装 订 线

高级语言程序设计实验报告
——汉诺塔综合演示

计算机一班
1850059
杨志远
2018年12月10日

1 题目及基本要求

本程序为汉诺塔的综合演示，用多个子菜单项目选择需要使用的功能，主要功能包括：

（1）求出非图形化的汉诺塔（1~10 层）的解，包括显示步数记录、横向内部数组和竖式的功能，支持分步操作。

（2）显示伪图形化的 n 层汉诺塔的解，包括绘制三根柱子、n 个圆环和动画化演示移动步骤，支持分步操作。

（3）用户操作移动汉诺塔的圆环，直到将所有圆环移动到指定柱子上。

1.1 非图形化汉诺塔

非图形化汉诺塔分为四个子菜单项目

（1）仅显示基本的移动步骤（例如 1#:A->C）

（2）显示步数和移动步骤（例如第 1 步 1#:A->C）

（3）显示步数、移动步骤和横向的内部数组（例如第 1 步(1#: A-->C) A: 3 2 B: C: 1）

（4）显示步数、移动步骤、横向的内部数组和竖式，支持以多种延迟自动显示移动步骤、也支持分步显示移动步骤

1.2 伪图形化汉诺塔

伪图形化汉诺塔分为四个子菜单项目

（1）绘制三根柱子

（2）绘制三根柱子和 n 个圆环

（3）绘制三根柱子和 n 个圆环，用动画演示汉诺塔的第一步移动步骤

（4）用横向内部数组、竖式和图形化的动画、演示汉诺塔的完整移动步骤，支持以多种延迟自动显示移动步骤、也支持分步显示移动步骤

1.3 游戏版汉诺塔

游戏版汉诺塔支持用户输入指令，将某个柱子上的圆环移动到另一个圆环上，直到所有圆环都在指定的柱子上，此时游戏结束

2 整体设计思路

这个程序的总体逻辑为：

准备工作（调整窗口大小、输出帮助信息） -> 选择模式① -> 如果选择退出，则退出程序，否则继续执行程序 -> 输入信息-> 初始化（包括内部数组初始化、再次调整窗口大小、清屏、绘制柱子和绘制初始圆环等，根据模式进行操作）

自动模式：递归执行汉诺塔程序② -> 移动内部数组 -> 输出移动步骤和内部数组（如果需要的话） -> 根据延迟输出竖式（如果需要的话） -> 根据延迟以动画演示当前移动步骤 -> 如果程序未完成，则回到②，否则回到主菜单①并等待选择模式

游戏模式：输入指令 -> 如果输入了退出指令，则回到主菜单①并等待选择模式，否则继续执行程序 -> 判断输入的指令知否合法（输入非法、起点为空、大环压小环等），如果合法则执行指令，移动圆环，如果非法则重新输入指令 -> 判断所有圆盘是否移动到指定柱子上，如果是则回到主菜单①并等待选择模式，否则重新输入指令

绝大部分的函数都会根据选择的模式调整需要进行的操作。

3 主要功能的实现

本程序的核心函数之一为汉诺塔的操作函数 Hanoi。该函数至少需要三个参数，包括当前操作的汉诺塔层数和当前操作的起始柱和终点柱。该函数以递归的思想，逻辑为：

（1） 如果当前层数为 1，则将起始柱上的圆环移动到终点柱并输出信息

（2） 如果当前层数不为 1，则先将当前圆环上所有圆环从起始柱移动到中间柱（调整参数，层数-1， 终点柱设为中间柱，递归调用该函数），再将当前圆环从起始柱移动到终点柱，最后将当前圆环上所有圆环从中间柱移动到终点柱（调整参数，层数-1，起点柱设为中间柱，递归调用该函数）。

汉诺塔函数中使用了另外一个函数，名为 move_stack_and_print_all_information，负责移动圆环和输出所有的信息。该函数会根据选择模式，执行部分或全部操作，每个操作用不同函数进行。执行的步骤为：

（1） 移动内部数组，代码为 stack[end - 'A'][top[end - 'A']++] = stack[start - 'A'][--top[start - 'A']];

