同济大学 实验报告

汉诺塔的综合演示

班级：信息类11班

学号：2151294

姓名：马威

完成日期：2021.12.9

1. 题目：集成所有之前做过的有关汉诺塔的小题，并加入图形化演示，用菜单项进行选择

1.1.基本解

显示每一步的移动情况，包括盘子的编号以及每一步起始柱、目标柱的编号

1.2.基本解（步数记录）

显示每一步的移动情况，以及每一步的步数

1.3.内部数组显示（横向）

显示每一步的移动情况，并以一横排的形式输出每根柱子上盘子编号、数量的实时情况

1.4.内部数组显示（纵向+横向）

在1.3的基础上，加上内部数组纵向输出，即以数字堆叠的形式模仿三根柱子上的盘子。此时每一步移动需擦除原有的内容后再输出这一步的内容。

1.5.图形解-预备-画三个圆柱

使用伪图形界面工具，画出汉诺塔的三根柱子。为了方便观察实现过程，需要加延时。

1.6.图形解-预备-在起始柱上画n个盘子

要求输入起始圆柱的编号，盘子的数量，在指定的起始圆柱上从小到大画n个盘子，其颜色各不相同。为了方便观察实现过程，需要加延时。

1.7.图形解-预备-第一次移动

1.在1.6的基础上，显示第一步移动的动画。

2.第一步不一定是源柱->目标柱，也可能是源柱->中间柱，所以一定要调用递归函数，指导第一步的行动。

3.不允许直接在两个圆柱间移动，必须先上移、再平移、再下移。

1.8.图形解-自动移动版本

在1.7的基础上，完整地显示每一步移动的动画，同时也进行内部数组的横向与纵向输出。每次圆盘的移动方式也必须是上移、平移、下移。

1.9. 图形解-游戏版

1. 每次键盘输入两个字母（A-C）之间，大小写均可，表示本次移动的源柱和目标柱。

2. 移动时要检查合理性，若不符合移动规则（大盘压小盘、源柱为空等）要提示出错并重输，每次合理的移动都必须记录步数。

3. 每次圆盘的移动方式也必须是上移、平移、下移。

4. 待所有盘子按序移动到结束柱则提示“游戏结束”

1.10. 退出

按下数字0能直接退出程序。

2. 整体设计思路

1、整个程序以一个永真循环为背景，通过菜单函数来控制菜单选择。

2、若菜单项为零，直接跳出循环，程序结束。

3、若菜单项不为零，调用一个总的输入函数（里面包含了多种正确性处理的函数），输入信息后再调用一个总控函数（里面根据菜单项选择调用不同函数，作用相当于一个控制台）实现主要功能。

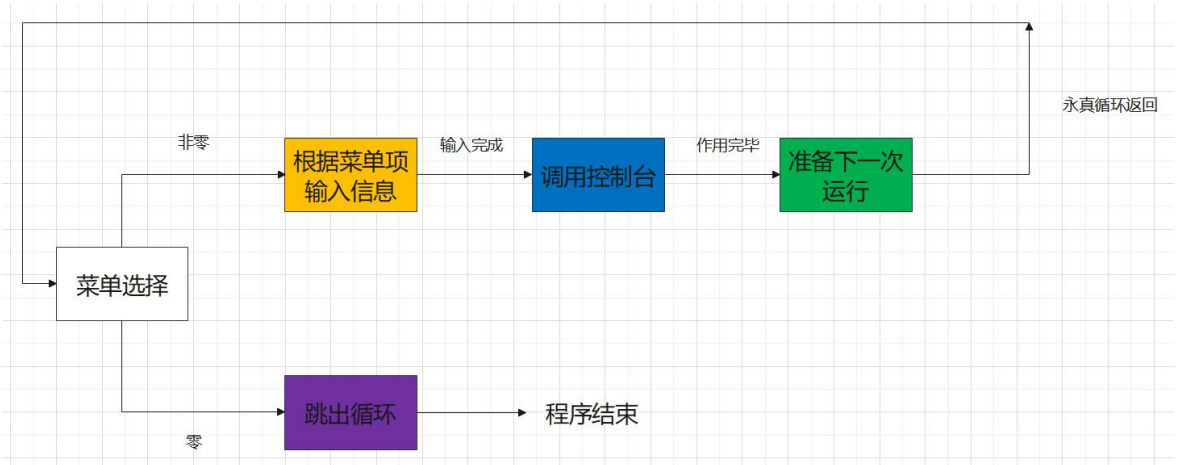
4、递归函数只有一个，所以第3点调用的是调用递归函数前，准备工作中需要的函数。而在递归过程中需要同时实现很多功能，所以在递归函数中会调用一个伴随函数，其通过菜单函数的返回值选择调用不同函数。

