

プログラム設計

<http://bit.ly/design4d>

構造化技法で使用するダイアグラム

後期 第12週

2019/12/16

今回学ぶダイアグラム




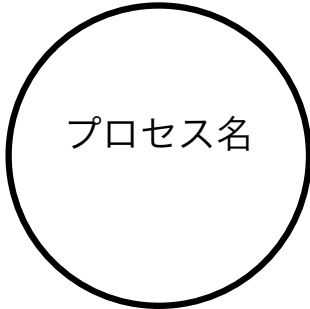
データフロー図 (DFD)

データの流れと処理の流れの観点で、システムの機能を整理するための図
(構造化手法の分析段階で利用される)

モジュール構造図

モジュール間の構造と引数を表現する
(DFDの「プロセス」をモジュールとする)

DFDの基本要素

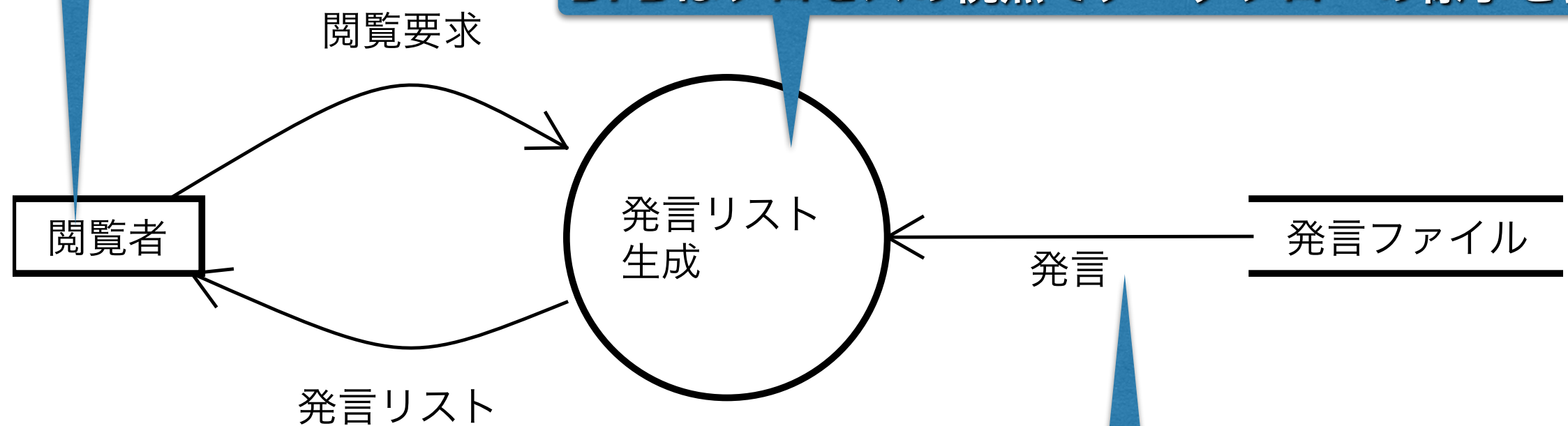
要素	説明	表記
源泉/吸収 (外部エンティティ)	データの入力元/ データの出力先	
データフロー	データの流れ	
データストア (ファイル)	データ蓄積場所	
プロセス (バブル)	データの処理	

DFDの例

掲示板の「閲覧」に関するデータフローを表す

システムの外部要素は外部エンティティとして表す

DFDはプロセスの視点でデータフローの様子を表す



ファイルからの読み込み（入力）を表している

DFDとユースケース図の関係

この2つの図の間では、次のような対応関係を作りやすくになっています。

- ▶ 外部エンティティ（源泉/吸収） → アクタ
- ▶ プロセス（バブル） → ユースケース

構造化技法とは

- ▶ **分割統治**や**モジュラリティ**を基盤とした開発技法
(これらについては後述)
- ▶ **手続き型言語** (C, Perlなど) と相性が良い

