同济大学计算机系

汉诺塔实验报告

学　　号 1850772

姓　　名 张哲源

专　　业 计算机科学与工程

授课老师 沈坚老师

目录

1. 题目（黑体，四号，段前段后间距各1行）

汉诺塔综合展示

1.1. 题目要求

将之前的汉诺塔功能集成到一个题目中，功能如下：

1. 基本解(打印盘移动和对应的柱子)
2. 基本解(同1，加步数记录)
3. 内部数组演示(打印三个数组，分别装盘的标号，每一步打印数组变化)
4. 内部数组演示(用纵向的数组表示塔，打印出变化，同时打印横向数组)
5. 图形解画三个圆柱
6. 图形解画n个柱子
7. 图形解画第一次移动
8. 图形解汉诺塔
9. 游戏解，将汉诺塔的移动用输入控制
0. 退出

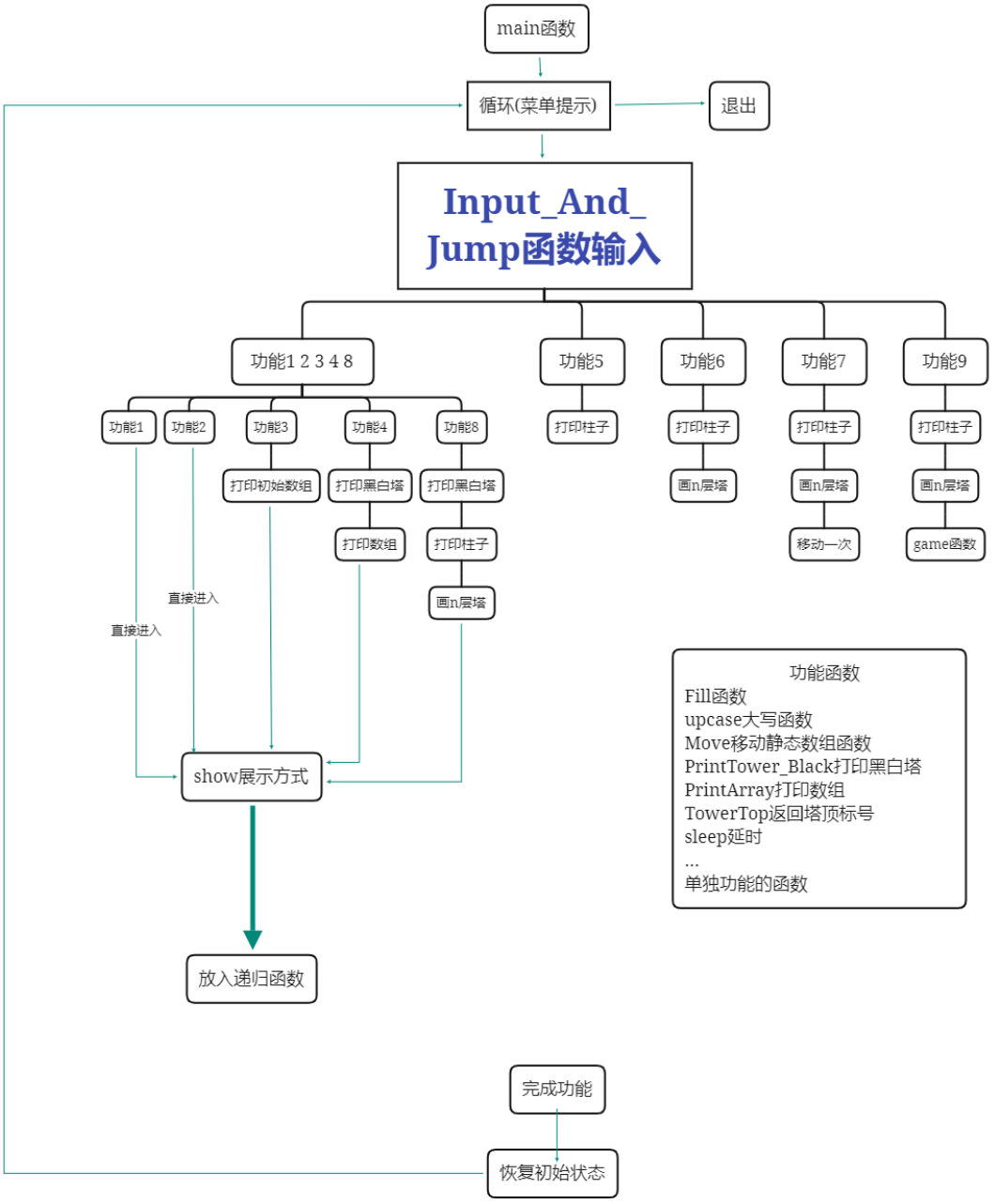
2. 整体设计思路

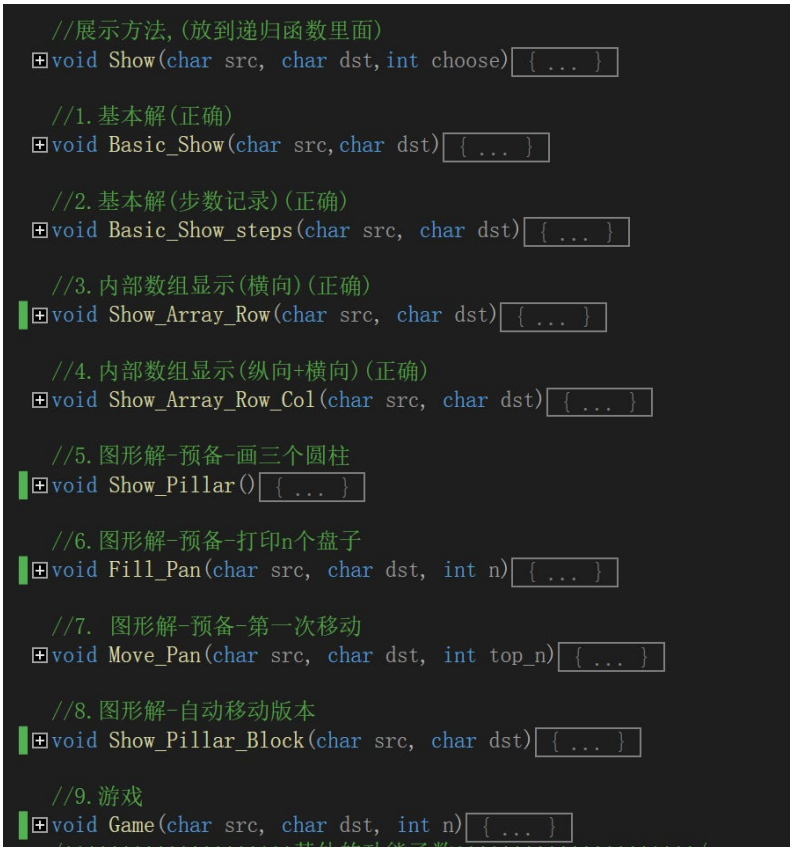
整体设计思路如下

1. 整合之前的作业，对应功能1， 2， 3， 4 ，用菜单函数装起来，能够跳转到不同的功能
2. 新增5个功能，分别是画彩色柱子，画初始化的n个盘子，画盘子移动的过程，输入控制移动的过程，分别对应功能 5，6，7，8，9
3. 程序有递归的部分和不能递归的部分，功能1，2，3，4，8对应递归功能，功能5，6，7，9不需要递归功能，递归函数里面需要放一个展示方式，输入的参数只限于1，2，3，4，8，其他的功能通过跳转函数跳转至其他的功能即可
4. 完成一个功能之后需要恢复默认状态，需要定义恢复函数，恢复初始状态

