MP 0 – Basic OCaml

CS 421 – Spring 2013
Revision 1.0

Assigned January 15, 2013
Due January 17, 2013, 9:30am
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 get you started using OCaml, which we will both study and use extensively in this class. OCaml is an example of a "functional language." In this MP, you will access the EWS system, download the package of code needed to do this MP, write solutions to simple problems, test them, and submit them using the handin system. (You may also download OCaml onto your own computers, to be less reliant on the EWS machines, although you will always need to use the EWS machines to run handin.)

This MP is ungraded, but it is strongly recommended. Problems are given point values below just to make the assignment more "realistic." We think the assignment should take about an hour.

The assignment asks you to do only simple things, but you may still find the information in the lecture slides to be inadequate. These resources are provided on the course website:

- The document "Basic guide to OCaml" is linked from the page where you got this MP. It contains everything you need to know about OCaml for this MP.
- The MPs page on the website contains a section (also linked from this page) entitled "Guide for Doing MPs." We wish this were less complicated, but it's the best we can do; this complexity is one reason we are assigning an MP0.
- The "Resources" and "faq" tabs on the website explain access to the EWS machines; hopefully, this is not new to you.
- Also on the Resources page are links to OCaml information; in particular, there is a link to www.ocaml.org, from which you can download OCaml.

3 Collaboration

Collaboration is NOT allowed in this assignment.

Each student needs to be familiar with our handin system, and know how to run a program in OCaml. If you are stuck on this assignment, ask for help from TAs right away.

4 Problems

In the problems below, we have in most cases given you a "skeleton" of the solution, of the form

```
# let funname arg = ... ;;
```

The pound sign is the OCaml prompt. The ellipsis is the part you need to fill in. This line is followed by a line representing OCaml's response to the input, and this is followed by a call to the function and, again, OCaml's response. In other words, if you fill in the ellipsis correctly, it should respond with the second line; then if you enter the third line, it should respond exactly as in the fourth (and sometimes fifth) line. (Questions 1 and 2 are exceptions, since they do not involve defining functions.)

1. (1 pt) Declare a variable `a` with the value `17`. It should have type `int`.

2. (1 pt) Declare a variable `s` with the value `"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. (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 `"Sam"`, it prints out the string

   `"Hi Sam!"`

   with no newline at the end. For any other argument, it 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 = ()
```

5. (3 pts) Write a function `greetstring` that is similar to the previous problem, but instead of *printing* the specified string, it *returns* the string as its result.

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

6. (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
```