$Id: asg2-ocaml-dc.mm,v 1.18 2017-10-11 15:18:18-07 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps112-wm/Assignments/asg2-ocaml-dc
URL: http://www2.ucsc.edu/courses/cmps112-wm/:/Assignments/asg2-ocaml-dc/

1. Overview

In this assignment, you will implement a desk calculator in Ocaml, a language with strong static type checking. Your program will be a strict subset of `dc`(1), although it will not have all of its functions. Begin by reading the man page for `dc`(1) and experimenting with it. Study its input format, output format, error messages, and exit status.

Your program will read the single file (if specified) as does `dc` and then read `stdin`. Implement the following `dc` functions: `+ - * / % ^ c d f l p s`. Note that all of these letters are lower case.

2. Implementation Notes

(1) You ***may not use*** the `Num` or `Big_int` modules in the Ocaml library. Instead, you will implement your own version of bigint by representing an integer by a product of a sign with a list of integers. The largest value of int in Ocaml is

$$4611686018427387903 = 0\text{x}3\text{FFFFFFFFFFFFFFF} = 2^{62} - 1,$$

which is one bit less than what you might normally expect. This is because one bit is used in each word for tagging.

(2) The ideal representation therefore would be to use eight-digit numbers in a list. However, in order to make sure that the lists are working, which is the point of the assignment, you can store only one digit in each element of a list. This is incredibly wasteful of storage, but possible makes the representation easier. The `dc` utility actually uses character arrays with two decimal digits per byte.

(3) Since arithmetic operations proceed from the lowest order digit to the highest, represent your numbers with the lowest order digit at the front of the list and the leftmost digit at the end.

(4) ***Do not*** use any loops in your program. All iteration should be done via recursion, and whenever possible, by using higher-order functions like map.

(5) First implement input and output of numbers. Make sure your output duplicates `dc` for very large numbers. Note that an underscore prefixing a number makes it negative. The minus sign is strictly for subtraction.

(6) Next, implement addition and subtraction. To do this, you will need two functions `add` and `sub` which just compare signs and then call `add'` or `sub'` as appropriate to do the work on their absolute values. When you subtract, make sure that the first argument is always the larger one.

(7) You will need a function `cmp` which returns a comparison value in the same way as does `strcmp` in C. This can move from the low order digits to the high order digits tail recursively and stop at the end of the shorter list, or by maintaining an actual comparison when the two lists turn out to be the same length.

(8) Make sure that you always canonicalize your answers by deleting leading 0 digits. This is only an issue with absolute subtraction, since addition can only lengthen the number. All other operations are implemented in terms of addition and subtraction.

(9) To implement multiplication, you add appropriate elements of the right column. To implement division, you add appropriate elements of the left column. The remainder is just whatever is left over after finishing the division, so your division function should return two results as a tuple, namely the quotient and remainder, and the main module then ignores the one not wanted.

(10) Exponentiation will then be trivial, since it is a simple matter to call the other functions.

3. Interactive Use

As with some other languages, **ocaml** can be used interactively for testing purposes:

```
-bash-1$ ocaml
        OCaml version 4.02.1
# let f x y = x + y;;
val f : int -> int -> int = <fun>
# f 3 4;;
- : int = 7
# ^D
```

You will, of course, need to add it to your **$PATH**:

```
-bash-2$ which ocaml
/afs/cats.ucsc.edu/courses/cmps112-wm/usr/ocaml/bin/ocaml
```

Unfortunately, while interacting with **ocaml**, the up arrow can not be used to recover earlier lines. Using it just causes **ocaml** to print **^[[A**. To get around this, you can use **rlwrap**, as in

```
-bash-3$ rlwrap ocaml
```

Then you can use the arrow keys to navigate.

```
-bash-4$ which rlwrap
/afs/cats.ucsc.edu/courses/cmps112-wm/usr/rlwrap/bin/rlwrap
```

4. What to Submit

Makefile, **bigint.ml**, **bigint.mli**, **maindc.ml**, and **scanner.mll**. Note that **scanner.ml** is a generated file and should be made by the **Makefile**. Also, **dc.ml** is a debugging tool, not to be submitted. Testing will be done on the **ocamlrun** script **ocamldc**, which should be runnable from the commandline.

Program testing: Test data will be fed to **dc**(1) as well as to your program and the output will be checked with **diff**(1).