Homework 1: Scheme Data and Operators

Due:

The section headings below give the sections of *Scheme and the Art of Programming* that you should read to help yourself with the problems.

1 Sections 1.1-1.3

1. (0 points; suggested practice) Do problems 1.1, 1.2, 1.3, and 1.4 (on page 13). For problem 1.1, the way to get a transcript is to use the Scheme procedure `transcript-on` as described in the course handout *How to Run Scheme*, section 1.3. To check your answers for problem 1.2, enter the definitions first, then the various parts. For problems 1.3 and 1.4 do as many parts as you need to be completely confident and quick.

   For these problems, it is important to think first, and check your work on the computer. Use the computer for feedback, not to think for you.

2. (3 points) Suppose that in a program you want to refer to the value of a variable `miles` and also want to use the word `miles` as data. How can you distinguish these two uses of `miles` in Scheme?

   (You can handwrite your answer to this.)

3. (6 points; basic) Suppose α, β, and γ are three numbers. Translate the following Scheme expressions into the usual arithmetical notation. For example: `(-` α `(-` γ β`))` translates into $(\alpha - (\gamma - \beta))$.

   (a) `(*` α `(* (+` β γ`)` β`))`

   (b) `(+ (*` α β`) (/` γ β`))`

   (c) `(- (*` α α`) (*` β β`))`

   As Greek letters are hard to print on US computers, turn in a handwritten answer for this problem.

4. (6 points; extra credit only) An *algorithm* is a precise description of a computational process, like a recipe in cooking, only more specific. Using English, write down two algorithms that another person could carry out for doing the two transformations described in exercises 1.4 and 1.5 (to and from "cambridge prefix notation"). What primitive operations do you expect a "computer" that carries out your algorithms to perform?

   You can handwrite your answer to this problem.

2 Section 1.4, Constructing Lists

5. (0 points; suggested practice) Do problem 1.6 and check your answers on the computer. (Don't leave out parts d and e, as they are somewhat different from parts a-c.)

6. (6 points; basic) The nesting of lists can be thought of as a tree structure. This is often used in artificial intelligence to do things like representing sentences in English. For example, the sentence "IBM sells it" could be represented by the list

   ```
   (sentence (np (noun IBM)) (vp (verb sells) (object it)))
   ```

   If you format it to show the tree structure more clearly, it looks like the following.

```
(sentence
  (np (noun IBM))
  (vp (verb sells)
      (object it)))
```

As in the book's problem 1.6, without using quoted lists (that is, using only `cons`, quoted symbols, and the empty list `'()`), define the the lists:

(a) `(object it)`,

(b) `(vp (verb sells) (object it))`, and

(c) `(sentence (np (noun IBM)) (vp (verb sells) (object it)))`.

Turn in a printout of a transcript (of a Scheme session) showing how to build these. Use `define` to reuse the answer of part (a) in part (b), and of part (b) in part (c). Thus what you type to the Scheme interpreter for your solution can look like the following. (Lines starting with a semicolon (`;`) are comments, and ignored by the Scheme interpreter.)

```
;; NAME: <your name here>     TA: <your grading TA's name here>
;; homework 1, problem 6
(define part-a (cons 'object ...))
part-a
(define part-b (cons ... part-a ...))
part-b
(define part-c (cons ... part-b ...))
part-c
```

Note that after you define `part-a`, to check to see if your definition is correct, you have to enter the variable name, `part-a`, at the Scheme prompt. That is why `part-a`, all by itself, follows the definition of `part-a`.

7. (0 points; suggested practice) Do problem 1.7.

3 Section 1.5, Taking Lists Apart

8. (0 points; suggested practice) Do problems 1.9 (page 22), 1.10 and 1.11 (page 25). Check your answers on the computer.

9. (0 points; suggested practice) Do as many parts of problem 1.12 as you need to feel comfortable and confident with `car` and `cdr`. Check your answers on the computer.

10. (8 points; basic) The ability to extract parts of a list using `car` and `cdr` is important in writing procedures that manipulate interesting list structures with unknown parts. Although this problem, like exercise like 1.13 in the text, has you manipulating list structures that are completely known, this is just a convenience to make the problem clear.

    In this problem, you are to write an expression around a given list, so that the entire expression returns the symbol `it`. We say that the expression "extracts the symbol `it` from the list." For example, we can extract the symbol `it` from the list `(walk (slowly towards) it)` using the expression `(car (cdr (cdr '(walk (slowly towards) it))))`. (See also exercise 1.13.)

    Write expressions using `car` and `cdr` to extract the symbol `it` from the following lists.

    (a) `(I he she it)`

    (b) `((german english) (das it))`

    (c) `((it pronoun) (go verb))`

    (d) `((((not) it)))`

Hand in a printout of a transcript (of a Scheme session) with your answers clearly labeled by comments, showing that your expression works. For example, our example of extracting `it` from `(walk (slowly towards) it)` would be typed as

```
;; homework 1, problem 10 example
(car (cdr (cdr '(walk (slowly towards) it))))
```

11. (0 points; suggested practice) Do problems 1.14 and 1.15.

12. (8 points) Ben Bittwiddle wonders what lists in Scheme are good for. John McCarthy says "nearly anything," but Ben wants examples. One example is English sentences; we have seen that a sentence can be represented as a list of the form `(sentence ...)` with nested lists describing each grammatical phrase, and each part of speech appearing in a list such as `(noun ...)`. For example, one can represent the English sentence "IBM sells it" by the following list.

```
(sentence
  (np (noun IBM))
  (vp (verb sells)
      (object it)))
```

As another example, we can represent relations (tables) by lists of the form `((`*key value*`) ...)`. For example the relation of the colors of a traffic light to their meanings would be represented by the following list

```
((green go) (yellow caution) (red stop))
```

There is usually more than one way to represent information. For example, one could also represent relations by a list of 2 lists of the same length, one of the keys, and the other of the associated values; with the ith element of the keys list related to the ith element of the values list. For example, with this representation the traffic light meaning relation would be represented as the following list.

```
((green yellow red) (go caution stop))
```

Show Ben at least one way to represent the following kinds of information as lists. For each one, give the general pattern in English, and how our example would be represented.

(a) Numbers with units. For example, 3 minutes.

(b) The time of day. For example, 3:10 PM.

(c) A catalog of a collection of records or compact disks, organized by type of music. For example, rock music: *Abbey Road* by The Beetles, *Like a Virgin* by Madonna, jazz: *Newport in New York* by various artists, *Take the A Train* by Ellington, and classical: *Symphony number 3 in E-flat* by Beethoven.

(d) Information about countries of the world, whether they belong to the United Nations, and the year they were admitted. For example, the USA is a charter member, Vatican City is not a member, Switzerland is not a member, Poland has been a member since 1945.

To see if your answer is good, ask yourself if you can tell someone (or Scheme) how to extract all the relevant information. In parts (c) and (d), make sure you can do this for examples of different sizes.

13. (6 points; extra credit only) As in the previous problem, how would you represent:

(a) A plan of study for a student that shows which courses must be taken before others (prerequisites). For example, the courses you have to take to get your degree at ISU,

(b) An outline of a book. For example the first part of *Scheme and the Art of Programming*.

(c) An algorithm. For example: if `n` is even, then divide `n` by 2, otherwise multiply `n` by 3 and add 1.