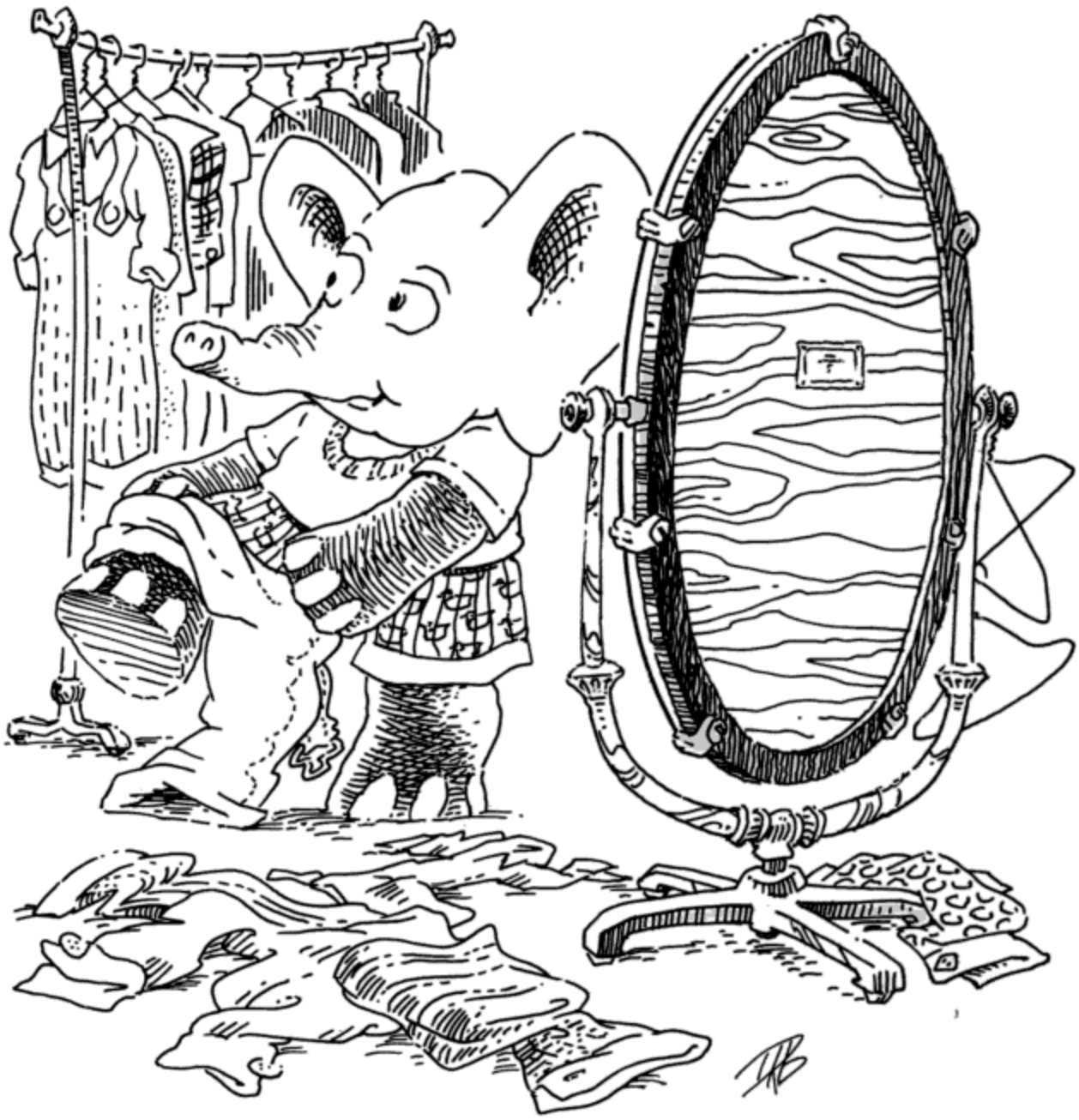
17.
We Change, Therefore We Are!



Is there a (**set!** m ...) in the value part of (**let** ((m n)) ...)

No. Are you asking whether we should unname again?

We could, couldn't we?

Yes, because now a name is replaced by a name.

Do it again!

```
(define deepM
  (let ((Rs (quote ()))
        (Ns (quote ())))
    (lambda (n)
      (let ((exists (find n Ns RS)))
        (if (atom? exists)
            (let ((result ...))
              (set! Rs (cons result Rs))
              (set! Ns (cons n Ns))
              result)
            exists)))))
```

```
...
(if (zero? n)
    (quote pizza)
    (cons (deepM (sub1 n))
          (quote ())))
...
```

Wouldn't you like to know how much help *deepM* gives?

What does that mean?

Once upon a time, we wrote *deepM* to remember what values *deep* had for given numbers.

Oh, yes.

How many *cons*es does *deep* use to build pizza

None.

How many *cons*es does *deep* use to build (((((pizza)))))

Five, one for each topping.

