MP 1 – Basic OCaml

CS 421 – su 2011

Assigned June 06, 2011
Due June 12, 2011, 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 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. (1 pt) Declare a variable `a` with the value `17`. It should have type `int`.

2. (1 pt) Declare a variable `s` with a value of `"Hi there"`. It should have the type of `string`.

3. (2 pts) Write a function `add_a` that adds the value of `a` from Problem 1 to its argument.

```
# let add_a n = ... ;;
val add_a : int -> int = <fun>
# add_a 13;;
- : int = 30
```

4. (2 pts) Write a function `s_paired_with_a_times` that returns the pair of the string `s` from Problem 2 paired with the result of multiplying the value `a` from Problem 1 by the integer input.

```
# let s_paired_with_a_times b = ... ;;
val s_paired_with_a_times : int -> string * int = <fun>
# s_paired_with_a_times 12;;
- : string * int = ("Hi there", 204)
```

5. (3 pts) Write a function `abs_diff` that takes two arguments of type `float` and returns the absolute value of the difference of the two. Pay careful attention to the type of this problem.

```
# let abs_diff x y = ... ;;
val abs_diff : float -> float -> float = <fun>
# abs_diff 15.0 11.5;;
- : float = 3.5
```

6. (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

```
"Hey Elsa, cool man!"
```

(no "newline" at the end), and for any other string, it first prints out `"Hello, "`, followed by the given name, followed by
`". I hope you enjoy CS421."` , followed by a "newline".

```
# let greetings name = ... ;;
val greetings : string -> unit = <fun>
# greetings "Angela";;
Hello, Angela. I hope you enjoy CS421.
- : unit = ()
```

7. (3 pts) Write a function `sign` that, when given an integer, returns `1` if the integer is positive, `0` if the integer is zero and `-1` if the integer is negative.

```
# let sign n = ... ;;
val sign : int -> int = <fun>
# sign 4;;
- : int = 1
```

8. (5 pts) Write a function `pre_post_compose` that takes a pair of functions as a first argument and then takes a second argument and returns the value resulting from applying the first function to the second argument, then applying the second function to the result of the first application, and lastly applying the first function to the result of the second application.

```
# let pre_post_compose (f,g) x = ... ;;
val pre_post_compose : ('a -> 'b) * ('b -> 'a) -> 'a -> 'b = <fun>
# pre_post_compose (sign, add_a) 10;;
- : int = 1
```

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

```
pre_post_compose : (int -> int) * (int -> int) -> int -> int
```

Warning: Any totally correct answer will have the polymorphic type given in the sample output, unless you simply placed a type restriction on one or more subexpressions forcing it to have a less general type than specified.