再帰（１）

クラスライブラリ応用

斉藤（裕）

本日の内容

再帰とは
例題: 階乗計算
例題: 真に再帰的なアルゴリズム
再帰とリスト

本日の内容

再帰とは

例題: 階乗計算

例題: 真に再帰的なアルゴリズム

再帰とリスト

「再帰」とは

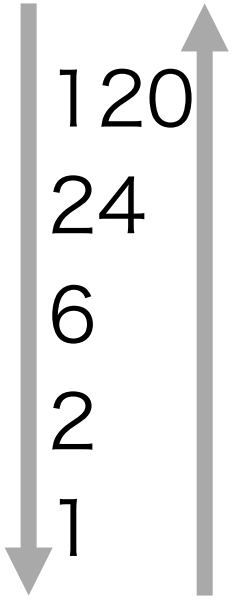
「ある処理」の方法を、「ある処理」自身を使って説明する方法

例

- 5の階乗は、4の階乗×5
- A→B→C→Dの経路を探すには、A→B→Cの経路を探し、CからDを見つければ良い

例題: 階乗計算 (pp.153-155)

- 階乗計算の例
  - 5! = 4! × 5
  - 4! = 3! × 4
  - 3! = 2! × 3
  - 2! = 1! × 2
  - 1! = 0! × 1
  - 0! = 1



(0だけは定義しておく)

再帰を理解するコツ

一つ前までは「できていることにする」

- 階乗計算の例
  - n! とは (n-1)! × n である

- プログラム的に書くと

```
階乗(n)の定義:
        階乗(n-1) * n
```

階乗を求めるメソッド

```
int fact(int n) {
    return  fact(n-1) * n;
}
```

考え方はほぼ合っているが、このままでは動かない。何を見落としているのだろうか？

階乗を求めるメソッド

```
int fact(int n) {
    if (n == 0)
        return 1;
    else
        return fact(n-1) * n;
}
```

