UMass Boston Computer Science

CS450 High Level Languages

Compound Data Definitions

Tuesday, February 18, 2025

```
class Circle {
  Num radius;
  Color col;
}
```

Logistics

- HW 2 in
  - ~~due: Tues 2/18, 11am EST~~
  - Files should not start `big-bang` loop automatically!

- HW 3 out
  - due: Tues 2/25 11am EST
  - Add keyboard input handler

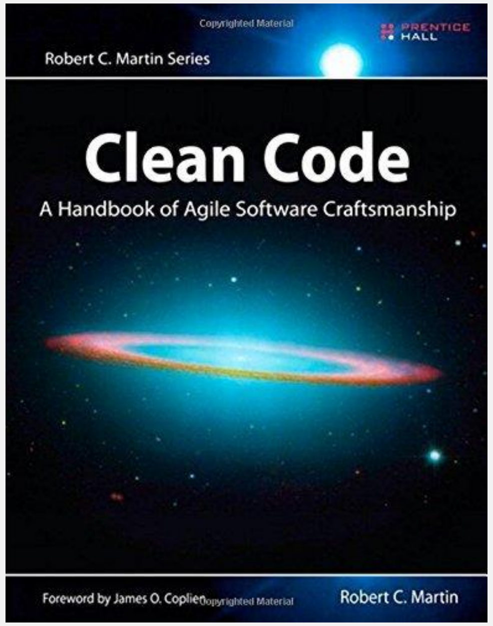
HW Advice

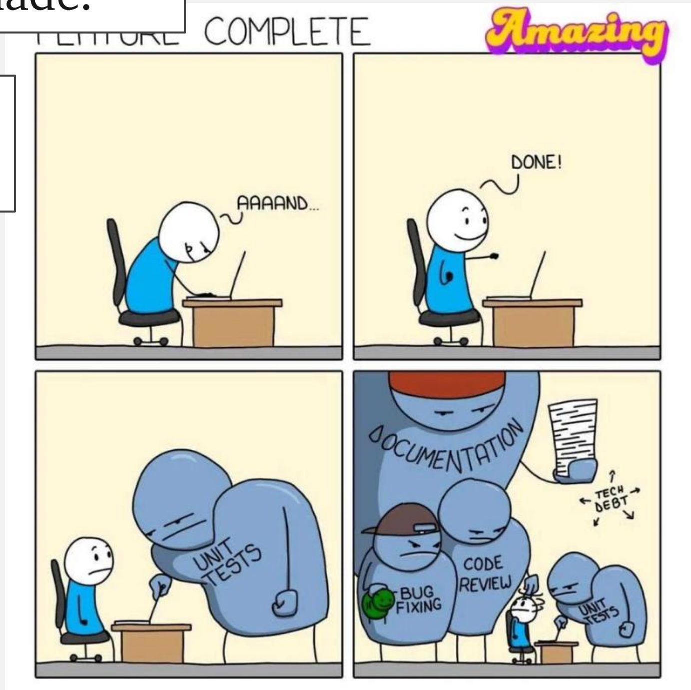
“Perhaps you thought that “**getting it working**” was the first order of business for a professional developer.

I hope by now, however, that this book has disabused you of that idea.

The functionality that you create today has a good chance of changing in the next release, but the **readability of your code** will have a profound effect on all the changes that will ever be made.”

— Robert C. Martin,
Clean Code: A Handbook of Agile Software Craftsmanship





HW Observations

- Not ok to submit
  - my code
  - Code that doesn’t (or hasn’t been) run
  - Failing / erroring tests
  - Code that doesn’t match Github (???)

- See: Incremental Programming Pledge!

Last Time

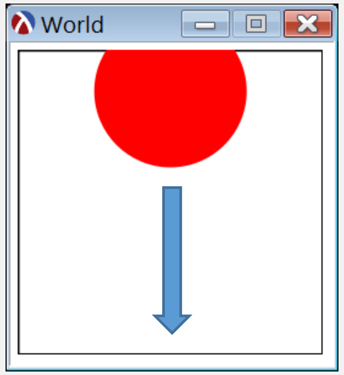
Kinds of Data Definitions

- Basic data
  - E.g., **numbers, strings,** etc
- Intervals
  - Data that is from a **range of values,** e.g., [0, 100)
- Enumerations
  - Data that is **one of a list of possible values**, e.g., `“green”`, `“red”`, `“yellow”`
- Itemizations
  - Data value that can be from a **list of possible other data definitions**
  - E.g., **either** a **string** or **number** (Generalizes enumerations)

