同济大学

Tongji University

《高级语言程序设计》

实验报告

报告名称：汉诺塔综合演示的实现和总结

班　　级：软件工程

学　　号：2253744

姓　　名：林觉凯

完成日期：2023年11月28日

1. 题目描述

1.1.实验的总体内容描述

这次汉诺塔实验大作业，总体的要求为将前几次作业的汉诺塔部分进行整合，并添加伪图形界面演示功能。最后在程序中功能用菜单的形式进行选择并进行相应的操作。

菜单中的前四项对应先前的作业，它们分别是“基本解”,“基本解（步数记录）”,“内部数组显示（横向）”,“内部数组显示（纵向+横向）”。5-9 项则是新增的图形化项，详细要求如下：

5.图形解-预备-画三个圆柱：要求在屏幕上画出三个圆柱，并且添加一定延时，以便观察实现的过程。

6.图形解-预备-在起始柱上画 n 个盘子：需要输入汉诺塔层数（1-10）、起始柱编号、目标柱编号（输入阶段有一定的错误处理），三根圆柱的编号从左到右分别是 A、B、C。同时在起始柱上从小到大画出 n 个圆盘，每层圆盘长度递减，颜色各不相同。为了方便观察实现的过程，同样需要添加延时。

7.图形解-预备-第一次移动：在菜单 6 的基础上，完成第一个圆盘的移动操作。由于输入的层数不同，第一次移动可能是从起始柱到目标柱，也可能是起始柱到中间柱。移动时，要求先向上垂直移动，再横向平移，最后向下垂直移动，不能直接在两个圆柱之间移动。在这个阶段也需要添加延时。

8.图形解-自动移动版本：这是对汉诺塔演示过程的完整图形化的实现，要求每次移动均像菜单 7 中的第一次移动一样，完整地将汉诺塔移动的整个过程展示出来。在移动时，也是要求先向上垂直移动，再横向平移，最后向下垂直移，不能直接在两个圆柱之间移动。在这个阶段也需要添加延时。

9.图形解-游戏版：由人工操作不同圆盘的移动。需要输入汉诺塔层数（1-10）、起始柱编号、目标柱编号（输入阶段有一定的错误处理），通过菜单 5 和菜单 6 的相应操作绘制好圆柱和圆盘后，需要每次键盘人工输入两个字母（A-C，大小写均可），表示本次移动的起始柱和目标柱。移动时要检查合理性，如果出现大盘压小盘、源柱为空等不符合移动规则的操作，要提示错误并要求重新输入。若是合理的移动，则需要记录下该步骤的步数。同样地，在移动时，要求先向上垂直移动，再水平向平移，最后向下垂直移动，不能直接在两个圆柱之间移动。在这个阶段也需要添加延时。如果所有圆盘均正确移动到目标柱上，则游戏结束，需要输出相应的提示语句；或者是输入 q（大小写均可）表示中止游戏，输出相应的提示语句。

1.2.实验的注意事项描述

1.2.1.菜单项的注意事项

1. 整个程序只能使用一个递归函数，菜单项的 1/2/3/4/8 必须共用一个递归函数，需要用参数解决每个菜单功能不同要求之间的差异，且递归函数不超过 15 行；
2. 菜单项中 1/2/3/4/6/7/8 中的多输入参数必须共用一个函数，菜单项 9 根据个人看是否共用；
3. 菜单项中 3/4/8 中横向输出必须共用一个函数，位置不同使用参数的变换来调整；
4. 菜单项中 4/8 中纵向输出必须共用一个函数，位置不同使用参数的变换来调整；
5. 菜单项中 5/6/7/8/9 中画柱子必须共用一个函数；

6. 菜单项中 7/8/9 中盘子的移动必须共用一个函数；

1.2.2.定义变量的注意事项

允许定义全局变量：

1. 用 1 个全局简单变量来记录总移动步数；
2. 用 3 个全局一维数组或 1 个全局二维数组来记录圆柱上现有圆盘的编号；
3. 用 3 个全局简单变量或 1 个全局一维数组来记录圆柱上现有圆盘的数量；
4. 用 1 个全局简单变量来记录延时；

其他不允许，const 和#define 不受限制。

（本份程序使用 3 个全局一维数组和 3 个全局简单变量来实现）

静态局部变量的数量不限制，但使用准则也是：少用、慎用、能不用尽量不用。

1.2.3.屏幕显示的注意事项

1. 为了方便观察实现的过程，我们需要添加延时，延时的系统函数为 Sleep(单位：毫秒)，同时需要在程序中包含头文件<Windows.h>

2. 输入完成后，我们可以使用 ccl_cls()来清除屏幕上现有的内容（输入提示及输入信息）。但是要注意，该命令只能使用一次，不允许每次移动一个元素后就清除屏幕并重新输出，而是只能擦除原有的位置，在新的位置上进行输出

