Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH-232-A
Date: 2021-10-08
Due: 2021-10-15

ICS 2021 Problem Sheet #5

Problem 5.1: *base b numbers closed form* (1+2 = 3 points)

Consider a base b number system ($b > 1$) with n digits and the number $1 \ldots 1_b$ (n times the digit 1).

a) Define a sum formula that provides the value of the n-digit number $1 \ldots 1_b$.

b) Proof via induction that the value of the n-digit number $1 \ldots 1_b$ is given by the following closed form:

$$\frac{1-b^n}{1-b}$$

For example, the 4-digit base 5 number 1111_5 has the value $\frac{1-5^4}{1-5} = \frac{-654}{-4} = 156$.

Solution:

a) From the definition of base b numbers we obtain:

$$1 \ldots 1_b = \sum_{i=1}^{n} b^{i-1} = \sum_{i=0}^{n-1} b^i$$

b) We show via induction over n that the following holds:

$$\sum_{i=0}^{n-1} b^i = \frac{1-b^n}{1-b}$$

Base case (n = 1):

$$\sum_{i=0}^{n-1} b^i = \sum_{i=0}^{0} b^i = b^0 = 1 = \frac{1-b}{1-b} = \frac{1-b^1}{1-b} = \frac{1-b^n}{1-b}$$

Induction step: assume the equation holds for n, lets consider $n+1$

$$\begin{aligned}
\sum_{i=0}^{(n+1)-1} b^i &= \sum_{i=0}^{n} b^i \\
&= \sum_{i=0}^{n-1} b^i + b^n \\
&= \frac{1-b^n}{1-b} + \frac{(1-b)b^n}{1-b} \\
&= \frac{1-b^n+b^n-b^{n+1}}{1-b} \\
&= \frac{1-b^{n+1}}{1-b}
\end{aligned}$$

Marking:

a) - *1pt for a correct proof of the base case*

b) - *1pt for the correct construction of the induction step*
- *1pt for the correct execution of the induction step*

Problem 5.2: *unicode and utf-8 encoding* (1+2+1 = 4 points)

The content of a file containing UTF-8 Unicode encoded text is given by the following sequence of bytes in hexadecimal notation:

```
48 65 6c 6c 6f 20 f0 9f 8c 8d 21 0a
```

a) Write each byte in binary notation.

b) Identify the unicode code points of the characters. What is the text stored in the file?

c) Which line end convention is used? What are other popular line end conventions?

Solution:

a) Conversion of each byte into binary:

```
0x48 = 0b01001000
0x65 = 0b01100101
0x6c = 0b01101100
0x6c = 0b01101100
0x6f = 0b01101111
0x20 = 0b00100000
0xf0 = 0b11110000
0x9f = 0b10011111
0x8c = 0b10001100
0x8d = 0b10001101
0x21 = 0b00100001
0x0a = 0b00001010
```

b) Unicode code points of the characters:

```
0x48 = 0b01001000 = U+0048 = 'H'
0x65 = 0b01100101 = U+0065 = 'e'
0x6c = 0b01101100 = U+006c = 'l'
0x6c = 0b01101100 = U+006c = 'l'
0x6f = 0b01101111 = U+006f = 'o'
0x20 = 0b00100000 = U+0020 = ' '
0xf09f8c8d = 0b11110000 10011111 10001100 10001101 = U+1f30d = '?'
0x21 = 0b00100001 = U+0021 = '!'
0x0a = 0b00001010 = U+000a = LF
```

Most characters are in the ASCII range and easy to decode. There is one character that is not in the ASCII range. It is encoded using four bytes. By removing the leading format bits, we get the bit sequence 000 011111 001100 001101. Aligning the bits into 8-bit blocks, we get 0b00001 11110011 00001101, which is 0x01f30d and hence the character is U+1f30d (Globe Showing Europe-Africa).

c) The line end convention is a LF character, which is the line end convention used on Unix and Unix-like systems. Microsoft Windows uses the CR LF line end convention while classic Apple computers used CR as a line end convention.

Marking:

a) - *-0.1pt for each incorrect binary conversion, not negative*

b) - *1pt for the "ASCII" unicode code points and characters*
- *1pt for the "emoji" unicode code point and character*

c) - *0.4pt for Unix line end convention*
- *0.3pt for describing the Windows line end convention*
- *0.3pt for describing another line end convention*

Problem 5.3: *long years (haskell)* (1 point)

According to the ISO8601 calendar, most years have 52 weeks, but some have 53 weeks. These are so called long years. There is a relatively simple way to calculate whether a given year y is a long year. The function $w(y)$ determines with the helper function $p(y)$ the number of weeks in the year y. (Note that the functions use integer division.)

$$p(y) = (y + \frac{y}{4} - \frac{y}{100} + \frac{y}{400}) \bmod 7$$

$$w(y) = 52 + \begin{cases} 1 & p(y) == 4 \wedge p(y-1) == 3 \\ 0 & \text{otherwise} \end{cases}$$

Implement a Haskell function `isLongYear :: Int -> Bool` to determine whether a year is a long year. Use the `isLongYear` function to calculate all long years in the range 2000..2100.

