# アルゴリズム論

2017年4月10日

樋口文人

# 担当

樋口文人

ひぐち ふみと

wenren@meiji.ac.jp

# 授業の概要・到達目標

- 種々のアルゴリズムとデータ構造の概念について講義を行う。また，説明したアルゴリズムの復習とその実践的な利用について演習を行う。
- 高度なアルゴリズムおよびその対となる概念であるデータ構造を基礎から学ぶことで，プログラミングの確かな基盤を習得する。また，アルゴリズムを数学的な観点から捉える考え方を学ぶことで，正しく効率的に動作するプログラムを組めるようになることを目標とする。

# アルゴリズム

- 問題解決の手順
- コンピュータの発達に伴い重要性を増す
  - 解ける問題，解けない問題
  - 抽象的
    - コンピュータの性能に依存しない問題の難しさの評価
    - 問題の規模に依存しない手順の性能評価
  - 具体的
    - 解答を得るのに掛かる時間の見積り
    - 「答えが出ない」ことを避ける

# アルゴリズム論

- アルゴリズへの理解
  - 問題の大きさとアルゴリズムの性能
- CPUの動作とメモリ上のデータの動きをイメージ
- メモリの活用
  - 線形なデータ構造
    - 番号による要素の指定，文字列による要素の指定
  - 木構造
  - ネットワーク構造（グラフ構造）の入り口

# 授業

- 今年度から実習を充実させたい
- 前半は講義
- 後半はプログラム作成の準備
- 宿題はプログラムの完成と実行結果の報告
- 中間と最終回にテスト
  - 期末試験とはしない
- 出欠をとる

# 授業内容

第１回：アルゴリズムの考え方
第２回：変数と配列
第３回：探索と整列
第４回：効率的な整列
第５回：効率的な探索
第６回：ハッシュと辞書
第７回：中間テストとその解説
第８回：集合と辞書
第９回：線形リスト
第１０回：木構造
第１１回：グラフ
第１２回：最短経路
第１３回：再帰
第１４回：再帰と計算効率
第１５回：確認テストとその解説

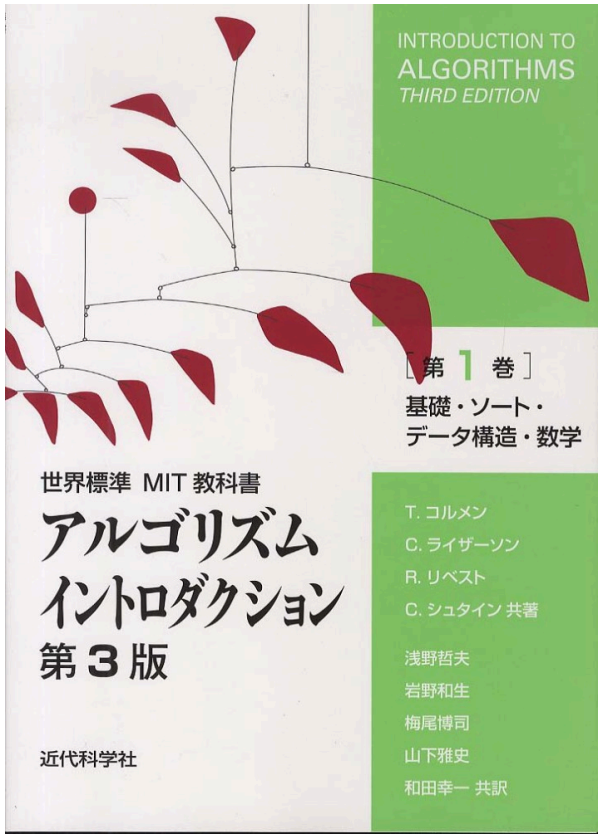
# 参考書

アルゴリズム・サイエンス:入口からの超入門 (アルゴリズム・サイエンスシリーズ 1—超入門編)


アルゴリズム・サイエンス:出口からの超入門 (アルゴリズム・サイエンスシリーズ 2—超入門編)

