CS251 Programming Languages **Handout # 15**
Prof. Lyn Turbak **February 7, 2007**
Wellesley College

Problem Set 2

Due: 11:59pm Wednesday, February 14

Overview:

The purpose of this assignment is to give you practice writing list recursions in OCAML. It is strongly recommended that you (1) start early and (2) work with a partner. Allocate time over several days to work on the problems; it is very unwise to start the assignment only a day or two before it is due. Don't hesitate to ask for help if you hit a roadblock. **This assignment also includes an individual problem that you must solve on your own.**

Reading:

- Handout #9 (*Introduction to the Objective Caml Programming Language*), Chapters 1–5.
- Handout #14 (*List Processing in* OCAML)

Individual Problem Submission:

Each student should turn in a hardcopy submission packet for the individual problem by slipping it under Lyn's office door by 6pm on the due date. The packet should include the individual problem header sheet and the final version of `factors.ml`.

Each student should also submit a softcopy (consisting of your final `ps2-individual` directory) to the drop directory `~cs251/drop/ps2/`*username*, where *username* is your username. To do this, execute the following commands in Linux in the account of the team member being used to store the code.

```
cd /students/username/cs251
cp -R ps2-individual ~cs251/drop/ps2/username/
```

Group Problem Submission:

Each team should turn in a single hardcopy submission packet for all problems by slipping it under Lyn's office door by 6pm on the due date. The packet should include:

1. a group problem header sheet indicating the time that you (and your partner, if you are working with one) spent on the parts of the assignment.
2. your final version of `listfuns.ml`.

Each team should also submit a single softcopy (consisting of your final `ps2-group` directory) to the drop directory `~cs251/drop/ps2/`*username*, where *username* is the username of one of the team members (indicate which drop folder you used on your hardcopy header sheet). To do this, execute the following commands in Linux in the account of the team member being used to store the code.

```
cd /students/username/cs251
cp -R ps2-group ~cs251/drop/ps2/username/
```

Starting the Assignment

To start this assignment, you will need to "download" the CVS-controlled files for the assignment. You can do this by executing the following in a Linux shell:

```
cd ~/cs251
cvs update -d
```

Executing the above commands will create two local directories, `~/cs251/ps2-individual` and `~/cs251/ps2-group` containing the following files:

1. `ps2-individual/factors.ml`: This file contains skeletons for the two OCAML functions you are asked to define in the Individual Problem. You should flesh out each of these skeletons. To obtain full credit in this problem, you should not define any auxiliary functions.

2. `ps2-group/listfuns.ml`: This file contains skeletons for each of the OCAML functions you are asked to define in the Group Problems. You should flesh out each of the skeletons as you do the problems. In many of the problems it will also be helpful to define additional auxiliary functions. You are welcome to use any functions defined in class, as well as any other functions you need.

3. `ps2-group/listfuns-test.ml`: This file contains code for testing each of your Group Problem functions on some simple test cases. You can test the function in Group Problem *n* by evaluating the function invocation `test`*n*`()` in the OCAML interpreter. You can test all the functions on the assignment by evaluating the function invocation `testall()`. Note that even if your function passes all the test cases, it is not guaranteed to be correct; you are encouraged to extend the test cases in the testing file.

4. `ps2-group/load-listfuns.ml`: This file is used to load the other two `ps2-group` files into the OCAML interpreter, as well as the file `~/cs251/download/utils/Handout14ListFuns.ml`. The latter file contains the solutions to the list-processing functions from Handout #14. It is loaded first in case you want to use any of its functions when constructing your solutions.

To start working on the rest of the problems on this assignment, you will need to launch the OCAML interpreter. There are several ways to do this; see Handout #10. It is recommended that you choose the approach in which you create a dedicated OCAML buffer within Emacs, since this simplifies many interactions.

Within the OCAML interpreter, execute the following to load the Individual Problem code:

```
#cd "/students/username/cs251/ps2-individual";;
#use "factors.ml";;
```

Note that the hash mark (`#`) is part of the command name and is *not* the prompt of the OCAML interpreter. After making any change to `factors.ml`, you should re-execute

```
#use "factors.ml";;
```

to inform the OCAML interpreter of your changes.

To load the Group Problem code, you should execute the following in an OCAML interpreter:

```
#cd "/students/username/cs251/ps2-group";;
#use "load-listfuns.ml";;
```

This will load both `listfuns.ml` and `listfuns-test.ml`. After making any change to `listfuns.ml` or `listfuns-test.ml`, you should re-execute

```
#use "load-listfuns.ml";;
```

to inform the OCAML interpreter of your changes.

