同济大学
TONGJI UNIVERSITY

《高级程序语言设计》

实验报告

实 验 名 称: 汉诺塔综合演示

专　　业: 数据科学与大数据技术

学　　号: 2351495

姓　　名: 闫业豪

完 成 时 间: 2025 年 5 月 21 日

1. 题目

1.1.总体描述

本次实验要求将之前完成的汉诺塔问题的各种解法集成到一个程序中，通过菜单方式选择不同的功能。程序需要包含从基本的文本输出到具有动画效果的图形界面演示，全面展示汉诺塔问题的求解过程。主要功能包括基本解法、步数记录、内部数组显示（横向和纵向）、图形解-预备（画柱子和盘子）、以及带动画效果的移动演示和用户交互功能。

1.2.题目要求

1.2.1. 函数设计要求

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. 变量设计要求

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。
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. 参数设计要求

为了能在显示位置、颜色、图形的粗细等方面做到灵活变化，即不在程序中写死，要求使用附件中的自带的hanoi_const_value.h,要求将任意一个改名为hanoi_const_value.h后，均能编译通过，并且达到对应demo的显示效果。

2. 整体设计思路

2.1.程序的主体架构设计

本程序采用模块化设计，主体分为两个部分：任务选择部分与任务实现部分。
任务选择部分包括：

1. main主函数：负责调用任务函数的主函数，包含主循环逻辑
2. menu函数：负责读取任务选择需求与输出菜单

任务实现部分主要可以分为5个部分：

1. 初始化选择部分：九个解函数（solve_basic_hanoi等）
2. 逻辑控制部分：核心递归函数hanoi和移动函数move_disk
3. 输入输出部分：input函数负责参数输入和验证
4. 数据表示部分：showshuiping和showshuzhi函数负责不同格式的数据显示
5. 图形界面部分：show_pillars、draw_disk、move_disk_animation等函数

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 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 menu()函数，根据不同的选项调用 hanoi_multiple_solutions.cpp 中的解函数来实现最后的问题解或者退出。

2.2.4. hanoi_multiple_solutions.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. init(int number, char src)函数

该函数用来初始化起始柱，将指定柱子的栈中按从大到小的顺序放入相应数量的圆盘。输入参数为number(汉诺塔层数)，src(起始柱编号 A、B、C)。此函数在每次开始新的汉诺塔操作前都需要调用，用于设置初始状态。

3.1.2. input(char* src, char* tmp, char* dst, int* number, bool need_delay, int menu_option = 0)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）。输入参数为 src(起始柱指针)，tmp(中转柱指针)，dst(目标柱指针)，number(层数指针)，need_delay(是否需要延时)，menu_option(菜单选项，默认为 0)。函数内部包含完整的输入验证逻辑，确保用户输入的有效性。

3.1.3. hanoi(int number, char src, char tmp, char dst, int format_type, int show_horizontal, int show_vertical, int show_graphical, int automatic)递归函数

该函数为汉诺塔问题的核心递归解决函数，实现经典的三步递归策略。输入参数为 number(汉诺塔层数)，src(起始柱编号)，tmp(中转柱编号)，dst(目标柱编号)，format_type(输出格式类型)，show_horizontal(是否显示横向数组)，show_vertical(是否显示纵向数组)，show_graphical(是否显示图形界面)，automatic(是否自动模式)。这是整个程序的核心递归函数，通过不同的参数控制不同菜单项的功能实现，函数行数严格控制在 15 行以内。

3.1.4. move_disk(int number, char src, char dst, int format_type, int show_horizontal, int show_vertical, int show_graphical, int automatic)函数

该函数用来实现单个圆盘的移动操作，包括栈操作和各种显示更新。输入参数为 number(移动的盘子编号)，src(源柱编号)，dst(目标柱编号)，format_type(输出格式类型)，show_horizontal(是否显示横向数组)，show_vertical(是否显示纵向数组)，show_graphical(是否显示图形界面)，automatic(是否自动模式)。函数通过栈的出栈入栈操作实现汉诺塔的塔栈内元素的相互移动，并根据参数控制不同的显示方式。

