FPSE Fall 2024

Please read all of the following items before beginning the quiz.

- You may assume that `Core` is opened above every block of code in this quiz.
- We hope and expect that the provided type signatures and examples completely describe the expected functionality.
- For discussion questions, your answers should be only one or two sentences. Provide brief answers only. We are not looking for lengthy justification.
- If your answer signficantly overflows the provided space, you might be on the wrong track. Use the extra sheets only if necessary.
- You are not to use mutation anywhere in any coding question. Coding answers using mutation will receive zero points.

I affirm that I have completed this quiz without unauthorized assistance from any person, materials, or device.

Signed: _____

Print name: _____

(Please print your name clearly so that Gradescope can recognize it.)

Date: 20 Nov 2024

Distribution of Marks

Question	Points	Score
1	9	
2	10	
3	16	
4	15	
Total:	50	

1. Coding in a functional language doesn't just mean we can't use mutation. There are some good things about functional languages, too! We get perks like first-class functions, algebraic data types, etc. Let's discuss some of the key aspects of functional programming and OCaml. Keep your answers brief and to the point. If you are too lengthy in your response, we might miss your point.

   (a) (3 points) What is a first-class function?

   (b) (3 points) What is the sequencing operator "`;`" (semicolon), and when is it used? Write a very short, one or two line program that uses it.

   (c) (3 points) What is one reason we might have a type `t` like the following instead of just `type t = string`?

```
type t = Ident of string
```

2. (10 points) Functors are a characteristic feature of OCaml, allowing polymorphism and dependent typing at the module level. We found them useful in Assignment 7 to generate functions that depended on a monad's type and implementations of `return` and `bind`.

   You'll write a simple functor in the same spirit of Assignment 7. We're giving you two module types, and you will write the implementation of a functor that satisfies the module type `F`.

```
module type M = sig
  type 'a t
  val return : 'a -> 'a t
  val extract : 'a t -> 'a
end

module type F = functor (_ : M) -> sig
  include M
  val lower : ('a t -> 'b t) -> ('a -> 'b)
end
```

   Now, write some functor below that has the signature `F`.

3. (16 points) The `Result` module in OCaml is a monad. It has the functions `return` and `bind`, which are useful when we want to produce an error in the program without raising an exception. You're going to write your own version of the module and call it `MyResult`, which in addition has a `fold_result` function. See the module signature and code skeleton below, where instructions are written in comments.

   If expected behavior at first seems unclear, remember to follow a type-directed-programming approach.

```
module MyResult : sig
  type ('a, 'e) t
  val return : 'a -> ('a, _) t
  val bind : ('a, 'e) t -> ('a -> ('b, 'e) t) -> ('b, 'e) t
  val fold_result : 'a list -> 'acc -> ('acc -> 'a -> ('acc, 'e) t) -> ('acc, 'e) t
end = struct
  type ('a, 'e) t =
    | All_good of 'a
    | Except of 'e

  (* Write an implementation to return a value, i.e. put it in "monad land". *)
  let return (x : 'a) : ('a, _) t =




  (* Write `bind` to apply `f` to `x`, or propagate the error if that
    application is not possible. *)
  let bind (x : ('a, 'e) t) (f : 'a -> ('b, 'e) t) : ('b, 'e) t =




  (* Write `fold_result` to fold through a list until `f` gives an error, at which
    point the error will propagate to the end. You must implement this without pattern
    matching on `t`, and instead, you must use `bind`. Feel free to use `let%bind`. *)
  let fold_result (ls : 'a list) (init : 'acc) ->
          (f : 'acc -> 'a -> ('acc, 'e) t) : ('acc, 'e) t =




end
```

4. (15 points) Functional linked lists have a constant time `cons` operation, but it is $O(n)$ time to get the length of the list because we must traverse the list until its end. Let's mend that.

   You will write a list module `Ls` to fix this problem, where all functions provided by the module are constant time. Credit is not awarded if the required time complexity is not met. Remember that you cannot use mutation.

```
module type L = sig
  type 'a t
  val empty : _ t
  val hd : 'a t -> 'a option
  val tl : 'a t -> 'a t option
  val cons : 'a -> 'a t -> 'a t
  val length : _ t -> int
end

module Ls : L = struct
  (* fill in with your answer below, where all functions are constant time *)









end
```

Use this page for scratch work or for your continued answers if you ran out of space. Clearly indicate if your work is an answer to a question in the quiz.

Use this page for scratch work or for your continued answers if you ran out of space. Clearly indicate if your work is an answer to a question in the quiz.