A TASTE OF SMALLTALK

1

THE EXAMPLE

THE TOWER OF HANOI

RECURSIVE: adj. see RECURSIVE.
STAN **KELLY-BOOTLE**, *Devil's DP Dictionary*

This program represents one of the few examples of agreement in computer science. Practically every programming text introduces recursion using this program, and, in this case, Smalltalk is no exception. The Tower of Hanoi is based on a game (popular in the 1960s) that had its own mythology (also popular in the 1960s):

> An ancient myth has it that in some temple in the Far East, time is marked off by monks engaged in the transfer of 64 disks from one of three pins to another. The universe as we know it will end when they are done. The reason we do not have to concern ourselves about the cosmological implications of this is that their progress is kept in check by some clever rules: the monks can only move one disk at a time; the disks all have different diameters; and no disk can ever be placed on top of a smaller one.*

* P.H. Winston and B.K.P. Horn, *LISP* (Reading, Massachusetts: Addison-Wesley Publishing Company, 1981), 88.

The rules seem easy enough, except with 64 disks you might get a little confused and never be sure if you were making forward progress. Luckily, there is a way to simplify things and find an algorithm. We quote from the second edition of *Oh! Pascal!* by Doug Cooper and Michael Clancy.

> Let's try to get a handle on how the moves are made for stacks of various heights. Clearly a height of 1 is trivial—we move the disk directly from A to C. What about a height of2? We put the top disk out of the way—to B. Then the bottom disk goes to C, and the smaller disk from B to C.
>
> With a stack of height 3 it gets interesting. Let's suppose, though, that we restate the problem (as we're liable to do when we're up to something). Instead of moving 3 disks from A to C, let's move 2 disks from A to B—we already know how to move two disks from one peg to another. Next, move the third disk directly to C. Finally, make another two-disk move, from B to C. We've switched all three disks.

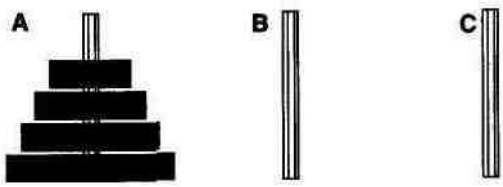

Figure 1.1

> How about starting with 4 disks? [See Figure 1.1.] Once more, let's begin by restating the problem. Can we move 3 disks from A to B? Sure—it's essentially the same as moving them from A to C. [See Figure 1.2.]

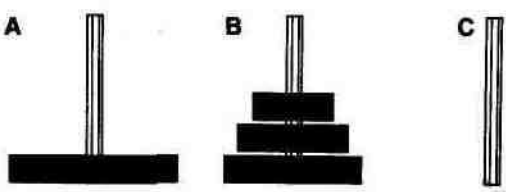

Figure 1.2

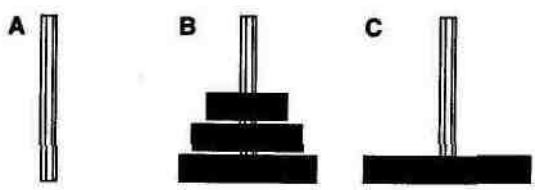

Figure 1.3

> Then we switch the fourth disk directly from A to C [Figure 1.3], and, Bnally, transfer 3 disks from B to C. [See Figure 1.4.]
>
> As you can probably gather, we've insisted on restating the problem in a particular way each time so that we can develop a special insight. We

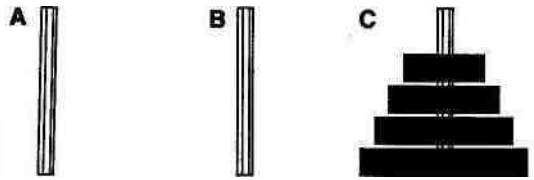

Figure 1.4

begin to solve the Towers of Hanoi problem for a stack of height *n* by trying to solve it for a stack of height n — *1.* This solution must wait until we solve for *(n — l) — l ,* and so on. Eventually we get to the trivial case of *n* equaling 1, and can begin to work our way back up.

