同济大学

实验报告

装

订

线

# 面向对象实验报告

## ——汉诺塔综合演示

**班级：** [redacted]

**学号：** [redacted]

**姓名：** [redacted]

**完成日期：2020.11.25**

# 1. 题目

实现一个汉诺塔综合演示。菜单如下：

1.基本解
2.基本解(步数记录)
3.内部数组显示(横向)
4.内部数组显示(纵向 + 横向)
5.图形解-预备-画三个圆柱
6.图形解-预备-在起始柱上画n个盘子
7.图形解-预备-第一次移动
8.图形解-自动移动版本
9.图形解-游戏版
0.退出

装

订

线

## 1.1.菜单项1：

实现汉诺塔基本的移动，包括移动的盘号、源柱和目标柱。

## 1.2.菜单项2：

实现汉诺塔基本的移动，包括移动的盘号、源柱和目标柱，并进行对步数的计数。

## 1.3.菜单项3：

实现汉诺塔基本的移动，包括移动的盘号、源柱和目标柱，对步数进行计数，并通过横向数组展示内部的组成，随着汉诺塔的移动实时切换状态。

## 1.4.菜单项4：

实现汉诺塔基本的移动，包括移动的盘号、源柱和目标柱，对步数进行计数，并通过横向、纵向数组展示内部的组成，随着汉诺塔的移动实时切换状态。

## 1.5.菜单项5：

在屏幕上画出三根圆柱，添加延时以体现过程。

## 1.6.菜单项6：

将三根源柱从左到右编号ABC，根据输入圆盘的数量、起始柱和目标柱画出指定个数的圆盘，

每个圆盘的颜色各不相同，添加延时以体现过程。

## 1.7.菜单项7：

在菜单6的基础上，完成第一个圆盘的移动，适当添加延时模拟移动效果，移动方式先上移、再平移、再下移。

## 1.8.菜单项8：

汉诺塔演示过程的完整实现，移动方式：先上移、再平移、再下移。

## 1.9.菜单项9：

汉诺塔游戏：

- 每次键盘输入两个A-C之间的字母，表示移动的源柱和目标柱。
- 移动检查合理性，若不符合移动规则（大盘压小盘、源柱为空等），提示出错并重输，每次合理的移动必须记录步数，并显示每次移动的盘子编号。
- 每次圆盘的移动方式必须是上移、平移、下移。
- 待所有盘子按序移动到结束柱则提示游戏结束

