Haskell presentation

Skye & Ollie

CCMI

December 1, 2025

Contents

Introduction

Haskell is a purely functional programming language. **No side effects, not even printing.**

- Lazy evaluation
- Strong type system
- Declarative style

This makes it ideal for critical applications.

Syntax & Types: The Basics

Everything in Haskell is a function. And all functions in Haskell have a type. Modules are in PascalCase and functions are in camelCase.

```
module MyModule where

binaryAdd :: Int -> Int -> Int
binaryAdd a b = a + b
add2 -> binaryAdd 2

-- in ghci
>> :load MyModule
>> binaryAdd 4 5
9
```

Syntax & Types: Types

Types determine what the expected input and output will be. These can be strict and static (Java, C++) or loose and dynamic (Python, Typescript).
The benefits of Haskell are

- **Type inference** The type checker can figure out exactly what type some data should.
- **Type classes** Similar to a `enum` except for types and can be used in `match` statements.
- **Higher-order types** A “functions” that take in a type and produce a brand new type.

Syntax & Types: Monads

Haskell has a lot of jokes about monads “monads are monoids in the category of endofunctors” or “monads are just borritos”.

Without going into details, these primarily allow you to have side effects in a purely functional language.

How to use

- **Glasgow Haskell Compiler**- Most widely used Haskell complier which supports optimisations and interactive REPL (ghci)
- **Linting** - Tools like hlint suggest idiomatic improvements
- **Hackage** - Central package repository for Haskell libraries which is managed via cabal or stack

Lazy evaluation

Haskell has lazy evaluation built into it. This means that expressions are not evaluated until their values are actually needed. **How it works:**

- Computations are deferred
- Values are represented as thunks
- Evaluation happens only when required for

Makes the code more efficient and can help stop runtime errors :).

Lazy evaluation - code snippet

We show the syntax for creating two mutually dependent lists.
Syntax:

```
main :: IO ()
main = do
 let xs = 0 : ys
     ys = 1 : xs
 print (take 20xs)
```

We will show this in action in VScode and what happens when we try to do it in C++.

Type-checking

It is possible to create a program that does not type check but does in C++

Syntax:

```haskell
import Text.Printf (printf)

main :: IO ()
main = do
 printf "%s" (1::Int)
```

We will show this in action in VScode and what happens when we try to do it in C++.

Applications

Haskell is used in Industry when is crucial that the code is fail-safe. Some applications:

- **Facebook** - anti-Spam systems
- **Github** - Entire blockchain is written in Haskell
- **Cardano Blockchain** - Semantic code for analysis tools

Other key reasons include Maintainability and formal verification.

Conclusion: Why use Haskell?

- Haskell an incredibly powerful language with a very steep learning curve.
- The strong typing and functional nature make it perfect for safe multi-threading.

Landin's knot

Landin's knot is a technique for implementing recursive structures without explicit recursion which is often used when implementing cyclic data structures or self-referential closures.

- **Manual Memory management** - In Haskell Garbage collection automatically handles cycles safely whereas in C++ cyclic structures risk memory leaks
- **Dangling pointers** - References are managed by the runtime system as Haskell uses no raw pointers. In C++, incorrect pointer handling can cause faults!
- **Complexity of implementing laziness**. Laziness is built into Haskell whereas manual tricks are needed in C++ which makes it error prone.

Monads

Monads have the following syntax

```
class Monad m where
  (>>=)    :: m a -> ( a -> m b ) -> m b
  return   ::   a                 -> m a
  -- (>>) :: m a ->  m b          -> m b
```

where

```
return a >>= k                  = k a
m        >>= return             = m
m >>= (\x -> k x >>= h) = (m >>= k) >>= h
```