Tongji University

汉诺塔综合演示

实验报告

国01 2253372 康嘉玮 2022/12/4

1. 题目：汉诺塔综合演示

本题要求用所学知识以及所给源文件当中给出的函数，通过多种方式实现汉诺塔的移动过程。

1.1. 菜单项（要实现的功能）

1/2：基本解，无/有步数显示。

3/4：内部数组显示，3仅显示横向，4同时显示横向和竖向（用数字模拟汉诺塔移动过程）。

5/6/7：汉诺塔移动图形化显示预备部分。5画柱子，6在指定柱子上画盘子，7实现第一次移动。

8/9：汉诺塔移动图形化。8为自动移动，9为游戏形式（需要玩家自己操作直至所有盘子全部转移到目标柱）。

1.2. 基本要求

可以使用静态全局变量记录总移动步数、每根柱子上的圆盘编号及数量和移动中的延时，除此之外不允许使用全局变量。

菜单项除5外都要遇到输入，要求除9外其他所有要用到输入的菜单项必须共用一个输入函数，且仅在此处可使用指针。除此之外禁止使用指针。

3、4、8中横向数组输出要共用一个函数，4、8中纵向数组输出也要共用一个函数，用参数解决输出位置等差异。

5~9画柱子要共用一个函数，7~9移动盘子要共用一个函数。

整个作业只允许使用一个递归函数模拟汉诺塔内部数组移动，且该函数不能超过15行。其余函数建议不超过50行。

每次移动应该是在原本位置擦除、在新位置输出，而非直接清屏并整个重新输出。

2. 整体设计思路

2.1. 菜单函数及主函数简述

2.1.1. 菜单函数

菜单函数负责打印菜单并返回输入的菜单号码。当读取的输入不在0~9时重读直到读到0~9的输入，否则在屏幕上显示并返回读取的输入值。

2.1.2. 主函数

进入一个while(1)循环，在循环中初始化屏幕（缓冲区大小、背景颜色和字体颜色等），并定义几个变量，分别存放菜单函数返回的菜单选项、盘子数量、起始柱、中间柱、目标柱及是否需要输入延时（当菜单选项为4或8时需要输入延时，其余情况均不需要）等信息，然后根据菜单选项调用相应函数。

在一次汉诺塔执行结束后，等待输入直到按下回车键，然后利用函数初始化各全局变量（不直接初始化是因为那些全局变量是静态的，和主函数不在同一.cpp文件，不能直接访问）并清屏，再从头开始循环。

2.2. 各个选择支在主函数中对应的函数调用

2.2.1. 选项1~3

这三个选项都需要输入汉诺塔的相关信息，因此要调用输入函数。在输入相应参数之后，调用对应的输出函数即可（详见“主要功能的实现”部分）。考虑到3如果加上初始数组的输出可能更好，我便在主函数中额外判断了一次选项是否为3，若是，则单独调用一次输出函数以输出初始状态。

2.2.2. 选项4~9

注意到选项5的特殊性（不需要输入），因此程序中判断了一次选项是否为5，若是，则不调用输入函数，否则调用输入函数。

与1~3不同的是，4~9在判断及调用输入函数结束之后，需要执行清屏操作，并将光标设置为不可见。

5~7是图形化汉诺塔输出的预备部分，不是完整的输出。其中，5只需要调用画柱子的函数，6在5的基础上再执行画盘子的函数，7则在5和6的基础上输出一次移动。也就是说，7实际上是要用到递归函数的，但只输出一次移动。因此我在递归函数里加了一个判断语句，以使菜单选项为7时递归函数正常执行但只输出第一次盘子的移动。

4和8则是对整个移动过程的输出，需要完整调用递归函数和相应的输出函数。输出时既要输出横向数组（与3不同，3是就地输出，4和8需要指定一个固定位置然后输出），还要输出纵向数组，对于选项8则还要输出图形化的汉诺塔及移动，这些根据条件调用相关函数即可。

9的输出和8类似，仅部分细节有差别。9和8最大的区别在于是否调用递归函数（8要调用而9不调用）以及移动指令的输入。所以，需要为9的实现专门写一个函数进行指令的读取和内部数组的移动，但完全可以套用8的输出方式输出9的移动过程。

