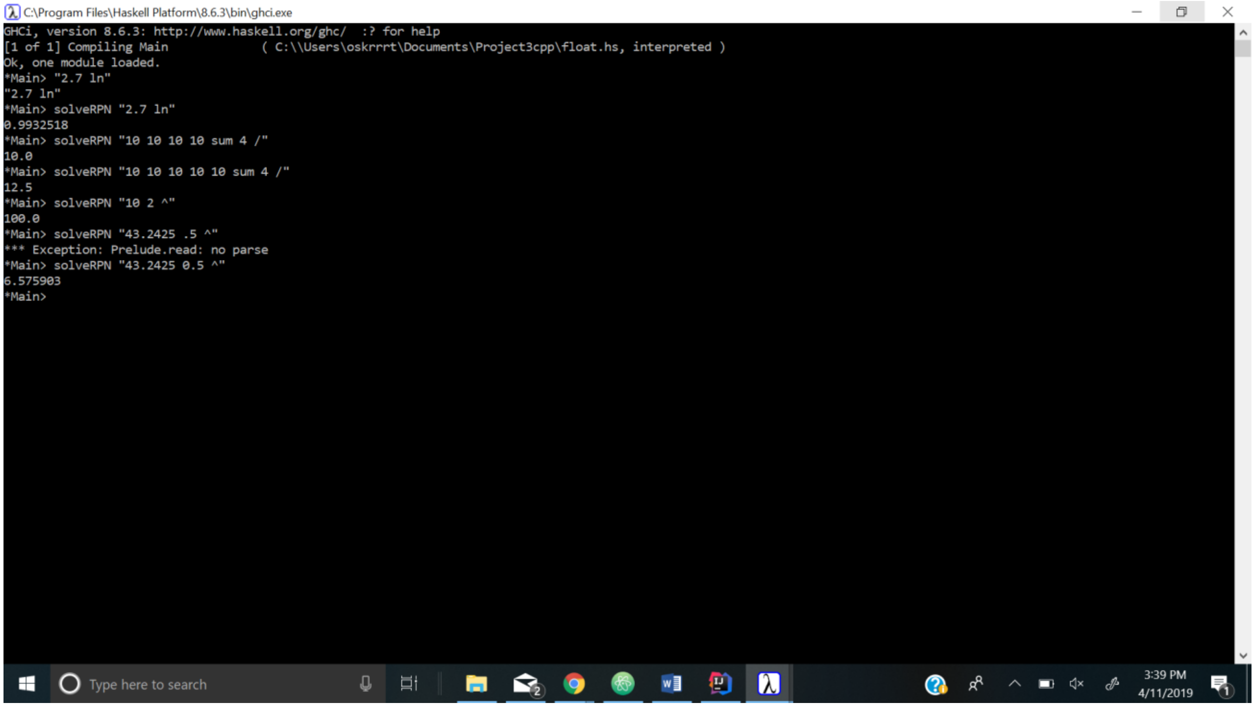
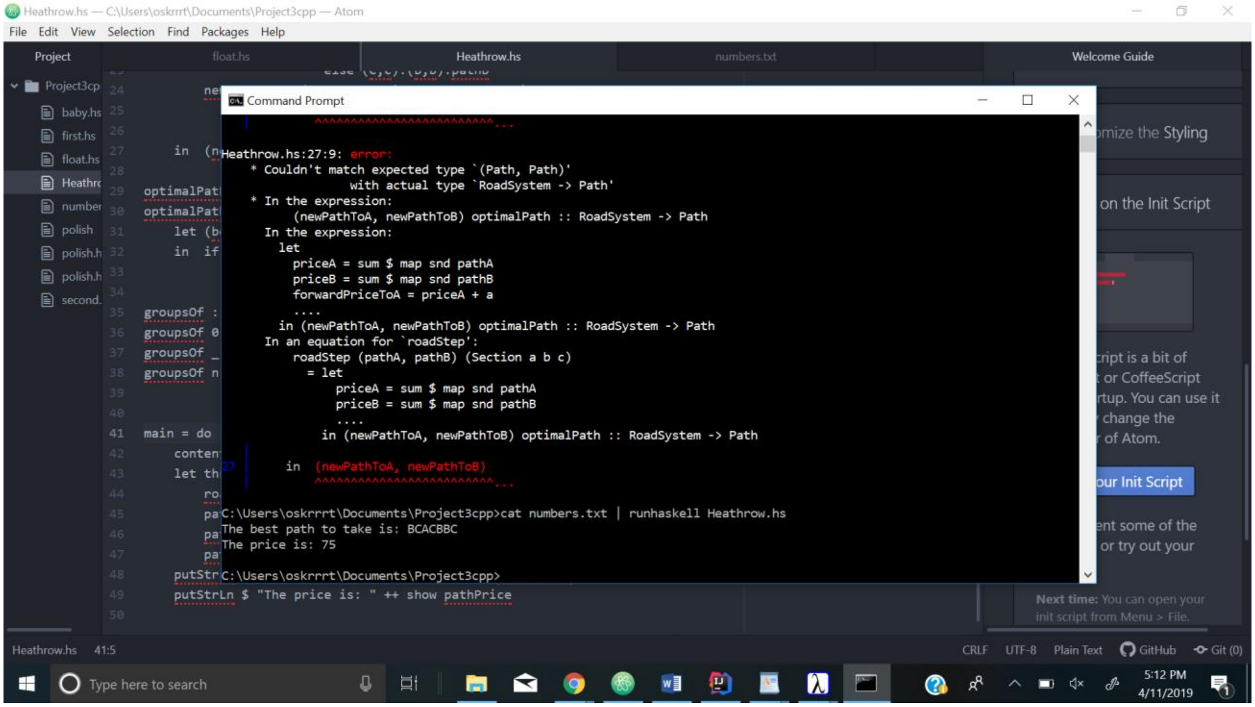


```
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( C:\\Users\oskrrrt\Documents\Project3cpp\float.hs, interpreted )
Ok, one module loaded.
