情報学基礎 第7回

4章　ソフトウェア

管理工学科
担当：篠沢 佳久

本日の内容

- 情報処理の基礎概念2:ソフトウェア
  - ソフトウェアの役割(4.1節)
  - アルゴリズムとプログラム(4.2節)
  - プログラミング言語(4.3節)
  - ソフトウェアの階層(4.4節)

- 第三回レポート課題

教科書を読んでおいて下さい

- 4.3節
  - (4)ライブラリとソフトウェア・フレームワーク
  - (5)ソフトウェア開発環境
  - (6)プログラミング言語の歴史

- 4.5節
  - アプリケーション・ソフトウェアの種類

ソフトウェアの役割

ソフトウェアの役割①（4.1節）

- ハードウェア
  - 電子機器としてのコンピュータ
  - 機能の追加・変更は簡単ではない
    - ハードウェアに設計ミスがあるとリコールになって大騒ぎ

- ソフトウェア
  - ハードウェアに「何をしてもらうのか」を具体的に指示するもの
  - 1 台のコンピュータでも，ソフトウェアさえ入れればいろいろなことができる
  - 機能の追加・変更がやりやすい
    - ソフトウェアを入れ替えればよい

ソフトウェアの役割②

- （例）iPhoneはハードウェア
- 　　iPhoneのアプリはソフトウェア
  - アプリさえ入れれば一台のiPhoneでいろいろなことが可能

コンピュータにできること

- 大ざっぱにいえば，次の2つのことしかできない
  - 2 進数で表された大量のデータを間違いなく記憶すること
  - 簡単な処理をものすごいスピードで間違いなく実行すること
    - 足し算，引き算，比較，分岐など
    - デジタル回路の組み合わせに過ぎない

- でも，複雑な処理をするように見えるんですけど？
  - 指示された通り，簡単な処理を繰り返しているだけ
  - コンピュータが自分で何かを考えることはない
  - コンピュータが気を利かせてくれることはない

  <u>間違った指示を出しても，間違った指示に従って忠実に処理</u>

アルゴリズムとプログラム

アルゴリズムとは？（4.2節）

- プログラムとは？
- コンピュータに対する指示を書き下したもの
  足し算，引き算，比較，分岐などの単純な処理を
  – どのような順番で
  – どのように行うのか
  をいちいち書き下したもの

- ソフトウェアとの違いは？
  – ソフトウェア ≒ プログラム と思っていてよい
  – 正確には，ソフトウェア = プログラム + データ etc.

- ソフトウェアを作ること ≒ プログラムを書くこと

プログラムのようなものを考えてみよう

- 例題:
  正の整数 n が素数かどうか判定するプログラム

- 考え方:
  - n を 2, 3, 4, ･･･, n – 1 まで順に割ってみる
  - もし，どれかで割り切れたら n は素数ではない
  - もし，2 から n – 1 のどの数でも割り切れなかったら素数

プログラムっぽく書くと・・・

- 与えられた正の整数 n が素数かどうか調べる
  1. 補助の変数 i を用意する
  2. 変数 i に 2 を代入する
  3. i が n 以上だったら「素数」と表示して終了
  4. n を i で割る
  5. 割り切れたら「素数ではない」と表示して終了
  6. 割り切れなかったら，i の値をひとつ増やす
  7. 3 に戻って繰り返し

コンピュータになったつもりでやってみる

- 自然数 5 が素数かどうか判定する
  すなわち, n の値は 5
  - 2. 変数 i に 2 を代入する
  - 3. i は 5 以上かどうか調べる
    - 2 は 5 より小さい
  - 4. 5 を i で割る
    - 5 / 2 は割り切れない
  - 6. i の値をひとつ増やす
    - i の値は 3 になる
  - 3. i は 5 以上かどうか調べる
    - 3 は 5 より小さい
  - . . .

アルゴリズムとプログラム①

- コンピュータに解かせたい問題

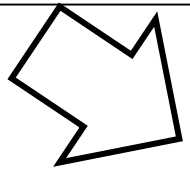
- 具体的な解法の手順（アルゴリズム）
  - その問題をコンピュータで解くために，すべての手順を明確にしたもの

- コンピュータが理解できる解法の手順（プログラム）
  - コンピュータが理解できる人工言語（プログラミング言語）で記述

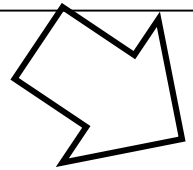
アルゴリズムとプログラム②

- アルゴリズムとプログラムは似ているがちょっと違う

問題：コンピュータに解かせたいこと
（例：素数かどうかを判定する）



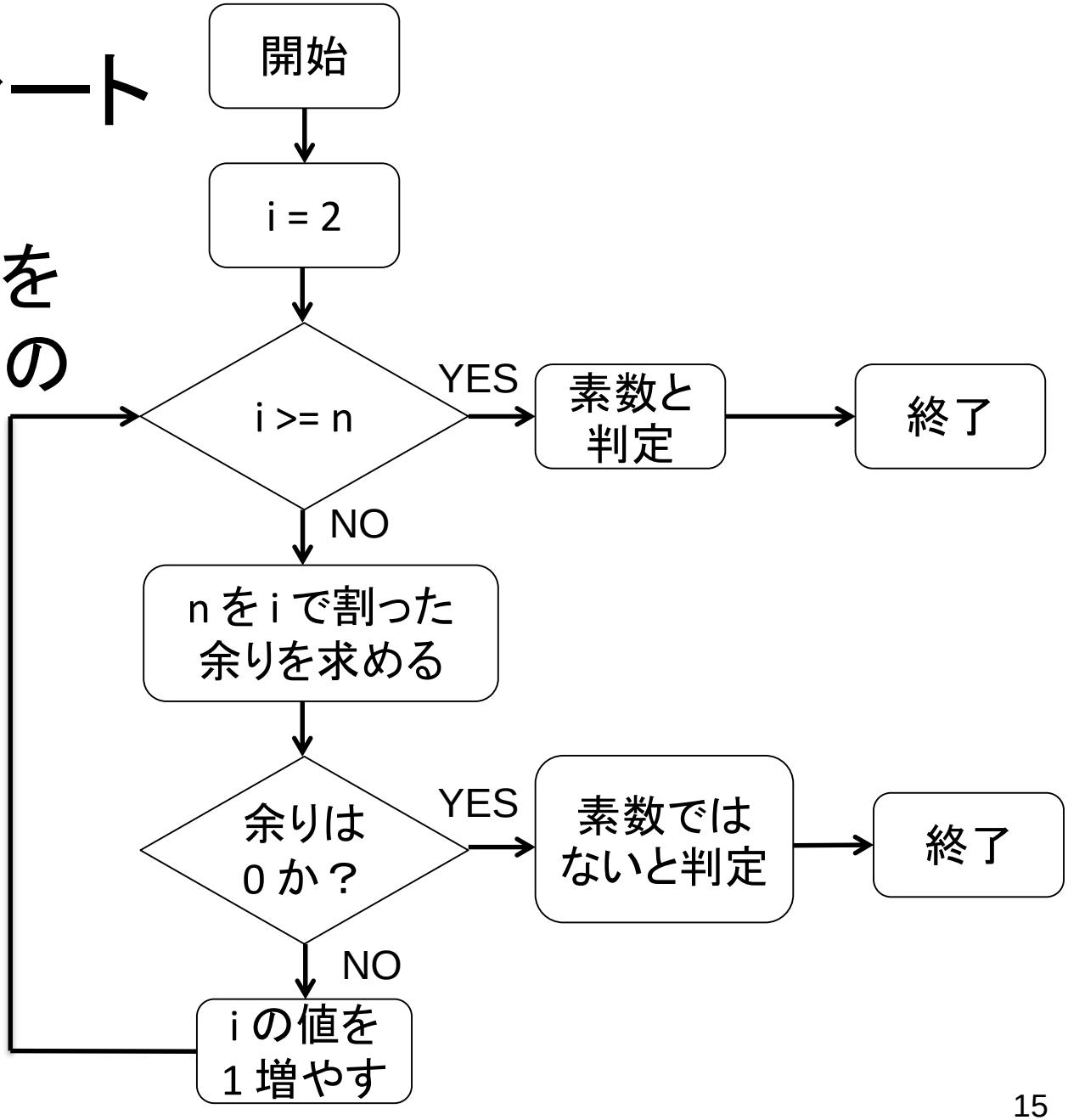
アルゴリズム：具体的な手順
（例：素数かどうか判定する方法を考える）



プログラム：アルゴリズムをプログラムとして記述したもの
（例：実際にプログラムを書く）

フローチャート

- プログラムを図示したもの



アルゴリズムの実行効率

- 同じ問題でもいろいろなアルゴリズムがある
  - 素数判定ひとつとっても何十種類もある

- アルゴリズムによって解くのにかかる時間が違う
  - 効率のよい手順であれば，短い時間で解きおわる
  - 効率の悪い手順であれば，解くのに長い時間がかかる
  - アルゴリズムの違いによって，数百万倍，あるいはもっと処理時間が違うこともある

計算量(オーダー)

