汉诺塔实验报告

实验目标：分不同方式展现汉诺塔移动过程。

题目描述：现有三个圆柱，分别是 A,B,C，随机挑选一座圆柱为起始圆柱，再次随机挑选一座圆柱为目标圆柱，起始圆柱上有 n 个圆盘，从低到高逐渐减小，现要求将这 n 个圆盘移动到目标柱上，要求移动过程中不允许有大圆盘压小圆盘，每次只能移动一个盘子，求整个过程中移动的总次数。并以不同形式将移动过程进行显示。

要求如下：

1：给出基本解,显示每一步中圆盘的起始柱和移动到的柱子

2：给出加数据记录的基本解，要求在 1 的基础上给出每次移动时的次数，即显示这是第几次移动。

3：在 2 的基础上，给出每次移动后各个圆柱上剩余的圆盘编号，编号原则为从低到高一次减小。

4：在 3 的基础上，添加纵向数组，同时设置单步演示和不同延时时长的自动演示，共 6 个版本。

5：图形解预备工作，画出三个圆柱。

6：图形解预备工作，在起始圆柱上画出 n 个盘子。

7：给出图形解的第一次移动。

8：给出图形解的自动移动。

9：给出图形解的游戏版，即手动控制，要求对输入加以判断，同时允许任意时刻的游戏退出。

整体设计思路：

该实验整体思路为递归，采用的方法是通过递归求解，对于有 n 个圆盘的圆柱，采用的思路为，将前 n-1 个圆盘视为一个整体，现将其移动到中间圆柱，再将最下面的圆盘移动到目标柱，最后将这 n-1 个圆盘从中间柱移动到目标柱。

主要功能实现：

1：递归算法，将前 n-1 个视作一个整体，依次逐渐移动即可。

2：定义一个静态全局变量，随着移动的过程依次递增。

3：该功能需要数组或全局变量记录每个圆柱上的盘子编号以及圆柱上的盘子个数。圆柱上的盘子个数两种办法，第一种：定义三个全局变量，第二种定义一个全局一维数组，数组大小为 3。记录盘子编号的方法，第一种，定义三个全局一维数组。第二种，定义一个全局二维数组，大小为 3*10。每次移动后，重新改变各个圆柱上的圆盘个数和圆盘编号，同时重新输出横向数组（两种方法，一种事从头重新输出一遍，没有更改的地方输出不会发生任何改变，另外一种是计算需要改变输出的坐标，计算坐标，精准改变输出。我选的第一种，推荐第二种）。

4：在 3 的基础上，加上了纵向数组，难点在于准确计算相应的坐标，同时改变输出时，只能用计算坐标的方法改变相应的输出。

5：确定好间隔，输出圆柱，在底盘的中间画出柱子。

6：在起始柱通过编号确定圆盘的长度，然后输出相应的圆盘，颜色有圆盘的长度决定。

7：确定好起始柱，第一次移动的目标柱，计算出起始位置的纵坐标，最终位置的纵坐标，起始和最终位置的横坐标，然后利用循环和输出字符函数，实现圆盘的移动。

8：在 7 的基础上，将 7 拓展到每一次移动，利用移动函数自带的延迟，从而实现自动移动。

9：在 8 的基础上加以改进，添加了横向数组和纵向数组显示，同时将模块 8 中自动利用递归输如起始柱和目标柱改为手动输入，添加输入函数，判断函数，并在执行函数中调用

这两个函数，每次移动后，圆柱上的圆盘个数和圆盘编号作出相应的改变，将目标柱上的圆盘个数作为游戏是否结束的判断依据。

调试过程中遇到的问题：

1 横向数组总是输出负数，后来发现是数组赋值是没有考虑负数，输出时也没有考虑负数。改进：赋值时保证数组中全为大于等于 0 的数，同时输出时如果是 0，则输出空格。

2 圆盘移动时，颜色总是为黑色，且总是从第二个开始，原因：圆盘移动时没有考虑好颜色，同时纵坐标计算的不对。

3 模块 9，输入后，字符显示的位置不对，改进方法：在输出前添加定位函数，准确回到目标位置。

心得体会：要认真领会老师给的提示，不要偷懒，思维的懒惰可能会带来一时的轻松，但是会带来后续很多困难。静下心来好好学习。

源程序：

1：相关参数的输入与错误处理

```
void initlize(int& layer, char& src, char& tmp, char& dst)
{
    while (1)
    {
        cout << "请输入汉诺塔的层数(1-10)：" << endl;
        cin >> layer;
        if (layer > 10 || layer < 1)
        {
            cin.clear();
            cin.ignore(65536, '\n');
            ;
        }
        else
            break;
    }
    while (1)
    {
        cout << "请输入起始柱(A-C)：" << endl;
        cin.ignore(65536, '\n');
        cin >> src;
        if (src == 'A' || src == 'a')
        {
            src = 'A';
            break;
        }
        if (src == 'B' || src == 'b')
        {
            src = 'B';
            break;
        }
        if (src == 'C' || src == 'c')
        {
            src = 'C';
            break;
        }
        if (src != 'A' && src != 'a' &&
            src != 'B' && src != 'b' &&
            src != 'C' && src != 'c')
        {
            cin.clear();
            cin.ignore(65536, '\n');
        }
    }
```

