Converting Numbers to Words

Oliver Krischer

August 2020

1 Problem Description

Write a function, that takes a nonnegative number less then one million and returns a string that represents the number in words, such that:

$convert :: Int \to String$

Examples:

$convert\ 308000 =$ `"three hundred and eight thousand"`
$convert\ 369027 =$ `"three hundred and sixty-nine thousand and twenty-seven"`
$convert\ 369401 =$ `"three hundred and sixty-nine thousand four hundred and one"`

2 Problem Solution

Define the names of the component numbers:

```
units, teens, tens :: [String]
units = ["zero", "one", "two", "three", "four", "five", "six", "seven",
         "eight", "nine", "ten"]
teens = ["ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen",
         "sixteen", "seventeen", "eighteen", "nineteen"]
tens = ["twenty", "thirty", "fourty", "fifty", "sixty", "seventy",
        "eighty", "ninety"]
```

Break down the problem into smaller steps. Begin with the conversion of a one-digit number, such that: $0 <= n < 10$.

```
convert1 :: Int -> String
convert1 n = units !! n
```

The next step is conversion of a two-digit number, such that: $0 <= n < 100$. At first, extract the digits into a tuple:

```
digits2 :: Int -> (Int, Int)
digits2 n = (n `div` 10, n `mod` 10)
```

Now combine the two digits into a string of words:

```
combine2 :: (Int, Int) → String
combine2 (t, u)
  | t ≡ 0  = units !! u
  | t ≡ 1  = teens !! u
  | u ≡ 0 = tens !! (t − 2)
  | otherwise = tens !! (t − 2) ++ "-" ++ units !! u
```

And then compose convert2 from digit2 and combine2:

```
convert2 :: Int → String
convert2 = combine2 ∘ digits2
```

Instead of combining two functions in this step we could have written a single function with a where-clause like so:

```
combine2 :: Int → String
combine2 n
  | t ≡ 0  = units !! u
  | t ≡ 1  = teens !! u
  | u ≡ 0 = tens !! (t − 2)
  | otherwise = tens !! (t − 2) ++ "-" ++ units !! u
  where (t, u) = (n `div` 10, n `mod` 10)
```

Now we can define convert3, which takes a number with three digits, such that $0 <= n < 1.000$.

```
convert3 :: Int → String
convert3 n
  | h ≡ 0       = convert2 t
  | t ≡ 0       = units !! h ++ " hundred"
  | otherwise   = units !! h ++ " hundred and " ++ convert2 t
  where (h, t) = (n `div` 100, n `mod` 100)
```

For converting a six-digit number such that $0 <= n < 1.000.000$, we can now use convert3 with the same pattern we used in step 3.

```
convert6 :: Int → String
convert6 n
  | m ≡ 0       = convert3 h
  | h ≡ 0       = convert3 m ++ " thousand"
  | otherwise   = convert3 m ++ " thousand" ++ link h ++ convert3 h
  where (m, h) = (n `div` 1000, n `mod` 1000)
```

Here we used a function ‘link‘ because we need the connecting word ”and” between words for m and h in the case that $0 < m$ and $0 < h < 100$. Thus

```
link :: Int → String
link h = if h < 100 then " and " else " "
```

Because our function convert6 is already able to handle all inputs for 0 <= $n < 1.000.000$, as it was stated in the problem desciption for ‘convert‘, we can simply substitute ‘convert6‘ with ‘convert‘:

```
convert :: Int → String
convert = convert6
```