MP 1 – Basic OCaml

CS 421 – Fall 2012
Revision 1.0

Assigned August 28, 2012
Due September 4, 2012, 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 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 `handin`

You should put code answering each of the problems below in a file called `mp1.ml`. A good way to start is to copy `mp1-skeleton.ml` to `mp1.ml` and edit the copy. If you choose not to start this way, please be sure to place

```
open Mp1common
```

at the top of you file. Please read the *Guide for Doing MPs* in

`http://courses.engr.illinois.edu/cs421/mps/index.html`

Also, please read the section *How do I `handin` my MPs and HWs?* in

`http://courses.engr.illinois.edu/cs421/faq.html#how_to_handin`

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 `greetings` with the value `"Hi there."`. It should have type `string`. It shoud not contain a "newline".

2. (1 pt) Declare a variable `pi` with a value of `3.1415926`. It should have the type of `float`.

3. (2 pts) Write a function `square` that returns the result of mulitplying a given integer by itself.

```
# let square n = ... ;;
val square : int -> int = <fun>
# square 7;;
- : int = 49
```

4. (2 pts) Write a function `plus_pi_times_3 y` that returns the result of multiplying by the float `3.0` the result of adding the value of `pi` from Problem 2 to the float-valued input.

```
# let plus_pi_times_3 y = ... ;;
val plus_pi_times_3 : float -> float = <fun>
# plus_pi_times_3 23.17;;
- : float = 78.9347778
```

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

5. (3 pts) Write a function `salute` 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

```
Halt! Who goes there!
```

followed by a "newline" at the end (that is, there should be only one "newline" produced). For any other string, it first prints out `"Hail, "`, followed by the given name, followed by `". We warmly welcome you!"`, followed by a "newline". Do not print the quotations; they were included to help make blank spaces visible. All spaces in the sample text above and below are one space long.

```
# let salute name = ... ;;
val salute : string -> unit = <fun>
salute "Malisa";;
Hail, Malisa. We warmly welcome you!
- : unit = ()
```

6. (4 pts) Write a function `has_smallest_square` that, when given one integer and then another, returns the one that has the smallest squared value. If they have the same squared value, but are not the same value, it should return the smallest value given.

```
# let has_smallest_square m n = ... ;;
val has_smallest_square : int -> int -> int = <fun>
# has_smallest_square 4 6;;
- : int = 4
```

7. (3 pts) Write a function `pivot` that takes a pair and returns the triple with the same first and second component as the pair, and repeats the first component as the third component.

```
# let pivot (x,y) = ... ;;
val pivot : 'a * 'b -> 'a * 'b * 'a = <fun>
# pivot (3, "hi");;
- : int * string * int = (3, "hi", 3)
```

8. (5 pts) Write a function `app_pair` that takes a pair of values as a first argument and then takes a function as a second argument and returns the pair resulting from applying the function to each of the first and the second component of the input pair.

```
# let app_pair (x,y) f = ... ;;
val app_pair : 'a * 'a -> ('a -> 'b) -> 'b * 'b = <fun>
# app_pair (7, 9) square;;
- : int * int = (49, 81)
```