5、根据不同的主要功能，有针对性地写出完成各种功能的函数。

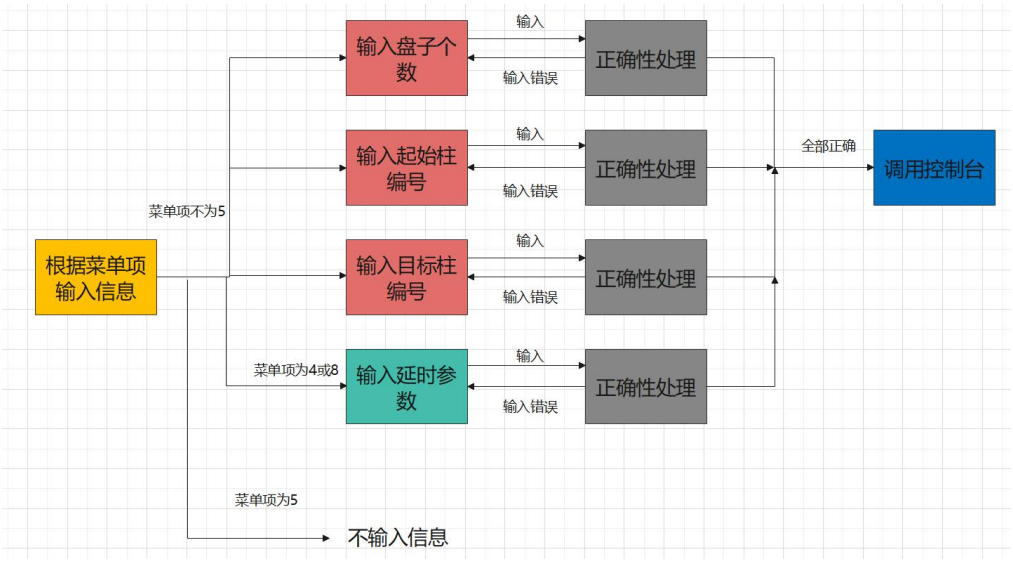
3. 主要功能的实现

3.1. 主函数及其有关函数

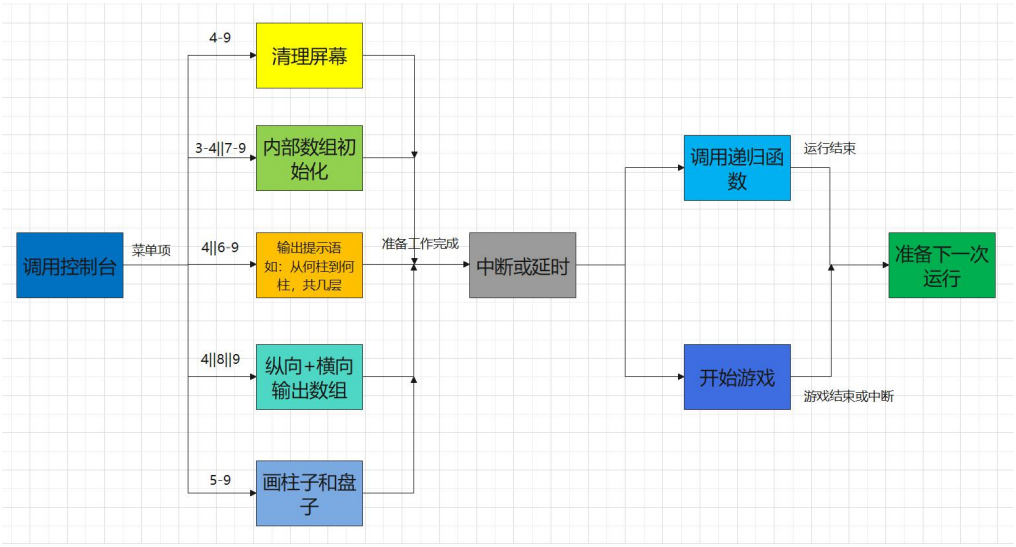
1、主函数的结构



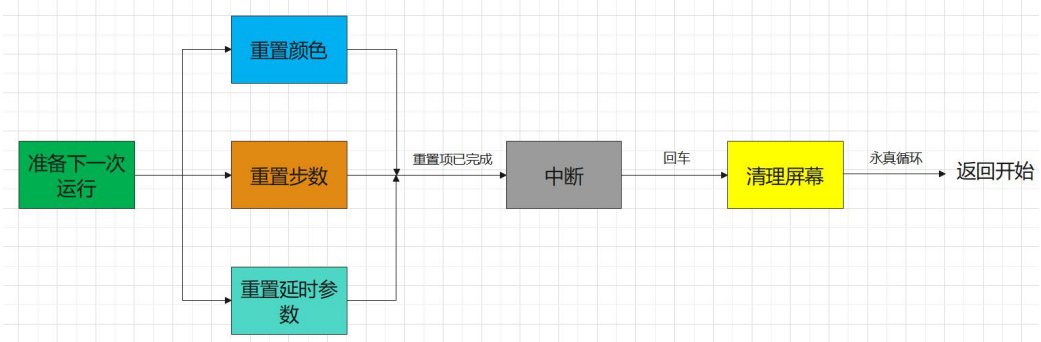
2、输入函数的结构



3、控制台函数的结构

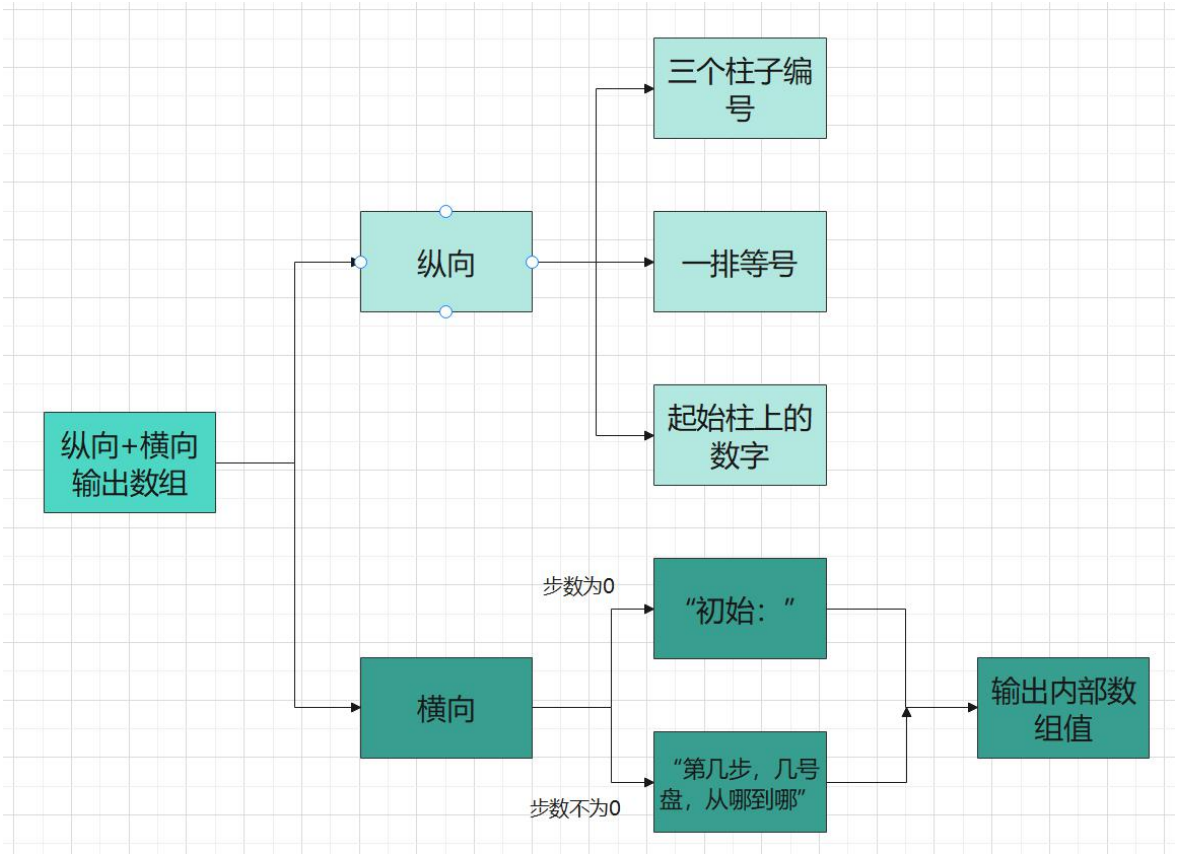


4、重设函数的结构

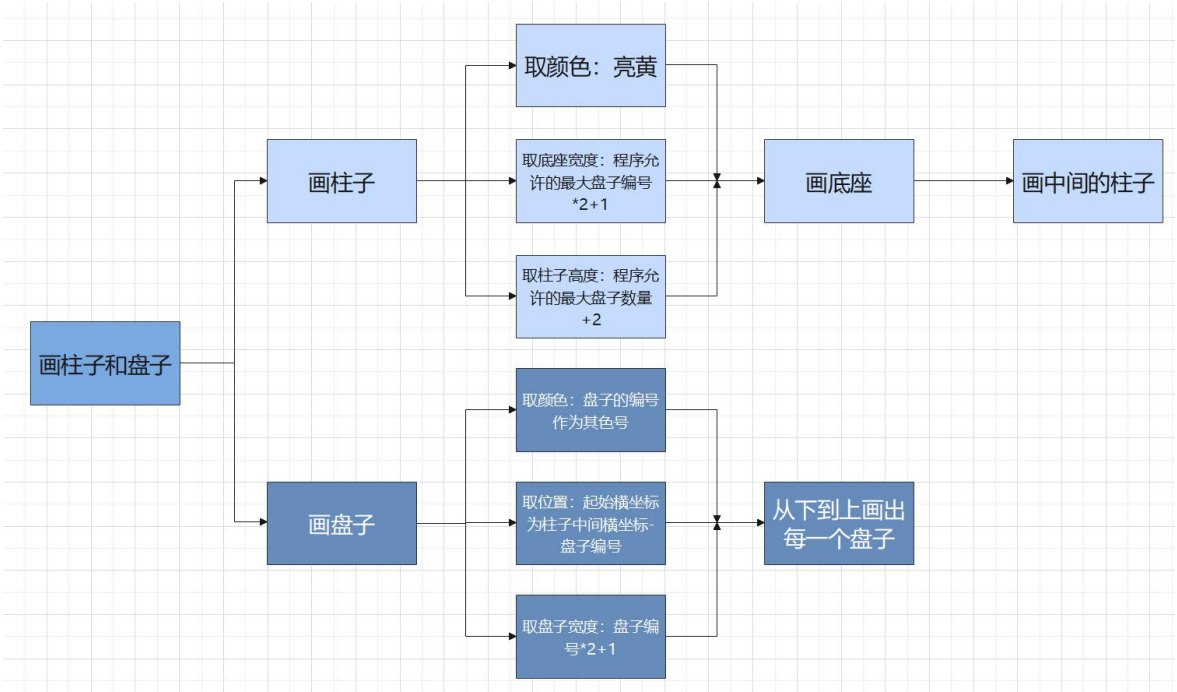


3.2.控制台有关函数

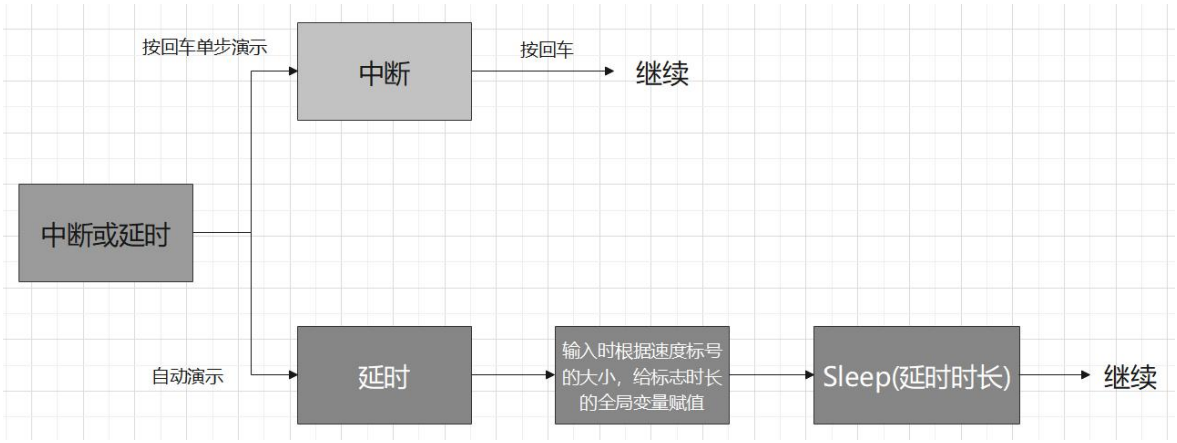
1、纵向、横向输出数组



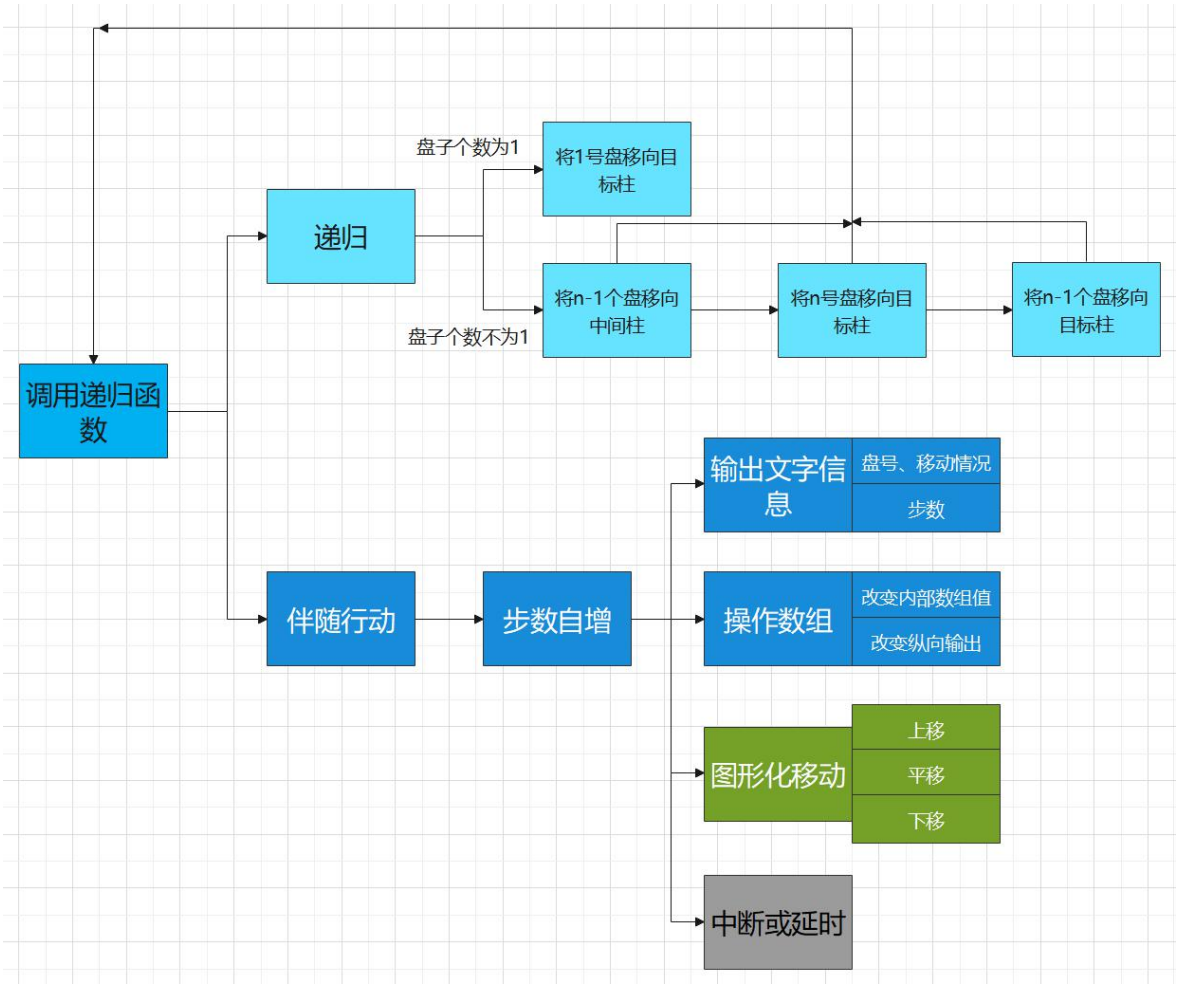
2、画柱子、画盘子



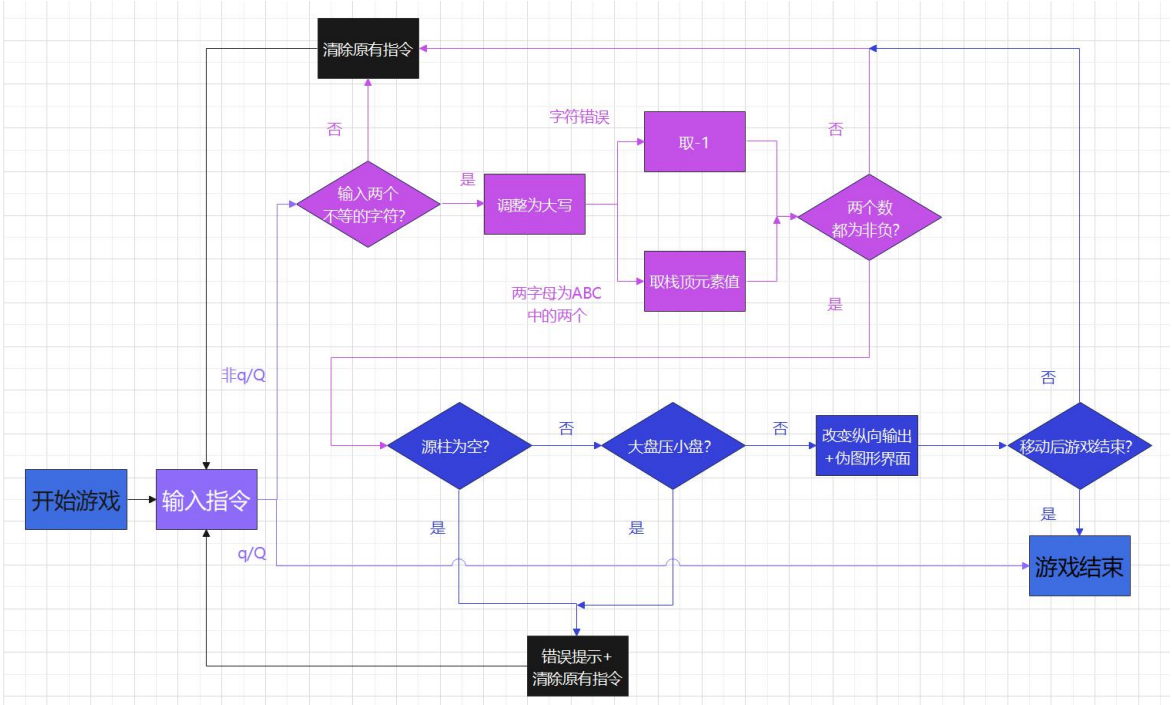
3、间断点


按回车单步演示
中断
按回车
继续
中断或延时
自动演示
延时
输入时根据速度标号的大小，给标志时长的全局变量赋值
Sleep(延时时长)
继续

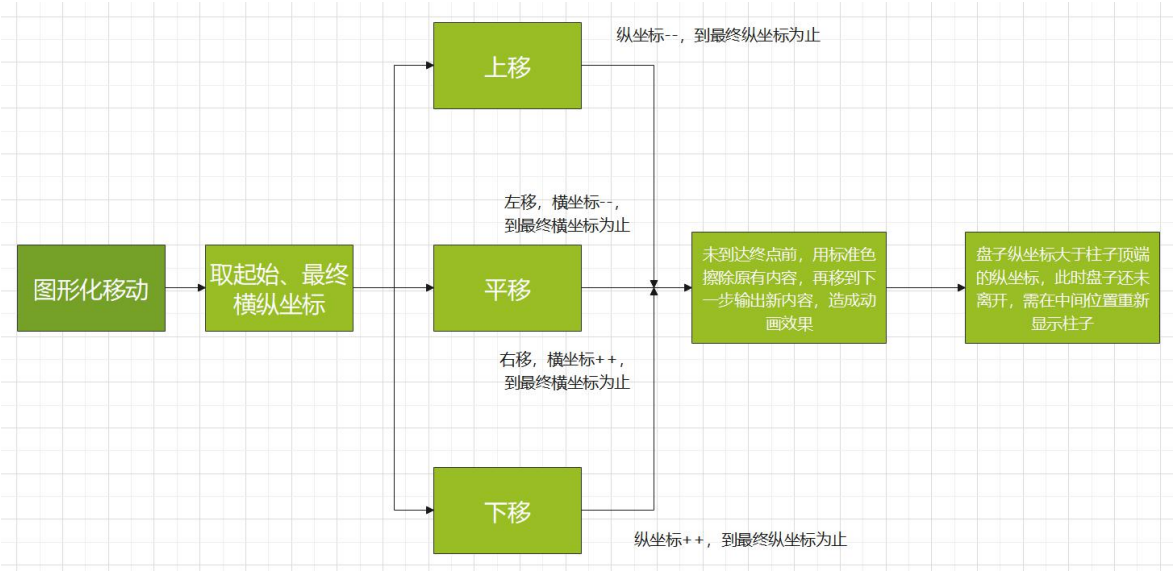
4、递归函数


盘子个数为1
将1号盘移向目标柱
递归
将n-1个盘移向中间柱
将n号盘移向目标柱
将n-1个盘移向目标柱
盘子个数不为1
调用递归函数
输出文字信息
盘号、移动情况
步数
伴随行动
步数自增
操作数组
改变内部数组值
改变纵向输出
图形化移动
上移
平移
下移
中断或延时

5、游戏



6、图形化移动



4. 调试过程碰到的问题

问题：写菜单项8的时候，横向、纵向数组显示正确，但伪图形界面里盘子移动总是不正确。

原因：查看递归函数中调用的伴随函数，发现逻辑上的错误：横向、纵向数组改变后，内部数组值也改变了，但盘子移动的函数需要利用移动前的内部数组值，所以产生了错误。

