CS 3110 Fall 2014 **Due at 11:59 PM, Thursday, 09/11/14**
Problem Set 1
Version 2 (last modified September 4, 2014)

Revision log

- [09/04/14] Revised problem 1f.

Objectives

- Gain familiarity with basic OCaml features such as lists, tuples, functions, pattern matching, datatypes, and basic features of the OCaml type system.
- Practice writing programs in the functional style using immutable data and recursions.
- Appreciate the impact of code style on readability, correctness, and maintainability.

Recommended reading

The following materials should be helpful in completing this assignment:

- Course readings: lectures 1, 2, 3, and 4; recitations 1 and 2
- The CS 3110 style guide
- The OCaml tutorial
- Real World OCaml, Chapters 1–3

What to turn in

Submit four files on CMS:

1. A file `ps1written.pdf` containing your answers to the written portions of this problem set, which are identified below as "[written]".
2. Files `ps1.ml` and `ps1_test.ml` containing your solutions and unit tests for the coding exercises of this problem set, which are identified below as "[code]".
3. A commit log `ps1log.txt` documenting your activity on the repository. You can generate a `git` log and store it in `ps1log.txt` with the following command:

```
git log --stat >ps1log.txt
```

Partners and source control

You are required to work with a partner for this problem set. You can use Piazza to help find a partner. Once you have found a partner, please join as partners for the PS1 assignment on CMS. You and your partner should meet early to jointly discuss your approach. Each partner is responsible for understanding all parts of the assignment. **Don't make the mistake of assigning half the problems to one partner and half to the other: you'll both end up regretting that division of labor when the prelims arrive.** Instead, work on the entire problem set together. Working together is a powerful tool!

You are required to use `git`, a version control system, to work with your partner. Both GitHub and BitBucket provide free private repositories to students. We expect the `git` log you submit to show evidence of work over a period of time, not just a single commit at the end.

> *Private repositories are of the utmost importance. A public* `git` *repository would share your code with the entire world, including your classmates, thus violating the course policy on academic integrity.* ***Therefore we require that you keep all your CS 3110 related code in private repositories.***

To create a private repository, simply make sure you select the "Private" radio button when creating a new repository at GitHub, or check the "This is a private repository" check box on BitBucket.

Instructions

Compile Errors

All code you submit must compile. **Programs that do not compile will be heavily penalized.** If your submission does not compile, we will notify you immediately. You will have 48 hours after the submission date to supply us with a patch. If you do not submit a patch, or if your patched code does not compile, you will receive an automatic zero.

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. Incorrectly named functions are **treated as compile errors** and you will have to submit a patch.

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. Good programming style is important for all assignments throughout the semester.

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.

No Imperative Features

Imperative features—such as `ref`'s, the `Array` module, and `mutable` fields—are not permitted in your solutions to this problem set. You have not seen these features in lecture or recitation, so we doubt you'll be tempted.

Problem 1: (10 pts)

[written] Give the OCaml type (if any) of each of the following OCaml expressions, and also give the value to which each well-typed expression reduces. For each expression that is not well-typed, briefly explain why.

(a) `22 + 20`

(b) `(+) 22 20`

(c) `[1;9;4;-3.;2]`

(d) `2::4::[6;8;10]`

(e) `["zar", "doz"]`

(f) `()`

(g) `Some 3110`

(h) `(fun zar -> zar*zar) 42`

(i) `let f x = x + 1 in f f 10`

(j) `let f x = x + 1 in f (f 10)`

Problem 2: (10 pts)

[written] Give OCaml expressions that have the following types.

(a) `int`

(b) `string list`

(c) `float list -> float`

(d) `int option -> int`

(e) `int list -> int list -> int list`

(f) `(int list -> int list) -> int list`

(g) `int list -> (int list -> int list)`

(h) `int*char list -> (int * char) list`

(i) `time`

(j) `time -> int`

For parts i and j, assume the following type definition:

```
type time = {hour:int; minute:int; am_pm:string}
```

Problem 3: (20 pts)

[written] The following function executes correctly, but it was written with poor style. Your task is to rewrite it with better style. Please consult the CS 3110 style guide on the course website. You should invent a new name for the function that appropriately conveys what it does.

```
let rec zardoz ((a:int list), (b:int list)) =
  if (List.length(a) = 0 && List.length(b) = 0) then (b) else
  if (List.length(b) = 0) then List.hd(a)::List.tl(a) else
  if (List.length(a) = 0) then [] @ b else
  if (List.hd(a) < List.hd(b)) = true then [List.hd(a)]
    @ (zardoz((List.tl(a)), b)) else [List.hd(b)]
    @ (zardoz(a, (List.tl(b))))
```

Problem 4: (120 pts)

[code] Complete each of the exercises below by following these instructions:

1. Write a function with the appropriate name and type.
2. Write a specification comment above the definition of the function that documents:
   - A brief description of the function (one or two sentences).
   - A brief description of each argument (a couple of words).
   - A concise and accurate description of the function's *precondition* and *postcondition.*
3. Write unit tests that demonstrate the function's correctness. If the function is named `f`, then these tests should be named `f_test1`, `f_test2`, ... `f_testn`. How many unit tests should you write? As many as necessary to make you confident that your solution is correct. Your tests should be in a separate file named `ps1_test.ml`.

