プログラム理論と言語　第6回
―帰納的定義とリスト―

有村博紀
吉岡真治

公開スライドPDF（情報知識ネットワーク研HP/授業）
http://www-ikn.ist.hokudai.ac.jp/~arim/pub/proriron/

前回の解答

- ２数の最大公約数を求めるプログラム
  Gcd(64,28) の実行過程を値呼び、名前呼びの二つの簡約手法で示せ。

- プログラム

```
gcd (x,y)  =
    if x > y then gcd(y,x)
    else if x = 0 then y
    else gcd(mod(y,x), x)
```

簡約の例（値呼び）

- gcd(64,28)
  = if 64 > 28 then gcd(28,64)
  else if 64 = 0 then 28
  else gcd(mod(28, 64), 64)
  = gcd(28,64)
  = if 28 > 64 then gcd(64,28)
  else if 28 = 0 then 64
  else gcd(mod(64, 28), 28)
  = if 28 = 0 then 64
  else gcd(mod(64, 28), 28)
  = gcd(mod(64, 28), 28)
  = gcd(8, 28)
  = if 8 > 28 then gcd(28,8)
  else if 8 = 0 then 28
  else gcd(mod(28, 8), 8)

簡約の例（値呼び）:続き

- = if 8 = 0 then 28
  else gcd(mod(28, 8), 8)
  = gcd(mod(28, 8), 8)
  = gcd(4, 8)
  = if 4 > 8 then gcd(4,8)
  else if 4 = 0 then 8
  else gcd(mod(8, 4), 4)
  = if 4 = 0 then 8
  else gcd(mod(8, 4), 4)
  = gcd(mod(8, 4), 4)
  = gcd(0, 4)
  = if 0 > 4 then gcd(4,0)
  else if 0 = 0 then 4
  else gcd(mod(4, 0), 0)
  = if 0 = 0 then 4
  else gcd(mod(4, 0), 0)
  =4

簡約の例（名前呼び）

- gcd(64,28)
  = if 64 > 28 then gcd(28,64)
  else if 64 = 0 then 28
  else gcd(mod(28, 64), 64)
  = gcd(28,64)
  = if 28 > 64 then gcd(64,28)
  else if 28 = 0 then 64
  else gcd(mod(64, 28), 28)
  = gcd(mod(64, 28), 28)
  = if mod(64, 28) > 28 then gcd(28, mod(64, 28))
  else if mod(64, 28) = 0 then 28
  else gcd(mod(28, mod(64, 28)), mod(64, 28))
  = if 8 > 28 then gcd(28, mod(64, 28))
  else if mod(64, 28) = 0 then 28
  else gcd(mod(28, mod(64, 28)), mod(64, 28))
  = if mod(64, 28) = 0 then 28
  else gcd(mod(28, mod(64, 28)), mod(64, 28))
  = if 8= 0 then 28
  else gcd(mod(28, mod(64, 28)), mod(64, 28))
  = gcd(mod(28, mod(64, 28)), mod(64, 28))

簡約の例（名前呼び）:続き

- = if mod(28, mod(64, 28)) > mod(64, 28) then gcd(mod(64, 28), mod(28, mod(64, 28)))
  else if mod(28, mod(64, 28))= 0 then mod(64, 28)
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(28, 8) > 8 then gcd(mod(64, 28), mod(28, mod(64, 28)))
  else if mod(28, mod(64, 28))= 0 then mod(64, 28)
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if 4 > 8 then gcd(mod(64, 28), mod(28, mod(64, 28)))
  else if mod(28, mod(64, 28))= 0 then mod(64, 28)
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(28, mod(64, 28))= 0 then mod(64, 28)
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(28, 8)= 0 then mod(64, 28)
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))