解决：只要在调用显示盘子移动的函数前，再调用一次改变内部数组的函数，撤销这一步移动带来的数组值改变，让内部数组值回到移动前的状态，伪图形界面里盘子移动就正确了。

5. 心得体会

5.1.完成本次作业得到的一些心得体会，经验教训

1.解决这类比较复杂且规模不小的问题时，先从整体入手，将整体设计思路写下来，着手将大框架的程序尝试写出来，比如规划主函数的大致内容、菜单函数的形式等。

2.在大框架下，按一定顺序（本题中按照菜单项的顺序）将小单元写出，先思考需要哪些功能，函数如何划分；再看看做过的小题中是否有可以使用的代码。总的来说就是逐个击破各个小问题，最终解决大问题。

3.在写具体函数的实现时，不要想到什么就一头扎进去写，要瞻前顾后。

4.瞻前：看看已经写出的代码，要写下来的是否与已有的重复？若重复了，如何在不影响功能的前提下用更精简而有条理地代码完成？然后再去修改、调试。

5.顾后：把自己想象成这个程序的客户，仔细思考自己对于这个产品是否会有不满的地方，会提出什么更高的要求，再把思路转换回来，想想面对这样的问题，是否只要在写程序时将方法稍微改变一下，后期维护升级时就会更轻松。

6.做第5点不宜太晚，应尽早规划好哪些地方在这个题目中可能是要“写死”的，但实际上升级时需要它是“活着”的，否则回想起来时，自己已经“写死”了太多的量，还想再改进时，工作会变得非常困难。

7.针对5、6点，一定要能用、会用、善用全局const和#define，不要在写具体实现时嫌写一个变量名比一个“死板的”数字来的麻烦，用上这些，日后只要通过改变参数的值就可以达到“牵一发而动全身”的改进效果。

5.2.在做一些较为复杂的程序时，是分为若干小题还是直接一道大题的形式更好？

分为若干小题更好。

从着手难度看：直接一道大题很难看一眼就直接上手写代码，至少需要构思总体的设计思路，再从比较简单的一些问题的解决上出发，而这已经需要不少的时间了。相比之下，分为若干个小题，在做每一个小提时，思考的问题不会太复杂，很快就能找到问题的突破口，仅仅对于完成题目而言，效率的确会快一点。

