装

订

线

高级语言程序设计

汉诺塔综合实验报告

作 者 姓 名：王灏廷

学 号：1953609

学院、 专业：软件学院 软件工程

同济大学

Tongji University

二〇二一年五月二十五日

1. 题目

1.1. 题目综述

将汉诺塔的各种实现方式集成在一个程序中，通过菜单进行选择，同时加入在 cmd 窗口中进行伪图形化界面演示的实现。提供 demo 程序和伪图形化界面工具集源程序。

1.2. 题目综述

整个菜单分别存在以下 10 个功能

1) 汉诺塔基本解，对应题目 4-b7。
2) 汉诺塔基本解，加入总步数记录，对应题目 4-b8。
3) 横向显示汉诺塔问题中对应的内部数组，对应题目 5-b6。
4) 横向与纵向显示汉诺塔内部数组，对应题目 5-b7。
5) 画出图形界面中的三个空柱，首次出现，用以图形化界面的初始化。
6) 在初始柱上画出 n 个盘子，做移动圆盘的准备。
7) 实现第一次移动，为自动移动作铺垫。
8) 实现汉诺塔的图形解，即自动移动。
9) 实现人工操作汉诺塔移动，即游戏版

1.3. 要求与限制

提交以下四个文件：

- hanoi.h : 本项目头文件
- hanoi_menu.cpp : 菜单的显示与选择
- hanoi_multiple_solutions.cpp: 菜单中各项功能的具体实现
- hanoi_main.cpp : main 函数

除下列内容外，其余内容均不允许使用全局变量进行记录：

总移动步数：允许使用 1 个全局简单变量进行记录。

圆柱上现有圆盘的编号：3 个全局一维数组或 1 个全局二维数组。

圆柱上现有圆盘的数量：3 个全局简单变量或 1 个全局一维数组。

延时：一个全局简单变量。

- 整个程序只允许使用一个递归函数，且该递归函数不得超过 15 行（包括独立成行的大括号）。
- 多个参数的输入必须共用一个函数。
- 内部数组的输出，画出三个基本盘，各小盘的移动必须共用一个函数。

2. 整体设计思路

2.1. 汉诺塔问题本身的求解思路

汉诺塔问题本质上是一个递归问题，整体思路通过递归算法解决，分为以下几步：

1.如果 A 柱子只剩一个盘子，那么直接移动到 C 柱子即可。

2.把 n-1 号盘子移动到缓冲区。

3.把 1 号从起点移到终点。

4.然后把缓冲区的 n-1 号盘子也移到终点。

2.2. 抽象化汉诺塔及其移动的思路

将汉诺塔的每个柱抽象为一个一维数组，数组中的元素对应不同标号的盘子。开始时对于数组做遍历初始化操作，将每个盘子对应的标号输入其中。对于移动操作，则将数组理解为栈，由一个柱移动到另一个柱对应着数组元素由现有栈出栈，进入目标栈中，需要移动起始柱与目标柱的栈顶指针。在移动过程中，需要一个简单变量代表栈顶指针，用三个全局变量表示，分别记作 topa,topb,topc，用以指向三个数组顶元素的上一个位置。如此便可以用数组表示汉诺塔的移动。

2.2. 程序整体实现思路

整个程序分为四个部分。

●头文件用于存放所有 cpp 文件中的函数声明，方便 cpp 之间能够互相访问。

●Menu 文件用以显示与调用菜单，并提供选项。

●Main 文件用于初始化屏幕、调用菜单函数并返回选项以及根据选项调用 solutions 中的 initial 函数。

●Solutions 存放被 Main 中根据菜单返回值调用的各菜单项对应的执行函数

Solutions 分为四个部分：

●第一部分是利用递归解决汉诺塔问题，所有功能均通用；

●第二部分是利用“栈”的思想实现数组元素的移动；

●第三部分是通过对于圆盘颜色的设定以及光标的移动，实现伪图形界面下图形的移动和生成；

●第四部分是根据 menu 菜单传入的值进行功能的初始化，负责调用以上三个不同的部分。

3. 主要功能的实现

3.1. 利用递归的方法解决汉诺塔问题

设共有 n 个盘子。由于每一步都相当于借助中间柱移动前 n-1 个盘子和第 n 个盘子，故利用递归解决汉诺塔问题。即设置条件，当 n>1 时调用函数 hanoi(n-1)，并以此类推；

