CSCI 4150 Introduction to Artificial Intelligence, Fall 2002
Assignment 1 (73 points): out Friday August 30, due Friday September 6

Notes on this assignment

- There is one support code file for this assignment: `a1code.scm`. You can download this from the course home page. Your solutions should load this file; *do not copy this code into file you turn in!*
- I will also provide a mostly empty file into which you should place your code. This file, `assign1.scm`, will also be on the course home page. It will contain a command to load the `a1code.scm` file as well as stubs for the procedures you need to write.
- Problems 1 and 2 are to be turned in on paper. The rest are to be turned in electronically. (See course home page for details.)
- You may always assume (on all assignments in this class) that your procedures will be given valid inputs.

Questions

All section references are to "How to Solve Problems Using Scheme".

1. (Written; 6 points) Do execrcise 1, parts (a)–(m) from Section 4.2.1 (on legal symbols).
2. (Written; 4 points) Do exercise 2 from Section 4.2.1 (on symbol similarity).
3. (12 points) The Income Tax problem: do exercise 1 from Section 6.2.
4. (14 points) List creation: do exercise 1, parts (b)–(h) from Section 8.1.4.

   In order to turn in your solutions electronically, you must write an procedure for each part. For example, a solution for part (a) would be:

   ```
   (define (p4a)
     (cons four numbers))
   ```

   The definitions given in this problem (i.e. `colors`, `numbers`, `nested-list`, and `four`) will be included in the `a1code.scm` file.

   Following this naming pattern, you will write procedures called `p4b`, `p4c`, ... `p4h`, all of which take zero arguments. To check your answer, you can call the procedure. For example:

   ```
   > (p4a)
   ;Value: (4 1 2)
   ```

5. (14 points) List access: do exercise 1, parts (b)–(h) from Section 8.4.

   Similar to the previous problem, you will write the procedures `p5b`, `p5c`, ... `p5h`, which all take zero arguments.

6. (4 points) Suppose we represent a date as a list where:

   - the first element is the one of the symbols: `jan`, `feb`, `mar`, `apr`, `may`, `jun`, `jul`, `aug`, `sep`, `oct`, `nov`, and `dec`
   - the second element in the day of the month, a positive integer no greater than 31

For example:

```
(mar 27)
```

Write the "accessor functions" `(date:month date)` and `(date:day date)`. For example:

```
> (date:month '(jul 14))
;Value: jul

> (day:month '(dec 6))
;Value: 6
```

7. (6 points) Write the procedure `(days-in-month m)` where `m` is assumed to be one of the three letter month symbols above. This procedure should return the number of days in that month. For example:

   ```
   > (days-in-month 'jul)
   ;Value: 31
   > (days-in-month 'feb)
   ;Value: 28
   ```

   Assume that we are dealing with a non-leap year.

8. (8 points) Write the procedure `(days-until-month m)` where `m` is one of the three letter month symbols above. This should return the total number of days in all the months that precede the month `m`. For example:

   ```
   > (days-until-month 'jan)
   ;Value: 0
   > (days-until-month 'mar)
   ;Value: 59
   ```

   Actually, you cannot do this using what Scheme we have covered in the first week. However, in the `a1code.scm` file, I have provided a procedure `(add-up-days months)` where `months` is a list of three letter month symbols. It returns the sum of the days in all months in the list. For example:

   ```
   > (add-up-days '())
   ;Value 0
   > (add-up-days '(jan feb))
   ;Value 59
   ```

   Using this procedure, you can write the `days-until-month` procedure. Again, assume that we are dealing with a non-leap year.

9. (5 points) Write the procedure `(date->daynumber date)` where `date` is in the date format above. It should return the day of the year. For example:

   ```
   > (date->daynumber '(jan 1))
   ;Value: 1
   > (date->daynumber '(jan 31))
   ;Value: 31
   > (date->daynumber '(feb 1))
   ;Value: 32
   ```

   As before, assume that we are dealing with a non-leap year.