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 OCAML, JAVA, and SCHEME, 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?

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

[5] Answer both of the following: (1) what problem does invoking the following C function lead to and (2) how can the problem be fixed?

```c
def nums (int n) {
  int a[n];
  for (n = n-1; n >= 0; n--) {
    a[n] = n;
  }
  return a;
}
```
Naming

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

((fun (a)
    (a b (fun (b) (+ b c)))))

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

- PASCAL
- C
- JAVA
- OCAMLL
- SCHEME

[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 statically 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 HOILIC expression under all four parameter passing mechanisms: call-by-value, call-by-reference, call-by-name, and call-by-lazy. Assume operands are evaluated in left-to-right order. (bind a 1

  (bind b a
    (bind c (seq (<- a (* a 2)) a)
      (seq (<- 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 if\(^0\) construct: the first defined by desugaring, the second defined as a function:

\[
(1) \text{if} \ E \ \text{num} \ E_\text{zero} \\
\sim \text{if} \ (= \ E \ 	ext{num} \ 0) \ E_\text{zero} \ E \text{num}
\]

\[
(2) \text{def if} \ \text{num} \ \text{zero} \\
\quad \text{if} (= \ \text{num} \ 0) \ \text{zero} \ \text{num})
\]

For (1) HOFL and (2) HOILIC, list all of the following parameter-passing mechanisms under which the two definitions are equivalent:

- call-by-value
- call-by-name
- call-by-lazy

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

\[
\text{take 5 (scanl (+) 0 ns)} \\
\text{where ns = 1 : (map (2 +) ns)}
\]


\[
\text{(let ((n 0))} \\
\text{(let ((add! (lambda (x) } \\
\text{\quad (begin (set! n (+ n x)) n)))} \\
\text{(let ((add1 (lambda () (inc! 1)))} \\
\text{\quad (add2 (delay (inc! 2))))} \\
\text{\quad (+ (* (add1) (force add2))} \\
\text{\quad (* (add1) (force add2)))))})
\]

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] Consider the following program transformation:

\[(+ E E) \Rightarrow (* 2 E)\]

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

[3] Consider the following Hoilic transformation:

\[((\lambda (x) 3) E) \Rightarrow 3\]

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-reference
- call-by-name
- call-by-lazy

[4] 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 \(or\):

\[(or E_1 E_2) \rightsquigarrow (let ((x E_1)) (if x x E_2))\]

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

[5] Give a translation of the following Fofl program into Postfix. You may use \texttt{bget} in your translation.

\[
(fofl (a b) (f (sq a) (sq b)))
(def (sq x) (* x x))
(def (f x y) (/ (+ x y) (- x y))))
Imperative Programming

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

- SCHEME
- OCAML
- JAVA
- HASKELL
- C

[2] What programming language property corresponds to the mathematical notion of “substituting equals for equals” (Purely functional languages have it; imperative languages don’t.)

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

```c
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?

```c
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 program in statically-scoped call-by-value HOIIC? Assume operands are evaluated from left to right. (Hint: draw environments!)

```plaintext
(bind f (bind a 0
    (fun ()
        (seq (<- a (+ a 1))
            (bindpar ((b a) (c 0)))
                (fun ()
                    (seq (<- c (+ c b))
                        c)))))))
(bindseq ((p (f)) (q (f)))
    (list (p) (q) (p) (q)))
```
Extra: What if (+ c b) were changed to (+ c a)?
Edsgar Dijkstra considered this control construct harmful.

Which one of the following most closely resembles Pascal’s goto construct?

- SCHEME’s error
- SCHEME’s call-with-current-continuation
- OCAML’s raise
- JAVA’s break
- JAVA’s try/catch

What is the value of the following expression in a version of SCHEME supporting raise and handle?

\[
\begin{align*}
\text{(handle err (lambda (y) (+ y 200)))} \\
\text{(let ((f (lambda (x) (+ (raise err x) 1000))))}) \\
\text{(handle err (lambda (z) (+ z 50)))} \\
\text{(f 4))}
\end{align*}
\]

Extra: what if the handles are replaced by traps?

Consider the following procedure in a version of SCHEME supporting label and jump:

\[
\begin{align*}
\text{(define test} \\
\text{(lambda (x))} \\
\text{(+ 1 (label a} \\
\text{(+ 20 (label b} \\
\text{(+ 300 (jump a} \\
\text{(label c} \\
\text{(if (> x 0)
\text{(+ 4000 (jump c x))} \\
\text{(jump b x))))))))})
\end{align*}
\]

What is the value of the expression (+ (test 0) (test 5))? 

What is the value of the following expression in a version of SCHEME supporting label and jump?

\[
\begin{align*}
\text{(let ((twice (lambda (f) (lambda (x) (f (f x))))))} \\
\text{(inc (lambda (x) (+ x 1))))} \\
\text{(let ((g (label a (lambda (z) (jump a z))))}) \\
\text{(((g twice) inc) 0))}
\end{align*}
\]
Types

[1] Name two ”real-world” statically-typed language that do not require explicit types.

[2] What feature is lacking in Java’s type system that makes it impossible to write a general Scheme or ML style `map` function in Java?

[3] What is the name of a transformation that can transform an OCAML function of type

\[
\text{int} * \text{char} \rightarrow \text{bool}
\]

to a function of type

\[
\text{int} \rightarrow \text{char} \rightarrow \text{bool}
\]  

[4] Write a declaration of an OCAML function `f` that has the following type:

\[\text{('a -> 'b list) \rightarrow ('b -> 'c list) \rightarrow ('a -> 'c list)}\]

You may find it helpful to use the following list functions in your definition:

\[
\text{List.map: ('a -> 'b) -> ('a list) -> ('b list)}
\]
\[
\text{List.flatten ('a list list) -> ('a list)}
\]

[5] For each of the following OCAML function declarations, either write down the type that would be reconstructed for the function or indicate that no type can be reconstructed:

\[
\text{let test1 (x, f, g) = (x, f(x), g(x))}
\]
\[
\text{let test2 (x, f, g) = (x, f(x), g(f(x)))}
\]
\[
\text{let test3 (x, f, g) = (x, f(x), g(f(x)), f(g(x)))}
\]
\[
\text{let test4 (x, f, g) = (x, f(x), g(x, f(x)))}
\]
\[
\text{let test5 (x, f, g) = (x, f(x), g(f(x), f(g(x))))}
\]
\[
\text{let test6 (x, f, g) = (x, f(x), g(x, f(g(x))))}
\]
[1] Who was the inventor of the lambda calculus, a formal system upon which functional programming is based?

[2] 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.

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

[4] List five properties that values must have in order to be considered “first-class”.

[5] We saw how to automatically translate FOFL programs to POSTFIX programs. Answer both of the following:

1. Describe a simple approach for translating FOBS programs to POSTFIX.

2. What feature does postfix lack that makes it difficult to translate HOFL programs to POSTFIX?