Homework 6: Interactive Programming

Due:

The homework for this chapter involves some larger problems. Don't start the work at the last minute!

All the directions from homework 2 apply to this homework. In particular, remember to hand in a printout for procedures, and either a calculation or a transcript showing your testing.

We highly recommend that you put each problem in a separate file, and when making a transcript use the Scheme procedure `load` to read that file in to the Scheme interpreter.

Some extra-credit problems that are not directly related to the chapter are suggested at the end of this homework. You can start on them any time you wish.

1 Section 6.2, Strings

1. (0 points; suggested practice) Do exercise 6.1. Check your work on the computer

2. (5 points; basic) Define a procedure `string-index` that takes two strings, `short` and `long` and returns the first index in `long` at which `short` occurs as a substring. If `short` does not occur as a substring in `long`, return -1. For example,

```
(string-index "bush" "Trees and bushes") ==> 10
(string-index "Tr" "Trees and bushes") ==> 0
(string-index "xyzzy" "Trees and bushes") ==> -1
(string-index "xyzzy" "xyzzy") ==> 0
(string-index "xyzzy" "x") ==> -1
(string-index "z" "xyzzy") ==> 2
```

3. (0 points; suggested practice) What does the procedure `substring-ref` in problem 6.2 do? Do problem 6.2 and check your answer on the computer.

4. (7 points) Do problem 6.3. Note that the empty string is a palindrome.

5. (5 points; extra credit) Define a procedure `zero-or-more?` that takes a string `pat` and a string `str` and returns `#t` if `str` is composed of 0 or more occurrences of `pat` and `#f` otherwise. For example,

```
(zero-or-more? "" "") ==> #t
(zero-or-more? "" "a b c") ==> #f
(zero-or-more? "hum" "") ==> #t
(zero-or-more? "hum" "humhum") ==> #t
(zero-or-more? "x" "xxxxxx") ==> #t
(zero-or-more? "x" "xxxxxx z x") ==> #f
```

6. (7 points; extra credit) Define a procedure `sep-by` that takes a list of strings `los`, a string `sep` and returns a string containing the elements of `los` separated by `sep`. For example,

```
(sep-by (list "a" "b" "c") ", ") ==> "a, b, c"
(sep-by (list "a" "b" "c") ".") ==> "a.b.c"
(sep-by '() "x") ==> ""
```

2 Section 6.3, Implicit Begin

7. (0 points; suggested practice) Do problem 6.4. The code for this problem is in `/home/cs227/lib/ch6.ss`.

8. (7 points; basic) This is an intentionally tricky problem, so watch out! Give the output of the following code. Circle the printed output, and show what is returned by underlining it.

```
(let ((x 3))
  (writeln "(a) x is " x)
  (let ((x (begin (writeln "(b) x is " x) (+ x 1)))
        (y (begin (writeln "(c) x is " x) (+ x 1))))
    (writeln "(d) x is " x))
  (writeln "(e) x is " x)
  (+ x 4)
  (writeln "(f) x is " x)
  (* x x))
```

3 Section 6.3, Input and Output

9. (0 points; suggested practice) Do exercise 6.5. Check your work on the computer.

10. (10 points; basic) In this problem you will write an interactive procedure, `odds`, that takes a natural number `num-to-pick`, and prompts for a number, prints the chance that picking `num-to-pick` of that number things will win (say in a lottery). It continues prompting and printing until `stop` is entered, and then prints "Bye." and exits. If the number entered is n and `num-to-pick` is p, then the fraction to print is $1/(n!/(p! \cdot (n-p)!))$, where $p!$ is the factorial of p. (The number $(n!/(p! \cdot (n-p)!))$ is the number of possible ways to choose p things from n things without replacement.) For example, consider the following transcript.

```
> (odds 6)
Enter the number to choose from, or stop to quit:  10
The odds of getting 6 right out of 10 are: 4.761904761904762e-3
Enter the number to choose from, or stop to quit:  20
The odds of getting 6 right out of 20 are: 2.5799793601651186e-5
Enter the number to choose from, or stop to quit:  40
The odds of getting 6 right out of 40 are: 2.6052657631605e-7
Enter the number to choose from, or stop to quit:  42
The odds of getting 6 right out of 42 are: 1.9062920218247562e-7
Enter the number to choose from, or stop to quit:  stop
Bye.
>  (odds 1)
Enter the number to choose from, or stop to quit:  10
The odds of getting 1 right out of 10 are: .1
Enter the number to choose from, or stop to quit:  20
The odds of getting 1 right out of 20 are: .05
Enter the number to choose from, or stop to quit:  stop
Bye.
```