当 n=1 时，则回溯，输出每一步的搬运过程。

3.2. 利用数组和栈显示每根柱子上盘子的情况并实现移动

3.2.1. 初始化汉诺塔

●定义 3 个全局一维数组 A[10]，B[10]，C[10] 来存放盘子的编号进而代表盘子的情况，其中 ABC 分别为柱的编号，10 代表问题中最多出现 10 个盘子。

●定义 3 个全局简单变量 top_a ， top_b ， top_c 来代表当前的栈顶位置,栈顶位置始终指向元素的待插入位置，即现有元素的上一个，便于后续的插入和移出。

●初始化时，以起始柱为 A 柱，共 3 个盘子举例：利用数组遍历将数组 A 初始化为：{3，2，1...}（其余元素均为 0），数组 B 与 C 均为全 0 数组，此时完成了移动前的准备工作。

3.2.2. 实现盘子的移动

●盘子的移动通过数组对应元素的“出栈”与“入栈”来实现。

●以 A 为出发柱，B 为接受柱举例：A 的栈顶元素出栈，此时 topa--，该元素进入 B 中，topb++，即 A[--topa] = B [topb++]

●输出时每个数组循环 10 次,用以保证所有盘都正常输出,若无盘则输出两个空格用以代替。

3.2.3. 使用图形化界面时盘子的移动

●利用头文件中的 cct_showch() 函数和 cct_gotoxy() 函数，在不同的位置输出字符并调整背景色，加入延时并隐去光标即可以实现伪图形化界面中盘子的移动。

●移动可以分解为上移，平移，下移。通过传入起始柱与目标柱来决定如何移动，故整个过程的六种可能都要考虑到，每种可能性的起始坐标、目标坐标与调用参数均不同，通过传入不同的 gotoxy 参数解决，只要相对位置是确定的，过程就都是大同小异的。

●上移次数根据起始柱栈顶指针决定，下移次数则根据目标柱栈顶指针决定，由柱高减去指针值即可得到，而平移的次数则又起始柱与目标柱的相对位置决定，每种情况是固定的参数，与其他参数无关，循环固定次数即可解决。

●延时功能通过 Windows.h 中的 Sleep 函数实现。输入数字越大延时越短。

●移动结束后需要将背景色与字体颜色调回黑白，否则会以圆盘颜色输出之后的内容，什么也看不到；光标也需要调回正常，否则后续无法正常显示光标。

3.2.4. 递归函数之外的调用

由于共用的递归函数不得超过 15 行，故每个独立的功能需要调用的不同函数必须放在递归函数之外，否则将会大大超标。因此我在递归函数之外定义了 transverse（）函数，通过输入功能的不同而选择调用不同的函数，用以实现不同的功能。而主递归函数 hanoi（）就只需要调用

transverse（）即可，大大提高了递归的整洁性。

3.2.5. 延时函数

延时函数需要实现 0~5 这几种不同的延时功能。总体来讲可以分为两个部分：

●1~5 实现不同的延时效果。使用 switch 判断输入的延时值，调用 Sleep 函数休眠由短及长的时间即可。

●0，实现按键控制移动功能。需要调用_getch()函数，读取输入的回车，若读到回车，则进行下一步，否则一直等待输入。

3.2.6. 游戏版控制移动功能

即游戏中的功能 9。需要读取输入的按键并通过按键控制圆盘的移动。

●我使用了两次 cin.get()来分别读取起始柱与目标柱，然后通过不停的 getchar（）直到回车，达到清空输入缓冲区的目的。若 getchar 读到了字符，说明输入超过了两个，此时直接清空输入并开始下一次输入，保证只读两个字符。

●若输入小写，则将小写转换为大写，然后进行两种判断：

1. 原柱为空。通过栈顶指针来判断，若起始柱栈顶指针为 0，说明起始柱中没有圆盘，此时输出原柱为空并清空输入区，直接开始下一次读取。

2. 大盘压小盘，非法移动。同样通过栈顶指针来判断，若起始柱指针指向的栈顶值大于目标柱指针指向的栈顶值，则说明移动后是大盘压小盘，此时输出大盘压小盘，非法移动，并清空输入区，直接开始下一次读取。