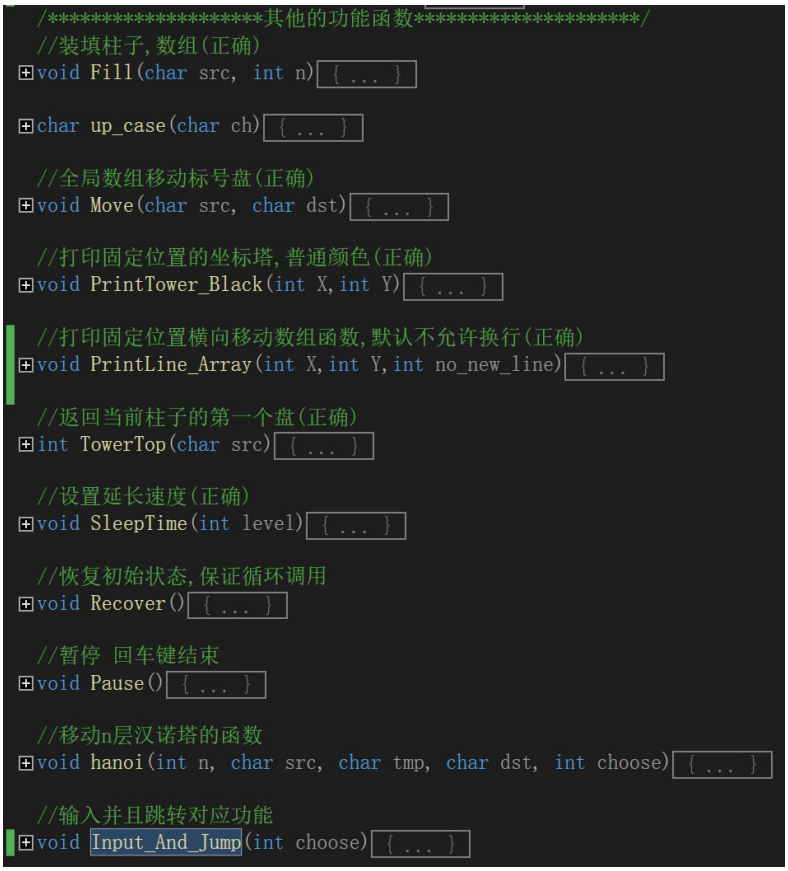
3. 主要功能的实现

9个功能定义9个功能函数，9个函数里面嵌套调用一些其他的辅助功能函数，通过跳转函数进入到不同的功能


main函数
循环(菜单提示)
退出
Input_And_
Jump函数输入
功能1 2 3 4 8
功能5
功能6
功能7
功能9
功能1
功能2
功能3
功能4
功能8
打印柱子
打印柱子
打印柱子
打印柱子
打印初始数组
打印黑白塔
打印黑白塔
画n层塔
画n层塔
画n层塔
打印数组
打印柱子
移动一次
game函数
画n层塔
直接进入
直接进入
功能函数
Fill函数
upcase大写函数
Move移动静态数组函数
PrintTower_Black打印黑白塔
PrintArray打印数组
TowerTop返回塔顶标号
sleep延时
...
单独功能的函数
show展示方式
放入递归函数
完成功能
恢复初始状态



```
//展示方法,(放到递归函数里面)
void Show(char src, char dst,int choose){ ... }

//1.基本解(正确)
void Basic_Show(char src,char dst){ ... }

//2.基本解(步数记录)(正确)
void Basic_Show_steps(char src, char dst){ ... }

//3.内部数组显示(横向)(正确)
void Show_Array_Row(char src, char dst){ ... }

//4.内部数组显示(纵向+横向)(正确)
void Show_Array_Row_Col(char src, char dst){ ... }

//5.图形解-预备-画三个圆柱
void Show_Pillar(){ ... }

//6.图形解-预备-打印n个盘子
void Fill_Pan(char src, char dst, int n){ ... }

//7. 图形解-预备-第一次移动
void Move_Pan(char src, char dst, int top_n){ ... }

//8.图形解-自动移动版本
void Show_Pillar_Block(char src, char dst){ ... }

//9.游戏
void Game(char src, char dst, int n){ ... }
```



```
/*********************其他的功能函数**********************/
//装填柱子,数组(正确)
void Fill(char src, int n){ ... }

char up_case(char ch){ ... }

//全局数组移动标号盘(正确)
void Move(char src, char dst){ ... }

//打印固定位置的坐标塔,普通颜色(正确)
void PrintTower_Black(int X,int Y){ ... }

//打印固定位置横向移动数组函数,默认不允许换行(正确)
void PrintLine_Array(int X,int Y,int no_new_line){ ... }

//返回当前柱子的第一个盘(正确)
int TowerTop(char src){ ... }

//设置延长速度(正确)
void SleepTime(int level){ ... }

//恢复初始状态,保证循环调用
void Recover(){ ... }

//暂停 回车键结束
void Pause(){ ... }

//移动n层汉诺塔的函数
void hanoi(int n, char src, char tmp, char dst, int choose){ ... }

//输入并且跳转对应功能
void Input_And_Jump(int choose){ ... }
```

主要的思路就是利用函数嵌套调用的方法,抽成9个功能模块,在Input_And_Jump函数里面实现跳转

4. 调试过程碰到的问题

问题 1:出现不能编译的问题

