CSE 413: Programming Languages and their Implementation

Ruth Anderson
Autumn 2007

Today's Outline

- Administrative Info
- Overview of the Course
- Introduction to Scheme

Staff

- Instructor
  - » Ruth Anderson (rea@cs.washington.edu)
- Teaching Assistants
  - » Jeremy Brudvik (jtbrudvi@cs.washington.edu)
  - » Paramjit Sandhu (paramsan@cs.washington.edu)

UNIVERSITY of VIRGINIA

Me (Ruth Anderson)

- Grad Student at UW (Programming Languages, Compilers, Parallel Computing)
- Taught Computer Science at the University of Virginia for 5 years
- Grad Student at UW (Educational Technology, Pen Computing)
- Just defended my PhD last fall!!

Web Page

- All info is on the web page for CSE 413
  (or at least will be once things are a bit further along…)
  - » http://www.cs.washington.edu/413
  - » also known as
    http://www.cs.washington.edu/education/courses/413/07au
- Look there for schedules, contact information, assignments, links to discussion boards and mailing lists, etc.

CSE 413 E-mail List

- If you are registered for the course you will be automatically registered.
- E-mail list is used for posting important announcements by **instructor** and **TAs**
- You are responsible for anything sent here

CSE 413 Discussion Board

- The course will have a Catalyst GoPost message board
- Students and Instructors can post and reply to posts. Please use this!
- Use:
  - » General discussion of class contents
  - » Hints and ideas about assignments (but **not** detailed code or solutions)
  - » Other topics related to the course

Course Computing

- College of Arts & Sciences Instructional Computing Lab (aka Math Science Computing Labs)
  - » Basement of Communications building: B-022/027
  - » http://depts.washington.edu/aslab
- Or work from home – all software available on course web

Grading: Estimated Breakdown:

- Approximate Grading:
  - » Homework + Project: 55%
  - » Midterm: 15% (TBA, in class)
  - » Final: 25% (Tues December 11 2:30-4:20)
  - » Participation 5%
- Assignments:
  - » Weights may differ to account for relative difficulty of assignments
  - » Assignments will be a mix of shorter written exercises and longer programming projects

Deadlines & Late Policy

- Assignments generally due Thursday evenings via the web
  - » Exact times and dates will be given for each assignment
- Late policy: NONE
  - » As in, no late assignments accepted

    (Talk to the instructor if something truly outside your control causes problems here)

Academic (Mis-)Conduct

- You are expected to do your own work
  - » Exceptions (group work), if any, will be clearly announced
- Things that are academic mis-conduct:
  - » Sharing solutions, doing work for or accepting work from others
  - » Searching for solutions on the web
  - » Consulting solutions to assignments or projects from previous offerings of this course
- Integrity is a fundamental principle in the academic world (and elsewhere) – we and your classmates trust you; don't abuse that trust

Policy on collaboration

- "Gilligan's Island" rule:
  - » You may discuss problems with your classmates to your heart's content.
  - » After you have solved a problem, *discard all written notes* about the solution.
  - » Go watch TV for a ½ hour (or more). Preferably *Gilligan's Island*.
  - » *Then* write your solution.

Homework for Today!!

1) **Assignment #1:** (posted in the next day or so)
2) **Preliminary Survey**: (On course web page) Fill out by evening of Thursday Sept 27th
3) **Information Sheet**: Bring to lecture on Friday Sept 28th
4) **Download and Install Dr. Scheme**
5) **Reading:** See "Scheme Resources" on Web page

Reading

- No required text, books on reserve
- Other references available from course web page

- Check "Functional Programming & Scheme" Link for:
  - » More notes on Scheme from previous course offerings
  - » *Revised[5] Report on the Algorithmic Language Scheme (R5RS)*
    - Section 2
  - » Link to *Structure and Interpretation of Computer Programs* (Abelson, Sussman, & Sussman)
    - Sections 1-1.1.5

Bring to Class on Friday:

- Name
- Email address
- Year (1,2,3,4)
- Major
- Hometown
- Interesting Fact or what I did over summer/break.

Handouts

- Slide handouts will be provided
  - » Also available on the web page
  - » Not…

- Homeworks not handed out, see the web page

Tentative Course Schedule

- Week 1: Scheme
- Week 2: Scheme
- Week 3: Scheme
- Week 4: Scheme wrapup/intro to C
- Week 5: Procedural programming issues, memory model, pointers, tools
- Week 6: Interlude: formal languages and grammars; language families, intro to compilers
- Week 7: compilers
- Week 8: Machine organization and runtime representation of languages
- Week 9: compilers
- Week 10: garbage collection; special topics

What is this course about?

- Programming Languages
- Their Implementation

Why Study Programming Languages?

