プログラミング言語 No.2-2

2025.4.18 中川博之

課題の解答例 演習問題2-1-1)

2-1-1) 整数x の立方を返す関数cube

- 「立方」とは3乗のこと
  x の立方は x * x * x.

解答例：

```
fun cube x = x * x * x;
```

- * は（何も指定しなければ）整数の乗算

課題の解答例 演習問題2-1-1) 補足

2-1-1a) 実数x の立方 の場合は？

- 「実数」という点がポイント
- * は何も指定しなければ整数の乗算
  - 実数の計算をさせたい場合，明示的にどこかで実数だと指定する必要あり

  → 引数のxが実数を扱うことを明示的に指定

解答例：

```
fun realCube (x:real) = x * x * x;
```

課題の解答例 演習問題 2-1-2)

2-1-2) 3 つの整数を引数として取り，そのうち最小のものと最大のものの対(pair)を返す関数minmaxpair

```
fun minmaxpair (a, b, c) =
    if a > b then
        if a > c then
            if b > c then (c, a)
            else      (b, a)
        else      (b, c)
    else
        if b > c then
            if a > c then (c, b)
            else      (a, b)
        else      (a, c);
```

パターン

パターン

- 入力によって評価する式が違う場合：
  - （if-else文でも書けるが）パターンを用いた関数定義が一般的

```
fun <関数名> <最初のパターン> = <最初の式>
|   <関数名> <2番目のパターン> = <2番目の式>
|   ...
|   <関数名> <最後のパターン> = <最後の式>;
```

パターンを使わない例：

```
fun fact n =
if n = 1 then 1
else n * fact (n-1);
```

パターンを使った例：

```
fun fact 1 = 1
| fact n = n * fact (n-1);
```

ML で使えるパターン

1. 定数　例えば １ ０ 0.0 []
2. 変数　例えば x L xs
3. コンス演算子を用いた式

   例 x::y::zs

   注：コンス演算子は右結合

   x::y::zs = x::(y::zs)

注意点：同じ変数を一つのパターンの中で2回使ってはいけない

× x::x::xs

○ x::y::xs

パターンと if-else文の違い

- 場合分けという意味で，基本的な考え方は同じ
- 処理系での実装が異なる
  - if-else文：関数呼び出し
    - 頭部や尾部を参照するためにhdやtlといった関数を呼び出さなくてはいけない
  - パターン：パターン照合
    - パターン照合でリストの頭部や尾部を参照できる

パターンの方が速い
再帰構造もわかりやすい

パターンの注意点

- パターンはすべて同じ型
- ＝の右の式もすべて同じ型
- 一致するかどうかを上から照合し，一致するものがあったらその式を評価
- MLの警告メッセージ：Warning: match not exhaustive
    - パターンがすべての場合を網羅していない時に出る

パターンを使った関数例

リストを逆順にする関数

```
fun reverse [] = []
| reverse (x::xs) = (reverse xs)@[x];
```

パターンを使った関数例

リストを逆順にする関数

```
fun reverse [] = []
| reverse (x::xs) = (reverse xs)@[x];
```

1つ目のパターン：空リストを逆順にしても空リスト

2つ目のパターン：尾部を逆順にしたものの後ろに頭部の要素を連結したものが逆順にしたリスト

パターンを使った関数例

リストを逆順にする関数

```
fun reverse [] = []
| reverse (x::xs) = (reverse xs)@[x];
```

1つ目のパターン：空リストを逆順にしても空リスト

2つ目のパターン：尾部を逆順にしたものの後ろに頭部の要素を連結したものが逆順にしたリスト

@はリストとリストの連結なので，[x]とすることで要素一つのリストを作って連結している．
（リストの連結を使わないバージョンのreverseはもう少し後で紹介する．）

関数 reverse の評価

[1, 2, 3] を　[3, 2, 1] にする.

```
fun reverse [] = []
| reverse (x::xs) = (reverse xs)@[x];
        - reverse [1,2,3];
```

まず，2つ目のパターンにマッチ

reverse [2, 3] @ [1]

また，2つ目のパターンにマッチ

reverse [3] @ [2]

また，2つ目のパターンにマッチ

reverse [] @ [3]

ここで，1つ目のパターンにマッチ

[]

関数 reverse の評価

[1, 2, 3] を　[3, 2, 1] にする.

```
fun reverse [] = []
| reverse (x::xs) = (reverse xs)@[x];
        - reverse [1,2,3];
```

まず，2つ目のパターンにマッチ

reverse [2, 3] @ [1]

また，2つ目のパターンにマッチ

reverse [3] @ [2]

