コンピュータ・サイエンス2

第12回
プログラミング・アルゴリズム(実習)

人間科学科コミュニケーション専攻
白銀 純子

第12回の内容

- プログラミング・アルゴリズムの実習

前回の復習

整列(ソート)アルゴリズム(p.123)

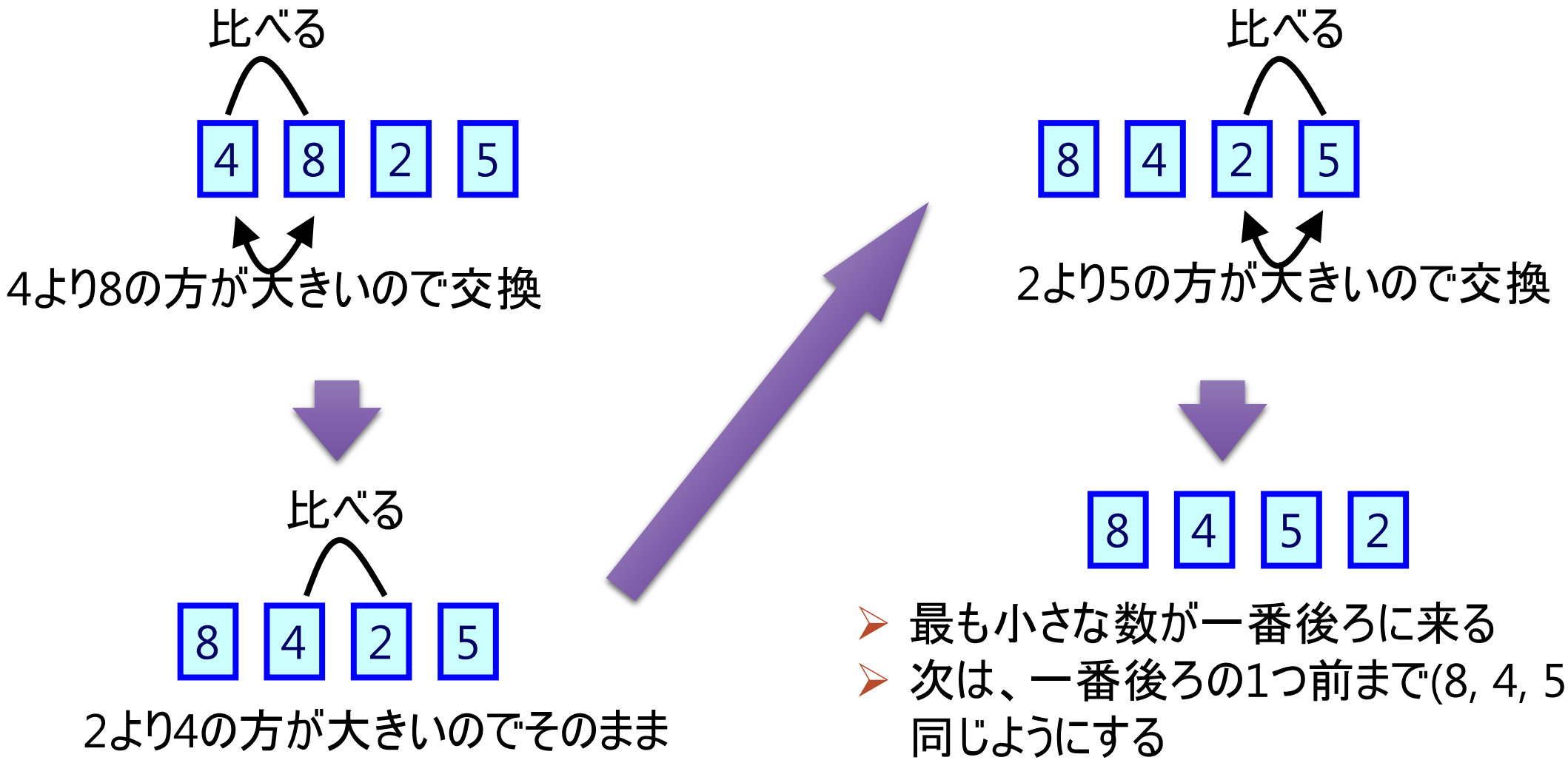
整列[ソート](p. 123)

- ソート: 複数の数を小さい順or大きい順に並べること
  - 選択ソート
  - バブルソート
  - 挿入ソート
  - クイックソート
  - etc.

バブルソート[1](p. 124)

- 前から2つずつ、数の大きさを比較して、小さい数を後ろに送っていく
  - 最後まで調べると、最も小さな数が一番後ろにある
  - この作業を、並べ替える数の個数だけ繰り返すと、数が大きい順に並ぶ

バブルソート[2](p. 124)



- 最も小さな数が一番後ろに来る
- 次は、一番後ろの1つ前まで(8, 4, 5)で同じようにする

アルゴリズムのよしあし

良いアルゴリズム(p. 126)

- そのアルゴリズムを使ったプログラムをコンピュータで実行するときの処理時間や記憶領域の使用量
- アルゴリズムのわかりやすさ・作りやすさ・修正の容易さ

アルゴリズムの計算時間(p. 126)

- アルゴリズムをコンピュータで実行したときの処理時間
  - CPUそのものの速さ
  - CPUとメインメモリとの間のアクセスの速さ
  - etc.

これらを除いても、同じ結果を出す複数のアルゴリズムで計算時間に違いが出る

➡ アルゴリズムの計算量

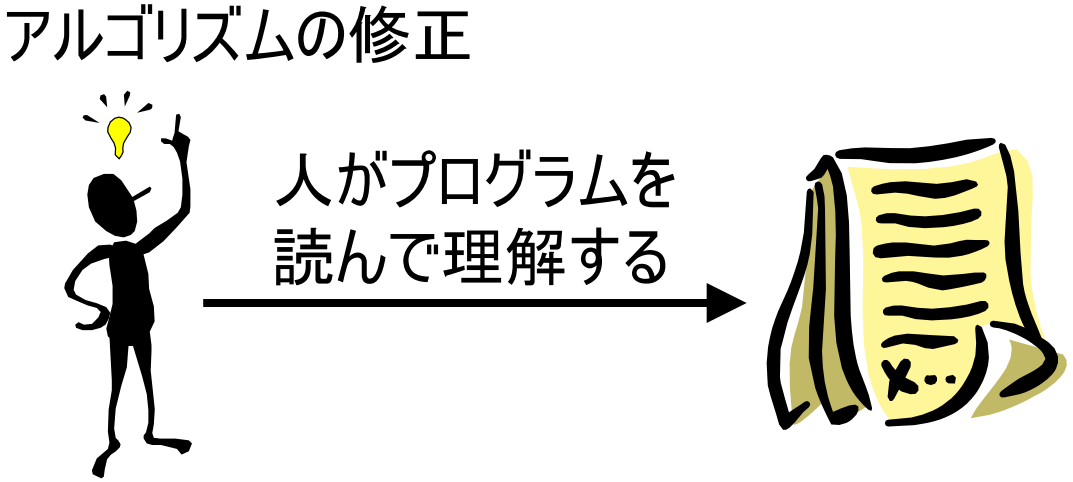
※同じコンピュータで同じアルゴリズムの処理をしても、そのときどきで処理に必要な時間が異なる

アルゴリズムの計算量(p. 126)

- CPUの速さなど、アルゴリズムには関係ない要因を除いた、アルゴリズムそのものの計算時間
  - アルゴリズムそのものの計算(処理)の速さ
- アルゴリズムでの計算の複雑さ

アルゴリズムのわかりやすさ(p. 126)

- 一旦完成したプログラム: 機能の追加などのためにプログラムの修正が必要なことも多い



プログラムを読んで理解する = 書かれてあるアルゴリズムの理解が必要

- アルゴリズムが難解 ≒ 修正が難しい
- アルゴリズムが簡単 ≒ 修正が容易

アルゴリズムを使う状況(p. 126)

- 多くの場合、計算量の少ない(処理時間の速い)アルゴリズムとわかりやすいアルゴリズムは対立関係
  - 計算量が少なければ、わかりにくいアルゴリズム
  - わかりやすければ、計算量が多いアルゴリズム

    速さをとるか、わかりやすさをとるかは、状況に応じて判断
    - めったに使わないプログラムや頻繁に修正するプログラム: わかりやすいアルゴリズム
    - よく使うプログラムや計算時間に制約があるプログラム: 速いアルゴリズム

ソートアルゴリズムの比較[1](p. 126)

- 結果が出るまでの基本処理の回数(アルゴリズムの計算量)
  - バブルソート: $N(N-1)/2$
  - 併合ソート: $N/2+(N-1)\log_2 N$

  } N: 並べ替える数の個数

  ※$\log_2 N$: Nを2^kとしたときの「k」

N	N(N-1)/2(バブルソート)	$N/2+(N-1)\log_2 N$(併合ソート)
8	28	25
32	496	171
64	2016	410
128	8128	953

➡ Nが大きければ大きいほど、併合ソートの方が速い

ソートアルゴリズムの比較[2](p. 126)

- 計算量: 入力(N: 並べ替えの場合は数の個数)に対して行われる基本処理の回数
  - Nが十分に大きなとき: 計算式の中の最も大きな項だけに着目して、大まかに計算
    = 各項の比例定数や次数の低い項は無視
    - バブルソート: $N(N-1)/2 = N^2/2 - N/2$
      →N^2のみに注目
    - 併合ソート: $N/2+(N-1)\log_2 N = N/2 + N\log_2 N - \log 2N$
      →$N\log_2 N$のみに注目

アルゴリズムの計算量は、正確な計算量ではなく、Nが大きくなればどの程度の割合で計算量が増えるかを大まかに知ることが重要なため

注目する項を取り出して、O(...)と表記
- $O(N^2)$や$O(N\log_2 N)$など

ソートアルゴリズムの比較[3](p. 126)

- 計算時間の速いアルゴリズム: NやlogNなどのみで計算量が計算できるアルゴリズム
- 計算時間の遅いアルゴリズム: N^2, N^3, ..., N^kやN!(1からNまでをかけあわせた数), 2^Nなど、多くのかけ算を計算に必要とするアルゴリズム
  - N^2, N^3などの計算を必要とするアルゴリズム: 多項式時間アルゴリズム
  - N!や2^Nなどの計算を必要とするアルゴリズム: 指数時間アルゴリズム