## 1.10. 菜单项0：

退出程序。

注：

1、菜单项1/2/3/4/8必须共用一个递归函数，且整个程序只允许使用一个递归函数，用参数解决菜单项之间的差异，递归函数不得超过15行。

2、菜单项1/2/3/4/6/7/8/9中的输入多个参数必须共用一个函数。

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。

4、菜单项4/8中的总想输出必须共用一个函数，用参数解决输出位置等擦会议。

5、菜单项5/6/7/8/9中画三个柱子必须共用一个函数。

6、菜单项7/8/9中盘子的移动必须共用一个函数。

7、共用函数中均允许调用其他函数。

8、1-4所需函数参数个数、类型等不受限制。

9、其他无强制要求。

# 2. 整体设计思路

总体思路如下：

main程序调用菜单函数、输入函数等部分。

menu程序展示菜单，返回输入的菜单选项。

hanoi_multiple_solutions程序实现整个程序的主要功能，包括汉诺塔递归函数、打印横向数组、打印纵向数组、打印柱子、打印盘子以及各种移动操作。

菜单项1调用共用函数，其中包含数组初始化函数。

菜单项2调用共用函数，其中包含数组初始化函数。

菜单项3调用共用函数，其中包含数组初始化函数、横向打印数组函数。

菜单项4调用共用函数，其中包含数组初始化函数、横向打印数组函数、纵向打印数组函数。

菜单项5调用共用函数，其中包含数组初始化函数、打印柱子函数。

菜单项6调用共用函数，其中包含数组初始化函数、打印柱子函数、打印盘子函数。

菜单项7调用共用函数，其中包含数组初始化函数、打印柱子函数、打印盘子函数、移动盘子函数。

菜单项8调用共用函数，其中包含数组初始化函数、打印柱子函数、打印盘子函数、移动盘子函数、横向打印数组函数、纵向打印数组函数。

菜单项9调用共用函数，其中包含数组初始化函数、打印柱子函数、打印盘子函数、移动盘子函数。

装 订 线

# 3. 主要功能的实现

## 3.1.初始化部分：

根据输入的起始柱和目标柱，决定src、tmp、dst柱，根据输入的层数等，初始化数组。数组模拟栈的操作。用一个一维数组记录栈顶指针指向的下标，用一个二维数组代表柱子，来存放盘子的编号。输入起始柱，填充对应的数组，有多少层就有多少个元素，从第一个元素起依次递减。

## 3.2.汉诺塔递归函数：

利用递归思想，实现汉诺塔盘子的移动的过程。

## 3.3.数组变化操作：

模拟出栈和入栈操作。指针起初均指向栈顶，则最上面的那个盘子的上方。若有盘子的移动，删去源柱数组最靠后的元素，将该元素添加到目标柱数组的末尾。此时，指向源柱栈顶的指针向下移动一个元素，指向目标柱栈顶的指针向上移动一个元素。此时一次移动的操作完成。

## 3.4.根据菜单项的不同选择进行不同的输出：

由于菜单项不同会导致输出差异，因此在汉诺塔递归函数中调用控制输出的函数。各菜单项调用的各个函数如“整体设计思路”模块所示。

## 3.5.横向打印部分：

横向打印数组，每次汉诺塔的移动都会引起该横向打印的变化。主要思路是数组有盘子的位置用盘子的编号表示，没有盘子的位置都用0来代替。若数组的元素不为0，就将盘子的编号以从柱底到柱顶的顺序以2位宽度输出；若数组的元素为0，不输出该元素，而是会输出2位宽度的空格。此处在数组变化操作之后进行。

## 3.6.纵向打印部分：

纵向打印数组，每次汉诺塔的移动都会引起该纵向打印的变化。主要思路是自下而上地打印数组中从柱底到柱顶的元素。当数组有变化的时候，纵向打印会在目标柱的柱顶加上一个盘子的编号，而源柱最上方的元素会被空格抹去。

## 3.7.打印柱子部分：

三个柱子同时打印，通过cct_showch函数实现。将输出的字符设置为空格，将背景颜色设置为亮黄色，并通过cct_showch的参数传递来控制柱底的长度。上方的柱子可通过一个循环，将柱子每一行的一小块一个个输出。

## 3.8.打印盘子部分：

经观察示例程序，可发现盘子可分为三部分：最中间的一小块以及中间小块左右两边的长条部分。而长条的长度恰好为盘子编号的大小。因此打印盘子的起始横坐标=中间小块的横坐标-盘子编号，而盘子长度=2*盘子编号+1，这样就能通过cct_showch函数画出对应编号的盘子。