Last Time

Falling "Ball" Example

```
;; A WorldState is a Non-negative Integer
;; Interp: Represents the y Coordinate of the center of a
;;          ball in a `big-bang` animation.
```



What if the **ball can also move side-to-side?**

`WorldState` would need <u>two</u> **pieces of data:**
the **x** *and* **y coordinates**

```
;; A WorldState is an Integer ...
;; ... and another Integer???
```

We need a way to create **compound data**
i.e., a **data definition** that
<u>combines</u> <u>values</u> of other data defs

Last Time

Kinds of Data Definitions

- Basic data
  - E.g., **numbers, strings,** etc
- Intervals
  - Data that is from a **range of values,** e.g., [0, 100)
- Enumerations
  - Data that is **one of a list of possible values**, e.g., "green", "red", "yellow"
- Itemizations
  - Data value that can be from a **list of possible other data definitions**
  - E.g., **either** a **string** or **number** (Generalizes enumerations)
- **Compound Data**
  - Data that is a **combination of values from other data definitions**

today

Falling "Ball" Example

???

```
;; A WorldState is a (mk-WorldState [x : Int] [y : Int])
;; Represents coordinate in big-bang animation where:
;; - x is ball (red solid circle) horizontal center
;; - y is ball vertical center
(struct world [x y] #:transparent)
(define/contract (mk-WorldState x y)
  (-> integer? integer? WorldState?)
  (world x y)
;; ...
```

a `struct` defines a new kind of **compound data**

Parts of a **`struct`** definition

field names

name

(needed for testing)

```
(struct world [x y] #:transparent)
```

Same as "name"

<u>(Implicitly) defines:</u>

- A **constructor** function → `world`
  - Creates instances of the struct
- **Accessor** functions → `world-x, world-y`
  - Get an instance's field value
- A **predicate** → `world?`
  - Returns true for struct instances

"name" + "-" + …

… field names

"name" + "?"

Falling "Ball" Example

```
;; A WorldState is a (mk-WorldState [x : Int] [y : Int])
;; Represents coordinate in big-bang animation where:
;; - x is ball (red solid circle) horizontal center
;; - y is ball vertical center
(struct world [x y] #:transparent)
(define/contract (mk-WorldState x y)
  (-> integer? integer? WorldState?)
  (world x y)
;; ...
```

a `struct` defines a new kind of **compound data**

Checked constructor (programmer must define)

Unchecked (internal) **constructor** (implicitly defined by **`struct`**)

```
(define INIT-WORLDSTATE (mk-WorldState 0 0))
```

Instances of the `struct` are values of that kind of data

Data Design Recipe

Data Definition

- Has **4** parts:
    1. **Name**
    2. Description of **all possible values** of the data
    3. **Interpretation** explaining the real world concepts the data represents
    4. **Predicate** returning `false` if given value is <u>not</u> in the Data Definition
        - If needed, define extra predicates for each **enumeration** or **itemization**

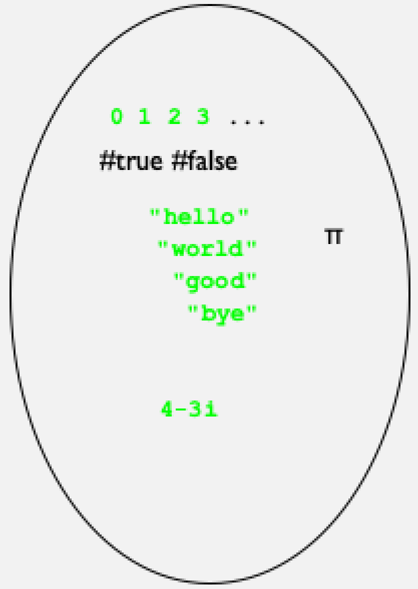
Data Design Recipe - Compound Data Update

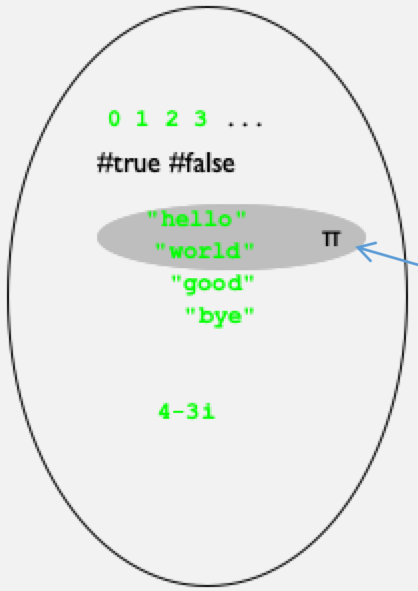
Data Definition

- Has ~~4~~ **maybe 5** parts:
  1. **Name**
  2. Description of **all possible values** of the data
  3. **Interpretation** explaining the real world concepts the data represents
  4. **Predicate** returning `false` if given value is <u>not</u> in the Data Definition
     - If needed, **define extra predicates for each enumeration** or **itemization**
  5. (checked) **Constructor** for compound data def values

