学　　号：	0120710340617

武漢理工大學

课 程 设 计

课程名称：数据结构

设计题目：马踏棋盘的程序设计

专　　业：计算机科学与技术专业

班　　级：0706

姓　　名：孙　禹

指导教师：刘　伟

2009 年 07 月 02 日

课程设计任务书

学生姓名：孙　禹　　专业班级：07 06 班

指导教师：刘　伟　　工作单位：计算机科学系

题　目：　马踏棋盘的程序设计

初始条件：

设计一个国际象棋的马踏遍棋盘的演示程序。

将马随机放在国际象棋的 8×8 棋盘 Board[8][8]的某个方格中，马按走棋规则（见题集 p98）进行移动。要求每个方格只进入一次，走边棋盘上全部 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 1,2,3,…,64 依次填入一个 8×8 的方阵，输出之。

测试用例见题集 P_{98}。

要求完成的主要任务：（包括课程设计工作量及其技术要求，以及说明书撰写等具体要求）

课程设计报告按学校规定格式用 A4 纸打印（书写），并应包含如下内容：

1、 问题描述

简述题目要解决的问题是什么。

2、 设计

存储结构设计、主要算法设计（用类 C 语言或用框图描述）、测试用例设计；

3、 调试报告

调试过程中遇到的问题是如何解决的；对设计和编码的讨论和分析。

4、 经验和体会（包括对算法改进的设想）

5、 附源程序清单和运行结果。源程序要加注释。如果题目规定了测试数据，则运行结果要包含这些测试数据和运行输出，

6、 设计报告、程序不得相互抄袭和拷贝；若有雷同，则所有雷同者成绩均为 0 分。

时间安排：

1、第 20 周（6 月 29 日至 7 月 3 日）完成。

2、7 月 3 日 8：00 到计算中心检查程序、交课程设计报告、源程序（CD 盘）。

指导教师签名：　　年　　月　　日

系主任（或责任教师）签名：　　年　　月　　日

一、问题描述

设计一个国际象棋的马踏棋盘的演示程序。

基本要求：将马随机放在国际象棋 8×8 的棋盘 Board[8][8]的某个方格中，马按走棋规则进行移动。要求每个方格只进入一次，走遍棋盘全部的 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 1，2，3，…,64 一次填入一个 8×8 的方阵输出之。

测试数据：可自行指定一个马的初始位置（i,j），0≤i，j≤7。

二、实验目的

1、对数据结构基本理论和存储结构及算法设计有更加深入的理解；

2、了解栈的特性，以便在实际问题背景下灵活运用他们；

3、提高在实际设计操作中系统分析、结构确定、算法选择、数学建模和信息加工的能力。

三、设计过程

第 1 步，实现提示

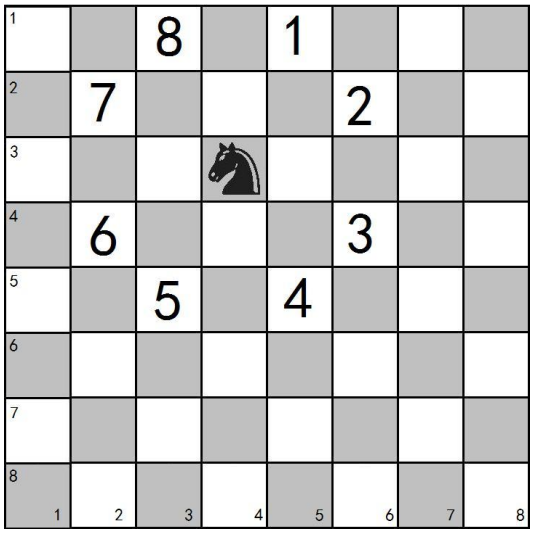
一般说来，当马位于位置（i,j）时，可以走到下列 8 个位置之一：

(i－2，j＋1)，(i－1，j＋2)

(i＋1，j＋2)，(i＋2，j＋1)

(i＋2，j－1)，(i＋1，j－2)

(i－1，j－2)，(i－2，j－1)



（图-1）

但是，如果（i，j）靠近棋盘的边缘，上述有些位置可能超出棋盘范围，成为不允许的位置。8 个可能位置可以用一维数组 HTry1[0…7]和 HTry2[0…7]来表示：