从代码利用看：直接做一道大题，全部的内容都是新的，需要“白手起家”，工作量不小。而分为多个小题后，较为复杂的小题总会使用到比较简单的小题中实现的功能，这时候就给了自己一次重新检视、利用、改进自己原先代码的机会，相当于再检查一次自己写的东西，总可以看出可以改进提升的地方；同时工作量也会小一些。

5.3.汉诺塔完成了若干小题，总结你在完成过程中是否考虑了前后小题的关联关系，是否能尽可能做到后面小题有效利用前面小题已完成的代码，如何才能更好地重用代码？

在完成过程中考虑到了前后小题之间的关联，也尽可能做到了利用已完成的代码。

重用代码不是简单的复制粘贴，能用就行。如5.2中所言，重用已有代码是给自己一个重新检视自己代码的机会，总的来说要重新检查三个方面：

1、能利用的地方有多少？不必利用、不能利用的部分是哪里？

2、是否精简、逻辑清晰明了？若不是，冗余代码如何寻找，又如何在不影响功能的前提下改进代码呢？

3、是否存在隐藏的逻辑错误？是否存在违反之前题目规定的做法？这些做法现在能用吗？这些做法会对自己和程序有什么影响？

总之，更好地重用代码，不能无所作为，而要努力思考、改错、改进、提升，然后才能更好地利用，否则有些问题可能越积越多，终会造成严重的后果。

5.4.以本次作业为例，说明如何才能更好地利用函数来编写复杂的程序

在构思整体设计思路的时候，会划分程序的各个主要功能。使用函数实现这些功能，在程序里一个复杂功能的实现就可以浓缩成一句函数的调用，使得逻辑非常的清晰明了，也能不难看出整个程序的结构。所以合理划分好各项需要实现的功能，再逐一使用函数实现，对于编写复杂程序会大有帮助。

例1：菜单项7、8中显示盘子的移动动画。

这个功能不是一个单一功能，它包括上移、平移、下移等多种功能，将其本身作为一个总的函数，其中调用上移函数、平移函数、下移函数等实现单一功能的函数，这样就很有条理地完成这个功能了。