3.1.5. showshuiping(int x, int y, bool simple = false, int format_type = 0)函数

该函数用来横向打印汉诺塔三个栈的当前状态，支持多种显示格式。输入参数为 x(输出的 x 坐标)，y(输出的 y 坐标)，simple(是否简单模式，默认 false)，format_type(格式类型，默认 0)。函数通过参数控制在不同位置显示横向数组信息，支持菜单项 3、4、8 的横向显示需求。

3.1.6. showshuzhi(int baseX, int baseY, char src, char dst, int speed, int init, int colA_offset = 0, int colB_offset = 0, int colC_offset = 0)函数

该函数用来纵向打印汉诺塔三个栈的当前状态，模拟实际汉诺塔的垂直摆放形式。输入参数为baseX(基础 x 坐标)，baseY(基础 y 坐标)，src(源柱编号)，dst(目标柱编号)，speed(速度设置)，init(是否初始化状态)，colA_offset(A 柱偏移量，默认 0)，colB_offset(B 柱偏移量，默认 0)，colC_offset(C 柱偏移量，默认 0)。函数支持菜单项 4、8 的纵向显示需求，并可通过偏移量参数调整显示位置。

3.1.7. show_pillars()函数

该函数用来在图形界面中绘制三个汉诺塔圆柱，为后续的圆盘绘制提供基础。该函数无输入参数。函数使用 HDC 图形库绘制柱子，包含延时操作和颜色设置，支持菜单项 5、6、7、8、9 的图形显示需求。

3.1.8. draw_disk(int center_x, int base_y, int disk_num, int position)函数

该函数用来在指定柱子的指定位置绘制指定大小的圆盘，支持不同颜色显示。输入参数为center_x(柱子中心 x 坐标)，base_y(底座 y 坐标)，disk_num(盘子编号，决定大小和颜色)，position(盘子在柱子上的位置，从下往上计数)。函数根据盘子编号计算圆盘宽度，使用不同颜色区分不同大小的圆盘。

3.1.9. move_disk_animation(int disk_num, char src_col, char dst_col, bool show_text_display = false)函数

该函数用来实现圆盘移动的完整动画效果，严格按照汉诺塔规则进行三段式移动。输入参数为disk_num(要移动的盘子编号)，src_col(源柱子编号 A、B、C)，dst_col(目标柱子编号 A、B、C)，show_text_display(是否显示文本信息，默认 false)。函数实现先向上垂直移动、再横向平移、最后向下垂直移动的完整动画效果，支持速度控制和单步演示功能。