	0	1	2	3	4	5	6	7
HTry1	－2	－1	1	2	2	1	－1	－2
HTry2	1	2	2	1	－1	－2	－2	－1

（表-2）

位于(i,j)的马可以走到新位置是在棋盘范围内的(i+HTry1[h],j+HTry2[h]),其中 h=0，1，…，7。

第 2 步，需求分析

（1)、输入的形式和输入值的范围：输入马的初始行坐标 X 和列坐标 Y，X 和 Y 的范围都是 [1,8]。

（2)、输出的形式：

①.以数组下标形式输入，i 表示行标，j 表示列标。

②.以棋盘形式输出，每一格打印马走的步数，这种方式比较直观。

（3)、程序所能达到的功能：让马从任一起点出发都能够历遍整个 8×8 的棋盘。

（4)、测试数据，包括正确的输入及输出结果和含有错误的输入及其输出结果。数据可以任定，只要 1≤x，y≤8 就可以了。

正确的输出结果为一个 2 维数组，每个元素的值表示马行走的第几步。若输入有错，程序会显示“输入有误!请您重新输入.....”并且要求用户重新输入数据，直至输入正确为止。

第 3 步，算法设计思想

①、输入马所在初始位置的坐标值，考虑到用户输入习惯，此处取 **1≤x，y≤8**；

②、将输入的初始位置进栈；

③、设置一个 **while** 循环，循环条件为 **count<64**；

④、取出栈顶元素；

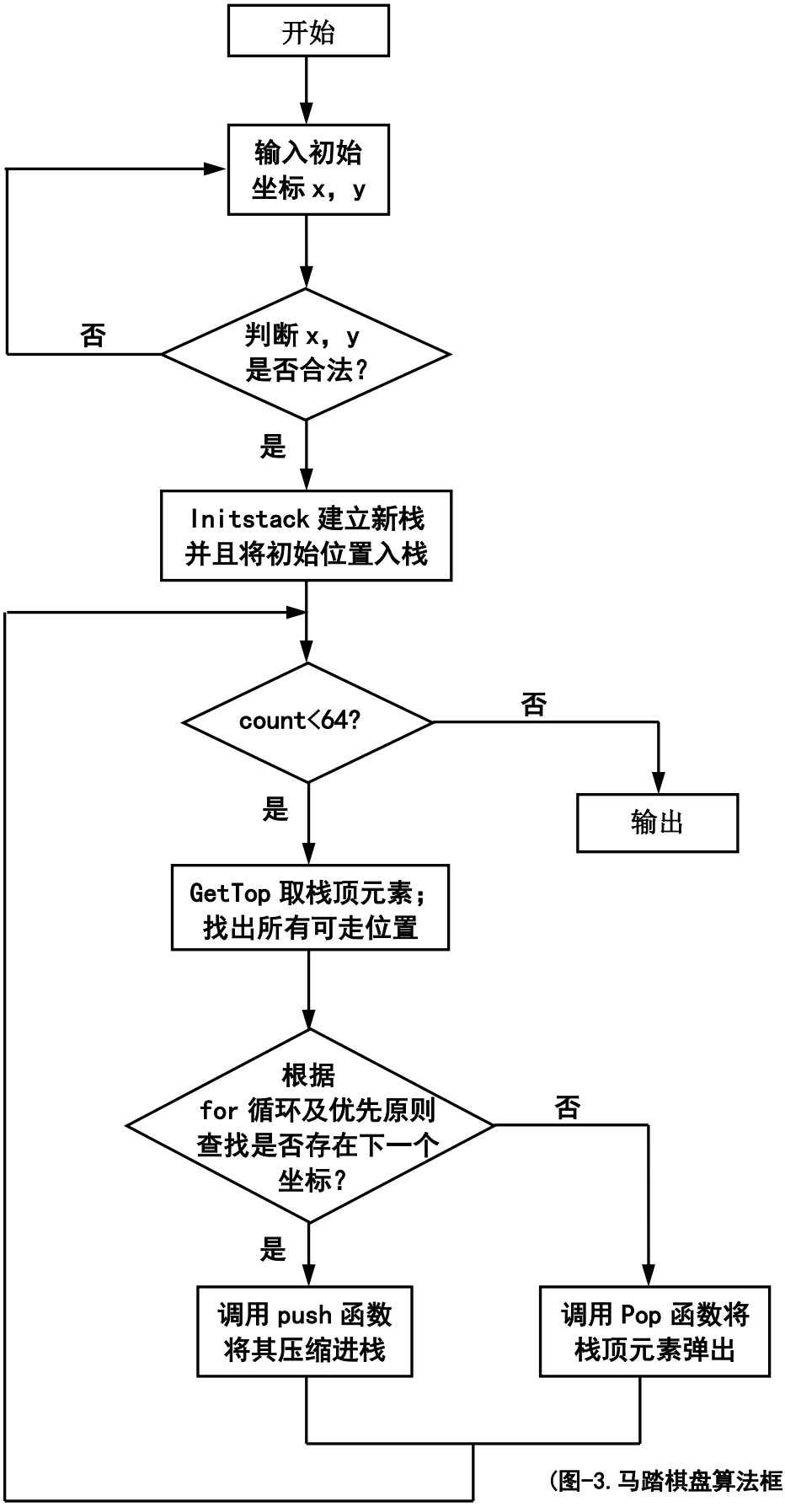
⑤、定义 **flag** 标志变量的值；

⑥、按照 **SetRound** 函数逆时针顺序优先原则，找栈顶元素周围未被占用的新位置。若存在该位置，则令 **order** 的值等于该新位置的坐标，并入栈；否则弹出栈顶元素；

⑦、再次回到第③步 **while** 循环进行判断；

⑧、输出一个 **8×8** 的方阵，所示数字即为相应步骤。

第 4 步，算法框图（如图-3 所示）



（图-3. 马踏棋盘算法框图）

第 5 步，存储结构设计

（1）、位置的存储表式方式

```
struct  Point
{   int  x;          //所在位置的横坐标值
    int  y;          //所在位置的纵坐标值
    int  from;       //前一个位置的最优点的序号
};
```

（2）、栈的存储方式

```
#define  STACKSIZE    70            //存储空间的初始分配
#define  STACKINCREASE    10     //存储空间分配增量
struct  Stack
