
MP 2 – Pattern Matching and Recursion

CS 421 – Fall 2008

Revision 1.1

Assigned January 17, 2008

Due January 22, 2008 23:59

Extension 48 hours (20% penalty)

1 Change Log

1.1 Corrected “January 23” to “January 22”

1.0 Initial Release.

2 Objectives and Background

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

1. pattern matching
2. recursion

3 Instructions

The problems below have sample executions that suggest how to write answers. Students have to use the same function name, but the name of the parameters that follow the function name need not be duplicated. That is, the students are free to choose different names for the arguments to the functions from the ones given in the example execution. We also will 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`.

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** (including `@`).

4 Problems

4.1 Pattern Matching

1. (3 pts) Write a function `plus_times` that adds the first component of a triple the the product of the other two.

```
# let plus_times ... = ...
val plus_times : int * int * int -> int = <fun>
# plus_times (3,4,5);;
- : int = 23
```

2. (4 pts) Write a function `sum_prod` that returns a pairs of the sum of the components of the input pair and the product of the components of the input pair.

```
# let sum_prod ... = ...
val sum_prod : int * int -> int * int = <fun>
# sum_prod (7, 9);;
- : int * int = (16, 63)
```

3. (5 pts) Write a function `midlist` that returns a list containing the second and third elements of a list, where the output list is in the same order as the input list. If the list does not have at least three elements, `midlist` should return the empty list.

```
# let midlist list = ...
val midlist : 'a list -> 'a list = <fun>
# midlist [4;6;8;10];;
- : int list = [6; 8]
```

4.2 Recursion

4. (6 pts) Write a function `all_nonneg` that will return whether all the elements in a list are greater than or equal to 0.

```
# let rec all_nonneg list = ...
val all_nonneg : int list -> bool = <fun>
# all_nonneg [1; 3; -5; -7];;
- : bool = false
```

5. (6 pts) Write a function `binom_list` whose argument is a list of pairs (n,m) (with $n \geq m$) that returns a list with elements $(n,m,\text{binom } n \text{ m})$.

Note: You may want to re-implement the function `binom` from the first MP (or just copy your solution over) as a helper function.

```
# let rec binom_list ... = ...
val binom_list : (int * int) list -> (int * int * int) list = <fun>
# binom_list [(4,1);(4,2);(4,3);(4,4)];;
- : (int * int * int) list = [(4, 1, 4); (4, 2, 6); (4, 3, 4); (4, 4, 1)]
```

6. (6 pts) Write a function `flatten` that returns the list consisting of the concatenation of the lists in the argument. Do **not** use the built-in `@` operator.

```
# let rec flatten ... = ...
val flatten : 'a list list -> 'a list = <fun>
# flatten [[1;2;3]; [4;5]; [8;2;3;4]];
- : int list = [1; 2; 3; 4; 5; 8; 2; 3; 4]
```

7. (6 pts) Write a function `expand`, that, for each pair (n,s) in the argument, adds n copies of s ($n \geq 0$) to the result.

```
# let rec expand ... = ...
val expand : (int * 'a) list -> 'a list = <fun>
# expand [(3,"ab"); (0, "c"); (4,"d")];;
- : string list = ["ab"; "ab"; "ab"; "d"; "d"; "d"; "d"]
```

8. (6 pts) Write a function `merge` whose arguments are lists in ascending order, and whose result is the merging of the two lists, also in ascending order.

```
# let rec merge ... .. = ...
val merge : 'a list -> 'a list -> 'a list = <fun>
# merge [1;3;5;7] [4;5;6;7];;
- : int list = [1; 3; 4; 5; 5; 6; 7; 7]
```

9. (5 pts) Write a function `sep_concat` that returns the result of concatenating from left to right all the string in the input list, with a blank space in between each, There should be no additional space added at the beginning or end of the string, and if the input is empty, the result should be the empty string ("").

```
# let rec sep_concat ... = ...
val sep_concat : string list -> string = <fun>
# sep_concat ["Hi"; "there, "; "Johnny"];;
- : string = "Hi there, Johnny"
```

4.3 Extra Credit

10. (2 pts) Write a function `zip` that takes two lists `lst1` and `lst2`, of lengths i and j , respectively, and returns a list of length $\min(i, j)$ whose k_{th} element is the pair with first part the k_{th} element of `lst1` and second part the k_{th} element of `lst2`.

```
# let rec zip ... .. = ...;;
val zip : 'a list -> 'b list -> ('a * 'b) list = <fun>
# zip [3;4;5;6] [1;2;8;9;4];;
- : (int * int) list = [(3, 1); (4, 2); (5, 8); (6, 9)]
```

11. (3 pts) Write a function `unzip` that takes a list `lst` of pairs and returns a pair whose first part is a list of the first parts of the elements of `lst`, and whose second part is a list of the second parts of the elements of `lst`.

```
# let rec unzip ... = ...;;
val unzip : ('a * 'b) list -> 'a list * 'b list = <fun>
# unzip [(3,1); (4,2); (5,8); (6,9)];;
- : int list * int list = ([3; 4; 5; 6], [1; 2; 8; 9])
```