プログラム設計

http://bit.ly/design4d

ソフトウェア工学について
ステートマシン図とプログラムの対応

後期 第14週

2020/1/6

本日は・・・

- **ソフトウェア開発の現状と問題**
- **ソフトウェア開発工程・方法論**
- **ステートマシン図とプログラムの対応**

本日は・・・

- ソフトウェア開発の現状と問題
- ソフトウェア開発工程・方法論
- ステートマシン図とプログラムの対応

ソフトウェア開発の現状

- 大規模化

  必要なソースコードの行数が増大

  → 多くの開発者が共同で開発する必要あり

- 複雑化

  社会が複雑化するのに合わせ、ソフトウェアへの要求も複雑化

ソフトウェア開発の問題

- 大規模化

  必要なソースコードの行数が増大

  → 多くの開発者が共同で開発する必要あり

  → 共同開発は難しい

  　共同開発によるバグの増大

- 複雑化

  社会が複雑化するのに合わせ、ソフトウェアへの

  要求も複雑化

  → 複雑化により設計, 実装がより難化

大規模化/複雑化したソフトウェアを完成させる方法論が必要

ソフトウェア危機

ソフトウェアへの依存度が大きくなっていることに対する危機感を以下の主要な5項目で表す

- ソフトウェアの巨大化, 複雑化に伴う開発費用の増大
- ハードウェア 対 ソフトウェアのコスト比の変化
  （「ハード ＞ ソフト」 から 「ハード ＜ ソフト」）
- ソフトウェア保守にかかる工数の増大
- ソフトウェア需要に対する供給能力の低下
- ソフトウェアトラブルの社会的問題化

ソフトウェア工学について

次のような目的をもった工学分野

- ソフトウェアの生産性，信頼性の低下を防ぐ
- 学術的(理論や方法論)および、実践的(技術や技法)な分野を目指している

ソフトウェア工学が確立した経緯

- 1968年：ソフトウェア工学とソフトウェア危機が提唱される
- 1970年代後半：ソフトウェア工学の分野が定着していく
  （ソフトウェア工学国際会議が開催される）