# 自習テキスト　1

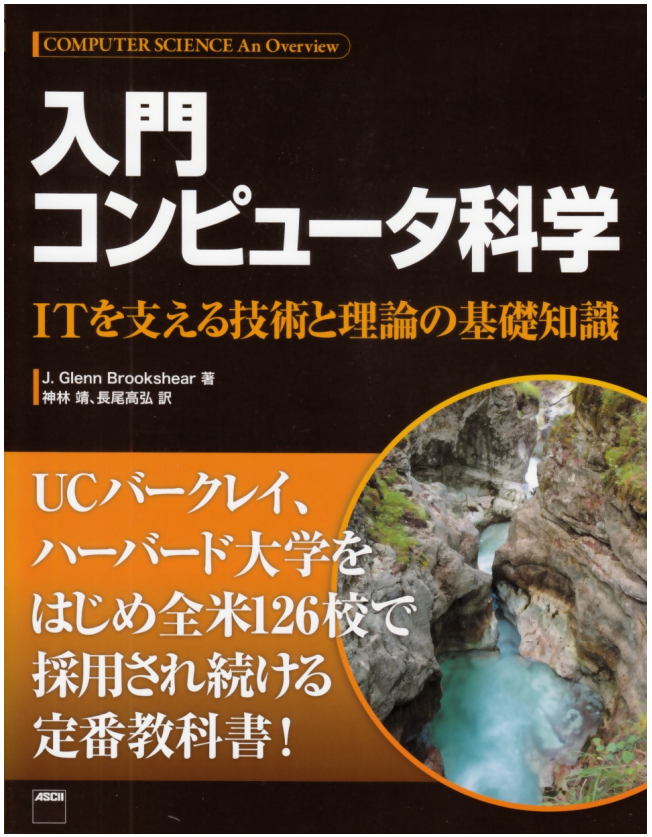
アルゴリズムイントロダクション 第3版 第1巻: 基礎・ソート・データ構造・数学 (世界標準MIT教科書)




アルゴリズムイントロダクション 第3版 第2巻: 高度な設計と解析手法・高度なデータ構造・グラフアルゴリズム (世界標準MIT教科書)

入門 データ構造とアルゴリズム


入門 コンピュータ科学
ITを支える技術と理論の基礎知識

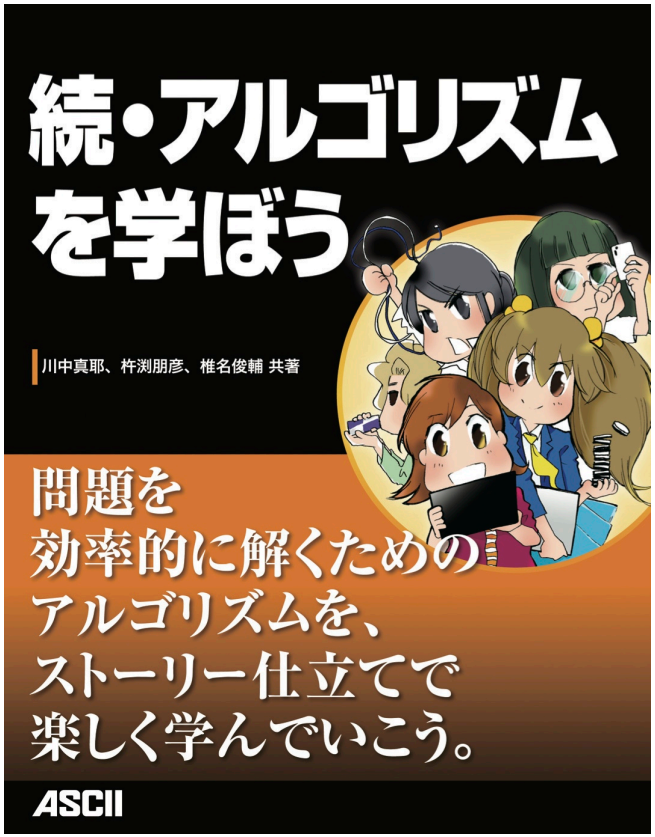


# 堅い本を敬遠したい人向け

誤植の指摘あり：https://sites.google.com/site/adtalgo/home

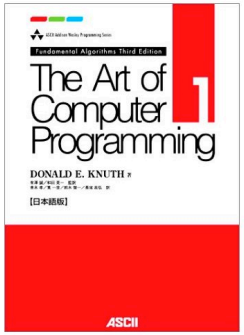
**アルゴリズムを学ぼう** 第1版第2刷

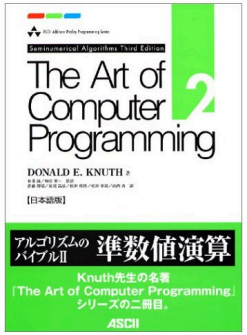

**続・アルゴリズムを学ぼう**

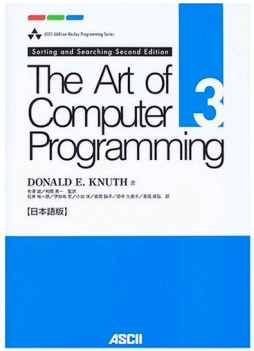


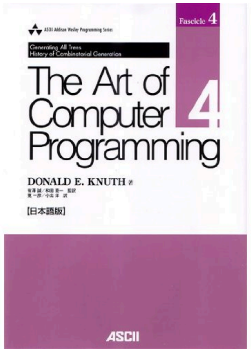
# 自習テキスト　４

ドナルド・E. クヌース（Donald E. Knuth）の歴史的な名著『The Art of Computer Programming 』の日本語版．アスキー．



