CSE 413 Winter 2002 Midterm Exam Sample Solution

Question 1. (16 points, 2 points each part) Suppose we enter the following top-level definitions into a Scheme interpreter.

```
(define one '(1 2 3))
(define taste '(sweet sour salt bitter))
(define nums  (list (+ 3 4) (- 6 4)))
(define vals '(list (+ 3 4) (- 6 4)))
```

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) (cddr taste) **(salt bitter)**

b) (cadr nums) **2**

c) (cadr vals) **(+ 3 4)**

d) (car (cdr (reverse taste))) **salt**

e) (list one one) **((1 2 3) (1 2 3))**

f) (cons one one) **((1 2 3) 1 2 3)**

g) (append one one) **(1 2 3 1 2 3)**

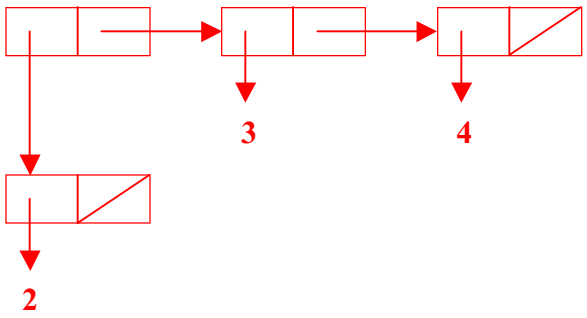
h) (or (< (cadr one) 10) (> (caar one 17))) **#t**

CSE 413 Winter 2002 Midterm Exam Sample Solution

Question 2. (10 points) Suppose we evaluate the following expression at the top level of a scheme interpreter

```
(define x (cons '(2) (list 3 4)))
```

(a) (3 points) Draw a boxes-and-arrows diagram of this data structure.



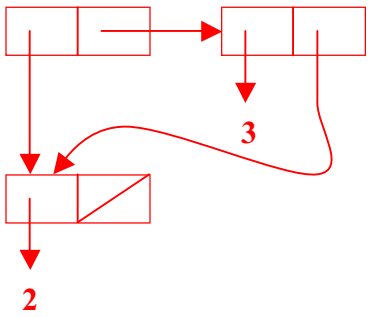
(b) (2 points) How would a Scheme interpreter print (display) the value of x?

```
((2) 3 4)
```

(c) (3 points) Now, suppose we evaluated the following:

```
(set-cdr! (cdr x) (car x))
```

Draw a boxes-and-arrows diagram showing this modified data structure (i.e., draw the modified data structure that x now refers to).



(d) (2 points) How would a Scheme interpreter print (display) this new value of x?

```
((2) 3 2)
```

Question 3. (9 points, 3 each)

(a) Some Scheme functions like `if` and `define` are referred to as special forms. What is the key difference between these special forms and ordinary functions like `cons`, `car`, and user-defined functions?

When an ordinary function is applied, all of its arguments are evaluated. Special forms do not necessarily evaluate all of their arguments.

(b) Scheme provides several different ways to bind values to names; two of them are `define` and `set!`. What is the key difference between these two?

define creates or changes a top-level binding; set! changes an existing binding but cannot bind a new name.

(c) What is the value of the following Scheme expression?

```
(map (lambda (f) (cons (f 3 2) '(4 5))) (list + * -))
```

```
((5 4 5) (6 4 5) (1 4 5))
```

CSE 413 Winter 2002 Midterm Exam Sample Solution

Question 4. (12 points) A car approaching an intersection can go one of three ways: left, straight, or right. For this problem, write a recursive scheme function `follow` that has two arguments: a description of an intersection (possibly nested), and a set of directions. The idea is that `(follow intersection directions)` should evaluate to the destination reached if the directions are followed starting at the given intersection. If the list of directions is too long and continues past the available intersections, function `follow` should return `#f`.

Directions are given by a list consisting of the symbols `l`, `s`, and `r`, meaning turn left, go straight, or turn right. The list `(r)` means turn right; the list `(s r l)` means go straight at the current intersection, turn right at the next one, and turn left at the last one.

An intersection is represented by a list (left straight right). Each choice is either a destination if it is a simple symbol, or the description of the next intersection in the given direction. Examples:

`(follow '(a b c) '(r)) => c` (i.e., turning right reaches `c`)

`(follow '(a (b c (d e f)) g) '(s r l)) => d`
(i.e., straight to `(b c (d e f))`, right to `(d e f)`, left to `d`)

`(follow '(a (b c (d e f)) g) '(s r)) => (d e f)`
(i.e., straight to `(b c (d e f))`, right to `(d e f))`

`(follow '(a b (x y z)) '(r l r l)) => #f`
(too many directions: right to `(x y z)`, left to `x`, but can't go further)

```
(define (follow intersection directions)
  (if (null? directions)
      intersection
      (if (pair? intersection)
          (cond ((eq? (car directions) 'l)
                 (follow (car intersection)
                         (cdr directions)))
                (cond ((eq? (car directions) 's)
                 (follow (cadr intersection)
                         (cdr directions)))
                (cond ((eq? (car directions) 'r)
                 (follow (caddr intersection)
                         (cdr directions))))
      #f)))
```

Correction

CSE 413 Winter 2002 Midterm Exam Sample Solution

Question 5. (12 points) The standard Scheme map function produces its result by applying a given function to each element of a list

```
(map (lambda (n) (+ n n)) '(2 -6 4 0 9)) => (4 -12 8 0 18)
```

Function map cannot, however, be used to apply a function to values in a nested list.

