以汉诺塔综合功能的实现为例研究如何设计实现复杂的多功能代码

班级：10071706
学号：2452654
姓名：郭炫君
完成日期：2025.5.23

1. 题目

1.1. 基本要求

要把之前的汉诺塔各种功能包括记录步数，打印内部数组，纵向打印圆盘等集成到一个程序中，并加入图形化演示的功能。通过菜单选择的方式来实现对应功能。

程序需要包含 10 个功能选项，分别是

1.基本解

2.带步数记录的基本解

3.带步数记录，横向显示内部数组

4.在 3 的基础上，加上纵向显示内部数组

5.画出三根圆柱

6.在起始柱上画出 n 个圆盘

7.画出第一次圆盘移动的全过程

8.图形解的自动移动版本

9.图形解的游戏版，输入起始柱和目标柱，画出相应的移动过程，当圆盘全部移动完成后，游戏结束，同时，不能出现起始柱是空柱，大盘压小盘等问题。

10.退出程序

1.2. 图形化要求

盘子移动动画必须呈现为上移-平移-下移的过程，为了方便观察，需要增加延时，同时不允许在动画中出现图形闪烁的现象，视觉上要有连贯性。

1.3. 代码组织要求

1. 整个程序只能有一个递归函数，且不超过 15 行
2. 功能 3/4/8 横向输出数组用一个函数，功能 4/8 纵向输出数组用一个函数，功能 5/6/7/8/9 画三根圆柱用一个函数，功能 7/8/9 移动圆柱用一个函数。
3. 尽量让每个函数不超过 50 行

1.4. 允许使用的全局变量

- 总移动步数：1 个变量
- 圆柱上圆盘编号：最多 3 个一维数组或 1 个二维数组
- 圆柱上圆盘数量：3 个变量或 1 个一维数组

- 延时控制：1 个变量

1.5. 窗口界面要求

控制台设为 windows 控制台主机，字体大小为新宋体 16，窗口大小为 120*40，分辨率最好设置为 1920*1080，缩放 100%。

1.6. 特殊要求

1. 代表图形粗细，颜色和位置的参数需用头文件里定义的常量来表示，不要写死，方便放入其他定义常量参数的头文件来检查代码。
2. 输入时需要考虑输入错误。

2.整体设计思路

整个代码运用到了许多重复的功能，因此把相同的功能用一个函数来实现，减少了代码冗余，最后通过不同功能函数之间的组合来实现要求的功能。

2.1. 整体架构

1. cmd_console_tools.cpp 提供控制台的函数，cmd_hdc_tools.cpp 提供了图形绘制的函数
2. hanoi_const_value.h 集中定义了各种常量
3. Hanoi_main.cpp 包含了主循环和开启程序的条件
4. Hanoi_menu.cpp 用于展示菜单页面和确定选择哪一个功能
5. Hanoi_multi_solutions.cpp 包含了具体实现各种功能的函数
6. 头文件 hanoi.h 将各个源文件连接在一起

2.2. 主要功能模块

1. 在 cmd_console_tools.cpp 中 cct_cls 用于清屏，cct_gotoxy 用于光标定位。在 cmd_hdc_tools.cpp 中，hdc_init 用于图形初始化，hdc_cls 用于图形清屏，hdc_set_pencolor 用于设置画笔颜色，hdc_rectangle 用于绘制矩形。
2. 使用二维数组 disks[3][10]表示三根柱子上的圆盘，用 tops[3]来记录栈顶。
3. hanoi 递归函数实现基本功能，而 muti_function 通过调用参数实现不同功能。
4. print_tower 和 print_tower_vertical 用于打印内部数组。

5. Paint_disks 用于画三根圆柱，paint_pan 画出初始圆盘，paint_move_pan 画出圆盘移动的动画，输入圆盘数就移动对应的圆盘。

6. Solution_9 单独解决第九个游戏功能。

7. Basis 函数用于判断输入是否正确以及调用各个功能函数来实现每一种功能。

8. 剩余的函数都是为了功能的具体实现而设计的，比如 move_disk 用于更新移动后数组信息，clean 用于每一次功能实现后初始化数组信息等。

3.主要功能的实现

3.1. 功能 1

直接调用 hanoi 函数，输出每一步的基本移动信息。结束后使用 clean 函数初始化全局变量数组，后面的功能有用到数组更新的都需要在结束之后使用 clean 来进行初始化，这样才能在不退出程序运行的情况下再次使用其他功能。

