Harvey Mudd College
Computer Science 60
Fall 2010

Assignment 3
Unicalc API and CLI
Due. 11:59 p.m., Wed., 22 September 2010
Pair-programming is optional on this assignment

Unicalc is a calculator that includes physical and other units, rather than just numbers. **CLI** stands for "Command-Line Interface", an application that executes user commands from a command line. The CLI we describe here is not the ultimate in user-friendliness, but should be regarded as a prototype for a more user-friendly interface to be done in a subsequent assignment.

Units are important to computation because they eliminate an element sometimes left to human interpretation. In the area of engineering, failure to interpret numbers without their units specified has been known to lead to failure of space missions for example. In the wikipedia article about NASA's Mars Climate Orbiter:

http://en.wikipedia.org/wiki/Mars_Climate_Orbiter

it is stated:

> "The Mars Climate Orbiter was intended to enter orbit at an altitude of 140–150 km (460,000-500,000 ft.) above Mars. However, a navigation error caused the spacecraft to reach as low as 57 km (190,000 ft.). The spacecraft was destroyed by atmospheric stresses and friction at this low altitude. The navigation error arose because a NASA subcontractor (Lockheed Martin) used Imperial units (pound-seconds) instead of the metric system."

A **Unicalc Quantity** consists of three parts:

- A number, called the **multiplier**
- A flat list of symbols, called the **numerator**
- A flat list of symbols, called the **denominator**

(*Flat* means there is no nesting.) We use symbols rather than strings because they are more convenient and efficient for our purposes. We don't have to do much with the content of strings in this assignment.

Our API specifies that a Quantity is represented by a list of these components, in this order, for example, using a Racket literal, the following would represent 2.5 kg m/s^2:

```
'(2.5 (kg meter) (second second))
```

For this problem, Quantities will be entered as test cases as literals. The CLI will be a Racket-like interpreter that accepts arithmetic expressions containing Quantities.

On the main course webpage, you will find **unicalc-db.rkt**. This is a Racket source file for the Unicalc **database**. The database is, in effect, a set of equations defining single symbols, called **units**, in terms of Quantities. As a Racket structure, it is an association list, **unicalc-db**. You will want to download this file and load it with your solution to this assignment, using this command after `#lang racket`.

```
(require "unicalc-db.rkt") .
```

The API is a set of functions that you will construct, as described below, along with other helper functions. To describe the API functions, we need a couple more definitions.

A symbol that is defined in the database (as the first element of an element of the association list `unicalc-db`) is called a **defined unit**. A symbol that is not defined is called a **basic unit**. Examples of defined units are: mile, day, pound. Examples of basic units are: kg, second, meter.

Defined units can be **expanded** into equivalent Quantities consisting of only basic units in one or more steps. (You can assume for now that there are no circular definitions in the database.) Also, basic units can be converted into quantities by making them the only element in the numerator list, accompanied by a multiplier of 1 and an empty denominator list.

A Quantity is called **normalized** provided that the following conditions are true:

- The numerator and denominator consist of only basic units.
- The numerator and denominator are *sorted* in ascending alphabetic order.
- No unit appears in both the numerator and denominator.

We don't require the **user** to provide normalized Quantities, but it helps make the processing more efficient if we use them exclusively inside our functions. For example, the sorted aspect allows us to use the "merge pattern" described in the appendix for normalization.

Here are the functions you need to provide in the API. Note that **divide** effectively achieves conversion from one kind of normalized Quantity to another.

Function Call Form	Meaning
(**`normalize-unit`** Unit)	Returns a normalized Quantity for a single unit.
(**`normalize`** Quantity)	Converts any Quantity to a normalized Quantity.
(**`multiply`** Quantity1 Quantity2)	Multiplies two normalized Quantities, returning a normalized Quantity.
(**`divide`** Quantity1 Quantity2)	Divides normalized Quantity1 by normalized Quantity2, returning a normalized Quantity.

Note that the database, `unicalc-db`, is global and behind the scenes, rather than being passed as an argument. Test cases will be provided as **`uncialc-tests.rkt`**. You will want to use the merge pattern in implementing some of these functions.

The CLI is started by a read-eval-print loop (REPL), which reads an S expression from the user, then prints the result.

The following are legitimate input ***expressions*** for the CLI (defined *recursively*):

a numeral (integer, rational, or scientific notation)

a symbol *not* beginning with $, which will be interpreted as a unit (such as mile, meter, second, etc.)

a symbol beginning with $, which will be interpreted as a user-defined *variable* that has a Quantity as its value.

(/ N D) where N and D are expressions, returning the quotient N divided by D, as a normalized Quantity

(* E_1 E_2 …) where E_1 E_2 … represents zero or more expressions, returning the product of its arguments, as a normalized Quantity. The result for zero arguments is the Quantity 1.

(**define** V E) where V is a variable (such as $x) and E is any expression. This sets the variable V to have the *normalized* value of the expression E in the current context.

Here are some examples of input to your CLI and the corresponding output:

```
> 3.14
‘(3.14 ()())

> mile
‘(1609.3149259968 (meter) ())

> (/ mile hour)
‘(0.447031923888 (meter) (second))

> (/ (* acre foot) year)
‘(3.9111332681348173e-05 (meter meter meter) (second))

> (/ joule (/ (* kg m m)(* second second)))
‘(1 ()())

> (define $x (* acre foot))
‘(1233.414987438996 (meter meter meter) ())

> (/ $x (* meter meter meter))
‘(1233.414987438996 ()())
```

Note that conversion from one type of unit to another is accomplished just by dividing the first by the second:

```
> (/ light_year mile)
‘(5878731576867.0205 ()())

> (/ acre liter)
'(4046710.204743883 () (meter))

> (/ (/ tadpole gallon) (/ tadpole liter))
‘(0.2641863702726775 ()())
```