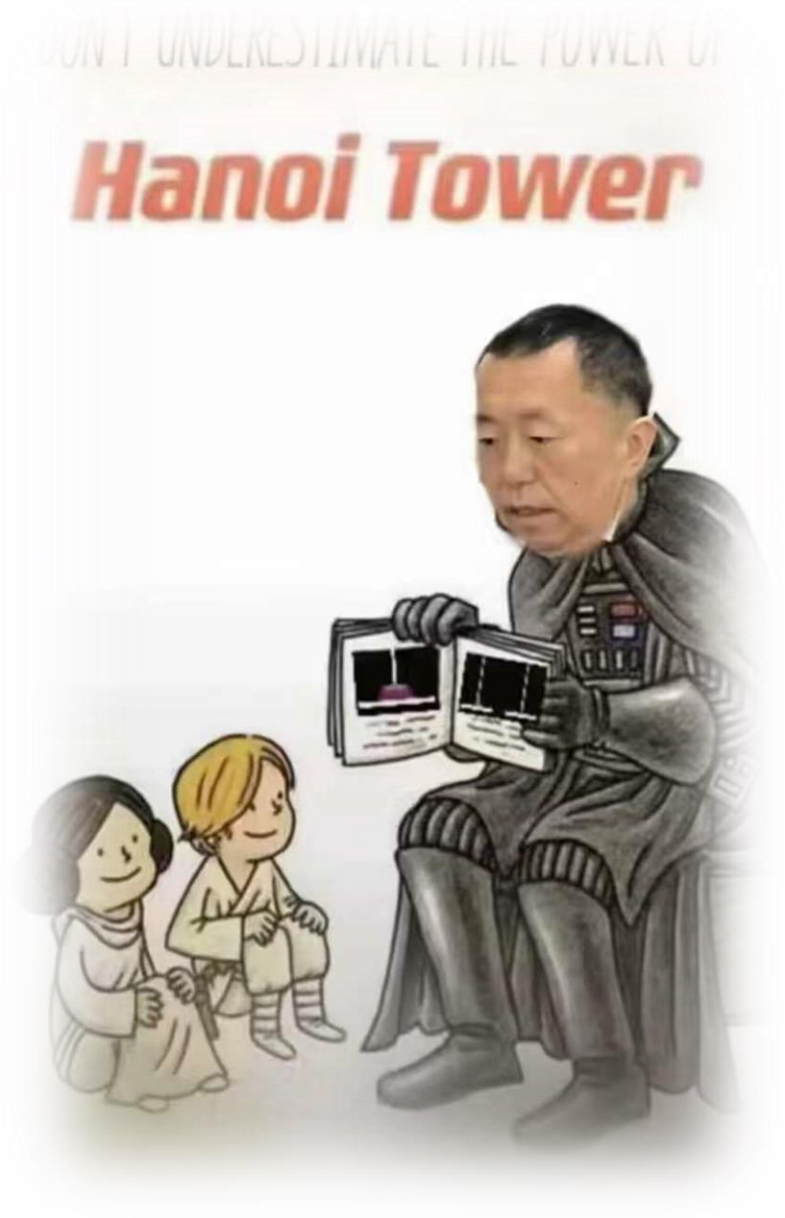
# 汉诺塔综合演示实验报告

**2353726 信01 付煜超 2024.5.16完成**



# 汉诺塔综合演示实验报告

## 1. 题目及基本要求描述

这次作业的目标是解决汉诺塔问题，并通过这个过程培养我们运用函数和参数控制多个变量的能力。汉诺塔问题是一个经典的数学问题，涉及到如何将一堆盘子从一个塔移到另一个塔，其中每个盘子都比它下面的盘子小。这个问题可以通过递归的方式来解决，因此，题目要求我们将程序拆分成多个函数，并确保递归函数的代码量不超过 15 行。这种方法旨在帮助我们保持思路的清晰，同时在需要修改或增加变量时，能够更轻松地对代码进行调整。