本日は・・・

- ソフトウェア開発の現状と問題
- ソフトウェア開発工程・方法論
- ステートマシン図とプログラムの対応

ソフトウェア開発手法

ソフトウェアを開発・運営する過程→ライフサイクル
ライフサイクルはプロセスモデルで表現される

代表的なプロセスモデル

- **ウォーターフォールモデル**
- **スパイラルモデル**
- **アジャイルソフトウェア開発**

ウォーターフォールモデル

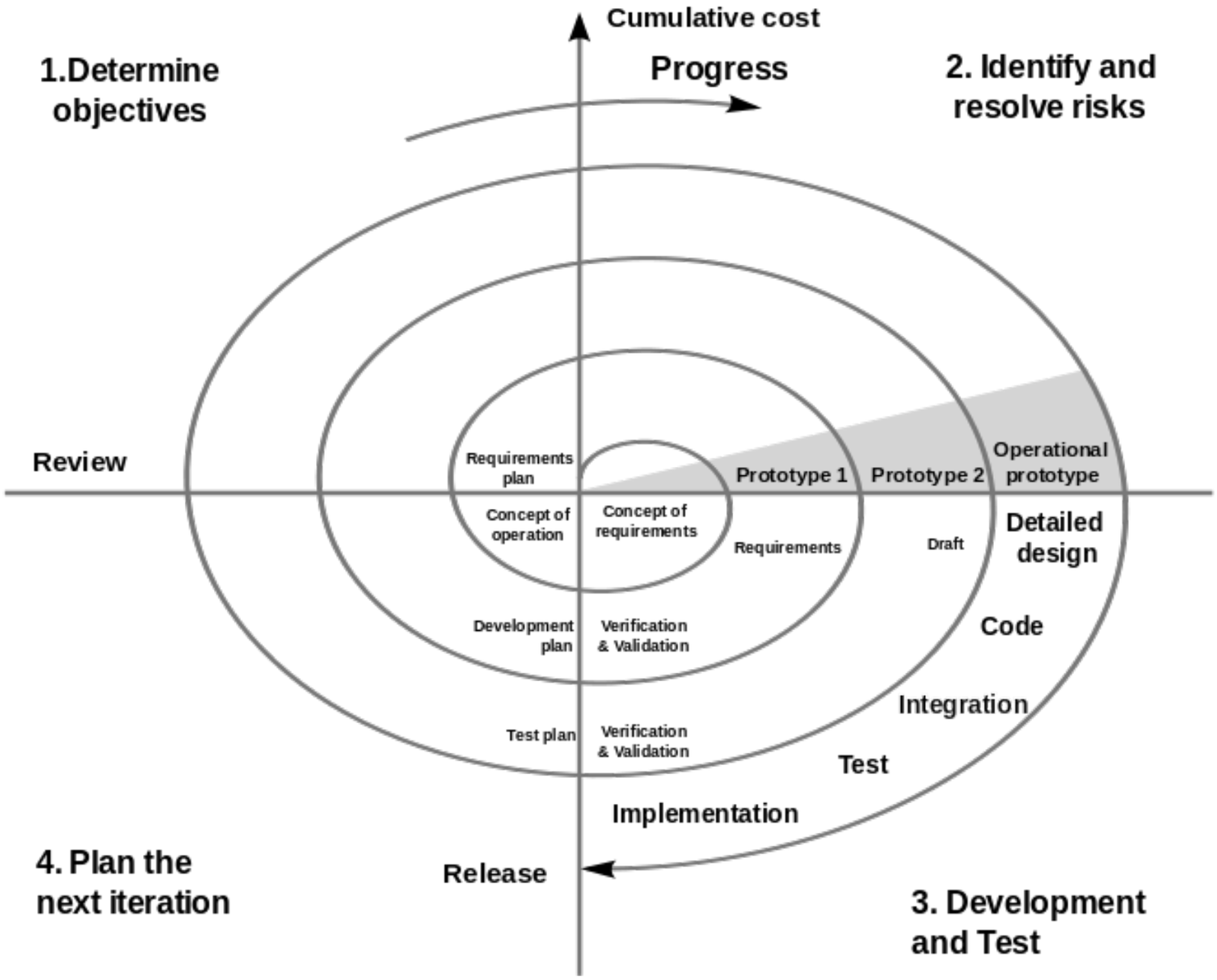
いくつかの工程に分けて開発を進める

上流工程：利用者側の視点（要求定義, 基本設計）

下流工程：開発者側の視点（詳細設計, 実装, テスト, 保守）

スパイラルモデル

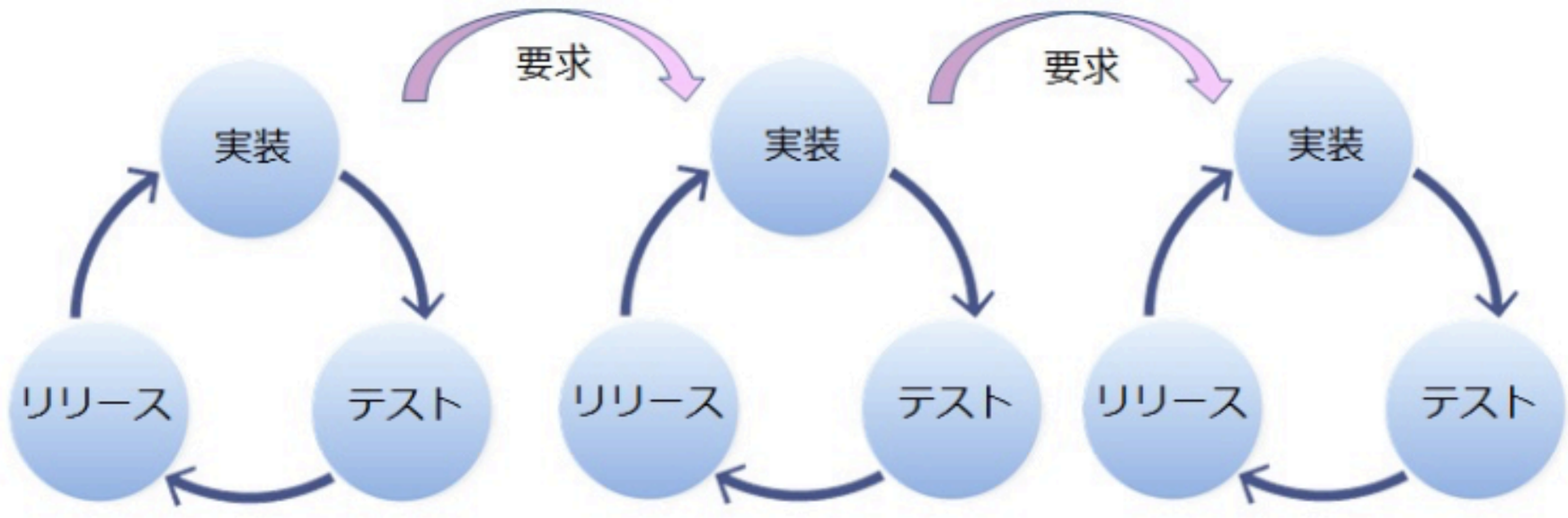
- 時間の経過とともに、ある方向へ向けて繰り返しながら成長する
- 4つのフェーズを1サイクルとし、プロトタイピングを繰り返す