- Become Better Software Engineer
  - » Understand How To Use Language Features
  - » Appreciate Implementation Issues
- Better Background For Language Selection:
  - » Familiar With Range Of Languages
  - » Understand Issues/Advantages/Disadvantages
- Better Able To Learn Languages:
  - » You Might Need To Know A Lot

Why Study Compilers?

- Better Understanding Of Implementation Issues in Programming Languages:
  - » How Is "This" Implemented?
  - » Why Does "This" Run So Slowly?
- Translation appears several places:
  - » Processing command line parameters
  - » Converting files/programs from one language/format to another

Why are there so many (1,000s) Programming Languages?

- **Evolution**: structured programming -> OO programming
- **Special Purposes**: Lisp for symbols, Snobol for strings, C for systems, Prolog for relationships
- **Personal Preference**: Programmers have their own personal tastes

What Makes a Programming Language Successful?

- Expressive power (more suited to a particular task)
- Easy to use (teaching/learning)
- Ease of implementation (easy to write a compiler/interpreter for)
- Good compilers (Fortran)
- Economics, patronage (Cobol, Ada)

- Donald Knuth:

» *Programming is the art of telling another human being what one wants the computer to do.*

Programming Domains

- Scientific Applications:
  - » Using The Computer As A Large Calculator
  - » FORTRAN, Mathematica
- Business Applications:
  - » Data Processing And Business Procedures
  - » COBOL, Some PL/I, Spreadsheets
- Systems Programming:
  - » Building Operating Systems And Utilities
  - » C, C++

Programming Domains (2)

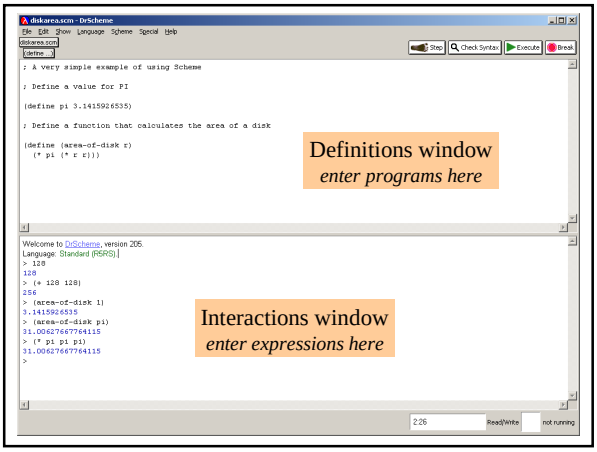
- Parallel Programming:
  - » Parallel And Distributed Systems
  - » Ada, CSP, Modula
- Artificial Intelligence:
  - » uses symbolic rather than numeric computations
  - » lists as main data structure, flexibility (code = data)
  - » Lisp 1959, Prolog 1970s
- Scripting Languages:
  - » A list of commands to be executed
  - » UNIX shell programming, awk, tcl, perl

Programming Domains (3)

- Education:
  - » Languages Designed Just To Facilitate Teaching
  - » Pascal, BASIC, Logo
- Special Purpose:
  - » Other Than The Above...
  - » Simulation
  - » Specialized Equipment Control
  - » String Processing
  - » Visual Languages

Why Scheme?

- The simplicity of the language lets us work on problem solving, rather than just syntax issues
- Flexibility of the language lets us see that the structure of C/Java/Basic is not the only way to express problem solutions
- Variety is the spice of life
  - » study more than one language paradigm and study the relationship between design paradigms
  - » professional programmers switch languages every few years anyway, so start practicing now



Definitions window

- Define programs in the Definitions window
  - » save the contents of the window to a file using menu item File - Save Definitions As …
  - » load existing files with menu item File - Open
  - » execute the contents of the definitions window by clicking on the “Run" button
  - » check and highlight syntax by clicking on the "Check Syntax" button
  - » Re-indent all with control-i

Interactions Window

- Evaluate simple expressions directly in the Interactions window
- Position the cursor after the ">", then type in your expression
  - » DrScheme responds by evaluating the expression and printing the result
  - » recall previous expression with escape-p
- Expressions can reference symbols defined when you executed the Definitions window

Think functionally

- Programming that makes extensive use of assignment is known as
  - » The order of assignments changes the operation of the program because the state is changed by assignment
- Programming without the use of assignment statements is known as
  - » In such a language, all procedures implement well-defined mathematical functions of their arguments whose behavior does not change
- Scheme is heavily oriented towards *functional* style

Primitive Expressions

- constants
  - » integer :
  - » rational :
  - » real :
  - » boolean :
- variable names (symbols)
  - » Names can contain almost any character except white space and parentheses
  - » Stick with simple names like **`value, x, iter,`** ...

