

Programming With Data



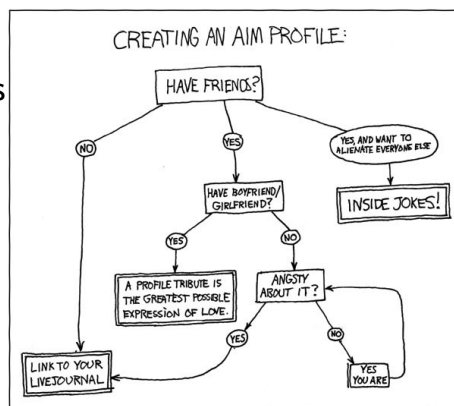
#1

One-Slide Summary

- A list is a **data structure**, a way of storing and organizing data.
- **cons** creates a **pair** of two values.
- **car** and **cdr** (or **first** and **rest**) extract the first and second elements of a cons pair.
- A **list** is a **recursive data structure**. A list is *either* empty (called **null**) or a cons pair where the second element is a list.
- A **recursive function** has a simple **base case** and a **recursive case** (where it calls itself).

#2

Outline



#3

Problem Set 1

- Colors and photomosaics, oh my!
- Comments?



The Patented RGB RMS Method

```

/* This is a variation of RGB RMS error. The final square-root has been eliminated to */
/* speed up the process. We can do this because we only care about relative error. */
/* HSV RMS error or other matching systems could be used here, as long as the goal of */
/* finding source images that are visually similar to the portion of the target image */
/* under consideration is met. */
for(i = 0; i < size; i++) {
    rt = (int) ((unsigned char)rmas[i] - (unsigned char)image->r[i]);
    gt = (int) ((unsigned char)gmas[i] - (unsigned char)image->g[i]);
    bt = (int) ((unsigned char)bmas[i] - (unsigned char)image->b[i]);
    result += (rt*rt + gt*gt + bt*bt);
}

```

Your code should never look like this! Use **new lines** and **indenting** to make it easy to understand the structure of your code! (Note: unless you are writing a patent. Then the goal is to make it as hard to understand as possible.)

#5

The Patented RGB RMS Method

```

rt = rmas[i] - image->r[i];
gt = gmas[i] - image->g[i];
bt = bmas[i] - image->b[i];
result += (rt*rt + gt*gt + bt*bt);

```

- Patent Requirements
 - New - must not be previously available
 - Ancient Babylonians made mosaics
 - Useful
 - Non-obvious
 - Many of you came up with this method!
 - Some of you used abs instead, which works as well

#6

History of Scheme

- **Scheme** [Guy Steele & Gerry Sussman, 1975]
Guy Steele co-designed Scheme and created the first Scheme interpreter for his 4th year project
More recently, Steele specified **Java** [1995]
- “Conniver” [1973] and “Planner” [1967]
- Based on **LISP** [John McCarthy, 1958]
- Based on Lambda Calculus (Alonzo Church, 1930s)
- Last few lectures in course on Lambda Calc

#7

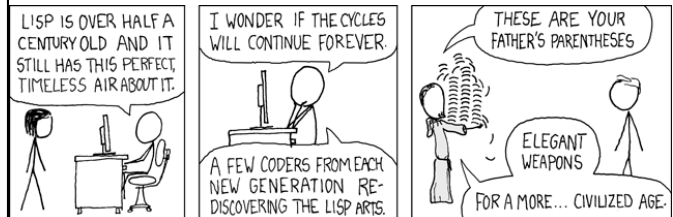
LISP

“Lots of Insipid Silly Parentheses”

“Lost In a Sea of Parentheses”

“**LISt** Processing language”

Lists are pretty important – hard to write a useful Scheme program without them.



#8

Ways to Design Programs

- Think about what you want to **do**, and turn that into code.
- Think about what you need to **represent**, and design your code around that.

Which is better?

