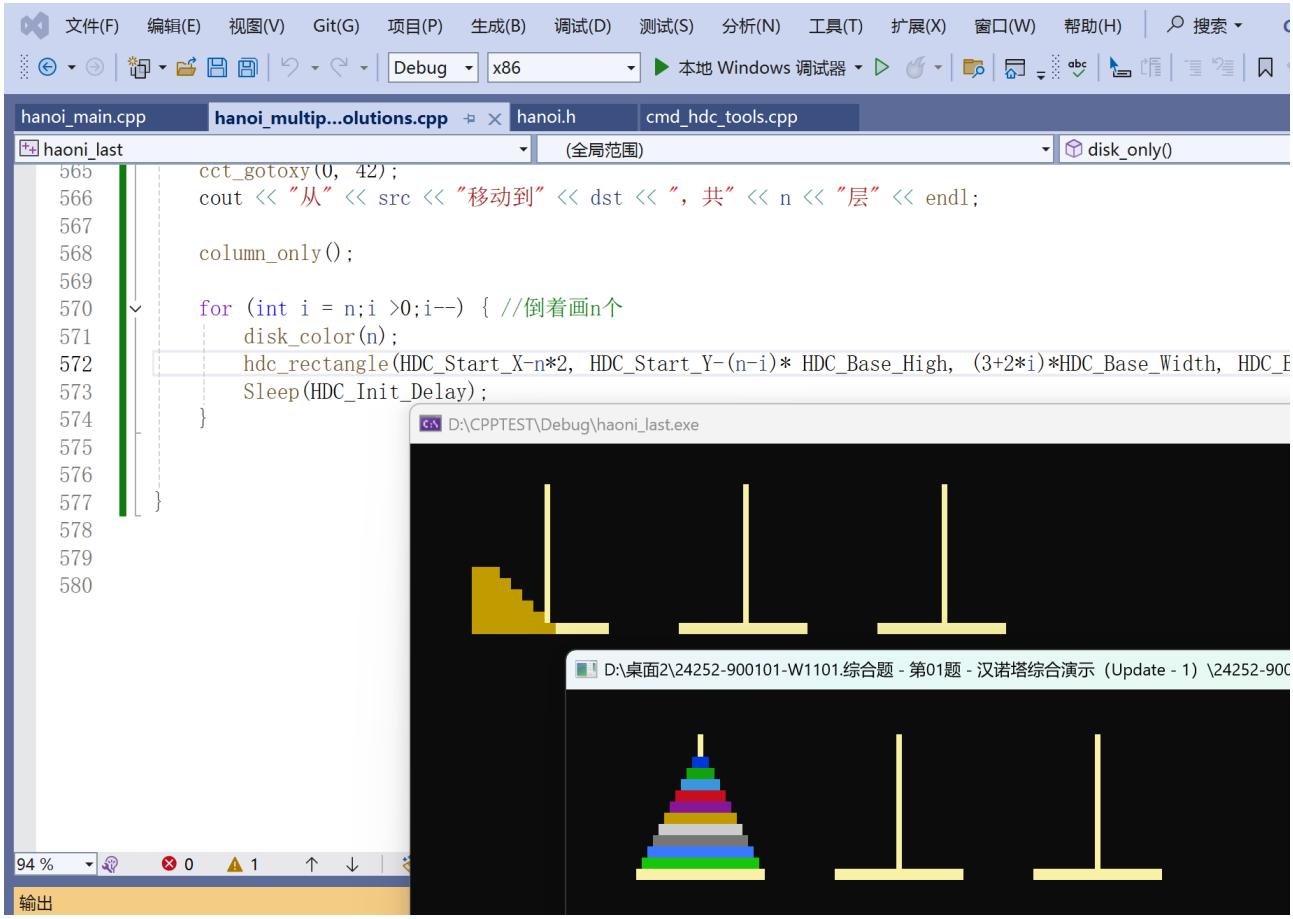
汉诺塔调试实验报告

2452545 计算机 课号5000244001608 刘晴语

2025.5.20

附作品：《代码实体化》



1. 题目

1. 整个程序只允许使用一个递归函数，即菜单项 1/2/3/4/8 必须共用一个递归函数，用参数解决各菜单项不同要求之间的差异,递归函数按一句一行计算(包含独立成行的左右大括号),不得超过 15 行
2. 1/2/3/4/6/7/8 中的输入多个参数必须共用一个函数,用参数解决输入不同内容的问题(本函数允许使用第 6 章的知识:函数形参为实参的指针，可以同时改变多个实参值)
3. 菜单项 3/4/8 中的横向输出必须共用一个函数，用参数解决输出位置等差异菜单项
4. 4/8 中的纵向输出必须共用一个函数，用参数解决输出位置等差异菜单项
5. 5/6/7/8/9 中画三个柱子的必须共用一个函数菜单项
6. 7/8/9 中盘子的移动必须共用一个函数

