First Exam
CS 1103 Computer Science 1 Honors

Fall 2022

Thursday October 6, 2022
Instructor Muller

KEY

Please write your name **on the back** of this exam.

You are free to use one 8.5 by 11 sheet of notes.

- Partial credit will be given so be sure to show your work.
- Feel free to write helper functions if you need them.
- **Please write neatly.**

Problem	Points	Out Of
Part 1		6
Part 2		14
Total		20

Part 1 (6 Points): Short Answer

1. (1 Point) Is the following well-formed? If not, what's wrong? If so, indicate the type and show the step-by-step simplification.

```
let pair = let n = 4 + 8 in (n, n) in pair :: [24]
```

Answer: This is ill-formed because the list elements must be of the same type.

2. (1 Point) Is the following well-formed? If not, what's wrong? If so, indicate the type and show the step-by-step simplification.

```
let x = (let y = not(x) in false || y) in not(x && x)
```

Answer: This is ill-formed because x cannot be used in the definition expression.

3. (2 Points) Let's say we have the following type definition `type t = {a : int; b : bool}`. Is the following well-formed? If not, what's wrong? If so, indicate the type and show the step-by-step simplification.

```
let r = {a = 2+3; b = not(false)} in if r.b then r.a + 10 else 20
```

Answer: This is well-formed.

```
let r = {a = 2+3; b = not(false)} in if r.b then r.a + 10 else 20 ->
let r = {a = 5; b = not(false)} in if r.b then r.a + 10 else 20 ->
let r = {a = 5; b = true} in if r.b then r.a + 10 else 20 ->
if {a=5; b=true}.b then {a=5; b=true}.a + 10 else 20 ->
if true then {a=5; b=true}.a + 10 else 20 ->
{a=5; b=true}.a + 10 ->
5 + 10 ->
15
```

4. (2 Points) Let's say we have the following definition `let double n = n * 2`. Is the following well-formed? If not, what's wrong? If so, indicate the type and show the step-by-step simplification.

```
(match (double 4) != 8 with | true -> (double 4) | false -> (double 5)) + 3
```

Answer: This is well-formed.

```
(match (double 4) != 8 with | true -> (double 4) | false -> (double 5)) + 3 ->
(match (4 * 2) != 8 with | true -> (double 4) | false -> (double 5)) + 3 ->
(match 8 != 8 with | true -> (double 4) | false -> (double 5)) + 3 ->
(match false with | true -> (double 4) | false -> (double 5)) + 3 ->
(double 5) + 3 ->
(5 * 2) + 3 ->
10 + 3 ->
13
```

Part 2 (14 Points): Writing Functions

Do both of problems 1. and 2. and any remaining problems adding up to **exactly** 12 points. Feel free to use solutions to one problem in solutions to subsequent problems. **Please circle the numbers of the problems you wish to be graded.**

1. (1 Point) Write a function `isEven : int -> bool` such that a call `(isEven n)` returns `true` if `n` is even and `false` if `n` is odd.

   Answer:

   ```
   let isEven n = n mod 2 = 0
   ```

2. (1 Point) The built-in function `Char.code` returns the ASCII code of a character. The call `(Char.code '0')` evaluates to 48, the call `(Char.code '1')` evaluates to 49 and so forth. Write a function `digitToInt : char -> int` so that a call `(digitToInt n)` returns the integer represented by the digit. For example, the call `(digitToInt '0')` should return 0 and the call `(digitToInt '3')` should return 3. You can assume that this function will only be used on digit characters.

   Answer:

   ```
   (* digitToInt : char -> int *)
   let digitToInt digit = (Char.code digit) - (Char.code '0')
   ```

3. (3 Points) Write a function `omitOdds : int list -> int list` such that a call `(omitOdds ns)` returns a list like `ns` but in which all of the odd numbers have been removed. For example, the call `(omitOdds [1; 2; 3; 4; 5])` should return the list `[2; 4]`.

