Tongji University

模块式编程实践

——以汉诺塔综合演示程序为例

智交 2051995 朱泽凯

2020-11-29

1. 题目

以伪图形界面以用户选择的方式演示九种模式演示汉诺塔求解。

本演示程序的汉诺塔柱子数量为3，盘子数量为1至10之间的整数，其它规则与常见的汉诺塔相同。

其中各模式要求如下：

1.基本解
- 提示用户输入盘数以及始末柱；
- 按顺序逐行输出最优解的每一步（包含信息：盘编号，移动前其所在柱编号，移动后其所在柱编号）。

2.基本解(步数记录)
- 提示用户输入盘数以及始末柱；
- 按顺序逐行输出最优解的每一步（包含信息：步数，盘编号，移动前其所在柱编号，移动后其所在柱编号）。

3.内部数组显示(横向)
- 提示用户输入盘数以及始末柱；
- 按顺序逐行输出最优解的每一步（包含信息：步数，盘编号，移动前其所在柱编号，移动后其所在柱编号，目前三个柱子上分别含有的盘子的编号）。

4.内部数组显示(纵向+横向)
- 提示用户输入盘数，始末柱以及动画播放模式；
- 输出最优解的每一步（包含信息：步数，盘编号，移动前其所在柱编号，移动后其所在柱编号，目前三个柱子上分别含有的盘子的编号）；
- 同时用数字简要模拟出汉诺塔的实物状态；
- 要求用动画演示出不同的步骤。

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
- 提示用户输入盘数，始末柱以及动画播放模式；
- 输出最优解的每一步（包含信息：步数，盘编号，移动前其所在柱编号，移动后其所在柱编号，目前三个柱子上分别含有的盘子的编号）；
- 利用色块模拟画出汉诺塔以及盘子的转移动态；
- 同时用数字简要模拟出汉诺塔的实物状态；
- 要求用动画演示出不同的步骤。
- 注：5.6.7.为 8.的预备。

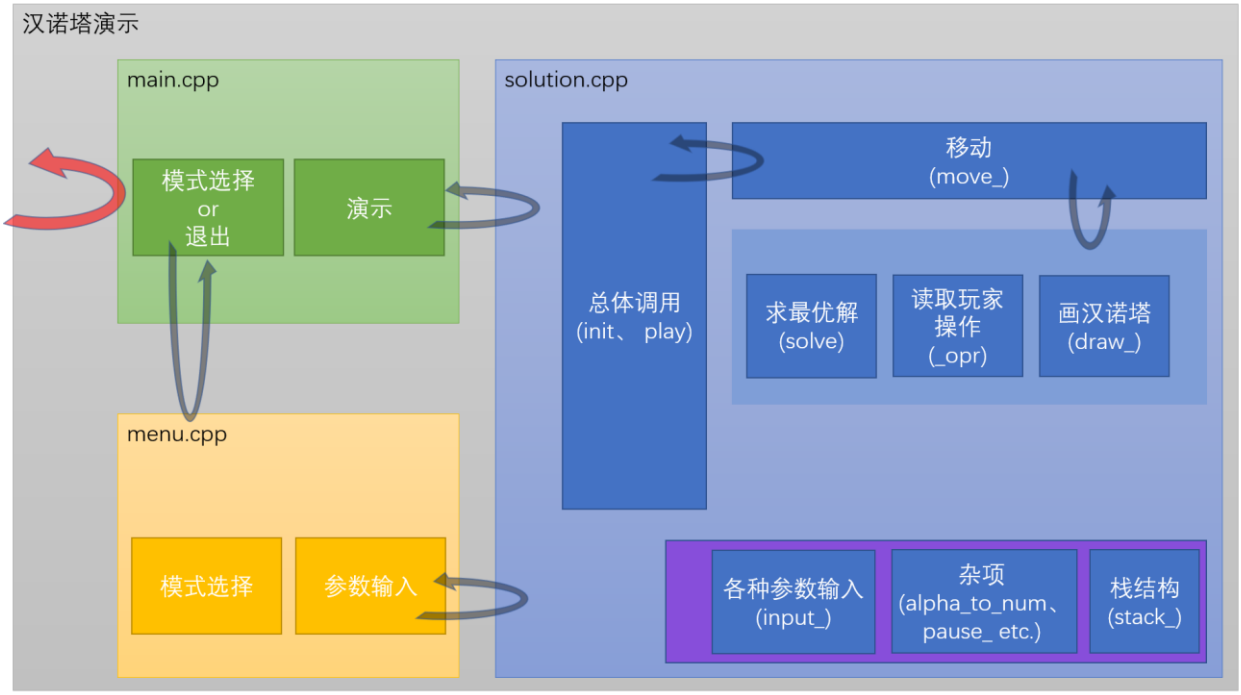
9.图形解-游戏版
- 显示格式要求与 8.相同，但由用户输入移动步骤，需要对步骤的合理性进行检测。