簡約の例(名前呼び):続き

- = if 4 = 0 then mod(64, 28)
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = gcd(mod(mod(64, 28), mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(mod(64, 28), mod(28, mod(64, 28))) > mod(28, mod(64, 28)) then gcd(mod(28, mod(64, 28)), mod(mod(64, 28), mod(28, mod(64, 28))))
  else if mod(mod(64, 28), mod(28, mod(64, 28))) = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(8, mod(28, 8)) > mod(28, 8) then gcd(mod(28, mod(64, 28)), mod(mod(64, 28), mod(28, mod(64, 28))))
  else if mod(mod(64, 28), mod(28, mod(64, 28))) = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(8, 4) > 4 then gcd(mod(28, mod(64, 28)), mod(mod(64, 28), mod(28, mod(64, 28))))
  else if mod(mod(64, 28), mod(28, mod(64, 28))) = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))

簡約の例(名前呼び):続き

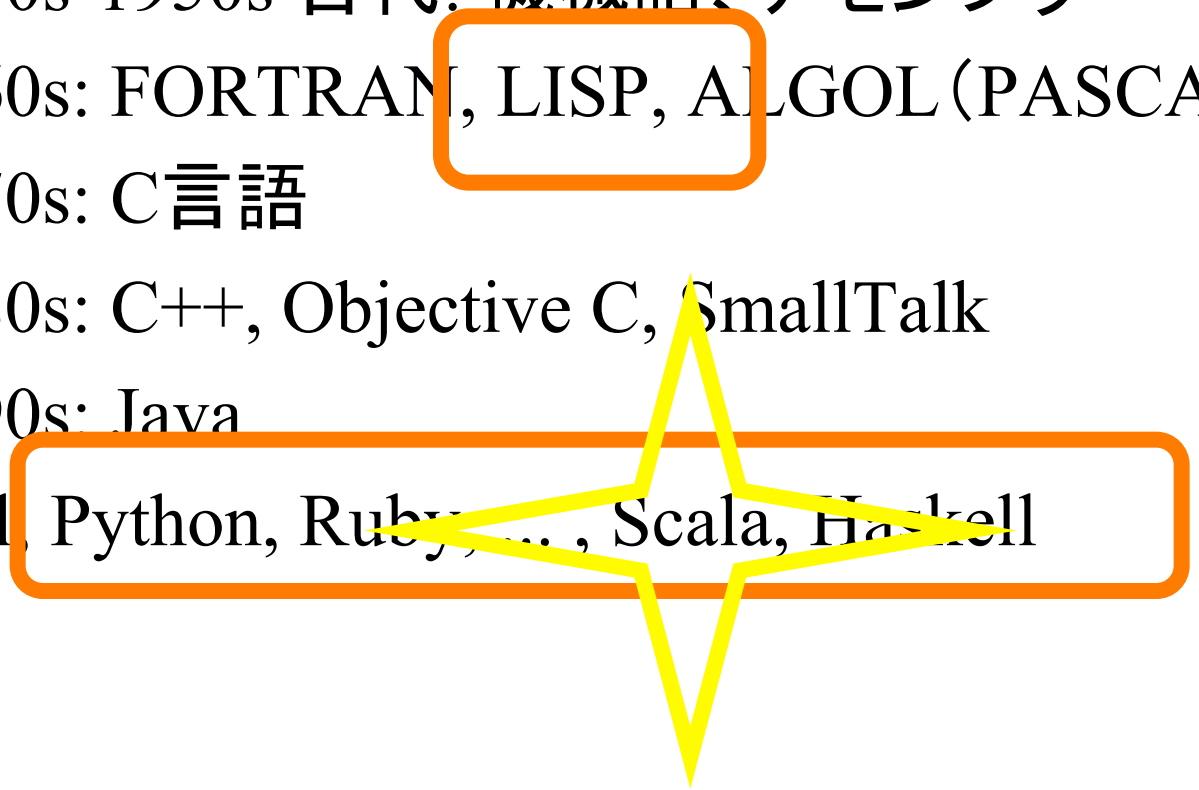
- = if 0 > 4 then gcd(mod(28, mod(64, 28)), mod(mod(64, 28), mod(28, mod(64, 28))))
  else if mod(mod(64, 28), mod(28, mod(64, 28))) = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(mod(64, 28), mod(28, mod(64, 28))) = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(8, mod(28, 8)) = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if mod(8, 4) = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = if 0 = 0 then mod(28, mod(64, 28))
  else gcd(mod(mod(64, 28) ,mod(28, mod(64, 28))), mod(28, mod(64, 28)))
  = mod(28, mod(64, 28))
  = mod(28,8)
  =4

