ARIZONA STATE UNIVERSITY, COMPUTER SCIENCE

CSE 340 – SPRING 2020

Homework 5

Due Thursday April 30 by 11:59 PM

General Instructions

1. When you submit your solution to GradeScope, you should enter the answers to separate problems separately
2. Your answers should be typed.

[Hindley-Milner Type Inference]. For each of the following definitions, give the type of the function and its arguments. If there is a type mismatch, you should explain the reason for the mismatch. Note that for recursive functions, **`let rec`** is used. If there is no type mismatch, you are only asked to give the answer without explanation. I realize that you can get the answer by typing the code in an OCaml editor, but if you do so, you will guarantee that you will not do well on the final on the Hindley-Milner section.

Note that recusive functions are declared with `let rec`
Note variable and function names in OCaml start with lowercase.

Problem 1. let f x = [1]

Problem 2. let f x = [1 ; 2.0]

Problem 3. let f x = [(1,2.0)]

Problem 4. let f x = [x ; x]

Problem 5. let f a b = if a then b else b + 1

Problem 6. let f a b = if a b then b else b + 1

Problem 7. function that applies f to every element of a list

```
let rec map f l = match l with
          []      -> []
        | h::t    -> f h :: map f t
```

Problem 8. Function that applies a function to the elements of a list and sums the results.

```
let rec sum f l  = match l with
          []      -> 0
        | h::t    -> f h + sum f  t
```

Problem 9. The previous function is restrictive. It only allows us to add the results together which, in addition to restricting us to addition, restricts us to integer functions. This problem defines a more

general function that takes a function to combine the results as an argument. This is mapReduce (some of you might have heard the term).

```
let rec mapReduce comb map l = match l with
          []      ->  None
        | h::t    ->  comb (map h)  (mapReduce  comb map t)
```

For problems 10-14, each problem builds on the previous ones. For the question on evaluation, you can use https://try.ocamlpro.com/. You can also use it for help with the types, but as I said, you should be able to do the derivations yourself.

Problem 10.
```
let rec mll l = match l with
          []      -> []
        | h::l1   -> [h]::(mll l1)
```

What is the type of mll?
What does mll [1;3;4;5] evaluate to?

Problem 11.
```
let rec merge l1 l2    = match (l1,l2) with
          ([],_)              ->        l2
        | (_,[])              ->        l1
        | (h1::t1,h2::t2) ->         if h1 > h2 then
                                              h1 :: ( merge t1 l2 )
                                     else
                                              h2::  ( merge l1 t2 )
```
what is the type of merge?
what does merge [5;3;1] [7;6;0] evaluate to?

Problem 12.
```
let rec mergecl l     = match l with
          []              -> []
        | a::[]           -> [a]
        | a::b::t         -> merge a b :: ( mergecl t )
```

What is the type of mergecl?
What does merge [[3];[1];[1];[3];[2];[5]] evaluate to?

Problem 13.
```
let rec mergesortll ll = match ll with
          [a]     -> a
        | _       -> mergesortll (mergecl ll)
```

What is the type of mergesortll?
What does mergesortll [[1];[2];[3];[4];[5]] evaluate to?

Problem 14.
```
let mergesort l = mergesortll (mll l)
```

What is the type of mergesort?