调用完毕后，重新将光标设置为可见。

2.2.3. 选项0

退出循环，结束程序。

3. 主要功能的实现

3.1. 定义静态全局变量

本题中定义了4个全局变量：移动步数_num_of_move，初值为0（初值不设为1是因为后期考虑到某些情况下要输出内部数组等的初始状态）；各个柱子上盘子的编号_col[3][10]（分别表示3根柱子上的10个

位置，非0值表示该位置上盘子的编号，0表示该位置没有盘子），所有元素初值均设为0；各个柱子上圆盘的数量_top[3]（这样命名是因为此数组元素的值表示对应柱子顶上的盘子上面的位置），所有元素初值均设为0；移动延时_speed，其值范围正常为0~5，其中0为手动按回车时继续演示下一次移动，1~5则是自动移动，且时间由1~5按顺序缩短，初值默认为3。

在每次汉诺塔程序执行完成并准备返回菜单时，都要调用一次reset函数。此函数原型为void reset(void)，作用是将所有的静态全局变量值恢复为初始状态。

3.2. 输入函数

输入函数原型为void input_hanoi(unsigned* num, char* src, char* dst, int is_input_speed)。其中num是输入盘子的数量，src和dst分别是起始柱和目标柱编号，is_input_speed是判定是否需要输入延时。

对于前三个形参，由于涉及到类似于“将形参的值传递给实参”之类的问题，所以采用了指针类型。但由于延时是静态全局变量，所以无需单独将其也作为一个形参，其输入（指用cin输入）也无需利用指针。

输入过程中要判断并处理输入错误，直到输入一个合法且合理的值时结束输入。

输入完成之后，给起始柱上的盘号和盘子数量赋值。其中盘号按照顺序由大到小赋值（例如放5个盘子，则起始柱上各位置的盘号分别为5，4，3，2，1，0，0，0，0，0）。

3.3. 递归函数

递归函数（汉诺塔移动）的函数原型为void hanoi(unsigned num, char src, char tmp, char dst, int number_of_menu)。具体思路及实现方式如下（假设总共有n个盘子）：

<table>
<tr><td>步骤序号</td><td colspan="2">做法</td></tr>
<tr><td>1</td><td colspan="2">将前(n-1)个盘子由起始柱移动到中间柱。</td></tr>
<tr><td rowspan="6">2</td><td rowspan="6">将第n个盘子由起始柱移动到目标柱。具体步骤如右侧顺序所示。</td><td>起始柱盘子数减少1。</td></tr>
<tr><td>起始柱顶部盘子的编号赋给目标柱顶部盘子上面的位置。</td></tr>
<tr><td>目标柱盘子数增加1。</td></tr>
<tr><td>起始柱原先顶部盘子位置上的值变为0。</td></tr>
<tr><td>移动步数增加1。</td></tr>
<tr><td>依据条件调用相应的输出函数。</td></tr>
<tr><td>3</td><td colspan="2">将前(n-1)个盘子由中间柱移动到目标柱。</td></tr>
</table>

以上步骤的前提是n>0。也就是说，n为1时第1步和第3步实际上等于根本没有执行，这与实际情况相符。按照规定，此函数不超过15行。

3.4. 输出函数

输出函数的原型为void print_hanoi(unsigned num, char src, char tmp, char dst, int

number_of_menu)。前四个参数中，num、src、dst含义同前，tmp表示中间柱，number_of_menu则是菜单选项，其值与函数功能实现关系如下表所示。

菜单选项	输出的项目
1	仅输出移动方式（不调用函数）
2	仅输出步数和移动方式（不调用函数）
3	横向打印内部数组（调用横向输出数组函数）
4	横向、纵向分别打印内部数组，数字模拟汉诺塔移动（调用横、纵向输出数组函数）
7	输出图形汉诺塔（调用图形汉诺塔输出函数），但是只输出一步移动
8	同时输出上面提到的所有内容（调用横、纵向输出数组函数和图形汉诺塔输出函数）。二者仅在最上方输出的提示信息有区别。
9	