n=1のときの計算

それ以外のときの計算

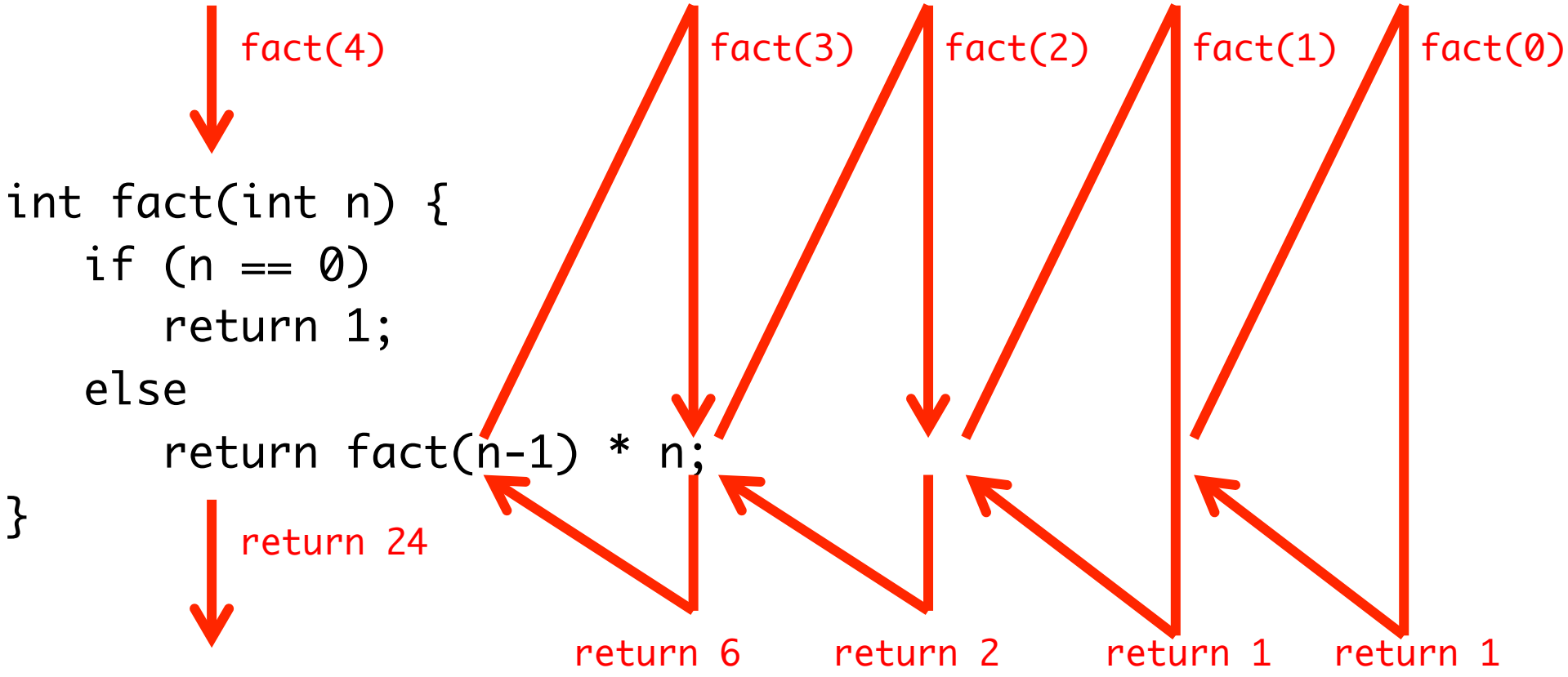
nが0のときはn!は1と定義する

実行の様子（１）

fact(5)=fact(4) x 5　そこで、fact(4)を求める
　fact(4)=fact(3) x 4　そこで、fact(3)を求める
　　fact(3)=fact(2) x 3　そこで、fact(2)を求める
　　　fact(2)=fact(1) x 5　そこで、fact(1)を求める
　　　　fact(1)=fact(0) x 1　そこで、fact(0)を求める
　　　　　fact(0)は定義から1
　　　　fact(1)=1×1=1
　　　fact(2)=1×2=2
　　fact(3)=2×3=6
　fact(4)=6×4=24
fact(5)=24×5=120

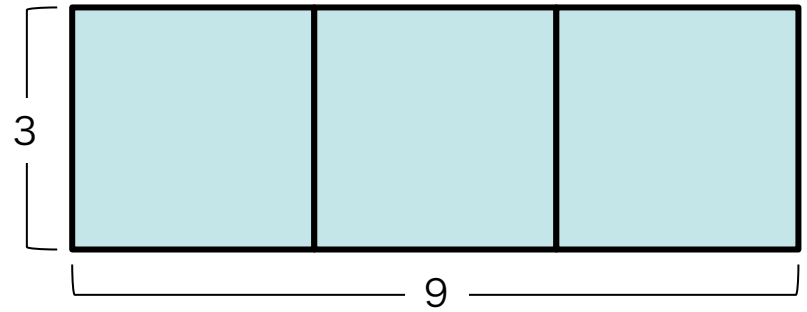
```
int fact(int n) {
    if (n == 0)
        return 1;
    else
        return fact(n-1) * n;
}
```

実行の様子（2）


fact(4)
fact(3)
fact(2)
fact(1)
fact(0)
int fact(int n) {
if (n == 0)
return 1;
else
return fact(n-1) * n;
}
return 24
return 6
return 2
return 1
return 1

