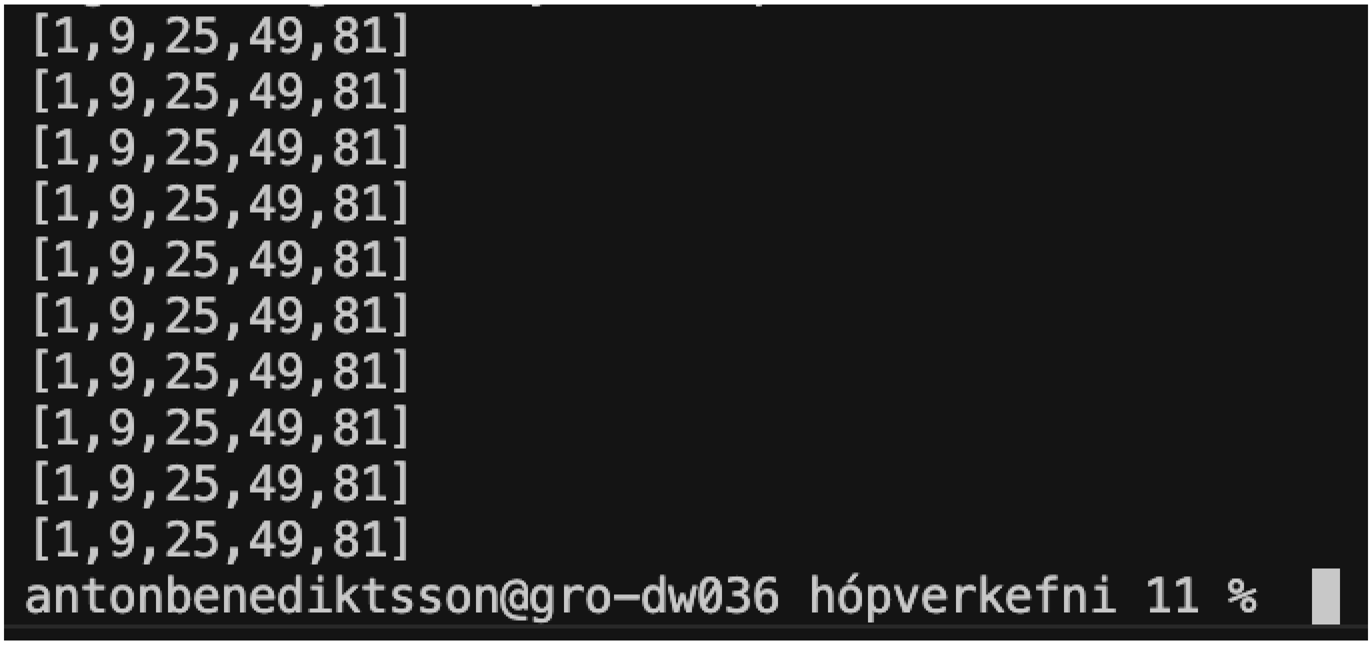
TÖL304G Forritunarmál Verkefnablað 11

Hópverkefni



```
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
[1,9,25,49,81]
antonbenediktsson@gro-dw036 hópverkefni 11 %
```

```haskell
module Main (main) where

{-
Save this file under the name condMap.hs on your
computer.

Finish programming the ten functions that have a
skeleton in this file. Most of them are one-liners.

Once you have programmed the functions you can
compile this file using the command
  ghc --make condMap.hs
and you can then run the resulting executable
file, which will be condmap.exe on Windows
and will be simply condmap on Unix. So use
this command on Windows:
  condMap
and use this command on Unix:
  ./condMap

Run the program and show the output.
-}

-- This loads the definition of MonadPlus:
import Control.Monad

{-
Usage: let y = condMapI p f x
       where I is one of 1,2,3,4,5,6,7,8,9,10.
Pre:   x=[x1,x2,...,xN] is a list of type [a]
       of values that are valid arguments for
       p and f. p is (a -> Bool), f is (a -> b).
Post:  y is a list of the values (f z) where
       the z values are the values in the x list
       such that (p z) is True.

ALTERNATIVELY, YOU MAY USE THE FOLLOWING,
MORE GENERAL, DESCRIPTION:

Usage: let y = condMapI p f x
       where I is one of 1,2,3,4,5,6,7,8,9,10.
