Faculty of Science
Final Examination
Computer Science COMP-302B
Programming Languages and Paradigms

Examiner: Prof. N. Friedman — **Thursday, April 14, 2005**
Associate Examiner: Prof. C. Verbrugge — **9:00-12:00**

Instructions:
This examination is open book. You may use any books or notes you have.
You may not use any calculators, computers or electronic aids.
Answer all questions in the exam book.
You may keep the exam.

This examination comprises 7 pages (including the title page).

1. (10 marks)

Write a definition for a Scheme procedure, freq, that takes a list of values as its argument. It computes the frequency of occurrence of each value in the list. That is, the result is a list of pairs (lists of length 2). Each pair consists of a value from the list and the number of times it occurs in the list.
For example:

```
> (freq '(a b c a a b c b b b c a))
((c 3) (b 5) (a 4))
> (freq '())
()
> (freq '(a a a))
((a 3))
> (freq '(1 2 3 4 1 2 3 1 2 1))
((4 1) (3 2) (2 3) (1 4))
```

2. (10 marks)

Consider the following grammar for the one variable lambda expressions.

```
<exp> ::= <symbol>
        |  ( lambda ( <identifier> ) <exp> )
        |  ( <exp> <exp>)
```

Write a procedure, max-depth, that takes a symbol, x, and a lambda expression and returns the maximal lexical depth of any reference to that variable in the expression.

3. (10 marks)

The binary operator **and** can be implemented in one of two different ways. In a language that uses **full evaluation**, both operands are always evaluated and a value representing *true* is returned if both of the operands evaluate to *true*. Otherwise a value representing *false* is returned.
In a language that uses **short circuit evaluation**, the first operand is evaluated. If it evaluates to *false*, the result *false* is returned (and the second operand is not evaluated). If it evaluates to *true*, the second operand is evaluated and its value is returned.

a) Suppose you require an **and** operator that uses full evaluation but Scheme uses short circuit evaluation. Write a Scheme procedure `full-and` that implements an **and** operator using full evaluation.

b) Suppose you require an **and** operator that uses short circuit evaluation but your version of Scheme uses full evaluation. Write a Scheme procedure `short-and` that implements an **and** operator using short circuit evaluation. (Hint use a lazy evaluation technique.)

4. (10 marks)

Consider the following expression in the language defined in class:

```
let x = 3
    y = 4
    z = 5
in let x = 13
       f = proc (a b) -(+(a,b), x)
   in let y = 24
          g = proc (a b) +((f a x), y)
      in (g x y)
```

a) Assume that the language uses static scoping. What is the result of this expression? Draw a representation of the environments that exist just before the call to procedure `f` terminates.

b) Assume that the language uses dynamic scoping. What is the result of this expression? Draw a representation of the environments that exist just before the call to procedure `f` terminates.

5. (10 marks)

The following grammar was used in class to define expressions.

```
<exp> ::= <number>
        | <identifier>
        | <primitive> <operands>
        | if <exp> then <exp> else <exp>
        | let {<decl>}* in <exp>
        | proc <varlist> <exp>
        | (<exp> {<expression}*)
```

The following is the abstract syntax for this grammar.

```
a-program (exp)
lit-exp (datum)
var-exp (id)
primapp-exp (prim rands)
if-exp (test-exp then-exp else-exp))
let-exp (ids rands body)
proc-exp (ids body)
app-exp (rator rands)
```

We gave a direct semantics for local definitions in class. Alternatively, the semantics for local definitions could be described in terms of anonymous procedure applications (as we did in Scheme). Write a syntax expand program that takes the AST of a program in the grammar and replaces all let expressions by equivalent procedure application expressions.

6. (15 marks)

Add the following construct to the language defined in class that includes assignment and sequencing (begin).

```
<exp> ::= do <exp> with <decls> end
<decls> ::= <decl> {; <decl>}*
```

Assume that the following is the abstract syntax for this construct.

```
do-with (body decls)
decl (var exp)
```

The semantic meaning of this construct is the following. The declarations in the list decls are processed sequentially. Then the body is evaluated in an environment that has the bindings for all the declarations. The result of the body is the result of the whole expression.

Sequential processing for declarations means that the expression in the first declaration in the list is evaluated in the original environment, then its binding is added to the environment used to process the remaining declarations. For example, in the defined language we would have

```
do x with x = 3 end => 3
do list(b,c,d) with a = 1;b = +(a,1);c = +(b,1);d = +(c,1) end
    -> (2 3 4)
do list(x,y) with x = 7;y = -(x,2) end => (7 5)
do begin set x = +(x,y);*(x,y) end with x = 7;y = -(x,2) end -> 60
```

Extend the interpreter, in eval-exp to handle this construct. Assume the parser has been extended to produce the appropriate AST. You only have to write the code for the do-with branch of the interpreter, as well as any auxiliary procedures that you call.

7. (15 marks)

Consider the following expression in the language defined in class. Assume the language has been augmented with a print command.

```
let x = 7
    a = 5
 in let a = x
        b = -(x, a)
        c = a
     in begin
          set a = 3;
          print (a, b, c, x);
          set c = 1;
          print (a, b, c, x)
        end
```

a) What is the output of this expression?

b) In class, we gave a semantics for local definitions in terms of extending environments. It is possible to describe the semantics in terms of procedure applications, as we did in Scheme question 5 above. Write an equivalent expression that uses anonymous procedures applications but does not use any local definitions.

c) Draw the environments created when this expression is evaluated. (Assume, as we did in class, that parameters are passed by value).

d) Suppose that we use call by reference to implement parameter passing. Draw the environments created when evaluating the expression you wrote in part (b). Indicate which values are direct targets and which are indirect targets.

e) What is the output of the expression when call by reference is used?

f) Draw the environments that would be created when evaluating the expression you wrote in part (b) if call by name were to be used for parameter passing. Indicate which values are direct, indirect or thunk targets.

g) What is the output of the expression when call by name is used?

8. (10 marks)

Consider the following expression in the language developed in class.

```
class foo extends object
  field i
  method initialize(n) set i = n
  method geti() i
  method showi() send self geti()
  method seti(x) begin set i = x; i end
class bar extends foo
  field i
  method initialize(n)
    begin
     set i = n;
     super seti (+(n,2))
    end
  method geti() i
  method seti(x) begin set i = x; i end
class baz extends bar
  field i
  method geti() i
  method showi() send self geti(i)
  method seti(x) begin set i = x; i end
let obj = new baz (7)
in list(send obj seti(3),
        send obj showi(),
        send obj seti(5),
        send obj geti(),
        send obj showi())
```

a) Assume that dynamic method dispatch is used. What is the result of this expression? what values will the three variables named i have after the expression is evaluated? Explain how you get your results.

b) Assume that static method dispatch is used. Answer the same questions with this assumption.

c) Suppose that super calls as well as other method calls use dynamic method dispatch. Answer the same questions under these assumptions.

9. (10 marks)

Object-Oriented languages frequently allow overloading of methods. This allows a class to have multiple methods of the same name provided they can be distinguished from one another by having distinct signatures. The signature of a method is typically the method name plus the types of its parameters. In the language we developed in class, we do not have types so the signature of a method could be defined as its name together with the number of parameters. For example, a class might have two initialize methods. One might have no parameters to be used when an initialization with a default field value is desired. The other might have a parameter for use when a specific initial value for a field is desired. Explain how you would extend our interpreter to allow overloading based on the number of method parameters. Be as specific as possible about the changes you would make to the interpreter.