Your last name

____

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

On my left: ____________ On my right: _______________

CSC 372 Midterm Exam
Wednesday, November 2, 2022

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 seven sheets.**

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

Problem 1: (5 points) (one point each)

What is the **type** of each of the following Haskell expressions? If the 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.

```
[1..3]
```

```
('x',['x'])
```

```
length
```

```
tail "head"
```

`map isDigit`   (Recall that `isDigit '9'` returns `True`.)

Problem 2: (6 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::...`)

```
f1 :: [a] -> Int
```

```
f2 :: a -> b -> c -> [b]
```

```
f3 :: [(a, b)] -> ([b], [c])
```

Problem 3: (4 points, as indicated)

This problem is like `warmup.hs` on the assignments—write the following Haskell Prelude functions.

Instances of poor style or needlessly using other Prelude or helper functions will result in deductions.

Be sure to use the wildcard pattern (underscore) when appropriate.

There's no need to specify function types.

`tail` [1 point] (Assume the list is never empty.)

`last` [1 point] (Assume the list is never empty.)

`zip` [2 points] (Remember that the length of the shorter list is the length of the result.)

DID YOU REMEMBER TO USE WILDCARDS WHEN APPROPRIATE?

Problem 4: (18 points) (8 points each + 2 points for types)

4

For this problem you are to **write both recursive and non-recursive versions** of a Haskell function `mkintlists` that takes a list of strings of digits and returns a list of `Int`s with corresponding values.

Example:

```
> mkintlists ["315","91", "", "713"]
[[3,1,5],[9,1],[],[7,1,3]]

> mkintlists []
[]
```

In the recursive version **ONLY**, use `error "even!"` to produce an error if an even digit is encounted:

```
> mkintlists ["31","12"]
*** Exception: even!
```
(Non-recursive version would produce `[[3,1],[1,2]]`*).*

Assume that strings consist only of decimal digits and that there is a `digitToInt` function:

```
> digitToInt '7'
7
```

Just like on assignments 3 and 4, the recursive version must not use any higher-order functions, and just like on assignment 5, write the non-recursive version imagining that you just don't know to how to write a recursive function.

BEFORE writing your two versions of `mkintlists`, what is the type of...

`mkintlist`?

`digitToInt`?

5

Problem 5: (15 points)

Without writing any recursive code, write a Haskell function `vcnp :: String -> Int` that counts the number of vowels in a string that are not immediately preceded by a vowel. Vowels may be in upper or lower case. Examples:

```
> vcnp "ate"
2

> vcnp "aeiou"
1

> vcnp "Oopsie!"
2

> vcnp "retreating"
3

> vcnp "a-e-i-o-u-A-E-I-O-U"
10
```

Recall :

(1) `value `elem` list` returns `True` iff `value` is an element of `list`.

(2) `toLower` converts letters to lower case and leaves non-letters unchanged.

HINT: My solution contains this fragment of code:

```
... foldl f ('x',0) ...
```

Problem 6: (17 points)

Write a Haskell function `coords rows cols` with type `Int -> Int -> IO ()` that PRINTS row and column coordinates for a grid with the given number of rows and columns. Example:

```
> coords 3 4
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1) (2,2) (2,3)
```

Assume you have a function `mkrc :: Int -> Int -> String` that creates a `"(`*`row`*`,`*`col`*`)"` string from row and column coordinates:

```
> mkrc 5 3
"(5,3)"
```

Two more examples: (blank line added before second call for readability)

```
> coords 2 2
(0,0) (0,1)
(1,0) (1,1)

> coords 3 1
(0,0)
(1,0)
(2,0)
```

Assume `rows` and `cols` are both greater than zero. **There are NO RESTRICTIONS on this problem.**

Problem 7: (5 points) (one point each unless otherwise indicated)

The following questions and problems are related to Haskell.

(1) The Haskell expression below has more parentheses than are needed! Mark **ALL** the parentheses that can be removed without changing the value of the expression.

```
(take 3) (show x) ++ (replicate 5) (chr 66)
```

(2) Given the type of a function, how can we quickly tell if the function is polymorphic?

(3) Add parentheses to the type below to fully show the right-associativity of the `->` type operator.

```
Int -> [Int] -> (Char -> Bool -> Bool) -> String
```

(4) Briefly explain how the following `map` works, paying particular attention to the function being mapped. (That function is the result of **`(uncurry $ flip replicate)`**.)

```
> map (uncurry $ flip replicate) [('a',3),('b',2)]
["aaa","bb"]
```

(5) Consider the Haskell expression below. Does it have any partial applications? If so, briefly describe what constitutes each of the partial applications.

```
map (take 5)
```

Problem 8: (7 points)

Using our DIY cons lists, write a Prolog predicate `eqalen/2` that succeeds iff its two arguments are lists that consist entirely of atoms **and** the length of all atoms in corresponding positions are equal. Examples:

```
?- eqalen(a:test:now:empty, i:went:too:empty).
true.        (Atoms in both lists have lengths of 1, 4, and 3, respectively.)

?- eqalen(a:test:now:empty, i:went:too:far:empty).
false.       (Four atoms in second list.)

?- eqalen(a:test:now:empty,a:failure:now:empty).
false.       (Length of second atom differs.)

?- eqalen(a:[b]:c:empty,a:b:c:empty).
false.       (First list contains a non-atom.)
```

Recall that `atom_length(+Atom, -Length)` can be used to get the length of an atom, and `atom/1` can be used to test for an atom.

