Basic Haskell Examples

haskellforall.com/2015/10/basic-haskell-examples.html

The Haskell community self-selects for people interested in unique things that Haskell can do that other languages cannot do. Consequently, a large chunk of Haskell example code in the wild uses advanced idioms (and I'm guilty of that, too).

So I would like to swing the pendulum in the other direction by just writing five small but useful programs without any imports, language extensions, or advanced features. These are programs that you could write in any other language and **that's the point**: you can use Haskell in the same way that you use other languages.

They won't be the prettiest or most type-safe programs, but that's okay.

Example #1: TODO program

This program is an interactive TODO list:

```
putTodo :: (Int, String) -> IO ()
putTodo (n, todo) = putStrLn (show n ++ ": " ++ todo)

prompt :: [String] -> IO ()
prompt todos = do
    putStrLn ""
    putStrLn "Current TODO list:"
    mapM_ putTodo (zip [0..] todos)
    command <- getLine
    interpret command todos

interpret :: String -> [String] -> IO ()
interpret ('+':' ':todo) todos = prompt (todo:todos)
interpret ('-':' ':num ) todos =
    case delete (read num) todos of
        Nothing -> do
            putStrLn "No TODO entry matches the given number"
            prompt todos
        Just todos' -> prompt todos'
interpret  "q"           todos = return ()
interpret  command       todos = do
    putStrLn ("Invalid command: `" ++ command ++ "`")
    prompt todos

delete :: Int -> [a] -> Maybe [a]
delete 0 (_:as) = Just as
delete n (a:as) = do
    as' <- delete (n - 1) as
    return (a:as')
delete _  []    = Nothing

main = do
```

```
    putStrLn "Commands:"
    putStrLn "+ <String> - Add a TODO entry"
    putStrLn "- <Int>    - Delete the numbered entry"
    putStrLn "q          - Quit"
    prompt []
```

Example usage:

```
$ runghc todo.hs
Commands:
+ <String> - Add a TODO entry
- <Int>    - Delete the numbered entry
q          - Quit

Current TODO list:
+ Go to bed

Current TODO list:
0: Go to bed
+ Buy some milk

Current TODO list:
0: Buy some milk
1: Go to bed
+ Shampoo the hamster

Current TODO list:
0: Shampoo the hamster
1: Buy some milk
2: Go to bed
- 1

Current TODO list:
0: Shampoo the hamster
1: Go to bed
q
```

Example #2 - Rudimentary TSV to CSV

The following program transforms tab-separated standard input into comma-separated standard output. The program does not handle more complex cases like quoting and is not standards-compliant:

```
tabToComma :: Char -> Char
tabToComma '\t' = ','
tabToComma  c   = c

main = interact (map tabToComma)
```

Example usage:

```
$ cat file.tsv
```

```
1   2   3
4   5   6
$ cat file.tsv | runghc tsv2csv.hs
1,2,3
4,5,6
```

Example #3 - Calendar

This program prints a calendar for 2015

```
data DayOfWeek
    = Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday
    deriving (Eq, Enum, Bounded)

data Month
    = January | February | March     | April   | May      | June
    | July    | August   | September | October | November | December
    deriving (Enum, Bounded, Show)

next :: (Eq a, Enum a, Bounded a) => a -> a
next x | x == maxBound = minBound
       | otherwise     = succ x

pad :: Int -> String
pad day = case show day of
    [c] -> [' ', c]
    cs  -> cs

month :: Month -> DayOfWeek -> Int -> String
month m startDay maxDay = show m ++ " 2015\n" ++ week ++ spaces Sunday
  where
    week = "Su Mo Tu We Th Fr Sa\n"

    spaces currDay | startDay == currDay = days startDay 1
                   | otherwise           = "   " ++ spaces (next currDay)

    days Sunday    n | n > maxDay = "\n"
    days _         n | n > maxDay = "\n\n"
    days Saturday  n              = pad n ++ "\n" ++ days  Sunday    (succ n)
    days day       n              = pad n ++ " "  ++ days (next day) (succ n)

year = month January   Thursday  31
    ++ month February  Sunday    28
    ++ month March     Sunday    31
    ++ month April     Wednesday 30
    ++ month May       Friday    31
    ++ month June      Monday    30
    ++ month July      Wednesday 31
    ++ month August    Saturday  31
    ++ month September Tuesday   30
```

```
    ++ month October   Thursday  31
    ++ month November  Sunday    30
    ++ month December  Tuesday   31

main = putStr year
```

