COMP 200
Structure & Interpretation of Computer Programs

- Today
  - The structure of COMP 200
  - The content of COMP 200
  - Beginning to Scheme

9/30/2015 1

COMP 200
Structure & Interpretation of Computer Programs

- Main sources of information on logistics:
  - General information handout
  - Course web page
    - http://courses.ku.edu.tr/comp200/
  - E-tutor
    - http://etutor.ku.edu.tr/

9/30/2015 2

COMP 200
Structure & Interpretation of Computer Programs

- Structure of the course
  - 2 lectures/week
  - 1 recitation/week
  - tutorials

9/30/2015 3

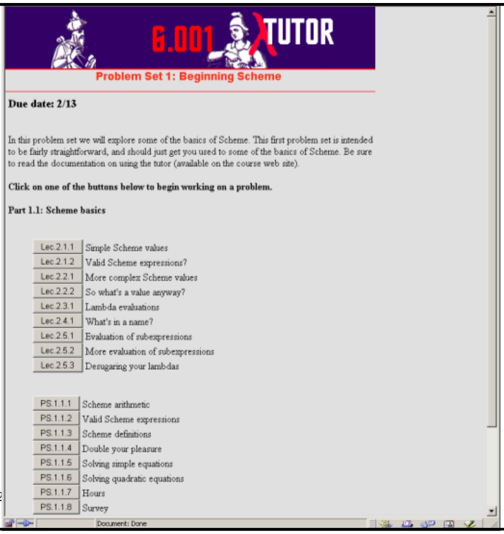
COMP 200
Structure & Interpretation of Computer Programs

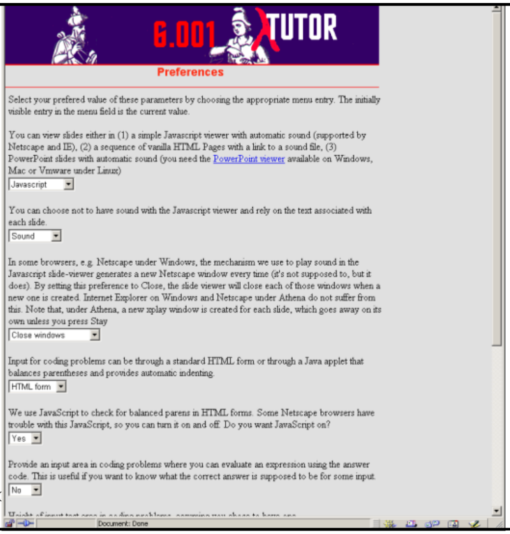
- Grades
  - Midterm 30%
  - Final exam 30%
  - 1 introductory project and 4 extended programming projects – 30%
  - Problem sets 10 %
  - Participation 10%
  - Attendance 0%

9/30/2015 4

Other logistics

- Problem sets
  - Are released online
- Projects
  - First one will be distributed later in the term – check website for updates.





COMP 200: Structure and Interpretation of Computer Programs

- Other issues
  - Collaboration – Read description on web site
  - Time spent on course
    - Survey shows 15-18 hours/week
    - Seeking help
      - Lab assistants
      - Other sources
  - Tutorials: Check your schedule
  - KOLT tutors
    - Watch for updates

COMP 200: Structure and Interpretation of Computer Programs

- Today
  - The structure of COMP 200
  - The content of COMP 200
  - Beginning to Scheme

What is the focus of COMP 200?

- This course is about ~~Computer~~ ~~Science~~

An analogy is to Geometry:
This comes from Ghia & Metra
or Earth & Measure

Geometry deals with Declarative or "What is" knowledge.

Computer Science deals with Imperative or "How to" knowledge

Declarative Knowledge

- "What is true" knowledge

$\sqrt{x}$ is the y such that $y^2 = x$ and $y \geq 0$

Imperative Knowledge

- "How to" knowledge

To find an approximation of square root of x:
- Make a guess G
- Improve the guess by averaging G and x/G
- Keep improving the guess until it is good enough

Example : $\sqrt{x}$ for $x = 2$.