# 確認

Processingを学んでいない人？

C/C++を学んでいない人？

ハードウェアを上手に使うための

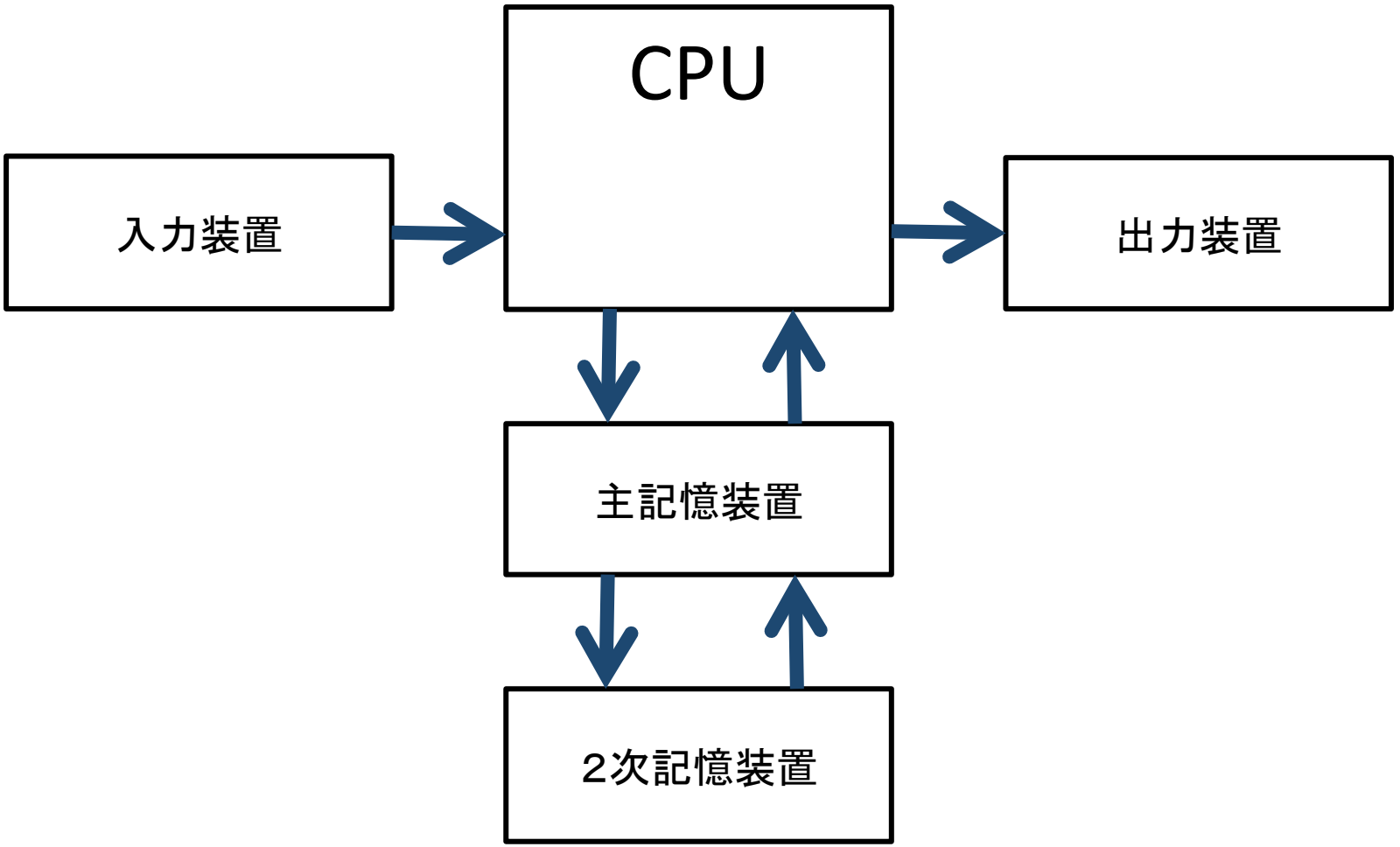
# ソフトウェア

# うまいやり方

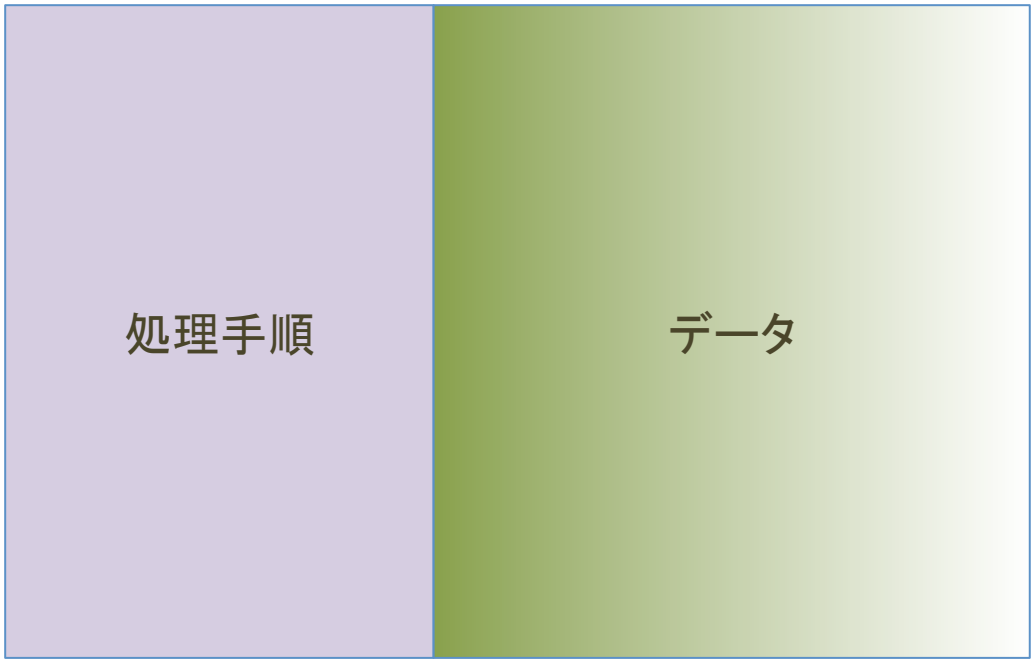
- 「この書類100部コピーしてください」
- 「それから，原稿に1カ所間違いがあるから直しておいてください」