2. 整体设计思路

2.1. main函数

为了保证整个游戏的循环进行，将整个main函数设为一个永真循环，只有输入0时才能退出（break;）。每次循环第一步就是先清空屏幕然后调用menu函数进行输入提示。

2.2. menu函数

根据输入的不同start值调用不同的solution函数。同时进行清屏、退出等操作。

2.3. solution函数

首先为了符合题目要求，整体全部利用同样的全局变量，即一个一维数组以记录每根柱子的圆盘数（0代表A,1代表B,2代表C），一个二维数组记录圆盘编号，和一个整型变量step来记录步数。

不同输出功能的实现：根据不同start值来设置不同参数，调用不同语句和不同函数，以实现不同功能。因此solution中的大多数功能函数都需要输入solution对应的编号（即start值）。

由于输入要求只用一个函数，但是一个函数只能返回一个值，所以采用指针方式，直接改变指针变量所指地址的值。

无论是否用到，每个solution对应的函数在最开始必须进行所有全局变量的初始化，以实现主函数的多次调用。

每个功能分开实现，以在每个solution中更加方便地调用，且减少代码行数，方便维护。

各函数中参数大多使用宏中定义的变量名而非定值，也是方便维护。

3. 主要功能的实现

3.1.动画函数

分为两步：（1）Y坐标改变，在新位置打印盘子；（2）在原位置打印柱子，如果是平移则用底色填充。在打印过程中加入延时Sleep（speed），以实现动画。

3.2.横向+纵向打印函数

先全部初始化为0，再根据输入的盘子数进行初始输入。当盘子移到另一组时，将原位置赋零。打印时，非零则打印数字，为零则打印空格覆盖。

3.3.输入函数

为了同时改变多个值，传入盘子数n、起始柱src，目标柱dst和速度speed的地址，进行输入时采用同名指针以直接改变相应地址对应的值。输入函数依旧采用永真循环，在输入正确时跳出循环。

3.4.设定盘子颜色函数

根据题目设定的几种参数，每次绘制盘子前直接调用这个函数，传入盘子编号，以改变画笔颜色。

4. 调试过程碰到的问题

问题1：参数太长了，而且每次都要重新查找柱子的特定坐标很麻烦
解决方法：引入函数int get_peg_center(char peg)，根据输入参数直接返回相应坐标值。

问题2：hanoi主函数没法缩到15行
解决方法：再新写一个专用的输出总函数output_final()。

问题3：动画演示时屏幕闪烁
没有解决。

问题4：整合时发现前面的功能2、3中，当输入n个盘子时会输出n-1个盘子的移动过程，而且整体柱子编号错位
解决方法：最后调试发现是把其他行的函数直接复制过来时，参数和参数顺序没有改。

问题4：调试到最后发现在进行到第三步时会莫名其妙出现一个新盘子（比如输入2个盘子，在第3步时突然凭空出现第3个盘子并且移动到较小的盘子上面），但是通过前四个solution的调试发现其他功能函数本身逻辑没有问题

解决方法：最后发现是一开始在写solution8对应函数的时候直接先调用了一次show_status()函数并且step++，然后再进入hanoi_final()导致整体逻辑混乱。

5. 心得体会

1. 定义函数再反复调用真的很香，方便看，方便改，还方便整理逻辑
2. 在几个函数之间把参数传来传去的时候要注意顺序，注意相同名字在不同函数中代表的不同含义（尤其是在汉诺塔这种重在参数顺序的函数逻辑中）
3. 一定要认真看完要求，这次把一堆函数整合成一个整了好久还调试了半天，明明一开始就应该整理好所有思路直接写一个的，为了偷懒直接复制了好多结果自食恶果

