インフォマティクス１

プログラミング

結縁（情報システム学）

大学の情報学部でプログラミングの何を学ぶ？

プログラミング＝コーディングではない

プログラミング ⊃ コーディング

普通のプログラムコードはそのうちに小中学生でも書けるようになる (数学と同じ道)

大学ではもっと面白いことを学ぼう！

正しく効率的に動作するプログラムの概念
新しいアイデアを素直に表現するプログラム
プログラミングの方法論　etc….

では大学の情報学部でプログラミングの何を学ぶ？

- プログラムとは何か？
- プログラムの数理的モデル
- 数理的理解のための方法
- プログラムの表現とその解析
- プログラムの正しさ

これらの基礎的な知識に基づいてコンピュータを研究し、利用する

情報学 （情報工学、コンピュータ科学）

内容

計算の方法

アルゴリズム

計算の表現方法

プログラムとプログラミング言語

計算の方法

二百十日は何月何日？

二百十日（にひゃくとおか）は、雑節のひとつで、立春を起算日として210日目（立春の209日後の日）である。日付ではおよそ9月1日ごろである。台風の多い日もしくは風の強い日といわれるが、必ずしも事実ではない。(Wikipedia)

2017年立春＝2月4日　209日後　⇒2月213日

2月213日　⇒3月185日　⇒4月154日
⇒5月124日　⇒6月93日　⇒7月63日
⇒8月32日　**⇒9月1日**

日付計算の方法

特定の日のn日後は何月何日かを計算する方法

入力：起点となる日付(a月b日)、n

出力：n日後の日付(c月d日)

mon[x] : x月の日数

Step 1　b を b + n でおきかえる。

Step 2　b>mon[a]のとき step 3 へ。b<=mon[a]ならば　Step 4へ

Step 3　a = 12 ならば a = 1 とする。
a < 12 ならば a を1つ増やし、
b を b + n – mon[a] で置き換えて、Step 2 にもどる