Submit your Haskell code as a plain text file.

Solution:

```haskell
module Main (main) where

import Test.HUnit

-- |The 'isLongYear' function indicates whether a year is a long year
-- (i.e., a year with 53 weeks instead of 52 weeks).
isLongYear :: Int -> Bool
isLongYear y = w y > 52
    where
      p y = (y + y `div` 4 - y `div` 100 + y `div` 400) `mod` 7
      w y = if (p y) == 4 || p (y-1) == 3 then 53 else 52

-- Below are some test cases...

isLongYearTests = TestList [
                   filter isLongYear [2000..2100] ~?= [
                    2004, 2009, 2015, 2020, 2026, 2032, 2037, 2043, 2048,
                    2054, 2060, 2065, 2071, 2076, 2082, 2088, 2093, 2099
                   ]
                  ]

main = runTestTT isLongYearTests
```

Marking:

- *0.5pt for a proper definition of `isLongYear`*
- *0.5pt for calculating the long years in the range 2000..2100*

Problem 5.4: *decimal to binary and binary to decimal (haskell)* (1+1 = 2 points)

Implement a function to convert a decimal number into a binary notation and one function to convert from a binary notation back.

a) Implement a function `dtob :: Int -> String` that converts a non-negative integer number into a `String` (consisting of the characters `'0'` and `'1'`) representing the integer number as a binary number. It is not necessary to handle negative integers in a meaningful way.

b) Implement a function `dtob :: String -> Int` that converts a `String` (consisting of the characters `'0'` and `'1'`) representing a binary number into the corresponding non-negative integer number. It is not necessary to handle unexpected strings in a meaningful way.

Submit your Haskell code as a plain text file. Below is a template file with a few unit test cases.

```haskell
module Main (main) where

import Test.HUnit

-- |The 'dtob' function converts a non-negative integer number into a
-- String providing a binary representation of the number.
dtob :: Int -> String
dtob _ = undefined

-- |The 'btod' function converts a String representing a non-negative
-- integer number as a binary number into an integer number.
btod :: String -> Int
btod _ = undefined

{-
   Below are some test cases.
-}

dtobTests = TestList [ dtob 0 ~?= "0"
                     , dtob 1 ~?= "1"
                     , dtob 2 ~?= "10"
                     , dtob 127 ~?= "1111111"
                     , dtob 12345 ~?= "11000000111001"
                     ]

btodTests = TestList [ btod "0" ~?= 0
                     , btod "1" ~?= 1
                     , btod "10" ~?= 2
                     , btod "1111111" ~?= 127
                     , btod "11000000111001" ~?= 12345
                     ]

main = runTestTT $ TestList [ dtobTests, btodTests ]
```

Solution:

```haskell
module Main (main) where

import Test.HUnit

-- |The 'dtob' function converts a non-negative integer number into a
-- String, providing a binary representation of the number.
dtob :: Int -> String
dtob 0 = "0"
dtob 1 = "1"
dtob n = dtob (div n 2) ++ (if odd n then "1" else "0")

-- |The 'btod' function converts a String representing a non-negative
-- integer number as a binary number into an integer number.
btod :: String -> Int
btod xs = btod' (reverse xs) where
    btod' "0" = 0
    btod' "1" = 1
    btod' (x:xs) = btod' [x] + 2 * btod' xs

-- Below are some test cases...

dtobTests = TestList [ dtob 0 ~?= "0"
                     , dtob 1 ~?= "1"
```

```
                      , dtob 2 ~?= "10"
                      , dtob 127 ~?= "1111111"
                      , dtob 12345 ~?= "11000000111001"
                      ]

btodTests = TestList [ btod "0" ~?= 0
                     , btod "1" ~?= 1
                     , btod "10" ~?= 2
                     , btod "1111111" ~?= 127
                     , btod "11000000111001" ~?= 12345
                     ]

main = runTestTT $ TestList [ dtobTests, btodTests ]
```

Marking:

a) - *1pt for a proper implementation of* `dtob`

b) - *1pt for a proper implementation of* `btod`