11. (10 points; extra credit) Modify the program in the previous problem so that it asks for both the number to pick and the number to choose from. Allow the user to stop at either prompt.

12. (15 points) Eva Lou Ator is going on a trip to sunny Italy to get away from the cold Iowa Spring. She wants to make sure she isn't cheated when buying goods in Italian shops, and so has decided to have you write her a program to tell her how much change she should get. She will try to sell this program for you to Italian businesses. The program should prompt for the

amount of a purchase, the amount paid, and then should describe how to give change using the minimum number of coins and bills. The Italian currency is called the "Lire". There are coins with values of 5, 10, 20, 50, 100, 200, and 500 Lire. There are bills with values of 1000, 2000, 5000, 10000, 50000, and 1000000 Lire. (A 1000 Lire is worth about one dollar.) The following is a sample transcript.

```
> (italian-cash-register)
price? 22345
Amount paid? 50000
Your change is 27655 Lire, which should be given as:
2 10000 Lire bills
1 5000 Lire bill
1 2000 Lire bill
1 500 Lire coin
1 100 Lire coin
1 50 Lire coin
1 5 Lire coin
Another? no
Bye
```

Note that getting the plurals right (bills vs. bill) is important for this program.

13. (0 points; suggested practice) Do problem 6.7. (Note that, according to the errata, if you have the first printing of the textbook, you should change "C" to "E" in the last two lines of page 177, and change "D" to "F", and on top of page 178, change step 5 to read: "Set R equal to F modulo 7".)

14. (25 points) Write an interactive program `histogram` that prompts for a list of integers, and prints a histogram (as shown below). It continues prompting and printing until an empty list is entered. The histogram uses asterisks to show occurrences. (Another name for this is a bar-graph.) For example,

```
> (histogram)
Enter a list of integers (no quote) or () to stop:
(3 4 3 20 0 20 8 8)
20: **
19:
18:
17:
16:
15:
14:
13:
12:
10:
 9:
 8: **
 7:
 6:
 5:
 4: *
 3: **
 2:
 1:
 0: *
```

```
Enter a list of integers (no quote) or () to stop:
(3 4 5 3 3 4)
5: *
4: **
3: ***
Enter a list of integers (no quote) or () to stop:
(-3 -4 2 2 2 2 0)
 2: ****
 1:
 0: *
-1:
-2:
-3: *
-4: *
Enter a list of integers (no quote) or () to stop:
()
```

You will only lose 1 point if you don't get the numbers to line up as in the above examples; if you don't line them up that way, use a tab to line up the asterisks. A string consisting of a single tab character is written `"\t"` in Scheme. (Hint: if you can't figure out where to start, think of a similar but easier problem and solve that first.)

15. (10 points; extra credit only) Add the ability to "bin" data in the histogram. That is, prompt the user for intervals in which to lump together points of data. Allow real numbers to be input as well as integers.

16. (20 points; extra credit only) Print the histogram horizontally instead of vertically. For example:

```
> (horizontal-histogram)
Enter a list of integers (no quote) or () to stop:
(3 4 5 3 3 4)
*
* *
* * *
3 4 5
Enter a list of integers (no quote) or () to stop:
()
```

4 Section 6.5, Two Famous Problems

17. (20 points) Write a procedure, `hanoi-states`, that takes a number `n` and returns a list of 3-element lists of numbers; these 3-element lists show how many disks are on each post as the moves are made by the priests for `n` disks. You may assume that `n` is at least 1.

```
(hanoi-states 3)
 ==> ((3 0 0) (2 0 1) (1 1 1) (1 2 0) (0 2 1) (1 1 1) (1 0 2) (0 0 3))
```

Hint: Don't try to do this iteratively. Is there any way to use work that's already been done for you to help with this? If all else fails, try using `reverse`.