装 订 线

2. 整体设计思路

将整个演示项目拆分成几个模块，一个模块实现一个相对独立的功能，最后用再将各模块整合在一起形成完整的项目。

3. 主要功能的实现



4. 调试过程碰到的问题

• 颜色属性问题

一开始用上 cct 工具库里的相关函数画色块的时候出现了后面输出的提示也被改变颜色了的问题，后来发现是 cct 里面 show 系列函数改变输出属性之后都没有把它们变回去，解决方法是每次使用完改变颜色的函数之后将颜色改回默认值（cct_setcolor）。

• 函数分配问题

虽然将整个项目拆分成了几个模块，但要实现每个模块的功能往往也需要不少的代码。此时，合理地拆分函数可以逐步完成该模块，每步调试好了再往后做，这样代码逻辑性更强，出错的概率降低，后期修改 bug 时目标性也更强。（其实这也是模块化编程的思维）

但代码拆分也不要太琐碎，这样会降低运行效率，而且有可能也会降低可读性。本次实践过程中，一个函数保持在一百行以内较为舒适。

·函数排版问题

项目中后期，源代码中的函数数量已经很多了，调试此时会出现翻找很久才能找到需要的函数的问题。可以在源代码中进行适当的排版，将功能关联性高的函数放在一起，还可以在函数上方写上简明的注释，VisualStudio2019有预览函数的功能。

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

·命名

变量以及函数的命名要简洁而又能让人很容易地感受到其功能。（如input_***，又如draw_*** 等等）

·排版与注释

函数前进行适当的注释，函数中某些操作逻辑较复杂的地方也可以适当地进行注释。但个人认为函数内部的注释还是尽量少写，函数实现中太多注释会让人感到杂乱。应依靠简明的变量名称或在整个函数上方的注释中描述（而非在函数内部穿插注释）来增加程序的可读性。（这同时也要求了单个函数的规模不要太大）

