Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2020-2021

Programming Exam 1

Due date: 08 April 2021, Thursday, 23:59

1 Problem Definition

In this programming exam, you will help your friends who work in the IT department of a bank by implementing some `Haskell` functions. Each function will correspond to an independent, small feature which is useful for their customers or themselves in some certain calculations.

1.1 General Specifications

- The signatures of the functions, their explanations and specifications are given in the following section. Read them carefully.
- Make sure that your implementations comply with the function signatures.
- You may define helper function(s) as you needed.
- Importing any modules is not allowed for this exam. All you need is already available in the standard `Prelude` module.
- Whenever you need to output a `Double` value (included in a list or by itself). You MUST use the following function to round it to 2 decimal places:

```
getRounded :: Double -> Double
getRounded x = read s :: Double
               where s = printf "%.2f" x
```

This tricky implementation uses `printf` function from `Text.Printf` module. This import and the implementation itself is already included in the template file. You can use it directly, while outputting Double values. The whole purpose of using this function is the simplification of the output, which will be useful in both debugging and evaluation processes.

2 Functions

2.1 convertTL (15 points)

You will implement a function named `convertTL` which takes a `Double` value and a `String` representing the name of currency which the given money in Turkish Liras will be converted to. Here is the signature of this function:

```
convertTL :: Double -> String -> Double
```

The abbreviations as representing `Strings` and the exchange rates [1] for the currencies are given in Table 1. No other currency will be tested.

Currency	Abbreviation	Exchange Rate
US Dollar	"USD"	8.18
Euro	"EUR"	9.62
Bitcoin	"BTC"	473497.31

Table 1: Exchange Rates with TL for Currencies

The function must convert the money in TL to the given currency using the proper exchange rate and return it as a `Double`.

SAMPLE I/O:

```
*PE1> convertTL 8180 "USD"
1000.0

*PE1> convertTL 12500 "EUR"
1299.38

*PE1> convertTL 1500000 "BTC"
3.17
```

2.2 countOnWatch (15 points)

As someone must monitor the servers of the bank at each night, some schedule is arranged for your friends. Your task is to implement a function named `countOnWatch` which takes a `list of Strings` as the schedule of the turns, a `String` representing the name of a certain employee and an `Int` representing the some number of days. This function shall calculate the number of watches assigned to this employee up to given days according to the arranged schedule. The function must return the calculated number as an `Int`.

Here is the signature of this function:

```
countOnWatch :: [String] -> String -> Int -> Int
```

[1] I know that they look a little over-detailed, but let's keep them in their realistic values for the time being. After all we have the ability of copy/paste for such a burden.

SAMPLE I/O:

```
*PE1> countOnWatch ["Ali", "Huseyin", "Ahmet", "Derya", "Ali", "Fatma",
   "Ali", "Ahmet"] "Ahmet" 5
1

*PE1> countOnWatch ["Ali", "Huseyin", "Ahmet", "Derya", "Ali", "Fatma",
   "Ali", "Ahmet"] "Ali" 5
2

*PE1> countOnWatch ["Ali", "Huseyin", "Ahmet", "Derya", "Ali", "Fatma",
   "Ali", "Ahmet"] "Ali" 8
3
```

2.3 encrypt (35 points)

You will implement a function named `encrypt` which takes an `Int` with 4 digits as the password of the customer and returns an `Int` as the encrypted password by processing the digits in the original one.

Here is the signature of this function:

```
encrypt :: Int -> Int
```

We will assume that the customers are not allowed to use a zero in any place for their password. Here are the rules to encrypt the rest of the digits:

- If the digit is divisible by `3`, you will `subtract 1 from the digit`.
- If the digit is divisible by `4`, you will `multiply the digit by 2`.
- If the digit is divisible by `5`, you will `add 3 to the digit`.
- For the rest of the cases, you will `add 4 to the digit`.
- If any of the above operation results in a double digit number, you will `take the number in the ones place` *(e.g. 6 for 16.)*.

SAMPLE I/O:

```
*PE1> encrypt 6475
5818

*PE1> encrypt 9812
8656

*PE1> encrypt 3597
2881
```

2.4 compoundInterests (35 points)

You will implement a function named `compoundInterests` which calculates the money the customers would have after the compound interests are applied according to their initial money and how many years they want to invest their money to the bank.

Rules for the calculation of the total money that each customer would get will be given below. Let's look at the signature of this function first:

```
compoundInterests :: [(Double, Int)] -> [Double]
```

This function takes a `list of (Double, Int) pairs` as the money and number of years for the investment of each customer. The function returns a `list of Double` as the total money that each customer would have for their investment. We will assume that there will be always at least one customer, when this function is called.

As the bank has the policy to compound the money monthly, here is the formula for the calculation of total money with compound interest:

$$T = P \times (1 + \frac{R}{12})^{12 \times N} \tag{1}$$

where T is the total money, P is the initial money, R is the annual interest rate and N is the number of years. We will use Table 2 to calculate annual interest rate (R) for the given initial money (P).

	Money $\geq$ 10000	**Money $<$ 10000**
Years $\geq$ 2	11.5%	9.5%
Years $<$ 2	10.5%	9.0%

Table 2: Annual Interest Rates

Let's say that a customer initially has 15000 TL, and wants to keep their money for 1 year to get the interest. Then we must use the annual interest rate as 10.5%. Hence, the calculation is done as below:

$$15000 \times (1 + \frac{0.105}{12})^{12 \times 1} \tag{2}$$

By rounding the result to 2 decimal places, we get 16653.05 as the total money that customer would have after 1 year.

SAMPLE I/O:

```
*PE1> compoundInterests [(15000, 1)]
[16653.05]

*PE1> compoundInterests [(17557.5, 2), (9800, 1)]
[22073.68,10719.31]

*PE1> compoundInterests [(1575.25, 3), (12500, 1), (100000, 2)]
[2092.36,13877.54,125722.25]
```

3 Regulations

1. **Implementation and Submission:** The template file named "pe1.hs" is available in the Virtual Programming Lab (VPL) activity called "PE1" on `OdtuClass`. At this point, you have two options:

   - You can download the template file, complete the implementation and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
   - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submit a file.

   The second one is recommended. However, if you're more comfortable with working on your local machine, feel free to do it. Just make sure that your implementation can be compiled and tested in the VPL environment after you submit it.

   There is no limitation on online trials or submitted files through OdtuClass. The last one you submitted will be graded.

2. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

3. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to consider the invalid expressions.

   Important Note: The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official testcases to determine your ***actual*** grade after the deadline.