関数型プログラミング
第1回　HASKELLの概要

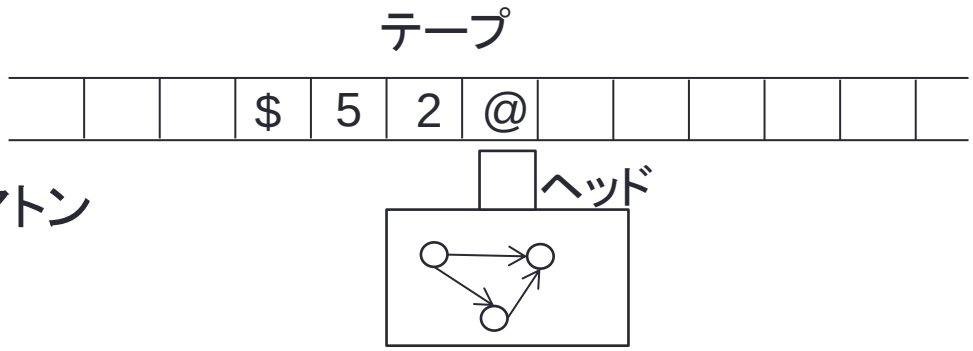
萩野　達也
hagino@sfc.keio.ac.jp

関数型プログラミング

- 手続き型プログラミング
  - 問題を解くための手順を書く
  - プログラムは行ごとに順番に実行される
  - 最も一般的なプログラミング言語
  - 例：FORTRAN, C, Java, Javascript, …
- 論理型プログラミング
  - 問題を解くための論理式を書く
  - 論理式の書く順序はあまり関係ない
  - 副作用がない
  - 代入文がない
  - 例：PROLOG
- 関数型プログラミング
  - 関数を組み合わせて問題を解く
  - 実行の順番は官界内
  - 副作用がない
  - 代入文がない
  - 例：LISP, FP, ML, Haskell

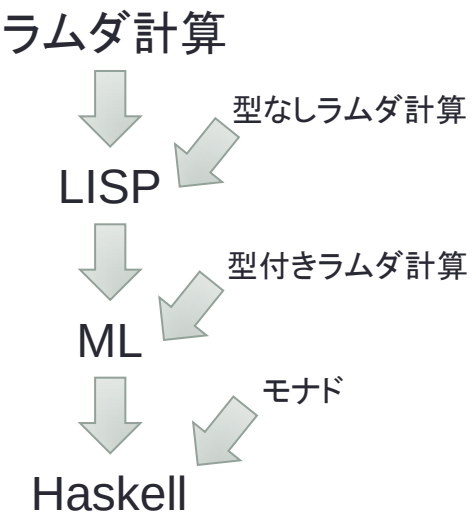
計算のモデル

- チューリング機械
  - 無限のテープと有限状態オートマトン
  - 万能チューリング機械



- 帰納的関数
  - 原始帰納的関数＋最小オペレータ
  - f(x+1) = (x+1)×f(x), f(0) = 1

- ラムダ計算
  - 関数抽象＋関数適用
  - (λx.(λy.xy))(λx.x) → (λy.(λx.x)y) → λy.y



Haskell

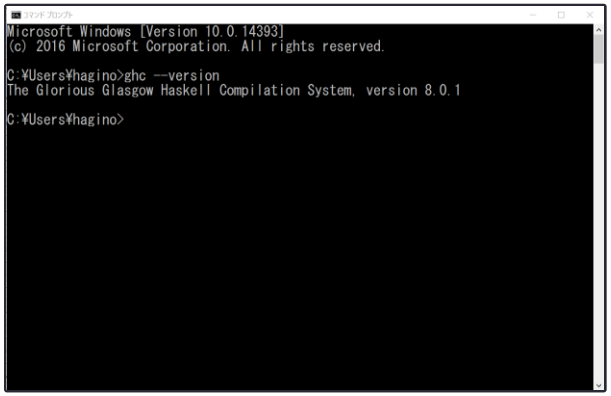
- 純粋は関数型プログラミング言語
  - 副作用がない
  - 参照透過性がある
- 強い型を持つ
  - コンパイル時に型チェックが行われる
- 多相型（Polymorphism）
  - 関数は複数の型に適用可能
- 正格ではない
  - 遅延評価
- モナド
  - 計算の順序を与える