·宏定义

为了让程序弹性更优，我将大部分默认的参数都用宏定义的方式定义在了文件头里。

·函数间的独立性

4中的颜色属性问题揭示了函数应该有的一个性质：尽量少地影响全局环境。
也可以定一个标准，让所有相关函数结束时将全局环境设置为默认标准。

要实现函数的独立性，让函数通过传参方式获取必要信息是一种很好的方法。

·先整体再局部的模块式思维

本次实践我采用了先整体再局部的编程实现方式，也就是先把总体性的程序架构编写调试完成，再去实现每一部分操作函数的具体实现。

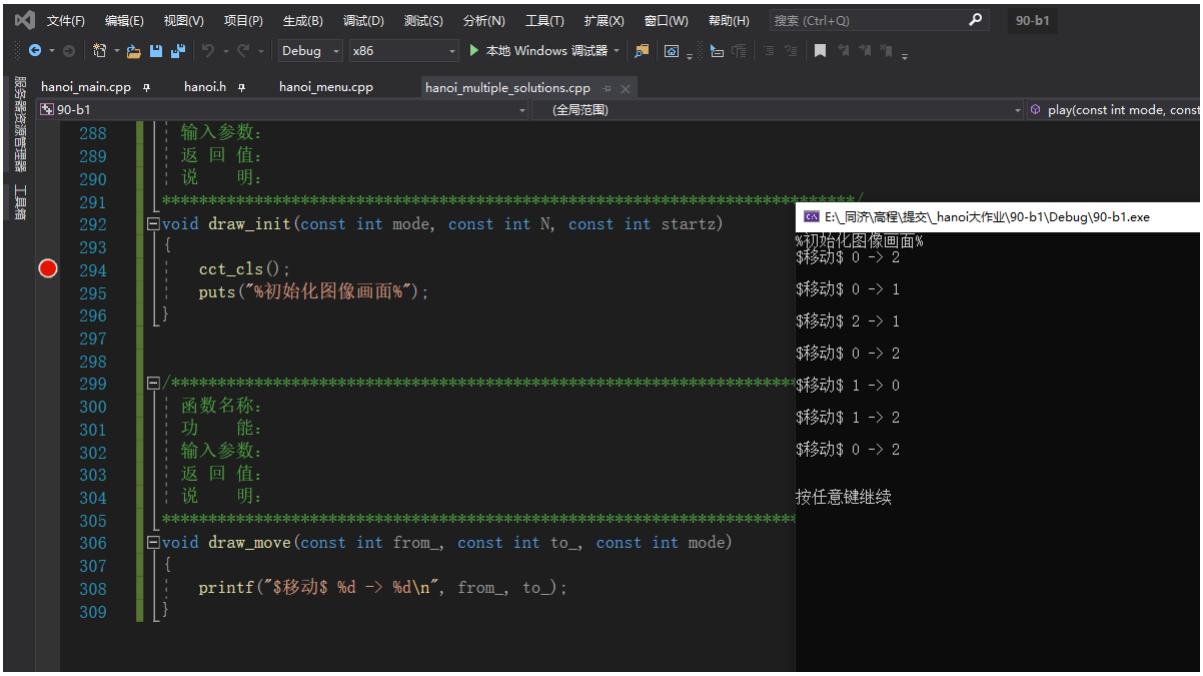
下图为我本次的分模块函数一览：

```c
/* 一些基础函数 */
int alpha_to_num(const char x);
void pause_(const int mode);

/* 栈相关 */
int stack_pop(const int x);
void stack_push(const int x, const int y);
int stack_top(const int x);
void stack_print(const int x);

/* hanoi_menu 中的函数 */
int MENU(const int lstloop, int* N_, int* startz_, int* endz_);
int menu_game_prepare(const int mode, int* N_, int* startz_, int* endz_);

/* 用于输入的一些函数 */
int input_N();
int input_startz();
int input_endz(const int startz);
int input_Time_mark();
int input_opr(char s[]);
int check_opr(const int len, const int st, const int ed);

/* 汉诺塔具体操作流程中用到的函数 */
void init(const int mode, const int N, const int startz, const int endz);
void play(const int mode, const int N, const int startz, const int endz);
void solve(const int n, const int from_, const int to_, const int mode);
void move(const int from_, const int to_, const int mode);

/* 画汉诺塔相关函数 */
void draw_init(const int mode, const int N, const int startz, const int endz, const int Y_ = Y_1);
void draw_move(const int X, const int from_, const int hf, const int to_, const int ht,  const int Y_, const int tag );
void draw_pan_add(const int pan, const int x, const int h, const int Y_);
void draw_pan_del(const int pan, const int x, const int h, const int Y_, const int tag);
```

具体一点来说，声明函数之前，我将需要的函数大致分成两类：“**整体调用函数**”和“**功能实现函数**”，之后先编写“整体调用函数”的逻辑，而在“功能实现函数”里面只简要输出某个特定的字符串（使调试“整体调用函数”的时候知道成功调用了该“功能实现函数”即可）。待“整体调用函数”完成且（初步）确认无误之后，再去进一步编写“功能实现函数”。

注：实际上“整体调用函数”与“功能实现函数”也是一个相对关系，有可能某个“功能实现函数”A 调用到了一些下一级的函数，那么对那些函数来说，A 就是它们的“整体调用函数”（以draw_系列函数为例）。我认为这也是模块化思维的体现。