Step 4　b <= mon[a] ならば c = a, d = b として終了

階乗

$$n! = n \times (n-1) \times \cdots \times 2 \times 1$$

6!の計算：

6×5= 30

30×4=120

120×3=360

360×2=720

720×1 = 720

階乗の計算方法

入力：n　　出力：n！

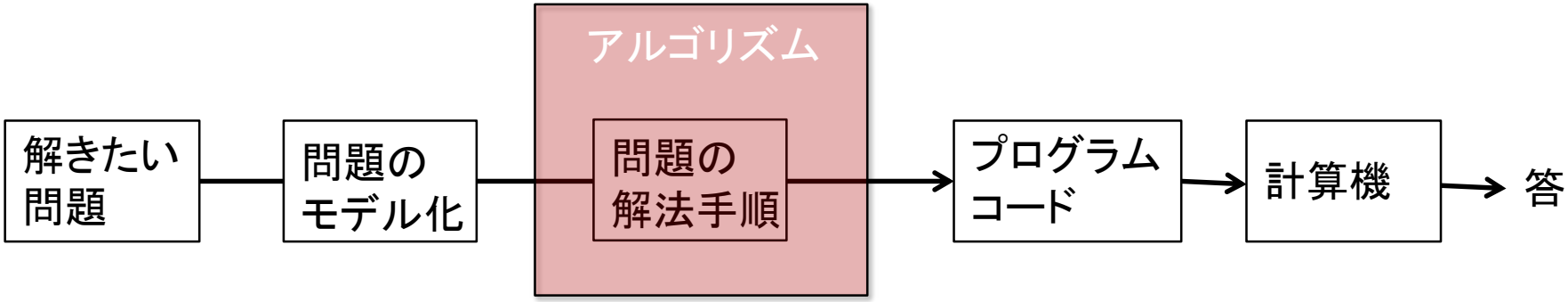
Step: r ← 1

Step2: n = 1 ならば Step 4 へ

Step 3: r ← r × n
　　n ← n − 1

Step 4: r を出力

アルゴリズム



1. 解きたい問題　：　2次方程式の解　$ax^2 + bx + c = 0$

2. 問題の解析・モデル化　：　解の公式　$x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

3. 解法の手順　：　解の公式の計算手順

判定式をチェックして、正ならば平方根を計算して、
-b と +/- して、2aで割る

フィボナッチ数列

0,1,1,2,3,5,8,13,21,34,55,…

直前の2つの数の和を次の数とする

フィボナッチ数列の計算の仕方？

Fib(n): フィボナッチ数列のn番目の数を計算する

Fib(1) = 0, Fib(2) = 1
n ≧3のとき、Fib(n)=Fib(n-1)+Fib(n-2)

どうやって計算する？

フィボナッチ数列の計算

n≧3のとき、n番目のフィボナッチ数はn-1番目とn-2番目の和だから
n番目のフィボナッチ数を fib(n) とすると、fib(n-1)とfib(n-2)を計算して
加えればよい。

n-1≧3ならば、
fib(n-1)を計算するには、fib(n-2)とfib(n-3)を計算して加算

これを繰り返していけばfib(n)が求められる。

フィボナッチ数列の計算（1）

$$
\begin{aligned}
fib(6) &= fib(5) + fib(4) \\
&= (fib(4) + fib(3)) + (fib(3) + fib(2)) \\
&= ((fib(3) + fib(2)) + (fib(2) + fib(1)) + (((fib(2) + fib(1)) + fib(2)) \\
&= (((fib(2) + fib(1)) + fib(2)) + ((fib(2) + fib(1)) + (((fib(2) + fib(1)) + fib(2)) \\
&= (((1 + 0) + 1) + (1 + 0) + ((1 + 0) + 1) \\
&= 5
\end{aligned}
$$

重複した計算が何度も行われる！

フィボナッチ数列の計算(2)

fib(n)の計算には、fib(n-1)とfib(n-2)があればよい

メモリを2つ用意して2個前の値を覚えておけば重複計算は不要

Step 1: x ← 0, y ← 1 (* xは2つ手前、yは1つ手前 *)

Step 2: n=1 のとき 0を返し, n = 2 のとき 1を返す。
n ≧ 3 のとき i ← 3 として Step 3へ

Step 3: n = i ならば x + y を返す。
そうでない場合は、Step 4へ

Step 4: t ← y, y ← x + y, x ← t
i を1つ増やして Step 3 へ

フィボナッチ数列の計算(2)

fib(6)

i	3	4	5
x	1	1	2
y	1	2	3

fib(6) = 5

重複した計算はない

計算量

入力に対してどの程度の計算が必要か？

fib(N)に対して、方法（1）では、$c \times (\frac{1+\sqrt{5}}{2})^{N-1}$回程度の計算が必要。

方法（2）では、$c \times N$回程度の計算が必要

方法（1）では、fib(N)の計算回数を C(n)として、

C(n) = C(n-1)+C(n-2)+5 回の計算が必要

Nが大きくなってくると絶対的な差となる

実際の計算(Cプログラム)

```
#include <stdio.h>

long fib01(long n) {
  if (n == 1) { return 0; }
  else if (n == 2) { return 1; }
  else { return fib01(n-2)+fib01(n-1); }
}

int main() {
   long n;

   printf("Enter n for Fib(n) ");
   scanf("%ld",&n);
   printf("%ld¥n", fib01(n));
}
```

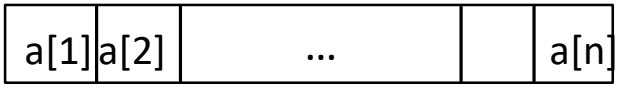
```
#include <stdio.h>

long fib02(long n) {
  long i;
  long x,y,t;
  if (n == 1) { return 0; }
  else if (n == 2) { return 1; }
  else { x = 0; y = 1;
    for(i=3;i<n;i++) {t=y;y=x+y;x=t;}
    return x+y;
  }
}

int main() {
   long n;
   printf("Enter n for Fib(n) ");
   scanf("%ld",&n);
   printf("%ld¥n", fib02(n));
}
```

