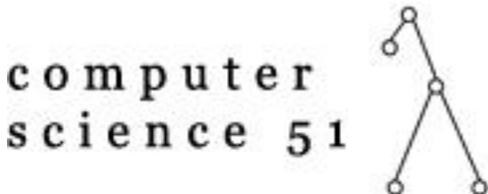
COMPUTER SCIENCE 51

Spring 2009

cs51.seas.harvard.edu

Prof. Greg Morrisett

Prof. Ramin Zabih



Logistical minutia

- We'll take a 5-minute break mid-class
  - Due to popular demand!
  - After break: Blackadder excerpt
- Grading policy questions are the purview of the TF's
  - Even if Greg or I contradict them
- Project 1 released shortly
  - Checkpoints each Tuesday
  - Choose a partner by Friday 11:59PM

Today: program design

- Hierarchical design of systems
  - In CS and outside of it
  - "Recursive decomposition"
- Contract versus implementation
  - What is computed, as opposed to how
- Proper design of datastructures

Basic idea

- Break your code into pieces, where each piece has a contract/spec
  - Exposes certain functions
  - **Invariant**: how the functions behave
  - Everything else about the code is hidden
- This sounds like a platitude
  - Let's demo its importance
    - See: Project1a
  - Modularity is like oxygen or gravity
    - Appreciated primarily in its absence

Point "contract"

- Constructor: `(make-point x y)`
- Accessors: `(point-x p)` `(point-y p)`
- Predicate: `(point? p)`
- **Invariant**: if we first evaluate

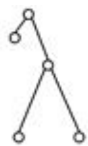
  `(define p (make-point x y))`

- Then:
  - `(point? p)` ⇓ `true`
  - `(point-x p)` ⇓ `x`
  - `(point-y p)` ⇓ `y`
- Else: `(point? p)` ⇓ `false`

A nightmare scenario

- Gideon and Victor implement the contract well
- Greg and Ramin abuse it badly
  - And suffer the consequences!

    (Exiled to Redmond?)

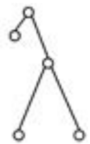


A good start

- Gideon and Victor use lists:

```
(define (make-point x y) (list x y))
(define (point? p)
  (cond [(not (list? p)) false]
        [else (> (length p) 1)]))
```

- Greg and Ramin build code using this initial implementation
  - Only calling functions in the contract
  - I.e., doing what they should



Greg and Ramin go wrong

- They notice points are actually lists
  - This allows them to do “clever” things
  - Find points on y-axis, given a list of points

```
(define (x-is-zero pts)
  (cond [(empty? pts) pts]
        [(= (car (car pts)) 0)
         (cons (car pts)
          (x-is-zero (cdr pts)))]
        [else (x-is-zero (cdr pts))]))
```

  - What’s the key error in this code?

Greg and Ramin go wild

- More fun can be had!
  - Is any point on x or y axis?

```
(define (squish pts)
  (if (empty? pts)
      empty
      (append (car pts)
              (squish (cdr pts)))))
(define (any-border pts)
  (member 0 (squish pts)))
```

Fun comes to an end

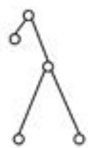
- Gideon and Victor upgrade the implementation
  - While still obeying the contract
- For example, they notice that their implementation of point? claims that lots of things are points
  - Examples?
- Real goal: catch bugs **early**!!

Defensive programming

```
(define (make-point x y)
  (list "Pointy!" x y))
; Never trust your caller – could be a prof!
(define (point? p)
  (cond [(not (list? p)) false]
        [(not (= (length p) 3)) false]
        [else (equal? (car p) "Pointy!")]))
```

Consequences

- What happens to Greg and Ramin's code when they install the upgrade?
- More frighteningly, what happens to another program that uses Greg and Ramin's code??
  - More or less what happens when you run a WinXP program under Vista
  - Or when you upgrade to MacOS X



Recall: structs

- Given:
  (**define-struct** *name* (x_1 x_2 … x_n))
- Records of values are values.
  - (make-*name* v_1 v_2 … v_n) is a value.
- **Invariant**:
  - (*name*? (make-*name* v_1 v_2 … v_n)) $\Downarrow$ true.
  - (*name*? *v*) $\Downarrow$ false for all other values.
  - (*name*-x_i (make-*name* v_1 v_2 … v_n)) $\Downarrow$ v_i

