MP 2 – Pattern Matching and Recursion

CS 421 – Fall 2014
Revision 1.2

Assigned September 2, 2014
Due September 9, 2014 23:59pm
Extension 48 hours (20% penalty)

1 Change Log

1.0 Initial Release.

1.1 Reverse order in Problem 11.

1.2 Moved Problems 6-12 to MP3.

2 Objectives and Background

The purpose of this MP is to help the student master:

- pattern matching
- recursion

3 What to `commit`

You should put code answering each of the problems below in a file called `mp2.ml`. A good way to start is with the stub file we have given you, editing to remove the stubs and insert you answers. Please read the *Guide for Doing MPs* in

`http://courses.engr.illinois.edu/cs421/mps/index.html`

The command to commit this file is:

```
svn commit -m "Turning in mp2." mp2.ml
```

4 Instructions for Doing the Problems

The problems below have sample executions that suggest how to write answers. You have to use the same function name, but the name of the parameters that follow the function name need not be duplicated. That is, you are free to choose different names for the arguments to the functions from the ones given in the example execution. We will sometimes use `let rec` to begin the definition of a function that is allowed to use `rec`. You are not required to start your code with `let rec`, and you may use `let rec` when we do not.

For all these problems, you are allowed to write your own auxiliary functions, either internally to the function being defined or externally as separate functions. In fact, you will find it helpful to do so on several problems. In this assignment, **you may not use any library functions other than `@`, `sqrt` and `mod`**.

5 Problems

1. (2 pts) Write a function `is_real_date : int * int -> bool` that, given a pair of integers, determines if it represents a “real date” this year, where the first integer is the month and the second is the day.

```
# let is_real_date (m, d)  = ... ;;
val is_real_date : int * int -> bool = <fun>
# is_real_date(1, 0);;
- : bool = false
```

2. (2 pts) Write a function `largest_of_3 : 'a -> 'a -> 'a -> 'a` that, when applied to three integers, returns the one that is the largest.

```
# let largest_of_3 x y z = ...;;
val largest_of_3 : 'a -> 'a -> 'a -> 'a = <fun>
# largest_of_3 "a" "c" "b";;
- : string = "c"
```

3. (2 pts) Write a function `do_both : ('a -> 'b) -> ('a * 'a) -> ('b * 'b)` that takes a function, and then a pair and applies the function to each element in the pair.

```
# let do_both f (x,y) = ...;;
val do_both : ('a -> 'b) -> 'a * 'a -> 'b * 'b = <fun>
# do_both (fun x -> x + 1) (3,4);;
- : int * int = (4, 5)
```

4. (2 pts) A triangular number is given by the following formula:

$$T_n = \sum_{k=1}^{n} = 1 + 2 + 3 + \cdots + n$$

Write a recursive function `triangular : int -> int` that computes the n^{th} triangular number. For $n < 1$, you should return 0.

```
let rec triangular n = ... ;;
# triangular 3;;
- : int = 6
```

5. (3 pts) Write a function `pascal : int -> int -> int` that returns the pascal number at position (c, r), where first parameter to the `pascal` function is the column c and the second is the row r (arguments are given in a curried form). The rows start with row 0, the elements in each row are numbered from the left beginning with 0. The value at $(0, 0)$ is 1, and all other columns in row 0 have a value of 0. For all rows lees than 0, all columns have the value of 0. For the other rows, the value of column c is the sum of the colunms $c - 1$ and c in the proceeding row.

1			
1	1		
1	2	1	
1	3	3	1
...	...	...	...

```
# let rec pascal c r = ... ;;
val pascal : int -> int -> int = <fun>
# pascal 1 3;;
- : int = 3
```