完成日期 2024.5.15

汉诺塔综合演示实验报告

姓名_____陈应波_____

班级 _____信 11_____

学号_____2352219____

1. 汉诺塔综合演示实验报告

汉诺塔问题（Hanoi Tower Puzzle），又称河内塔问题，据说起源于印度的一个传说，传说中有一个寺庙，里面有 64 个金盘，僧侣们的任务是将这些盘子从一个柱子移动到另一个柱子，当他们完成这个任务时，世界将会终结。是一个经典的数学游戏和算法问题。汉诺塔是由三根柱子和一堆大小不一的圆盘组成。初始时，所有圆盘按大小顺序（从大到小）叠放在起始柱上，目标是将所有圆盘移动到另一根柱上，且在整个移动过程中，始终遵循以下规则：

1. 每次只能移动一个圆盘。
2. 任何时候，在三根柱子的任何一根上，较大的圆盘不能放在较小的圆盘上面。

汉诺塔问题是一个经典的递归问题，它对于编程和算法设计具有重要的教育意义。通过解决这个问题，可以培养程序员的递归思维能力，理解如何将大问题分解为小问题，然后逐步解决的分而治之的思想。同时不同的算法和解决方案有自己的独特之处以及消耗的时间和内存不同，效率不同可以帮助程序员理解算法的时间复杂度。递归解决方案的时间复杂度是指数级的，这有助于程序员认识到不同算法效率的差异。

总之，汉诺塔问题是一个典型的程序设计例题。研究汉诺塔功能的实现和综合演示，可以提高我们的计算机思维和编程能力。

2. 整体设计思路

整个汉诺塔综合演示一共需要实现九个功能。用户通过菜单选项选择需要演示的功能，然后调用对应的函数，演示对应的功能，然后返回主菜单。整个程序的思路我们先是区分为几大板块功能区。分别为输入模块，菜单模块，功能实现模块。首先进入用户菜单，调用输入模块，接受用户的输入选项，然后根据用户的选项选择调用对应的功能函数，或者是退出程序。然后在功能模块实现相应的功能。这样就能实现不同函数之间的解耦，防止整个程序因为一个功能函数的错误而整个程序都不能正常使用。这样做就可以逐个实现对应功能，并且在开发过程中能实时调试，提高开发效率。

