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

**Examiner: Prof. N. Friedman** **Wednesday, December 15, 2004**
**Associate Examiner: Prof. C. Verbrugge** **9:00-12:00**

Instructions:
This examination is open book. You may use any books or notes including definition dictionaries.
You may not use any calculators, computers or electronic aids.
Answer all questions in the examination book.

This examination comprises 6 pages (including the title page).

## 1. (10 marks)

Consider the following Scheme expression.

```
((lambda (a b)
     ((lambda (x)
        (find x b))
      (map (lambda (y) (second y)) (rest b))))
   a (second z))
```

a) List the entire set of bound variables used in the above expression.
b) List the entire set of free variables used in the above expression.
c) Rewrite the expression replacing each bound variable by its lexical address. Each free variable is replaced by a lexical address which uses the symbol `free` for its position and includes its depth.

## 2. (10 marks)

Consider the following grammar for type expressions:

```
<type> ::= <symbol>
         |  ( -> ( <type-list> ) <type> )
         |  ( list-of <type> )
<symbol> ::= S | T | U
<type-list> ::= <empty> | <type> <type-list>
<empty> ::=
```

a) Write a leftmost derivation of `( -> ( (list-of S)) T)`
b) Give a datatype definition for representing the abstract syntax of this grammar.
c) Write a parse procedure that takes a type expression and returns the AST representation of the expression.

## 3. (10 marks)

It is possible to represent a two dimensional `n`x`m` matrix as a list of `n` rows where each row is a list of `m` numbers. (Assume that the rows are numbered from `0` to `n-1`). A matrix is said to be upper triangular if the first `i` values in row `i` are zero. Write a predicate **`triangular?`** that determines whether or not a given matrix is upper triangular. Do not assume that the matrix is square and make sure that all of the rows are of the same length.

## 4. (10 marks)

It is possible to think of a schedule as a mapping from time slots to events. For simplicity, we can assume the time slots are hourly and represent them by integers from 0 to 23.

a) Using a procedural representation of data, we could represent a schedule as a Scheme procedure. Implement the following operations using this representation. **Note that you will receive zero for this question if you use any mutators or global definitions.**

> **Empty-schedule** – takes no arguments. It returns a new empty schedule.
> **Add-event** – takes three arguments, a time, an event and an existing schedule. It returns a new schedule with the new association.
> **Event-at** – takes two arguments, a schedule and a time. It returns the event scheduled for that time. If there is no event scheduled, it returns the symbol 'empty

The following examples show how your code should work.

```
(define COMP302 (add-event 14 'COMP302 (empty-schedule)))
(define Wednesday (add-event 16 'seminar COMP302))

(event-at Wednesday 16) => seminar
(event-at Wednesday 11) => empty
(event-ad (empty-schedule) 18) => empty
```

b) Transform your procedural representation into a variant record representation by giving the define-datatype specification of the representation and implementing the operations using this representation.

## 5. (10 marks)

Consider the following expression in the language defined in class:

```
let f = proc (n) add1(n)
in let f = proc (n)
             if zero?(n)
             then 1
             else *(n,(f sub1(n)))
   in (f 4)
```

a) Assume that the language uses static scoping. What is the result of this expression? Draw a representation of the environments that are created when evaluating this expression.
b) Assume that the language uses dynamic scoping. What is the result of this expression? Draw a representation of the environments that are created when evaluating this expression.

## 6. (15 marks)

Answer the questions about the following expression in the language defined in class. **When you are asked to draw the bindings, indicate whether the values bound to the variables are direct targets, indirect targets or thunk targets.**

```
let x = 3
    y = 5
    z = 7
in let p = proc (a, b, c)
            begin
             set a = +(a, a);
             set x = +(x, b);
             set c = +(a, b);
             list (a, b, c, x, y, z)
            end
   in (p x +(x,y) z)
```

a) What is the value of this expression if call-by-value is used to pass the parameters?
b) Draw the bindings that exist for all of the variables in the environment just before the procedure returns when call-by-value is used.
c) What is the value of this expression if call-by-reference is used to pass the parameters?
d) Draw the bindings that exist for all of the variables in the environment just before the procedure returns when call-by-reference is used.
e) What is the value of this expression if call-by-name is used to pass the parameters?
f) Draw the bindings that exist for all of the variables in the environment just before the procedure returns when call-by-name is used.
g) What is the value of this expression if call-by-need is used to pass the parameters?
h) Draw the bindings that exist for all of the variables in the environment just before the procedure returns when call-by-need is used.

## 7. (10 marks)

Ass the following syntax to the language defined in class. This new kind of expression defines a new type of logical operation.

```
<exp> ::= implies <exp> => <exp>
```

The abstract syntax is

```
implies-exp (left-exp right-exp)
```

The semantics is as follows. Evaluate the left expression. If it evaluates to true, then evaluate the right expression. Otherwise return 1. This is equivalent to the expression `if left-exp then right-exp else 1`.

Write a syntax-expand procedure that takes the AST of an expression in the language which includes this construct and returns an AST without any implies expressions. You must fill in the missing code below.

```
(define syntax-expand
  (lambda (exp)
    (cases expression exp
      (lit-exp (datum) . . .)
      (var-exp (id) . . .)
      (primapp-exp (prim rands) . . .)
      (implies-exp (left-exp right-exp) . . .)
      (if-exp (test-exp true-exp false-exp) . . .)
      (let-exp (ids rands body) . . .)
      )))
```

## 8. (15 marks)

Add the following syntax to the language defined in class that includes lists and assignment. This new kind of expression allows us to define variable arity procedures.

```
<exp> ::= varargsproc ( <identifier> ) <exp>
```

Assume that the abstract syntax for this construct is:

```
varargsproc-exp (id body)
```

The semantics are as follows. Evaluating this expression forms a new kind of closure that includes the identifier, body and current environment. When applied we extend the environment in the closure with a binding of the identifier to a list of the values of the arguments.

Extend the interpreter to handle this construct. Specifically, show how you modify the following parts of the interpreter.

a) the definition of datatypes for Procval
b) the apply-procval function of the ADT Procval
c) the definition of the abstract syntax of expressions
d) the evaluator (by adding an extra clause to implement the new construct)

## 9. (10 marks)

Explain how to extend our Object-Oriented languages and the interpreter to allow direct referencing and modifying of field variables. That is add the expressions

```
fieldref object field-id
```

which returns the contents of the specified field of the object, and

```
fieldset object field-id exp
```

which sets the contents of the specified field to the value of the expression.

Be as specific as possible about the changes you would make to the interpreter.