关于相应的错误处理，采用的是死循环思路，如果输入的内容不合要求，则不断循环，否则退出循环，关于小写视同大写的处理，采用或运算，即可解决。

```
    while (1)
    {
        cout << "请输入目标柱(A-C):" << endl;
        cin.ignore(65536, '\n');
        cin >> dst;
        if ((dst == 'A' || dst == 'a') && src != 'A')
        {
            dst = 'A';
            break;
        }
        if ((dst == 'B' || dst == 'b') && src != 'B')
        {
            dst = 'B';
            break;
        }
        if ((dst == 'C' || dst == 'c') && src != 'C')
        {
            dst = 'C';
            break;
        }
        if (dst != 'A' && dst != 'a' &&
            dst != 'B' && dst != 'b' &&
            dst != 'C' && dst != 'c')
        {
            cin.clear();
            cin.ignore(65536, '\n');
        }
    }
    while (1)
    {
        if (src != 'C' && dst != 'C')
            tmp = 'C';
        if (src != 'B' && dst != 'B')
            tmp = 'B';
        if (src != 'A' && dst != 'A')
            tmp = 'A';
        break;
    }

}
```

2：简易的递归，总体思路演示

```
void one_move(int layer, char src, char tmp, char dst)
{
    if (layer == 1)
    {
        cout << setw(2) << layer << "#" << src << "-->" << dst << endl;
    }
    else
    {
        one_move(layer - 1, src, dst, tmp);
        cout << setw(2) << layer << "#" << src << "-->" << dst << endl;
        one_move(layer - 1, tmp, src, dst);
    }
}
```

3 添加了计数的递归演示：

```
void two_move(int layer, char src, char tmp, char dst)
{
    static int i = 0;
    if (layer == 1)
    {
        cout << "第" << setiosflags(ios::right) << setw(4) << ++i << "步 (";
        cout << layer << "#" << src << "--->" << dst<< ")" << endl;
    }
    else
    {
        two_move(layer - 1, src, dst, tmp);
        cout << "第" << setiosflags(ios::right) << setw(4) << ++i << "步 (";
        cout << layer << "#" << src << "--->" << dst << ")" << endl;
        two_move(layer - 1, tmp, src, dst);
    }
}
```

4：输出横向数组：

```
cct_gotoxy(X, Z);
for (int i = 0; i < 3; i++)
{
    cout << char(65 + i) << ":";
    for (int j = 0; j < 10; j++)
    {
        if (stack[i][j] == 0)
            cout << "  ";
        else
            cout << stack[i][j] << " ";
    }
    if (i == 2)
        cout << endl;
    else
        ;
}
```

5：输出纵向数组（仅一次）

```
for (int i = 0; i < 30; i++)
{
    cout << "=";
}
cout << endl;
cct_gotoxy(12, 26);
for (int i = 0; i < 3; i++)
{
    cout << "  " << char(65 + i) << "       ";
}
for (int i = 0; i < 3; i++)
{
    if (number[i])
    {
        for (int j = 0; j < layer; j++)
        {
            cct_gotoxy(A + 10 * i, X - j);
            cout << stack[i][j];
        }
    }
    else
        ;
}
```

6：处理圆柱上圆盘个数以及圆盘编号

```
t = stack[src - 'A'][--number[src - 'A']];
stack[dst - 'A'][number[dst - 'A']++] = t;
stack[src - 'A'][number[src - 'A']] = 0;
```

第一行，number[src-'A']先减去一，接着将 stack[src - 'A'][--number[src - 'A']]的值赋给变量 t。

第二行，先将 t 的值赋给 stack[dst - 'A'][number[dst - 'A']]，然后 number[dst - 'A'] 再自增加 1。

第三行，将 0 赋值给 stack[src - 'A'][number[src - 'A']]。

7：更改纵向数组中的元素

```
cct_gotoxy(A + (src - 'A') * 10, X - number[src - 'A']);
cout << " ";
cct_gotoxy(A + (dst - 'A') * 10, Y - number[dst - 'A']);
cout << t;
```