## 3.9.菜单9游戏部分：

该部分主要是输入处理和移动处理两部分。

## 3.9.1.输入处理：

·输入正确：输入的前两个字符是A-C（不区分大小写）之间的字符，后一个字符是回车，此时调用模拟栈的函数和移动盘子的函数，完成一次移动。

·输入非A-C之间的字符，程序判断出前两个字符不合法，当输入第三个字符时清除缓存区，输入过的字符全部清除。

·输入两个字符后未按回车，程序判断第三个字符非回车，清除缓存区，并且清除所有输入过的字符。

## 3.9.2.移动处理：

当输入合法并正确时，调用移动函数，将第一个字符对应柱子上的盘子移动到第二个字符对应柱子上。

# 4. 调试过程碰到的问题

## 4.1.数组模拟栈的操作

原本的方案是将栈顶指针直接指向数组的最后一个元素，但是在移动过程中出现了问题，当在处理空栈（即柱子上没有盘子的情况）时栈顶指针会是负数。因此经过修改，将栈顶指针指向了最后一个元素后一个元素的位置。栈的操作同时在移动顺序方面也有需要注意的地方。起初移动的顺序出现了问题，输出会输出0.后来将顺序改为：源柱最上方盘子赋给目标柱最上方，源柱的栈顶指针减一，源柱最上方盘子的位置置零，目标柱的栈顶指针加一，按照这个顺序，最终结果正确。

## 4.2.盘子上移和下移的操作

在起初画盘子上移和下移的动画时，盘子上移和下移可能会导致柱子的消失。因此在画盘子上移下移的过程中，除了要用黑色背景盖住之前的位置，还要在原来的位置中间添加一块亮黄色，保证柱子的完整。因此我又改善了画盘子的方法。用最中间那一块定位盘子，再画出整个盘子的形状，较好地完成了正常的盘子上移和下移操作。

# 5. 心得体会

在汉诺塔综合演示的制作过程中，我有以下一些心得体会：

1、在写程序之前，对程序的整体框架要有完整的把握。起初在写这个程序之前，我并没有对这个程序有一个比较完整的概念，只是想到哪写到哪，导致写出来的最初成品非常杂乱，函数之间的嵌套关系非常混乱模糊，给后期的代码修改和维护造成了很大的不便，浪费了很多无谓的时间和精力。

2、对于复杂的程序，在把握全局的同时，也要将程序分成各个小部分来进行，使程序清晰、有条理。我们这次的汉诺塔作业就是这么进行的，由起初的非常基本的汉诺塔程序，渐渐的添加修饰，变成了一个丰富完整的汉诺塔演示程序。

3、养成写注释的习惯非常重要。我在之前没有写注释的习惯，在写汉诺塔程序之前也没有写很多注释。因此在写程序的时候，过两天就忘了自己前两天写的是什么了。整体的感受就是思路非常杂乱无章，为写后续程序带来了障碍。

4、对于函数的设计，将一个很大的程序分割成很多有着特定功能的小函数效果会更佳。以此次汉诺塔程序为例，打印柱子、打印盘子、移动盘子、递归函数等的功能各自分配给多个函数，将不同的功能通过分割成不同的函数，使思路更加清晰明了，有利于理解程序并更好地实现功能，后期修改和完善也更加方便。

5、此次的作业中，很多后续的菜单选项都可以用到之前的函数，可以有效地减少重复冗余地代码。以此次的汉诺塔程序为例，汉诺塔递归函数、输入参数函数、打印柱子、打印盘子等功能都可以被各个菜单项反复利用，尤其是汉诺塔递归函数，贯穿了我们整个编写汉诺塔程序的过程。