扱いにくい問題

扱いにくい問題(p. 127)

- コンピュータでの処理が難しい問題も存在

データの個数に対する処理時間
(1回の処理に0.0000001秒かかるコンピュータ)

n	logn	n	nlogn	n^2	n^3	2^n	n!
10	0.0000003	0.000001	0.000003	0.00001	0.0001	0.0001024	0.36
20	0.0000004	0.000002	0.000009	0.00004	0.0008	0.1048576	7700年
30	0.0000005	0.000003	0.000015	0.00009	0.0027	107	8×10^{18}年
50	0.0000006	0.000005	0.000028	0.00025	0.012	3.4年	
100	0.0000007	0.00001	0.000066	0.001	0.1	4×10^{15}年	
10000	0.0000013	0.001	0.0133	10	1.2日		
1000000	0.0000023	1.0	23	116日	3200000年		

※「日」や「年」の書いていない数の単位は「秒」

扱いにくい問題[ナップザック](p. 127)

- 重さと値段のわかっているN個の荷物をナップザックに詰め込むとき、合計金額を最大いくらにできるか
  - 荷物の重さの合計はWを超えてはならない

解答例: 荷物の全ての組み合わせを作って
重さの合計と金額の合計を計算

荷物がN個の場合、荷物の組み合わせは2^N通り
= 重さの合計と金額の合計を2^N回計算する必要

例えば…
荷物が60個、計算の基本処理1回分が1000万分の1秒の場合:
$1/1000万 \times 2^{60}$ 秒 ≒ 3000年

➡ アルゴリズムを作ることはできるが、
計算時間が非現実的!

扱いにくい問題[セールス](p. 127)

- セールスパーソンが、A_0町(駅)からN個の町(駅)を回ってA_0町(駅)に帰るまでに最小のコスト(交通費)の経路を求める

解答例: A_0から始まって、N個の町を回って帰ってくることができる全ての経路を考え、それぞれの経路のコストの合計の中で最小のものを求める

町がN個の場合、経路の組み合わせは(N-1)!通り
= 調べなければならない経路が(N-1)!通り

例えば…
町が30個、計算の基本処理1回分が100万分の1秒の場合
$(30 - 1)! \times 1/100万 = 8.8 \times 10^{30}回 \times 1/100万 \fallingdotseq$ 3800年

➡ アルゴリズムを作ることはできるが、計算時間が非現実的!

扱いにくい問題への対応(p. 127)

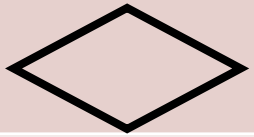
- 入力が特殊な条件を満たす場合は、扱いやすくなることもある
- 最適解でなく、近似解で良ければ、扱いやすくなる
  - 最適解: 最も良い答え
  - 近似解: 最も良いわけではないかもしれないが、他の多くの答えよりは良い答え

プログラムでの基本処理の復習～実習に向けて～

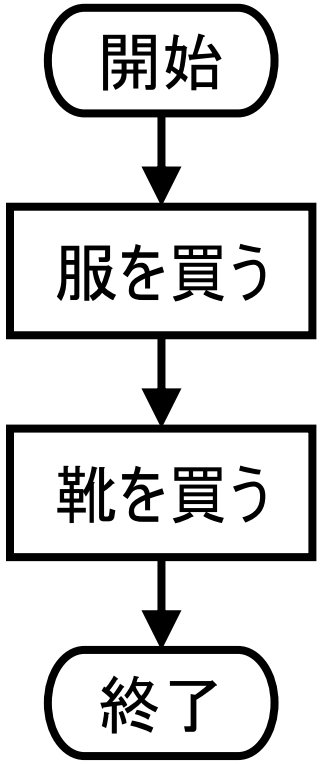
プログラムでの基本処理(p. 119)

- 順次処理、条件分岐、反復処理が基本
  - 順次処理:プログラム中に書いてある命令を、上から順に1つずつ処理すること
  - 条件分岐:ある条件を満たしたときとそうでないときで、処理内容が変わること
  - 反復処理:ある条件が満たされている限り、処理を繰り返すこと

フローチャート[1](p. 119)

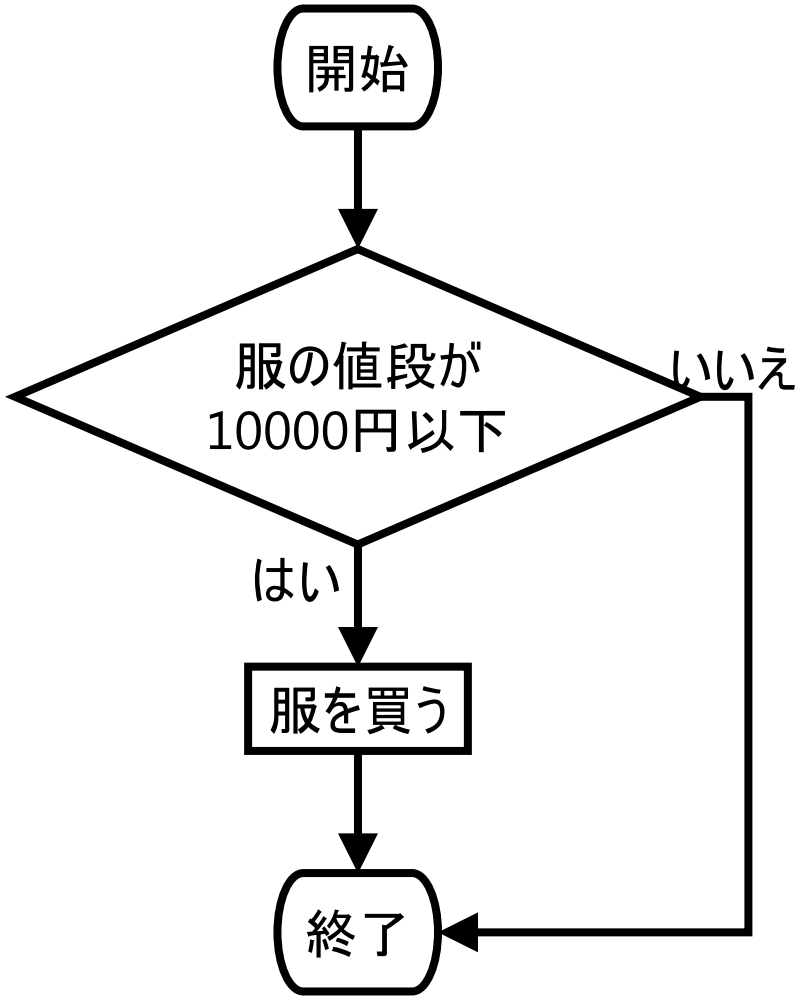
記号	意味
	開始と終了
	処理
	条件判断
	処理の流れ

順次処理の例

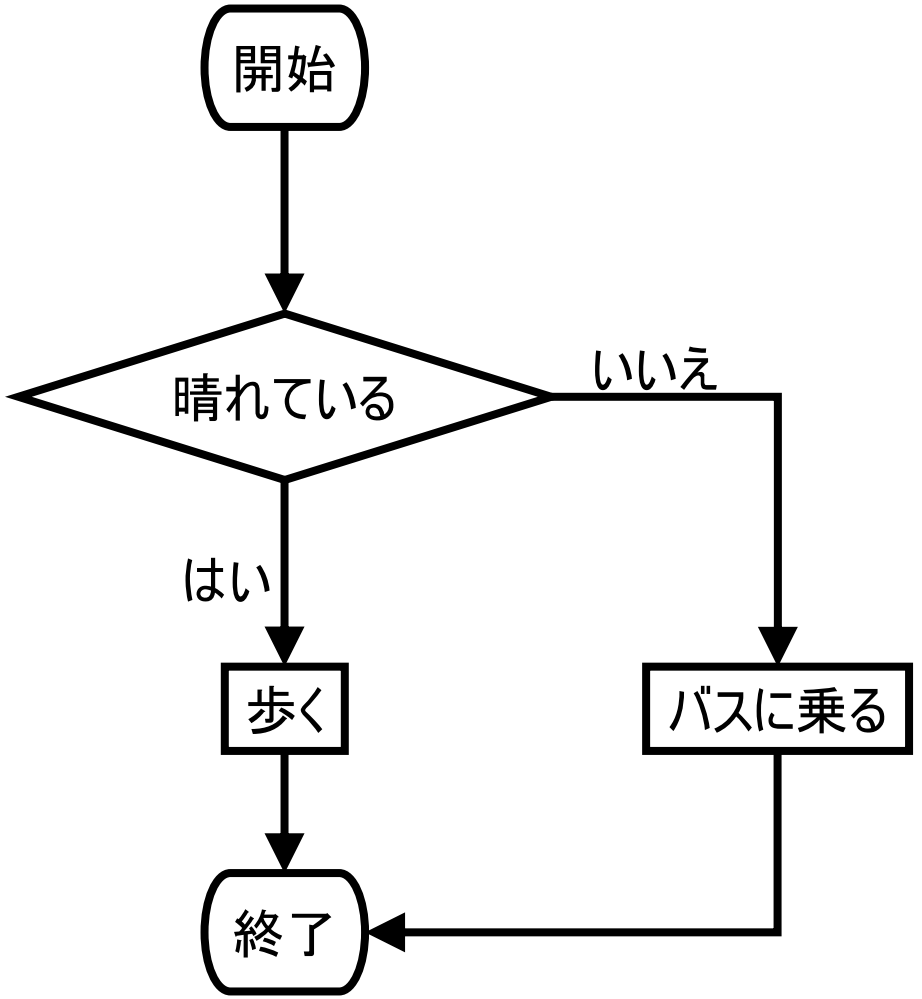


フローチャート[2](p. 119)

