CS61A – Homework 3.2 **Kurt Meinz**
University of California, Berkeley Summer 2002

Topic: Representing abstract data

Abelson & Sussman, Sections 2.4 through 2.5.2 (pages 169–200)

Lectures: Wednesday 7/10, Thursday 7/11

Reading: Homework due 10 AM Monday, 7/15:

Abelson & Sussman, exercises 2.75, 2.76, 2.77, 2.79, 2.80, 2.81, 2.83

Note: Some of these are thought-exercises; you needn't actually run any Scheme programs for them! (Some don't ask you to write procedures at all; others ask for modifications to a program that isn't online.)

Extra for experts:

Another approach to the problem of type-handling is *type inference.* If, for instance, a procedure includes the expression `(+ n k)`, one can infer that `n` and `k` have numeric values. Similarly, the expression `(f a b)` indicates that the value of `f` is a procedure.

Write a procedure called `inferred-types` that, given a definition of a Scheme procedure as argument, returns a list of information about the parameters of the procedure. The information list should contain one element per parameter; each element should be a two-element list whose first element is the parameter name and whose second element is a word indicating the type inferred for the parameter. Possible types are listed on the next page.

- `?` (the type can't be inferred)
- `procedure` (the parameter appeared as the first word in an unquoted expression or as the first argument of `map` or `every`)
- `number` (the parameter appeared as an argument of `+`, `-`, `max`, or `min`)
- `list` (the parameter appeared as an argument of `append` or as the second argument of `map` or `member`)
- `sentence-or-word` (the parameter appeared as an argument of `first, butfirst, sentence,` or `member?`, or as the second argument of `every`)
- `x` (conflicting types were inferred)

Continued on next page.

Homework assignment 3.2 continued...

You should assume for this problem that the body of the procedure to be examined does not contain any occurrences of `if` or `cond`, although it may contain arbitrarily nested and quoted expressions. (A more ambitious inference procedure both would examine a more comprehensive set of procedures and could infer conditions like ”nonempty list”.)

Here’s an example of what your inference procedure should return.

```
(inferred-types
  '(define (foo a b c d e f)
    (f (append (a b) c '(b c)) (+ 5 d) (sentence (first e) f)) ) )
```

should return

```
((a procedure) (b ?) (c list) (d number)
 (e sentence-or-word) (f x))
```

If you’re *really* ambitious, you could maintain a database of inferred argument types and use it when a procedure you’ve seen is invoked by another procedure you’re examining!

Unix feature of the assignment: `du`, `df`, `quota`

Emacs feature of the assignment: `M-q` (format paragraphs), `C-M-q` (format Scheme code)