3. 主要功能的实现

主要部分的流程图如下所示。

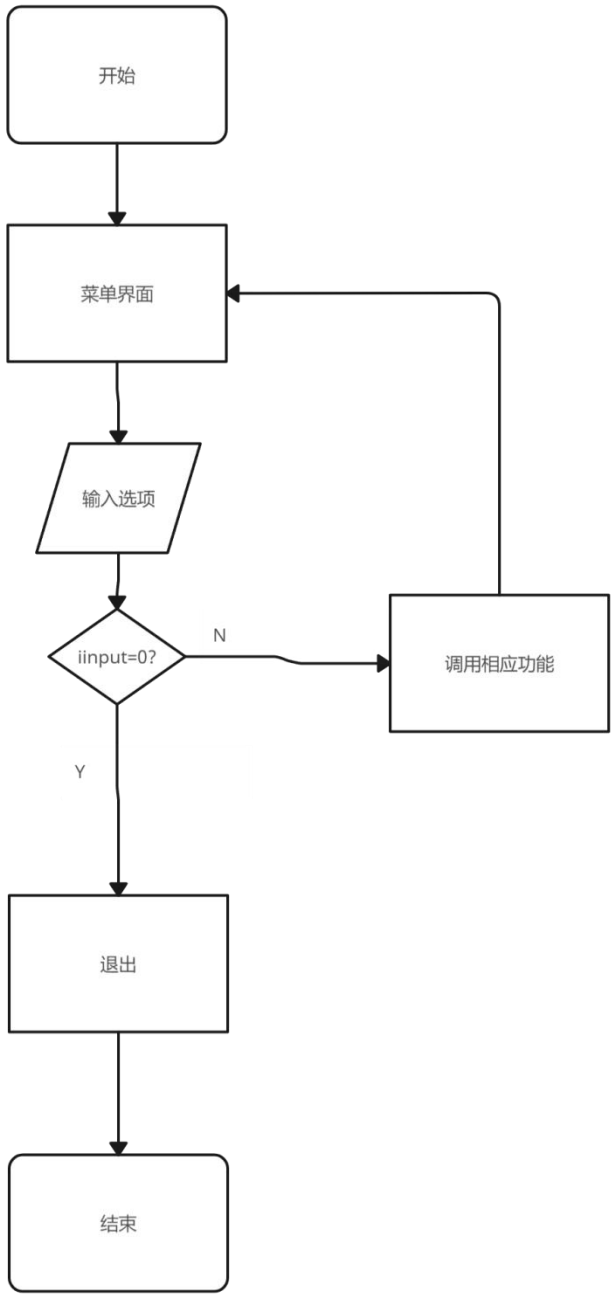
1

2

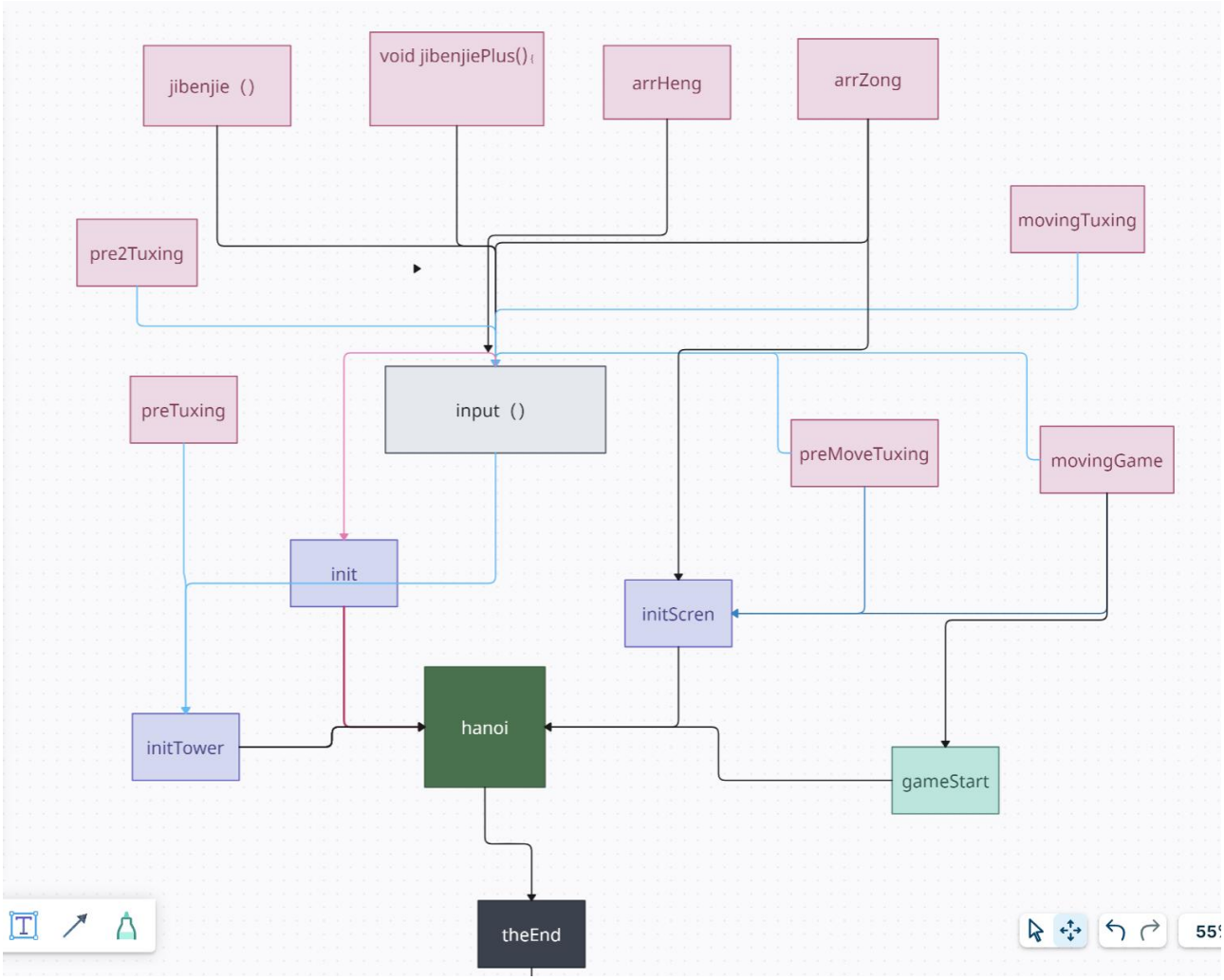
3

4

5



功能模块的实现关系图如下

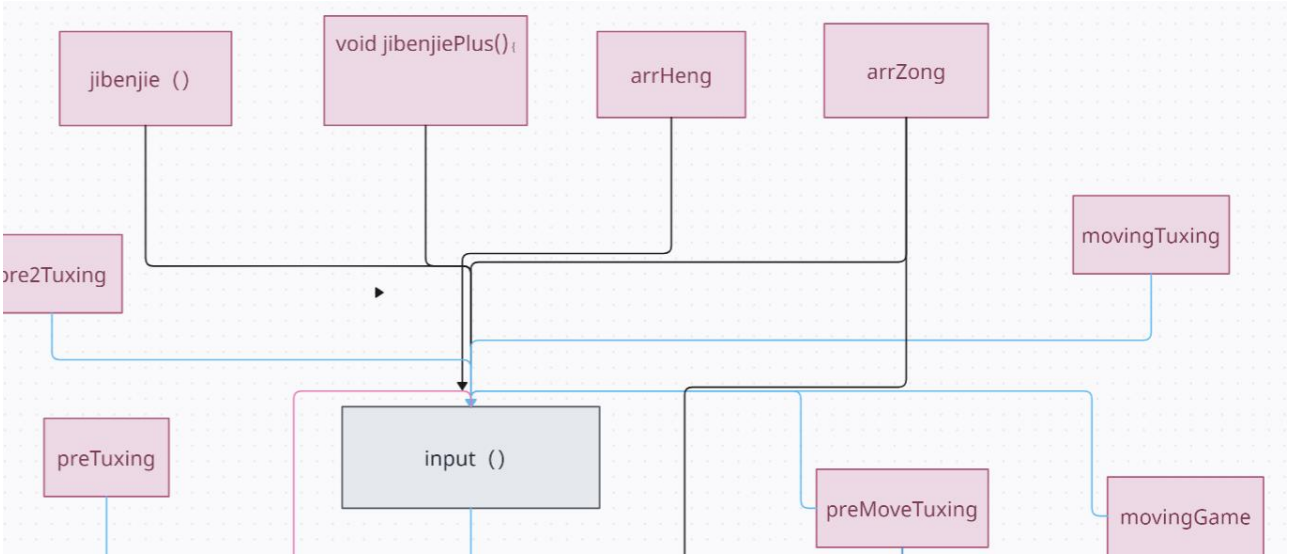


4. 调试过程碰到的问题

在调试过程中遇到了如下的问题：

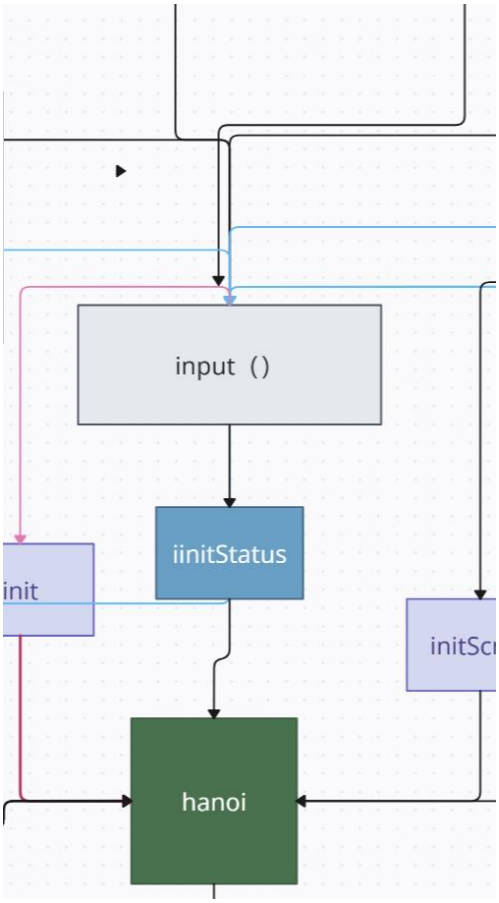
1. 因为使用了全局变量记录汉诺塔移动的总步数，以及汉诺塔内部数组的状态和指针的当前位置。所以在运行了一次游戏之后，再次返回菜单进行二次游戏的时候，上次残留的数据还继续保持在程序中，导致逻辑出现问题。

解决方法如下：



如图，每一个功能的实现首先都是通过input函数进行对初始变量的赋值，这其中就包括了上一次使用的全局变量，所以我在input函数中调用初始化函数initState（），对全局变量进行初始化，解决了问题。

如图：



2. 因为遇到了不同功能函数为了实现同一个功能而调用相同的函数，但是因为不同功能调用之间彼此需要实现的功能不经相同，所以需要对于这个共同调用的函数进行改造，使其对于不同的功能函数的调用产生不同的效果。

解决方法：

我在传递的参数中增加了一个way参数，用于标识调用这个函数的功能函数的信息。这样就可以在共用函数内部通过判断分别处理不同的功能。这样就实现了功能的复用。

