CHAPTER 3

第3章 程序设计案例与实训

本章提供一些综合设计案例，启发学生以C语言为工具解决实际问题，学会从问题分析到代码实现的开发过程，达到学以致用的目的。通过迷宫、骑士巡游程序，强调经典算法的重要性；通过五子棋、贪吃蛇案例增加C语言学习的趣味性；通过中文分词、图形操作程序，体会用C语言解决复杂问题的关键思路。

3.1 五子棋游戏程序

1. 问题描述

五子棋是一种双人棋类游戏，在以横线、竖线互相交叉（一般各为15条）的方形平面棋盘中，黑白两种棋子先后沿横线、竖线排列，在平面棋盘横线、竖线、斜线（无实线连接）上形成连续的同色五个棋子，则该方赢，如图3-1所示。

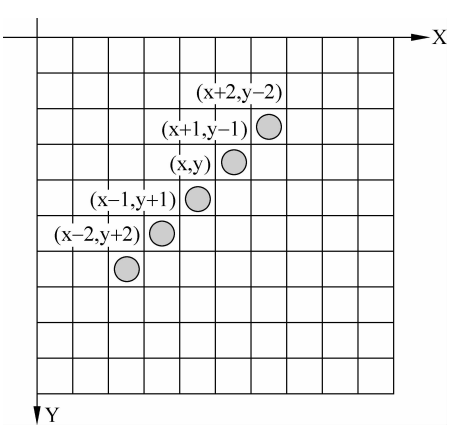

图3-1 五子棋坐标位置示意图

2. 算法思想

定义一个 N * N 的数组 ChessBoard[N][N]表示棋盘，数组元素 ChessBoard [x][y]的数值为 0 表示没有放棋子；为 1 表示玩家 1 放黑棋；为 2 表示玩家 2 放白棋。定义一个数组 ChessStep[N * N][2]表示两个玩家先后落子的位置坐标，第 i 步落子的为玩家 i%2+1，两人交替落子，按照 i%2 即可判定落子的玩家。

采用键盘上下左右箭头键在棋盘上移动位置，用空格键与 Enter 键确定落子，用退格键表示悔棋操作。

玩家代号为 v(v=1,2)，根据行、列、斜线几个方向是否构成 5 个连续相同的数值判定玩家输赢。即假设当前坐标位置为[x,y]，i=j=1，判定条件如下。

(1) 同一行上连续 5 个棋子。

```
while (ChessBoard[x][y-i] == v) i++;
while (ChessBoard[x][y+j] == v ) j++;
if (i+j>=6)
```

(2) 同一列上连续 5 个棋子。

```
while (ChessBoard[x-i][y] == v) i++;
while (ChessBoard[x+j][y] == v) j++;
if (i+j>=6)
```

(3) 左对角线上连续 5 个棋子。

```
while(ChessBoard[x-i][y-i] == v ) i++;
while (ChessBoard[x+j][y+j] == v ) j++;
if (i+j>=6)
```

(4) 右对角线上连续 5 个棋子。

```
while(ChessBoard[x+i][y-i] == v) i++;
while(ChessBoard[x-j][y+j] == v) j++;
if(i+j-1>=6)
```