（2） 横向输出当前移动步骤和内部数组

（3） 纵向输出竖式

（4） 动画演示当前移动步骤

输出竖式的函数 print_array_vertical 的实现步骤为：

（1） 隐藏光标

（2） 循环执行移动光标并输出内部数组中某个柱子上的数字，直到全部输出完成

（3） 将其之上的位置上的数字用空格擦除

这样执行三次，就能将三个柱子上的数字全部输出。最后根据设置的延迟模式调用 Sleep 函数并传入指定的延迟大小。

动画演示移动圆环的函数 move_ring 的大致实现步骤为：

（1） 隐藏光标

（2） 向上移动圆环

（3） 横向移动圆环

（4） 向下移动圆环

（5） 在最后的位置绘制圆环
（6） 将前景色和背景色改为默认值

向上或向下移动圆环的具体步骤为：
（1） 用带背景色的空格绘制圆环
（2） 调用 Sleep 函数
（3） 用空格擦除当前位置的圆环
（4） 如果当前位置在柱子上，则用一格带背景色的空格补齐柱子上缺失的那一格

横向移动圆环的步骤为与向上或向下移动圆环类似，但不需要补齐缺失的柱子。

游戏模式 game_mode 的实现步骤为：
（1） 进行准备工作（初始化，输出竖式）
（2） 循环执行以下步骤，直到收到退出的指令或判断到所有圆盘已经移动到指定柱子上：
  （1） 输入指令
  （2） 判断该指令是否合法（是否出现输入非法、起点为空、大环压小环的情况），如果合法则执行指令，调用 move_stack_and_print_all_information 来移动圆环并输出所有的信息，如果非法则重新输入指令

输入指令的操作为：
（1） 如果输入数据的函数 input_information 收到某个参数，则专门执行指令的输入：
  （1） 循环调用_getch 函数数次，将读入的字符写入一个字符串，如果该字符不是退格或回车，则将该字符输出，如果输入了回车，则退出循环
  （2） 将字符串中第一个和第二个字符从小写转换为大写
  （3） 判断整个字符串的长度是否符合要求、第一个字符是否为’Q’，如果是，则返回 0,如果不是：判断判断整个字符串的长度是否符合要求、第一个、第二字符是否为’A’或‘B’或‘C’并且两者不相等，如果是，则返回某个特定的值，代码为 return (str[0] - ’A’) * 10 + str[1] - ’A’;用两位数的十位和个位分别代表起始柱和终点柱，如果不是，则返回-1

根据选择的模式进行总操作的函数 choose_mode 的实现步骤为：
（1） 如果 mode 为 0，则退出程序
（2） 如果 mode 不为 5，则输入信息
（3） 如果 mode 为 4，则清屏
（4） 如果 mode 大于 4 且小于 9，则重新设置窗口大小并隐藏光标
（5） 如果 mode 大于 5 且小于 9，则绘制柱子
（6） 如果 mode 为 4 或 8 或 9，则将光标设置在（0，0）点
（7） 如果 mode 为 1 或 2 或 3 或 4 或 8 或 9，则执行初始化函数 initialize
（8） 如果 mode 大于 6 且小于 9，则绘制初始圆环
（9） 如果 mode 为 7，则调用 move_ring 函数，执行第一步移动
（10） 如果 mode 大于 1 且小于 4 或为 8，则执行汉诺塔函数 Hanoi
（11） 如果 mode 为 9，则执行游戏模式函数 game_mode

（12） 根据不同的模式，在适当的位置输出"按回车继续"的提示信息

4 调试过程碰到的问题

在编写移动圆环的函数的过程中，我曾经尝试使用圆环最左端的 x、y 坐标作为参数传入函数，但这样的坐标很不利于传入实参调用函数，极大地增加了编写程序的难度，也降低了代码的可读性。最大的问题是，如果简单粗暴地将当前位置已经绘制了的圆环擦除，那个已经绘制完成的柱子也会被一起擦除；而如果在此时调用绘制柱子的函数，那么其他还在柱子上的圆环会被截成两部分，不仅增加了屏幕的闪烁，而且几乎不可能再次将那些圆环恢复原状。

最终，我对这个函数进行了两个修改：首先，我将函数的形式参数中的 x 坐标的意义从圆环的最左端改为了圆环的最右端,这样就可以保证传入实参时只需要使用柱子的中心 x 坐标;其次，我在擦除已经绘制完成的圆环后，根据此时的 y 坐标，在柱子的中心位置输出一个与柱子颜色相同的空格，这样就可以有效防止柱子被擦除。