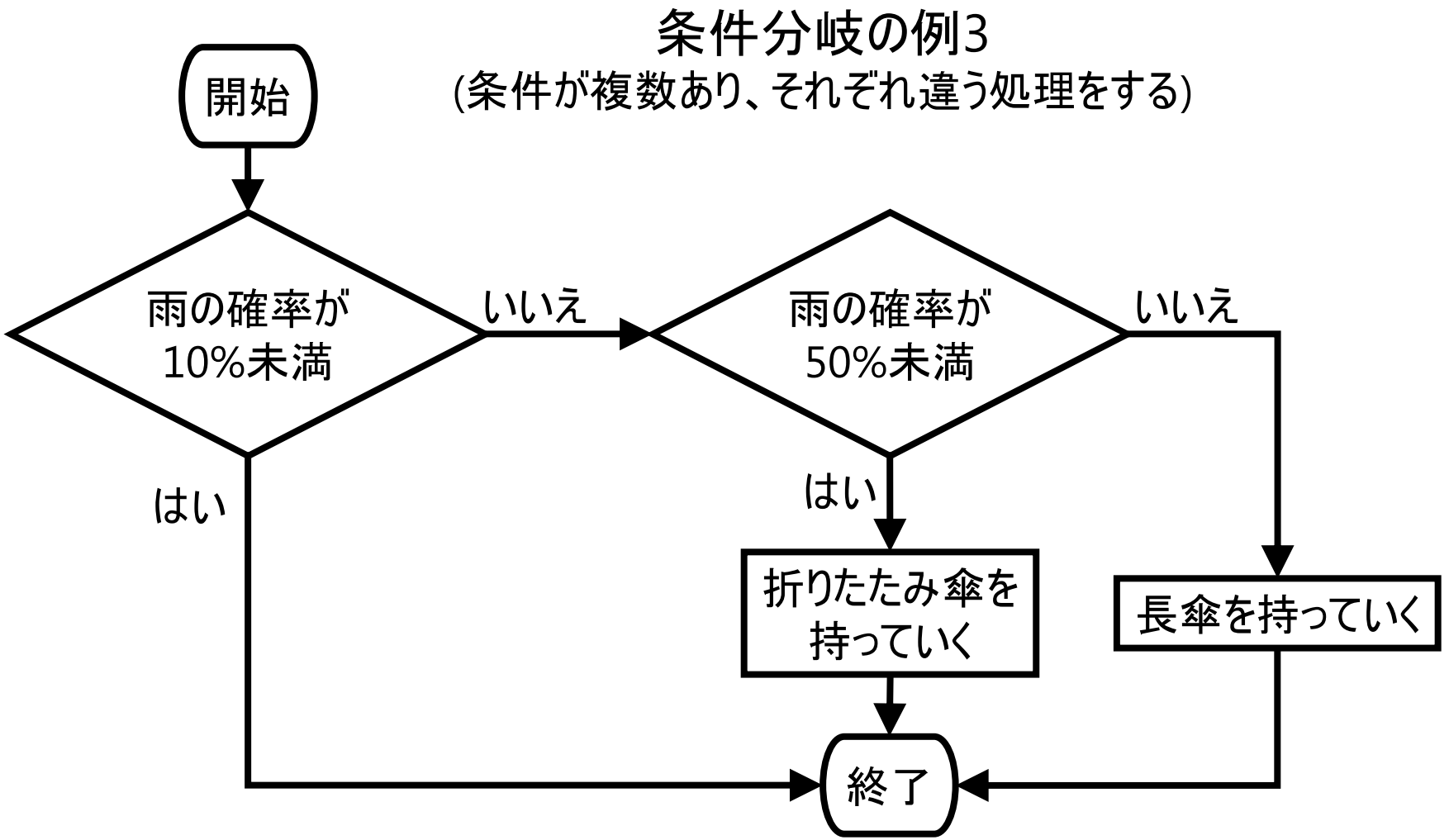
条件分岐の例1
(条件判断が「いいえ」の場合何もしない)



条件分岐の例2
(条件判断が「いいえ」の場合別のことをする)

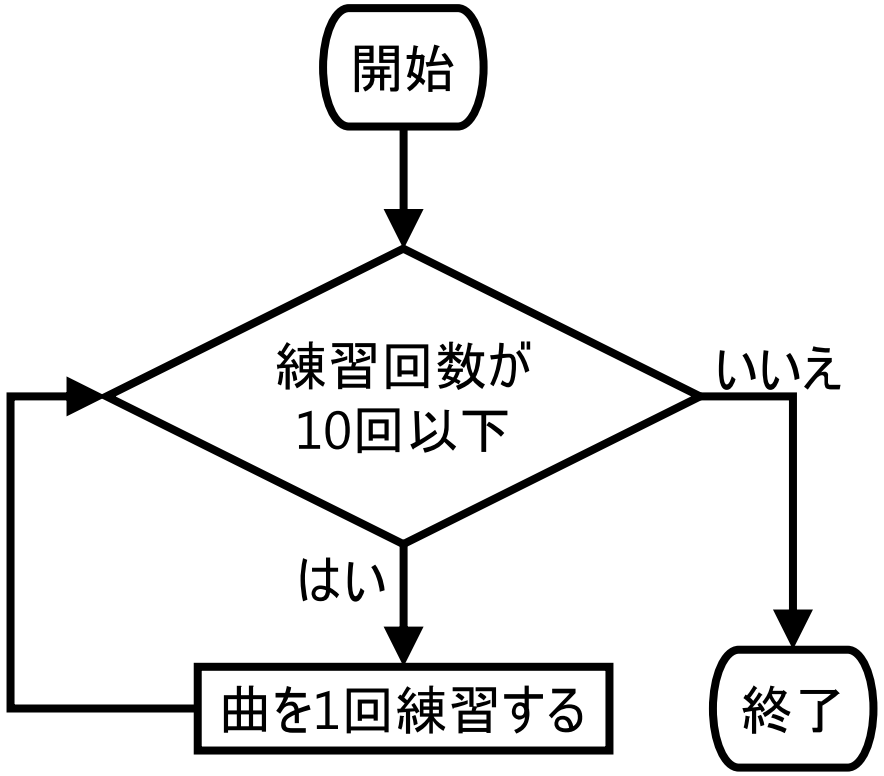


フローチャート[3](p. 119)

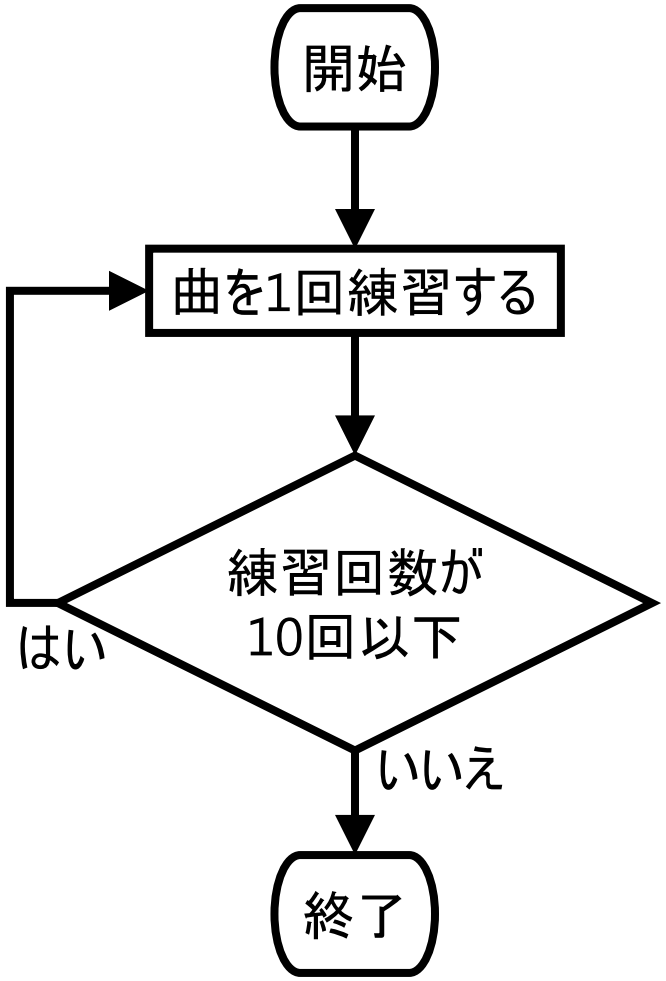


フローチャート[4](p. 119)

反復処理の例
(最初に条件を判断)



反復処理の例
(最初に処理をしてから条件を判断)



基本処理の実習

順次処理・条件分岐・反復処理を理解しよう!

- アルゴロジック(授業のページからアクセス可能)
  http://home.jeita.or.jp/is/highschool/algo/index.html
  - 順次処理・条件分岐・反復処理を理解するためのゲーム
    - 「アルゴロジック」で順次処理と反復処理
    - 「アルゴロジック2」で順次処理・条件分岐・反復処理の組み合わせ
  - マス上の人を、旗のマスまで移動させて旗をゲットするためのアルゴリズムを作成
    - 旗のマスが複数ある場合は、全ての旗をゲットさせる必要

アルゴロジック(使い方)[1]

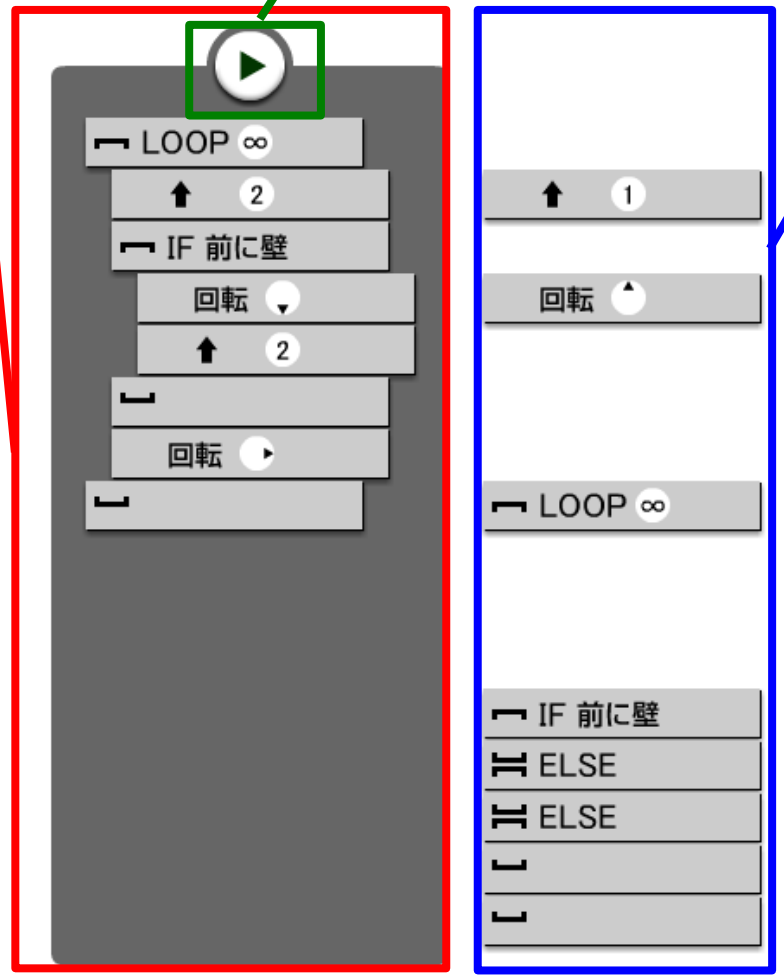
- アルゴロジックのページから「アルゴロジック」(または「アルゴロジック2」)の「ゲームスタート」から開始
  - アルゴロジック: 「アルゴロジックJr.(初心者問題)」または「アルゴロジック(チャレンジ問題)」
  - アルゴロジック2: 「START」
- 遊ぶコースを選択

アルゴロジック(使い方)[2]

- 「アルゴロジック2」をやってみよう!

