5.5.2 汉诺塔递归栈

 问题抽象

 3个塔，n个碟子

 初始：所有碟子放在1号塔，大的在底下，小的在上面

 任务：把碟子移动到2号塔，顺序不变， 可用3号塔辅助

 限制

 每次只能移动一个碟子

 总是大碟子在下，小的在上

 递归解法

 移动碟子的方法：move(n, t1, t2, t3)——

将n个碟子从t1移到t2，t3辅助

 可分解为3个步骤

 将n-1个碟子从t1移到t3：move(n-1, t1, t3, t2)

 将最大的碟子从t1移到t2

 将n-1个碟子从t3移到t2：move(n-1, t3, t2, t1)

 汉诺塔递归程序

void TowersOfHanoi(int n, int x, int y, int z)

{// Move the top n disks from tower x to tower y.

// Use tower z for intermediate storage.

if (n > 0) {

TowersOfHanoi(n-1, x, z, y);

cout << "Move top disk from tower " << x

<< " to top of tower " << y << endl;

TowersOfHanoi(n-1, z, y, x);}

}

 moves(n)=2n-1——最少次数，Θ(2n)

 一般递归程序转换为循环

 递归函数主体的转换

 转换为循环，while(1)即可

 递归调用的转换

 将当前参数、局部变量…（活动记录）压栈

 参照调用方式改变参数，继续循环

 函数执行结束——递归返回的处理

 若栈空，整个递归过程结束，跳出循环

 否则，将调用者的活动记录弹出栈，恢复其环境，继续循环

 递归函数不同入口的区分——返回地址的处理

 上例：ENTRANCE、FIRST、SECOND

 活动记录的一部分，与参数、局部变量一同压栈、出栈

 在循环主体中，根据当前活动记录的入口值，执行不同代码

 汉诺塔的递归栈实现

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

#include <stack>

using namespace ::std;

enum {

ENTRANCE = 0,

FIRST,

SECOND

};

struct ac {

int n, x, y, z;

int r;

};

 汉诺塔的递归栈实现

void hanoi(int n, int x, int y, int z)

{

stack<struct ac> stack;

struct ac ac = { n, x, y, z, ENTRANCE };

while (1)

{

if (ac.n <= 0)

{

if (stack.empty())

break;

ac = stack.top();

stack.pop();

if (ac.r == ENTRANCE)

ac.r = FIRST;

else

ac.r = SECOND;

}

 汉诺塔递归栈实现

if (ac.r == ENTRANCE)

{

stack.push(ac);

ac.n--;

swap(ac.y, ac.z);

}

else if (ac.r == FIRST)

{

cout << "Move top disk from tower " << ac.x

<< " to top of tower " << ac.y << endl;

stack.push(ac);

ac.r = ENTRANCE;

ac.n--;

swap(ac.x, ac.z);

}

 汉诺塔递归栈实现

else if (ac.r == SECOND)

{

if (stack.empty())

break;

ac = stack.top();

stack.pop();

if (ac.r == ENTRANCE)

ac.r = FIRST;

else

ac.r = SECOND;

}

}

}

 汉诺塔递归栈实现

int main(int argc, char \*argv[])

{

int n;

scanf("%d", &n);

hanoi(n, 1, 2, 3);

return 0;

}