18. (10 points; extra credit only) Do problem 6.8.

19. (30 points; extra credit only) Do problem 6.9. This is hard!

20. (10 points) Do problem 6.10.

21. (20 points) Do problem 6.11. Be sure to read the problem description in the book! The following is intended to clarify the book's description. In this problem, n is the length of the sequence, not a number that can appear in the sequence. Each good sequence can only contain the integers 1, 2, and 3. For example,

```
(good-sequences 0) ==> ()
(good-sequences 1) ==> ((3) (2) (1))
(good-sequences 2) ==> ((2 3) (1 3) (3 2) (1 2) (3 1) (2 1))
(good-sequences 3) ==> ((3 2 3) (1 2 3) (3 1 3) (2 1 3) (2 3 2) (1 3 2)
                        (3 1 2) (2 1 2) (2 3 1) (1 3 1) (3 2 1) (1 2 1))
(good-sequences 5) ==> ((3 1 3 2 3) (2 1 3 2 3) (2 3 1 2 3) (1 3 1 2 3)
 (3 2 1 2 3) (3 2 3 1 3) (1 2 3 1 3) (2 3 2 1 3) (1 3 2 1 3) (3 1 2 1 3)
 (3 1 2 3 2) (2 1 2 3 2) (2 3 1 3 2) (3 2 1 3 2) (1 2 1 3 2) (3 2 3 1 2)
 (1 2 3 1 2) (2 1 3 1 2) (2 3 2 1 2) (1 3 2 1 2) (1 3 2 3 1) (3 1 2 3 1)
 (2 1 2 3 1) (3 2 1 3 1) (1 2 1 3 1) (1 2 3 2 1) (3 1 3 2 1) (2 1 3 2 1)
 (2 3 1 2 1) (1 3 1 2 1))
```

Note that the order of the top-level items in the output of a call to good-sequences does not matter. For example, your procedure may give the result for 2 as follows,

```
(good-sequences 2) ==> ((2 1) (3 1) (1 2) (3 2) (1 3) (2 3))
```

or any other order with the same top-level items.

22. (0 points; suggested practice only) Do problem 6.12. This may help you visualize the backtracking process.

23. (0 points; suggested practice only) Do problem 6.13.

24. (10 points; extra credit only) Do problem 6.14.

25. (7 points; extra credit only) Do problem 6.15.

26. (0 points; suggested practice only) Do problem 6.16.

5 Extra Credit Explorations

27. (20 points; extra credit) If you have PC Scheme or access to a PC, read chapter 7 of *Programming in Scheme* by Eisenberg (on reserve at the library) and do exercises 1 and 5 on pages 76 and 77. (You can do just one of these for 10 points each.) These exercises show how to draw pictures using graphics-mode in PC Scheme. Note that in Scheme

```
(define (foo x y)
  ; ...
  )
```

is equivalent to

```
(define foo
  (lambda (x y)
    ; ...
   ))
```

28. (points given below; extra credit) Stan Schemer is a baseball fan. He likes to follow pitchers. He wants you to write some programs to figure some statistics about pitchers based on historical data.

In baseball, a pitcher throws the ball to start the play in motion. The pitcher's objective is to prevent the player on the opposing team (called the batter) from hitting the ball safely. When the batter does not hit the ball safely, the batter is "out". Each team gets 27 outs in a normal game; a normal game consists of 9 innings, each of which has 3 outs for each team. The game is won by whichever team scores the most runs; runs are earned by hitting the ball safely (and by some other things). Thus the pitcher both wants to get the batter out, and to prevent runs from scoring, to help the pitcher's team win games.

