Computer Science 320 (Spring Term, 2006)

Concepts of Programming Languages

Problem Set 1: Programming With Lists

Out: Friday, January 20

Due: ***5:00 PM*** on Friday, January 27, 200

There are 6 problems in this set. The harder problems are marked with a single * (average difficulty) or two ** (higher-than-average difficulty).

CS320 is sponsoring a 100 meter race, the *Scheme cup*, and all CS students are very excited to compete. You're asked to write a few procedures to help us analyze the outcome of the race. All problems below have to do with a list of racing times. The list is a list of pairs, each pair being of the form `(`⟨*name*⟩ `.` ⟨*seconds*⟩`)` where ⟨*name*⟩ is a character string (i.e., a Scheme *symbol*) and ⟨*seconds*⟩ is a non-negative real number. Such a pair is obtained by evaluating the expression `(cons` ⟨*name*⟩ ⟨*seconds*⟩`)`. For example, the pair `(alice . 15.7)` is obtained by evaluating the expression `(cons 'alice 15.7)`. A typical racing times look like this:

```
((don . 12.7) (bob . 13.5) (dan . 16.6) (ed . 9.8) (cal . 9.7)
 (jack . 10.1) (ben . 11.3))
```

For later reference, let "`scheme-cup`" be the list of racing times in the preceding line.

Problem 1 (10 points) Define a procedure `average-time` which consumes a list of racing times and returns the average time of all participants (i.e., the sum of the times divided by the number of racers).

Problem 2 (10 points) In two parts:

1. Define a procedure `all-equal?` that determines if all racing times are the same. For example, `(all-equal? '((bill . 12.7) (joe . 11.8) (brad . 12.2)))` evaluates to `#f` and `(all-equal? '((bill . 12.7) (joe . 12.7) (brad . 12.7)))` evaluates to `#t`.

2. Define a procedure `all-different?` that determines if all racing times are different. *Hint*: This is not simply the negation of `all-equal?`. Determine if every racing time in the list occurs only once.

* **Problem 3** (20 points) In two parts:

1. Define a procedure `mergesort-by-time` which consumes a list of racing times and returns another list of the same pairs, in order of numerically increasing time. For example, the evaluation of the expression `(mergesort-by-time scheme-cup)` returns:

   ```
   ((cal . 9.7) (ed . 9.8) (jack . 10.1) (ben . 11.3) (don . 12.7)
    (bob . 13.5) (dan . 16.6))
   ```

2. Define a procedure `mergesort-by-name` which consumes a list of racing times and returns another list of the same pairs, in order of lexicographically increasing names. For example, the evaluation of the expression `(mergesort-by-name scheme-cup)` returns:

```
((ben . 11.3) (bob . 13.5) (cal . 9.7) (dan . 16.6) (don . 12.7)
 (ed . 9.8) (jack . 10.1))
```

To get credit for Problem 3, you have to use "mergesort", not any other sorting method.

** **Problem 4** (20 points) Define a procedure `median-time` which consumes a list of racing times and returns the median of the times. *Reminder*: Given a list of $n \geqslant 1$ numbers in increasing order, their *median* is the number in position $\lceil n/2 \rceil$ counting from 1. For example, the median of the sorted list of numbers `(30 35 56 57 64 73 88)` is `57`.

You get 15 points for Problem 4 if your procedure `median-time` is correct. You get 20 points if your procedure `median-time` is correct *and* only uses list-processing functions, i.e., it does not use any arithmetical operations such as "`+`", "`*`", "`/`", etc., nor the function `length` which returns the number of items in a list. *Hint*: Consider sorting the "double" list formed by `(append xs xs)`.

* **Problem 5** (20 points) We're now to determine how to give out medals to participants. Rather than giving only three medals to the top three racers, we're giving everybody a medal according to the time taken to finish the 100 meter race.

Define a procedure `assign-medals` which consumes a list of racing times and returns another list where the second entry of each pair is the medal of the corresponding time in the input list, according to the following table:

Medal Type	Condition
`'gold`	Time less than 10 seconds
`'silver`	Time greater than or equal to 10 seconds but less than 11 seconds.
`'bronze`	Time greater than or equal to 11 seconds but less than 12 seconds.
`'athletic`	Time greater than or equal to 12 seconds but less than 14 seconds.
`'good-effort`	Time greater than or equal to 14 seconds but less than 18 seconds.
`'best-courage`	Time greater than or equal to 18 seconds.

For example, the evaluation of the expression `(assign-medals scheme-cup)` returns:

```
((don . athletic) (bob . athletic) (dan . good-effort) (ed . gold)
 (cal . gold) (jack . silver) (ben . bronze))
```

(Most implementations of Scheme, including the one on csa, are not case-sensitive.)

* **Problem 6** (20 points) Define a procedure `select-by-medal` which consumes a list of racing times `xs` together with a medal `m` to be selected from the set {`'gold`,`'silver`,`'bronze`,`'athletic`, `'good-effort`,`'best-courage`} and returns a list of all names in `xs` who received a medal `m`. For example, the evaluation of the expression `(select-by-medal scheme-cup 'gold)` returns the list `(ed cal)`.

What You Have To Turn In

All your solutions should be entered into a single Scheme file called `PSet01.scm` (or `PSet01.ss`), which we will load into the Scheme interpreter and run to test your answers. Make sure you comment out all lines that cannot be interpreted, by preceding them with a semi-colon ";". Your file should start with the following 4 lines:

```
; NAME: John Smith
; ID: U12345678
; CLASS: CS320
; PROBLEM SET 01
```

where you replace "`John Smith`" by your own name and "`U12345678`" by your own BU id number.