Pre:   x is a value of type (m a) where m is
       a monad such that (MonadPlus m) holds,
       containing values that are valid
       arguments for p and f. p is (a -> Bool),
       f is (a -> b).
Post:  y is a value of type (m b) containing the
       values (f z) where the z values are the
       values in the x container such that (p z)
       is True.

Note that this latter description fulfills
all the conditions that the former description
fulfills and more.
-}

-- Only use the functions map and filter,
-- and, of course f and p. Use no other built-in
-- functions.
-- Use no list comprehension and no do-notation.
-- You should use no if-expressions.
condMap1 :: (a->Bool)->(a->b)->[a]->[b]
condMap1 p f x = map f (filter p x)

-- Use list comprehension and use no functions
-- other than f and p.
-- You should use no if-expressions.
condMap2 :: (a->Bool)->(a->b)->[a]->[b]
condMap2 p f x = [f z | z <- x, p z]

-- Use the built-in function concatMap and no other built-in
-- function. You should also use an if-expression.
condMap3 :: (a->Bool)->(a->b)->[a]->[b]
condMap3 p f x = concatMap (\z -> if p z then [f z] else []) x

-- Use do-notation and use no built-in function and
-- do not use list comprehension. You should also use
-- an if-expression.
condMap4 :: (a->Bool)->(a->b)->[a]->[b]
condMap4 p f x =  do { z <- x; if p z then [f z] else [] }

-- Use (>>=) and return and no other built-in function.
-- You may also use the built-in special value mzero,
-- which will allow you to create a more general function
-- that works for all monads m such that (MonadPlus m)
-- holds. You should also use an if-expression.
```

```haskell
-- The type of condMap5 may be either
--  condMap5 :: (MonadPlus m) => (a->Bool)->(a->b)->m a->m b
-- OR
condMap5 :: (a->Bool)->(a->b)->[a]->[b]
condMap5 p f x = x >>= (\z -> if p z then return (f z) else mzero)

--Use (:) and no other built-in function.
-- You should not use any if-expressions.
condMap6 :: (a->Bool)->(a->b)->[a]->[b]
condMap6 _ _ [] = []
condMap6 p f (x:xs)
  | p x  = f x : condMap6 p f xs
  | True = condMap6 p f xs

-- Use head, tail and (:) and no other built-in function.
-- You should also use if-expressions.
condMap7 :: (a->Bool)->(a->b)->[a]->[b]
condMap7 p f x =
  if null x then
    []
  else
    if p(head x) then
      f(head x) : condMap7 p f (tail x)
    else
      condMap7 p f (tail x)

-- Use do-notation and use only the built-in function
-- return. Do not use any kind of list notation except [].
-- Instead of [] you may use the built-in special value
-- mzero, which will make the function more general so
-- that it works for monads m such that (MonadPlus m)
-- holds. You should also use an if-expression.
-- The type of condMap8 could be
-- condMap8 :: (a->Bool)->(a->b)->[a]->[b]
-- OR, MORE GENERAL GENERAL:
condMap8 :: MonadPlus m => (a->Bool)->(a->b)->m a->m b
condMap8 p f x = do { z <- x; if p z then return (f z) else mzero }

-- Use the built-in functions foldr and (:) and no
-- other built-in functions. Use no list notation
-- except []. You may use an anonymous helper function
-- and you may use if-expressions.
condMap9 :: (a->Bool)->(a->b)->[a]->[b]
```

```haskell
condMap9 p f x = foldr (\z acc -> if p z then f z : acc else acc) [] x

-- Use the built-in functions foldl, reverse and (:) and no
-- other built-in functions. Use no list notation
-- except []. You may use an anonymous helper function
-- and you may use if-expressions.
condMap10 :: (a->Bool)->(a->b)->[a]->[b]
condMap10 p f x = reverse (foldl (\acc z -> if p z then f z : acc else acc) [] x)

main :: IO ()
main = do
  print (condMap1 odd (^2) [1..10])
  print (condMap2 odd (^2) [1..10])
  print (condMap3 odd (^2) [1..10])
  print (condMap4 odd (^2) [1..10])
  print (condMap5 odd (^2) [1..10])
  print (condMap6 odd (^2) [1..10])
  print (condMap7 odd (^2) [1..10])
  print (condMap8 odd (^2) [1..10])
  print (condMap9 odd (^2) [1..10])
  print (condMap10 odd (^2) [1..10])
```