Exercise 1.

Write a function `is_mon_inc: int list -> bool` that takes an integer list and returns whether that list is monotonically increasing. For example,

- `is_mon_inc [1;2;3;6;9] = true`
- `is_mon_inc [1;3;5;7;5;9] = false`
- `is_mon_inc [1;1;2;3;4;4] = true`

Exercise 2.

Write a function `is_unimodal: int list -> bool` that takes an integer list and returns whether that list is unimodal. A *unimodal list* is a list that monotonically increases to some maximum value then monotonically decreases after that value. Either or both segments (increasing or decreasing) may be empty. For example:

- `is_unimodal [1;2;3;6;9;5;4] = true`
- `is_unimodal [1;3;5;7;5;6] = false`
- `is_unimodal [1;1;2;3;4;4;3;2;2;-1] = true`
- `is_unimodal [] = true`
- `is_unimodal [1;1;1] = true`
- `is_unimodal [1;2] = true`
- `is_unimodal [2;1] = true`

Exercise 3.

Write a function `powerset: int list -> int list list` that takes a set S represented as a list and returns the powerset of S. Recall that the *powerset* of a set S is the set of all subsets of S. For example:

- `powerset [1;2;3] = [[]; [1]; [2]; [3]; [1;2];[1;3];[2;3]; [1;2;3]]`
- `powerset [] = [[]]`

The order of the subsets in the powerset and the order of the elements in the subsets do not matter.

Hint: Consider the recursive structure of this exercise. Suppose you already have `p`, such that `p = powerset s`. How could you use `p` to compute `powerset (x::s)`?

Exercise 4.

Write a function `rev_int: int -> int` that takes an integer i and returns an integer whose digits are the reverse of i. The sign should remain unchanged. If the reversed integer is larger than `max_int`, the behavior is undefined: your function can do whatever you want. For example:

- `rev_int 1234 = 4321`
- `rev_int 4 = 4`
- `rev_int -1234 = -4321`

- `rev_int -10 = -1`
- `rev_int 1111111111 = 1111111111`
- `rev_int 1123456789 = (*undefined. 9876543211 > max_int on the 3110 VM *)`

Exercise 5.

Flattening is the process of converting a list of lists into a single list (see `List.flatten`). Write the reverse operation `unflatten: int -> 'a list -> 'a list list option` that takes an integer k and list `lst` and breaks it up into a list of lists, each of size k. In the case that `List.length lst` is not a multiple of k, the last list is allowed to be of size less than k. If `k` ≤ 0, then `unflatten k l` should return `None`. For example:

- `unflatten (-1) [1;2;3;4;5;6] = None`
- `unflatten 0 [1;2;3;4;5;6] = None`
- `unflatten 2 [1;2;3;4;5;6] = Some [[1;2]; [3;4]; [5;6]]`
- `unflatten 3 [1;2;3;4;5;6;7;8] = Some [[1;2;3]; [4;5;6]; [7;8]]`
- `unflatten 6 [1;2;3;4;5;6] = Some [[1;2;3;4;5;6]]`
- `unflatten 7 [1;2;3;4;5;6] = Some [[1;2;3;4;5;6]]`

Exercise 6.

A *Roman numeral* can be represented as a list of letters from the set {`I`, `V`, `X`, `L`, `C`, `D`, `M`}. The value of each letter is as follows:

Column 1		Column 2	
`I`	1	`V`	5
`X`	10	`L`	50
`C`	100	`D`	500
`M`	1000		

The letters are usually ordered from greater-valued to smaller-valued. If that ordering is broken, it means that the immediately preceding (lower) value is deemed to be "negative" and should be subtracted from the higher (out of place) value. For example, `IV` represents 4, and `XC` represents 90.

For purposes of this problem, define a *valid* Roman numeral as follows:

1. A letter from Column 1 may never appear more than three times in a row, and there may never be more than one additional occurrence of that letter.
2. A letter from Column 2 may never appear more than once.

3. Once a letter Z has been used in a "negative" position, all subsequent letters except the immediately following character may not be greater than Z.

For example, `MMMCX` is valid Roman numeral, but `XCMMM` is not.

We can define a type for Roman numerals in OCaml as follows:

```
type numeral = I | V | X | L | C | D | M
type roman = numeral list
```

Complete `int_of_roman`, assuming only valid Roman numerals as input:

```
let rec int_of_roman (r : roman) : int =
  let int_of_numeral = function
    | I -> 1
    | V -> 5
    | X -> 10
    | L -> 50
    | C -> 100
    | D -> 500
    | M -> 1000 in
  ???
```

For example:

- `int_of_roman [I; I; I] = 3`
- `int_of_roman [X; L; I; I] = 42`
- `int_of_roman [M; C; M; X; C; I; X] = 1999`

Exercise 7.

[written,ungraded] At the end of your file of written problems, please include any comments you have about the problem set or about your solutions. This would be a good place to list any known problems with your submission that you weren't able to fix, or to give us general feedback about how to improve the problem set.

Also, include a statement of what work in this problem set was done by which partner. The ideal case is that each of you contributed to every problem. But (especially since this exercise is ungraded) please be honest about how you divided the work.