
MP 0 – Basic OCaml (ungraded)

CS 421 – Spring 2010

Revision 1.0

Assigned January 19, 2010

Due January 20, 2010, 10:00 PM

Extension 48 hours (penalty would be 20% of total points possible if this were a graded assignment)

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);
- use handin to turn in your assignment electronically

For help with this MP, please see the handout on basic OCaml linked from the MP0 page.

The assignment is ungraded and therefore optional, i.e. you do not have to turn it in. However we strongly recommend that you do, in order to get practice both with OCaml and with the handin system. If you do not turn it in by the deadline, you have up to two days to try a “late” submission (which is meaningless for this assignment since we will not be grading it).

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.

If you are stuck on this assignment, ask for help from TAs right away.

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 `x` with the value 6. It should have type `int`.
2. (1 pt) Declare a variable `y` with a value of `98.1`. It should have type of `float`.

3. (2 pts) Write a function `minus_x` that subtracts the value of `x` from its argument.

```
# let minus_x z = ...
val minus_x : int -> int = <fun>
# minus_x 22;;
- : int = 16
```

4. (2 pts) Write a function `square_minus_y` that computes the result of subtracting `y` from the square of its input.

```
# let square_minus_y z = ...
val square_minus_y : float -> float = <fun>
# square_minus_y 16.23;;
- : float = 165.3129000000000042
```

5. (3 pts) Write a function `largest` that takes two integer arguments and returns the integer that is the largest if the largest is bigger than 0, and returns 0 otherwise.

```
# let largest n m = ...
val largest : int -> int -> int = <fun>
# largest 5 3;;
- : int = 5
```

6. (4 pts) Write a function `salutations` that takes a string, which is assumed to be a person's name, and prints out a greeting as follows: If the name is "Sam", it prints out the string

```
"Hi Sam, what's up?"
```

and for any other string, it first prints out "Dear " followed by the given name, followed by ", Welcome to CS421.". There should be no "newlines" printed in your results.

Note: Please use the function `print_string`, and only the function `print_string`, to print out your result. If you use a function other than `print_string`, we cannot test your output and the auto-grader will not be able to give you credit!

```
# let salutations name = ...
val salutations : string -> unit = <fun>
# salutations "Robert";;
Dear Robert, Welcome to CS421.- : unit = ()
```

7. (4 pts) Write a function `do4` that takes first a function `f` then takes an `n`. The argument `f` takes one argument and returns a result of the same type as that argument. The argument `n` is of the same type as the input to `f`. The function `do4` returns the result of applying `f` to `n` four times, *i.e.*, the result of applying `f` to the result of applying `f` to the result of applying `f` to `n`.

```
# let do4 f n = ...
val do4 : ('a -> 'a) -> 'a -> 'a = <fun>
# do4 (fun n -> n + 1) 2;;
- : int = 6
```

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 do4 : (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 some expression forcing it to have a less general type than specified.