X = 2	G = 1
X/G = 2	G = ½ (1+ 2) = 1.5
X/G = 4/3	G = ½ (3/2 + 4/3) = 17/12 = 1.416666
X/G = 24/17	G = ½ (17/12 + 24/17) = 577/408 = 1.4142156

9/30/2015 13

"How to" knowledge

To capture "how to" knowledge – want to describe a series of steps to be followed in order to deduce a particular value

-- a recipe

-- call this a procedure

Actual evolution of steps inside machine for a particular version of problem – called a process

Need a language for describing processes:

- vocabulary
- rules for writing compound expressions – syntax
- rules for assigning meaning to constructs – semantics

9/30/2015 14

Using procedures to control complexity

Goals

- Create a set of primitive elements in language – simple data and simple procedures
- Create a set of rules for combining elements of language
- Create a set of rules for abstracting elements – treat complex things as primitives

Why?

- Allows us to create complex procedures while suppressing details

Target:

Create complex systems while maintaining: robustness, efficiency, extensibility and flexibility.

9/30/2015 15

Key Ideas

- MANAGEMENT OF COMPLEXITY
  - Procedure and data abstraction
  - Conventional interfaces & programming paradigms
    - manifest typing
    - streams
    - object oriented programming
  - Metalinguistic abstraction
    - layered languages for new problems
    - hardware/register languages
    - Scheme evaluator(s)
    - manipulation of programs: compilation

9/30/2015 16

COMP 200: Structure and Interpretation of Computer Programs

- Today
  - The structure of COMP 200
  - The content of COMP 200
  - Beginning to Scheme

9/30/2015 17

Computation as a metaphor

- Capture descriptions of computational processes
- Use abstractly to design solutions to complex problems
- Using a language to describe processes
  - Primitives
  - Means of combination
  - Means of abstraction

9/30/2015 18

Describing processes

- Computational process:
  - Precise sequence of steps used to infer new information from a set of data
- Computational procedure:
  - The "recipe" that describes that sequence of steps

9/30/2015 19

Level of detail?

- Use bits?
  - Single piece of data: T or F, 0 or 1
  - Encode everything in terms of binary bits

TAG	**00200012000120001**

- Use higher level abstractions

9/30/2015 20

Rules for Scheme

1. Legal expressions have rules for constructing from simpler pieces
2. (Almost) every **expression** has a **value**, which is “returned” when an expression is **“evaluated”.**
3. Every value has a **type.**

9/30/2015 21

Language elements – primitives

- Self-evaluating primitives – value of expression is just object itself
  – Numbers: 29, -35, 1.34, 1.2e5
  – Strings: “this is a string” “ this is another string with %&^ and 34”
  – Booleans: #t, #f

9/30/2015 23

Language elements – primitives

- Built-in procedures to manipulate primitive objects
  – Numbers: +, -, *, /, >, <, >=, <=, =
  – Strings: string-length, string=?
  – Booleans: boolean/and, boolean/or, not

9/30/2015 24

Language elements – primitives

- Names for built-in procedures
  – +, *, -, /, =, …
  – What is the value of such an expression?
  – + → [#procedure …]
  – Evaluate by looking up value associated with name in a special table

9/30/2015 25

Language elements – combinations

- How do we create expressions using these procedures?

(+ 2 3)

Close paren

Other expressions

Open paren

Expression whose value is a procedure

- Evaluate by getting values of sub-expressions, then applying operator to values of arguments

9/30/2015 26

Language elements - combinations

- Can use nested combinations – just apply rules recursively

  (+ (* 2 3) 4) →10

  (* (+ 3 4) (- 8 2)) →42

9/30/2015 27

Language elements -- abstractions

- In order to abstract an expression, need way to give it a name

(define score 23)

- This is a special form
  - Does not evaluate second expression
  - Rather, it pairs name with value of the third expression
- Return value is unspecified

9/30/2015 28

Language elements -- abstractions

- To get the value of a name, just look up pairing in environment

  score → 23

  – Note that we already did this for +, *, …

  (define total (+ 12 13))

  (* 100 (/ score total)) → 92

- This creates a loop in our system, can create a complex thing, name it, treat it as primitive