*Main> "2.7 ln"
"2.7 ln"
*Main> solveRPN "2.7 ln"
0.9932518
*Main> solveRPN "10 10 10 10 sum 4 /"
10.0
*Main> solveRPN "10 10 10 10 10 sum 4 /"
12.5
*Main> solveRPN "10 2 ^"
100.0
*Main> solveRPN "43.2425 .5 ^"
*** Exception: Prelude.read: no parse
*Main> solveRPN "43.2425 0.5 ^"
6.575903
*Main>
```



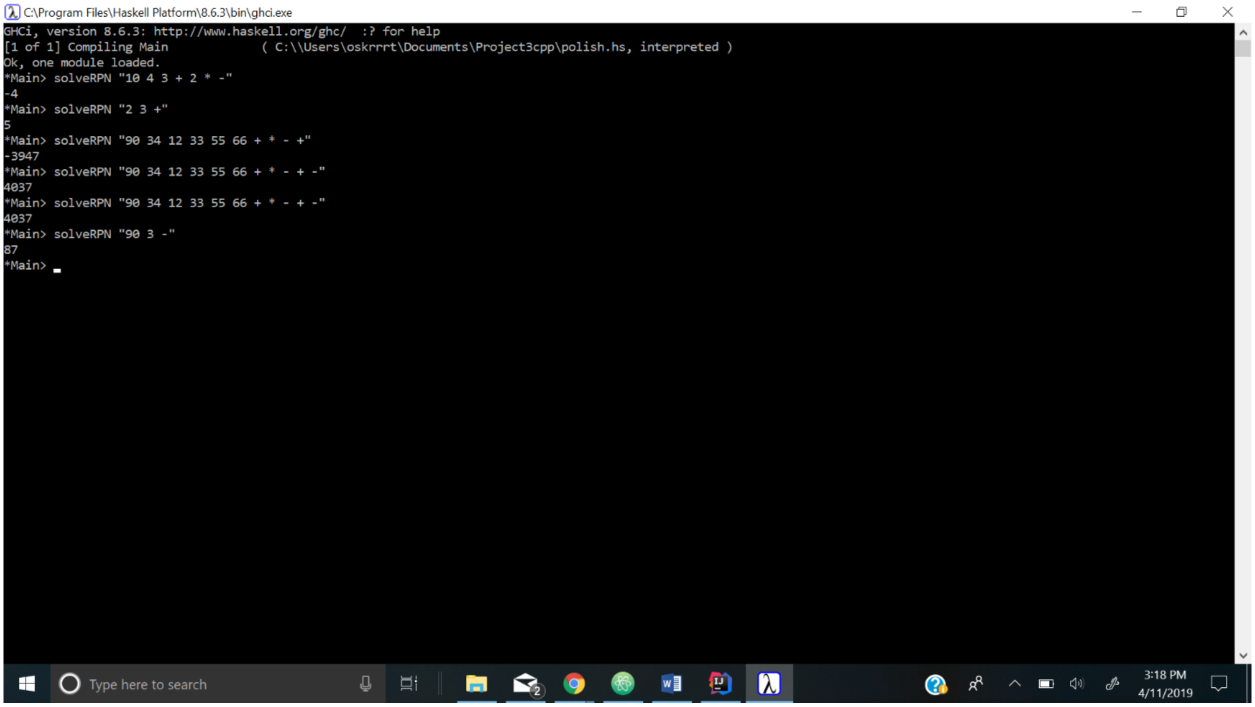
```
                ^^^^^^^^^^^^^^^^^^^^^^^^^...

