The University of Hong Kong

COMP3258: Functional Programming

Assignment 1

Deadline: 23:59, Oct 4, 2022 (HKT)

Problem 1. (10 pts.) The Pentanacci sequence is, by definition, the integer sequence in which every number after the first five is the sum of the five preceding numbers. To simplify:

0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 31, 61, 120, 236, 464, 912, 1793, ...

Given a number n, write a recursive function `pentanacci :: Int -> Int` which returns n-th pentanacci number.

```
*Main> pentanacci 1
0
*Main> pentanacci 5
1
*Main> pentanacci 7
2
*Main> pentanacci 10
16
```

Problem 2. (10 pts.) You get `n` boxes of mooncakes and `m` friends (`n` larger than `m`). You want to make a Mid-Autumn Festival present with mooncakes to each friend. You should do this in the fairest (that is, most equal) manner and the friend who gets the most number should differ from the friend who gets the least number as little as possible.

Your task is to complete the function `solve :: Int -> Int -> [Int]`, where the output integer list represents the i-th friend's number of moon cakes.

```
*Main> solve 12 3
[4, 4, 4]
*Main> solve 15 4
[3, 4, 4, 4]
```

```
*Main> solve 18 7
[2, 2, 2, 3, 3, 3, 3]
```

Hint:

- The function `rem` returns a reminder of the integer division of the arguments.
- The function `quot` divides the first argument by the second one discarding the remainder.

Problem 3. (10 pts.) You got a namebook that stores many names. However, the names have several different formats. A name `Danial John Velleman` may be stored as:

- `Danial John Velleman`
- `Danial John`
- `Danial-John-Velleman`
- `Danial-John Velleman`

and so on. In general, there may be hyphens `'-'` and spaces `' '` together with the words in a name string. Please note that a name can consist of one or multiple words.

Please write a function `lookupName :: [String] -> String -> [String]`, which takes a name book as a list of strings, and a prefix of the name as a single string then returns all names that start with the prefix.

Notice:

- Assume all inputs are valid as below:
  - All names in the namebook contain only hyphens `'-'`, spaces `' '`, and words
  - The prefix contains only letters
  - Upper case and lower case letters are different prefixes (e.g., `Dani` is not the same as `dani`).
- If the prefix is empty, return all the names in the namebook
- Names that are shorter than the prefix should not be returned
- The names returned by your function should be well formatted. That is, every name only contains words and spaces, without hyphens. There must be a space between two words of a name in the output.

Expected running results:

```
*Main> lookupName ["Daniel-John"] "Dani"
["Daniel John"]
*Main> lookupName ["Daniel John-Velleman"] "Dani"
```

```
["Daniel John Velleman"]
*Main> lookupName ["Daniel John", "Daniel-John"] "Dani"
["Daniel John","Daniel John"]
*Main> lookupName ["Dani"] "Daniel"
[]
*Main> lookupName ["Daniel-John-Velleman"] ""
["Daniel John Velleman"]
*Main> lookupName ["Daniel John", "Daniel-John"] "dani"
[]
*Main> lookupName ["Daniel John", "Daniel-John"] "John"
[]
```

Problem 4. (10 pts.) In the first lecture, we show the elegant implementation of `quicksort` in Haskell. The comparison of elements in the list relies on the method specified in `Ord`. Here we consider a general version of `sort` which lets you use a predicate `p : a -> a -> Bool`. The type of `sortp` should be

```
sortp :: [a] -> (a -> a -> Bool) -> [a]
```

Try to implement it by using recursion.