拆分程序成多个函数的做法是一种常见的编程实践，它有助于提高代码的可读性和可维护性。通过将程序分解为较小的功能模块，我们可以更容易地理解每个部分的作用，并且可以更方便地重用这些模块。在汉诺塔问题中，我们可以将程序分解为初始化塔、移动盘子、打印移动过程等多个函数，每个函数负责完成特定的任务，这样可以更好地组织代码结构。

同时，题目要求递归函数的代码量不超过 15 行，这也是有一定道理的。递归函数是一种特殊的函数，它在执行过程中会调用自身，通常用于解决可以被分解为相似子问题的问题。在汉诺塔问题中，递归函数可以很自然地应用，因为移动每个盘子的步骤都可以看作是一个类似的子问题。限制递归函数的代码量有助于保持代码的简洁性和清晰度，避免递归层次过深导致代码难以理解和维护。

这种方法还有助于加深对函数式编程和递归思想的理解。函数式编程是一种编程范式，它强调将计算视为数学函数的求值过程，通过组合和应用函数来实现程序的功能。在汉诺塔问题中，我们可以看到函数式编程的一些特点，比如将问题分解成多个函数、避免使用可变状态等。递归思想则是指通过将一个问题分解成更小规模的相似问题来解决原问题，这种思想在解决汉诺塔问题时得到了很好的体现。

总的来说，通过将程序拆分成多个函数并限制递归函数的代码量，我们不仅可以更好地解决汉诺塔问题，还可以提高代码的可读性和可维护性，同时加深对函数式编程和递归思想的理解。这种方法在实际的软件开发中也具有一定的指导意义，可以帮助我们写出更清晰、更易维护的代码。

## 2. 整体设计思路

在整体设计思路方面，我采取了逐步叠加的方法，从之前编写过的代码中逐个提取并整合，遇到问题时增加相应的变量，如果没有问题则继续编写。同时，我也不断增加函数的参数，以确保后续的需求可以在同一个函数内实现，提高了代码的灵活性和可维护性。

我的设计思路主要分为三类：横向输出、纵向输出和柱子与盘子的移动。通过综合前几道题的内容，我得以将横向和纵向输出结合起来，而第三类则需要进行更深入的思考和设计。其中，我主要考虑了

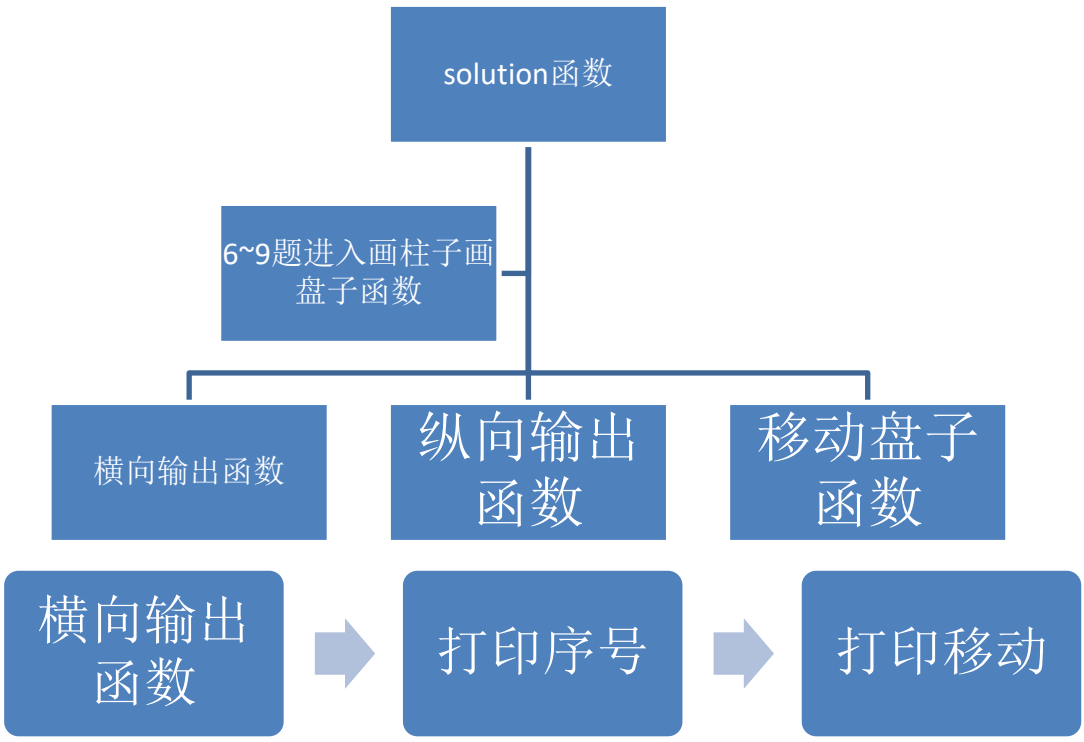
画柱子的函数、画盘子的函数以及移动盘子的函数。在这个过程中，我面临的一个主要问题是如何确保盘子被放置在合适的位置上。

