Quiz 2 (2013/09/25)

Put your name and student ID clearly on the quiz answer sheet and on a sheet of your own paper. Write "Q1", "Q2", "Q3", "Q4", "Q5", and "Q6" on your paper. For each question, write your answer on *both* sheets in the appropriate place. Hand in the quiz answer sheet *only*.

In every case, select the single *best* answer.

You're done with any question you answer correctly on the quiz.

By Tue 8PM, submit corrections. Include your name, student ID, the quiz number, and your collaboration statement. (**Even if you got everything right, please do submit for our records**. Just indicate that you got everything correct.) For each question you got incorrect, write the question number and then explain why the correct answer is correct and why the answer you chose is incorrect.

Running Example for Questions 1–3

Consider the following small interpreter from class (with space included for you to build on it). We would like to add an absolute value operator to the language.

```
(define-type ArithC
  ;; Here is the new variant to handle absolute values.
  [absC (body : ArithC)]
  [numC (n : number)]
  [plusC (l : ArithC) (r : ArithC)]
  [multC (l : ArithC) (r : ArithC)])

(define (interp [ast : ArithC]) : number
  (type-case ArithC ast
    [absC (body) ...]  ;; not yet complete
    [numC (n) n]
    [plusC (l r) (+ (interp l) (interp r))]
    [multC (l r) (* (interp l) (interp r))]))
```

Question 1: Changing an Interpreter

Which of these best completes the `absC` case of the interpreter?

1. `(local [(define b (numC-n body))] (if (< b 0) (* -1 b) b)`
2. `(local [(define v (interp body))] (if (< v 0) (* -1 v) v)`
3. `(if (< body 0) (* -1 body) body)`
4. `(if (< body 0) (multC (numC -1) body) body)`

-

Question 2: Interpreting and Parsing

We've decided that the concrete syntax should look like absolute value bars; so, for example, `|3|` evaluates to `3`, as does `|-3|`.

The built-in `read` function will not handle absolute value bars correctly. We will need a much more complex parser. How must the **interpreter** change in response to the need for more complex **parsing**?

1. Without the parser implementation, there is not enough information to tell what must change.
2. No part of the interpreter needs to change at all.
3. The current `ArithC` definition must change because the concrete syntax is no longer s-expressions.
4. `interp`'s `type-case` needs a case with a template like `| ... (interp body) ... |`

-

Question 3: Desugaring

Imagine we also had the following abstract syntax, which evaluates the `cond-exp` and then: if it's negative evaluates to the value of the `then-exp` and otherwise evaluates to the value of the `else-exp`.

```
[ifNegC (cond-exp : ArithC) (then-exp : ArithC) (else-exp : ArithC)]
```

Assuming we build the other parts of the desugarer (e.g., defining the full `ArithS` surface abstract syntax), which of these best describes how desugaring away absolute values could work?

1. We could desugar `(absS <body>)` to:

   ```
   (local [(define b (numC (interp (desugar <body>))))]
     (ifNegC b (multC (numC -1) b) b))
   ```

2. We could desugar `(absS <body>)` to:

   ```
   (local [(define b (desugar <body>))]
     (ifNegC b (multC (numC -1) b) b))
   ```

3. It could not work because we cannot tell until runtime (*dynamically*) whether the `cond-exp`'s value is negative.
4. It could not work because we cannot desugar to a conditional.

-

Question 4: How Interpreters Work

(Starting with this question, we no longer reference the running example.)

Which of these best describes the job of an interpreter?

1. To evaluate an expression to its value
2. To simplify the job of a desugarer
3. To run a program and display its result
4. To transform a program from concrete syntax into abstract syntax

-

Question 5: Key Features of Functions

Imagine these two snippets of code are part of a much larger `plai-typed` program: `(+ 1 2)` and `(define (foo x) (+ 1 2))`. Which of the following best describes the difference between the two expressions?

1. Only the first snippet evaluates to a value of type number.
2. Only the second snippet behaves differently depending on its argument.
3. Only the first snippet will run no matter what else happens in the program.
4. Only the second snippet can appear in more than one place in the program.

-

Question 6: Formal and Actual Parameters

Which of these best describes what a function **definition** can and cannot include, with respect to formal and actual parameters.

(Remember **all** the parts of a function definition!)

1. Cannot include appearances of either formal or actual parameters.
2. Can include appearances of formal but not actual parameters.
3. Can include appearances of actual but not formal parameters.
4. Can include appearances of both formal and actual parameters.

•