Ханойская башня

Это старая классическая задача, встречающаяся во многих учебниках и пособиях по программированию. Мы тоже не пройдем мимо и используем ее для того, чтобы еще раз поговорить о рекурсии. Общая структура рекурсивного решения такова:

1. Дан вычислительный (и не обязательно вычислительный) процесс, в котором результат каждого шага определяется через результат предыдущего.

2. В конечном итоге процесс обязательно приводит к какой-то тривиальной ситуации, в которой результат можно определить явным образом без дальнейших рекурсивных обращений.

Очевидно, что любая рекурсия обладает следующими свойствами: во-первых, в процессе рекурсивных вызовов изменяются какие-то данные, во-вторых, существует конфигурация данных, при которой рекурсивные вызовы прекращаются и что-то делается явным образом (обычно при вычислении каких-либо величин) или действие просто прекращается так. Кстати первое свойство вытекает из второго. Действительно, если ничего не будет изменяться, то говорить о том, что наступит тривиальная ситуация бессмысленно. Простейший пример, демонстрирующий сказанное, — это программа рекурсивного счета факториала.

Листинг

```
program example;
 uses crt;
 var
  n:integer;
function factorial(n:integer):integer;
begin
 if n=1 then factorial:=1
  else factorial:=n*factorial(n-1);
end;
begin
 clrscr;
 read(n);
 write(factorial(n));
end.
```

В процессе счета факториала тривиальная ситуация — это случай при `n=1`, а от вызова к вызову `n` уменьшается на 1, что и дает гарантию завершения рекурсивных вызовов. Гарантия завершения процесса рекурсивных вызовов очень важный момент. Надо всегда помнить, что рекурсия не завершается сама по себе, и что организованный вами процесс может и не дойти до тривиального случая, что создаст ошибку под названием "переполнение стека".

Если говорить об ошибках, то сразу стоит сказать еще об одной, более тонкой. Обратите внимание, что в той же программе счета факториала нет циклов, хотя нерекурсивное решение обязательно содержит цикл. Это означает, что рекурсия есть форма организации циклических процессов, и если в вашем рекурсивном решении появляются циклы, участвующие в организации основной логики, то это должно навести вас на размышление о верности логики. И, во всяком случае, рекурсивные вызовы внутри

циклов всегда сильно усложняют логику, если без них возможно обойтись, то лучше это сделать.

Программа счета факториала полезна только для демонстрации общих правил, вообще-то считать факториал рекурсивной программой вряд ли хорошо. Это обычный линейный процесс, и нерекурсивное решение, если не короче, то проще. Но мы немного на нем задержимся, чтобы напомнить еще одну важную деталь — рекурсивному решению всегда сопутствует рекуррентное определение. Например, наше рекурсивное решение задачи о факториале предполагает его рекуррентное определение n!=n*(n-1)!.

Если вспомнить задачу о снежинке, то также можно заметить, что большая снежинка состоит из маленьких, такой же формы, те в свою очередь из еще более маленьких и т. д.

Вернемся к задаче о Ханойской башне. Смысл ее в следующем: дано три подставки, на первой из них лежит некоторое количество дисков, таких что для любой пары дисков тот, что сверху, имеет радиус меньший того, что ниже. Необходимо переложить все диски на третью подставку, не нарушая следующих правил:

- за один раз можно брать только один диск;
- диск можно класть только на диск большего радиуса или на пустую подставку.

Начнем наши рассуждения с попытки обнаружить рекуррентную природу задачи. Это легко сделать, если внимательно рассмотреть два рисунка — 1 и.2. На рисунке 1 показана исходная задача.

Рисунок 1. Задача о Ханойских башнях (дано)

Рисунок 2. Задача о Ханойских башнях (процесс)

Если удастся получить ситуацию, изображенную на рисунке 2, то исходная задача перемещения пирамиды с левой подставки на правую решится в два шага. Во-первых, диск с левой подставки перемещается на правую, а затем решается задача перемещения меньшей пирамиды со средней подставки на правую. А для того, чтобы получить такую ситуацию надо сначала решить задачу перемещения ханойской башни без нижнего диска

на среднюю подставку. Таким образом, исходная задача о перемещении башни из пяти дисков сводится к двум задачам о перемещении четырех дисков.

Рекуррентный характер задачи налицо, следовательно, поиск рекурсивного решения вполне оправдан. Тривиальный случай для построения рекурсивной процедуры тоже понятен — это башня из одного диска.

Входные данные:

- номера подставок в следующем порядке: подставка, с которой необходимо переместить текущую пирамиду, вспомогательная подставка, и подставка, на которую требуется переместить пирамиду;
- высота перемещаемой пирамиды.

