CS 335 - Final Exam Study Suggestions

last modified: 5-5-11

- You are responsible for all material covered in lectures and homeworks; this is just a quick overview of especially-important material.
- You are permitted to bring into the exam a single piece of paper (8.5" by 11") on which you have handwritten whatever you wish on one or both sides. This paper must include your name, it must be handwritten by you, and it will not be returned.
  - Other than this piece of paper, the exam is closed-note, closed-book, and closed-computer.
- Note that final exams are **not** returned; they will be in my office for at least two years, however, and after they are graded you may come and view your final exam in my office if you would like.
- final is CUMULATIVE!
  - if it was fair game for Exam 1 or Exam 2, it is fair game for the final;
  - Thus, using the posted review suggestions for Exam 1 and Exam 2 in your studying for the final would be a good idea. (Note that they are still available from the public course web page, under "Homeworks and Handouts".)
  - studying your Exam 1 and Exam 2 would also be wise.
  - there may indeed be similar styles of questions on the final as on those exams.
  - NOTE that I will happen to include a references page with some examples of Scheme, Prolog, and Squeak code.
- this will be a pencil-and-paper exam, but you will be reading and writing code, statements, and expressions in this format. There could also be questions about concepts, of course.

programming models

- note that, at this point, we have discussed the following programming models: imperative/procedural, functional, logic, object-oriented
  - you should be able to name at least one programming language which has significant support for programming using that programming model (keeping in mind that some languages have support for multiple such paradigms)

automatic vs. manual storage reclamation, basics of garbage collection

- What is meant by manual storage reclamation? by automatic storage reclamation? Why is storage reclamation needed?
  - What are some advantages of manual storage reclamation? some disadvantages?
  - What are some advantages of automatic storage reclamation? some disadvantages?

- What is garbage collection? What is meant by "garbage-collected languages"?
  - note that some languages require garbage collection, either as part of the language specification or just for practical considerations; some were designed for use with manual memory management, although even these might have garbage collected implementations; and some allow for both manual and automatic storage reclamation, with variations;
  - you should know some examples of languages in each of these categories;
- You should be comfortable with, and able to describe and recognize, reference-counting, mark-sweep, copying collectors, heap compaction, and mark-compact approaches to garbage collection; you should be able to discuss advantages and disadvantages to each of these approaches.

parameter-passing modes

- parameter-passing mechanisms covered/discussed so far
  - pass by/call by reference
  - pass by/call by value
  - pass by/call by value-result
  - pass by-call by name
    - you should be able to describe each of these;
    - given a snippet of example code, you should be able to describe what the results of that code would be given a particular parameter-passing mechanism;
    - be able to discuss the trade-offs of these (in terms of efficiency, side-effects, etc.)
- How are parameters usually passed in FORTRAN? What are the benefits and drawbacks of that approach? What is the alternative sometimes used in FORTRAN implementations? Given appropriate fragments of code and the parameter-passing method, you should be able to determine the values that result within that fragment.
- what are the parameter-passing mechanisms in Algol-60? you need to be comfortable with these parameter-passing mechanisms; given Algol-60 code, you should be able to determine the effects of these parameter passing mechanisms, and should be able to determine the behavior of such code.
- what are the parameter-passing mechanisms in Pascal? you need to be comfortable with these parameter-passing mechanisms; given Pascal code, you should be able to determine the effects of these parameter passing mechanisms, and should be able to determine the behavior of such code.
- how is parameter passing unusual in Ada? (user specifies how parameter is to be used, but it is left to the compiler to decide what parameter-passing mechanism to use; can have default values for parameters; can give a parameter and its value in a call, and then the arguments, expressed in this way, can be given in ANY order)

Smalltalk-80/Squeak - extreme object-oriented programming

- (note: for the final, when you see "Smalltalk", assume that Smalltalk-80 is intended)

- where was Smalltalk-80 developed? What computer was it originally developed to be the language for? Who was its initial target audience?
- what are some of the potential benefits of object-oriented programming (OOP)?
- How object-oriented (comparatively) is Smalltalk? What is an unusual aspect of Smalltalk classes, for example, that is not the case (to the same degree) with classes in C++ or Java?
- do need to be comfortable with the essentials of the Smalltalk-80/Squeak language;
  - this includes, of course, being comfortable with string literals, symbol literals, numeric literals, booleans, arrays, etc.
  - What is Squeak's/Smalltalk's syntax intended to mimic? (which explains why an expression ends with the syntax that it does, and explains why messages are written in-fix...)
- Smalltalk has a "rich programming environment ... that is intimately related to the language" --- what are some advantages and disadvantages of this?
  - what are some of the features/capability of the Squeak environment? what are some of the tools provided?
  - what is the typical use of a Workspace? a Transcript? a System Browser? a Method Finder?
- in Smalltalk, what is meant by a keyword?
- does Smalltalk/Squeak have static or dynamic typing? What are the implications of this?
- what languages particularly influenced Smalltalk's design?
- in Smalltalk/Squeak, what are objects? classes? messages? methods? subclasses?
- when a message is sent to a method, what determines the actual method that is invoked?
- need to be comfortable with and familiar with the precedence rules for Smalltalk/Squeak messages;
- what is the distinction between self and super in Smalltalk? What do each mean/refer to?
- should be comfortable with the 8 syntactic forms in Smalltalk;
- need to be comfortable with the syntax and semantics of Smalltalk/Squeak unary, binary, and keyword-based messages;
  - given a statement, you should be able to identify/explain which is/are the receiver(s), which is/are the message(s), and which is/are the argument(s)/parameter(s);
  - note that a message with one argument may involve a binary operator, or it may involve a keyword; how can one tell by looking which is the case?
- what do square brackets represent in Smalltalk/Squeak? What is their significance?
- in setting up a Squeak class...
  - how do you indicate that a class is a subclass of another?
  - how do you indicate instance variables?
  - how do you indicate a category for a class?