3. 源程序

```
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>
#include<windows.h>                    //包含设置坐标函数
#define N 15

int ChessBoard[N][N] ={0};             //棋盘图
int Cursor[2]={8,8};                   //当前光标坐标 X、Y 的默认位置
int ChessStep[255][2]={0};             //记录按照落子次序的棋盘坐标位置,用于悔棋

void gotoxy(int x, int y)              // windows.h 包含坐标操作函数
{
    COORD pos;                         //坐标位置变量
    pos.X=x-1;
```

```
    pos.Y = y - 1;
    //Windows 自带的设置坐标位置函数
    SetConsoleCursorPosition (GetStdHandle(STD_OUTPUT_HANDLE), pos);
}

void MoveTo(int x,int y)
{
    x = 3 * x - 2;                          //x 方向每三个字符为一个棋盘位置单位
    y = 2 * y - 1;                          //y 方向每两个字符为一个棋盘位置单位
    gotoxy(x,y);
}

//输入参数为光标移动的方向(1 表示上,2 表示下,3 表示左,4 表示右)
void Move(int direction)
{
    switch (direction)
    {
        case 1:Cursor[1] - = 1;break;
        case 2:Cursor[1] + = 1;break;
        case 3:Cursor[0] - = 1;break;
        case 4:Cursor[0] + = 1;break;
        default:printf("Move 函数出错");
    }
    MoveTo(Cursor[0],Cursor[1]);
}

//功能: 将按键转化为数值代码: 0 1 2 3 4 5 6 7
//参数数值代码含义:
//      0 表示输入错误,1 表示上,2 表示下,3 表示左,4 表示右;
//      5 表示落子,6 表示悔棋,7 表示退出
//返回值含义: 1 表示方向键,2 表示落子,3 表示悔棋,4 表示退出 0 表示错误
int GetKey(int * data)
{
    unsigned char temp;
    while (1)
    {
        temp = getch();
        if (temp == 224)                    //功能键,方向键需要两次判定
        {
            temp = getch();
            switch (temp)
            {
                case 72: * data = 1;break;      //向上键
                case 80: * data = 2;break;      //向下键
                case 75: * data = 3;break;      //向左键
                case 77: * data = 4;break;      //向右键
                default:continue;
            }
            return 1;
        }
        else
```

```
        {
            switch (temp)
            {
                case 13 : *data = 5;return 2;  //Enter 键
                case 32 : *data = 5;return 2;  //空格键
                case 8 : *data = 6;return 3;   //Backspace 键
                case 27 : *data = 7;return 4;  //Esc 键
                default : *data = 0 ;return 0;
             }
        }
    }
}

int InBoundry(int x,int y)                          //位置是否出界
{
    if (x == -1||x == N||y == -1||y == N) return 0;
    else return 1;
}

int Win(int v)                                      //构成连续 5 子返回 1,否则返回 0
{
    int i = 1,j = 1,x = 0,y = 0;
    //左对角线上连续 5 个棋子
    while (ChessBoard[x = Cursor[0] - i - 1][y = Cursor[1] - i - 1] == v&&InBoundry(x,y)) i++;
    while (ChessBoard[x = Cursor[0] + j - 1][y = Cursor[1] + j - 1] == v&&InBoundry(x,y)) j++;
    if (i + j>= 6) return 1;

    //右对角线上连续 5 个棋子
    i = 1,j = 1,x = 0,y = 0;
    while (ChessBoard[x = Cursor[0] + i - 1][y = Cursor[1] - i - 1] == v&&InBoundry(x,y)) i++;
    while (ChessBoard[x = Cursor[0] - j - 1][y = Cursor[1] + j - 1] == v&&InBoundry(x,y)) j++;
    if (i + j>= 6) return 1;

    //同一列上连续 5 个棋子
    i = 1,j = 1,x = 0,y = 0;
    while (ChessBoard[x = Cursor[0] - i - 1][y = Cursor[1] - 1] == v&&InBoundry(x,y)) i++;
    while (ChessBoard[x = Cursor[0] + j - 1][y = Cursor[1] - 1] == v&&InBoundry(x,y)) j++;
    if (i + j>= 6) return 1;

    //同一行上连续 5 个棋子
    i = 1,j = 1,x = 0,y = 0;
    while (ChessBoard[x = Cursor[0] - 1][y = Cursor[1] - i - 1] == v&&InBoundry(x,y)) i++;
    while (ChessBoard[x = Cursor[0] - 1][y = Cursor[1] + j - 1] == v&&InBoundry(x,y)) j++;
    if (i + j>= 6) return 1;
    return 0;
}

void GameStart()                                    //初始化棋盘坐标值,画棋盘
{
    int i,j;
    for (i = 0;i< N;i++)