Алгоритм рекурсивной процедуры будет выглядеть так:

```
Если перемещаемая высота равна 1 то
  Переместить один диск с исходной подставки на подставку-цель.
  Иначе
    Вызвать процедуру, перемещающую пирамиду с высотой на единицу
    меньше с исходной подставки на вспомогательную (то есть кроме
    нижнего диска).
    Переместить нижний диск с исходной подставки на подставку-цель.
    Вызывать процедуру, перемещающую пирамиду с вспомогательной
    подставки на подставку-цель.
```

Текст программы показан в листинге

Листинг

```
program example;
 uses crt,graph;
 type
  mas=record
        krug:array[1..10] of integer;
        h:integer;
      end;
 var
  a:array[1..3] of mas;
  dr,md,i,n:integer;
procedure ris;
var
 i:integer;
begin
 cleardevice;
 for i:=1 to n do
  begin
   if a[1].krug[i]>0 then circle(100,200,a[1].krug[i]);
   if a[2].krug[i]>0 then circle(300,200,a[2].krug[i]);
   if a[3].krug[i]>0 then circle(500,200,a[3].krug[i]);
  end;
end;

procedure hanoy(a1,a2,a3,m:integer);
procedure perest;
begin
    a[a3].h:=a[a3].h+1;
    a[a3].krug[a[a3].h]:=a[a1].krug[a[a1].h];
```

```
    a[a1].krug[a[a1].h]:=0;
    a[a1].h:=a[a1].h-1;
    ris;
    readkey;
end;
begin
 if m=1 then perest
 else
  begin
   hanoy(a1,a3,a2,m-1);
   perest;
   hanoy(a2,a1,a3,m-1);
  end;
end;
begin
 dr:=detect;initgraph(dr,md,'');
 read(n);
 for i:=1 to n do
   a[1].krug[i]:=(n-i+1)*10;
 a[1].h:=n;a[2].h:=0;a[3].h:=0;
 ris;
 readkey;
 hanoy(1,2,3,n);
end.
```

Полученное решение — хороший пример формального подхода. Мы уже говорили о том, что для построения алгоритма весьма желательно выработать хорошее представление о процессе. Попробуйте выработать у себя хорошее представление о процессе перемещения дисков. Для двух дисков это легко, для трех — затруднительно, а попробуйте представить себе процесс для пяти или шести дисков. Это будет уже весьма непросто. Или даже попробуйте, поняв алгоритм на четырех дисках, повторить его на шести. Совершенно не очевидно, что у вас это получится. Кстати даже в анализе задачи мы нашли существенную сложность. Рекуррентное определение говорит о том, что исходная задача сводится к двум задачам, при этом целевое использование подставок меняется. Вот эта постоянная перемена предназначения подставок сильно осложняет понимание процесса.

Но к процессу поиска решения можно подойти и иначе. Мы уже описали некую общую технологию построения рекурсивных программ. О ней сказано достаточно много, сейчас отметим только, что она состоит из ряда формальных шагов, фактически трех: определение условия завершения, организация рекурсивного вызова, выполнение текущих операций. Если это сделать корректно, то беспокоиться о правильном понимании процесса уже не надо, он будет идти так, как надо.

В рассмотренной задаче на каждом шагу выполняется элементарное перемещение дисков и два рекурсивных вызова, отличающихся друг от друга назначением подставок (какая из них конечная, а какая вспомогательная). Точное определение назначения подставок при каждом вызове избавляет нас от необходимости понимания всего процесса перемещений. Что-то подобное уже было в задаче о снежинке. Вспомните, мы строили рекурсивную процедуру шаг за шагом, не слова не говоря о процессе рисования в целом.

Второе решение.

В листинге приведен текст второго решения той же задачи. Оно также имеет рекурсивный характер. Текст рекурсивной процедуры несколько сложнее (во всяком случае длиннее), но структура данных, реализующая пирамиду существенно проще. Мы

не будем подробно анализировать эту программу, попробуйте разобраться в принципах ее работы самостоятельно.

Листинг

```
program example;
   uses crt,graph;
 var
   driver,mode,i,r,n:integer;
   x:array [0..15] of integer;
 procedure gr(x:array of integer);
  var
    c:char;
  begin
   for i:=1 to n do
    begin
     circle(x[i],239,r*i);
     setfillstyle(1,i);
     floodfill(x[i]+r*i-1,239,15);
    end;
   c:=readkey;
   setfillstyle(1,16);
   floodfill(1,1,14);
  end;
procedure abc(a,b,c,a1,b1,c1:integer);
   begin
    if a>1 then  abc(a-2,a-1,a,a1,b1,c1);
    if a=1 then  begin x[a]:=b1; gr(x); end;
    x[b]:=c1; gr(x);
    if a>1 then abc(a-2,a-1,a,b1,c1,a1);
    if a=1 then  begin x[a]:=c1; gr(x); end;
    x[c]:=b1; gr(x);
    if a>1 then  abc(a-2,a-1,a,c1,a1,b1);
    if a=1 then  begin  x[a]:=a1;  gr(x); end;
    x[b]:=b1; gr(x);
    if a>1 then  abc(a-2,a-1,a,a1,b1,c1);
    if a=1 then  begin x[a]:=b1;  gr(x); end;
   end;
 begin
  clrscr;
  writeln('number of circle');
  readln(n);
  r:=round(100/n);
   driver:=detect; initgraph(driver,mode,'');
  for i:=1 to n do  x[i]:=110;
 gr(x);
 abc(n-2,n-1,n,100,530,320);
 end.
```

В заключение.

У двух приведенных рекурсивных решений даже на первый взгляд есть что-то общее, но видна и существенная разница. Обратите внимание, что вызов рекурсивной процедуры во втором варианте требует шесть аргументов. Можно найти и другие отличия. Корень различий в представлении подставок и колец структурами данных. Проведите анализ структур данных, который даст понимание, почему эти два решения так сильно отличаются.