{  Point  *top;     //在栈构造之前和销毁之后，base 的值为 NULL
    Point  *base;   //栈顶指针
   int  length;      //当前已分配的存储空间
};
```

第 6 步，设计功能的分析与实现

（1）设定栈的抽象数据类型定义：

ADT Stack{

数据对象：$D=\{a_i|a_i \in ElemSet，i=1，2，\cdots，n，n \geq 0\}$

数据关系:$R1=\{<a_{i-1},a_i>|a_{i-1},a_i \in D，i=1，2，\cdots，n\}$

基本操作:(这里仅列举本题中使用的操作)

InitStack(&S)

操作结果:构建一个空栈 S。

StackEmpty(S)

操作结果:若栈 S 为空栈，则返回 TURE,否则 FALSE。

GetTop(S, &e)

操作结果：用 e 返回 S 的栈顶元素。

Push(&S,e)

操作结果：在栈顶插入新的元素 e。

Pop（&S,&e）

操作结果：删除 S 的栈顶元素，并用 e 返回其之。

}ADT Stack

（2）主要函数及算法说明。

①.函数： `void SetRound(Point cur)`

参数：Point;

功能：找出当前位置下一步的八个位置，将其赋给 g_round[8];

算法描述：接受参数传来的值，按逆时针次序加上

g_round[i].x ={-2,-1, 1, 2,2,1,-1,-2};

g_round[i].y ={1,2,2,1,-1,-2,-2,-1};

再根据 GetRound 函数来判断是否合法（x[0~7]，y[0~7]）；若合法返回 TRUE；否则返回 FALSE。

②.函数： `bool GetRound(int i,Point &pt)`

参数：int ,Point;

功能：将所在位置周围所有八个位置坐标赋予指针变量，并判断其合理性;

算法描述：用赋值语句

`pt.x = g_round[i-1].x;pt.y = g_round[i-1].y;`

分别将现在周围 8 个指定位置赋予指针变量 pt，并用 `if(pt.x<0 || pt.y<0 || pt.x>7 || pt.y>7)` 条件语句判断其合理性，合理返回返回 TRUE；否则返回 FALSE。

③.语句：

```
for(int i=cur.from+1;i<=8;i++)
{
        if(GetRound(i,next) && order[next.x][next.y]==0)
        {                    语句… …
        }
}
```