2. 整体设计思路

2.1.程序的主体架构设计

本程序的主体可以大致分为两个部分，任务选择部分与任务实现部分。任务选择部分包括负责调用任务函数的 main 主函数与负责读取任务选择需求与输出菜单的 menu 函数。任务实现部分是该程序主体部分，主要可以分为 5 个部分：

1. 初始化的选择部分，直接供 main 函数调用的九个解函数；
2. 逻辑控制部分，负责控制汉诺塔的内部计算方式和汉诺塔的游戏控制；
3. 输入输出部分，负责显示界面、输入的健壮性判断和读取用户输入的参数；
4. 数据的表示部分，负责打印汉诺塔内部的数组、圆盘的移动过程和步骤；
5. 数据的内部计算部分，负责进行汉诺塔内部数据的计算变化(如递归和栈的变化)。

初始化的选择部分根据题目的要求，调用相应的解函数，利用相应的输入输出部分来从用户获得初始化参数，为逻辑控制部分提供相应的模式设置。随后逻辑控制部分就会执行具体的各项操作，来选择相应的数据内部计算方式来达成相应的解，最后通过数据的表示部分将最后的计算结果输出。

2.2.实验项目的文件组成

2.2.1.hanoi.h

为了保证 hanoi_main.cpp/hanoi_menu.cpp/hanoi_multiple_solutions.cpp 能相互访问，本头文件存放了相应的函数的函数声明；除此之外，还存放了宏定义的全局常量。（比如延迟的基本时间、圆盘底座的横纵坐标、中心横纵坐标和塔的高度）

2.2.2.hanoi_menu.cpp

该 cpp 文件存放了被 hanoi_main.cpp 调用的菜单函数 Hanoi_Menu()，主要用于打印菜单信息给用户，并读取菜单的选项返回给 main 函数。

2.2.3.hanoi_main.cpp

该 cpp 文件存放 main()函数，它利用循环调用 Hanoi_Menu()函数，根据不同的选项调用 hanoi_multiple_solution.cpp 中的解函数来实现最后的问题解或者退出。

2.2.4.hanoi_multiple_solution.cpp

所有的任务实现函数都定义在此文件中，包括解函数、输入输出、数据的内部计算、初始化的选择函数、数据的表示函数等等，它是实现各个选项汉诺塔功能的主要文件。

3. 主要功能的实现

3.1.主要功能的函数实现

3.1.1.Reset()函数

该函数用来初始化静态全局变量,在需要运用到栈的操作前都需将移动步数 Movement 初始化为 1，三个栈顶和三个栈中内容初始化为 0。此函数重要，没有这个函数的话只能正常运行一次。

3.1.2.Time_Sleep(int speed)函数

该函数用来设置延时功能,输入延时功能的选项 speed，当 speed 为 0 时表示按回车单步演示，1 延时最长，5 延时最短，1-5 中间的选项根据不同的调试速度进行呈现。

3.1.3.void Input(char* src, char* tmp, char* dst, int* n, int speed_flag)函数

该函数用来确定汉诺塔层数和三根柱，并确定是否使用延时（做过错误处理），输入汉诺塔层数、起始柱和目标柱，延时标志，当 speed_flag 为-1 时表示该操作不使用延时操作。

3.1.4.void Init(int n, char src)函数

该函数用来初始化起始柱，输入汉诺塔层数和起始柱，将起始柱的对应栈中元素初始为相应盘。

3.1.5.void Hanoi_Stack(char src, char dst)函数

该函数用来实现元素在汉诺塔塔栈里的移动，输入此步移动的起始柱和目标柱通过栈的出栈入栈操作实现汉诺塔的塔栈内元素的互相移动，是汉诺塔移动盘子的主要函数之一。

3.1.6.void Print_Bottom(int x, int y)函数

该函数用来打印选项 4 中的底盘，输入是底盘最左侧的坐标 x,y，输入的 x,y 用于定位。

3.1.7.void Print_Crosswise(int x, int y)函数

该函数用来横向打印汉诺塔数组各元素，输入汉诺塔横向打印数组的位置坐标，输入的 x,y 用于定位。

3.1.8.void Print_Vertical(char src, char dst, int init, int bottom_x, int bottom_y, int x, int y)函数

该函数用来纵向打印汉诺塔数组各元素，输入起始柱、目标柱、是否初始化状态、底盘的 x、y 坐标和打印纵向数组的 x、y 坐标 init 为 1 是表示初始化状态，此时打印起始柱的数组内容即可。