把前面的作业拿过来的时候,抽成4个模块实现功能时发现不能正常编译

群里问了才发现是因为宏定义的问题,和系统文件有冲突,导致出现了错误

```
#define A 0
#define B 1
#define C 2                 //全局变量二维数组的三个塔

#define SIZE 10             //汉诺塔最多的层数
#define BEGIN_X 10          //要打印的塔的横坐标位置
#define BEGIN_Y 11          //要打印的塔的纵坐标位置
#define LINE 17             //“==========”的位置

#define PILLAR_COLOR 14     //定义柱子的颜色是黄色
#define BG_COLOR 0          //定义背景颜色黑色
#define P_WIDTH  20         //定义柱子底座的宽度
#define P_HEIGHT 12         //定义柱子的长度
#define P_SEPERATE 5        //定义柱子之间的距离
#define P_X 1
#define P_Y 15              //柱子的起始坐标

int count_n = 1;            //计数器,记录第几步
int Top[3] = { 0 };         //栈顶指针
int Tower[3][SIZE] = { {0} };   //三个塔
int level = 0;              //延时等级
```

```
    DWORD Reg : 3;
    DWORD R : 1;
    DWORD L : 1;
    DWORD C : 1;
    DWORD StackAdjust : 10;
} DUMMYSTRUCTNAME;
```

而且与此同时根据沈坚老师的点播,发现不能把全局变量放到头文件里面,回去复习了下第四章的内容,把全局变量放回到了功能 cpp 文件里面

问题 2:盘色块移动的问题

实现盘色块功能时出现了很多的问题,例如出现不想要的色块,以及色块擦不干净等

中间错误的问题没有过多截图,最后调出来了之后,总结解决思路大概如下

1. 总的移动方块有三个步骤:

   把柱子挪上去

   把柱子平移

   把柱子放下来

```
//每一个块的颜色
int block_color = top_n, fg_color = 0;//字体颜色不需要

//让坐标到x点对应宽度的位置,比如说10号盘就是x前10个位置
cct_gotoxy(x - top_n, y);

//从柱子上挪上去
while (y >= P_Y - P_HEIGHT) { ... }

//盘的平移
if (src - dst < 0) { ... }
else { ... }

cct_gotoxy(x - top_n, y);//此时回到了对应柱子的上方

//盘的下降,和盘的上升逻辑一样
while (y < P_Y - Top[dst - 'A'] - 1) { ... }
```

2. 移动的单个过程擦一块,画一块

   上升和下降的时候给中间的柱子要补颜色

   平移不用,

   每次画完之后到新的位置即可

```
//先删掉一行,在往上面画一行
for (int i = -top_n; i <= top_n; i++)
{
    if (i == 0)//柱子中间补一块颜色
        cct_setcolor(PILLAR_COLOR, 0);
    else
        cct_setcolor();
    cout << " ";//输出黑色擦掉当前色块
}

//擦完之后x回到两格前,y要往上走一格
cct_gotoxy(x - top_n, --y);

//然后再打印一格
cct_setcolor(block_color, fg_color);//设置回颜色
for (int i = -top_n; i <= top_n; i++)
    cout << " ";
```