例2：菜单项9的游戏

这一项包含的功能更多，包括输入、判断、赋值、移动等。可以先以菜单项9整体作为一个总的函数，找出高频使用的功能，每一个都用函数来实现，按需取用，如：

①消除上次输入的指令

②根据输入的指令给起始柱标号等变量赋值

③用②结果判断是否能移动

④输出移动错误的信息并擦除

⑤根据②的结果，调用多个函数完成移动

……

这样菜单项9的程序结构就比较清晰了，就算其功能比较复杂，完成起来还是比较有条理的。

6. 附件：源程序

1.hanoi_multiple_solutions.cpp

```
/*2151294 信11 马威*/
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<iomanip>
#include<conio.h>
#include<Windows.h>
#include"cmd_console_tools.h"
#include"hanoi.h"
using namespace std;

static int __stepnumber = 0;

static int __delay_time;

int TOWER[3][PLATE_NUM_LIMIT];

int MARK[3];
```

```
void hanoi(int number_of_plates, char source_tower, char
temporary_tower, char destination_tower, const int menu)
{
	if (number_of_plates == 1) {/*若汉诺塔层数为 1，将该盘从起
始柱直接移向目标柱*/
		__stepnumber++;
		hanoi_act(number_of_plates, source_tower,
destination_tower, menu);
	}

	else {  /*若汉诺塔层数不为 1*/
		hanoi(number_of_plates - 1, source_tower,
destination_tower, temporary_tower, menu);  /*将 n-1 个盘子移向中
间柱*/
		__stepnumber++;  /*步数自增*/
		hanoi_act(number_of_plates, source_tower,
destination_tower, menu);  /*将第 n 个盘子直接移向目标柱*/
		hanoi(number_of_plates - 1, temporary_tower,
source_tower, destination_tower, menu);  /*将其余 n-1 个盘子移向目
标柱*/
	}
}

void hanoi_act(int number_of_plates, char source_tower, char
destination_tower, const int menu)
{
	hanoi_act_info(number_of_plates, source_tower,
destination_tower, menu);    /*1、输出每一步的文字信息*/
	hanoi_act_array(number_of_plates, source_tower,
destination_tower, menu);  /*2、对数组进行操作*/

	/*3、菜单项 7 或 8：图形输出+再次改变数组值*/
	/*注：菜单项 7 只输出第一步的移动*/
	if ((menu == 7 && __stepnumber == 1) || menu == 8) {
		console_show_move(number_of_plates, source_tower,
destination_tower, menu);
		array_move_plates(source_tower, destination_tower);
	}

	/*4、菜单项 4 或 8：每一步后的暂停或延时*/
	if (menu == 4 || menu == 8) {
		pause(__delay_time);  /*每一步后的暂停或延时*/
	}
}

void hanoi_act_info(int number_of_plates, char source_tower, char
destination_tower, const int menu)
{
	/*菜单项 1：打印每一步的解*/
	if (menu == 1)
		cout << number_of_plates << "# " << source_tower <<
"---->" << destination_tower << endl;

	/*菜单项 2 或 3：打印每一步的步数+解*/
	else if (menu == 2 || menu == 3) {
		cout << "第" << setw(4) << __stepnumber << " 步(" <<
setw(2) << number_of_plates << "#: "
			<< source_tower << "-->" << destination_tower
<< ")";

		/*若菜单项为 2，直接在此换行*/
		if (menu == 2)
			cout << endl;
	}
}

void hanoi_act_array(int number_of_plates, char source_tower, char
destination_tower, const int menu)
{
	/*菜单项 3、4 或 8：改变数组值+横式打印内部数组*/
	if (menu == 3 || menu == 4 || menu == 8) {
		array_move_plates(source_tower, destination_tower);
		array_print_tower(number_of_plates, source_tower,
destination_tower, menu);
	}

	/*菜单项 4 或 8：改变数值位置*/
	if (menu == 4 || menu == 8)
		array_show_move(source_tower, destination_tower,
menu);

	/*菜单项 8：将这一步数组值复位，以便下一步图形化输出能使用
移动前的内部数组值*/
	if (menu == 8)
		array_move_plates(destination_tower, source_tower);
}

void array_tower_initialize(const int number_of_plates, const char
source_tower)
{
	int i, j;

	/*给所有元素赋初值*/
	for (i = 0; i < 3; i++) {
		for (j = 0; j < PLATE_NUM_LIMIT; j++)
			TOWER[i][j] = 0;
	}

	/*根据起始柱的代号，给起始柱的对应元素赋上 1-n 的值，代表初
始时的各个盘子*/
	for (i = 0; i < number_of_plates; i++)
		TOWER[source_tower - 'A'][i] = number_of_plates - i;

	/*起始柱栈顶指针的初始化*/
	for (i = 0; i < 3; i++)
		MARK[i] = 0;

	MARK[source_tower - 'A'] = number_of_plates;
}

void array_print_base(const int number_of_plates, const char
source_tower, const int menu)
{
	int i, BASE_Y;  /*定义所需变量，BASE_Y 是坐标参数*/

	if (menu == 4)
		BASE_Y = PLATE_NUM_LIMIT + 2;

	else
		BASE_Y = CONSOLE_BASE_Y + PLATE_NUM_LIMIT + 2;

	/*循环输出代表三个柱子的字母*/
	for (i = 0; i < 3; i++) {
		cct_gotoxy(ARRAY_BASE_X + (ARRAY_DISTANCE + 1) * i,
BASE_Y + 1);
		cout << char('A' + i);
	}

	/*循环输出一排等号，使输出结果更具形象化*/
	cct_gotoxy(ARRAY_BASE_X - 2, BASE_Y);
	for (i = 0; i < 6 + 2 * ARRAY_DISTANCE + 1; i++)
		cout << "=";

	/*在指定的位置，形象化输出每根柱子上的情况，光标移动前后都
需移动到指定位置*/
	cct_gotoxy((ARRAY_DISTANCE + 1) * (source_tower - 'A') +
ARRAY_BASE_X - 1, BASE_Y - 1);

	for (i = 0; i < PLATE_NUM_LIMIT; i++) {
		if (TOWER[source_tower - 'A'][i])
			cout << setw(2) << TOWER[source_tower -
'A'][i];

		cct_gotoxy((ARRAY_DISTANCE + 1) * (source_tower - 'A')
+ ARRAY_BASE_X - 1, BASE_Y - i - 2);
	}

	cct_gotoxy((ARRAY_DISTANCE + 1) * (source_tower - 'A') +
ARRAY_BASE_X + 1, BASE_Y - number_of_plates);
}

void array_show_move(const char source_tower, const char
destination_tower, const int menu)
{
	int BASE_Y;  /*定义所需变量，BASE_Y 是坐标参数*/

	/*按菜单函数返回值决定参数值*/
	if (menu == 4)
```

```
		BASE_Y = PLATE_NUM_LIMIT + 2;

	else
		BASE_Y = CONSOLE_BASE_Y + PLATE_NUM_LIMIT + 2;

	/*覆盖原有内容*/
	cct_gotoxy((ARRAY_DISTANCE + 1) * (source_tower - 'A') +
ARRAY_BASE_X + 1, BASE_Y - 1 - MARK[source_tower - 'A']);
	cout << "\010\010\040\040";

	/*输出现有内容*/
	cct_gotoxy((ARRAY_DISTANCE + 1) * (destination_tower - 'A')
+ ARRAY_BASE_X - 1, BASE_Y - MARK[destination_tower - 'A']);
	cout << setw(2) << TOWER[destination_tower -
'A'][MARK[destination_tower - 'A'] - 1];
}

void array_print_tower(const int number_of_plates, const char
source_tower, const char destination_tower, const int menu)
{
	int i, j, y = 0;

	if (menu == 4)
		y = PLATE_NUM_LIMIT + 7;  /*菜单项 4 的位置*/

	else if (menu == 8 || menu == 9)
		y = CONSOLE_BASE_Y + PLATE_NUM_LIMIT + 7;  /*菜单项
8、菜单项 9 的位置*/

	if (menu != 3) {  /*菜单项 3 不需要擦除再输出*/
		cct_gotoxy(SCREEN_WIDTH - 1, y);

		/*消除原有内容*/
		for (i = 0; i < SCREEN_WIDTH; i++)
			cout << '\010';
		for (i = 0; i < SCREEN_WIDTH; i++)
			cout << '\040';

		/*根据步数决定输出提示内容*/
		cct_gotoxy(0, y);

		if (__stepnumber)  /*已经开始移动，步数不为零，输出
移动情况*/
			cout << "第" << setw(4) << __stepnumber << "
步(" << number_of_plates << "#: " << source_tower << "-->" <<
destination_tower << ") ";

		else  /*还未移动，步数为零，输出初始情况*/
			cout << "初始: ";
	}

	/*显示内部数组值*/
	for (i = 0; i < 3; i++) {
		cout << ' ' << char('A' + i) << ":";

		for (j = 0; j < PLATE_NUM_LIMIT; j++) {
			if (TOWER[i][j] != 0)
				cout << setw(2) << TOWER[i][j];

			else
				cout << "  ";
		}
	}

	cout << endl;
}

void array_move_plates(const char source_tower, const char
destination_tower)
{
	TOWER[destination_tower - 'A'][MARK[destination_tower -
'A']++] = TOWER[source_tower - 'A'][--MARK[source_tower - 'A']];
	TOWER[source_tower - 'A'][MARK[source_tower - 'A']] = 0;
}

void console_print_base()
{
	const int bg_color = COLOR_HYELLOW;    //背景为亮黄色
	const int fg_color = COLOR_HBLUE;      //前景为亮蓝色
	int i, j;  /*计数变量*/

	/*光标位置初始化*/
	cct_gotoxy(CONSOLE_BASE_X, CONSOLE_BASE_Y);

	/*输出底盘*/
	for (i = 0; i < 3; i++) {
		Sleep(50);
		cct_showch(CONSOLE_BASE_X + i * (TOWER_WIDTH +
TOWER_DISTANCE), CONSOLE_BASE_Y, ' ', bg_color, fg_color,
TOWER_WIDTH);
	}

	/*输出柱子*/
	for (i = 0; i < TOWER_HEIGHT; i++) {
		for (j = 0; j < 3; j++) {
			Sleep(50);
			cct_showch(CONSOLE_BASE_X + (TOWER_WIDTH - 1)
/ 2 + j * (TOWER_WIDTH + TOWER_DISTANCE), CONSOLE_BASE_Y - i - 1,
' ', bg_color, fg_color, 1);
		}
	}

	/*输出完成后的颜色重设*/
	cct_setcolor();
}

void console_print_plates(const int number_of_plates, const char
source_tower)
{
	const int mid = CONSOLE_BASE_X + (TOWER_WIDTH - 1) / 2 +
(TOWER_WIDTH + TOWER_DISTANCE) * (source_tower - 'A'); /*每根柱子
中间的位置*/
	int X, Y;                /*横坐标、纵坐标*/
	int bg_color, fg_color;  /*背景色、字体色*/
	int LENGTH;              /*盘子长度*/
	int i = 0;               /*计数变量*/

	/*输出盘子*/
	for (i = 0; i < number_of_plates; i++) {

		/*根据参数，确定各个变量的值*/
		X = mid - number_of_plates + i;
