Problem 1: (10 points) (two points each) (mean: 7.3, median: 7, 3rd quartile: 10)

What is the type of each of the following Haskell expressions? If an expression is invalid, briefly state why.

- `show 10`  
  `String`

- `[1, 'x']`  
  Invalid—a list with a mix of types

- `head "tail"`  
  `Char`

- `zip`  
  `[a] -> [b] -> [(a, b)]`

- `replicate 5 "a"`  
  `[String]`

Problem 2: (7 points) (mean: 6.2, median: 7, 3rd quartile: 7)

Write functions `f1`, `f2`, and `f3` having each of the following types.

- `f1 :: a -> b -> a`  
  `f1 x y = x`

- `f2 :: a -> b -> [(b,a)]`  
  `f2 a b = [(b,a)]`

- `f3 :: a -> [a] -> [a]`  
  `f3 a b = a:b`

Problem 3: (16 points) (mean: 15.8, median: 16, 3rd quartile: 16)

*Without using any higher-order functions*, write a function `largers` that takes a pair of equal-length lists, `L1` and `L2`, and returns a list `R` of the larger values, position by position.

```haskell
> largers [7,3,9,5] [4,6,2,5]
[7,6,9,5]
```

Solution:

```haskell
largers [] [] = []
lagers (a:as) (b:bs)
    | a > b = a : largers as bs
    | otherwise = b : largers as bs
```
Problem 4: (16 points) (mean: 14.1, median: 15, 3<sup>rd</sup> quartile: 16)

Write a function `chunks` that breaks a list into a series of chunks whose lengths are specified by the elements of `lengths`.

```haskell
> chunks [1..9] [3,2,3,1]
[[1,2,3],[4,5],[6,7,8],[9]]
```

Solution:

```haskell
chunks _ [] = []
chunks list (n:ns) |
  length list >= n = take n list : chunks (drop n list) ns
  otherwise = []
```

Problem 5: (16 points) (mean: 14.4, median: 15, 3<sup>rd</sup> quartile: 16)

Write a function `printsums` that takes a list of `Int` lists and prints, one-per-line, the position of each list in turn (one-based!) and the sum of that list's elements.

```haskell
> printsums [[3,4],[9],[],[3,1,2]]
1: 7
2: 9
3: 0
4: 6
```

If the list is empty, print "Empty!":

```haskell
> printsums []
Empty!
```

Solution:

```haskell
printsums [] = putStrLn "Empty!\n"
printsums lists = putStrLn result
  where
    pairs = zip [1..] (map sum lists)
    result = unlines $ map f pairs

    f (n, sum) = show n ++ " : " ++ show sum
```
Problem 6: (15 points) (mean: 12.2, median: 13, 3rd quartile: 14)

Without writing any recursive code, write a function `addRs` that takes a non-empty string and between each pair of characters inserts either '<' or '>' depending on the ordering relationship between the two characters.

```haskell
> addRs "sorted"
"s>o<r<t>e>d"
```

Solution:

```haskell
addRs string = foldr f [] string

f c [] = [c]
f c acm@(h:_)
| c < h = c:'<':acm
| otherwise = c:'>':acm
```

Problem 7: (10 points) (two points each) (mean: 8.5, median: 9, 3rd quartile: 10)

The following questions and problems are related to Haskell.

1. As taught in this class, what's the recommended order of preference for if-then-else, guards, and patterns?
   patterns, guards, if-then-else

2. What would be awkward about implementing `zip` using folding?
   Folding fundamentally operates on one list but `zip` operates on two lists. You might first `zip` the two lists together and then fold that list of tuples, but that seems pretty silly!
   However, we'll see that Racket generalizes folds to N lists.

3. Rewrite the following function using guards:
   ```haskell
   f x = if x > 3 then 'a' else 'b'
   ```
   ```haskell
   f x
   | x > 3 = 'a'
   | otherwise = 'b'
   ```

4. If a function `f` has the type `String -> Int -> Char` and is called with `f "testing"` what is the type of the value produced? (If it's an error, explain why.)
   ```haskell
   Int -> Char
   ```

5. Is it possible to fold a list of `Bool` values into a list of `(String, [Int])` tuples? Briefly justify your answer.
   ```haskell
   This does it: foldr (\_ acm -> acm) ["x", [1::Int]] [True]
   ```
Problem 8: (10 points) (two points each) (mean: 7.7, median: 8, 3rd quartile: 10)

Briefly answer the following general questions.

(1) In any language you know, write an expression that has both a value and a side-effect. Cite the language, too.
Java: \texttt{x = 1}

We saw a lot of mistakes like those we saw on Quiz 2. See Sam's Piazza post \texttt{@34} for a discussion of them.