4. 调试过程碰到的问题

4.1. 游戏版控制移动功能读取问题

●起初我使用的是_getch()进行读取功能，这样方便实现游戏化的输入，不需要回车即可读到。然后发现问题多多：

1. 起初的逻辑是读到 ABCabc 才输出在屏幕上，后来在调试过程中发现第一次是正确的，但是在之后的次数中缓冲区里还有其他字符，无法正确读到需要的字符。或者输入了”BD”，然后起始柱读到了 B，目标柱读取错误，但是并没有任何显示和清除，也无法输入其他字符。

2. 其次是不好清除。因为输入不会全部打在屏幕上，所以怎样清除、清除多少就成为一个很大的问题，没法解决。

最终直接放弃了_getch()，改用 cin.get（）读取数据，getchar（）清空缓冲区，然后把屏幕上前后两行清空，完美解决。

4.2. 游戏版控制“大盘压小盘”判断

●本来觉得通过栈顶指针可以轻松解决。调试时发现栈顶指针指向的是数组现有元素的上一个，所以想直到栈顶元素指针值需要减 1，但指针值可以是 0，若减 1 则越界，VS 会报 warning。

最终在每一个判断前加入一个是否大于 0 的判断，等于 0 就跳过，大于 0 才判断，最终解决了这个 warning。

4.3.图形化自动移动的函数顺序问题

●同样函数写得很顺利，在 transverse（）中判断出了大问题。

在 Demo 中自动移动时，是横纵向显示的数组先移动，然后图形化的圆盘才移动。但如果这样做的话，数组内容和栈中内容在横纵向显示中就改变了，在图形化移动时调用的是移动后的参数，会导致出现错误，但是这二者之间的前后顺序又无法调整，因为是横纵向显示移动先于图形化，所以想了很多解决方法，比如记录上一个状态的值等等，但是尝试都失败了。

最终迫不得已，调用了控制数组和栈移动的函数三次：横纵向显示先调用一次，然后再把移动过的盘再移回原柱，再调用图形化移动函数，再把盘子移到目标柱。虽然这种方法很笨，但是解决效果还是良好的。

5.心得体会

5.1.在完成本次作业中的心得体会

1)反复出现的常量应该用常变量或者宏定义的方式写在头文件中，便于维护。

2)变量命名要准确易于理解，最好使用匈牙利命名法或驼峰命名法。

3)在源程序中要善于写注释，并且对不同的函数和不同的定义做分类以及说明，减少维护线程序时浪费的时间。在同一函数中也要注意用空行来区分不同代码块，便于区分。

4)要善于构建临时测试，并学会打断点，边写代码边做测试，这样可以大大节省后续调试的时间，同时也不会因为程序过于庞大而找不到错误在哪。

5)充分思考后再开始写代码，增加代码精巧程度，减少无效代码和垃圾代码。

5.2.对于较复杂程序编写的体会

●一个庞杂的程序由很多功能构成，如果不化整为零的话很容易造成无从下手的局面。捡了东头丢了西头，十分不利于整体框架的构建。

●在编写较复杂程序时，要善于将大的功能拆分成为小的模块，首先实现各个模块的功能，然后再将不同模块封装成为大的模块，最终实现与其功能。

●同样，复杂程序也不一定是只由一个 cpp 构成的，使用多个 cpp 和头文件的声明有利于分解不同功能，将不同的功能在不同的文件中实现，也能够有效地化繁为简，使任务简单化。

5.3.汉诺塔作业总结

5.3.1. 对于各题前后关系以及已有代码的利用

这次作业中我充分利用了各题的前后关系以及前题的有效代码，相对减少了时间量。

●之前做过的作业可以直接解决 1-4 的功能，但是在实现后续功能时，需要在一些函数中增加调用参数。前后小题的本质都是汉诺塔的移动，核心思想是同样的，所以在增加调用参数的情况下，移动或者递归时可以做到代码复用。