- アルゴリズムの実行効率を数学的に定義したもの

- あるアルゴリズムが与えられたとき,
  - もっとも時間のかかる演算に注目する
  - その演算が何回行われるのかを見積もる
  - 演算回数が少なければ少ないほど効率のよいアルゴリズム

- 例:
  - 正数 n が素数かどうか調べるのに n 回程度の割り算が必要
  - これを O(n) と書く

計算量の例①

自然数 n が偶数か奇数か調べたい

n を2で割ればいいので，この作業は n の大きさに関係なく，一定時間で済む

⇒ この処理の計算量は O(1) である

計算量の例②

自然数 n が素数であるか調べたい

[手法1] 2 から n-1 までの数で n を割ってどれかで割り切れるかどうか調べる

[手法2] 2 から$\sqrt{n}$ までの整数で n を割ってどれかで割り切れるかどうか調べる

割り算の回数に着目すると,

[手法1]の計算量は O(n)

[手法2]の計算量は O($\sqrt{n}$)

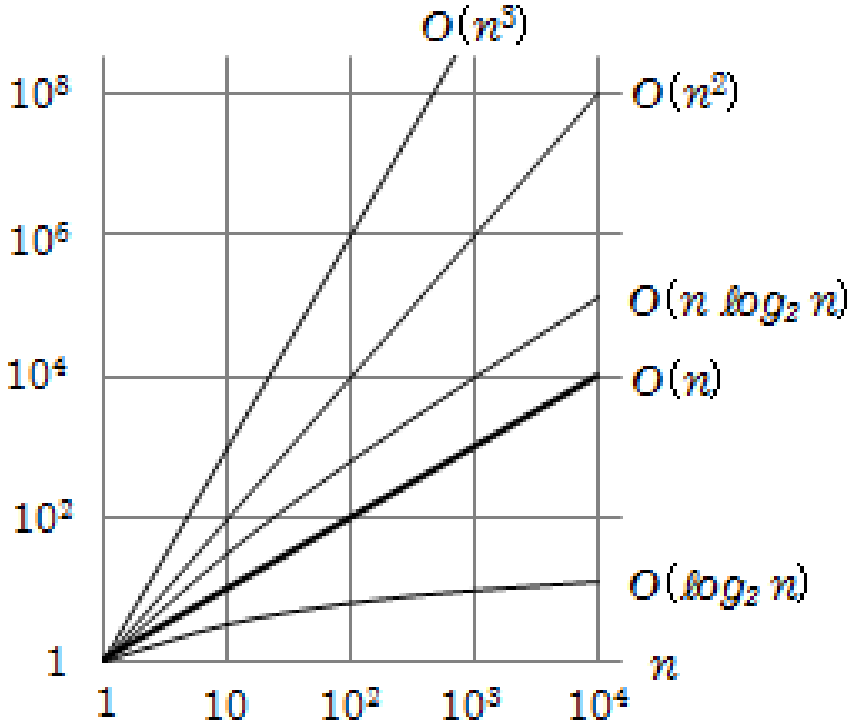
計算量の例③

N × N の行列があるとする．これが単位行列であることを，すべての要素を1つ1つ調べることで確かめる

要素の数は N^2 個あるので，この作業の計算量は $O(N^2)$ である

さまざまな計算量

- 実用上，よく出てくるのは
  O(log n), O(n), O(n log n), O(n^2) など
  - log の底は暗黙に 2．計算量を議論するときは無視してよい



- n の値が大きいと，
  O(log n) と O(n^2) では
  効率が大きく違う

- アルゴリズムによる効率の違い：
  - 「基本的な考え方」のソート：O(n^2)
  - 効率のよいソート：O(n log n)
  - 100万個のデータをソートしたら，
    効率は何倍違う？

プログラミング言語

プログラミング言語（4.3節）

- コンピュータに解かせたい問題

- 具体的な解法の手順（アルゴリズム）
  - 効率の良いアルゴリズムを用いる

- コンピュータが理解できる解法の手順（プログラム）
  - コンピュータが理解できる人工言語（プログラミング言語）で記述

プログラミング言語

- アルゴリズムをコンピュータにわかる形で記述するための（人工）言語
  - 自然言語と違い，意味が正確に定義されている

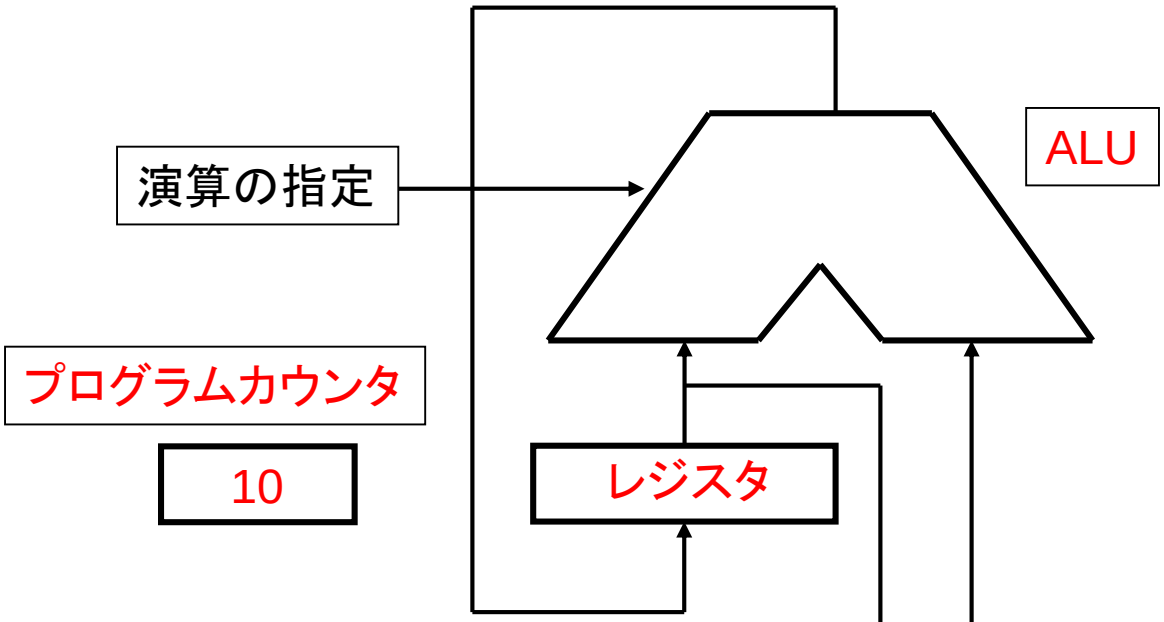
- 素数判定の例でもアルゴリズムを正確に記述するのは大変
- まともなソフトでは膨大な量の指示を書き下す必要がある
  - 人間の理解力をはるかに超えてしまう
  - ソフトウェア研究は人間の認知能力とプログラムの複雑さとの戦い
- プログラミング言語により，ソフトウェアの複雑さを覆い隠す
  - コンピュータサイエンスの重要な使命のひとつ

（1）機械語とアセンブリ言語

- プログラミング言語以前の原始段階

- 機械語（マシン語）
  - コンピュータが直接理解できる形式
    - コンピュータが理解できるのは2進数のみ
  - プログラムも2進数の羅列として記述する
    - 人間が理解するのはほとんど不可能