Interlude: Data Definitions (ch 5.7)

All possible data values





A **data definition**
= (a named) subset of all possible values

We are **defining** (and naming) **the <u>valid</u> data values <u>our program</u>!**

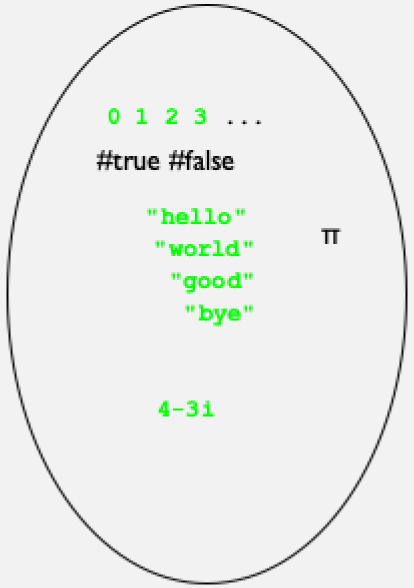
All programs manipulate some set of data values …

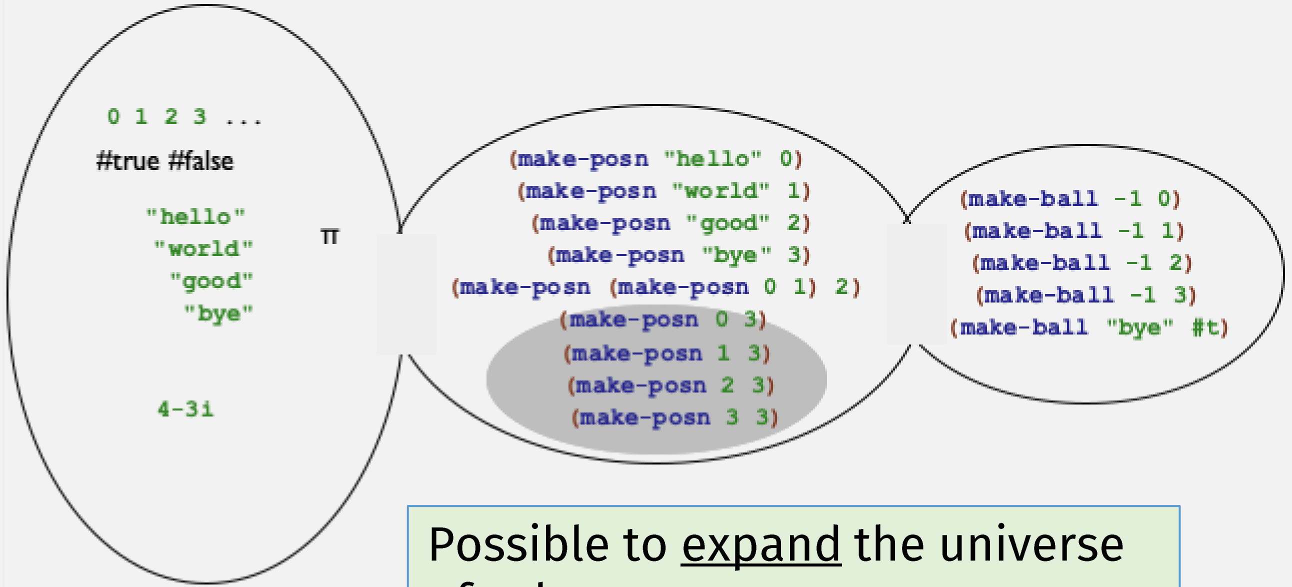
So this must be the <u>first step</u> of programming!

Also makes "error handling" easy

Interlude: Data Definitions (ch 5.7)

All possible basic data values





Possible to expand the universe of values, e.g., new **compound data definitions** (`struct`, or other data structure)

Predicates for Compound Data

```
;; A WorldState is a (mk-WorldState [x : Int] [y : Int])
;; Represents coordinate in big-bang animation where:
;; - x is ball (red solid circle) horizontal center
;; - y is ball vertical center
(struct world [x y] #:transparent)
```

<u>Compound data predicates</u> should be **"shallow"** checks, i.e., `world?`

predicate?

struct already defines `world?`, what about **fields?**

```
(define (WorldState? arg)
  (and (world? arg)
       (integer? (world-x arg))
       (integer? (world-y arg))))
```

???

