CS251 Programming Languages **Handout # 38**
Prof. Lyn Turbak **May 6, 2004**
Wellesley College

CS251 Jeopardy: The Home Version
The game that turns CS251 into CS25*fun*

Data

[**1**] What data structure is commonly used in interpreters to associate names with values?

[**2**] What feature in Scheme, ML, and Java is responsible for reclaiming storage used by values that are no longer accessible from the program?

[**3**] How are "sum-of-product" data structures expressed in (i) OCAML and (ii) JAVA?

[**4**] What is the value of the following OCAML program?

```
let yourMom = [[1;2]; [3;4;5;6;7], [8]]
 in map (foldr (fun (_,x) -> 1+x) 0)
        yourMom
```

[**5**] What problem does invoking the following C function lead to?

```
int* elts (int c, int n) {
  int a[n];
  int i;
  for (i = 0; i < n; i++) {
    a[i] = c*i;
  }
  return a;
}
```

Extra: : How can the problem be fixed?

Naming

[**1**] List all of the free variables of the following HOFL expression:

```
(abs (a)
   (a b (abs (b) (+ b c))))
```

[**2**] List *all* of the following languages that are block structured:

- Pascal
- C
- Java
- Scheme
- ML

[**3**]
The following Common Lisp program denotes the factorial function, but a SCHEME program written in the same way would not. What language property accounts for the difference in which the program is treated in the two languages?

```
(defun fact (fact)
  (if (= fact 0)
      1
      (* fact (fact (- fact 1)))))
```

[**4**] Give the value of the following expression in both lexically scoped and dynamically scoped versions of Scheme:

```
(let ((a 1)
      (b 2))
  (let ((f (let ((a 10))
             (lambda () (+ a b)))))
    (let ((b 20))
      (f))))
```

[**5**] Give the value of the following Scheme expression under all four parameter passing mechanisms: call-by-value, call-by-name, call-by-need, call-by-reference. Assume procedure arguments are evaluated in left-to-right order.

```
(let ((a 1))
  (let ((b a))
    (let ((c (begin (set! a (* a 2) a))))
      (begin (set! b 10)
             (+ a (+ c c))))))
```

Laziness

[1] Which one of the following does not belong:

- lazy data
- call-by-value
- memoization
- call-by-need.

[2] In his paper "Why Functional Programming Matters", John Hughes argues that laziness is important because it enhances something? What?

[3] Below are two definitions of an if0 construct: the first defined by desugaring, the second defined as a function:

(1) (if0 E_{num} E_{zero} $E_{nonzero}$)
desugars to (if (= E_{num} 0) E_{zero} $E_{nonzero}$)

```
(2) (define if0
      (lambda (Enum Ezero Enonzero)
        (if (= Enum 0) Ezero Enonzero)))
```

List *all* of the following parameter-passing mechanisms under which the two definitions are equivalent:

call-by-value call-by-name call-by-need

[4] What are the elements of the list returned by evaluating the following Haskell expression?

```
take 5 (scanl (+) 0 elts)
  where elts = 1 : (map (2 *) elts)
```

[5] What is the value of the following statically-scoped call-by-value Scheme expression? Assume left-to-right operand evaluation.

```
(let ((n 0))
  (let ((inc! (lambda (x)
                (begin (set! n (+ n x)) n))))
    (let ((inc1 (lambda () (inc! 1)))
          (inc2 (delay (inc! 2))))
      (+ (* (inc1) (force inc2))
         (* (inc1) (force inc2))))))
```

Extra: : What if the operand evaluation order is right-to-left?

Transformations

[1] What common program transformation have we studied that Alan Perlis once quipped could cause "cancer of the semi-colon"?

[2] What is the name of a transformation that can transform an ML function of type

```
int * char -> bool
```

to a function of type

```
int -> char -> bool
```
?

[3] Consider the following program transformation:

(+ E E) => (* 2 E)

For each of the following programming paradigms, indicate whether the above transformation is safe - that is, it preserves the meaning of the expression for all possible expressions E.

