

**计算机软件基础实习报告**

|  |  |
| --- | --- |
| 题 目： | 马踏棋盘 |
| 班 级： | 4171090701 |
| 姓 名: | 焦瑞鹏 |
| 学 号： | 417109070107 |
| 时 间： | 2019/01/07—2019/01/18 |
| 指导教师： | 张敬敏 张有华 |

信息工程学院

2019年1月编制

**马踏棋盘**

**目录**

[**一、需求分析 1**](#_Toc535390502)

[**二、概要设计 2**](#_Toc535390503)

[**三、详细设计 4**](#_Toc535390504)

[**四、调试分析 9**](#_Toc535390505)

[**五、用户手册 10**](#_Toc535390506)

[**六、测试结果 10**](#_Toc535390507)

[**七、附录 源程序文件名 12**](#_Toc535390508)

# 一、需求分析

1.问题描述

将马随机放在国际象棋的8X8棋盘 [1..8,1..8]的某个方格中，马按走棋规则（按照“日”字行走）进行移动。要求每个方格上只经过一次，走遍棋盘上全部64个方格。编制非递归程序，求出马的行走路线，并按求出的行走路线，将数字1，2，…，64依次填入8X8的方阵输出之。用<i,j>表示马的初始位置。

2.输入要求

由用户指定,可自行指定一个马的初始位置[定义i.j（i,j都为整数）一个初始的值。范围(0<i<8；0<j<8）]。

3.输出要求

马的行走路线，即根据马对棋盘上每个格子的访问顺序并输出该序号，即为马的形式路径。

4.测试数据

（1）输入： i=2，j=3.

输出：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 2 | 31 | 4 | 19 | 24 | 29 | 14 | 17 |
| 2 | 5 | 20 | 1 | 30 | 15 | 18 | 25 | 28 |
| 3 | 32 | 3 | 56 | 23 | 58 | 27 | 16 | 13 |
| 4 | 21 | 6 | 33 | 46 | 55 | 40 | 59 | 26 |
| 5 | 34 | 53 | 22 | 57 | 44 | 47 | 12 | 39 |
| 6 | 7 | 64 | 45 | 54 | 37 | 60 | 41 | 48 |
| 7 | 52 | 35 | 62 | 9 | 50 | 43 | 38 | 11 |
| 8 | 63 | 8 | 51 | 36 | 61 | 10 | 49 | 42 |

（2）输入： i= 9，j=8

输出：“您的输入有误，请重新输出”

# 二、概要设计

为了实现上述程序功能，应以数组和栈来表示棋盘和马儿的行驶过程。

用一个二维数组来存放棋盘，假设马儿的坐标为(x，y)，那么可供选择的下一个位置共有8种可能。我们所要做的，就是从0号位置开始，依次判断新的马儿位置是否可用，若不可用，则遍历下一个可能的1号位置，直到7号位置停止，如果没有可用位置，则进行回溯，如果回溯到了起始位置，则表示此路不通，即无法从该位置开始遍历整个棋盘。如果在遍历0-7号位置的过程中，发现有可用位置，则将该位置坐标赋予(x，y)。之后，利用递归，再次寻找马儿的新的跳跃位置。直到马儿跳了64次时停止，此时，马儿就已经将整个棋盘走过了。

1.栈的抽象函数类型：

ADT Stack {

数据对象：D={ai|aiElemSet，i=1，2，…，n，n≥0}

数据关系：R1={<ai-1,ai>|ai-1,aiD,i=2,…,n}

约定an端为栈顶，a1端为栈底。

基本操作：

Initstack(&s)

操作结果：构建一个空栈s

Destroystack(&s)

初始条件：栈s已经存在

操作结果：栈s被销毁

Clearstack(&s)

初始条件：栈s已经存在

操作结果：将s清为空栈

Stacklength(s)