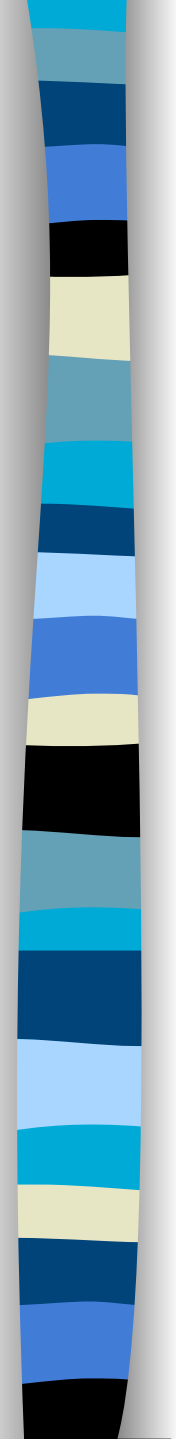
前回の訂正：簡約の例（名前呼び）

- 定義：fact(n) = if n =0 then 1 else n * fact(n-1)
- fact(3) = if 3 = 0 then 1 else 3 * fact(3-1)
  = 3 * fact(3-1)
  = 3 * (if 3-1 = 0 then 1 else (3-1) * fact((3-1)-1))
  = 3 * (if 2 = 0 then 1 else (3-1) * fact((3-1)-1))
  = 3 * ((3-1) * fact((3-1)-1))
  = 3 * ((3-1) * (if ((3-1)-1) =0 then 1 else ((3-1)-1) * fact (((3-1)-1)-1)))
  = 3 * ((3-1) * (if (2-1) =0 then 1 else ((3-1)-1) * fact (((3-1)-1)-1)))
  = 3 * ((3-1) * (if 1 =0 then 1 else ((3-1)-1) * fact (((3-1)-1)-1)))
  = 3 * ((3-1) * ((3-1)-1) * fact (((3-1)-1)-1)))
  = 3 * ((3-1) * ((3-1)-1) * (if (((3-1)-1) -1) =0
  then 1 else (((3-1)-1)-1) * fact (((3-1)-1)-1))
  = 3 * ((3-1) * ((3-1)-1) * (if ((2-1) -1) =0
  then 1 else (((3-1)-1)-1) * fact (((3-1)-1)-1))
  = 3 * ((3-1) * ((3-1)-1) * (if (1 -1) =0
  then 1 else (((3-1)-1)-1) * fact (((3-1)-1)-1))
  = 3 * ((3-1) * ((3-1)-1) * (if 0 =0
  then 1 else (((3-1)-1)-1) * fact (((3-1)-1)-1))
  = 3 * ((3-1) * ((3-1)-1) * 1)
  = 3 * ((3-1) * ((2-1) * 1)
  = 3 * ((3-1) * (1 * 1)
  = 3 * (2 * (1* 1))
  = 3 * (2 * 1)
  = 3 * 2
  = 6

プログラム言語の歴史

- 1940s-1950s 古代: 機械語、アセンブラ
- 1960s: FORTRAN, LISP, ALGOL（PASCAL）
- 1970s: C言語
- 1980s: C++, Objective C, SmallTalk
- 1990s: Java
- Perl, Python, Ruby, ... , Scala, Haskell





関数型プログラミング

リスト処理

集合の帰納的定義

おまけ

- 基底(basis)となる要素に関する定義と基底に対する操作(構成子:Constructor)により得られる要素により集合を定義する
  - 例:自然数
    - 0は自然数である。
    - xが自然数であれば、xに1を加えた物も自然数である。
    - これらの2つの操作によって自然数とわかる物のみが自然数である。