3. 移动的位置由栈顶的元素决定

以挪上去为例

从栈顶开始挪,初始化的x和y的坐标指向栈顶的元素

移动的三个临界点

栈顶块的位置减去柱长,坐标柱的横坐标,栈顶块的位置

每次擦完就往上挪一格

然后画一层

再擦一层

依次循环往复

```
//初始化两个坐标,指向起始柱的底部
int x = P_X + P_WIDTH / 2 + (P_WIDTH + P_SEPERATE) * (src - 'A');
int y = P_Y - Top[src - 'A'];

//每一个块的颜色
int block_color = top_n, fg_color = 0;//字体颜色不需要

//让坐标到x点对应宽度的位置,比如说10号盘就是x前10个位置
cct_gotoxy(x - top_n, y);

//从柱子上挪上去
while (y >= P_Y - P_HEIGHT)//没移动到顶
{
    //先删掉一行,在往上面画一行
    for (int i = -top_n; i <= top_n; i++){ ... }

    //擦完之后x回到两格前,y要往上走一格
    cct_gotoxy(x - top_n, --y);

    //然后再打印一格
    cct_setcolor(block_color, fg_color);//设置回颜色
    for (int i = -top_n; i <= top_n; i++)
        cout << " ";

    //打印完之后回到初始点,等待下一次循环擦掉
    cct_gotoxy(x - top_n, y);

    SleepTime(t);
}
```

问题 3:数组变化和画出这个变化怎么放到递归里面

移动函数有两个,一个是数组变化,一个是画出来这个块,一开始以为要先变化数组,然后再画移动,后来发现不是这样,结果是错误的

调试发现,原来我定义移动时就是基于当前状态到下一步的,所以应该把画移动放到前,然后再变化数组

```
void Show_Pillar_Block(char src, char dst)
{
    PrintTower_Black(BEGIN_X, BEGIN_Y + MOVE_Y_ARRAY);
    SleepTime(level);
    Move_Pan(src, dst, TowerTop(src));
    cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY);
    Basic_Show_steps(src, dst);
    PrintLine_Array(ARRAY_X,ARRAY_Y+MOVE_Y_ARRAY);
}
```

```
void hanoi(int n, char src, char tmp, char dst, int choose)
{
    if (n == 0)
        return;
    hanoi(n - 1, src, dst, tmp, choose);//把n-1层的柱子放到tmp
    step_n++;//计数器增加
    /*****************展示方式*****************/

    Show(src, dst, choose);

    /*****************************************/
    hanoi(n - 1, tmp, src, dst, choose);
}
```

```
void Show(char src, char dst,int choose)
{
    //5 6 7 9应该是用不到递归的
    switch (choose)
    {
        case 1://1.基本解
            Basic_Show(src, dst);
            break;
        case 2://2.基本解(步数记录)
            Basic_Show_steps(src, dst);
            cout << endl;
            break;
        case 3://3.内部数组显示(横向)
            Show_Array_Row(src, dst);
            break;
        case 4://4.内部数组显示(横向+纵向)
            Show_Array_Row_Col(src, dst);
            break;
        case 8://8.图形解,自动移动版本
            Show_Pillar_Block(src,dst);
            break;
        default:
            break;
    }
```

其实这个是试出来的,但是调出来之后再看,就能理解这个为什么是这样做的了

5. 心得体会

总的感觉有两个方面吧,第一次完成大作业,像是完成了一个工程,就感觉像自己写了一个小游戏发到网上了,然后等着甲方爸爸给钱就行了.

(虽然我知道自己还不配hhhhha)

1. 嵌套调用函数和功能的抽离

在添加新的功能之前,在抽离之前自己写的东西时,会发现这个过程会不好实现,尤其是发现第一

次抽离的时候,从主函数到后面,发现会有很多的重复部分,虽然汉诺塔9个功能中本来就有很多的重叠部分,但是我觉着其实重点不是减少代码量,提高代码复用率,而是能通过抽调,让整个程序有更清晰的逻辑,这样调试的时候,就知道改修改哪些功能,哪些功能是不用动的.

自己在一开始调的过程中,也能感觉到逻辑清晰的重要性,一开始我的想法是把整个程序弄成树那样的逻辑形式,就是根据输入,然后把对应的功能加一点点东西,但是发现9个功能中不全是递归,这样会弄的递归函数里面很麻烦.而且不利于出现单个问题的调试.

所以最后9个功能,我用了9个函数,然后用一个跳转函数来读取对应的参数,让函数能够跳转到对应的功能.

这9个函数中嵌套调用基本功能函数,然后通过逐一调试,确保基本的功能函数绝对的正确,然后再把基本功能放到9个函数中就行了.发现这样下来,调试效率就会非常高

2. 小工具的使用

看小工具的演示时,体验还是比较好的,就发现原来控制台程序居然也能这么牛逼,然后在小工具里面找个几个自己能用的功能,调试会用之后,然后就开始用到自己的函数里面去.

感觉这些工具还是非常好用的,而且调试过程中也没有出现任何的错误,而且小工具的代码可读性也很强,调试过程也不至于再打开小工具cpp文件再看注释.