3.1.9.void Print_Column()函数

该函数用来完成选项 5 中的预备画三个圆柱。（画圆柱时有延时操作和颜色的相应设置）

3.1.10.void Print_Plate(int n, char src)函数

该函数用来完成选项 6 中的在起始柱上画 n 个盘子的要求，输入汉诺塔层数和起始柱。

3.1.11.void Move_Vertical_Up(char src, int speed)函数

该函数用来完成盘子的垂直向上移动，输入是起始柱的信息和移动速度设置，Length_Flag 并不是盘

的真实长度,而是每个盘的长度标志,还需乘 2 加 1 才表示该盘的实际长度。

3.1.12.void Move_Crosswise(char src, char dst, int speed)函数

该函数用来完成盘子的水平方向的移动，输入起始柱、目标柱的信息和移动速度设置，在水平移动的过程中需要考虑起始柱和目标柱的相对位置来决定左移和右移。

3.1.13.void Move_Vertical_Down(char src, char dst, int speed)函数

该函数用来完成盘子的垂直向下的移动，输入起始柱、目标柱和移动速度设置。

3.1.14.void Move_Plate(char src, char dst, int speed)函数

该函数用来完成盘子的移动操作，输入起始柱、目标柱和移动速度设置，如果 speed 为 0，即对应着按回车单步演示，此时我们将 speed 的值手动设为 10。

3.1.15.void Hanoi_Game(int n, char dst)函数

该函数用来汉诺塔游戏（选项 9）的实现，输入汉诺塔层数和目标柱的信息。在此函数的用户输入是有较为复杂的错误处理，最后通过判断目标柱的顶是否为盘 1 来判断游戏是否结束。

3.1.16.void Operation(char src, char tmp, char dst, int n, int speed, int choice)函数

该函数用来完成选项 1、2、3、4、7、8 中在递归过程中的各类打印操作（是缩短递归函数行数的关键），输入起始柱、中间柱、目标柱的信息、汉诺塔层数、速度设置和菜单选项。

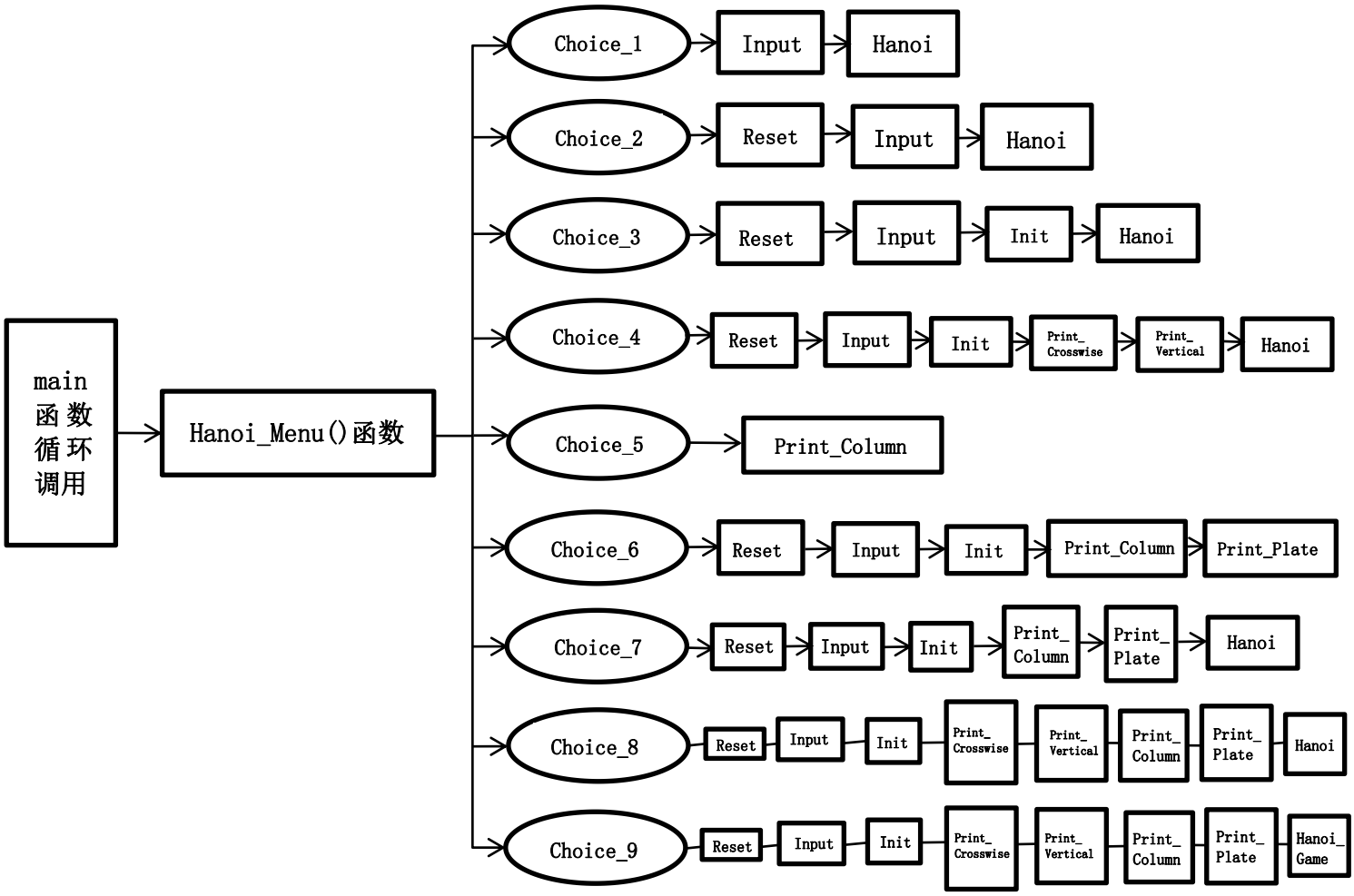
3.1.17.void Hanoi(char src, char tmp, char dst, int n, int speed, int choice)函数

