Comprehensive Exam: Programming Languages Autumn 2006

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 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	# 5	Total
Score						
Max	21	9	10	12	8	60

1. (*21 points*) ..... Short Answer

Answer each question in a few words or phrases.

(a) (*3 points*) Why is memory usually separated into a stack and a heap?

(b) (*3 points*) Why is it possible to implement C without creating closures?

(c) (*3 points*) What is the difference between overloading and polymorphism?

(d) (*3 points*) Explain the difference between subtyping and inheritance.

(e) (*3 points*) If `A` <: `B` and `B` <: `C` is `A`→`B` <: `A`→`C` ? Why or why not?

(f) (*3 points*) Explain how the Java compiler treats this statement and, if the statement compiles, what happens at run time?

```
Character c = (Character) new Object();
```

(g) (*3 points*) What property of the Java architecture makes it necessary to do bytcode rewriting to get efficient method lookup?

2. (*9 points*) ..... Pointer arithmetic

Unlike most programming languages, C allows pointer arithmetic. This question asks you about advantages and disadvantages of this language feature, with some parts of the question referring to the following excerpt from a JPEG decoder that performs a discrete cosine transformation using a loop that iterates through an array of elements.

```
void jpeg_fdct_ifast (DCTELEM *data)
{
   ...
   DCTELEM *dataptr;
   int ctr;
   ...
   /* Pass 1: process rows. */
   dataptr = data;
   for (ctr = DCTSIZE-1; ctr>= 0; ctr--) {
      tmp0 = dataptr[0] + ... + dataptr[DCTSIZE-1];
      ...
      dataptr += DCTSIZE;  /* advance pointer to next row */
   }
   ...
```

(a) (*2 points*) Name two disadvantages of pointer arithmetic. In answering the question, consider compiler features, desirable language properties, or language run-time features that are difficult or impossible in languages with pointer arithmetic.

(b) (*3 points*) One argument sometimes given in favor of pointer arithmetic is efficiency. In the code example above, each loop iteration processes one row of a DCTSIZE×DCTSIZE matrix whose entries are stored sequentially in memory. Iteration number `ctr` accesses the elements `data`[`DCTSIZE` $*$ ((`DCTSIZE` $-$ `1`) $-$ `ctr`)] to `data`[`DCTSIZE` $*$ ((`DCTSIZE` $-$ `1`) $-$ `ctr`) $+$ (`DCTSIZE` $-$ `1`)]. Explain why this loop above might be more efficient than a loop that uses expressions of the form `data`[`DCTSIZE` $*$ ((`DCTSIZE` $-$ `1`) $-$ `ctr`) $+$ `i`] to index into the data array.

(c) (*2 points*) Suppose that the omitted code inside the loop accesses memory using $*$`dataptr` $+$ `j`, with `j` set by a `for` loop that only gives `j` values between 0 and `DCTSIZE` $-$ 1. Explain why all memory accessed by the loop is memory that is allocated to the program, assuming that the function parameter `data` points to a heap-allocated region of `DCTSIZE` $*$ `DCTSIZE` locations.

(d) (*2 points*) Consider the more general setting of a pair of nested loops of the following form:

```
for (i = 0; i < imax; i++) {
   for (j = 0; j < jmax; j++) {
      ...
      ... data[i*A+j*B] ...
      ...
   }
}
```

i. (*1 points*) Assuming a simple-minded compiler with no loop-related optimizations, what arithmetic operation is used to compute the location `data[i*A+j*B]` in each execution of the inner loop ?

ii. (*1 points*) Suppose `A` and `B` are constants and `i` and `j` are only used inside array indices. What could an optimizing compiler do to simplify the arithmetic that is needed on each execution of the inner loop?

3. (*10 points*) ..... Scope and parameter passing

(a) (*4 points*) Consider this simple program.

```
1 { int x;
2   int y;
3   int z;
4   x := 3;
5   y := 7;
6   { int f(int y) { return x*y };
7     int y;
8     y := 11;
9     { int g(int x) { return f(y) };
10      { int y;
11        y := 13;
12        z := g(2);
13      };
14    };
15  };
16 }
```

What value is assigned to `z` in line 12 under static scoping?

What value is assigned to `z` in line 12 under dynamic scoping?

(b) (*6 points*) What are the values of `y` and `z` at the end of the following block under the assumption that parameters are passed:

i. call by value

ii. call by reference

iii. call by value-result

```
{ int y;
  int z;
  y := 7;
  { int f(int x) {
       x := x+1;
       y := x;
       x := x+1;
       return y
    };
    int g(int x) {
       y := f(x)+1;
       x := f(y)+3;
       return x
    }
    z := g(y);
  };
}
```

4. (*12 points*) ..... Dynamic Lookup

Answer each of the following questions about dynamic lookup in Smalltalk, C++, and Java in a few concise, carefully worded and legible sentences. Focus on the main points that distinguish these languages.

(a) (*3 points*) Briefly describe the C++ implementation of dynamic lookup, mentioning the key data structure(s).

(b) (*3 points*) Briefly describe the Smalltalk implementation dynamic lookup, mentioning the key data structure(s).

(c) (*2 points*) Which implementation would you expect to run faster? Why?

(d) (*2 points*) Could Smalltalk use the C++ implementation of dynamic lookup? Why or why or not?

(e) (*2 points*) How is Java similar or different from each of these implementations?

5. (*8 points*) ..... Concurrency and Parallelism

Graphics processors (GPUs) are one example of a parallel processor that nearly everyone has on their desktop. The frame buffer is a region of memory on the GPU that stores pixel colors. In a single rendering pass, the frame buffer can only be written to and can not be read from. Conversely, the rest of the memory on the graphics card can be read from at any time but can not be written to. After a rendering pass, the frame buffer can be copied into another part of memory where the values can be read in a subsequent rendering pass. These restrictions are enforced by the device driver for the GPU.

(a) (*3 points*) What general problem with concurrent memory access do we solve by having a write-only frame buffer with all other memory being read-only?

(b) (*3 points*) Java concurrency primitives allow us to write a multi-threaded software simulation of the parallel processors in a GPU. Suppose we wanted to use our GPU simulator to see what happens when we change the frame buffer from having write-only access to having both read and write access.

Fill in the blanks in the following Java FrameBuffer class. You may need to write more than word per blank. Explain in one sentence.

```
class FrameBuffer {
  private FBdata[] pixels;
  ...

  public ____________________ read( int addr ) {
    return pixels[addr];
  }

  public ____________________ write( int addr, FBdata data ) {
    pixels[addr] = data;
  }

  ...
}
```

(c) (*2 points*) After running several experiments with your Java GPU simulation, you decide that allowing frame buffer reads would be a very useful feature. (This is actually true for some graphics algorithms.) There is a condition, restricting which threads get to write to each pixel, that would allow writes without introducing concurrency problems. Since a hardware implementation of pixel locking is too expensive to be feasible, what condition on access to pixels or groups of pixels could a device driver enforce to prevent concurrency problems? Explain.