MATHFUN
Discrete Mathematics and Functional Programming

Worksheet 2: Introduction to Functional Programming II

Introduction

The first part of this worksheet is designed to give you practice in writing function definitions, particularly using guards, but also possibly with local definitions. Put your solutions to these exercises into a file called week2.hs, which should be made available to show us for feedback during next week's practical session. The second part of the worksheet is intended to give you practice in evaluating expressions using calculations.

Programming exercises

1. Write a new version of your function:

   ```
   absolute :: Int -> Int
   ```

   from worksheet 1, this time using guards rather than an if ... then ... else.

2. Write a function:

   ```
   sign :: Int -> Int
   ```

   that returns 1 for positive arguments, -1 for negative arguments and 0 for zero-valued arguments.

3. Write a function:

   ```
   howManyEqual :: Int -> Int -> Int -> Int
   ```

   which determines how many of its three arguments are equal (i.e. it returns either 0, 2 or 3).

4. Write a function:

   ```
   sumDiagonalLengths :: Float -> Float -> Float -> Float
   ```

   which takes the side-lengths of three squares as its arguments, and returns the sum of the lengths of the squares' diagonals.

5. A taxi company calculates fares based on distance travelled. Fares start at £2.20; 50p is added for each kilometre covered for the first 10 kilometres; and 30p is added for each additional kilometre. Write a function:

   ```
   taxiFare :: Int -> Float
   ```

   which takes the distance in kilometres, and returns the fare in pounds.

6. Write a function:

   ```
   howManyAboveAverage :: Int -> Int -> Int -> Int
   ```

   which returns how many of its three integer arguments are greater than their average value. (Hint: First consider what the possible results could be.)

7. Write a function:

```
validDate :: Int -> Int -> Bool
```

   which takes integers representing a day and month, and returns True if, and only if, the date is valid. For example `validDate 29 3` gives True (since 29th March is a valid date), but `validDate 30 2`, `validDate 25 13` and `validDate (-4) 6` all give False. Assume February always has 28 days.

8. Assuming that all years divisible by 4 are leap years, write a function:

```
daysInMonth :: Int -> Int -> Int
```

   which returns the number of days in a given month and year (for example, `daysInMonth 2 2012` should be 29).

Written exercises

1. For your `sumThree` function from worksheet 1, give calculations that evaluate the following expressions:
   - `sumThree 3 5 7`
   - `sumThree 8 (1 + 3) 2`
2. For your `threeDifferent` function from worksheet 1, give calculations that evaluate the following expressions:
   - `threeDifferent 1 4 2`
   - `threeDifferent 1 7 7`
3. For your `howManyEqual` function from this worksheet, give calculations that evaluate the following expressions:
   - `howManyEqual 3 5 2`
   - `howManyEqual 5 2 5`