9/30/2015 29

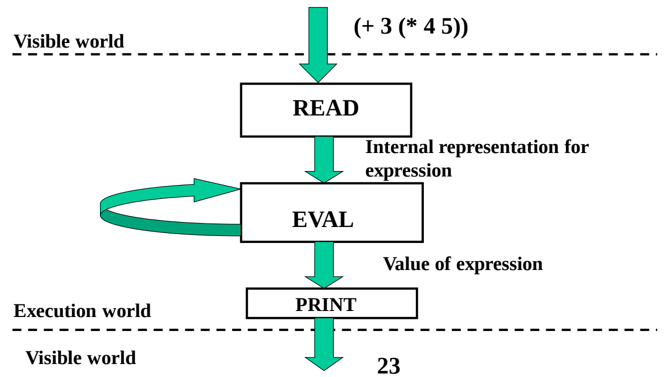
Scheme Basics

- Rules for evaluation

1. If **self-evaluating,** return value.
2. If a **name,** return value associated with name in environment.
3. If a **special form,** do something special.
4. If a **combination,** then
   a. *Evaluate* all of the subexpressions of combination (in any order)
   b. *apply* the operator to the values of the operands (arguments) and return result

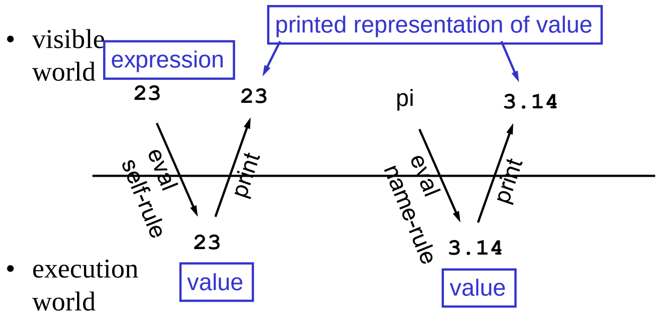
9/30/2015 30

Read-Eval-Print



9/30/2015 31

A new idea: two worlds

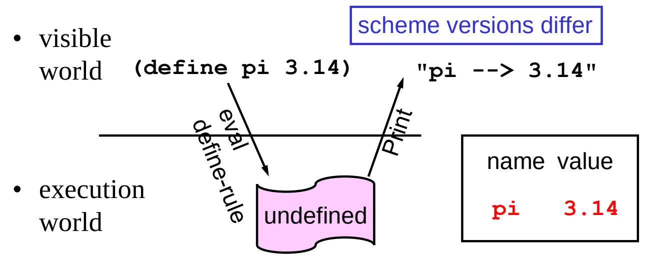


name-rule: look up value of name in current environment

9/30/2015 32

Define special form

- define-rule:
  – evaluate 2nd operand only
  – name in 1st operand position is bound to that value
  – overall value of the define expression is undefined



9/30/2015 33

Mathematical operators are just names

`(+ 3 5)` ➔ `8`

`(define fred +)` ➔ `undef`

`(fred 4 6)` ➔ `10`

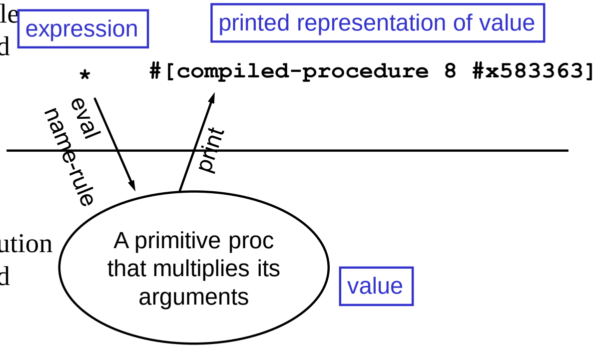
- How to explain this?

- Explanation
  - + is just a name
  - + is bound to a value which is a procedure
  - line 2 binds the name **`fred`** to that same value

9/30/2015 34

Primitive procedures are just values

- visible world
- execution world



9/30/2015 35

Summary

- Primitive data types
- Primitive procedures
- Means of combination
- Means of abstraction – names

9/30/2015 36