また，2つ目のパターンにマッチ

reverse [] @ [3]

ここで，1つ目のパターンにマッチ

[]

[3, 2, 1]

評価の結果

[3, 2] @ [1]

評価の結果

[3] @ [2]

評価の結果

[] @ [3]

評価の結果

再帰

再帰

- 定義中に自分自身への参照があるもの
  - 関数の再帰呼び出し：関数の定義の中で自分自身を呼んでいる
  - リストの再帰的定義：リストの記述中にリストが出てくる
- 「再帰」が意味のあるものになるためには条件がある

再帰の構成

1. 基底 (基礎) (basis) ：再帰を必要としない十分に小さな部分の定義
2. 帰納 (帰納段階) (inductive step) ：基底では定義できない部分の定義
   - 自分自身を再帰的に参照することにより定義する
   - 再帰的参照を繰り返すごとで，最終的に基底に辿り着く

再帰関数で計算が停止する (＝ 結果が得られる) ためには

a) 帰納段階で呼ばれる度に引数が小さくなること
b) 最終的に基底が呼ばれること

再帰的定義の例

- リスト
  a) [] はリスト
  b) リストの頭に要素を加えたものはリスト

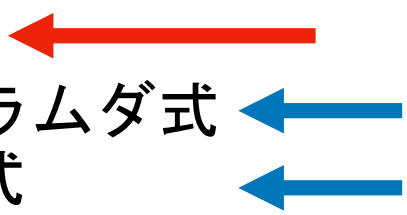
- ラムダ式
  a) 変数 x0, x1, … はラムダ式
  b) Mがラムダ式で x が変数のとき (λx. M) はラムダ式
  c) MとNがラムダ式のとき　(M N) はラムダ式

再帰的定義の例

- リスト
  a) [] はリスト
  b) リストの頭に要素を加えたものはリスト

基底
帰納

- ラムダ式
  a) 変数 x0, x1, … はラムダ式
  b) Mがラムダ式で x が変数のとき (λx. M) はラムダ式
  c) MとNがラムダ式のとき　(M N) はラムダ式



基底
帰納

再帰関数の例

階乗を計算する関数fact

1. 基底：
   n が 1 の時，結果は無条件に 1.
   (再帰呼び出しは行われない）

2. 帰納：
   基底以外の場合 n * fact (n-1).
   再帰的に fact を評価する．その時引数はn-1,
   つまり，呼び出すごとに引数の値が一つずつ小さくなっていく.
   最終的に基底が呼び出される.

注）この関数factは引数に0以下の値を入力すると，
無限に再帰呼び出しを行い停止しない (結果が得られない)

再帰関数の例2

整数のリストの要素の値の合計を計算する関数sum

```
fun sum [] = 0
| sum (x::xs) = x + sum (xs);
```

1. 基底
   リストが空のとき，結果は無条件に0.

2. 帰納
   リストが x::xs のとき，結果は引数をxsとして自分自身を呼び出した結果に x を足したもの.

リストの長さが再帰呼び出しの度に小さくなって最終的に空リストになる（基底に到達する）

再帰関数の例3

リストを逆順にする関数 reverse

```
fun reverse [] = []
| reverse (x::xs) = reverse (xs) @ [x];
```

1. 基底
   リストが空のとき，結果は無条件に空リスト.

2. 帰納
   リストが x::xs のとき，結果は引数をxsとして自分自身を呼び出した結果の 後ろに [x] を足したもの.

リストの長さが再帰呼び出しの度に小さくなって最終的に空リストになる（基底に到達する）

例題: 問題 2-2-1)

リストの各要素を複製する関数 cpelm.
　　リスト[a1, a2, …,an]が与えられたら
　　[a1, a1, a2, a2, …, an, an]を返す.

例題: 問題 2-2-1)　解答例

リストの各要素を複製する関数 cpelm.
　　リスト[a1, a2, …,an]が与えられたら
　　[a1, a1, a2, a2, …, an, an]を返す.

```
fun cpelm []= []
| cpelm (x::xs) = x::x::cpelm(xs);
```

例題: 問題 2-2-1)　解答例

リストの各要素を複製する関数 cpelm.
　　リスト[a1, a2, …,an]が与えられたら
　　[a1, a1, a2, a2, …, an, an]を返す.

```
fun cpelm []= []
| cpelm (x::xs) = x::x::cpelm(xs);
```

パターンを使わないならば

```
fun cpelm (L) = if L = [] then []
                else hd(L)::hd(L)::cpelm(tl(L));
```

