

# EPITA SPE promo 2017

## Practical Programming - Machine Test

Marwan Burelle\*

Friday, November 8, 2013

### Instructions:

**You must read the whole subject and all these instructions. Every explicit instructions in the subject are mandatory. Points lost for ignoring subject rules are not open to arguments, including compiling issues or usage of directory hierarchy.**

*In the  `sujet`  directory, you'll find a sub-directory called  `Skel` , you have to copy the **content** of this directory in your  `rendu`  directory.*

*Here is given as example, commands to perform the needed copy from  `sujet/Skel`  directory to  `rendu`  directory:*

```
> cd
> cd sujet
> cp Skel/* ~/rendu/
```

*In this directory you'll find: a **Makefile** offering targets to compile your code, some annex files, a file for each questions named  `questionXX.c`  or  `questionXX.ml` . Only question files can be modified, all other files will be replaced by the original one during the automatic correction, and thus all modifications will be lost.*

*The **Makefile** offers a target building a test program for each question. This test program will perform all the interaction part (input and output) and call your (or yours) function(s) with the correct expected parameters. In order to target the build of these test programs, you need to issue the command (for question number  `XX` ):  `make questionXX` .*

*During automatic correction, this **Makefile** will be used and thus the question  `XX`  will be evaluated only if  `make questionXX`  succeed. Of course, **the grade will depends on the correctness of your answer.***

*For each question, the number of points is indicated on the right of the question number (between parenthesis.) **These points are given for information only and can be changed later.***

*At the end of the document, you'll find the extra sections providing:*

- *Advices about test programs;*
- *List of all files that **must** be in your  `rendu`  directory.*

**There are 25 points and 12 questions in this test.**

---

\*marwan.burelle@lse.epita.fr

# OCaml: Bases

## Question 1

(1)

Write the following function(s):

```
val fact : int -> int
```

fact n compute n!. Your fonction must raise Neg\_param(n) exception if n is negative.

### Exemple 1.1:

```
un_shell> ./question01 0 5
```

Fixed values:

```
fact 0 = 1 (should be 1)
```

```
fact 1 = 1 (should be 1)
```

```
fact 5 = 120 (should be 120)
```

Random tests:

```
fact 12 = 479001600
```

```
fact 10 = 3628800
```

```
fact 14 = 87178291200
```

```
fact 16 = 20922789888000
```

```
fact 2 = 2
```

```
fact (-4) = Neg_param (-4)
```

## Question 2

(2)

Write the following function(s):

```
val int_sqrt : int -> int
```

int\_sqrt x compute the integer approximate of  $\sqrt{x}$ . The result must satisfy:

```
(int_sqrt x) * (int_sqrt x) <= x < (1 + int_sqrt x) * (1 + int_sqrt x)
```

For the square root computation of  $x$ , we'll use the following principle:

- let  $y$  be an upper approximate of the square root (we'll start with  $y = x$ )
- We know that  $x/y \leq \sqrt{x} \leq y$ . Thus, we can have a better approximate using:

$$y' = \frac{y + \frac{x}{y}}{2}$$

- We start over using  $y'$  instead of  $y$ .
- We stop when  $y \leq y'$ .

**Using sqrt function (and any translation to float) is strictly forbidden, such usage may be detected during automatic evaluation and will be punished by a 0.**

**Exemple 2.1:**

```
un_shell> ./question02 0 5
Power of 2:
  int_sqrt 4 = 2 (should be 2)
  int_sqrt 16 = 4 (should be 4)
  int_sqrt 64 = 8 (should be 8)
  int_sqrt 256 = 16 (should be 16)
  int_sqrt 1024 = 32 (should be 32)
Random tests:
  int_sqrt 985520 = 992
  int_sqrt 370270 = 608
  int_sqrt 710610 = 842
  int_sqrt 158504 = 398
  int_sqrt 335194 = 578
```

## OCaml: Lists

**Question 3**

(1)

Write the following function(s):

```
val digit : int -> int list
```

`digit n` builds the list of all decimal digit representing the number  $n$ . If  $n = 0$  the function returns an empty list.

For example, `digit 123` will return the list `[ 1; 2; 3 ]`.

