Your last name

____

Last name of classmates beside you (or "wall"/ "aisle")

On my left: ____________ On my right: _______________

CSC 372 Midterm Exam
Wednesday, March 15, 2023

READ THIS FIRST

Read this page now but do not turn this page until you are told to do so. Go ahead and fill in the boxes above with your last name and the last names of any classmates sitting beside you.

This is a 65-minute exam with a total of 100 points of regular questions and an extra credit section.

The last five minutes of the exam is a "seatbelts required" period, to avoid distractions for those who are still working. If you finish before the "seatbelts required" period starts, you may turn in your exam and leave. If not, you must stay quietly seated—no "packing up"— until time is up for all.

You are allowed no reference materials whatsoever.

If you have a question, raise your hand. We will come to you. DO NOT leave your seat.

If you have a question that can be safely resolved with a minor assumption, like the name of a function or the order of function arguments, state the assumption and proceed.

Feel free to use abbreviations, like "`otw`" for "`otherwise`".

It's fine to use helper functions or predicates unless they are specifically prohibited or a specific form for the function or predicate is specified.

Don't make a programming problem hard by assuming that it needs to do more than is specifically mentioned in the write-up or that the solution that comes to mind is "too easy."

If you're stuck on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that's a sure zero.

It is better to put forth a solution that violates stated restrictions than to leave it blank—a solution with violations may still be worth partial credit.

When told to begin, double-check that your name is at the top of this page, and then **put your initials in the lower right-hand corner of the top side of each sheet, checking to be sure you have all five sheets.**

BE SURE to enter your last name on the sign-out log when turning in your completed exam.

Problem 1: (10 points) (two points each)

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

Assume integers have the type `Int`. Remember that `String` is a type synonym for `[Char]`— the two can be used interchangeably.

Important: Remember that the type of a function includes the type of the arguments as well as the type of the value returned.

```
show 10


[1,'x']


head "tail"


zip


replicate 5 "a"
```

Problem 2: (7 points)

This problem is like `ftypes.hs` on `a3`. Write functions `f1`, `f2`, and `f3` having each of the following types. There are no restrictions other than you may not use explicit type declarations. (e.g. `f1::...`) The functions are worth two, two, and three points, respectively.

```
f1 :: a -> b -> a


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


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

Problem 3: (16 points)

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.

That is, the first element of `R` is the larger of the first element in `L1` and `L2`; the second element of `R` is the larger of the second element in `L1` and `L2`, etc.

Example:

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

> largers "turnip" "caveat"
"tuvnit"

> largers [] []
[]
```

4

Problem 4: (16 points)

Write a function `chunks list lengths` of type `[a] -> [Int] -> [[a]]` that breaks `list` into a series of chunks whose lengths are specified by the elements of `lengths`.

Assume all lengths are greater than zero.

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

> chunks "abcdefghi" [3,2,3,1]
["abc","de","fgh","i"]

> chunks "abcdefghi" [3,2]
["abc","de"]

> chunks "abcd" []
[]
```

If a length is encountered that's longer than what remains, `chunks` produces no further lists:

```
> chunks "abcd" [3,2,1]
["abc"]

> chunks "abcd" [5,2,1]
[]
```

My solution is recursive and uses `take` and `drop`.

Problem 5: (16 points)

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. Examples:

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

> printsums [[],[1,1,1],[]]
1: 0
2: 3
3: 0
```

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

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

My solution uses `zip` and `sum`.

6

Problem 6: (15 points)

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.

Example:

```
> addRs "abc"
"a<b<c"
```

The resulting string shows that `'a' < 'b'` and `'b' < 'c'`.

Two more examples:

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

> addRs "c"
"c"
```

As a simplification, assume characters are never equal.

My solution uses `foldr`.

7

Problem 7: (10 points) (two points each)

<u>The following questions and problems are related to Haskell.</u>

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

(2) What would be awkward about implementing `zip` using folding?

(3) Rewrite the following function using guards:

```
f x = if x > 3 then 'a' else 'b'
```

(4) If a function `f` has the type `String -> Int -> Char` and is called with

```
f "testing"
```

what is the <u>type</u> of the value produced? (If it's an error, explain why.)

(5) Is it possible to fold a list of `Bool` values into a list of `(String,[Int])` tuples? Briefly justify your answer.

Problem 8: (10 points) (two points each)

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.

(2) With programming languages in general, what's the fundamental difference between a statement and an expression?

(3) Cite a design decision related to Java's + operator.

(4) What does the syntax of a programming language specify?

(5) What does the semantics of a programming language specify?

Extra Credit Section (½ point each unless otherwise noted)

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

```
(.) . (.)
```

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

(3) Write `zip` in Python.

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

(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?

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

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