ソート（１）：バブルソート

N個のデータを小さい順に並べる



方法（１）：

a[1]とa[2]を比べて、小さい方をa[1]に入れ、大きい方をa[2]に入れる。
a[2]とa[3]を比べて、小さい方をa[2]に入れ、大きい方をa[3]に入れる。
　　これをa[n-1],a[n]まで繰り返す。　a[n]に最大の要素が格納
a[1]からa[n-1]に同じことを繰り返す。
　　これを繰り返して a[1],a[2] まで繰り返す。

8,2,4,6,5 ⇒ 2,8,4,6,5 ⇒ 2,4,8,6,5 ⇒ 2,4,6,8,5 ⇒ 2,4,6,5,8

⇒ 2,4,6,5,8 ⇒ 2,4,6,5,8 ⇒ 2,4,6,5,8 ⇒ 2,4,5,6,8

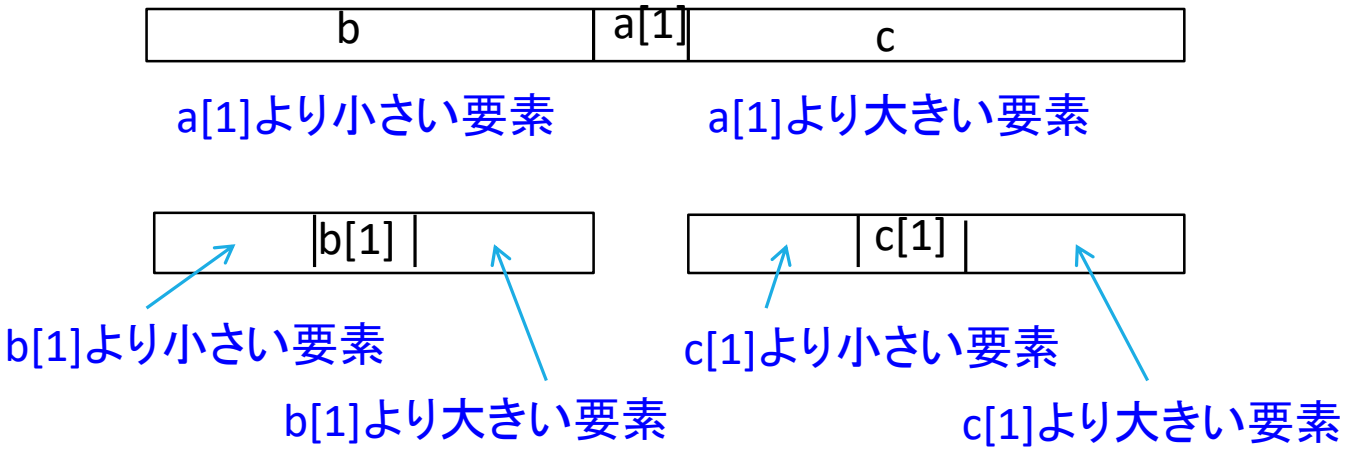
⇒ 2,4,5,6,8 ⇒ 2,4,5,6,8 ⇒ 2,4,5,6,8

⇒ 2,4,5,6,8 ⇒ 2,4,5,6,8

N^2 の計算量

ソート（2） クイックソート

N個のデータを小さい順に並べる



それぞれの要素が1つになるまで繰り返す

うまく分割できれば長さが半分ずつになる

ソート（2） クイックソート

8,3,6,1,9,4,2,5 ⇒ 3,6,1,4,2,5,8,9

⇒ 3,6,1,4,2,5,8,9 ⇒ 1,2,3,6,4,5,8,9

⇒ 1,2,3,6,4,5,8,9 ⇒ 1,2,3,4,5,6,8,9

⇒ 1,2,3,4,5,6,8,9 ⇒ 1,2,3,4,5,6,8,9

ピボット（赤の数字）の選び方によって異なる
最悪は N^2

プログラムの実行例

バブルソートのプログラム

10000個の整数を乱数で発生させて実行

約30秒

```
Sorting...done
29.798u 0.059s 0:29.95 99.6%      0+0k 0+0io 0pf+0w
```

クイックソートのプログラム

100000個の整数を乱数で発生させて実行

約0.2秒

```
Sorting...done
0.190u 0.003s 0:00.19 100.0%      0+0k 0+0io 0pf+0w
```