**Exemple 3.1:**

```
un_shell> ./question03 0 5
digit 985520 = [ 9; 8; 5; 5; 2; 0; ]
digit 370270 = [ 3; 7; 0; 2; 7; 0; ]
digit 710610 = [ 7; 1; 0; 6; 1; 0; ]
digit 158504 = [ 1; 5; 8; 5; 0; 4; ]
digit 335194 = [ 3; 3; 5; 1; 9; 4; ]
digit 651659 = [ 6; 5; 1; 6; 5; 9; ]
```

**Question 4**

(1)

Write the following function(s):

```
val bin_digit : int -> int list
```

`bin_digit n` builds the list of all binary digit representing the number  $n$ . If  $n = 0$  the function returns an empty list.

For example, `bin_digit 42` will return the list `[ 1; 0; 1; 0; 1; 0 ]` (42 is 101010 in binary.)

**Exemple 4.1:**

```
un_shell> ./question04 0 5
bin_digit 432 = [ 1; 1; 0; 1; 1; 0; 0; 0; 0; ]
bin_digit 606 = [ 1; 0; 0; 1; 0; 1; 1; 1; 1; 0; ]
bin_digit 978 = [ 1; 1; 1; 1; 0; 1; 0; 0; 1; 0; ]
bin_digit 808 = [ 1; 1; 0; 0; 1; 0; 1; 0; 0; 0; ]
bin_digit 346 = [ 1; 0; 1; 0; 1; 1; 0; 1; 0; ]
bin_digit 395 = [ 1; 1; 0; 0; 0; 1; 0; 1; 1; ]
```

**Question 5**

(3)

Write the following function(s):

**val** `join : ('a * 'b) list -> ('b * 'c) list -> ('a * 'b * 'c) list`

`join l1 l2` builds the *join* between two lists: the result is a list containing elements of the form  $(a,b,c)$  and such that there exists in `l1` a pair  $(a,b)$  and in `l2` there exists a pair  $(b,c)$ . The order of the resulting list doesn't matter (the test program will sort it.)

In other words, the function is joining all the pairs with a common element. **Beware**, you must compare **all** the pairs of the first list with **all** the pairs of the second list.

**Exemple 5.1:**

```
un_shell> ./question05 0 10
l1 = [ (14,12); (03,24); (30,23); (23,06); (01,24);
       (17,10); (18,20); (11,26); (08,18); (30,16); ]
l2 = [ (03,04); (13,25); (28,00); (16,07); (31,06);
       (02,25); (01,14); (04,26); (09,26); (12,13); ]
join l1 l2 = [ (14,12,13); (30,16,07); ]
```

**Question 6**

(2)

Write the following function(s):

**val** `product : 'a list -> 'b list -> ('a * 'b) list`

`product l1 l2` builds the cartesian product of the two lists:  $\{(a,b) \mid \forall a \in l1 \text{ et } \forall b \in l2\}$ .

**Exemple 6.1:**

```

un_shell> ./question06 0 3
l1 = [ 0432; 0606; 0978; ]
l2 = [ 0346; 0395; 0808; ]
product l1 l2 =
  [ (0432,0346); (0432,0395); (0432,0808); (0606,0346);
    (0606,0395); (0606,0808); (0978,0346); (0978,0395);
    (0978,0808); ]

```

**Question 7**

(2)

Write the following function(s):

```
val pipe : 'a -> ('a -> 'a) list -> 'a
```

pipe x [f0; f1; ...; fn] returns (fn ( ... (f1 (f0 x))))

**Exemple 7.1:**

```

un_shell> ./question07 42 10
s = "rRGddY RsV"
pipe s [lowercase; capitalize] = "Rrgddy rsv"
pipe s [uppercase; uncapitalize] = "rRGDDY RSV"
pipe s [rot47] = "C#v55* #D'"
pipe s [rot47; rot47] = "rRGddY RsV"

```

**OCaml: Advanced Types****Question 8**

(2)

Write the following function(s):

```
val check : 'a mutable_list -> bool
```

check l verifies that the size embedded in list l (in field size) is coherent with the list real size (obtained using a traversal of the list.)

