University of Illinois at Urbana-Champaign
Department of Computer Science

Midterm 2
CS 421 Programming Languages and Compilers
Spring 2006

Name: ____________________________
Netid: ____________________________

- Print your name and netid, neatly in the space provided above; print your name at the upper right corner of every page. Please print legibly.

- This is a closed book exam. No notes, books, dictionaries, or calculators are permitted.

- Write your answers in the space provided for the corresponding problem. Let us know if you need more paper.

- Suggestions: Read through the entire exam first before starting work. Do not spend too much time on any single problem. If you get stuck, move on to something else and come back later.

- If you run short on time, remember that partial credit will be given.

- If any question is unclear, ask one of us for clarification.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>15</td>
</tr>
<tr>
<td>Problem 2</td>
<td>20</td>
</tr>
<tr>
<td>Problem 3</td>
<td>15</td>
</tr>
<tr>
<td>Problem 4</td>
<td>15</td>
</tr>
<tr>
<td>Problem 5</td>
<td>20</td>
</tr>
<tr>
<td>Problem 6</td>
<td>15</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
Problem 1  Currying (15 Points)

Extend the interpreter from MP4 so that it uses currying. That is to say, a function application expression, such as "(f 1 2 3)" should be interpreted as "(((f 1) 2) 3)", and a procedure declaration "proc (x y z) ..." should be interpreted as "proc (x) proc (y) proc (z) ...".

Limit your changes to the apply-procval function. You may assume that you have two helper functions, first-n and last-n that return the first n and last n elements of a list, and that only environments (and not an environment and store setup) are being used.

```
(define-datatype procval procval?
  (closure
    (ids (list-of symbol?))
    (body expression?)
    (env environment?)))

(define apply-procval
  (lambda (proc args)
    (cases procval proc
      (closure (ids body env)
        (if (eq? (length args) (length ids))
          (eval-expression body (extend-env ids args env))
          (if (< (length args) (length ids))
            (closure (last-n ids (- (length ids) (length args)))
              body (extend-env (first-n ids (length args)) args env))))))))
```
Problem 2  Interpreter (20 Points)

2.a)  (10 Points)

Consider the following chunks of code from a call-by-need interpreter. Make the appropriate modifications to turn it into a call-by-name interpreter.

```scheme
(define eval-expression
  (lambda (exp env store)
    (cases expression exp
      (lit-exp (datum) (cons datum store))
      (var-exp (id) (get-store-help store (apply-env env id)))
      (lexvar-exp (d l) (get-store-help store (apply-env env (cons d l))))
      ...))))

(define get-store-help
  (lambda (store i)
    (let ((val (get-store store i)))
      (if (specval? val)
        (cases specval val
          (needval (exp env)
            (let* ((val (eval-expression exp env store))
              (store (set-store (cdr val) i (car val))))
              (cons (car val) store))))
        (cons val store))))))

(define eval-rands
  (lambda (rands env store)
    (eval-rand rands env store ())))

(define eval-rand
  (lambda (rands env store acc)
    (if (eq? rands ())
      (cons (reverse acc) store)
      (let* ((loc (car (get-empties store 1)))
        (store (set-store store loc (needval (car rands) env))))
        (eval-rand (cdr rands) env store (cons loc acc)))))))
```
2.b) (10 Points)

Modify the following interpreter to support a do-while statement. do-while is identical to while, except that the expression is tested after the body of the while is executed.

```
(define execute-statement
  (lambda (stmt env)
    (cases statement stmt
        ...
        (while-statement (exp statement)
          (let loop ()
            (if (true-value? (eval-expression exp env))
              (begin
                (execute-statement statement env)
                (loop))))))
        (do-while-statement (exp statement)
          (let loop ()
            (begin (execute-statement statement env)
              (if (true-value? (eval-expression exp env))
                (loop)))))))
```
Problem 3  Call-by conventions (15 Points)

For the following piece of code:

let x = 42  
y = 2778 in  
let f = proc(y) +(y, x) in  
let g = proc(x) (f x) in  
  (g y)

3.a)  (5 Points)

Draw all the environments that exist when the expression + (y, x) is being evaluated, assuming that the language is statically scoped and call-by-value.

