1 Introduction

The purpose of this assignment is to improve your skills writing Haskell functions over lists.

- For the purposes of this assignment, unless otherwise stated, don’t use any of the built-in library functions. — I want you to write all functions “from scratch”! Simple list manipulation functions (head, tail, :, ++, length) are OK to use, of course.
- Unless otherwise specified, all functions should use Haskell’s pattern or guard syntax, not the if-then-else syntax!
- You may freely introduce auxiliary functions if that makes your program cleaner. Also, feel free to introduce local definitions (where-clauses) to make your code easier to read.
- You will be graded primarily on correctness and style, not on the execution efficiency of your code.
- Functions written for one problem may be freely used in subsequent problems. In fact, this is encouraged!
- All functions must be commented. At the very least, each function should start out with a description of what it does, what input parameters it takes, what output it produces, and an example of how it is invoked.
- All functions must have a function signature.

2 Grading Problems

1. The final grade assignment for a student in this class with a total score of \( t \) and final exam score of \( f \) is given by the formula

\[
\text{grade} = \begin{cases} 
E & \text{if } f < 50 \\
A & \text{if } t \in [90, 100] \\
B & \text{if } t \in [80, 89] \\
C & \text{if } t \in [70, 79] \\
D & \text{if } t \in [60, 69] \\
E & \text{if } t < 60
\end{cases}
\]

[5 points]

Code this formula in Haskell:
grade :: Float -> Float -> Char
grade f t ...

> grade 45.0 85.0
'E'
> grade 90.0 85.0
'B'
> grade 90.0 105.0
program error: illegal score
> grade (-5.0) 99.0
Program error: illegal score

Scores greater than 100.0 or less than 0.0 should generate a illegal score error message.

2. Write a function member x ys which returns True if x is an element of ys, False otherwise. [5 points]

member :: Float -> [Float]-> Bool
member x xs ...

> member 4 [1,2,3]
False
> member 4 [1,2,3,4.0,5]
True

3. Write a function maxl xs that generates an error "empty list" if xs==[] and otherwise returns the largest element of xs: [5 points]

maxl :: [Float] -> Float
maxl xs = ...

> maxl [2,3,4,5,1,2]
5.0
> maxl []
Program error: empty list

4. Write a function mull xs m which returns a new list containing the elements of xs multiplied by m: [5 points]

mull ::[Float] -> Float -> [Float]
mull xs x = ...

> mull [1,2,3,4,5] 0.0
[0.0,0.0,0.0,0.0,0.0]
> mull [1,2,3,4,5] 2
[2.0,4.0,6.0,8.0,10.0]

5. Write a function setsub xs ys that takes two integer lists xs and ys as input, both lists representing sets. In other words, xs and ys are unsorted lists that contain no duplicate elements. setsub xs ys returns xs-ys, i.e. the list containing the elements in xs that are not in ys: [10 points]

2
setsub :: [Float] -> [Float] -> [Float]
setsub xs ys = ...

> setsub [1,2,3] []
[1.0,2.0,3.0]
> setsub [1,2,3] [3]
[1.0,2.0]
> setsub [1,2,3] [1,2,3]
[]
> setsub [] [1,2,3]
[]
> setsub [] []
[]

6. In computing the grades for this class, each individual score (final, midterm, quizzes, assignments) will be curved by multiplying each score by \(100.0/2\text{nd}\_\text{highest}\_\text{class}\_\text{score}\). \([15\text{ points}]\)
This can be computed by the formula
\[
\min(100, (100.0/ \max(x - \max(x))))s
\]
where \(-\) is set-subtraction, \(x\) is the set of grades, and \(s\) is the score of a particular student.
Use \texttt{maxl}, \texttt{mul}, and \texttt{setsub} from the problems above to write a function \texttt{curve xs} that scales the scores in \(xs\).
For example, assume the following final exam scores:

\[
34\ 45\ 66\ 88\ 98
\]

After the curve has been applied, the scores will be

\[
38.6\ 51.1\ 75\ 100\ 100
\]

Note that no score can be higher than 100.0.

\texttt{curve :: [Float] -> [Float]}
\texttt{curve xs = ...}

> curve [34,45,66,88,98]
[38.63636,51.13636,75.0,100.0,100.0]

For this problem you don’t have to check for illegal scores.
Depending on which computer you run this on the result can vary slightly.

7. Define a recursive function \texttt{histogram xs} which takes a list of grades as input (pairs of \(\text{final, total}\)) and returns a string which is a histogram of the grades. You don’t have to check for illegal scores.
Example: \([15\text{ points}]\)

\[
> \text{putStr (histogram [(60,90),(50,38),(100,100),(100,100),(34,80),(92,92)])}
'A' ***** (4)
'B' (0)
'C' (0)
'D' (0)
'E' ** (2)
\]
Each * represents one student who got a particular grade.

