COMP105 Lecture 23

IO

Recap: pure functions

So far, we have studied **pure** functional programming

Pure functions

- Have no side effects
- Always return a value
- Are deterministic

All **computation** can be done in pure functional programming

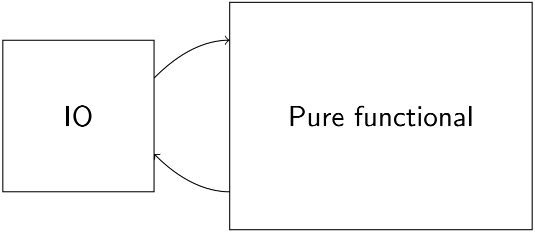
Input and output

Sometimes programs need to do **non-pure** things

- Print something to the screen
- Read or write a file
- Communicate over a network
- Create a GUI
- ...

Haskell includes mechanisms to do these impure things

IO vs Pure functional



- Impure IO code talks to the **outside world**
- Pure functional code does the **interesting computation**

IO code can call pure functions; Pure functions cannot call IO code

getLine

getLine reads a line of input from the console

```
ghci> getLine
hello
"hello"
```

```
ghci> :t getLine
getLine :: IO String
```

The IO type

The IO type marks a value as being **impure**

```
ghci> :t getLine
getLine :: IO String
```

```
ghci> :t getChar
getChar :: IO Char
```

If a function returns an IO type then it is impure

- It may have side effects
- It may return different values for the same inputs

The `IO` type

The `IO` type should be thought of as a **box**

- The box holds a value from an impure computation
- We can use `<-` to get the value out

```
ghci> x <- getLine
hello

ghci> x
"hello"

ghci> :t x
x :: String
```

The IO type

Values **must** be unboxed before they are used in pure functions

```
ghci> head (getLine)

Couldn't match expected type ‘[a]’
    with actual type ‘IO String’
```

```
ghci> x <- getLine
hello

ghci> head x
'h'
```

putStrLn

putStrLn prints a string onto the console

```
ghci> putStrLn "hello"
hello
```

```
ghci> :t putStrLn
putStrLn :: String -> IO ()
```

The return type indicates that it returns nothing useful

- It has the IO type, indicating that it has a side effect

Exercise

What do these ghci queries do?

```
ghci> x <- getLine
ghci> y <- getLine
ghci> putStrLn (x ++ " " ++ y)
```

```
ghci> n <- getLine
ghci> let num = (read n) :: Int
ghci> putStrLn (show (num + 1))
```

```
ghci> putStrLn (getLine)
```