CSE 413 Spring 2000 Midterm Exam

Sample Solution

Name _____ ID # _____ Score _____

1	2	3	4	5	6	7	8	9	10

There are 10 questions worth a total of 42 points. Please budget your time so you get to all of the questions. ***Keep your answers brief and to the point.***

You may refer to the textbook (Budd's *Understanding Object-Oriented Programming using Java*) and the Scheme report *only*. No other books or notes are allowed.

Question 1. (5 points) Write a recursive Scheme function `powers-of-two`. Given a list of integers as input, `powers-of-two` should return a list whose elements have the value 2^n for each corresponding **positive number** *n* in the original list. Any elements of the original list that are not positive or are not integers should be ignored. Examples:

```
(powers-of-two '(1 2 3 a 0 10))     => (2 4 8 1024)
(powers-of-two '(-2))               => ()
(powers-of-two '(-3 5 oops 6 2.4)) => (32 64)
```

Feel free to use appropriate Scheme library functions in your solution.

```
(define (powers-of-two alist)
    (cond
        ((null? alist) ())
        ((and (integer? (car alist)) (positive? (car alist)))
            (cons (expt 2 (car alist)) (powers-of-two (cdr alist))))
        (else (powers-of-two (cdr alist)))))
```

CSE 413 Spring 2000 Midterm Exam

Question 2. (8 points) Suppose we enter the following top-level definitions into a Scheme interpreter.

```
(define a 100)
(define b '(200 49))
(define c 17)
(define (pizza a) (odd? a))
(define (pie a) (pizza (- 3 a)))
(define apple (lambda (a) (> a c)))
```

What is the value of each of the following expressions, given that the above definitions are in effect? If evaluating the expression produces an error, explain what is wrong.

a) `(cons a b)` => `(100 200 49)`

b) `(cons b b)` => `((200 49) 200 49)`

c) `(append a b)` error. Both arguments to append must be lists; a is bound to the integer 17.

d) `(apple 13)` => `#f`

e) `(pizza 12)` => `#f`

f) `(map pizza b)` => `(#f #t)`

g) `(map pie '(7 8 9))` => `(#f #t #f)`

h)
```
(let ((c 12)
      (b (+ c a)))
  (+ b c))
```
=> 129

CSE 413 Spring 2000 Midterm Exam

Question 3. (4 points) Suppose we have defined the following Scheme functions.

```
;; = "is num a prime number?"
(define (is-prime? num)
   (prime-helper num (- num 1)))

(define (prime-helper num divisor)
   (cond ((< divisor 1) #f)
         ((= 1 divisor) #t)
         ((= 0 (remainder num divisor)) #f)
         (else (prime-helper num (- divisor 1)))))
```

The following questions refer to the auxiliary function `prime-helper` ***only.***

.

a) Is `prime-helper` tail recursive or NOT tail recursive? In 1 or 2 sentences explain why or why not.

Yes. The only recursive call is in the else clause, and the result of that recursive call is the direct result of prime-helper; no further processing needs to be done.

b) What is the time complexity of `prime-helper`? ($O(n \log n)$, $O(1)$, $O(n^2)$, etc.).

Given a divisor d, `prime-helper` executes $O(d)$ steps.

c) What is the space complexity of `prime-helper`?

O(1). (Scheme requires that tail recursion be implemented without requiring a separate stack frame for each tail-recursive call .)

Question 4. (3 points) Write a function `(are-primes? alist)` that returns a list indicating which of the corresponding numbers in the original list are prime. Your answer should use `is-prime?` from the previous question as needed, and ***may not contain any loops or recursive function calls.*** (Hint: higher-order functions) You may assume that all elements in the original list are positive integers. Example:

```
(are-primes? '(3 42 17 12)) => (#t #f #t #f)
```

```
(define (are-primes? lst)
  (map is-prime? lst))
```

CSE 413 Spring 2000 Midterm Exam

Question 5. (4 points) Generational garbage collectors classify objects as "new", "not so new", and "really old".

a) How is this information used by the garbage collector? (i.e., what is the strategy for collecting different kinds of objects, how does the collector decide which objects are new/not so new/old, etc.) Describe briefly.

Objects are initially allocated in new space. If they survive some number of garbage collections, they are promoted to "not so new", and if they survive additional collections while there, they are eventually promoted to old space.

The GC collects the new space frequently, "not so new" less frequently, and old space rarely (exactly how often depends on the particular algorithms used).

b) What is the rationale for doing this? Why is this a useful/good idea?

The key observation is that most dynamically allocated objects live for a short time and become garbage quickly. Objects that do not become garbage quickly tend to have long lifetimes. That means that collections of new space are likely to reclaim a high percentage of that pool of storage, while collections of older spaces have much lower yields for the amount of work done. So frequent collections of new space are more cost effective (yield the most free storage for the effort involved).

CSE 413 Spring 2000 Midterm Exam

Question 6. (6 points) This question concerns static (lexical) vs dynamic scope rules.

Suppose we have the following definitions:

```
(define a 100)

(define b 17)

(define (squid)
    (let ((b 5))
      (clam b)))

(define (clam a)
    (+ a b))
```

a) (2 points)What is the result of evaluating `(squid)` using the normal lexical scoping rules of Scheme?

22

b) b. (2 points) What is the result of evaluating `(squid)` using dynamic scoping?

10

c) (2 points) Modern programming languages generally use lexical scoping instead of dynamic scoping. Why? (Give a software engineering or other technical reason.)

If a language provides static scoping, it is possible to analyze functions and understand them without having to take into account how they are used. With dynamic scope, what happens when a function is executed depends on who calls it, and what local names the caller(s) define. Besides being harder to analyze, it also makes the code more fragile – changes in the calling program (renaming of local variables) can change the behavior of functions that it calls.

Question 7. (5 points) Circle true or false.

In the Java programming language…

a) True | False Private fields and methods declared in a class can be accessed by member functions of that class and from member functions of classes that extend the original class.

FALSE. Private fields and methods can only be accessed directly inside the class, not in other classes (even those that extend the original class).

b) True | False Methods and data can both be declared final.

TRUE

c) True | False The number of bits used to store a variable of type `int` depends on the kind of machine or particular Java implementation that you're using.

FALSE

d) True | False The default assignment operator(=) does a shallow copy.

TRUE

e) True | False A class that implements an interface must either implement all methods in the interface, or inherit those that it does not implement from a parent class or interface.

FALSE. Code cannot be inherited from interfaces; they are purely specifications with no implementations.

Question 8. (2 points) What's wrong with this Java class definition, if anything? (If something is wrong, it will be more significant than a missing semicolon or other trivial punctuation error.)

```
class Whazzup {
    private int v;
    public static int getv() {return v;}
    public static void setv(int val) { v = val; }
}
```

Static functions cannot reference non-static instance variables because they are not associated with an instance (they are associated with the class). So the references to variable `v` in the static functions are illegal.

Question 9. (3 points)

a) If a Java method is declared "synchronized", what does that mean?

A synchronized method obtains a lock on the associated object while it executes, and any other thread that calls a synchronized method for the same object will block until the first one releases the lock.

b) Why would anyone ever do this?

The primary use is to keep two threads from interfering with each other, by keeping a second thread from accessing data that has been partially updated by the first one.

Question 10. (2 points) Java. A window in Java can be defined like this.

```
class NewFrame extends Frame {
        ...
        public void paint(Graphics g) {
           g.drawline(100,100,200,200);
        }
        ...
  }
```

When is `paint()` called to draw the window and who calls it?

`paint()` (or `update()`, which calls `paint()`) is called by the underlying window manger whenever some or all of the window needs to be redrawn.