※Wikipediaより

アジャイルソフトウェア開発

- Extreme Programmingが有名
- 軽量で短期間の開発に適している
- リリース（ある程度の形になって動作するソフトウェア）
  → 2〜3ヶ月間隔
- イテレーション（ソフトウェアの部分的な設計・実装・テスト）
  → 2〜3週間



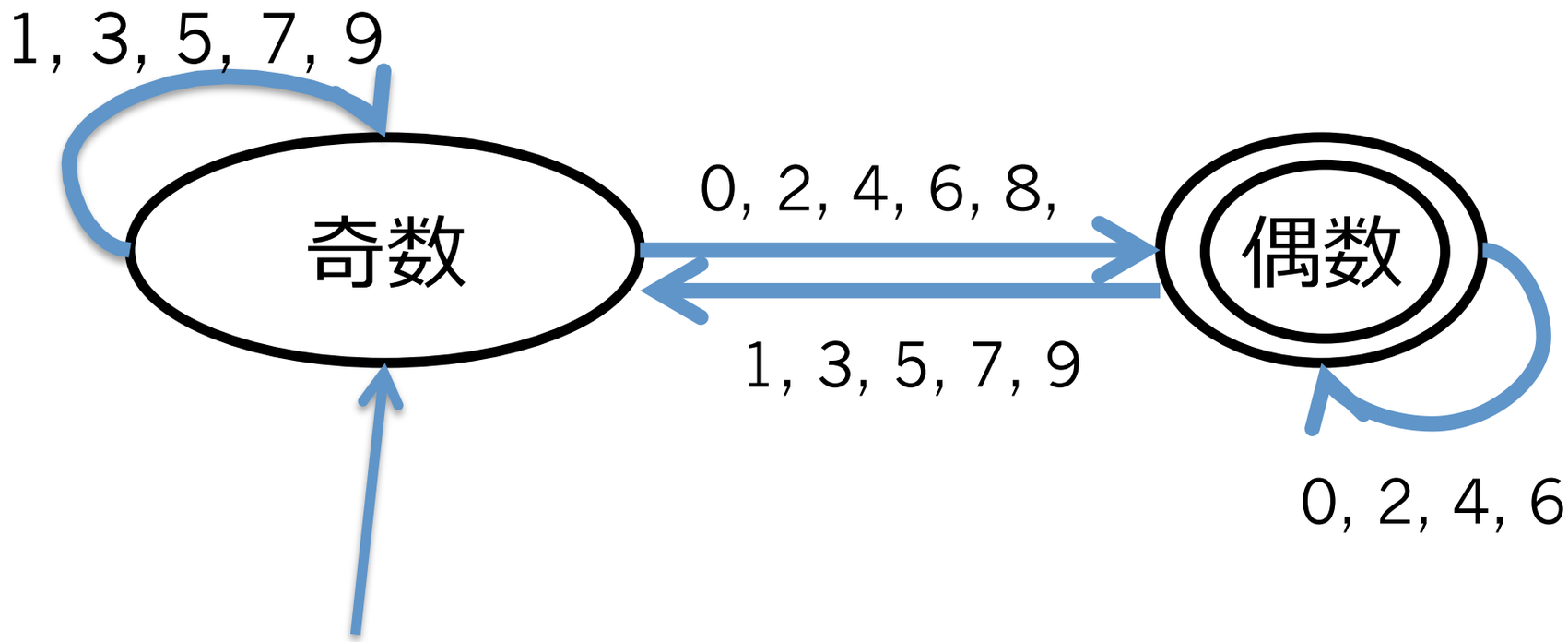
いくつかのイテレーションを完了することでリリースを達成し、
リリースを繰り返すことで完成度を上げる