（下图为例）



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。

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？

是；有尽量这样做，但可能还有提升空间；可以通过集成类似功能到一个总的选择函数中来更好地重用代码，具体可参考源代码中draw_各函数以及move函数的实现。

这种写法的好处在本次实践中也有多处体现，比如我是最后才做模式9（玩家操控）的，而在此之前已经将所有准备工作都调试好了，只差识别用户操作这一部分，所以只用再加一小部分读取操作以及判定操作的逻辑即可完成模式9。（见附件中move函数片段以及模式9的部分逻辑片段）

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

- （以下只列出梗概，更具体的见5.1）
- 对任务进行模块划分。
- 合理地分配函数规模。
- 函数的独立性，足够的必要传参。
- 使用简洁明了歧义少的名称。
- 以合乎逻辑的顺序（例如先整体再局部）逐步将各函数编写、调试完成。
- 勤备份，这样写砸了还能有退路。

6. 附件：源程序

（只给出部分程序，最后还有若干运行截图）

装 订 线

Hanoi_main.cpp

```
int main()
{
    cct_setconsoleborder(120, 40, 120, 9000);
    int N, startz, endz;
    for (int loop = 1; loop; ) {

        loop = MENU(loop, &N, &startz, &endz);

        play(loop, N, startz, endz);

    }
    return 0;
}
```

hanoi_multiple_solutions.cpp

```
/***************************************************************************
  （input_ 模块的代表）
  功    能：（带错误处理地）读入盘子个数
  返 回 值：输入的盘子个数（0~MaxN）
***************************************************************************/
int input_N()
{
    int N;
    for (N = -1; N < 1 || N>MaxN; )
    {
        cout << "请输入汉诺塔的层数(1-" << MaxN << "): " << endl;
        cin >> N;
        cin.clear();
        cin.ignore(65535, '\n');
    }
    return N;
}


/***************************************************************************
  （模式9的部分逻辑）
***************************************************************************/


ready_to_move = 0;

while (!ready_to_move) {  //一直要求输入操作，直到收到可以移动(>0) 或游戏中止(<0) 的操作

    cct_setcursor(CURSOR_VISIBLE_NORMAL);        //光标显示

    cct_gotoxy(60, Y_2 + 7);
    len = input_opr(s);         //读取操作

    cct_setcursor(CURSOR_INVISIBLE);    //光标隐藏

    st = alpha_to_num(s[0]);
    ed = alpha_to_num(s[1]);
```

```
    ready_to_move = check_opr(len, st, ed);

}

if (ready_to_move < 0) {        //游戏中止
    cct_gotoxy(0, Y_2 + 8);
    puts("游戏中止 :(");
    break;
}
else {
    move(st, ed, mode);
    if (top[endz] == N) {
        cct_gotoxy(0, Y_2 + 8);
        puts("游戏结束 :)");
        break;
    }
}



/*****************************************************************************
  求最优解“整体调用函数”代表1
*****************************************************************************/

void solve(const int n, const int from_, const int to_, const int mode)
{
    if (n == 0)
        return;
    int tmp = (from_ + to_) * 2 % 3;
    solve(n - 1, from_, tmp, mode);
    move(from_, to_, mode);
    solve(n - 1, tmp, to_, mode);
}


/*****************************************************************************
  移动操作“整体调用函数”代表2
*****************************************************************************/
void move(const int from_, const int to_, const int mode)
{
    int X = stack_top(from_),hf,ht;

    hf = top[from_];        //获取移动前的高度
    stack_push(to_, stack_pop(from_));    //栈中真实的移动
    ht = top[to_];              //获取移动后的高度

    switch (mode) {
        case 1:
            draw_move(X, from_, 0, to_, 0, Y_1,1);          //基本解输出
            putchar('\n');
            break;

        case 4:
            draw_move(X,from_, hf, to_, ht, Y_1,3);         //数字盘画
            cct_gotoxy(0, Y_1 + 5);
        case 2:
```

```
        case 3:
            cout << "第" << setw(4) << (++CNT) << " 步: "; //步数输出
            draw_move(X, from_, 0, to_, 0,  Y_1,1);            //基本解输出

            if (mode == 3 || mode==4)
                draw_move(X, from_, 0, to_, 0, Y_1, 2);        //栈输出

            putchar('\n');
            pause_(mode);
            break;

        case 7:
            draw_move(X, from_, hf, to_, ht, Y_1, 4); //图画盘
            break;
        case 8:
        case 9:
            draw_move(X, from_, hf, to_, ht, Y_2, 3);      //修改数字盘
            cout << "第" << setw(4) << (++CNT) << " 步: "; //步数输出
            draw_move(X, from_, 0, to_, 0, Y_2, 1);            //基本解输出
            draw_move(X, from_, 0, to_, 0, Y_2, 2);        //栈输出
            draw_move(X, from_, hf, to_, ht, Y_1, 4); //修改画盘
            pause_(mode);
            cct_gotoxy(0, Y_2 + 5);
            break;
    }
}
```

