汉诺塔实验报告

信06 2352018 刘彦

2024/5/16

1. 汉诺塔的移动设计思路（主要函数说明）

1.1.汉诺塔递归函数hanoi

完成递归并增加移动次数计数器

1.2.输入函数my_input

输入层数，起始柱，目标柱等，同时在读取输入值的时候进行判断和错误处理。

1.3.移动函数

1.3.1. move函数

用于移动数组，利用一个一维数组作为栈顶指针，函数调用了moveA, moveB, moveC函数来分别移动ABC三根柱子。

1.3.2. move_plate函数

用于移动盘子，通过调用cct_showch函数，完成盘的初始化，上升下降和平移，主体是三个循环。

1.4.输出函数

1.4.1. print函数

用于打印出汉诺塔数组（hanoi0）中的数，函数调用了3个printTower函数来打印ABC三根柱子。

1.4.2. print_line函数

逐行输出数组，实现最基础的移动步数的输出。

1.4.3. print_row函数

横向输出数组，并调用move函数，实现数组的横向输出和移动。

1.4.4. print_column函数

分为两个函数，print_column_base函数用来打印柱名和横线，print_column_tower函数通过调用cct_gotoxy函数，实现hanoi0数组的纵向输出和移动过程。

1.4.5. draw_tower函数

打印底座，柱子和盘，确定底座柱子和盘的宽度和高度。

1.5.初始化函数

1.5.1. init函数

在输入src，dst和storey后确定并打印初始状态。

1.5.2. init_plate函数

共有四个函数，均在画盘时使用。init_plate_height函数用于确定每个盘的高度，init_plate_width函数用于确定每个盘的宽度，init_plate_color函数用于确定每个盘的颜色，init_plate_dst是一个辅助函数，用于确定目标柱。

1.6.菜单函数hanoi_menu

显示各菜单项，读入正确的选项后返回，输入不用回显。作用是做输入提示，自动清除错误输入并刷新。

1.7.汇总函数hanoi_all

汇总整个汉诺塔的输出函数。

2. 过程及设计思路

2.1.基本解

调用递归函数即可完成。

2.2.基本解(步数记录)

在基本解的基础上添加一个静态全局变量进行步数记录。

2.3.内部数组显示(横向)

设置了一个二维数组hanoi0[3][10]来表示三个柱，一个一维数组top来做栈顶指针，负责控制进栈出栈，然后调用print_row函数打印结果。

2.4.内部数组显示(横向+纵向)

纵向与横向本质相同，不同的是调用cct_gotoxy函数将横向的数组转向纵向，都利用栈顶指针控制进出栈。

2.5. 图形解-预备-画三个圆柱

利用伪图形界面工具函数cct_showstr和cct_showch在屏幕上画三根柱子，设置为亮黄色，设定好底座的宽度和柱子的高度即可。并利用cct_gotoxy函数将柱子放在适宜位置。在生成时，由于有sleep函数，是一点点逐步画成的。

2.6. 图形解-预备-在起始柱上画n个盘子

在画三个圆柱的基础上绘制盘子，盘子的位置由柱的位置稍作调整后确定，在画盘前，应将文本颜色设置为黑色，背景颜色设置为白色。然后用循环绘制动态矩形，用迭代确定颜色和位置，画好后再将文本颜色设置为黑色，背景颜色设置为白色。

2.7. 图形解-预备-第一次移动

在画好柱和盘后，调用hanoi_move函数来确定第一步的目标柱，然后在进行盘的移动。调用move_plate函数，绘制盘后利用cct_gotoxy函数和参数自加自减完成平移的操作，使盘移到新位置，盘的原位置用黑色覆盖，以达到模拟盘移动的效果。

