MP 1 – Basic OCaml

CS 421 – Summer 2013
Revision 1.0

Assigned May 30, 2012
Due June 5, 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 your file. Please read the *Guide for Doing MPs* in

`http://courses.engr.illinois.edu/cs421/su2013/mps/index.html`

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

`http://courses.engr.illinois.edu/cs421/su2013/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 as a guide. However, you have to use the indicated name for your functions, and the functions 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 `e` with the value `2.71828`. It should have type `float`.

2. (1 pt) Declare a variable `welcome` with a value of `"Hello and welcome!"`. It should have the type of `string`.

3. (2 pts) Write a function `ten_less` that returns the result of subtracting 10 from a given integer.

```
# let ten_less n = ... ;;
val ten_list : int -> int = <fun>
# ten_less 17;;
- : int = 7
```

4. (2 pts) Write a function `e_to_the_square y` that returns the result of squaring its (float-valued) input, then raising the value of `e` from Problem 1 to the power of the result. You can use the `**` operator for exponentiation (recall that the `^` operator is string concatenation).

```
# let e_to_the_square y = ... ;;
val e_to_the_square : float -> float = <fun>
# e_to_the_square 1.4;;
- : float = 7.0993177054147738
```

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

5. (3 pts) Write a function `recognize` 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 `"Ah, I know you."`, followed by a "newline" at the end (there should be only one "newline" produced, and it should not print the quotation marks). For any other string, it first prints out `"Your name is "`, followed by the given name, followed by `", you say? Welcome to CS 421."`, followed by a "newline".

```
# let recognize name = ... ;
val recognize : string -> unit = <fun>
# recognize "John";;
Your name is John, you say? Welcome to CS 421.
- : unit = ()
```

6. (4 pts) Write a function `has_smallest_floor` that, when given one float and then another, returns the one that has the smallest floor (you can use the function `floor` to find the floor of a number). If they have the same floor, it should return the smaller value given.

```
# let has_smallest_floor m n = ... ;;
val has_smallest_floor : float -> float -> float = <fun>
# has_smallest_floor 4.3 6.2;;
- : float = 4.3
```

7. (3 pts) Write a function `first_two` that takes a triple (3-tuple) and returns the pair (2-tuple) of the first two elements of the triple.

```
# let first_two (x,y,z) = ... ;;
val first_two : 'a * 'b * 'c -> 'a * 'b = <fun>
# first_two (3, 5, 9);;
- : int * int = (3, 5)
```

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

```
# let app_triple f (x,y,z) = ... ;;
val app_triple : ('a -> 'b) -> 'a * 'a * 'a -> 'b * 'b * 'b = <fun>
# app_triple ten_less (10, 11, 12);;
- : int * int * int = (0, 1, 2)
```