MP 2 – Pattern Matching and Recursion

CS 421 – Spring 2014
Revision 1.0

Assigned January 30, 2014
Due February 9, 2014 11:59pm

1 Change Log

1.0 Initial Release.

2 Objectives and Background

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

- pattern matching
- recursion

3 Instructions

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`**.

4 Problems

1. (2 pts) Write `dist : float * float -> float * float -> float` that takes two float points and calculates the Euclidean distance between them.

```
# let dist (x1, y1) (x2, y2) = ...;;
val dist : float * float -> float * float -> float = <fun>
# dist (0.,0.) (3., 4.);;
- : float = 5.
```

2. (2 pts) Write a function `fibo_num : int -> int` that takes in a number n and returns the n-th Fibonacci number counting from position 0. Specifically, the Fibonacci series is 0 for $n \leq 0$, then 1 for $n = 1$, and then each successive Fibonacci number is the sum of the previous two. So fibo_num 0 would return 0, fibo_num 1 would return 1, fibo_num 4 would return 3, and etc.

```
# let rec fibo_num n = ...
val fibo_num : int -> int = <fun>
# fibo_num 4;;
- : int = 3
```

3. (2 pts) Write a function `fibo_sum : int -> int` that takes in a number n and returns the sum of the first n Fibonacci numbers counting from position 0. If $n <= 0$ the result is 0. The Fibonacci numbers are defined as in the previous problem.

```
# let rec fibo_sum n = ...
val fibo_sum : int -> int = <fun>
# fibo_sum 5;;
- : int = 12
```

4. (2 pts) Write a OCaml function `reverse_triple_to_list : 'a * 'a * 'a -> 'a list` to reverse a triple and convert it into a list

```
# let reverse_triple_to_list (a, b, c) = ...;;
val reverse_triple_to_list : 'a * 'a * 'a -> 'a list = <fun>
# reverse_triple_to_list (1,2,3);;
- : int list = [3; 2; 1]
```

5. (2 pts) Write a function `sum : int list -> int` to find the sum of an integer list. The sum of an empty list is `0`.

```
# let rec sum l = ...;;
val sum : int list -> int = <fun>
# sum [1;2;3];;
- : int = 6
```

6. (4 pts) Write a function `min : 'a list -> 'a -> 'a` that returns the smallest element (note: watch out for the type) in a list. (The comparison operator < is polymorphic, with the restriction that it cannot be applied to values involving functions.) If the list is empty it returns the second argument.

```
# let rec min l default = ...
val min : 'a list -> 'a -> 'a = <fun>
# min [1;2;3] 0;;
- : int = 1
```

7. (4 pts) Write a function `is_sorted_ascend : 'a list -> bool` that checks if the input list is sorted in ascending order. Note, [1;2;2;3;5] is sorted in ascending order. A list is sorted in ascending order if each element in the list is less than or equal to every element that comes after it in the list.

```
# let rec is_sorted_ascend l = ...;;
val is_sorted_ascend : 'a list -> bool = <fun>
# is_sorted_ascend [1;2;3];;
- : bool = true
```

8. (4 pts) Write a function `zip : 'a list -> 'b list -> ('a * 'b) list` that takes two lists, and return a one-to-one mapping pair list. That is, each element of the first list is paired with the element of the second list in the same position. If either list is longer than the other the extra elements at the end from the longer list will be left out.

```
# let rec zip l1 l2 = ...;;
val zip : 'a list -> 'b list -> ('a * 'b) list = <fun>
# zip [1;2] [3;4];;
- : (int * int) list = [(1, 3); (2, 4)]
```

9. (4 pts) Write a function `unzip : ('a * 'b) list -> 'a list * 'b list` that restores a zipped list into a pair of two original lists. This is basically a reverse process of the zip function.

```
# let rec unzip l = ...;;
val unzip : ('a * 'b) list -> 'a list * 'b list = <fun>
# unzip [(1,3);(2,4)];;
- : int list * int list = ([1; 2], [3; 4])
```

10. (4 pts) Write a function `add_odd_pos : int list -> int` that takes in a list of integers and return a sum of all elements at odd positions. For this problem, the first element in the list (the head of the list) is at position 1, not zero. The sum of all elements at odd positions of the empty list is 0.

```
# let rec add_odd_pos l = ...
val add_odd_pos : int list -> int = <fun>
# add_odd_pos [1;2;3;4;5];;
- : int = 9
```

11. (4 pts) Write a function `insert : 'a -> 'a list -> 'a list` that takes in a element n and a list. It returns a list with the element inserted immediately before the first element that is greater than n. If there is no such position exists, insert n at the end of the list. Note, if the list is sorted in ascending order, the order is perserved by the insertion.

```
# let rec insert n l = ...
val insert : 'a -> 'a list -> 'a list = <fun>
# insert 2 [1;3];;
- : int list = [1; 2; 3]
```

5 Extra Credit

12. (5 pts) Write a function `primes : int -> int list` that takes an integer n and gives back a list of the first n primes, listed largest to smallest. If $n \leq 0$, you should return the empty list. A prime is an integer that is greater than 1, and has exactly two divisors, 1 and iteself. You will probably want to use the infixed function `mod` for this, and you will probably want to write one or more auxiliary functions.

```
# let rec primes n = ...
val primes : int -> int list = <fun>
# primes 2;;
- : int list = [3; 2]
```