22c:111 (CS:3820) Programming Language Concepts
Fall 2014
Micro-Assigment 2
Due: Monday, Oct 6 at 1.30pm

Solutions must be uploaded to the respective Dropbox section on ICON before the due date and time. Typed solutions can be submitted as one file in either of the following formats: PDF (strongly recommended), plain text, OpenOffice/LibreOffice, RTF or MS Word. Handwritten solutions are perfectly fine, but must be scanned and submitted as a *single PDF file* (no JPEGs). *Please make sure your handwriting is clearly legible on the scanned file*. Whatever the format, every page should clearly contain the name of the author and a page number.

1 Types

1. Are the following expressions well-typed? If so, what are their types and why?

   (a) `'a' + 'b'` (2 points)

   (b) `'a' > 'b'` (2 points)

   (c) `let f x y = if x > 0 then x = y else x = (-y)` (4 points)

2. Why do we not just use, say, the integer `0` to represent **`false`** and the integer `1` to represent **`true`**? Why do we have a separate type `bool` at all? (2 points)

2 OCaml

Let us implement the insertion sort algorithm, which has a runtime of $O(n^2)$, but is conceptually simple. It inserts elements one by one at the right position into a list that is assumed to be sorted already. Initially this list is empty.

Work in the OCaml toplevel and make sure that you test your functions properly with the test cases given, and try to come up with additional cases. Copy your two functions from the toplevel into the file `micro2.ml` at the appropriate places, run `ocaml micro2.ml` and make sure the tests pass. Submit the file `micro2.ml` in the Dropbox of this assignment.

1. Write a function `insertion : int -> int list -> int list` that takes two parameters, an element `x`, and a list `y`. It should return a list that contains the elements of `y` in the same order with the element `x` inserted.

   Fill in the following template and submit it to the OCaml toplevel. Remember to terminate your input with `;;`.

   ```
   let rec insertion x y =
     match y with
       | [] ->
       | h :: tl ->
   ```

   Make sure that your code is type correct, then evaluate the following expressions and that have to evaluate to exactly these values.

Input	Output
`insertion 0 [];;`	`- : int list = [0]`
`insertion 1 [0];;`	`- : int list = [0; 1]`
`insertion 1 [2];;`	`- : int list = [2; 1]`
`insertion 1 [0; 2];;`	`- : int list = [0; 1; 2]`
`insertion 0 [1; 2];;`	`- : int list = [0; 1; 2]`
`insertion 2 [0; 1];;`	`- : int list = [0; 1; 2]`

   Hints: Look at the first test case above to find out what the result of calling `insertion` with an empty list is. Then look at the following test cases, where the list `y` is not empty. Remember that in the case of a non-empty list you have a head element `h` that you can compare with `x`, and the tail `tl` of the list that you really cannot do much with. You will want to call the function `insertion` recursively with modified arguments. To add some element `e` to the front of a list `l` use the infix function `e :: l`. (10 points)

2. Now write a function `insertion_sort: int list -> int list` that sorts a list using the function `insertion` from the previous question. Test your function on the lists `[]`, `[1; 2; 3]`,`[3; 1; 2]` `[3; 2; 1]`.

   Hints: Again, you will want to pattern match on the shape of the input list `x`. What is the result of sorting an empty list? What is the result of sorting a list with one single element? In the case of a non-empty list, how can you sort `x` using the function `insertion` and the function `insertion_sort` recursively. (4 points)