3.2. 功能 2

在功能 1 的基础上增加了 step 用于计步数，调用 muti_function 中参数为 2 的部分，即可得到结果。

3.3. 功能 3

在功能 2 的基础上调用 print_tower 横向打印内部数组。

3.4. 功能 4

先让光标移动到指定位置，输入总体信息“从...移动到...，共...层，延时设置为...ms”，然后再次移动光标到事先设定好的位置打印初始横向数组和竖向数组，进行延时，然后调用 hanoi 函数的参数 4 部分。因为初始状态的打印和之后的循环打印有差异，所以把这一部分单拎出来写，使得 hanoi 函数对其他几个功能也适用，而不是单一服务于功能 4。由于不能每一次都清屏，在纵向打印时先在每一个圆盘处输入空格来更新，相当于对纵向打印的数组进行清屏，再纵向输入数组。

3.5. 功能 5

直接调用 paint_disks 函数，也就是先初始化背景，然后根据柱子大小和位置利用循环的方式画出底座和立柱。

3.6. 功能 6

先让光标移动到指定位置，输入总体信息“从...移动到...”，然后先画出三根柱子，再根据圆盘个数和初始柱信息利用循环来画出起始圆盘。

3.7. 功能 7

先执行功能 6 的内容，然后使用 move_disk 更新 1 次数组，然后调用 paint_move_pan 来进行一次移动，移动动画函数先计算出需要移动圆盘的起始、目标位置，大小信息，根据这些信息移动光标，先擦去原来位置一个步长高度，再在此处画出立柱，然后在移动后位置新增一个步长高度，而不是擦去整个画出整个，这样能从视觉上产生连续的动画效果。分为上移，平移和下移三个步骤，每个步骤终止条件是圆盘到达对应的位置。

3.8. 功能 8

将之前各种功能整合在一起即可，先把初始的各个部分单拎出来打印一遍，然后进入 hanoi 循环，先更新打印数组，然后画出圆盘移动的动画。功能 5 到 7 相当于对功能 8 的分解，功能 7 是对第一个圆盘进行移动，输入实参是 1，而功能 8 是输入 n 来移动第 n 个圆盘。

3.9. 功能 9

其他部分与功能 8 相似，但是单独使用函数 solution_9 来实现输入字母移动圆盘以及判断游戏结束的功能。首先清除上一次的输入，然后输入这一回的字母，函数中前半段都在判断输入是否正确，包括字母是不是 A 到 C、两者是否相同、是不是 Q、字母数量有没有超过 2 个、源柱是否为空、是否大盘压小盘等等。之后根据起始柱和目标柱信息打印数组和画动画，一直循环，直到移动完成循环结束。

4. 调试过程碰到的问题

4.1. 如何只使用一个递归函数实现不同的任务。

通过观察可以发现之前几个作业里的 hanoi 递归函数基本结构都是一致的，只有一句话不一样，因此可以把这句话提出来放到另一个函数里，通过不同参数调用这个函数不同的语句，实现不同的 hanoi 功能。

4.2. 如果把“从...移动到...，共...层” 放到 muti_function 函数里每一次递归都打印一遍，那么很多函数里就需要增加层数、起始柱、目标柱三个形参，大大增加了工作量。

直接在 basis 函数里面开头打印这句话，每一次移动圆盘时不清屏，而是现在纵向打印数组上面全部替换为空格再打印，横向数组直接打印即可在原来位置替换掉前一次的数组，这样最底下的基本信息就不会消失，只需要开头打印一遍即可。

