Problem Set 1: An Introduction to OCaml

Due Thursday February 2, 23:59

Please be sure to be using Ocaml 3.12

Version: 5
Last modified: Wednesday, February 1, 20:40
Please be sure to be using Ocaml 3.12
Changes:

- Version 5 fixes the due date and a misspelling in **`cond_dup`**
- Version 4 fixes a mismatched parenthesis in **`n_times`** and uses semicolons rather than commas in **`cond_dup`**
- Fixed a typo in range which had the numbers in the wrong order.
- Fixed a type error in the example of buckets. Fixed an example error in the example of buckets

---

Instructions

This problem set has three parts. You should write your solution to each part in a separate file: **`part1.txt`**, **`part2.ml`**, and **`part3.ml`**. To help get you started, we have provided a stub file for each part on CMS. You should download and edit these files. Once you have finished, submit your solution using CMS, linked from the course website.

Important notes about grading:

1. **Compile errors:** All code you submit must compile. **Programs that do not compile will probably receive an automatic zero.** If you are having trouble getting your assignment to compile, please visit consulting hours. If you run out of time, it is better to comment out the parts that do not compile, than hand in a more complete file that does not compile.
2. **Naming:** We will be using an automatic grading script, so it is **crucial** that you name your functions and order their arguments according to the problem set instructions, and that you place the functions in the correct files. Otherwise you may not receive credit for a function properly written.
3. **Code style:** Finally, please pay attention to style. Refer to the **CS 3110 style guide** and lecture notes. Ugly code that is functionally correct may still lose points. Take the extra time to think out the problems and find the most elegant solutions before coding them up. Even though only the second part of the assignment explicitly addresses the style issue, good programming style is also required for the other parts of this assignment, and for all the subsequent assignments in the rest of the semester.
4. **Late assignments:** Please carefully review the course website's policy on late assignments, as **all** assignments handed in after the deadline will be considered late. Verify on CMS that you have submitted the correct version, *before* the deadline. Submitting the incorrect version before the deadline and realizing that you have done

so after the deadline will be counted as a late submission.

Part 1: Expression Types *(25 pts)*

Give the types of each of the following OCaml expressions. For example, the type of `(1, 2)` is `int * int`.

a. `[(3 , 4); (5, 6)]`

b. `(['a'], 'b')`

c. `[[None]]`

d.
```
fun f (z, v, t) -> if f z == "inf" && v t == "mar" then "win"
else "lose"
```

Give expressions that have the following OCaml types:

e. `char list option`

f. `bool -> 'a -> ('b -> 'a) -> 'b -> 'a`

g. `('a -> 'a) -> 'a -> 'a`

h. `char * int * float -> int * string -> int`

Replace `???` with an expression that (1) makes the code type check correctly and (2) does not cause the code to raise an exception. Give the type of the expression you chose.

i.
```
let num = ??? in
let first = num / 100 in
let second = num mod 100 in
let third = first - second - second - second in
let ans = first + second + third in
if ans != 42 then failwith "argh"
```

j.
```
let nums = ??? in
let filterer zardoz =
  List.filter (fun x -> x > 0) zardoz in
let splitter zardoz =
  let (a,b) =  List.split zardoz in b in
let zardoz = filterer (splitter nums) in
if List.length zardoz = 0 then failwith "argh"
```

Part 2: Code Style *(15 pts)*

The following function executes correctly, but it was written with poor style. Rewrite it with better style. Please consult the **CS 3110 style guide**.

```
let rec zardoz f lst =
```

```
if (List.length lst == 0) != false then []
else if (List.length lst == 1) == true && true
  then [f (List.hd lst)] @ zardoz f []
else
  let hd = List.hd lst in
  let tl = List.tl lst in
  let v = f hd in
  let lstdv = [v] in
  let rest = zardoz f tl in
  lstdv @ rest
```

Part 3: OCaml Functions *(60 pts)*

a. Write a function **`cond_dup : 'a list -> ('a -> bool) -> 'a list`** that takes in a list and preciate and duplicates all elements which satisfy the condition expressed in the predicate. For example, **`cond_dup [3;4;5] (fun x -> x mod 2 = 1) = [3;3;4;5;5]`** .

b. Write a function **`n_times : ('a -> 'a) * int * 'a -> 'a`** such that **`n_times (f,n,v)`** applies **`f`** to **`v`** **`n`** times. For example, **`n_times((fun x-> x+1), 50, 0) = 50`**.

c. Write a function **`range : int -> int -> int list`** such that **`range num1 num2`** returns an ordered list of all integers from **`num1`** to **`num2`** inclusive. For example, **`range 2 5 = [2;3;4;5]`**. Use failwith if **`num2`** < **`num1`**.

d. Write a function **`zipwith : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list`** such that **`zipwith f l1 l2`** generates a list whose ith element is obtained by applying **`f`** to the ith element of **`l1`** and the ith element of **`l2`** . If the lists have different lengths, the extra elements in the longer list should be ignored. For example, **`zipwith (+) [1;2;3] [4;5] = [5;7]`** .

e. Recall the Pythagorean Theorem. We can use it to find the hypotenuse of a right triangle, given two sides by: **`let hyp a b = sqrt((a *. a ) +. (b *. b))`** . The following code uses higher order functions to calculate the same value. Fill in the ??? below so that the whole function correctly calculate the hypotenuse of a right triangle.

```
let hyp a b =
  let d x = ??? in
  let u (x1, x2) f = ??? in
  let p f x = ??? in
  p sqrt (u (u (d a) ( *.), u (d b) ( *.)) (+.))
```

f. Write a function **`buckets : ('a -> 'a -> bool) -> 'a list -> 'a list list`** that partitions a list into equivalence classes. That is, **`buckets equiv lst`** should return a list of lists where each sublist in the result contains equivalent elements, where two elements are considered equivalent if **`equiv`** returns true. For example:

```
buckets (=) [1;2;3;4] = [[1];[2];[3];[4]]
buckets (=) [1;2;3;4;2;3;4;3;4] = [[1];[2;2];[3;3;3];[4;4;4]]
buckets (fun x y -> (=) (x mod 3) (y mod 3)) [1;2;3;4;5;6] =
  [[1;4];[2;5];[3;6]]
```