2.8. 图形解-自动移动版本

图形解-自动移动版本与第一次移动的方法类似，只是需要调用汉诺塔递归函数hanoi，确定下一次要移动的盘和位置即可，二者盘的移动方法相同。

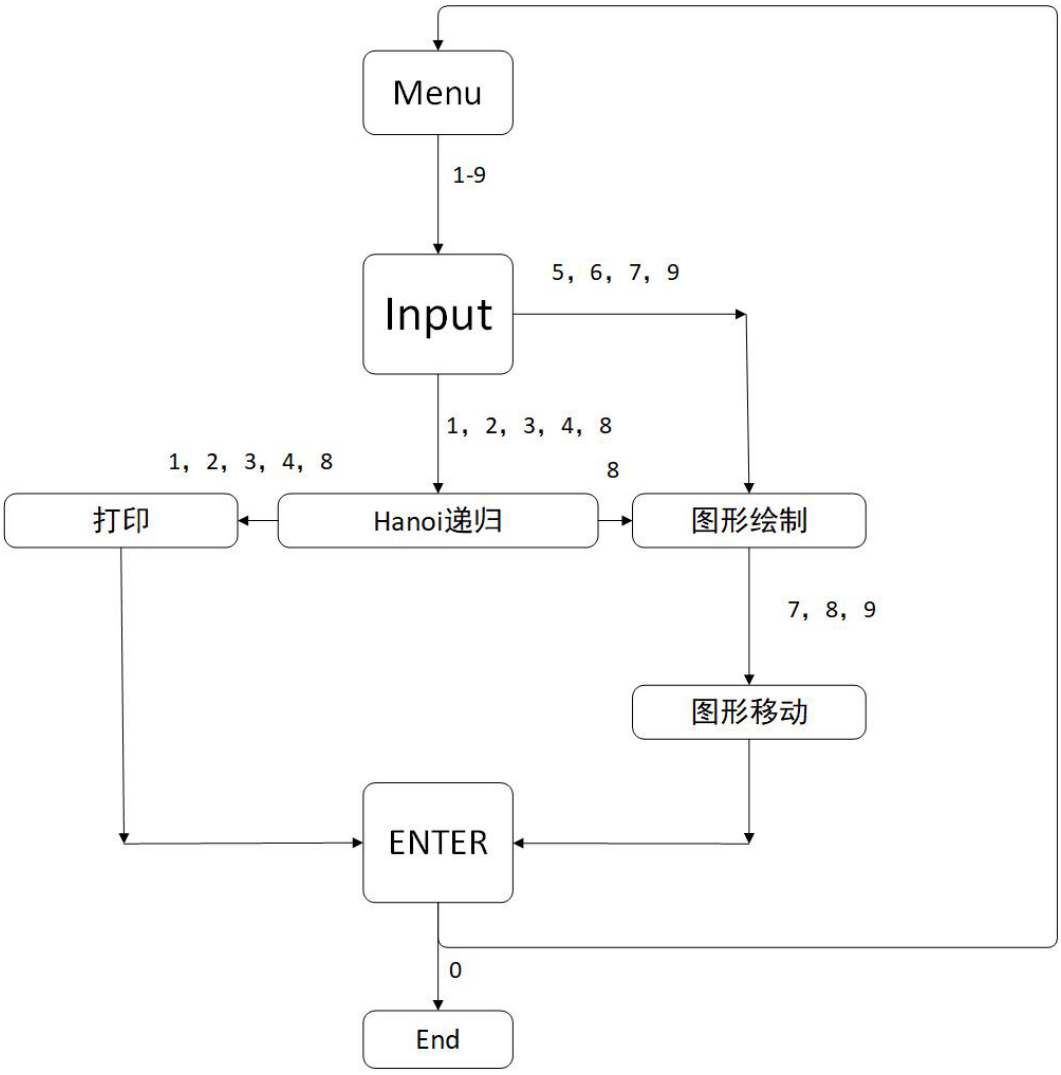
2.9. 图形解-游戏版

游戏版与自动移动版本的不同是不用调用汉诺塔递归函数hanoi，而是通过输入来确定下一步的目标柱，盘的移动方法与之前都相同。难点是错误处理，即判断是否大盘压小盘以及src柱是否为空，需要增设条件语句。并且还要在dst柱上盘满后判断“游戏结束”。

2.10. 退出

输入0是程序退出，否则是无限循环，再一次操作结束后按回车键可以重新开始选择。

3. 主要功能的实现



3.1. 输入

该程序可以通过参数的输入选择功能，输入参数后首先判断正确性，，进行错误处理。然后根据参数来选择输入方式和调用函数，同样要进行错误处理，随后将输入数据传入相应函数中。