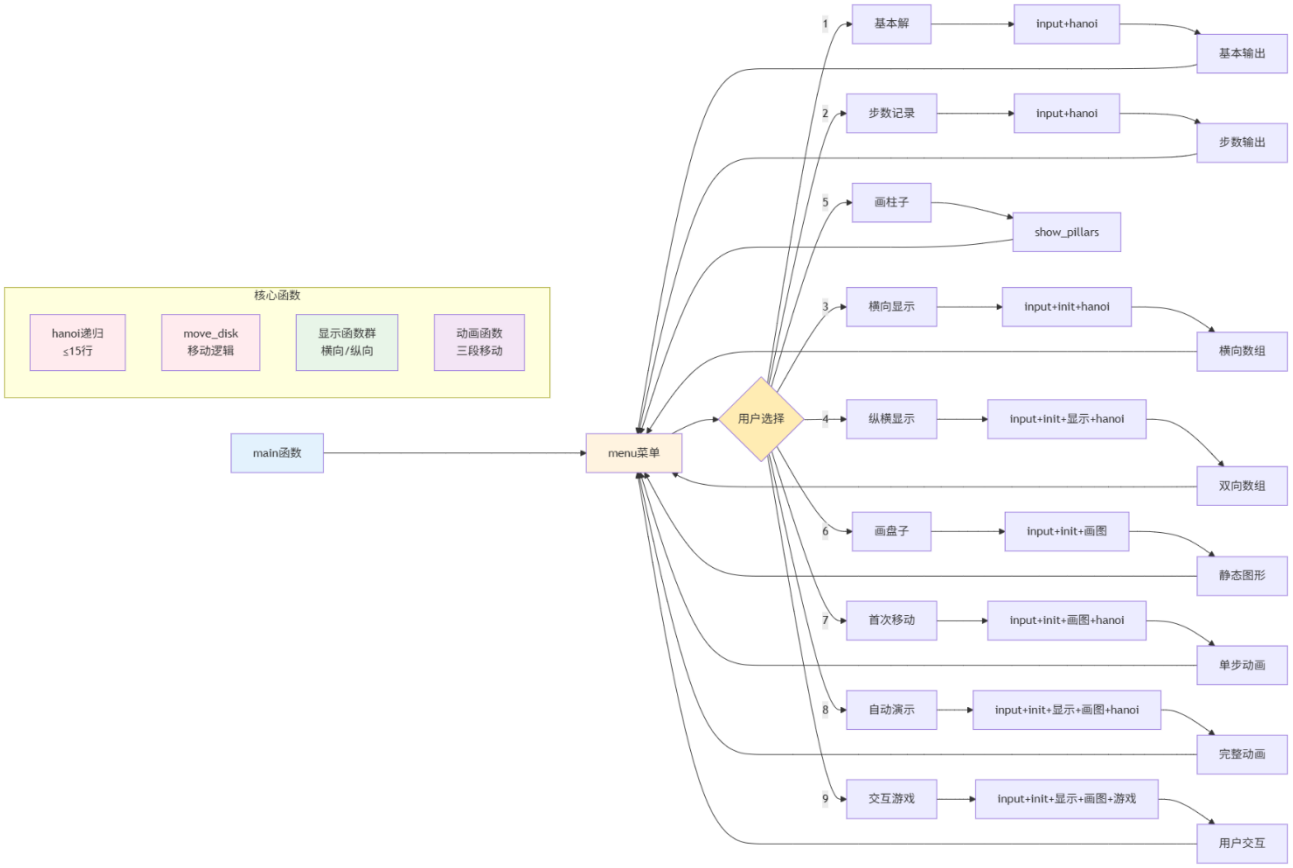
3.1.10. 辅助函数群与解函数

除上述核心函数外，程序还包含多个辅助函数，如 updateStackTop(更新栈顶)、getColumnCenterX(获取柱子中心坐标)、waitForEnter(等待回车键)等，以及九个菜单解函数solve_basic_hanoi()、solve_with_step_count()、solve_shuiping()、solve_both_arrays()、prepare_graphical_pillars()、prepare_graphical_disks()、first_disk_move()、auto_move_hanoi()、interactive_move_hanoi()，用来完成每一个菜单选项的功能。每个解函数都按照具体要求将上述核心函数组合调用，实现对应的功能需求。

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interactive_move_hanoi函数手动进行解决移动柱的问题



4. 调试过程碰到的问题

4.1. 问题1：递归函数参数传递复杂性

在实现核心递归函数时，由于要求所有菜单项（1/2/3/4/8）共用同一个递归函数，需要通过参数控制不同的显示效果。最初设计时参数过多导致函数调用混乱，经常出现参数传递错误。解决方法：重新设计参数结构，使用format_type统一控制输出格式，用布尔值控制是否显示横向、纵向、图形等，使参数意义更加明确，减少了传参错误

4.2. 问题2：栈操作与动画显示不同步

在实现图形动画时，发现栈的逻辑操作和图形显示更新不同步，导致动画效果与实际数据状态不匹配。有时候动画还在播放，但栈数据已经更新完成，造成显示错误。解决方法：在move_disk_animation函数中，将栈操作分为两部分：动画开始前先从源栈移除数据，动画结束后再添加到目标栈，确保动画过程中的数据一致性。