6. 附件：源程序

(Hanoi_mutiple_solution)

```
/* 1850772 张哲源 计科 */

#include"hanoi.h"
#include"cmd_console_tools.h"
#include<windows.h>
#include<conio.h>
#include<iomanip>
#include<iostream>
using namespace std;

int step_n = 0;                    //计数器,记录第几步
int Top[3] = { 0 };                //栈顶指针
int Tower[3][TSIZE] = { {0} };     //三个塔
int level = -1;                    //延时等级
//展示方法,(放到递归函数里面)
void Show(char src, char dst,int choose)
{
    //5 6 7 9 应该是用不到递归的
    switch (choose)
    {
        case 1://1.基本解
            Basic_Show(src, dst);
            break;
        case 2://2.基本解(步数记录)
            Basic_Show_steps(src, dst);
            cout << endl;
            break;
        case 3://3.内部数组显示(横向)
            Show_Array_Row(src, dst);
            break;
        case 4://4.内部数组显示(横向+纵向)
            Show_Array_Row_Col(src, dst);
            break;
        case 8://8.图形解,自动移动版本
            Show_Pillar_Block(src,dst);
            break;
        default:
            break;
    }
}
//1.基本解(正确)
void Basic_Show(char src,char dst)
{
    cout << TowerTop(src) << "# " << src << "---->" << dst<<endl ;
    Move(src, dst);
}
//2.基本解(步数记录)(正确)
void Basic_Show_steps(char src, char dst)
{
    cout << "第" << setw(4) << step_n << "步";
    cout << '(' << TowerTop(src) << "#: ";
    cout << src << "-->" << dst << ')';
```

```
    Move(src, dst);
}
//3.内部数组显示(横向)(正确)
void Show_Array_Row(char src, char dst)
{
    Basic_Show_steps(src, dst);
    PrintLine_Array(0,0,0);//换行打印
}
//4.内部数组显示(纵向+横向)(正确)
void Show_Array_Row_Col(char src, char dst)
{
    SleepTime(level);
    PrintTower_Black();
    cct_gotoxy(0, ARRAY_Y);
    Basic_Show_steps(src, dst);
    PrintLine_Array();
    SleepTime(level);
    PrintTower_Black();
}
//5.图形解-预备-画三个圆柱
void Show_Pillar()
{
    int t = 4;//定义延长的时间,方便后期调试
    cct_setcursor(CURSOR_INVISIBLE);//让光标看不见
    cct_setcolor(PILLAR_COLOR, BG_COLOR);//黄柱子,黑背景
    for (int pillar = 0; pillar < 3; pillar++)//三个柱子打印三次
    {
        for (int i = 0; i <= P_WIDTH - 1; i++)//打印底座
        {
            cct_gotoxy(P_X + pillar * (P_WIDTH + P_SEPERATE) + i, P_Y);
            cout << " ";
        }
        for (int i = 0; i <= P_HEIGHT; i++)//打印纵柱
        {
            cct_gotoxy(P_X + (P_WIDTH / 2) + pillar * (P_WIDTH + P_SEPERATE), P_Y - i);
            cout << " ";
            SleepTime(t);
        }
    }
    cct_setcursor(CURSOR_VISIBLE_NORMAL);//恢复光标
    cct_setcolor();//设置回颜色,默认参数是黑白
    cct_gotoxy(0, 30);//光标返回到下面,防止不必要的错误
}
//6.图形解-预备-打印 n 个盘子
void Fill_Pan(char src, char dst, int n)
{
    int t = 5;
    cct_setcursor(CURSOR_INVISIBLE);//让光标看不见
    cct_gotoxy(0, 0);//光标归位
    cout << "从" << src << "移动到" << dst << ",共" << n << "层";
    //初始坐标在起始柱的底部
    int x = P_X + P_WIDTH / 2 + (P_WIDTH + P_SEPERATE) * (src - 'A');
    int y = P_Y - 1;
    for (int i = n; i >= 1; i--)//从 n 层打印到 1 层
    {
        cct_setcolor(i);//设置颜色
        cct_gotoxy(x - i, y - (n - i));
        for (int j = -i; j <= i; j++)
            cout << " ";//打印 2n+1 个块的颜色
        SleepTime(t);
    }
    cct_setcursor(CURSOR_VISIBLE_NORMAL);//恢复光标
    cct_setcolor();
    cct_gotoxy(0, 30);
}
//7. 图形解-预备-第一次移动
void Move_Pan(char src, char dst, int top_n)
{
    int t = (level == -1||level==0) ? 3 : level;//延时时间,方便修改
    cct_setcursor(CURSOR_INVISIBLE);//让光标看不见
    //初始化两个坐标,指向起始柱的底部
    int x = P_X + P_WIDTH / 2 + (P_WIDTH + P_SEPERATE) * (src - 'A');
    int y = P_Y - Top[src - 'A'];
    //每一个块的颜色
    int block_color = top_n, fg_color = 0;//字体颜色不需要

    //让坐标到 x 点对应宽度的位置,比如说 10 号盘就是 x 前 10 个位置
    cct_gotoxy(x - top_n, y);

    //从柱子上挪上去
    while (y >= P_Y - P_HEIGHT)//没移动到顶
```

```
    {
        //先删掉一行,在往上面画一行
        for (int i = -top_n; i <= top_n; i++)
        {
            if (i == 0)//柱子中间补一块颜色
    cct_setcolor(PILLAR_COLOR, 0);
            else
                cct_setcolor();
            cout << " ";//输出黑色擦掉当前色块
        }

        //擦完之后 x 回到两格前,y 要往上走一格
        cct_gotoxy(x - top_n, --y);

        //然后再打印一格
        cct_setcolor(block_color,  fg_color);//设置回颜色
        for (int i = -top_n; i <= top_n; i++)
            cout << " ";
        //打印完之后回到初始点,等待下一次循环擦掉
        cct_gotoxy(x - top_n, y);

        SleepTime(t);
    }
    //盘的平移
    if (src - dst < 0)//如果目标柱子在右边,就右移
    {
        while (x < P_X + P_WIDTH / 2 + (P_WIDTH + P_SEPERATE) * (dst - 'A'))
        {
            //先回到那个位置擦掉
            cct_gotoxy(x - top_n, y);
            cct_setcolor();
            for (int i = -top_n; i <= top_n; i++)
                cout << " ";
            //然后到下一个位置打印
            cct_gotoxy(++x - top_n, y);
            cct_setcolor(block_color, 0);