   Answer:

   ```
   (* 3: omitOdds : int list -> int list *)
   let rec omitOdds ns =
     match ns with
     | [] -> []
     | n :: ns ->
       let almost = omitOdds ns
       in
       if isEven n then
         n :: almost
       else
         almost

   let omitOdds ns = List.filter isEven ns

   let omitOdds = List.filter isEven
   ```

4. (3 Points) Write a function `doubleTreble : int list -> int list` so that a call `(doubleTreble ns)` returns a list like `ns` but in which all of the odd numbers have been multiplied by 2 and all of the even numbers have been multiplied by 3. For example, the call `(doubleTreble [1; 2; 3; 4; 5])` should return the list `[2; 6; 6; 12; 10]`.

Answer:

```
(* 3: doubleTreble : int list -> int list *)
let rec doubleTreble ns =
  match ns with
  | [] -> []
  | n :: ns ->
    if isEven n then
      (n * 3) :: doubleTreble ns
    else
      (n * 2) :: doubleTreble ns
```

5. (3 Points) Write a function `concatAll : string list -> string` such that a call `(concatAll strings)` returns the string obtained by concatenating all of the strings in `strings` together using OCaml's string concat operator `^`. The resulting string should introduce a space between the words. For example, the call `(concatAll [])` should evaluate to `""`. The call `(concatAll ["Alice"; "Bob"])` should evaluate to the string `"Alice Bob"`. Note that there is no trailing space.

Answer:

```
(* concatAll : string list -> string *)
let rec concatAll strings =
  match strings with
  | [] -> ""
  | [only] -> only
  | string :: strings -> string ^ " " ^ concatAll strings
```

6. (3 Points) Let's say we're keeping track of students with a type

```
type student = { name : string
               ; year : int
               }
```

Write a function `studentNames : student list -> int -> string list` such that a call

```
(studentNames students n)
```

returns a list of the names of the students graduating in year `n`. For example, given the list

```
let students = [ {name="Alice"; year=2025};
                 {name="Bob";   year=2026};
                 {name="Mei";   year=2025}
               ]
```

The call `(studentNames students 2025)` should return the list of strings `["Alice"; "Mei"]`.

Answer:

```
(* studentNames : student list -> int -> string list *)
let rec studentNames students year =
  match students with
  | [] -> []
  | student :: students ->
    if student.year = year then
      student.name :: studentNames students year
    else
      studentNames students year
```

7. (3 Points) Write a function `every : ('a -> bool) -> 'a list -> bool` in such a way that a call `(every test xs)` returns `true` if the testing function `test` returns `true` for every element of `xs`. For example, the call `(every isEven [2; 4; 5])` should return `false` because `5` isn't even. The call `(every test [])` should return `true`.

Answer:

```
(* every : ('a -> bool) -> 'a list -> bool *)
let rec every test xs =
  match xs with
  | [] -> true
  | x :: xs -> test x && every test xs
```

8. (6 Points) The function `Lib.explode : string -> char list` converts a string to a list of characters. For example, a call `(Lib.explode "342")` returns the list `['3'; '4'; '2']`. Write a function `atoi : string -> int` such that a call `(atoi digits)` returns the integer represented by `digits`. For example, the call `(atoi "342")` should return the integer `342`. You may assume that this function will only be called with strings made up of digits.

Answer:

```
(* atoi : string -> int *)
let atoi string =
  let digits = Lib.explode string in
  let rec loop digits result =
    match digits with
    | [] -> result
    | digit :: digits ->
      let digit = digitToInt digit
      in
      loop digits (10 * result + digit)
  in
  loop digits 0
```

9. (6 Points) The famous Fibonacci sequence is the infinite sequence of integers

```
1, 1, 2, 3, 5, 8, ...
```

The first two Fibonacci numbers are 1 and from the third number on, the number is obtained by adding the previous two. For $n > 0$, the nth Fibonacci number can be computed using the wildly inefficient code

```
let rec fib n =
  match n < 3 with
  | true  -> 1
  | false -> (fib (n - 2)) + (fib (n - 1))
```

Can you write an efficient version?

Answer:

```
(* fib : int -> int *)
let fib n =
  let rec loop a b n =
    match n = 1 with
    | true  -> a
    | false -> loop b (a + b) (n - 1)
  in
  loop 1 1 n
```