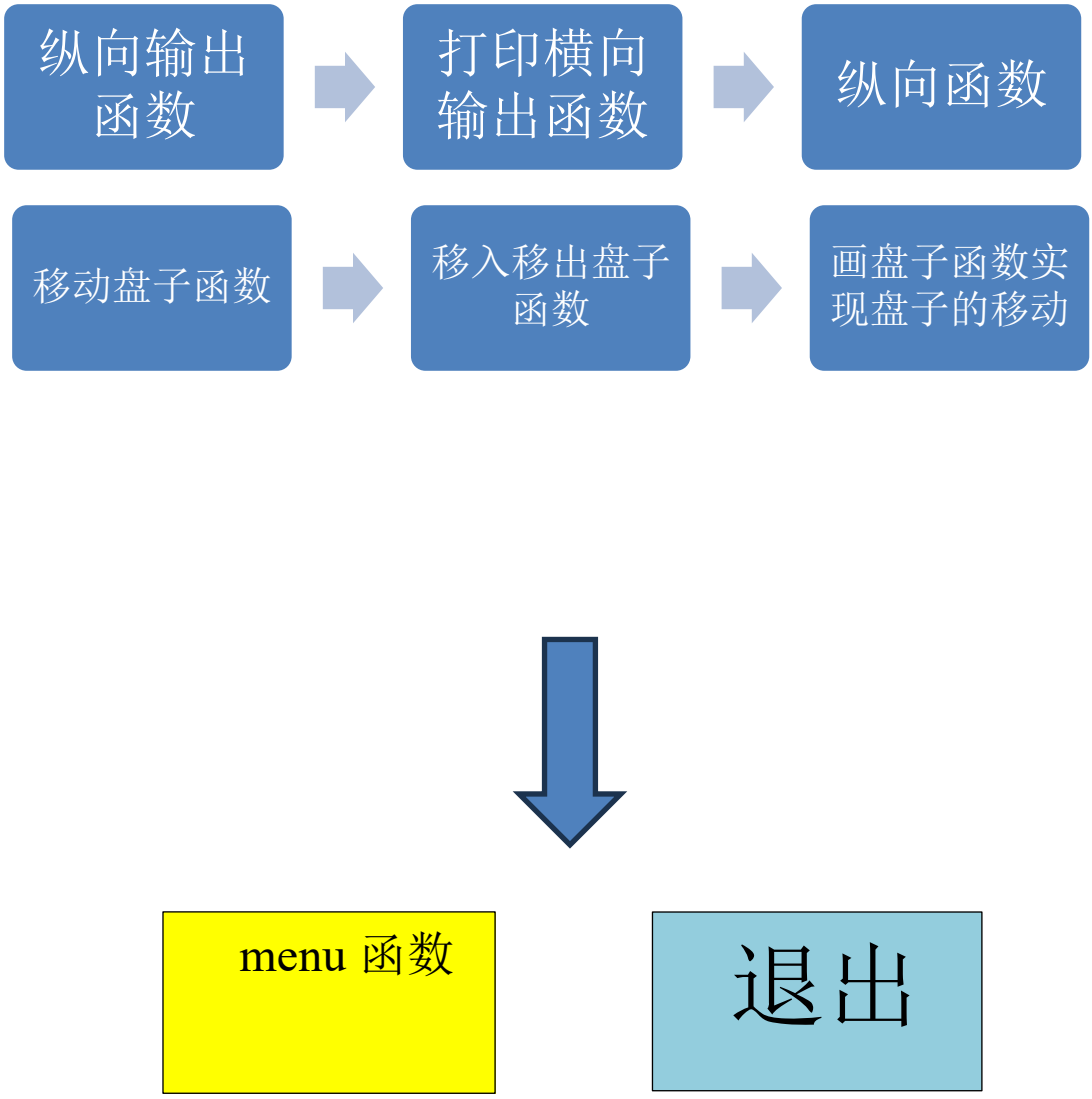
首先，我致力于设计横向输出函数。这个函数的主要目的是将当前盘子的状态以横向的形式展示出来，以便于我们观察和理解问题。然后，我着手设计纵向输出函数，这样可以在横向输出的基础上更清晰地展示出盘子的垂直堆叠情况，有助于我们更直观地理解问题。