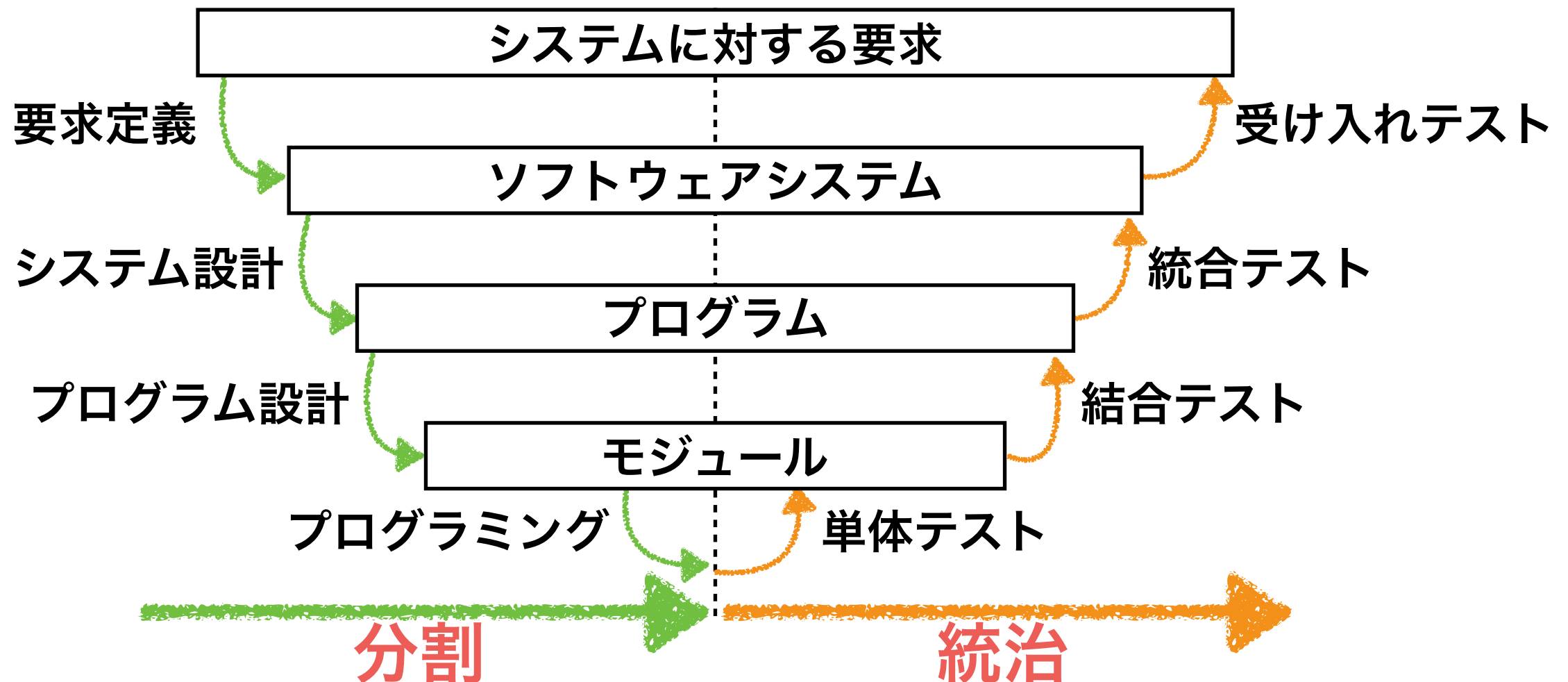


オブジェクト指向技法

- ▶ **クラス**や**インスタンス**の概念を基盤とした開発技法
- ▶ **オブジェクト指向型言語** (Java, Python, C++, C#, Ruby, Swiftなど) と相性が良い

分割統治

ソフトウェアをいくつかの独立した部分（モジュール）に**分割**していき、単純化した部分をそれぞれ開発することで、最終的にそれらの部分を**統治**（統合）するように開発するという考え方。



