

# CS Qualifying Exam Spring 2003 Programming Languages

by Adriana Compagnoni

January 2003

## Important remarks

- This test has 3 questions.
- The grading scale is: A= 90-100, B= 80-90, C=70-80, D=60-70, else Fail.
- This test is closed book, closed notes, and closed laptops.
- If you find any question unclear, please specify your understanding of what is being asked.

## Subset of Scheme You May Use

Unless otherwise specified, when defining procedures you may only use: helping procedures that you define yourself, comments, and the procedures and keywords that are included in the following list. The notation `c...r` means `caar`, `cadr`, `cddr`, etc.

```
', #t, #f, *, +, -, /, <, <=, =, >=, >,
and, andmap, append, apply, boolean?,
car, cdr, c...r, char? cond, cons,
define, display, else, eq? equal?, eqv?, error, if, let, letrec,
lambda, list, length, list?, map, newline, not, null?, number?,
or, pair?, procedure?, quote,
string, string?, string=?, string-append,
string-ci=?, string-length, string-ref,
string->list, string->number,
string->symbol, substring, symbol?
vector, vector, vector?, vector-length,
vector->list, vector-ref, zero?
define-datatype, cases.
```

1. (20 points) Consider the following definition of binary trees.

```
(define-datatype bintree bintree?
  (leaf-node
    (datum number?))
  (interior-node
    (key symbol?)
    (left bintree?)
    (right bintree?)))
```

Implement a procedure `bintree-to-list` for binary trees, so that

```
(bintree-to-list (interior-node 'a (leaf-node 8) (leaf-node 5)))
```

returns the list

```
(interior-node
  a
  (leaf-node 8)
  (leaf-node 5))
```

2. (20 points) Trace the following program under call-by-reference.

```
begin
  set a = 4;
  set b = 7;
  let p = proc(q, r, s)
    begin
      set q = 8;
      set r = s;
      set a = s;
      set s = 5;
      set s = b;
    end
  in
    begin
      p(a+2, b, b)
      write(a, b)
    end
end
```

3. Consider the following grammar defining the concrete syntax `<exp>` and the corresponding datatype defining the abstract syntax:

```
<exp> ::= <identifier>
        | <boolean>
        | (if <boolean> <exp> <exp>)
        | (lambda (<identifier> <type-exp>) <exp>)
        | (<exp> <exp>)
```

```
(define-datatype exp exp?
  (var-exp (var symbol?))
  (bool-exp (bool boolean?))
  (if-exp (test exp?)
          (then exp?)
          (else exp?))
  (lambda-exp
   (id symbol?)
   (type type-exp?)
   (body exp?))
  (app-exp
   (rator exp?)
   (rand exp?)))
```

- (a) (20 points) Define the BNF and corresponding datatype for type expressions `<type-exp>`
- (b) (40 points) Write a Scheme procedure (`type-of-expression exp tenv`) that computes the type of expression `exp` in type environment `tenv`. Write any auxiliary procedure that you may need.