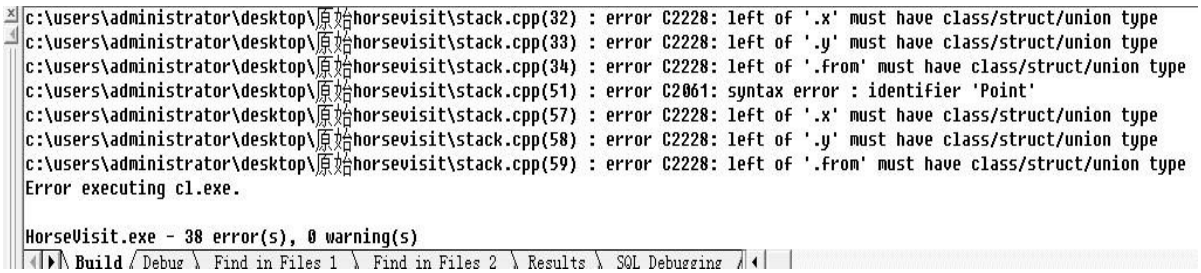
功能描述：按照逆时针的优先规则，选出下一个可用的新位置。通过 if 语句判断所选出的下一个位置是否可用。若可用，则使其进栈；否则运行④。

④. 语句：

```
if(!flag)
 {                    语句… …
          while(j<p && GetDeep(horseVisit)>1)
          {                    语句… …
          }
 }
```

功能描述：如果当前位置周围不存在任何新路径，则根据 while 循环进行退栈，直至退到存在有最佳位置的坐标。但必须保证栈不为空。所以此出栈中设定了 int GetDeep(Stack S)的基本操作，就是防止空栈是还在继续作弹出操作的情况发生。