#9

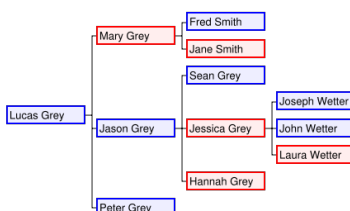
Data Structure

- A **data structure** is a way of storing and **organizing** data so that it can be **used** efficiently by a computer program.
- A well-designed data structure allows **many operations** to be performed, using as **few resources** (such as time and memory space) as possible.
- When designing of many computer programs, the choice of data structures is a **primary consideration**. Experience in building large systems has shown that the **difficulty** of implementation and the **quality** and **performance** of the final result depend heavily on choosing the best data structure.

#10

Data Structure Examples

- single **integer**: 16777216
- **string**: “aافتاب label boomeh”
- <x,y> **pair**: <38.0292, -78.5662>
- Family **tree**

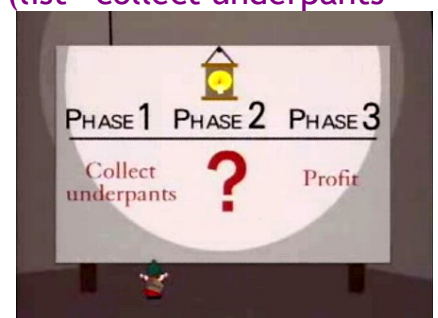


#11

Data Structure Example: List

- List of classes, list of students, list of French war heroes, list of countries in the UN, list of X-men, list of groceries, ...

(define gnome-plan (list “collect underpants” “?” “profit”))



#12

Liberal Arts Trivia: Philosophy

- In the Utopian Kallipolis, philosopher kings ruled the ideal city state: "Philosophers [must] become kings...or those now called kings [must] ...genuinely and adequately philosophize." In the same book, the author fashions the *ship-of-state* metaphor: "[A] true pilot must of necessity pay attention to the seasons, the heavens, the stars, the winds, and everything proper to the craft if he is really to rule a ship". Name the philosopher *and* the book.

#13

Liberal Arts Trivia: Neuroscience

- *These parts* of a neuron are cellular extensions with many branches, and metaphorically this overall shape and structure is referred to as a tree. This is where the majority of input to the neuron occurs. Information outflow (i.e. to other neurons) can also occur, but not across chemical synapses; there, the backflow of a nerve impulse is inhibited by the fact that an axon does not possess chemoreceptors and *these parts* cannot secrete neurotransmitter chemicals. This unidirectionality of a chemical synapse explains why nerve impulses are conducted only in one direction.

#14

Making Lists

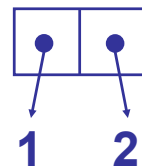
- Lists are **so important** that we will now discuss how to make them. (define villains-1984 ...)



#15

Making a Pair

> (cons 1 2)
(1 . 2)



cons constructs a pair

#16

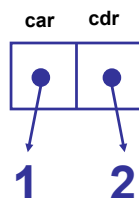
Splitting a Pair

> (car (cons 1 2))

1

> (cdr (cons 1 2))

2



car extracts first part of a pair
cdr extracts second part of a pair

#17

Why “car” and “cdr”?

- Original (1950s) LISP on IBM 704
 - Stored cons pairs in memory registers
 - **car** = “Contents of the Address part of the Register”
 - **cdr** = “Contents of the Decrement part of the Register” (“could-er”)
- Doesn’t matter unless you have an IBM 704
- Think of them as **first** and **rest**

(define first car)
(define rest cdr)

(The DrScheme “Pretty Big” language already defines these, but they are not part of standard Scheme.)

#18

Implementing cons, car and cdr

```
(define (cons a b)
  (lambda (w) (if w a b)))
```

Advanced
Detail!

```
(define (car pair) (pair #t))
(define (cdr pair) (pair #f))
```

Scheme provides primitive implementations for cons, car, and cdr. But, we could define them ourselves.

#19

Pairs are fine, but how do we make threesomes?