Compound Expressions

- Either a *combination* or a *special form*

1. Combination : (operator operand operand …)
   - » there are quite a few pre-defined operators
   - » We can define our own operators
2. Special form
   - » keywords in the language
   - » eg, define, if, cond

Combinations

- (operator operand operand …)

- this is *prefix* notation, the operator comes first
- a combination always denotes a procedure application
- the operator is a symbol or an expression, the applied procedure is the associated value
  - » +, -, abs, my-function
  - » characters like * and + are not special; if they do not stand alone then they are part of some name

Evaluating Combinations

- To evaluate a combination
  - » Evaluate the subexpressions of the combination
  - » Apply the procedure that is the value of the leftmost subexpression (the operator) to the arguments that are the values of the other subexpresions (the operands)
- For example

Percolate values up a tree

Evaluate

```
(* (+ 2 (* 4 6))
   (+ 3 5 7))
```

Evaluating Special Forms

- Special forms have unique evaluation rules
- **`(define x 3)`** is an example of a special form; it is not a combination
  - » the evaluation rule for a simple define is "associate the given name with the given value"
- There are more special forms which we will encounter, but there are surprisingly few of them compared to other languages

Procedures

References

- Section 15.5, *Concepts of Programming Languages*
- Section 4.1, *Revised^5 Report on the Algorithmic Language Scheme (R5RS)*
- For more help:
  - » Sections 1.1.6-1.1.8, *Structure and Interpretation of Computer Programs* (Abelson, Sussman, & Sussman)

Recall the *define* special form

- Special forms have unique evaluation rules
- **`(define x 3)`** is an example of a special form; it is not a combination
  - » the evaluation rule for a simple define is "associate the given name with the given value"

Define and name a variable

- **`(define`** ⟨*name*⟩ ⟨*expr*⟩**`)`**
  - » **`define`** - special form
  - » *name* - name that the value of *expr* is bound to
  - » *expr* - expression that is evaluated to give the value for *name*
- **`define`** is valid only at the top level of a <program> and at the beginning of a <body>

Define and name a procedure

- **`(define (`**⟨*name*⟩ ⟨*formal params*⟩**`)`** ⟨*body*⟩**`)`**
  - » **`define`** - special form
  - » *name* - the name that the procedure is bound to
  - » *formal params* - names used within the body of procedure
  - » *body* - expression (or sequence of expressions) that will be evaluated when the procedure is called.
  - » The result of the last expression in the body will be returned as the result of the procedure call

Example definitions

```
(define pi 3.1415926535)

(define (area-of-disk r)
  (* pi (* r r)))

(define (area-of-ring outer inner)
  (- (area-of-disk outer)
     (area-of-disk inner)))
```


Defined procedures are "first class"

- Compound procedures that we define are used exactly the same way the primitive procedures provided in Scheme are used
  - » names of built-in procedures are not treated specially; they are simply names that have been pre-defined
  - » you can't tell whether a name stands for a primitive (built-in) procedure or a compound (defined) procedure by looking at the name or how it is used

Evaluation example

- `(area-of-ring 4 1)`

Booleans

- Recall that one type of data object is boolean
  - » `#t` (true) or `#f` (false)
- We can use these explicitly or by calculating them in expressions that yield boolean values
- An expression that yields a true or false value is called a predicate
  - » `#t` =>
  - » `(< 5 5)` =>
  - » `(> pi 0)` =>

Conditional expressions

- As in all languages, we need to be able to make decisions based on inputs and do something depending on the result

Predicate **Consequent**

Special form: **cond**

- `(cond` ⟨*clause*$_1$⟩ ⟨*clause*$_2$⟩ `...` ⟨*clause*$_n$⟩`)`
- each clause is of the form
  - » `(`⟨*predicate*⟩ ⟨*expression*⟩`)`
- the last clause can be of the form
  - » `(else` ⟨*expression*⟩`)`

Example: sign.scm

```
; return the sign of x as -1, 0, or 1

(define (sign x)
  (cond
    ((< x 0) -1)
    ((= x 0) 0)
    ((> x 0) +1)))
```


Special form: **`if`**

- **`(if`** ⟨*predicate*⟩ ⟨*consequent*⟩ ⟨*alternate*⟩**`)`**
- **`(if`** ⟨*predicate*⟩ ⟨*consequent*⟩ **`)`**

Examples : abs.scm

```
; absolute value function
(define (abs a)
```


Examples : true-false.scm

```
; return 1 if arg is true, 0 if arg is false
(define (true-false arg)
```


Logical composition

- **`(and`** ⟨e_1⟩ ⟨e_2⟩... ⟨e_n⟩**`)`**
- **`(or`** ⟨e_1⟩ ⟨e_2⟩... ⟨e_n⟩**`)`**
- **`(not`** ⟨e⟩**`)`**

- Scheme interprets the expressions e_i one at a time in left-to-right order until it can tell the correct answer

in-range.scm

```
; true if val is lo <= val <= hi

(define (in-range lo val hi)
  (and (<= lo val)
       (<= val hi)))
```