機械語（マシン語）



人にとっては意味不明

番地	命令，データ
10	C=A+B
20	E=C-D
30	A=10
40	B=20

命令は二進数

1011 0111 1001 1100
0111 1111 0110 0011
1001 1001 1011 0101

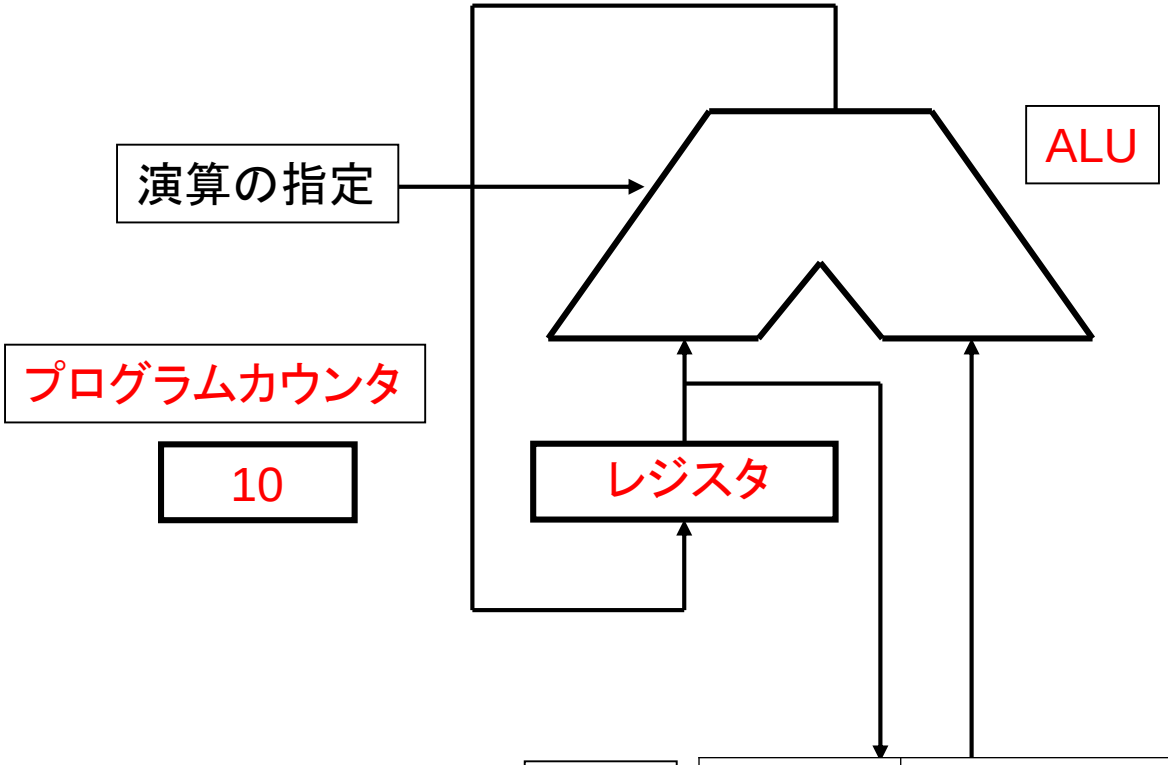
アセンブリ言語①

- アセンブリ言語
  - 機械語と比較して，人間にとって分かりやすくした言語
  - 機械語と一対一に対応した言語

- ADD R1, R2, R3
  - レジスタ2とレジスタ3を加算し，レジスタ1に格納
  - ADDをオプコード
  - 操作対象（R1，R2，R3）をオペランド

アセンブリ言語②

プログラマー



アセンブリ言語

```
ADD  R1, R2, R3
ADD  R1, R3, R4
BEQZ  1000
```

アセンブラにより変換

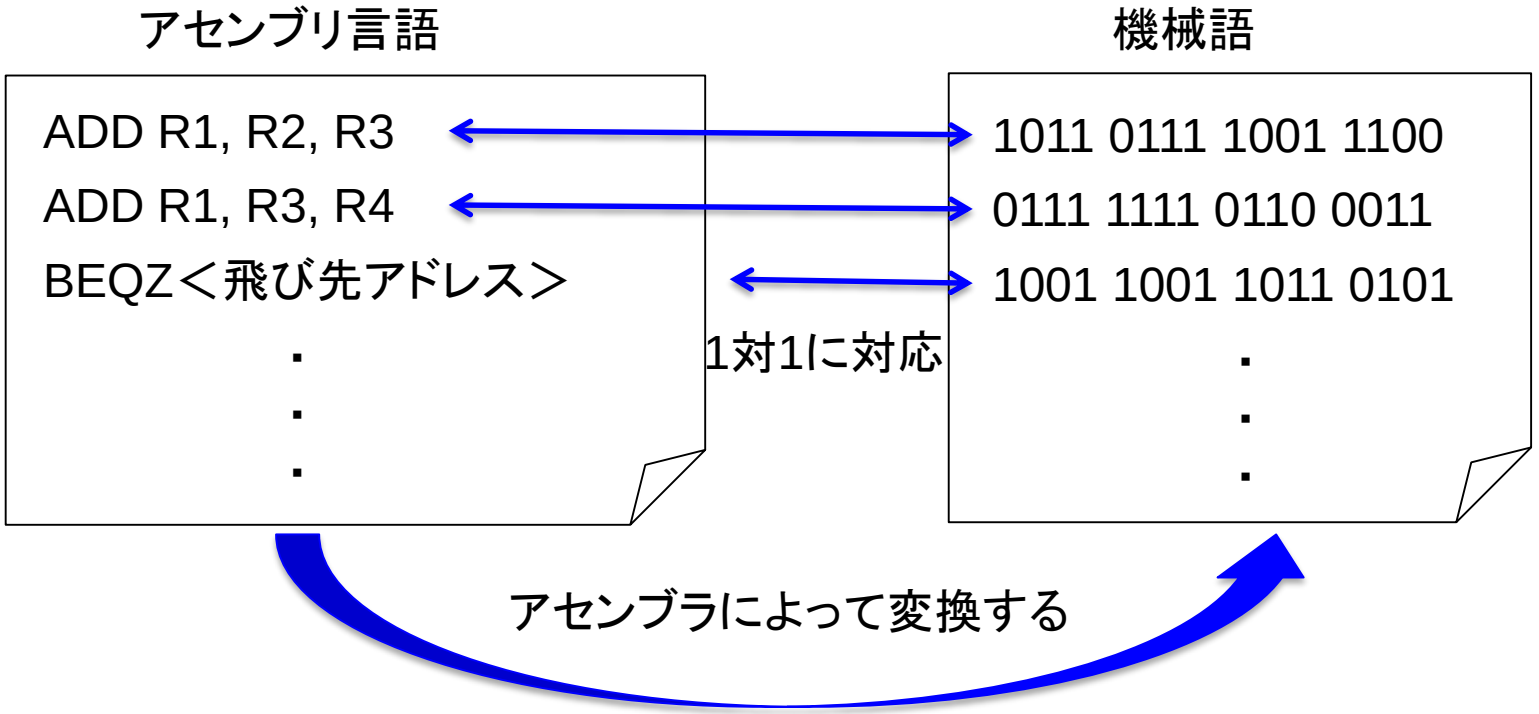
番地	命令，データ
10	C=A+B
20	E=C-D
30	A=10
40	B=20

マシン語

```
1011 0111 1001 1100
0111 1111  0110 0011
1001 1001 1011 0101
```

アセンブリ言語と機械語の関係

- アセンブリ言語と機械語はほぼ一対一に対応する
- 機械語に変換してから実行する
- 機械語に変換するソフトウェアのことをアセンブラという



低水準言語と高水準言語

- 低水準(低級)言語
  - アセンブリ言語，機械語
  - ハードウェアに近いことを低水準と呼ぶ
  - ハードウェアの知識が必要

- 高水準(高級)言語
  - ハードウェアから遠いことを高水準と呼ぶ

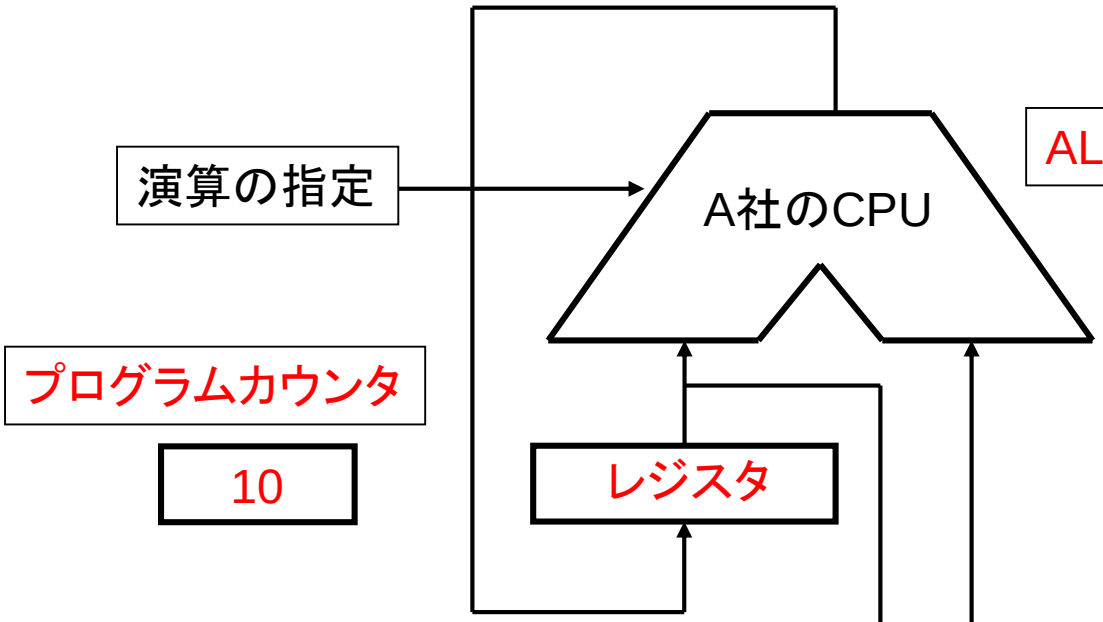
機械語とアセンブリ言語

- 機械語
  - CPUが異なれば，同じ動作をさせようとした場合においても，命令が異なる→アセンブリ言語も異なる

- （例）c = a + b
- A社のアセンブリ言語の場合
  - R2にa，R3にbの値が入っている場合
  - ADD R1, R2, R3（R2とR3を足した結果をR1に入れる）

- B社のアセンブリ言語の場合
  - Raにa，Rbにbの値が入っている場合
  - MOV Rc, Ra（Raの値をRcにコピー）
  - ADD Rc, Rb（RcとRbを足した結果をRcに入れる）

