CS 342 — Principles of Programming Languages December 19, 1989

Final Exam

Please write your name here:

Before you begin, please take a moment to look over the entire exam so that you can budget your time.

For each of the multiple choice questions, please circle the correct answer. You will lose one point for an incorrect answer on a multiple-choice question, so don't guess randomly.

Answers to essay questions will be judged on their completeness and clarity. However, please do not feel obligated to fill up all the empty space on the page. Conversely, do not feel limited by the space on a page, since you may continue an answer on the back of a page if you wish.

problem	subject	points	score
1	Structure principle	4	
2	Type checking	4	
3	Aliasing	3	
4	Unification	4	
5	Exceptions	5	
6	LISP's syntax	5	
7	Lambda abstraction	5	
8	Garbage collection	5	
9	Logic programming	5	
10	Information Hiding	15	
11	Inheritance	15	
12	Smalltalk	8	
13	Scheme	10	
14	Prolog	12	
Total		100	

This exam (only) counts for 15% of your final grade.

1. (4 points) What is wrong with the following argument: the **goto** statement is harmful because it allows one to write programs that are difficult to follow.

   (a) Nothing, the argument is valid.

   (b) It should say that the **goto** "encourages" programmers to write programs that are difficult to follow.

   (c) One can write programs that are difficult to follow in any language.

2. (4 points) Which of the following best compares the type system of Smalltalk-80 to that of CLU?

   (a) There is no type checking in Smalltalk, which makes the language simpler. CLU is statically type checked, so CLU is more secure than Smalltalk. However, Smalltalk provides for information hiding and data abstraction because it is object-oriented.

   (b) Smalltalk has static type checking as does CLU. However, because types in Smalltalk correspond to classes, type checking in Smalltalk is much more flexible than in CLU.

   (c) Smalltalk has dynamic type checking, which consists of ensuring that the message sent to an object is one that is defined in that object's class. This dynamic type checking is more flexible than CLU's static type checking, but is less efficient at run-time.

3. (3 points) In a purely applicative programming language, can a program observe aliasing? (That is, can one write a procedure that prints "1" if two arguments are aliased and "0" otherwise?)

   (a) Yes.

   (b) No.

4. (4 points) Which of the following is a most general unifier of the terms `foo(X,Y,bar(3))` and `foo(t,Z,bar(W))`?

   (a) the substitution s defined by $s(\texttt{X}) = \texttt{t}$, $s(\texttt{Y}) = \texttt{a}$, $s(\texttt{Z}) = \texttt{a}$, $s(\texttt{W}) = \texttt{3}$.

   (b) the substitution s' defined by $s'(\texttt{X}) = \texttt{t}$, $s'(\texttt{Y}) = \texttt{U}$, $s'(\texttt{Z}) = \texttt{U}$, $s'(\texttt{W}) = \texttt{3}$.

   (c) the term `foo(t,U,bar(3))`.

   (d) the term `foo(X,U,bar(W))`.

5. (5 points) CLU's exception handling mechanism illustrates which of the following principles? (Circle all that apply.)
   (a) automation
   (b) preservation of information
   (c) regularity
   (d) simplicity
   (e) none of the above
6. (5 points) Which of the following correctly describes LISP's syntax? (Circle all that apply.)
   (a) LISP's fixed format syntax makes it difficult to write programs.
   (b) LISP's syntax is more complex than Smalltalk's.
   (c) LISP's syntax is simpler than Smalltalk's.
   (d) LISP's syntax makes it easy to treat programs as data.
   (e) LISP does not have precedence rules, and hence it is easy to be confused about the order of evaluation of subexpressions.
   (f) LISP's syntax violates the zero-one-infinity principle, because the `if` expression takes three (3) arguments.
7. (5 points) Which of the following accurately describe `lambda` abstractions in Scheme? (Circle all that apply.)
   (a) Lambda abstractions are only needed when one needs to pass a function as an argument but does not wish to give the function a name.
   (b) `lambda` is the same as `quote`, as `lambda` just returns its arguments without evaluating them.
   (c) Lambda abstraction can be used to express delayed evaluation, recursion, and all kinds of functionals.
   (d) `lambda` allows one to dynamically construct a function and remember its environment of definition.
   (e) Activation records can always be allocated on the run-time stack, because the function returned by a `lambda` can only be called from a more deeply nested function.
   (f) Activation records must be allocated on the heap in general, because the function returned by a `lambda` may refer to the environment in which the `lambda` was executed at any later point in the program's execution.

