
MP 1 – Basic OCaml and Recursion

CS 421 – Spring 2007

Revision 1.1

Assigned January 17, 2007

Due January 24, 2007, 11:59 PM

Extension 48 hours (20% penalty)

1 Change Log

1.0 Initial Release.

1.1 Removed the answers to the problems accidentally included.

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 pattern matching.

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. (2 pts) Write a function `squares` that computes the sum of the squares of two real (float) numbers.

```
# let squares x y = ...
val squares : float -> float -> float = <fun>
# squares 13.5 16.23;;
- : float = 445.6629000000000036
```

2. (4 pts) Write a function `biggest` that takes a pair of integers and returns the integer that is the largest, if the largest is bigger than 0, and returns 0 otherwise.

```
# let biggest (x,y) = ...
val biggest : int * int -> int = <fun>
# biggest (5, -3);;
- : int = 5
```

3. (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

```
"Hi theya, Elsa, whatcha doin?"
```

and for any other string, it first prints out "Hello, " followed by the given name, followed by " I hope you are well." There should be no "newlines" printed in your results.

```
# let greetings name = ...
val greetings : string -> unit = <fun>
# greetings "Robert";;
Hello, Robert I hope you are well.- : unit = ()
```

4. (3 pts) Write a function `rev3_print` that takes the strings and prints them in reverse order.

```
# let rev3_print (s, t, u) = ...
val rev3_print : string * string * string -> unit = <fun>
# rev3_print ("Bob", "likes", "Mike");;
MikelikesBob- : unit = ()
```

5. (5 pts) Write a function `do3` that takes first a function `f` that needs two integers as input, and returns an integer, then takes an integer `n`. It returns the result of applying `f` to `n` and the result of applying `f` to `n` and `f` applied to 0 and 0.

```
# let do3 f n = ...
val do3 : (int -> int -> int) -> int -> int = <fun>
# do3 (fun n -> (fun m -> (n + 5) * (m + 1))) 2;;
- : int = 301
```

6. (3 pts) Write a function `circle_info` that when applied to a radius given as a float that is greater than 0.0, returns the pair of the circumference and area of a circle of that radius. If the given radius is less than 0, it should return a pair of 0 and 0. You should use 3.14159 as the value of π . The circumference of a circle of radius r is $2\pi r$ and the area is πr^2 .

```
# let circle_info r = ...
val circle_info : float -> float * float = <fun>
# circle_info 3.333;;
- : float * float = (20.94183894, 34.89957459351)
```

7. (4 pts) Write a function `eq_rat` that takes two pairs of integers, and tells if the pairs are equal when viewed as rational numbers represented by a numerator and a denominator. If either denominator is 0, the result should be `false`. Recall that $\frac{m}{n} = \frac{p}{q}$ if and only if $mq = np$ (and $n \neq 0 \neq q$).

```

# let eq_rat (m,n) (p,q) = ...
val eq_rat : int * int -> int * int -> bool = <fun>
# eq_rat (4, 10) (6, 15);;
- : bool = true

```

8. (4 pts) Write a function `int_list_head` that returns the head of a list of integers, if it has one, and otherwise returns 0.

```

# let int_list_head l = ...
val int_list_head : int list -> int = <fun>
# int_list_head [5; 3; 7];;
- : int = 5

```

9. (4 pts) Write a function `third_hi` that returns `true` if it is applied to a list that has at least three elements and the third element is "hi", and returns `false` otherwise.

```

# let third_hi l = ...
val third_hi : string list -> bool = <fun>
# third_hi ["I"; "said"; "hi"; "there"];;
- : bool = true

```

Extra Credit Problem

10. (3 pts) Write a function `sort_triple` that takes a triple of pairs of integers, and returns the triple sorted from largest to smallest, where the pairs are order by reverse alphabetic order. Reverse alphabetic ordering gives $(m,n) < (p,q)$ if $n < q$ or $n = q$ and $m < p$. It is possible to write correct code for this problem that has a polymorphic type, more general than the type given here.

```

# let sort_triple (a, b, c) = ...
val sort_triple :
  (int * int) * (int * int) * (int * int) ->
  (int * int) * (int * int) * (int * int) = <fun>
# sort_triple ((2,3), (4,1), (3,2));;
- : (int * int) * (int * int) * (int * int) = ((2, 3), (3, 2), (4, 1))

```