処理内容を実行させるためのボタン
- 処理が止まらなくなったら、もう一度押す

マス上の人を動かす
処理内容
- 上から順に実行される



処理に使う部品
- マウスでドラッグして左の枠に移動させる

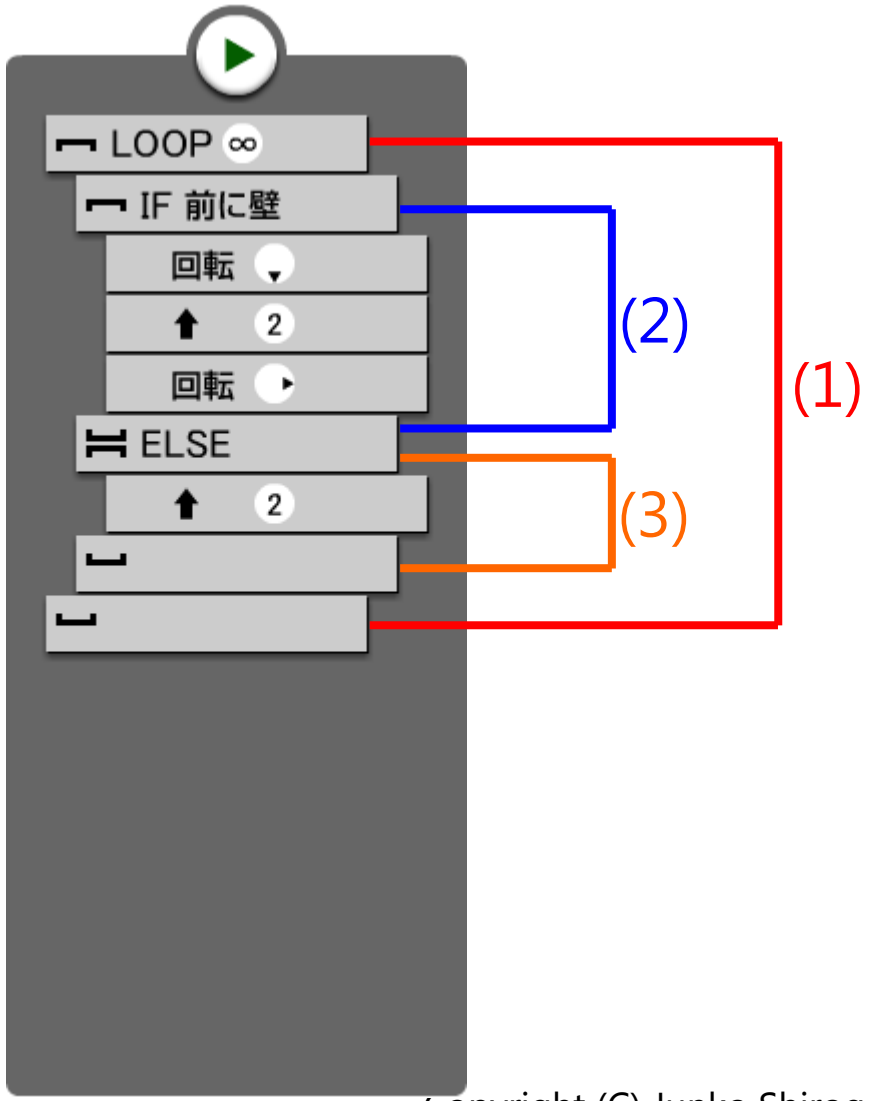
アルゴロジック(使い方)[3]

部品	使い方
⬆ 1	➢ 人を数字の分だけマスを移動させる ➢ 数字部分をクリックすることで、数字を変更する
回転 ▲	➢ 人を「▲」の方向に方向転換させる ➢ 「○」内の上下左右をクリックすることで、方向を変更する ➢ 方向は、人が現在向いている方向を基準にして設定する ✓ 例えば人が現在下を向いていて、「▼」の方向を設定すると、人は上を向く
IF 前に壁	人の前が壁だった場合に、この部品から ⌙ の部品までの処理をする
ELSE	➢ 「IF前に壁」の処理の後に置き、人の前が壁でなかった場合の処理を定義する ➢ ⌙ までで、人の前が壁でなかった場合の処理を定義する
LOOP ∞	➢ この部品から ⌙ の部品までの処理を繰り返す ➢ 「∞」の部分をクリックすることで、繰り返す回数を変更する

※「IF前に壁」や「LOOP」は、対になる ⌙ が配置されていなければエラー

アルゴロジック(使い方)[4]

- 処理の設定例



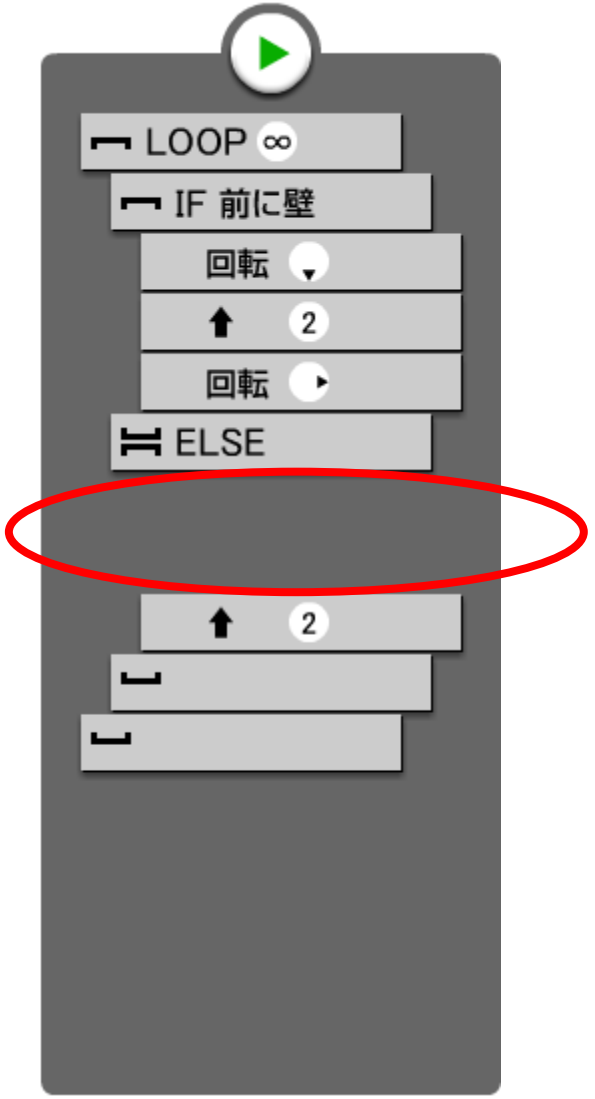
(1) 「LOOP」の処理
- (2)と(3)の処理を繰り返す
- ただし、(2)と(3)はどちらか一方のみ処理される
  - ✓ 人の前に壁があるかないかでどちらが処理されるかが決まる

(2) 「IF前に壁」の処理: 人の前に壁があった場合
1. 人を逆方向に回転させる
2. 人を2マス進める
3. 人を、人の右手方向に方向転換させる

(3) 「ELSE」の処理: 人の前に壁がなかった場合
- 人を2マス進める

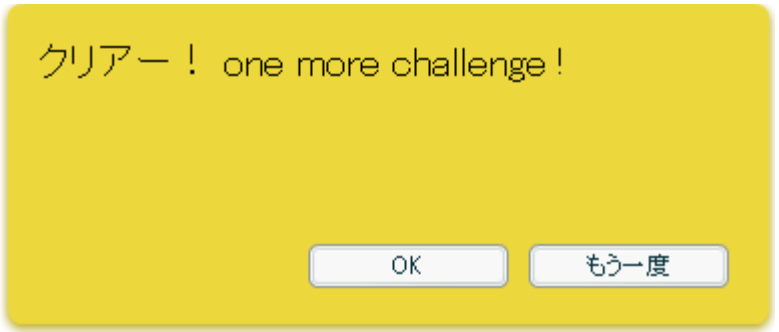
アルゴロジック(使い方)[5]

- 部品をドラッグして、部品と部品との間を空けることも可能
  - 部品の順序を変更したいとき
  - 配置する部品を追加したいとき
  - etc.

