Abstraction, List, & Cons

CS61A Lecture 7

2011-06-29
Colleen Lewis

Very sad code

(define (total hand)
  (if (empty? hand)
      0
      (+ (butlast (last hand))
          (total (butlast hand))))
STk> (total '(3h 10c 4d))
  17
STk> (total '(3h ks 4d))
EEEK!

Happier Code

(define (total hand)
  (if (empty? hand)
      0
      (+ (rank (one-card hand))
          (total (remaining-cards hand)))))
(define (rank card) (butlast card))
(define (suit card) (last card))
(define suit last)
(define (one-card hand) (last hand))
(define (remaining-cards hand) (butlast hand))

Selectors

Goals

• To talk about things using meaning not how it is represented in the computer
• To be able to change how it is represented in the computer without people who use our program caring
• Invented by: Turing Award Winner: Barbara Liskov

Constructors

• GOAL: To talk about things using meaning not how it is represented in the computer
You still have to teach people to use your program

Constructors

STk> (total
    (make-hand
        (make-card 3 'heart)
        (make-card 10 'club)
        (make-card 4 'diamond)))
(define (make-card rank suit)
  (word rank (first suit)))
(define make-hand se)
Data Abstraction 😊

```
(define (total hand)
  (if (empty? hand)
      0
      (+ (rank (one-card hand))
          (total (remaining-cards hand)))))
```

```
(define (rank card)
  (butlast card))

(define (suit card)
  (last card))

(define (one-card hand)
  (last hand))

(define (remaining-cards hand)
  (bl hand))

(define (make-card rank suit)
  (word rank (first suit)))

(define make-hand se)
```

Try It!

Rewrite what you need to:

- Cards are represented as numbers 1-52
  - 1-13 is A-K of Hearts
  - 14-26 is A-K of Spades
  - 27-39 is A-K of Diamonds
  - 40-52 is A-K of Clubs

Runtimes Continued

Exponential Runtime

\[ 2^n \]

Fibonacci

\[ \text{fib}_n = \text{fib}_{n-1} + \text{fib}_{n-2} \]

\[ n \text{ branches} = 2^{b+1} - 1 \text{ calls} \]
Logarithmic Runtime

Log₂(N)

Number Guessing Game

• I’m thinking of a number between 1 and 100
• How many possible guesses could it take you? (WORST CASE)

• Between 1 and 10000000?
• How many possible guesses could it take you? (WORST CASE)

Divide and Conquer

• If we can divide the problem up in half each time
  – like the number guessing game
• How many recursive calls will it take?

n is the original problem size
if h calls then:

\[ n = 2^{h+1} - 1 \]

Log₂(N)

• When we’re able to keep dividing the problem in half (or thirds etc.)
• Looking through a phone book
Asymptotic Cost

- We want to express the speed of an algorithm independently of a specific implementation on a specific machine.
- We examine the cost of the algorithms for large input sets i.e. the asymptotic cost.
- In later classes (CS70/CS170) you’ll do this in more detail

Which is fastest after some value $N$?

- $time = \sqrt{n} - 1000$
- $time = 0.75\sqrt{n} + 50$

**WAIT – who cares?**

<table>
<thead>
<tr>
<th>Function</th>
<th>Common Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Constant</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>Logarithmic</td>
</tr>
<tr>
<td>$O(\log^2 n)$</td>
<td>Log-squared</td>
</tr>
<tr>
<td>$O(\sqrt{n})$</td>
<td>Root-n</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>Linear</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Quadratic</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>Cubic</td>
</tr>
<tr>
<td>$O(n^4)$</td>
<td>Quartic</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>Exponential</td>
</tr>
<tr>
<td>$O(e^n)$</td>
<td>Bigger exponential</td>
</tr>
</tbody>
</table>

Important Big-Oh Sets

*Formal definition*

$$T(n) \in O(f(n))$$

if and only if

$$T(n) \leq c \cdot f(n)$$

for all $n > N$
Simplifying stuff is important

\[ f(n) \in O(5n^3 + 10n^2 + 1000n) \]

\[ T(n) \in O(n^3) \]

Cons and Lists

**cons**

STk> (cons 1 2)

(1 . 2)

STk> (define a (cons 1 2))

a

STk> (define b (cons 'hi 'bye))

b

STk> b

(hi . bye)

**car / cdr**

STk> (car a)

1

STk> (cdr a)

2

STk> (car b)

hi

STk> (cdr b)

bye

Data Abstraction 😊

Pairs

<table>
<thead>
<tr>
<th>car</th>
<th>cdr</th>
</tr>
</thead>
</table>

Selectors

car
cdr

Constructors

cons

Uses of Pair from textbook

- Points
- Intervals
- Fractions
- Complex #
- Lines
Lists

Demo

Lists are made with pairs!

STk> (define a (list 1 2 3 4))

STk> (define b (list 1 2))

The Empty List

STk> (cons 2 `())

How can you make the list (1 2)?

a) (define a (cons 1 2 `()))
b) (define a (cons 1 (cons 2)))
c) (define a (cons 1 (cons 2 `())))
d) (define a (cons (cons 2 `()) 1)))
e) ???

How many calls to cons are made?

STk> (define a (list 1 2 3 4))
How many calls to cons are made?

STk> (define a (list 1 2 (list 3 4) 5))

A) 2  B) 3  C) 4  D) 5  E) 6

Accessing Elements

Using car and cdr

The Empty List w/ car & cdr

STk> (define x (cons 2 '()))

x

STk> x

(2)

STk> (car x)

2

STk> (cdr x)

()

How do you get the 2?

STk> (define a (list 1 2 3 4))

1

2

3

4

A) (car (cdr a))
B) (cdr (car a))
C) (cdr (cdr (car a)))
D) (car (cdr (cdr a)))
E) (cdr (car (car a)))

How do you get the 3?

STk> (define a (list 1 2 (list 3 4) 5))

1

2

3

4

5

A) (car (car (cdr (cdr a)))))
B) (cdr (car (car a)))))
C) (cdr (car (cdr (car a))))
D) (car (cdr (car (cdr a))))
E) ???

Cons makes a pair

(cons a b)
Dots

Demo

STk> (cons 1 2)
   (1 . 2)

STk> (cons 1 '())
   (1)

STk> (cons 1 '())
   (1 . ())

CONSTRUCTOR SOLUTION

(define (make-card rank suit)
  (cond
   ((equal? suit 'heart) rank)
   ((equal? suit 'spade) (+ rank 13))
   ((equal? suit 'diamond) (+ rank 26))
   ((equal? suit 'club) (+ rank 39))
   (else (error "say what?")) ))

SELECTOR SOLUTION

(define (card-rank card)
  (remainder card 13))

(define (suit card)
  (cond
   ((> 14 card) 'heart)
   ((> 27 card) 'spade)
   ((> 40 card) 'diamond)
   (else 'club)))

How many calls to cons are made?

STk> (define a (list 1 2 (list 3 4) 5))

A) 2   B) 3   C) 4   D) 5   E) 6

Solution: (cons 1 (cons 2 (cons (cons 3 (cons 4 '())) (cons 5 '()))))