初始条件：栈s已经存在

操作结果：返回s的元素个数。即栈的长度

Gettop（s，&e）

初始条件：栈s已经存在

操作结果：用e返回s的栈顶元素

Push（&s，e）

初始条件：栈s已经存在

操作结果：插入元素e为新的栈顶元素

Pop（&s，&e）

初始条件栈：s已经存在

操作结果：删除s的栈顶元素，并用e返回其值

Stacktracerse（s，visit（））

初始条件：

操作结果：从栈底到栈顶依次对s的每个元素调用函数visit（）。一旦visit（）失败，则操作失败

} ADT stack

2.程序中的功能模块

(1)主程序模块

Void main( ){

设置棋盘

输入初始坐标

根据走起规则输出马儿的行驶过程

}

(2)马踏棋盘单元模块

该模块包含三个部分，分别是对马初始位置，马尝试路径输出马的运动过程。

①起始坐标函数模块

②探寻路径函数模块

③输出路径函数模块

(3)栈的模块

各级模块关系：

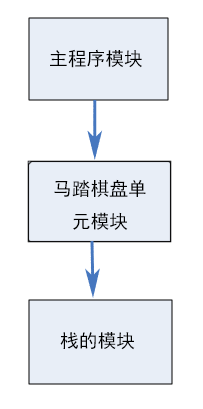


图2-1模块关系

# 三、详细设计

1.结点类型和指针类型

struct Stack{ //定义栈类型

int i; //行坐标

int j; //列坐标

int director; //存储方向

selemtype \*top //栈顶指针

}sqstacksize; /\*结点类型,指针类型\*/

2.棋盘定义

int board[8][8]; //定义棋盘

int Htry1[8]={1,-1,-2,2,2,1,-1,-2};

/\*存储马各个出口位置相对当前位置行下标的增量数组\*/

int Htry2[8]={2,-2,1,1,-1,-2,2,-1};

/\*存储马各个出口位置相对当前位置列下标的增量数组\*/

3.函数声明

void horseweizhi(int xi,int yi);

int changshi(int i,int j);

void display();

4.主程序

int main()

{

int i,j;

int x,y;

for(i=0;i<N;i++) //初始化棋盘

for(j=0;j<N;j++)

board[i][j]=0;

for(;;)

{

printf("请输入棋子起始坐标(1<=x<=8 and 1<=y<=8)\n");

printf("请输入行坐标 x = ");

scanf("%d",&x); //输入起始位置的横坐标

printf("请输入列坐标 y = ");

scanf("%d",&y); //输入起始位置的纵坐标

if(x>=1&&x<=8&&y>=1&&y<=8)break;

printf("Your input is worng!!!\n");

}

printf("从这里这个位置开始 %d :\n\n", 8\*(x-1)+y);

horseweizhi(x-1,y-1); //调用起始坐标函数

}

5.起始坐标模块

void horseweizhi(int xi,int yi)

{

int x,y;

top++;

stack[top].i=xi;

stack[top].j=yi;

stack[top].director=-1;

board[xi][yi]=top+1;

x=stack[top].i;

y=stack[top].j;

if(changshi(x,y))

display();

else

printf("无解");

}

6.路径模块

int changshi(int i,int j)

{

int find,director,number,min;

int i1,j1,h,k,s;

int a[8],b1[8],b2[8],d[8];

while(top>-1)

{

for(h=0;h<8;h++)

{

number=0;

i=stack[top].i+Htry1[h];

j=stack[top].j+Htry2[h];

b1[h]=i;

b2[h]=j;

if(board[i][j]==0&&i>=0&&i<8&&j>=0&&j<8)

{

for(k=0;k<8;k++)

{

i1=b1[h]+Htry1[k];

j1=b2[h]+Htry2[k];

if(board[i1][j1]==0&&i1>=0&&i1<8&&j1>=0&&j1<8)

number++;

}

a[h]=number;

}

}

for(h=0;h<8;h++)

{

min=9;

for(k=0;k<8;k++)

if(min>a[k])

{

min=a[k];

d[h]=k;

s=k;

}

a[s]=9;

}

director=stack[top].director;

if(top>=63)

return (1);

find=0;

for(h=director+1;h<8;h++)

{

i=stack[top].i+Htry1[d[h]];

j=stack[top].j+Htry2[d[h]];

if(board[i][j]==0&&i>=0&&i<8&&j>=0&&j<8)

{

find=1;

break;

}

}

if(find==1)

{

stack[top].director=director;

top++;

stack[top].i=i;

stack[top].j=j;

stack[top].director=-1;

board[i][j]=top+1;

