Comprehensive Exam: Programming Languages Autumn 2003

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?

   Answer: Mark-and-sweep sets all mark bits to 0, then begins a mark phase by marking all data reachable from pointers on the run-time stack. If an item is reachable, its mark bit is set to 1. All locations not marked reachable are then returned to the available storage list. This algorithm only collects garbage, since the program can only use reachable data. However, some data that is never accessed by the program will not be collected.

   (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++?

   Answer: One approach is conservative garbage collection. In conservative garbage collection, data that may be used as a pointer is treated as a pointer. For example, an integer variable containing a value that is a valid address can be treated as if it is a pointer. Since it is hard to tell what locations might be accessed using pointer arithmetic, conservative garbage collection may not work properly for some programs. But conservative garbage collectors have been used successfully by many programmers.

   (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?

   Answer: Some kind of locking is needed to keep the program from writing to areas that are under examination by the garbage collector. However, concurrent reads can be allowed for appropriate garbage collection algorithms.

2. (6 points) *Subtyping.*

   (a) (3 points) Describe the standard subtyping rules for function types.
   Answer: If $A <: B$ and $C <: D$, then $B \to C <: A \to D$.

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

   Answer: Assume $A <: B$. Then $ref(A) <: ref(B)$ is not right because a program writing into a B cell cannot use an A cell instead. Conversely, $ref(B) <: ref(A)$ is not right because a program reading an A cell may generate a type error if given a B cell instead.

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.

Answer: Suppose class C is derived from A and B. When an A pointer is assigned to a C object, the pointer may point to the top of the object. However, to make the data access and virtual function lookup work properly, something different may happen when a B pointer is assigned to the C object. In this case, the pointer will point to a different part of the object, where a second vptr pointing to a second vtable is located. When a C object is treated as a B object, the second vtable allows the standard virtual function offsets associated with class B to be used. The "deltas" are used when a virtual function defined in class C appears in the second B vtable for class C. In this case, the virtual function will be expecting the `this` pointer to point to the top of the object. Therefore, a "delta" will have to be added to the pointer to compute the right value for the `this` argument to the virtual function. A picture as in the *C++ Annotated Reference Manual* or *Concepts in Programming Languages* will help.

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?

Answer: The control link is used to find the previous record on the run-time stack.

(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.

Answer: No, the number of control pointers to follow depends on the calling sequence.

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

Answer: A jump resulting from an exception could prevent an instruction from being executed. This would cause a problem if an instruction that should have been executed before the jump was placed later by an optimizing compiler.

(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?

Answer: Whether the function can raise/throw an exception, and if so, which ones.

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

Answer: Yes, for programmer-defined exception classes, but not for Runtime_Exception or Error