接下来，我转向了柱子与盘子的移动函数的设计。这一部分的设计相对复杂，需要考虑如何有效地将盘子从一个柱子移动到另一个柱子，并且确保移动的过程符合汉诺塔问题的规则。在这个过程中，我思考了如何设计一个函数来画出柱子的状态，以及如何设计一个函数来画出盘子的状态。最后，我着手设计了一个函数，用于实现盘子的移动，并确保移动的合法性和正确性。

在整个设计过程中，我不断地思考和调整每个函数的参数，以确保它们可以灵活地适应不同的情况和需求。

同时，我也积极地利用之前编写的代码和经验，以提高代码的复用性和效率。通过这样的设计思路和方法，我成功地完成了汉诺塔问题的综合程序，同时也提高了自己的编程能力和思维方式。




## 3. 调试过程碰到的问题

### 3.1在画图移动盘子时如何定位要移动的盘子

在本次程序中，我遇到最大的问题就是如何定位要移动的盘子和盘子该移动到哪个位置。这个要我们思考如何把纵向打印汉诺塔的指针函数pop和push函数与画盘子函数的结合。此外边界条件也需要一点一点调试才会逐步正确，柱子的重新打印、盘子的重新打印可能第一次边界条件会设置成一样的，然后在debug的时候会发现问题，这时候就可以根据自己的边界做出适当的调整。

在解决汉诺塔问题时，要确定要移动的盘子以及移动到哪个位置是至关重要的。这需要我们将纵向打印汉诺塔的指针功能（pop和push函数）与画盘子的功能结合起来。

首先，我们需要考虑每个盘子在柱子上的位置以及它们的移动规则。在汉诺塔问题中，每次移动都涉及三个柱子之间的操作。我们需要选择一个源柱子、一个目标柱子和一个辅助柱子。通常情况下，我们会将某个盘子从源柱子移动到目标柱子，并利用辅助柱子作为过渡。

在编写代码时，我们可以设计一个函数来模拟这个过程。该函数可能接受源柱子、目标柱子和辅助柱子作为参数，并根据规则选择要移动的盘子以及移动的方向。通过调用这个函数，我们可以逐步解决汉诺塔问题，直到所有盘子都移动到目标柱子上为止。

调试边界条件是解决问题的关键之一。在处理汉诺塔问题时，我们需要考虑盘子数量为0或1的特殊情况，并确保程序在这些情况下能够正确运行。此外，还需要注意柱子的重新打印和盘子的重新打印。有时候，可能会出现柱子和盘子的初始状态设置不正确或移动过程中出现错误的情况。通过逐步调试和根据实际情况调整边界条件，我们可以逐步解决这些问题，并确保程序的正确性和稳定性。