A number of students wrote \texttt{i++;} for Java or C. We didn't count it wrong but adding the semicolon turns the expression \texttt{i++} into a statement. Section 14.8 of the Java Language Specification identifies \texttt{i++;} as an \textit{expression statement}.

(2) With programming languages in general, what's the fundamental difference between a statement and an expression?
An expression can be evaluated to produce a value, assuming no error. An expression might have a side effect or not. Statements don't produce a value and unless they have a side effect, they serve no purpose. Note that with \texttt{gcc} on lectura, the line of code \texttt{3+4;}
produces a warning:
\texttt{x.c:8:5: warning: statement with no effect}
\texttt{8 | 3+4;}
With Java, it's an error.

(3) Cite a design decision related to Java's \texttt{+} operator.
My first thought was,
"Adding" a \texttt{String} and a primitive type produces a \texttt{String},
but many things were cited for this.

(4) What does the syntax of a programming language specify?
The sequences of characters that are valid programs in the language.

(5) What does the semantics of a programming language specify?
The meaning of a sequence of characters.
Extra Credit Section (½ point each unless otherwise noted) (mean: 2.3, median: 2.5, 3rd quartile: 3)

(1) Martin came across the following Haskell expression on the net. Briefly, what is it doing?

( . ) . ( . )

It's composing two composition operators!

(2) What's the key aspect of folding that Mr. Mitchell admits he didn't "get" when first teaching functional programming?

I didn't understand that a list of anything can be folded into a completely different type.

(3) Write zip in Python.

```python
def myzip(a,b):
    result = []
    for i in range(min(len(a),len(b))):
        result.append((a[i],b[i]))
    return result
```

A number of students used a recursive solution, and that was fine.

A few students used result[i] = ... rather than append. That happens to work in Ruby, not in Python, but we accepted that answer.

As I write this it crosses my mind that one student used a list comprehension and we counted that correct but at the moment I can't see how that can be made to work. Grading is an imperfect process.

(4) What is the title or who is the author of the book that allegedly caused the word "paradigm" to come into popular use as in "programming paradigms"?

Thomas Kuhn wrote The Structure of Scientific Revolutions.

(5) We started the class with the quote, “A language that doesn't affect the way you think about programming is not worth knowing.” Who said it?

Alan J. Perlis

(6) Write a Haskell function `samelen list1 list2` that returns `True` iff `list1` and `list2` are the same length. The only function you may use in your solution is `samelen`. (1 point)

```haskell
samelen [] [] = True
samelen (x:xs) (y:ys) = samelen xs ys
samelen _ _ = False
```

A few students used operators, like `==`, but operators are simply functions bound to symbolic names.

(7) Write a good extra credit question related to the course material and answer it. (1 point)

Among other things, a number of questions about Ralph Griswold were posed.
Statistics

Here are all 58 scores, in descending order.

104.00, 103.00, 101.50, 101.00, 100.00, 100.00, 100.00, 99.50, 99.50, 99.50, 98.50, 96.50, 96.50, 96.50, 96.50, 96.00, 96.00, 95.50, 94.00, 93.50, 93.50, 92.50, 91.50, 91.00, 91.00, 91.00, 91.00, 91.00, 90.50, 90.00, 90.00, 90.00, 90.00, 89.50, 89.00, 88.50, 88.50, 87.00, 86.00, 85.50, 85.50, 85.50, 84.00, 83.50, 83.50, 83.00, 82.50, 81.50, 79.00, 79.00, 79.00, 78.50, 77.50, 74.50, 74.00, 73.00, 57.00, 56.50, 51.00

Mean: 88.37
Median: 90.5
3rd Quartile: 96.5

Here's a table with per-problem statistics. Median/possible shows per-problem median scores divided by possible points, expressed as a percentage.

<table>
<thead>
<tr>
<th>exprtypes</th>
<th>ftypes</th>
<th>lagers</th>
<th>chunks</th>
<th>printsums</th>
<th>addrS</th>
<th>Haskell S/A</th>
<th>Gen S/A</th>
<th>EC</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>7.3</td>
<td>6.2</td>
<td>15.8</td>
<td>14.1</td>
<td>14.4</td>
<td>12.2</td>
<td>8.5</td>
<td>7.7</td>
<td>2.3</td>
</tr>
<tr>
<td>Median</td>
<td>7</td>
<td>7</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>13</td>
<td>9</td>
<td>8</td>
<td>2.5</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>10</td>
<td>7</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>14</td>
<td>10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Median/possible</td>
<td>70.0</td>
<td>100.0</td>
<td>100.0</td>
<td>93.8</td>
<td>93.8</td>
<td>86.7</td>
<td>90.0</td>
<td>80.0</td>
<td>55.6</td>
</tr>
</tbody>
</table>

CSC 372 Spring 2023 Midterm Exam