```

```
        for (j = 0;j < N;j++)
            ChessBoard[i][j] = 0;

    system("cls");
    for (i = 1;i < 30;i++)
    {
        for (j = 1;j <= 43;j++)
            if (i % 2 == 1) printf("-");
            else if (j % 3 == 1) printf("|");
                else printf(" ");
        printf("\n");
    }

    Cursor[0] = 8;
    Cursor[1] = 8;
    gotoxy(50, 10);
    printf("使用");
    for (i = 24;i < 28;i++)
        printf("%2c",i);
    printf(" 键移动光标 ");
    gotoxy(50, 12);
    printf("使用空格或 Enter 键落子");
    gotoxy(50, 14);
    printf("按 Backspace 键悔棋 ");
    gotoxy(50, 16);
    printf("按 Esc 键返回主菜单 ");
    gotoxy(50, 20);
    printf("请玩家 1 下棋... ");
    MoveTo(8,8);
}

int OutBoundry(int data)                          //沿 data 方向移动是否出边界
{
    int StepXY[2],New[2];
    switch (data)
    {
        case 1: StepXY[0] = 0;StepXY[1] = -1 ;break;
        case 2:StepXY[0] = 0;StepXY[1] = 1;break;
        case 3:StepXY[0] = -1;StepXY[1] = 0;break;
        case 4:StepXY[0] = 1;StepXY[1] = 0 ;break;
        case 5:StepXY[0] = 0;StepXY[1] = 0 ;break;
    }

    New[0] = Cursor[0] + StepXY[0];
    New[1] = Cursor[1] + StepXY[1];
    if (New[0]> N||New[0]< 1||New[1]> N||New[1]< 1) return 1;
    else return 0;
}

//第一个输入值为按键的玩家号
//第二个为按下键的对应值
```

```
int KeyError(int v,int direction)
{
    if (v == 3||v == 4) return 0;                    //如果输入的为其他按键,则 KeyError 为假
    else if (v == 0) return 1;                       //如果输入错误,则 KeyError 为真
    if (OutBoundry(direction)) return 1;             //如果移动方向出边界,则 KeyError 为真
        return 0;
}

int luozi(int v)                                     //玩家 v 落子
{
    ChessBoard[Cursor[0] - 1][Cursor[1] - 1] = v;
    if (v == 1) printf("O");
    else if (v == 2) printf("X");
    if (Win(v)) {
        gotoxy(50,20);
        printf("玩家 %d 赢了! ",v);
        MoveTo(Cursor[0],Cursor[1]);
        getch();
        return 1;
    }
    gotoxy(50,20);
    printf("现在请玩家 %d 下棋 ",v % 2 + 1);
    MoveTo(Cursor[0],Cursor[1]);
    return 0;
}

void HuiQi(int step)                                 //step 为当前要悔的是第几步棋
{
    Cursor[0] = ChessStep[step - 1][0];
    Cursor[1] = ChessStep[step - 1][1];
    ChessBoard[Cursor[0] - 1][Cursor[1] - 1] = 0;

    MoveTo(Cursor[0],Cursor[1]);
    printf(" - ");
    gotoxy(50,20);
    printf("现在请玩家 %d 下棋 ",(step + 1) % 2 + 1);
    MoveTo(Cursor[0],Cursor[1]);
}

int Play()
{                                                    /* direction 0 表示不可移动,1 表示上,2 表示下,3 表示左
                                                       4 表示右,5 表示落子,6 表示悔棋,7 表示退出 */
    int direction = 0;
    int i = 0,temp;

    while (1)
    {
        while (temp = GetKey(&direction),KeyError(temp,direction));
        switch (direction)
        {
            case 1:
```

```
            case 2:
            case 3:
            case 4:Move(direction);break;
            case 5:
                if (ChessBoard[Cursor[0]-1][Cursor[1]-1]) continue;
                    if (luozi(i%2+1)) return 0;
                    ChessStep[i][0]=Cursor[0];      //记录落子步骤
                    ChessStep[i][1]=Cursor[1];
                    i++;
                    break;
            case 6:
                if (i==0)
                {
                    gotoxy(50,24);
                    printf("现在无法悔棋 ");
                    MoveTo(Cursor[0],Cursor[1]);
                }
                else HuiQi(i--);
                break;
            case 7:return 0;
        }
    }
    return 1;
}

int main()
{
    GameStart();
    Play();
    gotoxy(50,22);
    return 0;
}
```