For this problem, write a function `deep-map` that applies a function to each element of a possibly nested list data structure, and returns a result that has the same shape as the original list argument.

```
(deep-map (lambda (n) (+ n n)) '((2 -6) 4 (((0)) 9)) ) =>
                                        ((4 -12) 8 (((0)) 18))
```

```
(define (deep-map f arg)
  (if (pair? arg)
      (cons (deep-map f (car arg))
            (deep-map f (cdr arg)))
      (if (null? arg)
          ()
          (f arg))))
```

Question 6. (12 points) If you recall from lecture, the Fibonocci numbers are defined using as the sequence

```
fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2)
```

In other words, each number in the sequence is the sum of the previous two: 0, 1, 1, 2, 3, 5, 8, 13, … . When we coded this up directly as a Scheme function, we discovered it ran very slowly for even moderate values of n, because its execution time was an exponential function of n.

For this problem, write a properly *tail-recursive* function `(fibt n)` that computes the n^{th} Fibonocci number. For full credit, your function should run in linear time as a function of `n` (i.e., the number of recursive calls should be proportional to `n`.) You will probably need an auxiliary (helper) function in addition to function `fibt`.

```
(define (fibt n)
  (if (< n 2)
      n
      (fibt-aux (- n 2) 1 0)))

(define (fibt-aux nremaining prev 2ndprev)
  (if (= nremaining 0)
      (+ prev 2ndprev)
      (fibt-aux (- nremaining 1) (+ prev 2ndprev) prev)))
```

CSE 413 Winter 2002 Midterm Exam Sample Solution

Question 7. **(12 points, 3 each) A few short-answer questions about Java.**

a) Can a class be both `final` and `abstract`? If so, give an example of where this would be useful. If not, explain why not.

No. A final class may not be extended, while an abstract class cannot be used to create objects unless it is extended and concrete instances of any abstract methods are supplied. It doesn't make sense to label a single class as both final and abstract.

b) Java references and C/C++ pointers are similar in that they can be used to link objects together to form data structures like linked lists and trees, and they can be used as function (method) parameters to pass an object to a function (method). But compared to C/C++ pointers, Java references are more limited: you can't use & to create a reference to an arbitrary variable, you can't do pointer arithmetic, etc. Describe an advantage of restricting references in Java this way compared to C/C++ pointers? Be specific but brief (i.e., saying "it makes software better" is not specific enough since it doesn't give a technical reason).

There are a couple of key advantages. One is security and reliability since a reference can only refer to an object in the heap or null; it can't refer to an arbitrary address in memory. A second advantage is that the garbage collector can make correct assumptions about which parts of objects refer to other objects.

Question 7 (cont).

c) In Java, a new type can be defined in two ways: by defining a class or by defining an interface. What are two fundamental differences between classes and interfaces?

A class can provide a partial or complete implementation of the methods belonging to the new type; an interface is pure specification and cannot provide implementations of any methods.

A single class may only extend one class (inheritance) but may implement many interfaces.

d) One of your colleagues, A. Hacker, is working on the bank's automatic teller machine software. Individual bank accounts are represented by instances of a class `BankAccount`. Hacker is an old-time Fortran programmer who is new to Java and isn't comfortable yet with containers like `ArrayList`. So his/her code uses arrays instead. The following code is supposed to create an array of `BankAccounts` and set their balances to 0.

```
BankAccount[ ] accounts = new BankAccount[10];
for (int k = 0; k < accounts.length; k++) {
     accounts[k].setBalance(0.0);
}
```

There is a `setBalance` method in class `BankAccount`, which can be used to change the account balance. But when this code is executed, a `nullPointerException` is thrown. What went wrong?

The references in the new array are initialized to null and do not refer to any BankAccount objects. For the code to work, the array elements would have to refer to existing or newly created BankAccount objects.

Question 8. (12 points, 2 each) Every Java application begins execution in a main method that has the following signature (i.e., function heading)

```
public static void main (String[ ] args) { … }
```

The question is, what does all that gibberish mean anyway? Below, give a brief definition or explanation of the meaning of each keyword or other item in the heading.

`public`

This method can be accessed by any client code that has permission to access the class.

`static`

This is a class method, i.e., it is not associated with instances of the class (objects). Instead there is a single copy of the method associated with the class itself.

`void`

This method does not return a value.

`main`

Function name. The name main signifies a function that can be used as the starting point when executing a Java class as an application program.

`String[]`

Type of the parameter: an array of strings.

`args`

Parameter name. The actual name has no significance – it could be anything, but it must be the name of an array of strings.