4.3. 画圆盘移动动画的时候如果先擦去整个圆盘再在新的位置画出整个圆盘，动画会闪烁，视觉效果差。

擦去一个步进单位大小的矩形，再在新的位置画一个步进大小的矩形，这样能实现视觉连续性。

5.心得体会

5.1. 完成本次作业的经验教训

5.1.1. 模块化设计的重要性

这个程序体量很大，要完成 9 个功能，每个功能下又需要若干个小功能来完成，如果按照功能顺序一点一点写下来，不仅思路会很混乱，而且还很容易出现很多重复的代码，事倍功半。把每一个功能细分出来，一个小模块一个小模块完成，让我思路更加清晰，而且每完成一个小功能，就知道自己又更进了一步，让我写的时候更有动力。

5.1.2. 定义 const 参数而不是直接把数字写到代码里

在写这种大体量的代码时，会经常用到参数，如果把参数写死在代码里，后续修改会非常困难。虽然我一开始使用 const 参数时感觉很不习惯，有点麻烦，但是随着进度不断推进，我发现如果真的是一个一个数字往里面填写，很容易混淆，因为你可能会忘记这个参数的意义是什么，而变量名就能提醒你，还更方便记忆。

5.1.3. 在计算参数时要注意细节，比如是不是要+1，是<=还是<等等

在写圆盘移动函数的时候，需要计算圆盘终止位置的准确数值，而且判断终止条件的<=和<也需要考虑使用哪一种，一定要先仔细在脑中模拟一下整个过程，对边界数据的确定十分关键，我就是在边界数据的确定和终止条件判断上卡了很久，其实就是对某几个数据进行一个小调整，但是这对于结果而言有巨大的影响。

5.2. 在做复杂程序的时候是分为若干个小题好还是一个大题好？

显然是分为多个小题来完成更好。首先，这样能够降低程序的复杂程度，提高开发的效率，比如如果没有把控制窗口的 cct_cls 和 cct_gotoxy 分出来，那么在写需要控制窗口功能的时候就会非常麻烦，而分出来以后就可以明确地知道在这个位置是要用到控制窗口的功能。整体开发存在许多潜在问题，一个是调试苦难，由于所有功能模块都合在一起，很难分清楚到底是那个部分出了问题，还有一个是会发生“代码纠缠”，不同功能的代码全部写在一起会十分混乱，把输入、算法等部分全都混合在一起。同时，分多个小题完成在后期对程序进行维护时也更加方便。

5.3.

在做前面小题的时候，我有考虑后面小题与前面的关联，比如 hanoi 函数前后几个功能可以共用，只需要修改根据参数调用不同语句即可。想要更好地重复利用代码，在开始写代码之前就应该整体构思一下每一个功能大概需要怎么写，在设计函数的时候也要考虑后面的功能能不能再重复使用或者是只需要扩展一下，函数的形参数量会不会因为后面功能的需要而增加。要提早为后面的功能做规划，否则很有可能写道后面发现不适配，需要全部推翻重新写。

5.4. 如何更好地利用函数来编写复杂的程序

1. 尽量让每个函数都只完成单一的任务，因为把多个功能放到一个函数里面以后，这个函数可能就无法在其他功能里面被重复使用，而且代码阅读起来也会变得不易理解。
2. 要合理地分解复杂逻辑，变为多个子函数，就比如第 8 个功能就能被分解为横向打印内部数组、纵向打印数组、打印圆柱、打印起始圆盘以及移动圆盘等多个子函数的组合，方便阅读也易于后续修改调试。
3. 仔细观察各个语句之间的相似性，如果相同且重复，可以提取出来写成一个函数，就比如 hanoi 函数中的 muti_function，如果没有这个函数，就需要多写好几个函数用来实现汉诺塔的不同功能，工作量会大大增加。
4. 在每个函数前写上注释标明函数的用途，同时函数名应当指向其功能，便于记忆。

