Nome: GUILHERME LUCAS PEREIRA BERNARDO

LISTA DE EXERCÍCIO

1. **Contagem de Dígitos**: Diana escreverá uma lista com todos os inteiros positivos entre A e B, inclusive, na base decimal e sem zeros à esquerda. Ela quer saber quantas vezes cada um dos dígitos irá ser usado.

   Entrada

   Cada caso de teste é dado uma entrada com dois inteiros A e B.

   Saída

   Para cada caso de teste, imprima uma única linha com 10 inteiros representando o número de vezes que cada dígito é usado dentro do intervalo de inteiros entre A e B. Escreva a contagem de cada dígito em ordem crescente do 0 até o 9.

   R:

```
import Text.Printf(printf)


contRockeiro:: Int->String->Int

contRockeiro numToSearch number = let found = [head(show(numToSearch))
== x|x <- number]

                                      onlyFound = [x | x <- found, x]

                                  in length onlyFound


questao1 val1 val2 = let lista = [val1..val2]

                         recur0 = sum [contRockeiro 0 (show x)|x<-lista]

                         recur1 = sum [contRockeiro 1 (show x)|x<-lista]

                         recur2 = sum [contRockeiro 2 (show x)|x<-lista]

                         recur3 = sum [contRockeiro 3 (show x)|x<-lista]

                         recur4 = sum [contRockeiro 4 (show x)|x<-lista]

                         recur5 = sum [contRockeiro 5 (show x)|x<-lista]

                         recur6 = sum [contRockeiro 6 (show x)|x<-lista]

                         recur7 = sum [contRockeiro 7 (show x)|x<-lista]

                         recur8 = sum [contRockeiro 8 (show x)|x<-lista]

                         recur9 = sum [contRockeiro 9 (show x)|x<-lista]

                     in
[recur0,recur1,recur2,recur3,recur4,recur5,recur6,recur7,recur8,recur9]
```

2. **Conversão entre Bases**: O professor de matemática de Juliano marcou uma prova cujo conteúdo será apenas conversão entre valores decimais, hexadecimais e binários. Uma das coisas mais complexas para Juliano é fazer estas conversões de base entre números. Por mais que estude, tem muita dificuldade para entender. Portanto, como você entende de

computação e é amigo(a) de Juliano, ele solicitou a tua ajuda para que faça um programa que verifique se as conversões feitas por ele estão correta.

Entrada

A entrada contém vários casos de teste. A primeira linha da entrada contém um inteiro N, indicando o número de casos de teste que virão a seguir, um por linha. Cada caso de teste contém um valor X (X > 0) seguido de um texto Y com três caracteres, indicando se o valor X está no formato binário, decimal ou hexadecimal. Independente do formato, qualquer dos números deverá caber em um inteiro de 32 bits.

Saída

Para cada caso de teste, você deve apresentar o número de caso de teste seguido por duas linhas, que contém a conversão do valor fornecido para as outras duas bases. A sequência das bases de saída será sempre: decimal, hexadecimal (em minúsculo) e binário, ou seja deve-se respeitar esta ordem excluindo obviamente o formato de entrada.

Exemplo:

Exemplo de Entrada	Exemplo de Saída
3 101 bin 101 dec 0f hex	Case 1: 5 dec 5 hex Case 2: 65 hex 1100101 bin Case 3: 143 dec 10001111 bin

