

CS Qualifying Exam Spring 2002 Programming Languages

by Adriana Compagnoni

22 January 2002

Important remarks

- This test has 5 questions of 20 points each.
- 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. (a) Write the BNF for multi-lists of numbers.
 (b) Write a procedure (`multi-increase-even ml`), which given a multi list of numbers `ml` returns a list of numbers where the even numbers in `ml` have been increased by 1.
 For example,


```
> (multi-increase-even '(1 2 3 4 5))
(1 3 3 5 5)
> (multi-increase-even '(1 (2 3) ((4)) 5))
(1 (3 3) ((5)) 5)
> (multi-increase-even '())
()
>
```
2. Write half a page describing and comparing interpreters and compilers considering similarities and differences.
3. This question is about the differences in the parameter-passing mechanisms call-by-value, call-by-reference, and call-by-value-result.
 - (a) Trace the following program under call-by-value
 - (b) Trace the following program under call-by-reference
 - (c) Trace the following program under call-by-value-result

```
begin
  set a = 1;
  set b = 2;
  set c = 3;
  let p = proc(q, r, s, t)
    begin
      set q = 8;
      set r = t;
      set a = s;
      set t = 5;
      set s = c;
      set c = b
    end
  in
    begin
      p(a+2, b, b, c);
      write(a, b, c)
    end
end
```

Hint: `write(a, b, c)` has different output in all 3 cases.

4. Consider the following grammar for the language of programs and statements, where expressions are left unspecified for the purpose of this exercise.

```

<program> ::= <statement>

<statement> ::= <identifier> = <expression>
              | print (<expression>)
              | {{<statement>}*(;)}
              | if <expression> <statement> <statement>
              | while <expression> do <statement>
              | do <statement> while <expression>
              | var {<identifier>}*(,); <statement>

```

Where `{{<statement>}*(;)}` is a sequence of statements separated by ‘;’, and similarly for `{<identifier>}*(,)`

Complete the procedure `execute-statement` in the following interpreter.

```

(define execute-program
  (lambda (pgm)
    (cases program pgm
      (a-program (statement)
        (execute-statement statement (init-env))))))

(define execute-statement
  (lambda (stmt env)
    (cases statement stmt
      (assign-statement (id exp)
        ...)
      (print-statement (exp)
        ...)
      (compound-statement (statements)
        ...)
      (if-statement (exp true-statement false-statement)
        ...)
      (while-statement (exp statement)
        ...)
      (do-while (statement exp)
        ...)
      (block-statement (ids statement)
        ...)
    )))

```

5. Assuming that `zero?` has type `int -> bool`, `sub1` has type `int -> int` and `*` has type `int x int -> int`, calculate the type of the following expression. Justify your answer giving types to all sub-expressions.

```

letrec
  int fact (int x) =
    if zero? (x) then 1 else *(x, fact sub1(x))
in (fact 5)

```