`putStr` prints out a string on the console. You may need to use the built-in `show` function, which converts a number/character/etc. to a string:

```haskell
> show 6
"6"
> show '6'
"'6'"
> show 6.6
"6.6"
```

You will find that for a more complicated function like `histogram` you must break it up into several smaller functions. For example, you could organize your program like this:

```haskell
-- Return the number of students who got a particular
-- grade. For example,
--   > grades [(60,90),(50,38),(100,100),(100,100),(34,80),(92,92)] 'A'
-- 4
grades :: [(Float,Float)] -> Char -> Int
grades ((f,t):xs) g = ...

-- This is a helper function for histogram. It does the
-- actual work of generating the histogram. The first
-- argument is the list of grades that we are going to
-- print out the histogram for.
-- Example:
--   > putStr (hist "ABCDE" [(60,90),(50,38),(100,100),(100,100),(34,80),(92,92)])
--   'A' **** (4)
--   'B'   (0)
--   'C'   (0)
--   'D'   (0)
--   'E' ** (2)
hist :: [Char] -> [(Float,Float)] -> String
hist (g:gs) ft = ...

-- The main function to print out a histogram.
histogram :: [(Float,Float)] -> String
histogram ft = hist "ABCDE" ft

-- Generate a string consisting of n copies of c
stringDup Int -> String -> String
stringDup n c = ..
3 Computing Primes

Define a function `sieve xs` which computes a list of prime numbers using a Eratosthenes' sieve. For example, `sieve [2..10]` should return the list `[2,3,5,7]`. [20 points]

The method is as follows:

1. start with a list of numbers beginning with 2, for example `[2,3,4,5,6,7,8,9,10]`.
2. The first number in the list is prime. Remove all its multiples. In this case we get `[3,5,7,9]`.
3. Repeat the previous step: 3 is prime, and after 'sieving' out the multiples we are left with `[5,7]`.
4. Repeat the step again: 5 is prime, and sieving leaves `[7]`.
5. Do it again: 7 is prime, and sieving leaves `[]`.
6. When no numbers remain, we've found all the primes in the given range.

First implement a function `siever n xs` which returns the elements of `xs` that are not multiples of `n`:

```haskell
siever :: Int -> [Int] -> [Int]
siever n xs ... 
> siever 2 [3..10]
[3,5,7,9]
```

Hint: use this helper function:

```haskell
relprime p n = n `mod` p > 0
```

Using `siever`, implement the `sieve` function:

```haskell
sieve :: [Int] -> [Int]
sieve xs ... 
> sieve [2..10]
[2,3,5,7]
> sieve [2..20]
[2,3,5,7,11,13,17,19]
> sieve [2..] 
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67, 
71,73,79,83,89,97,101,103,107,109,113,127,131,137,139, 
149,151,157,163,167,173,179,181,191,193] {Interrupted!}
```
4 Sorting

The final problem is to implement a sorting algorithm.

1. Write a function `sort3` that can sort lists of length 3 or less: \[5 \text{ points}\]

   ```haskell
   sort3 :: [Float] -> [Float]
   sort3 xs = ...
   > sort3 []
   []
   > sort3 [42]
   [42.0]
   > sort3 [4,2]
   [2.0,4.0]
   > sort3 [4,2,1]
   [1.0,2.0,4.0]
   > sort3 [1,3,2]
   [1.0,2.0,3.0]
   > sort3 [1,3,2,4]
   Program error: can't sort more than 3 elements!!!
   
   For `sort3` you should not use recursion! Instead, implement it by performing the minimal number of comparisons. For example, for a two element list you only need to make one comparison, for a three element list you only need to make two or three comparisons. You can use any combination of guard syntax, if-then-else, or pattern syntax you like.

2. Write a function `merge xs ys` that merges two sorted lists into a sorted list: \[5 \text{ points}\]

   ```haskell
   merge :: [Float] -> [Float] -> [Float]
   merge xs ys = ...
   > merge [] []
   []
   > merge [1,2,3] []
   [1.0,2.0,3.0]
   > merge [] [1,2,3]
   [1.0,2.0,3.0]
   > merge [3,4,7] [1,2,3]
   [1.0,2.0,3.0,3.0,4.0,7.0]
   
   3. Write a function `msort xs` that returns the list `xs` sorted: \[5 \text{ points}\]

   ```haskell
   msort :: [Float] -> [Float]
   msort xs = ...
   > msort []
   []
   > msort [1]
   [1.0]
   > msort [2,1]
   [1.0,2.0]
   > msort [3,2,1]
   [1.0,2.0,3.0]
You should use the `mergesort` algorithm to implement `msort`. I.e. at every recursive step split `xs` into two halves (hint: use `take` and `drop`), sort them recursively using `msort`, and then use `merge` to construct a sorted list.

4. Construct a faster version of `msort` called `fsort` that uses the technique from `sort3` to treat lists of three or fewer elements specially. I.e. whenever `fsort` gets down to short lists it won’t call itself recursively but rather sort them directly. To measure the “time” (actually, number of operations) that `msort` and `fsort` take, start `hugs` with the `+s` option:

```haskell
$ hugs +s
> :load ass.hs
> msort [10,9..1]
[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0]
(1401 reductions, 1886 cells)
> fsort [10,9..1]
[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0]
(911 reductions, 1266 cells)
```

5 Submission and Assessment

The deadline for this assignment is noon, Tue Sep 22. It is worth 5% of your final grade.

You should submit the assignment electronically using d2l.

Don’t show your code to anyone, don’t read anyone else’s code, don’t discuss the details of your code with anyone. If you need help with the assignment see the instructor or the TA.