6. 源代码

```
#include <iostream>
#include <conio.h>
#include <iomanip>
#include <windows.h>
#include "cmd_console_tools.h"
#include "hanoi_const_value.h"
#include "cmd_hdc_tools.h"
using namespace std;
int tops[3] = { -1, -1, -1 };  // 三根柱子的栈顶指针
int disks[3][10];              // 三根柱子上的圆盘
int step = 0;          // 移动步数计数
int sleep;         //延时
//完成一个功能，重新开始
void clean()
{
    step = 0;
    for (int i = 0; i < 3; i += 1) {
        tops[i] = -1;  //这里变为-1，而不是0!
        for (int j = 0; j < 3; j += 1) {
            disks[i][j] = 0;
        }
    }
}
//处理延时
void delay_time(int sleep)
{
    if (sleep == 0) {
        while (_getch() != '\r');
     }
    else {
        Sleep(sleep);
    }
}
//移动圆盘
void move_disk(char from, char to)
{
    int from_index = from - 'A';
    int to_index = to - 'A';
    disks[to_index][tops[to_index]    +    1]    =
disks[from_index][tops[from_index]];
    tops[to_index] += 1;
    tops[from_index] -= 1;
}
//横向数组打印函数 功能3/4/8
void print_tower() //打印每根柱子圆盘
{
    for (int i = 0; i < 3; i += 1) {
        char zhuzi = 'A' + i;
        cout << " " << zhuzi << ":";
        int space = 20 - ((tops[i] + 1) * 2);
        for (int j = 0; j <= tops[i]; j += 1) {
            if (disks[i][j] == 10)
                cout << disks[i][j];
            else {
                cout << " " << disks[i][j];
            }
        }
        for (int i = 0; i < space; i++) {
            cout << " ";
        }
    }
    cout << endl;
}
//纵向打印函数,功能4/8
void print_tower_vertical(int choose)
{
    int const max_height = 9;
    for (int i = max_height; i >= 0; i -= 1){
        if (choose==4)
            cct_gotoxy(MenuItem4_Start_X+
Underpan_A_X_OFFSET,              MenuItem4_Start_Y+
Underpan_A_Y_OFFSET-i- 2);
        if (choose==8)
            cct_gotoxy(MenuItem8_Start_X          +
Underpan_A_X_OFFSET,       MenuItem8_Start_Y      +
Underpan_A_Y_OFFSET - i - 2);
        cout << " ";
        if (choose==4)
            cct_gotoxy(MenuItem4_Start_X          +
Underpan_A_X_OFFSET+             Underpan_Distance,
MenuItem4_Start_Y + Underpan_A_Y_OFFSET -i-2);
        if (choose==8)
            cct_gotoxy(MenuItem8_Start_X          +
Underpan_A_X_OFFSET      +       Underpan_Distance,
MenuItem8_Start_Y + Underpan_A_Y_OFFSET - i - 2);
        cout << " ";
        if (choose == 4)
            cct_gotoxy(MenuItem4_Start_X          +
Underpan_A_X_OFFSET     +     2*Underpan_Distance,
MenuItem4_Start_Y + Underpan_A_Y_OFFSET -i-2);
        if (choose == 8)
            cct_gotoxy(MenuItem8_Start_X          +
Underpan_A_X_OFFSET   +   2   *   Underpan_Distance,
MenuItem8_Start_Y + Underpan_A_Y_OFFSET - i - 2);
        cout << " ";
    }
    for (int i = max_height; i >= 0; i -= 1) {
        if (choose==4)
            cct_gotoxy(MenuItem4_Start_X          +
Underpan_A_X_OFFSET,       MenuItem4_Start_Y      +
Underpan_A_Y_OFFSET - i - 2);
        if (choose==8)
            cct_gotoxy(MenuItem8_Start_X          +
Underpan_A_X_OFFSET,       MenuItem8_Start_Y      +
Underpan_A_Y_OFFSET - i - 2);
        if (i <= tops[0]) {
            cout << disks[0][i];
        }
```

```
        if (choose==4)
            cct_gotoxy(MenuItem4_Start_X + Underpan_A_X_OFFSET + Underpan_Distance, MenuItem4_Start_Y + Underpan_A_Y_OFFSET - i - 2);
        if (choose==8)
            cct_gotoxy(MenuItem8_Start_X + Underpan_A_X_OFFSET + Underpan_Distance, MenuItem8_Start_Y + Underpan_A_Y_OFFSET - i - 2);
