
\$Id: cse112-study-guide-2021-q2spring.mm,v 1.79 2021-06-01 23:57:54-07 - - \$
PWD: /afs/cats.ucsc.edu/courses/cse112-wm/Syllabus/study-guide-2021-q2spring.d
URL: https://www2.ucsc.edu/courses/cse112-wm/:/Syllabus/study-guide-2021-q2spring.d/

1. Week 1 — Tuesday March 30

- (a) Syllabus, pair programming, course overview. Lab0 intro unix, and review of Data Structures labs. Reference to study guides.
- (b) **Languages/Hello, Languages/Collatz**— Examples of simple programs in multiple languages: Bash, C++, Ocaml, Perl, Prolog, Scheme, Smalltalk.
- (c) Lecture notes: **scheme-1-language.pdf** (p. 1–4).
- (d) **Languages/scheme/Examples/a-simple** — Simple introductory Scheme programs: **hello.scm**, **false.scm**, **factorial.scm**, **fibonacci.scm** (with tracing).

2. Week 1 — Thursday April 1

- (a) Finish **a-simple/**.
 - (1) **derivation-factorial** — showing tail vs non-tail call formula derivations.
 - (2) **stack-tail-usage.scm** — program using small stack space with tail recursion but blowing up stack without tail calls.
- (b) **Examples/b-arith** — examples showing interaction with the command line, hash-bang (#!) scripts, and other examples.
- (c) **asg1-scheme-mbir** specifications. Writing an interpreter in Scheme for a small mini-basic language.
- (d) **code/mbir.scm** — brief overview of the starter code and introduction to interpretation.

3. Week 2 — Tuesday April 6

- (a) **Examples/c-evalexpr** — examples related to evaluating expressions and scanning for labels.
 - (1) **euler.scm** — simple examples of using built-in **eval** function to evaluate lists as expressions.
 - (2) **simple-eval.scm** — hand-coded simple evaluator of expressions.
 - (3) **evalexpr.scm** — hand-coded evaluator of expressions with operators and variables stored in hash tables.
 - (4) **hashexample.scm** — use of hash tables for storing information.
 - (5) **labelhash.scm** — scanning a program list and identifying labels.
 - (6) **readnums.scm** — reading numbers and detecting non-numbers and end of file.
- (b) **misc-cons-lists.d/** — Simple diagram of some node configurations.
picture-21-let-if contains a diagram of the **mbir** program structure.
- (c) **mbir.scm** — detailed dissection of the starter code for the interpreter.

4. Week 2 — Thursday April 8

- (a) **Lecture-notes/scheme-1-language.pdf** — notes on Scheme made from Dybvig's text (p. 4–13).
- (b) Numerous online examples.

5. Week 3 — Tuesday April 13

- (a) **Lecture-notes/scheme-1-language.pdf** (p. 13–end).
- (b) **scheme-2-higherorder.pdf** — Scheme higher order functions (p. 1–12).

6. Week 3 — Thursday April 15

- (a) **scheme-2-higherorder.pdf** — Scheme higher order functions (p. 12–end).
- (b) **Examples/d-functions** — wrap up examples with some programs. **mergesort.scm**, **mutualrec.scm**, **facfun.scm**, **quine.scm**
- (c) **Lecture-notes/ocaml-1-notes.pdf** (p. 1–2), plus numerous online examples. Introduction to Ocaml.

7. Week 4 — Tuesday April 20

- (a) **Examples/a-simple**
 - (1) **hello.ml**, **helloworld.ml**
 - (2) **argv.ml** — access to the command line
 - (3) **epsilon.ml** — showing $1 + \epsilon \equiv 1$
 - (4) **factorial.ml**, **fibonacci.ml** — repeat of examples from Scheme tail call and tail recursions discussion.
 - (5) **length.ml** — another example of internal function with tail recursion
- (b) **Examples/b-evalexpr**
 - (1) **eval1-simple.ml**, **eval2-symbols.ml** — examples of recursive evaluations of expressions, with and without a symbol table.
 - (2) **find.ml** — example of type '**a option**' for returning an object that might not exist
- (c) Assignment 2 specifications
- (d) **asg2/code**
 - (1) **absyn.mli** — abstract syntax of the interpreter
 - (2) **parser.mly**, **scanner.mll** — brief overview of parser and scanner (provided, not written by students).
 - (3) **main.ml** — main function calling interpreter