找到以前的位置，输出空格覆盖原来的内容。

找到目标位置，输出数字。

8：输出圆柱

```
cct_showch(1, 15, ' ', 14, 0, 23);
cct_showch(34, 15, ' ', 14, 0, 23);
cct_showch(67, 15, ' ', 14, 0, 23);
for (int y = 14; y > 3; y--)
{
    cct_showch(0, y, ' ', 14, 0, 1);
    cct_showch(0 + 33, y, ' ', 14, 0, 1);
    cct_showch(0 + 66, y, ' ', 14, 0, 1);
    Sleep(100);
}
Sleep(100);
cct_gotoxy(0, 27);
cct_setcolor();
```

找到基准位置，输出底盘和圆柱。

9：画圆盘

```
for (int i = layer; i > 0; i--)
{
    cct_showch((int(src) - 'A') * 33 + 11 - i, 14 + i - layer, ' ', i, 0, 2 * i + 3);
}
```

10：移动

```
void seven_graph_move(char src,char dst)
{
    int src_x = (src - 'A') * 33 + 11 - stack[src - 'A'][number[src - 'A']];
    int src_y = 14 - (number[src - 'A']);
    int dst_x=(dst - 'A') * 33 + 11 - stack[src - 'A'][number[src - 'A']];
    int dst_y = 14 - (number[dst - 'A'])+1;
    /*上移*/
    for (int y = src_y; y > 2; y--) { ... }
    /*左右移动*/
    if (dst_x > src_x) { ... }
    else { ... }
    /*下移*/
    for(int y=3; y<dst_y+1; y++) { ... }
}
```

其中

src_x 是起始圆柱上要移动的圆盘对应的最左侧位置横坐标

src_y 是起始圆柱上要移动的圆盘对应的纵坐标

dst_x 是目标圆柱上要移动的圆盘对应的最左侧位置横坐标

dst_y 是目标圆柱上要移动的圆盘对应的纵坐标

10-1：上移函数

```
/*上移*/
for (int y = src_y; y > 2; y--)
{
    cct_showch(src_x, y, ' ', stack[src - 'A'][number[src - 'A']], 0, 2*stack[src - 'A'][number[src - 'A']]+3);
    Sleep(300);
    if (y>3) {
        /* 清除显示(最后一次保留)，清除方法为用正常颜色+空格重画一遍刚才的位置 */
        cct_showch(src_x, y, ' ', COLOR_BLACK, COLOR_WHITE,stack[src - 'A'][number[src - 'A']]+1);
        cct_showch(src_x + stack[src - 'A'][number[src - 'A']]+1, y, ' ', 14, 0, 1);
        cct_showch(src_x + stack[src - 'A'][number[src - 'A']]+2, y, ' ', COLOR_BLACK, COLOR_WHITE, stack[src - 'A'][number[src - 'A']]+1);
    }
}
```

10-2：左右移动

```
/*左右移动*/
if (dst_x > src_x)
{
    for (int x = src_x; x <dst_x+1; x++)
    {
        cct_showch(x, 3, ' ', stack[src - 'A'][number[src - 'A']], 0, 2 * stack[src - 'A'][number[src - 'A']] + 3);
        Sleep(300);
        if(x<dst_x)
            cct_showch(x, 3, ' ', COLOR_BLACK, COLOR_WHITE, 2 * stack[src - 'A'][number[src - 'A']] + 3);
    }
}
else
{
    for (int x = src_x; x>dst_x-1; x--)
    {
        cct_showch(x, 3, ' ', stack[src - 'A'][number[src - 'A']], 0, 2 * stack[src - 'A'][number[src - 'A']] + 3);
        Sleep(300);
        if (x>dst_x)
            cct_showch(x, 3, ' ', COLOR_BLACK, COLOR_WHITE, 2 * stack[src - 'A'][number[src - 'A']] + 3);
    }
}
```

10-3：下移函数

```
/*下移*/
for(int y=3; y<dst_y+1; y++)
{
    cct_showch(dst_x, y, ' ', stack[src - 'A'][number[src - 'A']], 0, 2*stack[src - 'A'][number[src - 'A']]+3);
    Sleep(300);
    if (y <dst_y) {
        /* 清除显示(最后一次保留)，清除方法为用正常颜色+空格重画一遍刚才的位置 */
        cct_showch(dst_x, y, ' ', COLOR_BLACK, COLOR_WHITE, stack[src - 'A'][number[src - 'A']] + 1);
        cct_showch(dst_x + stack[src - 'A'][number[src - 'A']] + 1, y, ' ', 14, 0, 1);
        cct_showch(dst_x + stack[src - 'A'][number[src - 'A']] + 2, y, ' ', COLOR_BLACK, COLOR_WHITE, stack[src - 'A'][number[src - 'A']] + 1);
    }
}
```