4. 实训题目

查看五子棋游戏的其他规则，修改程序，使其适宜作为一个游戏软件。

3.2　迷宫问题

1. 问题描述

用一个 m 行 n 列的二维数组来表示迷宫，数组中每个元素的取值为 0 或 1，其中值为 0 表示通路，值为 1 表示阻塞，迷宫的入口在左上方(1,1)处，出口在右下方(m,n)处。要求求出从迷宫入口到出口有无通路，若有通路，则指出其中一条通路的路径，即输出找到通路的迷宫数组，其中通路上的“0”用另外一个数字 8 替换，同时打印所走通路径上每一步的位置坐标及下一步的方向，如图 3-2 所示。

2. 算法描述

计算机解迷宫，就是从入口出发，顺某一方向向前探索，若能走通，则继续往前走；否则

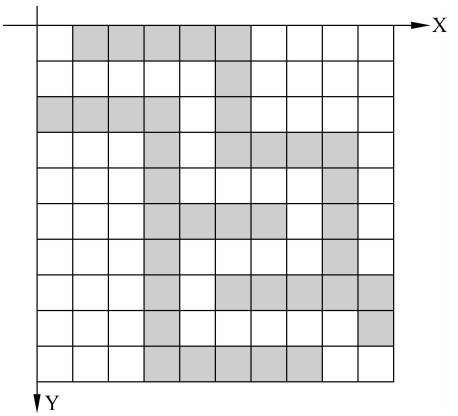

图 3-2　迷宫路线图

沿原路退回，换一个方向再继续探索，直至所有可能的通路都探索到为止。因此，迷宫可以看做一个图，只要找到一条通路即可，可以用深度优先算法实现。用邻接矩阵表示的深度优先搜索算法如下：

```
void DFS(Graph * g,int i)              //设邻接矩阵 g 是 0,1 矩阵,i 表示顶点 vi
{                                      //以 vi 为出发点,对邻接矩阵表示的图 g 进行 DFS 搜索
   int j;                              //vj 为 vi 的邻接点
   printf("visit vertex: %c",g->vexs[i]);   //访问顶点 vi
   visited[i] = 1;                     //visited 为标志数组.0:vi 未被访问过;1:表示 vi 访问过
   for(j = 0; j<g->n; j++)             //依次搜索 vi 的邻接点
        if (g->edges[i][j] == 1&&!visited[j])
            DFS(g,j)                   //(vi,vj)∈E,且 vj 未访问过,故 vj 为新出发点 DFS
    }
```

针对迷宫问题，用二维数组表示邻接矩阵，用当前位置相邻 4 个方向有无通路(元素值为 1 或 0)表示边的权值，每个顶点最多有 4 条边。

(1) 以二维数组 maze[m][n]表示迷宫，并设 maze[1][1]处为迷宫入口，maze[m][n]处为迷宫出口，迷宫中的任一位置以 maze[x][y]来表示。为了简化边界位置的检测，将二维数组 maze[m][n]扩充到 maze[m+2][n+2]，且令其四周边界位置的值均为 1。

(2) 对于迷宫中的每个位置(x,y)处，可能移动的路线有 4 个方向，用一维数组 dx[4]、dy[4]表示这 4 个方向上的一种跳法对应 x、y 坐标的变化量。

```
int dx[4] = {0, -1,0,1};//沿北、西、南、东方向 x 坐标的变化量
int dy[4] = { -1,0,1,0};//沿北、西、南、东方向 y 坐标的变化量
```

如果(x,y)下一位置的坐标为(xNext,yNext)，(xNext,yNext)不超出边界，没有走过，则把(xNext,yNext)作为当前的位置坐标，则进入下一步递归“SeekPath (xNext,yNext);”。

(3) 具体算法可以概括如下：

① 设初始位置为 (startx,starty)，终止位置为(endx,endy)，走迷宫过程中，如果当前位置(x,y)已到达出口，即(x,y)=(endx,endy)，则说明已找到一条通路，返回值为 1，结束递归。

② 如果当前位置的各个方向上都没有通路，若此时的位置在(startx，starty)，给出“没有通路”的信息，则递归返回值 0；否则，退回上一个位置，沿其他方向搜索。