素数の計算

1からNまでの素数を列挙する

Step1 : i ← 1

Step2: j ← 1
Step3: j < i のとき、Step4へ　j = i のとき i を出力
Step4 : i を j で割り切れる場合 step 5へ。
　　割り切れないときは、j ← j +1 としてstep 3へ
Step5: i ← i + 1, i ≦ N ならば Step 2へ
　　i > N で終了

素数のチェック

xが素数かどうかチェックするのに割ってみる数は、$\lfloor \sqrt{x} \rfloor$までで十分

ところで

素数は無限に存在する(wikipedia)

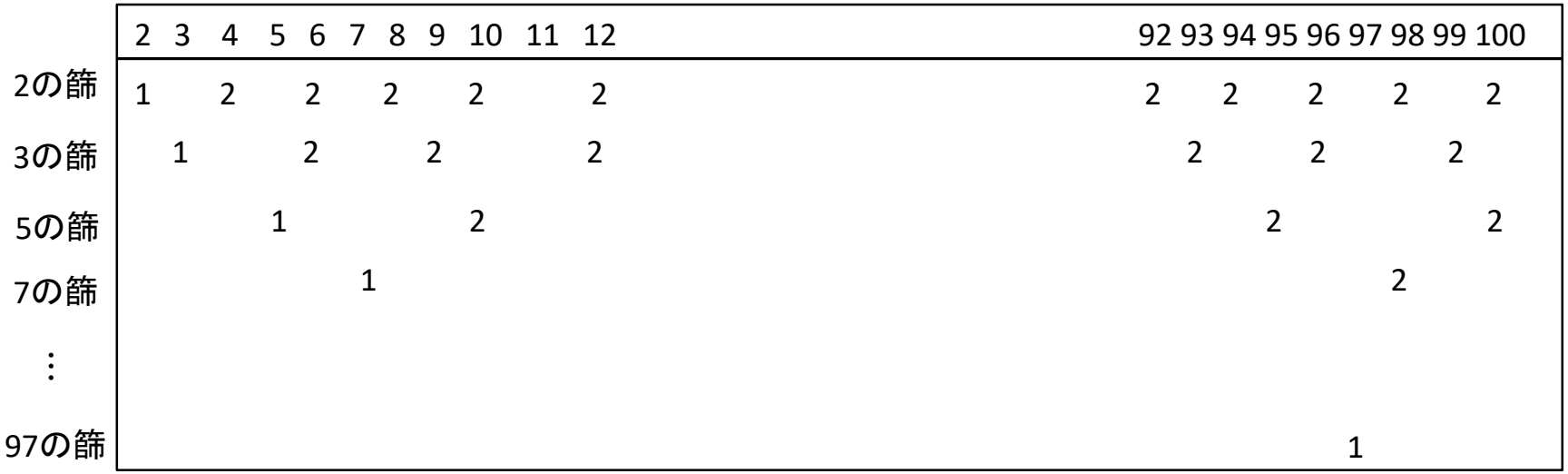
> 素数の個数が有限と仮定し、p1, … pn が素数の全てとする。その積 P = p1 × … × pn に 1 を加えた数 P + 1 は、p1, …, pn のいずれでも割り切れないので、素数でなければならない。しかし、これは p1, …, pn が素数の全てであるという仮定に反する。よって、仮定が誤りであり、素数は無限に存在する。

https://primes.utm.edu/largest.html $2^{77,232,917} - 1$ 23249425桁

現在知られている最大の素数

昨年 $2^{74207281} - 1$ 22338617桁

エラトステネスの篩（ふるい）



	2	3	4	5	6	7	8	9	10	11	12	…	92	93	94	95	96	97	98	99	100
2の篩	1		2		2		2		2		2		2		2		2		2		2
3の篩		1			2			2			2			2			2			2	
5の篩				1					2							2					2
7の篩						1													2		
⋮																					
97の篩																		1			