帰納的定義の表現

おまけ

- 帰納的定義による自然集の集合N
  - Nは自然数全体の集合で以下で定義される
    - 0は自然数である：$0 \in N$
    - xが自然数であれば、xに1を加えた物も自然数である：1を加える(s)：$N \rightarrow N$ （1を加える操作は、自然数を自然数に写像する）
- 表記
  - Inductive set N with
    0:N
    s：$N \rightarrow N$
- 自然数の集合
  - 0,s(0),s(s(0)),s(s(s(0))),....
  - sの数を数え、数字に置き換えると、我々が良く知っている自然数の表現と等しくなる

自然数上の関数

おまけ

- 帰納的定義で得られた集合上での関数
  - is_0 : 0(基底)であったかどうかを調べる関数
  - y≠0の自然数(y∈N) には、構成子を適用する前の自然数が存在するはずである : s(x)=y
  - ここで、構成子を適用する前の自然数を求める操作を $s^{-1}(y)=x$と表記する。
    - $s^{-1}(0)$は定義されない。
    - $s^{-1}(s(x))=x$, $s\,(s^{-1}(x))=x$

基本関数・基本述語による関数定義

おまけ

- 基本関数・基本述語をリデックスと考え、簡約を行うことにより、計算を行う
- 原始帰納的関数で表現される
  - 例：足し算

    plus(x,y) =

    if is_0(x) then y

    else s(plus(s^{-1} (x),y))
  - 実行例：
    - plus(s(s(0)), s(s(s(0))))

      s(plus(s^{-1} (s(s(0))), s(s(s(0)))))

      s(plus(s(0), s(s(s(0)))))

      s(s(plus(s^{-1}(s(0)), s(s(s(0))))))

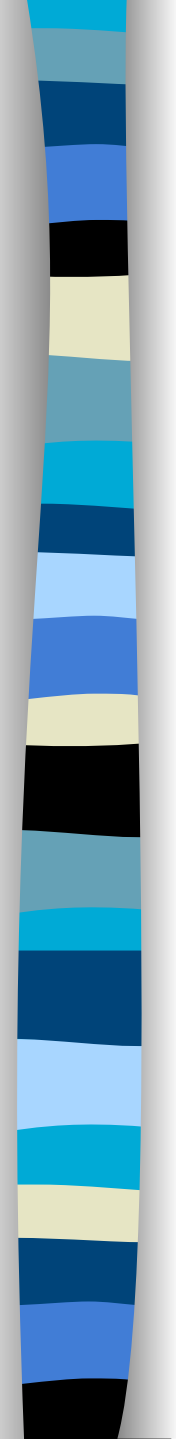
      s(s(plus(0, s(s(s(0))))))

      s(s(s(s(s(0)))))

練習問題

おまけ

- 自然数の積を表す関数times(x,y)を定義し、times(s(s(0),s(s(0)))を計算せよ。
- 必要であれば、先に定義したplus(x,y)を利用しても良い。



関数型プログラミング

リスト処理

関数型言語によるリスト処理

ここは大事！

- リストとは
  - 有限個のデータ(要素)の並び
  - 例) L = (a, b, c, d) は，要素a, b, c, d が順に並んだリスト.
  - 言語によっては，コンマを省略して (a b c d)とも書いたり(LISP言語など)，四角カッコをつかって [a, b, c, d] と書くこともある(Python).
- 空リスト：要素数0の特別なリスト
  - ()やNilで表す(またはともかく．要素なしで四角かっこ[と]を書く)
- リストの長さ： その要素の数
  - 例) 上のリストLの長さは4

リストに対する基本操作

説明だけなので簡単に....