该函数为汉诺塔递归主函数，输入起始柱、中间柱、目标柱、汉诺塔层数、速度设置和菜单选项，在选项 7 中只显示第一次移动步骤，故当 choice == 7 进行一步操作后直接 return。

3.1.18. void Choice_1()函数、void Choice_2()函数、void Choice_3()函数、void Choice_4()函数、void Choice_5()函数、void Choice_6()函数、void Choice_7()函数、void Choice_8()函数、void Choice_9()函数

这九个函数为解函数，用来完成每一个菜单选项的功能，每一个函数中的具体实现操作都按照具体要求将上述函数组合，从而形成解。

3.2.主要功能的流程框架



该程序的主要流程为：通过 main 函数中的循环不断调用 Hanoi_Menu()函数进行选择解决函数，不同的解决函数通过调用不同的任务函数来完成该项解法的要求。其中 1-8 项需要用到 Hanoi 递归函数，而第 9 项则需要通过 Hanoi_Game 函数手动进行解决移动柱的问题。

4. 调试中碰到的问题

4.1.问题1

在进行模拟调试的时候，发现进行过一次操作之后，后续的结果都出现了异常。在检查的过程中，除了第一次操作时六个全局变量（三个栈顶、三个栈数组）初始都为0，后面不是；解决方法是写了一个Reset()函数来专门重置全局变量，在每一次调用解决函数的时候首先调用Reset()函数，将六个全局变量重置之后，再进行之后的操作。

4.2.问题2

在刚开始做伪图形界面的那几项的时候，经常发现使用颜色函数打印带有颜色盘子之后再次进行其他操作的时候满屏都是之前的颜色或者总是有一整行是之前盘子的颜色；认真学习之后我发现在每次使用打印盘子设定颜色的那个函数之后需要在使用cct_setcolor(COLOR_BLACK, COLOR_WHITE)将颜色调回。

4.3.问题3

在编写 Move_Plate 函数的时候经常找不到光标的具体位置，方向也经常搞错。所以我选择设定几个全局变量（比如 Base_X、Base_Y、Base_Distance、A_Base_Middle、B_Base_Middle、C_Base_Middle 和 Column_Height，方便直接找到位置；同时注意控制台中向下 y 坐标增大，向上减少。

没有其他较为主要的问题出现。

5. 心得与体会

5.1.本次作业的经验教训与体会

积累的经验主要有以下几点：更加熟练地了解并运用了老师在 cmd_console_tools.cpp 中给的函数。之前几次老师有给过类似的作业，但是这次我更加系统地学习了这些有关光标、键盘和颜色的函数，在之后的学习中肯定有用；再次体会到利用多个函数写复杂程序的巧妙之处，比如可以用同一个函数（只是中间的变量稍微进行调整，就可以达到函数多次利用的效果；同时我更加熟练地掌握了使用同一个项目，多文件能互相访问的操作；更加深入地了解了递归内涵，之前一个一个作业分开做可能没有这么深刻的感觉。

5.2.复杂的题目希望多小题还是直接一道大题

对于编写代码、初学者学习来说，希望多个小题，因为这样可以循序渐进，逐步体会同一个主题下的内涵；当然，在写完多个小题之后再来一道大题，我觉得可以更加深刻地将本主题的所有知识点串联起来，互相对比，互相影响，会有更好的巩固效果。所以我认为本课程对汉诺塔的知识点分析练习方法十分到位！

5.3.如何更好地重用代码

我在编写此次程序的过程中前 4 项几乎都是前几次代码的整合与归纳，我可以做到有效利用。我觉得更好地重用代码，最主要还是清晰地将函数分模块，同时要善于找出不同操作中相似的部分进行归纳整合。

5.4.如何更好利用函数编写复杂程序

首先我认为需要注意分层分块，将一个比较巨大的问题用多个函数有序地用逻辑顺序表现出来；其次需要善于归纳，很多看似需要用多个函数表达的问题可能这多个函数之间存在相似的逻辑，要善于将这些类似的操作归类成同一函数；最后，遇到一个复杂的程序的时候要分块看，分区实现，尽量不受到其他区块的逻辑影响；将每一块的基本操作都实现后，通过每一块之间的某一相关系的量联系在一起。