## 6. 附件：源程序

初始化部分：

```
void initial(char src, int layer, int order)
{
    if (order == 4 || order == 9)
        cct_gotoxy(0, 17);
    if (order != 5 && order != 6 && order != 7)
    {
        if (order == 8 || order == 9)
            cct_gotoxy(0, 32);
        cout << "初始:  ";
    }//通过参数区分输出区别
    if (src == 'A')
    {
        capsule[0][0] = layer;
        pointer[0] = layer;
        if (order == 4 || order == 8 || order == 9)
            cout << " A:" << setw(2) << capsule[0][0];
        for (int i = 1; i < layer; i++)
        {
            capsule[0][i] = capsule[0][i - 1] - 1;
            if (order == 4 || order == 8 || order == 9)
                cout << setw(2) << capsule[0][i];
        }
        for (int i = layer; i < 10; i++)
            if (order == 4 || order == 8 || order == 9)
                cout << "  ";
        if (order == 4 || order == 8 || order == 9)
        {
            cout << " B:
";
            cout << " C:" << endl;
        }
    }//分别进行初始化操作，若选择了显示内部数组，则输出各个盘子内部的数字
    else if (src == 'B')
    {
        capsule[1][0] = layer;
        pointer[1] = layer;
        if (order == 4 || order == 8 || order == 9)
        {
            cout << " A:
";
            cout << " B:" << setw(2) << capsule[1][0];
        }
        for (int i = 1; i < layer; i++)
        {
            capsule[1][i] = capsule[1][i - 1] - 1;
            if (order == 4)
                cout << setw(2) << capsule[1][i];
        }
        if (order == 4 || order == 8 || order == 9)
        {
            for (int i = layer; i < 10; i++)
                cout << "  ";
            cout << " C:" << endl;
        }
    }
    else if (src == 'C')
    {
        capsule[2][0] = layer;
        pointer[2] = layer;
        if (order == 4 || order == 8 || order == 9)
        {
            cout << " A:
";
            cout << " B:
";
            cout << " C:" << setw(2) << capsule[2][0];
        }
        for (int i = 1; i < layer; i++)
        {
            capsule[2][i] = capsule[2][i - 1] - 1;
            if (order == 4 || order == 8 || order == 9)
                cout << setw(2) << capsule[2][i];
        }
        if (order == 4 || order == 8 || order == 9)
        {
            for (int i = layer; i < 10; i++)
                cout << "  ";
        }
    }//初始化操作
}
```

汉诺塔递归实现：

```
void hanoi(int n, char src, char tmp, char dst, int order)
```

```
{
    if (n == 1)
    {
        step++;
        hanoiSolutions(n, src, tmp, dst,
order);
    }
    else
    {
        hanoi(n - 1, src, dst, tmp,
order);//移到tmp,dst作辅助塔
        step++;
        hanoiSolutions(n, src, tmp, dst,
order);
        hanoi(n - 1, tmp, src, dst,
order);//移到dst,src作辅助塔
    }
}
```

数组变化操作：

```
void move(char src, char dst)
{

    capsule[dst - 'A'][pointer[dst - 'A']] =
capsule[src - 'A'][pointer[src - 'A'] - 1];
    pointer[src - 'A']--;
    capsule[src - 'A'][pointer[src - 'A']] =
0;
    pointer[dst - 'A']++;//用数组模拟栈的操
作，pointer指针指向最后一个元素的后一个位置。
若有移动操作，则模拟出栈入栈
}
```

根据菜单选项的不同选择不同输出方式：

```
void hanoiSolutions(int n, char src, char tmp,
char dst, int order)
{
    if (order == 1)
        cout << setw(2) << n << "# " << src
<< "-->" << dst << endl;
    if (order == 2)
        cout << setw(5) << step << ":" <<
setw(3) << n << "# " << src << "-->" << dst
<< endl;
    if (order == 3)
    {
        cout << "第" << setw(4) << step <<
"步(" << n << "#: " << src << "-->" << dst <<
") ";
        move(src, dst);
        print();
    }
    if (order == 4)
    {
        move(src, dst);
        printSteps(n, src, dst, order);
        printVer(n, src, dst, order);//先
输出横向，再输出纵向
    }
    if (order == 8)
    {
        if (!interval)
        {
            while (getchar() != '\n')
                ;
        }
        movePlate(12 + 32 * (src - 'A'), 14
- pointer[src - 'A'], src, dst, order);
        move(src, dst);
        printVer(n, src, dst, order);
        printSteps(n, src, dst, order);//
先对伪图形界面进行画图，再横向、纵向打印
    }
}
```