This "deep" predicate <u>checks too much</u> ...

... because it's the **job of "field data type" processing functions to check those kinds of data**

also, **maybe exponential overhead ...**

Checked constructor ensures that <u>**only valid instances may be created**</u>!

```
(define/contract (mk-WorldState x y)
  (-> integer? integer? WorldState?)
  (world x y))
```

Data Design Recipe - Predicate Update

Data Definition

- Has **maybe 5** parts:
  1. **Name**
  2. Description of **all possible values** of the data
  3. **Interpretation** explaining the real world concepts the data represents
  4. **Predicate**
     - Evaluates to `true` for some values in the Data Definition
       - False positives **ok**
     - Evaluates to `false` for some values not in the Data definition
       - False negatives **not ok**
  5. (checked) **Constructor** for compound data def values

Last Time

Function Design Recipe

1. **Name**

2. **Signature** – types of the function input(s) and output

3. **Description** – explain (in English prose) the function behavior

4. **Examples** – show (using `rackunit`) the function behavior

5. **Code** – implement the rest of the function (arithmetic)

6. **Tests** – check (using `rackunit`) the function behavior

Last Time

Function Design Recipe

1. **Name**
2. **Signature** – types of the function input(s) and output
3. **Description** – explain (in English prose) the function behavior
4. **Examples** – show (using `rackunit`) the function behavior
5. **Template** – sketch out the function structure (using input's Data Definition)
6. **Code** – implement the rest of the function (arithmetic)
7. **Tests** – check (using `rackunit`) the function behavior

Functions For Compound Data

- A **function that processes compound data** must
  - **extract the individual pieces,** using accessors
  - **combine them,** with arithmetic

Functions For Compound Data - Template

- A **function that processes compound data** must
  - **extract the individual pieces,** using accessors ← Done with template
  - **combine them,** with arithmetic

```
;; A WorldState is a (mk-WorldState [x : Int] [y : Int])
;; Represents coordinate in big-bang animation where:
;; - x is ball (red solid circle) horizontal center
;; - y is ball vertical center
(struct world [x y] #:transparent)
```

```
;; TEMPLATE for WorldState-fn: WorldState -> ???
(define        (WorldState-fn w)

  .... (world-x w) ....
  .... (world-y w) .... )
```

A **function's template** is completely **determined by** the input's **Data Definition**

Functions For Compound Data - Template

- A **function that processes compound data** must
  - **<u>extract</u> the individual pieces,** using accessors ← Done with template
  - **<u>combine</u> them,** with arithmetic

A **function's template** is completely **determined by the input's Data Definition**

```
;; A WorldState is a (mk-WorldState [x : Int] [y : Int])
;; Represents coordinate in big-bang animation where:
;; - x is ball (red solid circle) horizontal center
;; - y is ball vertical center
(struct world [x y] #:transparent)
```

```
;; TEMPLATE for WorldState-fn: WorldState -> ???
(define/contract (WorldState-fn w)
  (-> WorldState? ??? )
  .... (world-x w) ....
  .... (world-y w) .... )
```

Signatures / Contracts Redundant?

Redundant?

```
;; TEMPLATE for WorldState-fn: WorldState -> ???
(defin Redundant? (WorldState-fn w)
  (-> WorldState? ??? )
  .... (world-x w) ....
  .... (world-y w) .... )
```

Function Design Recipe - Signature / Contract Update

Submitted code no longer needs both Signature and Contract

- The **Contract is the Signature!**
- This assumes:
  - Contract predicates represent valid Data Definitions!
- **NOTE – this does not change the Design Recipe!**
  - Only submission requirements

```
;; TEMPLATE for WorldState-fn~~: WorldState -> ???~~
(define/contract (WorldState-fn w)
  (-> WorldState? ??? )
  .... (world-x w) ....
  .... (world-y w) .... )
```

Function Design Recipe

Still must program with these steps, in this order!

1. **Name**
2. **Signature** – types of the **function input(s)** and **output**

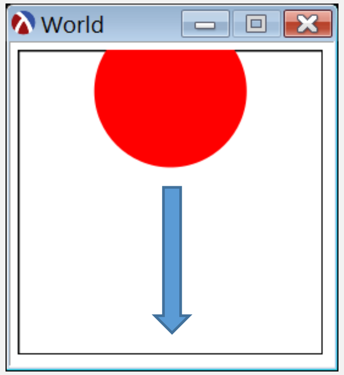
   (not submitted in comments, if there are valid contracts)

3. **Description** – explain (in **English prose**) the **function behavior**
4. **Examples** – show (using `rackunit`) the **function behavior**