Heathrow.hs:27:9: error:
    * Couldn't match expected type `(Path, Path)'
                  with actual type `RoadSystem -> Path'
    * In the expression:
        (newPathToA, newPathToB) optimalPath :: RoadSystem -> Path
      In the expression:
        let
          priceA = sum $ map snd pathA
          priceB = sum $ map snd pathB
          forwardPriceToA = priceA + a
          ....
        in (newPathToA, newPathToB) optimalPath :: RoadSystem -> Path
      In an equation for `roadStep':
          roadStep (pathA, pathB) (Section a b c)
            = let
                priceA = sum $ map snd pathA
                priceB = sum $ map snd pathB
                ....
              in (newPathToA, newPathToB) optimalPath :: RoadSystem -> Path
   |
27 |     in  (newPathToA, newPathToB)
   |         ^^^^^^^^^^^^^^^^^^^^^^^^^...

C:\Users\oskrrrt\Documents\Project3cpp>cat numbers.txt | runhaskell Heathrow.hs
The best path to take is: BCACBBC
The price is: 75

C:\Users\oskrrrt\Documents\Project3cpp>
```

```
    putStrLn $ "The price is: " ++ show pathPrice
```



```
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( C:\\Users\oskrrrt\Documents\Project3cpp\polish.hs, interpreted )
Ok, one module loaded.
*Main> solveRPN "10 4 3 + 2 * -"
-4
*Main> solveRPN "2 3 +"
5
*Main> solveRPN "90 34 12 33 55 66 + * - +"
-3947
*Main> solveRPN "90 34 12 33 55 66 + * - + -"
4037
*Main> solveRPN "90 34 12 33 55 66 + * - + -"
4037
*Main> solveRPN "90 3 -"
87
*Main>
```

```
import Data.List


solveRPN :: String -> Float
solveRPN = head . foldl foldingFunction [] . words
   where   foldingFunction (x:y:ys) "*" = (x * y):ys
         foldingFunction (x:y:ys) "+" = (x + y):ys
         foldingFunction (x:y:ys) "-" = (y - x):ys
         foldingFunction (x:y:ys) "/" = (y / x):ys
         foldingFunction (x:y:ys) "^" = (y ** x):ys
         foldingFunction (x:xs) "ln" = log x:xs
         foldingFunction xs "sum" = [sum xs]
         foldingFunction xs numberString = read numberString:xs


import Data.List
```

```
data Section = Section { getA :: Int, getB :: Int, getC :: Int } deriving (Show)
type RoadSystem = [Section]

heathrowToLondon :: RoadSystem
heathrowToLondon = [Section 50 10 30, Section 5 90 20, Section 40 2 25, Section 10 8 0]

data Label = A | B | C deriving (Show)
type Path = [(Label, Int)]

roadStep :: (Path, Path) -> Section -> (Path, Path)
roadStep (pathA, pathB) (Section a b c) =
   let priceA = sum $ map snd pathA
       priceB = sum $ map snd pathB
       forwardPriceToA = priceA + a
       crossPriceToA = priceB + b + c
       forwardPriceToB = priceB + b
       crossPriceToB = priceA + a + c
       newPathToA = if forwardPriceToA <= crossPriceToA
                    then (A,a):pathA
                    else (C,c):(B,b):pathB
       newPathToB = if forwardPriceToB <= crossPriceToB
                    then (B,b):pathB
                    else (C,c):(A,a):pathA
   in  (newPathToA, newPathToB)

optimalPath :: RoadSystem -> Path
optimalPath roadSystem =
   let (bestAPath, bestBPath) = foldl roadStep ([],[]) roadSystem
   in  if sum (map snd bestAPath) <= sum (map snd bestBPath)
```

```
            then reverse bestAPath
            else reverse bestBPath
groupsOf :: Int -> [a] -> [[a]]
groupsOf 0 _ = undefined
groupsOf _ [] = []
groupsOf n xs = take n xs : groupsOf n (drop n xs)


main = do
    contents <- getContents
    let threes = groupsOf 3 (map read $ lines contents)
        roadSystem = map (\[a,b,c] -> Section a b c) threes
        path = optimalPath roadSystem
        pathString = concat $ map (show . fst) path
        pathPrice = sum $ map snd path
    putStrLn $ "The best path to take is: " ++ pathString
    putStrLn $ "The price is: " ++ show pathPrice
```

```
50
10
30
5
90
20
40
2
25
10
8
```

0

```
solveRPN :: (Num a, Read a) => String -> a
solveRPN = head . foldl foldingFunction [] . words
   where   foldingFunction (x:y:ys) "*" = (x * y):ys
        foldingFunction (x:y:ys) "+" = (x + y):ys
        foldingFunction (x:y:ys) "-" = (y - x):ys
        foldingFunction xs numberString = read numberString:xs
```