演習1: 最大公約数

- 最大公約数：2つの数に存在する約数のうち最大のもの
- 【考え方】例：**9と3の最大公約数**



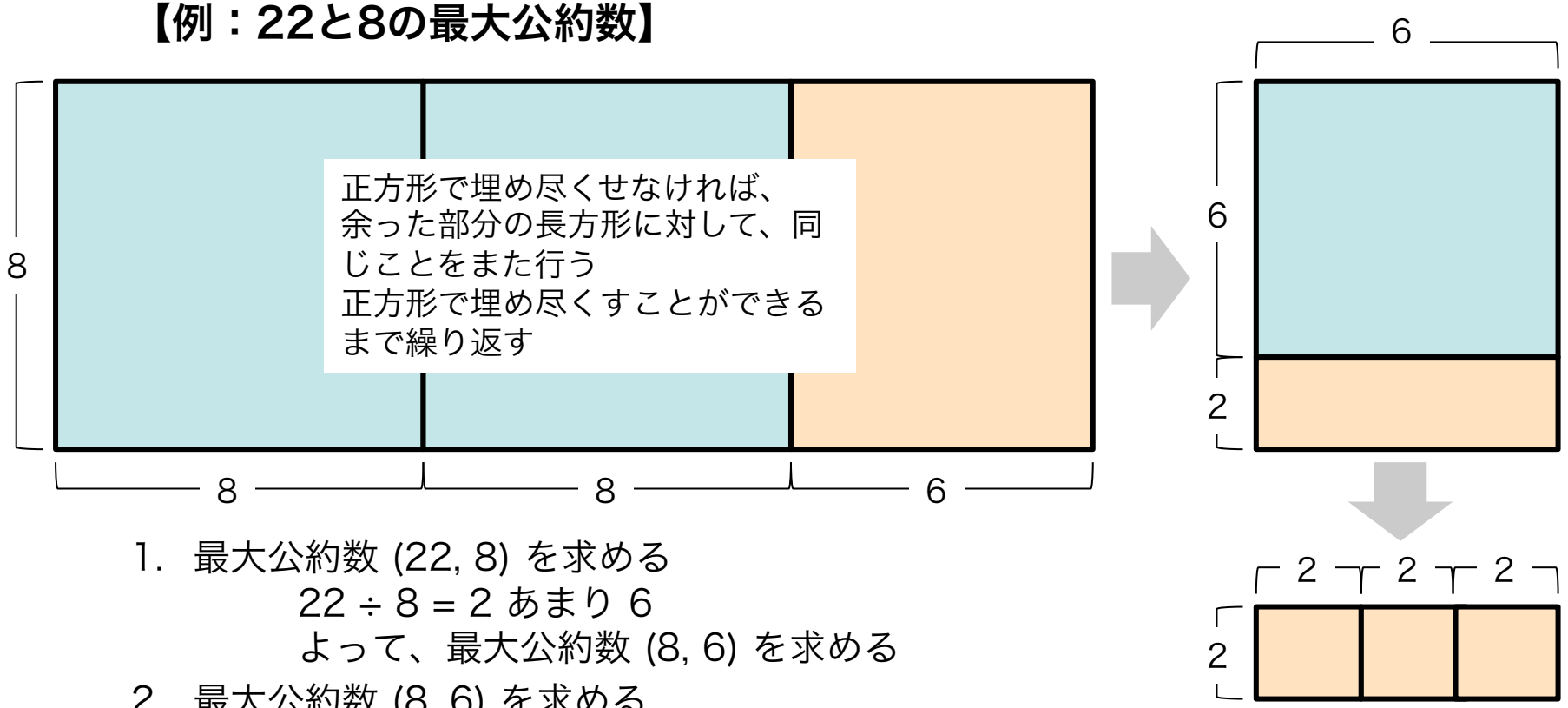
短い方の辺の長さを一辺とする正方形で埋め尽くす
ぴったり埋め尽くせれば、短い辺の長さが答え＝3

【最大公約数の定義（不完全版）】

最大公約数 (a, b)：
a ÷ b が割り切れたとき b が、最大公約数となる
では、割り切れなかったら？？？

演習1: 最大公約数

【例：22と8の最大公約数】



1. 最大公約数 (22, 8) を求める
   22 ÷ 8 = 2 あまり 6
   よって、最大公約数 (8, 6) を求める
2. 最大公約数 (8, 6) を求める
   8 ÷ 6 = 1 あまり 2
   よって、最大公約数 (6, 2) を求める
3. 最大公約数 (6, 2) を求める
   6 ÷ 2 = 3 (割り切れた)
   前のスライドより、答えは 2

演習1：最大公約数

以上の考え方に基づき、最大公約数の計算を再帰的に定義しなさい

【最大公約数の定義】

最大公約数（a, b）：
・［　？　］のときは、最大公約数（［?］, ［?］）を求める
・ただし［　？　］のときは、［?］を結果の値とする

（参考）このアルゴリズムを「ユークリッドの互除法」と呼ぶ

本日の内容

再帰とは

例題: 階乗計算

例題: 真に再帰的なアルゴリズム

再帰とリスト

例題: フィボナッチ数

fib(n) = fib(n-2) + fib(n-1)　　(n>2のとき)

fib(n) = 1　　(n=2またはn=1のとき)

fib(4)　= fib(2) + fib(3)
　= 1 + fib(1) + fib(2)
　= 1 + 1 + 1
　= 3

fib(5)　= fib(3) + fib(4)
　= fib(1) + fib(2) + fib(2) + fib(3)
　= 1 + 1 + 1+ fib(1) + fib(2)
　= 1 + 1 + 1 + 1 + 1
　= 5