Some statistics for pitchers are related directly to games. For example, the number of games the pitcher plays (G), the number of games the pitcher "starts" (GS) (that is in which the pitcher is the first pitcher for his team), the number of games the pitcher both starts and finishes (CG or "complete games"), the number of games the pitcher wins (W), loses (L), and saves (SV) for another pitcher, the number of complete games in which the other team scores no runs (SHO).

Other statistics are related to the number of innings a pitcher pitches (IP). The earned run average (ERA) is the number of runs the opposing team earns, times nine, divided by the innings the pitcher pitched. For reasons to be discussed below, we also offer the ERA of the league in which the pitcher pitched (LGERA) for that year.

Finally, some statistics just count events. If the batter fails to hit the 3 pitches in the "strike zone" then the batter "strikes out", which means that the batter is out. These strike outs are counted tabulated under the heading "SO". If the pitcher fails to throw 4 pitches in the strike zone, then the batter gets a "base on balls" (BB). Also counted are the number of times the opposing batters hit safely (H), and the number of "home runs" (HR). (Home runs are hits that immediately score the batter).

A pitcher's statistics are tabulated for several years. For example, here are the statistics for two famous pitchers for some of their best years.

```
     YEAR   W  L  G GS CG SHO SV  IP   H HR  BB  SO  ERA LGERA
Walter Johnson
     1910  25 17 45 42 38   8  1 374 269  1  76 313 1.35  2.51
     1911  25 13 40 36 36   6  1 323 292  8  70 207 1.89  3.34
     1912  33 12 50 37 34   7  2 368 259  2  76 303 1.39  3.34
     1913  36  7 48 36 29  11  2 346 232  9  38 243 1.14  2.93
     1914  28 18 51 40 33   9  1 372 287  3  74 225 1.72  2.73
Sandy Koufax
     1963  25  5 40 40 20  11  0 311 215 18  58 306 1.88  3.29
     1964  19  5 29 28 15   7  1 223 154 13  53 223 1.74  3.54
     1965  26  8 43 41 27   8  2 336 216 26  71 382 2.04  3.54
     1966  27  9 41 41 27   5  0 323 241 19  77 317 1.73  3.61
```

(a) (5 points; extra credit) Write an interactive program that takes lines of input as above, and produces a list for each line; for example,

```
( 1910  25 17 45 42 38   8  1 374 269  1  76 313 1.35  2.51 )
```

It's up to you to decide on prompts, error checking, etc.

(b) (5 points; extra credit) Write procedures that take a year record as a list like in the previous part, and return the following statistics:

- The won-loss fraction (PCT) for the year. This is defined as W/(W+L). For example, in 1910, Walter Johnson's won-loss fraction was approximately 0.595. (Do round off this to 3 decimal places, as the true fan always wants it that way.)

- Hits per game for the year. This is not defined, as the name would indicate, as H/G, but as H/(IP/9). The reason is that a pitcher may play a different number of innings per game. For example, in 1910, Walter Johnson's hits per game was approximately 6.5.
- Bases-on-balls per game, defined as BB/(IP/9). For example, in 1910, Walter Johnson's bases-on-balls per game was approximately 1.8.
- Strikeouts per game, defined as SO/(IP/9). For example, in 1910, Walter Johnson's strikeouts per game was approximately 7.5 (which lead the American League!).

(c) (5 points; extra credit) The above statistics are not the best way to compare pitchers, especially pitchers like Koufax and Johnson who pitched in different leagues at different times. To be more precise, the book *Total Baseball* (edited by Thorn and Palmer, Warner Books, 1989) uses "pitching runs". Pitching runs (PR) are defined by the formula: (IP/9)×(ERA - LGERA). The idea is to measure the runs saved beyond what a league average pitcher might have saved (where the league average pitcher's runs are defined as zero). Write a procedure that takes a year record as a list and returns the pitcher's pitching runs. For example, in 1910, Walter Johnsons pitching runs was 49 (this was close to the lead in his league). Who had the best year: Johnson or Kofax?

