

第2章:基本データ型

胡 振江

数型

- 整数 Int
 - 42 -42 0 100 ...
- 浮動小数点 Float
 - -42.10 42.10 0.0 100.0
- 算術演算子

+	加算	$2+3 \Rightarrow 5$
-	減算	$2-3.0 \Rightarrow -1.0$
*	乗算	$4*5 \Rightarrow 20$
/	除算	$8/3 \Rightarrow 2.66667$
^	べき乗算	$2^3 \Rightarrow 8$
div	整数除算	$\text{div } 5 \ 3 \Rightarrow 1$
mod	整数剰余	$\text{mod } 5 \ 3 \Rightarrow 2$

結合順位、順序

- 結合順位(異なる演算):
関数適用: 一番強い
^
* / div mod
+-
- 例
 - $3^4 * 5$
 - $3 * 7 + 4.1$
 - square $3 * 4$
- 結合順序(同じの演算)
 - 左結合: $5-4-3$
 - 右結合: 5^4^3
 - 結合性: $5+4+3$

演算子とセクション

- セクション: 括弧でくくられた演算子
 $(+) :: \text{Num } a \Rightarrow a \rightarrow a \rightarrow a$
 $(+) \ x \ y = x + y$
- 更に拡張: 引数を演算子とともに括弧でくくる
 $(x \oplus) \ y = x \oplus y$
 $(\oplus x) \ y = y \oplus x$
 - (*)2): 2倍する 関数
 - (1/): 逆数を求める 関数
 - (/2): 2分する 関数
 - (+1): つぎの値を得る 関数

練習問題

- つぎの関数はどのような引数に対して Trueを返すか。
 - $(==9) \cdot (2+) \cdot (7*)$
- 正しいのはどれ?
 - $(*) \ x = (*x)$
 - $(+) \ x = (x+)$
 - $(-) \ x = (-x)$

例題: 平方根の計算sqrt

- 仕様
 - $X \geq 0$ の時、 $\text{sqrt } x \geq 0$ かつ $(\text{sqrt } x)^2 = x$
 - この仕様が平方根を計算する方法を述べているのではない
 - 実際の計算機上の有限精度の算術計算を許容しない。



小さい $\text{eps} > 0$ に対して
 $\text{sqrt } x \geq 0$ かつ $\text{abs}((\text{sqrt } x)^2 - x) \leq \text{eps}$

例題: 平方根の計算sqrt (cont)

■ Newton法

$$y(n+1) = (y(n) + x / y(n)) / 2$$

例: 2の平方根の計算

$$\begin{aligned} y(0) &= 2 \\ y(1) &= (2+2/2)/2 = 1.5 \\ y(2) &= (1.5+2/1.5)/2 = 1.4167 \\ y(3) &= (1.4167+2/1.4167)/2 = 1.4142157 \\ &\dots \end{aligned}$$

例題: 平方根の計算sqrt (cont)

■ 近似値yから新しい近似値を生成する関数

$$\text{improve } x \ y = (y + x/y) / 2$$

■ 終了条件を判定する関数

$$\text{satis } x \ y = \text{abs } (y^2 - x) < \text{eps}$$

■ ある条件pが真になるまで初期値に関するfを繰り返し適用する関数

$$\begin{aligned} \text{until } p \ f \ x \mid p \ x = x \\ \mid \text{otherwise} = \text{until } p \ f \ (f \ x) \end{aligned}$$

■ メイン関数

$$\text{sqrt } x = \text{until } (\text{satis } x) \ (\text{improve } x) \ x$$

プログラム sqrt.hs

```
sqrt1 x = until (satis x) (improve x) x
where
```

$$\text{improve } x \ y = (y + x/y) / 2$$

$$\text{satis } x \ y = \text{abs } (y^2 - x) < \text{eps}$$

$$\text{eps} = 0.0001$$

単純な関数の組み合わせ → 修正しやすくなる

(教科書 p.24, 練習問題 2.1.7)

論理型 Bool

■ 論理値: True, False

■ 述語: 論理値を返す関数

■ E.g. even :: Int → Bool

■ 比較演算子

== 等しい 1==1
 /= 等しくない True /= False
 < より小さい
 > より大きい
 <= より小さいかまたは等しい
 >= より大きいまたは等しい