#20

Triple

A **triple** is just a **pair** where one of the parts is **also a pair**!

```
(define (triple a b c)
  (cons a (cons b c)))
(define (t-first t) (car t))
(define (t-second t) (car (cdr t)))
(define (t-third t) (cdr (cdr t)))
```

#21

Quadruple

A **quadruple** is a **pair** where the second part is a triple

```
(define (quadruple a b c d)
  (cons a (triple b c d)))
(define (q-first q) (car q))
(define (q-second q) (t-first (cdr q)))
(define (q-third q) (t-second (cdr q)))
(define (q-fourth q) (t-third (cdr q)))
```

#22

Multiples

- A quintuple is a pair where the second part is a quadruple
- A sextuple is a pair where the second part is a quintuple
- A septuple is a pair where the second part is a sextuple
- An octuple is group of octupi
- A ? is a pair where the second part is a ...?

The Feynman point is the sequence of six 9s which begins at the 762nd decimal place of π . It is named after physicist Richard Feynman, who once stated during a lecture he would like to memorize the digits of π until that point, so he could recite them and quip "nine nine nine nine nine nine and so on."

#23

Lists

List ::= (cons *Element* *List*)

A **list** is a **pair** where the second part is a **list**.

One big problem: how do we stop?
This only allows infinitely long lists!

#24

Lists

List ::= (cons Element List)

List ::=

It's hard to write this!

A **list** is either:

a pair where the second part is a *list*
or, empty

The function **list?** returns **#t** for a list and
#f for all other values.

#25

Null

List ::= (cons Element List)

List ::= null

A **list** is either:

a pair where the second part is a *list*
or, empty (**null**)

The function **null?** returns **#t** for the empty
list and **#f** for all other values.

#26

List Examples

```
> null
()
> (cons 1 null)
(1)
> (list? null)
#f
> (list? (cons 1 2))
#f
> (list? (cons 1 null))
#t
```

Why #f?

#27

More List Examples

```
> (list? (cons 1 (cons 2 null)))
#t
> (car (cons 1 (cons 2 null)))
1
> (cdr (cons 1 (cons 2 null)))
(2)
> (null? (list 1 2))
#f
> (null? null)
#t
```

Try these on paper!

#28

More List Examples

```
> (list? (cons 1 (cons 2 null)))
#t
> (car (cons 1 (cons 2 null)))
1
> (cdr (cons 1 (cons 2 null)))
(2)
> (null? (list 1 2))
#f
> (null? null)
#t
```

#29

Recap

- A **list** is either:
 - a **pair** where the second part is a **list**
or **null** (note: some books use **nil**)
- Pair primitives:
 - (cons a b) Construct a pair <a, b>
 - (car pair) First part of a pair
 - (cdr pair) Second part of a pair

#30

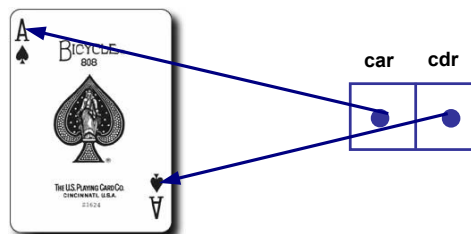
Card Tricks for Problem Set 2



#31

Problem Set 2: Programming with Data

- Representing a card

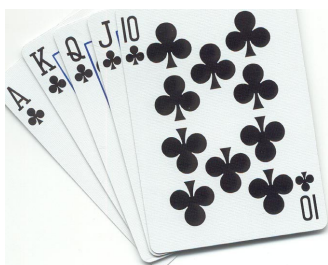


Pair of rank (Ace) and suit (Spades)

#32

Problem Set 2: Programming with Data

- Representing a card: `(cons <rank> <suit>)`
- Representing a hand:

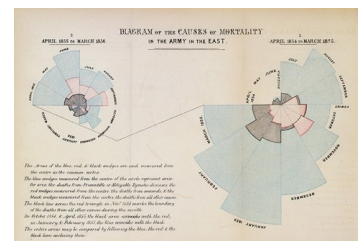


(list (make-card Ace clubs)
(make-card King clubs)
(make-card Queen clubs)
(make-card Jack clubs)
(make-card 10 clubs))

#33

Liberal Arts Trivia: Nursing

- This "Lady with The Lamp" was a nurse, writer and statistician. Her *Diagram of the Causes of Mortality in the Army in the East* was a pioneering use of statistical graphics, including the pie chart and polar area diagram (Crimean War, 1854).