2がついていない数から始めて、倍数に2をつけていく。
最後に1がついている数を表示する

実行結果

2から1000000まで素数がいくつあるか数える

定義どおり

78498 primes
115.034u 0.950s 1:59.79 96.8%　　0+0k 0+0io 0pf+0w

115秒

$\lfloor \sqrt{x} \rfloor$ までチェックする

78498 primes
0.238u 0.004s 0:00.24 95.8% 0+0k 0+0io 0pf+0w

0.2秒

エラトステネスの篩

78498 primes
0.022u 0.003s 0:00.02 100.0%　　0+0k 0+0io 0pf+0w

0.22秒

計算の表現方法

命令型 (Imparative)

行うべき操作を命令として順番に記述
条件分岐と繰返しを一列に並べて記述

宣言型 (Declarative)

何を入力として何を求めたいかを書く

(a) 関数型

y=f(x)の形で書く　x: 入力　y: 出力

(b) 論理型

R(x,y)が真になるような述語を定義 x: 入力　y:出力

命令型

変数への代入　　$x \leftarrow a$　　$x \leftarrow exp$

自然数の操作(四則演算)　　0　　1

$e_1 + e_2$　　$e_1 \times e_2$　　$e_1 - e_2$　　e_1/e_2

逐次実行　　$Cmd_1; Cmd_2$　　**skip**

条件分岐　　**if** $Cond$ **then** Cmd_1 **else** Cmd_2

条件付きループ構造 (whileループ)

while $Cond$ **do** $Cmd_1; \cdots, Cmd_n$ **done**

命令型の例

$$\textbf{if } a > b \textbf{ then } t \leftarrow a; a \leftarrow b; b \leftarrow t \textbf{ else skip};$$

$$\textbf{while } b \neq 0 \textbf{ do } t \leftarrow a; a \leftarrow b; t \leftarrow t;$$

$$\quad \textbf{while } b \geq a \textbf{ do } b \leftarrow b - a; q \leftarrow q + 1 \textbf{ done};$$

$$\textbf{done};$$

何を計算するプログラム？

ユークリッドの互除法

a,b の最大公約数をaに計算

宣言型:関数型

プログラムは関数とみなすことが自然 [Backus]

$$y = f(\vec{x}) \qquad \vec{x} \text{ 入力} \qquad y \text{ 出力}$$

求めたい関数を数学的に記述すればプログラムに変換できる(はず)

$$\begin{aligned} fib(n) \quad = \quad & \text{if } n = 1 \text{ then } 0 \\ & \text{else if } n = 2 \text{ then } 1 \\ & \text{else } fib(n-1) + fib(n-2) \end{aligned}$$

必ずしも効率的なプログラムが生成できるとは限らない

関数の型を詳細に解析することで実行前にかなりの誤りを発見できる

宣言型:論理型

入出力はある条件を満たす関係である

$R(x_1, \cdots, x_n, y)$　　$(x_1, \cdots, x_n)$ 入力　y 出力

入力値 $v_1, \cdots, v_n$ に対する出力値が u のとき

$R(v_1, \cdots, v_n, u)$ が真となるように R を定義

関係を定義してプログラムの振舞いを定める

一階述語論理を利用する

導出原理(resolution principle)を用いた効率的な計算

理論的には関係なので入出力を限定する必要はない:
→計算の方向性はあまり関係ない

加算の論理的な定義

- X に 0 を加算すると(無条件に)Xである。
- X に Y を加算してZになるならば、Xに(Y+1)を加算するとZ+1である。

```
Add(X,0,X).
Add(X,Y+1,Z+1) :- Add(X,Y,Z).
```