- purely functional
- imperative
- object-oriented

[4] Consider the following transformation in an imperative version of Scheme:

((lambda (x) 3) E) => 3

List all of the following parameter passing mechanisms for which the above transformation is safe - that is, it preserves the meaning of the expression for all possible expressions E.

- call-by-value
- call-by-name
- call-by-need
- call-by-reference

[5] In Scheme, the special form (or E_1 E_2) first evaluates E_1 to a value V_1. If V_1 is not false, it is returned without evaluating E_2. If V_1 is false, the value of E_2 is returned. Bud Lojack suggests the following desugaring rule for (or E_1 E_2):

(or E_1 E_2)
desugars to
(let ((x E_1)) (if x x E_2))

Unfortunately, this desugaring has a bug. Give a concrete expression in which Bud's desugaring fails to have the right meaning.

Imperative Programming

[**1**] List all of the following languages in which a variable is always bound to an implicit mutable cell.

- Scheme
- ML
- Java
- Haskell
- C

[**2**] What programming language property corresponds to the mathematical notion of "substituting equals for equals" (Functional languages have it; imperative languages don't.)

[**3**] What is the value of executing `f(5)`, where `f` is the following C function?

```
int f (int n) {
  int ans = 1;
  while (n > 0) {
    n = n - 1;
    ans = n * ans;
  }
  return ans;
}
```

[**4**] What is the value of executing `g(1,2)` in the context of the following C definitions?

```
void h (int x, int* y) {
  x = x + *y;
  *y = *y + x;
}

int g (int a, int b) {
  h(a, &b);
  return a * b;
}
```

[**5**] What is the value of the following Scheme program? Assume operands are evaluated from left to right. (Hint: draw environments!)

```
(let ((f (let ((a 0))
           (lambda ()
             (begin (set! a (+ a 1))
                    (let ((b 0))
                      (lambda ()
                         (begin (set! b (+ a b))
                                b))))))))
  (let ((p (f))
    (+ (p)
       (let ((q (f))
         (+ (q)
            (+ (p) (q))))))
```

Control

[**1**] Name the property that allows Scheme to perform iterations in constant space without explicit looping constructs.

[**2**] Which one of the following most closely models Pascal's `goto` construct?

- Scheme's `error` construct
- Scheme's `call-with-current-continuation` construct
- ML's `raise` construct
- Java's `try/catch` construct
- Java's `break` construct

[**3**] What is the value of the following expression in a version of Scheme supporting raise and handle?

```
(handle err (lambda (y) (+ y 200))
  (let ((f (lambda (x) (+ (raise err x) 1000))))
    (handle err (lambda (z) (+ z 50))
      (f 4)))
```

Extra: what if the handles are replaced by traps?

[**4**] Consider the following procedure in a version of Scheme supporting `label` and `jump`:

```
(define test
  (lambda (x)
    (+ 1 (label a
           (+ 20 (label b
                   (+ 300 (jump a
                            (label c
                              (if (> x 0)
                                  (+ 4000 (jump c x))
                                  (jump b x))))))))))
```

What is the value of the expression `(+ (test 0) (test 5))`? Assume operands are evaluated left-to-right.

[**5**] What is the value of the following expression in a version of Scheme supporting `label` and `jump`?

```
(let ((twice (lambda (f) (lambda (x) (f (f x))))))
  (let ((g (label a (lambda (z) (jump a z)))))
    (((g twice) 1+) 0)))
```

Potpourri

[1] Complete the following Guy Steele poem by filling in the *???*:

> A one slot cons is called a *???*
> A two-slot cons makes lists as well
> And I would bet a coin of bronze
> There isn't any three-slot cons.

[2] Who was the inventor of the lambda calculus, a formal system upon which functional programming is based?

[3] Is it possible to write an interpreter for an imperative language in a purely functional language?

[4] Fill in the *???* in the following Norman Adams quote: "Objects are a poor man's *???*".

[5] List five properties that values must have in order to be considered "first-class".