
MP 1 – Basic OCaml

CS 421 – Fall 2009

Revision 1.0

Assigned August 25, 2008

Due Sept 1, 2008, 11:59 PM

Extension 48 hours (penalty 20% of total points possible)

1 Change Log

1.0 Initial Release.

2 Objectives and Background

The purpose of this MP is to test the student's ability to

- start up and interact with OCaml;
- define a function;
- write code that conforms to the type specified (this includes understanding simple Ocaml types, including functional ones).

Another purpose of MPs is to provide a framework to study for the exam. Several of the questions on the exam will appear similar to the MP problems. By the time of the exam, your goal is to be able to solve any of the following problems with pen and paper in less than 2 minutes.

3 Problems

Note: In the problems below, you do not have to begin your definitions in a manner identical to the sample code, which is present solely as a guide. However, you have to use the indicated names for your functions, and the functions will be expected to conform to any type information supplied and to yield the same results as any sample executions given.

1. (1 pt) Declare a variable `b` with the value `false`. It should have type `bool`.
2. (1 pt) Declare a variable `r` with a value of `12.4`. It should have type `float`.
3. (2 pts) Write a function `sub_r` that subtracts the value of `r` from its argument.

```
# let sub_r z = ...
val sub_r : float -> float = <fun>
# sub_r 14.1;;
- : float = 1.7
```

4. (2 pts) Write a function `double_pair_with_b` that returns two times its integer input paired with the boolean `b` of Problem 1.

```
# let double_pair_with_b x = ...
val double_pair_with_b : int -> int * bool = <fun>
# double_pair_with_b 15;;
- : int * bool = (30, false)
```

5. (4 pts) Write a function `pos_min` that takes two integer arguments and returns the smaller of the two integers, if they are both non-negative, and returns 0 otherwise. Pay careful attention to the type of this function.

```
# let pos_min n m = ...
val pos_min : int -> int -> int = <fun>
# pos_min 9 13;;
- : int = 9
```

6. (3 pts) Write a function `greetings` that takes a string, which is assumed to be a person's name, and prints out a greeting as follows: If the name is "Elsa", it prints out the string

```
"Hey Elsa man, waz happn'n?"
```

and for any other string, it first prints out "Greetings, " followed by the given name, followed by ", warmest welcome to CS421.", followed by a "newline".

```
# let greetings name = ...
val greetings : string -> unit = <fun>
# greetings "Mica";;
Greetings, Mica, warmest welcome to CS421.
- : unit = ()
```

7. (5 pts) Write a function `do_each` that takes a pair of functions as a first argument and a pair of values as a second argument, then returns the pair formed by applying the first function to the first argument and the second function to the second argument.

```
# let do_each (f,g) (x,y) = ...
val do_each : ('a -> 'b) * ('c -> 'd) -> 'a * 'c -> 'b * 'd = <fun>
# do_each ((fun n -> n - 4), pos_min 3) (5, 7);;
- : int * int = (1, 3)
```

The correct answer will have the polymorphic type given above, but since we have not discussed polymorphism yet, it will only be tested at the type

```
val do_each : (int -> int) * (int -> int) -> int * int -> int * int = <fun>
```

Warning: Any totally correct answer will have the polymorphic type given in the sample output, unless you simply placed a type restriction on one or more subexpressions forcing it to have a less general type than specified.