6. 附件：源程序

```
#include"hanoi.h"
#include"cmd_hdc_tools.h"
#include"cmd_console_tools.h"
#include <iostream>
#include<iomanip>
#include <cstdio>
#include<conio.h>
#include<limits>


using namespace std;


int pole_count[3] = { 0 };        // 记录每根柱子的圆盘数（0A,1B,2C）
int pole_disks[3][10];  // 记录圆盘编号
int step = 0;

void get_hanoi_input(int* n, char* src, char* dst, int* speed, int mode)
{
    // 输入层数
    while (true) {
        cout << "请输入汉诺塔的层数(1-10)" << endl;
        cin >> *n;
        if (cin.good() && *n >= 1 && *n <= 10) {
            std::cin.ignore((std::numeric_limits<std::streamsize>::max)(), '\n');
            break;
        }
        else {
            cin.clear();

            std::cin.ignore((std::numeric_limits<std::streamsize>::max)(), '\n');
        }
    }

    // 输入起始柱
    while (true) {
        cout << "请输入起始柱(A-C)" << endl;
        cin >> *src;
        if (cin.good() && (*src == 'A' || *src == 'a' || *src == 'B' || *src == 'b' || *src == 'C' || *src == 'c')) {

            std::cin.ignore((std::numeric_limits<std::streamsize>::max)(), '\n');
            break;
        }
        else {
            cin.clear();
```

```
    std::cin.ignore((std::numeric_limits<std::st
reamsize>::max)(), '\n');
        }
    }

    // 转换为大写
    if (*src >= 'a' && *src <= 'c') {
        *src -= 32;
    }

    // 输入目标柱
    while (1) {
        cout << "请输入目标柱(A-C)" << endl;
        cin >> *dst;
        if (cin.good() && *dst != *src && (*dst
== 'A' || *dst == 'a' || *dst == 'B' || *dst ==
'b' || *dst == 'C' || *dst == 'c')) {

    std::cin.ignore((std::numeric_limits<std::st
reamsize>::max)(), '\n');
            break;
        }
        else if (cin.good() && *dst == *src) {
            cout << "目标柱(" << *src << ")不能
与起始柱(" << *src << ")相同" << endl;

    std::cin.ignore((std::numeric_limits<std::st
reamsize>::max)(), '\n');
        }
        else {
            cin.clear();

    std::cin.ignore((std::numeric_limits<std::st
reamsize>::max)(), '\n');
        }
    }

    // 转换为大写
    if (*dst >= 'a' && *dst <= 'c') {
        *dst -= 32;
    }

    // 输入速度
    if (mode == 4 || mode == 5 || mode == 6 ||
mode == 7 || mode == 8 || mode == 9) {
        while (true) {
            cout << "请输入移动速度(0-20：0-按回
车单步演示 1-20:延时 1-20ms)" << endl;
            cin >> *speed;
            if (cin.good() && *speed >= 0 &&
*speed <= 20) {

    std::cin.ignore((std::numeric_limits<std::st
reamsize>::max)(), '\n');
                break;
            }
            else {
                cin.clear();

    std::cin.ignore((std::numeric_limits<std::st
reamsize>::max)(), '\n');
            }
        }
    }
}


void my_exit()
{

    cout << "按回车键继续" << endl;
    char k = _getch();
    while (k != '\n' && k != '\r')
        ;
}

// 柱子中心 X 坐标
int get_peg_center(char peg) {
    int x = HDC_Start_X + 23 * HDC_Base_Width / 2;
    if (peg == 'B') x += 23 * HDC_Base_Width +
HDC_Underpan_Distance;
    else if (peg == 'C') x += 2 * (23 *
HDC_Base_Width + HDC_Underpan_Distance);
    return x;
}


//设定盘子画笔颜色
void disk_color(int n)
{
    if (n == 1)
        hdc_set_pencolor(0, 55, 218);
    else if (n == 2)
        hdc_set_pencolor(19, 161, 14);
    else if (n == 3)
        hdc_set_pencolor(58, 150, 221);
    else if (n == 4)
        hdc_set_pencolor(197, 15, 31);
    else if (n == 5)
        hdc_set_pencolor(136, 23, 152);
    else if (n == 6)
        hdc_set_pencolor(193, 156, 0);
    else if (n == 7)
        hdc_set_pencolor(204, 204, 204);
    else if (n == 8)
        hdc_set_pencolor(118, 118, 118);
    else if (n == 9)
        hdc_set_pencolor(59, 120, 255);
    else if (n == 10)
        hdc_set_pencolor(22, 198, 12);

}

//内部数组显示(横向+纵向)
```

```
void show_status(char from = 0, char to = 0, int
disk = 0, int speed = 10, int mode = 3) {