5. 心得体会

5.1.本次作业的经验教训与体会

通过本次汉诺塔综合演示实验，我在多个方面都获得了显著的提升。在技术层面，我系统掌握了console控制台编程和HDC图形编程技术，从简单的文本输出到复杂的动画效果实现，让我对图形界面编程有了深入理解。在程序设计思维方面，我真正体会到了函数复用和模块化编程的优势，通过巧妙的参数设计让单个递归函数支撑多个菜单功能，这种抽象设计能力对我今后的编程实践具有重要指导意义。同时，多文件项目的管理经验让我学会了如何进行有效的模块划分，调试过程中遇到的各种问题也培养了我分析复杂bug的能力和细致的编程习惯。

5.2.复杂的题目希望多小题还是直接一道大题

我认为当前"小题铺垫+大题整合"的设计是最理想的学习方式。多个小题的渐进式学习让我能够逐步掌握递归思想、栈操作、图形编程等独立知识点，每个阶段都有明确的目标和成就感，避免了直接面对复杂问题时的畏难情绪。而最终的综合大题则提供了知识整合的机会，让我在回顾所有技术点的同时思考如何优雅地组织它们，这个过程让我对汉诺塔问题有了全面而深刻的理解。因此，这种教学模式既保证了学习的循序渐进，又提供了综合运用的实践机会，是非常有效的学习安排！

5.3.如何更好地重用代码

代码重用的关键在于合理的抽象设计和模块化思维。在本次实验中，我通过将功能相似的操作抽象成通用函数，如用参数控制不同显示格式的showshuiping函数、通过偏移量适应不同位置的showshuzhi函数，实现了高效的代码复用。核心递归函数hanoi通过format_type、show_horizontal等参数控制不同行为模式，避免了为每个菜单项编写独立函数的冗余。我认为更好地重用代码需要在设计阶段就考虑通用性，善于发现不同操作中的相似部分并进行归纳整合，同时保持参数设计的合理性，这样才能构建出既灵活又易维护的代码结构。

5.4.如何更好利用函数编写复杂程序

编写复杂程序的关键是运用分层分治的思想和模块化的开发方法。我将整个程序分为菜单控制、逻辑处理、数据操作和显示控制等不同层次，每层专注于特定职责，通过明确的接口进行通信。在具体实现中，我采用自底向上的策略，先确保各个独立模块的正确性，再考虑模块间的集成，这种方式有效降低了调试难度。同时，抽象归纳能力也很重要，当发现多个功能存在相似逻辑时，及时进行抽象和统一处理，如将所有移动动画抽象为统一的move_disk_animation函数。良好的函数设计让模块协作变得简单高效，通过精心设计的参数列表使函数具有更好的通用性和扩展性。

6. 附件：源程序

Hanoi.h关键代码

```
// 窗口大小
const int win_width = 1000, win_high = 900;
// 基本尺寸
const int pillar_width = HDC_Base_Width;
const int base_height = HDC_Base_High;
// 计算最大圆盘宽度并确保底座比它大
const int max_layer = MAX_LAYER;
const int max_disk_width = (3 + 2 * (max_layer - 1)) *
pillar_width;
const int base_width = max_disk_width * 11/10;
// 柱子高度
const int pillar_height = (HDC_Start_Y - HDC_Top_Y) * 8 /
10;
// 基本起始坐标
const int start_x = HDC_Start_X;
const int base_y = HDC_Start_Y;
// 三个柱子中心坐标
const int center1_x = win_width / 5;
const int center3_x = win_width * 4 / 5;
const int center2_x = (center1_x + center3_x) / 2;
```

hanoi_main.cpp关键循环

```
char choice;
while (true)
{
	choice = menu();
	if (choice == '0')
	{
		cct_gotoxy(20, 35);
		return 0;
	}
	cout << endl << endl;
	switch (choice)
	{
	case '1':
		solve_basic_hanoi();
		cout << endl;
		break;
	 //2-8 情况类似，省略
	 case '9':
		interactive_move_hanoi();
		cout << endl;
		break;
	}
}
```