            for (int i = -top_n; i <= top_n; i++)
                cout << " ";
            SleepTime(t);
        }
    }
    else //在左边就左移
    {
        while (x > P_X + P_WIDTH / 2 + (P_WIDTH + P_SEPERATE) * (dst - 'A'))
        {
            //先回到那个位置擦掉
            cct_gotoxy(x - top_n, y);
            cct_setcolor();
            for (int i = -top_n; i <= top_n; i++)
                cout << " ";

            //然后到下一个位置打印
            cct_gotoxy(--x - top_n, y);
            cct_setcolor(block_color, 0);
            for (int i = -top_n; i <= top_n; i++)
                cout << " ";

            SleepTime(t);
        }
    }
    cct_gotoxy(x - top_n, y);//此时回到了对应柱子的上方

    //盘的下降,和盘的上升逻辑一样
    while (y < P_Y - Top[dst - 'A'] - 1)
    {
        for (int i = -top_n; i <= top_n; i++)
        {
            if (i == 0 && y != P_Y - P_HEIGHT - 1)

    cct_setcolor(PILLAR_COLOR);
            else
                cct_setcolor();
            cout << " ";
        }
        cct_gotoxy(x - top_n, ++y);
        cct_setcolor(block_color);
        for (int i = -top_n; i <= top_n; i++)
            cout << " ";
        cct_gotoxy(x - top_n, y);

        SleepTime(t);
    }

    cct_gotoxy(0, y + 2);//往下走两行,防止被别的东西打印覆盖了
    //把颜色设置回来
    cct_setcolor();
    cct_setcursor(CURSOR_VISIBLE_NORMAL);//恢复光标
}

//8.图形解-自动移动版本
void Show_Pillar_Block(char src, char dst)
{
    PrintTower_Black(BEGIN_X,  BEGIN_Y + MOVE_Y_ARRAY);
```

```
    SleepTime(level);
    Move_Pan(src, dst, TowerTop(src));
    cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY);
    Basic_Show_steps(src, dst);
    PrintLine_Array(ARRAY_X,ARRAY_Y+ MOVE_Y_ARRAY);
}

//9.游戏
void Game(char src, char dst, int n)
{
    PrintTower_Black(BEGIN_X, BEGIN_Y + MOVE_Y_ARRAY);
    while (1)
    {
        cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY + 2);//到原先打印数组的下一行打印提示信息
        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) ：";

        int x, y;
        char move1, move2;//输入指令
        move1 = _getche();
        move2 = _getche();
        move1 = up_case(move1);
        move2 = up_case(move2);
        cct_getxy(x, y);//清除当前指令行
        cct_gotoxy(x - 3, y);
        SleepTime(5);
        cout << "          ";
        //退出
        if (move1 == 'Q' && move2 == 13)
        {
            cout << endl << "游戏终止";
            break;
        }

        //移动
        if (move1 >= 'A' && move1 <= 'C' && move2 >= 'A' && move2 <= 'C')
        {
            if (move1 == move2)//相同的情况
            {
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY + 3);
                cout << "两柱不能相同" << endl;
                SleepTime(1);// 过一会清除提示信息
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY + 3);
                cout << "                ";//清除提示消息
            }

            else if (!Top[move1 - 'A'])//源柱为空
            {
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY + 3);
                cout << "源柱为空" << endl;
                SleepTime(1);// 过一会清除提示信息
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY + 3);
                cout << "                ";//清除提示消息
            }
            else if (TowerTop(move1) > TowerTop(move2) && TowerTop(move2))
            {//大盘压小盘的情况
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY + 3);
                cout << "大盘压小盘" << endl;
                SleepTime(1);// 过一会清除提示信息
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY + 3);
                cout << "                ";//清除提示消息
            }
            else//不然就移动盘
            {
                Move_Pan(move1, move2, TowerTop(move1));
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY);
                Basic_Show_steps(move1, move2);

    PrintTower_Black(BEGIN_X, BEGIN_Y + MOVE_Y_ARRAY);

    PrintLine_Array(ARRAY_X, ARRAY_Y + MOVE_Y_ARRAY);
            }

            if (Top[dst - 'A'] == n)
            {
                cct_gotoxy(0, ARRAY_Y +
```

```
MOVE_Y_ARRAY + 3);
                    cout << "游戏结束!!!" << endl;
                    break;
                }
            }
            //非法输入就继续
        }//end while

}
/********************* 其 他 的 功 能 函 数 *********************/
//装填柱子,数组(正确)
void Fill(char src, int n)
{
    switch (src)
    {
        case 'A':
            Top[TA] = n;
            for (int i = 0; i < n; i++)
                Tower[TA][i] = n - i;
            break;
        case 'B':
            Top[TB] = n;
            for (int i = 0; i < n; i++)
                Tower[TB][i] = n - i;
            break;
        case 'C':
            Top[TC] = n;
            for (int i = 0; i < n; i++)
