Comprehensive Exam: Programming Languages Autumn 2007

This is a 60-minute closed-book exam and the point total for all questions is 60.

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 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	# 5	Total
Score						
Max	15	10	11	12	12	60

1. (*15 points*) .... Short Answer

Answer each question in a few words or phrases.

(a) (*3 points*) Can a language that does not allow explicit deallocation and uses a correct implementation of garbage collection have dangling pointers? Justify your answer.

(b) (*3 points*) What is a closure and what problem does it solve?

(c) (*3 points*) Explain the difference between subtyping and inheritance (in at most two sentences).

(d) (*3 points*) Assume that `Rectangle` is a subtype of `Shape`, written `Rectangle` <: `Shape`. Which of the following subtype relationships hold in principle?

i. (`Shape` → `Rectangle`) <: (`Rectangle` → `Rectangle`)

ii. (`Rectangle` → `Shape`) <: (`Rectangle` → `Rectangle`)

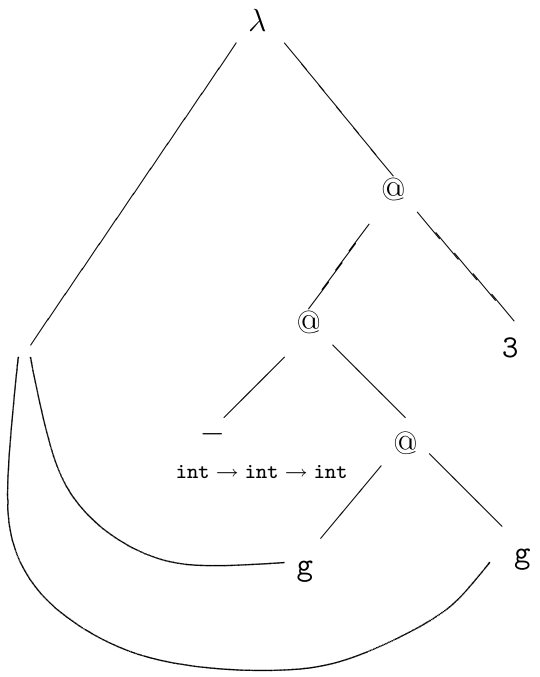
(e) (*3 points*) Why do static fields of a Java class have to be initialized when the class is loaded? Why can't we initialize static fields when the program starts?

2. (*10 points*) .... Type Inference on Parse Graph

Use the parse graph below to follow the steps of the ML type inference algorithm on the function declaration

```
fun f(g) = g(g) - 3;
```

Write the type associated with each node of the graph, as the type inference algorithm proceeds from the bottom of the graph up towards the root. What is the output of the type checker?



3. (*11 points*) .... Parameter passing comparison

For the following Algol-like program, write the number printed by running the program under each of the listed parameter passing mechanisms.

In *pass-by-value-result*, also called call-by-value-result and copy-in/copy-out, parameters are passed by value, with an added twist. More specifically, suppose a function `f` with a pass-by-value-result parameter `u` is called with actual parameter `v`. The activation record for `f` will contain a location for formal parameter `u` that is initialized to the R-value of `v`. Within the body of `f`, the identifier `u` is treated as an assignable variable. On return from the call to `f`, the actual parameter `v` is assigned the R-value of `u`.

```
begin
   integer i;

   procedure pass ( x, y );
      integer x, y;  // types of the formal parameters
      begin
         x := x + 3;
         y := x + 5;
         x := y;
         i := i + 7
      end

   i := 1;
   pass (i, i);
   print i
end
```

(a) (*3 points*) pass-by-value

(b) (*4 points*) pass-by-reference

(c) (*4 points*) pass-by-value/result

4. (*12 points*) .... Phantom Members

A C++ class may have virtual members that may be redefined in derived classes. However, there is no way to "undefine" a virtual (or non-virtual) member. Suppose we extend C++ by adding another kind of member, called a *phantom* member, that is treated as virtual, but only defined in derived classes if an explicit definition is given. In other words, a "phantom" function is not inherited unless its name is listed in the derived class. For example, if we have two classes

```
class A {
...
public:
   phantom void f(){...}
   ...
};
class B : public A {
...
public:
   ... /* no definition of f */
};
```

then `f` would appear in the `vtbl` for `A` objects and, if `x` is an `A` object, `x.f()` would be allowed. However, if `f` is not declared in `B`, then `f` might not need to appear in the `vtbl` for `B` objects and, if `x` is a `B` object, `x.f()` would not be allowed.

(a) (*8 points*) Are phantom members consistent with the design of C++, or is there some general property of the way the language is designed and implemented that would be destroyed? If so, explain what this property is, why it is important, and why it is destroyed.

(b) (*4 points*) JavaScript does allow a method of an object to be removed. Explain why this is consistent with the design goals and implementation mechanisms for JavaScript, referring to your answer to part (a) as appropriate.

5. (*12 points*) .... Race Conditions

The program on the following page contains two classes: `RaceInducer` creates a field `counter` that is the source of data races, and field `counter` is an object of class `DoubleCounter`, which supports methods `incrementBoth` and `getDifference`. For each of the following questions, justify your answer *briefly.*

(a) (*3 points*) If `incrementBoth` and `getDifference` were never allowed to execute at the same time, what would this program print?

(b) (*3 points*) In the program, `dif_value` in `run` is always either 0 or 1. Why is that? Why can't the difference exceed 1 or become negative?

(c) (*2 points*) One way to ensure that data races do not occur would be to insert synchronization primitives. For example, declaring

```
public synchronized int getDifference() {...}
public int incrementBoth() {...}
```

would prevent two threads from executing in method `getDifference` at the same time. Is this enough to ensure that `getDifference` always returns 0?

(d) (*2 points*) Is the following declaration

```
public int getDifference() {...}
public synchronized int incrementBoth() {...}
```

sufficient to ensure that `getDifference` always returns 0?

(e) (*2 points*) If the following declaration is used,

```
public synchronized int getDifference() {...}
public synchronized int incrementBoth() {...}
```

what will the output be? Explain.

```
class RaceInducer extends Thread {
//  this object is shared between all instances of this class
    static DoubleCounter counter;
    static volatile boolean done = false;

    public void run() {
        try {
            for (int i = 0; i < 1000; i++) {
                if (i % 60 == 0) {
                    // insert line break
                    System.out.println();
                }
                int dif_value = counter.getDifference();
                // prints either a '+' or a '-'
                System.out.print("+-".charAt(dif_value));
                sleep(20); // suspends the current thread
            }
            done = true;
        } catch (InterruptedException e) {
            return;
        }
    }

    // entry point into the program
    public static void main(String[] x) {
        Thread ri = new RaceInducer();
        counter = new DoubleCounter();
        try {
            ri.start();     // starts a new thread and calls run()
            while (!done) {
                counter.incrementBoth();
                sleep(30);  // suspends the current thread
            }
            ri.join();
        } catch (InterruptedException e) {
            return;
        }
    }
}

/**
 * Shared data structure.
 * */
class DoubleCounter {
    protected int x = 0, y = 0;

    public int getDifference() {
        return x - y;
    }

    public void incrementBoth() throws InterruptedException {
        x++;
        Thread.sleep(9);
        y++;
    }
}
```