A社のアセンブリ言語



A社のアセンブリ言語

```
ADD  R1, R2, R3
ADD  R1, R3, R4
BEQZ  1000
```

アセンブラにより変換

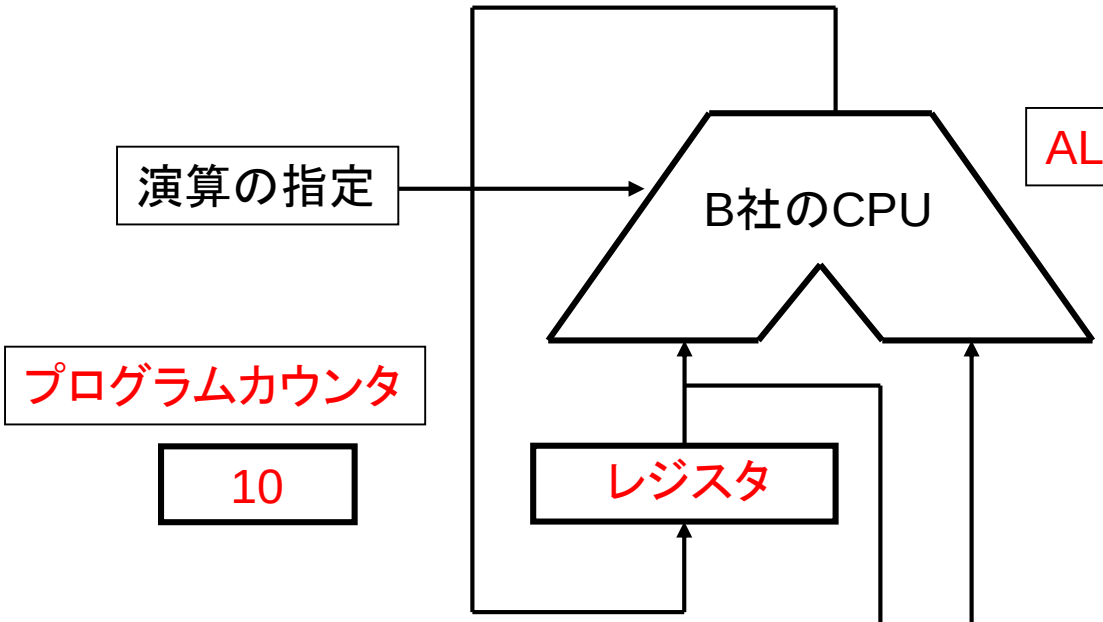
メモリ

番地	命令，データ
10	C=A+B
20	E=C-D
30	A=10
40	B=20

A社のマシン語

```
1011 0111 1001 1100
0111 1111  0110 0011
1001 1001 1011 0101
```

B社のアセンブリ言語



B社のアセンブリ言語

```
MOV  Rc, Ra
ADD  Rc, Rb
BEQ  Ra  1100
```

アセンブラにより変換

メモリ

番地	命令，データ
10	C=A+B
20	E=C-D
30	A=10
40	B=20

B社のマシン語

```
0111 0001 0110 1001
0111 0001 1001 1110
1010 1011 1100 1100
```

アセンブリ言語の問題点

- 移植
  - A社用のCPUのために開発したプログラム
  - B社のCPUで利用できるようにするためには，プログラムを書き換えなければならない（移植，ポーティング）
  - 移植性が低い
  - 人にとって書きづらい
  - 大規模なプログラムを書くことが困難

(2)高水準言語

- 高水準(高級)言語
  - ハードウェアに依存したプログラムを書く必要がない
  - 大規模プログラムを書くための機能を備えている
  - 人にとって分かりやすい
  - プログラミング言語といえば高水準言語を指す
  - 高水準言語で書かれたプログラムのことをソースコードとも呼ぶ
- 例:
  - c = a + b;
    などと書けば, a と b を足した結果が c になる
- いろいろな高水準言語がある(図4.7)
  - Java, Lisp, FORTRAN など

高水準言語の役割

- 大規模プログラミングのための諸機能を備えている
  - こういう機能を抽象化という言い方をすることがある
    - プログラムの複雑さを隠蔽するための機能
    - プログラムの再利用性を高めるための機能
    - プログラミング上のケアレスミスを発見してくれる機能など

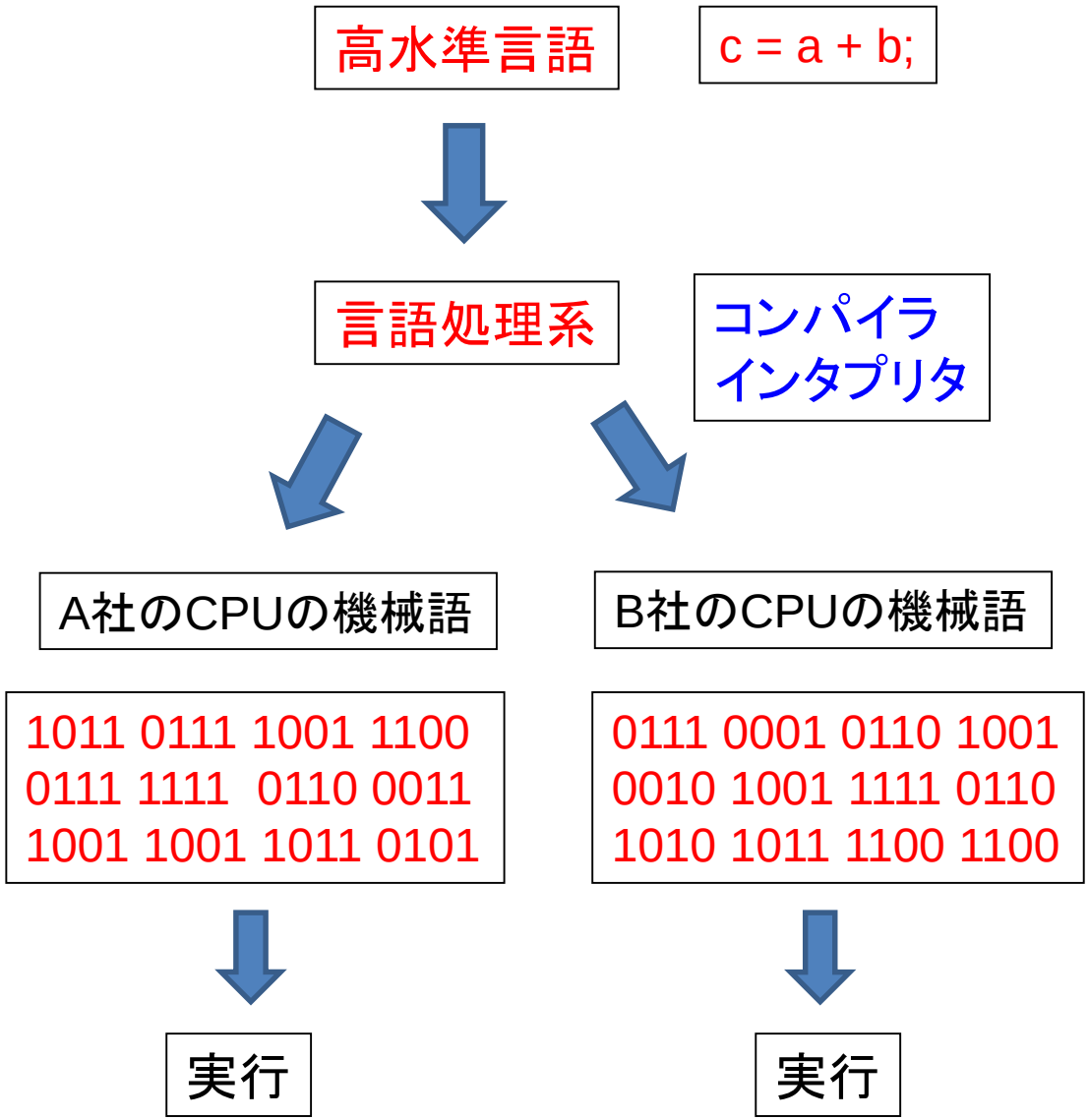
- 高水準言語を使いこなすためにいろいろと学ぼう
  - 最近のプログラミング言語は次のような機能や概念を持つ
    - 関数，手続き，メソッド，クラス，モジュール，型，型多相性，型推論
    - リフレクション，例外，スコープ，ガベージコレクションなどなど
  - 経験的に理解していくのが一番
    - プログラミング関係の演習科目だけではなく自分でもチャレンジしよう