●就此，我觉得每个函数应当尽可能实现一个小的、具体的功能，不应把很多不同的功能杂糅在一起，这样会导致后续需要实现相似功能时，因为功能不完全相同而无法直接调用，还需要重新写一个大部分重复的函数，得不偿失。每个函数只要实现了核心思想，就不需要同时完成很多功能，而是应该让它们各司其职，互相调用，这样可以大大减少重复代码量。

5.3.2. 对于代码重用的感悟

●好的代码应当是复用性强的代码。代码中的共性完全可以提炼出来，写成一个具体的函数，这样之后遇到相似思路问题时便可以调用求解；同时如果考虑到之后的使用的话，预留接口和参数也十分重要，这样方便了将一个小函数改造成一个解决整体问题的大函数，可以有效的节约代码编写的时间和空间，减少冗余代码的产生。

5.4.对于函数利用的感悟

●通过这次的作业，我充分体会到了提炼函数的重要性。比如在这次作业的输入处理中，不同类型的变量可以通过编写对应类型的输入处理函数来实现输入，而提示信息又可以通过装一个字符串参数来解决，这样做大大提高了效率。又比如图形化界面中盘子的移动函数，面对不同对象操作完全可以通过同一函数实现，极大的提高了代码效率。

●我认识到，要对于重复代码多加思考，提炼共性使之成为函数，不同之处利用参数来解决。可以使得代码简洁而有效。

6.附件：源程序

6.1.hanoi.menu.cpp

```
char menu()
{
	cct_setconsoleborder(120, 40, 120, 9000);
	cct_cls();
	cout << "--------------------------------" << endl
		<< "1.基本解" << endl
		<< "2.基本解(步数记录)" << endl
		<< "3.内部数组显示(横向)" << endl
		<< "4.内部数组显示(纵向 + 横向)" << endl
		<< "5.图形解 - 预备 - 画三个圆柱" << endl
		<< "6.图形解 - 预备 - 在起始柱上画 n 个盘子" << endl
		<< "7.图形解 - 预备 - 第一次移动" << endl
		<< "8.图形解 - 自动移动版本" << endl
		<< "9.图形解 - 游戏版" << endl
		<< "0.退出" << endl
		<< "--------------------------------" << endl
		<< "[请选择:] ";
	while (1)
	{
		const char num = _getch();
		if (num<'0'||num>'9')
		{
			cin.clear();
			continue;
		}
		return num;
	}
}
```

6.2.hanoi.main.cpp

```
int main()
{
	/* demo 中首先执行此句，将 cmd 窗口设置为 40 行 x120 列（缓冲区宽度 120 列，行数 9000 行，即 cmd 窗口右侧带有垂直滚动杆）*/
	cct_setconsoleborder(120, 40, 120, 9000);
	while (1)
	{
		const int select = menu();
		switch (select)	//输入参数 根据输入分别处理
		{
			case '0':
				cout << endl;
				return 0;
			case '1':
				initial(select);
```

```
                break;
            case '2':
                initial(select);
                break;
            case '3':
                initial(select);
                break;
            case '4':
                initial(select);
                break;
            case '5':
                initial(select);
                cct_cls();
                print_disk();
                break;
            case '6':
                initial(select);
                break;
            case '7':
                initial(select);
                break;
            case '8':
                initial(select);
                break;
            case '9':
                initial(select);
                break;
            default:
                break;
        }
        cout << endl << "按回车键继续" << endl;
        int cmd;
        while (1)
        {
            cmd = _getch();
            if (cmd == 13)
            {
                cct_cls();
                break;
            }

        }
    }
    return 0;
}
```

6.3. hanoi_multiple_solutions.cpp 部分函数

6.3.1. hanoi 递归函数

```
void hanoi(int n, char src, char tmp, char dst,char sign)
{
    if (n == 1)
        transverse(n,src,tmp,dst,sign);
```

```
        else
        {
            hanoi(n - 1, src, dst, tmp,sign);
            transverse(n, src, tmp, dst,sign);
            hanoi(n - 1, tmp, src, dst,sign);
        }
    }
```

