

関数束縛の基本形

プログラミングAIII

2025年度講義資料 (3)

新潟大学 工学部工学科 知能情報システムプログラム

青戸等人

関数束縛の基本形

```

1 # fun 関数名 仮引数 = 式 ;      (*ユーザの入力*)
2 val 関数名 = fn : 関数の型
3                                         (*インターフリタからの返答*)

```

- これによって、関数名に関数が束縛される。関数名や仮引数に使える識別子は、変数名のときと同じ。
- 変数のときはその値が表示されたが、関数の値は表わせないので、どの場合も fn と表示される。
- 関数の型 $X \rightarrow Y$ は、関数が X 型の値を受けとて、 Y 型の値を返すことを表わす。

目次

① 関数の束縛

② 組型

③ 基本データ型

4 / 26

関数の束縛

```

# fun double x = x * 2;
val double = fn : int -> int
# double;
val it = fn : int -> int
# double 3 + 5;
val it = 11 : int
# double (double 3);
val it = 12 : int
# double double 3;
(val it : int)
(interactive):6.0-6.12(85) Error:
  (type inference 016) operator and operand don't agree
    operator domain: int
    operand: int -> int

```

- 関数定義により、その識別子が束縛される。
- 関数名を入力すると、関数の型が見える。
- 関数適用の結合力は最も強い。** したがって、中置演算子より、関数適用の結合力は強い。
- 関数適用は左結合。** つまり、「 $f g x$ 」は「 $(f g) x$ 」の省略形。

目次

① 関数の束縛

② 組型

③ 基本データ型

5 / 26

実習課題(1)：関数定義の基本

- ① 与えられた n 時間の秒数を返す関数 hoursToSeconds を定義せよ。

```
# hoursToSeconds 3;
val it = 10800 : int
```

- ② 与えられた文字列を、括弧で囲む関数 addPar を定義せよ。

```
# addPar "aaa";
val it = "(aaa)" : string
```

- ③ 与えられた整数が正の数なら true, そうでないなら false を返す関数 isPositive を定義せよ。

```
# isPositive 3;
val it = true : bool
```

関数によるプログラミング

2 / 26

6 / 26

実習課題(1)：関数定義の基本

- ④ 先の問題3は if 文を使わずに定義できる。もし先の解答で if 文を使っていた場合には、if 文を使わない解答を与えて。

- ⑤ "apple"なら"orange"を返し、"orange"なら"apple"を返し、それ以外ならそのまま引数文字列を返す exchangeAppleAndOrange を定義せよ。

```
# exchangeAppleAndOrange "apple";
val it = "orange" : string
# exchangeAppleAndOrange "orange";
val it = "apple" : string
# exchangeAppleAndOrange "banana";
val it = "banana" : string
#
```

```

# fun double x = x * 2;      (* 関数 double の定義 *)
val double = fn : int -> int
# double 3;                  (* 関数 double の利用 *)
val it = 6 : int

```

- 関数型プログラミング言語では、「関数」を自分で定義することにより、目的の計算を実現する。
- さまざまな繰り返し構造や制御構造は、「関数」を使って実現できる。
- 関数型プログラミング言語(最初の段階)では、「プログラミング」 = 「関数を定義する」

目次

組型の実行例 (2)

```
# val pair = (4,3); sqdiff pair;
val pair = (4, 3) : int * int (* 組型の変数 *)
val it = 7 : int
# val (x,y) = mkincpair 2; (* 組型のパターン *)
val x = 2 : int
val y = 3 : int
# (1) = 1; (* 1つ組は組型でない *)
val it = true : bool
# (1,2,3); (1,(2,3)); ((1,2),3); (* 組のネスト *)
val it = (1, 2, 3) : int * int * int
val it = (1, (2, 3)) : int * (int * int)
val it = ((1, 2), 3) : (int * int) * int
# #1 (1,2); (* 要素の取り出し *)
val it = 1 : int
```

8 / 26

12 / 26

組型 (教p.22-23) (1)

関数によっては、複数の引数をとりたいことがよくある。数学では、複数の対象と一緒に扱うときには、対や組といった構造が用いられる。関数プログラムでも同じ。

組型の基本 (1)

- 式 e_1, \dots, e_k の組を (e_1, \dots, e_k) と書く (ただし, $k \geq 2$).
- 式 e_1, \dots, e_k の型をそれぞれ A_1, \dots, A_k とするとき、組 (e_1, \dots, e_k) の型は $A_1 * \dots * A_k$ となる。 A_1, \dots, A_k は同じ型でも異なる型でもよい。
- 式 e_1, \dots, e_k の値をそれぞれ v_1, \dots, v_k とするとき、組 (e_1, \dots, e_k) の値は (v_1, \dots, v_k) となる。
- 組 (e_1, \dots, e_k) と組 (e'_1, \dots, e'_k) が等しくなるのは、 $i = 1, \dots, k$ のそれぞれで式 e_i と e'_i が等しい値をもつ場合。