11 自动移动

```
void eight_move(int layer,char src,char tmp,char dst)
{
    static int t = 0;
    if (layer == 1)
    {
        t = stack[src - 'A'][--number[src - 'A']];
        stack[dst - 'A'][number[dst - 'A']++] = t;

        seven_graph_move(src, dst);
        stack[src - 'A'][number[src - 'A']] = 0;
    }
    else
    {
        eight_move(layer - 1, src, dst, tmp);
        t = stack[src - 'A'][--number[src - 'A']];
        stack[dst - 'A'][number[dst - 'A']++] = t;
        seven_graph_move(src, dst);
        stack[src - 'A'][number[src - 'A']] = 0;
        eight_move(layer - 1, tmp, src, dst);
    }
}
```

在递归的基础上添加移动函数，每一次递归均实现移动，通过递归自动输入起始柱和目标柱，然后执行函数，进行移动。

12 游戏模块手动输入

12-1：输入函数

```
void night_scanf(char str[3])
{
    while (1)
    {
        int number = 0;
        cct_gotoxy(56, 33+1);
        cout << "                          ";
        cct_gotoxy(56, 33+1);
        for (int i = 0;; i++, number++)
        {
            str[i] = getchar();
            if (str[i] == '\n')
                break;
            else
                ;
        }
        cct_gotoxy(0, 35);
        cout << "                                                  ";
        if (str[0] == 'q' || str[0] == 'a' || str[0] == 'b' || str[0] == 'c' || str[0] == 'Q' || str[0] == 'A' || str[0] == 'B' || str[0] == 'C')
        {
            if (str[1] == 'a' || str[1] == 'b' || str[1] == 'c' || str[1] == 'A' || str[1] == 'B' || str[1] == 'C' || str[1] == '\n')
                break;
            else
                ;
        }
        else
            ;
    }
}
```

利用死循环，设置退出条件，从而完成错误处理，保证输入的要么是 q 和回车，要么是起始柱，目标柱，回车。

12-2：检查函数

```
int check_is_true(char src, char dst)
{
    if (src == 'A' || src == 'a')
        src = 'A';
    if (src == 'B' || src == 'b')
        src = 'B';
    if (src == 'C' || src == 'c')
        src = 'C';
    if (dst == 'A' || dst == 'a')
        dst = 'A';
    if (dst == 'B' || dst == 'b')
        dst = 'B';
    if (dst == 'C' || dst == 'c')
        dst = 'C';
    if (number[src - 'A'] == 0)
    {
        cout << "移动非法，柱源为空";
        return 0;
    }
    else
    {
        if (number[dst - 'A'] > 0)
        {
            if (stack[src - 'A'][number[src - 'A'] - 1] > stack[dst - 'A'][number[dst - 'A'] - 1] && stack[dst - 'A'][number[dst - 'A'] - 1] > 0)
            {
                cout << "移动非法，大盘压小盘";
                return 0;
            }
            else
                return 1;
        }
        else
            ;
    }
}
```

12-3 递归处理函数

```
void night_move(char src, char dst)
{
    static int t = 0;
    static int i = 1;
    t = stack[src - 'A'][--number[src - 'A']];
    stack[dst - 'A'][number[dst - 'A']++] = t;
    seven_graph_move(src, dst);
    cct_setcolor(0, 7);
    stack[src - 'A'][number[src - 'A']] = 0;
    cct_gotoxy(0, Z+5+1);
    cout << "第" << setw(4) << i++ << "步(" <<t<< "):" << src << "-->" << dst << " ";
    cct_gotoxy(A + (src - 'A') * 10, X - number[src - 'A']+1);
    cout << " ";
    cct_gotoxy(A + (dst - 'A') * 10, Y - number[dst - 'A']+1);
    cout << t;
    cct_gotoxy(X, Z+5+1);
    for (int i = 0; i < 3; i++)
    {
        cout << char(65 + i) << ":";
        for (int j = 0; j < 10; j++)
        {
            if (stack[i][j] == 0)
                cout << "  ";
            else
                cout << stack[i][j] << " ";
        }
        if (i == 2)
            cout << endl;
        else
            ;
    }
}
```

13 游戏总框架

```
for (;;)
{
    while (1)
    {
        int w = 0;
        night_scanf(str);
        while (1)
        {
            if (str[0] == 'q' || str[0] == 'Q')
            {
                check = 0;
                break;
            }
            else
            {
                check = 1;
                break;
            }
        }
        if (check == 0)
        {
            cct_gotoxy(0,35+1);
            cout << "游戏终止"<<endl;
            is_jinxing = 0;
            break;
        }
        else
        {
            cct_gotoxy(0, 35);
            w = check_is_true(str[0], str[1]);
        }
        if (w == 0)
            ;
        else
            break;
    }
    if (is_jinxing == 0)
        break;
    else
        ;
    night_move(str[0], str[1]);
    if (number[dst - 'A'] == layer){ ... }
    else
        ;
}
```

代码缺点:过于冗杂,没有做到合理分解函数并加以调用,各个模块中有很多重复代码。