装

另外，在自动移动圆环的函数的过程中，还出现了其他的问题。移动圆环时，画面出现明显的错误，不仅圆环的起点和终点错误，在分步操作的时候还出现了不同步的现象，圆环移动滞后一步于竖式的移动。

订

在深入分析程序之后，我发现原因在于执行移动圆环的函数 move_ring 时，传入的参数出现了错误。在重新计算了起点和终点的 x、y 坐标后，我成功地解决了显示的错误。之后，我再调整了等待读入回车的代码的位置，成功地保持了圆环移动和竖式移动的同步。

线

在编写游戏模式输入指令的代码时，如果直接使用 cin 读入字符串，因为无法判断用户输入了多少字符，所以当用户输入了足够长的字符串时，会导致控制台中的输出信息被向下拖动、所有基于 gotoxy 函数的输出全部都会错位。一开始，我尝试使用循环和_getche 函数逐个读入字符并存入字符串来解决这个问题，但如果用户输入大量的退格，会将该行全部擦除，如果用户输入大量回车，同样会出现错位现象。

我的解决方法是该用_getche 并输出那些可以被输出的字符，如果输入了回车则跳过循环。这样实现的一个要点是需要根据上一个输入的字符判断是否需要输出当前字符，因为如果键盘按下了方向键或 F1~F12，会转化为两个字符，其中一个的 ASCII 码对应了可输出的字符，也就是说如果不对其进行处理，会输出一个字符。相关的代码如下：

```
char str[20] = "*";  //任意单个字符皆可
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)：";
for (int i = 1; i <= 15; i++)  //最多输入 15 个字符
{
 str[i] = _getch(); //读入按键
 if (str[i] == '\r')  //如果按下了回车，则退出循环
     break;
 if(str[i] >= 33 && str[i] <= 126 && str[i - 1] >= 33 && str[i - 1] <= 126) //判断是否为可输出字符
     cout << str[i];
```

```
}
```

同时，需要注意的一点是，_getch 会将回车中的’\r’也读入字符串，也就是说，键盘输入”AB”时，字符串的长度为 4，这在之后的处理中非常重要。

5 心得体会

5.1 心得体会和经验教训

我在逐渐实现一个个模式的过程中，没有为将来的模式做好充分的准备，这就导致了在之后实现其他模式时，不得不多次修改之前的程序来保证程序不出错。这样频繁的修改也导致了程序代码显得啰嗦，还有一些重复的无效操作和低效率的实现方法使得代码的可读性下降。事实上，我撰写实验报告的大部分时间，都在尽力修改和完善源程序，现在的程序代码与我开始写报告时的代码有很多差别。今后我在编写有一定规模的程序时，应当尽量考虑到未来的扩展性，减少因缺乏远见而导致的重复劳动。

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？

分为若干个小题的形式更好。如果直接以完成一道大题的思路去编写程序，不仅会加大编程的难度，还会导致很多小缺陷甚至bug会被忽视。当程序规模越来越大时，这些问题会越来越难发现，给程序带来巨大的隐患。而如果分为若干小题，就能更容易发现问题并及时解决。

5.3 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？

在编写程序的过程中，我在绝大部分函数中利用传入的参数表示当前的模式，并以此运用if语句调整需要执行的代码。以这样的方法，我在实现之后的小题时，尽可能地重用了之前的代码，减少代码量。尽管我在编程的过程中，尽可能地考虑了未来小题的实现方法，但是还有一些没有考虑到的情况，这就导致了我不得不多次修改原来的程序来适应目前的要求。