6. 附件：源程序

hanoi.h中主要部分宏定义：

```
#define Time 500
#define Base_X 1
#define Base_Y 15
#define Base_Distance 33
#define A_Base_Middle 12
#define B_Base_Middle 45
#define C_Base_Middle 78
#define Column_Height 12
```

hanoi_menu.cpp 中主要部分返回字符：

```
char keydown;
while (1)
{
    keydown = _getch();
    if (keydown >= '0' && keydown <= '9')
    {
        putchar(keydown);
        break;
    }
}
cout << endl;
return keydown;
```

hanoi_main.cpp 中主要部分循环：

```
while (1)
{
    choice = Hanoi_Menu();
    if (choice == '0')
    {
        cct_gotoxy(0, 37);
        return 0;
    }
    cout << endl << endl;
    switch (choice)
    {
    case '1':
        Choice_1();
        cout << endl;
        break;
    case '2':
        Choice_2();
        cout << endl;
        break;
    case '3':
        Choice_3();
        cout << endl;
        break;
    case '4':
        Choice_4();
        cout << endl;
        break;
    case '5':
        Choice_5();
        break;
    case '6':
        Choice_6();
        break;
    case '7':
        Choice_7();
        break;
    case '8':
        Choice_8();
        break;
    case '9':
        Choice_9();
        break;
    }
```