- **first(L)** :リストLの最初の要素
  - 例）first(0 1 2 3) = 0
- **rest(L)** :リストLの残りの要素
  - 例）rest(0 1 2 3) = (1 2 3)
  - 大事な性質：firstとrestを合わせると元に戻る！
- **is_cons(L) :** リストLが，空リスト(nil)かどうか？
  - 例）is_cons(nil) = false, is_cons((1 2))= true, is_cons(1) = false
- **is_nil(L) :** リストLが，空リスト(nil)か？
  - 例）is_nil(nil) = true, is_nil((1 2))= false, is_nil(1)= false

あとでScala言語で演習をします

リストに対する関数の定義

- 多くのプログラミング言語*は，組み込み関数としてリストの処理関数をもっている
  - length(L) = リストLの長さ（要素数）
  - sum(L) = 数のリストLの要素の総和
  - append(X, Y) = リストXとリストYの連結
  - reverse(L) = リストLの反転
- これらを自分で書いてみることは，関数型言語の良い練習問題
  - 副作用は使わないこと.
  - 配列アクセスは使わないこと

注*）このスライドの演習ができるプログラミング言語:
◎scala, ○LISP, scheme, Haskell. △Python, Ruby

例:リスト要素の足し算

説明だけなので簡単に....

- リスト要素に含まれる全ての数の総和を計算

```
sum(x) =
    if is_nil(x) then 0
    else first(x)+ sum(rest(x))
```

- 実行過程
  - sum((1 2 3)) = 1 + sum((2 3))
    = 1 + 2 + sum((3))
    = 1 + 2 + 3 + sum(nil)
    = 1 + 2 + 3 + 0
    = 6

あとでScala言語で演習をします

例:リストの長さ

説明だけなので簡単に....

- リスト要素に含まれる要素の数を計算

```
length(x) =
    if is_nil(x) then 0
    else 1+ length(rest(x))
```

- 実行過程
  - length((1 2 3)) = 1 + length ((2 3))
    = 1 + 1 + length ((3))
    = 1 + 1 + 1 + length (nil)
    = 1 + 1 + 1 + 0
    = 3

あとでScala言語で演習をします

例:リストの連結

演習問題

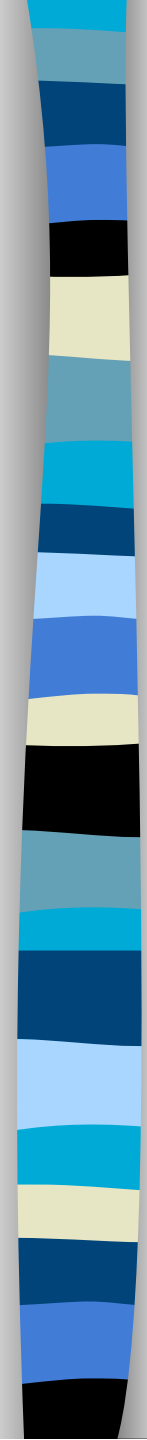
説明だけなので簡単に....

- 2つの異なるリストを連結

```
append(x, y) =
    if is_nil(x) then y
    else cons((first(x), append(rest(x), y))
```

- 実行過程
  - append((1 2 3), (4 5))
    = cons(1, append((2 3), (4 5)))
    = cons (1, cons(2, append((3), (4 5))))
    = cons (1, cons(2, cons(3, append(nil, (4 5)))))
    = cons (1, cons(2, cons(3, (4 5))))
    = cons (1, cons(2, (3 4 5)))
    = cons (1, (2 3 4 5))
    = (1 2 3 4 5)

あとでScala言語で演習をします



関数型源と

SCALA言語でリスト処理プログラムを書こう

Scala言語早わかり(1)

- Scala言語
  - 現在，広く使われている関数型言語の一つ
  - 手続き型やオブジェクト型の命令も使える
  - インタプリタ（REPLループ）と，Java仮想機械（JVM）へのコンパイラの両方が使える

- 変数の型宣言は "変数：型"のように書く.
  - 例）x : Int, y : String
- いろいろな型がある
  - Int　　整数型
  - String　　文字列型
  - List[Int]　　整数のリスト型（後述）

