CS251 Programming Languages **Handout # 11**
Prof. Lyn Turbak **January 28, 2004**
Wellesley College

Problem Set 1

Due: 6pm Thursday, February 5

Overview:

The purpose of this assignment is to give you practice writing list recursions in OCAML. Since learning a new programming language and programming environment takes time, 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.

Reading:

- Handouts #1 – #10 (only Chapters 1–5 of Handout #8 (Introduction to OCAML)).

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 team header sheet (see the end of this assignment for the 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 `ps1.ml`.

Each team should also submit a single softcopy (consisting of your final `ps1` directory) to the drop directory `~cs251/drop/ps1/`*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 ps1 ~cs251/drop/ps1/username/
```

Problem 0: Getting Started

a. : Linux

You should begin this assignment by learning (or reminding yourself) how to use Linux. See the information in Handouts #3 and #4 on using Linux/Unix.

b. : Emacs

Next you should learn (or remind yourself) how to use Emacs. See the information in Handouts #3 and #5 on using Emacs. Learning to execute all cursor-motion and editing commands via keystrokes (rather than via mouse and menus) is an important skill that will save you lots of time over the semester. It will also make it easier for you to work remotely via `telnet`/`ssh`. A good way to begin learning the keystroke commands is taking the online interactive Emacs tutorial (see Handout #3 for how to do this).

c. : CVS

To do the rest of the problems on this assignment, you will need to use several files that are in the CVS-controlled CS251 repository. Follow the directions in Handout #7 for how to install your local CVS filesystem. You only need to install it once.

Once you have installed your local CVS filesystem, you can access all CVS-controlled files by executing the following in a Linux shell:

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

Indeed, every time you log in to a Linux machine to work on a CS251 assignment, you should execute the above commands to ensure that you have the most up-to-date versions of the problem set materials.

On this assignment, executing the above commands will create the local directory `~/cs251/ps1` containing three files:

1. `ps1.ml`: This file contains skeletons for each of the OCAML functions you are asked to define. 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.
2. `ps1-test.ml`: This file contains code for testing each of your functions on some simple test cases. You can test the function in 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.
3. `load-ps1.ml`: This file is used to load the other two files into the OCAML interpreter. (See below.)

d. : OCAML Interpreter

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 #9. It is recommended that you choose one of the ways to work within Emacs, since this simplifies many interactions.

Within the OCAML interpreter, execute the following to load the PS1 code:

```
#cd "/students/username/cs251/ps1";;
#use "load-ps1.ml";;
```

Note that the hash mark (`#`) is part of the command name and is *not* the prompt of the OCAML interpreter. For some reason, OCAML does not understand the abbreviation `~` for `/students/`*username*, so you must write out the long form. After making any change to `ps1.ml` or `ps1-test.ml` you should re-execute

```
#use "load-ps1.ml";;
```

to inform the OCAML interpreter of your changes.

Below are the specifications for nine functions. 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-and-glue strategy you are familiar with from CS111 and CS230 can be use to solve all problems.

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. *)
```

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
```

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
```

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 = []
```

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 = ([], [])
```

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 = []
```

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]
```

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]]
```

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.

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

Problem Set Header Page
Please make this the first page of your hardcopy submission.

CS251 Problem Set 1

Due 6pm Thursday, February 5

Names of Team Members:

Date & Time Submitted:

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

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		