オブジェクト指向プログラミング

第７回

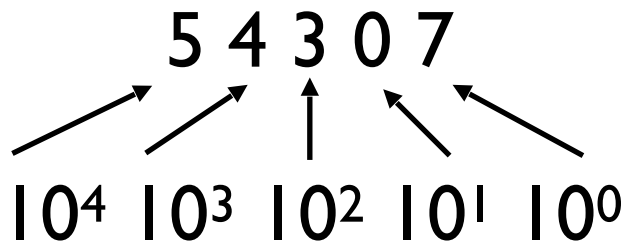
箕原辰夫

整数のアルゴリズムと演算

- n進数との変換
- 約数や倍数
- 階差方程式
- 整数のビット演算子

n進数と位取り法

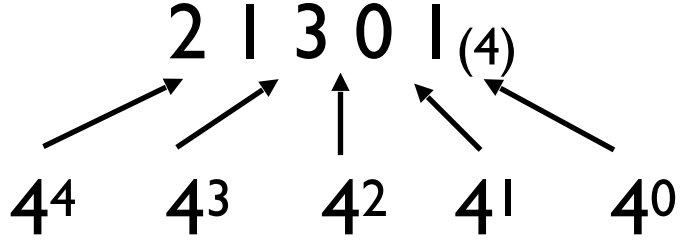
- それぞれの桁が基数のべき乗との掛け算である。



5 x 10000
4 x 1000
3 x 100
0 x 10
7 x 1

n進数の値を求める

- 各桁をnのべき乗数と掛け算し、それらの総和を求める



2 x 256
1 x 64
3 x 16
0 x 4
\+ 1 x 1

$=625_{(10)}$

Question 1:
$12031_{(4)}$

Question 2:
$356_{(7)}$

2進数と16進数

- 2進数を最下位（右側）の桁から４桁ずつ区切って読むと16進数になる。
  - 1110101010010101
  - 11/1010/1001/0101
  - 3　A　9　5

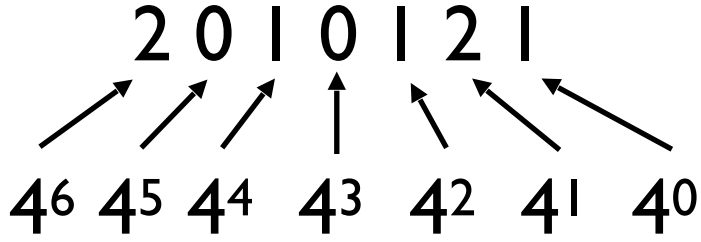
2進数	16進数	2進数	16進数
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

n進数への変換

- 基数のnで割っていく

4)8473
4)2118 ... 1
4) 529 ... 2
4) 132 ... 1
4) 33 ... 0
4) 8 ... 1
4) 2 ... 0
0 ... 2

検算してみる



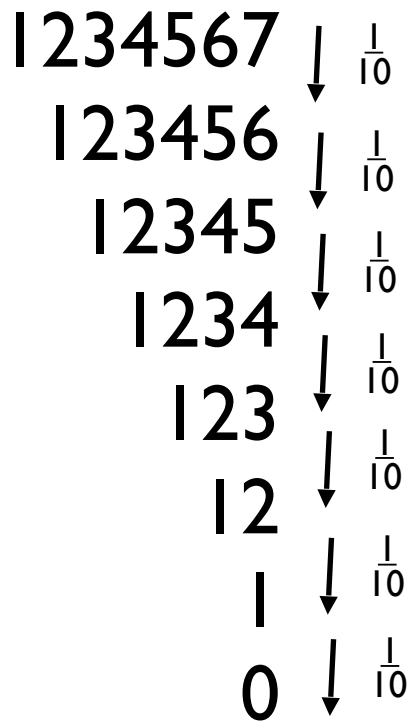
2 x 4096
0 x 1024
1 x 256
0 x 64
1 x 16
2 x 4
1 x 1