   (not submitted)

5. **Template** – sketch out the **function structure** (using input's **Data Definition**)
6. **Code** – implement the **rest of the function** (arithmetic)
7. **Tests** – check (using `rackunit`) the **function behavior**

Falling "Ball" Example



What if the **ball can also move side-to-side?**

`WorldState` would need **two pieces of data**: the **x** *and* **y coordinates**

```
;; A WorldState is a (mk-WorldState [x : Int] [y : Int])
;; Represents coordinate in big-bang animation where:
;; - x is ball (red solid circle) horizontal center
;; - y is ball vertical center
```

```
(check-equal?
  (next-WorldState
    (mk-WorldState 0 0))
  (mk-WorldState X-VEL Y-VEL))
```

```
;; next-WorldState : WorldState -> WorldState
;; Computes the ball position after 1 tick
```

```
;; TEMPLATE for WorldState-fn: WorldState -> ???
(define/contract (WorldState-fn w)
  (-> WorldState? ??? )
  .... (world-x w) ....
  .... (world-y w) .... )
```

```
(check-equal?
  (next-WorldState
    (mk-WorldState 0 0))
  (mk-WorldState X-VEL Y-VEL))
```

```
;; next-WorldState : WorldState -> WorldState
;; Computes the ball position after 1 tick
```

```
(define/contract (next-WorldState w)
  (-> WorldState? WorldState?)
  .... (world-x w) ....
  .... (world-y w) .... )
```

```
(check-equal?
  (next-WorldState
    (mk-WorldState 0 0))
  (mk-WorldState X-VEL Y-VEL))
```

```
;; next-WorldState : WorldState -> WorldState
;; Computes the ball position after 1 tick
```

```
(define/contract (next-WorldState w)
  (-> WorldState? WorldState?)
  (mk-WorldState
    (+ (world-x w) X-VEL)
    (+ (world-y w) Y-VEL)))
```

Extract Compound Pieces – `let`

```
(define/contract (next-WorldState w)
  ; ...
  (let ([x (world-x w)]
        [y (world-y w)])
    (mk-WorldState (+ x X-VEL) (+ y Y-VEL))))
```

```
(let ([id val-expr] ...) body ...+)
```

Extract all compound data pieces first, before doing “arithmetic”

Defines new local variables

in scope only in the body

Local variables **shadow** previously defined vars

Extract Compound Pieces – (internal) **define**

```
(define/contract (next-WorldState w)
  ; …
  (define x (world-x w))
  (define y (world-y w))
  (mk-WorldState (+ x X-VEL) (+ y Y-VE
```

Extract all compound data pieces first, before doing "arithmetic"

(is there an easier way to do this?)

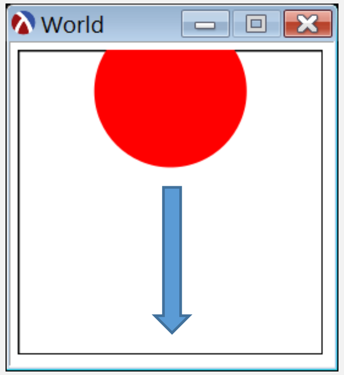
Extract Compound Pieces – Pattern Match!

```
(define/contract (next-WorldState w)
  ; ...
  (match-define (world x y) w)

  (mk-WorldState (+ x X-VEL) (+ y Y-VEL))))
```

Extract all compound data pieces, at the same time!

Falling “Ball” Example



What if the **ball can also move side-to-side**?

on a key-press?

`WorldState` would need <u>**two**</u> **pieces of data**:
the **x** *and* **y coordinates**

Last Time

KeyEvent Enumeration (predefined)

```
; A KeyEvent is one of:
; - 1String
; - "left"
; - "right"
; - "up"
; - ...
```

“Key event fn”

(result must be **WorldState**)

But remember:

1 function does
1 task which processes
1 kind of data

```
; WorldState KeyEvent -> ..
(define (handle-key-events w ke)
  (cond
    [(= (string-length ke) 1) ...]
    [(string=? "left" ke) .. (handle-left w) ???
    [(string=? "right" ke) . (handle-right w) ???
    [(string=? "up" ke) ...]
    [(string=? "down" ke) ...]
    ...))
```

WorldState

Template

Give to: **big-bang on-key** clause

Must call separate: (WorldState-fn w)

```
; A 1String is a String of length 1,
; including
; - "\\" (the backslash),
; - " " (the space bar),
; - "\t" (tab),
; - "\r" (return), and
; - "\b" (backspace).
; interpretation represents keys on the keyboard
```

In-class exercise 2/18

on gradescope