如图：

```
void printStep(int way, int n, char src, char dst) {
    switch (way)
    {
        case 1:
            cout << " " << n << "#" << " " << src << "-->" << dst << endl;
            break;

        case 2:
            cout << "第" <<setw(4)<< sum++ << " 步( " << n << "#: " << src << "-->" << dst <
            break;

        case 3:
            cout << "第" << setw(4) << sum++ << " 步( " << n << "#: " << src << "-->" << dst
            changeStatus(n, src, dst);
            showStatus();

            break;

        case 4:
            if(delay == 0)
                while (getchar() != '\n');
            else
                Sleep(250 - delay * 50);
```

其中way表示调用的功能函数是哪一个。然后用switch加以判断。

3. 在函数的传递过程中会有时候会出现需要传递数组作为参数的时候。这个时候因为传递的数组参数属于引用，在内部处理的过程中外部数组的值也会随着发生改变，这个时候就需要就会对需要原数组的值作为参数的函数的功能产生影响。比如

在菜单功能9中，需要更具用户的选择移动图形塔，同时需要提前显示文字塔的变化过程。这个时候我使用的函数都是根据内部数组的值来判断打印的盘数。当调用文字塔打印的时候，内部数组因为是公用的所以导致在打印移动动画的时候产生了不同的逻辑。

解决方法：

这个问题暂时没有想到更好的解决办法，所以我在打印完文字塔之后又重新将圆盘移动回原来的柱子上，再次打印动画，能比较好的实现功能，但是会出现效率变低的问题。

如图所示：

```
changeStatus(n, src, dst);
cct_gotoxy(20, 34);
cout << "第" << sum++ << "步(" << n << "#: " << src << "-->" << dst << ")  ";
showStatus();
showStep(11, n, src, dst);
changeStatus(n, dst, src);
movingTable(src, dst);
changeStatus(n, src, dst);
break;
```

