HW 3 – Order of Evaluation

CS 421 – Fall 2014
Revision 1.0

Assigned Tuesday, September 9, 2014
Due Tuesday, September 16, 2014, 23:59pm
Extension 48 hours (20% penalty)

1 Change Log

1.0 Initial Release.

2 Turn-In Procedure

Unlike previous two homework assignments, homework 3 requires you to write OCaml code, so before submitting please make sure that your solution successfully compiles.

You should put code answering each of the problems below in a file called `hw3.ml`

The command to commit this file is:

```
svn commit -m "Turning hw3." hw3.ml
```

3 Objectives and Background

The purpose of this HW is to test your understanding of:

- Order of evaluation in OCaml

Another purpose of HWs is to provide you with experience answering non-programming written questions of the kind you may experience on the midterms and final.

4 Problems

Note: In OCaml, in the application of an expression of function type to an argument, the argument is evaluated to a value first, then the expression of function type is evaluated to a functional value. If the functional value is a closure (as opposed to a primitive operation, or a partial application of a primitive operation), then the resulting application of the closure to a value is done as described in class.

1. (4 pts) Write a declaration of a name `verbose_inc` that takes one integer argument, increments it by one, and meets the following two requirements:

   1. The result of the declaration prints:

      ```
      declaring f
      val verbose_inc : int -> int = <fun>
      ```

   2. Evaluation of the function `verbose_inc` on an integer argument produces:

      ```
      evaluating:
      - : int ...
      ```

where `...` will be replaced with the provided integer argument incremented by one.

For example:

```
# verbose_inc 2;;
evaluating:
- : int = 3
```

2. (5 pts) Write a function `f` that takes another function (`g`) as an argument. Function `g` evaluates to an integer when applied to unit, `()`. The main computation `f` is to extract the value from its argument by applying it to unit, and then to return the result of adding 2 to that value. However, this function is also to have side-effects consistent with the behavior as shown in the examples below.

   Example 1:

   ```
   # f (fun () -> 1);;
   ab- : int = 3
   ```

   Example 2:

   ```
   # f (fun () -> print_string "x"; 5);;
   axb- : int = 7
   ```

3. (6 pts)

   a. (3pts) Modify the order of evaluation imposed by the function `f` above so that Example 1 prints the same output, while Example 2 prints:

   ```
   # f (fun () -> print_string "x"; 5);;
   xab- : int = 7
   ```

   b. (3 pts) Modify the order of evaluation imposed by the function `f` above so that Example 1 prints the same output, while Example 2 prints:

   ```
   # f (fun () -> print_string "x"; 5);;
   abx- : int = 7
   ```