R:

```
import Numeric (showHex, showIntAtBase)
import Data.Char (intToDigit)
import Text.Printf (printf)

bintodec :: Integral i => i -> i
bintodec 0 = 0
bintodec i = 2 * bintodec (div i 10) + (mod i 10)

hexChar :: Char -> Integer
hexChar ch
    | ch == '0' = 0
    | ch == '1' = 1
    | ch == '2' = 2
    | ch == '3' = 3
    | ch == '4' = 4
    | ch == '5' = 5
    | ch == '6' = 6
    | ch == '7' = 7
```

```
      | ch == '8' = 8
      | ch == '9' = 9
      | ch == 'A' = 10
      | ch == 'B' = 11
      | ch == 'C' = 12
      | ch == 'D' = 13
      | ch == 'E' = 14
      | ch == 'F' = 15
      | ch == 'a' = 10
      | ch == 'b' = 11
      | ch == 'c' = 12
      | ch == 'd' = 13
      | ch == 'e' = 14
      | ch == 'f' = 15
      | otherwise     = 0

  parseHex :: String -> Integer
  parseHex [] = 0
  parseHex hxStr = hexChar (last hxStr) + (16 * parseHex (init
hxStr))

  converter:: String -> String -> [String]
  converter x y --["valor dec","valor hex","dec","hex"]
      |y == "bin" = [(show (bintodec (read x::Int))), (showHex
(bintodec (read x::Int))("")),"dec","hex"] --Num em Dec/Hex
      |y == "dec" = [(showHex (( read x::Int)) ("")),
showIntAtBase (2) intToDigit (read x::Int) (""),"hex","bin"]
--Num em Hex/Bin
      |y == "hex" = [show(parseHex x),showIntAtBase (2)
intToDigit (parseHex x) (""),"dec","bin"]

  main::IO()
  main = do arg1<-getLine::IO String
            pinga (read(arg1)::Int)
            printf ""

  pinga 0 = do printf ""
  pinga x = do pinga (x-1)
               arg1<-getLine::IO String
```

```
                let y = converter (head(words arg1)) (words arg1
!!1)
                printf"Caso %d:\n" x
                printf "%s %s\n" (y!!0) (y!!2)
                printf "%s %s\n" (y!!1) (y!!3)
                printf"\n"
```

3. **Volta à Faculdade de Física**: Uma partícula tem velocidade inicial e aceleração constante. Se a sua velocidade após certo momento é v então qual será seu deslocamento no dobro deste tempo?

   Entrada

   A entrada é um caso de teste e contém dois inteiros v e t (t significa o momento no qual a partícula ganha aquela velocidade).

   R:

```
deslocDobro::Double->Double->Double

deslocDobro v t = let accel = v/t

                  in (accel)*((2*t)**2)/2
```

4. **Encaixa ou Não I**: Paulinho tem em suas mãos um pequeno problema. A professora lhe pediu que ele construísse um programa para verificar, à partir de dois valores inteiros A e B, se B corresponde aos últimos dígitos de A.

   Entrada

   A entrada contém dois inteiros A e B positivos.

   Saída

   Imprima uma mensagem “encaixa” para quando corresponder os dígitos e “não encaixa” para o caso contrário.

   Ex.:

Entrada	Saída
5678690 78690	encaixa

R:

```
import Data.List

compareWith::String->String->Bool

compareWith x y

    |length y /= length x = False

    |otherwise = let isElemEqual =  zipWith (==) x y

                     difValuesList = [item | item <- isElemEqual, (not
item)]

                 in  null difValuesList
```

```
comparar::Int->Int->String
comparar n1 n2
    | length (show n1) >= length (show n2) = do if compareWith (show n1)
(show n2)
                                                then "encaixa"
                                                else let
n1tail=read(drop 1 (show n1))::Int
                                                     in comparar
n1tail n2
    | otherwise = "nao encaixa"
```

5. **Fibonacci de Novo**: A famosa sequência de Fibonacci pode ser definida da seguinte maneira:
   - Fib(1) = Fib(2) = 1
   - Fib(N) = Fib(N-1) + Fib(N-2), para N > 2

   Sua tarefa é simples, calcular o valor do resto de Fib(Fib(N)) por M.

   Entrada

   A entrada é composta por dois inteiros N e M (1 ≤ N ≤ 109, 2 ≤ M ≤ 106).

   Saída

   Imprima uma linha contendo um inteiro igual ao resto de Fib(Fib(N)) por M.

   Exemplo:

Exemplo de Entrada	Exemplo de Saída
1 100 2 100 3 100 4 100 5 100 5 2 6 100	1 1 1 2 5 1 21

R:

R:

```
fib 0 = 0
fib 1 = 1
fib 2 = 1
fib n = fib (n - 1) + fib (n - 2)

restfib n m = (fib((fib n)) `mod` m)
```