Example usage:

```
$ runghc calendar.hs
January 2015
Su Mo Tu We Th Fr Sa
              1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

February 2015
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28

March 2015
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

April 2015
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

May 2015
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

June 2015
Su Mo Tu We Th Fr Sa
```

```
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

```
July 2015
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

```
August 2015
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

```
September 2015
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

```
October 2015
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

```
November 2015
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

```
December 2015
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
```

```
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

Example #4 - Decode RNA

This program converts an RNA sequence read from standard input into the equivalent sequence of amino acids written to standard output, using the genetic code:

```
data RNA = A | U | C | G
    deriving (Read)

data AminoAcid
    = Ala | Cys | Asp | Glu | Phe | Gly | His | Ile | Lys | Leu
    | Met | Asn | Pro | Gln | Arg | Ser | Thr | Val | Trp | Tyr
    | Stop
    deriving (Show)

decode :: RNA -> RNA -> RNA -> AminoAcid
decode U U U = Phe
decode U U C = Phe
decode U U A = Leu
decode U U G = Leu
decode U C _ = Ser
decode U A U = Tyr
decode U A C = Tyr
decode U A A = Stop
decode U A G = Stop
decode U G U = Cys
decode U G C = Cys
decode U G A = Stop
decode U G G = Trp
decode C U _ = Leu
decode C C _ = Pro
decode C A U = His
decode C A C = His
decode C A A = Gln
decode C A G = Gln
decode C G _ = Arg
decode A U U = Ile
decode A U C = Ile
decode A U A = Ile
decode A U G = Met
decode A C _ = Thr
decode A A U = Asn
decode A A C = Asn
decode A A A = Lys
decode A A G = Lys
decode A G U = Ser
```

```
decode A G C = Ser
decode A G A = Arg
decode A G G = Arg
decode G U _ = Val
decode G C _ = Ala
decode G A U = Asp
decode G A C = Asp
decode G A A = Glu
decode G A G = Glu
decode G G _ = Gly

decodeAll :: [RNA] -> [AminoAcid]
decodeAll (a:b:c:ds) = decode a b c : decodeAll ds
decodeAll  _         = []

main = do
    str <- getContents
    let rna :: [RNA]
        rna = map (\c -> read [c]) str

    let aminoAcids :: [AminoAcid]
        aminoAcids = decodeAll rna

    putStrLn (concatMap show aminoAcids)
```

Example usage:

```
$ echo "ACAUGUCAGUACGUAGCUAC" | runghc decode.hs
ThrCysGlnTyrValAlaThr
```

Example #5 - Bedtime story generator

This generates a "random" bedtime story:

```
stories :: [String]
stories = do
    let str0 = "There once was "

    str1 <- ["a princess ", "a cat ", "a little boy "]

    let str2 = "who lived in "

    str3 <- ["a shoe.", "a castle.", "an enchanted forest."]

    let str4 = "  They found a "

    str5 <- ["giant ", "frog ", "treasure chest " ]

    let str6 = "while "
```

```
    str7 <- ["when they got lost ", "while strolling along "]

    let str8 = "and immediately regretted it.  Then a "

    str9 <- ["lumberjack ", "wolf ", "magical pony "]

    let str10 = "named "

    str11 <- ["Pinkie Pie ", "Little John ", "Boris "]

    let str12 = "found them and "

    str13 <- ["saved the day.", "granted them three wishes."]

    let str14 = "  The end."

    return (  str0
           ++ str1
           ++ str2
           ++ str3
           ++ str4
           ++ str5
           ++ str6
           ++ str7
           ++ str8
           ++ str9
           ++ str10
           ++ str11
           ++ str12
           ++ str13
           ++ str14
           )

main = do
    let len = length stories
    putStrLn ("Enter a number from 0 to " ++ show (len - 1))
    n <- readLn
    putStrLn ""
    putStrLn (stories !! n)
```

Example usage:

```
$ runghc story.hs
Enter a number from 0 to 971
238

There once was a princess who lived in an enchanted forest.  They found a giant
while while strolling along and immediately regretted it.  Then a lumberjack
named Boris found them and saved the day.  The end.
```

Conclusion

If you would like to contribute a simple example of your own, try sharing a paste of your program under the #Haskell #BackToBasics hashtags.