}

else

{ board[stack[top].i][stack[top].j]=0;

top--;}

}

return (0);

}

7.路径输出模块

void display()

{ int i,j;

for(i=0;i<N;i++)

{

for(j=0;j<N;j++)

printf("\t%d ",board[i][j]);

printf("\n\n");

}

printf("\n");

}

8.函数调用关系图

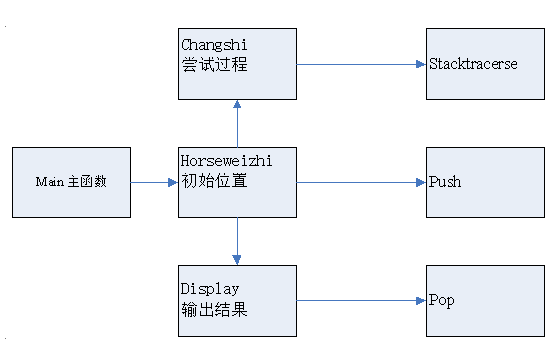


图3-1函数调用关系图

# 四、调试分析

遇到的问题如下：

1.指针位置、地址混乱，导致报错。将栈的结构和操作熟悉了几次以后解决了这个问题。

2.正确输入坐标后，程序不显示行驶路径。最后发现在马的行驶路径中应该定义一个变量来进行计数。

3.程序最后调用主函数时出现以下情况

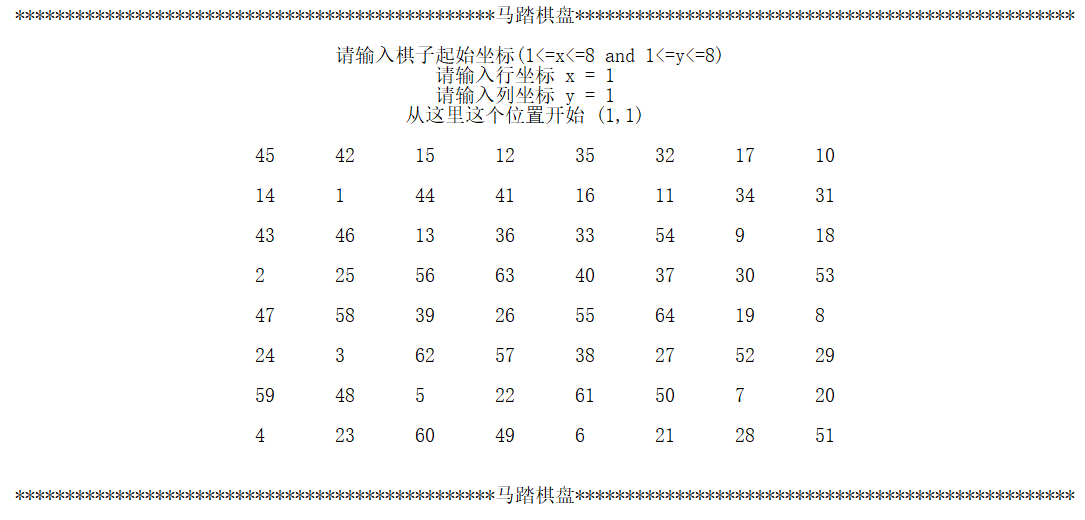


图4-1 运行结果错误图

虽然输入的是（1,1）但是起始位置是从（2,2）开始。后来是发现在调用马位置函数的时候应该在输入的坐标上都-1，因为定义的是从（0,0）开始而不是（1,1）。



图4-2 代码修正

4.算法的时空分析

l)从数据结构的角度看，栈是线性表，其特殊性在于栈的基本操作是线性表操作的子集，所以何种操作的时间复杂度比较合理。

2)基于栈实现各种运算和操作的时间复杂度分析如下：定义马儿初始位置的函数中void horseweizhi( )函数下只需要将i(横坐标输入n次)和j（纵坐标输入n次）输入栈中所以此函数的时间复杂度为O(n^2)。

3)马儿在探索路径int changshi( )函数下利用Stacktracerse遍历栈中的元素，读取i和j（横纵坐标）n次；2层嵌套循环，每层循环n次故时间复杂度为0(n^2)。