env0 = ((x, y), (42, 2778), env0)
env1 = ((y), (2778), env1)
env2 = ((x), (2778), env1)
env3 = ((y), (2778), env1)
env4 = ((f), (<y, +(y, x), env1>, env1))
env5 = ((g), (<x, (f x), env4>, env4))

3.b)  (5 Points)

Do the same as above, but assume that the language is statically scoped and call-by-reference. Draw references as arrows pointing to the appropriate value.

store = (42, 2778, <y, +(y, x), env1>, <x, (f x), env4>)
env1 = ((x, y), (0, 1), env0)
env2 = ((y), (1), env1)
env3 = ((x), (1), env1)
env4 = ((f), (3), env1)
env5 = ((g), (4), env4)

3.c)  (5 Points)

Do the same as above, but assume that the language is dynamically scoped and call-by-reference.

store = (42, 2778, <y, +(y, x)>, <x, (f x)>)
env1 = ((x, y), (0, 1), env0)
env2 = ((f), (3), env1)
env3 = ((g), (4), env2)
env4 = ((x), (1), env3)
env5 = ((y), (1), env4)
Problem 4  Environments (15 Points)

The environment

![Environment Diagram]

is active at the point indicated by the arrow in the following code:

```plaintext
let
  incr = proc (x) <body>
in
  <===
  (incr 5)
```

4.a)  (6 Points)

Write the code and specify location where the following environment is active

![Environment Diagram]

```plaintext
let
  incr = proc (x) <body>
in
  let
    incr2 = proc (x) <body>
in
    <===
    ...
```
4.b) (9 Points)

Write the code and specify location where the following environment is active

```
letrec
  double = proc (x) <body>
  triple = proc (x,y) <body>
in
<===
...
Problem 5  Type Inference (20 Points)

For each of the following expressions, write in the type of the formal arguments, and the type of the result of each application. For each application, draw a curly bracket under the application and indicate it’s type. Indicate unspecified types, if any, by ‘a, ’b, c’ and so on. Assume ‘add1’ is of type (int → int) and ‘+’ is of type (int * int → int). Further assume that the result of the test expression in the conditional must be of bool type.

5.a) (5 Points)

\[ \text{proc ((int → bool) → int → int f, int → bool p, int x)} \]
\[ \text{if (p x)} \]
\[ \{ \text{bool } \} \]
\[ \text{then add1(x)} \]
\[ \{ \text{int } \} \]
\[ \text{else (f p x)} \]
\[ \{ \text{int } \} \]

5.b) (5 Points)

\[ \text{proc (int → int f, (int → int) → int → int g, int → bool p, int x)} \]
\[ \text{if (p add1(x))} \]
\[ \{ \text{bool } \} \]
\[ \text{then (g f x)} \]
\[ \{ \text{int } \} \]
\[ \text{else add1((f x))} \]
\[ \{ \text{int } \} \]

5.c) (5 Points)

\[ \text{let int x = 3} \]
\[ \text{int → int f = proc (int x) add1(x) in (f x)} \]
\[ \{ \text{int } \} \]

5.d) (5 Points)

\[ \text{proc (τ → int f, int → τ → int g, int → bool p, τ x)} \]
\[ \text{if (p (f x))} \]
\[ \{ \text{bool } \} \]
\[ \text{then (g 1 x)} \]
\[ \{ \text{int } \} \]
\[ \text{else add1((f x))} \]
\[ \{ \text{int } \} \]
Problem 6  Method dispatch (15 Points)

class point extends object
field x
field y
method move (dx,dy)
  begin
  set x = +(x,dx)
  set y = +(y,dy)
  send self draw ()
  end
method draw () 'pointDrawn
class circle extends point
  field radius
methods draw () 'circleDrawn
let o = new circle ()
in send o move()

6.a)  (5 points)
What is the value returned by the above program if static method dispatch is used?
  pointDrawn

6.b)  (5 points)
What is the value returned by the above program if dynamic method dispatch is used?
  circleDrawn

6.c)  (5 points)
What does the code below evaluate to?

class c1 extends object
  method initialize () 1
  method m1 () send self m2 ()
  method m2 () 12
class c2 extends c1
  method m1 () 21
  method m2 () 22
  method m3 () super m1 ()
class c3 extends c2
  method m1 () 31
  method m2 () 32
let o = new c3 ()
in send o m3 ()
  32