A1,…,AnならばBを　B:-A1,…,An　と書く

Add(2,3,X)の計算

2+3=Xであるためには、2+2=X', X=X'+1である。
2+2=X'であるためには、2+1=X'', X'=X''+1である。
2+1=X''であるためには、2+0=X''',X''=X'''+1である。
2+0=X'''であるので、X'''=2である。これからX=((2+1)+1)+1=5

プログラムとプログラミング言語

プログラム：計算の方法を計算機に伝える記述

計算方法と計算機に応じた記述

命令型、関数型、論理型を組み合わせて書く

どの書き方が中心的かに応じて、
関数型言語、論理型言語などと呼ばれる。

命令型は普通なので特に何も言わない

記述：　構文（シンタックス：Syntax）
動作：　意味（セマンティックス：Semantics）
非常に単純には入出力対応のこと

関数型言語による記述

関数型言語　Ocaml　による最大公約数を求める記述

```
let rec gcd a b =
    if b = 0 then a else gcd b (a mod b);;

Printf.printf "gcd(%d,%d)=%d¥n" 999 148 (gcd 999 148);;
```

関数型言語：Haskell, SML(Standard ML)　など

論理型による記述

prologによる最大公約数の記述

論理式（ホーン節）そのものをプログラムとみなす

```
greatestCommonDiviser(M,N,G):-
    Integer(M),Integer(N),M>0,N>0,gcd(M,N,G)
gcd(M,0,M).
gcd(M,N,G):-mod(M,N,R),gcd(N,R,G).
```

`gcd(M,N,G)` MとNの最大公約数はGである。

```
mod(X,Y,Z):-X>=Y,mod(X-Y,Y,Z).
mod(X,Y,X):-X<Y.
```

`mod(X,Y,Z)` XをYで割った時のあまりはZである。

論理型言語の実行

```
greatestCommonDivser(999,148,G):-
    Integer(999),Integer(148),999>0,148>0,gcd(999,148,G).

gcd(999,148,G):-mod(999,148,R),gcd(148,R,G).

  mod(999,148,R):-999>=148,mod(851,148,R).
  mod(851,148,R):-851>=148,mod(703,148,R).
  mod(703,148,R):-703>=148,mod(555,148,R).
  mod(555,148,R):-555>=148,mod(407,148,R).
  mod(407,148,R):-407>=148,mod(259,148,R).
  mod(259,148,R):-259>=148,mod(111,148,R).
  mod(111,148,111):-111<148.

  gcd(148,111,G):-mod(148,111,R),gcd(148,R,G).
```

```
mod(148,111,R):-148>=111,mod(37,111,R).
mod(37,111,37):-37<111.

gcd(148,37,G):-mod(148,37,R),gcd(37,R,G).

  mod(148,37,R):-148>=37,mod(111,37,R).
  mod(111,37,R):-111>=37,mod(74,37,R).
  mod(74,37,R):-74>=37,mod(37,37,R).
  mod(37,37,R):-37>=37,mod(0,37,R).
  mod(0,37,0):-0<37.

  gcd(37,0,37).
gcd(148,37,37).
gcd(148,111,37).
gcd(999,148,37).
```

gcd(999,148,37) が真
999 と 148 の最大公約数は 37

プログラムモデルに基づく構文

オブジェクト指向　(Object Oriented)

すべての要素：オブジェクト

ClassとInstance

Class: メソッドを定義　　Instance: Classで定義されるメソッドを持つ実体

オブジェクト指向の考え方はほぼすべての言語に導入可能

オブジェクト指向モデルを積極的に取り込んだ言語：Smalltalk, Java, C++ など

まとめ

- プログラミング：

  問題の同定→モデル化→解法（アルゴリズム）→コーディング→実行

  コーディング（プログラムを書く）は最後の作業

  モデル化のタイプに応じてさまざまなプログラミング言語が存在

- 正しく動作するプログラムを書くことはそう簡単でない

  プログラムの検証は基本的に決定不能

- プログラムを正しく動くように維持することは難しい

  パッチ（更新プログラム）の発行　：　セキュリティに対する要求