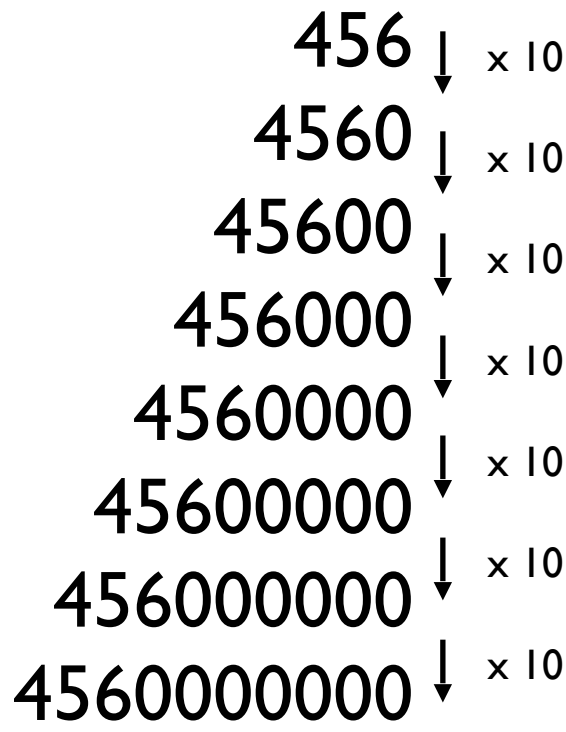
Question 3:
9876を7進数で

n進数の分解

- 基数での整数除算と整数剰余
- 基数で割っていく＋基数で整数剰余を取る

➡ それぞれの桁が求まる





基数と整数剰余・整数除算

- 各桁に分解できる（**div**は、Pythonの場合、// ）
  - 3456 % 10 = 6
  - 3456 **div** 10 % 10 = 5
  - 3456 **div** 10 **div** 10 % 10 = 4
  - 3456 **div** 10 **div** 10 **div** 10 % 10 = 3

- n進数においても同じ
  - 39 % 5 = 4
  - 39 **div** 5 % 5 = 2
  - 39 **div** 5 **div** 5 % 5 = 1

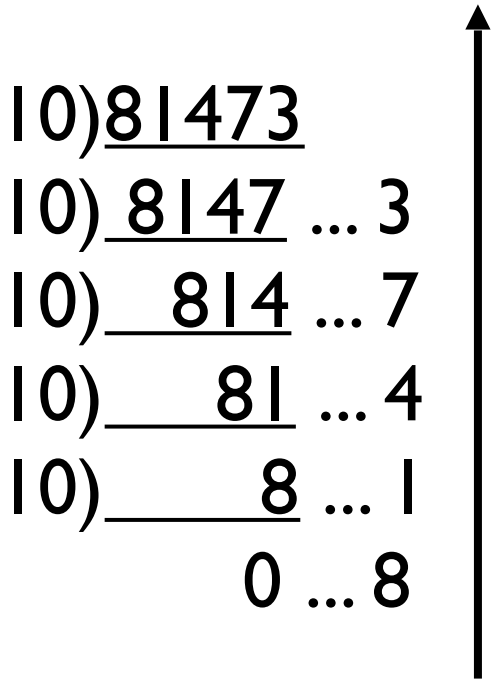
n進数の必要性

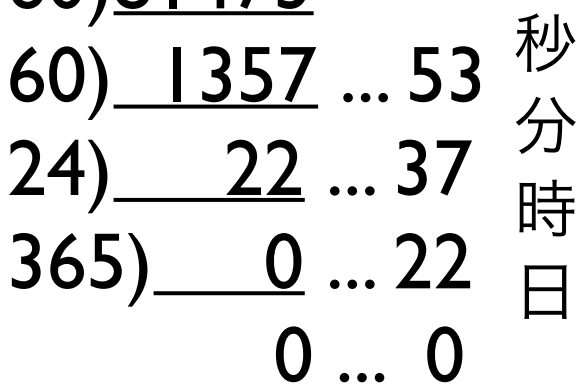
- 2進数、16進数はコンピュータで基本的に使われている
- 7進数は暦の上で曜日（週）との関連性が強い
- 12進数は、時間や月などで用いられている
- 60進数は、時間（分や秒）で用いられている
- 360進数は、角度で用いられている

- n進数はこのように実は広く生活の中に浸透している。ただ、それをn進数として学んでこなかっただけ

各桁への分解

- 基数の10で割っていく　　時分秒の場合は60と24





n進数変換

- n進数を10進数に変換する
  - n進数で表記された各桁をnのべき乗（最下位は0から始まる）と掛け算して、足し合わせる
  - 11進数以上は、アルファベットへの対応が必要