相关的具体输出函数将在后面提到。

3.5. 横向输出数组函数

此函数原型为void print_array(unsigned num, char src, char tmp, char dst, int number_of_menu)。各形参的含义见前述。

此处的number_of_menu参数是为了确定横向数组输出的位置。该值为3时，数组是就地输出，输出一次换一行；其他情况则是在一个固定的位置（此位置的X和Y坐标由宏定义表示）开始输出，每次不换行。

此函数实际执行时，将ABC三根柱子上的所有圆盘编号全部打印出来。如果相应的位置没有盘子（表现在数组中就是值为0），则输出与数字相同宽度的空格（如果该位置原本由数字，则新输出的空格会将其盖掉）。

此函数还会同时输出前述总输出函数中菜单选项为1和2时的所有项目。

3.6. 纵向输出数组函数

此函数原型为void print_visible_number_hanoi(char src, char tmp, char dst, int number_of_menu)。函数名是鉴于此函数在某种程度上算是实现了汉诺塔移动的“可视化”，尽管只是输出内部数组变化而非真实汉诺塔的盘子移动。各形参含义见前述。

number_of_menu参数的设置是因为菜单项4，8，9都要用到此函数但延时表现不同。8和9的延时（9的延时默认为3，见前述）在图形化输出中体现即可，在此函数中则可体现可不体现（我按照不体现处理了，节省时间），4的延时则必须在此处体现。

如果是初始状态（移动步数为0），则先输出底座，在其上由下到上按照类似横向数组输出的方式输出即可。如果不是初始状态，则用空格覆盖掉这一步被移动的盘子，并将其输出到目标柱顶部数字的上方。如果菜单选项为4，则在此处根据_speed的值停顿指定时间或者等待输入回车。注意根据要求，此函数不可以在每步移动输出完成之后都清屏并全部重新打印。

根据要求，菜单选项4和8要共用这一函数，且要用参数解决输出位置差异等问题。但由于我在本程序中并未区分4和8输出纵向数组的位置，所以也就没有此问题。此处用参数解决的是延时表现的差异。

3.7. 图形汉诺塔输出函数

函数原型为void print_visible_colour_hanoi(char src, char tmp, char dst)，各参数含义见前述。此函数本体非常简单，仅用了10行左右，但其中调用的函数却很不简单。详细内容如下。

3.7.1. 画柱子

一个无参的void型函数，作用是在某个位置（文件中是用宏定义表示的）画3根柱子（包括底座）。其中柱子宽度设为1，底座宽度自设（此处是设为21）。具体操作是在指定位置（该位置需要计算，因为要控制的是输出的中心位置，而此处的参数是起始位置）输出指定前、背景色的空格若干次，与下面的画盘子的方式类似。这一过程以及接下来的过程都要借助cct_showch函数（在指定起始X，Y位置用指定前、背色输入指定字符若干次，共6个参数）实现。

3.7.2. 画盘子

一个无参的void型函数，作用是根据全局二维数组的情况在柱子上画出盘子。

具体操作：以起始柱为A，共3个盘子的情况为例。

此时_col[0]的十个元素之值分别为3，2，1，0，0，0，0，0，0，0。先找到起始柱A在屏幕上的X坐标（为叙述方便，下文用x称呼）和柱子底座的Y坐标（在源文件里是用宏表示的），上移一行，准备开始输出盘子。

在该行X坐标为x-3的起始位置（3是盘子编号，这样选取起始位置是为了使盘子的中心位置的X坐标正好为x）用指定前、背景色输出3*2+1=7个空格（此处设盘子宽度为盘号*2+1），再上移一行后对编号为2的盘子做类似输出，如此进行下去，直至起始柱上当前位置没有盘子。注意新画的盘子会覆盖原本的柱子。

