
MP 1 – Basic OCaml

CS 421 – Fall 2008

Revision 1.0

Assigned August 26, 2008

Due Sept 2, 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 for guiding you better. However, you have to use the indicated name for your functions, and the functions will have 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 `true`. It should have type `bool`.
2. (1 pt) Declare a variable `r` with a value of `17.9`. It should have type of `float`.
3. (2 pts) Write a function `add_r` that adds the value of `r` to its argument.

```
# let add_r z = ...
val add_r : float -> float = <fun>
# add_r 2.3;;
- : float = 20.2
```

4. (2 pts) Write a function `square_pair_with_b` that returns the square of integer input paired with the boolean `b` of Problem 1.

```
# let square_pair_with_b x = ...
val square_pair_with_b : int -> int * bool = <fun>
# square_pair_with_b 17;;
- : int * bool = (289, true)
```

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

```
# let pos_max n m = ...
val pos_max : int -> int -> int = <fun>
# pos_max 7 12;;
- : int = 12
```

6. (3 pts) Write a function `aloha` 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 "Aloha, " followed by the given name, followed by ", Warmest welcome to" followed by a "newline".

```
# let aloha name = ...
val aloha : string -> unit = <fun>
# aloha "Mica";;
Aloha, Mica, Warmest welcome to CS421.
- : unit = ()
```

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

```
# let do_pair (f,g) x = ...
val do_pair : ('a -> 'b) * ('a -> 'c) -> 'a -> 'b * 'c = <fun>
# do_pair ((fun n -> n + 1), pos_max 8) 2;;
- : int * int = (3, 8)
```

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_pair : (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.