装 订 线

（draw_部分，集成了数字盘和色块盘）

```
/****************************************************************************
  功    能：为移动画图
  输入参数：tag
                1.基本解
                2.栈
                3.数字盘
                4.图画盘
****************************************************************************/
void draw_move(const int X,const int from_,const int hf, const int to_, const int ht, const int Y_,
const int tag)
{
    switch (tag) {
        case 1:
            cout << setw(2) << X << "# " << char(from_ + 'A') << "---->" << char(to_ + 'A');
            break;
        case 2:
            cout << "  ";
            for (int i = 0; i < 3; i++)
                stack_print(i);
            break;
        case 3:
            cct_gotoxy(X_4[from_], Y_ - hf);
            puts("    ");
            cct_gotoxy(X_4[to_], Y_ - ht);
            cout << X;
            cct_gotoxy(0, Y_ + 5);
            break;
        case 4:
```

```
            for (int i = hf, j = H_TOP-1; i <= j; i++) {   //提起
                draw_pan_del(X, X_8[from_], i, Y_,H_ZHU-i);
                //pause_(-1);
                draw_pan_add(X, X_8[from_], i + 1, Y_);
                pause_(Time_mark ? Time_mark * 4 - 23 : -3);
            }

            for (int i = X_8[from_], j = X_8[to_], k = ((j - i > 0) ? 1 : -1); i != j; i += k){
    //平移
                draw_pan_del(X, i, H_TOP, Y_,H_ZHU-i);
                //pause_(-1);
                draw_pan_add(X, i + k, H_TOP, Y_);
                pause_(Time_mark?Time_mark * 4 - 21:-2);
            }

            for (int i = H_TOP ; i > ht; i--) {       //放下
                draw_pan_del(X, X_8[to_], i, Y_,H_ZHU-i);
                //pause_(-1);
                draw_pan_add(X, X_8[to_], i - 1, Y_);
                pause_(Time_mark ? Time_mark * 4 - 23 : -3);
            }
            cct_gotoxy(0, Y_ + 5);
            break;
    }
    //printf("$移动$ %d -> %d\n", from_, to_);      （最初的临时代码）
}
```