                Tower[TC][i] = n - i;
            break;
        default:
            break;
    }
}

char up_case(char ch)
{
    if (ch >= 'a' && ch <= 'z')
        return ch - 32;
    else
        return ch;
}
//全局数组移动标号盘(正确)
void Move(char src, char dst)
{
    if (src == 'A' && dst == 'B')
    {//A 挪到 B,A 减去一个，B 增加一个
        Tower[TB][Top[TB]++] = Tower[TA][Top[TA] - 1];
        Tower[TA][Top[TA] - 1] = 0;
        Top[TA]--;
    }
    if (src == 'A' && dst == 'C')
    {//A 挪到 C,A 减去一个，C 增加一个
        Tower[TC][Top[TC]++] = Tower[TA][Top[TA] - 1];
        Tower[TA][Top[TA] - 1] = 0;
        Top[TA]--;
    }
    if (src == 'B' && dst == 'A')
    {//B 挪到 A,B 减去一个，A 增加一个
        Tower[TA][Top[TA]++] = Tower[TB][Top[TB] - 1];
        Tower[TB][Top[TB] - 1] = 0;
        Top[TB]--;
    }
    if (src == 'B' && dst == 'C')
    {//B 挪到 C,B 减去一个，C 增加一个
        Tower[TC][Top[TC]++] = Tower[TB][Top[TB] - 1];
        Tower[TB][Top[TB] - 1] = 0;
        Top[TB]--;
    }
    if (src == 'C' && dst == 'A')
    {//C 挪到 A,C 减去一个，A 增加一个
        Tower[TA][Top[TA]++] = Tower[TC][Top[TC] - 1];
        Tower[TC][Top[TC] - 1] = 0;
        Top[TC]--;
    }
    if (src == 'C' && dst == 'B')
    {//C 挪到 B,C 减去一个，B 增加一个
        Tower[TB][Top[TB]++] = Tower[TC][Top[TC] - 1];
        Tower[TC][Top[TC] - 1] = 0;
        Top[TC]--;
    }
    return;
}

//打印固定位置的坐标塔,普通颜色(正确)
void PrintTower_Black(int X,int Y)
{
    for (int i = 9; i >= 0; i--)
    {
        if (Tower[TA][i])//有就打印，不然就把之前的消去
        {
            cct_gotoxy(X, Y - i);
            cout << setw(2) << Tower[TA][i];
        }
        else if (!Tower[TA][i])
        {
            cct_gotoxy(X, Y - i);
            cout << "  ";
        }
```

```
            if (Tower[TB][i])
            {
                cct_gotoxy(X + 10, Y - i);
                cout << setw(2) << Tower[TB][i];
            }
            else if (!Tower[TB][i])
            {
                cct_gotoxy(X + 10, Y - i);
                cout << "  ";
            }
            if (Tower[TC][i])
            {
                cct_gotoxy(X + 20, Y - i);
                cout << setw(2) << Tower[TC][i];
            }
            else if (!Tower[TC][i])
            {
                cct_gotoxy(X + 20, Y - i);
                cout << "  ";
            }
        }
        cct_gotoxy(X - 1, Y + 1);// 9    12 横线的位置
        for (int i = 0; i < 25; i++)
            cout << "=";//打印横线
        cct_gotoxy(X + 1, Y + 2);//11 13 ‘A’的位置
        cout << 'A';
        cct_gotoxy(X + 11, Y + 2);// 21  13 ‘B’的位置
        cout << 'B';
        cct_gotoxy(X + 21, Y + 2);//31    13  'C'的位置
        cout << 'C' << endl;//打印完下一行
    }
    //打印固定位置横向移动数组函数,默认不允许换行(正确)
    void PrintLine_Array(int X,int Y,int no_new_line)
    {
        if(no_new_line)
            cct_gotoxy(X, Y);
        cout << "A:";
        for (int i = 0; i < 10; i++)
            if (Tower[TA][i])
                cout << setw(2) << Tower[TA][i];
            else
                cout << "  ";
        cout << " B:";
        for (int i = 0; i < 10; i++)
            if (Tower[TB][i])
                cout << setw(2) << Tower[TB][i];
            else
                cout << "  ";
        cout << " C:";
        for (int i = 0; i < 10; i++)
            if (Tower[TC][i])
                cout << setw(2) << Tower[TC][i];
            else
                cout << "  ";
        cout << endl;
    }

    //返回当前柱子的第一个盘(正确)
    int TowerTop(char src)
    {
        switch (src)
        {
            case 'A':
                return Tower[TA][Top[TA] - 1];
                break;
            case 'B':
                return Tower[TB][Top[TB] - 1];
                break;
            case 'C':
                return Tower[TC][Top[TC] - 1];
                break;
            default:
                return 0;
                break;
        }
        return 0;
    }

    //设置延长速度(正确)
    void SleepTime(int level)
    {
        switch (level)
        {
            case 0:
                while (1)
                {
                    char c = _getch();
                    if (c == 13)
                        break;
                }
                break;
            case 1:
                Sleep(1000);
                break;
            case 2:
                Sleep(500);
                break;
            case 3:
                Sleep(200);
                break;
            case 4:
```

```
                Sleep(50);
                break;
            case 5:
                Sleep(10);
                break;
            default:
                break;
        }
    }
    //恢复初始状态,保证循环调用