本日は・・・

- ソフトウェア開発の現状と問題
- ソフトウェア開発工程・方法論
- ステートマシン図とプログラムの対応

【例】偶数を受理する状態遷移図

入力を先頭から読み込み末尾の数値が偶数ならば受理



「124」を入力した場合

開始状態（奇数）

→1（奇数のまま）

→2（偶数に遷移）

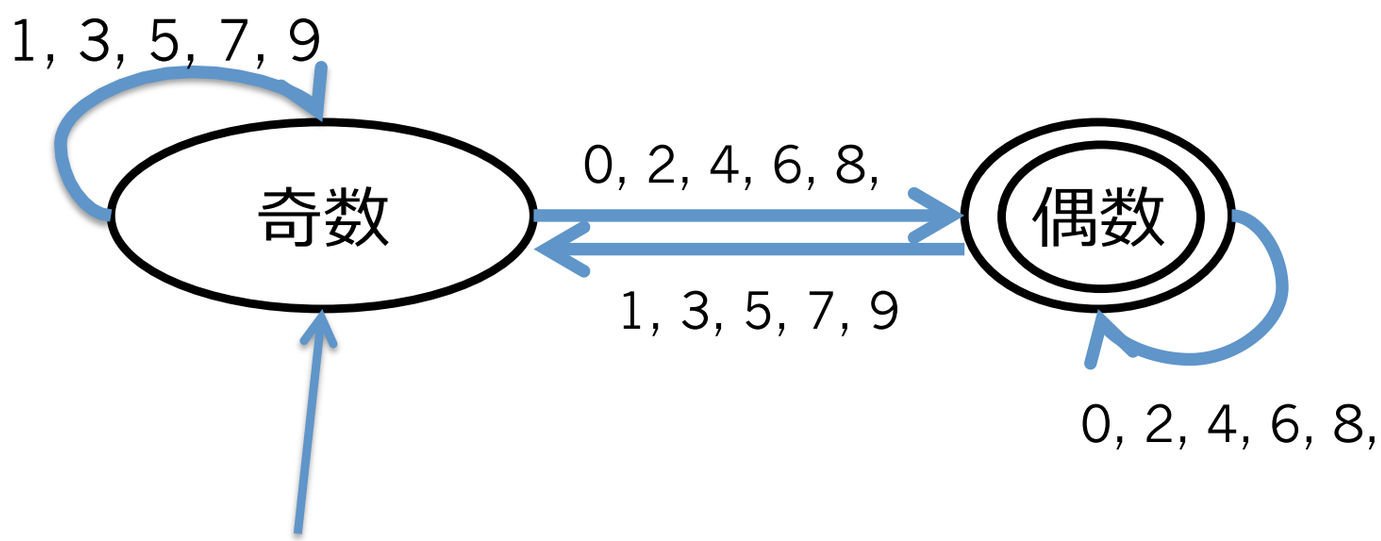
→4（偶数状態で終了=受理）

※3Dプログラミング応用より

C言語によるオートマトンの実装

大まかな流れ

①必要な変数を宣言
　（ユーザからの入力を格納する変数, 現在の状態を保持する変数）
②文字列を入力させる
③while文で先頭から1文字ずつ読み込みながら、
　switch文で状態を遷移する
④すべて読み込み終わったときに終了状態に居れば受理



※3Dプログラミング応用より

①②の処理

```
/* (1) 変数を宣言 */
char input[100];        //入力を格納する配列
int i=0;                //入力数をカウントする

//状態は、0が奇数、1が偶数の状態を表す
int current_state=0; //現在の状態（初期状態）
int fin_state=1;     //終了状態

/* (2) 文字列の入力 */
printf("数字を入力してください。\n");
scanf("%s", input);
```