実習課題(2)：組を用いた関数定義

- 与えられた整数の対 (x, y) に対して、 $x + y$ が偶数かを布尔型で返す関数 `hasEvenSum` を定義せよ。
- 与えられた整数の対 (x, y) に対して、 $x \bmod y$ と $x \bmod y$ の対を返す関数 `modAndDiv` を定義せよ。
- 与えられた整数の対 (x, y) に対して、(小さい方、大きい方)の対を返す関数 `orderPair` を定義せよ。
- 3つの文字列が与えられて、その連結を返す関数 `concat3` を定義せよ。
- 整数の3つ組 (x, y, z) (ただし, $0 \leq x < 24, 0 \leq y < 60$ とする) が与えられたときに、 x 時 y 分から z 分後の時刻が x' 時 y' 分となるような (x', y') を計算する関数 `ellipseMinutes` を定義せよ。

9 / 26

13 / 26

組型の実行例 (1)

```
# (9,12); (1, 2.0, "aaa"); (* さまざまな型の組 *)
val it = (9, 12) : int * int
val it = (1, 2.0, "aaa") : int * real * string
# (4 + 2, 2.0 * 3.0); (* 組型の式の評価 *)
val it = (6, 6.0) : int * real
# (1+2, 2+3) = (13 mod 5, 10 div 2);
val it = true : bool (* 組型の式の同値性 *)
# fun sqdiff (x,y) = x * x - y * y; (* 2引数の関数 *)
val sqdiff = fn : ['a::{int, ...}. 'a * 'a -> 'a]
# sqdiff (3,2);
val it = 5 : int
# fun mkincpair x = (x, x + 1); mkincpair 2;
val mkincpair = fn : int -> int * int (* 組を返す関数 *)
val it = (2, 3) : int * int
```

目次

① 関数の束縛

② 組型

③ 基本データ型

10 / 26

14 / 26

組型 (教p.22-23) (2)

基本データ型

組型の基本 (2)

- 組を1つの式として扱うことができる。変数に束縛したり、関数に与えたり、関数の結果として返すことができる。
- 組をパターンとして使うことができる。つまり、変数束縛 `val =` で変数の組を左辺におくことができる。
- 括弧で囲んだ中がコンマ(,)で区切られていないければ組型にはならない。つまり(1)と1は等しい。
- 組の要素が組であってもよい(組がネストしていくてもよい)。ただし、結合が異なる組は区別される：

$$(a, b, c) \quad ((a, b), c) \quad (a, (b, c))$$

は異なる式。それぞれの型は、

$$A*B*C \quad (A*B)*C \quad A*(B*C)$$

これらの型では、括弧の有無に意味があることに注意。

- SMLの型には、それ以上分解できないような **基本データ型** と、すでにある型から作られているような型(**複合型** という)の2種類がある。
- 1つ前でみた組型はもっとも基本的な複合型の1つである。
- ここでは、前回の講義で学習した整数型や文字列型を始めとする、さまざまな基本データ型と、それらの型に用意されている関数や演算について、少し詳しく見ていく。

11 / 26

15 / 26

単位(unit)型 (教4.1節)

単位(unit)型

- 単位型は、ただ1つの値「()」をもつ型。
- print文は、引数の文字列を副作用として表示し、値()を返す。このように、単位型は、副作用が目的である場合など、評価された値が重要でない時に用いられる。

```
# () ; print;
val it = () : unit
val it = fn : string -> unit
# print "Hello\nWorld!";
Hello
World!val it = () : unit
# print "AAA" = print "aaa";
AAAaaaaval it = true : bool
```

真理値(bool)型 (教4.2節) (1)

論理演算子

- andalso, orelse, not が、論理積、論理和、否定のブール演算。

```
# not true; not false;
val it = false : bool
val it = true : bool
# true andalso true; true andalso false; false andalso true; false andalso false
val it = true : bool
val it = false : bool
val it = false : bool
val it = false : bool
# true orelse true; true orelse false; false orelse true; false orelse false;
val it = true : bool
val it = true : bool
val it = true : bool
val it = false : bool
```

真理値(bool)型 (教4.2節) (2)

論理演算子の結合力

- andalso と orelse は最も弱い結合力をもつ。
- ただし、andalso の方が orelse より結合力が強い。

```
# true orelse true andalso false;
val it = true : bool
# (true orelse true) andalso false;
val it = false : bool
# false = true andalso false;
val it = false : bool
# false = (true andalso false);
val it = true : bool
#
#
```

