CS3L: Introduction to Symbolic Programming

Lecture 22:
Midterm Review

Summer 2008

Gilbert Chou
cs3-ta@imail.eecs.berkeley.edu

Announcements

- Midterm two Tuesday July 29th
  - This one will probably be harder than the first
- Miniproject 3
  - Due Friday July 25th at 11:59 pm
- Practice problems on course website

Useful Procedures

appearances

item (1-index)

position (0-index)

member *

odd?, even?, equal?, empty?, word?, member?

and, or, not, >, <, >=, <=, =

count

first, last, bf, bl

word, sentence

+, -, *, /, quotient, remainder, sqrt

every, keep, accumulate

Recursion

- Base case
- Reduce arguments
- Leap of Faith
- Box model
- Passing information
  - Counters
  - Flags – explicit, implicit
  - Tagging data

Common bugs...

Recursive Patterns

- Mapping – every, replace
- Counting – count-evens
- Finding – first-choice
- Filtering – keep, keep-wd
- Testing – all-odd?
- Combining – accumulate, sum-of-all

Higher Order Functions

- Every
  - 1st argument – procedure with one argument that returns a word or sentence
  - 2nd argument – a word or sentence
  - ALWAYS returns a sentence

Common bugs...

Higher Order Functions

- Keep
  - 1st argument – procedure with one argument and returns a boolean value (#tor #f)
    - In other words, a predicate
  - 2nd argument – a word or sentence
  - Returns a word or sentence (depending on input)

Common bugs...

Higher Order Functions

- Accumulate
  - 1st argument – procedure with two arguments
    - Responsible for combining values
  - 2nd argument – a word or sentence
  - Returns whatever the procedure from the 1st argument returns

Common bugs...

Higher Order Functions

- Write a higher order function for:
  - Counting
  - Finding
  - Testing
  - paired-every
  - double-every

Lambda, Let, Define

- Lambda syntax:
  - (lambda (<arguments>) <body>)
- Lambda returns a procedure!
- Lambdas can be avoided altogether, but they are useful for short procedure definitions

Examples:

```
(define (add-n n)
  (lambda (x) (+ x n)))
(define (keep-wd wd sent)
  (keep (lambda (elem) (equal? elem wd)) sent))
```

Lambda, Let, Define

- Let syntax:

```
(let ( (<name> <value>)
       (<name> <value>)
        ...                    )
            <body>  )
```

- Useful for storing procedure calls that are used multiple times

Lambda, Let, Define

- Define syntax:
  - Defining variables
    - (define <name> <value>)
  - Defining procedures
    - (define (<proc_name> <args>) <body>)
    - Equivalent to:

      (define <proc_name> (lambda (<args>) <body>))

Problem Solving

- Check domain and range
  - What do we take in...
  - What do we return...
- Think abstractly
- Work backwards
- Break into steps
  - Helper procedures