Expected running results:

```
g :: Bool -> Bool -> Bool
g True True = True
g True False = True
g False True = False
g False False = False

*Main> sortp [True,False,True,False] g
[True,True,False,False]
*Main> sortp [1,2,3,4,5] (>)
[5,4,3,2,1]
```

Problem 5. (20 pts.) `foldl` is a function to fold containers (e.g., list) into a summary value. We show some examples of `foldl`:

```
foldl :: (b -> a -> b) -> b -> [a] -> b

*Main> foldl (+) 0 [1,2,3,4] -- ((((0 + 1) + 2) + 3) + 4)
10
*Main> foldl (-) 10 [4,3,2,1] -- ((((10 - 4) - 3) - 2) - 1)
-10
```

However, the `foldl` only accounts for finite lists. For example, `foldl (+) 0 [1..]` does not terminate. We want you to implement an improved version of `foldl` which accepts a predicate (if the summary value doesn't satisfy the predicate, the `foldl` terminates and returns the value) to deal with the infinite lists.

```
foldlp :: (b -> Bool) -> (b -> a -> b) -> b -> [a] -> b
```

```
*Main> foldlp (< 50) (+) 0 [1..]
45
```

Problem 6. (20 pts.) A list is defined to be twin paired if its even-valued elements (if any) are in ascending order and its odd-valued elements (if any) are in descending order.The list `[-6, 12, 5, 24, 3, 1]` is twin paired because the even-valued elements `[-6, 12, 24]` are in ascending order and the odd-valued elements `[5, 3, 1]` are in descending order. However, the list `[1, 2, 3]` is not twin paired because the odd numbers are not in descending order. Write a function named `isTwinPaired :: [Int] -> Bool` that returns `True` if its list argument is twin paired, otherwise, it returns `False`.

Expected running results:

```
*Main> isTwinPaired [2,4,32]
True
*Main> isTwinPaired [2,2,2,1,1,1]
True
*Main> isTwinPaired [1,19,23]
False
*Main> isTwinPaired [3,2,1]
True
*Main> isTwinPaired [20,22,24,27,25]
True
```

Problem 7. (20 pts.) You may be familiar with the infix notation in mathematical expressions. For instance, in `(7 - 2) * 3`, `-` and `*` are infix operators. Reverse Polish Notation (RPN) is another way to represent expressions. In such a postfix manner, we do not need to worry about precedence and parens. For instance, the `(7 - 2) * 3` will be transformed into `7 2 - 3 *`. The task of this question is to implement an RPN calculator in Haskell. The function you need to implement will be `solveRPN :: String -> Int`. And it is based on the following rules:

- The calculator only deals with integers.
- Four basic operators `+`, `-`, `*`, `/` should be supported.
- Two bitwise operators `&` (bitwise and), `|` (bitwise or) should be supported.
- Besides binary operations, your calculator should also support the unary ones: `inc` (`1 inc` returns 2), `dec` (`2 dec` returns 1).

- Usually, we use a stack to store the expressions parsed, you need to implement some stack operations: `dup` (push a duplicate of top element in the stack), `clear` (clear the stack)

Expected running results:

```
*Main> solveRPN ""
0
*Main> solveRPN "2 3 + 4 *"
20
*Main> solveRPN "2 inc 3 + 4 *"
24
*Main> solveRPN "2 3 & 1 +"
3
*Main> solveRPN "2 dup +"
4
*Main> solveRPN "2 3 + 4 * clear 2 dup +"
4
```

Hints

- To deal with bitwise operations, `Data.Bits` is your friend.
- `words` breaks a string up into a list of words, which were delimited by white space.
- The `read` function reads input from a string.

```
>>> read "123" :: Int
123
>>> read "hello" :: Int
*** Exception: Prelude.read: no parse
```

Code style and submission (5 pts.)

All functions should be implemented in a single Haskell file, named as `A1_XXX.hs`, with `XXX` replaced by your UID. Your code should be well-written (e.g. proper indentation, names, and type annotations) and documented. Please submit your solution on Moodle before the deadline.

Notice: there are cases that students cannot upload a Haskell file on Moodle. Then please compress it into `A1_XXX.zip`, which contains only one file `A1_XXX.hs`.

Plagiarism

Please do this assignment on your own; if, for a small part of an exercise, you use something from the Internet or were advised by your classmate, please mark and attribute the source in a comment. Do not use publicly accessible code sharing websites for your assignment to avoid being suspected of plagiarism.