Scala言語早わかり (2)

- 変数は使う前に変数宣言する
  - **val** x　　変更できない変数の宣言（ふつう）
  - **var** x　　変更可能な変数の宣言（使わない）
- 関数宣言（手続き宣言）は "def"で定義する.
  - 前から順に，関数名，引数と型の並び，返り値の型の順に書き，"="のあとに関数の本体を書く.
  - 例）def add(x : Int, y : Int) : Int = { x + y }
- 構造化プログラミング（手続き的言語）のほとんどの制御構文をもつ
  - 例）if-then-else文, while文
- パターンマッチ文（match-case文）で，if文の代わりに場合分けができる.（リスト処理に強力）

2016.12.22: 関数宣言の例の誤りを訂正しました.

Scalaプログラムを書こう（scala言語）

```
arim@lecture$ scala
Welcome to Scala 2.11.8
 (Java HotSpot(TM) 64-Bit ....

scala> val msg = "hello world!"
msg: String = hello world!

scala> val x = 2 + 3
x: Int = 5

scala> :quit
arim@material$
```

unix/linuxのコマンドラインからscalaコマンドを実行

valは書き換えられない変数の宣言

変数msgはString型（文字列型）

変数xはInt型（整数型）

scalaのセッションを終了

Scalaプログラムを書こう（scala言語）

```
scala> def add(x : Int, y : Int) : Int = x + y
add: (x: Int, y: Int)Int

scala> val z = add(2, 3)
z: Int = 5

scala> val z = add(2, add(3, 4))
z: Int = 9

scala>
```

def で関数宣言

Scalaプログラムを書こう（scala言語）

```
scala> def max(x: Int, y: Int) : Int = {
     | if (x > y) x
     | else y
     | }
max: (x: Int, y: Int)Int
scala> val z = max(2, 3)
z: Int = 3
scala> def max(x: Int, y: Int) =
     | if (x > y) x else y
max: (x: Int, y: Int)Int
scala> val z = max(1, max(2, 3))
z: Int = 3
```

関数宣言

IF-ELSE文．関数型なのでRETURNはいらない

関数定義のカッコは省略できる

関数は何回でも組み合わせて使える

リスト処理プログラムを書こう(scala言語)

- Scala言語によるリスト処理
  - リスト構成子::とパターンマッチで，リストの処理ができる.

//scala

- 空リスト: List() または Nil
- 要素がa, b, cのリスト:
  - List(a, b, c)　と書く
  - a :: b :: c :: Nil ともかける
- 要素aをリストMの先頭につけたリストL
  - L = a :: M

Scala言語によるリスト処理

- リスト構成子::とパターンマッチで，リストの処理ができる．

- 定義（リスト構成子） L = x**::xs**で，先頭が要素xで，残りの部分がリストxsであるようなリストを表す．
  - xをLの先頭要素（=first(L)）という
  - xsをLの後続要素（=rest(L)）という

- 例）x = a, xs = [b, c, d]のとき，

  x::xs = a::[b,c,d] = [a,b,c,d]

scala言語

- 定義（パターンマッチ）
- Nil は空リストにマッチする
- X::Lは，空でないリスト（つまり長さ1以上のリスト）にマッチする．さらに，先頭要素X(つまりfirst(L))と後続リストM(つまりrest(L))を取りだせる

```
//scala リストのパターンマッチ文
   L match {
      case Nil => 何か処理をする
      case x :: xs =>
        先頭要素xと後続リストxsで
        何か処理をする
   }
```

リスト処理プログラムを書こう(scala言語)

演習問題

- 数のリストLの要素の足し算 sum(L)
- 例）L = List(1, 2, 3)のとき，sum(L) = 1 + 2 + 3 = 6

```
//scala リスト要素の足し算
def sum(L : List[Int]) : Int =
    L match {
        case Nil => 0
        case x :: xs => x + sum(xs)
    }
```

```
//本スライドの再帰的な定義
sum(x) =
        if is_nil(x) then 0
        else first(x)+ sum(rest(x))
```