hanoi_menu.cpp关键代码

```
char menu()
{
	char choice;
	cct_cls();
	 //页面省略
	 while (true)
	{
		choice = _getch();

		if (choice >= '0' && choice <= '9')
		{
			cout << choice << endl;
			break;
		}
	}
	return choice;
}
```

hanoi_multi_solutions.cpp 关键代码

```
// 全局变量
int totalstep = 0;
int stackA[MAX_LAYER] = { 0 }, stackB[MAX_LAYER] = { 0 },
stackC[MAX_LAYER] = { 0 };
int topA = 0, topB = 0, topC = 0;
int speed = 0;
//初始化
void init(int number, char src)
{
	topA = topB = topC = 0;
	for (int i = 0; i < MAX_LAYER; i++)
		stackA[i] = stackB[i] = stackC[i] = 0;
	switch (src)
	{
	case 'A':
		for (int i = number; i >= 1; i--)
			stackA[topA++] = i;
		break;
	case 'B':
		for (int i = number; i >= 1; i--)
			stackB[topB++] = i;
```

```
                break;
        case 'C':
                for (int i = number; i >= 1; i--)
                        stackC[topC++] = i;
                break;
        }
}
```

// 绘制底座

```
void showdizuo(int baseX, int baseY)
{
        int columnA_X = baseX + Underpan_A_X_OFFSET;
        int columnB_X = columnA_X + Underpan_Distance;
        int columnC_X = columnB_X + Underpan_Distance;
        int lineStartX = columnA_X - 2;
        int lineLength = (columnC_X - columnA_X) + 5;
        cct_gotoxy(lineStartX, baseY);
        for (int i = 0; i < lineLength; i++)
                cout << "=";
        cct_gotoxy(columnA_X, baseY + 1);
        cout << "A";
        cct_gotoxy(columnB_X, baseY + 1);
        cout << "B";
        cct_gotoxy(columnC_X, baseY + 1);
        cout << "C";
}
```

// 横向显示函数

```
void showshuiping(int x, int y, bool simple = false, int
format_type = 0)
{
        if (format_type == 3) // 菜单项 3
        {
                const int fieldWidth = Underpan_Distance;
                cout << " A:";
                for (int i = 0; i < topA; i++)
                        cout << setw(2) << stackA[i];
                for (int i = 1; i < fieldWidth - topA * 2;
i++)
                        cout << " ";
                cout << "B:";
                for (int i = 0; i < topB; i++)
                        cout << setw(2) << stackB[i];
                for (int i = 1; i < fieldWidth - topB * 2;
i++)
                        cout << " ";
                cout << "C:";
                for (int i = 0; i < topC; i++)
                        cout << setw(2) << stackC[i];
                cout << endl;

        }
```

//菜单 4、8 类似处理，省略

// 纵向显示函数

```
void showshuzhi(int baseX, int baseY, char src, char dst,
int speed, int init,
        int colA_offset = 0, int colB_offset = 0, int
colC_offset = 0)
{
        int columnA_X, columnB_X, columnC_X;
        columnA_X = baseX + Underpan_A_X_OFFSET +
colA_offset;
        columnB_X = columnA_X + Underpan_Distance +
colB_offset;
        columnC_X = columnB_X + Underpan_Distance +
colC_offset;
        if (init == 1) //菜单项 4
        {
                showdizuo(baseX, baseY);
        }
```

