Compound Data and Data Abstraction

The Game of Nim

Rules:

- There are two piles of items.
- Players take turns.
- You take as many as you want from a single pile.
- The player who takes the last item wins.

```
(define play-with-turns
  (lambda (game-state player)
    (display-game-state game-state)
    (cond ((over? game-state)
           (announce-winner player))
          ((equal? player 'human)
           (play-with-turns (human-move game-state) 'computer))
          ((equal? player 'computer)
           (play-with-turns (computer-move game-state) 'human))
          (else
           (error "player wasn't human or computer:" player)))))
```

```
(define computer-move
  (lambda (game-state)
    (let ((pile (if (> (size-of-pile game-state 1) 0)
                    1
                    2)))
      (display "I take 1 coin from pile ")
      (display pile)
      (newline)
      (remove-coins-from-pile game-state 1 pile))))
```

```
(define prompt
  (lambda (prompt-string)
    (newline)
    (display prompt-string)
    (newline)
    (read)))

(define human-move
  (lambda (game-state)
    (let ((p (prompt "Which pile will you remove from?")))
      (let ((n (prompt "How many coins do you want to remove?")))
        (remove-coins-from-pile game-state n p)))))
```

```
(define over?
  (lambda (game-state)
    (= (total-size game-state) 0)))

(define announce-winner
  (lambda (player)
    (if (equal? player 'human)
        (display "You lose. Better luck next time.")
        (display "You win. Congratulations."))))
```

```
(define remove-coins-from-pile
  (lambda (game-state num-coins pile-number)
    (if (= pile-number 1)
        (make-game-state (- (size-of-pile game-state 1)
                            num-coins)
                         (size-of-pile game-state 2))
        (make-game-state (size-of-pile game-state 1)
                         (- (size-of-pile game-state 2)
                            num-coins)))))
```

```
;; Implementation
;; The state is a two-digit integer whose 10's digit
;; is the number of items in the first pile and 1's
;; digit is the number of items in the second pile.
;; Assumes no more than 9 coins per pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
    (+ (* 10 n) m)))

(define size-of-pile
  ;Returns an integer equal to the number of coins in
  ;pile pile-number of the game-state.
  (lambda (game-state pile-number)
    (if (= pile-number 1)
        (quotient game-state 10)
        (remainder game-state 10))))
```

```
;; Utilities

(define display-game-state
  (lambda (game-state)
    (newline)
    (newline)
    (display "    Pile 1: ")
    (display (size-of-pile game-state 1))
    (newline)
    (display "    Pile 2: ")
    (display (size-of-pile game-state 2))
    (newline)
    (newline)))

(define total-size
  (lambda (game-state)
    (+ (size-of-pile game-state 1)
       (size-of-pile game-state 2))))
```

Alternate implementations

Alternate implementations depend only on redefining `make-game-state` and `size-of-pile`.

Alternate implementation

The game state is a single integer of the form $2^n \times 3^m$.

Alternate implementation

The game state is a single integer of the form $2^n \times 3^m$.

To make the game state use the `expt` function.

Alternate implementation

The game state is a single integer of the form $2^n \times 3^m$.

To make the game state use the `expt` function.

Problem: How do you get the size of each pile from a single number?

Alternate implementation

The game state is a single integer of the form $2^n \times 3^m$.

To make the game state use the `expt` function.

Problem: How do you get the size of each pile from a single number?

Example: If the game state is the single number 648 and you know that $648 = 2^n \cdot 3^m$ how do you compute n and m?

Successively divide by 2.

648 even

Successively divide by 2.

648 even

324 even

Successively divide by 2.

648	even
324	even
162	even

Successively divide by 2.

648 even

324 even

162 even

81 not even

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

648 divisible by 3

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

648	divisible by 3
216	divisible by 3

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

648 divisible by 3

216 divisible by 3

72 divisible by 3

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

648 divisible by 3

216 divisible by 3

72 divisible by 3

24 divisible by 3

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

648 divisible by 3

216 divisible by 3

72 divisible by 3

24 divisible by 3

8 not divisible by 3

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

$$\left.\begin{array}{ll} 648 & \text{divisible by 3} \\ 216 & \text{divisible by 3} \\ 72 & \text{divisible by 3} \\ 24 & \text{divisible by 3} \end{array}\right\} n = 4$$

8 not divisible by 3

Successively divide by 2.

$$\left.\begin{array}{ll} 648 & \text{even} \\ 324 & \text{even} \\ 162 & \text{even} \end{array}\right\} n = 3$$

81 not even

Successively divide by 3.

$$\left.\begin{array}{ll} 648 & \text{divisible by 3} \\ 216 & \text{divisible by 3} \\ 72 & \text{divisible by 3} \\ 24 & \text{divisible by 3} \end{array}\right\} n = 4$$

8 not divisible by 3

So, `(exponent-of-in 2 648)` should return 3, and `(exponent-of-in 3 648)` should return 4.

Exercise for the student

For a three-pile game of Nim, the game state can be a single digit of the form $2^n \times 3^m \times 5^k$

The procedure implementation

The game state is a function with one parameter, the pile number, that returns the number of items on that pile.

The procedure implementation

The game state is a function with one parameter, the pile number, that returns the number of items on that pile.

Example:

Pile 1: 3 items

Pile 2: 4 items

`(game-state 1)` should return 3

`(game-state 2)` should return 4

`(game-state 1)` should return 3

`(game-state 2)` should return 4

`(game-state 1)` should return 3

`(game-state 2)` should return 4

How would you program `game-state` directly?

`(game-state 1)` should return 3

`(game-state 2)` should return 4

How would you program `game-state` directly?

```
(define game-state
```

`(game-state 1)` should return 3

`(game-state 2)` should return 4

How would you program `game-state` directly?

```
(define game-state
  (lambda (x)
```

`(game-state 1)` should return 3

`(game-state 2)` should return 4

How would you program `game-state` directly?

```
(define game-state
  (lambda (x)
    (if (odd? x) 3 4)))
```

`(game-state 1)` should return 3

`(game-state 2)` should return 4

How would you program `game-state` directly?

```
(define game-state
  (lambda (x)
    (if (odd? x) 3 4)))
```

Function `make-game-state` is a factory!

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
    (lambda (x)
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
    (lambda (x)
      (if (odd? x)
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
    (lambda (x)
      (if (odd? x)
          n
          m))))
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
    (lambda (x)
      (if (odd? x)
          n
          m))))

(define size-of-pile
  ;Returns an integer equal to the number of coins in
  ;pile pile-number of the game-state.
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
    (lambda (x)
      (if (odd? x)
          n
          m))))

(define size-of-pile
  ;Returns an integer equal to the number of coins in
  ;pile pile-number of the game-state.
  (lambda (game-state pile-number)
```

```
;; Implementation
;; The state is a function of one parameter, the pile
;; number, that returns the number of items on that pile.

(define make-game-state
  ;Returns a game state with n coins in the first pile
  ;and m coins in the second pile.
  (lambda (n m)
    (lambda (x)
      (if (odd? x)
          n
          m))))

(define size-of-pile
  ;Returns an integer equal to the number of coins in
  ;pile pile-number of the game-state.
  (lambda (game-state pile-number)
    (game-state pile-number)))
```