We'll use a *home made* type definition for mutable lists:

```

type 'a inner = {
  mutable content : 'a;
           next    : 'a inner option;
}

type 'a mutable_list = {
  mutable size  : int;
  mutable inner : 'a inner option;
}

```

The 'a option type is defined (in OCaml core library) as:

```
type 'a option =  
  None  
  | Some of 'a
```

**Exemple 8.1:**

```
un_shell> ./question08 0 5  
l = (size=5)[ 62; 36; 54; 90; 52; ]  
check l = OK  
l = (size=6)[ 62; 36; 54; 90; 52; ]  
check l = KO  
l = (size=1)[ ]  
check l = KO
```

**Question 9**

(2)

Write the following function(s):

```
val update : ('a -> 'a) -> 'a mutable_list -> unit
```

update f l replace all elements of the list l with their image through the function f. We'll use the same type as the previous question.

**Exemple 9.1:**

```
un_shell> ./question09 0 5  
l = (size=5)[ 02; 06; 04; 00; 02; ]  
f x = x*x  
update f l  
l = (size=5)[ 04; 36; 16; 00; 04; ]
```

**Question 10**

(2)

Write the following function(s):

```
val bin_search : 'a -> 'a array -> bool
```

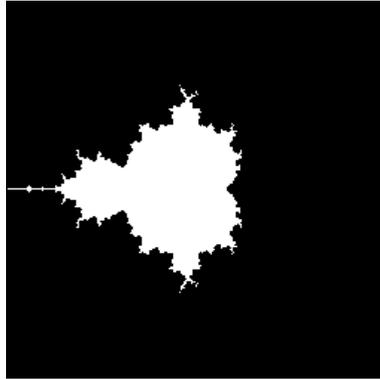
bin\_search x tab look for x in the sorted array tab.

**Remarque 1:**

*Since the array is sorted, we expect your function to be efficient (at least in advanced test.) I thus advice you to use algorithm such as dichotomy. Automatic evaluation will call your function on much larger arrays.*

**Exemple 10.1:**

```
un_shell> ./question10 0 10 90
tab = [| 03; 22; 36; 52; 54; 57; 62; 68; 90; 94; |]
bin_search 90 tab = found
```

Figure 1: Display for the 11<sup>th</sup> question**Question 11**

(3)

Write the following function(s):

```
val escape : Complex.t -> int -> bool
```

escape c bound returns true if the complex Mandelbrot sequence is still converging at rank  $n$ . The complex Mandelbrot sequence is defined as follow:

$$\begin{aligned} z_0 &= c && \text{with } c \text{ the function parameter} \\ z_n &= z_{n-1}^2 + z_0 \end{aligned}$$

We know that the sequence is converging while  $|z_n| \leq 2$  (the modulus of term  $z_n$  is bellow 2, concretely we should rather verify that  $|z_n|^2 \leq 4$  in order to avoid a square root.) Thus, we can approximate the convergence of the sequence, by computing the terms  $z_n$  to until we found  $z_{bound}$  or we found some escaping term.

We'll use the Complex module provided by OCaml:

```
type t = { re: float; im: float }
(** The type of complex numbers. [re] is the real part and
    [im] the imaginary part. *)
```

```
val zero: t
(** The complex number [0]. *)
```

```
val add: t -> t -> t
(** Addition *)
```

```

val mul: t -> t -> t
(** Multiplication *)

val norm2: t -> float
(** Norm squared: given [x + i.y], returns [x^2 + y^2]. *)

val norm: t -> float
(** Norm: given [x + i.y], returns [sqrt(x^2 + y^2)]. *)

```

### Exemple 11.1:

```

un_shell> ./question11 0 5
(-0.53125,0.75) : converge
(-1.375,1.28125) : diverge
(0.171875,-0.59375) : converge
(0.28125,1.8125) : diverge
(0.265625,0.15625) : converge

```

*We can also call the program with a third parameter (that can be anything), in that case, the tes program will switch to graphic mode: it will draw in black the point for wich the function returns false. For example the following command will display what you can found in figure 1:*

```

un_shell> ./question11 0 12 draw

```

## Question 12

(4)

Write the following function(s):

```

val normalize : peano -> peano
val add : peano -> peano -> peano

```

`normalize p` *normalize* a Peano integer: a normalized integer is either: 0 or composed only one constructor (successor or predecessor.)

