MP1 – Basic OCaml

CS 421 – Summer 2009
Revision 1.3

Assigned June 1, 2009
Due June 3, 2009, 1:00 PM
Extension 48 hours (penalty 20% of total points possible)

1 Change Log

1.3 The example for Problem 5 listed an incorrect answer. This has now been fixed.
1.2 Due to a careless instructor, the example for Problem 5 listed an incorrect answer. This has now been fixed.
1.1 Due to scheduled Compass downtime, submission deadline changed to 1:00PM.
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 Collaboration

Collaboration is NOT allowed in this assignment.
This MP is critical for a student to be familiar with our handin system, and programming in OCaml. If you are stuck on this assignment, ask for help from the course staff right away.

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 the functions will have to conform to any type information supplied, and to yield the same results as any sample executions given.

1. (1pt) Declare a variable `x` with value `1304`. It should have type `int`.

2. (1pt) Declare a variable `y` with value `4.21`. It should have type `float`.

3. (2pts) Write a function `square_minus_x` that subtracts the value of `x` from the square of its argument.

```
# let square_minus_x z = ...
val square_minus_x : int -> int = <fun>
# square_minus_x 10;;
- : int = -1204
```

4. (2pts) Write a function `square_plus_y` that adds the value of `y` to the square of its argument.

```
# let square_plus_y z = ...
val square_plus_y : float -> float = <fun>
# square_plus_y 10.0;;
- : float = 104.21
```

5. (3pts) Write a function `abs_largest` that takes two integer arguments and returns the absolute value of the largest argument (*not* the largest absolute value!).

```
# let abs_largest n m = ...
val abs_largest : int -> int -> int = <fun>
# abs_largest 3 (-5);;
- : int = 3
```

6. (4 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 "`Kirill`", it prints out the string

```
"Greetings, Kirill."
```

and for any other string, it first prints out "`Hi `" followed by the given name, followed by "`, welcome to CS421!`". There should be no "newlines" printed in your results.

```
# let greetings name = ...
val greetings : string -> unit = <fun>
# greetings "Alice";;
Hi Alice, welcome to CS421!- : unit = ()
```

7. (4pts) Write a function `triple_the_fun` that takes first a function `f` then takes an `n`. The argument `f` takes one argument and returns a result of the same type as that argument. The argument `n` is of the same type as the input to `f`. The function `triple_the_fun` returns the result of applying `f` to `n` three times, *i.e.*, the result of applying `f` to the result of applying `f` to the result of applying `f` to `n`.

```
# let triple_the_fun f n = ...
val triple_the_fun : ('a -> 'a) -> 'a -> 'a = <fun>
# triple_the_fun (fun n -> n * 2) 2;;
- : int = 16
```

The correct answer will have teh polymorphic type given above, but since we have not discussed polymorphism yet, it will only be tested at the type

```
val triple_the_fun : (int -> int) -> int -> int = <fun>
```

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