MP 1 – Basic OCaml

CS 421 – Spring 2014
Revision 1.0

Assigned January 23, 2014
Due February 2, 2014, 11:59 PM

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 in general 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 What to turn in

You should put code answering each of the problems below in the file called `mp1.ml`. The file already contains a stub function for each problem. Please read the *Guide for Doing MPs* in:
`http://courses.engr.illinois.edu/cs421/mps/index.html`

4 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 values, and they will have to conform to any type information supplied, and have to yield the same results as any sample executions given, as well as satisfying the specification given in English.

1. (1 pt) Declare a variable `title` with the value `"MP 1 -- Basic OCaml"`. It should have type `string`. It shoud not contain a "newline".

2. (1 pt) Declare a variable `e` with a value of `2.71828`. It should have the type of `float`.

3. (2 pts) Write a function `firstFun` that returns the result of mulitplying a given integer by 2 and adding 5.

```
# let firstFun n = ... ;;
val firstFun : int -> int = <fun>
# firstFun 12;;
- : int = 29
```

4. (2 pts) Write a function `divide_e_by` that returns the result of dividing the value you gave in Problem 2 by the given `float`. You will not be tested on the value `0.0`.

```
# let divide_e_by x = ...;;
val divide_e_by : float -> float = <fun>
# divide_e_by e;;
- : float = 1.
```

(Your value may vary slightly from that printed here if you use a machine of different precision.)

5. (3 pts) Write a function `diff_square_9` that takes an integer and if the integer is between 3 and -3, squares it and substracts 9, and otherwise substracts its square from 9.

```
# let diff_square_9 m = ...;;
val diff_square_9 : int -> int = <fun>
# diff_square_9 5;;
- : int = -16
```

6. (3 pts) Write a function `dist_double` that, when given a string and an integer, first prints the string, a comma and a space, then "`I guess it's double or nothing!`" followed by a newline, and then returns double the integer it was given.

```
# let dist_double s n = ...;;
val dist_double : string -> int -> int = <fun>
# dist_double "Well, Sam" 8;;
Well, Sam, I guess it's double or nothing!
- : int = 16
```

7. (3 pts) Write a function `swizzle` that takes a 4-tuple (quadruple) and returns the 4-tuple with the first component moved to the third, the third moved to the second, the second moved to the fourth and the fourth moved to the first.

```
# let swizzle (w,x,y,z) = ... ;;
val swizzle : 'a * 'b * 'c * 'd -> 'd * 'c * 'a * 'b = <fun>
# swizzle (1,2,3,4);;
- : int * int * int * int = (4, 3, 1, 2)
```

8. (4 pts) Write a function `left_right_compose` that takes a first function f, then a second function g, and returns the result of pre- and post-composing g with f to get $f \circ g \circ f$.

```
# let left_right_compose f g = ... ;;
val left_right_compose : ('a -> 'b) -> ('b -> 'a) -> 'a -> 'b = <fun>
# left_right_compose firstFun (dist_double "Oh my") 17;;
Oh my, I guess it's double or nothing!
- : int = 161
```