移动盘子部分：

```
void movePlate(int x, int y, char src, char
dst, int order)//盘子的移动操作
{
    cct_setcursor(CURSOR_INVISIBLE);
    int color = capsule[src -
'A'][pointer[src - 'A'] - 1];
    cct_gotoxy(x, y - pointer[src - 'A']);
    while (y > 0)
    {
        if (order != 9)
        {
            if (interval)
                Sleep(500 - interval *
99);//根据输入的延时参数调整延长时间
            else
            {
                Sleep(100);
            }
        }
        printPlate(x, y, capsule[src -
'A'][pointer[src - 'A'] - 1], color);
        printPlate(x, y + 1, capsule[src -
'A'][pointer[src - 'A'] - 1], COLOR_BLACK);
        if (y != 1)
            cct_showch(x, y + 1, ' ',
COLOR_HYELLOW, COLOR_HYELLOW, 1);
        y--;
    }
    y++;
    if (src < dst)//对源柱和目标柱的位置进
行分类讨论
    {
```

装 订 线

```
        while (x < 12 + 32 * (dst - 'A'))
        {
            if (order != 9)
            {
                if (interval)
                    Sleep(500 - interval
* 99);
                else
                {
                    Sleep(100);
                }
            }
            cct_showch(x - capsule[src -
'A'][pointer[src - 'A'] - 1], y, ' ',
COLOR_BLACK, COLOR_BLACK, 1);
            x++;
            printPlate(x, y, capsule[src -
'A'][pointer[src - 'A'] - 1], color);
        }
    }
    else
    {
        while (x > 12 + 32 * (dst - 'A'))
        {
            if (order != 9)
            {
                if (interval)
                    Sleep(500 - interval
* 99);
                else
                {
                    Sleep(100);
                }
            }//延时
            cct_showch(x + capsule[src -
'A'][pointer[src - 'A'] - 1], y, ' ',
COLOR_BLACK, COLOR_BLACK, 1);//移动的时候，
原来有的位置需要被黑色填上
            x--;
            printPlate(x, y, capsule[src -
'A'][pointer[src - 'A'] - 1], color);
        }
    }
    while (y < 15 - pointer[dst - 'A'])
    {
        if (order != 9)
        {
            if (interval)
                Sleep(500 - interval *
99);
            else
            {
                Sleep(100);
            }
        }
        printPlate(x, y, capsule[src -
'A'][pointer[src - 'A'] - 1], color);
        if (y != 1)
            printPlate(x, y - 1,
capsule[src - 'A'][pointer[src - 'A'] - 1],
COLOR_BLACK);
        if (y != 1 && y != 2 && y != 3)
            cct_showch(x, y - 1, ' ',
COLOR_HYELLOW, COLOR_HYELLOW, 1);//当移动的
时候，需要还原之前被盖住的柱子
        y++;
    }
}
```

横向打印部分：

```
void print()
{
    for (int i = 0; i < 3; i++)
    {
        cout << setw(2) << char('A' + i) <<
':';
        for (int j = 0; j < 10; j++)
        {
            if (j < pointer[i])
                cout << setw(2) <<
capsule[i][j];
            else
                cout << "  ";
        }
    }
    cout << endl;//横向打印数组内容
}
```

纵向打印部分：

```
void printVer(int n, char src, char dst, int
order)//对纵向的汉诺塔数组进行输出
{
    cct_setcolor(COLOR_BLACK,
COLOR_WHITE);
    for (int i = 0; i < 3; i++)
    {
        int y = 0;
        if (order == 4)
            y = 12;
        else if (order == 8 || order == 9)
            y = 29;//由于不同菜单选项位置
不同，通过参数的区别进行区分
        for (int j = 0; j < 10; j++)
        {
            cct_gotoxy(10 + 10 * i, --y);
            if (j < pointer[i])
                cout << setw(2) <<
capsule[i][j];
            else if (j == pointer[i])
                cout << "  ";//纵向打印
```

```
        }
    }
}
```