    const int X = 10, Y = 29;

    if (step == 0) {
        cct_gotoxy(0, 41);
        cout << left << setw(17) << "初始 : ";
    }
    // 显示步骤信息
    if (step > 0) {
        // 处理暂停和光标位置
        if (mode == 7 || mode == 8 || mode == 9)
{
            Sleep(speed);
            cct_gotoxy(0, 41);
        }
        else if (mode == 4) {
            if (speed != 0) {
                Sleep(speed);
                cct_gotoxy(0, 41);
            }
            else {
                cct_gotoxy(0, 41);
                while (_getch() != 13)
                    ;
            }
        }

        cout << "第 " << setw(3) << step << " 步
(" << setw(2) << disk << "): "
            << from << "-->" << to << " ";
    }

    // 横向显示
    if (mode == 7 || mode == 8 || mode == 4 ||
mode == 9) {
        cct_gotoxy(30, 41);
    }
    for (int peg = 0; peg < 3; peg++) {
        cout << char('A' + peg) << " : ";
        for (int level = 0; level <= 9; level++)
{
            if (pole_disks[peg][level] != 0)
                cout << pole_disks[peg][level]
<< " ";
            else
                cout << "  ";
        }
    }
    cout << endl;

    // 纵向显示
    if (mode != 3)
    {
        for (int i = 0; i < 10; i++) {
            cct_gotoxy(X, Y - i);
            if (pole_disks[0][i] != 0)
                cout << pole_disks[0][i];
            else
                cout << "  ";

            cct_gotoxy(X + 10, Y - i);

            if (pole_disks[1][i] != 0)
                cout << pole_disks[1][i];
            else
                cout << "  ";

            cct_gotoxy(X + 20, Y - i);
            if (pole_disks[2][i] != 0)
                cout << pole_disks[2][i];
            else
                cout << "  ";
        }
    }
    if (step == 0)
        Sleep(HDC_Init_Delay);
}

void move_disk_with_status(char from, char to, int
speed, int mode) {

    int from_idx = from - 'A';
    int to_idx = to - 'A';
    int                 disk                =
pole_disks[from_idx][pole_count[from_idx] - 1];

    // 1. 显示移动提示（盘子还在原位）
    if (mode == 7 || mode == 8 || mode == 9) {
        Sleep(HDC_Init_Delay);
    }

    // 2. 从 src 移除盘子
    pole_disks[from_idx][--pole_count[from_idx]]
= 0;


    // 3. 盘子已在目标位置

   pole_disks[to_idx][pole_count[to_idx]] = disk;
    show_status(from, to, disk, speed, mode);

    if (mode == 7 || mode == 8 || mode == 9)
    {
        // 4. 动画
        int disk_size = disk;
        int from_x = get_peg_center(from);
        int to_x = get_peg_center(to);
        int  width  =  (1  +  2  *  disk_size)  *
HDC_Base_Width;

        // 上升
        int current_x = from_x - width / 2;
        int    current_y    =    HDC_Start_Y    -
```

```
(pole_count[from_idx] + 1) * HDC_Base_High;
        while (current_y > HDC_Top_Y) {
            hdc_rectangle(current_x, current_y,
width, HDC_Base_High, RGB(12, 12, 12));
            hdc_rectangle(from_x -
HDC_Base_Width / 2, current_y, HDC_Base_Width,
HDC_Base_High, RGB(249, 241, 165));
            current_y -= HDC_Step_Y;
            disk_color(disk_size);
            hdc_rectangle(current_x, current_y,
width, HDC_Base_High, INVALID_RGB);

            // 控制步骤显示和暂停
            if (speed == 0 && (step != 0 || mode
== 9)) {
                while (_getch() != 13)
                    ;
            }
            else if (step != 0) {
                if (step <= 7)
                    Sleep(speed);
            }
        }

        // 水平移动
        while (current_x != to_x - width / 2) {
            hdc_rectangle(current_x, current_y,
width, HDC_Base_High, RGB(12, 12, 12));
            if (current_x < to_x - width / 2)
current_x += HDC_Step_X;
            else current_x -= HDC_Step_X;
            disk_color(disk_size);
            hdc_rectangle(current_x, current_y,