3.2. 内部数组打印

该题中用一个全局二维数组来储存三个柱子上的盘号，利用一个全局一维来充当栈顶指针并储存三根柱子上的圆盘的个数。

盘子移动后，对应的柱子数组值就会改变，通过栈顶指针控制进栈出栈，在一次进出栈后重新打印（水平数组打印，纵向数组打印），展现柱上圆盘的情况。

3.3. 图形绘制

使用伪图形工具函数集生成三根柱子和相应的圆盘数量，保证每个柱子高度与颜色不变，每个圆盘

颜色不同且从上到下依次变大。

3.4.图形移动

使用图形绘制函数将盘的移动过程可视化，通过sleep函数，可以看到盘每一步的移动信息。在进行盘子移动的时候首先要确定盘子的起始和终点的位置坐标，移动可以分为三种情况。

- 向上移动，起点为盘子的所在的高度，终点高度为柱高以上的定值
- 向下移动，起点为盘子所在的柱高以上的定值，终点高度为目标盘子所在高度。
- 平移，起点为起始柱的水平坐标，终点为目标柱的水平坐标。

移动时，在下一个目标位置重新绘制盘，将原位置的盘用黑色抹去，以此来实现移动的效果。

4. 调试过程碰到的问题

4.1.纵向输出时字符位置的问题

起初调试纵向输出时，输出字符与下面横线及横向输出之间的位置较为混乱，调试起来找不到哪里出了问题，并容易将原本对的程序改错。为解决这个问题，我将这些位置坐标都用宏定义表示，这样在描述其相对位置上就会方便很多，并且为之后的改正和纠错提供便利。

并且，在做到图形解-自动移动版本时，宏定义为横向与纵向输出的位置调整也提供了便利，避免了很多计算的麻烦，提高了代码可读性。

4.2.第一次移动盘时目标柱的问题

刚开始在第一次移动盘时，程序在目标柱的计算上出现了错误，我写了较长代码但依旧不能发挥作用。后来，我尝试采用构造辅助函数的方法来实现计算，结果依然没有满足。在仔细检查后，我发现是因为我当时忽视了函数形参不能双向传值的问题，从而导致第一次直接移到输入的dst柱上，造成错误。

最后，通过修改函数类型，调整函数逻辑，将该函数返回值作为第一次移动的目标柱，从而解决问题。

4.3.盘水平移动方向的问题

第一次写好水平移动的代码时，判断其移动方向的逻辑并不清晰，以至于出现了错误，例如移到了柱子外面。经调试发现，是因为在迭代过程中“+1”或“-1”的问题，对横坐标加减可以控制其移动方向。

为了解决这个问题，我写了一个辅助函数，并设置条件判断（dst - src的正负）该加还是该减，控制不让盘移到柱外去。

5. 心得体会

5.1.在做作业时要不断优化以前代码

做作业时，直接把之前的程序程序粘贴过来可能不是最佳选择，反而有可能因为逻辑混乱增加工作量，而应该不断优化，使其逻辑更加清晰。适合在大作业中使用。

5.2.要多设定函数

在实际操作中我发现，多设计函数可以帮助我理清程序的逻辑，从而在查找和修改错误是效率更高，提高了代码可读性。在编写复杂程序的时候，将预期实现功能大致相同部分对应的代码整合成一个函数，可以有效的减少代码量。