3.7.3. 移动盘子

此void型函数有两个参数，分别为本次移动的起始柱和目标柱。此处只简述向上和向右移动盘子的部分，向左和向下的移动逻辑与之类似，只需将坐标的增减等反过来即可。

注意由于是先进行内部数组的变化再进行输出表现，所以被移动的盘子应该是本次移动目标柱最顶上的盘子，但目前该盘子实际上是显示在起始柱的最顶上。

先定义两个变量，分别存储当前该盘子的X坐标和Y坐标-1的值（设为Y坐标-1是为了使一步向上移动结束时盘子位置坐标等于变量值）。为叙述方便，下文分别记作x和y。

向上移动时，先将盘子原来所在位置（Y坐标为y+1）用初始背景色覆盖（与前面输出盘子的原理类似），且如果被涂黑的位置并没有超过柱子的最顶端则需要重新将该位置的柱子补回去（此处所给的demo是添加了延时的，但考虑到此处不加延时仍然能表现动画效果且省时，所以我没加），然后在Y坐标为y的位置画一个和原来的盘子一模一样的盘子。这几步输出完成后，根据延时选项确定程序暂停时间。然后y值自减1（因为是向上），再对该盘子执行同样的过程，直到y的值超过预设的最高处为止（此处是为了叙述方便，实际程序中是当型循环，下同），此时盘子在屏幕上的显示高度就是预设的高度。整个纵向移动过程中，x的值不变。执行完成后再让y的值自增1，这样在上移结束后y的值就是盘子的Y坐标。

向右移动前，先使x的值自增1，目的与前面设y=Y-1相似。移动时，将盘子最左边的位置用初始背景色覆盖，再将该位置之后紧邻的一个位置作为初始位置画一个与原来一模一样的盘子，接着根据延时选项确定程序暂停时间（此处横向移动每步暂停时间设置为短于纵向为妙，因为此时实际在控制台中的字体宽度要小于高度，如果横纵向暂停时间相同，则实际表现效果是横向移动明显慢于纵向），然后将x的值自增1，如此进行，直到x的值超过预设终点处为止。此时盘子在屏幕上的位置与最终的位置正好对齐。整个横向移动过程中，y的值不变。执行完成后再让x的值自减1，这样在右移结束后x的值就是盘子的X坐标。

当一个盘子经过上移、平移、下移的步骤移动到目标位置之后，如果延时设定为0，则暂停程序直到手动按下回车键，然后执行接下来的操作；如果延时设定非0，则自动继续进行。在我写的整个程序中，延时设定为0时盘子的移速与延时设定为1时相同

3.8. 汉诺塔游戏

菜单选项9的汉诺塔游戏功能是利用一个函数实现的。该函数原型为void hanoigame(unsigned num, char src, char tmp, char dst)，各形参含义同前所述。

执行函数时，先调用一次总输出函数以输出初始状态，接下来进入一个while(1)循环。

进入循环之后，先输出提示语句提示命令输入形式，再清除掉下一行的命令（此程序的移动命令输入和提示语句输出不在同一行，与所给demo不同）。

然后定义两个变量start_col和end_col，储存输入的起始柱和目标柱编号。利用_getch()分别读取输入（与所给demo处理不同）。只有当读到的起始柱是ABCQ中的任何一个（此处不区分大小写，输入小写时转化成大写处理）时，结束读取。如果读到Q，则退出while(1)循环，否则继续等待输入移动目标柱。用类似的方式读入目标柱即可。

输入完成后，开始判断命令是否合法。非法命令有两种：一是起始柱为空，从空柱子上拿盘子显然有悖常理；二是大盘压小盘，这不符合游戏规则。前者只需判断_top数组相应元素的取值是否为0，后者则需判断目标柱是否为空以及比较两柱顶部盘子编号的大小。如果命令合法，则用与递归函数当中相同的方法更改内部数组和步数，然后调用总输出函数输出变化。

当总的目标柱（指在输入函数中输入的目标柱，而非每次移动的目标柱）上的盘子数等于总的盘子数时，游戏目标达成，游戏结束，输出提示信息后退出循环。

4. 调试过程碰到的问题

4.1. 有关全局变量的问题

写菜单选项2时试着运行了一下程序，发现在第一次输出基本移动步骤和计数时正常，但返回主菜单后在进行第二次同样操作时就会出现计数等不正常的问题。经检查才反应过来需要在返回主菜单前重置全局变量的值。本来是将这个过程放进主函数的，但在仔细阅读要求后发现全局变量应该定义为静态的，这意味着主函数没有权限访问这些变量。鉴于这种情况，我在全局变量所在的文件中写了个重置函数，由主函数调用，才算解决了这个问题。