        if (i <= tops[1]) {
            cout << disks[1][i];
        }
        if (choose==4)
            cct_gotoxy(MenuItem4_Start_X + Underpan_A_X_OFFSET + 2 * Underpan_Distance, MenuItem4_Start_Y + Underpan_A_Y_OFFSET - i - 2);
        if (choose==8)
            cct_gotoxy(MenuItem8_Start_X + Underpan_A_X_OFFSET + 2 * Underpan_Distance, MenuItem8_Start_Y + Underpan_A_Y_OFFSET - i - 2);
        if (i <= tops[2]) {
            cout << disks[2][i];
        }
    }
    if (choose==4)
        cct_gotoxy(MenuItem4_Start_X + Underpan_A_X_OFFSET-2, MenuItem4_Start_Y + Underpan_A_Y_OFFSET-1);
    if (choose==8)
        cct_gotoxy(MenuItem8_Start_X + Underpan_A_X_OFFSET - 2, MenuItem8_Start_Y + Underpan_A_Y_OFFSET - 1);
    for (int i = 0; i <= Underpan_Distance * 2 + 4; i += 1) {
        cout << "=";
    }
    for (int i = 0; i < 3; i += 1) {
        if (choose==4)
            cct_gotoxy(MenuItem4_Start_X + Underpan_A_X_OFFSET+i* Underpan_Distance, MenuItem4_Start_Y + Underpan_A_Y_OFFSET);
        if (choose==8)
            cct_gotoxy(MenuItem8_Start_X + Underpan_A_X_OFFSET + i * Underpan_Distance, MenuItem8_Start_Y + Underpan_A_Y_OFFSET);
        cout << (char)('A' + i);
    }
}
//画出三根柱子
void paint_disks()
{
    const int   win_width = 1400, win_high = 900;
    const int   win_bgcolor = 7, win_fgcolor = 0;
    hdc_init(win_bgcolor, win_fgcolor, win_width, win_high);
    hdc_cls();
    for (int i = 0; i < 3; i += 1) {
        hdc_rectangle(HDC_Start_X + (HDC_Underpan_Distance + 23 * HDC_Base_Width) * i, HDC_Start_Y, 23 * HDC_Base_Width, HDC_Base_High,HDC_COLOR[MAX_LAYER + 1]);
        Sleep(HDC_Init_Delay);
    } //三根基座
    for (int i = 0; i < 3; i += 1) {
        hdc_rectangle(HDC_Start_X + (HDC_Underpan_Distance + 23 * HDC_Base_Width) * i + 11 * HDC_Base_Width, HDC_Start_Y - 12 * HDC_Base_High, HDC_Base_Width, 12 * HDC_Base_High, HDC_COLOR[MAX_LAYER + 1]);
        Sleep(HDC_Init_Delay);
    }//三根立柱
}
//画出起始的圆盘
void paint_pan(int n,char src)
{
    for (int i = n; i > 0; i -= 1) {
        hdc_rectangle(HDC_Start_X + (HDC_Underpan_Distance + 23 * HDC_Base_Width) * (int)(src - 'A') + (10 - i+1) * HDC_Base_Width, HDC_Start_Y - (n - i + 1) * HDC_Base_High, (1 + 2 * i) * HDC_Base_Width, HDC_Base_High, HDC_COLOR[i]);
        Sleep(HDC_Init_Delay);
    }
}
//画出圆盘移动过程
void paint_move_pan(int n, char from, char to)
{
    //盘子信息
    int width_pan = (1 + 2 * n) * HDC_Base_Width;
    int height_pan = HDC_Base_High;
    //起始柱信息
    int rank_from = from - 'A';//在第几根柱子上，从0开始
    int rank_pan_from = tops[rank_from]+2 ;//盘子叠在第几个,先使用 move_disk 函数，需要再加一
    int X_pan_from = HDC_Start_X + (HDC_Underpan_Distance + 23 * HDC_Base_Width) * rank_from + (11 - n ) * HDC_Base_Width;
    int Y_pan_from = HDC_Start_Y - rank_pan_from * HDC_Base_High;
    //目标柱信息
    int rank_to = to - 'A';//在第几根柱子上，从0开始
    int rank_pan_to = tops[rank_to] +1;
    int X_pan_to = HDC_Start_X + (HDC_Underpan_Distance + 23 * HDC_Base_Width) * rank_to + (11 - n) * HDC_Base_Width;
    int Y_pan_to = HDC_Start_Y - rank_pan_to * HDC_Base_High;
    //上移动画
    for (int i = Y_pan_from; i > HDC_Top_Y; i -= HDC_Step_Y) {
        hdc_rectangle(X_pan_from, i+ height_pan- HDC_Step_Y, width_pan, HDC_Step_Y, HDC_COLOR[0]);
```

```
        if (i+height_pan-HDC_Step_Y>= HDC_Start_Y - 12 * HDC_Base_High)