输出马儿行驶路径void display()函数下元素读取n次，2层嵌套循环，每层循环n次，所以此函数的时间复杂度为0(n^2);其中 n为棋盘的行(列)数。

5. 本作业采用数据抽象的程序设计方法，将程序划分为四个层次结构；主程序模块，起始坐标函数模块，探寻路径函数模块，输出路径函数模块。使得设计时思路清晰，实现时调试顺利。各模块具有较好的可重用性、确实得到了一次良好的程序设计训练。

# 五、用户手册

（1）本程序的运行环境为仿真 DOS 系统，执行文件为 main.exe

（2）运行程序以后根据要求输入数据

（3）正确输出为8×8数组矩阵，错误输入提示用户再次输入

（4）根据要求结束程序

# 六、测试结果

运行后首先进入欢迎界面



图6-1 欢迎界面

用户根据提示进行输入，当输入x，y的数据时，输出起始坐标位置

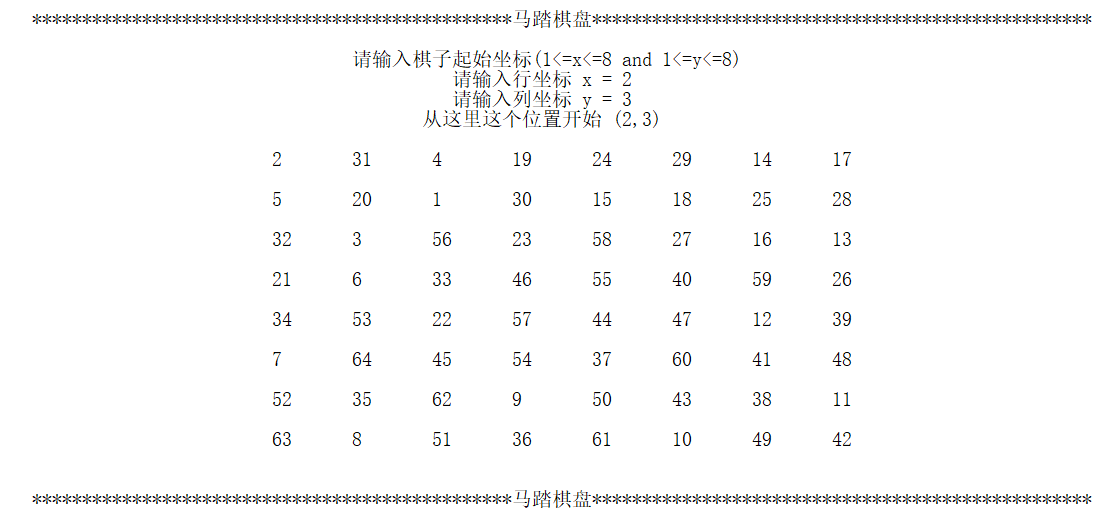


图6-2 正确马路径输出图

当用户输入错误的x，y数据，输入输出错误显示

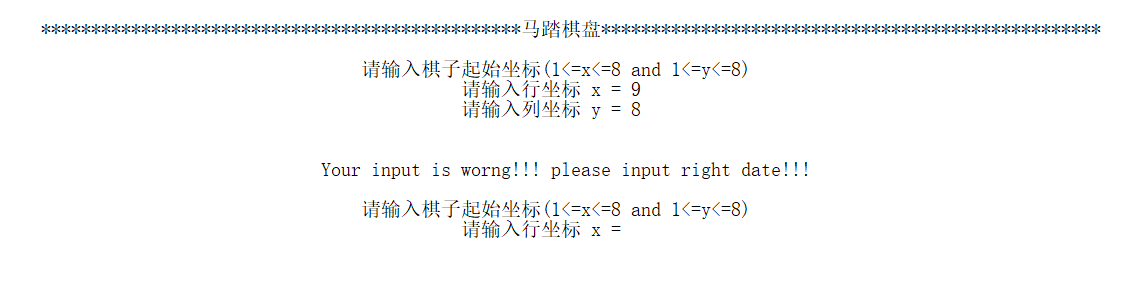


图6-3错误马路径输出图

# 七、附录 源程序文件名

[**main.cpp**](main.cpp)