例題: 問題 2-2-1)　解答例

リストの各要素を複製する関数 cpelm.
　　リスト[a1, a2, …,an]が与えられたら
　　[a1, a1, a2, a2, …, an, an]を返す.

```
fun cpelm []= []
| cpelm (x::xs) = x::x::cpelm(xs);
```

コンス演算子ではなく連結演算子を使うと

```
fun cpelm []= []
| cpelm [x::xs] = [x,x]@cpelm(XS);
```

しかし，コンス演算子の方が早いし分かりやすいのでおすすめ

例題: 問題 2-2-2)

整数iとリストLを取り，Lをi回巡回する関数 multicycle.
L=[a_1, a_2, …, a_n]のとき，結果は [a_{i+1}, a_{i+2}, …, a_n, a_1, a_2, …, a_i] となる.

考え方：
まず，リストLを1回巡回する関数を作る.
その関数を i回呼び出す.

例題: 問題 2-2-2)　解答例

整数iとリストLを取り，Lをi回巡回する関数 multicycle.
L=[a_1, a_2, …, a_n]のとき，結果は [a_{i+1}, a_{i+2}, …, a_n, a_1, a_2, …, a_i] となる.

```
fun cyclelist [] = []
| cyclelist (x::xs) = xs @ [x];
fun multicycle (1, L) = cyclelist L
| multicycle (i, L) = multicycle (i-1, cyclelist L);
```

問題 2-2-3) n個から m個を選ぶ組み合わせの数

$(n \geqq m > 0)$

$$\binom{n}{m}$$

$$\binom{n}{m} = \frac{n!}{(n-m)!\,m!}$$

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

n個の中からm個を選ぶ組み合わせの数は
n個のうちの最初の1個を選ばなかった場合と選んだ時の和

a) 最初の1個を選ばなかったので, n-1個の中からm個を選ぶ場合
b) 最初の1個を選んだので, 残りの n-1個の中からm-1個を選ぶ場合

問題 2-2-3) 組み合わせの数を返す関数comb

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

基底は　m = 0 のときと　n = m のとき．どちらも組み合わせの個数は1

```
fun  comb (_, 0) = 1
| comb (n, m) =
  if n = m then 1
  else
```

続きを考えてみてください（ミニレポート）

問題 2-2-3) 組み合わせの数を返す関数comb

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

基底は　m = 0 のときと　n = m のとき．どちらも組み合わせの個数は1

```
fun  comb (_, 0) = 1
| comb (n, m) =
  if n = m then 1
  else
```

_ はワイルドカード．つまり，どんな値ともマッチする．

n = m という条件はパターンでは書けないのでif文を使う．

続きを考えてみてください（ミニレポート）

問題 2-2-4) 2つの自然数 a, b の最大公約数を求める関数mygcd

(ユークリッドの互除法）

1. 基底
   b = 0 ならば a .
2. 帰納
   a を b で割った余りを b とし，元のbをa として再帰する.

MLの剰余演算子はMOD. 中置記法で定義されている.

関数mygcdをMLで定義せよ（ミニレポート）

問題 2-2-5)

リストの奇数番目と偶数番目の要素を入れ換える関数swapodevを書け.

すなわち，リスト[a1, a2, ... , an]が与えられると[a2, a1, a4, a3, a6, a5, ...] を生じる.

もしn が奇数ならばan がそのまま最後の要素となる.

関数 swapodev をMLで定義せよ（ミニレポート）

問題 2-2-6)

リストで集合を表す．集合の要素はどんな順番でリスト中に表れてもよいが，同じ要素は高々一度しかリストには表れないとする．集合における以下の操作を実現する関数を書け．

- a) x がS に含まれるときtrue, 含まれないときfalse を返す関数 member (x, S). x はS のどこに現れてもよい．
- b) x をS から削除する関数delete (x,S). x はリストS 中にたかだか一度しか現れないと仮定してよい．
- c) x をS に追加する関数insert (x, S). x はS 中にたかだか一度しか現れないという条件を満たすために，x がすでにS 中に存在するかどうかを確認しなければならない．ただ単純にSの頭部(head) にx を追加するだけでは正しくない．

各関数をMLで定義せよ（ミニレポート）

本日の演習およびミニレポート

1. 問題 2-2-3) 関数combの続きを完成させよ
2. 問題 2-2-4) 関数mygcdを定義せよ
3. 問題 2-2-5) 関数swapodevを定義せよ
4. 問題 2-2-6) 各関数を定義せよ

- じっくり考えても分からない場合は，正しく動かなくても良いので，考えたプログラムとなぜ上手く動かないのか，どうすればうまく動きそうかを考察して提出すること
- 締切：次回金曜講義前日の23:59