6.001: Structure and Interpretation of Computer Programs

• Today
  – The structure of 6.001
  – The content of 6.001
  – Beginning to Scheme

What is the focus of 6.001?

• This course is about computer science.
• Geometry was once equally misunderstood.
  • The term comes from ghia & metra or earth & measure
  • But in fact it’s about…
• Computer Science deals with imperative or “how to” knowledge

Declarative Knowledge

• “What is true” knowledge

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

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 \( x = 2 \).

\[
\begin{array}{|c|c|}
\hline
X & G \\
\hline
2 & 1 \\
\hline
\end{array}
\]

“How to” knowledge

Why “how to” knowledge?
• Could just store tons of “what is” information
• Much more useful to capture “how to” knowledge – a series of steps to be followed to deduce a particular value
  – a recipe
  – called a procedure
• Actual evolution of steps inside machine for a particular version of the problem – called a process
• Distinguish between procedure (recipe for square root in general) and process (computation of specific result)

Describing “How to” knowledge

Need a language for describing processes:
• Vocabulary – basic primitives
• Rules for writing compound expressions – syntax
• Rules for assigning meaning to constructs – semantics
• Rules for capturing process of evaluation – procedures
Using procedures to control complexity

Goals:
- Create a set of primitive elements—simple data and 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? — Can create complex procedures while suppressing details

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

Key Ideas in 6.001

- Management of complexity:
  - Procedure and data abstraction
  - Conventional interfaces & programming paradigms
    - manifest typing
    - streams
    - object oriented programming
  - Metalinguistic abstraction:
    - creating new languages
    - evaluators

Computation as a metaphor

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

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 in general, independent of specific instance

Representing basic information

- Numbers
  - Primitive element—single binary variable
    - Takes on one of two values (0 or 1)
    - Represents one bit (binary digit) of information
  - Grouping together
    - Sequence of bits
      - Byte—8 bits
      - Word—16, 32 or 48 bits
  - Characters
    - Sequence of bits that encode a character
      - EBCDIC
      - ASCII

Binary numbers and operations

- Unsigned integers
  \[ \sum_{j=0}^{n-1} b_j \cdot 2^j \quad \text{where} \quad b_j \text{ is 0 or 1} \]
Binary numbers and operations

- Addition

<table>
<thead>
<tr>
<th>+0</th>
<th>+1</th>
<th>+0</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

10101

111

11100

Binary numbers and operations

- Can extend to signed integers (reserve one bit to denote positive versus negative)
- Can extend to character encodings
  - Representation is too low level!
  - Need abstractions!!

Assuming a basic level of abstraction

- We assume that our language provides us with a basic set of data elements
  - Numbers
  - Characters
  - Booleans
- And with a basic set of operations on these primitive elements
- Can then focus on using these basic elements to construct more complex processes

Our language for 6.001

- Scheme
  - Invented in 1975
- Dialect of Lisp
  - Invented in 1959

Rules for describing processes in 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.

Kinds of Language Constructs

- Primitives
- Means of combination
- Means of abstraction
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

Language elements – primitives

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

Language elements – primitives

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

Language elements – combinations

• How do we create expressions using these procedures?

\[(+ (* 2 3) 4) \rightarrow 10\]

\[(* (+ 3 4) (- 8 2)) \rightarrow 42\]

Language elements -- abstractions

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

\[(define \text{score} \text{23})\]
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

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

Read-Eval-Print
Visible world
 Execution world

A new idea: two worlds
visible world
expression
value
printed representation of value
execution world

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

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
Primitive procedures are just values

Summary

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