■ 論理演算子

&& 論理和 and p q (p `and` q)
 || 論理積 or p q
 not 論理否定 not p

例題: 閏年の判定

- 閏年とは、4で割り切れる年であるが、100で割り切れるならば400でも割り切れなくてはならない。

$$\begin{aligned} \text{leap } y &= y \text{ `mod` } 4 == 0 \ \&\& \\ &\quad (y \text{ `mod` } 100 /= 0 \mid y \text{ `mod` } 400 == 0) \end{aligned}$$

または

$$\begin{aligned} \text{leap } y \mid y \text{ `mod` } 100 == 0 &= y \text{ `mod` } 400 == 0 \\ \mid \text{otherwise} &= y \text{ `mod` } 4 == 0 \end{aligned}$$

その他の例題

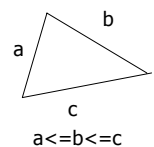
analysis a b c

$$\mid a+b<=c \quad = 0$$

$$\mid a==b \ \&\& \ b==c \quad = 1$$

$$\mid a/=c \ \&\& \ (a==b \mid b==c) \quad = 2$$

$$\mid a<b \ \&\& \ b<c \quad = 3$$



$$\text{pred } n \mid n==0 \quad = 0$$

$$\mid n>0 \quad = n-1$$

練習問題

- 3個の数を取り、そのうちの大きい2数の2乗の和を返す関数 `sumsq` を定義せよ。

文字と文字列

- 文字型 `Char`
`'a'` `'7'` `''`
- 基本関数
 - `ord :: Char → Int`
 - `chr :: Int → Char`
 - 例: `ord 'b' → 98`
`chr 98 → 'b'`
`chr (ord 'b' + 1) → 'c'`

例題

- 文字が数字であることを判定する関数
`isDigit x = '0' <= x && x <= '9'`
- 小文字を大文字に変える関数
`capitalise x`
 - | `isLower x` = `chr (offset + ord x)`
 - | otherwise = `x`where `offset = ord 'A' - ord 'a'`

文字列

- 文字列型 `String` (e.g., `[Char]`)
`"a"` `"hello"`
- 文字列の比較は通常の辞書式順に従う
`"hello" > "hallo"`
`"Jo" < "Joanna"`
- 関数
 - `show :: a → String`
`show 100 → "100"`
`show True → "True"`
`show (show 100) → ""100""`
 - 文字列をつなぐ演算子 `++`
`"hello" ++ " " ++ "world" → "hello world"`

組

- (T_1, T_2, \dots, T_n)
 - $(17.3, '+') :: (\text{Float}, \text{Char})$
 - $(3, 6) :: (\text{Int}, \text{Int})$
- 順序: 辞書式順序
 - $(("s", 4) < ("s", 5))$
- 関数
 - `fst (x, y) = x`
 - `snd (x, y) = y`

例題1: 2次方程式

- 2次方程式の根を求める関数

$$\begin{array}{ccc} a x^2 + b x + c = 0 & & (a, b, c) \\ \downarrow & & \downarrow \\ \text{Let } d = b^2 - 4ac. & & \\ \text{If } d \geq 0 \text{ then} & & (r_1, r_2) \\ r_1 = (-b + \sqrt{d}) / 2a & & \\ r_2 = (-b - \sqrt{d}) / 2a & & \end{array}$$

```

roots :: (Float,Float,Float) → (Float,Float)
roots (a,b,c) | d>=0 = (r1,r2)
  where
    r1 = (-b+r) / (2*a)
    r2 = (-b-r) / (2*a)
    r  = sqrt d
    d  = b^2 - 4*a*c

```

例題2:有理数

- 有理数の表現: 対
 $x/y \rightarrow (x,y)$
- 問題:
 - 有理数の正規化 $(18,16) \rightarrow (9,8)$
 - 有理数の四則演算
 - 有理数の比較
 - 有理数の表示

有理数の正規化

$$\frac{x}{y} = s(x,y) \frac{|x|/\gcd(|x|,|y|)}{|y|/\gcd(|x|,|y|)}$$

```

norm (x,y)
  | y /= 0 = (s * (u `div` d), v `div` d)
  where u = abs x
        v = abs y
        d = gcd u v
        s = sign (x*y)

sign x | x>0 = 1
      | x==0 = 0
      | x<0  = -1

```

有理数上の四則演算

```

radd (x,y) (u,v) = norm (x*v+u*y,y*v)
rsub (x,y) (u,v) = norm (x*v-u*y,y*v)
rmul (x,y) (u,v) = norm (x*u,y*v)
rdiv (x,y) (u,v) = norm (x*v,y*u)