四、调试报告

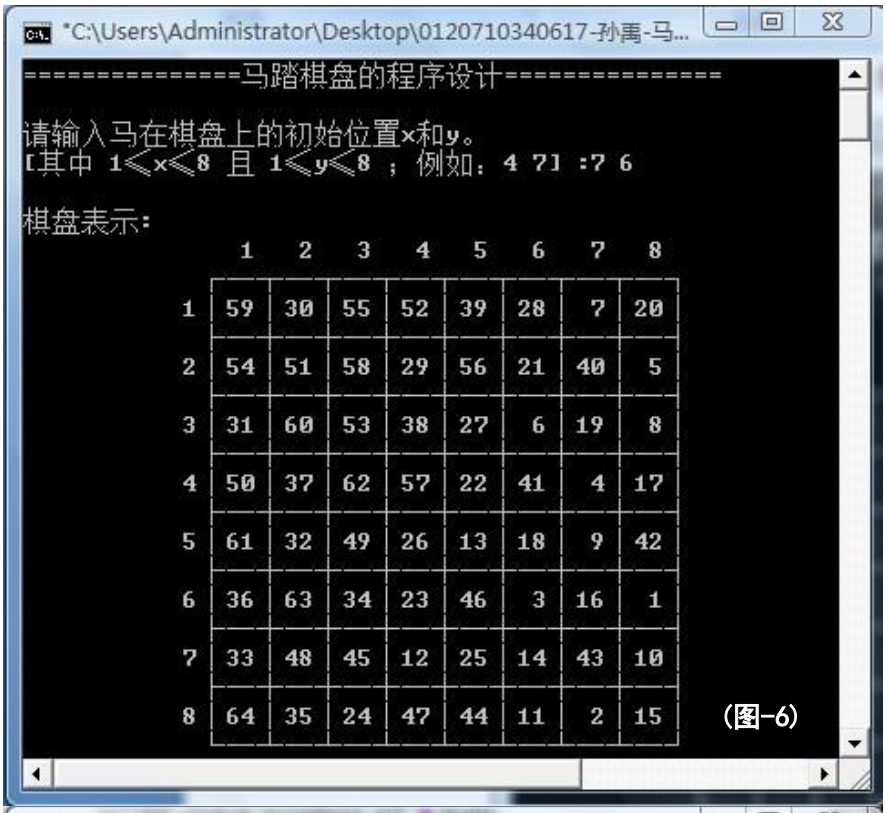


（图-4）

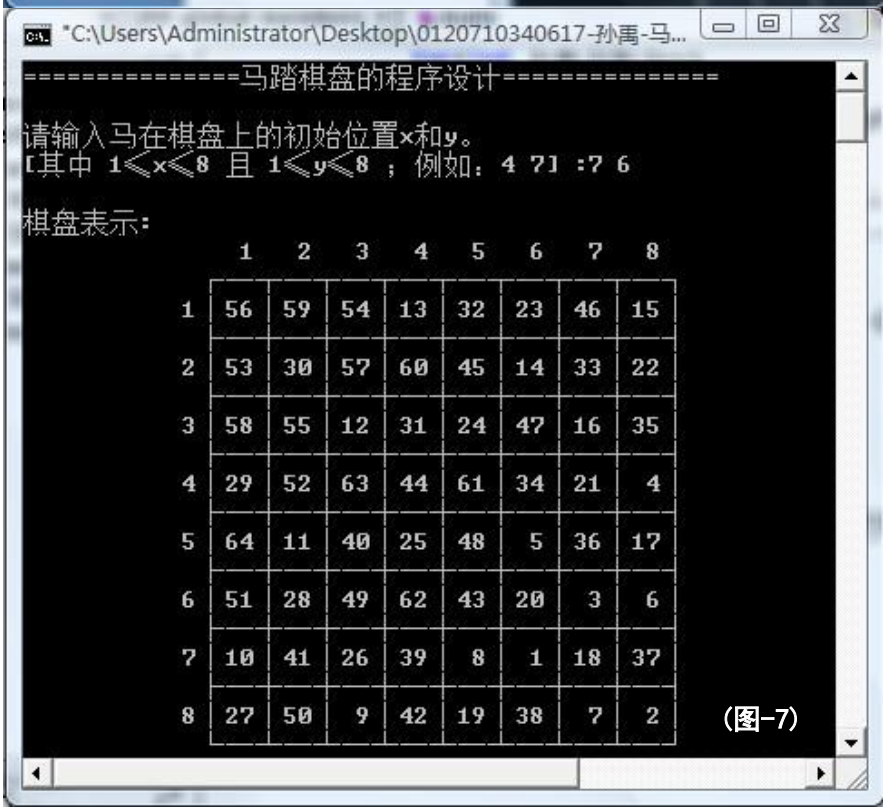
- **问 题 一：**刚开始写完程序后初次运行发现有较多错误，如图-4 所示。
- **原　　因：**大多数为语法和字符错误，由于粗心大意造成的。
- **解决方法：**根据提示通过调试一一解决这些简单的错误。直到“0 error（s），0 warning（s）”。

- **问 题 二：**成功运行后发现当输入某些初始坐标时，输出结果初始位置的数值不为“1”。（如图-6 所示）

- **原　　因：**检查发现这是由于进入 while(j<5)循环时，进行退栈，直至退到存在有最佳位置的坐标。但栈为空时，电脑对栈是还在继续进行弹出操作。
- **解决方法：在**栈中设定了 int GetDeep(Stack S)的基本操作。确保栈不弹出过头。改正后正确输出。（如图-7 所示）



(图-6)



(图-7)

- **问 题 三：**成功运行后发现，对于不同的初始坐标。电脑输出结果的速度也不尽相同；
- **原　　因：**与同学讨论发现语句 **while(j<4 && GetDeep(horseVisit)>1) {Pop(horseVisit,cur); order[cur.x][cur.y] = 0; count--; j++; }**中执行的次数会对程序运行的速度成影响，且对弹出也有影响；
- **解决方法：**将“j<4 改成 j< 5”时对于整个程序的速度是最佳的；

- **问 题 四：**解决问题三后发现输入坐标（**3**，**7**）时无法显示结果，但是发现当 **j<4** 时可以正常显示；
- **原　　因：**分析可能与弹出过度有关。
- **解决方法：**利用一个 if 语句判定其坐标为（3，7）时用 j<4，其他情况用 j< 5，语句如下：

```
if(begin.x==2 && begin.y==6) p=4;
else p=5;
while(j<p && GetDeep(horseVisit)>1)
{                    语句… …
}
```

- **问 题 五：**选取位置时超出边界；
- **原　　因：**循环次数大；
- **解决方法：**我已经按马的下一位置最难的先走，再走难度次之的位置。 函数的功能尽量单一，调用函数时要注意是否修改了其他参数的值。一定要静下心去做。调试也很重要，从中能知道错在哪里。用 **C++**一定要学会这功能。

五、用户手册

（1）、本程序运行环境为 DOS 操作系统，运行工作空间 HorseVisit.dsw；

（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、经验和体会

本次数据结构被分配到的是马踏棋盘的程序设计，刚看题目觉得有点没有思绪，但是通过课本提示及相关资料的查阅。编程的思路就愈加清晰。经过我一天的奋战，终于把程序编出来了。通过这次课程设计，我也学会了很多东西。