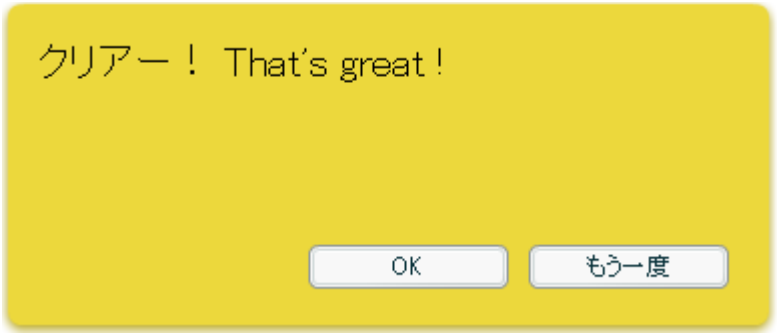


アルゴロジック(使い方)[6]

- 人が全ての旗をゲットできるアルゴリズムを作れば、そのコースはクリア
  - ただ単にクリアするアルゴリズムであれば、「one more challenge!」の表示
    - コース一覧の画面で、そのコースに「○」印がつく



  - 最も良いアルゴリズムであれば、「That's great!」の表示
    - コース一覧の画面で、そのコースに「◎」印がつく



本日の出席確認

- アルゴロジック2のコースに挑戦しよう!
  - 目標: 指定されたコースのうち、異なるグループから2つ以上をクリアすること
    - 「one more challenge!」の表示でOK(もちろん、「That's great!」の表示もOK)

指定コース

出席確認[2]

- 挑戦結果のスナップショットを junko@cis.twcu.ac.jp 宛にメールに添付して送信

- スナップショット: 画面の表示を撮影したもの
  - 「Command」キーと「Shift」キーの2つのキーを押したまま「3」キーを押す
  - デスクトップに、画面の内容を撮影した画像ファイルが保存される
  - 保存された画像ファイルをメールに添付して送る

プログラミング言語の復習

プログラミング言語の変遷[1](p. 76)

- プログラム: コンピュータに命令を伝えるための文書
- プログラミング言語: プログラムを記述するための言葉

初期: 機械語でプログラムを記述

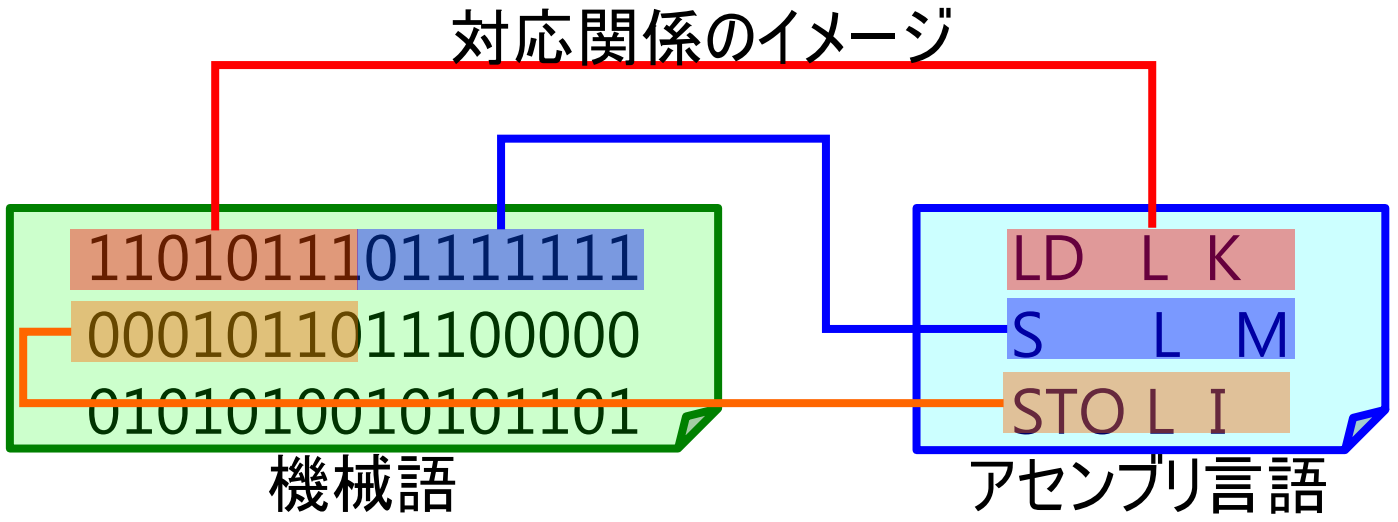
- 機械語: 0と1の2進数の形式の言葉

→ 不便!

→ アセンブリ言語が登場

アセンブリ言語

- 英語に似せた言語
- 機械語と1対1で対応
- アセンブリ言語のプログラムを機械語に翻訳
  - 翻訳ソフトウェア: アセンブラ



プログラミング言語の変遷[2](p. 76)

- 1954年にFORTRAN(FORmula TRANSlator)
  - IBM社が科学技術用言語として提唱
- 1959年にCOBOL(Common Business Oriented Language)
  - アメリカ国防省が商用言語として提唱
- 1962年にBASIC(Beginner's All purpose Symbolic Instruction Code)
  - ダートマス大学で初心者でも使える言語として提唱
  - ビル・ゲイツがよく利用し、Microsoft社が開発に注力

プログラミング言語の変遷[3](p. 76)

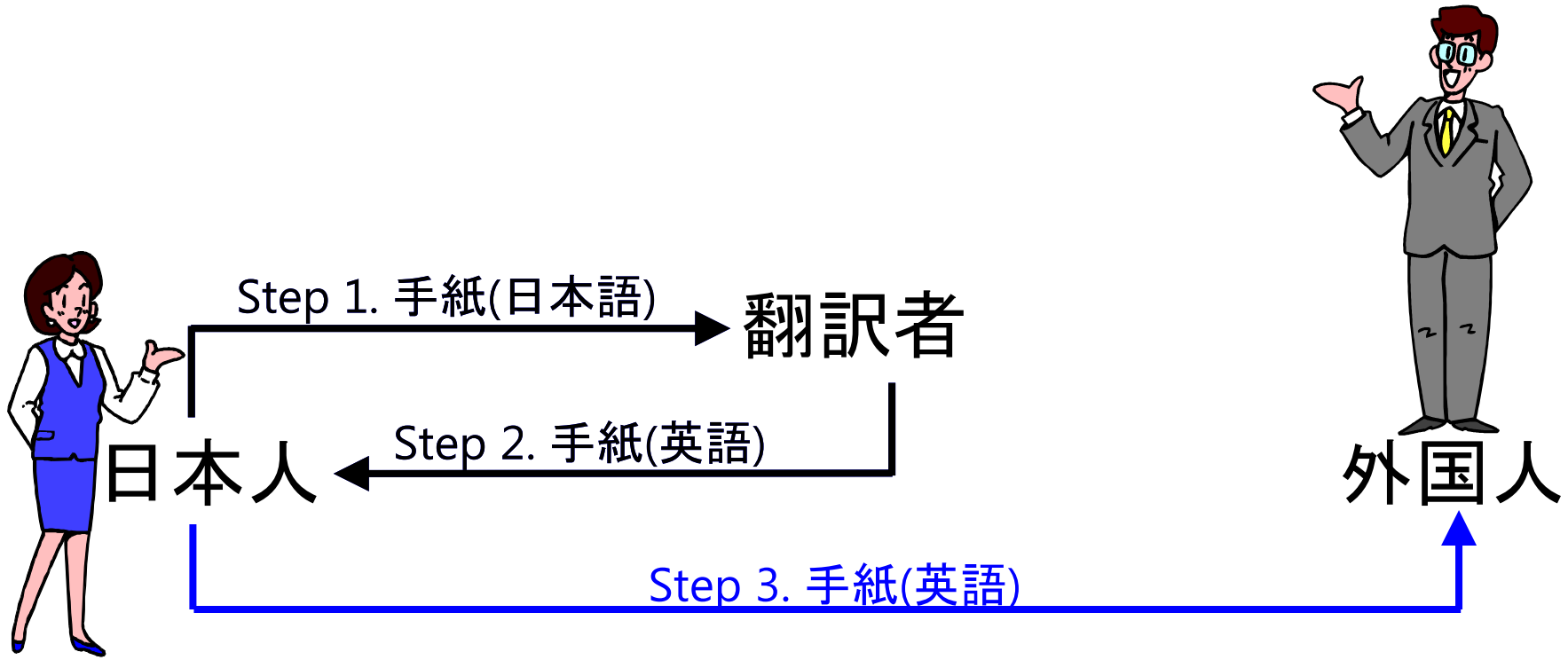
- 1972年にC言語
  - ベル研究所がOSなどの基幹ソフトウェアの開発用言語として開発
  - UNIXがOSとして初めてC言語で記述
- 1972年にSmalltalk
  - ゼロックス社のパロアルト研究所でオブジェクト指向言語として開発
- 1995年にJava
  - サン・マイクロシステムズ社(現オラクル社)でネットワーク対応言語として開発

変換方式

- プログラミング言語で書かれた命令書: 機械語に変換しなければ、コンピュータは実行不可能
- 変換方式は大きく分けて2種類
  - コンパイラ型: 翻訳
  - インタプリタ型: 通訳