```

有理数の比較

```

compare' op (x,y) (u,v) = op (x*v) (y*u)

requals = compare' (==)
rless   = compare' (<)
rgreater = compare' (>)

```


有理数の表示

```
showrat (x,y)
= if v==1 then show u
  else show u ++ "/" ++ show v
where (u,v) = norm (x,y)
```

パターン

- 等式の左側にパターンを用いて関数を定義することができる。
- 論理値パターン

```
cond True x y = x
cond Flase x y = y
```




```
cond p x y | p == True  = x
           | p == Flase = y
```

- 自然数(負でない整数)パターン

```
pred 0 = 0
pred (n+1) = n

count 0 = 0
count 1 = 1
count (n+2) = 2
```

関数

- 関数はあらゆる型の値を引数にとりうるし、あらゆる種類の値を結果として返すことができる。
 - 例: 高階関数
 - 引数として関数をとる、あるいは
 - 結果として関数を返す
- 微分演算子: 引数 - 関数
結果 - 導関数
- 

関数合成

- 二つの関数を合成する演算子 .
 $(.) :: (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow (a \rightarrow c)$
 $(f . g) x = f (g x)$
- 結合的
 $(f . g) . h = f . (g . h)$

演算子と関数

- 二項演算子は関数とよく似ている。異なる点は2つの引数の前に置くのではなく間に書くということだけある。
 $\otimes \oplus \clubsuit \diamond \heartsuit \spadesuit$
- セクション: 演算子 \rightarrow 関数
 $2 + 3 \rightarrow (+) 2 3 \rightarrow (2+) 3 \rightarrow (+3) 2$
- バッククオート: 二引数関数 \rightarrow 演算子
 $\text{div } 5 \ 3 \rightarrow 5 \text{ `div ` } 3$

逆関数

- 単射関数
 - $\forall x \text{ in } A, y \text{ in } B. f\ x == f\ y \rightarrow x == y$
- 全射関数
 - $\forall y \text{ in } B, \exists x \text{ in } A. f\ x = y$
- 逆関数
 - $f^{-1}(f\ x) = x$
 - 例 $f\ x = (\text{sign } x, \text{abs } x)$
 $f^{-1}(s,a) = s*a$

正格関数と非正格関数

- 正格関数
 - 定義: $f\ \perp = \perp$
 - 例: $\text{square } (1/0) = \perp$
- 非正格関数: 正格でない関数
 - 例: $\text{three } x = 3$
 $> \text{three } (1/0)$
 3

非正格な意味論の利点

- 相等性に関する議論しやすい
 - $2 + \text{three } x = 5$
 - 単純で統一的な置換操作
 - プログラムの正当性を議論しやすい
- 関数を定義して、新たら制御構造を定義することができる。
 - $\text{cond } p\ x\ y \mid p = x$
 $\mid \text{otherwise} = y$
 - $\text{recip } x = \text{cond } (x=0)\ 0\ (1/x)$
 - 正格な意味論では $\text{recip } 0 = \perp$
 - 非正格な意味論では $\text{recip } 0 = 0$

簡約戦略

$f\ x_1\ x_2 \dots x_n$ の評価

- 先行評価
 - 引数優先評価戦略
 x_1, x_2, \dots, x_n を評価したら f を評価する。
- 遅延評価
 - (外側の) 関数優先評価戦略
 f をまず評価する。

型の同義名

- 距離、角度、位置を引数にとり、角度と距離で示される新しい位置に場所を移動する関数 `move`:
 $\text{move} :: \text{Float} \rightarrow \text{Float} \rightarrow (\text{Float}, \text{Float}) \rightarrow (\text{Float}, \text{Float})$
 $\text{move } d\ a\ (x,y) = (x+d*\cos\ a, x+d*\sin\ a)$

 $\text{type Position} = (\text{Float}, \text{Float})$
 $\text{type Angle} = \text{Float}$
 $\text{type Distance} = \text{Float}$

 $\text{move} :: \text{Distance} \rightarrow \text{Angle} \rightarrow \text{Position} \rightarrow \text{Position}$

型推論

- 適用規則
 - $f\ x :: t \rightarrow \exists t'. x :: t', f :: t' \rightarrow t$
- 相等性規則
 - $x :: t, x :: t' \rightarrow t = t'$
- 関数の規則
 - $t \rightarrow u = t' \rightarrow u' \rightarrow t = t', u = u'$