8. (5 points) Which of the following correctly describes a language with garbage collection? (Circle all that apply.)

   (a) In a language with garbage collection, there can be no aliasing.

   (b) In a language with garbage collection, there can be no dangling references.

   (c) A language with garbage collection is more secure than it would be with explicit erasure.

   (d) A language with explicit erasure is more secure than one with garbage collection, because programmers can state their intentions about what storage should be reclaimed.

   (e) A language with garbage collection generally makes more efficient use of the computer at run-time than a language with explicit erasure.

9. (5 points) Which of the following are features of Prolog that make it a less-than-ideal logic programming language? That is, which of the following are *bad* features of Prolog from the viewpoint of logic programming? (Circle all that apply.)

   (a) Logical variables.

   (b) Unification.

   (c) Depth-first search.

   (d) Conjunctive queries (using `,`).

   (e) Cut (`!`).

   (f) Assert and retract.

10. (15 points) Briefly describe how Smalltalk-80 allows programmers to hide information (such as the representations of objects). What are the significant costs and benefits of these mechanisms? (Hint: you may want to discuss how these mechanisms differ from CLU's.)

11. (15 points) What are the significant costs and benefits of the inheritance mechanism of Smalltalk-80?

12. (8 points) Consider the following Smalltalk program, written in the Little Smalltalk dialect we used in class.

Suppose that the class `C` has the following instance methods (separated here by blank lines).

```
"Class C"
m1
   ^ self m2

m2
   ^ 'C'

m3
   ^ 'm3'
```

Suppose that the class `D` is defined as a subclass of `C`, with the following instance method definitions.

```
"Class D"
m2
   ^ super m3

m3
   ^ 'E'
```

(a) Given the above class definitions, what is the result (if any) of the following Smalltalk expression?

```
(C new) m1
```

(b) Given the above class definitions, what is the result (if any) of the following expression?

```
(D new) m1
```

13. (10 points) Consider the following Scheme program.

```
(define (m x y)
  (lambda (f) (f x y)))

(define (a z)
  (z (lambda (p q) p)))

(define (r z)
  (z (lambda (p q) q)))
```

(a) Given the above definitions, what is the result of the following expression?

```
(let ((x 5) (y 6))
  (a (r (m 3 (m 1 2)))))
```

(b) Given the above definitions, what would be the result of the following expression if Scheme used dynamic scoping?

```
(let ((x 5) (y 6))
  (r (m 1 2)))
```

14. (12 points) Consider the following Prolog rules and facts.

```
algebraic(X).
classes(smalltalk).
loops(clu).
loops(fortran).
loops(smalltalk).
messages(smalltalk).
modules(clu).
procedures(clu).
procedures(fortran).
procedures(prolog).
procedures(smalltalk).
type_checking(clu,static).
type_checking(clu,name).
type_checking(fortran,static).
type_checking(smalltalk,dynamic).
abstraction(L) :- loops(L), procedures(L), information_hiding(L).
automation(L) :- algebraic(L), loops(L).
information_hiding(L) :- modules(L), type_checking(L,name).
information_hiding(L) :- classes(L), messages(L),
                         type_checking(L,dynamic).
```

Please answer the following with respect to the above rules and facts.

(a) Give the output of the query `?- abstraction(clu).`

(b) Give the output of the query `?- abstraction(ada).`

(c) Give *all* answers (i.e., all substitutions, if any) to the query

`?- abstraction(Y), automation(Y).`