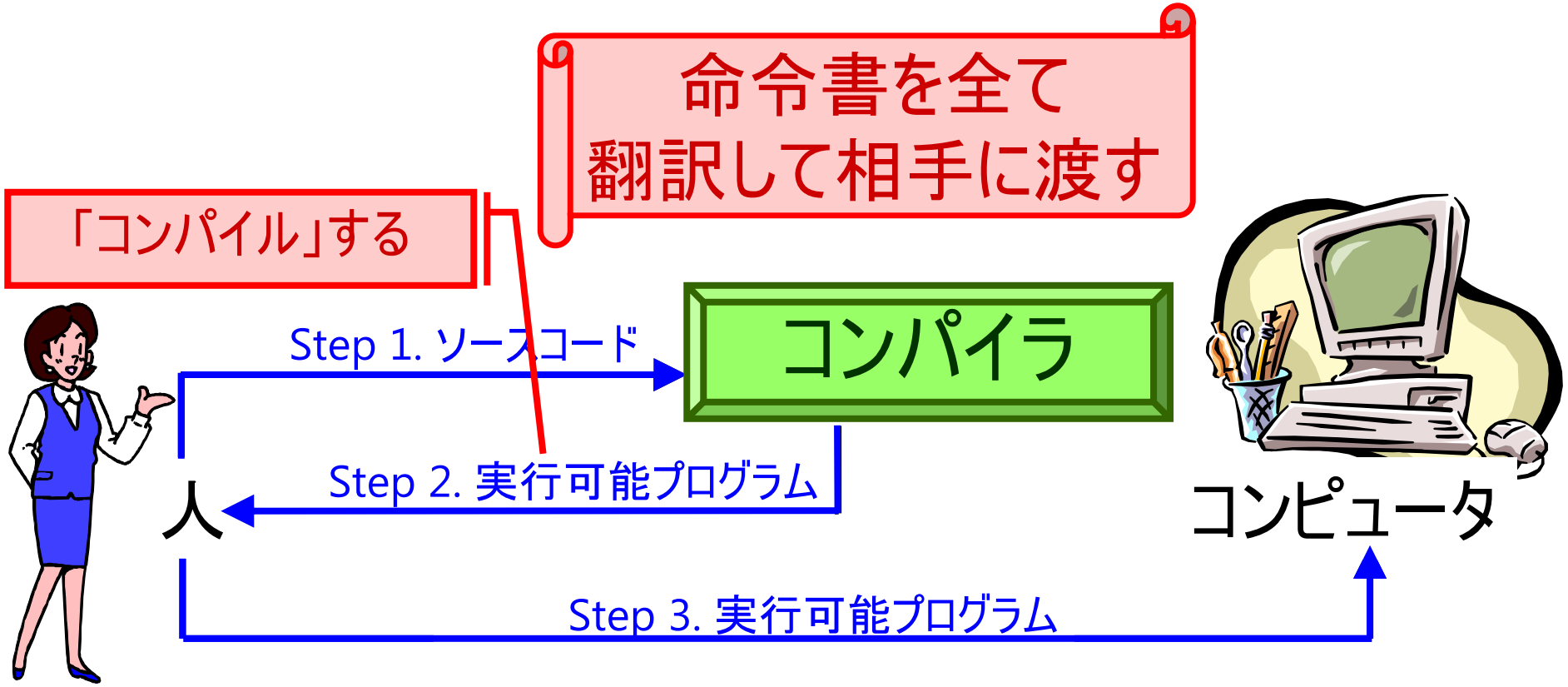
コンパイラ[概要](p. 78)

- コンパイラ: 命令書を機械語に翻訳し、コンピュータで実行可能にするためのソフトウェア

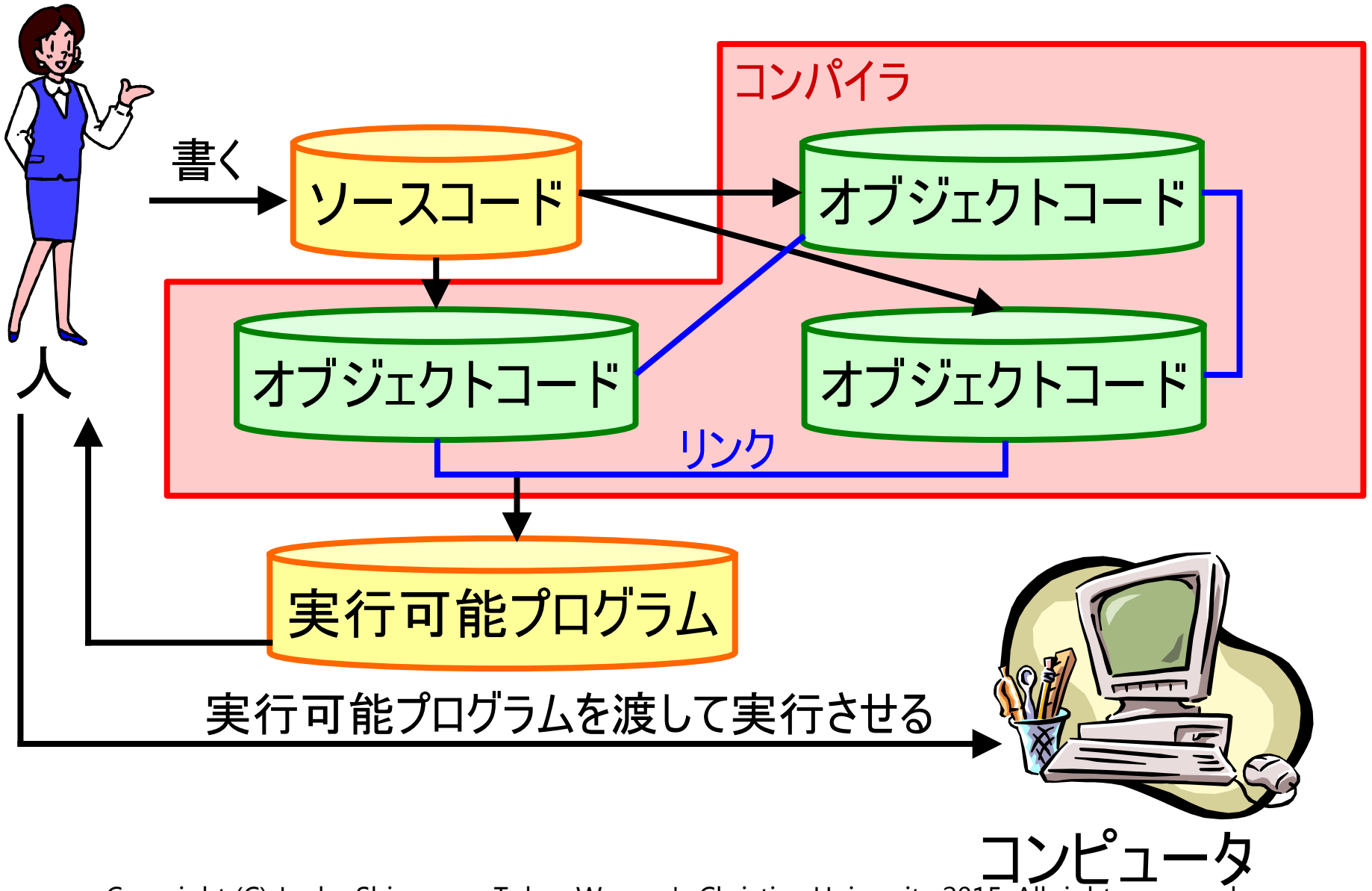


コンパイラ[概要](p. 78)

- コンパイラ: 命令書を機械語に翻訳し、コンピュータで実行可能にするためのソフトウェア



コンパイラ[詳しく](p. 78)

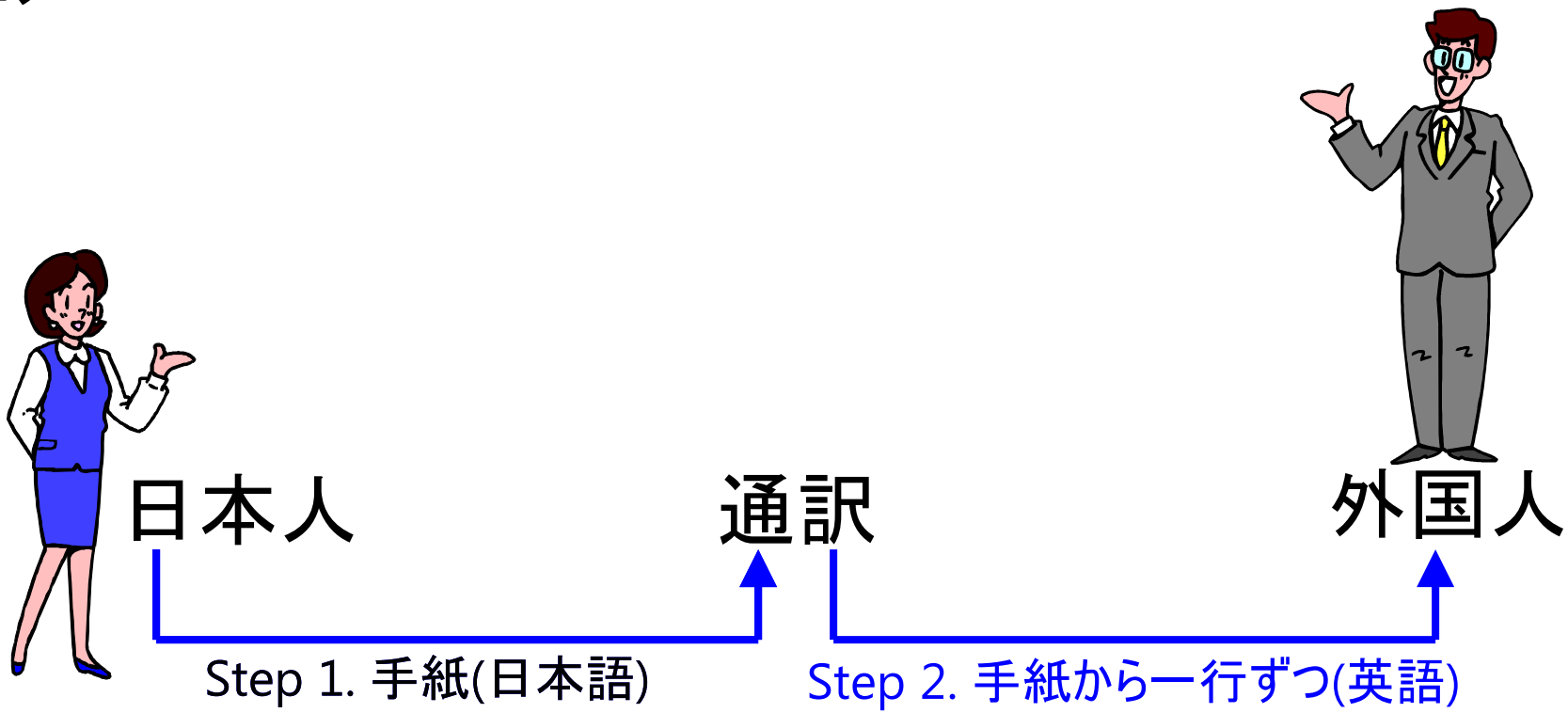


用語[1](p. 78)

- ソースコード: プログラミング言語で記述した命令書
- オブジェクトコード: ソースコードを翻訳したもの
- 実行可能プログラム: オブジェクトコードを連携させて、動作可能な形にしたもの
- プログラム: ソースコードと実行可能プログラムの双方の意味
  - どちらの意味で使われるかは、その時々で異なる