実行例: リストの要素の総和

```
scala> def sum(L : List[Int]) : Int =
     | L match {
     | case List() => 0
     | case x :: xs => x + sum(xs)
     | }
sum: (L: List[Int])Int

scala> val L = List(1, 2, 3)
L: List[Int] = List(1, 2, 3)

scala> sum(L)
res8: Int = 6
```

リスト処理プログラムを書こう(scala言語)

演習問題

- 2つのリストXとYの連結 append(X, Y)
- 例）X = List(1, 2), Y = List(3, 4, 5)のとき，
  append(X, Y) = List(1, 2, 3, 4, 5)

```
//scala 2つのリストの連結
def append(x : List[Int], y : List[Int]) : List[Int] =
    x match {
        case Nil => y
        case z :: zs => z :: append(zs, y)
    }
```

```
append(x, y) =     //本スライドの再帰的定義
    if is_nil(x) then y
    else cons((first(x), append(rest(x), y))
```

リスト処理プログラムを書こう(scala言語)

演習問題

- 2つのリストXとYの連結 append(X, Y)

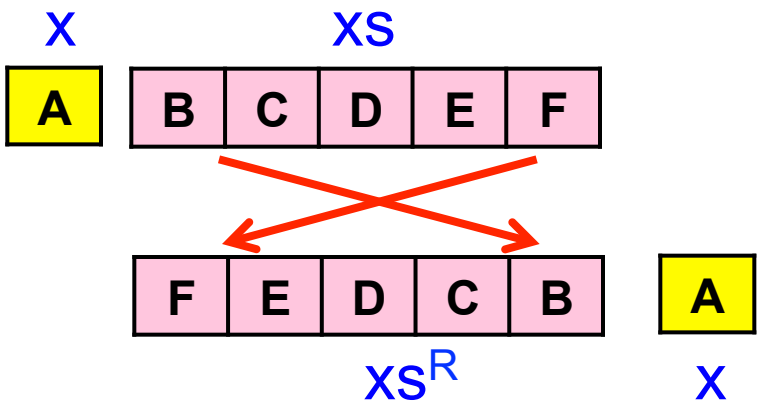
```
scala> def append(x : List[Int], y : List[Int]) : List[Int] =
     | x match {
     | case Nil => y
     | case z :: zs => z :: append(zs, y)
     | }
append: (x: List[Int], y: List[Int])List[Int]

scala> val z = append(List(1,2), List(3,4,5))
z: List[Int] = List(1, 2, 3, 4, 5)
scala>
```

リスト処理プログラムを書こう(scala言語)

演習問題

- 演習問題：リストXの反転 reverser(X)
  - 例）reverse(List()) = List()
  - 例）reverse(List(a, b, c, d)) = List(d, c, b, a)
- プログラムの考え方
  - 場合分けして，帰納法で考える
  - (i) 空リストの反転は，空リスト．(ii) 先頭要素が x で，後続リストがxsとなるリストの反転は，xsを反転しておいて，末尾にxを連結すれば良い．
  - リストの連結には，前につくったappendを使う！



リスト処理プログラムを書こう(scala言語)

演習問題

```
//scalaリストXの反転のプログラム
//* リストXをもらって，その反転Xを返す
//* 連結プログラムappendを使う版

def reverse(xs: List[Int]) : List[Int] = {
  xs match {
    case Nil => Nil
    case y :: ys => append(reverse(ys), y::Nil)
  }
}
```

(i)　空リストの反転は，空リスト.
(ii) 先頭要素が x で，後続リストがxsとなるリストの反転は，xsを反転しておいて，末尾にxを連結する

実行例

```
arim@scala$ scala
scala> :load append.scala
Loading append.scala...
append: (xs: List[Int], y: List[Int])List[Int]
reverse: (xs: List[Int])List[Int]

scala> reverse(List())
res1: List[Int] = List()

scala> reverse(List(1,2,3))
res2: List[Int] = List(3, 2, 1)

scala> reverse(List(1,2,3,4,5,6,7,8,9))
res3: List[Int] = List(9, 8, 7, 6, 5, 4, 3, 2, 1)
```