どうします？

# プログラムを実行する



# プログラムの内部構造



# プログラムの状態

# ソフトウェアの特性

一つに機能に焦点を絞って考えると

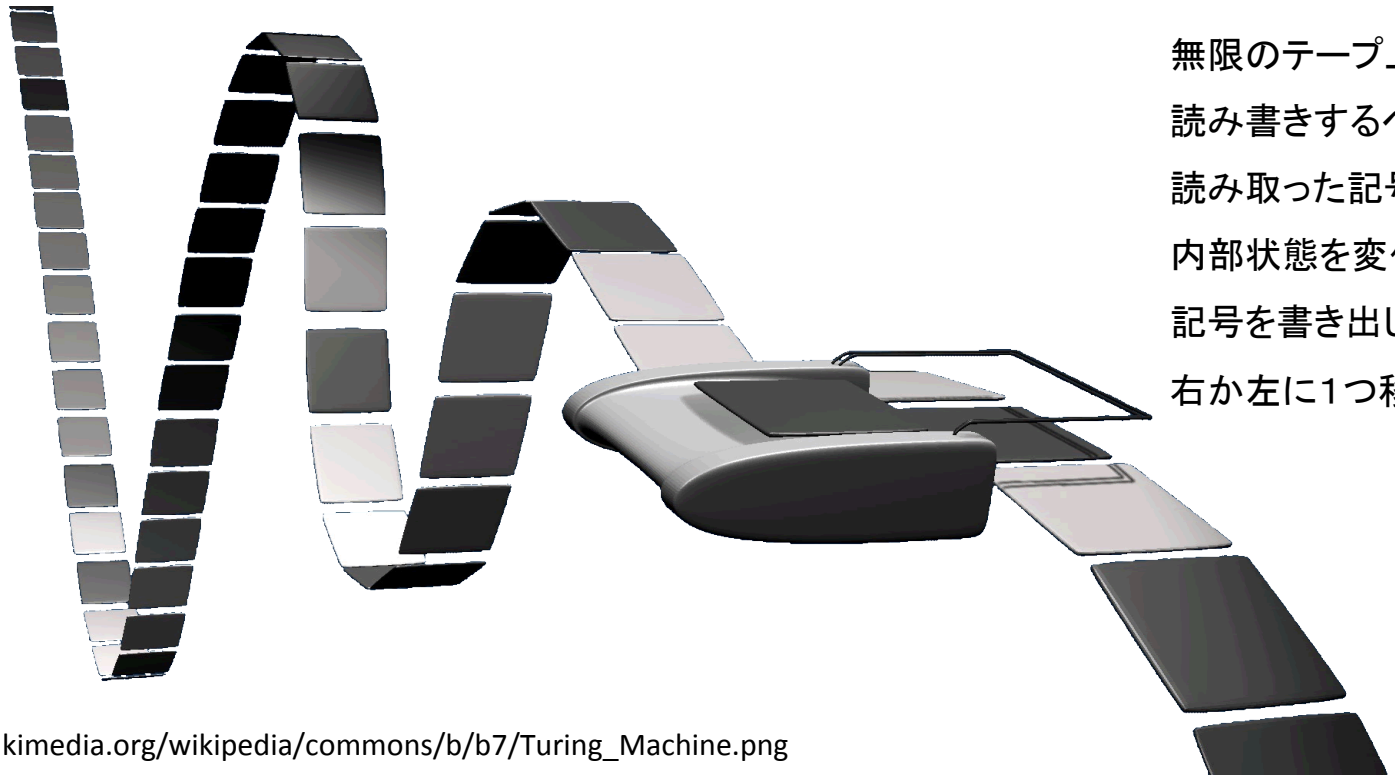
- 仕事をさせると，それなりに時間がかかる
- 仕事の量が増えれば増えるだけ時間がかかる
- 仕事の量をどう記述するか？
- かかる時間をどう記述するか？

