P.F. Reynolds, Jr. **23 March 1999**

CS655 Midterm - Part I

This is part I of a two part exam. Answer *three* of the four questions in this part. Questions in this part make up 45% of your point total (15 points each). Submit your answers on separate 8-1/2 by 11 sheets of paper. Start a new page for each answer and put answers on only one side of a page. Limit answers in this part to one page each. Put your name at the top of each page. Identify your answers with the five-character mnemonics associated with the questions. *Submit your answers stapled and in lexicographic order.* Type your answers. This exam is open books, *your* notes, etc. However, you are to speak to no one other than me or Jim Gunderson about it. You must stop answering questions on this part of the exam within six hours after you first read any part of it, or any posting about its contents. Turn your exam in to the dept secretaries as soon after your six hours are up as CS office business hours will permit, but no later than 5PM ET, Friday, 26 March. Late exams accepted by prior arrangement only.

Pledge this part of your exam: I have neither given nor received unauthorized aid on this exam, I have abided by the six hour rule, and I have not been a party to the removal of materials from the CS or Sci/Tech libraries to the detriment of others taking this exam.

Watch the CS655 homepage for clarifications and answers to your questions. A copy of this exam can be found on the CS655 homepage.

Answer *three* out of four

1APAR) Recall that Ada designers identified orthogonal aspects of parameter passing that predecessors had missed: the separation of specification from implementation. Parameter passing methods in Ada are IN, IN-OUT and OUT. Implementation of IN-OUT can be by reference or by value-result, implementor's choice. Identify what can go wrong with this design (hint: consider aliasing). Also, consider adding pass by name as an implementation choice for IN-OUT and discuss its ramifications.

1BCMS) Block structure, class hierarchies, modules and subprograms all provide some control over the scope and visibility of named entities (e.g. type names, variable names and constant names, among others). Construct a matrix in which these four structuring methods label individual rows, and issues such as vulnerability and indiscriminate access, *and other issues you identify*, label the columns. Entries in the matrix (i.e. for each structuring method / issue pair) should be your assessment of whether or not the issue is an issue for the structuring method, and if so, any brief thoughts you have about the nature of the issue for that structuring method. For example, for the Blocks / Vulnerability pair, a reasonable entry would be: "yes; intervening blocks can introduce new meanings for non-local references in inner blocks" Evaluation of your answer will be based on quality and extent of the issues you identify, and the correctness of your entries. You may present your matrix in list form if you wish.

1DENO) Extend the denotational semantics of the small language we considered (the one with conditional, assignment, DIVERGE and sequence) to include a C-like FOR statement. Expressions in the small language FOR statement should be limited to features occurring in the original small language (e.g. assignment and side-effect free expressions, and perhaps others, but no function calls).

1STRC) Give possible meanings for the **following**, and explain your reasoning for choosing those meanings. In particular, discuss when assumptions about strictness/laziness apply. Consider explaining your answers in terms of passing arguments by name / by value.

a) **(λx. 3) 1/0**

b) for int y, (F y) = ((> y 1) –> ((odd y) –> (F (+ (* 3 y) 1)) | (F (div x 2))) | 1) **(λx. (F x)) z**

c) for G = function which generates an infinite sequence **(λx. (car x)) G**