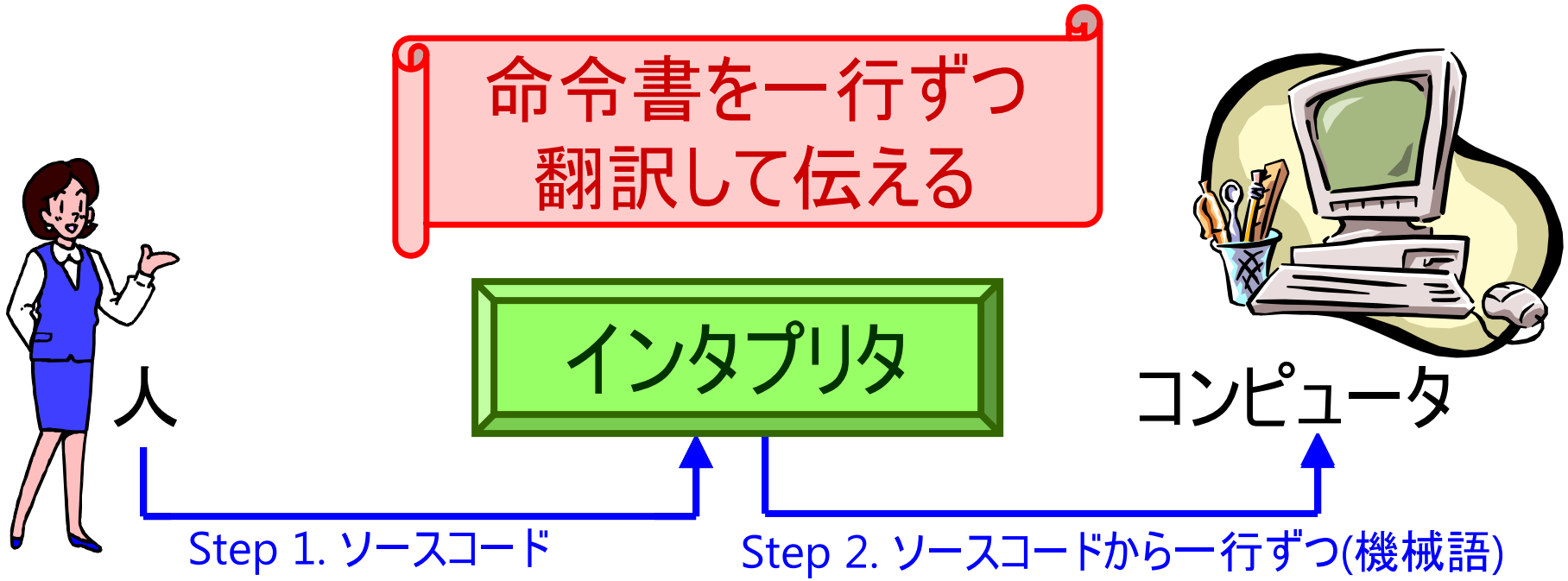
インタプリタ

- インタプリタ: 命令書を最初から1行ずつ読んで機械語に通訳するためのソフトウェア



インタプリタ

- インタプリタ: 命令書を最初から1行ずつ読んで機械語に通訳するためのソフトウェア



プログラミング実習

実習内容

- プログラミング実習
  - 命令書(プログラム)とはどのようなものか?
  - どうやって翻訳・通訳するか?
  - どうやって実行するか?
- アルゴリズム実習
  - アルゴリズムによって、そんなに処理時間に違いがあるか?

準備

- 授業のページから4つのファイルをダウンロード
  - BubbleSort.c
    - バブルソートをするC言語のプログラム
  - BubbleSort.html
    - バブルソートをするJavaScriptのプログラム
  - MergeSort.c
    - 併合ソートをするC言語のプログラム
  - MergeSort.html
    - 併合ソートをするJavaScriptのプログラム

※リンクをクリックするのではなく、右クリック→「ファイルをダウンロード」でダウンロードすること

C言語とJavaScript

- C言語
  - プログラミング言語の一種
  - 記述された命令書を機械語に翻訳した命令書を作成する形式の言語(コンパイラ型の言語)
- JavaScript
  - プログラミング言語の一種
  - 記述された命令書を機械語に通訳する形式の言語(インタプリタ型の言語)
  - Webページでよく利用

BubbleSort.c[前半1]

```
#include<stdio.h>
#include<time.h>
int main() {
  int i, j, temp, num = 30000;
  int random[num];
  for (i = 0; i < num; i++) {
    random[i] = rand();
  }
  FILE *randFile = fopen("BubbleRandomNum.txt", "w");
  for (i = 0; i < num; i++) {
    fprintf(randFile, "%d¥n", random[i]);
  }
  fclose(randFile);

  int start, end, procTime;
  start = clock();
```

並べ替えをする数の個数

BubbleSort.c[前半2]

```
#include<stdio.h>
#include<time.h>
int main() {
  int i, j, temp, num = 30000;
  int random[num];
  for (i = 0; i < num; i++) {
    random[i] = rand();
  }
  FILE *randFile = fopen("BubbleRandomNum.txt", "w");
  for (i = 0; i < num; i++) {
    fprintf(randFile, "%d¥n", random[i]);
  }
  fclose(randFile);

  int start, end, procTime;
  start = clock();
```

並べ替えをする数を
自動的に作成

BubbleSort.c[前半3]

```
#include<stdio.h>
#include<time.h>
int main() {
  int i, j, tem
  int random
  for (i = 0;
    random[i] = rand();
  }
  FILE *randFile = fopen("BubbleRandomNum.txt", "w");
  for (i = 0; i < num; i++) {
    fprintf(randFile, "%d¥n", random[i]);
  }
  fclose(randFile);

  int start, end, procTime;
  start = clock();
```

自動的に作った数を
「BubbleRandomNum.txt」ファイルに書き出し

BubbleSort.c[後半1]

```
for (i = 0; i < num; i++) {
  for (j = 0; j < num - i - 1; j++) {
    if (random[j] > random[j + 1]) {
      temp = random[j + 1];
      random[j + 1] = random[j];
      random[j] = temp;
    }
  }
}
  end = clock();
  procTime = end - start;
  printf("Time(Bubble Sort of %d Numbers): %lf second¥n", num,
          (double) procTim

  FILE *bubbleFile = fopen("BubbleSortNum.txt", "w");
  for (i = 0; i < num; i++) {
    fprintf(bubbleFile, "%d¥n", random[i]);
  }
  fclose(bubbleFile);
}
```

バブルソートをする処理部分

BubbleSort.c[後半2]

```
for (i = 0; i < num; i++) {
  for (j = 0; j < num - i - 1; j++) {
    if (random[j] > random[j + 1]) {
      temp = random[j + 1];
      random[j + 1] = random[j];
      random[j] = temp;
    }
  }
}
end = clock();
procTime = end
printf("Time(Bu
        (double) procTime / CLOCKS_PER_SEC);

FILE *bubbleFile = fopen("BubbleSortNum.txt", "w");
for (i = 0; i < num; i++) {
  fprintf(bubbleFile, "%d¥n", random[i]);
}
fclose(bubbleFile);
}
```

数を並べ替えた結果を
「BubbleSortNum.txt」ファイルに書き出し

BubbleSort.html[前編1]

```
<html>
<head>
<title>バブルソート</title>
<script language="JavaScript">
NUM = 5000;
random = new Array();
for (i=0; i<=NUM; i++) {
random[i] = Math.floor(NUM * Math.random());
}
function bubbleSort() {
  for (i = 0; i < NUM; i++) {
    for (j = 0; j < NUM - i - 1; j++) {
      if (random[j] > random[j + 1]) {
        temp = random[j + 1];
        random[j + 1] = random[j];
        random[j] = temp;
      }
    }
  }
}
```

並べ替えをする数の個数

BubbleSort.html[前編2]

```
<html>
<head>
<title>バブルソート</title>
<script language="JavaScript"><!--
NUM = 5000;
random = new Array();
for (i=0; i<=NUM; i++) {
random[i] = Math.floor(NUM * Math.random());
}
function bubbleSort() {
  for (i = 0; i < NUM; i++) {
    for (j = 0; j < NUM - i - 1; j++) {
      if (random[j] > random[j + 1]) {
        temp = random[j + 1];
        random[j + 1] = random[j];
        random[j] = temp;
      }
    }
  }
}
```

並べ替えをする数を
自動的に作成

BubbleSort.html[前編3]

```
<html>
<head>
<title>バブルソート</title>
<script language="JavaScript"><!--
NUM = 5000;
random = new Array();
for (i=0; i<=NUM; i++) {
random[i] = Math.floor(NUM *
}
function bubbleSort() {
 for (i = 0; i < NUM; i++) {
  for (j = 0; j < NUM - i - 1; j++) {
   if (random[j] > random[j + 1]) {
    temp = random[j + 1];
    random[j + 1] = random[j];
    random[j] = temp;
   }
  }
 }
}
```

バブルソートをする処理部分

BubbleSort.html[中編]

```
function writeNumber() {
 for (i=0; i<NUM; i++) document.write(random[i]," ");
 document.write("<br>");
}
// --></script>
</head>
<body>
<h1>バブルソート</h1>
<h2>並び替え前</h2>
<script langauge="JavaScript"><!--
writeNumber();
// --></script>
```

並べ替えた数をブラウザに
表示する部分

BubbleSort.html[後編]

```
<h2>並び替え後</h2>
<script langauge="JavaScript"><!--
Start = new Date();
bubbleSort();
Stop = new Date();
startTime = Start.getTime();
stopTime = Stop.getTime();
resultTime = stopTime - startTime;
writeNumber();
document.write("<br>かかった時間: " + resultTime + "msec<br>");
// --></script>
</body>
</html>
```

MergeSort.c[前編1]

```
#include<stdio.h>

int num = 30000;
void mergeSort(int random[], int start, int end) {
  int middle, i, j, k;
  int temp[num];
  if (start >= end) {
    return;
  }
  middle = (start + end) / 2;
  mergeSort(random, start, middle);
  mergeSort(random, middle + 1, end);

  for (i = start; i <= middle; i++) {
    temp[i] = random[i];
  }
  for (i = middle + 1, j = end; i <= end; i++, j--) {
    temp[i] = random[j];
  }
  i = start;
  j = end;
```