  - for a method, how do you indicate what a method returns?
  - to follow good Smalltalk/Squeak style practice, how are parameters expected to be named within a method? Why?
  - In general, how long do Squeak/Smalltalk methods tend to be?
- How many buttons is a Squeak/Smalltalk-80mouse expected to have?
- you will very likely have to read/write Squeak code;
- what are Smalltalk's/Squeak's strengths? weaknesses? OOP's strengths? weaknesses?

Language Generations

- You should be comfortable with MacLennan's language generations (first generation through fourth generation)
- What are the distinguishing characteristics of each of these generations?
- What is a classic example of a language in each of these generations?
- Why can C said to show characteristics from multiple generations?

Control Flow

- What are Scott's eight principal categories for control-flow mechanisms? Why might it be beneficial to think in terms of these categories rather than in terms of the syntax of some particular language?
- How is control flow handled in assembly language?
- Consider early FORTRAN. What was FORTRAN's biggest goal? What were most of FORTRAN's control structures based on? What FORTRAN statement "is the workhorse of control flow"? What was the only "high-level" control structure provided by FORTRAN? Why could it be provided?
  - How can you write a leading-decision loop in FORTRAN? ...a trailing-decision loop in FORTRAN?
  - You should be comfortable with both FORTRAN's arithmetic IF and its logical IF.
- What is the dangling else problem? How do Algol-60, Pascal, and Ada address this problem?
- Why was Algol's for-loop considered to be baroque? You should be able to trace the behavior of a given Algol for-loop.
- How do you write a simple Python for-loop? How does it differ from FORTRAN's and Algol's for-loops?

History of Programming Languages

- (note: most of this was sprinkled throughout the semester, with a few exceptions...)
- who is often considered the first programmer, having described examples that could be performed by Babbage's Analytical Engine (if it had ever been completed)?

  - what much-later programming language was named in honor of this programmer?
- considered the first high-level language, and developed by a team at IBM lead by John Backus from 1954-1957, this language showed that compiled languages could produce machine code that was reasonably efficient as compared to "hand-written" assembly code;
  - what is this language's name an acronym for?
- the popular business-oriented language developed by the U.S. Department of Defense between 1959-1960 by a team led by Grace Hopper;
  - what is this language's name an acronym for?
- this language, designed to help in describing algorithms, was also developed by committee between 1958-1960 --- but it was rather more influential in future language development, since Pascal, C, and Ada (amongst others) are considered to be derivatives of it.
  - what are some of the concepts that it introduced?
- this language used in many artificial intelligence applications was developed by John McCarthy at MIT in the late 1950's.
  - what is this language's name an acronym for?
- what are two prominent languages designed for teaching programming (one from the 1960's, the other from 1971)?
- what language is sometimes described as high-level assembly language?
- what three languages are sometimes described as being from the "Swiss-army knife" school of language design? (trying to include as many features as possible...) One was designed in 1963-1964, another's design was fixed in 1980, and the third is a prominent scripting language;
  - the first was supposed to combine all the best features of FORTRAN, COBOL, Algol60, and more;
  - the second tried to contain all the best software-engineering ideas from 1970's research;
  - the third was developed in the late 1980's, and was "kind of designed to make awk and sed semi-obsolete"
- this language, designed from 1965-1967, is often considered the first object-oriented language, introducing the concept of a class; it was designed originally for simulations.
- this language (whose 1980 version is especially significant) reached back to the one mentioned above for inspiration, and is often considered the "purest" example of an object-oriented language.
- this language, with roots in BCPL and Algol-68, was developed at Bell Labs as part of the Unix development effort (Unix's kernel was eventually rewritten in it), and can be argued to show features of first generation, second generation, and third generation programming languages;
  - what are some ways it is like first generation languages? ... second generation languages? ... third generation languages?
- this language was designed by Stroustrup beginning in 1980, which started out with the goal of extending C using ideas from that same language referred to in three top-level bullets above;

- you should be aware of at least two languages with significant support for functional programming;
  - the first was already described above;
  - the second is a variant of that earlier one; this second was developed between 1975-1978 by Sussman and Steele, and was designed to more closely resemble the lambda calculus;
  - other functional languages include Erlang, Haskell, and ML (I've seen APL classified as a functional programming language, too...!)
- this language is the most-well-known practical example of a logic programming language, and was developed beginning in 1972.
- this language is often considered one of the most significant general-purpose languages introduced in the 1990's; it was originally developed for embedded consumer-electronic applications, but was then revised for Web and network use.
  - one could say that it is both compiled and interpreted -- why?
  - how does it make use of a virtual machine to beneficial effect?
- this scripting language was developed in the early 1990's, when its developer wanted to extend the language ABC for use on the Amoeba distributed operating system.