原本定义的步数初值是1，且步数增加在输出之后，但写到菜单功能4时，想起输出函数可能还要输

出某些情况下的初始状态，此时需要进行特殊判定，于是将步数初值改为0，并对调了步数增加和输出的位置。

4.2. 清屏问题

在写完菜单选项2对应的功能并在Debug模式下运行时，我突然发现清屏函数cct_cls()似乎不能正常工作了。当时我打开了调试，发现那个函数可以正常进入，并且在某些情况下可以正常清屏，但另一些情况则不能清屏了，而且换用理论上应该与之功能相同的system("cls")时不会出现此问题。和同学交流了一晚上，仍然没能解决问题。第二天再次处理时，我试着调节了一下控制台的字体，然后调了回去，发现cct_cls()竟然可以正常清屏了。

当时出现这种状况的原因暂时仍不清楚，猜想与控制台有关。

5. 心得体会

本次作业中我学会了制作一个简单的菜单并根据菜单的输入值执行相应的操作，与此同时学到了一些简单的图形化处理函数的使用方法。在写汉诺塔内部数组的改变时，需要用到递归函数，这要求作者对递归函数的使用有一定的掌握；恰当的函数定义和宏定义有利于简化程序，使程序易懂。

部分功能的实现可能需要实现一些前置功能，因此将一个较大任务分解为多个较小任务的方式更容易给代码的作者提供思路。通常来说，每一个较小的任务都可以通过若干个函数实现，而大任务的完成实际就是根据条件调用这些实现小任务的函数，这样设计更易使代码简短，减少冗余。