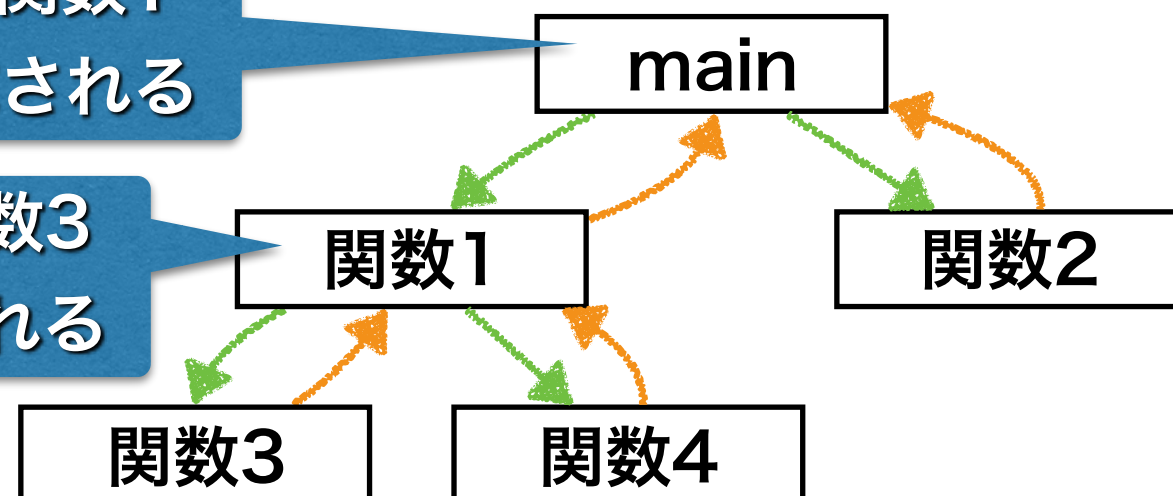
モジュラリティ

ソフトウェアを複数の**モジュール**に分割し、それぞれのモジュールに独立した機能を与え、ソフトウェア全体を管理すること。

「モジュール = 関数」の場合

mainの処理は、関数1
と関数2から構成される

関数1の処理は、関数3
と関数4から構成される



呼び出す →
戻る →

モジュール構造図

DFDの「プロセス」をモジュールとみなして、モジュール間の**構造**と**引数**を表現する。

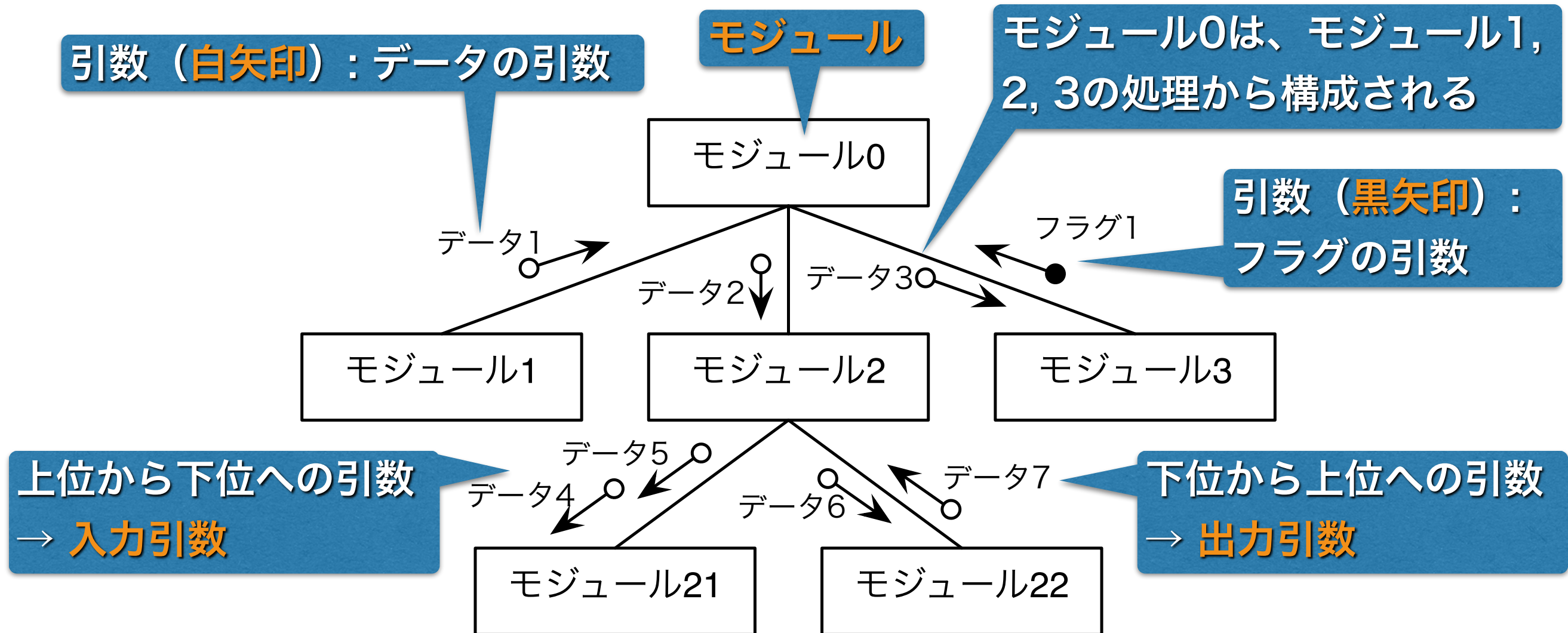
▶ モジュール分割の構造

利用する側（上位モジュール）と利用される側（下位モジュール）の**木構造**

▶ モジュール間の関係

上位と下位の間のインタフェースは、**引数**（入力引数, 出力引数）によってデータをやりとりする

モジュール構造図の基本要素



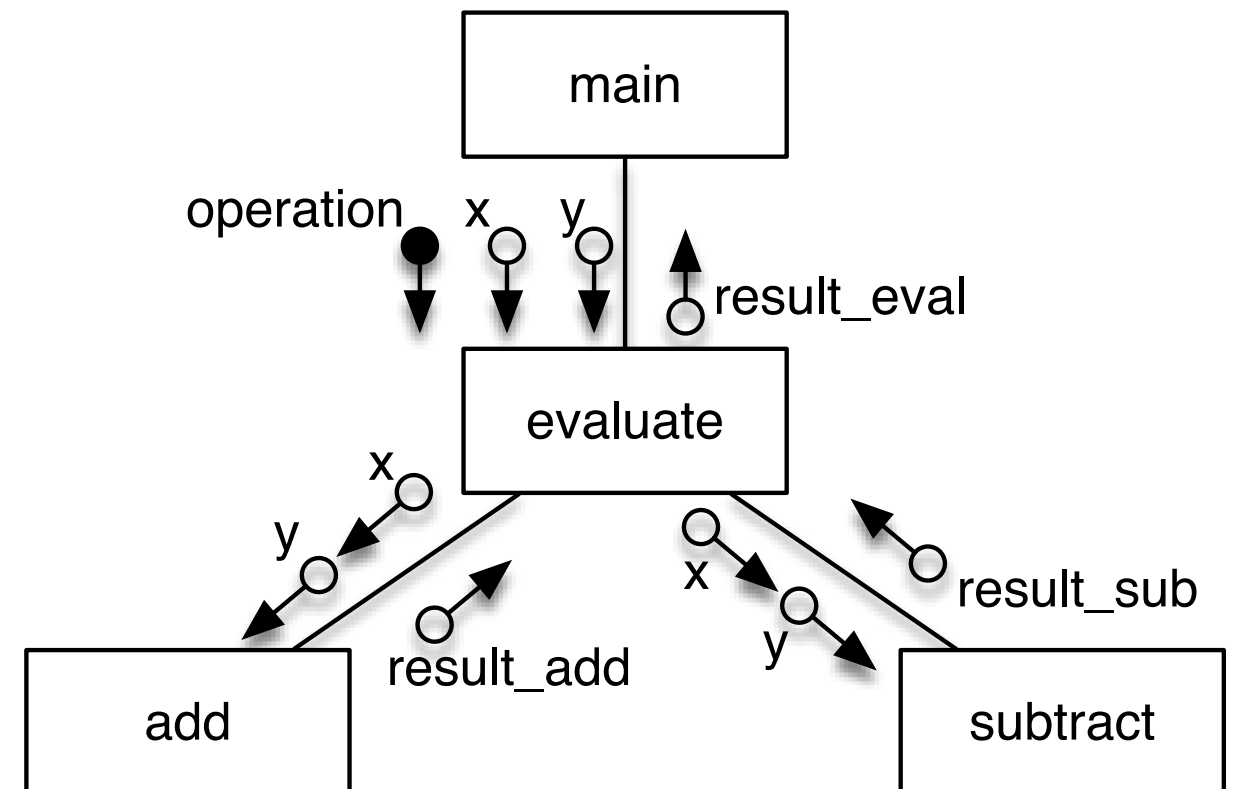
モジュール構造図とプログラムの関係

```
int add(int x, int y)
{
    int result_add;
    result_add = x + y;
    return result_add;
}

int subtract(int x, int y)
{
    int result_sub;
    result_sub = x - y;
    return result_sub;
}

int evaluate(int x, int y, int operation)
{
    int result_eval = 0;
    if (operation == 1) {
        result_eval = add(x, y);
    } else if (operation == 2) {
        result_eval = subtract(x, y);
    }
    return result_eval;
}
```

```
int main()
{
    int x, y, r;
    x = 20;
    y = 30;
    r = evaluate(x, y, 1);
    printf("r: %d\n", r);
    r = evaluate(x, y, 2);
    printf("r: %d\n", r);
    return 0;
}
```