(d) (10 points; extra credit) Write an interactive program to prompt for a year record for a pitcher (not as a list), and to print in response the above statistics. For example:

```
> (year-pitching-averages)
Enter the pitcher's record for the following items (or quit)
     YEAR   W  L   G GS CG SHO SV  IP   H HR  BB  SO  ERA LGERA:
     1911  25 13 40 36 36   6  1 323 292  8  70 207 1.89  3.34
YEAR  W  L  PCT  G GS CG SHO SV  IP   H H/G HR  BB BB/G SO  SO/G  ERA LGERA PR
1911 25 13 .658 40 36 36   6  1 323 292 8.1  8  70  2.0 207  5.8 1.89  3.34 52
Enter the pitcher's record for the following items (or quit)
     YEAR   W  L   G GS CG SHO SV  IP   H HR  BB  SO  ERA LGERA:
quit
Bye.
```

The new columns are as follows: PCT is the won-loss fraction, H/G is hits per game, BB/G is bases-on-balls per game, SO/G is strikeouts per game, and PR is pitching runs.

(e) (10 points) Write a procedure, `career-pitching-runs`, that takes a list of year record lists, and returns the pitcher's pitching runs for all of those years. This is not the average of PR for each year, but a recomputation of the formula, weighted by innings pitched. That is, figure the earned runs for each year, ER = ERA * (IP/9), and add up all of the earned runs for the career. Then figure the career ERA as ERA = ER/(IP/9), where both ER and IP should be career totals. Then use that and the average of the LGERAs to figure PR.

29. (5 points; extra credit only) This problem is about doing Unix programming in Scheme.

The simplest way to make a Unix command call a Scheme program is by using the Unix Shell programming language. A simple version of this is your file `~/bin/scheme`. Here is another one...

```
#!/bin/sh
# fact -- factorial function
scheme fact.ss <<!
(fact $1)
(exit)
!
```

The first two lines are comments. The first comment is a special comment that tells Unix what kind of shell this is written in (the Bourne shell). The characters `<<!` tell the shell that what

comes afterwards is a "here document", processed as input to Scheme. The input to Scheme ends with the `!` on the last line. The argument to the `fact` function comes from the command line; that is `$1` means the first command line argument. If you put this in a file, `fact.sh`, change it to be executable by executing the Unix command

```
chmod +x fact.sh
```

you can then run it as follows

```
% fact.sh 4
Chez Scheme Version 4.1
Copyright (c) 1991 Cadence Research Systems

loading /home/cs227/lib/atomic-item.ss ...
loading /home/cs227/lib/print-poly.ss ...
> 24
> %
```

To get rid of the prompts and the 227 specific loading messages, we can make a slight change as follows to fact.sh

```
#!/bin/sh
# fact -- factorial function
/home/scheme/bin/dec/scheme /home/cs227/lib/call-from-shell.ss fact.ss <<!
(fact $1)
(exit)
!
```

The code in `call-from-shell.ss` turns off the > prompts. The use of `/home/scheme/bin/dec/scheme` bypasses your `~/bin/scheme` program, and thus suppresses the loading of the Com S 227 file `springer-friedman.ss`.

```
% fact.sh 4
Chez Scheme Version 4.1
Copyright (c) 1991 Cadence Research Systems

24
%
```

The use of `call-from-shell.ss` is not necessary for interactive Scheme programs that have no command line arguments. They can exit by putting a call to `exit` in the program itself. For example, if the interactive Italian cash register program is in `italian-cash-register.ss`, then the following shell script will make run the code.

```
#!/bin/sh
/home/scheme/bin/dec/scheme /home/cs227/lib/writeln.ss italian-cash-register.ss
```

You have to put this in a file, and make it executable as described above.

Convert your Italian cash register program to run as a Unix program as described above. Test the program. Describe any problems you encounter. Hand in a printout of your Scheme code and your Unix shell script.