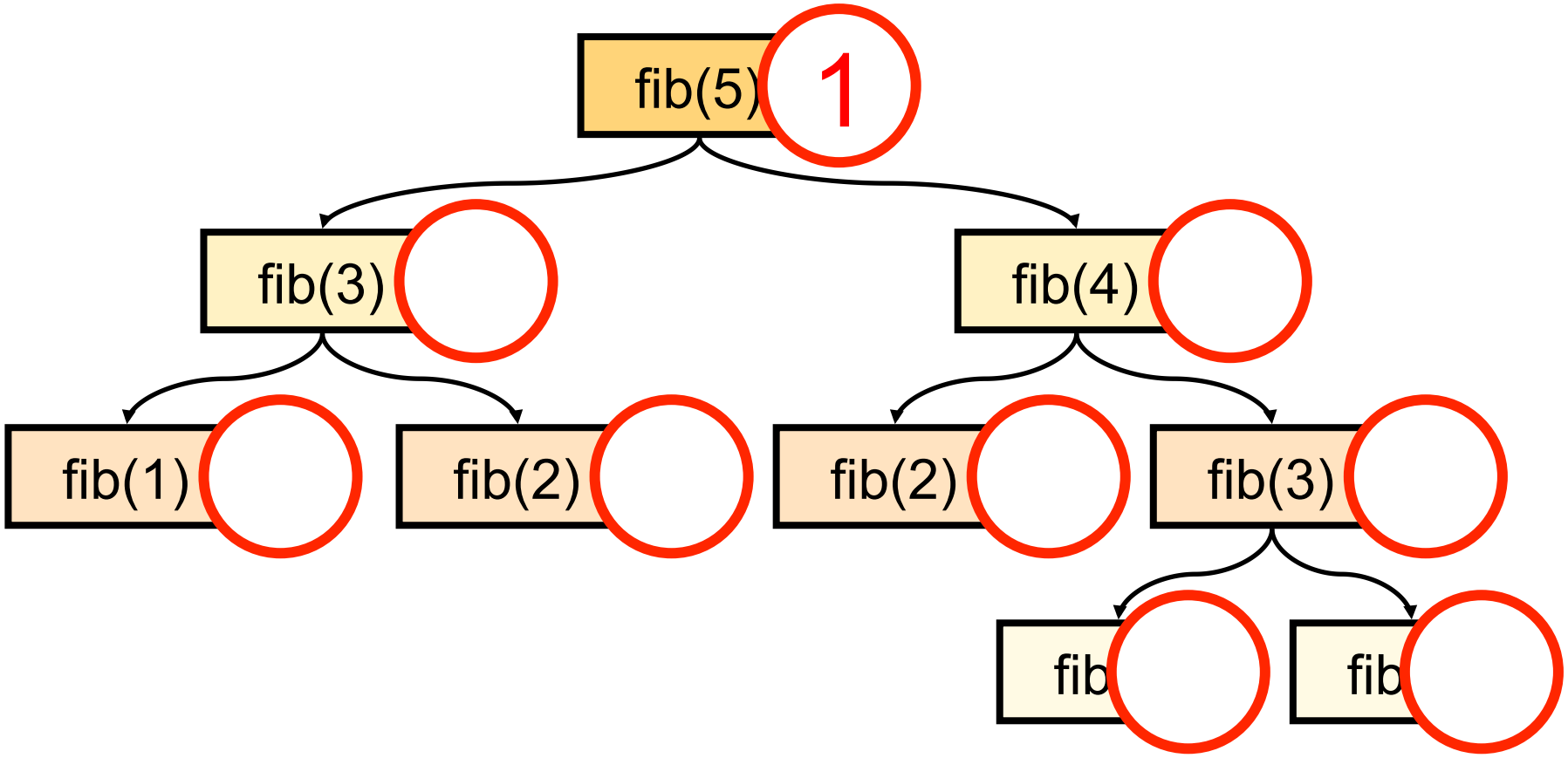
フィボナッチ数を求めるプログラム

fib(n) = fib(n-2) + fib(n-1)　　(n>2のとき)
fib(n) = 1　　(n=2またはn=1のとき)

```
int fib(int n) {
    if (n == 2 || n == 1)
        return 1;
    int ans = fib(n-2) + fib(n-1);
    return ans;
}
```