【課題の準備】

演習室で作業する前に、以下のコマンドを
入れるだけで準備が完了する

```
$ mygitclone4d 「自分のGitHubユーザ名」  
$ cd prog4d-(ユーザ名)  
$ ./myconf
```

※本体をシャットダウンするまでは、
上記「mygitclone」と「myconf」の設定は有効です

【課題の準備】

以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して後期第11週のフォルダを作る
\$ cd prog4d-(ユーザ名) (←既に移動しているなら不要)
\$ mkdir week212
\$ cd week212

※課題で作るファイル名は各自で決めて構いません。

【練習12-1】

astahを使って、データフロー図のスライドで示した例を作ってみましょう。

【課題12-1】

練習12-1で作成したデータフロー図に、次のような要素を追加して、「発言の書き込み」処理を表すデータフロー図を描いて下さい。

▶ 源泉：発言者

▶ プロセス：発言書き込み

▶ データフロー：発言

「発言者」から「発言書き込み」を通して、
「発言ファイル」へと流れる

【課題の提出】

以下の流れで、GitHubにプッシュしてWebサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
```

```
$ git commit -m “課題12-1提出”
```

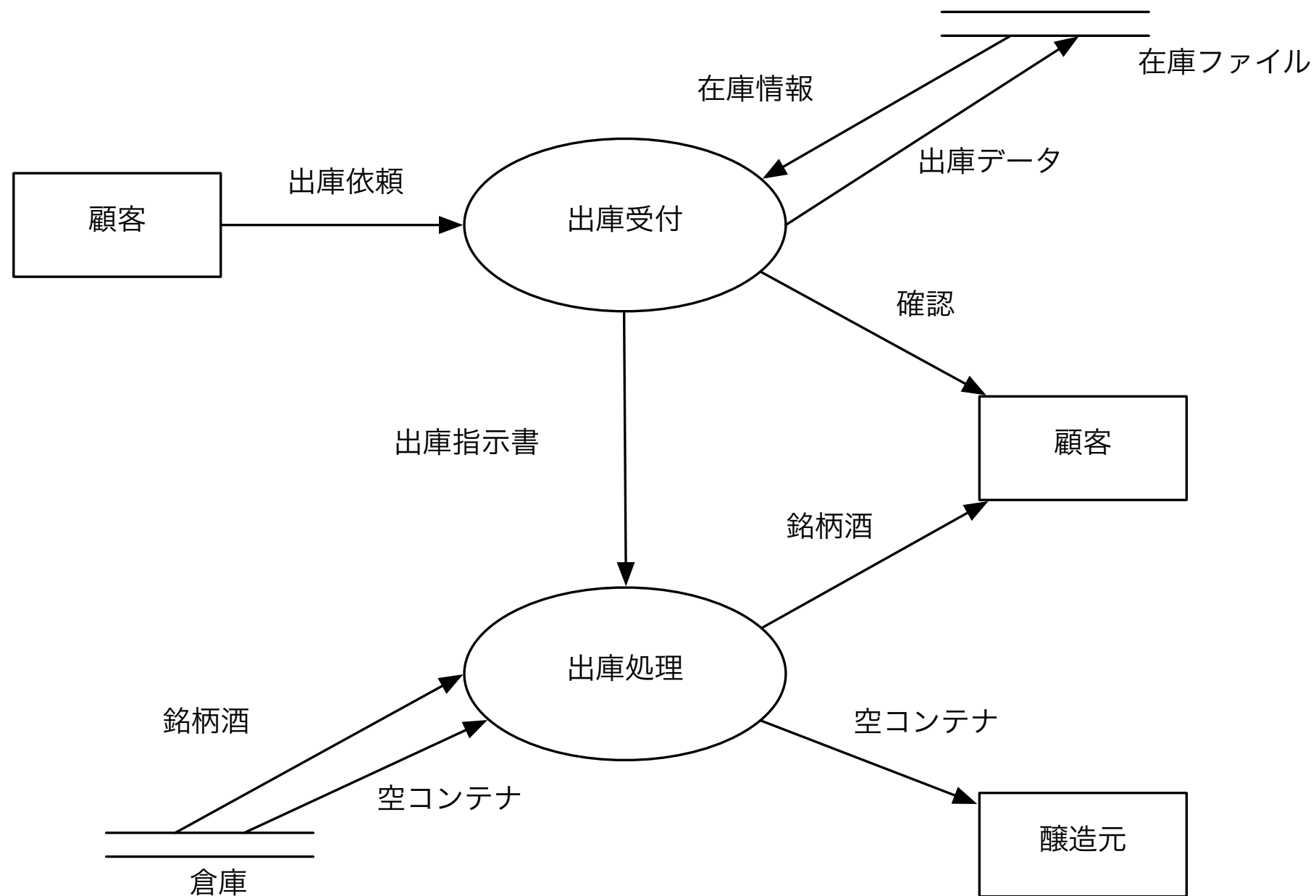
```
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog4d-2019/prog4d-\(ユーザー名\)](https://github.com/nit-ibaraki-prog4d-2019/prog4d-(ユーザー名))

【課題12-2】

次のような「酒屋の出庫」に関するデータフロー図をastahで作成して下さい。



【課題の提出】

以下の流れで、GitHubにプッシュしてWebサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
```

```
$ git commit -m “課題12-2提出”
```

```
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog4d-2019/prog4d-\(ユーザー名\)](https://github.com/nit-ibaraki-prog4d-2019/prog4d-(ユーザー名))

小テストについて

小テストの注意点

- 他人の力は借りずに、自分だけでプログラムを作成する。（つまり定期試験と同様）
- プログラムの提出はGitHubを使用する。

小テストについて

小テスト中に参照できるもの

- 教科書, 参考書, 配付資料
- 自分のホームディレクトリ（ホームフォルダ）以下に保存されているファイル
- 小テストでは紙媒体のものは参照可能
- 上記以外の情報を参照することは不正行為とする
例：USBで接続された機器に保存されているファイルの参照
Webブラウザ、ネットワークを介した情報の参照
自分のPCを使用する、など