Comprehensive Exam: Programming Languages Autumn 2003

This is a 30-minute closed-book exam and the point total for all questions is 30.

All of the intended answers may be written within the space provided. (*Do not use a separate blue book.*) Succinct answers that do not include irrelevant observations are preferred. You may use the back of the preceding page for scratch work. If you to use the back side of a page to write part of your answer, be sure to mark your answer clearly.

The following is a statement of the Stanford University Honor Code:

A. *The Honor Code is an undertaking of the students, individually and collectively:*

   (1) *that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*

   (2) *that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*

B. *The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*

C. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

By writing my "magic number" below, I certify that I acknowledge and accept the Honor Code.

____

(Number)

Prob	# 1	# 2	# 3	# 4	Total
Score					
Max	6	6	8	10	30

1. (6 points) *Garbage Collection.*

   This question is about the design and use of a garbage collector as part of the runtime system.

   (a) (2 points) Assume you have a language like ML, in which all pointers are statically typed. Describe the general operation of a garbage collector that may be run while the application program is stopped. Your collector should only mark objects as garbage and delete them if really are garbage. Does your collector find and delete all inaccessible objects?

   (b) (2 points) Garbage collection for languages like C and C++ is more complicated because casting and unions make pointers difficult to recognize. What changes are needed in the garbage collector you proposed in (a) to support languages like C and C++?

   (c) (2 points) Some languages, like Java, allow simultaneous execution of multiple threads. In a multi-threaded run-time system, it is advantageous to allow the garbage collector to run concurrently with one or more program threads. What changes are needed for your garbage collector to make it concurrent?

2. (6 points) *Subtyping.*

   (a) (3 points) Describe the standard subtyping rules for function types.

   (b) (3 points) Why is a type of mutable cells (ML reference cells, or records with a single assignable field) not subtypable?

3. (8 points) *Method Lookup.*

   In a multiple inheritance language such as C++, one needs to put “deltas” in the virtual function table along with pointers to functions. Explain why the “deltas” are needed in the virtual function table. Illustrate with an example.

4. (10 points) *Implementing Exceptions*

   One way of implementing exceptions is to make a table mapping exception names to code for handlers, for each scope, and store this table on the run-time stack. This has little performance impact at run-time, unless an exception is raised, since the tables can be determined at compile time. However, when an exception is raised, there is some cost. Specifically, if the current activation record contains a handler, control is transferred to this handler. If not, then the exception will have to be "re-raised" in another scope.

   (a) If an exception is raised and there is no handler in the current scope, which pointer in the activation record should be used to find the next scope?

   (b) Can the compiler determine the number of pointers to follow, for a given exception and scope, at compile time? Explain why or why not.

   (c) Optimizing compilers often change the order of instructions for various reasons. Why do languages with exceptions make this kind of optimization more difficult?

   (d) What information would a compiler like to know, at compile time, about a given expression such as a function call, in order to reorder instructions?

   (e) Does Java provide this information for method calls? If so, for all exceptions or just some exceptions?