### 3.2如何把之前写过的各种函数整合进一个大框架内

整合之前写过的各种函数到一个大框架内是一个关键的步骤，这有助于确保代码的可维护性和扩展性。在开始大作业之前，我并没有意识到这一点，因此每个作业的适配性都相对较差。在将之前的代码整合到大作业中时，我不得不花费大量的时间去调整各种参数，这让整个过程变得非常耗时。

为了解决这个问题，我首先对之前写过的各个函数进行了梳理和分类。我将它们分为横向输出、纵向输出以及柱子与盘子的移动等不同类别。然后，我逐个将这些函数适配到大作业的框架中，确保它们可以无缝地衔接在一起。

在整合的过程中，我遇到了一些挑战。首先是参数的调整。由于之前的函数可能是根据各自的需求设计的，所以它们的参数可能不太适用于大作业的整体框架。因此，我不得不花费一些时间来调整参数，以确保它们可以正确地与其他函数进行交互。

其次是函数之间的依赖关系。有些函数可能会依赖于其他函数的输出，而这些函数的输出又可能会作为其他函数的输入。因此，我需要仔细地分析和设计函数之间的调用顺序，以确保整个程序能够顺利运行。

另外，我还需要考虑如何处理异常情况和错误。在将各种函数整合到一个大框架中时，难免会出现一些意想不到的问题，比如参数不匹配、函数调用顺序不正确等。因此，我需要添加适当的错误处理机制，以确保程序能够在出现问题时能够及时地进行处理并给出合适的提示。

为了更好地整合之前的各种函数，我还采取了一些策略。比如，我尽量保持函数的独立性和模块化，这样可以使得它们更容易被复用和调试。同时，我也尽量减少函数之间的耦合度，以降低整个程序的复杂度和维护成本。

总的来说，将之前写过的各种函数整合到一个大框架内是一个复杂而繁琐的过程，但它也是非常必要的。通过这样的整合，我不仅提高了对代码结构和组织的理解，还提高了编程的效率和质量。这个经验也让我意识到在以后的项目中，提前规划和设计好整体框架是非常重要的。

### 3.3错误处理的问题

处理错误可能是开发过程中最棘手的部分之一。在解决第九题的错误时，我发现自己陷入了一个循环：需要不断地运行演示来揭示各种错误，然后相应地修改代码或者添加参数到原有函数中。这个过程

耗费了大量的时间和精力。更糟糕的是，之前的错误处理代码已经变得相当臃肿，不得不通过反复调试来理清其中的问题。重新编写一个新的错误处理函数无疑是一个艰巨的任务，它需要仔细地审视每一个可能的错误情况，并确保新的处理方式不会引入更多的问题。这个过程不仅是一项挑战，而且还会大大延写程序的时间。

## 4. 心得体会

汉诺塔问题是一个经典的递归问题，其代码具有很高的通用性。无论是求解3个盘子的汉诺塔问题，还是任意数量盘子的情况，都可以使用相同的递归算法来解决。这种通用性主要体现在以下几个方面：

递归算法的通用性： 汉诺塔问题的解决方案基于递归算法，递归的思想是一种非常通用的编程技巧。无论是解决汉诺塔问题，还是其他类似的问题，都可以使用递归来简洁地描述解决过程。

参数化实现： 汉诺塔问题的解决方案可以通过参数化实现，即将盘子的数量、源柱子、目标柱子和辅助柱子等作为函数的参数传入。这样一来，无论是解决3个盘子还是更多盘子的问题，只需调整参数即可。