		Y = CONSOLE_BASE_Y - i - 1;
		bg_color = number_of_plates - i;
		fg_color = 7;
		LENGTH = 2 * number_of_plates + 1 - 2 * i;

		/*延时、输出*/
		Sleep(30);
		cct_showch(X, Y, ' ', bg_color, fg_color, LENGTH);
	}

	/*输出完成后的颜色重设*/
	cct_setcolor();
}

void console_show_move(const int number_of_plates, const char
source_tower, const char destination_tower, const int menu)
{
	const int bg_color = number_of_plates;        /*背景色（根
据盘子编号来取，保证移动过程中盘子不变色）*/
	const int fg_color = COLOR_WHITE;             /*字体色*/
	const int LENGTH = 2 * number_of_plates + 1;  /*盘子长度*/

	/*起始柱中间位置*/
	const int src_mid = CONSOLE_BASE_X + (TOWER_WIDTH - 1) / 2
+ (TOWER_WIDTH + TOWER_DISTANCE) * (source_tower - 'A');
	/*目标柱中间位置*/
	const int dst_mid = CONSOLE_BASE_X + (TOWER_WIDTH - 1) / 2
+ (TOWER_WIDTH + TOWER_DISTANCE) * (destination_tower - 'A');

	const int START_X = src_mid - TOWER[source_tower -
'A'][MARK[source_tower - 'A'] - 1];  /*起始横坐标*/
	const int START_Y = CONSOLE_BASE_Y - MARK[source_tower - 'A'];
/*起始纵坐标*/
	const int END_X = dst_mid - number_of_plates;
/*最终横坐标*/
	const int END_Y = CONSOLE_BASE_Y - MARK[destination_tower -
```

```
'A'] - 1;  /*最终纵坐标*/
	int time = 0;  /*延时参数*/

	/*取延时参数，若为菜单项 7、8 中的单步演示，延时 100；若为菜单项 8 中的连续演示，延时 delay_time；若为菜单项 9，延时 1，造成瞬移效果*/
	if (menu == 7 || (menu == 8 && __delay_time == 0))
		time = 100;
	else if (menu == 8 && __delay_time != 0)
		time = __delay_time;
	else if (menu == 9)
		time = 1;

	/*向上移动*/
	console_show_move_up(START_X, START_Y, bg_color, fg_color, src_mid, LENGTH, time);

	/*左右移动*/
	if (destination_tower < source_tower)
		console_show_move_left(START_X, END_X, bg_color, fg_color, LENGTH, time);
	else
		console_show_move_right(START_X, END_X, bg_color, fg_color, LENGTH, time);

	/*向下移动*/
	console_show_move_down(END_X, END_Y, bg_color, fg_color, dst_mid, LENGTH, time);

	/*移动完成后的颜色重设*/
	cct_setcolor();
}

void console_show_move_up(int START_X, int START_Y, int bg_color, int fg_color, int src_mid, int LENGTH, int time)
{
	int Y;  /*显示动画中变动的当前纵坐标*/

	for (Y = START_Y; Y >= 1; Y--) {
		cct_showch(START_X, Y, ' ', bg_color, fg_color, LENGTH);

		/*若连续演示速度不为 5，则启动延时*/
		if (time != 1)
			Sleep(time);

		/*到达最高点之前，用标准色擦除原有内容，造成动画效果*/
		if (Y > 1)
			cct_showch(START_X, Y, ' ', COLOR_BLACK, COLOR_WHITE, LENGTH);

		/*若未离开柱子，则需在柱子中间位置重新显示柱子，否则会被上一句擦除掉*/
		if (Y > TOWER_TOP)
			cct_showch(src_mid, Y, ' ', COLOR_HYELLOW, COLOR_WHITE, 1);
	}
}

void console_show_move_down(int END_X, int END_Y, int bg_color, int fg_color, int dst_mid, int LENGTH, int time)
{
	int Y;  /*显示动画中变动的当前纵坐标*/

	for (Y = 1; Y <= END_Y; Y++) {
		cct_showch(END_X, Y, ' ', bg_color, fg_color, LENGTH);

		/*若连续演示速度不为 5，则启动延时*/
		if (time != 1)
			Sleep(time);

		/*到达最低点之前，用标准色擦除原有内容，造成动画效果*/
		if (Y < END_Y)
			cct_showch(END_X, Y, ' ', COLOR_BLACK, COLOR_WHITE, LENGTH);

		/*若进入柱子，则需在柱子中间位置重新显示柱子，否则会被上一句擦除掉*/
		if (Y > TOWER_TOP && Y < END_Y)
			cct_showch(dst_mid, Y, ' ', COLOR_HYELLOW, COLOR_WHITE, 1);
	}
}

void console_show_move_left(int START_X, int END_X, int bg_color, int fg_color, int LENGTH, int time)
{
	int X;  /*显示动画中变动的当前横坐标*/

	for (X = START_X; X >= END_X; X--) {
		cct_showch(X, 1, ' ', bg_color, fg_color, LENGTH);

		/*若连续演示速度不为 5，则启动延时*/
		if (time != 1)
			Sleep(time);

		/*到达最左侧之前，用标准色擦除原有内容，造成动画效果*/
		if (X > END_X)
			cct_showch(X, 1, ' ', COLOR_BLACK, COLOR_WHITE, LENGTH);
	}
}

void console_show_move_right(int START_X, int END_X, int bg_color, int fg_color, int LENGTH, int time)
{
	int X;  /*显示动画中变动的当前横坐标*/

	for (X = START_X; X <= END_X; X++) {
		cct_showch(X, 1, ' ', bg_color, fg_color, LENGTH);

		/*若连续演示速度不为 5，则启动延时*/
		if (time != 1)
			Sleep(time);

		/*到达最右侧之前，用标准色擦除原有内容，造成动画效果*/
		if (X < END_X)
			cct_showch(X, 1, ' ', COLOR_BLACK, COLOR_WHITE, LENGTH);
	}
}

void game_command_center(const int total_number_of_plates, const char destination_tower, const int menu)
{
	cct_setcursor(CURSOR_VISIBLE_NORMAL);  /*显示光标*/
	cct_gotoxy(0, GAME_TIP_Y);     /*移动光标至指定位置*/
	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)：";  /*输出提示语。改变提示语长度时，请同步改变#define TIP_LENGTH*/

	while (1) {
		int i = 0;  /*计数变量，每一次循环结束归零*/
		char src = 0, dst = 0, c;  /*起始柱编号（每一次循环结束归零）、目标柱编号（每一次循环结束归零）、提取字符*/

		/*1、输入指令*/
		while (1) {
			c = getchar();  /*逐个提取字符*/

			if (c == '\n' || c == 'q' || c == 'Q')  /*若有换行符或退出指令，结束输入*/
				break;

			if (i == 0)  /*若输入未结束，第一个字符赋给 src*/
				src = c;

			if (i == 1)  /*若输入未结束，第二个字符赋给 dst*/
				dst = c;

			i++;  /*每输入一个字符，计数变量自增，统计输入的字符数*/
```

```
            }

            /*2、若 c 为退出指令，输出提示语并结束游戏*/
            if (c == 'q' || c == 'Q') {
                  cout << "游戏中止!!!!!";
                  break;
            }

            /*3、若刚好输入两个不等的字符，调整大小写，判断是否可以移动并移动盘子*/
            if (i == 2 && src != dst) {
                  if (src >= 'a' && src <= 'c')
                        src = src - 'a' + 'A';
                  if (dst >= 'a' && dst <= 'c')
                        dst = dst - 'a' + 'A';

                  game_get_info_and_move(src, dst, menu);
            }

            /*4、若 1 号盘已经到达最终位置，则判断游戏结束*/
            if (TOWER[destination_tower - 'A'][total_number_of_plates - 1] == 1) {
                  cct_gotoxy(0, GAME_TIP_Y + 1);
                  cout << "游戏结束!!!!!";
                  break;
            }

            /*5、输入指令错误，或者是每完成一步移动后，清除原有指令（位置靠后，不会在游戏结束或错误提示时先清空原有指令）*/
            else
                  game_clear_input();
      }
}

void game_clear_input()
{
      int j;

      cct_gotoxy(TIP_LENGTH, GAME_TIP_Y);

      for (j = 0; j < 60; j++)
            cout << '\040';

      cct_gotoxy(TIP_LENGTH, GAME_TIP_Y);
}

void game_get_info_and_move(const char source_tower, const char destination_tower, const int menu)
{
      /*根据柱子编号，取栈顶元素的值（空为零，错误为负）*/
      int number_of_src = game_get_plate_num(source_tower);
      int number_of_dst = game_get_plate_num(destination_tower);

      /*只有当两个栈顶元素值都不为负（都正确）的时候，才进行下一步判断*/
      if (number_of_src >= 0 && number_of_dst >= 0) {

            /*起始柱为空的情况*/
            if (TOWER[source_tower - 'A'][0] == 0)
                  game_err(EmptySourceTower);

            /*大盘压小盘的情况*/
            else if (number_of_src > number_of_dst && number_of_dst != 0)
                  game_err(BigOnSmall);

            /*准备就绪，可以移动的情况*/
            else
                  game_ok_move(number_of_src, source_tower, destination_tower, menu);
      }
}

int game_get_plate_num(const char tower)
{
      /*柱子编号范围正确*/
      if (tower >= 'A' && tower <= 'C') {

            /*使用只读字符变量接收参数 tower，消除智能提示*/
            const char Tower = tower;

            /*若栈底元素不为零，说明该柱不为空，返回栈顶元素的值*/
            if (TOWER[Tower - 'A'][0])
                  return TOWER[Tower - 'A'][MARK[Tower - 'A'] - 1];

            /*若栈底元素为零，说明该柱为空，返回零*/
            else
                  return 0;
      }

      /*若柱子编号不正确，返回负数*/
      else
            return -1;
}

void game_err(const int err_info)
{
      unsigned int i;  /*计数变量*/
      char c[128];     /*接收提示语的字符数组*/

      /*根据错误信息制定提示语*/
      if (err_info == 1)
            strcpy(c, "源柱为空!");
      else if (err_info == 2)
            strcpy(c, "大盘压小盘，非法移动!");

      /*输出提示语并暂停*/
      cout << c;
      Sleep(1000);

      /*消除提示语并使光标复位*/
      cct_gotoxy(0, GAME_TIP_Y + 1);

      for (i = 0; i < strlen(c); i++)
            cout << '\040';

      cct_gotoxy(TIP_LENGTH, GAME_TIP_Y);
}

void game_ok_move(const int number_of_plates, const char source_tower, const char destination_tower, const int menu)
