Structure and Interpretation of Computer Programs

COMP200
TODAY
Outline

• The structure of COMP200
• The content of COMP200
• Introduction to Scheme
• The structure of COMP200
• The content of COMP200
• Introduction to Scheme
COURSE INFORMATION

Logistics

- Syllabus and other handouts
- Course webpage
  https://sites.google.com/a/ku.edu.tr/comp200/fall2019
- Problem sets
  http://etutor.ku.edu.tr/comp200/tutor.cgi
- Email TAs and me!
COURSE INFORMATION

Structure

• 2 lectures a week
• Labs
COURSE INFORMATION

Grades

- 2 midterms 40%
- Final exam 25%
- 5 programming projects 25%
- Problem sets 10%
- Participation
COURSE INFORMATION

Grades

• 2 midterms 40%
• Final exam 25%
• 5 programming projects 25%
• Problem sets 10%
• Participation
COURSE INFORMATION

Grades

• 2 midterms 40%
• Final exam 25%
• 5 programming projects 25%
• Problem sets 10%
• Participation
COURSE INFORMATION
Contact

• Lecturer: Fatma Guney
  Office Hours: ENG-203A
  Time: TBD

• TAs: Ozan Arkan Can, Berkay Furkan Onder, Ulas Sert
COURSE INFORMATION

Other

• Problem Sets
  • through the online tutor
  • due electronically
COURSE INFORMATION

Other

• Problem Sets
  • through the online tutor
  • due electronically

• Projects
• Collaboration

• Time spent on course 10-15 hours a week

• Get help!
• The structure of COMP200
• The content of COMP200
• Introduction to Scheme
COMP200
What is it about?

Computer Science
COMP200
What is it about?

Computer Science
COMP200
What is it about?

Computer Science
COMP200

What is it about?
DECLARATIVE KNOWLEDGE

“What is true” knowledge

$\sqrt{x}$ is the $y$ such that $y^2 = x$ and $y \geq 0$
COMP200

What is it about?

Imperative Knowledge

“How to” knowledge
COMP200

What is it about?

Imperative Knowledge

“How to” knowledge
Find an approximation of $\sqrt{x}$

Make a guess $G$

Improve the guess by averaging $G$ and $x/G$

Keep improving the guess until it is good enough
Find an approximation of $\sqrt{x}$

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 $x = 2$
Find an approximation of $\sqrt{x}$
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 $x = 2$

\[
\begin{array}{c|c}
  x = 2 & G = 1 \\
\end{array}
\]
**IMPERATIVE KNOWLEDGE**

“How to” knowledge

Find an approximation of $\sqrt{x}$

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 $x = 2$**

<table>
<thead>
<tr>
<th>$x$ = 2</th>
<th>$G$ = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x/G = 2$</td>
<td>$G = \frac{1}{2}(1 + 2) = 1.5$</td>
</tr>
</tbody>
</table>
**IMPERATIVE KNOWLEDGE**

"How to" knowledge

Find an approximation of $\sqrt{x}$

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 $x = 2$**

<table>
<thead>
<tr>
<th>$x = 2$</th>
<th>$G = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x/G = 2$</td>
<td>$G = \frac{1}{2}(1 + 2) = 1.5$</td>
</tr>
<tr>
<td>$x/G = 4/3$</td>
<td>$G = \frac{1}{2}(3/2 + 4/3) = 17/12 = 1.416666$</td>
</tr>
</tbody>
</table>
**IMPERATIVE KNOWLEDGE**

“How to” knowledge

Find an approximation of $\sqrt{x}$

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 $x = 2$**

<table>
<thead>
<tr>
<th>$x = 2$</th>
<th>$G = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x/G = 2$</td>
<td>$G = \frac{1}{2}(1 + 2) = 1.5$</td>
</tr>
<tr>
<td>$x/G = 4/3$</td>
<td>$G = \frac{1}{2}(3/2 + 4/3) = 17/12 = 1.416666$</td>
</tr>
<tr>
<td>$x/G = 24/17$</td>
<td>$G = \frac{1}{2}(17/12 + 24/17) = 577/408 = 1.4142156$</td>
</tr>
</tbody>
</table>
IMPERATIVE KNOWLEDGE

“How to” knowledge

in comparison to declarative knowledge
IMPERATIVE KNOWLEDGE

“How to” knowledge

A series of steps to be followed a recipe or a *procedure*
IMPERATIVE KNOWLEDGE

“How to” knowledge

Actual evolution of steps inside a machine
Actual evolution of steps inside a machine a process
IMPERATIVE KNOWLEDGE

“How to” knowledge

A language for describing processes:
A *language* for describing processes:

- vocabulary
A *language* for describing processes:

