

DRAFT: The factorial function in Clarity

Part 4

Phillip G. Bradford*

November 29, 2024

Abstract

Clarity forbids recursion or unbounded iteration. So computing ‘naturally recursive’ functions in Clarity is interesting.

1 Classic Recursive Factorial

Factorials are classical. The factorial function is a classic example of recursion. For instance,

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n \cdot (n - 1)! & \text{otherwise.} \end{cases}$$

Basic factorials are $0! = 1$, $1! = 1 \times 0!$, $2! = 2 \times 1! = 2$, $3! = 3 \times 2! = 6$, $4! = 4 \times 3! = 24$, and in full detail $5!$ is

$$\begin{aligned} 5! &= 5 * 4! \\ &= 5 * 4 * 3! \\ &= 5 * 4 * 3 * 2! \\ &= 5 * 4 * 3 * 2 * 1! \\ &= 5 * 4 * 3 * 2 * 1 * 0! \\ &= 120 \end{aligned}$$

*phillip.bradford@uconn.edu, phillip.g.bradford@gmail.com, UNIVERSITY OF CONNECTICUT, SCHOOL OF COMPUTING, STAMFORD, CT USA

Many programming languages use the factorial function to illustrate recursion, see [1]. The recursive Scheme factorial function in Listing 1 cannot be directly implemented in Clarity since Clarity forbids recursion.

Listing 1: Scheme factorial function

```
(define (factorial n)
  (cond
    ((= n 0) 1)
    (else
     (* n (factorial (- n 1))))))
```

To its credit, the recursive nature of a standard factorial definition is cleanly and clearly implemented using recursion. The factorial function in Listing 1 builds a stack as seen in Listing 2. At the end when all arguments are fully evaluated, the expression `(* 5 4 3 2 1 1)` is evaluated.

Listing 2: Recursive factorial substitution

```
(factorial 5)
(* 5 (factorial 4))
(* 5 4 (factorial 3))
(* 5 4 3 (factorial 2))
(* 5 4 3 2 (factorial 1))
(* 5 4 3 2 1 (factorial 0))
(* 5 4 3 2 1 1)
120
```

Clarity has the following restrictions,

1. Recursion is illegal
2. Looping can only be done using `map`, `filter`, or `fold`.

So, how can we implement the factorial function in Clarity? Central objectives include,

1. Correctness
2. Elegance

The value of correctness is self-evident. The value of elegance comes from several places

1. Elegant things are easy to remember
2. Revisiting elegant things is a pleasure so we will do it more

Of course, efficiency plays a role as well. Efficiency is also related to elegance.

2 Factorial in Clarity

2.1 The map function

Listing 3: The Clarity map function

```
(map f lst)
```

Consider the definition,

```
(define-constant lst (list u1 u2 u3 u4 u5)).  
  
(define-private (psquare (num uint))  
  (* num num))  
  
(define-public (square-numbers (numbers (list 10 uint)))  
  (ok (map psquare numbers)))
```

The `fold` function takes three arguments. The first argument is a binary function, the second argument is an iterable item, and the last argument is the first value for the binary function application.

Listing 4: The Clarity fold function

```
(fold f lst base)
```

which means f is applied to `base` and `lst`.

```
(f base lst )
```

2.2 The filter function

```
(define-private (is-even (num uint))  
  (is-eq (mod num u2) u0))  
  
(define-public (filter-even-numbers (numbers (list 10 uint)))  
  (ok (filter is-even numbers)))
```

Executing this in the Clarinet console,

Listing 5: The Clarity filter function

```
(contract-call? .c1 filter-even-numbers (list u1 u2 u3 u4 u5))
```

2.3 The map function

Listing 6: The Clarity map function

```
(map f lst)
```

Consider the definition,

```
(define-constant lst (list 1 2 3 4 5)).
```

2.4 The map function

Listing 7: The Clarity map function

```
(map f lst)
```

Consider the definition,

```
(define-constant lst (list 1 2 3 4 5)).
```

Given 1st, how can $5!$ be computed?

Listing 8: A start of a factorial function in a Clarity contract `c1`

```
;;  
;;  
;;  
;;  
(define-constant lst (list 1 2 3 4 5))  
;;  
;;  
;;  
(define-public (factorial-5 (v int))  
  ;;
```

For example, we desire,

```
>> (contract-call? .c1 factorial-5 1)
>> (ok 120)
```

A version without an argument is,

```
(define-public (fact-1)
  ;;
```

Also giving,

```
>> (contract-call? .c1 fact-1)
>> (ok 120)
```

3 Exercises

1. The Fibonacci sequence is

$$f(n) = \begin{cases} f(n-1) + f(n-2) & \text{if } n > 1, \\ 0 & \text{if } n \leq 1. \end{cases}$$

How can we generate elements of the Fibonacci sequence in Clarity?

2. A recursive function may have k different recursive invocations in its definition. For example, the code in Listing 1 has one recursive invocation in its definition. The Fibonacci function has two recursive invocations in its definition.

In Clarity, how might we code a function that has k recursive invocation in its definition?

References

- [1] Harold Abelson, Gerald Jay Sussman, with Julie Sussman: *Structure and Interpretation of Computer Programs*, Second Edition, MIT Press, 1996.
- [2] Clarity Book, <https://book.clarity-lang.org/> 2023-04-11.