CSE 341 Midterm — May 5, 2000 — Answer Key

1. (6 points) Write a Scheme expression to pick out the atom `squid` in the following lists (which will each be bound to `x`). You can use an expression you want, as long as there are no side effects or definitions of new global variables.

   (a) `(define x '(octopus shrimp squid clam))`
   Answer: `(caddr x)`

   (b) `(define x (cons shrimp (cons squid (cons smelt nil))))`
   Note: there were missing quotes on the original version of the test – but nobody had trouble because of this. The corrected question is:
   `(define x (cons 'shrimp (cons 'squid (cons 'smelt ()))))`
   Answer: `(cadr x)`

   (c) `(define x '( ((octopus)) ((shrimp)) ((squid)) ((clam)) ))`
   Answer:

   ```
   (let ((third (caddr x)))
      (caar third))
   ```

2. (6 points) Zero or more of the following Perl expressions are true. Circle all true expressions.

   (a) `"spring"`
   True All strings are true except "" or "0".

   (b) `"spring" =~ /^[aeiou]*.*/`
   True `^[aieou]*` matches zero or more vowels at the start of the string, which means it matches nothing. The given regular expression actually matches any string, because it matches zero or more vowels followed by zero or more of any character.

   (c) `"spring" =~ /^[aeiou].*/`
   False `^[aeiou]` matches exactly one vowel at the beginning of the string. "spring" does not begin with a vowel.

3. (6 points) What does the following Perl code fragment print?

   ```
   $x = "the wonderful thing about Tiggers is Tiggers are wonderful things";
   $x =~ s/on(.*)ge//;
   print "$1\n$x";
   ```

   Output:

   ```
   derful thing about Tiggers is Tig
   the wrs are wonderful things
   ```

   Note that the `*` operator in Perl matches greedily, so it does not stop at the first "ge". Also note that, although we match on "onderful thing about Tiggers is Tigge", the "on" and "ge" are not saved by the parens, and so not printed.

4. (12 points) Consider the following Java class definitions. (These compile correctly.)

```
class Appliance {
  private static int applianceTag = 0;

  public int tag() {
    return applianceTag;
  }

  public void setTag(int t) {
    applianceTag = t;
  }

 public void printDescription() {
    System.out.println("a generic appliance");
  }
}


class Stove extends Appliance {

 public void printDescription() {
    System.out.println("a stove");
    super.printDescription();
  }
}


class SelfCleaningStove extends Stove {

  private int myTag = 0;

  public int tag() {
    return myTag;
  }

  public void setTag(int t) {
    myTag = t;
  }

 public void printDescription() {
    System.out.println("a self cleaning stove");
    super.printDescription();
  }
}
```

Now suppose we also have a class `Kitchen` with a `main` method. What is the result of compiling and executing the Java program for each of the following versions of `Kitchen` and `main`? The result might be that the program runs correctly, or it might have a compile time error, or a runtime error. If the program compiles correctly and can be run, give the output. Otherwise explain what the error is.

(a)
```
public class Kitchen {
    public static void main (String [ ] args) {
      Appliance a = new Appliance();
      Appliance b = new Stove();
    Appliance c = new SelfCleaningStove();
    a.printDescription();
    b.printDescription();
    c.printDescription();
  }
}
```

Answer:

```
a generic appliance

a stove
a generic appliance

a self cleaning stove
a stove
a generic appliance
```

(b)

```
public class Kitchen {
  public static void main (String [ ] args) {
    SelfCleaningStove c = new SelfCleaningStove();
    Stove b = c;
    b.printDescription();
    c.printDescription();
  }
}
```

Answer:

```
a self cleaning stove
a stove
a generic appliance

a self cleaning stove
a stove
a generic appliance
```

(c)

```
public class Kitchen {
  public static void main (String [ ] args) {
    Appliance a = new Appliance();
    Appliance b = new Stove();
    Appliance c = new SelfCleaningStove();
    a.setTag(100);
    b.setTag(200);
    c.setTag(300);
    System.out.println(a.tag());
    System.out.println(b.tag());
    System.out.println(c.tag());
  }
}
```

Answer:

```
200
200
300
```

5. (10 points) Tacky but easy-to-grade true/false questions!

   (a) Java bytecodes are machine instructions for a virtual register-based machine that includes 32 general-purpose and 16 floating point registers.
   False

   (b) As soon as an object in Java is no longer accessible from any live variable, its "finalize" method will be invoked and it will be garbage collected.
   False. (Java *may* garbage collect it as soon as it is no longer accessible, but there is no requirement that it be garbage collected at any particular time.)

   (c) Scheme is type safe, but not statically type checked.
   True.

   (d) Java is type safe, but not statically type checked.
   False.

   (e) Perl's type declarations, along with its static type checking, mean that type errors in a Perl program are discovered at compile time.
   False. (Perl is dynamically typed.)

6. (8 points) Draw a box-and-arrow diagram that shows the list structure and bindings of variables after the following Scheme expressions have been evaluated:

```
(define x '(a b c))
(define y (append (cons 1 nil) x))
(define z (cdr y))
```

```
        -------------      ------------      -----------
        |      |     |     |     |     |     |    |   / |
x ----> |  a   |   --|---->|  b  |  ---|---->| c  |  /  |
z ----> |______|_____|     |_____|_____|     |____|_/___|
            ^
            |
            |___________
                        |
        -------------   |
        |      |     |  |
y ----> |  1   |   --|---
        |______|_____|
```

7. (12 points) What is the result of evaluting the following Scheme expressions?

   (a) `(* (+ 3 4) (- 6 2))  ; an easy one to start off ...`
   Answer: `28`

   (b) `(cddr '(a (b)))`
   Answer: `nil`

   (c)
```
(let ((x '(a b))
      (y '(1 2))
   (let ((x '(c d))
         (y '(3 4 5))
         (f (lambda (z) (append x z))))
       (f x))))
```
   Question thrown out! (mismatched parenthesis)

(d)
```
(let ((a cons)
      (b 3)
      (c '(+ 1 2)))
  (eval (list a b c) user-initial-environment))
```
Question thrown out! (The answer is `(3 . 3)` however.)

8. (20 points) Java includes 8 primitive types (which aren't objects). What are the advantages and disadvantages of having these 8 primitive types separate from ordinary objects, as opposed to making everything be an object? Consider the regularity and usability of the language, efficiency of runtime storage management, potential for efficiency of code generated by a JIT, and any other issues you think are relevant.

   Answer:

   The advantages are almost entirely related to efficiency. The primitive types take up less space in memory. They can also be stored on the stack rather than in the heap, thus avoiding the use of the garbage collector. Java can compile more efficient and specialized byte codes for them, and JITs can then compile more efficient native code from that.

   The disadvantages are that the language is less regular. For methods that expect objects as arguments, or variables that hold objects, one can't use a primitive type but must instead wrap it. This is a common problem when using Java's collection library – a HashSet, for example, can't contain ints. The primitive types cannot be subclassed or extended, and one can't add new primitives.