プログラミング言語処理系①

- ただし，高水準言語のままではコンピュータは理解できない
  - 機械語に変換する必要がある

- プログラミング言語処理系
  - ソースコードを機械語に変換するソフトウェアのこと
  - ある言語を使うにはその言語用の処理系が必要となる

プログラミング言語処理系②

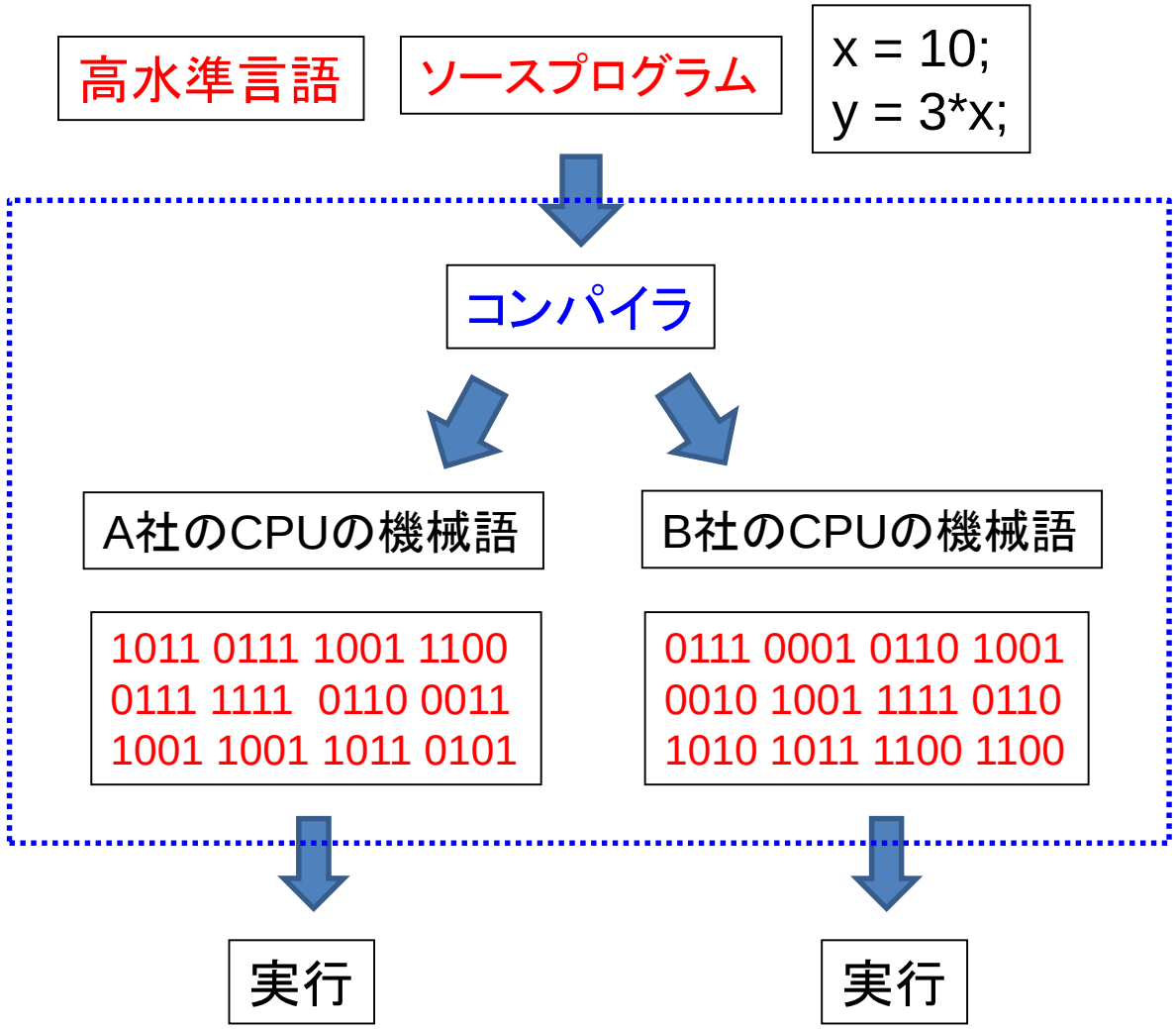


（3）インタプリタとコンパイラ

- プログラミング言語処理系
- コンパイラ
  - 高水準言語で書かれたソースコードを機械語に変換（コンパイル）するためのソフトウェア
- インタプリタ
  - 高水準言語で書かれたソースコードを逐次，機械語に変換しながら実行するソフトウェア