盘子移动顺序的问题涉及到了汉诺塔问题的核心逻辑。在汉诺塔问题中，每次移动都必须遵循以下规则：

只能移动一个盘子。

每次移动，盘子都必须从一个柱子移动到另一个柱子。

不能将一个大盘子放在一个小盘子之上。

如果移动顺序不正确，就会破坏这些规则，导致无法正确解决汉诺塔问题。因此，在编写代码时，需要仔细考虑每一步的移动顺序，确保符合这些规则。

另外，错误处理也是编写汉诺塔代码时需要考虑的一个重要方面。可能的错误包括输入参数错误、柱子编号错误等。为了提高代码的健壮性，可以在代码中添加错误处理机制，例如：

参数验证： 在接受用户输入之前，可以添加参数验证功能，确保用户输入的盘子数量、柱子编号等参数符合要求。

异常处理： 在代码中添加异常处理机制，捕获并处理可能发生的异常，以确保程序能够正常运行并给出友好的错误提示。

边界情况处理： 考虑到汉诺塔问题的特殊情况，比如盘子数量为0或1时，可以添加相应的处理逻辑，使程序更加健壮。

通过考虑这些问题，编写出具有良好通用性、正确移动顺序和健壮的错误处理机制的汉诺塔代码，能够更好地满足实际需求，并且能够应对各种可能的情况，提高代码的可靠性和稳定性。

除了以上提到的技术层面的收获，我认为最大的心得是意识到在处理高程作业时，早早地完成任务至关重要。这次的经历让我深刻领悟到，在项目中及早完成任务可以为自己留出更多的时间来处理可能出现的问题、进行调试和优化，从而提高最终的完成质量。因此，我打算将这种经验运用到今后的学习和工作中，尽早规划并完成任务，以提高效率和准确性。

## 5. 附件：源程序

### 5.1 menu函数

```
int menu()
{
    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 a = _getch();
    cout << a - 48;
    return a;

    }
```

### 5.2 main函数

```
int main()
{
    while (true)
    {
        step = 1;
        b0 = 0;
        c0 = 0;                                    //记录圆盘个数//
        for (int i = 0; i < 10; i++)
            A[i] = 0;
        for (int i = 0; i < 10; i++)
            B[i] = 0;
        for (int i = 0; i < 10; i++)
            C[i] = 0;
        t = 0;                                     //延时
        display = 0;                               //显示数值
        /* demo中首先执行此句，将cmd窗口设置为40行x120列（缓冲区宽度120列，行数9000行，即cmd窗口右侧
带有垂直滚动杆）*/
        cct_setconsoleborder(120, 40, 120, 9000);

        int menudig;
        menudig = menu();
        cout << endl;
        cout << endl;
        cout << endl;
        if (menudig == 48)
        {
            break;
```

```
        }
        if (menudig == 49)
            hanoi_s1(1);
        if (menudig == 50)
            hanoi_s2(2);
        if (menudig == 51)

                hanoi_s348(3);
```