How many *cons*es does *deep* use to build (((pizza)))

Three.

How many *cons*es does *deep* use to build pizza with a thousand toppings?

1000.

How many *cons*es does *deep* use to build all possible pizzas with at most a thousand toppings?

That's a big number:
the *cons*es of (*deep* 1000), and
the *cons*es of (*deep* 999), and
..., and
the *cons*es of (*deep* 0).

You mean 500,500?

Yes, thank you, Carl F. Gauss (1777–1855).

Yes, there is an easy way to determine this number, but we will show you the hard way. It is far more exciting.

Okay.

Guess what it is?

Can we write a function that determines it for us?

Yes, we can write the function *consC* which returns the same value as *cons* and counts how many times it sees arguments.

This is no different from writing *deepR* except that we use *add1* to build a number rather than *cons* to build a list.

```
(define consC
  (let ((N 0))
    (lambda (x y)
      (set! N (add1 N))
      (cons x y))))
```

Don't forget the imaginary name.

```
(define N1 0)
```

```
(define consC
  (lambda (x y)
    (set! N1 (add1 N1))
    (cons x y)))
```

Could we use this function to determine 500,500?

Sure, no problem.

How?

We just need to use *consC* instead of *cons* in the definition of *deep*:

```
(define deep
  (lambda (m)
    (if (zero? m)
        (quote pizza)
        (consC (deep (sub1 m))
               (quote ())))))
```

Wasn't this exciting?

Well, not really.

So let's see whether this new *deep* counts *cons*es

How about determining the value of (*deep* 5)?

That is easy; we shouldn't bother. What is the value of $\underline{N}_1$

We don't know, it is imaginary.

But that's how we count *cons*es

How could we possibly see something that is imaginary?

Here is one way.

```
(define counter)
```

```
(define consC
  (let ((N 0))
    (set! counter
      (lambda ()
        N))
    (lambda (x y)
      (set! N (add1 N))
      (cons x y))))
```

Is this as if we had written:

```
(define N_2 0)
```

```
(define counter
  (lambda ()
    N_2))
```

```
(define consC
  (lambda (x y)
    (set! N_2 (add1 N_2))
    (cons x y)))
```

But?

It changed $\underline{N}_3$ to 0.

What is the value of (*supercounter f*) where *f* is *deep*

500500.

Is this what we expected?

Yes!

It is time to see how many *cons*es are used for (*deepM* 5)

Don't we need to modify its definition so that it uses *consC*?

Of course! What are you waiting for?

```
(define deepM
  (let ((Rs (quote ()))
        (Ns (quote ())))
    (lambda (n)
      (let ((exists (find n Ns RS)))
        (if (atom? exists)
            (let ((result
                   (if (zero? n)
                       (quote pizza)
                       (consC
                         (deepM (sub1 n))
                         (quote ())))))
              (set! Rs (cons result Rs))
              (set! Ns (cons n Ns))
              result)
            exists)))))
```

How many *cons*es does *deepM* use to build (((((pizza)))))

Probably five?

What is the value of (*counter*)

500505.

Yes!

Yes, but it means we forgot to initialize with *set-counter*.

What is the value of (*set-counter* 0)	
How many *cons*es does *deepM* use to build (((((pizza)))))	Five.
What is the value of (*counter*)	5.
What is the value of (*deep* 7)	(((((((pizza))))))).
What is the value of (*counter*)	Obvious: 7.
Didn't we need to *set-counter* to 0	No, we wanted to count the number of *cons*es that were needed to build (*deepM* 5) and (*deepM* 7).
Why isn't this 12	Because that was the point of *deepM*.
What is (*supercounter f*) where *f* is *deepM*	Don't we need to initialize?
No. What is (*supercounter f*) where *f* is *deepM*	1000.
How many more *cons*es does *deep* use to return the same value as *deepM*	499,500.
"A LISP programmer knows the value of everything but the cost of nothing."	Thank you, Alan J. Perlis (1922–1990).

But we know the value of food!

((((((((((((((((more pizza))))))))))))))))
((((((((((((((more pizza))))))))))))))
((((((((((((more pizza))))))))))))
((((((((((((more pizza))))))))))))
((((((((((more pizza))))))))))
((((((((((more pizza))))))))))
((((((((more pizza))))))))
((((((((more pizza))))))))
(((((((more pizza)))))))
(((((((more pizza)))))))
((((((more pizza))))))
(((((more pizza)))))
((((more pizza))))
(((more pizza)))
((more pizza))
(more pizza)
more pizza)

What is the value of (counter)	5.
What is the value of (*set-counter* 0)	
(*rember1*C2 a l*) where *a* is noodles and *l* is ((food) more (food))	((food) more (food)), because this list does not contain noodles.
And what is the value of (*counter*)	5, because *rember1*C2* needs five *consC*s to rebuild the list ((food) more (food)).
What food are you in the mood for now?	Find a good restaurant that specializes in it and dine there tonight.