width, HDC_Base_High, INVALID_RGB);

            // 控制步骤显示和暂停
            if (speed == 0 && (step != 0 || mode
== 9)) {
                while (_getch() != 13)
                    ;
            }
            else if (step != 0) {
                if (step <= 7)
                    Sleep(speed);
            }
        }

        // 下降
        int target_y = HDC_Start_Y -
(pole_count[to_idx] + 1) * HDC_Base_High;
        while (current_y < target_y) {
            hdc_rectangle(current_x, current_y,
width, HDC_Base_High, RGB(12, 12, 12));
            if (current_y > HDC_Top_Y +
HDC_Base_High)
                hdc_rectangle(to_x -
HDC_Base_Width / 2, current_y, HDC_Base_Width,
HDC_Base_High, RGB(249, 241, 165));
            current_y += HDC_Step_Y;
            //盘子
            disk_color(disk_size);
            hdc_rectangle(current_x, current_y,
width, HDC_Base_High, INVALID_RGB);
            if (speed == 0 && (step != 0 || mode
== 9)) {
                while (_getch() != 13)
                    ;
            }
            else if (step != 0) {
                if (step <= 7)
                    Sleep(speed);
            }
        }
    }

    // 5. 正式更新高度
    pole_disks[to_idx][pole_count[to_idx]++] =
disk;


    // 6. 显示最终状态
    if (mode == 7 || mode == 8 || mode == 9) {
        show_status(from, to, disk, speed, mode);
    }
}

void output_final(int n, char src, char dst, int
speed, int mode)
{
    if (mode == 8 || mode == 7)
        move_disk_with_status(src, dst, speed,
mode);
    else if (mode == 2)
        cout << setw(5) << step << ": " << setw(2)
<< n << '#' << ' ' << src << "-->" << dst << endl;
    else if (mode == 1)
        cout << setw(2) << n << '#' << ' ' << src
<< "-->" << dst << endl;
    else if (mode == 3 || mode == 4)
        move_disk_with_status(src, dst, speed,
mode);
}



void hanoi_final(int n, char src, char dst, char
tmp, int speed, int mode)
{
    if (n == 1) {

        step++;
        output_final(n, src, dst, speed, mode);
    }
```

```
    else {
        hanoi_final(n - 1, src, tmp, dst, speed,
mode);
        step++;

        output_final(n, src, dst, speed, mode);

        hanoi_final(n - 1, tmp, dst, src, speed,
mode);
    }
}



//1. 基本解

void basic1()
{
    int n, speed;
    char src, dst;
    step = 0;



    get_hanoi_input(&n, &src, &dst, &speed, 1);

    char tmp = 198 - src - dst;
    hanoi_final(n, src, dst, tmp, speed, 1);

    my_exit();
}




//2. 基本解(步数记录)

void basic2_steps()
{
    step = 0;
    int n, speed;
    char src, dst;



    get_hanoi_input(&n, &src, &dst, &speed, 2);

    char tmp = 198 - src - dst;
    hanoi_final(n ,  src,dst, tmp, speed, 2);

    my_exit();
}




//3. 横向
void inside1()
{
    step = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 10; j++) {
            pole_disks[i][j] = 0;
        }
    }


    for (int i = 0; i < 3; i++) {
        pole_count[i] = 0;
    }


    // 清空

    int n, speed;
    char src, dst;


    get_hanoi_input(&n, &src, &dst, &speed, 3);

    char tmp = 198 - src - dst;
    int src_pole = src - 'A';
    for (int i = n; i >= 1; i--) {

    pole_disks[src_pole][pole_count[src_pole]++]
= i;
    }



    hanoi_final(n, src, dst, tmp, speed, 3);

    my_exit();

}




//4. 内部数组显示(纵向+横向)

void inside2()
{
    int n, speed;
    char src, dst;

    get_hanoi_input(&n, &src, &dst, &speed, 4);

    // 初始化全局变量
    step = 0;
    for (int i = 0; i < 3; i++) {
        pole_count[i] = 0;
        for (int j = 0; j < 10; j++) {
            pole_disks[i][j] = 0;
        }
    }
```

```
    // 初始化源柱子
    int src_idx = src - 'A';
    for (int i = n; i >= 1; i--) {

    pole_disks[src_idx][pole_count[src_idx]++] = i;
    }

    cct_cls();
    cout << "从" << src << "移动到" << dst << ", 共" << n << "层, " << "延时设置为" << speed << "ms";