# 抽象化

- アルゴリズム
  - プログラムの中の計算手順
- チューリングマシン
  - ハードウェアの抽象化

# チューリング・マシン

- ハードウェアのモデル
  - 現在のコンピュータにできることは全てできる



無限のテープ上の記号を
読み書きするヘッド
読み取った記号に応じて
内部状態を変化させ
記号を書き出し
右か左に1つ移動

http://upload.wikimedia.org/wikipedia/commons/b/b7/Turing_Machine.png

# アルゴリズムの評価

- アルゴリズムの考察対象
  - 問題の規模
  - 必要なステップ数
- 問題の規模
  - データの個数
- 必要なステップ数
  - 本来はチューリングマシンの動作回数
  - 演算回数または実行する命令数

# 実行時間とステップ数

- CPUはクロックによって動作が進む
- 命令を実行にはほぼ一定の時間がかかる
- アルゴリズムの性能評価
- 時間計測が基本

# 処理に要する時間の評価

- 演算の回数に応じた処理時間がかかる
- 演算の回数が処理に要する時間の目安
- アルゴリズムの観点
  - 処理時間そのものより
  - 処理の規模 n の増加による影響に着目

# 時間とスペース

メモリ量と処理時間の関係は？

一般には両者はトレードオフの関係

# プログラムの実行に要する時間

- 原理
  - 処理開始時の時刻を知る
  - 処理終了時の時刻を知る
  - 終了時刻から開始時刻の差をとる

# プログラムの実行環境

- OSがプログラムをロード
  - ２次記憶装置から主記憶装置上へ展開
- 通常は同時に他のプログラムも起動中
  - 複数の処理を同時並行に実行
    - マルチコア
    - 時分割処理

# n の増加の影響

- 漸近的な振る舞い
  - 計算１： $n^2/2 + 3n/2$
  - 計算２: $2 ( n + 1 )$
- O記法
  - 計算１の処理時間 $f(n^2/2 + 3n/2) \rightarrow O(n^2)$
  - 計算２の処理時間 $f(2 ( n + 1 ) ) \rightarrow O(n)$
  - Oの定義
  - $O(g(n)) = \{ f(n): c,n_0 >0, n>n_0, 0 \leq f(n)<cg(n)\}$

