Functional Programming

Mock test for WISEflow

Exam Question 1, May 27, 2024

Version 0.98 of April 6, 2025

These exam questions comprise 3 pages. Check immediately that you have all the pages.

The exam duration is 4 hours.

There is 1 question. To obtain full marks you must answer all the subquestions satisfactorily.

You do not have general Internet access at the exam. You can only access WISEflow. You are allowed to use books, lecture notes, lecture slides, hand–ins, solutions to assignments, calculators, computers, and software used throughout the course including your IDE of choice. You are **not** allowed to use software utilizing AI, e.g., any form of copilot support.

You are allowed, unless otherwise stated, to use the .NET library including the modules described in the book, e.g., `List`, `Set`, `Map` etc.

If a subquestion requires you to define a particular function, then you may **use that function in subsequent subquestions**, even if you have not managed to define it yourself.

If a subquestion requires you to define a particular function, then you may **define as many helper functions as you want**, but in any case you must define the required function so that it has exactly the type and effect that the subquestion asks for.

The grading will favour functional solutions, i.e., solutions without side effects. Recursion is also favoured over loops. An imperative solution is preferred over no solution.

You should hand–in one file only, e.g., `ksfupr2024.fsx`. Do not use time on formatting your solution in Word or PDF.

You are welcome to use the accompanying file `may2024Snippets.fsx`. The file contains some of the code snippets included in the exam set for your convenience.

You must include explanations to support your solutions. You must demonstrate your solutions work by running the examples provided.

Your exam hand–in must be made by yourself and yourself only, and this holds for program code, examples, the explanations you provide for the code, and all other parts of the answers. It is illegal to make the exam answers as group work or to enlist the help of others in any way. This includes copying solutions from other resources and utilizing help from AI based tools.

Your hand–in must contain the following declaration:

I hereby declare that I myself have created this exam hand–in in its entirety without help from AI tooling and anybody else.

Question 1 (30%)

We consider problems that can be solved with recursion and tail–recursion. The concept of recursion is described in HR Chapter 1 and used throughout the book. Tail recursion is described in HR Chapter 9.

Question 1.1

It is allowed to use the collection library functions in this sub–question, as described in HR Chapter 5.

- Declare an F# function `noRemainderP` *m n* of type `int -> int -> bool` that returns `true` if *m* modulo *n* is 0, that is *m* `%` *n* is 0. For instance, `noRemainderP 10 2` evaluates to `true` and `noRemainderP 10 3` evaluates to `false`. The `noRemainderP` function is not recursive.
- Declare an F# function `checkNumber` *n m* of type `int -> int -> bool` that evaluates to `true` if `noRemainderP` *m l* is true for all $1 \leq l \leq n$; otherwise `checkNumber` evaluates to `false`. For instance, `checkNumber 10 2520` evaluates to `true` and `checkNumber 11 2520` evaluates to `false`.
- Declare a tail–recursive F# function `untilTrue` *f* of type `(int -> bool) -> int`, that repeatedly evaluates the function *f* on the integers $n \geq 1$ until the first n is found where *f* evaluates to `true`. The first n found is the result of the evaluation. For instance, `untilTrue (fun n -> n % 3 = 0)` evaluates to 3 because 1 % 3 is 1, 2 % 3 is 2 and 3 % 3 is 0.

  Explain why your solution is tail–recursive.

  Explain what happens if you evaluate `untilTrue (fun _ -> false)`.
- Declare an F# function `findSmallest` *n* of type `int -> int`, that evaluates to the smallest number M, that can be evenly divided by the numbers $1, \ldots, n$. That is, M can be divided with any number between 1 and n without any remainder. For instance `findSmallest 3` evaluates to 6, because 6 % 0 is 0, 6 % 1 is 0, 6 % 2 is 0, and 6 % 3 is 0. Also, the numbers 1 to 5 are not solutions, e.g., 1%2 is 1, 2%3 is 1, 3%2 is 1, 4%3 is 1 and 5%3 is 2.

  Hint: Can be implemented using `untilTrue` and `checkNumber`.

Question 1.2

It is **not** allowed to use any collection library functions in this sub–question.

- Declare an F# function `revAppend` *xs ys* of type `'a list -> 'a list -> 'a list` that appends the reverse of the list *xs* to the list *ys*. That is, `revAppend [`x_1`; ...; `x_N`] [`y_1`; ...; `y_M`]` evaluates to the list `[`x_N`; ...; `x_1`; `y_1` ...;`y_M`]`. For instance, `revAppend [1;2] [3;4]` evaluates to `[2;1;3;4]`.
- Declare an F# function `average` *xs* of type `float list -> float` that evaluates to the average of the values in the list *xs*. That is, `average [`x_1`; ...;`x_N`]`, evaluates to $\frac{x_1+\cdots+x_N}{N}$. The function `average` must evaluate to 0.0 in case the list xs is empty. For instance, `average [1.0;2.0]` evaluates to 1.5 and `average []` evaluates to 0.0.
- Declare an F# function `maxBy` *f xs* of type

  `('a -> 'b) -> 'a list -> 'a when 'b: comparison`

  that evaluates to the element x_i in the list *xs* where $f x_i$ evaluates to a larger value than any other $f x_j$ for $i \neq j$ and x_j also in the list *xs*. That is, evaluates to an element x_i in *xs*, where $f x_i \geq f x_j$ for any x_j in *xs*. The function should fail with message "maxBy: empty list" in case *xs* is empty. For instance, `maxBy ((+)1) [1..10]` evaluates to 10.

Question 1.3

Consider the F# function `collect` *f xs* of type `('a -> 'b list) -> 'a list -> 'b list`.

```
let rec collect f = function
    [] -> []
  | x::xs -> f x @ collect f xs
```

The function applies *f* on each element *x* in *xs* and appends each result list. For instance, `collect id [[1;2];[3;4]]` evaluates to `[1;2;3;4]`.

- Explain why the function `collect` is not a tail–recursive function.
- Declare a tail–recursive version of `collect`. The function must have same type and effect.
  Hint: The function `revAppend` may be useful.