③ 对当前位置的各个方向依次搜索的过程中，发现某个方向的位置可以走通(即 maze 与 visited 数组中该位置的值均为 0)，则把该位置作为当前位置，继续向前搜索。

在记录走通的每步位置时，考虑递归的特点，若要正序打出走通的路径，则可以令起始位置和终止位置颠倒，即可实现目的。

3．源程序

```
#include <stdio.h>
#define m 12
#define n 15                          //迷宫的大小
//输入迷宫地图
int maze[m+2][n+2]={ 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                     1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,
                     1,0,0,0,0,1,1,0,1,1,1,0,0,1,1,1,1,
                     1,0,1,1,0,0,0,0,1,1,1,1,0,0,1,1,1,
                     1,1,1,0,1,1,1,0,0,1,1,0,1,1,0,0,1,
                      1,1,1,0,1,1,1,0,0,1,1,0,1,1,0,0,1,
                      1,1,1,0,1,0,0,1,0,0,1,1,1,1,1,1,1,
                      1,0,0,1,1,0,1,1,1,0,1,0,0,1,1,1,1,
                      1,0,1,1,1,1,0,0,1,0,1,1,1,1,1,1,1,
                      1,0,0,1,1,0,1,1,0,0,1,1,1,1,0,1,1,
                      1,1,1,0,0,0,1,1,0,1,1,0,0,0,0,0,1,
                      1,0,0,1,1,1,1,1,0,0,0,0,0,0,1,0,1,
                      1,0,1,0,0,1,1,1,1,1,0,1,0,1,0,0,1,
                      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
int visited[m+2][n+2];                //设立迷宫位置走过标志
int dx[4]={0,-1,0,1};                 //沿北、西、南、东方向
int dy[4]={-1,0,1,0};
int SeekPath(int x,int y)
{
    int next,xNext,yNext;
    if (x==1&&y==1)                   //如果达到终点，则说明找到路径，返回成功
        return 1;

    for (next=0;next<4;next++)        //沿每一个方向搜索
    {
        xNext=x+dx[next];             //设置该方向对应点的新坐标
        yNext=y+dy[next];
        //如果该点走得通且没有被探索过
        if (maze[xNext][yNext]==0&&visited[xNext][yNext]==0)
        {
            visited[xNext][yNext]=1;          //将该点置为探索过
            if (SeekPath(xNext,yNext))        //从该点开始新的探索，假定探索成功
            {
            printf("(%d,%d)",xNext,yNext);    //输出该点坐标
            if (dx[next]==1) printf("North->");               //判断前一点到该点的方向
                if (dx[next]==-1)
                    printf("South->");
```

```
                if (dy[next] == 1)
                    printf("West->");
                if (dy[next] == -1)
                    printf("East->");
                printf("\n");
                maze[xNext][yNext] = 8; //把该点设为通路
                return 1;
            }
        }
    }

    if (x == m&&y == n)
        printf("No path!\n");       //如果最后返回起点,则说明没有通路
    return 0;
}

int main()
{
    int row,col;
    for (row = 0;row < m + 2;row++)
        for (col = 0;col < n + 2;col++)
            visited[row][col] = 0;  //先将所有通路设置为没有走过状态
    visited[m][n] = 1;              //将起点设置为走过状态

    if (SeekPath(m,n))              //如果走通
    {
        printf("(%d,%d)\n",m,n); //先输出起点坐标
        maze[m][n] = 8;             //将起点设为通路
        for (row = 0;row < m + 2;row++)
            for (col = 0;col < n + 2;col++)
            {
                 printf("%d ",maze[row][col]);
                 if (col == n + 1)
                    printf("\n");
            }                       //打印出走通后的迷宫
    }
    return 0;
}
```

4. 实训题目

(1) 如果从迷宫的某一点到达另一点,如何实现?

(2) 查资料,迷宫问题有哪些实际应用价值?

3.3 骑士巡游问题

1. 问题描述

在n行n列的棋盘上,一位骑士(按象棋中“马走日”的行走法)从初始坐标位置(x1,y1)出发,要遍访(巡游)棋盘中的每一个位置一次,共有多少种走法?并打印出所有走法。