6. 附件：源程序

```
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <Windows.h>
#include "hanoi.h"
#include "cmd_console_tools.h"

using namespace std;
static int _num_of_move = 0;
static int _col[3][10] = { 0 };
static int _top[3] = { 0,0,0 };
static unsigned _speed = 3;

void hanoi(unsigned num, char src, char tmp, char dst, int number_of_menu)
{
    if (num)
    {
        hanoi(num - 1, src, dst, tmp, number_of_menu);
        _col[dst - 'A'][_top[dst - 'A']++] = _col[src - 'A'][--_top[src - 'A']];
        _col[src - 'A'][_top[src - 'A']] = 0;
        _num_of_move++;
        if (number_of_menu != 7 || _num_of_move <= 1)
            print_hanoi(num, src, tmp, dst, number_of_menu);
        hanoi(num - 1, tmp, src, dst, number_of_menu);
```

```
    }
}

void print_visible_number_hanoi(char src, char tmp, char dst, int number_of_menu)
{
    int y, i, j;
    int x[4] = { A_X - (B_X - A_X),A_X,B_X,C_X };
    if (!_num_of_move)
    {
        cct_gotoxy(A_X - 1, NUM_COL_Y);
        printchar('=', C_X - A_X + 3);
        cct_gotoxy(A_X, NUM_COL_Y + 1);
        cout << "A";
        printchar(' ', B_X - A_X - 1);
        cout << "B";
        printchar(' ', C_X - B_X - 1);
        cout << "C";
        for (i = 0; i < 3; i++)
        {
            y = NUM_COL_Y - 1;
            for (j = 0; j < 10; j++, y--)
            {
                cct_gotoxy(x[i] + 1, y);
                cout << setw(x[i + 1] - x[i]);
                if (_col[i][j])
                    cout << _col[i][j];
                else
                    cout << ' ';
            }
        }
    }
    else
    {
        cct_gotoxy(x[src - 'A'] + 1, NUM_COL_Y - _top[src - 'A'] - 1);
        cout << setw(x[src - 'A' + 1] - x[src - 'A']) << ' ';
        cct_gotoxy(x[dst - 'A'] + 1, NUM_COL_Y - _top[dst - 'A']);
        cout << setw(x[dst - 'A' + 1] - x[dst - 'A']) << _col[dst - 'A'][_top[dst - 'A'] - 1];
    }

    if (number_of_menu == 4)
    {
        switch (_speed)
        {
            case 0:
                while (_getch() != '\r')
                    ;
                break;
            case 1:
                Sleep(600);
                break;
            case 2:
                Sleep(300);
                break;
            case 3:
                Sleep(150);
                break;
            case 4:
```

```
                Sleep(75);
                break;
            case 5:
                Sleep(25);
                break;
        }
    }
}

void hanoigame(unsigned num, char src, char tmp, char dst)
{
    print_hanoi(num, src, tmp, dst, 9);

    while (1)
    {
        cct_gotoxy(0, ARRAY_Y + 1);
        cout << "输入本次移动的起始柱和目标柱(如AC表示将A顶端的盘子移动到C,按Q退出): " << endl;
        cout << "  \r";

        char start_col, end_col;
        while (1)
        {
            start_col = _getch();
            if (start_col >= 'a' && start_col <= 'c' || start_col >= 'A' && start_col <= 'C' || start_col
== 'Q' || start_col == 'q')
            {
                cout << start_col;
                if (start_col >= 'a' && start_col <= 'c' || start_col == 'q')
                    start_col -= 32;
                break;
            }
        }
        if (start_col == 'Q')
        {
            cout << endl << "游戏中止! ";
            break;
        }
        while (1)
        {
            end_col = _getch();
            if (end_col >= 'a' && end_col <= 'c' && end_col - start_col != 32 || end_col >= 'A' && end_col
<= 'C' && end_col != start_col)
            {
                cout << end_col;
                if (end_col >= 'a' && end_col <= 'c')
                    end_col -= 32;
                break;
            }
        }

        if (_top[end_col - 'A'] && _col[start_col - 'A'][_top[start_col - 'A'] - 1] > _col[end_col -
'A'][_top[end_col - 'A'] - 1])
        {
            cout << endl << "非法移动(大盘压小盘)! ";
            Sleep(500);
            cout << '\r' << "                        ";
        }
```

```
        else if (!_top[start_col - 'A'])
        {
            cout << endl << "非法移动(起始柱为空)！";
            Sleep(500);
            cout << '\r' << "                        ";
        }
        else
        {
            _col[end_col - 'A'][_top[end_col - 'A']++] = _col[start_col - 'A'][--_top[start_col - 'A']];
            _col[start_col - 'A'][_top[start_col - 'A']] = 0;
            _num_of_move++;
            print_hanoi(_col[end_col - 'A'][_top[end_col - 'A'] - 1], start_col, 'A' + 'B' + 'C' -
start_col - end_col, end_col, 9);
        }

        if (_top[dst - 'A'] == num)
        {
            cct_gotoxy(0, ARRAY_Y + 2);
            cout << endl << "游戏结束！";
            break;
        }
    }
    cout << endl;
}