            hdc_rectangle(X_pan_from+ HDC_Base_Width*n,  i+height_pan  -  HDC_Step_Y, HDC_Base_Width, HDC_Step_Y, HDC_COLOR[MAX_LAYER + 1]);
        hdc_rectangle(X_pan_from,  i-HDC_Step_Y, width_pan, HDC_Step_Y, HDC_COLOR[n]);
        if (step <= 7)
            delay_time(sleep);
    }
    //左右移动
    int  step_X  =  (X_pan_to  >  X_pan_from)  ? HDC_Step_X : -HDC_Step_X;
    for (int i = X_pan_from; i != X_pan_to; i += step_X) {
        if (step_X > 0) { // 向右移动
            hdc_rectangle(i, HDC_Top_Y, HDC_Step_X, height_pan, HDC_COLOR[0]);
            hdc_rectangle(i + width_pan, HDC_Top_Y, HDC_Step_X, height_pan, HDC_COLOR[n]);
        }
        else { // 向左移动
            hdc_rectangle(i + width_pan - HDC_Step_X, HDC_Top_Y, HDC_Step_X, height_pan, HDC_COLOR[0]);
            hdc_rectangle(i - HDC_Step_X, HDC_Top_Y, HDC_Step_X, height_pan, HDC_COLOR[n]);
        }
        if (step<=7)
            delay_time(sleep);
    }
    //下移动画
    for (int i = HDC_Top_Y; i < Y_pan_to; i += HDC_Step_Y) {
        //到最后一格的时候不用擦去上面，否则会少一格
        hdc_rectangle(X_pan_to,  i,  width_pan, HDC_Step_Y, HDC_COLOR[0]);
        if (i >= HDC_Start_Y - 12 * HDC_Base_High && i<Y_pan_to)//同理
            hdc_rectangle(X_pan_to + HDC_Base_Width *  n,  i,  HDC_Base_Width,  HDC_Step_Y, HDC_COLOR[MAX_LAYER + 1]);
        hdc_rectangle(X_pan_to,  i  +height_pan, width_pan, HDC_Step_Y, HDC_COLOR[n]);
        if (step <= 7)
            delay_time(sleep);
    }
}
//hanoi 各种功能函数
void muti_function(int n, char src, char tmp, char dst, int choose)
{
    if (choose == 1) {
        cout << n << "# " << src << "---->" << dst << endl;
        move_disk(src, dst);
    }
    if (choose == 2) {
        cout << "第" << setw(4) << step << " 步( " << n << "#: " << src << "-->" << dst << ")"<<endl;
        move_disk(src, dst);
    }
    if (choose == 3) {
        cout << "第" << setw(4) << step << " 步( " << n << "#: " << src << "-->" << dst << ")";
        move_disk(src, dst);
        print_tower();
    }
    if (choose == 4) {
        move_disk(src, dst);
        print_tower_vertical(4);
        cct_gotoxy(MenuItem4_Start_X, MenuItem4_Start_Y);
        cout << "第" << setw(4) << step << " 步(" << n << "#: " << src << "-->" << dst << ") ";
        print_tower();
        delay_time(sleep);
    }
    if (choose == 8) {
        move_disk(src, dst);
        print_tower_vertical(8);
        cct_gotoxy(MenuItem8_Start_X, MenuItem8_Start_Y);
        cout << "第" << setw(4) << step << " 步(" << n << "#: " << src << "-->" << dst << ") ";
        print_tower();
        paint_move_pan(n, src, dst);
    }
}
//hanoi 递归函数
void hanoi(int n, char src, char tmp, char dst,int choose)
{
    if (n == 1) {
        step += 1;
        muti_function(n, src, tmp, dst,choose);
        return;
    }
    hanoi(n - 1, src, dst, tmp,choose);
    step += 1;//把最大的圆盘移到目标柱
    muti_function(n, src, tmp, dst,choose);
    hanoi(n - 1, tmp, src, dst,choose);
}
void solution_9(int n, char src, char tmp, char dst)
{
    char ssrc;
    char sdst;
    while (1) {
        cct_gotoxy(60, Status_Line_Y - 1);
        for (int i = 0; i < 40; i += 1)
            cout << " ";
        cct_gotoxy(60, Status_Line_Y - 1);
        cin>>ssrc;
```

```
        if (cin.fail()) {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        ssrc = toupper(ssrc);
        if (ssrc < 'A' || ssrc > 'C') {
            cin.ignore(1024, '\n');