8. Week 4 — Thursday April 22

- (a) **asg2/code** — finish discussion from last lecture.
 - (a) **interp.{mli,ml}** — extensive discussion of interpreter, where students do the majority of coding for this project.
 - (b) **tables.{mli,ml}** — variable and function tables for maintaining data.
 - (c) **dumper.{mli,ml}** — data dumper and stringification functions for absyn.
 - (d) **etc.{mli,ml}** — miscellaneous other functions.
- (b) **Examples/b-evalexpr**
 - (1) **find-opt-exn.ml** — function for searching a list. Type '**a option**'.
 - (2) **hashtable.ml** — use of **Hashtbl.find** and **Hashtbl.find_opt**
 - (3) **readnumber.ml** — scanning input for individual numbers from stdin.
- (c) **Examples/c-functions** — brief look during remaining lecture time

9. Week 5 — Tuesday April 27

- (a) **Lecture-notes/ocaml-1-notes.pdf** — review.
- (b) **Examples/c-functions** — miscellaneous functions showing more Ocaml style :
 - (1) **ackermann.ml** — test of computational complexity
 - (2) **complex-nrs.ml** — module **Complex** and **float** numbers
 - (3) **exponent.ml** — efficient integer exponent computation
 - (4) **mergesort.ml** — polymorphic efficient sorting of a list
 - (5) **odd-even.ml** — mutually recursive functions
- (c) **Examples/x86-64-code** — generated x86-64 code showing how compilers eliminate tail recursion and rewrite as loops. Source code in Ocaml and C.
 - (1) **boolconst.ml.s**, **boolvar.ml.s** — constant propagation to eliminate a boolean test always false.
 - (2) **length.ml.s**, **facrec.c.s**, **factorial.ml.s** — tail recursive functions implemented as loops by the code generator.

10. Week 5 — Thursday April 29

- (a) **Lecture-notes/ocaml-2-higherorder.pdf** — higher-order functions in Ocaml.
- (b) **Examples/d-higherorder** — examples of higher-order functions in Ocaml.
 - (1) **p1-uncurry.ml** — functions **curry** and **uncurry**
 - (2) **p2-apply.ml** — application of function argument to other arguments.
 - (3) **p3-foldl.ml** — functions written directly using tail recursion, and functions written as arguments to fold left : **sum**, **length**, **reverse**, **member**.
 - (4) **p3-foldl.ml** — **reduce**, exception producing folding, e.g., **find_minimum** using **failwith** and returning an '**a option**'.
 - (5) **p4-foldr.ml** — fold right functions that can not be written tail recursively, implementation using direct recursion and as a parameter to fold right : **map**, **filter**, **append**.
 - (6) **p5-zip.ml** — merging and splitting lists : **unzip**, **zip**, **zipwith**, **inner_product**.

11. Week 6 — Tuesday May 4

- (a) **Examples/a-trivial.d**
 - (1) **hello.st**, **usage.st**, **echoargs.st**, **showargv.st** — trivial examples involving command line.
 - (2) **arithmetic.st**, **divide.st**, **dictionary.st**, **fns-radix.st**, **intsort.st** — simple examples showing arithmetic and some library data structures.
 - (3) **collatz-block.st**, **collatz-class.st** — coding examples: blocks and a simple class.
- (b) **Examples/b-simple.d** — slightly more advanced examples.
 - (1) **arraysum.st**, **ashex.st**, **isgraph.st** — extending a class on the fly with new methods.
 - (2) **perform.st** — use of keyword method **perform:** and **perform:with:** as an analog to a functional language's **apply** function.
 - (3) **sillypet.st** — simple example of class inheritance of methods, dynamic typing, and “duck typing”.
 - (4) **simple-eval.st** — example of a numeric and expression class with inheritance, including **value**, **printOn:**, and **perform:with:** methods.
 - (5) **filein.st**, **parseargs.st**, **priority.st**, **string.st**, **terminalecho.st** — miscellaneous other simple examples.