並べ替えをする
数の個数

MergeSort.c[前編2]

```
#include<stdio.h>

int num = 30000;
void mergeSort(int random[], int start, int end) {
  int middle, i, j, k;
  int temp[num];
  if (start >= end) {
    return;
  }
  middle = (start + end) / 2;
  mergeSort(random, start, middle);
  mergeSort(random, middle + 1, end);

  for (i = start; i <= middle; i++) {
    temp[i] = random[i];
  }
  for (i = middle + 1, j = end; i <= end; i++, j--) {
    temp[i] = random[j];
  }
  i = start;
  j = end;
```

併合ソートの処理部分(前半)

MergeSort.c[中編1]

```
  for (k = start; k < end; k++) {
    if (temp[i] <= temp[j]) {
      random[k] = temp[i++];
    } else {
      random[k] = temp[j--];
    }
  }
}
int main() {
  int i, j, temp;
  int random[num];
  for (i = 0; i < num; i++) {
    random[i] = rand();
  }
  FILE *randFile = fopen("MergeRandomNum.txt", "w");

  for (i = 0; i < num; i++) {
    fprintf(randFile, "%d¥n", random[i]);
  }
  fclose(randFile);
```

併合ソートの処理部分(後半)

MergeSort.c[中編2]

```
  for (k = start; k < end; k++) {
    if (temp[i] <= temp[j]) {
      random[k] = temp[i++];
    } else {
      random[k] = temp[j--];
    }
  }
}
int main() {
  int i, j, temp;
  int random[num];
  for (i = 0; i < num; i++) {
    random[i] = rand();
  }
  FILE *randFile = fopen("MergeRandomNum.txt", "w");

  for (i = 0; i < num; i++) {
    fprintf(randFile, "%d¥n", random[i]);
  }
  fclose(randFile);
```

並べ替えをする数を
自動的に作成

MergeSort.c[中編3]

```
  for (k = start; k < end; k++) {
    if (temp[i] <= temp[j]) {
      random[k] = temp[i++];
    } else {
      random[k] = temp[j--];
    }
  }
}
int main() {
  int i, j, te
  int rand
  for (i = 0
    random[i] = rand();
  }
  FILE *randFile = fopen("MergeRandomNum.txt", "w");

  for (i = 0; i < num; i++) {
    fprintf(randFile, "%d¥n", random[i]);
  }
  fclose(randFile);
```

自動的に作った数を
「MergeRandomNum.txt」ファイルに書き出し

MergeSort.c[後編]

```
  int start, end, procTime;
  start = clock();
  mergeSort(random, 0, num - 1);
  end = clock();

  procTime = end - start;
  printf("Time(Merge Sort of %d Numbers): %f second¥n", num,
          (double) procTime / CLOCKS_PER_SEC);
  FILE *mergeFile = fopen("MergeSortNum.txt", "w");
  for (i = 0; i < num; i++) {
    fprintf(mergeFile, "%d¥n", random[i]);
  }
  fclose(mergeFile);
}
```

数を並べ替えた結果を
「MergeSortNum.txt」ファイルに書き出し

MergeSort.html[前編1]

```
<html>
<head>
<title>併合ソート</title>
<script language="Jav
NUM = 5000;
random = new Array();
for (i = 0; i <= NUM; i++) {
random[i] = Math.floor(NUM * Math.random());
}
temp = new Array();
function mergeSort(start,end) {
    if (start >= end) return;
        var middle = Math.floor((start + end) / 2);
        mergeSort(start, middle);
        mergeSort(middle + 1, end);
        var p = 0;
        for (i = start; i <= middle; i++) temp[p++] = random[i];
            var i = middle + 1;
            var j = 0;
            var k = start;
```

並べ替えをする数の個数

MergeSort.html[前編2]

```
<html>
<head>
<title>併合ソート</title>
<script language="JavaScript"><!--
NUM = 5000;
random = new Array();
for (i = 0; i <= NUM; i++) {
random[i] = Math.floor(NUM * Math.random());
}
temp = new Array();
function mergeSort(start,end) {
    if (start >= end) return;
        var middle = Math.floor((start + end) / 2);
        mergeSort(start, middle);
        mergeSort(middle + 1, end);
        var p = 0;
        for (i = start; i <= middle; i++) temp[p++] = random[i];
            var i = middle + 1;
            var j = 0;
            var k = start;
```

並べ替えをする数を
自動的に作成

MergeSort.html[前編3]

```
<html>
<head>
<title>併合ソート</title>
<script language="JavaScript"><!--
NUM = 5000;
random = new Array();
for (i = 0; i <= NUM; i++) {
random[i] = Math.floor(NUM * Math.random());
}
temp = new Array();
function mergeSort(start,end) {
    if (start >= end) return;
        var middle = Math.floor((start + end) / 2);
        mergeSort(start, middle);
        mergeSort(middle + 1, end);
        var p = 0;
        for (i = start; i <= middle; i++) temp[p++] = random[i];
            var i = middle + 1;
            var j = 0;
            var k = start;
```

併合ソートの処理部分(前半)

MergeSort.html[中編1]

```
            while ((i <= end) && (j < p)) {
                if (temp[j] <= random[i])  {
                    random[k++] = temp[j++];
                } else {
                    random[k++] = random[i++];
                }
            }
            while (j < p) {
                random[k++] = temp[j++];
            }
}
function writeNumber() {
    for (i = 0; i < random.length; i++) {
        document.write(random[i]," ");
    }
    document.write("<br>");
}
// --></script>
</head>
```

併合ソートの処理部分(後半)

MergeSort.html[中編2]

```
            while ((i <= end) && (j < p)) {
                if (temp[j] <= random[i])  {
                    random[k++] = temp[j++];
                } else {
                    random[k++] = random[i++];
                }
            }
            while (j < p) {
                random[k++] = temp[j+
            }
}
function writeNumber() {
    for (i = 0; i < random.length; i++) {
        document.write(random[i]," ");
    }
    document.write("<br>");
}
// --></script>
</head>
```

並べ替えた数をブラウザに
表示する部分

MergeSort.html[後編]

```
<body>
<h1>併合ソート</h1>
<h2>並び替え前</h2>
<script langauge="JavaScript"><!--
writeNumber();
// --></script>

<h2>並び替え後</h2>
<script langauge="JavaScript"><!--
Start = new Date();
mergeSort(0,random.length-1);
Stop = new Date();
startTime = Start.getTime();
stopTime = Stop.getTime();
resultTime = stopTime - startTime;
writeNumber();
document.write("<br>かかった時間: " + resultTime + "msec<br>");
// --></script>
</body>
</html>
```

ファイルの役割

- BubbleSort.c
  - 「BubbleRandomNum.txt」に、並び替え前の数を保存
  - 「BubbleSortNum.txt」に、並び替え後の数を保存
- BubbleSort.html
  - 並び替え前と後の数をブラウザに表示
- MergeSort.c
  - 「MergeRandomNum.txt」に、並び替え前の数を保存
  - 「MergeSortNum.txt」に、並び替え後の数を保存
- MergeSort.html
  - 並び替え前と後の数をブラウザに表示

BubbleSort.c, MergeSort.c[1]

- コンパイルして機械語のファイルを作成
  1. Finder→「アプリケーション」→「ユーティリティ」→「ターミナル」をダブルクリック
  2. それぞれのファイルをコンパイル
     - BubbleSort.cの場合:
       gcc -o BubbleSort BubbleSort.c
       と入力し、「Return」キーを押す
     - MergeSort.cの場合:
       gcc -o MergeSort MergeSort.c
       と入力し、「Return」キーを押す

BubbleSort.c, MergeSort.c[2]

3. Finderで、「BubbleSort」ファイルと「MergeSort」ファイルができていることを確認
   - BubbleSort: BubbleSort.cを機械語に翻訳したファイル
   - MergeSort: MergeSort.cを機械語に翻訳したファイル

BubbleSort.c, MergeSort.c[2]

- プログラミング言語が機械語に翻訳されているかを確認
  - BubbleSort.cをJeditで開いてみる
  - BubbleSortをJeditで開いてみる
  - MergeSort.cをJeditで開いてみる
  - MergeSortをJeditで開いてみる

➢ gcc: C言語のプログラムを機械語に翻訳するためのコンパイラ
  ✓ 「gcc -o ファイル1 ファイル2」で、「ファイル2のプログラムを機械語に翻訳し、ファイルに保存する」という命令

BubbleSort.c, MergeSort.c[3]

- プログラムを実行
  - BubbleSort.cの場合: ターミナルで「./BubbleSort」と入力し、「Return」キーを押す
  - MergeSort.cの場合: ターミナルで「./MergeSort」と入力し、「Return」キーを押す

- BubbleRandomNum.txtとBubbleSortNum.txtを開き、数が並び替えされているかどうかを確認
- MergeRandomNum.txtとMergeSortNum.txtを開き、数が並び替えされているかどうかを確認
- 「Time: かかった時間 second」とターミナル上に表示されるので、かかった時間を比較

※かかった時間の単位は秒

BubbleSort.html, MergeSort.html

- BubbleSort.htmlとMergeSort.htmlをWebブラウザで表示

  - ブラウザで表示された数が、「並び替え前」と「並び替え後」で並び替えられているかどうかを確認
  - 「かかった時間msec」(ブラウザの一番下に表示)ので、BubbleSort.htmlとMergeSort.htmlでかかった時間を比較
  - BubbleSort.cとBubbleSort.htmlでかかった時間を比較
    (※BubbleSort.cは数が30000個、BubbleSort.htmlは数が5000個)
  - MergeSort.cとMergeSort.htmlでかかった時間を比較
    (※MergeSort.cは数が30000個、MergeSort.htmlは数が5000個)

おまけの演習

- 配ったカードを、順番をバラバラにして…
  1. バブルソートで昇順に並べ替え(練習)
  2. バブルソートで昇順に並べ替え、かかった時間を計測(本番)
  3. 自分のやりやすい方法で昇順に並べ替え、かかった時間を計測
  4. 2. と3. の時間を比較(どちらが時間がかかっているか??)

補講と期末試験

- 補講: 1月21日(木) 3限 24101教室
- 期末試験: 1月26日(火) 1限 24101教室
  - 時間: 60分
  - 持ち込み: すべて不可
  - 内容: 後期の講義内容すべて
    - 用語の意味の選択・説明
    - 各種概念に関しての説明
    - 回路図・真理値表
    - etc.