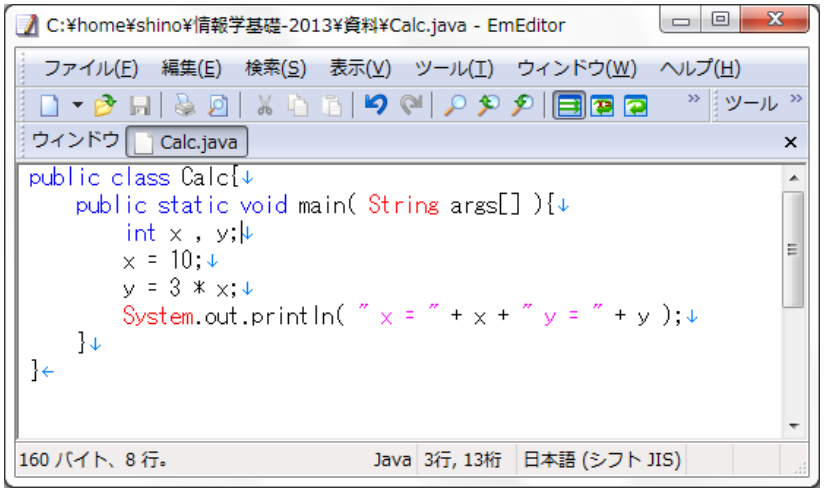
インタプリタとコンパイラ①

- コンパイラ



コンパイラ（Java言語）

Javaプログラム



コンパイル

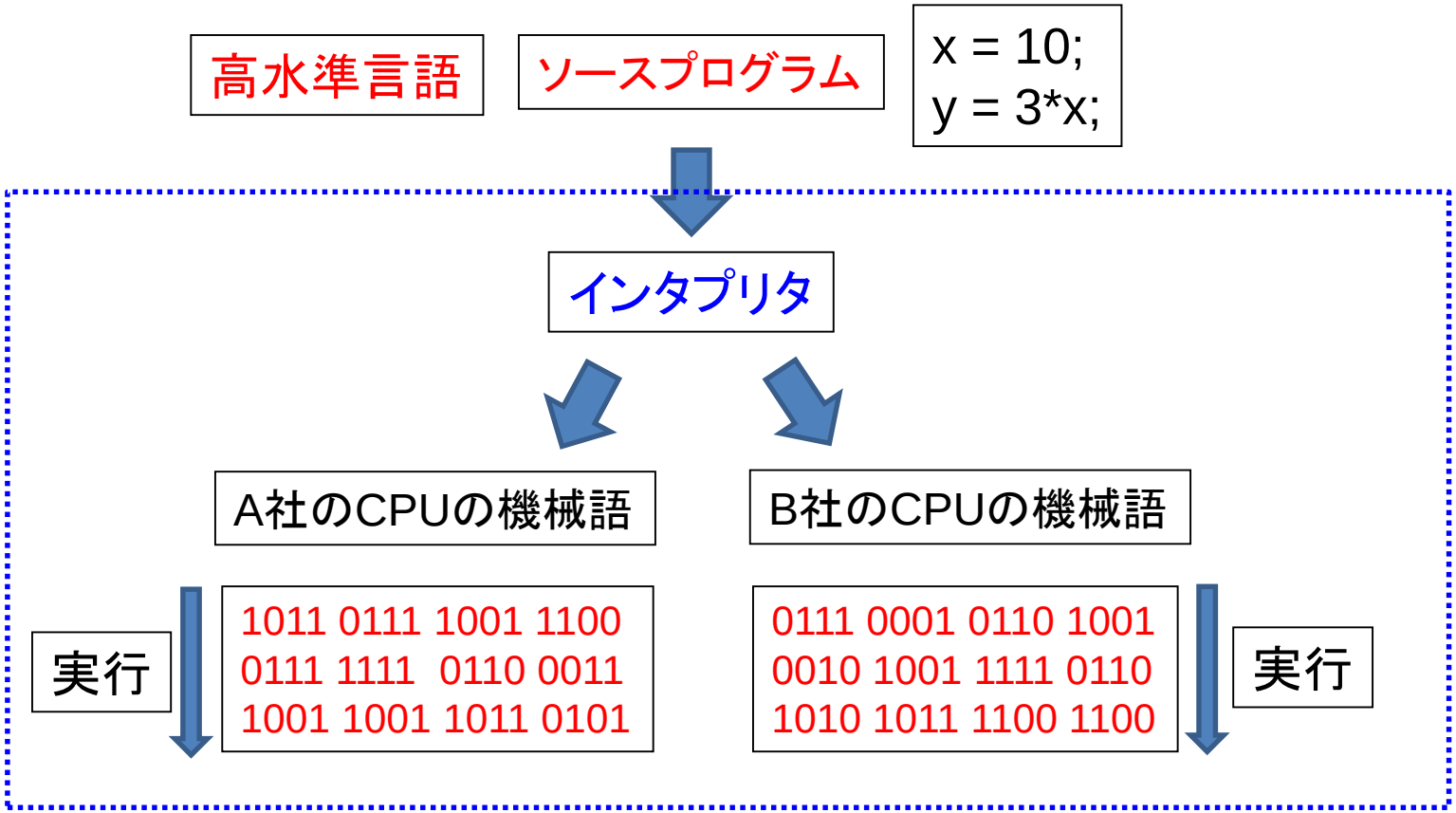
```
C:¥>javac Calc.java
```

実行

```
C:¥> java Calc
 x = 10 y = 30
```

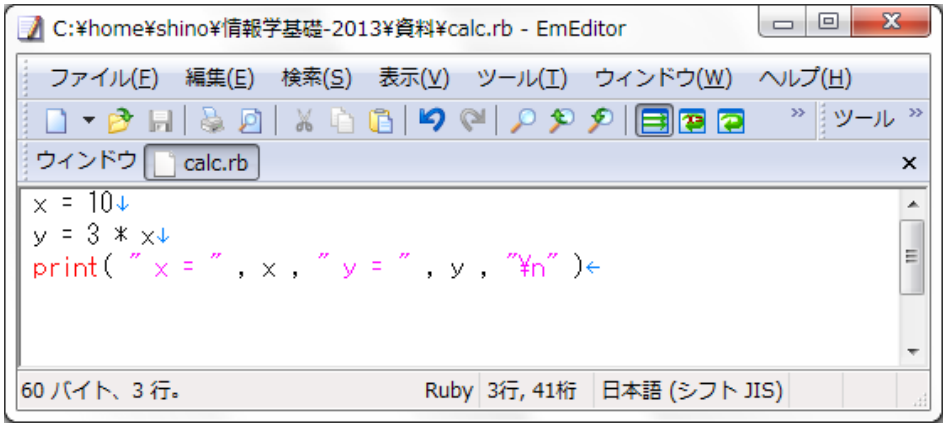
インタプリタとコンパイラ②

- インタプリタ



インタプリタ（Ruby言語）

Rubyプログラム



インタプリター（Ruby）

実行

```
C:>ruby calc.rb
 x = 10 y = 30
```

インタプリタとコンパイラ③

- コンパイラ
  - 事前にまとめて機械語に翻訳するため，高速に実行が可能
  - プログラムを修正した場合，コンパイルを再度行わなければならない
- インタプリタ
  - 逐次，機械語に翻訳→実行するため，実行速度が遅い
  - プログラムを修正した場合，コンパイルは不必要

(4)ライブラリとフレームワーク

- 高水準言語を使っても，大規模プログラミングは大変
- ゼロからプログラムを書かなくていいようになっている
  - 誰かが書いてくれたプログラムを再利用しながら書く
  - すでに用意された部品を変更しつつ，貼り合わる感じ

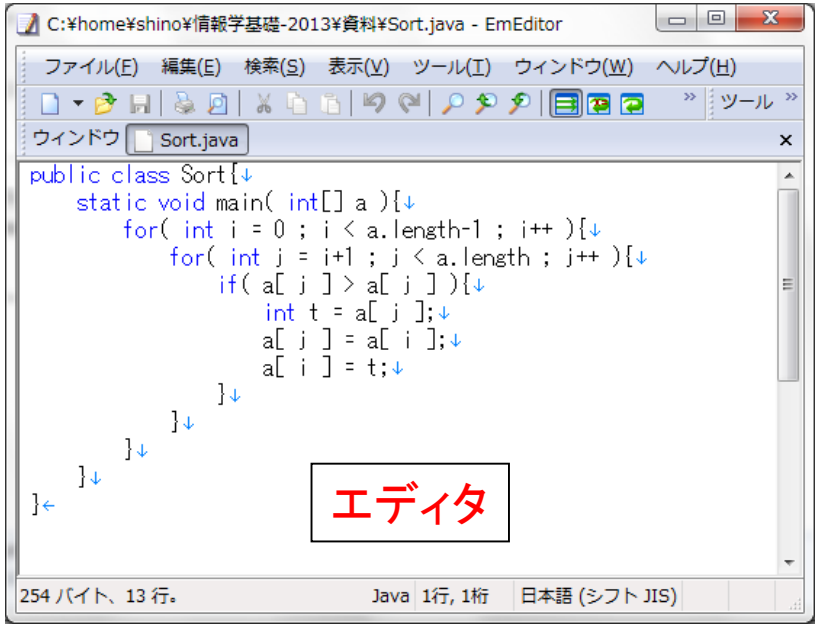
- ライブラリとソフトウェア・フレームワーク
  - よく使う機能を系統立ててまとめたもの
  - フレームワークそのものを作るのは大変
  - ソフトウェア科学の研究で得られた知見が詰まっている

(5)ソフトウェア開発環境①

- ソフトウェア開発を容易にするソフトウェアのこと
  - ソフトウェア開発にはいろんなステップがある
    1. プログラムをエディタを用いて書く
    2. 書いたプログラムをコンパイルする
    3. コンパイルしたプログラムを実行して，動作チェックをする
    4. 期待した通りに動くことは稀．プログラムの間違い（バグ）を探して修正
    5. もう一度，コンパイルして動作チェック．以下，繰り返し

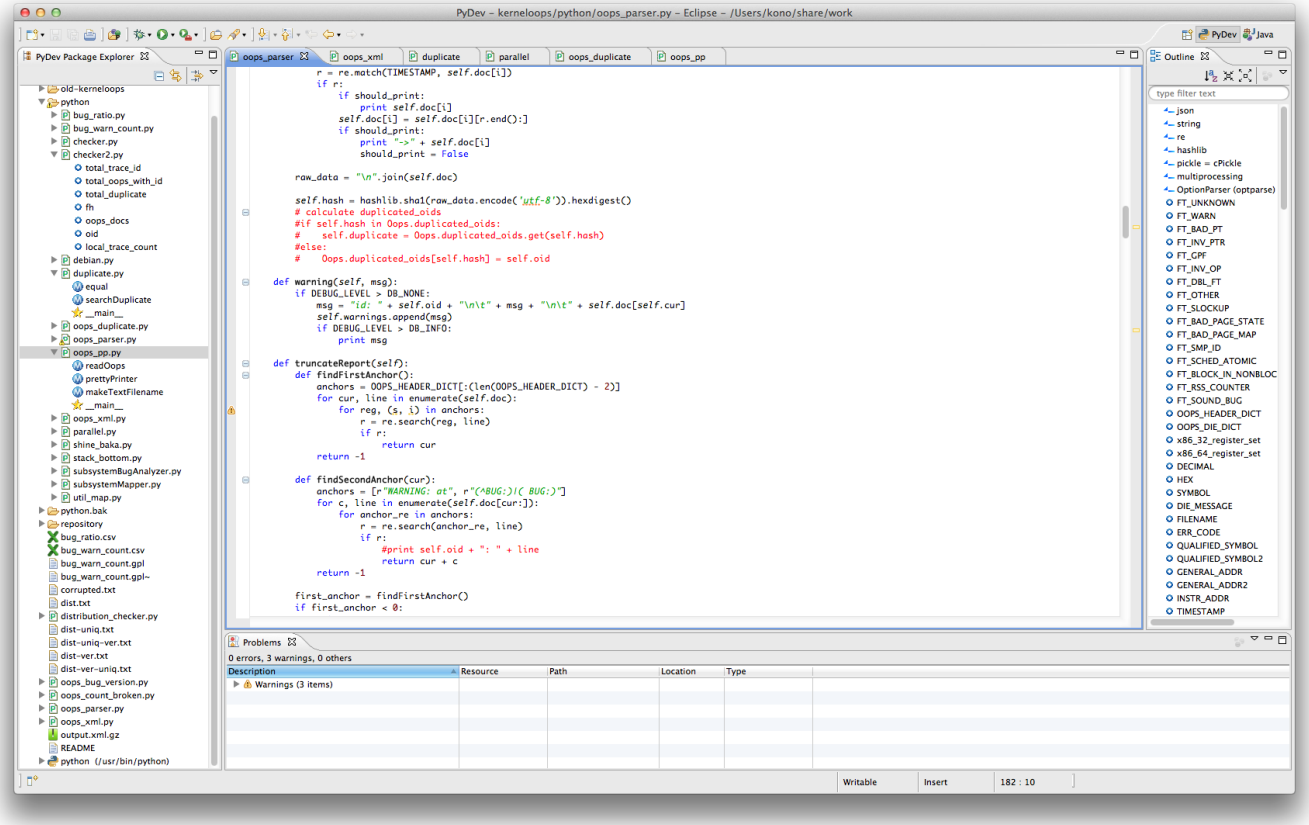
ソフトウェア開発環境②

- それぞれのステップをサポートするソフトウェアがある
  - エディタ: プログラムを書く作業を支援する
  - コンパイラ: プログラムをコンパイルする
  - デバッガ: プログラムの間違いを探すのを助けてくれる