演習２：フィボナッチ数を求める再帰呼び出しの解析

fib(5) の計算において、メソッドが呼び出される順番を以下の○の中に書き込みなさい



本日の内容

再帰とは

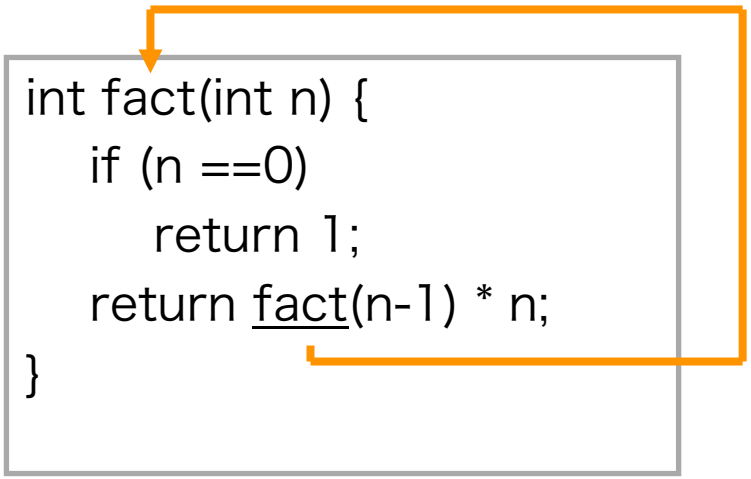
例題: 階乗計算

例題: 真に再帰的なアルゴリズム

再帰とリスト

再帰とリストの類似性

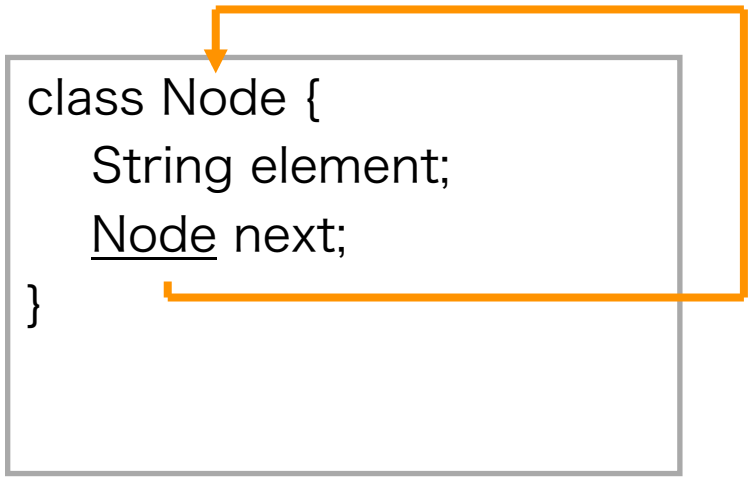
再帰



```
int fact(int n) {
    if (n ==0)
        return 1;
    return fact(n-1) * n;
}
```

自分から自分自身を呼び出す

リスト

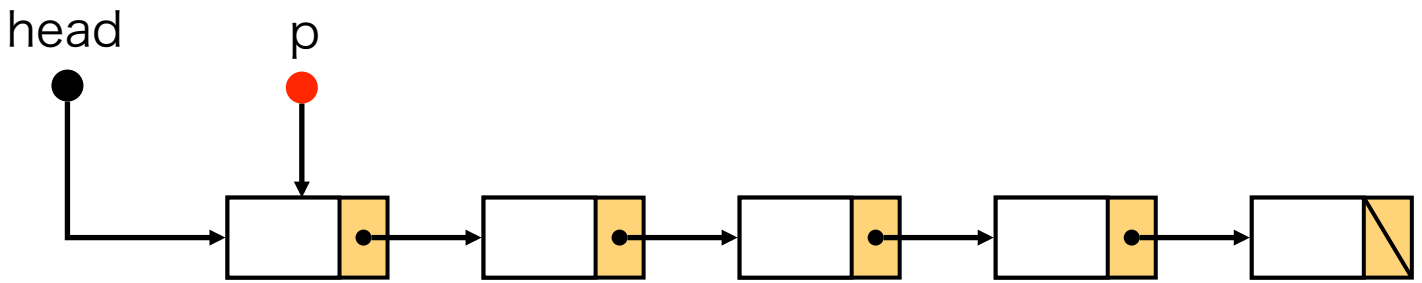


```
class Node {
    String element;
    Node next;
}
```

自分が自分自身を参照する

[復習] リスト上のノードを順にアクセス

```
void follow(Node p) {
    Node p = head;
    while (p != null) {
            pに関する処理;
            p = p.getNext();
        }
  }
```



演習3: 再帰を使い書き換える

これまでの書き方

```
void follow() {
    Node p = head;
    while (p != null) {
        pに関する処理;
        p = p.getNext();
    }
}
```

再帰を使った書き方

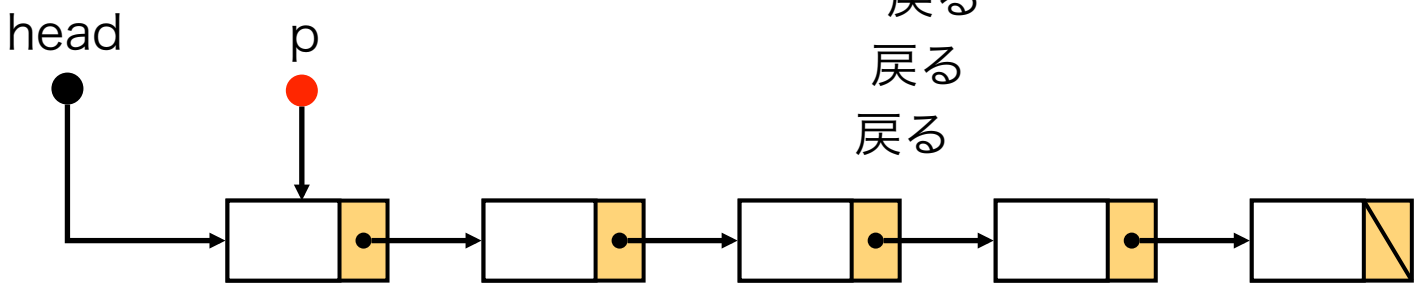
(最初に呼び出すときの
pの値はheadとする)

```
void follow_r(Node p) {
    ?
    pに関する処理;
    follow_r( ? );
}
```

ヒント：実行の様子

```
void follow_r(Node p) {
    if (     ?     )
        ?
    pに関する処理;
    follow_r(     ?     );
}
```

follow_r(head)
 follow_r(2番目のノード)
  follow_r(3番目のノード)
   follow_r(4番目のノード)
    follow_r(5番目のノード)
     follow_r(6番目のノード)
      ここでpがnullになるので戻る
    戻る
   戻る
  戻る
 戻る
戻る



プログラムの構造

- **書き換えが簡単**

```
メソッドA {
    処理A;
    メソッドA();
}
```

メソッドの最後に再帰呼び出しが行われる場合

- **（少し）難しい**

```
メソッドB {
    メソッドB();
    処理B;
}
```

再帰呼び出しのあとに他の処理が行われる場合

「末尾再帰」と呼ぶ

再帰呼び出しの除去

- 再帰呼び出しは、定義から分かりやすいプログラムを作成することができるが、実行効率がやや悪い
  - メソッドの実行にともなうオーバヘッド等

- そこで、再帰呼び出しを単純な繰り返しに書き換える方法を考えよう

一般の末尾再帰除去の形式

- **末尾再帰の形式**

```
F(x) {
    処理
    if (条件式)
        F(α)
    else
        return
}
```

- **繰り返しの形式**

```
F(x) {
    while (条件式) {
        処理
        x = α
    }
}
```