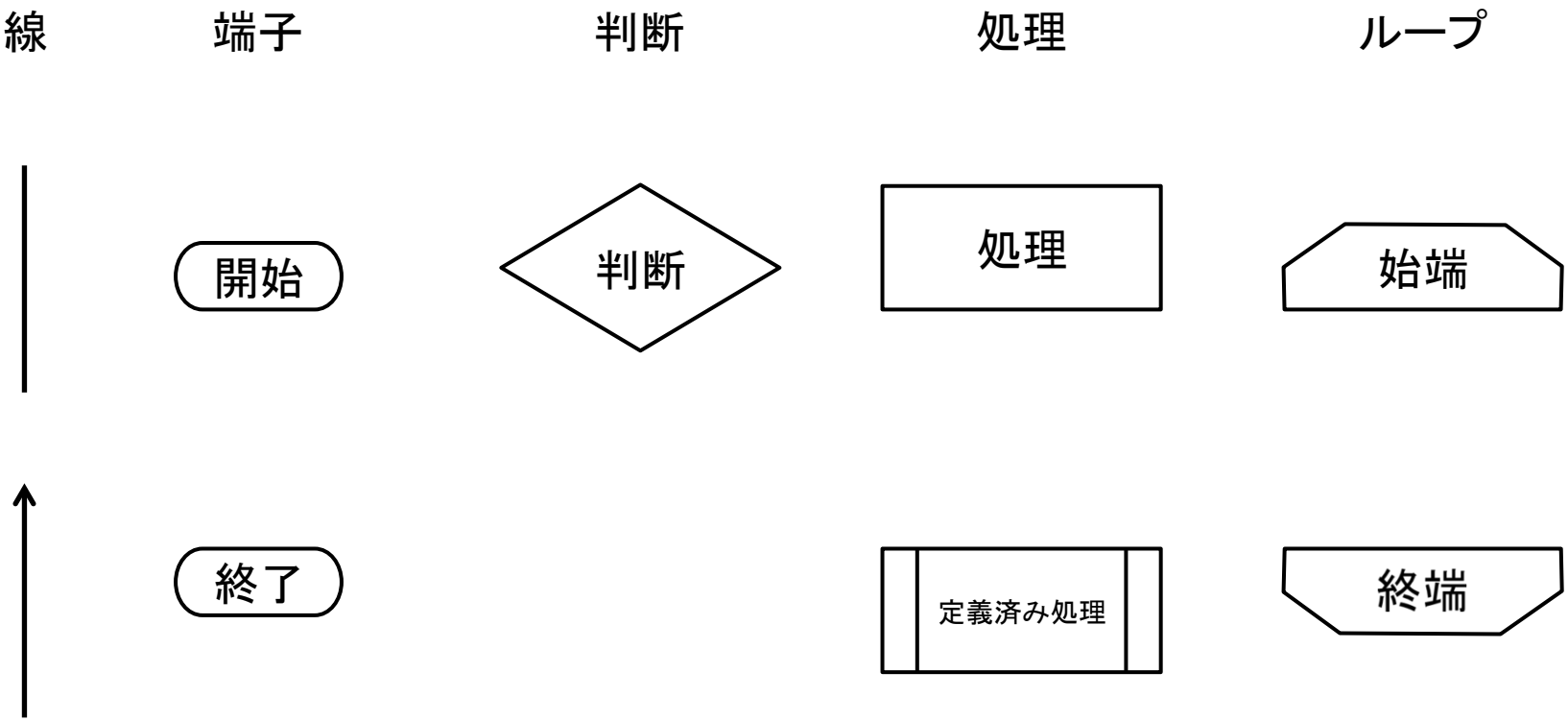
# O記法

- 漸近的な上界を与える
- Ω記法
  - 漸近的な下界
- Θ記法
  - 漸近的に上界と下界に挟まれる
- その他
  - o記法
  - ω記法

# アルゴリズムの図示

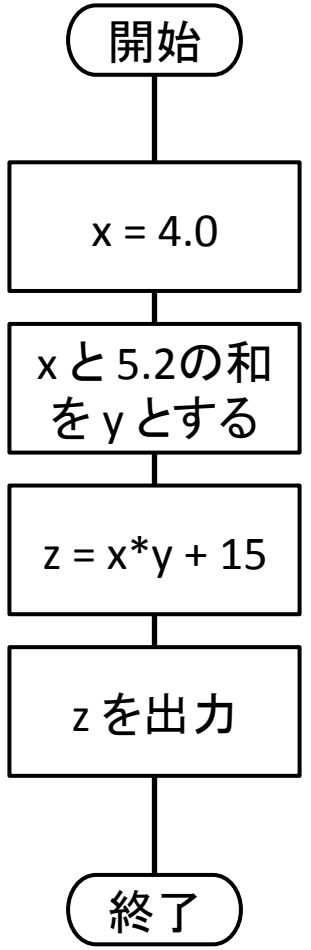
- 様々な方法が使われている
  - フローチャート
    - 古くからある．JIS規格もある．
    - この言葉が指すものは多様．
  - Problem Analysis Diagram (PAD)
    - フローチャートより良いという意見もある．
    - 日立で開発．今はあまり使われていない．
  - アクティビティ図（UML)
    - オブジェクト指向言語向け．UMLの一部．