    cct_gotoxy(8, 30);
    cout << "========================" << endl;

    cout << "          A         B         C  ";

    cct_gotoxy(3, 25);
    cout << left << setw(21) << "初始:";
    show_status(src, dst, n, speed, 4);

    // 开始移动
    char tmp = 198 - src - dst;
    hanoi_final(n, src, dst, tmp, speed, 4);

    my_exit();
}
```

```
//5.图形解-预备-画三个圆柱
void column_only()
{
    hdc_init();
    hdc_set_pencolor(249, 241, 165);//柱子的黄色

    cct_gotoxy(0, 40);
    Sleep(HDC_Init_Delay);


    hdc_rectangle(HDC_Start_X - 6, HDC_Start_Y, 25 * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
    Sleep(HDC_Init_Delay);
    hdc_rectangle(HDC_Start_X + HDC_Underpan_Distance + 23 * HDC_Base_Width, HDC_Start_Y, 23 * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
    Sleep(HDC_Init_Delay);
    hdc_rectangle(HDC_Start_X + 2 * (HDC_Underpan_Distance + 23 * HDC_Base_Width), HDC_Start_Y, 23 * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
    Sleep(HDC_Init_Delay);
    for (int i = 0; i < 3; i++) {
        int x = get_peg_center('A' + i);
        hdc_rectangle(x - HDC_Base_Width / 2, HDC_Base_High + 24,
            HDC_Base_Width, 230, INVALID_RGB);
        Sleep(HDC_Init_Delay);
    }


}
```

```
//6. 图形解-预备-在起始柱上画 n 个盘子
void disk_only()
{
    int n, speed;
    char src, dst;


    get_hanoi_input(&n, &src, &dst, &speed, '6');

    char tmp = 198 - src - dst;

    cct_cls();
    cct_gotoxy(0, 38);
    cout << "从" << src << "移动到" << dst << ", 共" << n << "层" << endl;

    column_only();

    if (src == 'A') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }
    else if (src == 'B') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 + HDC_Underpan_Distance + 23 * HDC_Base_Width - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }
    else if (src == 'C') {
```

```
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 + HDC_Underpan_Distance + 23 * HDC_Base_Width + HDC_Underpan_Distance + 23 * HDC_Base_Width - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }


}



//8.图形解 - 自动移动版本

void hanoi_automove()
{
    step = 0;
    // 输入
    int n, speed;
    char src, dst;

    get_hanoi_input(&n, &src, &dst, &speed, 8);

    char tmp = 198 - src - dst; // ASCII 码计算中间柱子 (A=65, B=66, C=67 -> 65+66+67=198)

    cct_cls();

    // 初始化
    int src_idx = src - 'A';
    pole_count[0] = pole_count[1] = pole_count[2] = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 10; j++) {
            pole_disks[i][j] = 0;
        }
    }

    // 根据源柱子初始化 disk_stacks
    for (int i = 0; i < n; i++) {
        pole_disks[src_idx][i] = n - i; // 底部放最大的盘子
    }
    pole_count[src_idx] = n;

    column_only();
    if (src == 'A') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }
    else if (src == 'B') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 + HDC_Underpan_Distance + 23 * HDC_Base_Width - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }
    else if (src == 'C') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 + HDC_Underpan_Distance + 23 * HDC_Base_Width + HDC_Underpan_Distance + 23 * HDC_Base_Width - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }
    // 显示初始状态
    show_status(src, dst, n, speed, 8);

    cct_gotoxy(0, 39);
    cout << "从" << src << "移动到" << dst << ", 共" << n << "层, " << "延时设置为" << speed << "ms" << "（前 7 步，后面自动变为 0ms）";

    cct_gotoxy(8, 30);
    cout << "=========================" << endl;
    cout << "          A         B         C  ";




    // 开始移动
    hanoi_final(n, src, dst, tmp, speed, 8);
}
```

```
//7.一步
void hanoi_firststep()
{
    step = 0;
    int n, speed;
    char src, dst;


    get_hanoi_input(&n, &src, &dst, &speed, 7);

    char tmp = 198 - src - dst;


    cct_cls();


    // 初始化