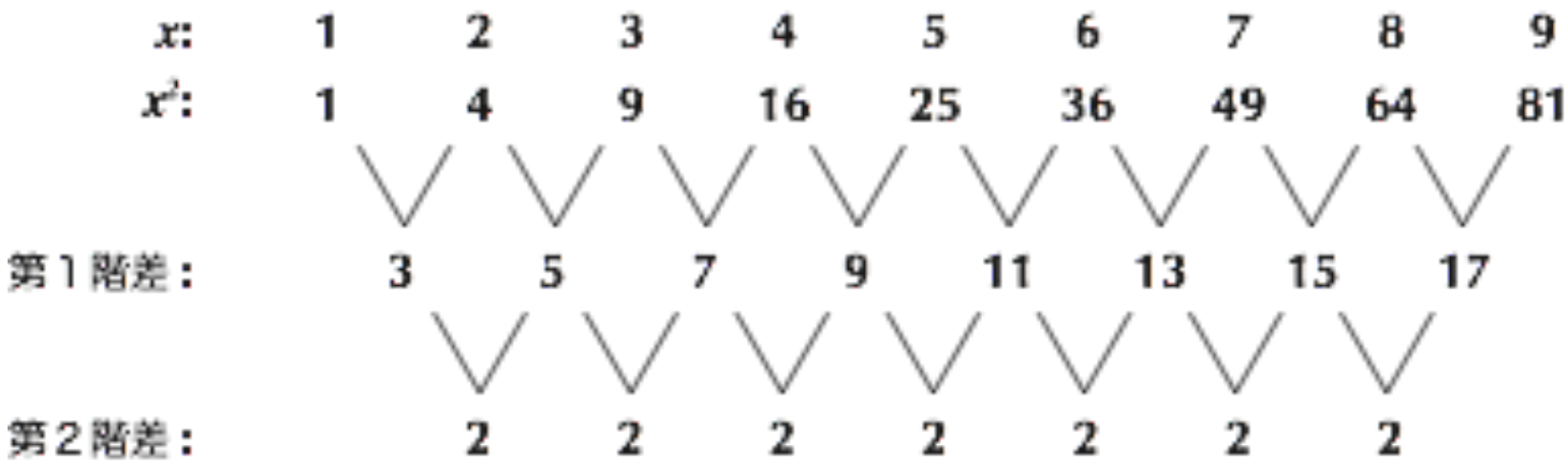
- 10進数をn進数に変換する
  - 数をnで割っていき（整数除算）、余りを下の桁から文字列として足していく
  - 商が0になった時点で終了

