CS251 Jeopardy: The Home Version
The game that turns CS251 into CS25\textit{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?

\begin{verbatim}
let yourMom = [[1;2]; [3;4;5;6;7], [8]]
in map (foldr (fun (_,x) -> 1+x) 0)
yourMom
\end{verbatim}

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

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

\textit{Extra:} How can the problem be fixed?
Naming

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

\[(\text{abs} (a)
\quad (a \ b \ (\text{abs} \ (b) \ (+ \ b \ c))))\]

[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?

\[
(\text{defun} \ \text{fact} \ (\text{fact})
\quad (\text{if} \ (= \ \text{fact} \ 0)
\quad \ 1
\quad \ (* \ \text{fact} \ (\text{fact} \ (- \ \text{fact} \ 1))))\)

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

\[
(\text{let} \ ((a \ 1)
\quad (b \ 2))
\quad (\text{let} \ ((f \ (\text{let} \ ((a \ 10))
\quad \ (\text{lambda} \ () \ (+ \ a \ b)))))
\quad (\text{let} \ ((b \ 20))
\quad \ (f)))
\]


\[
(\text{let} \ ((a \ 1))
\quad (\text{let} \ ((b \ a))
\quad \ (\text{let} \ ((c \ (\text{begin} \ (\text{set!} \ a \ (* \ a \ 2) \ a)))
\quad \ (\text{begin} \ (\text{set!} \ b \ 10)
\quad \ (+ \ 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:

\[
\begin{align*}
&\text{(1) } \text{if0 } E_{\text{num}} E_{\text{zero}} E_{\text{nonzero}} \\
&\text{desugars to } \text{if } (= E_{\text{num}} 0) E_{\text{zero}} E_{\text{nonzero}}
\end{align*}
\]

\[
\begin{align*}
&(2) \text{define if0} \\
&(\text{lambda} \ (\text{Enum} \ E_{\text{zero}} \ E_{\text{nonzero}}) \\
&(\text{if } (= \text{Enum} 0) \ E_{\text{zero}} \ E_{\text{nonzero}}))
\end{align*}
\]

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?

\[
\text{take 5 (scanl (+) 0 elts)}
\]

where \( \text{elts} = 1 : \text{map} \ (2 \ *) \ \text{elts} \)

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

\[
\begin{align*}
&(\text{let } ((n 0)) \\
&(\text{let } ((\text{inc!} \ (\text{lambda} \ (x) \\
&(\text{begin} \ (\text{set!} \ n \ (+ \ n \ x)) \ n)))) \\
&(\text{let } ((\text{inc1} \ (\text{lambda} \ () \ (\text{inc!} \ 1)))) \\
&(\text{inc2} \ (\text{delay} \ (\text{inc!} \ 2)))) \\
&(\text{+} \ (\text{*} \ (\text{inc1}) \ (\text{force} \ \text{inc2})) \\
&(\text{*} \ (\text{inc1}) \ (\text{force} \ \text{inc2}))))
\end{align*}
\]

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
\[
\text{int} \times \text{char} \to \text{bool}
\]
to a function of type
\[
\text{int} \to \text{char} \to \text{bool}
\]?

[3] 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

[4] Consider the following transformation in an imperative version of Scheme:
\[
((\text{lambda} (x) 3) E) \Rightarrow 3
\]
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-name
- call-by-need
- call-by-reference

[5] In Scheme, the special form $(\text{or} \ E_1 \ E_2)$ 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 $(\text{or} \ E_1 \ E_2)$:
\[
(\text{or} \ E_1 \ E_2) \quad \text{desugars to} \\
(\text{let} \ ((x \ E_1)) \ (\text{if} \ x \ x \ E_2))
\]
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?

```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 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 (lambda 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”.