打印柱子部分：

```
void printPillar(int layer, char src)
{
    const int WIDTH = 23;
    const int HEIGHT = 12;
    for (int i = 0; i < WIDTH; i++)
    {
        Sleep(100);
        cct_showch(i + 1, 15, ' ', COLOR_HYELLOW, COLOR_HYELLOW, 1);
        cct_showch(i + 33, 15, ' ', COLOR_HYELLOW, COLOR_HYELLOW, 1);
        cct_showch(i + 65, 15, ' ', COLOR_HYELLOW, COLOR_HYELLOW, 1);//打印柱子的底座
    }

    for (int i = 0; i < HEIGHT; i++)
    {
        Sleep(100);
        cct_showch(12, 14 - i, ' ', COLOR_HYELLOW, COLOR_HYELLOW, 1);
        cct_showch(44, 14 - i, ' ', COLOR_HYELLOW, COLOR_HYELLOW, 1);
        cct_showch(76, 14 - i, ' ', COLOR_HYELLOW, COLOR_HYELLOW, 1);//打印柱子竖着的部分
    }
}
```

打印盘子部分：

```
void printPlate(int x, int y, int half_length, int color)
{
   cct_showch(x - half_length, y, ' ', color, color, half_length * 2 + 1);//盘号即半边的长度，用于打印盘子
}
```

菜单 9 的游戏模式：

```
void gameVer(int layer, char src, char dst, int order)
{
    while (1)
    {
        cct_gotoxy(0, 34);
        cct_setcolor(COLOR_BLACK, COLOR_WHITE);

    cct_setcursor(CURSOR_VISIBLE_NORMAL);
       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）：  \b\b";
        char temp_src, temp_dst, temp;
        temp_src = _getche();
        temp_dst = _getche();

        if ((temp_src == 'q' || temp_src == 'Q') && temp_dst == '\r')
        {
            cout << endl << "游戏结束！" << endl;
            break;
        }//当接收到q指令，就结束游戏
        temp = _getch();
        if (temp != '\r')
        {
            cct_setcolor(COLOR_BLACK, COLOR_WHITE);
            cout << "\b\b  \b\b";
            continue;
        }//若输入超过两个字符，清除缓存区
        else
            cout << endl;
        if (temp_src >= 'a')
            temp_src -= 32;
        if (temp_dst >= 'a')
            temp_dst -= 32;
        if(temp_src>'C'||temp_src<'A'||temp_dst>'C' || temp_dst < 'A')
        {
            cct_setcolor(COLOR_BLACK, COLOR_WHITE);
            cout << "\b\b  \b\b";
            continue;
        }//若输入字符不满足abc条件，清除缓存区
        if (pointer[temp_src - 'A'] == 0)
        {
            cout <<"源柱为空,请重新输入";
            Sleep(1000);
            int x = 19, y = 35;
            while (x >= 0)
            {
                cct_gotoxy(x, y);
                cout << ' ';
                x--;
            }
            continue;
        }
        else if (capsule[temp_src - 'A'][pointer[temp_src - 'A'] - 1] > capsule[temp_dst - 'A'][pointer[temp_dst - 'A'] - 1] && pointer[temp_dst - 'A'] != 0)
        {
            cout << "大盘压小盘,非法移
```

```
动!";
                Sleep(1000);
                int x = 21, y = 35;
                while (x >= 0)
                {
                    cct_gotoxy(x, y);
                    cout << ' ';
                    x--;
                }
                continue;
            }//大盘压小盘的非法操作
            else
            {
                step++;
                movePlate(12 + 32 * (temp_src -
'A'), 14 - pointer[temp_src - 'A'], temp_src,
temp_dst, order);
                move(temp_src, temp_dst);
                printVer(layer, temp_src,
temp_dst, order);
                printSteps(layer, temp_src,
temp_dst, order);
            }
            if (pointer[src - 'A'] == 0 &&
pointer[dst - 'A'] == layer)
            {
                cct_gotoxy(0, 35);
                cout << "游戏结束! " << endl;
                break;
            }//若完成了汉诺塔游戏，则游戏结束
    }
}
```

装 订 线