另外，函数的命名应当体现其功能，这样在调用和检查时会更加方便，不能因为追求省事而随意使用简单字母命名。

5.3.充分利用注释

在规模较大的程序中，代码的逻辑可能比较复杂，所以应该对每个函数和要有清晰的注释对函数和模块的用途和功能进行描述，解释输入和输出的定义和对变量及函数命名的解释。

通过注释快速的理解该代码的逻辑，增强可读性，且便于修改。

5.4.及时完成作业

人在紧张时往往会产生错误，不应赶ddl，应该及时完成作业，这样才有充分时间思考。

6. 附件：源程序

```
void draw_tower(int n, int src, int choice)
{
    int i, j, k, m;
    int bottom, stick;   //打印底座和柱子
    int color1, color2;   //盘和背景的颜色
    int startX, startY;   //打印盘的位置
    for (i = 0; i < 3; i++)   //打印底座
    {
        bottom = BOTTOM_X + LOC_TOWER * i;
        Sleep(TIME_SLEEP);
        cct_showstr(bottom,    BOTTOM_Y,    ″    ″,
COLOR_HYELLOW, COLOR_HYELLOW, BOTTOM_WIDTH);
    }

    for (j = 14; j > 1; j--)   //打印柱子
    {
        for (k = 0; k < 3; k++)
        {
            stick  =  BOTTOM_Y  +  LOC_TOWER  *  k  -
BOTTOM_X - 2;
            Sleep(45);
            cct_showch(stick,   j   +   1,   ’   ’,
COLOR_HYELLOW, COLOR_HYELLOW, STICK_WIDTH);
        }
    }
    // 将文本颜色设置为黑色，背景颜色设置为白色
    cct_setcolor(COLOR_BLACK, COLOR_WHITE);

    if (choice == 6 || choice == 7 || choice == 8)
    {
        int num = src - ’A’;

        // 循环绘制动态矩形
        for (m = n; m >= 1; m--)
        {
            Sleep(TIME_SLEEP);

            // 计算当前迭代的颜色值
            color1 = COLOR_BLUE + m;
            color2 = COLOR_BLUE + m;
```

```
            // 计算当前迭代的位置
            startX = LOC_LINE_X + LOC_TOWER * num - m + 2;
            startY = LOC_LINE_Y - n + m + 1;
            cct_showch(startX, startY, ' ', color1, color2, 1 + 2 * m);
        }
        // 将文本颜色设置为黑色，背景颜色设置为白色
        cct_setcolor(COLOR_BLACK, COLOR_WHITE);
    }
}

void draw_move(char src, char x, int i)
{
    int height = init_plate_height(src);
    int width = init_plate_width(src);
    int loc_x = (x - 'A') * LOC_TOWER + (BOTTOM_WIDTH + BOTTOM_X - width + 1) / 2;
    int color = init_plate_color(src);

    cct_showch((x - 'A') * LOC_TOWER + 1, i, ' ', COLOR_BLACK, COLOR_WHITE, BOTTOM_WIDTH);
    if (i > 2 && i < 15)   //打印柱
        cct_showch((x - 'A') * LOC_TOWER + (BOTTOM_WIDTH + BOTTOM_X) / 2, i, ' ', COLOR_HYELLOW, COLOR_HYELLOW, STICK_WIDTH);
    //在屏幕上的特定位置绘制一个盘
    cct_showch(loc_x, i + 1, ' ', COLOR_BLACK + color, COLOR_BLACK + color, width);
    height = i - 1;
}

void drawCharacter(int x, int y, int flag, int color, int width)
{
    cct_showch(x, y, ' ', COLOR_BLACK, COLOR_WHITE, width);

    // 根据水平距离的值决定绘制颜色字符的位置
    int offset = (flag > 0) ? 1 : -1;
    int color0 = COLOR_BLACK + color;

    // 绘制具有特定颜色的字符
    cct_showch(x + offset, y, ' ', color0, color0, width);
}

void printTower1(int tower[], char label)
{
    cout << label << ":";
    for (int i = 0; i < 10; ++i)
    {
        if (hanoi0[0][i] == 0)
        {
            cout << "  "; // 两个空格表示没有盘
        }
        else
        {
            cout << setw(2) << tower[i];
        }
    }
    cout << " ";
}

void print()
{
    printTower1(hanoi0[0], 'A');
    printTower2(hanoi0[1], 'B');
    printTower3(hanoi0[2], 'C');
}

void moveA(char dst)
{
    int a;
    a = hanoi0[0][--top[0]];
    hanoi0[0][top[0]] = 0;
    if (dst == 'B')
    {
        hanoi0[1][top[1]++] = a;
    }
    if (dst == 'C')
    {
        hanoi0[2][top[2]++] = a;
    }
}
void move(char src, char dst)
{
    if (src == 'A')
        moveA(dst);
    if (src == 'B')
        moveB(dst);
    if (src == 'C')
        moveC(dst);
}

void move_plate(char src, char dst)
{
    int height = init_plate_height(src);
    int width = init_plate_width(src);
    int loc_x = (src - 'A') * LOC_TOWER + (BOTTOM_WIDTH + BOTTOM_X - width + 1) / 2;
    int color = init_plate_color(src);
    int loc_y = init_plate_dst(dst);
    int flag = (dst - src) * LOC_TOWER;
    int i = height, j = height;
    //绘制盘
    for (; i > 1; i--)
    {
        Sleep(TIME_SLEEP);
        cct_showch((src - 'A') * 32 + 1, i, ' ', COLOR_BLACK, COLOR_WHITE, BOTTOM_WIDTH);
        if (i > 2 && i < 15)   //打印柱
```

```