※3Dプログラミング応用より

③の処理

```
/* (3) 先頭から順に読み込みながら状態遷移 */
while(input[i]!='\0') {
    switch(current_state) {
    case 0:
        if(input[i]=='0' || input[i]=='2'
           || input[i]=='4' || input[i]=='6'
           || input[i]=='8') {
            current_state = 1;
        } else {
            current_state = 0;
        }
        break;
    case 1:
        if(input[i]=='1' || input[i]=='3'
           || input[i]=='5' || input[i]=='7'
           || input[i]=='9') {
            current_state = 0;
        } else {
            current_state = 1;
        }
        break;
    }
    printf("読み込んだ数値 : %c 遷移先 : %d\n", input[i], current_state);
    i++;
}
```

入力された文字列の終端文字まで繰り返す

現在の状態が0の場合（奇数）

次の状態に遷移

現在の状態が1の場合（偶数）

次の状態に遷移

※3Dプログラミング応用より

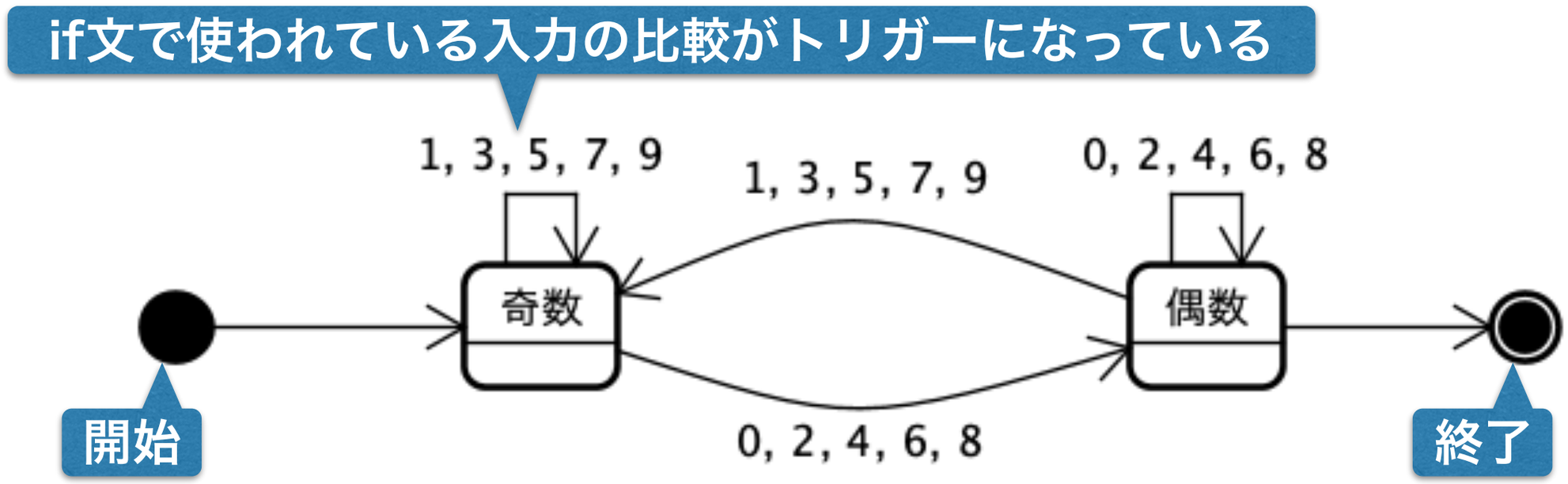
④の処理

```
/* (4) 終了状態にいるか判定 */
if(current_state == fin_state) {
    printf("受理する。\n");
} else {
    printf("受理しない。\n");
}
```

全ての入力に対して遷移をした後に、
終了状態ならば「受理する」

※3Dプログラミング応用より

ステートマシン図


if文で使われている入力の比較がトリガーになっている
1, 3, 5, 7, 9
1, 3, 5, 7, 9
0, 2, 4, 6, 8
奇数
偶数
開始
0, 2, 4, 6, 8
終了

【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14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 week214
$ cd week214
```

※課題で作るファイル名は各自で決めて構いません。

【練習14-1】

偶数を受理するステートマシン図の例を、astahを使って作りましょう。

【課題14-1】

「pd14-1-sample.c」に示した「3の倍数なら受理する」プログラムのステートマシン図を、astahを使って作りましょう。

【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14-1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-(ユーザ名)