約数・倍数を求める

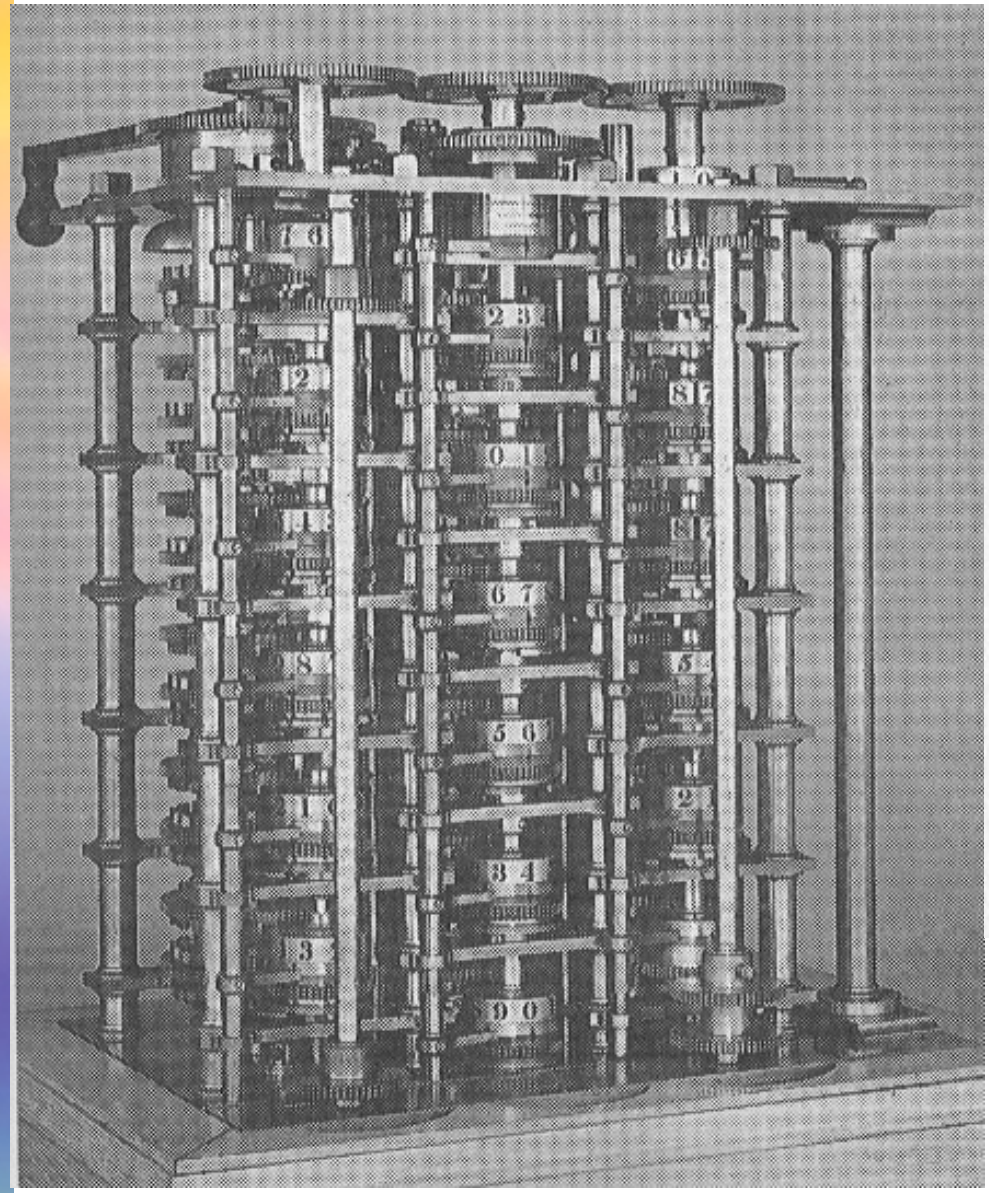
- 約数の個数を求める
  - ➔ 約数が何個あるかを数えるための変数を用意
- 約数の和を求める
  - ➔ 約数の和を求めるための変数を用意
- 倍数の個数を求める
  - ➔ 倍数が何個あるかを数えるための変数を用意
- 素数と完全数
- ピタゴラスの平方数を求める
- 素因数分解