RESTRICTION: The symbol = may NOT APPEAR in your solution! (This rules out == and \==, too, for example.)

Problem 9: (12 points)

Write a Prolog predicate `lines(+Pairs, +Separator)` with this behavior:

```
?- lines(three-x:five-ok:four-oops:empty,'.').
x.x.x
ok.ok.ok.ok.ok
oops.oops.oops.oops
true.
```

`Pairs` is a DIY cons list with "pairs" of the form *`EnglishNumber-Atom`*. For each pair, a line with `EnglishNumber` repetitions of `Atom` is printed with `Separator` between the atoms.

If `lines` is run with no arguments, it prints a usage message:

```
?- lines.
Usage: lines(+Pairs,+Sep)
true.
```

Two more examples follow.

```
?- lines(four-four:five-[]:empty,<>).
four<>four<>four<>four
[]<>[]<>[]<>[]<>[]
true.

?- lines(three-xx:empty,'X').
xxXxxXxx
true.
```

Assume that you have a predicate `n/2` with twenty facts like these:

```
n(one,1). n(two,2). n(three,3). ...  n(twenty,20).
```

Assume that there is at least one element in `Pairs` and that each pair specifies 1-20 replications.

Assume that you have `member_cl`:

```
?- member_cl(X,three-x:five-ok:four-oops:empty).
X = three-x ;
X = five-ok ;
X = four-oops ;
false.
```

Space for your solution is provided on the next page, but you can write it below if you wish.

Space for solution for `lines(+Pairs, +Separator)`.

For reference:

```
        ?- lines(three-x:five-ok:four-oops:empty,'.').
        x.x.x
        ok.ok.ok.ok.ok
        oops.oops.oops.oops
        true.

        ?- lines.
        Usage: lines(+Pairs,+Sep)
        true.
```

Assume that `lines` is used properly, i.e., don't worry about handling any errors.

BE SURE that `lines` always succeeds!

Problem 10: (5 points) (one point each unless otherwise indicated)

The following questions and problems are related to Prolog.

(1) `x(3,4)` is an example of a Prolog structure. Without using any parentheses, commas, or square brackets, write another example of a Prolog structure.

(2) Prolog anatomy question: What are the three parts of a Prolog rule? (½ point)

(3) What are three distinct ways in which `length/2` can be used? (1½ points)

(4) What is the **output** of the following query?

```
?-  between(1,3,A), writeln(A), A > 5,
     between(5,7,B), writeln(B), B < 10,
     writeln('Done!').
```

(5) How do the following two goals differ in meaning?

```
A+B == 5

A+B =:= 5
```

Problem 11: (6 points) (one point each unless otherwise indicated)

Briefly answer the following general questions.

(1) Who founded The University of Arizona's Computer Science department and when?

(2) What is the value and side effect of the following Python expression?

```
print("testing")
```

(3) What's the one feature/property that a language must have in order to do anything that even remotely resembles functional programming?

(4) Which <u>one</u> of the following language elements is most essential for imperative programming AND why? (2 points)

(a) some sort of looping construct like a `while` or a `for`
(b) an assignment operation
(c) some sort of "print" statement
(d) procedures

(5) Early in the semester we talked about three aspects of expressions that are often important to understand and reason about. What are those three aspects?

Extra Credit Section (½ point each unless otherwise noted)

(1) Collectively, the body of facts and rules that implement a Prolog predicate is known as the _______________ for the predicate.

(2) Sadly, I never got around to answering this Haskell puzzle posed on 284:

Make the list `[take, tail, init]` valid by adding two characters.

Answer it now!

(3) What's a fundamental difference between using `>>> type(x)` in the Python shell and using `> :type x` in `ghci`?

(4) Write our beloved `map` function in Python. (1 point)

(5) Several places in the Haskell slides mention "H10". Example: "Lambda abstraction (H10)". What is H10?

(6) What would be a big simplification in the following Haskell code?

```
    g list = foldl1 (\acm elem -> f acm elem) list
```

(7) Write a non-imperative version of C's `strcpy(...)` function. (If you haven't had 352 nor are taking it now, plead ignorance for a half-point!)

(8) Once `whm` decided he should teach Racket instead of Ruby, he held onto Racket like a monkey holding onto ______________________!

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