首先，我又再次感受到了 C++功能的强大。这个学期的数据结构的学习不仅让我们学到了很多东西，在编程上有很大的提高；同时也是对我们以前所学的 C++的复习与巩固。每次的编程都是对以前各种知识点的升华，如果说以前的编程仅仅是按照课本的要求进行的话，拿这个课程设计相对可要难得多。它让我们知道很多东西我们都要系统的、理论的将其联系到一起去。这样才能让我们的程序更加的强大。

其次谈谈我对本次课程设计的一些感受。我认为我们这学期所学的数据结构给我们很大的帮助。我的题目主要是考察栈的特性与实现，以便在实际问题背景下灵活运用他们。本次程序主要用到栈的建立、销毁、进栈、出栈、取栈顶、求栈长等等基本操作。通过这些基本操作来实现坐标的存放。实验调试的过程中，由于粗心大意，很多错误可以避免，这也告诉我无论做什么都要细心，编程也是一个的道理。

再次，我认为世界上没有完美的程序。任何一个程序都需要不断地完善，不断的更改。所以与同学之间的讨论与交流也尤为重要。这次程序要感谢一些同学在我调试过程中给与的帮助与建议。比如说在输出格式方面的问题，原来打算只是按行列打印一个数组，但是有同学提醒说可以让其更美观，通过输出语句打印格子。我通过上网查找找到了一种较好的输出方式，运用到了我的程序之中。

最后，还要感谢我们实验室指导老师的细心指导、热心帮助。路漫漫其修远兮，虽然本学期数据结构课程结束了，但仍觉得自己还有很多东西要学，我会在自己在以后的学习生活中不断努力、不断提高，争取更大的进步。