Almost without realizing it, we've used a high-priced method of thinking called *induction.* We start by solving a simple case of the Towers of Hanoi problem—a tower of height one or two. Then, we show that even if we start with a larger number, we can always work our way down to a problem that we know how to solve. This is the heart of what will become our recursive solution to the problem.

Now we're ready to make a recursive statement of our solution. To move *n* disks from peg A to peg C:

1. Move *n—1* disks from A to B.
2. Move 1 disk from A to C.
3. Move *n—1* disks from B to C.

In steps 1 and 3, of course, we will use the remaining peg as an auxiliary "holding" peg. *

Versions of this algorithm in Pascal, C, LISP, and Smalltalk are shown on the next pages. In the programs "frompin" refers to the index of the pin a disk will move from, "topin" refers to its destination, and "usingpin" refers to the remaining pin. To describe the poles that hold the disks, we will use the word "pole" in the text and "pin" in the programs. We were surprised at the variety of notation for the same program in the three "normal" programming languages.

A ROSETTA STONE: PASCAL, C, LISP, AND SMALLTALK

The Tower of Hanoi In Pascal

This program is derived from a program found on pages 102-103 in *Programming in Pascal* by Peter Grogono, Addison-Wesley, Reading, Massachusetts, 1978.

• Doug Cooper and Michael Clancy, *Oh! Pascal!,* 2nd edition (New York: Norton, 1985), 237-38.

```
program hanoi(input, output);
  var total: integer;

  procedure movetower (Height, Frompin, Topin, Usingpin : integer);

    procedure movedisk ( Frompin, Topin : integer);
    begin
          writeln( Frompin, '->', Topin)
    end; { movedisk }

  begin { movetower}
    if Height > 0 then
      begin
          movetower(Height -1, Frompin, Usingpin, Topin);
          movedisk( Frompin, Topin);
          movetower(Height -1, Usingpin, Topin, Frompin)
      end
  end; {movetower}

begin {hanoi}
    readftotal);
    movetowerftotal, 1,3,2)
end. {hanoi}
```

If you wanted to run the program on Berkeley Unix 4.2 on the VAX-11/780, you would type the above program into hanoi.p and then type:

pix hanoi.p

(which compiles and interprets the program). It replies with:

Execution begins. . .

You now type the number of disks:

3

and here is what happens:

```
1 -> 3
1 -> 2
3 -> 2
1 -> 3
2 -> 1
2 -> 3
1 -> 3
```

and at the end of the program the Pascal interpreter types:

Execution terminated.
45 statements executed in 0.05 seconds cpu time.

The Tower of Hanoi in C

Here is a C version to examine:

```
main()
{
   int howmany;
   howmany = getchar() - '0';
   movetower(howmany, 1,3,2);
}

movetower(height, frompin, topin, usingpin)
intheight,frompin,topin,usingpin;
{
   if(height > 0) {
      I* Move the whole stack on frompin to usingpin */
      movetower(height-1, frompin, usingpin, topin);
      I* Move the bottom disk */
      movedisk( frompin, topin)
      I* Move the whole stack on usingpin to topin *I
      movetower(height-1,usingpin,topin,frompin);
   };
}

movediskffrompin,topin) int frompin,topin;
{
      printf("%d  -> %d\n",frompin,topin);
}
```

If you wanted to run the program on Berkeley Unix 4.2 on the VAX-11/780, you would type the above program into hanoi.c and then type:

ec hanoi.c

(which compiles the program). You then type

a.out

(the name of the object file) and then you type the number of disks:

3

and here is what happens:

```
1 -> 3
1 -> 2
3 -> 2
1 -> 3
2 -> 1
2 -> 3
1 -> 3
```

