1 Quines for Everyone

This is an interruption to the sequence on provability logic (though it's not entirely unrelated).

The code below spits out a Haskell program that prints out a Perl program that prints out a Python program that prints out a Ruby program that prints out a C program that prints out a Java program that prints out the original program. Nothing new, an obvious generalisation of this. (Well, truth be told, it's a block of HTML that generates some Haskell code...)

But there's one big difference. To allow everyone else to join in the fun you can configure it yourself! Any non-empty list of languages will do, including length one. You may start hitting language line length limits if you make your list too long. Note that you can repeat languages so you can give someone some Haskell which decays into Perl after n iterations.

The code is geared towards generating tightly packed code but it's easily adapted to generate something slightly more readable. Apologies for the C warnings. Trivial to fix.

It's easily extended to support many more languages. C++, C#, obj-C, ocaml, Prolog, Lisp, Scheme and Go, say, should all be trivial apart from maybe a tiny bit of work with delimiting strings. (Eg. for Go you may need to tweak the import statement slightly so it doesnt't use double quotation marks.)

The code leaves many opportunities for refactoring but it's not like anyone is actually going to use this code for real production so I'm leaving it as is now.

I've only tested it under MacOS X. I don't know if there are carriage return/linefeed issues with other OSes. The shell script at the end is a regression test.

There was a little bit of theory involved which I learnt from Vicious Circles.

Here's a challenge for you: write a quine that takes as input the name of a

language and outputs the same thing implemented in the input language. Much harder than what I just wrote.

```
import Data.List
```

Here's the bit you can easily play with:

```
langs = [Haskell, Perl, Python, Ruby, C, Java]
```

2 Implementation

```
data Languages = Haskell | Ruby | Perl | C | Python | Java

sequenceFromString Haskell s = "map toEnum[" ++ (intercalate "," $
    map (\c -> show (fromEnum c)) s) ++ "]"
sequenceFromString Perl s    = (intercalate "," $
    map (\c -> "chr(" ++ show (fromEnum c) ++ ")") s)
sequenceFromString Python s  = (intercalate "+" $
    map (\c -> "chr(" ++ show (fromEnum c) ++ ")") s)
sequenceFromString Ruby s    = (intercalate "+" $
    map (\c -> show (fromEnum c) ++ ".chr") s)
sequenceFromString C s       = concatMap
    (\c -> "putchar(" ++ show (fromEnum c) ++ ");") s
sequenceFromString Java s    = concatMap
    (\c -> "o.write(" ++ show (fromEnum c) ++ ");") s

paramList' Haskell = intercalate " " .
                map (\n -> "a" ++ show n)
paramList' C       = intercalate "," .
                map (\n -> "char *a" ++ show n)
paramList' Python  = intercalate "," .
                map (\n -> "a" ++ show n)
paramList' Ruby    = intercalate "," .
                map (\n -> "a" ++ show n)
paramList' Java    = intercalate "," .
                map (\n -> "String a" ++ show n)

paramList Perl    _ = ""
paramList lang n = paramList' lang [0..n-1]
```

```
driver l args = defn l ++
        intercalate (divider l) args ++ endDefn l

divider C       = "\",\""
divider Perl    = "',"
divider Ruby    = "\",\""
divider Python  = "\",\""
divider Haskell = "\" \""
divider Java    = "\",\""

defn C       = "main(){q(\""
defn Perl    = "&q('"
defn Python  = "q(\""
defn Ruby    = "q(\""
defn Haskell = "main=q \""
defn Java    = "public static void main(String[]args){q(\""

endDefn C       = "\");}"
endDefn Perl    = "')"
endDefn Python  = "\")"
endDefn Ruby    = "\")"
endDefn Haskell = "\""
endDefn Java    = "\");}}"

arg Haskell n = "a" ++ show n
arg Perl n    = "$_[" ++ show n ++ "]"
arg C n       = "printf(a" ++ show n ++ ");"
arg Python n  = "a" ++ show n
arg Ruby n    = "a" ++ show n
arg Java n    = "o.print(a" ++ show n ++ ");"

argDivide Haskell l = "++" ++
       sequenceFromString Haskell (divider l) ++ "++"
argDivide Perl l    = ","    ++
       sequenceFromString Perl (divider l) ++ ","
argDivide C l       =
       sequenceFromString C (divider l)
argDivide Python l  =
       "+" ++ sequenceFromString Python (divider l) ++ "+"
argDivide Ruby l    =
```

```
       "+" ++ sequenceFromString Ruby (divider l) ++ "+"
argDivide Java l    =
       sequenceFromString Java (divider l)

argList lang1 lang2 n = intercalate (argDivide lang1 lang2) $
    map (arg lang1) ([1..n-1] ++ [0])

fromTo Haskell l n = "q " ++ paramList Haskell n ++
    "=putStrLn$a0++" ++
    sequenceFromString Haskell ("\n" ++ defn l) ++ "++" ++
    argList Haskell l n ++ "++" ++
    sequenceFromString Haskell (endDefn l)
fromTo Perl    l n = "sub q {" ++ "print $_[0]," ++
    sequenceFromString Perl ("\n" ++ defn l) ++
    "," ++ argList Perl l n ++ "," ++
    sequenceFromString Perl (endDefn l ++ "\n") ++ "}"
fromTo Python  l n = "def q(" ++ paramList Python n ++
    "): print a0+" ++
    sequenceFromString Python ("\n" ++ defn l) ++
    "+" ++ argList Python l n ++ "+" ++
    sequenceFromString Python (endDefn l)
fromTo Ruby    l n = "def q(" ++ paramList Ruby n ++
    ") print a0+" ++
    sequenceFromString Ruby ("\n" ++ defn l) ++
    "+" ++ argList Ruby l n ++ "+" ++
    sequenceFromString Ruby (endDefn l ++ "\n") ++ " end"
fromTo C       l n = "q(" ++ paramList C n ++ "){" ++
    "printf(a0);" ++
    sequenceFromString C ("\n" ++ defn l) ++ argList C l n ++
    sequenceFromString C (endDefn l ++ "\n") ++ "}"
fromTo Java    l n =
   "public class quine{public static void q(" ++
    paramList Java n ++
    "){java.io.PrintStream o=System.out;o.print(a0);" ++
    sequenceFromString Java ("\n" ++ defn l) ++ argList Java l n ++
    sequenceFromString Java (endDefn l ++ "\n") ++ "}"

main = do
    let n = length langs
```

```
    let langs' = cycle langs
    putStrLn $ fromTo (head langs') (head (tail langs')) n
    putStrLn $ driver (head langs') $
        zipWith (\lang1 lang2 -> fromTo lang1 lang2 n)
        (take n (tail langs')) (tail (tail langs'))
```

3 Regression Test

Assuming this article is stored in -quineCentral.lhs-.

```
runghc quineCentral.lhs>1.hs
cat 1.hs
echo "--------------------------------"
runghc 1.hs>2.pl
cat 2.pl
echo "--------------------------------"
perl 2.pl>3.py
cat 3.py
echo "--------------------------------"
python 3.py>4.ruby
cat 4.ruby
echo "--------------------------------"
ruby 4.ruby>5.c
cat 5.c
echo "--------------------------------"
gcc -o 5 5.c
./5>quine.java
cat quine.java
echo "--------------------------------"
javac quine.java
java quine>7.hs
cat 7.hs
diff 1.hs 7.hs
```