            continue;
        }
        else if (ssrc == 'Q') {
            cct_gotoxy(Status_Line_X,
Status_Line_Y);
            cout << "游戏中止!!!!!";
            break;
        }
        cin >> sdst;
        if (cin.fail()) {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        sdst = toupper(sdst);
        if (getchar() != '\n')
            continue;//字符数量>2
        if (sdst < 'A' || sdst > 'C' || ssrc == sdst)
{
            continue;
            cin.clear();
            cin.ignore(1024, '\n');
        }

        //检查移动合理性
        if (tops[ssrc - 'A'] == -1) {
            cct_gotoxy(Status_Line_X,
Status_Line_Y);
            cout << "源柱为空!";
            Sleep(1000);
            cct_gotoxy(Status_Line_X,
Status_Line_Y);
            for (int i = 0; i < 9; i += 1) {
                cout << " ";
            }
            continue;
        }
        if (disks[ssrc - 'A'][tops[ssrc - 'A']] >
disks[sdst - 'A'][tops[sdst - 'A']] && tops[sdst -
'A']!=-1) {
            cct_gotoxy(Status_Line_X,
Status_Line_Y);
            cout << "大盘压小盘，非法移动！";
            Sleep(1000);
            cct_gotoxy(Status_Line_X,
Status_Line_Y);
            for (int i = 0; i < 22; i += 1) {
                cout << " ";
            }
            continue;
        }
        move_disk(ssrc, sdst);
        print_tower_vertical(8);
        step += 1;
        cct_gotoxy(MenuItem8_Start_X,
MenuItem8_Start_Y);
        cout << "第" << setw(4) << step << " 步(" <<
n << "#: " << ssrc << "-->" << sdst << ") ";
        print_tower();
        paint_move_pan(disks[sdst - 'A'][tops[sdst -
'A']], ssrc, sdst);
        if (tops[dst - 'A'] == n-1) {
            cct_gotoxy(Status_Line_X,
Status_Line_Y);
            cout <<"游戏结束";
            break;
        }
    }
}

int basis(int choose)//主函数
{
    if (choose == 5) {
        cct_cls();
        paint_disks();
        cct_gotoxy(Status_Line_X, Status_Line_Y+1);
        cout << "按回车键继续";
        while (_getch() != '\r');
        return 0;
    }
    //计算中间柱的值
    char tmp = 'A' + 'B' + 'C' - start - ending;

    //对起始柱初始化
    int start_index = start - 'A';
    for (int i = 0; i < layer; i += 1) {
        disks[start_index][i] = layer - i;
    }
    tops[start_index] = layer - 1;

    //功能 4/8 的速度选择
    if (choose == 4) {
        while (1) {
            cout <<"(请输入移动速度(0-200：0-按回车
单步演示 1-200:延时 1-200ms)";
            cin >> sleep;
            if (cin.good()) {
                if (sleep >= 0 && sleep <= 200) {
                    break;
                }
                else
                    continue;
            }
            cin.clear();
            cin.ignore(1024, '\n');
        }
```

```
        cin.ignore(1024, '\n');
    }

    if (choose == 7 || choose==8 || choose==9) {
        while (1) {
            cout << "(请输入移动速度(0-20：0-按回车单步演示 1-20:延时 1-20ms)";
            cin >> sleep;
            if (cin.good()) {
                if (sleep >= 0 && sleep <= 20) {
                    break;
                }
                else
                    continue;
            }
            cin.clear();
            cin.ignore(1024, '\n');
        }
        cin.ignore(1024, '\n');
    }

    //菜单参数
    switch (choose) {
        case 1:
            hanoi(layer, start, tmp, ending, 1);
            clean();
            cout << endl;
            cout << "按回车键继续";
            while (_getch() != '\r');