統合開発環境

- ソフトウェア開発のための機能が統合されている
  - エディタ，コンパイラ，デバッガなどが統合されている
- 例：Eclipse



(6)プログラミング言語の歴史

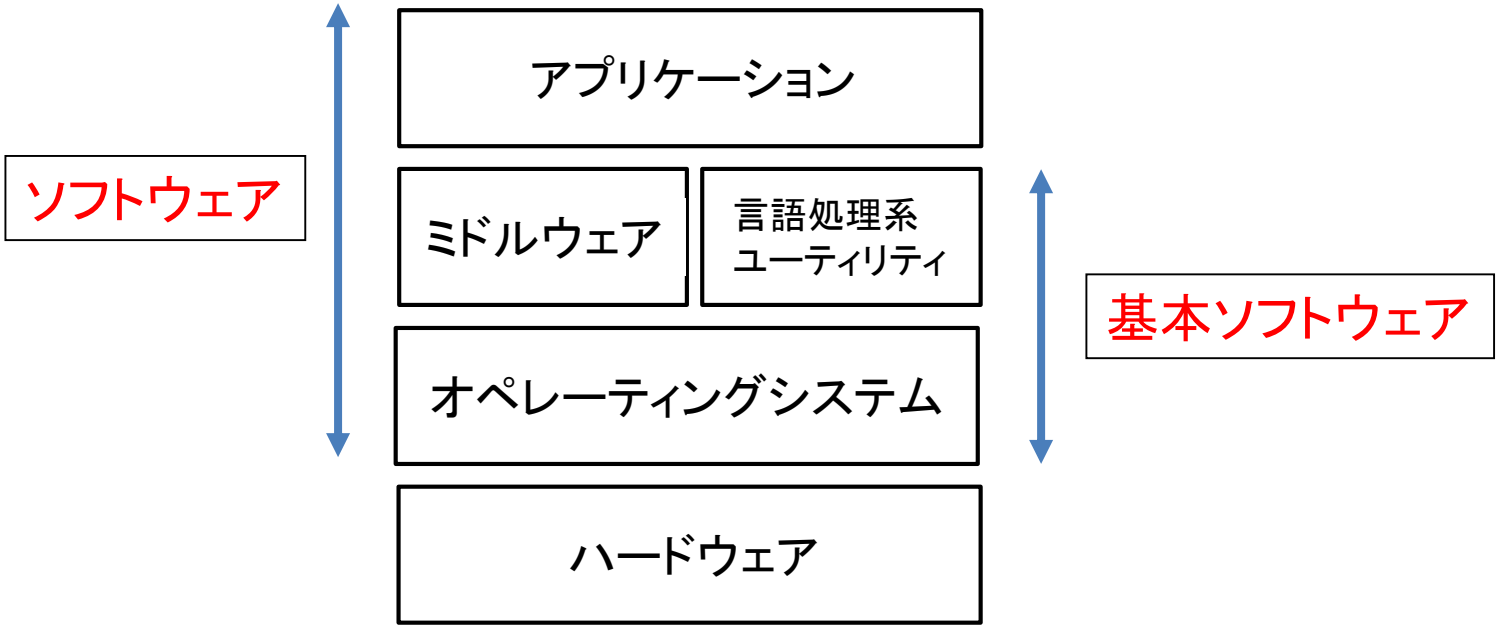
- これまでに多くの言語が開発されてきた
  - 研究・実験目的で開発された言語も入れるとその数は膨大
- プログラミング言語の研究成果と経験則から発展
  - 理論的な研究，実装技術の研究，ソフトウェア工学的な研究

- 素朴な機能から洗練された機能へと発展
  - FORTRAN: なんとなく必要そうな機能を入れたもの
  - 構造化言語：FORTRAN の反省からうまれた一連の言語
    - 手続き抽象という機能を中心に設計されている
  - オブジェクト指向言語：
    - データ抽象という考え方をさらに発展させた言語
  - 関数型言語：
    - 理論的な取り扱いがしやすい
    - 先進的な機能は関数型言語起源のものも多い

ソフトウェアの階層

ソフトウェアの階層（4.4節）

- ソフトウェアにはいろいろな種類がある
  - ハードウェアに近いところで動くオペレーティングシステムから，ユーザに近いところで動くアプリ-ケーションまで
- さまざまなソフトウェアが互いに連携しながら動いている



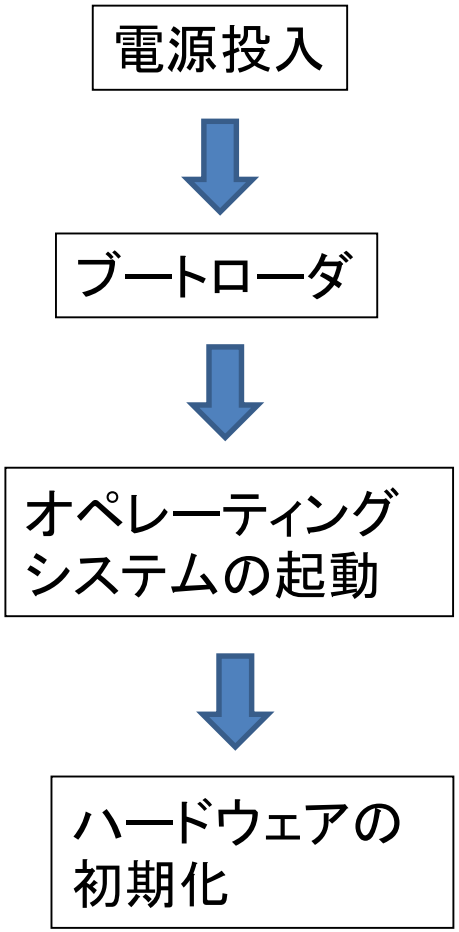
（1）オペレーティングシステム①

- 英語で Operating System. 略して OS
  - Windows, Mac OS，Linux, Android などが OS の代表例

- コンピュータに電源を入れると？
  - 電子機器としてのコンピュータが動き出す
  - しかし，コンピュータはプログラムに従って動作するだけ
  - 電源投入直後にはどんなプログラムを実行するの？
    - OS を実行するように仕組まれている

オペレーティングシステム②

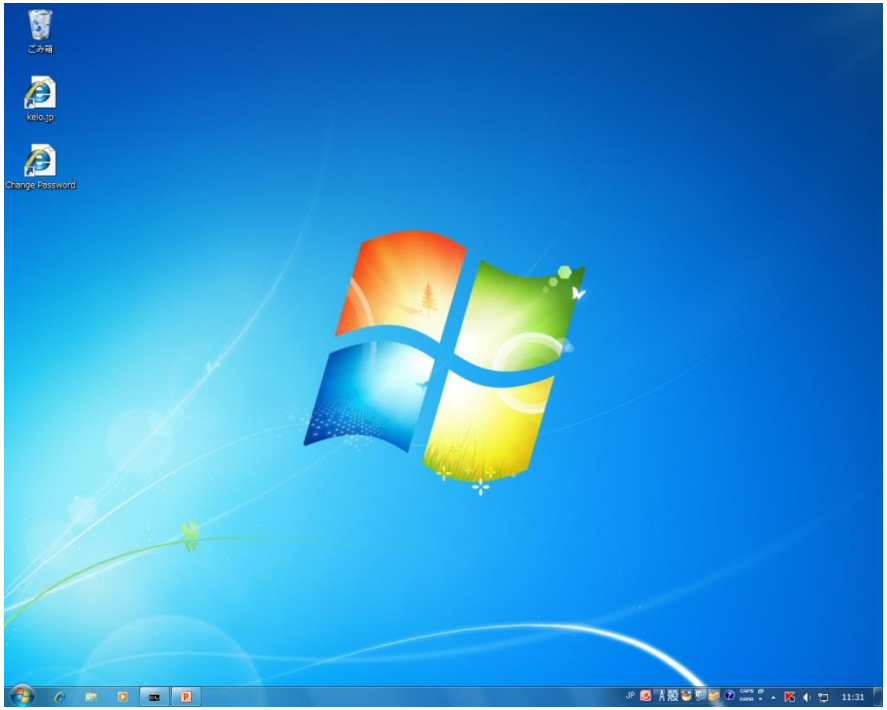
- コンピュータの起動（ブート，ブートストラップ）
- ブートローダ―
  - IPL（Initial Program Loader）
  - オペレーティングシステムを動かすプログラム



オペレーティングシステム③

- OS が動き始めると・・・
  - ハードウェアの初期化を行う
    - コンピュータにどんな機器が接続されているのかを調べる
    - 接続されている機器を初期設定を行ない，使える状態にする
  - OS 自身の初期設定を行う
  - さまざまなソフトウェアを立ち上げ，コンピュータを使える状態にする