void print_move(char src, char dst)
{
    int y;
    int col_x[3] = { A_X,B_X,C_X };
    int x = col_x[src - 'A'];
    int colour_of_disc[10] = { COLOUR_OF_DISC0, COLOUR_OF_DISC1, COLOUR_OF_DISC2, COLOUR_OF_DISC3,
COLOUR_OF_DISC4,
                               COLOUR_OF_DISC5, COLOUR_OF_DISC6, COLOUR_OF_DISC7, COLOUR_OF_DISC8,
COLOUR_OF_DISC9 };

    /*将盘子拿起来*/
    for (y = COL_COL_Y - 1 - _top[src - 'A'] - 1; y >= COL_COL_Y - 13; y--)
    {
        cct_showch(x - _col[dst - 'A'][_top[dst - 'A'] - 1], y + 1, ' ', COLOUR_OF_BACKGROUND,
COLOUR_OF_BACKGROUND, _col[dst - 'A'][_top[dst - 'A'] - 1] * 2 + 1);
        if (y >= COL_COL_Y - 11)
            cct_showch(x, y + 1, ' ', COLOUR_OF_COLOMN, COLOUR_OF_COLOMN, 1);
        cct_showch(x - _col[dst - 'A'][_top[dst - 'A'] - 1], y, ' ', colour_of_disc[_col[dst -
'A'][_top[dst - 'A'] - 1] - 1], colour_of_disc[_col[dst - 'A'][_top[dst - 'A'] - 1] - 1], _col[dst -
'A'][_top[dst - 'A'] - 1] * 2 + 1);

        switch (_speed)
        {
            case 0:
            case 1:
                Sleep(200);
                break;
            case 2:
                Sleep(100);
                break;
            case 3:
                Sleep(50);
```

```
                break;
            case 4:
                Sleep(15);
                break;
            case 5:
                Sleep(3);
                break;
        }
    }
    y++;

    /*横向平移*/
    if (src < dst)
    {
        for (x++; x <= col_x[dst - 'A']; x++)
        {
            cct_showch(x - _col[dst - 'A'][_top[dst - 'A'] - 1] - 1, y, ' ', COLOUR_OF_BACKGROUND,
COLOUR_OF_BACKGROUND, 1);
            cct_showch(x - _col[dst - 'A'][_top[dst - 'A'] - 1], y, ' ', colour_of_disc[_col[dst -
'A'][_top[dst - 'A'] - 1] - 1], colour_of_disc[_col[dst - 'A'][_top[dst - 'A'] - 1] - 1], _col[dst -
'A'][_top[dst - 'A'] - 1] * 2 + 1);

            switch (_speed)
            {
                case 0:
                case 1:
                    Sleep(60);
                    break;
                case 2:
                    Sleep(30);
                    break;
                case 3:
                    Sleep(15);
                    break;
                case 4:
                    Sleep(5);
                    break;
                case 5:
                    static int i = 0;
                    i++;
                    if (i % 3 == 0)
                        Sleep(1);
                    break;
            }
        }
        x--;
    }
    else
    {
        for (x--; x >= col_x[dst - 'A']; x--)
        {
            cct_showch(x + _col[dst - 'A'][_top[dst - 'A'] - 1] + 1, y, ' ', COLOUR_OF_BACKGROUND,
COLOUR_OF_BACKGROUND, 1);
            cct_showch(x - _col[dst - 'A'][_top[dst - 'A'] - 1], y, ' ', colour_of_disc[_col[dst -
'A'][_top[dst - 'A'] - 1] - 1], colour_of_disc[_col[dst - 'A'][_top[dst - 'A'] - 1] - 1], _col[dst -
'A'][_top[dst - 'A'] - 1] * 2 + 1);
```

```
            switch (_speed)
            {
                case 0:
                case 1:
                    Sleep(60);
                    break;
                case 2:
                    Sleep(30);
                    break;
                case 3:
                    Sleep(15);
                    break;
                case 4:
                    Sleep(5);
                    break;
                case 5:
                    static int i = 0;
                    i++;
                    if (i % 3 == 0)
                        Sleep(1);
                    break;
            }
        }
        x++;
    }

    /*放下盘子*/
    for (y++; y <= COL_COL_Y - _top[dst - 'A']; y++)
    {
        cct_showch(col_x[dst - 'A'] - _col[dst - 'A'][_top[dst - 'A'] - 1], y - 1, ' ',
COLOUR_OF_BACKGROUND, COLOUR_OF_BACKGROUND, _col[dst - 'A'][_top[dst - 'A'] - 1] * 2 + 1);
        if (y >= COL_COL_Y - 9)
            cct_showch(col_x[dst - 'A'], y - 1, ' ', COLOUR_OF_COLOMN, COLOUR_OF_COLOMN, 1);
        cct_showch(col_x[dst - 'A'] - _col[dst - 'A'][_top[dst - 'A'] - 1], y, ' ', colour_of_disc[_col[dst
- 'A'][_top[dst - 'A'] - 1] - 1], colour_of_disc[_col[dst - 'A'][_top[dst - 'A'] - 1] - 1], _col[dst -
'A'][_top[dst - 'A'] - 1] * 2 + 1);

        switch (_speed)
        {
            case 0:
            case 1:
                Sleep(200);
                break;
            case 2:
                Sleep(100);
                break;
            case 3:
                Sleep(50);
                break;
            case 4:
                Sleep(15);
                break;
            case 5:
                Sleep(3);
                break;
        }
    }
```

```
    cct_setcolor(COLOUR_OF_BACKGROUND, COLOUR_OF_DEFAULT);
}
```