ASSIGNMENT 5 — FUNCTIONAL PROGRAMMING

COMP 3010 — ORGANIZATION OF PROGRAMMING LANGUAGES

1. Lisp & Scheme

Exercise 1. Consider the Scheme definition:

```
(define (mystery x y)
  (lambda (z) (x (y (x z)))))
```

(1) Translate the `mystery` function to a λ-calculus expression. *HINT:* You will need λs introducing the parameters x and y around the body of `mystery`.

(2) What happens when you evaluate

```
(mystery (lambda (x) (+ 1 x)) (lambda (y) (* 2 y)))
```

(3) What happens when you evaluate

```
((mystery (lambda (x) (+ 1 x)) (lambda (y) (* 2 y))) 5)
```

(4) What happens when you evaluate

```
(((mystery (lambda (x) (+ 1 x)) (lambda (y) (* 2 y))) 5) 6)
```

Exercise 2. Remember the `map` function, which changes every element of a list using a given operation, is written in Scheme as

```
(define (map change-car xs)
  (cond [(null? xs) xs]
        [(cons? xs) (cons (change-car (car xs))
                          (map change-car (cdr xs)))]))
```

so that a list `(list x y ... z)` is transformed like so

```
(map f (list x y ... z)) = (list (f x) (f y) ... (f z))
```

`reduce`, which compresses a list by replacing every `cons` with a chosen binary operation and the final `null` with a chosen constant, is written in Scheme as

```
(define (reduce change-cons change-null xs)
  (cond [(null? xs) change-null]
        [(cons? xs) (change-cons (car xs)
                                 (reduce change-cons change-null (cdr xs)))]))
```

so that a list `(cons x (cons y (cons ... (cons z null))))` is transformed as

```
(reduce f e (cons x (cons y (cons ... (cons z null)))))
= (f x (f y (f ... (f z e))))
```

(1) What is the result of evaluating

```
(map (lambda (x) (* x x)) (list 1 2 3 4 5))
```

(2) What is the result of evaluating

```
(reduce + 0 (list 1 2 3 4 5))
```

(3) What is the result of evaluating

```
(reduce + 0 (map (lambda (x) (* x x)) (list 1 2 3 4 5)))
```

Date: Spring 2023.

(4) (Multiple Choice) Consider this definition of the function `f`:

```
(define (f xs)
  (reduce + 0 (map (lambda (x) (* x x)) xs)))
```

Which of the following alternate definitions of `f` is equivalent to the one above that used `map` and `reduce`?

(a)
```
(define (f xs)
  (cond [(null? xs) 0]
        [(cons? xs) (* (+ (car xs) (car xs)) (f (cdr xs)))]))
```
(b)
```
(define (f xs)
  (cond [(null? xs) 0]
        [(cons? xs) (+ (* (car xs) (car xs)) (f (cdr xs)))]))
```
(c)
```
(define (f xs)
  (cond [(null? xs) 0]
        [(cons? xs) (+ (car xs) (f (cdr xs)))]))
```
(d)
```
(define (f xs)
  (cond [(null? xs) 0]
        [(cons? xs) (* (car xs) (f (cdr xs)))]))
```

2. ML Family

Exercise 3. Do *Concepts In Programming Languages* Exercise 5.3 on Nonlinear Pattern Matching (page 123).

Note, for parts (a) and (b), you can write the described functions in SML syntax as asked by the exercise, *OR* in your choice of Ruby, Python, or C syntax.

Exercise 4. Do *Concepts In Programming Languages* Exercise 5.7 on Disjoint Unions (page 125).

Exercise 5. In SML, *all* references must point to real values in the heap. In other words, SML does not support implicit null pointers in place of a reference. Instead, the SML data type declaration

```
datatype 'a option = NONE | SOME of 'a;
```

defines the generic type `'a option` of references which could *either* point to nothing (represented by the `NONE` constructor containing no data) *or* point to some actual `'a` in the heap (represented by the `SOME` constructor containing a value of type `'a`).

For example, the integer division operation `x div y` will raise an exception when the divisor `y` is 0. A safe version of division, which never raises an exception, can be written in SML as

```
fun safe_div(x, 0) = NONE
  | safe_div(x, y) = SOME(x div y);
```

which takes a pair of `int`s and returns an `int option`.

(1) What is the difference between the result of evaluating `10 div 0` versus `safe_div(10, 0)`?
(2) What is the difference between the result of evaluating `10 div 5` versus `safe_div(10, 5)`?
(3) What happens when you try to evaluate `2 * (10 div 5)`? What happens when you try to evaluate `2 * (safe_div(10, 5))`?