Individual Problem [20]: Major and Minor Factors

This is an individual problem. Each student must solve this problem on her own without consulting any other person (except Lyn).

Any integer d that divides a positive integer n without leaving a remainder is said to be a *factor* of n. Every positive integer $n \geq 2$ has a unique *prime factorization* – a multiset[1] of prime numbers ≥ 2 whose product is n. For example, here are some numbers and their prime factorizations:

Integer	**Prime Factorization**	**Integer**	**Prime Factorization**
60	$2 \cdot 2 \cdot 3 \cdot 5$	88	$2 \cdot 2 \cdot 2 \cdot 11$
72	$2 \cdot 2 \cdot 2 \cdot 3 \cdot 3$	90	$2 \cdot 3 \cdot 3 \cdot 5$
75	$3 \cdot 5 \cdot 5$	91	$7 \cdot 13$
80	$2 \cdot 2 \cdot 2 \cdot 2 \cdot 5$	96	$2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3$
81	$3 \cdot 3 \cdot 3 \cdot 3$	97	97
82	$2 \cdot 41$	100	$2 \cdot 2 \cdot 5 \cdot 5$

Suppose $n \geq 2$. For the purposes of this problem, we shall define the *major factor* of n to be the product of exactly one occurrence of each of its prime factors and the *minor factor* to be the product of all remaining occurrences of its prime factors. For example, the prime factorization of 360 is $2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5$, so its major factor is $2 \cdot 3 \cdot 5 = 30$ (one occurrence each of the three prime factors 2, 3, and 5) and its minor factor is $2 \cdot 2 \cdot 3 = 12$ (the extra 2s and 3s). Note that every integer n is the product of its major and minor factors. Here are some examples:

Integer	**(Major Factor, Minor Factor)**	**Integer**	**(Major Factor, Minor Factor)**
60	$(30, 2)$	88	$(22, 4)$
72	$(6, 12)$	90	$(30, 3)$
75	$(15, 5)$	91	$(91, 1)$
80	$(10, 8)$	96	$(6, 16)$
81	$(3, 27)$	97	$(97, 1)$
82	$(82, 1)$	100	$(10, 10)$

a. [**8**] In the file `~/cs251/ps2-individual/factors.ml`, define a function

```
val smallestPrimeFactor : int -> int
```

such that `smallestPrimeFactor` *n* returns the smallest prime factor ≥ 2 of an integer $n \geq 2$. If n itself is prime number, then the result of `smallestPrimeFactor` *n* is n. For simplicity, you should determine the smallest prime factor by trying every possible divisor between 2 and $\lfloor \sqrt{n} \rfloor$ (this includes many non-primes) until either one is found or it is determined that n is prime. Convert between `int` and `float` values using `int_of_float` and `float_of_int`.

b. [**12**] In the file `~/cs251/ps2-individual/factors.ml`, define a function

```
val majorMinorFactors : int -> int * int
```

such that `majorMinorFactors` *n* returns a pair of the major and minor factors of n. For full credit, your definition should not mention any user-defined functions other than `smallestPrimeFactor` and `majorMinorFactors`. (*Hint:* Use divide/conquer/glue!) Partial credit will be awarded for correct definitions of `majorMinorFactors` in which you need to define additional functions. Turn in a transcript showing that `majorMinorFactors` works correctly on all the examples in the table shown above.

[1]Recall from CS230 that a multiset (also known as a bag) is an unordered collection of elements that may contain duplicates.

Group Problems

Below are the specifications for nine functions. In the file `~/cs251/ps2-group/listfuns.ml`, write definitions for each of the nine functions. Thinking carefully about your strategy before you start coding will save you lots of time! The divide/conquer/glue strategy you are familiar with from CS111 and CS230 can be use to solve all the problems.

Group Problem 1 [**10**] `val sum_multiples_of_3_or_5 : int * int -> int`
`sum_multiples_of_3_or_5 (`*m*`,`*n*`)` returns the sum of all integers from *m* up to *n* (inclusive) that are multiples of 3 and/or 5. For example:

```
# sum_multiples_of_3_or_5 (0,10);;
- : int = 33 (* 3 + 5 + 6 + 9 + 10 *)
# sum_multiples_of_3_or_5 (-9,12);;
- : int = 22
# sum_multiples_of_3_or_5 (18,18);;
- : int = 18
# sum_multiples_of_3_or_5 (10,0);;
- : int = 0 (* The range "10 up to 0" is empty. *)
```

