Lab – 7

Week of Mar. 4, 2018

1 Instructions

1. Use the OCaml top-level to develop and debug your code.
2. Please write the code in a file first and then test your code in the OCaml top-level using the directive *#use "foo.ml"* for the file *foo.ml*
3. You may assume that all inputs are valid unless otherwise stated in the problem.
4. In questions that require string outputs, be careful not to include leading or trailing whitespace.
5. You may submit and evaluate your code a *maximum* of 15 times without penalty. Subsequently, you will lose 2 marks per additional evaluation. Therefore, please ensure that you have thoroughly debugged your code before submitting.

The following submission file is *required*:

1. `BigInt.ml`

2 Learn on your own

1. Learn how to read from and write to a file (or channel) here: `https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html`. Here is a short example of how to do so.

Reading file using standard input

```
let rec read_input () =
(* try ... with used for exception handling*)
try
        print_string (read_line()^"\n");    (*read_line() Read a line from stdin*)
        read_input ()                               (* recursive call*)
with e ->                                           (* catch exception e*)
        print_string ("File reading complete\n");;  (* reached end of file*)

read_input ();;
```

Writing to a file

```
open Printf                   (* module used for input-output*)

let file = "example.txt";;    (* name of the output file*)
let message = "Hi";;          (* message you want to write to the output file*)

(* Write message to file *)
let output_channel = open_out file in    (* create or truncate file, return channel *)
fprintf output_channel "%s\n" message;   (* write to the output channel *)
close_out output_channel;                (* flush and close the channel *)
```

2. Go through the module `String` from `https://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html` to learn how to manipulate strings. Some of the functions there will be useful in implementing the functions below.

3 Assignment

3.1 Functionality

This week you will implement a module called `BigInt`, to handle *large* non-negative integers that are > `max_int`. Define the module `BigNumber` that contains the type `bignumber`. You may choose whatever representation you like. Note that the domain of numbers is restricted to non-negative integers only. Implement functions to:

1. Add a given set of k numbers and return their sum
2. Subtract a number from another number.
3. Multiply a given set of k numbers and return their product
4. Divide one number by the other and return the quotient

3.2 How your program will be run

Your program will be run from the *command line* after compiling it into an executable. That is, your program will be compiled using the following command: `$ ocamlc -o bigint BigInt.ml`. The executable `bigint` is then run using the command: `$ ./bigint < inFile`, where `inFile` is a text file whose format is described in the next section. Note that, your program should consist of a *main* function that will read the contents of the `inFile` and then perform the operations as specified.

3.3 Input

The input to your program will be a file called `inFile` that will consist of multiple lines, each with the following format:

1. The *first* line of the file always contains the name of the output file into which you should write your results.
2. From the *second* line onwards, the file contains a line with one of the following formats:
   (a) `ADD <#1> <#2> ... <#k>` – note that each number is separated by exactly one space and the first number is separated from the command by one space as well. *Expected output:* The sum of the k numbers. $k >= 2$
   (b) `SUB <#1> <#2>` – *Expected output:* The second number subtracted from the first. You may assume that the second number is smaller than or equal to the first.
   (c) `MULT <#1> <#2> ... <#k>` – *Expected output:* Product of the k numbers. $k >= 2$
   (d) `DIV <#1> <#2>` – *Expected output:* The first number divided by the second. Note that you need to return the quotient only. Output the string "NAN" if the second number is zero.

3.4 Output

Your program should write its results to the output file specified in the first line of `inFile`. *Each line* in the output file should contain exactly *one* number, corresponding to the result of the operation.
Make sure that there are no leading zeros in the output number. For example, *120* is a valid output but *0120* is not a valid output.

Input File

```
out1.txt
ADD 1 2 3
MULT 1 2 3 4
SUB 5 3
DIV 9 4
DIV 2 0
ADD 50000000000000000000000000000000000000 1500000000000000000000000000000
```

Output File - out1.txt

```
6
24
2
2
NAN
50000001500000000000000000000000000000
```