5.4 如何才能更好地利用函数来编写复杂的程序

（1）对函数传入参数，可以指定需要执行函数的哪些语句，只要语句的先后顺序合理，就能节省大量重复的代码。

（2）在一个功能强大的函数中，可以适当地穿插一些规模小的函数，既可以用来代替一些高度重复的代码，也可以用来增加代码的可读性。

（3）适当地运用函数划分一个复杂的问题，可以使程序模块化，增加程序的可读性，大幅度降低维护的难度。

6 源代码

```
//1850059 计1班 杨志远
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <Windows.h>
#include <conio.h>
#include "cmd_console_tools.h"
using namespace std;

const int maxLayer = 10;
const int moveDown = 10;
const int moveDownGraphics = 5;
const int windowX = 100;
const int windowY = 30;
const int SPEED1 = 1000;
const int SPEED2 = 250;
const int SPEED3 = 70;
const int SPEED4 = 30;
const int SPEED5 = 0;
int stack[3][maxLayer];
int top[5];
int steps;

int Speed(int speed) //根据输入的模式输出延迟
{
    if (speed == 1)
        return SPEED1;
    else if (speed == 2)
        return SPEED2;
    else if (speed == 3)
        return SPEED3;
    else if (speed == 4)
        return SPEED4;
    else if (speed == 5)
        return SPEED5;
    else
        return -1;
}

//输出栈内信息
void print_stack(int layer)
{
    cout << " A:";
    for (int i = 0; i < top[0]; i++)
        cout << setw(2) << stack[0][i];
    for (int i = top[0]; i < 10; i++)
        cout << "  ";
    cout << " B:";
    for (int i = 0; i < top[1]; i++)
        cout << setw(2) << stack[1][i];
    for (int i = top[1]; i < 10; i++)
        cout << "  ";
    cout << " C:";
    for (int i = 0; i < top[2]; i++)
        cout << setw(2) << stack[2][i];
    for (int i = top[2]; i < 10; i++)
        cout << "  ";
    cout << endl;
}

void print_array_vertical(int mode, int speed)
//输出竖式
{
    for (int i = 0; i < top[0]; i++)
    {
        gotoxy(10, maxLayer + ((mode == 8 ||
mode == 9) ? moveDown + moveDownGraphics: 0) -
i + 1);
        cout << setw(2) << stack[0][i];
    }
    gotoxy(10, maxLayer + ((mode == 8 || mode
== 9) ? moveDown + moveDownGraphics: 0) -
(top[0]) + 1);
    cout << "  ";
    for (int i = 0; i < top[1]; i++)
    {
        gotoxy(20, maxLayer + ((mode == 8 ||
mode == 9) ? moveDown + moveDownGraphics: 0) -
i + 1);
        cout << setw(2) << stack[1][i];
    }
    gotoxy(20, maxLayer + ((mode == 8 || mode
== 9) ? moveDown + moveDownGraphics: 0) -
(top[1]) + 1);
    cout << "  ";
    for (int i = 0; i < top[2]; i++)
    {
        gotoxy(30, maxLayer + ((mode == 8 ||
mode == 9) ? moveDown + moveDownGraphics: 0) -
i + 1);
        cout << setw(2) << stack[2][i];
    }
    gotoxy(30, maxLayer + ((mode == 8 || mode
== 9) ? moveDown + moveDownGraphics: 0) -
(top[2]) + 1);
    cout << "  ";
    if (Speed(speed) == -1 && (mode == 4))
        while (_getch() != '\r')
            ;
}

void move_ring_horizonal(int x, int i, int
color, int length, int speed) //（图形化）竖
直移动环的一步
{
    showch(x - length / 2, i, ' ', color,
COLOR_WHITE, length);
    Sleep(Speed(speed) >= 0 ? Speed(speed) :
0);
```

```
    showch(x - length / 2, i, ' ', COLOR_BLACK,
COLOR_WHITE, length);
    if (i >= moveDownGraphics - 2)
        showch(x, i, ' ', COLOR_HYELLOW,
COLOR_WHITE, 1);
}

void move_ring(int x1, int x2, int y1, int y2,
int color, int length, int speed) //（图形化）
移动环
{
    setcursor(CURSOR_INVISIBLE);
    for (int i = y1; i != moveDownGraphics - 4;
i--)
        move_ring_horizonal(x1, i, color,
length, speed);
    for (int i = x1 - length / 2; i != x2 - length
/ 2; x1 < x2 ? i++ : i--)
    {
        showch(i, moveDownGraphics - 4, ' ',
color, COLOR_WHITE, length);
        Sleep(Speed(speed) >= 0 ?
Speed(speed) : 0);
        showch(i, moveDownGraphics - 4, ' ',
COLOR_BLACK, COLOR_WHITE, length);
    }
    for (int i = moveDownGraphics - 4; i != y2;
i++)
        move_ring_horizonal(x2, i, color,
length, speed);
    showch(x2 - length / 2, y2, ' ', color,
COLOR_WHITE, length);
    showch(0, windowY - 3, ' ', COLOR_BLACK,
COLOR_WHITE, 1);
    if (Speed(speed) == -1)
        while (_getch() != '\r')
            ;
}

void print_Hanoi_one_line(int mode, int layer,
char start, char end, int steps, int speed) //
横向输出汉诺塔移动信息
{
    if (mode == 4)
    {
        if (Speed(speed) != -1)
            Sleep(Speed(speed));
    }
    if (mode == 4 || mode == 8 || mode == 9)
        gotoxy(0, maxLayer + ((mode == 8 ||
mode == 9) ? moveDownGraphics + moveDown : 0)
+ 5);
    if (mode == 2 || mode == 3 || mode == 4 ||
mode == 8 || mode == 9)