hanoi_multiple_solutions.cpp 中主要函数：

```
static int speed;
static int movecount;
static int A_stack[10], B_stack[10], C_stack[10];
static int A_top, B_top, C_top;
void Reset()
{
    movecount = 1;
    A_top = B_top = C_top = 0;
    for (int i = 0; i < 10; i++)
        A_stack[i] = B_stack[i] = C_stack[i] = 0;
}
void Time_Sleep(int speed)
{
    if (speed)
        Sleep(Time / speed);
    else
        while (getchar() != '\n');
}
void Input(char* src, char* tmp, char* dst, int* n, int
speed_flag)
{
    while (1)
    {
        cout <<"请输入汉诺塔的层数(1-10)：" << endl;
        cin >> *n;
        if (cin.good() == 0 || *n < 1 || *n > 10)
        {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        break;
    }
    while (1)
    {
        cout << "请输入起始柱(A-C)" << endl;
        cin >> *src;
        if (cin.good() == 0)
        {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        if (*src >= 'a' && *src <= 'c')
```

```
            *src -= 32;
        if (*src > 'C' || *src < 'A')
        {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        cin.clear();
        cin.ignore(1024, '\n');
        break;
    }
    while (1)
    {
        cout << "请输入目标柱(A-C)" << endl;
        cin >> *dst;
        if (cin.good() == 0)
        {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        if (*dst >= 'a' && *dst <= 'c')
            *dst -= 32;
        if (*dst > 'C' || *dst < 'A')
        {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        if (*dst == *src)
        {
            cout << "目标柱(" << *dst << ")不能与起始柱(" << *src << ")相同" << endl;
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        cin.clear();
        cin.ignore(1024, '\n');
        break;
    }
    *tmp = 'B' * 3 - *src - *dst;
    if (speed_flag)
    {
        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() == 0 || speed > 5 || speed < 0)
            {
                cin.clear();
                cin.ignore(1024, '\n');
                continue;
            }
            cin.clear();
            cin.ignore(1024, '\n');
            break;
        }
    }
}
void Init(int n, char src)
{
    if (src == 'A')
    {
        while (A_top < n)
            A_stack[A_top++] = n - A_top;
    }
    当 src 为 B 或 C 时与 A 操作类似，省略
}
void Hanoi_Stack(char src, char dst)
{
    int element = 0;
    if (src == 'A' && A_top > 0)
    {
        element = A_stack[--A_top];
        A_stack[A_top] = 0;
    }
    当 dst 为 B 或 C 时与 A 操作类似，省略
    if (dst == 'A' && A_top < 10)
        A_stack[A_top++] = element;
}
void Print_Bottom(int x, int y)
{
    cct_gotoxy(x, y);
    cout << "         ========================" <<
endl;
    cout << "           A         B         C";
}
void Print_Crosswise(int x, int y)
{
    cct_gotoxy(x, y);
    cout << " A:";
    for (int i = 0; i < 10; i++)
    {
        if (A_stack[i])
            cout << setw(2) << A_stack[i];
        else
            cout << "  ";
    }
    cout << " B:";
    for (int i = 0; i < 10; i++)
    {
        if (B_stack[i])
            cout << setw(2) << B_stack[i];
        else
            cout << "  ";
    }
    cout << " C:";
    for (int i = 0; i < 10; i++)
    {
        if (C_stack[i])
            cout << setw(2) << C_stack[i];
```

```
        else
            cout << "  ";
    }
    cout << endl;
}
void Print_Vertical(char src, char dst, int init, int
bottom_x, int bottom_y, int x, int y)
{
    Print_Bottom(bottom_x, bottom_y);
    if (init)
    {
        for (int i = 0; i < A_top; i++)
        {
            cct_gotoxy(x, y - i);
            if (A_stack[i])
                cout << setw(2) << A_stack[i];
        }
        for (int i = 0; i < B_top; i++)
        {
            cct_gotoxy(x + 10, y - i);
            if (B_stack[i])
                cout << setw(2) << B_stack[i];
        }
        for (int i = 0; i < C_top; i++)
        {
            cct_gotoxy(x + 20, y - i);
            if (C_stack[i])
                cout << setw(2) << C_stack[i];
        }
    }
    else
    {
        if (src == 'A')
        {
            cct_gotoxy(x, y - A_top);
            cout << "  ";
        }
        当 src 为 B 或 C 时与 A 操作类似，省略
        if (dst == 'A')
        {
            cct_gotoxy(x, y + 1 - A_top);
            cout << setw(2) << A_stack[A_top - 1];
        }
        当 dst 为 B 或 C 时与 A 操作类似，省略
    }
}
void Print_Column()
{
    cct_setcursor(CURSOR_INVISIBLE);
    const int Column_Color = COLOR_HYELLOW;
    cct_showch(Base_X, Base_Y, ' ', Column_Color,
Column_Color, 23);
    cct_showch(Base_X + Base_Distance, Base_Y, ' ',
Column_Color, Column_Color, 23);
    cct_showch(Base_X + 2 * Base_Distance, Base_Y, '
', Column_Color, Column_Color, 23);
    for (int i = 0; i < Column_Height; i++)
    {
        cct_showch(A_Base_Middle, Base_Y - 1 - i, ' ',
Column_Color, Column_Color, 1);
        Sleep(50);
        cct_showch(B_Base_Middle, Base_Y - 1 - i, ' ',
Column_Color, Column_Color, 1);
        Sleep(50);
        cct_showch(C_Base_Middle, Base_Y - 1 - i, ' ',
Column_Color, Column_Color, 1);
        Sleep(50);
    }
}
void Print_Plate(int n, char src)
{
    int Plate_Base_x;
    int Plate_Base_y = Base_Y;
    if (src == 'A')
        Plate_Base_x = A_Base_Middle;
    else if (src == 'B')
        Plate_Base_x = B_Base_Middle;
    else
        Plate_Base_x = C_Base_Middle;
    for (int i = n; i > 0; i--)
    {
        cct_showch(Plate_Base_x - i, Plate_Base_y - 1
+ i - n, ' ', i, i, 2 * i + 1);
        Sleep(30);
    }
}
void Move_Vertical_Up(char src, int speed)
{
    int Origin_X, Start_Location, Length_Flag;
    if (src == 'A')
    {
        Origin_X = A_Base_Middle;
        Start_Location = A_top;
        Length_Flag = A_stack[A_top - 1];
    }
    else if (src == 'B')
    {
        Origin_X = B_Base_Middle;
        Start_Location = B_top;
        Length_Flag = B_stack[B_top - 1];
    }
    else
    {
        Origin_X = C_Base_Middle;
        Start_Location = C_top;
        Length_Flag = C_stack[C_top - 1];
    }
    for (int i = Base_Y - Start_Location; i > 0; i--)
    {
        cct_showch(Origin_X - Length_Flag, i, ' ',
Length_Flag, Length_Flag, 2 * Length_Flag + 1);
        Sleep(Time / speed);
        if (i <= Base_Y - Start_Location)
        {
```

```
            cct_showch(Origin_X - Length_Flag, i, ' ',
COLOR_BLACK, COLOR_WHITE, 2 * Length_Flag + 1);
            if (i > 2)
                cct_showch(Origin_X,    i,    '    ',
COLOR_HYELLOW, COLOR_HYELLOW, 1);
        }
    }
}
void Move_Crosswise(char src, char dst, int speed)
{
    int Src_X, Dst_X, Length_Flag;
    if (src == 'A')
    {
        Src_X = A_Base_Middle;
        Length_Flag = A_stack[A_top - 1];
    }
    当 src 为 B 或 C 时与 A 操作类似，省略
    if (dst == 'A')
        Dst_X = A_Base_Middle;
    当 dst 为 B 或 C 时与 A 操作类似，省略
    if (src < dst)
    {
        for (int i = Src_X - Length_Flag; i <= Dst_X
- Length_Flag; i++)
        {
            cct_showch(i,   1,   '  ',   Length_Flag,
Length_Flag, 2 * Length_Flag + 1);