* vocabulary
* rules for writing compound expressions
A *language* for describing processes:

- vocabulary
- rules for writing compound expressions
  
  *syntax*
A language for describing processes:

• vocabulary

• rules for writing compound expressions

• rules for assigning meaning to constructs
A language for describing processes:

- vocabulary

- rules for writing compound expressions

- rules for assigning meaning to constructs
COMPLEXITY

The devil
Goal:

- Create a set of **primitive elements**
  simple data and simple procedures
COMPLEXITY
The devil

Goal:

• Create a set of primitive elements
  simple data and simple procedures

• Create a set of rules for combining elements
Goal:

• Create a set of primitive elements simple data and simple procedures

• Create a set of rules for combining elements

• Create a set of rules for abstracting elements complex things => primitives
COMPLEXITY

The devil

Why?
Why?

Create complex procedures while

- suppressing details
- maintaining robustness, efficiency, extensibility, and flexibility
Managing complexity
COMP200

Key Ideas

Managing complexity

• Procedure and data abstraction
COMP200
Key Ideas

Managing complexity

• Procedure and data abstraction

• Conventional interfaces & programming paradigms
Managing complexity

- Procedure and data abstraction

- Conventional interfaces & programming paradigms
  manifest typing, streams, object oriented programming
Managing complexity

- Procedure and data abstraction
- Conventional interfaces & programming paradigms
  manifest typing, streams, object oriented programming
- Meta-linguistic abstraction
Managing complexity

- Procedure and data abstraction
- Conventional interfaces & programming paradigms
  manifest typing, streams, object oriented programming
- Meta-linguistic abstraction
  - layered languages for new problems
  - hardware/register languages
  - scheme evaluator(s)
  - manipulation of programs: compilation
• The structure of COMP200
• The content of COMP200
• Introduction to Scheme
INTRODUCTION

Computation as a Metaphor
INTRODUCTION
Computation as a Metaphor

• Capture descriptions of computational processes
INTRODUCTION

Computation as a Metaphor

• Capture descriptions of computational processes
• Use abstractly to design solutions to complex problems
INTRODUCTION
Computation as a Metaphor

• Capture descriptions of computational processes
• Use abstractly to design solutions to complex problems
• Use a language to describe processes
INTRODUCTION
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
INTRODUCTION
Describing Processes

- Computational process
Computational process

Precise sequence of steps used to infer new information from a set of data
INTRODUCTION
Describing Processes

- **Computational process**
  Precise sequence of steps used to infer new information from a set of data

- **Computational procedure**
INTRODUCTION
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
INTRODUCTION
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
INTRODUCTION

Level of Detail
INTRODUCTION

Level of Detail

• Bits?
INTRODUCTION
Level of Detail

• Bits?
  • Single bits of data: T or F, 0 or 1
INTRODUCTION

Level of Detail

• Bits?

  • Single bits of data: T or F, 0 or 1

  • Encode everything in terms of binary bits

TAG

101011010110101
INTRODUCTION
Level of Detail

• Bits?
  • Single bits of data: T or F, 0 or 1
  • Encode everything in terms of binary bits

• Higher level abstractions

| TAG | 101011010110101 |
INTRODUCTION

Rules for Scheme
• Legal expressions have rules for constructing from simpler pieces.
• Legal expressions have rules for constructing from simpler pieces.

• (Almost) every expression has a value, which is “returned” when an expression is “evaluated”.

INTRODUCTION
Rules for Scheme
INTRODUCTION
Rules for Scheme

• Legal expressions have rules for constructing from simpler pieces.

• (Almost) every expression has a value, which is “returned” when an expression is “evaluated”.

• Every value has a type.
LANGUAGE ELEMENTS

Primitives

• Self-evaluating primitives
• Self-evaluating primitives
value of expression is just object itself
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
Built-in procedures to manipulate primitive objects
• Built-in procedures
to manipulate primitive objects

• *Numbers*: 
LANGUAGE ELEMENTS
Built-in Procedures

• Built-in procedures to manipulate primitive objects
  • *Numbers*: +, -, *, /, >, <, >=, <=, =
• Built-in procedures to manipulate primitive objects
  
  • *Numbers*: +, -, *, /, >, <, >=, <=, =
  
  • *Strings*: 
Built-in procedures to manipulate primitive objects

- **Numbers**: +, -, *, /, >, <, >=, <=, =
- **Strings**: string-length, string=?
Built-in procedures to manipulate primitive objects

- **Numbers**: +, -, *, /, >, <, >=, <=, =
- **Strings**: `string-length`, `string=?`
- **Booleans**: 
Built-in procedures to manipulate primitive objects

- **Numbers**: +, -, *, /, >, <, >=, <=, =
- **Strings**: string-length, string=?
- **Booleans**: boolean/and, boolean/or, not
• Names for built-in procedures
• Names for built-in procedures
  • +, *, -, /, =, ...
  • What is the value of such an expression?
LANGUAGE ELEMENTS
Built-in Procedures

• Names for built-in procedures
  • +, *, -, /, =, ...
  • What is the value of such an expression?
  • Evaluate by looking up value associated with name in a special table
How do we create expressions using these procedures?

\[(+ \ 2 \ 3)\]

- open parentheses
- expression whose value is a procedure
- other expressions
- close parentheses
How do we create expressions using these procedures?

