CSCI 2041: Advanced Programming Principles
Second Mid-Term Exam
April 6, 2015, 13:25 - 14:15 p.m.

Before doing anything else, please put aside all notes, books and electronic devices. This is a **closed book** exam. Also, please turn off your cell phones if you have them with you in class.

There are four questions in this exam all of which must be answered for full credit. The exam paper consists of 12 pages. Before reading any further, please make sure you have all of the pages.

Before you begin to write anything, please print your name and university id (i.e. **your university email address**) legibly in the space provided below and then check your lab section time. Please also have your student id ready for inspection during the exam and as you turn in your exam paper.

Name:

University Id (Email Address):

Lab Section Time: ( ) 9:05-9:55 ( ) 10:10-11:00 ( ) 11:15-12:05 ( ) 12:20-13:10 ( ) 13:25-14:15

Now, here are some guidelines that you should follow in doing the exam:

- Read all the questions through first to decide on a preferred order for doing them in.
- *Write your answers to the questions in the space provided in the exam paper.* Typically, you should not need more space. However, if you do need extra space
  - continue your work into the extra sheet at the end of the exam,
  - indicate under the problem that you are continuing the answer onto the extra space, and
  - in the extra space at the end, indicate *clearly* each problem that you are continuing there.
- Use only one side of each sheet, i.e., *do not write on the back of any page.*
- Use a number 2 pencil or pen for writing your answers. Also, write legibly—we cannot grade what we cannot read.
- Where you are asked to explain the reasons for your answer, note that the explanation is *crucial*; you will get very little credit without the explanation.

Good luck in the exam!

Problem I *(7 + (5+3) + (4+6) points)*

Recall the definitions of the functions `accumulate` and `reduce` that we saw in class:

```
let rec accumulate f lst u =
   match lst with
   | [] -> u
   | (h::t) -> accumulate f t (f h u)

let rec reduce f lst u =
   match lst with
   | [] -> u
   | (h::t) -> f h (reduce f t u)
```

1. We want to use `reduce` to define a function `pairwith` that takes an item `x` and a list of items `lst` and produces a new list in which `x` is paired with each item from `lst`. Your task is to provide an OCaml expression *that does not use any library functions* that can be plugged into the blank space in the definition shown below to yields such a function.

   ```
    let pairwith x lst = reduce ____  lst []
   ```

   Just to be clear about what `pairwith` should do, here are some example uses:

   ```
   # pairwith "string" [1;2;3];;
   - : (string * int) list = [("string", 1); ("string", 2); ("string", 3)]
   # pairwith 5 [1;2;3];;
   - : (int * int) list = [(5, 1); (5, 2); (5, 3)]
   # pairwith 5 [];;
   - : (int * 'a) list = []
   #
   ```

   Provide the expression below that should be used to complete the definition of `pairwith`:

2. The *difference* between two sets A and B is the set of all the elements of A that are not elements of B. Assume that we represent sets using lists in which each item has at most one occurrence. Your task is to provide expressions that can be plugged into the blank spaces below to yield a function `difference` that computes the difference between two sets.

```
difference set1 set2 = reduce ____ set1 ____
```

In filling in the blanks, you may use the function `member: 'a -> 'a list -> bool` that returns `true` if the first argument appears in the list that constitutes the second argument and `false` otherwise. Do not use any other library functions.

Just to be clear, some example uses of `difference` are shown below:

```
# difference [1; 2] [4; 7; 14; 1];;
- : int list = [2]
# difference [1; 2] [1;3;4;2];;
- : int list = []
# difference ["a"; "good"; "string"] ["just"; "string"];;
- : string list = ["a"; "good"]
#
```

Indicate what should be written in the first blank space in the definition:

Now indicate what should be written in the second blank space:

3. Suppose that we present OCaml with the following let declaration:

```
let whatever =
   (reduce (fun w x -> (accumulate (fun y z -> y + z) w 0) * x)
           [[1;2;3]; [4;5;6]; [1;2;3]]
           1)
```

   (a) What type will OCaml infer for `whatever`?