            Sleep(Time / speed);
            if (i < Dst_X - Length_Flag)
                cct_showch(i, 1, ' ', COLOR_BLACK,
COLOR_WHITE, 2 * Length_Flag + 1);
        }
    }
    else
    {
        for (int i = Src_X - Length_Flag; i >= Dst_X
- Length_Flag; i--)
        {
            cct_showch(i,   1,   '  ',   Length_Flag,
Length_Flag, 2 * Length_Flag + 1);
            Sleep(Time / speed);
            if (i > Dst_X - Length_Flag)
                cct_showch(i, 1, ' ', COLOR_BLACK,
COLOR_WHITE, 2 * Length_Flag + 1);
        }
    }
}
void Move_Vertical_Down(char src, char dst, int speed)
{
    int Origin_X, End_Location, Length_Flag;
    if (src == 'A')
        Length_Flag = A_stack[A_top - 1];
    else if (src == 'B')
        Length_Flag = B_stack[B_top - 1];
    else
        Length_Flag = C_stack[C_top - 1];
    if (dst == 'A')
    {
        Origin_X = A_Base_Middle;
        End_Location = A_top;
    }
    else if (dst == 'B')
    {
        Origin_X = B_Base_Middle;
        End_Location = B_top;
    }
    else
    {
        Origin_X = C_Base_Middle;
        End_Location = C_top;
    }
    for (int i = 1; i < Base_Y - End_Location; i++)
    {
        cct_showch(Origin_X - Length_Flag, i, ' ',
Length_Flag, Length_Flag, 2 * Length_Flag + 1);
        Sleep(Time / speed);
        if (i < Base_Y - End_Location - 1)
        {
            cct_showch(Origin_X - Length_Flag, i, ' ',
COLOR_BLACK, COLOR_WHITE, 2 * Length_Flag + 1);
            if (i > 2)
                cct_showch(Origin_X,    i,    '    ',
COLOR_HYELLOW, COLOR_HYELLOW, 1);
        }
    }
}
void Move_Plate(char src, char dst, int speed)
{
    if (!speed)
        speed = 10;
    Move_Vertical_Up(src, speed);
    Move_Crosswise(src, dst, speed);
    Move_Vertical_Down(src, dst, speed);
}
void Hanoi_Game(int n, char dst)
{
    char Input_src, Input_dst;
    int Get_element, Endbase_element;
    while (1)
    {
        cct_gotoxy(0, 34);
        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) ：";
        cct_showch(60,   34,   '  ',   COLOR_BLACK,
COLOR_WHITE, 10);
        cct_gotoxy(60, 34);
        Input_src = getchar();
        if (Input_src == '\n')
            continue;
        if (cin.good() == 0 || (Input_src != 'A' &&
Input_src != 'B' && Input_src != 'C' && Input_src !=
'Q' && Input_src != 'a' && Input_src != 'b' &&
Input_src != 'c' && Input_src != 'q'))
        {
```

```
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        Input_dst = getchar();

        if ((Input_src == 'Q' || Input_src == 'q') && Input_dst ==  '\n' )
        {
            cout << "游戏中止!!!!!" << endl << endl;
            break;
        }
       if (cin.good() == 0 || (Input_dst != 'A' && Input_dst != 'B' && Input_dst != 'C' && Input_dst != 'a' && Input_dst != 'b' && Input_dst != 'c'))
        {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
    char Judge = getchar();
    if (Judge != '\n')
    {
        cin.clear();
        cin.ignore(1024, '\n');
        continue;
}       if (Input_src >= 'a' && Input_src <= 'z')
            Input_src -= 32;
        if (Input_dst >= 'a' && Input_dst <= 'z')
            Input_dst -= 32;
        if (Input_src == Input_dst)
            continue;
        if (Input_src == 'A' && A_top > 0)
            Get_element = A_stack[A_top - 1];
        当 Input_src 为 B 或 C 时与 A 操作类似，省略
        else
        {
            cct_showstr(0,  35,  "源柱为空!", COLOR_BLACK, COLOR_WHITE, 1);
            Sleep(2 * Time);
            cct_showstr(0,  35,  "                ", COLOR_BLACK, COLOR_WHITE, 1);
            continue;
        }
        if (Input_dst == 'A' && A_top > 0)
            Endbase_element = A_stack[A_top - 1];
        else if (Input_dst == 'B' && B_top > 0)
            Endbase_element = B_stack[B_top - 1];
        else if (Input_dst == 'C' && C_top > 0)
            Endbase_element = C_stack[C_top - 1];
        else
            Endbase_element = 11;
        if (Get_element > Endbase_element)
        {
            cct_showstr(0, 35, "大盘压小盘，非法移动!", COLOR_BLACK, COLOR_WHITE, 1);
            Sleep(2 * Time);
            cct_showstr(0,          35,          "                ", COLOR_BLACK, COLOR_WHITE, 1);
            continue;
        }
        int x, y;
        cct_gotoxy(0, 32);
        cout << "第" << setw(4) << movecount++ << " 步(" << Get_element << "#: " << Input_src << "-->" << Input_dst << ")  ";
        Hanoi_Stack(Input_src, Input_dst);
        cct_getxy(x, y);
        Print_Crosswise(x, y);
        Print_Vertical(Input_src, Input_dst, 0, 0, 27, 10, 26);
        Hanoi_Stack(Input_dst, Input_src);
        Move_Plate(Input_src, Input_dst, 10);
        cct_setcolor(COLOR_BLACK, COLOR_WHITE);
        Hanoi_Stack(Input_src, Input_dst);
        int Final_Plate;
        if (dst == 'A')
            Final_Plate = A_stack[n - 1];
        else if(dst =='B')
            Final_Plate = B_stack[n - 1];
        else
            Final_Plate = C_stack[n - 1];
        if (Final_Plate == 1)
        {
            cct_showstr(0,  35,  "游戏结束!!!!!", COLOR_BLACK, COLOR_WHITE, 1);
            break;
        }
    }
}
void Operation(char src, char tmp, char dst, int n, int speed, int choice)
{
    if (choice == 1)
        cout << n << "# " << src << "---->" << dst << endl;
    else if (choice == 2)
        cout << "第" << setw(4) << movecount++ << " 步(" << setw(2) << n << "#: " << src << "-->" << dst << ")" << endl;
    else if (choice == 3)
    {
        int x, y;
        cout << "第" << setw(4) << movecount++ << " 步(" << setw(2) << n << "#: " << src << "-->" << dst << ")";
        Hanoi_Stack(src, dst);
        cct_getxy(x, y);
        Print_Crosswise(x, y);
    }
    else if (choice == 4)
    {
        cct_gotoxy(0, 17);
```

```
        cout << "第" << setw(4) << movecount++ << " 步
(" << setw(2) << n << "#: " << src << "-->" << dst <<
")";
        Hanoi_Stack(src, dst);
        Print_Crosswise(22, 17);
        Print_Vertical(src, dst, 0, 0, 12, 10, 11);
        Time_Sleep(speed);
    }
    else if (choice == 7)
    {
        Move_Plate(src, dst, 10);
    }
    else if (choice == 8)
    {
        int x, y;
        cct_gotoxy(0, 32);
        cout << "第" << setw(4) << movecount++ << " 步
( " << n << "#: " << src << "-->" << dst << ")  ";
        Hanoi_Stack(src, dst);
        cct_getxy(x, y);
        Print_Crosswise(x, y);
        Print_Vertical(src, dst, 0, 0, 27, 10, 26);
        Hanoi_Stack(dst, src);
        Move_Plate(src, dst, speed);
        Hanoi_Stack(src, dst);
        cct_setcolor(COLOR_BLACK, COLOR_WHITE);
        Time_Sleep(speed);
    }
}
void Hanoi(char src, char tmp, char dst, int n, int
speed, int choice)
{
    if (n == 1)
        Operation(src, tmp, dst, n, speed, choice);
    else
    {
        Hanoi(src, dst, tmp, n - 1, speed, choice);
        if (choice == 7)
            return;
        Operation(src, tmp, dst, n, speed, choice);
        Hanoi(tmp, src, dst, n - 1, speed, choice);
    }
}
```