            cct_showch((src - 'A') * LOC_TOWER + (BOTTOM_WIDTH + BOTTOM_X) / 2, i, ' ', COLOR_HYELLOW, COLOR_HYELLOW, STICK_WIDTH);
        //在屏幕上的特定位置绘制一个盘
        cct_showch(loc_x, i - 1, ' ', COLOR_BLACK + color, COLOR_BLACK + color, width);
        height = i - 1;
    }
    i = height;
    //平移盘
    while (1)
    {
        Sleep(TIME_SLEEP);
        cct_showch(loc_x, height, ' ', COLOR_BLACK, COLOR_WHITE, width);
        drawCharacter(loc_x, height, flag, color, width);

        // 更新 x 的位置
        loc_x += (flag > 0) ? 1 : (flag < 0) ? -1 : 0;

        // 检查是否达到停止条件
        int locate = flag + (src - 'A') * LOC_TOWER + (BOTTOM_WIDTH + BOTTOM_X - width + 1) / 2;
        if (loc_x == locate)
            break;
    }
    //降落盘
    j = height;
    while (j < loc_y)
    {
        Sleep(TIME_SLEEP);
        draw_move(src, dst, j);
        j++;
    }
    cct_setcolor(COLOR_BLACK, COLOR_WHITE);
}

void print_all(int n, char src, char tmp, char dst, int choice)
{
    char dst_7 = 0;
    switch (choice)
    {
        case 1:
            print_line(n, src, tmp, dst, 1);
            break;
        case 2:
            print_line(n, src, tmp, dst, 2);
            break;
        case 3:
            print_row(n, src, tmp, dst, 3);
            break;
        case 4:
            switch (speed)
            {
                case 0:
                    if (_getch() == 13)
                    {
                        print_row(n, src, tmp, dst, 4);
                        print_column_base(4);
                        print_column_tower(4);
                    }
                    break;
                case 1:
                    Sleep(250);
                    print_row(n, src, tmp, dst, 4);
                    print_column_base(4);
                    print_column_tower(4);
                    break;
                case 2:
                    Sleep(200);
                    print_row(n, src, tmp, dst, 4);
                    print_column_base(4);
                    print_column_tower(4);
                    break;
                case 3:
                    Sleep(150);
                    print_row(n, src, tmp, dst, 4);
                    print_column_base(4);
                    print_column_tower(4);
                    break;
                case 4:
                    Sleep(100);
                    print_row(n, src, tmp, dst, 4);
                    print_column_base(4);
                    print_column_tower(4);
                    break;
                case 5:
                    Sleep(50);
                    print_row(n, src, tmp, dst, 4);
                    print_column_base(4);
                    print_column_tower(4);
                    break;
            }
            break;
        case 7:
            dst_7 = hanoi_move(n, src, tmp, dst);
            move_plate(src, dst_7);
            break;
        case 8:
            switch (speed)
            {
                case 0:
                    if (_getch() == 13)
                    {
                        print_row(n, src, tmp, dst, 8);
                        move(dst, src);
                        move_plate(src, dst);
                        move(src, dst);
                    }
```

```
                break;//按回车单步演示,回车的ASCII码的值为13
            case 1:
                Sleep(250);
                print_row(n, src, tmp, dst, 8);
                move(dst, src);
                move_plate(src, dst);
                move(src, dst);
                break;
            case 2:
                Sleep(200);
                print_row(n, src, tmp, dst, 8);
                move(dst, src);
                move_plate(src, dst);
                move(src, dst);
                break;
            case 3:
                Sleep(150);
                print_row(n, src, tmp, dst, 8);
                move(dst, src);
                move_plate(src, dst);
                move(src, dst);
                break;
