3,2,3)
(3,1,2)
(4,1,2)
(5,1,1)
(5,2,1)
(5,3,2)
(6,3,1)
(6,4,1)
(6,5,4)
(5,5,1)
(5,6,1)
(5,7,2)
(6,7,2)
(7,7,2)
(8,7,2)
(9,7,1)
(9,8,666)

非递归所得路径:
■■■■■■■■■■
■◇◇■      ■  ■
■  ◇■      ■  ■
■◇◇    ■■  ■■
```



"C:\Users\THCMAZJ\Desktop\newman2\Debug\newman2.exe"

```
(9,8,666)

非递归所得路径:
■■■■■■■■■■
■◇◇■      ■  ■
■  ◇■      ■  ■
■◇◇    ■■  ■■
■◇■■■    ■  ■
■◇◇◇■◇◇◇  ■
■  ■◇◇◇■◇■■
■  ■■■■  ◇■■
■■■      ■◇■■
■■■        ◇◇■
■■■■■■■■■■

*****非递归结束*****

******递归法求解迷宫路径******:

从文件maze.txt重新读取迷宫
读取文件:maze.txt
你建立的迷宫为(最外圈为墙)...
1 1 1 1 1 1 1 1 1 1
1 0 0 1 0 0 0 1 0 1
1 0 0 1 0 0 0 1 0 1
1 0 0 0 0 1 1 0 1 1
1 0 1 1 1 0 0 1 0 1
1 0 0 0 1 0 0 0 0 1
1 0 1 0 0 0 1 0 1 1
1 0 1 1 1 1 0 0 1 1
1 1 1 0 0 0 1 0 1 1
1 1 1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1

方阵形式输出原迷宫:
■■■■■■■■■■
■    ■      ■  ■
■    ■      ■  ■
■        ■■  ■■
■  ■■■    ■  ■
■      ■        ■
■  ■      ■  ■■
■  ■■■■    ■■
■■■      ■  ■■
■■■            ■
■■■■■■■■■■

递归所得路径:
■■■■■■■■■■
■◇◇■      ■  ■
■  ◇■      ■  ■
■◇◇    ■■  ■■
■◇■■■    ■  ■
■◇◇◇■◇◇◇  ■
■  ■◇◇◇■◇■■
■  ■■■■  ◇■■
■■■      ■◇■■
■■■        ◇◇■
■■■■■■■■■■
*******递归结束******
*****程序结束*****
Press any key to continue
```

Accessories: (maze for testing)

maze.txt:

```
0 0 1 0 0 0 1 0
0 0 1 0 0 0 1 0
0 0 0 0 1 1 0 1
0 1 1 1 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 1
0 1 1 1 1 0 0 1
1 1 0 0 0 1 0 1
1 1 0 0 0 0 0 0
```

```
0 0 1 0 0 0 1 0
0 0 1 0 0 0 1 0
0 0 0 0 1 1 0 1
0 1 1 1 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 1
0 1 1 1 1 0 0 1
1 1 0 0 0 1 0 1
1 1 0 0 0 0 0 0
```