All functions in Lisp are written in prefix notation: If `f` is a defined function, then

```
(f e0 e1 e2 ... en)
```

applies `f` to the $n+1$ arguments `e0`, `e1`, `e2`, ..., `en`. Thus

```
(+ 1 -2 3)
```

adds the three numbers and returns the value 2;

```
(<= 1 2 3)
```

checks that the three numbers are in nondecreasing order and yields true (`t` in Lisp).

Lists are Lisp's main data structure. To construct a list `(e0 e1 e2 ... en)` the expression

```
(list e0 e1 e2 ... en)
```

is used. The function `(nth i l)` extracts the *i*th element of the list `l`, indexing from zero; the predicate `(member x l)` tests if `x` is an element of `l`. To get the first (indexed zero), second, or third elements of a list, we use the functions `first`, `second`, and `third`, respectively. The empty list is represented by `nil`.

Functions are defined using the `defun` command, which has the following syntax:

```
(defun function-name (param1 ... paramn)
   expression)
```

For example, we can define a function (unavailable in Common Lisp) to return the (truncated) integer quotient of two integers:

```
(defun quotient (m n)
   (floor m n))
```

We will represent all dates on the Gregorian, Julian, Islamic, and Hebrew calendars by a list of the form (*month day year*) in which *month*, *day*, and *year* are each integers. (Common Lisp places no *a priori* upper bound on the size of integers; none of our calculations require more than 32-bit integers for dates in the next twenty thousand years; 24 bits suffice for all of the calculations, except as noted.) To extract the individual components of such a date we use the following access functions:

```
(defun extract-month (date)
;; Month field of date = (month day year).
  (first date))
(defun extract-day (date)
;; Day field of date = (month day year).
  (second date))
(defun extract-year (date)
;; Year field of date = (month day year).
  (third date))
```