            case 4:
                Sleep(100);
                print_row(n, src, tmp, dst, 8);
                move(dst, src);
                move_plate(src, dst);
                move(src, dst);
                break;
            case 5:
                Sleep(50);
                print_row(n, src, tmp, dst, 8);
                move(dst, src);
                move_plate(src, dst);
                move(src, dst);
                break;
        }

        break;

    }
}

void print_row(int n, char src, char tmp, char dst, int choice)
{
    switch (choice)
    {
        case 3:
            break;
        case 4:
            cct_gotoxy(LOC_ROW_X, LOC_ROW_Y);
            break;
        case 8:
            cct_gotoxy(LOC_ROW_X, LOC_ROW_Y + 18);
            break;
    }
    cout << "第" << setw(4) << Count << " 步" << "(" << setw(2) << n << ")" << ": " << src << "-->" << dst << " ";
    move(src, dst);
    print();
    cout << endl;
}

void print_column_tower(int choice)
{
    int i, j, h;

    switch (choice)
    {
        case 4:
            for (i = 0; i < 3; i++)
            {
                h = LOC_Y_4 - 2;
                for (j = 0; j < 10; ++j)
                {
                    cct_gotoxy(LOC_COLUMN * (i + 1), h);
                    if (hanoi0[i][j] == 0)
                    {
                        cout << "  "; // 两个空格表示没有盘
                    }
                    else
                    {
                        cout   <<   setw(2)   <<   hanoi0[i][j];
                    }
                    h--;
                }
                cout << " ";
            }
            break;
        case 8:
            for (i = 0; i < 3; i++)
            {
                h = LOC_Y_8 - 2;
                for (j = 0; j < 10; ++j)
                {
                    cct_gotoxy(LOC_COLUMN * (i + 1), h);
                    if (hanoi0[i][j] == 0)
                    {
                        cout << "  "; // 两个空格表示没有盘
                    }
                    else
                    {
                        cout   <<   setw(2)   <<   hanoi0[i][j];
                    }
                    h--;
```

```
                }
                cout << " ";
            }
            break;
    }
}

void init(int n, int src, int choice)
{
    int i;
    if (src == 'A')
    {
        for (i = 0; i < n; i++)
        {
            hanoi0[0][i] = n - i;
        }
        top[0] = n;
    }
    if (src == 'B')
    {
        for (i = 0; i < n; i++)
        {
            hanoi0[1][i] = n - i;
        }
        top[1] = n;
    }
    if (src == 'C')
    {
        for (i = 0; i < n; i++)
        {
            hanoi0[2][i] = n - i;
        }
        top[2] = n;
    }
    if (choice == 3 || choice == 4 || choice == 8)
    {
        cout << "初始:";   // 打印初始状态
        cout << "                  ";
        print();
        cout << endl;
    }
}

int init_plate_color(char src)
{
    int color = 0;
    if (src == 'A')
    {
        color = hanoi0[0][top[0] - 1];
    }
    if (src == 'B')
    {
        color = hanoi0[1][top[1] - 1];
    }
    if (src == 'C')
    {
        color = hanoi0[2][top[2] - 1];
    }

    return color + 1;
}

int init_plate_dst(char dst)
{
    int loc_y = BOTTOM_Y - top[dst - 'A'] - 1;
    return loc_y;
}

void my_speed()
{
    while (1)
    {
        cout << "请输入移动速度(0-5: 0-按回车单步演示 1-延时最长 5-延时最短) ";
        cin >> speed;
        if (cin.good() && speed >= 0 && speed <= 5)
            break;
        else
        {
            cin.clear();
            cin.ignore(65536, '\n');
        }
    }
}

void hanoi(int n, char src, char tmp, char dst, int choice)
{
    if (n == 1)
    {
        Count++;
        print_all(n, src, tmp, dst, choice);
    }
    else
    {
        hanoi(n - 1, src, dst, tmp, choice);
        Count++;
        print_all(n, src, tmp, dst, choice);
        hanoi(n - 1, tmp, src, dst, choice);
    }
}
```