{
      __stepnumber++;  /*步数自增*/
      array_move_plates(source_tower, destination_tower);  /*改变数组值，内部数组达到移动盘子的效果*/
      array_print_tower(number_of_plates, source_tower, destination_tower, menu);  /*横式打印内部数组*/
      array_show_move(source_tower, destination_tower, menu);  /*改变数值位置，纵向输出达到移动盘子的效果*/

      array_move_plates(destination_tower, source_tower);  /*将这一步数组值复位，以便下一步图形化输出能使用移动前的内部数组值*/
      console_show_move(number_of_plates, source_tower, destination_tower, menu);  /*改变盘子位置，图形输出达到移动盘子的效果*/
      array_move_plates(source_tower, destination_tower);  /*改变数组值，内部数组达到移动盘子的效果*/
}

void input_info(int* number_of_plates, char* source_tower, char* destination_tower, int* delay_time_set, const int menu)
{
      if (menu != 5) {  /*菜单项 5 不需要输入信息*/
            input_number_of_plates(number_of_plates);
            input_source_tower(source_tower);
            input_destination_tower(destination_tower, source_tower);
      }

      if (menu == 4 || menu == 8)  /*菜单项 4 和菜单项 8 需要输入延时参数*/
            input_delay(delay_time_set);
}

void input_number_of_plates(int* number_of_plates)
{
```

```
	int n;

	while (1) {
		cout << "请输入汉诺塔的层数(1-" << PLATE_NUM_LIMIT << "): " << endl;
		cin >> n;
		if (n >= 1 && n <= PLATE_NUM_LIMIT && cin.good() == 1)
{
			cin.clear();
			cin.ignore(INT_MAX, '\n');
			break;
		}

		/*只剩非法输入，或数值超类型范围的处理*/
		else if ((n < 1 || n > PLATE_NUM_LIMIT) && cin.good()
== 0) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}

		/*读取状态正常，超过用户范围时的处理*/
		else if ((n < 1 || n > PLATE_NUM_LIMIT) && cin.good()
== 1) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}
	}

	*number_of_plates = n;
}

void input_source_tower(char* source_tower)
{
	char src;
	while (1) {
		cout << "请输入起始柱(A-C): " << endl;
		cin >> src;
		if (((src >= 'a' && src <= 'c') || (src >= 'A' && src
<= 'C')) && cin.good() == 1) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
			break;
		}

		/*只剩非法输入，或数值超类型范围的处理*/
		else if (((src < 'a' || src > 'c') || (src < 'A' || src >
'C')) && cin.good() == 0) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}

		/*读取状态正常，超过用户范围时的处理*/
		else if (((src < 'a' || src > 'c') || (src < 'A' || src >
'C')) && cin.good() == 1) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}
	}

	if (src >= 'a' && src <= 'c') {
		*source_tower = src - 32;
	}

	else {
		*source_tower = src;
	}
}

void input_destination_tower(char* destination_tower, const char*
source_tower)
{
	char dst;
	while (1) {
		cout << "请输入目标柱(A-C): " << endl;
		cin >> dst;
		if (((dst >= 'a' && dst <= 'c') || (dst >= 'A' && dst
<= 'C')) && cin.good() == 1) {
			if (dst != *source_tower && dst != *source_tower
+ 32 && dst != *source_tower - 32) {
				cin.clear();
				cin.ignore(INT_MAX, '\n');
				break;
			}

			/*dst 与 src 相等时的处理*/
			else {

				/*为了使提示语中的 dst 输出为大写,对 dst
的值进行处理*/
				if (dst >= 'a' && dst <= 'c') {
					dst = dst - 32;
				}

				cout << "目标柱(" << dst << ")不能与起
始柱(" << *source_tower << ")相同" << endl;
				cin.clear();
				cin.ignore(INT_MAX, '\n');
			}
		}

		/*只剩非法输入，或数值超类型范围的处理*/
		else if (((dst < 'a' || dst > 'c') || (dst < 'A' || dst >
'C')) && cin.good() == 0) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}

		/*读取状态正常，超过用户范围时的处理*/
		else if (((dst < 'a' || dst > 'c') || (dst < 'A' || dst >
'C')) && cin.good() == 1) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}
	}

	if (dst >= 'a' && dst <= 'c') {
		*destination_tower = dst - 32;
	}

	else {
		*destination_tower = dst;
	}
}

void input_delay(int* delay_time_set)
{
	int i;
	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)" << endl;
		cin >> i;
		if (i >= 0 && i <= 5 && cin.good() == 1) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
			break;
		}

		/*只剩非法输入，或数值超类型范围的处理*/
		else if (cin.good() == 0) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}

		/*读取状态正常，超过用户范围时的处理*/
		else if ((i < 0 || i > 5) && cin.good() == 1) {
			cin.clear();
			cin.ignore(INT_MAX, '\n');
		}
	}

	/*根据 i 的输入值，给全局变量 delay_time 赋值*/
	if (i == 0)
		__delay_time = 0;

	else if (i >= 1 && i <= 3)
		__delay_time = 130 - 30 * i;

	else if (i == 4)
```

```
            __delay_time = 15;

      else if (i == 5)
            __delay_time = 1;

      *delay_time_set = i;
}

void pause(const int delay_time_set)
{
      int c;

      if (delay_time_set > 0 && delay_time_set != 5)
            Sleep(__delay_time);

      else {
            while (1) {
                  c = _getch();

                  if (c == '\r')
                        break;
            }
      }
}

void reset(const int menu)
{
      /*菜单项 1-3 输出完成后不需要移动光标，输出换行符即可*/
      if (menu >= 1 && menu <= 3)
            cout << endl;

      /*重置颜色*/
      cct_setcolor();

      /*菜单项 4-9 输出完成后需要移动光标，根据菜单函数返回值调整参数*/
      if (menu >= 4 && menu <= 8)
            cct_gotoxy(0, CONSOLE_BASE_Y + PLATE_NUM_LIMIT + 9);

      else if (menu == 9)
            cct_gotoxy(0, CONSOLE_BASE_Y + PLATE_NUM_LIMIT + 13);

      cout << "按回车键继续"; /*输出提示语*/
      __stepnumber = 0;          /*重置记录步数的静态全局变量*/
      __delay_time = 0;          /*重置标志延时时长的静态全局变量*/

      pause(0);   /*所有重置项完成后的中断*/
      cct_cls();  /*清理屏幕*/
}

void command_center(int number_of_plates, char source_tower, char temporary_tower, char destination_tower, int delay_time_set, int menu)
{
      /*清空屏幕*/
      if (menu >= 4 && menu <= 9)
            cct_cls();

      /*内部数组初始化*/
      if (menu == 3 || menu == 4 || menu == 7 || menu == 8 || menu == 9)
            array_tower_initialize(number_of_plates, source_tower);

      command_info(number_of_plates, source_tower, destination_tower, delay_time_set, menu);  /*输出提示语*/
      command_array_base(number_of_plates, source_tower, destination_tower, menu);  /*输出纵向数组*/
      command_console_base(number_of_plates, source_tower, menu);  /*输出柱子和盘子*/
      command_delay(menu);  /*控制中断或延时*/

      /*调用递归函数*/
      if (menu != 5 && menu != 6 && menu != 9)
            hanoi(number_of_plates, source_tower, temporary_tower, destination_tower, menu);

      /*运行游戏*/
      if (menu == 9)
            game_command_center(number_of_plates, destination_tower, menu);
}

void command_info(const int number_of_plates, const char source_tower, const char destination_tower, const int delay_time_set, const int menu)
{
      if (menu == 4 || menu == 6 || menu == 7 || menu == 8 || menu == 9)
            cout << "从 " << source_tower << " 移动到 " << destination_tower << ", 共 " << number_of_plates << " 层";

      if (menu == 4 || menu == 8)
            cout << " , 延时设置为 " << delay_time_set;
}

void command_array_base(const int number_of_plates, const  char source_tower, const char destination_tower, const int menu)
{
      if (menu == 4 || menu == 8 || menu == 9) {
            array_print_tower(number_of_plates, source_tower, destination_tower, menu);
            array_print_base(number_of_plates, source_tower, menu);
      }
}

void command_console_base(const int number_of_plates, const  char source_tower, const int menu)
{
      if (menu >= 5 && menu <= 9)
            console_print_base();

      if (menu >= 6 && menu <= 9)
            console_print_plates(number_of_plates, source_tower);
}

void command_delay(const int menu)
{
      if (menu == 4 || menu == 8)
            pause(__delay_time);

      else if (menu == 7)
            Sleep(1000);

      else if (menu == 9)
            Sleep(100);
}
```