Group Problem 2 [**5**] `val contains_multiple : int * int list -> bool`
`contains_multiple (`*n*`,`*ns*`)` returns `true` if *n* evenly divides at least one element of the integer list *ns*; otherwise it returns `false`. Use the infix `mod` function to determine divisibility. E.g. `17 mod 5` denotes 2.

```
# contains_multiple (5, [8;10;14]);;
- : bool = true
# contains_multiple (3, [8;10;14]);;
- : bool = false
# contains_multiple (5, []);;
- : bool = false
```

Group Problem 3 [**5**] `val all_contain_multiple : int * int list list -> bool`
`all_contain_multiple (`*n*`,`*nss*`)` returns `true` if each list of integers in `nss` contains at least one integer that is a multiple of n; otherwise it returns `false`.

```
# all_contain_multiple (5, [[17;10;12]; [25]; [3;7;5]]);;
- : bool = true
# all_contain_multiple (3, [[17;10;12]; [25]; [3;7;5]]);;
- : bool = false
# all_contain_multiple (3, []);;
- : bool = true
```

Group Problem 4 [10] `val merge : 'a list * 'a list -> 'a list`
Assume that *xs* and *ys* are both lists ordered from small to large by `<=`. Then `merge (`*xs*`,`*ys*`)` returns a list containing all the elements of *xs* and *ys* in sorted order.

```
# merge ([1;4;5;7], [2;3;5;9]);;
- : int list = [1; 2; 3; 4; 5; 5; 7; 9]
# merge (['a';'d';'f'], ['b'; 'c'; 'e']);;
- : char list = ['a'; 'b'; 'c'; 'd'; 'e'; 'f']
# merge ([], []);;
- : '_a list = []
```

Group Problem 5 [15] `val alts : 'a list -> 'a list * 'a list`
Assume that the elements of a list are indexed starting with 1. `alts` *xs* returns a pair of lists, the first of which has all the odd-indexed elements (in the same relative order as in *xs*) and the second of which has all the even-indexed elements (in the same relative order as in *xs*).

```
# alts [7;5;4;6;9;2;8;3];;
- : int list * int list = ([7; 4; 9; 8], [5; 6; 2; 3])
# alts [7;5;4;6;9;2;8];;
- : int list * int list = ([7; 4; 9; 8], [5; 6; 2])
# alts [7];;
- : int list * int list = ([7], [])
# alts [];;
- : '_a list * '_a list = ([], [])
```

Group Problem 6 [15] `val cartesian_product : 'a list * 'b list -> ('a * 'b) list`
`cartesian_product (`*xs*`,`*ys*`)` returns a list of all pairs `(`*x*`,`*y*`)` where *x* ranges over the elements of *xs* and *y* ranges over the elements of *ys*. The pairs should be sorted first by the *x* entry (relative to the order in *xs*) and then by the *y* entry (relative to the order in *ys*).

```
# cartesian_product ([1; 2], ['a'; 'b'; 'c']);;
- : (int * char) list =
[(1, 'a'); (1, 'b'); (1, 'c'); (2, 'a'); (2, 'b'); (2, 'c')]
# cartesian_product ([2; 1], ['a'; 'b'; 'c']);;
- : (int * char) list =
[(2, 'a'); (2, 'b'); (2, 'c'); (1, 'a'); (1, 'b'); (1, 'c')]
# cartesian_product (['c'; 'a'; 'b'], [2; 1]);;
- : (char * int) list =
[('c', 2); ('c', 1); ('a', 2); ('a', 1); ('b', 2); ('b', 1)]
# cartesian_product ([1], ['a']);;
- : (int * char) list = [(1, 'a')]
# cartesian_product ([], ['a'; 'b'; 'c']);;
- : ('_a * char) list = []
```

Group Problem 7 [10] `val bits : int -> int list`
`bits` *`n`* returns a list of the bits (0s and 1s) in the binary representation of n.

```
# bits 5;;
- : int list = [1; 0; 1]
# bits 10;;
- : int list = [1; 0; 1; 0]
# bits 11;;
- : int list = [1; 0; 1; 1]
# bits 22;;
- : int list = [1; 0; 1; 1; 0]
# bits 23;;
- : int list = [1; 0; 1; 1; 1]
# bits 46;;
- : int list = [1; 0; 1; 1; 1; 0]
# bits 1;;
- : int list = [1]
# bits 0;;
- : int list = [0]
```