//菜单 8、9 类似处理，省略

// move_disk 函数，处理选择

```
void move_disk(int number, char src, char dst,
        int format_type, int show_horizontal, int
show_vertical,
        int show_graphical, int automatic)
{
        // 获取源栈信息
        int srcTop = getStackTop(src);
        int dstTop = getStackTop(dst);
        int disk = getStackValue(src, srcTop - 1);
        if (!show_graphical && format_type != 7 &&
format_type != 8)
        {
                // 从源栈移除元素
                setStackValue(src, srcTop - 1, 0);
                updateStackTop(src, srcTop - 1);
                // 添加到目标栈
                setStackValue(dst, dstTop, disk);
```

```
            updateStackTop(dst, dstTop + 1);
        }
        // 处理功能 7：只移动动画，不显示文本
        if (format_type == 7)
        {
            move_disk_animation(disk, src, dst, false);
            return;
        }

        if (format_type == 1)
            cout << number << "# " << src << "---->" <<
dst << endl;
```

//其它 format_type 处理方法类似，省略

// 递归汉诺塔函数

```
void hanoi(int number, char src, char tmp, char dst,
        int format_type, int show_horizontal, int
show_vertical,
        int show_graphical, int automatic)
{
        if (number == 0)
            return;
        hanoi(number - 1, src, dst, tmp, format_type,
show_horizontal, show_vertical, show_graphical, automatic);
        if (format_type == 7 && number > 1)
            return;
        totalstep++;
        move_disk(number, src, dst, format_type,
show_horizontal, show_vertical, show_graphical, automatic);
        hanoi(number - 1, tmp, src, dst, format_type,
show_horizontal, show_vertical, show_graphical, automatic);
}
```

// 绘制柱子

```
void show_pillars()
{
        hdc_init(HDC_COLOR[0], HDC_COLOR[11], win_width,
win_high);
        hdc_set_pencolor(HDC_COLOR[11]);
```

// 绘制三个底座，只展示一个，其他省略

```
        Sleep(HDC_Init_Delay);
        hdc_rectangle(center1_x - base_width / 2, base_y,
base_width, base_height, HDC_COLOR[11]);

        // 绘制三个柱子，只展示一个，其他省略
        Sleep(HDC_Init_Delay);
        hdc_rectangle(center1_x - pillar_width / 2, base_y
- pillar_height, pillar_width, pillar_height,
HDC_COLOR[11]);
}
```

// 绘制圆盘

```
void draw_disk(int center_x, int base_y, int disk_num, int
position)
{
        int disk_width = (3 + 2 * (disk_num - 1)) *
pillar_width;
        int disk_height = base_height;
        int disk_x = center_x - disk_width / 2;
        int disk_y = base_y - disk_height * (position + 1);
        Sleep(HDC_Init_Delay);
        hdc_rectangle(disk_x, disk_y, disk_width,
disk_height, HDC_COLOR[disk_num]);
        Sleep(HDC_Init_Delay);
}
```

//移动盘子动画

```
void move_disk_animation(int disk_num, char src_col, char
dst_col, bool show_text_display)
{
        int src_x = getColumnCenterX(src_col);
        int dst_x = getColumnCenterX(dst_col);
        int disk_width = (3 + 2 * (disk_num - 1)) *
pillar_width;
        int disk_height = base_height;
        int srcTop = getStackTop(src_col);
        int dstTop = getStackTop(dst_col);
        int current_y = base_y - srcTop * disk_height;
        int target_y = base_y - (dstTop + 1) * disk_height;
        updateStackTop(src_col, srcTop - 1);
        //上升动画
        int y;
        for (y = current_y; y >= HDC_Top_Y; y -=
HDC_Step_Y)
        {
            hdc_rectangle(src_x - disk_width / 2, y +
disk_height - HDC_Step_Y, disk_width, HDC_Step_Y,
HDC_COLOR[0]);
            if (y + disk_height > base_y -
pillar_height)
```

```
                hdc_rectangle(src_x - pillar_width / 2, y + disk_height - HDC_Step_Y, pillar_width, HDC_Step_Y, HDC_COLOR[11]);
                if (y > HDC_Top_Y)
                        hdc_rectangle(src_x - disk_width / 2, y - HDC_Step_Y, disk_width, disk_height, HDC_COLOR[disk_num]);
                else
                        hdc_rectangle(src_x - disk_width / 2, HDC_Top_Y, disk_width, disk_height, HDC_COLOR[disk_num]);
                if (speed == 0)
                {
                        Sleep(5);
                        cct_gotoxy(0, Status_Line_Y + 2);
                        Sleep(5);
                        cout << "按回车键继续...";
                        char ch = _getch();
                        while (ch != '\r')
                                ch = _getch();
                        cout << "\r                                \r";
                }
                else
                        Sleep(speed);
        }
        y += HDC_Step_Y;
```

