HW 3 – Order of Evaluation
CS 421 – Fall 2012
Revision 1.0

Assigned Tuesday, September 18, 2012
Due Tuesday, September 25, 2012, 11:59pm
Extension 48 hours (20% penalty)

1 Change Log

1.0 Initial Release.

2 Objectives and Background

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

• Order of evaluation in OCaml

3 What to handin

Answer each problem below, save your work as a PDF (either scanned if handwritten or converted from a program), and hand in the PDF.

4 Problems

1. (20 pts) Below is a fragment of Ocaml code. Describe everything that is displayed on the screen (its observable behavior) after this code has been cut-and-pasted into an interactive Ocaml session, and explain why this behavior is observed. This should include both the type information that the compiler gives back for each declaration, and any other things printed to the screen. For the type information, no explanation is required (but it should be correct). Give explanations for all other things printed, and the order in which they occur.

   let f g x =
   (let r =
    if ((print_string "a"; x > 5) && (g(); x > 10))
    then
     (print_string "b"; x - 7)
    else
     let z = (print_string "c"; 15) in (print_string "d"; z)
    in (g(); r));;
   let u = (f (fun () -> print_string "e\n") (f (fun () -> print_string "f\n") 3));;

5 Solution

1. The expression (f (fun () -> print_string "e\n") (f (fun () -> print_string "f\n") 3)) is evaluated and the value is bound to u. The following is output by ocaml:
2. This involves invoking function \( f \) with two arguments: `(fun () -> print_string "e\n")` and `(f (fun () -> print_string "f\n") 3)`.

3. The right-most argument gets evaluated first. Thus, function \( f \) is invoked again with two arguments: `(fun () -> print_string "f\n")` and `3`.

4. The invocation of function \( f \) mentioned in step 3 results in the following:

(a) the argument `(fun () -> print_string "f\n")` binds to the formal parameter \( g \) and the argument `3` binds to the formal parameter \( x \).

(b) in the body of \( f \), the value of the if-then-else expression is evaluated and bound to \( r \) as follows:

i. the expression `(print_string "a"; x > 5) && (g(); x > 10)` in the if condition is first evaluated. The two arguments of the && operator in this expression are `(print_string "a"; x > 5)` and `(g(); x > 10)`. Since && uses short-circuit evaluation, the left argument is evaluated first, which causes the string "a" to be output, but evaluates to false since \( 3 < 5 \). Since the left argument of the && operator is found to be false, owing to short-circuit evaluation, the right argument is not evaluated anymore and the if condition is evaluated to be false.

ii. the expression under the else part is then evaluated. First, the expression being bound to \( z \) in the let expression is evaluated. This causes the string "c" to be output and the value `15` to be bound to \( z \). Next, the main body of the let expression is evaluated, and this causes the string "d" to be output and the value of the else part evaluates to the value of the expression \( z \), which is `15`.

iii. the value of the expression under the else part is bound to \( r \). Thus, \( r \) is bound to `15`.

(c) Finally, the sequence of expressions \( g() \) and \( r \) is evaluated. Since \( g \) was bound to `(fun () -> print_string "f\n")`, it causes the string "f\n" to be output, and since \( r \) was bound to `15`, the body of the function \( f \) evaluates to `15`.

5. Then the invocation of the function \( f \) mentioned in step 2 is completed. The formal parameters of \( f \) get bound to the arguments `(fun () -> print_string "e\n")` and `15` (since the operations in step 4 evaluated to `15`). In the body of \( f \), the value of the if-then-else expression is evaluated and bound to \( r \) as follows:

(a) the expression `(print_string "a"; x > 5) && (g(); x > 10)` in the if condition is first evaluated. The two arguments of the && operator in this expression are `(print_string "a"; x > 5)` and `(g(); x > 10)`. Since && use short-circuit evaluation, the left argument is evaluated first, which causes the string "a" to be output and evaluates to true since `15 > 5`. Then the right argument of && is evaluated, which results in invoking the anonymous function `(fun () -> print_string "e\n")` and thus the string "e\n" is output; and the sequence evaluates to true since `15 > 10`. Therefore, the if condition evaluates to true.

(b) The expression under the then part is then evaluated. This results the string "b" being output and evaluates to the value `15 - 7 = 8`. Thus \( r \) is bound to `8`.

(c) Finally, the sequence of expressions \( g() \) and \( r \) is evaluated. Since \( g \) was bound to `(fun () -> print_string "e\n")`, it causes the string "e\n" to be output, and since \( r \) was bound to `8`, the body of the function \( f \) evaluates to `8`.

6. The variable \( u \) thus gets bound to the value `8`, which is output by the top-level OCaml loop.