\((+\ 2\ 3)\ \ 5\)

Evaluate by getting values of sub-expressions, then applying operator to values of arguments
Nested combinations
just apply the rules recursively

\[( + ( \ast \ 2 \ 3 \ 4 ) \)
Nested combinations
just apply the rules recursively

\((+ (\ast 2 3) 4)\) 10
Nested combinations
just apply the rules recursively

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

\[(* (+ 3 4) (- 8 2))\]
Nested combinations
just apply the rules recursively

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

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

Abstractions
In order to abstract an expression, give it a name:
In order to abstract an expression, give it a name:

```
(define score 42)
```
In order to abstract an expression, give it a name:

\[(\text{define score 42})\]

A special form - does not evaluate the 2nd expression.
In order to abstract an expression, give it a name:

```
(define score 42)
```
In order to abstract an expression, give it a name:

\[
(\text{define score } 42)
\]

No return value.
In order to abstract an expression, give it a name:

```
(define score 42)
```

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

Abstractions

More examples:

```
(define sum23 (+ 2 3))
(define score 42)
(* 100 (/ score sum23))
```
More examples:

```
(define sum23 (+ 2 3))
(define score 42)
(* 100 (/ score sum23))
```

create a complex thing, name it, treat it as primitive
SCHEME
Rules for Evaluation

• If self-evaluating, return the value.
SCHEME
Rules for Evaluation

• If self-evaluating, return the value.

• If a name, return the value associated with the name in the environment.
• If self-evaluating, return the value.

• If a name, return the value associated with the name in the environment.

• If a special form, do something special.
SCHEME
Rules for Evaluation

• If self-evaluating, return the value.

• If a name, return the value associated with the name in the environment.

• If a special form, do something special.

• If a combination, then
  • evaluate all of the sub-expressions of the combination (in any order)
  • apply the operator to the values of the operands (arguments) and return the result
SCHEME

Read-Eval-Print
Visible world
Visible world

Execution world

Visible world
SCHEME
Two Worlds

Visible world

Execution world

Visible world
Visible world

Execution world

Visible world

SCHEME
Read-Eval-Print

Internal representation for expression

(+ 3 (* 4 5))
SCHEME
Read-Eval-Print

Visible world

Execution world

Visible world
SCHEME
Read-Eval-Print

Visible world

Visible world

Execution world

READ

Internal representation for expression

EVAL

Value of expression

(+ 3 (* 4 5))
SCHEME
Read-Eval-Print

Visible world

Execution world

Visible world

\( (+ \ 3 \ (\ast \ 4 \ 5) ) \)

READ

EVAL

PRINT

Internal representation for expression

Value of expression

23
SCHEME
Two Worlds

Visible world

23

Execution world
Two Worlds

Visible world

Execution world

SCHEME

23

eval

self-true

23
SCHEME
Two Worlds

Visible world

Execution world

23

23

self-true

eval

print
SCHEME
Two Worlds

Visible world

expression

23

self-true

eval

Execution world

23

value

print

23

printed representation of value
SCHEME
Two Worlds

Visible world

pi

Execution world
**SCHEME**

Two Worlds

Visible world

\[ \text{pi} \]

Execution world

\[ 3.141592653589793... \]

*name-rule*: look up the value of the name in the current environment
Two Worlds

SCHEME

Visible world

Execution world

\[ \pi \approx 3.141592653589793 \]

\[ 3.141592653589793 \ldots \]
Two Worlds

**SCHEME**

Visible world

**expression**

pi

3.141592653589793...

Execution world

**eval**

**name-rule**

3.141592653589793...

**print**

printed representation of value

value
SCHEME
Two Worlds

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

Visible world

Execution world

(define theAnswer 42) > theAnswer 42

undefined

SCHEME

Two Worlds

define-rule

eval

print
Two Worlds

Visible world

(define theAnswer 42)

Execution world

undefined

Name | Value
--- | ---
theAnswer | 42

SCHEME
SCHEME
Two Worlds

Visible world

(define theAnswer 42)

Execution world

undefined

Name | Value
--- | ---
theAnswer | 42
SUMMARY

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