Liberal Arts Trivia: Geography

Name the largest enclosed body of water on Earth by area, variously classed as the world's largest lake or a full-fledged sea. It has a surface area of 371,000 square kilometers and is bounded by northern Iran, southern Russia, western Kazakhstan and Turkmenistan, and eastern Azerbaijan.



#35

Liberal Arts Trivia: Jewish Studies

This record of rabbinic discussions pertaining to Jewish law, ethics, customs and history is a central text of mainstream Judaism. It is considered cryptic and hard to understand, containing obscure Greek and Persian words. Scholars often produce running commentaries that explicate sections.



#36

How To Write A Procedure

- Find out what it is supposed to do.
 - What are the **inputs**? What types of values?
 - What is the **output**? A number? Procedure? List?
- Think about some example inputs and outputs
- **Define your procedure**
 - More on this next slide
- **Test** your procedure

#37

Defining A Procedure

- **Be optimistic!**
- **Base case:** Think of the simplest input to the problem that you know the answer to.
 - For number inputs, this is often zero.
 - For list inputs, this is often the empty list (null).
- **Recursive step:** Think of how you would solve the problem in terms of a smaller input. Do part of the work now, then make a **recursive call** to handle the rest.
 - For numbers, this usually involves subtracting 1.
 - For lists, this usually involves cdr.

#38

Procedure Skeleton

- The vast majority of recursive functions look like this:

```
(define (my-procedure my-input)
  (if (is-base-case? my-input)
      (handle-base-case my-input)
      (combine (first-part-of my-input)
                 (my-procedure (rest-of my-input))))))
```

#39

Example: max-elt

- “Define a procedure **max-elt** to find the maximum element in a list of positive integers. If the list is empty, return 0.”
 - What is the input?
 - What is the output?
 - Example input:
 - (list 1 2) -> 2
 - (list 7 5 3) -> 7

#40

max-elt Skeleton

```
(define (my-procedure my-input)
  (if (is-base-case? my-input)
      (handle-base-case my-input)
      (combine (first-part-of my-input)
                 (my-procedure (rest-of my-input))))))
```

- is-base-case? = handle-base-case =
- combine = first-part-of = rest-of =



#41

max-elt Skeleton

```
(define (my-procedure my-input)
  (if (is-base-case? my-input)
      (handle-base-case my-input)
      (combine (first-part-of my-input)
                 (my-procedure (rest-of my-input))))))
```

- is-base-case? = null? handle-base-case = 0
- combine = max first-part-of = first rest-of = rest

#42

max-elt defined!

```
(define (my-procedure my-input)
  (if (is-base-case? my-input)
      (handle-base-case my-input)
      (combine (first-part-of my-input)
                 (my-procedure (rest-of my-input)))))
- is-base-case? = null?   handle-base-case = 0
- combine = max          first-part-of = first   rest-of = rest

(define (max-elt my-input)
  (if (null? my-input)
      0
      (max (first my-input)
             (max-elt (rest my-input)))))
```

#43

List Length

- Define a procedure that takes as input a list, and produces as output the length of that list.

(length null) → 0

(length (list 1 2 3)) → 3

(length (list 1 (list 2 3 4))) → 2

Do this now on paper. Yes, really. *Hint0: what is the definition of a list? Hint1: use if and null?.*

#44

List Length

- List is a recursive *data structure*, so length must be a *recursive function*.
- By definition, a list is *either empty or* a pair containing another list.
 - The length of the empty list is 0.
 - The length of a non-empty list is 1 + the length of its tail.

```
(define (length x)
  (if (null? x) 0 (+ 1 (length (rest x)))))
```

#45

Homework

- It's OK if you are confused now.
- Lots of opportunities to get unconfused:
 - Problem Set 2 (and PS3 and PS4)
 - Forum!
 - Read the Course Book
 - Next Class - lots of examples programming with recursive functions and definitions
 - Office and Lab Hours!

#46