Group Problem 8 [15] `val inserts : 'a * 'a list -> 'a list list`
Assume that *ys* is a list with n elements. `insert (`*`x`*`,`*`ys`*`)` returns a $n+1$-length list of lists showing all ways to insert a single copy of x into *xs*.

```
# inserts (3, [5;7;1]);;
- : int list list = [[3; 5; 7; 1]; [5; 3; 7; 1]; [5; 7; 3; 1]; [5; 7; 1; 3]]
# inserts (3, [5;3;1]);;
- : int list list = [[3; 5; 3; 1]; [5; 3; 3; 1]; [5; 3; 3; 1]; [5; 3; 1; 3]]
# inserts (3, []);;
- : int list list = [[3]]
```

Group Problem 9 [15] `val permutations : 'a list -> 'a list list`
Assume that *xs* is a list of distinct elements (i.e., no duplicates). `permutations` *`xs`* returns a list of all the permutations of the elements of *xs*. The order of the permutations does not matter. *Hint:* Use `inserts` as a helper function.

```
# permutations [];;
- : '_a list list = [[]]
# permutations [4];;
- : int list list = [[4]]
# permutations [3;4];;
- : int list list = [[3; 4]; [4; 3]]
# permutations [2;3;4];;
- : int list list =
[[2; 3; 4]; [3; 2; 4]; [3; 4; 2]; [2; 4; 3]; [4; 2; 3]; [4; 3; 2]]
# permutations [1;2;3;4];;
- : int list list =
[[1; 2; 3; 4]; [2; 1; 3; 4]; [2; 3; 1; 4]; [2; 3; 4; 1];
 [1; 3; 2; 4]; [3; 1; 2; 4]; [3; 2; 1; 4]; [3; 2; 4; 1];
 [1; 3; 4; 2]; [3; 1; 4; 2]; [3; 4; 1; 2]; [3; 4; 2; 1];
 [1; 2; 4; 3]; [2; 1; 4; 3]; [2; 4; 1; 3]; [2; 4; 3; 1];
 [1; 4; 2; 3]; [4; 1; 2; 3]; [4; 2; 1; 3]; [4; 2; 3; 1];
 [1; 4; 3; 2]; [4; 1; 3; 2]; [4; 3; 1; 2]; [4; 3; 2; 1]]
```

Extra Credit 1 [20]: Permutations in the Presence of Duplicates

This problem is optional. You should only attempt it after completing the rest of the problems.

Define a version of the `permutations` function from Problem 9 named `permutationsDup` that correctly handles lists with duplicate elements. That is, each permutation of such a list should only be listed once in the result. You should *not* generate duplicate permutations and then remove them; rather, you should just not generate any duplicates to begin with. As before, the order of the permutations does not matter.

```
# permutationsDup [2;1;2];;
- : int list list = [[1;2;2];[2;1;2];[2;2;1]] (* order doesn't matter *)
# permutationsDup [1;2;1;2;2]
- : int list list =
[[1;1;2;2;2]; [1;2;1;2;2]; [1;2;2;1;2]; [1;2;2;2;1];
 [2;1;1;2;2]; [2;1;2;1;2]; [2;1;2;2;1];
 [2;2;1;1;2]; [2;2;1;2;1]; [2;2;2;1;1]] (* order doesn't matter *)
```

Individual Problem Header Page
Please make this the first page of your hardcopy submission of individual problems.

CS251 Problem Set 2 Individual Problems
Due 11:59pm Wednesday, February 14

Name:

Date & Time Submitted:

By signing below, I attest that I have followed the policy for individual problems set forth in the Course Information handout. In particular, I have not consulted with any person except Lyn about these problems and I have not consulted any materials from previous semesters of CS251.

Signature:

In the **Time** *column, please estimate the time you spent on the parts of this problem set. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the* **Score** *column when grading you problem set.*

Part	**Time**	**Score**
General Reading		
Problem 1a [8]		
Problem 1b [12]		
Total		

Group Problem Header Page
Please make this the first page of your hardcopy submission for group problems.

CS251 Problem Set 2 Group Problems
Due 11:59pm Wednesday, February 14

Names of Team Members:

Date & Time Submitted:

Collaborators (*anyone you or your team collaborated with on the problem set*):

By signing below, I/we attest that I/we have followed the collaboration policy as specified in the Course Information handout.
Signature(s):

In the **Time** *column, please estimate the time you or your team spent on the parts of this problem set. Team members should be working closely together, so it will be assumed that the time reported is the time for each team member. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the* **Score** *column when grading you problem set.*

Part	**Time**	**Score**
General Reading		
Problem 1 [10]		
Problem 2 [5]		
Problem 3 [5]		
Problem 4 [10]		
Problem 5 [15]		
Problem 6 [15]		
Problem 7 [10]		
Problem 8 [15]		
Problem 9 [15]		
Total		