            break;
        case 2:
            hanoi(layer, start, tmp, ending, 2);
            clean();
            cout << endl;
            cout << "按回车键继续";
            while (_getch() != '\r');
            break;
        case 3:
            hanoi(layer, start, tmp, ending, 3);
            clean();
            cout << endl;
            cout << "按回车键继续";
            while (_getch() != '\r');
            break;
        case 4:
            cct_cls();
            cct_gotoxy(Status_Line_X, Status_Line_Y);
            cout << "从 " << start << " 移动到 " << ending<<", 共 "<<layer<<" 层，延时设置为"<<sleep << "ms" << endl;
            cct_gotoxy(MenuItem4_Start_X, MenuItem4_Start_Y);
            cout<<"初始: ";
            print_tower();
            print_tower_vertical(4);
            delay_time(sleep);
            hanoi(layer, start,tmp, ending, 4);
            clean();
            cct_gotoxy(Status_Line_X, Status_Line_Y+1);
            cout << "按回车键继续";
            while (_getch() != '\r');
            break;
        case 6:
            cct_cls();
            cct_gotoxy(Status_Line_X, Status_Line_Y );
            cout << "从 " << start << " 移动到 " << ending << ", 共 " << layer << " 层";
            paint_disks();
            paint_pan(layer, start);
            clean();
            cct_gotoxy(Status_Line_X, Status_Line_Y + 2);
            cout << "按回车键继续";
            while (_getch() != '\r');
            break;
        case 7:
            cct_cls();
            cct_gotoxy(Status_Line_X, Status_Line_Y);
            cout << "从 " << start << " 移动到 " << ending << ", 共 " << layer << " 层";
            move_disk(start, ending);
            paint_disks();
            paint_pan(layer, start);
            paint_move_pan(1, start, ending);
            clean();
            cct_gotoxy(Status_Line_X, Status_Line_Y + 1);
            cout << "按回车键继续";
            while (_getch() != '\r');
            break;
        case 8:
            cct_cls();
            cct_gotoxy(Status_Line_X, Status_Line_Y);
            cout << "从 " << start << " 移动到 " << ending<<",共 "<<layer<<" 层，延时设置为"<<sleep << "ms（前 7 步，后面自动变为 0ms）";
            cct_gotoxy(MenuItem8_Start_X, MenuItem8_Start_Y);
            cout << "初始: ";
            print_tower();
            print_tower_vertical(8);
            paint_disks();
            paint_pan(layer, start);
            delay_time(sleep);
            hanoi(layer, start, tmp, ending, 8);
            clean();
            cct_gotoxy(Status_Line_X, Status_Line_Y + 1);
            cout << "按回车键继续";
```

```
            while (_getch() != '\r');
            break;
        case 9:
            cct_cls();
            cct_gotoxy(Status_Line_X,
Status_Line_Y);
            cout << "从 " << start << " 移动到 " <<
ending << ", 共 " << layer << " 层";
            cct_gotoxy(MenuItem8_Start_X,
MenuItem8_Start_Y);
            cout << "初始: ";
            print_tower();
            print_tower_vertical(8);
            paint_disks();
            paint_pan(layer, start);
            cct_gotoxy(Status_Line_X,
Status_Line_Y-1);
            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) ：";
            solution_9(layer, start, tmp, ending);
            clean();
            cct_gotoxy(Status_Line_X, Status_Line_Y
+ 1);
            cout << "按回车键继续";
            while (_getch() != '\r');
            break;
    }
    return 0;
}
```