# フローチャートの記号



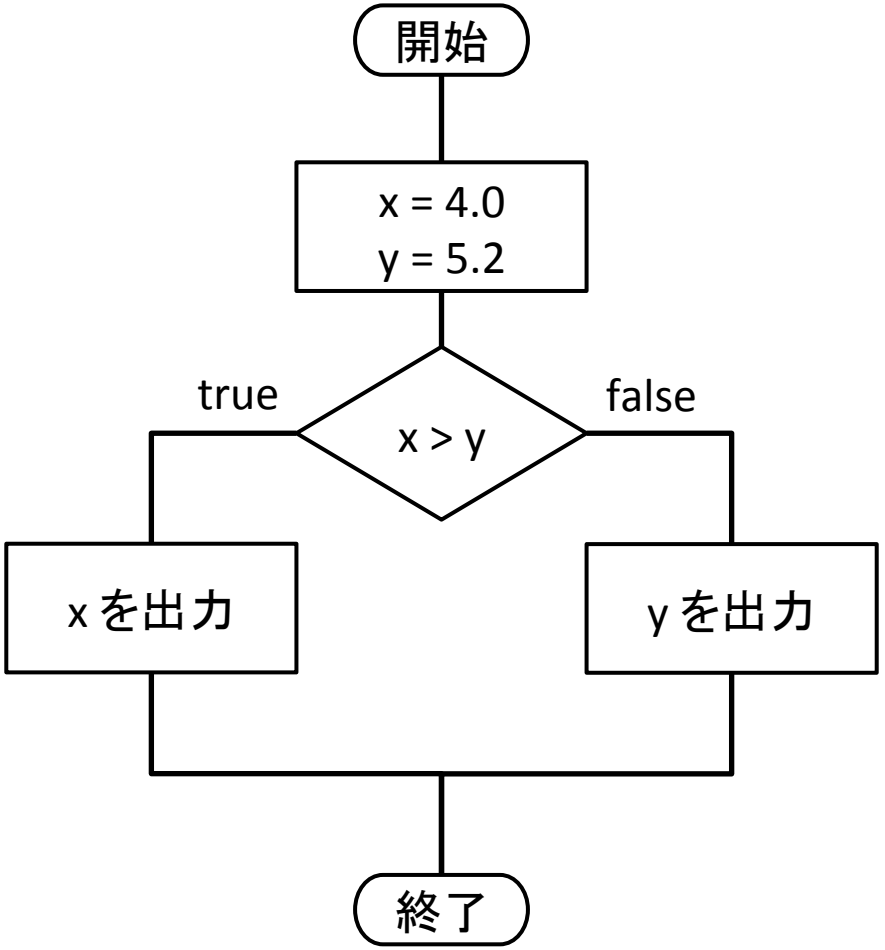
# フローチャートの例

```
float x = 4.0;
float y = x + 5.2;
float z = x*y + 15.0;
println(z);
```



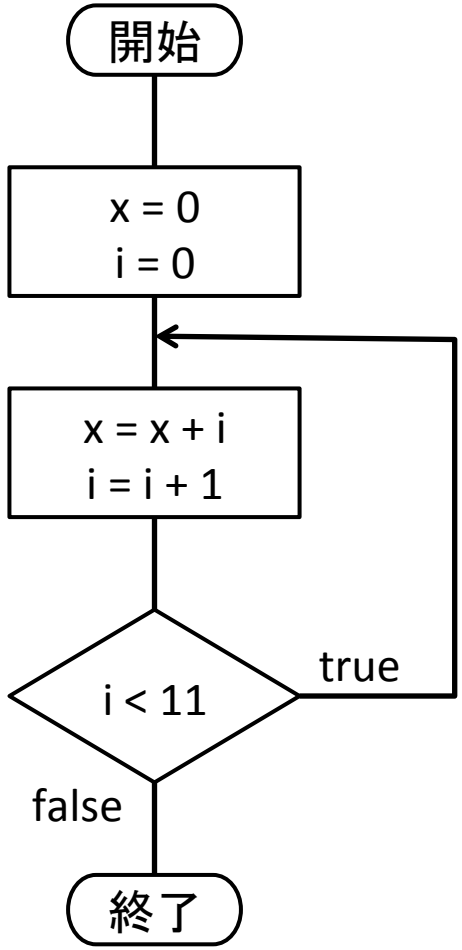
# フローチャートの例2

```
float x = 4.0;
float y = 5.2;
if (x > y) ;
println(z);
```



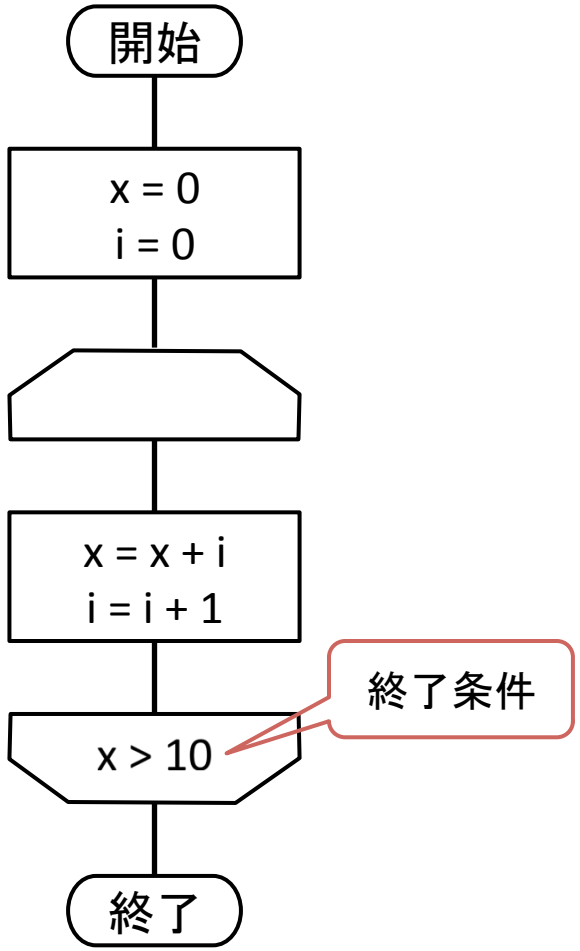
# フローチャートの例3

```
int x = 0;
for (int i = 1; i < 11; i++) {
  x = x + i;
}
println(x);
```