        cout << "第" << setw(4) << steps << "
步(" << setw(2) << layer << "#: " << start <<
"-->" << end << ")";
    if (mode == 1)
        cout << layer << "#: " << start <<
"-->" << end;
    if (mode == 1 || mode == 2)
        cout << endl;
    if (mode == 3 || mode == 4 || mode == 8 ||
mode == 9)
        print_stack(layer);
}

void move_stack_and_print_all_information(int
mode, int layer, char start, char end, int steps,
int speed) //移动和输出总函数
{
    stack[end - 'A'][top[end - 'A']++] =
stack[start - 'A'][--top[start - 'A']];
    print_Hanoi_one_line(mode, layer, start,
end, steps, speed);
    if (mode == 4 || mode == 8 || mode == 9)
        print_array_vertical(mode, speed);
    if (mode == 8 || mode == 9)
        move_ring(12 + (start - 'A') * 32, 12
+ (end - 'A') * 32, maxLayer + moveDownGraphics
- top[start - 'A'] - 1, maxLayer +
moveDownGraphics - top[end - 'A'], layer, layer
* 2 + 1, speed);
}

void Hanoi(int mode, int layer, char start, char
end, int speed) //汉诺塔核心函数
{
    if (layer == 1)
    {

    move_stack_and_print_all_information(mod
e, layer, start, end, ++steps, speed);
        return;
    }
    char middle = 'A' + 'B' + 'C' - start - end;
    Hanoi(mode, layer - 1, start, middle,
speed);
    move_stack_and_print_all_information(mod
e, layer, start, end, ++steps, speed);
    Hanoi(mode, layer - 1, middle, end, speed);
}

void welcome_information()//输出主界面的提示
{
    cout <<
"--------------------------------" << endl;
    cout << "1.基本解" << endl;
    cout << "2.基本解(步数记录)" << endl;
    cout << "3.内部数组显示(横向)" << endl;
    cout << "4.内部数组显示(纵向 + 横向)" <<
endl;
    cout << "5.图形解 - 预备 - 画三个圆柱" <<
endl;
```

```
    cout << "6.图形解 - 预备 - 在起始柱上画 n 个盘子" << endl;
    cout << "7.图形解 - 预备 - 第一次移动" << endl;
    cout << "8.图形解 - 自动移动版本" << endl;
    cout << "9.图形解 - 游戏版" << endl;
    cout << "0.退出" << endl;
    cout << "----------------------------------" << endl;
    cout << "[请选择:]";
}

int select_mode()//选择模式
{
    while (1)
    {
        char ch = _getch();
        if (ch >= '0' && ch <= '9')
            return ch - '1' + 1;
    }
}

inline void clearCin()//清除输入缓冲区
{
    cin.clear();
    cin.ignore(999999999, '\n');
}

int input_information(int mode, int *layer, char *start, char *end, int *speed) //输入数据总函数
{
    while (mode != 10)
    {
        cout << "请输入汉诺塔的层数(1-" << maxLayer << "): " << endl;
        cin >> *layer;
        clearCin();
        if (*layer <= 0 || *layer > maxLayer)
            continue;
        break;
    }
    while (mode != 10)
    {
        cout << "请输入起始柱(A-C): " << endl;
        cin >> *start;
        clearCin();
        (*start) = ((*start) >='a' && (*start) <= 'c') ? (*start) - ('a' - 'A') : (*start);
        if (!(((*start) >='A' && (*start) <= 'C')))
            continue;
        break;
    }
    while (mode != 10)
    {
        cout << "请输入目标柱(A-C): " << endl;
        cin >> *end;
        clearCin();
        (*end) = (*end >='a' && *end <='c') ? (*end) - ('a' - 'A') : (*end);
        if (!(((*end) >= 'A' && (*end) <= 'C')))
            continue;
        if (*start == *end)
        {
            cout << "目标柱(" << (*end) << ")不能与起始柱(" << (*start) << ")相同" << endl;
            continue;
        }
        break;
    }
    while (mode == 4 || mode == 8)
    {
        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;
        if (!(cin >> *speed))
        {
            clearCin();
            continue;
        }
        clearCin();
        if (*speed < 0 || *speed > 5)
            continue;
        break;
    }
    while (mode == 10)
    {
        char str[20] = "*";  //任意单个字符皆可
        gotoxy(0, maxLayer + moveDown + moveDownGraphics + 6);
        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) ：";
        for (int i = 1; i <= 15; i++)  //最多输入 15 个字符
        {
            str[i] = _getch(); //读入按键
            if (str[i] == '\r')  //如果按下了回车，则退出循环
                break;
            if (str[i] >= 33 && str[i] <= 126 && str[i - 1] >= 33 && str[i - 1] <= 126) //判断是否为可输出字符
                cout << str[i];
        }
        str[1] >='a' && str[1] <='z' ? str[1] = str[1] - 'a' + 'A' : 0 ;
        str[2] >='a' && str[1] <='z' ? str[2] = str[2] - 'a' + 'A' : 0 ;
        gotoxy(0, maxLayer + moveDown + moveDownGraphics + 6);
```