//水平和竖直移动同理，省略

// 处理文本显示，添加延迟避免冲突

```
         if (show_text_display)
        {
                Sleep(15);
                int textBaseY = MenuItem8_Start_Y;
                cct_gotoxy(MenuItem8_Start_X, textBaseY + 1);
                cout << "                                ";
                Sleep(5);
                cct_gotoxy(MenuItem8_Start_X, textBaseY + 1);
                cout << "第" << setw(4) << totalstep << "步(" << setw(2) << disk_num << "#: " << src_col << "-->" << dst_col << ")";
                showshuiping(MenuItem8_Start_X + 30, textBaseY + 1, false, 0);
                Sleep(5);
                showshuzhi(MenuItem8_Start_X, textBaseY + Underpan_A_Y_OFFSET, 0, 0, 0, 1);
                Sleep(5);
        }
}
```

//解法 3

```
void solve_shuiping()
{
    int number;
    char src, tmp, dst;
    input(&src, &tmp, &dst, &number, false);
    init(number, src);
    totalstep = 0;
    hanoi(number, src, tmp, dst, 3, 0, 0, 0, 0);
    waitForEnter();
}
```

//解法 1、2、4 同理，省略

//解法 8

```
void auto_move_hanoi()
{
    int number;
    char src, tmp, dst;
    input(&src, &tmp, &dst, &number, true, 8);
    init(number, src);
    totalstep = 0;
    cct_cls();
    int textBaseY = MenuItem8_Start_Y;
    for (int i = 0; i < 15; i++)
    {
        cct_gotoxy(0, textBaseY - i - 1);
        cout << "                                ";
    }
    cct_gotoxy(Status_Line_X, Status_Line_Y);
    cout << "从 " << src << " 移动到 " << dst << "，共 " << number << " 层，延时设定为 " << speed << "ms" << endl;
    hdc_init(HDC_COLOR[0], HDC_COLOR[11], win_width, win_high);
    show_pillars();
    int center_x = getColumnCenterX(src);
```

```
    for (int i = number; i >= 1; i--)
    {
        int position = number - i;
        draw_disk(center_x, base_y, i, position);
    }
    showshuzhi(MenuItem8_Start_X, MenuItem8_Start_Y +
Underpan_A_Y_OFFSET, 0, 0, 0, 1);
    cct_gotoxy(MenuItem8_Start_X, MenuItem8_Start_Y + 1);
    cout << "初始:";
    showshuiping(MenuItem8_Start_X + 30, MenuItem8_Start_Y
+ 1, false, 0);
    Sleep(10);
    hanoi(number, src, tmp, dst, 8, 1, 1, 1, 1);
    cct_gotoxy(Status_Line_X, Status_Line_Y + 1);
    cout << "移动完成，共" << totalstep << "步          ";
    cct_gotoxy(Status_Line_X, Status_Line_Y + 2);
    waitForEnter();
}
```