    int src_idx = src - 'A';
    pole_count[0] = pole_count[1] = pole_count[2] = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 10; j++) {
            pole_disks[i][j] = 0;
        }
    }

    // 根据源柱子初始化 disk_stacks
    for (int i = 0; i < n; i++) {
        pole_disks[src_idx][i] = n - i;
    }
    pole_count[src_idx] = n;

    column_only();

    if (src == 'A') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }
    else if (src == 'B') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 + HDC_Underpan_Distance + 23 * HDC_Base_Width - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }
    else if (src == 'C') {
        for (int i = n;i > 0;i--) { //倒着画 n 个
            disk_color(i);
            hdc_rectangle(HDC_Start_X + 23 * HDC_Base_Width / 2 + HDC_Underpan_Distance + 23 * HDC_Base_Width + HDC_Underpan_Distance + 23 * HDC_Base_Width - (1 + 2 * i) * HDC_Base_Width / 2 + 1, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, INVALID_RGB);
            Sleep(HDC_Init_Delay);
        }
    }


    src_idx = src - 'A';
    for (int i = 0; i < n; i++) {
        pole_disks[src_idx][i] = n - i; // 底部放最大的盘子
    }
    pole_count[src_idx] = n;

    cct_gotoxy(0, 39);
    cout << "从" << src << "移动到" << dst << ",共" << n << "层," << "延时设置为" << speed << "ms" << "（前 7 步，后面自动变为 0ms）";


    cct_gotoxy(8, 30);
    cout << "========================" << endl;

    cout << "          A         B         C  ";


    Sleep(HDC_Init_Delay);


    // 开始移动
    show_status(src, dst, n, speed, 7);
    step++;
    move_disk_with_status(src, dst, speed, 7);
}


//9.自助
void hanoi_own() {
    //初始化
```

```
    step = 0;
    int n, speed;
    char src, dst;


    get_hanoi_input(&n, &src, &dst, &speed, 9);

    char tmp = 198 - src - dst;

    cct_cls();


    int src_idx = src - 'A';
    pole_count[0] = pole_count[1] = pole_count[2]
= 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 10; j++) {
            pole_disks[i][j] = 0;
        }
    }
    for (int i = 0; i < n; i++) {
        pole_disks[src_idx][i] = n - i;
    }
    pole_count[src_idx] = n;


    column_only();




    // 绘制初始盘子
    for (int i = n; i > 0; i--) {
        disk_color(i);
        int x = get_peg_center(src);
        int width = (1 + 2 * i) * HDC_Base_Width;
        hdc_rectangle(x - width / 2 + 1,
HDC_Start_Y - (n - i + 1) * HDC_Base_High, width,
HDC_Base_High, INVALID_RGB);
        Sleep(HDC_Init_Delay);
    }



    cct_gotoxy(8, 30);
    cout << "=======================" << endl;

    cout << "          A         B         C  ";

    cct_gotoxy(0, 41);
    cout << left << "初始 : ";
    // 4. 显示初始状态
    show_status(src, dst, n, speed, 9);


    while (true) {
        cct_gotoxy(0, 42);
        cout << "请输入移动的柱号(命令形式：AC=A
顶端的盘子移动到C，Q=退出) ：";

        char input[3];
        cin >> input;
        cct_gotoxy(60, 42);
        cout << "  ";

        input[0] = toupper(input[0]);
        input[1] = toupper(input[1]);
        // 退出条件
        if (toupper(input[0]) == 'Q') {
            cout << "游戏中止!!!!!";
            break;
        }

        // 输入处理
        if (strlen(input) < 2 || !(input[0] >=
'A' && input[0] <= 'C') || !(input[1] >= 'A' &&
input[1] <= 'C')) {
            cct_gotoxy(0, 43);
            cout << "  ";
            continue;
        }

        char from = input[0];
        char to = input[1];

        if (pole_count[from - 'A'] == 0) {
            continue;
        }

        if (pole_count[to - 'A'] > 0 &&
pole_disks[from - 'A'][pole_count[from - 'A'] - 1] >
pole_disks[to - 'A'][pole_count[to - 'A'] - 1]) {
            continue;
        }



        // 移动
        step++;
        move_disk_with_status(from, to, speed,
9);

        // 完成
        if (pole_count[dst - 'A'] == n) {
            break;
        }
    }
}
```