5. 心得体会

在完成汉诺塔移动程序的综合开发过程中，我获得了许多宝贵的心得体会和经验教训。

1. 递归思维的重要性

汉诺塔问题的解决充分体现了递归思想的威力。通过将大问题分解为小问题，然后再将小问题缩小规模，直至达到最简单的情况（基准情况），递归方法简化了问题解决的过程。

2. 分治策略的应用

汉诺塔问题的解法是分治策略的一个典型例子，通过将原问题分解成几个规模较小但类似于原问题的子问题，递归地解决这些子问题，然后再合并这些子问题的解以得到原问题的解。

3. 算法效率的考量

虽然递归方法为解决问题提供了一种清晰、简洁的途径，但也需要考虑到其效率和可能的性能问题，比如栈溢出等。在开发过程中，需要权衡递归深度和性能之间的关系。良好的代码结构和清晰的逻辑是软件开发的基础。在编写汉诺塔程序时，递归函数的简洁性有助于提高代码的可读性和维护性。

同时我还获得了许多的经验和教训。比如：

1. 在实际开发中，过深的递归可能导致栈溢出。因此，在设计递归算法时，应当考虑到递归深度的限制，并尽可能优化算法，减少递归深度。递归虽好，但并非所有情况下都是最优解。在一些情况下，迭代方法可能更加高效。因此，开发者应根据实际问题的特点，选择最合适的方法。

递归程序的调试通常比较复杂，因为需要追踪每次递归调用的状态。在开发过程中，应当编写清晰的测试用例，并利用调试工具逐步跟踪递归调用的过程，以便于发现和修正错误。在实际开发中，对输入参数的验证不可忽视。应确保输入参数的有效性，并且对可能的错误情况进行适当的处理，以增强程序的健壮性和用户体验。

