CS154: Paradigms for Programming Lab

End-Sem Exam (2 hours; 25 marks)

November 15^{th}, 2023

Instructions:

- Save each solution in a file `Qi.ext`, with the appropriate extension. Enclose all files in a folder `rollnum`, where `rollnum` is your roll number. Create and upload a zip that contains this folder on Moodle.
- You can refer to your notes, but Internet usage is prohibited.

THE BRACKETY LANGUAGE

Q1 [3]. If f is a numerical function and n is a positive integer, then we can form the n^{th} repeated application of f, which is defined to be the function whose value at x is $f(f(...(f(x))...))$. For example, if f is the function $x \to x + 1$, then the n^{th} repeated application of f is the function $x \to x + n$. If f is the operation of squaring a number, then the n^{th} repeated application of f is the function that raises its argument to the 2^n-th power. Write a procedure that takes as inputs a procedure that computes f and a positive integer n and returns the procedure that computes the n^{th} repeated application of f. Your procedure should be able to be used as:

```
> ((repeated square 2) 5)
625
```

Q2 [4]. Write a procedure `stream-limit` that takes as arguments a stream and a number (the `tolerance`). It should examine the stream until it finds two successive elements that differ in absolute value by less than the tolerance, and return the second of the two elements. Verify that your procedure works by using it to compute square roots up to a given tolerance as follows:

```
(define (sqrt-improve guess x)
  (average guess (/ x guess)))
(define (sqrt-stream x)
  (define guesses
    (cons-stream 1.0 (stream-map (lambda (guess) (sqrt-improve guess x)) guesses)))
  guesses)
(define (sqrt x tolerance)
  (stream-limit (sqrt-stream x) tolerance))
```

For your convenience, standard stream constructs are defined in a file `streams.scm` on Moodle.

Q3 [5]. The procedures `+`, `*`, and `list` take arbitrary numbers of arguments. One way to define such procedures is to use `define` with *dotted-tail notation.* In a procedure definition, a parameter list that has a dot before the last parameter name indicates that, when the procedure is called, the initial parameters (if any) will have as values the initial arguments, as usual, but the final parameter's value will be a *list* of any remaining arguments. For instance, given the definition

```
(define (f x y . z) <body>)
```

the procedure `f` can be called with two or more arguments. If we evaluate

```
(f 1 2 3 4 5 6)
```

then in the body of `f`, `x` will be `1`, `y` will be `2`, and `z` will be the list `(3 4 5 6)`.

Use this notation to write a procedure `same-parity` that takes one or more integers and returns a list of all the arguments that have the same even-odd parity as the first argument. For example:

```
> (same-parity 1 2 3 4 5 6 7)
(1 3 5 7)
> (same-parity 2 3 4 5 6 7)
(2 4 6)
```

THE TINY LANGUAGE

Q4 [2]. Write a predicate `oddSum(L,N)` that captures the condition that the sum of the odd elements contained in a list `L` is `N`.

Q5 [2]. Given a list containing an arbitrary number of occurrences of three symbols `r`, `b` and `w`, write a predicate `arrange(L, M)` that rearranges the list `L` in `M` such that all the `r`s precede all the `w`s precede all the `b`s.

THE LAZY LANGUAGE

Q6 [3]. Give a definition of the function

```
disjoint :: (Ord a) => [a] -> [a] -> Bool
```

that takes two lists in ascending order, and determines whether or not they have an element in common.

Q7 [3]. Consider a `grid` function that generates a *coordinate* grid of all numbers in a range:

```
grid m n = [(x,y) | x <- [0..m], y <- [0..n]]
```

Using a list comprehension and the `grid` function above, define a function `square :: Int -> [(Int,Int)]` that returns a coordinate square of size n, excluding the diagonal from $(0, 0)$ to n, n. For example:

```
> square 2
[(0,1),(0,2),(1,0),(1,2),(2,0),(2,1)]
```

Q8 [3]. Complete the following *insertion sort* implementation:

```
sort [] = <??>
sort <??> = insert x (sort xs)
insert x <??> = [x]
insert <??> = if <??>
                then x:y:ys
              else y:<??>
```

NOT THE END YET!