//解法 5、6、7 同理，省略

// 解法 9

```
void interactive_move_hanoi()
{
    int number;
    char src, tmp, dst;
    input(&src, &tmp, &dst, &number, true, 9);
    init(number, src);
    totalstep = 0;
    cct_cls();
    int textBaseY = MenuItem8_Start_Y;
    for (int i = 0; i < 15; i++)
    {
        cct_gotoxy(0, textBaseY - i - 1);
        cout << "
";
    }
    showshuzhi(MenuItem8_Start_X, textBaseY +
Underpan_A_Y_OFFSET, 0, 0, 0, 1);
    cct_gotoxy(0, Status_Line_Y);
    cout << "从 " << src << " 移动到 " << dst << ", 共 " <<
number << " 层，延时设定为 " << speed << "ms" << endl;
    show_pillars();
    int center_x = getColumnCenterX(src);
    for (int i = number; i >= 1; i--)
    {
        int position = number - i;
        draw_disk(center_x, base_y, i, position);
    }
    cct_gotoxy(MenuItem8_Start_X, textBaseY + 1);
    cout << "初状:  ";
    showshuiping(MenuItem8_Start_X + 30, textBaseY + 1,
true, 0);
    char move_src, move_dst;
    char command[3];
    while (true)
    {
        cct_gotoxy(0, textBaseY + 4);
        cout << "请输入移动的柱号(命令形式: AC=A 顶上的圆盘
移动到C, Q=退出): ";
        cin >> command;
        cin.clear();
        cin.ignore(2048, '\n');
        if (command[0] == 'Q' || command[0] == 'q')
            break;
        if (command[0] == '\0' || command[1] == '\0' ||
command[2] != '\0' ||
            (command[0] != 'A' && command[0] != 'B' &&
command[0] != 'C' &&
                command[0] != 'a' && command[0] != 'b' &&
command[0] != 'c') ||
            (command[1] != 'A' && command[1] != 'B' &&
command[1] != 'C' &&
                command[1] != 'a' && command[1] != 'b' &&
command[1] != 'c'))
        {
            cct_gotoxy(0, textBaseY + 5);
            cout << "命令错误，请重新输入!" << endl;
            continue;
        }
        move_src = (command[0] >= 'a' && command[0] <=
'c') ? command[0] - 32 : command[0];
        move_dst = (command[1] >= 'a' && command[1] <=
'c') ? command[1] - 32 : command[1];
        if (move_src == move_dst)
        {
            cct_gotoxy(0, textBaseY + 5);
            cout << "源柱(" << move_src << ")和目标柱(" <<
move_dst << ")相同，请重新输入!" << endl;
            continue;
```

```
        }
        // 检查源柱是否为空
        int srcTop = getStackTop(move_src);
        bool source_empty = (srcTop == 0);
        if (source_empty)
        {
            cct_gotoxy(0, textBaseY + 5);
            cout << "源柱(" << move_src << ")上没有圆盘，请
重新输入!" << endl;
            continue;
        }
        int source_disk = getStackValue(move_src, srcTop -
1);
        int dstTop = getStackTop(move_dst);
        int target_top_disk = (dstTop > 0) ?
getStackValue(move_dst, dstTop - 1) : 100;
        if (source_disk > target_top_disk &&
target_top_disk != 100)
        {
            cct_gotoxy(0, textBaseY + 5);
            cout << "非法移动: 不能将大圆盘(" <<
source_disk << ")放在小圆盘(" << target_top_disk << ")上!"
<< endl;
            continue;
        }
        cct_gotoxy(0, textBaseY + 5);
        cout << "
" << endl;
        totalstep++;
        move_disk_animation(source_disk, move_src,
move_dst, true);
        cct_gotoxy(MenuItem8_Start_X, textBaseY + 1);
        cout << "第" << setw(4) << totalstep << " 步(" <<
setw(2) << source_disk << "#: " << move_src << "-->" <<
move_dst << ")      ";
        cct_gotoxy(0, textBaseY + 2);
        cout << "
";
        if (speed > 0)
            Sleep(speed);
        if ((dst == 'A' && topA == number) ||
            (dst == 'B' && topB == number) ||
            (dst == 'C' && topC == number))
        {
            cct_gotoxy(0, textBaseY + 6);
            cout << "恭喜！您已经成功地将所有圆盘从 " <<
src << " 柱移动到了 " << dst << " 柱，共 " << totalstep <<
" 步!" << endl;
            break;
        }
    }
    cct_gotoxy(0, textBaseY + 7);
    waitForEnter();
}
```