    void Recover()
    {
        //指针清零
        for (int i = 0; i < 3; i++)
            Top[i] = 0;

        //盘清零
        for (int i = 0; i < TSIZE; i++)
        {
            Tower[TA][i] = 0;
            Tower[TB][i] = 0;
            Tower[TC][i] = 0;
        }
        //延时和步数恢复
        level = -1;
        step_n = 0;
        //清屏,用来下一次输出
        cct_setcolor();//颜色归位
        cct_cls();
    }
    //暂停  回车键结束
    void Pause()
    {
        cout << endl;
        cout << "按回车键继续" << endl;

        while (1)
        {
            char ch = _getch();
            if (ch == 13)
                break;
        }
    }
    //移动 n 层汉诺塔的函数
    void hanoi(int n, char src, char tmp, char dst, int
    choose)
    {
        if (n == 0)
            return;
        hanoi(n - 1, src, dst, tmp, choose);//把 n-1
    层的柱子放到 tmp 柱子上面
        step_n++;//计数器增加
        /*****************  展  示  方  式
    ******************/

        Show(src, dst, choose);

        /*********************************
    ************/
        hanoi(n - 1, tmp, src, dst, choose);
    }
    //输入并且跳转对应功能
    void Input_And_Jump(int choose)
    {
        int n;
        char src, tmp, dst;

        if (choose == 5)//功能 5  是不用输入的
        {
            cct_cls();//清屏

            Show_Pillar();//打印柱子

            return;
        }
        //输入汉诺塔的层数(其他都要输入)
        while (1)
        {
            cout << "请输入汉诺塔的层数(1-
    10)" << endl;
            cin >> n;

            cin.clear();
            cin.ignore(100000, '\n');
            if (n >= 1 && n <= 16)
                break;
        }
        //输入起始柱(其他都要输入)
        while (1)
        {
            cout << "请输入起始柱(A-C)" <<
    endl;
            cin >> src;
            cin.clear();
            cin.ignore(100000, '\n');
            if (src >= 'a' && src <= 'c')
                src += 'A' - 'a';
            if (src >= 'A' && src <= 'C')
                break;
        }

        //输入目标柱(其他都要输入)
        while (1)
        {
            cout << "请输入目标柱(A-C)" <<
    endl;
```

```
            cin >> dst;
            cin.clear();
            cin.ignore(100000, '\n');
            if (dst >= 'a' && dst <= 'c')
                dst += 'A' - 'a';
            if (dst >= 'A' && dst <= 'C')
            {
                if (dst == src)
                    cout << "目标柱(" << src << ")不能与起始柱(" << src << ")相同" << endl;
                else
                    break;
            }
        }
        //输入移动速度(功能 4 和功能 8 需要输入)
        if (choose == 4 || choose == 8)
        {
            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 >> level;
                cin.clear();
                cin.ignore(100000, '\n');
                if (level >= 0 && level <= 5)
                    break;
            }
        }
        //转换输入大小写
        {
            if (src <= 'A' && dst == 'B' || dst == 'A' && src == 'B')
                tmp = 'C';
            if (src <= 'A' && dst == 'C' || dst == 'A' && src == 'C')
                tmp = 'B';
            if (src == 'C' && dst == 'B' || dst == 'C' && src == 'B')
                tmp = 'A';
        }
        //装填柱子
        Fill(src, n);
        cct_cls();//清屏

        //打印提示信息(3 需要有初始 ,4 和 8 需要有延时)
        if (choose == 3)
        {
            cout << "初始:                    ";
            PrintLine_Array(19, 0);
        }
        if (choose == 4 || choose == 8)
        {
            cout << "从 " << src << " 移动到 " << dst << "，共 " << n << " 层，延时设置为 " << level << "，";
            cout << "显示内部数组值";
            if (choose == 4)
            {
                cct_gotoxy(0, ARRAY_Y);
                cout << " 初 始 : ";
                PrintLine_Array();
            }
            else
            {
                cct_gotoxy(0, ARRAY_Y + MOVE_Y_ARRAY);
                cout << " 初 始 : ";
                PrintLine_Array(ARRAY_X, ARRAY_Y + MOVE_Y_ARRAY);
            }

        }
        //6 7 8 9 需要画盘子填充盘子
        if (choose == 6 || choose == 7 || choose == 8 || choose == 9)
        {
            Show_Pillar();
            Fill_Pan(src, dst, n);
            if (choose == 6)
                return;//填充完毕就结束
            if (choose == 7)
            {
                Move_Pan(src, dst, TowerTop(src));
                Move(src, dst);
                return;//移动一次就结束
            }
        }
        if (choose == 9)
        {
            level = 5;//快一点不然要等死了
            Game(src, dst, n);
            return;
        }
        //5 6 7 9 不用递归
        if (choose != 5 && choose != 6 && choose != 7 && choose != 9)
            hanoi(n, src, tmp, dst, choose);
        return;
    }
```