### 5.3 hanoi_s56789函数

```
if (sj == 8)
{
    printcolumn(sj, n, x, y, z);
    hanoi(n, x, y, z, sj);
    system("pause");
}
while (sj == 9)
{
    cout << "请输入汉诺塔的层数(1-10)" << endl;
    cin >> n;
    if (cin.good() == 0)
    {
        cin.clear();
        cin.ignore(1024, '\n');
        continue;
    }
    if (n < 1 || n>16)
    {
        cin.clear();
        cin.ignore(1024, '\n');
        continue;
    }
    else
    {
        cin.clear();
        cin.ignore(1024, '\n');
        break;
    }
}
while (sj == 9)
{
    cout << "请输入起始柱(A-C)" << endl;
    int k = getchar();
    if (k == 'a') {
        k = 'A';
    }
    if (k == 'b') {
        k = 'B';
    }
    if (k == 'c') {
        k = 'C';
    }
```

```
if (cin.good() == 0)
{
    cin.clear();
    cin.ignore(1024, '\n');
    continue;
}
if (k != 65 && k != 66 && k != 67 && k != 97 && k != 98 && k != 99)
{
    cin.clear();
    cin.ignore(1024, '\n');
    continue;
}
else
{
    if (k == 65 || k == 97)
        k = 65;

    if (k == 66 || k == 98)
        k = 66;

    if (k == 67 || k == 99)
        k = 67;
    cin.clear();
    cin.ignore(1024, '\n');
}
while (1)
{
    cout << "请输入目标柱(A-C)" << endl;
    int l = getchar();
    if (l == 'a') {
        l = 'A';
    }
    if (l == 'b') {
        l = 'B';
    }
    if (l == 'c') {
        l = 'C';
    }
    if (cin.good() == 0)
    {
        cin.clear();
        cin.ignore(1024, '\n');
        continue;
    }
    if (l != 65 && l != 66 && l != 67 && l != 97 && l != 98 && l != 99)
    {
        cin.clear();
        cin.ignore(1024, '\n');
        continue;
    }
    if (l == k)
    {
        cout << "目标柱(" << char(l) << ")不能与起始柱(" << char(k) << ")相同" << endl;
        cin.clear();
```

```
            cin.ignore(1024, '\n');
            continue;
        }
        else
        {
            if (l == 65 || l == 97)
                l = 65;

            if (l == 66 || l == 98)
                l = 66;

            if (l == 67 || l == 99)
                l = 67;
            int m;
            if (k != 65 && l != 65 && k != 97 && l != 97)
                m = 65;
            if (k != 66 && l != 66 && k != 98 && l != 98)
                m = 66;
            if (k != 67 && l != 67 && k != 99 && l != 99)
                m = 67;
            if (k == 'A')
            {
                for (int i = 0; i < n; i++)
                {
                    A[i] = n - i;
                }
                a0 = n;
            }
            else if (k == 'B')
            {
                for (int i = 0; i < n; i++)
                {
                    B[i] = n - i;
                }
                b0 = n;
            }
            else
            {
                for (int i = 0; i < n; i++)
                {
                    C[i] = n - i;
                }
                c0 = n;
            }
            x = k;
            y = m;
            z = l;
            break;
        }
    }
    break;
}
if (sj == 9)
{
```

```
    printcolumn(sj, n, x, y, z);

}
```

### 5.4 hanoi函数

```
void hanoi(int n, char src, char tmp, char dst, int sj)
{
    if (n == 0)
        return;
    else
    {
        hanoi(n - 1, src, dst, tmp, sj);
        movedisk(src, dst, n);
        printstep(src, dst, n, sj);
        tree(sj, n, src, dst);
        step = step++;
        hanoi(n - 1, tmp, src, dst, sj);
    }
}
```

### 5.5 柱子移动函数

```
for (int l = 15 - i; l > 2; l--)
{
    cct_showch(srccol - n, l, ' ', n, n, 2 * n + 1);
    Sleep(100);
    cct_showch(srccol - n, l, ' ', 0, 0, 2 * n + 1);
    if (l > 3)
        cct_showch(srccol, l, ' ', 14, 14, 1);
}
cct_setcolor(); //恢复缺省颜色
//上移盘子
if (dstcol > srccol)//右移
{
    for (int m = srccol - n; m <= dstcol - n; m++)
    {
        cct_showch(m, 3, ' ', n, n, 2 * n + 1);
        Sleep(100);
        if (m < dstcol - n)
        {
            cct_showch(m, 3, ' ', 0, 0, 2 * n + 1);
        }
    }
}
if (dstcol < srccol)//左移
{
    for (int m = srccol - n; m >= dstcol - n; m--)
    {
        cct_showch(m, 3, ' ', n, n, 2 * n + 1);
        Sleep(100);
        if (m > dstcol - n)
        {
```

```
            cct_showch(m, 3, ' ', 0, 0, 2 * n + 1);
        }
    }
}
cct_setcolor(); //恢复缺省颜色
for (int m = 3; m <= 16 - k; m++)//下移
{
    cct_showch(dstcol - n, m, ' ', n, n, 2 * n + 1);
    Sleep(100);
    if (m < 16 - k)
    {
        cct_showch(dstcol - n, m, ' ', 0, 0, 2 * n + 1);
    }
    if (m > 3)
        cct_showch(dstcol, m, ' ', 14, 14, 1);
}
cct_showch(dstcol - n, 16 - k, ' ', n, n, 2 * n + 1);
cct_setcolor(); //恢复缺省颜色
```