装 订 线

```
        for (int i = 0; i < windowX; i++)
            cout << "  ";
        if (strlen(str) == 3 && str[1] == 'Q')
            return 0;
        if(strlen(str) == 4 && str[1] >= 'A'
&& str[1] <= 'C' && str[2] >= 'A' && str[2] <=
'C' && str[1] != str[2])
            return (str[1] - 'A') * 10 + str[2]
- 'A';
    }
    return -1;
}

void initialize(int mode, int layer, char start,
char end, int speed) //初始化
{
    if (mode == 3 || mode == 4 || mode == 8 ||
mode == 9)
    {
        top[0] = top[1] = top[2] = 0;
        for (int i = 0; i < layer; i++)
            stack[start - 'A'][i] = layer -
i;
        top[start - 'A'] = layer;
    }
    if (mode == 4 || mode == 8 || mode == 9)
    {
        cout << " 从 " << start << " 移动到 "
<< end << ", 共 " << layer << " 层, 延时设置
为 " << speed << endl;
        gotoxy(9, maxLayer + ((mode == 8 ||
mode == 9) ? moveDownGraphics + moveDown : 0)
+ 2);
        cout << "=======================";
        gotoxy(9, maxLayer + ((mode == 8 ||
mode == 9) ? moveDownGraphics + moveDown : 0)
+ 3);
        cout << "  A         B         C";
        gotoxy(0, maxLayer + ((mode == 8 ||
mode == 9) ? moveDownGraphics + moveDown : 0)
+ 5);
        cout << "初始:";
        print_stack(layer);
        Sleep(200);
        if (Speed(speed) == -1 && (mode == 4))
            while (_getch() != '\r')
                ;
        gotoxy(0, maxLayer + ((mode == 8 ||
mode == 9) ? moveDownGraphics + moveDown : 0)
+ 5);
    }
    steps = 0;
    cout << endl;
}

void draw_columns()//（图形化）绘制柱子
{
    showch(1 , maxLayer + moveDownGraphics, '
', COLOR_HYELLOW, COLOR_WHITE, 23);
    showch(33, maxLayer + moveDownGraphics, '
', COLOR_HYELLOW, COLOR_WHITE, 23);
    showch(65, maxLayer + moveDownGraphics, '
', COLOR_HYELLOW, COLOR_WHITE, 23);
    Sleep(Speed(3));
    for (int i = maxLayer + moveDownGraphics;
i >= moveDownGraphics - 2; i--)
    {
        showch(12, i, ' ', COLOR_HYELLOW,
COLOR_WHITE, 1);
        showch(44, i, ' ', COLOR_HYELLOW,
COLOR_WHITE, 1);
        showch(76, i, ' ', COLOR_HYELLOW,
COLOR_WHITE, 1);
        Sleep(Speed(3));
    }
    showch(0, windowY - 3, ' ', COLOR_BLACK,
COLOR_WHITE, 1);
    Sleep(Speed(2));
}

void draw_ring(int layer, char start) //（图
形化）绘制环
{
    for (int i = layer; i > 0; i--)
    {
        showch( - i + 12 + (start - 'A') * 32,
maxLayer + moveDownGraphics + i - layer - 1,
            ' ', i, COLOR_WHITE, i * 2 + 1);
        Sleep(Speed(3));
    }
    showch(0, windowY - 3, ' ', COLOR_BLACK,
COLOR_WHITE, 1);
    Sleep(Speed(2));
}

bool game_over(int layer, int end) //判断游戏
结束
{
    return (top[end - 'A'] == layer);
}

void game_mode(int layer, char start, char end)
//游戏模式的控制函数
{
    print_array_vertical(8, 1);
    while (1)
    {
        int move = input_information(10, 0, 0,
0, 0);
        char playerStart = 'A' + move / 10;
        char playerEnd = 'A' + move % 10;
        gotoxy(0, maxLayer + moveDown +
moveDownGraphics + 6);