   (b) What will the declaration bind `whatever` to? If you think it is a function of some kind, you should explain what the function will do.

[This space is left blank so as to start the next problem on a new page]

Problem II *(15 points)*

Without using any previously defined functions, define the function

```
filter_pair : ('a -> 'b -> bool) -> ('a list) -> ('b list) -> ('a * 'b) list
```

that takes a boolean valued function and two lists and produces a list of pairs of corresponding elements in the two lists that are related by the input function. If the two lists are of unequal length, the extra elements in the longer list are ignored. Some example uses of this function that illustrate the desired behaviour:

```
# filter_pair (fun x y -> x < y) [1; 7; 3; 14] [2; 6; 8; 12];;
- : (int * int) list = [(1, 2); (3, 8)]
# filter_pair (fun x y -> x < y) [1; 7; 3; 14] [2; 6; 8];;
- : (int * int) list = [(1, 2); (3, 8)]
# filter_pair (fun x y -> x < y) [] [2; 6; 8];;
- : (int * int) list = []
#
```

Problem III. *(12 + 17 points)*

Transform the functions shown below into tail-recursive form by using continuations in the manner discussed in class and also done in Homework 4. You **must** follow that process to get credit.

1. This part concerns the *insert* function on binary trees. The type for binary trees and the function are both shown below.

```
type 'a btree =
  | Empty
  | Node of 'a * 'a btree * 'a btree

let rec insert t i =
   match t with
   | Empty -> Node (i,Empty,Empty)
   | Node (i',l,r) ->
        if (i < i') then Node (i',insert l i,r)
        else Node (i',l,insert r i)
```

   For this part, you should provide a definition below for the function

```
   cont_insert : 'a btree -> 'a -> ('a btree -> 'b) -> 'b
```

2. In this part you will transform the Fibonnaci function on positive numbers that is given as follows:

```
let rec fib n =
  match n with
  | 1 -> 1
  | 2 -> 1
  | n -> fib (n-1) + fib (n-2)
```

For this part, you should provide a definition below for the function

```
cont_fib : int -> (int -> 'a) -> 'a
```

Problem IV. *((2 + 4) + (3 + 6) + (3 + 4 + 9) points)*

Given below are the definitions of two familiar OCaml functions:

```
let rec append l1 l2 =
   match l1 with
   | [] -> l2
   | (h::t) -> h :: (append t l2)

let rec length lst =
   match lst with
   | [] -> 0
   | (h::t) -> (length t) + 1
```

In this problem, we are going to use induction on lists to show that *append* is length preserving, a property expressed precisely as follows:

$$\forall l_1 \in ('a\ list).\ \forall l_2 \in ('a\ list).\ length\ (append\ l_1\ l_2) = (length\ l_1) + (length\ l_2). \qquad (*)$$

Recall that in such propositions = means that the left and right sides have the same termination behaviour and that they evaluate to the same value if they terminate.

Note: I used the long names `append` and `length` above because those are the names we have been using for these functions, but it is okay for you to shorten these names to `app` and `len` in writing your answers below.

1. Inductive proofs begin with viewing the property to be proved as of the form $\forall l.\ P(l)$.

   (a) In the property marked (*), indicate which of the lists l_1 and l_2 you would like to base the induction on; this will be the $\forall l$ part when you write the property as $\forall l.P(l)$.

   (b) Based on your choice in the previous subpart, show below what $P(l)$ is.

2. In this part you will consider the base case of your argument using the formulation you have chosen in the first part of the problem.

   (a) Write down below the property you have to prove in the base case.

   (b) Provide an argument below for why the base case must hold. You must justify each of your steps for credit.

3. In this part you will consider the inductive step in the argument.

   (a) Write below the induction hypothesis for your argument.

   (b) Write below what you have to prove to complete the inductive step.

   (c) Provide an argument for why what you have written down subpart (b) must hold. You must justify each of your steps for credit.

Continue your answer onto this page if the space provided below some question is not sufficient for your answer. *Make sure to indicate clearly which answer is being continued here.*

This additional page has been provided as scratch area