注意:リストの反転

発展的内容

- このプログラムreverserは，要素数nのリストを反転するのに，$O(n^2)$時間を要する.
- これはなぜ？

  - 理由:
    - O(n)個の先頭要素xを一つ取る.
    - 先頭要素xごとに，appendで長さO(n)の後続リストxsの末尾にxを（正確には長さ1のリストx::NIlを）連結する．これはO(N)時間を要する.
    - よって，全てを反転するのに，$O(n)\times O(n) = O(n^2)$時間かかる
  - 挑戦！ O(n)時間で反転できないか？

解決編：高速リスト反転プログラム

発展的内容

```
//reversefast.scala
//2つのリストをO(n)時間で連結する

def reverse(xs: List[Int]) : List[Int] =
  reverse1(xs, Nil)

def reverse1(xs, ys) : List[Int] = {
  xs match {
    case Nil => ys
    case z :: zs => reverse1(zs, z :: ys)
  } //第二引数ysに反転リストを蓄積する
}
```

2016.12.22: reverse/1の定義の誤りを訂正しました.

実行例

発展的内容

```
arim@scala$ scala
scala> :load append.scala
Loading append.scala...
append: (xs: List[Int], y: List[Int])List[Int]
reverse: (xs: List[Int])List[Int]

scala> reverse(List())
res1: List[Int] = List()

scala> reverse(List(1,2,3))
res2: List[Int] = List(3, 2, 1)

scala> reverse(List(1,2,3,4,5,6,7,8,9))
res3: List[Int] = List(9, 8, 7, 6, 5, 4, 3, 2, 1)
```

リストに対する原始帰納的関数

発展的内容

- 自然数と同様にリストに対しても原始帰納法による計算の停止性に関する議論が可能
- 原始機能的関数
  - 関数f(x)の値がfirst(x), rest(x), f(rest(x))により既知の関数g(定義にfを含まない)で求めることができ、f(nil)の値cが決まっている関数
  - 再帰的関数定義の特殊形
  - 定義の例:

    f(x) = if is_nil(x) then c

    else g(first(x), rest(x), f(rest(x)))
  - 関数gの計算が全てのリストについて停止するなら、f(x)の計算は停止する

この授業の進め方(再)

- イントロダクション(有村・原口)
  - プログラミング言語の歴史
  - プログラミング言語と背景理論
- 手続き型と関数型 (有村)
- オブジェクト指向型と論理型(原口)

プログラム言語の歴史

手続き型

- 1940s-1950s 古代: 機械語、アセンブラ
- 1960s FORTRAN, LISP, ALGOL（PASCAL）
- 1970s C言語
- 1980s: C++, Objective C, SmallTalk
- 1990s: Java
- Perl, Python, Ruby, .. , Scala, Haskell

関数型

- 発展（主に関数型言語）
  - 高階関数（Scala, Haskel, javaのlambda）
  - 型理論（ScalaやSMLの型推論）
  - 大規模実行手法と専用言語（データ処理言語等）

今後の勉強のために

- 次は広く使われており，玄人と一般の両方に人気がある「関数型系」プログラミング言語
  - LISP　（エディタのemacsの設定言語）
  - Python　（「普通の書き方のLISP」？）
  - Ruby　（一部で圧倒的人気）
  - Haskell　（純粋な関数型言語ではこれ！）
- 次は
  - Java　（Java8から導入されたLambda)
  - scheme　（小さくてきれいなLISP）

まとめ

- 集合の帰納的定義
  - 基底と操作の組合せにより集合を操作的に定義
  - 帰納的定義で与えられた集合上の操作を表す関数が原始的帰納法で表現可能であれば、必ず計算が終了することが保障される。
- リスト
  - 帰納的定義により定義される
  - 関数型言語向きのデータ構造