        if (top[playerStart - 'A'] == 0)
```

```
        {
            cout << endl << "输入错误：起点柱为空      " << endl;
            Sleep(750);
            gotoxy(0, maxLayer + moveDown + moveDownGraphics + 6);
            cout << endl << "                                                  " << endl;
            continue;
        }
        if (top[playerEnd - 'A'] > 0 && stack[playerEnd - 'A'][top[playerEnd - 'A'] - 1] > 0 && stack[playerStart - 'A'][top[playerStart - 'A'] - 1] > stack[playerEnd - 'A'][top[playerEnd - 'A'] - 1])
        {
            cout << endl << "输入错误：大环不能在小环上" << endl;
            Sleep(750);
            gotoxy(0, maxLayer + moveDown + moveDownGraphics + 6);
            cout << endl << "                                                  " << endl;
            continue;
        }

    move_stack_and_print_all_information(8, stack[playerStart - 'A'][top[playerStart - 'A'] - 1], playerStart, playerEnd, ++steps, 5);
        if (move == 0 || game_over(layer, end))
        {
            gotoxy(0, windowY + 5 - 3);
            cout << "游戏终止!!!                                                  " << endl;
            return;
        }
    }
}
void choose_mode(int mode) //根据选择的模式进行操作
{
    int layer, speed = 1;
    char start, end;
    if (mode == 0)
        exit(0);
    if (mode != 5)
        input_information(mode, &layer, &start, &end, &speed);
    if (mode == 4)
        cls();
    if (mode >= 4 && mode <= 9)
    {
        setcursor(CURSOR_INVISIBLE);
        setconsoleborder(windowX, (mode != 8 && mode != 9) ? windowY : windowY + 5);
    }
    if(mode >= 5 && mode <= 9)
        draw_columns();
    if (mode == 4 || mode == 8 || mode == 9)
        gotoxy(0, 0);
    if (mode >= 1 && mode <= 4 || mode == 8 || mode == 9)
        initialize(mode, layer, start, end, (mode != 9) ? speed : 5);
    if (mode >= 6 && mode <= 9)
        draw_ring(layer, start);
    if (mode == 7)
    {
        char middle = 'A' + 'B' + 'C' - start - end;
        move_ring(12 + (start - 'A') * 32, 12 + ((layer % 2 ? end : middle) - 'A') * 32, maxLayer + moveDownGraphics - layer, maxLayer + moveDownGraphics - 1, COLOR_BLUE, 3, 4);
    }
    if (mode >= 1 && mode <= 4 || mode == 8)
    {
        if (mode == 8 && Speed(speed) == -1)
            while (_getch() != '\r')
                ;
        Hanoi(mode, layer, start, end, speed);
    }
    if (mode == 9)
        game_mode(layer, start, end);
    if(mode == 4 || mode == 5 || mode == 8 || mode == 9)
        gotoxy(0, ((mode != 8 && mode != 9) ? windowY : windowY + 5) - 3);
    cout << endl << "按回车继续" << endl;
}

int main()
{
    while (1)
    {
        setconsoleborder(windowX, windowY, windowX, 5000);
        setcursor(CURSOR_VISIBLE_NORMAL);
        welcome_information();
        int mode = select_mode();
        cout << mode << endl << endl;
        Sleep(200);
        choose_mode(mode);
        while (_getch() != '\r')
            ;
    }
    return 0;
}
```