附件：源程序

```
cct_setconsoleborder(120, 40, 120, 9000);

    while(true)
    {
     int in;
     in = menu();
     switch (in)
     {

          case 2:
               jibenjiePlus();
               break;

          case 3:
               arrHeng();
               break;

          case 4:
               arrZong();
               break;

          case 5:
               preTuxing();
               break;

          case 6:
               pre2Tuxing();
               break;

          case 7:
               preMoveTuxing();
               break;

          case 8:
               movingTuxing();
               break;

          case 9:
               movingGame();
               break;
     }
}



//hanoi_menu
cct_setconsoleborder(120, 40, 120, 9000);
while(true)
{
     int in;
     in = menu();
     switch (in)
```

```
          case 0:
               return 0;

          case 1:
               jibenjie();
               break;

{
          case 0:
               return 0;

          case 1:
               jibenjie();
               break;

          case 2:
               jibenjiePlus();
               break;

          case 3:
               arrHeng();
               break;

          case 4:
               arrZong();
```

```
            break;

        case 5:
            preTuxing();
            break;

        case 6:
            pre2Tuxing();
            break;

        case 7:
            preMoveTuxing();
            break;

        case 8:
            movingTuxing();
            break;

        case 9:
            movingGame();
            break;
    }
}

static int delay = 0;
int sum = 1;
int arr[3][10] = { 0 };
int stack[3] = { 0 };


void theEnd() {
    cout << "按回车键继续";
    while (_getch() != '\r')
        ;
    cct_cls();
}


void initState() {
    sum = 1;
    for (int i = 0; i < 3;i++) {
        stack[i] = 0;
    }

    for (int i = 0; i < 3;i++) {
        for (int j = 0; j < 10; j++)
        {
            arr[i][j] = 0;
        }
    }
}

//in[] n, src,dst,tmp
void input(int isDelay, int in[]) {
    //初始化全局变量，这一步很重要，防止再次玩游戏的时候上一次的数据残留。
    initState();
    char src, dst;
        cout << "请输入汉诺塔的层数(1-10)" << endl;
        cout << "请输入起始柱(A-C)" << endl;

    while (isDelay) {
        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 >> delay;

    }
}

//根据输入初始化数组和指针
void init(int n, char src, char tmp, char dst)
{
    //对应关系为，arr顺序ABC，num顺序src,dst,tmp

    for (int i = 0; i < n; i++)
    {
        arr[src - 65][i] = n - i;
    }
    stack[src - 65] = n;

}
//改变数组状态
void changeStatus(int n, char src, char dst) {
    //arr[][],num[]
    //对应关系为，arr顺序ABC，stack[]指向顶
    arr[dst - 65][stack[dst - 65]++] = n;
    arr[src - 65][--stack[src - 65]] = 0;

}

//显示数组状态
void showStatus() {
    //A:                    B:
C:10 9 8 7 6 5 4 3 2 1
    for (int i = 0; i < 3; i++)
    {
        cout << (char)(65 + i) << ':';
        if (arr[i][0] == 0) {
            cout << "   ";
        }
        else if (arr[i][0] != 10) {
            cout << ' ' << arr[i][0] << ' ';
```

```
        }
        else {
            cout << arr[i][0] << ' ';
        }

        for (int j = 1; j < 10; j++)
        {
            if (arr[i][j] != 0) {
                cout << arr[i][j] << ' ';
            }
            else {
                cout << "  ";
            }
        }
    }
    cout << endl;
}

//显示文字塔
void showStep(int first, int n, char src, char
dst) {

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (arr[i][j] != 0) {
                cct_gotoxy((13 + i * 10), 19 -
j + first);
                cout << arr[i][j];
            }
            else {
                cct_gotoxy((13 + i * 10), 19 -
j + first);
                cout << ' ';
                if (n == 10 && j == 9) {
                    cct_gotoxy((14 + i * 10),
19 - j + first);
                    cout << ' ';
                }
            }
        }
    }

}

void initScren(int first, int n, char src, char
tmp, char dst) {
    cct_cls();
    cout << "从" << src << "移动到 " << dst <<
", 共 " << n << " 层, 延时设置为 " << delay;
    cct_gotoxy(20, 24 + first);