七、参考文献

[1] 闵联营，何克右.《C++程序设计教程》.武汉理工大学出版社；

[2] 严蔚敏，吴伟民.《数据结构（C 语言版）》.清华大学出版社；

[3] 张文祥， 肖四友.《C++实验与案例分析》. 科学出版社；

[4] 谭浩强.《C++程序设计》. 清华大学出版社；

[5] 朱战立.《数据结构(C++语言描述)》(第二版本).高等教育出版社。

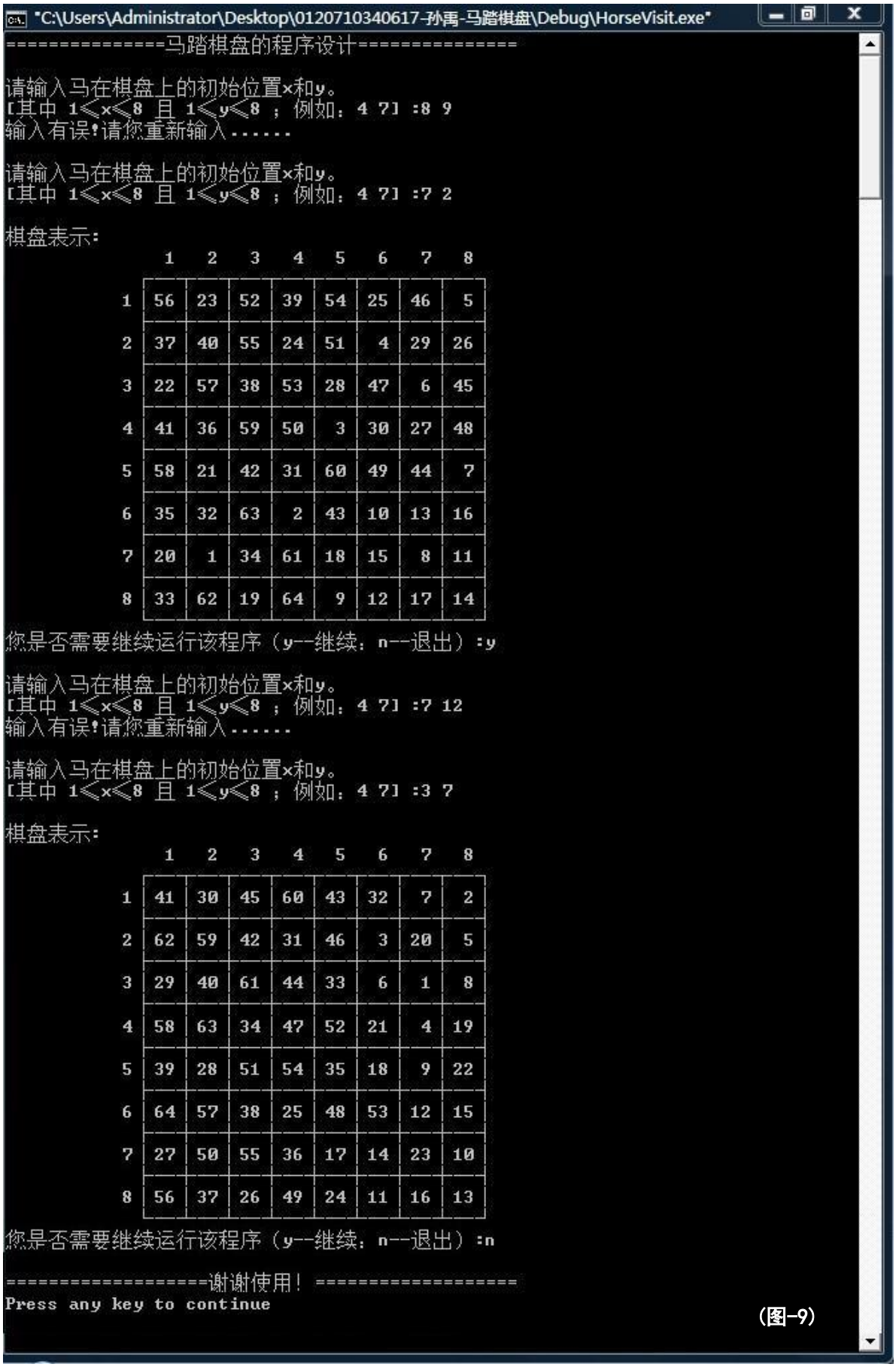
八、附运行结果和源程序清单。

附件 1：运行结果说明

输入任意两组正确的坐标：

（图-8）

输入任意坐标检测，然后退出程序：：



（图-9）

附件 2：程序清单

```
/* Header  Files / Stack.h */
#include <malloc.h>
#ifndef          STACK_H
#define          STACK_H
//-------------位置的存储表式方式
struct Point
{
    int x;
    int y;
    int from;
};
//--------------栈的存储方式--------------
#define STACKSIZE          70
#define STACKINCREASE     10
struct Stack
{
    Point *top;
    Point *base;
    int length;
};

//--------------操作集函数-----------------
bool Initstack(Stack &s);
bool Push(Stack &s,Point e);
bool Pop(Stack &s,Point &e);
void DestroyStack(Stack &s);
bool StackEmpty(Stack S);
bool GetTop(Stack S,Point &e);
int GetDeep(Stack S);

#endif

/*Source  Files / Stack.cpp*/
#include"stack.h"
bool Initstack(Stack &s)
{//构造一个空栈 S
    s.base = (Point*)malloc(STACKSIZE*sizeof(Point));
    if(!s.base)return false;
    s.length = STACKSIZE;
    s.top = s.base;
    return true;
}
bool Push(Stack &s,Point e)
{//插入元素 e 为新的栈顶元素
    if(s.top - s.base >= s.length)
    {
        s.base = (Point*)realloc(s.base,
            (s.length+STACKINCREASE)*sizeof(Point));
        if(!s.base)return false;
        s.length +=STACKINCREASE;
        s.top = s.base + s.length;
    }
    (*s.top).x = e.x;
    (*s.top).y = e.y;
    (*s.top).from = e.from;
    s.top ++;
    return true;
}

bool Pop(Stack &s,Point &e)
{ //若栈不为空，则删除 S 的栈顶元素
    if(s.top == s.base)return false;
    e.x = (*--s.top).x;
    e.y = (*s.top).y;
    e.from = (*s.top).from;
    return true;
}

void DestroyStack(Stack &s)
{ //销毁栈 S，S 不在存在
    free(s.base);
}

bool StackEmpty(Stack S)
{//若栈 S 为空栈，则返回 ture；否则返回 false
    if(S.base == S.top)
        return true;
    else
        return false;
}

bool GetTop(Stack S,Point &e)
{//若栈不为空，则取栈顶元素
    if(StackEmpty(S))
        return false;
    else
```

```
    {
        e.x = (*(S.top-1)).x;
        e.y = (*(S.top-1)).y;
        e.from = (*(S.top-1)).from;
        return true;
    }
}

int GetDeep(Stack S)
{   //取栈的深度
    return (S.top-S.base);
}

/*Source  Files/HorseVisit.cpp */
#include<iostream.h>
#include"stack.h"
#include <iomanip.h>

Point g_round[8] = {0,0,0};
void SetRound(Point cur)
{//查找所在位置的所有可走位置的坐标，将其赋给g_round[8]
    Point round[] =
    {
        cur.x-2,cur.y+1,0,cur.x-1,cur.y+2,0,
        cur.x+1,cur.y+2,0,cur.x+2,cur.y+1,0,
        cur.x+2,cur.y-1,0,cur.x+1,cur.y-2,0,
        cur.x-1,cur.y-2,0,cur.x-2,cur.y-1,0
    };
    for(int i=0;i<8;i++)
    {
        g_round[i].x = round[i].x;
        g_round[i].y = round[i].y;
    }
}

bool GetRound(int i,Point &pt )
{//将所在位置周围所有八个位置坐标赋予指针变量 pt，并判断其合理性
    pt.x = g_round[i-1].x;
    pt.y = g_round[i-1].y;
    if(pt.x<0 || pt.y<0 || pt.x>7 || pt.y>7)//判断其合理性
        return false;
    else
        return true;
}
void main()
{
    int s=1;
    char yn;
    cout<<"===============马踏棋盘的程序设计===============<<endl;
    cout<<endl;
