1 n-1个盘子：src→mid

2 第n个盘子：src→dst

3 n-1个盘子：mid→dst

1 A B C

1 A C B 1 A C B

2 A C B 1 C A B 1 C A B 1 C A B 1 B C A

1 A B C 1 A B C 1 A B C 1 A B C 1 A B C 1 B A C 1 B A C

3 A B C 2 B A C 2 B A C 2 B A C 2 B A C 2 B A C 2 B A C 1 A B C 1 A B C

#include <iostream> // 用栈代替递归来处理Hanoi（汉诺塔）问题

#include <stack>

using namespace std;

struct Problem{

int n;

char src,mid,dst; // src表示第一个柱子（源），dst表示第三个柱子（目的地）

Problem(int nn,char s,char m,char d):n(nn),src(s),mid(m),dst(d){}

}; // 一个Problem变量代表一个子问题，将src上的n个盘子以mid为中介，移动到dst

stack<Problem> stk; // 一个元素代表一个子问题，若有n个盘子，则栈的高度不超过n\*3

void main()

{

int n;

printf("请输入盘子数量n:\n");

cin>>n;

stk.push(Problem(n,'A','B','C')); // 初始化问题入栈

while(!stk.empty())

{

Problem cp = stk.top(); // cp表示当前子问题current sub-problem

stk.pop();

if(cp.n==1) // 可以直接输出的情形

cout<<cp.src<<"->"<<cp.dst<<endl;

else // 分解子问题并入栈

{

stk.push(Problem(cp.n-1, cp.mid, cp.src, cp.dst)); // 3 n-1个盘子：mid→dst

stk.push(Problem(1, cp.src, cp.mid, cp.dst)); // 2 第n个盘子：src→dst

stk.push(Problem(cp.n-1, cp.src, cp.dst, cp.mid)); // 1 n-1个盘子：src→mid

}

}

system("pause");

}

#include <stdio.h> // 用递归来处理Hanoi（汉诺塔）问题

#include <stdlib.h>

void hanoi(int n, char from, char temp, char to)

{

if (n==1)

printf("%c→%c(1)\n",from,to);

else

{

//printf("hanoi(%d,%c,%c,%c)\n",n-1,from,to,temp);

hanoi(n-1,from,to,temp); // 将x上编号为1到n-1的圆盘移到y，z作辅助塔

printf("%c→%c\n",from,to); // 将x上编号为n的圆盘移到z

//printf("hanoi(%d,%c,%c,%c)\n",n-1,temp,from,to);

hanoi(n-1,temp,from,to); // 将y上编号为1到n-1的圆盘移到z，y作辅助塔

}

}

int main()

{

int n;

printf("请输入盘子数量n:\n");

scanf("%d",&n);

hanoi(n,'A','B','C');

system("pause");

return 0;

}

/\*

请输入盘子数量n:

4

A→B(1)

A→C

B→C(1)

A→B

C→A(1)

C→B

A→B(1)

A→C

B→C(1)

B→A

C→A(1)

B→C

A→B(1)

A→C

B→C(1)

\*/

//http://www.7k7k.com/swf/37623.htm

/\*

你如果掌握了第一个（最小的）盘子的移动规模，这个游戏你玩起来就会得心应手。

当n是奇数时，第一个盘子只能在柱子ACBACB……间循环移动；

当n是偶数时，第一个盘子只能在柱子ABCABC……间循环移动；

请输入盘子数量n:

5

A→C(1)

A→B

C→B(1)

A→C

B→A(1)

B→C

A→C(1)

A→B

C→B(1)

C→A

B→A(1)

C→B

A→C(1)

A→B

C→B(1)

A→C

B→A(1)

B→C

A→C(1)

B→A

C→B(1)

C→A

B→A(1)

B→C

A→C(1)

A→B

C→B(1)

A→C

B→A(1)

B→C

A→C(1)

\*/

#endif