    cout << "初始:                ";
    showStatus();
    switch (delay)
    {
        case 0:
            while (getchar() != '\n');
            break;
        default:
            Sleep(250 - delay * 50);
            break;
    }
    cct_gotoxy(10, 20 + first);
    cout << "======================";
    cct_gotoxy(13, 21 + first);
    cout << "A         B         C";
    int i = src - 'A';
    for (int j = 0; j < n; j++)
    {
        cct_gotoxy((13 + i * 10), 19 - j +
first);
        cout << arr[i][j];
    }

}


void initTower(int first, int n, char src, char
dst) {
    if (n && !first) {
        cct_cls();
        cct_setcursor(CURSOR_INVISIBLE);
        cout << "从" << src << " 移动到" << dst
<< ", 共 " << n << " 层";
    }
    cct_showch(1, 20, ' ', 14, 0, 23);
    Sleep(100);
    cct_showch(34, 20, ' ', 14, 0, 23);
    Sleep(100);
    cct_showch(67, 20, ' ', 14, 0, 23);
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 12; j++)
        {
            Sleep(100);
            cct_showch((12 + i * 33), 19 - j, '
', 14, 0, 1);
        }
    }
    cct_setcolor();
    cct_gotoxy(1, 37);
    cct_setcursor(CURSOR_VISIBLE_NORMAL);
}
```

```
void printTower(int n, char src, char dst, char
tmp) {
    int x = 3;
    cct_setcursor(CURSOR_INVISIBLE);
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (arr[i][j] != 0) {
                Sleep(100);
                cct_showch((12 + i * 33 -
arr[i][j]), 19 - j, ' ', x + arr[i][j], 0,
(arr[i][j] * 2 + 1));
            }
            else {
                Sleep(100);
                cct_showch((12 + i * 33), 19 -
j, ' ', 14, 0, 1);
            }
        }
    }
    cct_setcolor();
    cct_gotoxy(1, 37);
    cct_setcursor(CURSOR_VISIBLE_NORMAL);
}
void movingTable(char src, char dst) {
    int n = 0, x, x2, begin, end;
    int a = src - 'A';
    int b = dst - 'A';
    for (int i = 9; i >= 0; i--)
    {
        if (arr[a][i] != 0) {
            n = arr[a][i];
            x = (12 + a * 33 - n);
            begin = 19 - i;
            break;
        }
    }
    x2 = (12 + b * 33 - n);
    end = 19;
    for (int i = 9; i >= 0; i--)
    {
        if (arr[b][i] != 0) {
            end = end - i - 1;
            break;
        }
    }
    cct_setcursor(CURSOR_INVISIBLE);
    movingUp(n, x, begin);
    x2 > x ? movingRight(n, x, x2) : movingLeft(n,
x, x2);
    movingDown(n, x2, end);
    cct_setcolor();
    cct_gotoxy(1, 37);
    cct_setcursor(CURSOR_VISIBLE_NORMAL);
}



void printStep(int way, int n, char src, char dst)
{
    switch (way)
    {
        case 1:
            cout << " " << n << "#" << " " << src
<< "-->" << dst << endl;
            break;

        case 2:
            cout << "第" <<setw(4)<< sum++ << "
步(" << n << "#: " << src << "-->" << dst << ")
" << endl;
            break;

        case 3:
            cout << "第" << setw(4) << sum++ <<
" 步(" << n << "#: " << src << "-->" << dst <<
")  ";
            changeStatus(n, src, dst);
            showStatus();

            break;

        case 4:
            if(delay == 0)
                while (getchar() != '\n');
            else
                Sleep(250 - delay * 50);

            changeStatus(n, src, dst);
            cct_gotoxy(20, 24);
            cout << "第" << sum++ << "步(" << n
<< "#: " << src << "-->" << dst << ")  ";
            showStatus();
            showStep(0, n, src, dst);
            break;
        case 5:
            if (delay == 0)
                while (getchar() != '\n');
            else
                Sleep(250 - delay * 50);
```

```
            changeStatus(n, src, dst);
            cct_gotoxy(20, 34);
            cout << "第" << sum++ << "步(" << n
<< "#: " << src << "-->" << dst << ")  ";
            showStatus();
            showStep(11, n, src, dst);
            movingTable(src, dst);
            break;
    }
}

void hanoi(int way, int n, char src, char tmp,
char dst)
{
    if (n == 1) {
        printStep(way, n, src, dst);
    }
    else {
        hanoi(way, n - 1, src, dst, tmp);
        printStep(way, n, src, dst);
        hanoi(way, n - 1, tmp, src, dst);