# ループ（繰り返し）の別表現

```
int x = 0;
for (int i = 1; i < 11; i++) {
  x = x + i;
}
println(x);
```



# フローチャート作成の注意点

- 開始端子から始まり，終了端子で終わる
  - 複数箇所で終了するときは，終了端子まで線を延ばす
- 線は，垂直か水平に延ばす
  - 斜めの線や曲線は使わない
- 線は分岐させない
  - 並列動作の記号がある
- 複数の線を合流させても良い
  - 合流点から出ていく線は一つ
  - 流れを明示するために矢印を付けても良い

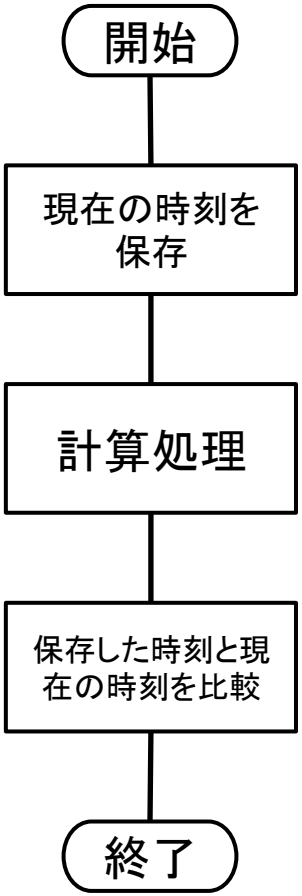
# 参考

- ざっくりわかる！プログラミングのためのフローチャートの書き方
  - http://eng-entrance.com/programming_flowchart
- フローチャート（流れ図）
  - http://wwwpat.eng.u-toyama.ac.jp/flowchart/fc_sct.html
- フローチャートの書き方
  - http://www.ced.is.utsunomiya-u.ac.jp/lecture/2005/prog/common/flow_guide.pdf
- (参照 2017-04-09)

# やってみよう

- フローチャートを描いてください
  - N個の乱数の和を求める手順
  - 上記の手順をQ回繰り返す手順
  - 処理の最初から最後までに掛る時間tを計測する手順を追加
  - t/Qを求めNまでの和を求めるのに必要な時間の平均値を計算。Nの変化でどのように変わるか？

# ヒント



# 宿題：ex01

- フローチャートに描いたプログラムを Processing で作成してください。
  - N個の乱数の和を求め，その手順をQ回繰り返す
  - この処理の最初から最後までに掛る時間tを計測する手順を追加
  - t/Qを求めN個の乱数の和を求めるのに必要な計算の平均時間を求める。
- 試すこと
  - NやQにどれくらいの値を与えると計測可能になるでしょうか？
  - Nの変化で平均時間はどのように変わるでしょうか？
  - 乱数の代わりに定数を用いたらどのように変わりますか？
  - 結果はコメントで

# コーディング

- コメントとして:
  - プログラムの冒頭にMS, NDの別，クラス，番号，氏名を記述
  - 試した結果，考察，感想など
    - 量的にはA4レポート用紙 ½ ページ以内
    - 個人での努力が読み取れるように！
  - （人的資源を含む）参考にした資料等があれば出典を書いておく
- プログラム本体は上記コメントの後
  - プログラムは実行可能なこと

# 提出についての注意

- Processing のプログラム名
  - デフォルトでは sketch_yymmdda などだが…
  - higuchi_fumito_ex01 のように氏名と宿題番号に変えること
- Processingのプログラムはフォルダごと提出
  - 必要ならzipファイルとしてまとめてください
- Oh-o!Meijiから提出（次回の授業開始までに）

# python によるサンプルコード

```
import time
import random

Q = 200
N = 10000
sum = 0
t_sum = 0

t_start = time.time()
for ii in range(Q):
    sum = 0
    for j ii in range(N):
        sum += random.random()

t_sum = time.time() – t_start
print(1000*t_sum/Q, “ ms for “, N, “ additions”)
```

# ヒント

- Processing では変数に型があります
  - 場合によっては型の変換が必要です
- 時間に関係する関数は関数によって仕様が微妙に違います
  - よく Reference Manual を読みましょう
- python では，for 文の範囲は{}ではなく，インデントで表現しています
  - range(Q) はOからQ-1までの数値のリストを作ります