12. Week 6 — Thursday May 6

- (a) **asg3-smalltalk-mbst/**
 - (1) asg3-smalltalk-mbst specifications overview.
 - (2) discussion of **.mbst** intermediate code files for interpreter
 - (3) **code/mbint.st** — line by line analysis of starter code for project.

13. Week 7 — Tuesday May 11

- (a) Midterm Examination. No lecture.

14. Week 7 — Thursday May 13

- (a) **asg3-smalltalk-mbst/**
 - (1) **code/mbint.st** — repetition of review of starter code.
- (b) **asg3/misc** — comparison of similar problems illustrated by code in Scheme, Ocaml, and Smalltalk, all with the same purpose.
 - (1) **euler.{scm,ml,st}** — examples of complex number arithmetic, somewhat combersome in Ocaml.
 - (2) **perform.{scm,ml,st}** — application of operators to operands in various ways.
 - (3) **evalexpr.{scm,ml,st}** — evaluations of symbolic expressions from data structures.
- (c) **Examples/c-involved.d** — various further examples of Smalltalk code:
binepsilon.st, **complexx.st**, **epsilon.st**, **euler.st**, **floatd-inspect.st**, **floatd-format.cpp**, **initarray.st**, **sorted-names.st**, **treeleaf.st**

15. Week 8 — Tuesday May 18

- (a) **perl/Examples/a-simple** Some simple examples of Perl scripts involving command line, etc.: **hello.perl**, **argv.perl**, **ncat.perl**, **count.perl**, **xref.perl**.
- (b) Lecture notes. **asg4/Perl-notes.d** (lines 1–298). Introduction to Perl syntax and semantics.

16. Week 8 — Thursday May 20

- (a) Review midterm exam with answers and variations.
- (b) **asg4-perl-pmake** — specifications for project, a small subset of the **make** utility.
- (c) **asg4/misc** — some examples related to project.
 - (1) **modtime.perl** — obtain modification time from a file.
 - (2) **runcmd.perl** — run a command using the shell, obtain the resulting success or error status as a message.
 - (3) **graph.perl** — simple example of how to create a graph in Perl.
- (d) **asg4/code/pmake** — preliminary tour of the starter code, including parsing a **Makefile**.
- (e) **asg4/dot.score** and subdirectories discussion of the test data and attendant **Makefiles**.
- (f) **asg4/sample-output.d** — test data showing effects of actual **make**.

17. Week 9 — Tuesday May 25

- (a) **asg4-perl-pmake** — details of pseudo-code for the `make_goal` algorithm and calling sequence for `run_command`. Special details of the `@` and `-` command markers.
- (b) **asg4/code/pmake** — detailed dissection of the starter code and discussion of the data structures.
- (c) **asg4/misc** — finish discussion of miscellaneous related Perl programs.
- (d) Answered many questions from zoom chat about the program and Perl in general.
- (e) Lecture notes. **asg4/Perl-notes.d** (lines 298–380).

18. Week 9 — Thursday May 27

- (a) Lecture notes. **asg4/Perl-notes.d** (lines 380–end), including regular expressions.
- (b) **Perl/Examples/b-moreso** :
 - (1) **cat.perl**, accessing files and iterating `@ARGV`.
 - (2) **dumper**, showing `Data::Dumper`.
 - (3) **switch.perl**, implementing a switch statement by putting subs (lambdas) in a hash table.
 - (4) **text2html.perl**, converting plain text into html and recognizing URLs.
- (c) A short overview discussion of Javascript and timers on web pages.

19. Week 10 — Tuesday June 1

- (a) Further discussion of **pmake** assignment, answering students' questions.
 - (1) Details of test scripts, discussion of **bash** code in each of the testing directories.
 - (2) Example of copying test scripts to a **/tmp** directory and replacing **pmake** with **make** to determine expected output.
 - (3) **asg4/misc/quotameta.perl** — handling wildcards such as **%o : %c**.
 - (4) **asg4/misc/subst-macros.perl** — substituting the values of **pmake** macros.
 - (5) **asg4/misc/modtime.perl** — finding the modification time of a file and determining existence of a file.
- (b) Discussion of shell scripting using **bash** with examples of some production bash scripts in the **bin/** directory.
- (c) Finish up lecture with brief introduction to lambda calculus.

20. Week 10 — Thursday June 3

- (a) Review of exams from previous quarters.