    }
}

void gameInput(char put[]) {
    cct_gotoxy(1, 36);
    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) :    ";
    cct_gotoxy(61, 36);
    while (true) {
        cin.getline(put, 255,
        if (put[1] == 'A' || put[1] == 'B' ||
put[1] == 'C') {
        }
        else if (put[1] == 'a' || put[1] == 'b'
|| put[1] == 'c') {
            put[1] -= 32;
        }
        else {
            cct_showch(61, 36, ' ', 0, 7, 100);
            cct_gotoxy(61, 36);
            continue;
        }
        break;
    }
}

int isCorrect(char src, char dst) {
    int x = -1, y = 100;
    int a = src - 'A';
    int b = dst - 'A';
    for (int i = 9; i >= 0; i--)
    {
        if (arr[a][i] != 0) {
            x = arr[a][i];
            break;
        }
    }
    for (int i = 9; i >= 0; i--)
    {
        if (arr[b][i] != 0) {
            y = arr[b][i];
            break;
        }
    }
    return x < y ? x : 0;
}

int isWin(int n, char src, char dst) {
    int a = src - 'A';
    int b = dst - 'A';
    for (int i = 0; i < n; i++)
    {
        if (arr[b][i] != n - i || arr[a][i] !=
0)
            return 0;
    }
    return 1;
}
void gameStart(int n, char src, char dst) {
    while (!isWin(n, src, dst)) {
        char put[255];
        gameInput(put);
        if (put[0] == 'q' || put[0] == 'Q') {
            cout << "游戏中止!!!!!";
            return;
        }
        int n = isCorrect(put[0], put[1]);
        if (n == -1) {
            cout << "柱源为空!";
            Sleep(1000);
            cct_showch(0, 37, ' ', 0, 7, 10);
        }
        else if(n) {
            changeStatus(n, put[0], put[1]);
            cct_gotoxy(20, 35);
            cout << "第" << sum++ << "步(" << n
<< "#: " << put[0] << "-->" << put[1] << ")  ";
            showStatus();
            showStep(11, n, put[0], put[1]);
            changeStatus(n, put[1], put[0]);
            movingTable(put[0], put[1]);
            changeStatus(n, put[0], put[1]);
```

```
        }
        else {
            cout << "大盘压小盘，非法移动!";
            Sleep(1000);
            cct_showch(0, 37, ' ', 0, 7, 30);
        }
    }
    cout << "挑战成功！";
}


void jibenjie() {
    int in[4] = { 0 };
    input(0, in);
    cout << "移动步骤为:" << endl;
    hanoi(1, in[0], in[1], in[2], in[3]);
    theEnd();
}


void jibenjiePlus() {
    int in[4] = { 0 };
    input(0, in);
    cout << "移动步骤为:" << endl;
    hanoi(2,in[0], in[1], in[2], in[3]);
    theEnd();
}

void arrHeng() {
    int in[4] = { 0 };
    input(0, in);
    init(in[0], in[1], in[2], in[3]);
    cout << "移动步骤为:" << endl;
    hanoi(3, in[0], in[1], in[2], in[3]);
    theEnd();
}

void arrZong() {
    int in[4] = { 0 };
    input(1, in);
    init(in[0], in[1], in[2], in[3]);
    initScren(0, in[0], in[1], in[2], in[3]);
    hanoi(4, in[0], in[1], in[2], in[3]);
    cct_gotoxy(1,27);
    theEnd();
}

void preTuxing() {
    cct_cls();
    initTower(0, 0, 65, 66);
    cct_gotoxy(1, 38);
    theEnd();
}

void pre2Tuxing() {
    int in[4] = { 0 };
    input(0,in);
    init(in[0], in[1], in[2], in[3]);
    initTower(0, in[0], in[1], in[2]);
    printTower(in[0], in[1], in[2], in[3]);
    cct_gotoxy(1, 38);
    theEnd();
}

void preMoveTuxing() {
    int in[4] = { 0 };
    input(0, in);
    init(in[0], in[1], in[2], in[3]);
    initTower(0, in[0], in[1], in[2]);
    printTower(in[0], in[1], in[2], in[3]);
    movingTable(in[1], in[2]);
    cct_gotoxy(1, 38);
    theEnd();
}

void movingTuxing() {
    int in[4] = { 0 };
    input(1, in);
    init(in[0], in[1], in[2], in[3]);
    initScren(11, in[0], in[1], in[2], in[3]);
    initTower(1, in[0], in[1], in[2]);
    printTower(in[0], in[1], in[2], in[3]);
    hanoi(5, in[0], in[1], in[2], in[3]);
    cct_gotoxy(1, 38);
    theEnd();
}


void movingGame() {
    int in[4] = { 0 };
    input(0, in);
    delay = 2;
    init(in[0], in[1], in[2], in[3]);
    initScren(11, in[0], in[1], in[3], in[2]);
    initTower(1, in[0], in[1], in[2]);
    printTower(in[0], in[1], in[2], in[3]);
    gameStart(in[0], in[1], in[2]);
    cct_gotoxy(1, 38);
    theEnd();
}
```