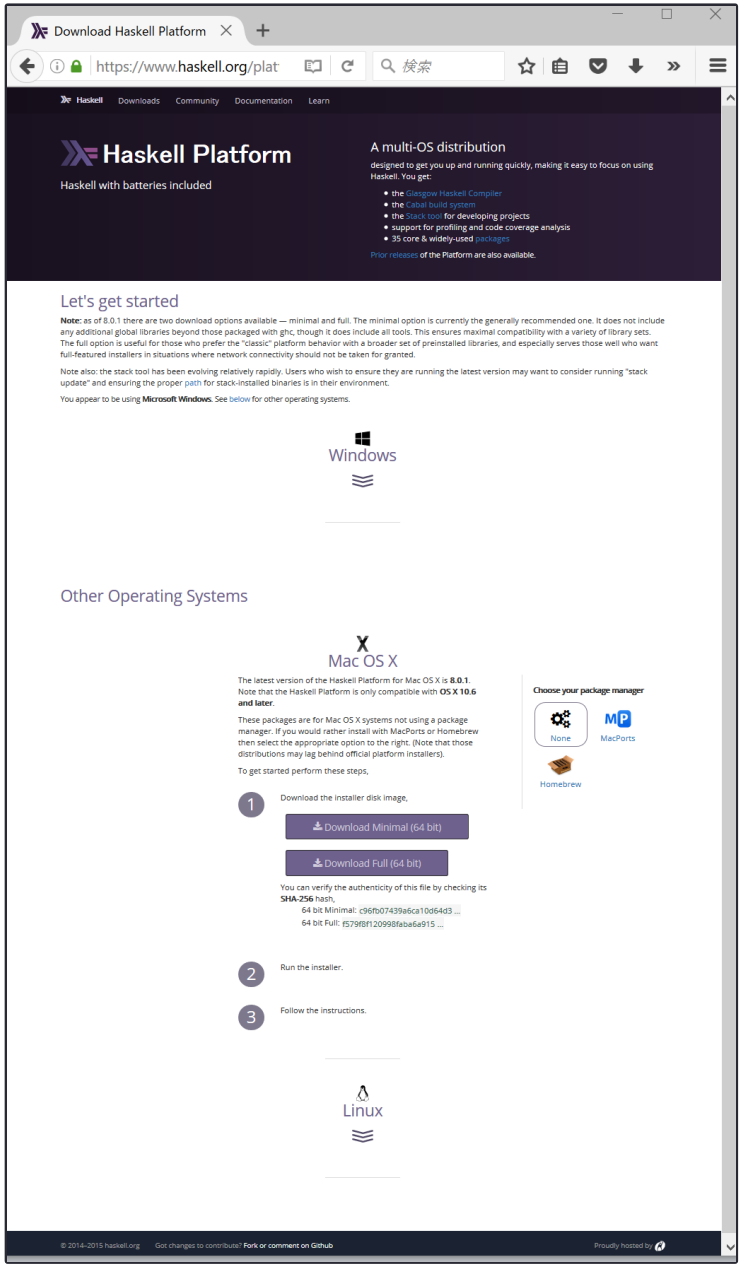
Haskell Brooks Curry

- アメリカの数学者（1900/9/12 ～ 1982/9/1）
- 組み合わせ論理
  - S, K, I
  - ラムダ計算と同値
- Curryのパラドックス
  - 「この文が真なら，サンタクロースは実在する．」
- Curry-Howard対応
  - 論理学 ⇔ 計算
  - 証明はプログラム
- カリー化（Currying）
  - $(A \times B \to C) \to (A \to (B \to C))$

GHC

- Glasgow Haskell Compiler
  - Haskell Platform
  - 実行環境：Windows, Mac OX X, Linux, other UNIX
- インストール
  - https://www.haskell.org/platform/
- コマンドプロンプト（Windows）あるいはターミナル（mac）で正しくインストールできたか確かめる
  - **`ghc --version`**





Hello, World!

- Haskellのプログラムを実行してみる.
  1. 次のプログラムをhello.hsに準備する(テキストエディタを利用).

```
main = putStrLn "Hello, World!"
```

  2. ghcコマンドを使ってコンパイルする(コマンドプロンプト, ターミナル).

```
C:¥> ghc hello.hs
[1 of 1] Compiling Main ( hello.hs, hello.o )
Linking hello.exe ...
```

  3. コンパイルされたプログラムを実行する(コマンドプロンプト, ターミナス).

```
C:¥> hello
Hello, World!
```

直接実行と対話的実行

- プログラムを開発中などは，コンパイルする時間が面倒になる時があります．
- runghcコマンドを使って直接実行することも可能です．

```
C:¥> runghc hello.hs
Hello, World!
```

- 対話的にghcを起動することも可能です．

```
C:¥> ghci
GHCi, version 8.0.1: http://www.haskell.org/ghc/  :? for help
Prelude> putStrLn "Hello, World!"
Hello, World!
Prelude> 1 + 1
2
Prelude> :quit
```

mainアクション

```
main = putStrLn "Hello, World!"
```

- これは変数mainの定義です.
- mainの値は関数ではなく, アクションです.
- putStrLnは関数です.
  - putStrLnの引数は文字列リテラル"Hello, World!"です.
  - putStrLnは文字列を出力するアクションを返す関数です.
- Haskellのプログラムが起動されるとmainのアクションが評価されます.
- 関数の適用に括弧は必要ありません.

```
putStrLn "Hello, World!"
(putStrLn "Hello, World!")
putStrLn("Hello, World!")
```

- 上記のどれも同じです.

2つ以上のアクションを行う

- "Hello, World!" と "Hello, SFC!" を2行に書いてみよう！

```
main = putStrLn "Hello, World!" "Hello, SFC!"
```


```
main = (putStrLn "Hello, World!") "Hello, SFC!"
```

- 動かない

```
main = putStrLn "Hello, World!"
       putStrLn "Hello, SFC!"
```

- 動かない

```
main = (putStrLn "Hello, World!") >> (putStrLn "Hello, SFC!")
```

- 動く

```
main = do putStrLn "Hello, World!"
          putStrLn "Hello, SFC!"
```

- 動く

do構文

- 名前を入力してもらい，出力するプログラム

```
main = do putStrLn "What is your name?"
          name <- getLine
          putStrLn ("Hello, " ++ name ++ "!")
```

つなげるアクションをそろえる必要がある

- **getLine**
  - 標準入力（ターミナル）から1行読み込むアクション
- **name <- getLine**
  - 代入文ではありません．
  - アクション（**getLine**)が成功すると，その値が**name**に束縛される．
- ++
  - 2項演算子
  - 2つの文字列をつなげる

練習問題1-1

- 2人の名前を尋ね，2人が友達であるというメッセージを出すプログラムfriend.hsを書きなさい．

```
% ex1-1
Input a name?
Taro
Input another name?
Hanako
Taro and Hanako are friends.
```

入力

- 宿題は次のサイトから提出してください．

https://vu5.sfc.keio.ac.jp/kadai/

- 宿題の期限はその週の土曜日までです．

- 成績は，宿題と期末試験で評価します．