真理値(bool)型 (教4.2節) (3)

論理演算子の評価順序

- e_1 andalso e_2 では、 e_1 が評価されて、その値がtrueなら、 e_2 の値が評価されて、その値が返る。 e_1 の値がfalseなら、 e_2 の値が評価はスキップされて、全体の値(false)が返る。
- e_1 orelse e_2 も同様
- if e_0 then e_1 else e_2 では、 e_0 が評価されて、その値がtrueなら、 e_1 の値が評価されて、全体の値が返る。 e_1 の値がfalseなら、 e_2 の値が評価されて、全体の値が返る。

```
# not (print "a" = print "b") andalso (print "c" = print "d");
abval it = false : bool
# (print "a" = print "a") andalso (print "c" = print "d");
aacdval it = true : bool
#
```

整数型 (教4.3節)と実数型 (教4.4節)

0 <= 1; 0 >= 1; 0 <> 1;
 val it = true : bool
 val it = false : bool
 val it = true : bool
 # abs ~2;
 val it = 2 : int
 # real 12;
 val it = 12.0 : real
 # floor 12.9; ceil 12.1;
 val it = 12 : int
 val it = 13 : int

- 整数に関する数のいろいろ(教科書p.64-65)。
- 実数に関する数のいろいろ(教科書p.65)。
- real型は浮動小数点表現された実数の型。表現と精度は処理系依存。
- 実数については、等価性判定(=)は利用不可。

文字(char)型 (教4.5節)と文字列(string)型 (教4.6節)

```
# print "a\tc\naa\tc\n\n";
a c
aa c
val it = () : unit
# "Hello, \
> \World!";
val it = "Hello, World!" : string
# "Hello," ^ " World!";
val it = "Hello, World!" : string
# size "abcde";
val it = 5 : int
# substring ("abcde",2,3);
val it = "cde" : string
#
```

- 特殊文字のいろいろ(教科書p.66)
- 文字に関する関数のいろいろ(教科書p.67)
- 長い文字列の入力
- 文字列に関する関数のいろいろ(教科書p.68)
- 何番目かは0からカウント

基本ライブラリ(教12章)

トップ環境で使える関数以外にも、さまざまな有用な関数が基本ライブラリに用意されている。そのような関数の多くは、カテゴリ別にストラクチャ(structure)とよばれるモジュールの中に用意されている。そのような関数を用いるには、

「ストラクチャ名.関数名」

の形を用いる。

- Int ストラクチャ... 整数関連の関数など
- Real ストラクチャ... 実数関連の関数など
- Char ストラクチャ... 文字関連の関数など
- String ストラクチャ... 文字列関連の関数など

基本ライブラリ関数の利用例(1)

- 同じ名前をもつ関数が、別のストラクチャに含まれていることもある。
- ストラクチャの中にサブストラクチャがあることもある。例えば、Real ストラクチャの中には Math サブストラクチャがある。

```
# Int.toString 12;
val it = "12" : string
# Int.max (9,12);
val it = 12 : int
# Real.toString 12.0e~3;
val it = "0.012" : string
# Real.Math.pi;
val it = 3.14159265359 : real
# Real.Math.sin (Real.Math.pi / 2.0);
val it = 1.0 : real
# Real.Math.sqrt 3.0;
val it = 1.73205080757 : real
```

基本ライブラリ関数の利用例(2)

実習課題(3)：基本データ型の関数の利用

```
# Char.isAlpha;
val it = fn : char -> bool
# Char.isSpace; Char.isAlphaNum;
val it = fn : char -> bool
val it = fn : char -> bool
val it = fn : char -> bool
# Char.isLower; Char.isUpper;
val it = fn : char -> bool
val it = fn : char -> bool
# Char.contains;
val it = fn : string -> char -> bool
# Char.contains "alpha" #"h";
val it = true : bool
# String.isPrefix;
val it = fn : string -> string -> bool
# String.isPrefix "ab" "abc";
val it = true : bool
```

実習課題(3)：基本データ型の関数の利用

- ❶ 与えられた整数の対(x,y)に対して、xとyの両方が偶数であるかをブール型で返す関数 isBothEven を定義せよ。
- ❷ 与えられた文字列を表示して改行する関数 prStringLn を定義せよ。

```
# prStringLn "hello";
hello
val it = () : unit
#
❸ (print "a" = ()) orelse (print "b" = ())
を評価せよ。何がわかるか。同様にして、e1 andalso e2において、e1の値がfalseとなるとき、e2は評価されないこと確かめよ。
```