COMP105 Lecture 24

Writing Programs

Writing programs

To write a program in Haskell we write a **main** function

```haskell
main :: IO ()
main = putStrLn "Hello world!"
```

`main` always

- Takes no arguments
- Returns an IO type

Writing programs

To run the program, we first need to **compile** it

```
$ ghc hello.hs
[1 of 1] Compiling Main    ( hello.hs, hello.o )
Linking hello ...

$ ./hello
Hello world!
```

Compiling on Windows

1. Save your code as `M:\code.hs`
2. Open the Command Prompt (search for `cmd`)
3. Switch to the M drive with `"M:"`
4. Compile with `"ghc code.hs"`
5. To run, just type `"code"`

You can also compile and run code in subdirectories, but you will need to use `"cd"` to first switch to the right directory

Command line arguments

Most command line programs take **arguments**

- `getArgs :: IO [String]` returns those arguments
- This function is in `System.Environment`

```haskell
import System.Environment

main :: IO ()
main = do
    args <- getArgs
    let output = "Command line arguments: " ++ show args
    putStrLn output
```

Looping in IO code

The only way to **loop** in IO code is to use recursion

```
printN :: String -> Int -> IO ()

printN _   0 = return ()
printN str n =
    do
        putStrLn str
        printN str (n-1)
```

- No variables!
- No loops!

Using command line arguments

```
main :: IO ()
main = do
    args <- getArgs
    let n = read (args !! 0) :: Int
    printN (args !! 1) n
```

```
$ ./repeat_string 3 hello
hello
hello
hello
```

File IO

`readFile :: String -> IO String` reads the contents of a file

Suppose that `example.txt` contains:

```
line one
line two
line three
```

```
ghci> readFile "example.txt"
"line one\nline two\nline three\n"
```

`writeFile`

`writeFile` writes a string **to a file**

- `writeFile :: String -> String -> IO ()`
- The file will be overwritten!

```
ghci> writeFile "output.txt" "hello\nthere\n"
```

The file `output.txt` contains:

```
hello
there
```

Finishing the marks.csv example

We wrote the **report** function in Lecture 18

- Now we can turn it into a program

```
main :: IO ()
main = do
    args <- getArgs
    let infile = args !! 0
        outfile = args !! 1
    input <- readFile infile
    writeFile outfile (report input)
```

Exercises

1. Write a program that takes one command line argument that is a file, and then prints the first line of that file to the screen
2. Write a program that takes one command line argument that is an integer x, and prints $x + 1$ to the screen
3. Write a program that asks the user to input a line of text, and then writes that text to a file called "output.txt"