    while(s)
    {
        int order[8][8] = {0};      //初始化
        int count = 0; //计数器，记录的是第几步棋

        Point begin;
        cout<<"请输入马在棋盘上的初始位置 x 和 y。"<<endl;
        cout<<"[其中 1≤x≤8 且 1≤y≤8 ；例如：4 7] :";
        cin>>begin.x>>begin.y;
        cout<<endl;
        begin.from = 0;
while(begin.x>8 || begin.x<1 || begin.y>8 || begin.y<1)
        {
            cout<<" 输 入 有 误 ！请 您 重 新 输 入....."<<endl;
            cout<<endl;
            cout<<"请输入马在棋盘上的初始位置 x 和 y。"<<endl;
            cout<<"[其中 1≤x≤8 且 1≤y≤8 ；例如：4 7] :";
            cin>>begin.x>>begin.y;
            cout<<endl;
        }
        begin.x--;      //实际下标是 0~7，
        begin.y--;

        Stack horseVisit;
        Point cur,next;
        Initstack(horseVisit);
        Push(horseVisit,begin);      //首位置进栈
        order[begin.x][begin.y] = ++count; //计数器+1
    while(count<64)
        {       //其余 63 步棋的走法
```

```
                GetTop(horseVisit,cur);
                SetRound(cur);

                bool flag = false;
                for(int i=cur.from+1;i<=8;i++)
                {   //按照逆时针的优先规则，选出下一个可用的新位置
      if(GetRound(i,next)&&order[next.x][next.y]==0)
                    {//可用位置未曾使用，则进栈，计数器加 1
                flag = true;
                 order[next.x][next.y] = ++count;

                Pop(horseVisit,cur);
                 cur.from = i;
                 Push(horseVisit,cur);

                next.from = 0;
                Push(horseVisit,next);
                break;
                    }
                }
                if(!flag)
                {   //如果当前位置周围没有路径，则退栈，直至退到存在有最佳位置的坐标
                int j=0,p;
                if(begin.x==2 && begin.y==6) p=4;
                else p=5;
                while(j<p && GetDeep(horseVisit)>1)
                    {
                        Pop(horseVisit,cur);
                        order[cur.x][cur.y] = 0;
                        count--;
                        j++;
                    }
                }
            }
            DestroyStack(horseVisit); //完成后销毁栈
            cout<<"棋盘表示:"<<endl;
            cout<<"                    1    2    3    4    5    6    7    8"<<endl;
            cout<<"                  ┌────┬────┬────┬────┬────┬────┬────┬────┐ "<<endl;
            for(int i=0;i<8;i++)
            {       //输出 order 数组，数组上数值为路径
                cout<<setw(12)<<i+1<<setw(2)<<"│";
                for(int j=0;j<8;j++)
                {

      cout<<resetiosflags(ios::right)<<setw(2)<<order[i][j]<<"│";
                }
                cout<<endl;
                if(i==7)cout<<"                  └────┴────┴────┴────┴────┴────┴────┴────┘ "<<endl;
                else  cout<<"                  ├────┼────┼────┼────┼────┼────┼────┼────┤ "<<endl;
            }
            cout<<endl;
            cout<<"您是否需要继续运行该程序（y--继续：n--退出）:";
            cin>>yn;
            cout<<endl;
            if(yn=='y'||yn=='Y')    s=1;
            else
            {
                s=0;
                cout<<"==================== 谢 谢 使用！====================="<<endl;
            }
        }
}
```