（“功能函数”）

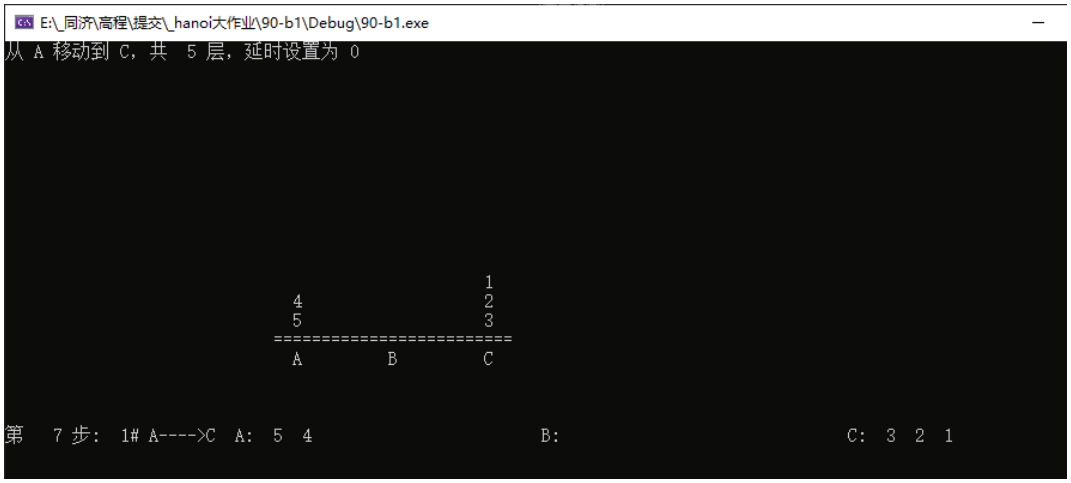
```
void draw_pan_add(const int pan, const int x, const int h, const int Y_)
{
    int bgc, fgc;
    cct_getcolor(bgc, fgc);

    cct_showch(x - pan, Y_ -h, ' ', COLOR_PAN[pan],COLOR_FG, pan * 2 + 1);

    cct_setcolor(bgc, fgc);
}

void draw_pan_del(const int pan, const int x, const int h, const int Y_,const int tag)
{
    int bgc, fgc,tmpbg;
    cct_getcolor(bgc, fgc);

    tmpbg = (tag <0) ? COLOR_BG : COLOR_ZHU;
    cct_showch(x - pan, Y_ - h, ' ', COLOR_BG, COLOR_FG, pan * 2 + 1);
    cct_showch(x, Y_ - h, ' ', tmpbg, COLOR_FG, 1);

    cct_setcolor(bgc, fgc);
}
```

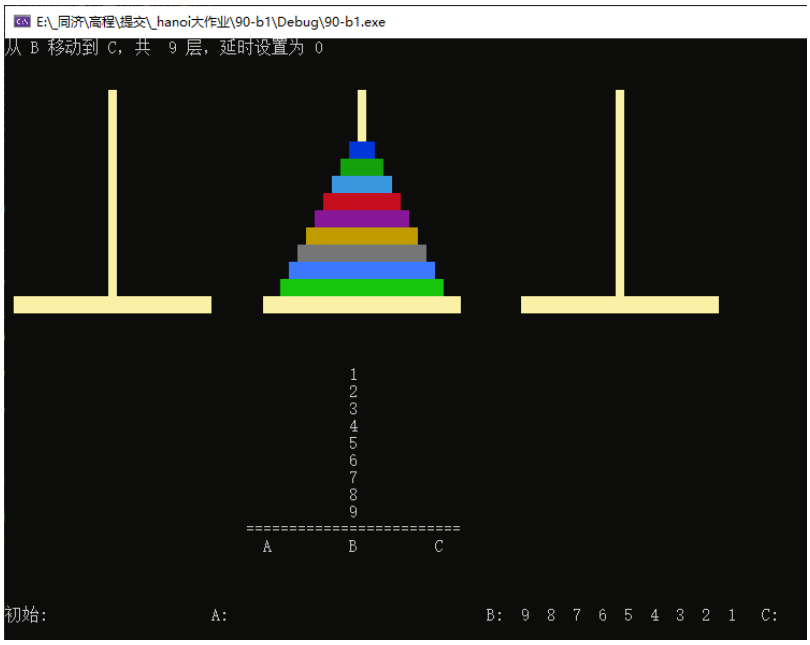
装 订 线

装 订 线

运行截图


E:_同济\高程\提交_hanoi大作业\90-b1\Debug\90-b1.exe
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
[请选择:] 4
请输入汉诺塔的层数(1-10):
5
请输入起始柱(A-C):
a
请输入目标柱(A-C):
c
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)
0


E:_同济\高程\提交_hanoi大作业\90-b1\Debug\90-b1.exe
从 A 移动到 C, 共 5 层, 延时设置为 0
1
4 2
5 3
A B C
第 7 步: 1# A----->C A: 5 4 B: C: 3 2 1


E:_同济\高程\提交_hanoi大作业\90-b1\Debug\90-b1.exe
从 B 移动到 C, 共 9 层, 延时设置为 0
1
2
3
4
5
6
7
8
9
A B C
初始: A: B: 9 8 7 6 5 4 3 2 1 C: