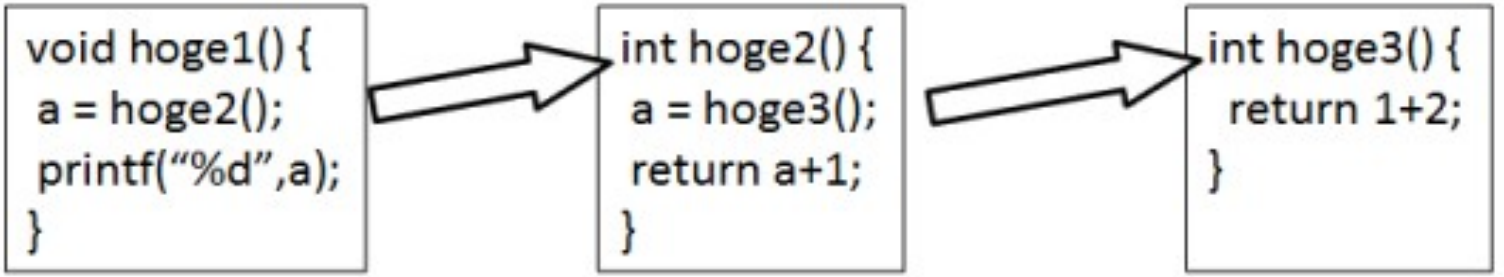
ひらメソッド初心者奮闘記

2007/4/28 LMS発表
2007/4/29 一部訂正
2007/4/30 ライセンス追記
2007/5/07 クレジット変更
野田

ひらメソッドってなに？

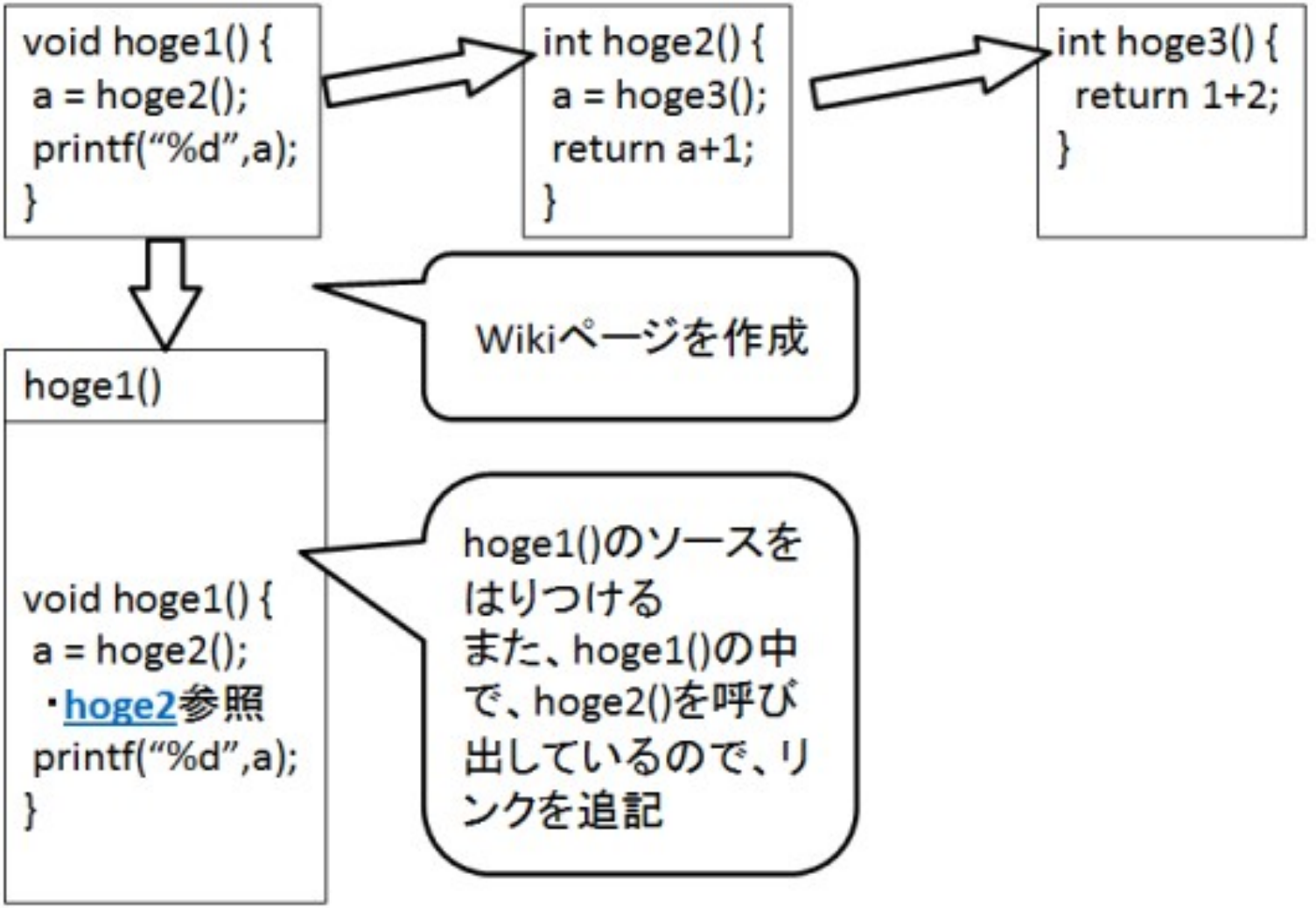
- ひらさんが考案したコードリーディング手法
- LKH-jp（旧読学のススメ）で公開

ページの作り方①

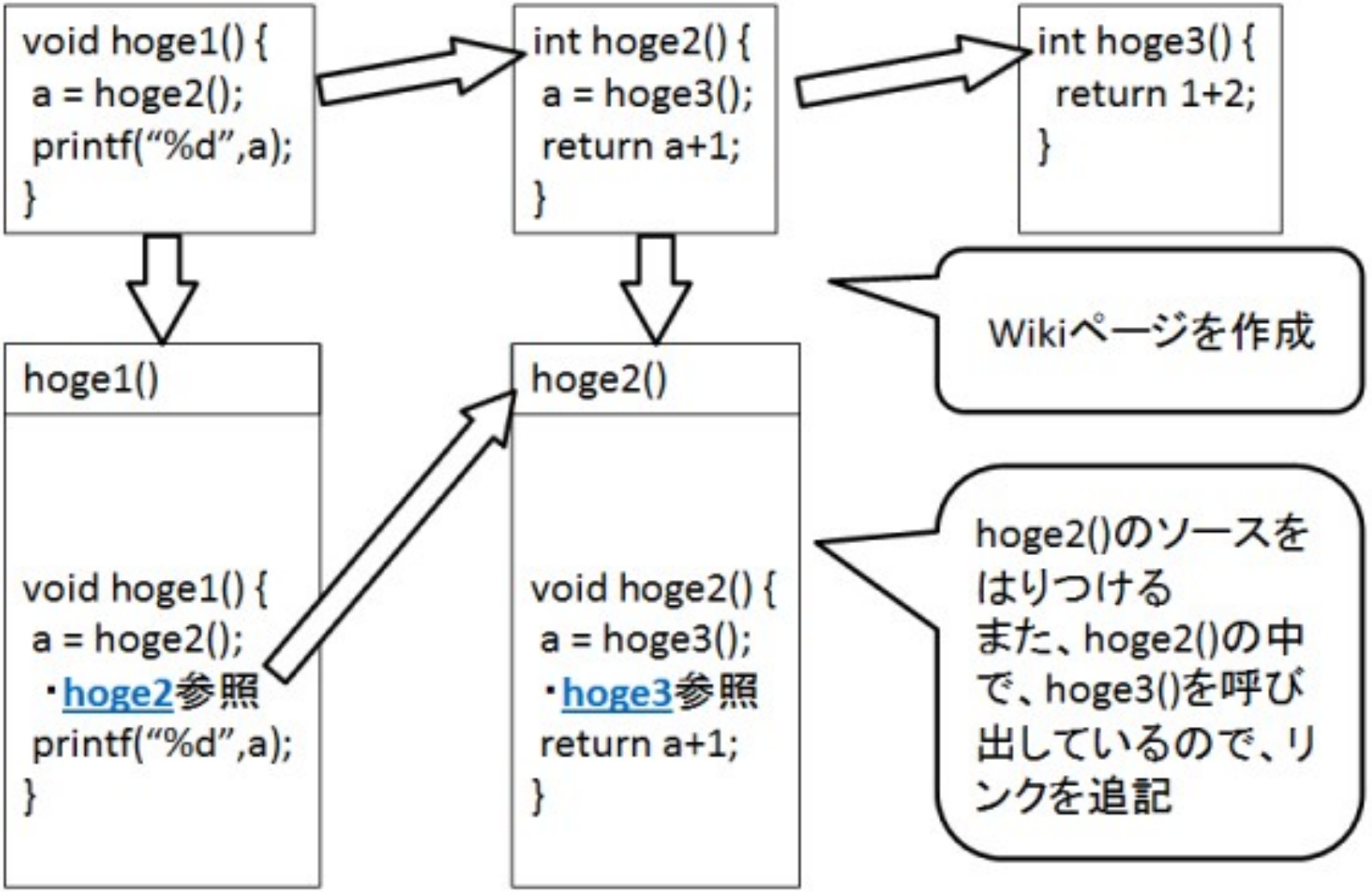


上のような構成のソースコードがあると仮定

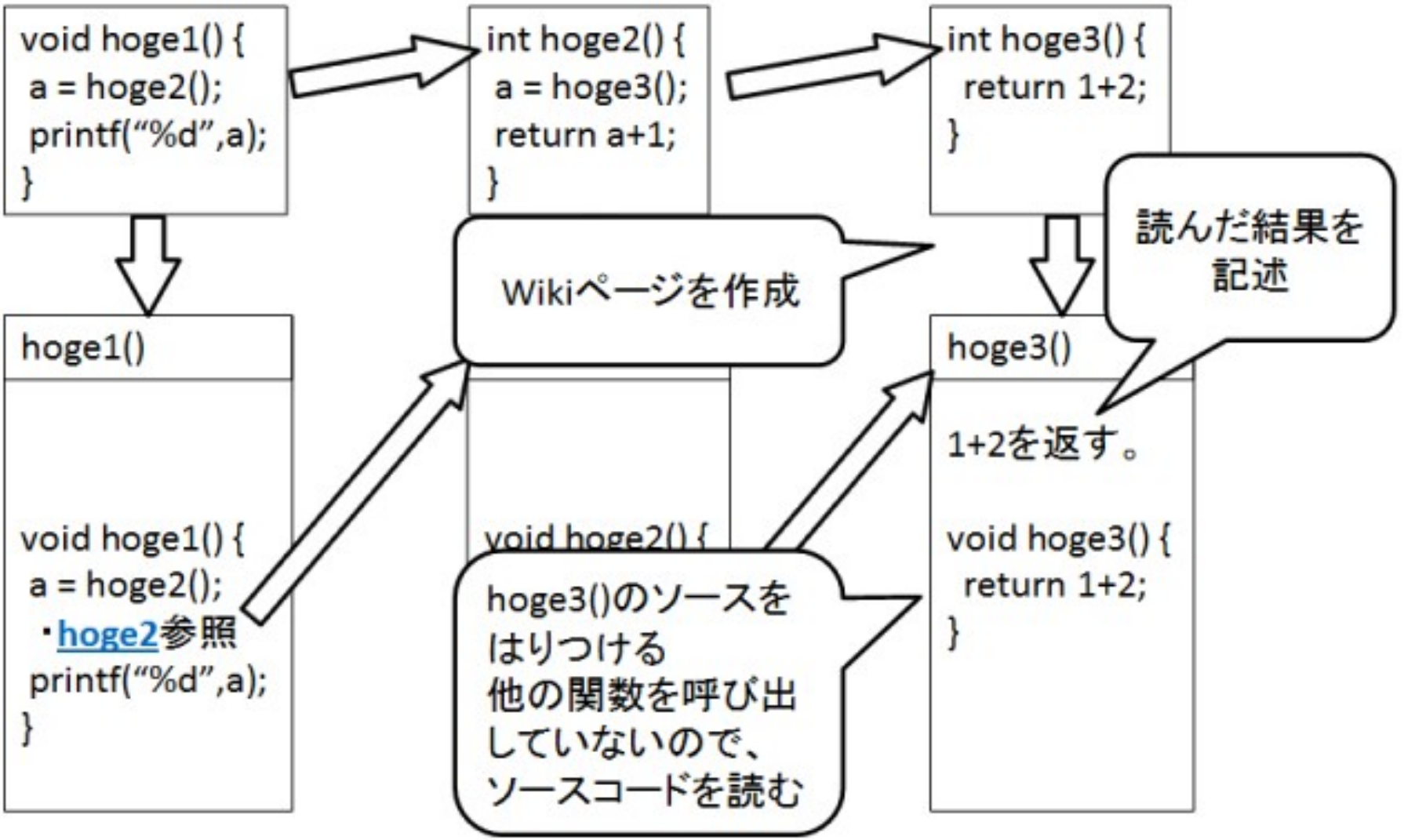
ページの作り方②


void hoge1() {
a = hoge2();
printf("%d",a);
}
int hoge2() {
a = hoge3();
return a+1;
}
int hoge3() {
return 1+2;
}
Wikiページを作成
hoge1()
void hoge1() {
a = hoge2();
・hoge2参照
printf("%d",a);
}
hoge1()のソースをはりつける
また、hoge1()の中で、hoge2()を呼び出しているので、リンクを追記

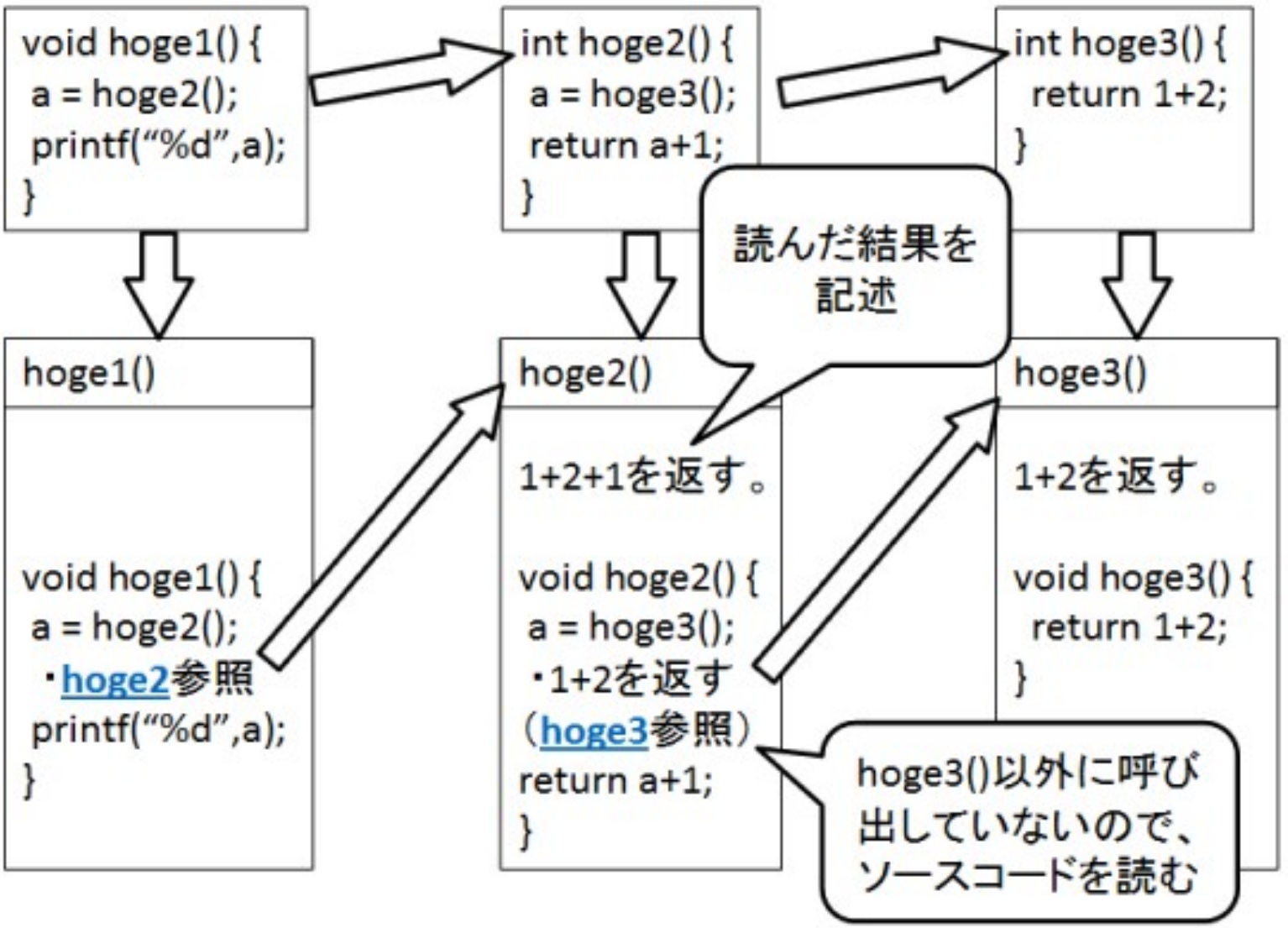
ページの作り方③


void hoge1() {
a = hoge2();
printf("%d",a);
}
int hoge2() {
a = hoge3();
return a+1;
}
int hoge3() {
return 1+2;
}
hoge1()
void hoge1() {
a = hoge2();
・hoge2参照
printf("%d",a);
}
hoge2()
void hoge2() {
a = hoge3();
・hoge3参照
return a+1;
}
Wikiページを作成
hoge2()のソースをはりつける
また、hoge2()の中で、hoge3()を呼び出しているので、リンクを追記

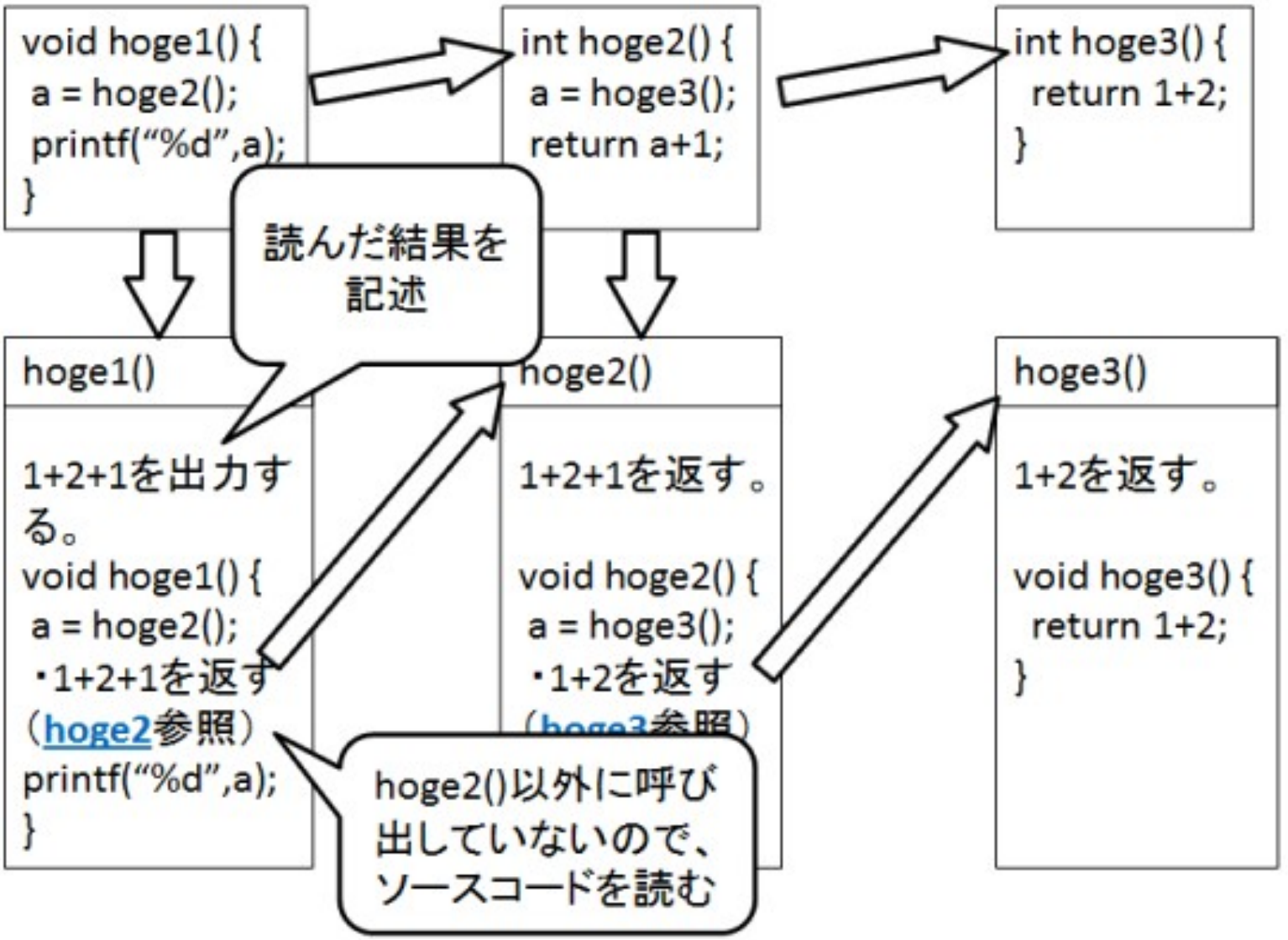
ページの作り方④


void hoge1() {
a = hoge2();
printf("%d",a);
}
int hoge2() {
a = hoge3();
return a+1;
}
int hoge3() {
return 1+2;
}
hoge1()
void hoge1() {
a = hoge2();
・hoge2参照
printf("%d",a);
}
void hoge2() {
hoge3()
1+2を返す。
void hoge3() {
return 1+2;
}
Wikiページを作成
読んだ結果を
記述
hoge3()のソースを
はりつける
他の関数を呼び出
していないので、
ソースコードを読む

ページの作り方⑤


void hoge1() {
a = hoge2();
printf("%d",a);
}
int hoge2() {
a = hoge3();
return a+1;
}
int hoge3() {
return 1+2;
}
読んだ結果を記述
hoge1()
void hoge1() {
a = hoge2();
・hoge2参照
printf("%d",a);
}
hoge2()
1+2+1を返す。
void hoge2() {
a = hoge3();
・1+2を返す
(hoge3参照)
return a+1;
}
hoge3()
1+2を返す。
void hoge3() {
return 1+2;
}
hoge3()以外に呼び出していないので、ソースコードを読む

ページの作り方⑥


void hoge1() {
a = hoge2();
printf("%d",a);
}
int hoge2() {
a = hoge3();
return a+1;
int hoge3() {
return 1+2;
}
読んだ結果を記述
hoge1()
1+2+1を出力する。
void hoge1() {
a = hoge2();
・1+2+1を返す
(hoge2参照)
printf("%d",a);
}
hoge2()
1+2+1を返す。
void hoge2() {
a = hoge3();
・1+2を返す
hoge3()
1+2を返す。
void hoge3() {
return 1+2;
}
hoge2()以外に呼び出していないので、ソースコードを読む

実際にやってみよう！

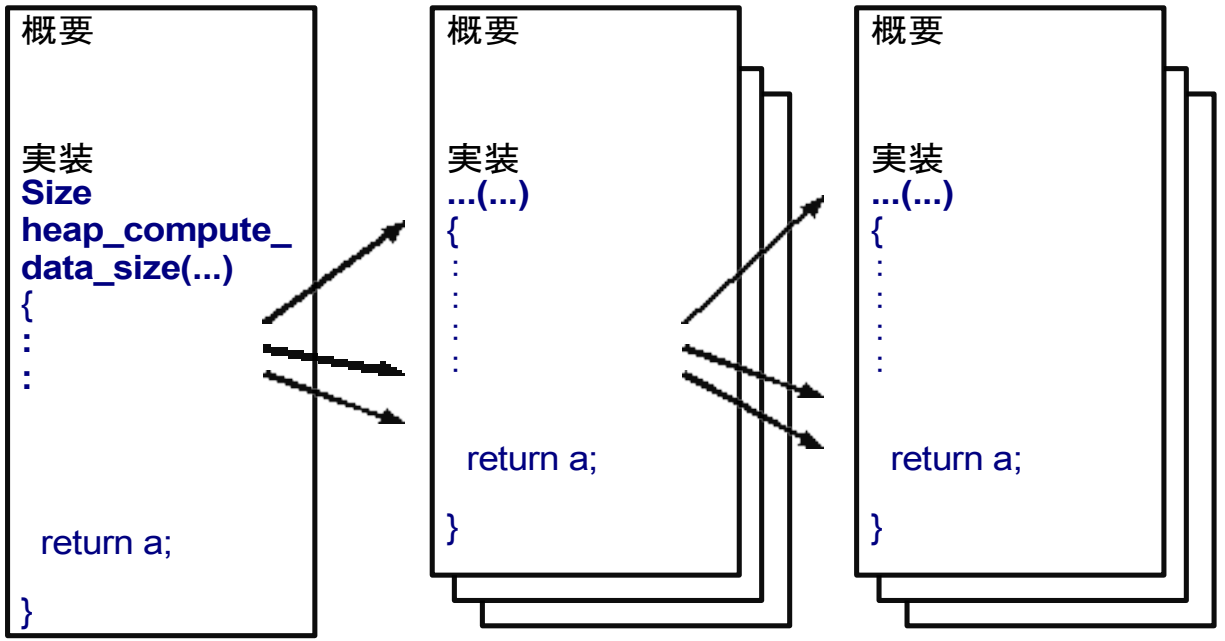
- PostgreSQL-8.1.4のソースを読むことにしました。
- 「PostgreSQL解読室」というWikiサイトを作りました。

どこから読みはじめる？

- とりあえず、PostgreSQLのソースから適当に選んだheap_compute_data_size()という関数を読むことにした。

どこから読み始める？

順番に読まず、まず
heap_compute_data_size()から呼ばれる関数・マクロ・変数・データ型等のページを一度に作ってしまった。



概要をどう書くか？

```
/*
 * Object ID is a fundamental type in Postgres.
 */
typedef unsigned int Oid;
```

- 例えば上記のような、いろいろな変数が出てくる。
- 「Object ID is a fundamental type in Postgres.」とコメントに書いてある。
- うーん、じゃあ、Oidの概要には、「オブジェクトIDの定義」って書いておくか。

最上位関数に戻ってきた

```
/*
 * heap_compute_data_size
 *      Determine size of the data area of a tuple to be constructed
 */
Size
heap_compute_data_size(TupleDesc tupleDesc,
                                    :
                                    :
```

- 下位のページについて概要を(大体)全部書いた。
- heap_compute_data_size()の概要を書こう！
- コメントに「Determine size of the data area of a tuple to be constructed」と書いてある。
- うーん、じゃあ、これを訳して、「構成されるタプルのデータ領域のサイズを決定する」って書いておくか。

あれ？なんか変だ？

```
Size
heap_compute_data_size( ･･･
{
                                        :

    data_length = att_align(data_length, att[i]->attalign);
```

- data_lengthに、以下の値を代入。
  - attalignの値が'i'の場合、…
  - attalignの値が'c'の場合、…

```
                                        :
    data_length = att_addlength(data_length
```

- data_lengthに、以下の値を加算。
  - att[i]->attlenが0より大きい場合、…
  - att[i]->attlenが-1の場合、…

```
                                        :
```

下位関数の概要を無視してないか？

- コメントを概要にしていいなら、なんで下位関数を読まないといけないんだろう？

そういえば

ソースコード

読んでなかった！

orz

反省

```
/*
 * Object ID is a fundamental type in Postgres.
 */
typedef unsigned int Oid;
```

ここを読む

- コメントにとらわれてはいけない！
- コードから読み取れないことは、書くのをやめよう！
- Oidの概要には、「オブジェクトIDの定義」ではなく、「unsigned intの別名」のように書こう！

全部作り直しorz

- コメントの訳は備考に移動しました。
- 当面、「コードから読み取れないこと」を概要に書くことをやめました。（今でもそのままです）

よーしどんどん読むぞー！

- コードに書いてあることだけを忠実に追うことにした！
- 下位関数の概要から、上位関数の概要をかけるようになった！
- これは、とても楽しい！

最上位関数に戻ってきた！

- 上位関数の概要を書くのは結構難しい！
- 書けた時は感動した！

再び気が付いた

- 適当な順番で読んでいたので、ページの作り忘れがある。

Byひらさん「順番に読もうよー」

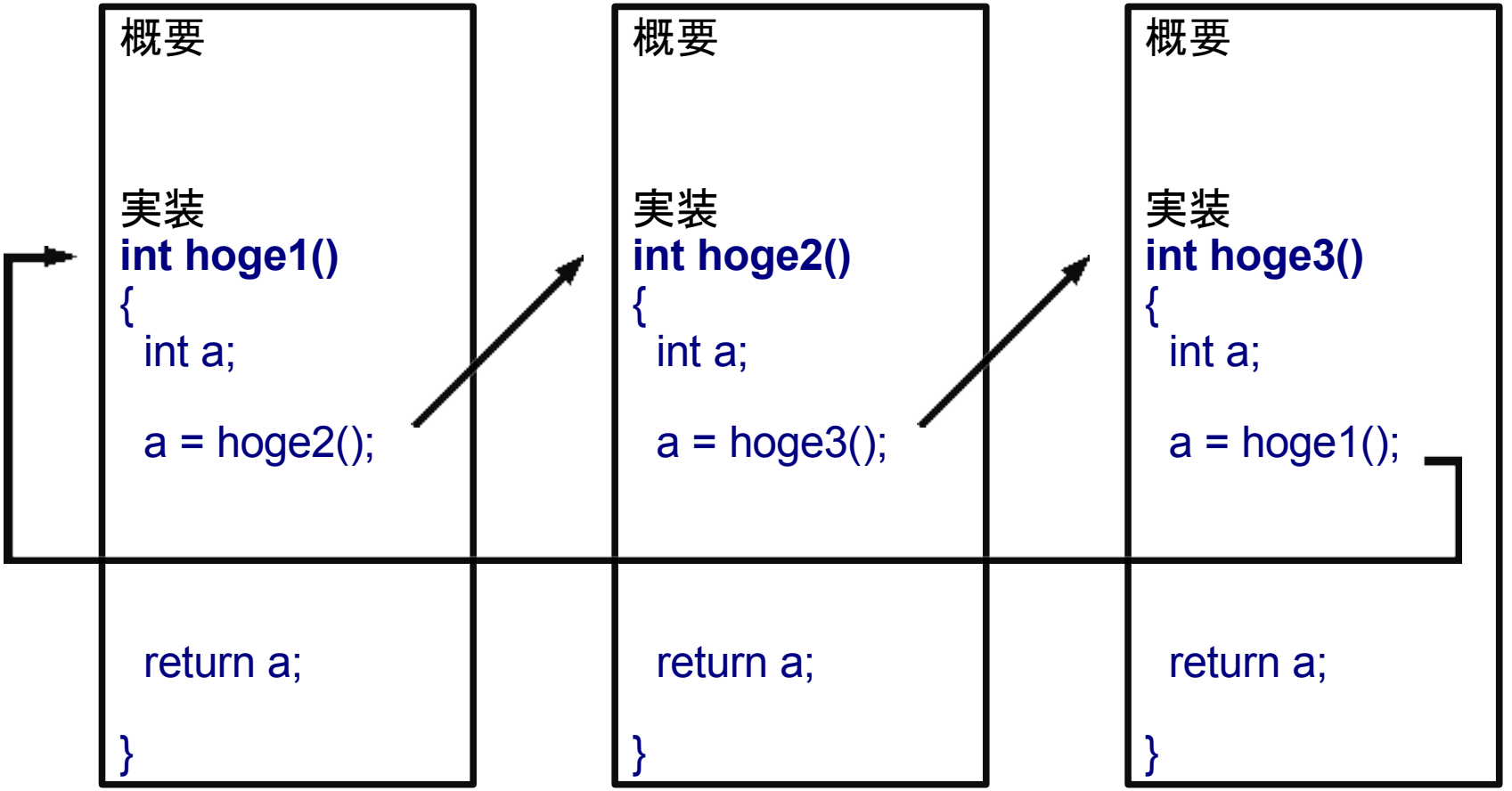
- ひらさんに話したらおこられた！
- しかし、この時点では、なんで順番にこだわるのか、いまひとつわからなかった！

よーし順番に読むぞー！

- とにかく、順番に読むようにして継続。

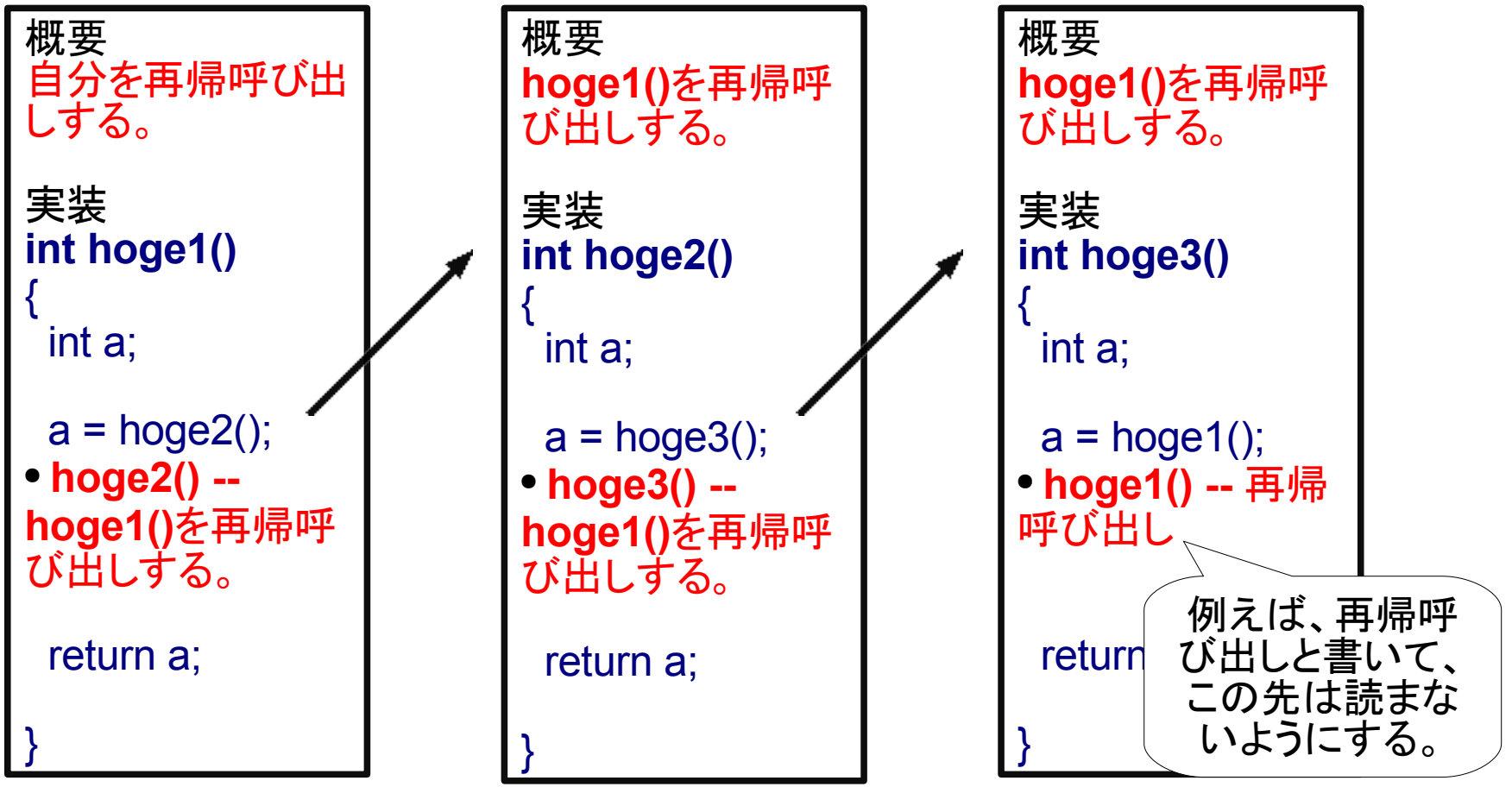
再帰呼び出しにぶつかった！

どれが最上位なのか？



順番に読んでれば大丈夫！

先にページを作った関数が必ず上位



教訓

- 順番を守ろう！
- ひらメソッドで読んでいくために、読み順を理解していることが非常に重要だとわかりました。
- 補足すると、正確には「CPUが実行するのと同じ順序で読む」ということのようです。

その他の失敗談

- ソースコードをgrepしても定義が見つからない！
- いそがしくて、続かない！
- むずかしくて、読んでもわからない！

ソースコードに定義がない

- ソースをgrepしても、定義が見つからない

どこかにある！

- ググってみる
  - C言語や標準ライブラリは、入門書にかかれているよりずっと多機能です。
  - ネットにあるC言語の解説サイトが参考になります。
- ググっても見つからない場合
  - マクロなどの識別子が、configureスクリプトなどで定義されていることもあります。
    - ソースをgrepする際に、configureスクリプト、makefileなどをgrep対象に含めておくと良いです。
  - ビルド環境のヘッダーファイルなども参考になります。

いそがしくて続かない

- 仕事がいそがしい
- モチベーションが続かない
- 疲れた
- 友達が電話をかけてくる

解読日記をつけて、人に見せよう！

4/29訂正：
mixiでないといけない、というわけではなく、blogなどでももちろんよいです。自分の性格に一番合う方法を選ぶのがいいと思います。

- ~~日記をつけてmixiなどに載せていると、ゆっくりでも続きます。~~日記をつけて友人・知人に見せるなどをしていると、ゆっくりでも続きます。
  - 失敗談など、自分のダメなところを晒すのがモチベーション維持につながります。
- 人に見せることが重要です。
- 解読日記は、自分の無知や恥をさらすくらいの気持ちで書いた方が、いいものが書けて、スキルアップにつながります。

読んでもわからない

たとえば、

```
#define TYPEALIGN(ALIGNVAL,LEN) ¥
(((long) (LEN) + ((ALIGNVAL) - 1)) & ~((long) ((ALIGNVAL) - 1)))
```

処理内容をコードからそのまま表現すると、

「LEN+ALIGNVAL-1の値と、ALIGNVAL-1の値の補数の間で、ビットごとのAND演算を行い、その結果を返す。」

これでは、上位関数の概要が書けません

読んでもダメなら、動かしてみよう！

- サンプルを作って動かす。
- 紙に実行結果を何通りか書いてみる。

ALIGNVAL	LEN	結果
4	9	12
4	7	8
4	5	8
4	4	4

前スライドの場合：

「LEN以上であるALIGNVALの倍数のうち、最小値を返す。」

- コメントに逃げてはいけない！
  - 自分の場合、コメントに逃げると動作が追えなくなりました。

ありがとうございました。

本資料のライセンスについて

- 本資料は、GPLv2にて公開いたします。