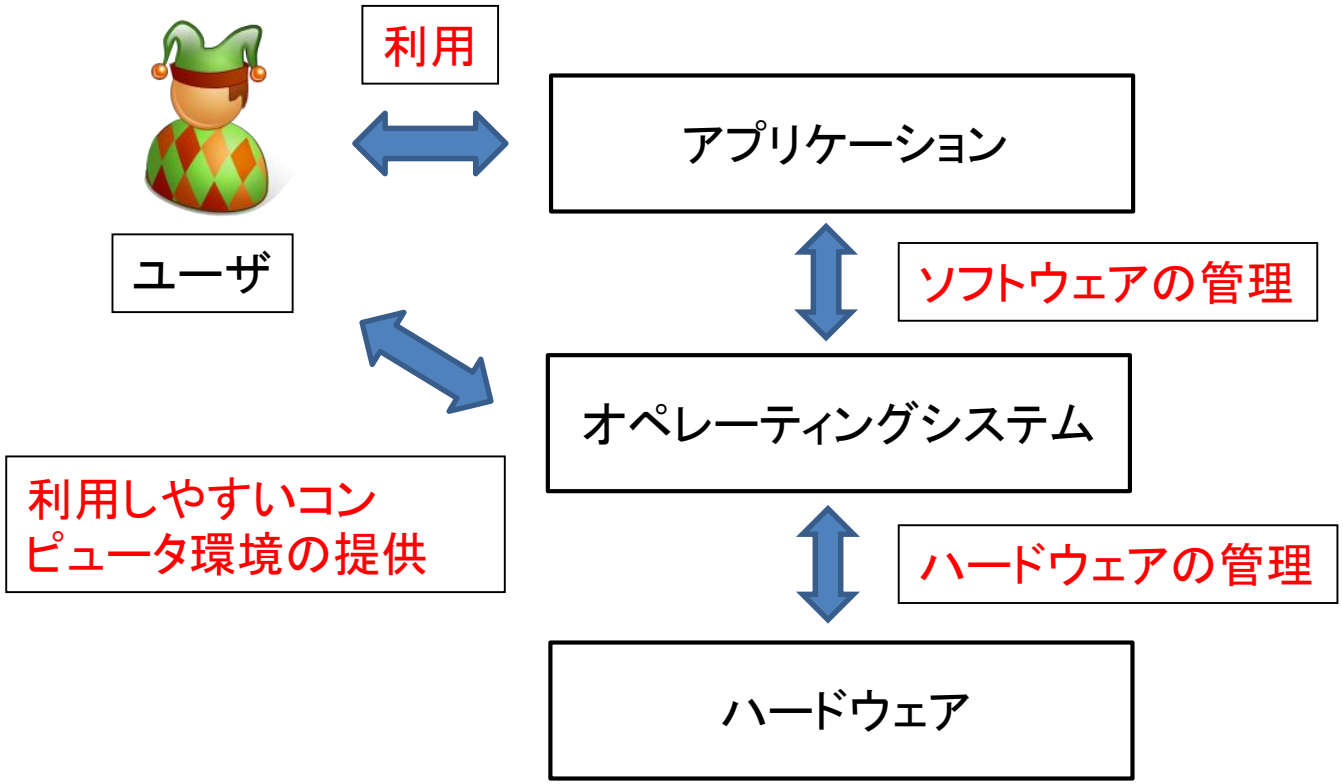
オペレーティングシステム④



Microsoft Windows

Apple Mac OS

オペレーティングシステムの役割



OS が提供する機能の例：ファイル

- ファイルという機能がないと・・・
  - ハードディスクはセクタという番号のついた箱が並んだもの
  - ひとつの箱（セクタ）には 512バイトのデータが入れられる
  - ファイルの保存：
    ファイルを 512バイト毎に分け，それぞれをセクタに入れる
    たとえば，100番と101番のセクタに入れる
  - ファイルの読み出し：
    セクタ番号を指定して読み出す
    セクタ番号を覚えていないといけない
- ファイルという機能があると・・・
  - ファイル名だけを覚えてけばよい

ハードディスク

ファイル名と拡張子（1）

- ファイル名の拡張子ファイル名の末尾につけられた . （ピリオド）で始まる部分
  - 例 :
    - ファイル名 : word.doc 拡張子は .doc
    - ファイル名 : sample.pdf 拡張子は .pdf
- 最近の OS では拡張子を表示しないことが多い
  - 表示するように設定を変更すると拡張子が見えるようになる
- 拡張子はファイルの種類を表す（表4.1）
  - Microsoft Word 用のファイルなら .doc, .docx
  - PDF 形式のファイルなら .pdf

ファイル名と拡張子（2）

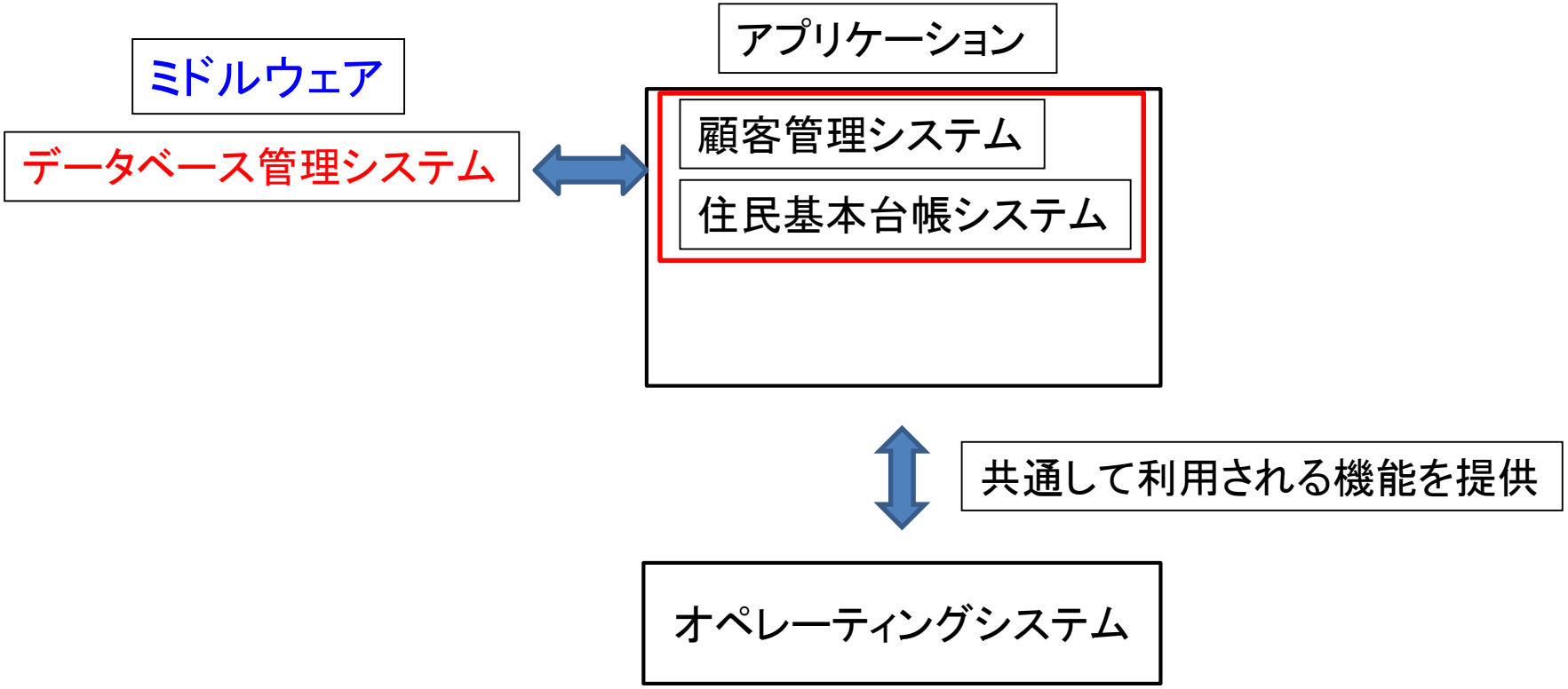
- 拡張子を見て表示するアイコンを決めている
  - .doc, .docx なら 
  - .pdf なら 
- ファイルの中身を見て表示するアイコンを決めているわけではない
  - .doc を .pdf に無理矢理変更するとアイコンが変わる
  - しかし，ファイルの中身は変わらない．ワード形式のまま
  - したがって，PDF 形式のファイルとして扱うことはできない

(2)基本ソフトウェア

- ユーティリティプログラム
  - OSに指示を出すための一連のプログラム群のこと
    - ファイル管理(ファイル，フォルダのコピー，変更，削除)
  - OS入門はユーティリティ・プログラムの使い方を解説
- プログラミング言語処理系
  - コンパイラ，インタプリタ
- 基本ソフトウェア
  - OS，ユーティリティプログラム，言語処理系

(3)ミドルウェア

- アプリケーションの実行を支えるソフトウェア
  - アプリケーションを支えるという点は OS と同じ
- 特定の分野で共通して利用される機能の提供



データベース管理システム

- データベース
  - 特定の目的のために収集したデータ

個人番号	名前	性別	年齢	住所
600001	慶應 太郎	男	22	横浜市港北日吉3-14-1
600002	三田 花子	女	22	港区三田2-14-45

- データベース管理システム
  - データベース構築，運用，管理のためのソフトウェア

(4)アプリケーション・ソフトウェア

- ワードプロセッサ，ウェブブラウザ，メーラなど
  - iPhone などのスマートフォンのアプリを想像すればよい
  - ユーザのために直接，何かをやってくれるソフトウェア

- OS やミドルウェアはアプリケーションを支えている
  - アプリケーションは OS やミドルウェアの力を借りて動いている

アプリケーション・ソフトウェア

- ユーザの望む処理を直接行なうソフトウェア
  - 文書処理
  - 作図と画像処理
  - 表計算とデータ処理
  - プレゼンテーションソフトウェア
  - 数式処理と数値解析
  - ネットワークアプリケーション
  - マルチメディアアプリケーションとオーサリングツール

アプリケーションソフトウェアの種類

アプリケーションソフトウェアの種類（4.5節）

- 文書処理
- 作図と画像処理
- 表計算とデータ処理
- プレゼンテーションソフトウェア
- 数式処理と数値解析
- ネットワークアプリケーション
- マルチメディアアプリケーションとオーサリングツール

本日の内容

- 情報処理の基礎概念2：ソフトウェア
  - ソフトウェアの役割（4.1節）
  - アルゴリズムとプログラム（4.2節）
  - プログラミング言語（4.3節）
  - ソフトウェアの階層（4.4節）

- 次回は5章を予習して来て下さい（復習も忘れずに）