2. hanoi_main.cpp

```
/*2151294 信11 马威*/
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <Windows.h>
#include "cmd_console_tools.h"
#include "hanoi.h"
using namespace std;

int main()
{
      cct_setconsoleborder(120, 40, 120, 9000);

      while (1) {
            int MENU = menu();
            cout << MENU << "\n\n\n";
            Sleep(200);

            if (MENU == 0)
                  break;

            int number_of_plates, * nop = &number_of_plates;
            int delay_time_set = 0, * dts = &delay_time_set;
            char src_tower, * st = &src_tower;
            char dst_tower, * dt = &dst_tower;
```

```
            input_info(nop, st, dt, dts, MENU);

            char tmp_tower = 'A' + 'B' + 'C' - src_tower -
dst_tower;

            cct_setcursor(CURSOR_INVISIBLE);

            command_center(number_of_plates, src_tower,
tmp_tower, dst_tower, delay_time_set, MENU);

            cct_setcursor(CURSOR_VISIBLE_NORMAL);

            reset(MENU);
      }

      return 0;
}
```

3. hanoi_menu.cpp

```
/*2151294 信11 马威*/
#include<iostream>
#include<conio.h>
using namespace std;

int menu()
{
      int ret;
      cout << "--------------------------------\n"
            << "1.基本解\n"
            << "2.基本解(步数记录)\n"
            << "3.内部数组显示(横向)\n"
            << "4.内部数组显示(纵向+横向)\n"
            << "5.图形解-预备-画三个圆柱\n"
            << "6.图形解-预备-在起始柱上画 n 个盘子\n"
            << "7.图形解-预备-第一次移动\n"
            << "8.图形解-自动移动版本\n"
            << "9.图形解-游戏版\n"
            << "0.退出\n"
            << "--------------------------------\n"
            << "[请选择:] ";

      while (1) {
            ret = _getch();

            if (ret >= '0' && ret <= '9') {
                  break;
            }
      }

      return ret - '0';
}
```