1 Introduction

The purpose of this assignment is to get started writing Haskell functions. For the purposes of this assignment, don’t use any of the higher-order built-in functions such as \texttt{map}, \texttt{foldr}, etc. — I want you to write all functions “from scratch”! You may use the ++-function for string concatenation.

Hint: It’s never wrong to introduce an auxiliary function when it makes the program easier to write or prettier to read! In fact, breaking up a larger function in two or more smaller ones is encouraged.

Unless otherwise specified, you should use the \textit{guard syntax} rather than the \texttt{if-then-else} syntax when you define recursive functions.

2 Simple Non-Recursive Functions

1. Define a function \texttt{doublestring \ s} which takes a string argument \texttt{s} and returns a new string consisting of two copies of \texttt{s}: [5 points]

\[
\texttt{doublestring :: String \to String} \\
\texttt{doublestring \ s = ...}
\]

\begin{verbatim}
> doublestring ""
"
> doublestring "hello"
"hellohello"
\end{verbatim}

2. Write a function \texttt{charToString \ a} which returns the one-character string consisting only of the character \texttt{a}: [5 points]

\[
\texttt{charToString :: Char \to String} \\
\texttt{charToString \ c = ...}
\]

\begin{verbatim}
> charToString 'A'
"A"
\end{verbatim}

3. Using the formula

\[
V = 2\pi rh + 2\pi r^2
\]

define a function \texttt{cylinderSurfaceArea \ (r, h)} which computes the surface area of a cylinder of height \texttt{r} and radius \texttt{h}, rounded down to the nearest integer: [5 points]

\[
\texttt{cylinderSurfaceArea :: (Float, Float) \to Int} \\
\texttt{cylinderSurfaceArea \ (r, h) = ...}
\]
cylinderSurfaceArea :: (Float,Float) -> Int
cylinderSurfaceArea (r,h) = ...

> cylinderSurfaceArea (2.0,5.0)
87

Use the \texttt{floor} function to round down and the \texttt{pi} constant function to approximate $\pi$:

> floor 5.5
5
> pi
3.14159

4. Use \texttt{head} and \texttt{tail} to write a non-recursive function \texttt{third xs} which returns the third element of a list of \texttt{Ints}:

\begin{verbatim}
third :: [Int] -> Int
third xs = ...
\end{verbatim}

> third [1,2,3,4,5]
3

You don’t have to check that the list contains at least three elements.

5. Use \texttt{head} and \texttt{tail} to write a non-recursive function \texttt{yahtzee xs} which takes a list of five \texttt{Ints} (between 1 and 6) as argument (the result of rolling five dice) and returns \texttt{True} if all numbers are the same, and \texttt{False} otherwise.

\begin{verbatim}
yahtzee :: [Int] -> Bool
yahtzee xs = ...
\end{verbatim}

> yahtzee [1,1,1,1,1]
True
> yahtzee [1,1,1,1,2]
False
> yahtzee [6,6,6,6,6]
True

You don’t have to check the input for correctness — we assume that there are exactly five elements in the list and that the numbers are between 1 and 6.

\section*{3 Simple Recursive Functions}

1. Write a recursive function \texttt{msum n} that returns the sum of the integers $0 + 1 + 2 + 3 + \ldots + n$, where $n \geq 0$:

\begin{verbatim}
msum :: Int -> Int
msum n = ...
\end{verbatim}

> msum 0
msum should make use of a *conditional expression* (*if-then-else* syntax).

2. Write a recursive function \( gsum \ n \) that returns the sum of the integers \( 1 + 2 + 3 + \ldots + n \), where \( n \geq 0 \):  

\[
gsum :: \text{Int} \to \text{Int} 
gsum \ldots
\]

\( gsum \) should make use of *guards*.

3. Define a recursive function \( \text{copystring} \ (s,n) \) which returns a string consisting of \( n \) copies of the string \( s \):  

\[
\text{copystring} :: (\text{String},\text{Int}) \to \text{String} 
\text{copystring} \ (s,n) \ldots
\]

Your function should have the following behavior:

\[
\begin{align*}
> \text{copystring} \ ("hello",-1) & \quad \Rightarrow \ "" \\
> \text{copystring} \ ("hello",0) & \quad \Rightarrow \ "" \\
> \text{copystring} \ ("hello",1) & \quad \Rightarrow \ "hello" \\
> \text{copystring} \ ("hello",2) & \quad \Rightarrow \ "hellohello" \\
> \text{copystring} \ ("hello",10) & \quad \Rightarrow \ "hellohellohellohellohellohellohellohellohellohellohellohellohellohellohellohellohello"
\end{align*}
\]

I.e., for any values of \( n \leq 0 \) \( \text{copystring} \) should return the empty string, for positive values it should return \( n \) copies of the string, concatenated together.

4. Write recursive function \( \text{numlist} \ n \) which generates a list of the integers \([0, 1, 2, 3, \ldots, n]\), where \( n \geq 0 \). Generate the error "illegal argument" for \( n < 0 \).  

\[
\text{numlist} :: \text{Int} \to [\text{Int}] 
\text{numlist} \ n \ldots
\]

\[
\begin{align*}
> \text{numlist} \ 0 & \quad \Rightarrow \ [0] \\
> \text{numlist} \ 4 & \quad \Rightarrow \ [0,1,2,3,4] \\
> \text{numlist} \ (-5) & \quad \text{program error: illegal argument}
\end{align*}
\]

5. Write a recursive function \( \text{allsame} \ xs \) which takes a list of \text{Ints} and returns \text{True} if all numbers are the same, and \text{False} otherwise.  

\[
\text{allsame} :: [\text{Int}] \to \text{Bool} 
\text{allsame} \ xs \ldots
\]
allsame :: [Int] -> Bool
allsame xs ... 

> allsame []
True
> allsame [1]
True
> allsame [1,2]
False
> allsame [1,1,1,1,1]
True
> allsame [1,1,1,1,2]
False

6. Write a recursive function swap xs which takes a list of Ints and returns and new list where pairs of adjacent elements have been swapped. [12 points]

swap :: [Int] -> [Int]
swap xs ...

> swap []
[]
> swap [1]
[1]
> swap [1,2]
[2,1]
> swap [1,2,3]
[2,1,3]
> swap [1,2,3,4]
[2,1,4,3]
> swap [1,2,3,4,5,6,7,8]
[2,1,4,3,6,5,8,7]

7. Write a recursive function split xs which takes a list of Ints and returns and tuple of two new list such that the odd numbered elements go in the first one, the even numbered elements in the second one. [13 points]

split :: [Int] -> ([Int],[Int])
split xs ...

> split []
([],[])
> split [1]
([1],[])
> split [1,2]
([1],[2])
> split [1,2,3]
([1,3],[2])
> split [1,2,3,4]
([1,3],[2,4])
> split [1,2,3,4,5,6,7,8,9,10]
([1,3,5,7,9],[2,4,6,8,10])
> split [10,1000,111,42,666,99,999,9999]
  ([10,111,666,999],[1000,42,99,9999])

Note: The function \texttt{fst} returns the first element of a tuple, \texttt{snd} returns the second one.

4 Submission and Assessment

The deadline for this assignment is noon, Thu Sep 10. It is worth 5\% of your final grade.

You should submit the assignment to \texttt{d2l.arizona.edu}.

\begin{center}
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.
\end{center}