九个解函数的主要操作调用的函数：

```
void Choice_1()
{
    Input(&src, &tmp, &dst, &n, 0);
    Hanoi(src, tmp, dst, n, -1, 1);
}
void Choice_2()
{
    Reset();
    Input(&src, &tmp, &dst, &n, 0);
    Hanoi(src, tmp, dst, n, -1, 2);
}
void Choice_3()
{
    Reset();
    Input(&src, &tmp, &dst, &n, 0);
    Init(n, src);
    Hanoi(src, tmp, dst, n, -1, 3);
}
void Choice_4()
{
    Reset();
    Input(&src, &tmp, &dst, &n, 1);
    Init(n, src);
    Print_Crosswise(x, y);
    Print_Vertical(src, dst, 1, 0, 12, 10, 11);
    Hanoi(src, tmp, dst, n, speed, 4);
}
void Choice_5()
{
    Print_Column();
}
void Choice_6()
{
    Reset();
    Input(&src, &tmp, &dst, &n, 0);
    Init(n, src);
    Print_Column();
    Print_Plate(n, src);
}
void Choice_7()
{
    Reset();
    Input(&src, &tmp, &dst, &n, 0);
    Init(n, src);
    Print_Column();
    Print_Plate(n, src);
    Hanoi(src, tmp, dst, n, -1, 7);
}
void Choice_8()
{
    Reset();
    Input(&src, &tmp, &dst, &n, 1);
    Init(n, src);
    Print_Crosswise(x, y);
    Print_Vertical(src, dst, 1, 0, 27, 10, 26);
    Print_Column();
    Print_Plate(n, src);
    Hanoi(src, tmp, dst, n, speed, 8);
}
void Choice_9()
{
    Reset();
    Input(&src, &tmp, &dst, &n, 0);
    Init(n, src);
    Print_Crosswise(x, y);
    Print_Vertical(src, dst, 1, 0, 27, 10, 26);
    Print_Column();
    Print_Plate(n, src);
    Hanoi_Game(n, dst);
}
```