



AC21007: Haskell Lecture 1

An Introduction to Haskell

František Farka

Syllabus of this part of the AC21007 module

Lectures

- ▶ Lecture 1: An introduction to Haskell
- ▶ Lecture 2: List functions, function polymorphism
- ▶ Lecture 3: Folds, tail recursion
- ▶ Lecture 4: Data types
- ▶ Lecture 5: Type classes
- ▶ Lecture 6: Sorting algorithms in Haskell
- ▶ Lecture 7: AI Algorithms: BFS, DFS, ...
- ▶ Lecture 8: tbd

Labs

- ▶ each week, half a slot
- ▶ please use the Lab 2
- ▶ weekly lab exercises, a final assignment




Resources


► Books:

 Thompson, Simon. Haskell: The craft of functional programming(3rd Edition)

 Mena, Alejandro Serrano. Beginning Haskell: A Project-Based Approach

► Free online resources:

 Learn You a Haskell for Great Good!
<http://learnyouahaskell.com/>

 Real World Haskell
<http://book.realworldhaskell.org/>



History of Haskell

- ▶ Named after the logician Haskell Curry
- ▶ Conference on Functional Programming Languages and Computer Architecture in Portland, Oregon. 1987
- ▶ Common language for *lazy functional programming languages* research
- ▶ Matured a lot over the past 29 years
- ▶ Several standards: Haskell98, Haskell2010
- ▶ De facto implementation: Glasgow Haskell Compiler (GHC)*
- ▶ New development of the language via GHC language extensions



*Current: The Glorious Glasgow Haskell Compilation System, version 7.10.2

Notable Features of Haskell



Haskell /'hæskəl/ is a standardised, general-purpose purely functional programming language, with non-strict semantics and strong static typing.*

- ▶ purely functional
- ▶ non-strict (also lazy) semantics
- ▶ (strong) static typing

*[https://en.wikipedia.org/wiki/Haskell_\(programming_language\)](https://en.wikipedia.org/wiki/Haskell_(programming_language))

Programming Language Paradigms

- ▶ Procedural languages: Pascal, C, Python, ...
- ▶ Object Oriented languages: C++, Java, C#, Python...
- ▶ Functional languages: Haskell, ML, OCaml, Lisp,...
- ▶ Functional and Object Oriented languages: Scala,...
- ▶ Declarative-Logic languages: Prolog, ...
- ▶ and many others: Erlang, Go,...



Imperative Style: C

$$\text{Power function: } b^n = \begin{cases} 1 & n = 0 \\ b * b^{n-1} & n > 0 \end{cases}$$

An implementation in C:

```
int power(int b, int n)
{
    int p = 1;

    for ( ; 0 < n; --n)
        p = p * b;
    return p;
}
```

`power(7,3) ==> 343`



Imperative Style: C

A bit more verbose implementation in C:

```
int power(b, n)
    int b;
    int n;
{
    int p;
    p = 1;

forcycle:
    if (! (0 < n )) goto endfor;
    p = p * b;
    --n;
    goto forcycle;
endfor:

    return p;
}
```



Functional Style: Haskell

$$\text{Power function: } b^n = \begin{cases} 1 & n = 0 \\ b * b^{n-1} & n > 0 \end{cases}$$

A Haskell implementation:

```
power :: Int -> Int -> Int
```

```
power b 0 = 1
```

```
power b n = b * (power b (n - 1))
```

```
power 7 3
```

```
==> 7 * (power 7 (3 - 1))
```

```
==> 7 * (power 7 2)
```

```
==> 7 * (7 * (power 7 (2 - 1)))
```

```
==> 7 * (7 * (power 7 1))
```

```
==> 7 * (7 * (7 * (power 7 (1 - 1))))
```

```
==> 7 * (7 * (7 * (power 7 0)))
```

```
==> 7 * (7 * (7 * (1)))
```

```
==> 7 * (7 * (7))
```

```
==> 7 * (49)
```

```
==> 343
```



Functional Style: Haskell (cont.)

Logical negation: $\neg p = \begin{cases} \textit{False} & p = \textit{True} \\ \textit{True} & p = \textit{False} \end{cases}$

A Haskell implementation:

```
not :: Bool -> Bool
not True  = False
not False = True
```



Purity

```
int counter = 0;
int power(int b, int n)
{
    if (counter > 5) return 42;
    counter++;

    int p = 1;
    for (; 0 < n; --n)
        p = p * b;
    return p;
}
```

```
power :: Int -> Int -> Int
power b 0 = 1
power b n = b * (power b (n-1))
```

- ▶ Both power functions have similar type
- ▶ However, Haskell's power function is *pure* - has no side effects



Summary



Imperative Style:

- ▶ Close to machine
- ▶ Programming by iteration and by modifications of state (for-loops, updates of variables)
- ▶ Side-effects

Functional Style:

- ▶ Close to problem description
- ▶ Program by recursion (no for-loops, no updates of variables)
- ▶ No side-effects

Development environment



- ▶ A text editor of your choice
- ▶ Haskell Platform - <https://www.haskell.org/platform/>
 - ▶ The Glasgow Haskell Compiler
 - ▶ The Cabal build system
 - ▶ 35 core & widely-used packages (libraries)

Development environment (cont.)

- ▶ A word about GHC
 - ▶ a compiler `ghc`
 - ▶ an interpreter `ghci`
 - ▶ a limited set of standard packages (`base`, ...)
- ▶ A word about Cabal
 - ▶ a package manager `cabal`
 - ▶ used both for installing new packages and project management
 - ▶ uses online repository Hackage
- ▶ A word about Hackage (<https://hackage.haskell.org/>)
 - ▶ community's central package archive
 - ▶ contains generated documentation
 - ▶ search engine Hoogle (<https://www.haskell.org/hoogle/>)



Hello World!

- ▶ Because every language tutorial has it, and
- ▶ to show you few other basics.

a file *Main.hs*:

```
module Main where
```

```
{- Simple function to create a hello  
   message. -}
```

```
hello :: String -> String
```

```
hello s = "\n\tHello " ++ s
```

```
-- Entry point of a program
```

```
main :: IO ()
```

```
main = putStrLn (hello "World")
```



Next time



- ▶ one extra lecture, Wednesday the the 20th of January, 11-12AM, **Wolfson LT, QMB**
- ▶ lists and list functions
- ▶ non-strict (lazy) semantics
- ▶ function polymorphism

GHCi demo



`http:
//www.tutorialspoint.com/compile_haskell_online.php`