`add p1 p2` returns the sum of the two Peano integer `p1` and `p2`. **Beware**, you must respect the recursive equations provided in the subject !

A Peano integer is described as either 0 or the successor of a Peano integer or the predecessor of a Peano integer. The corresponding OCaml type is:

```

type peano =
  | Zero
  | Succ of peano
  | Pred of peano

```

The sum of two Peano integer is recursively defined as:

$$\begin{aligned}0 + p &= p \\p + 0 &= p \\Succ(p) + Succ(p') &= p + Succ(Succ(p')) \\Pred(p) + Pred(p') &= p + Pred(Pred(p')) \\Succ(p) + Pred(p') &= p + p' \\Pred(p) + Succ(p') &= p + p'\end{aligned}$$

**Exemple 12.1:**

```
un_shell> ./question12 0 7
```

```
p =
```

```
  Pred ( Succ ( Succ ( Pred ( Pred ( Succ ( Pred ( Zero))))))
    )
  )
```

```
normalize p = Pred ( Zero)
```

```
p1 = Pred ( Succ ( Pred ( Zero)))
```

```
p2 = Pred ( Succ ( Pred ( Zero)))
```

```
p1 + p2 = Pred ( Pred ( Succ ( Pred ( Zero))))
```

## About The Test Session

Once the test is over, you must leave your session by closing the clock (that's the only way.) Note that when the test is over, your session will close directly.

When the session closed, you'll be prompted for your password (the one used to login.) This will end the test (your *rendu* directory will be archived and sent to the collecting server.) **You must not shutdown the computer before the completion of this final step, otherwise your work will be lost.**

You can send intermediary versions of your test by using the shell command `rendu`. It is strongly advised that you do so to prevent data lost before the end of the test.

Even after the end of the test (in the few minutes following the test, of course), you can restart your computer to eventually re-send your work (this may be required sometimes if something goes wrong during the final step.)

## About Questions Skel

For every question, a skeleton of code is provided. This code is the minimal requirement for the compilation of the file *w.r.t.* the test program. The content of the skeleton will also induce a failure at execution time and thus you must remove the the body of the function(s). In OCaml, be sure to remove (or comment) the `assert false` line of code: if it's still in the file, it will probably be executed anyway.

### Exemple 1:

*For example, if you're asked for the following function:*

```
val identity: 'a -> 'a
```

*identity x returns x.*

*You'll find the following skeleton:*

```
let identity _ =  
  (* FIX ME *)  
  assert false
```

*Your answer will look like:*

```
let identity x = x
```

## About test programs

When invoked with `make questionXX`, `make` will build a binary program named `questionXX`. This program can be used to test your answer to the question X in this subject. All binary wait for parameters et display a small help when run with `-help`.

**Exemple 2:**

*For example, the program for question 1 (this is an example and may not corresponds to the actual question 1) will display:*

Question 1:

```
./question01 graine taille  
-help  Display this list of options  
--help Display this list of options
```

The two parameters are thus: `graine` (seed) and `taille` (size). These parameters are present in most question: the seed is used to initialize the random number generator (for a given seed, the generator will produce the same sequence of number) and the size can be either the size of generated data (for list or strings ... ) or the number of tested ...

If you need more detail, read the `test_qXX.ml` files.

## Lists of Files

### Immutable Files

Makefile
question01.mli
question02.mli
question03.mli
question04.mli
question05.mli
question06.mli
question07.mli
question08.mli
question09.mli
question10.mli
question11.mli
question12.mli
test_frame.ml
test_q01.ml
test_q02.ml
test_q03.ml
test_q04.ml
test_q05.ml
test_q06.ml
test_q07.ml
test_q08.ml
test_q09.ml
test_q10.ml
test_q11.ml
test_q12.ml

### Answer Files

question01.ml
question02.ml
question03.ml
question04.ml
question05.ml
question06.ml
question07.ml
question08.ml
question09.ml
question10.ml
question11.ml
question12.ml