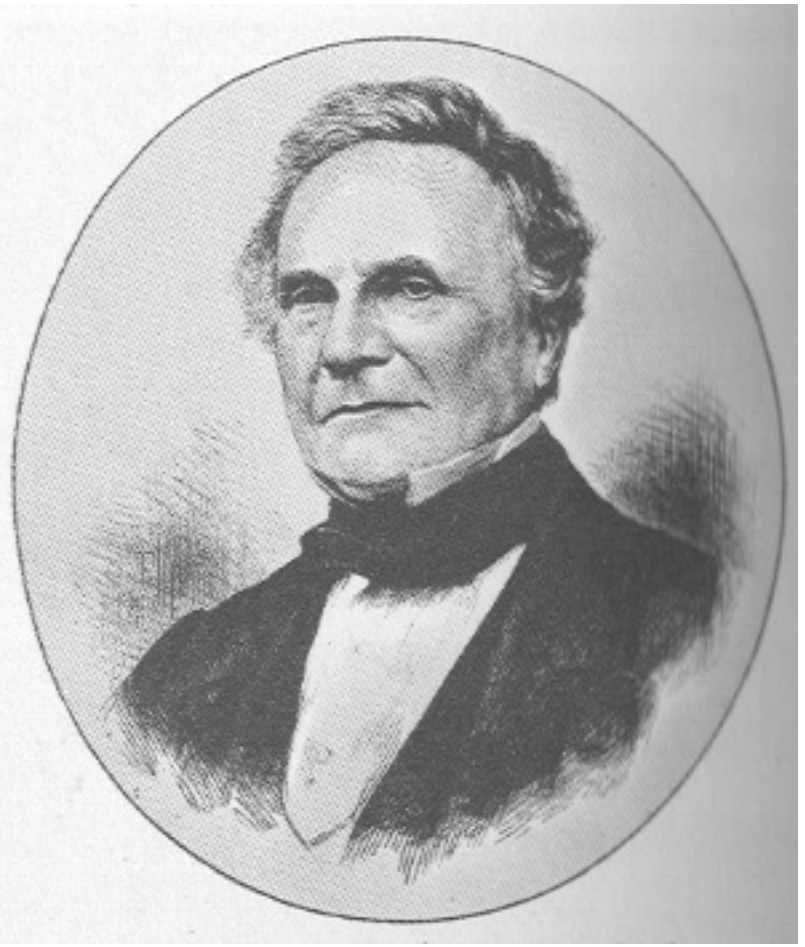
バベッジの階差機関

- ２次多項式に値を代入して計算値を求める
- 掛け算を足し算で計算していく方法



バベッジの階差機関





求め方

- 計算結果を求める変数
  - *result* = *result* + 第１階差；
- 第１階差を求める変数
  - *diff* = *diff* + 第２階差；
- 多項式が$px^2 + qx + r$の場合、
  - 最初の結果は、r
  - 最初の第１階差は、$p + q$
  - 第２階差は、一定で、$p + p$ (=$2p$)

ビット演算

- ~　ビットの反転（単項演算子）
  - 例： ~x
- &　ビットのAND（二項演算子）
  - 例： x & 7
- |　ビットのOR（二項演算子）
  - 例： x | 5
- ^　ビットの排他的論理和（二項演算子）
  - 例： x ^ 0b1101

8 bitで演算

- 0xa3と ~0xa3
  - 0b10100011　各ビットを反転する(Pythonでは-(n+1))

  - 0b01011100
- 0xa3 & 0x7a
  - 0b10100011 0xa3
  - 0b01111010 0x7a
  - 0b00100010 0x22 両方とも1のビットだけ1
- 0xa3 | 0x7a
  - 0b10100011 0xa3
  - 0b01111010 0x7a
  - 0b11111011 0xfb いずれかのビットが1なら1

ビット演算続き

- 0xa3 ^ 0x7a
  - 0b10100011 0xa3
  - 0b01111010 0x7a
  - 0b11011001 0xd9 どちらかのビットが1なら1

A	B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

ビット演算を目に見える形で

- 入力された２つの数をそれぞれ２進数で表示する
- 入力された文字列の２進数を整数に変換するには、int(文字列, base=2)を用いる
- 整数を２進数で表示するには、bin( )関数を用いる

シフト算

- ２のべき乗とのかけ算・割り算になる
  - ➡ >> 右シフト　２のべき乗で割った

例：　12 >> 2　　≡　　12 / / (2*2)

  - ➡ << 左シフト　２のべき乗と掛けた

例：　　3 << 4　　≡　　3 * (2 * 2 * 2 * 2)

format関数とzfill関数

- format関数
  - format(数値, 書式文字列)
    - "b"…２進数
    - "d"…１０進数
    - "o"…８進数
    - "x"…16進数

- 文字列のzfill関数
  - 文字列.zfill(0で埋める桁数)

2進数の表示

- format(式, "b").zfill(桁数)
  - 式の値を、桁数分の0で埋めてから、２進数を表示する

- 表示する有効の下位の桁数を指定したいときは、「値 & 0b1111」などを使う
- 例：format(~(0b101) & 0xff, "b").zfill(8)
  - 下位８桁だけを表示する

補数

- 足してその数になる数の組み合わせ
- １０の補数
  - ➔ 足して１０になる数の組み合わせ

例：　1と9, 2と8, 3と7, 4と6, 5と5

6に対する10の補数は？　　… 4

9の補数　0と9, 1と8, 2と7, 3と6, 4と5

補数

- 補数は引き算の替わりに用いられる
- 基数の補数（2の補数、10の補数）
- 基数-1の補数（１の補数、9の補数）
- 基数の補数 = 基数-1の補数　+　1
- 例：　10000 –　5678 =　？
  - 9999　–　5678　+ 1　=　4321 + 1 = 4322
  - 15678 – 9876 = 5678 + 10000 – 9876
  - = 5678 + 9999 – 9876 + 1 =　5678 + 123 + 1
  - = 5802

２進数では？

- 10010 – 01101 = 18 – 13
  - = 10010 + ~(01101) + 1
  - = 10010 + 10010 + 1
  - = 100100 + 1 = 100101 = 00101 = 5
- 00000 – 01101 = 0 – 13
  - = ~(01101) + 1 = 10010 + 1 = 10011