6.3.2.各分支调用函数

```
    void transverse(int n, char src, char tmp, char dst,char sign)
    {
        if (sign == '1' || sign == '2')
            print_move_one(n,src,dst,sign);
        if (sign == '3')
        {
            print_move(n, src, dst, sign);
            move(src, dst);
            print();
            return;
        }
        if (sign == '4')
        {
            wait();
            print_move(n, src, dst, sign);
            move(src, dst);
            print();
            wait();
            print_tower(sign);
        }
        if (sign == '8')
        {
            wait();
            print_move(n, src, dst, sign);
            move(src, dst);
            print();
            print_tower(sign);
            move(dst, src);
            disk_move(n, src, dst, sign);
            move(src, dst);
        }
    }
```

6.3.2.汉诺塔的移动与数组的打印

```
void move(char src, char dst)
{
    int temp = 0;
    switch (src)
    {
        case 'A':
            temp = A[topa - 1];
            A[topa - 1] = 0;
            topa--;
```

```
                break;
            case 'B':
                temp = B[topb - 1];
                B[topb - 1] = 0;
                topb--;
                break;
            case 'C':
                temp = C[topc - 1];
                C[topc - 1] = 0;
                topc--;
                break;
            default:
                break;
        }
        switch (dst)
        {
            case 'A':
                A[topa] = temp;
                topa++;
                break;
            case 'B':
                B[topb] = temp;
                topb++;
                break;
            case 'C':
                C[topc] = temp;
                topc++;
                break;
            default:
                break;
        }
}
void print()
{
        int i;
        cout << "A:";
        for (i = 0; i < 10; i++)
        {

            if (A[i] == 0)
                cout << "    ";
            else
                cout << setw(2) << A[i];
        }
        cout << " B:";
        for (i = 0; i < 10; i++)
        {

            if (B[i] == 0)
                cout << "    ";
            else
                cout << setw(2) << B[i];
        }
        cout << " C:";
        for (i = 0; i < 10; i++)
```

装 订 线

```
    {
        if (C[i] == 0)
            cout << "    ";
        else
            cout << setw(2) << C[i];
    }
    cout << endl;
}
void print_move(int n, char src, char dst,char sign)
{
    if (sign == '4')
        cct_gotoxy(0, 17);
    else
        cct_gotoxy(0, 33);
    cout << "第" << setw(4) << num << "步" << "(" << setw(2) << n << "#: " << src << "-->" << dst
<< ")    ";
    num++;
}
```

6.3.4. 图形化界面相关函数

```
void print_disk()
{
    for (int i = 0; i < 3; i++)
    {
        cct_showch(2 + 30 * i, 17, 0, 14, 14, 25);
    }
    for (int j = 0; j < 15; j++)
    {
        cct_showch(14, 17 - j, 0, 14, 14, 1);
        cct_showch(44, 17 - j, 0, 14, 14, 1);
        cct_showch(74, 17 - j, 0, 14, 14, 1);
        Sleep(50);
    }
    cct_gotoxy(0, 35);
    cct_setcolor(0, 7);
}
void print_disk_ini(char src,int sum)
{
    int sum1 = sum;
    if (src == 'A')
    {
        for (int i = 0; i < sum; i++)
        {
            cct_showch(14 - sum1, 16 - i, 0, sum1, sum1, sum1 * 2 + 1);
            Sleep(50);
            sum1--;
        }
    }
    else if (src == 'B')
    {
        for (int i = 0; i < sum; i++)
        {
```

```
                cct_showch(44 - sum1, 16 - i, 0, sum1, sum1, sum1 * 2 + 1);
                Sleep(50);
                sum1--;
            }
        }
        else if (src == 'C')
        {
            for (int i = 0; i < sum; i++)
            {
                cct_showch(74 - sum1, 16 - i, 0, sum1, sum1, sum1 * 2 + 1);
                Sleep(50);
                sum1--;
            }
        }
        cct_gotoxy(0, 35);
        cct_setcolor(0, 7);
}
```

6.3.5.延迟函数

```
void wait()
{
    switch (moving)
    {
        case 0:
            int cmd;
            while (1)
            {
                cmd = _getch();
                if (cmd == 13)
                    break;
            }
            break;
        case 1:
            Sleep(1000);
            break;
        case 2:
            Sleep(500);
            break;
        case 3:
            Sleep(200);
            break;
        case 4:
            Sleep(100);
            break;
        case 5:
            Sleep(50);
            break;
        default:
            break;
    }
}
```