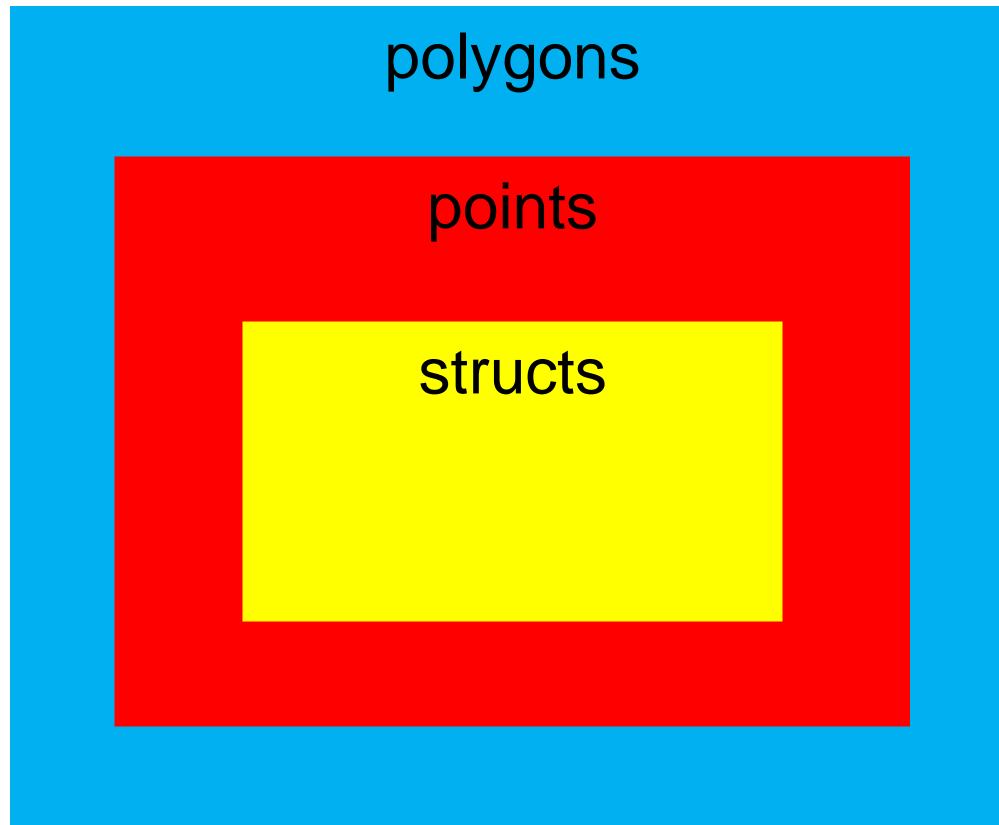
An even worse case

- Gideon and Victor move to structs
  - Sound predicate
  - Information hiding
  - One-line implementation
- What happens to Greg and Ramin?
  - Answer: nothing good!
- Suppose they also switched to polar coordinates?

Break time!

- Thought question:
  - Why does this nightmare happen so incredibly often?

Program design



Stepping back

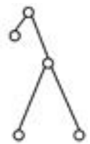
- Why do programmers often behave like Greg and Ramin did
  - Breaking the abstraction barrier
  - Even though they often "know better"?
- Progam time is less valuable than programmer time
  - Moore's law does not apply to coders
- Breaking abstraction barriers is often driven by focus on the common cases

Optimize the common case

http://www.youtube.com/watch?v=4rPbXHvsLbI

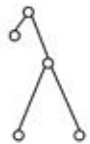
Many famous failures

- History of Windows is full of them
  - Originally there wasn't really a contract
  - Programmers took advantage of accidental implementation properties
    - Compatibility nightmare (still!)
- Intel also has had its share
- Most software "meltdowns" occur due to bad design decisions
  - Not, e.g., lack of smart coders



Proper design

- Divide problem into the right pieces
  - Solve them independently
  - Communicate only via interfaces
- Ideally you can “hot swap”
- Examples?
  - Any plug (USB, wall socket)
    - Any standard/API is a contract!
  - Networking software and hardware



A pretty example