The Tower of Hanoi in LISP

This program is derived from a program found on pages 88-90 in LZSP by P.H. Winston and B.K.P. Horn, Addison-Wesley, Reading, Massachusetts, 1981.

```
(defun Tower-01'-Hanoi nil (Transfer '1 '3'2 (read)))    ; N disks on 1 first.
(defun Move-Disk (From-pin To-pin)
    (print (list From-pin ' - > To-pin))                 ; Print instruction.
    (terpri))                                            ; Start new line.
(defun Transfer (From-pin To-pin Using-pin Height)
    (cond ((equal Height 1)
      (Move-Disk From-pin To-pin))                       ; Transfer one disk.
    ft (Transfer   From-pin                              ; Move from From-pin
        Using-pin                                        ; to Using-pin
        To-pin                                           ; using To-pin as space
        (sub1 Height))                                   ; (n -1) disks.
    (Move-Disk From-pin To-pin)                          ; Move lowest disk.
    (Transfer   Using-pin                                ; Move from Using-pin
        To-pin                                           ; to To-pin
        From-pin                                         ; using From-pin as space
        (sub1 Height)))))                                ; (n -1) disks.
```

If you wanted to run the program on Berkeley Unix 4.2 on the VAX-11/780, you would type the above program into hanoi.lisp and then type:

lisp

It replies with:

Franz Lisp, Opus 38.72
->

You now load the program:

(load 'hanoi. lisp)

and it replies with:

[load hanoi.lisp]
t

To invoke the program you type:

(Tower-Of-Hanoi)

then tell it the number of disks:

3

and here is what happens:

```
(1 ->3)
(1 ->2)
( 3 - > 2 )
(1 ->3)
( 2 - > 1 )
( 2 - > 3 )
(1 ->3)
nil
```

You then type control D and it says:

```
Goodbye
```

The Tower of Hanoi in Smalltalk

In the Smalltalk version of the program, there are two procedures. Each is typed separately into the bottom pane of a window on the screen (called the System Browser) and then compiled and merged into the system by choosing a command from a pop-up menu. (Don't try to type this in now. If you absolutely can't stand not running the Smalltalk version now, you may skip to the second section of Chapter 2, and continue on into Chapter 3.)

```
moveTower: height from: fromPin to: toPin using: usingPin
  "Recursive procedure to move the disk at a height from one
  pin to another pin using a third pin"
  (height > 0) ifTrue: [
    self moveTower: (height-1) from: fromPin to: usingPin using: toPin.
    self moveDisk: fromPin to: toPin.
    self moveTower: (height-1) from: usingPin to: toPin using: fromPin]

moveDisk: fromPin to: toPin
  "Move disk from a pin to another pin. Print the results in the
    transcript window"
  Transcript cr.
  Transcript show: (fromPin printString,' - > ', toPin printString).
```

To run the program, type the text given below into any window that can contain Smalltalk code, select it, and choose **do** it from the middle-button pop-up menu.

```
(Object new) moveTower: 3 from: 1 to: 3 using: 2.
```

The Transcript Window will show:

```
1 ->3
1 ->2
3 ->2
1 ->3
2 ->1
2 ->3
1 ->3
```

We chose the Tower of Hanoi as an example because it is a short program that solves an interesting problem, and because it is so widely used. However, we are astonished at how unclear this recursive solution is. The program has no permanent variables and no assignment statements. The "movedisk" routine does not actually move anything. If you stop the program in the middle, where is the information stored? Generations of students have been confused by the fact that the entire state of the towers is held "on the stack." The arguments to all of the "movetower" routines that are suspended, partially finished, hold the information about the locations of the disks.

The recursive solution presented here is not "object-oriented." The program in Smalltalk looks very much like the program in the other languages. It contains objects, but they are not exhibiting the full power of object-oriented programming. In Chapter 6, we will explore a more intuitive algorithm for solving the Tower of Hanoi problem. The Smalltalk program we write for it will show off objects and object-oriented programming to their full advantage.

Let's face it, except for all those parentheses in one language (we won't point fingers), and all those colons in another, there isn't much difference between the programs. Smalltalk, however, has an unconventional model underneath that belies its surface similarity to other languages.