I111 アルゴリズムとデータ構造
第1回: プログラミングの基礎

担当: 上原 隆平(uehara)

講義概要

- I111アルゴリズムとデータ構造
- 担当: 上原 隆平
- 目的: アルゴリズムの意味と意義を理解

問題を解く手順のことをアルゴリズムといい，計算機内部にデータを蓄える形式のことをデータ構造という．一般に一つの問題に対して，何通りものアルゴリズムとデータ構造の組み合わせが考えられる．それらを計算時間やデータ構造のサイズなどで評価し，状況に応じて最適なアルゴリズムを選択することが必要であるが，単に従来のアルゴリズムを理解し，記憶するだけではなく，アルゴリズム設計の考え方を身につけることが重要である．本科目では，適切な例題を用いて，アルゴリズムの正当性を確認し，その効率の解析により改善の余地があるかどうかを調べることの重要性を認識させ，アルゴリズム設計の基礎を教授する．

講義情報について

- http://www.jaist.ac.jp/~uehara/course/2015/i111/index.html
  - 一度熟読すること
  - 頻繁にチェックすること

- jenzabar.jaist.ac.jp
  - 講義のビデオなどあり

教科書・参考書

- 教科書
  - 浅野，和田，増澤著『アルゴリズム論』オーム社.
  - 上原著『はじめてのアルゴリズム』近代科学社.
- 参考書
  - エイホ，ホップクロフト，ウルマン 著，野崎，野下 訳，『アルゴリズムの設計と解析 I, II』サイエンスライブラリ，サイエンス社.
  - 石畑著『アルゴリズムとデータ構造』岩波書店.
  - Cormen, Leiserson, Rivest, Stein著 “Introduction to Algorithms, 3rd ed.” MIT Press. （浅野他訳:『アルゴリズム・イントロダクション』）

評価方法

- 観点
  - 基礎理論の理解度と応用力
- 方法
  - レポート問題, 期末試験
- 基準
  - レポート問題 30%, 期末試験 70%
  - Web講義受講者は期末試験のみ

http://www.jaist.ac.jp/~uehara/course/2015/ti111/index.html

受講条件

- 条件: なし
  - プログラミング経験があることが望ましい
  - プログラミング言語は何でもよい
    - C, C++, Java, C# (?), Ruby, Python, Scheme, Haskell, …
    - C# は上原の環境で実行できないかも…
    - 上原が「読める」プログラムでない場合は減点対象にする可能性あり（こちらでも努力はします）
  - まったく経験がなければプログラム演習Iを受講しましょう.
    - 本来は「アルゴリズム」はプログラムとは違うので，問題などで配慮はします.

アルゴリズム(algorithm)とは

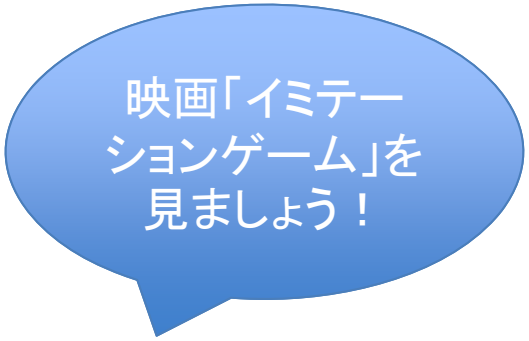
計算機を用いて解ける問題に対する解法を抽象的に記述したもの

- 問題が解けるとは？
  - どんな入力に対しても正しい解が得られる
  - 妥当な時間内に解が得られる
    - 入力サイズの多項式時間で計算ができる
    - 入力サイズの多項式空間（メモリ）で計算できる
- 解けない問題とは
  - 入力によっては非常に長い時間が必要
  - 入力によっては非常に大きなメモリが必要
  - (そもそもプログラムが作れない問題(I216))

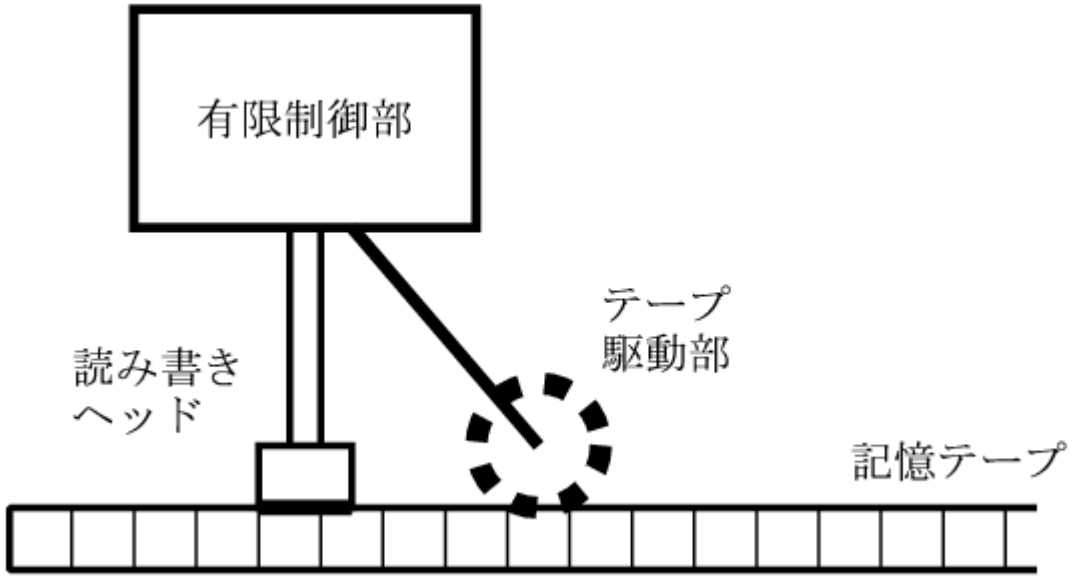
計算モデルの話

そもそもコンピュータってどういう仕組みで動いているの？

- 計算モデルによって，アルゴリズムの記述や効率は違ってくる
  - なにが「基本演算」なのか？
  - どんなデータが記憶できるのか？
    - 自然数，実数（？），画像，音楽データ…?
- いくつかの標準的なモデルがある
  - チューリング機械：かのアラン・チューリングが考案．すべての議論の基礎となっている．
  - RAMモデル：アルゴリズムの話をするときはこれが標準．

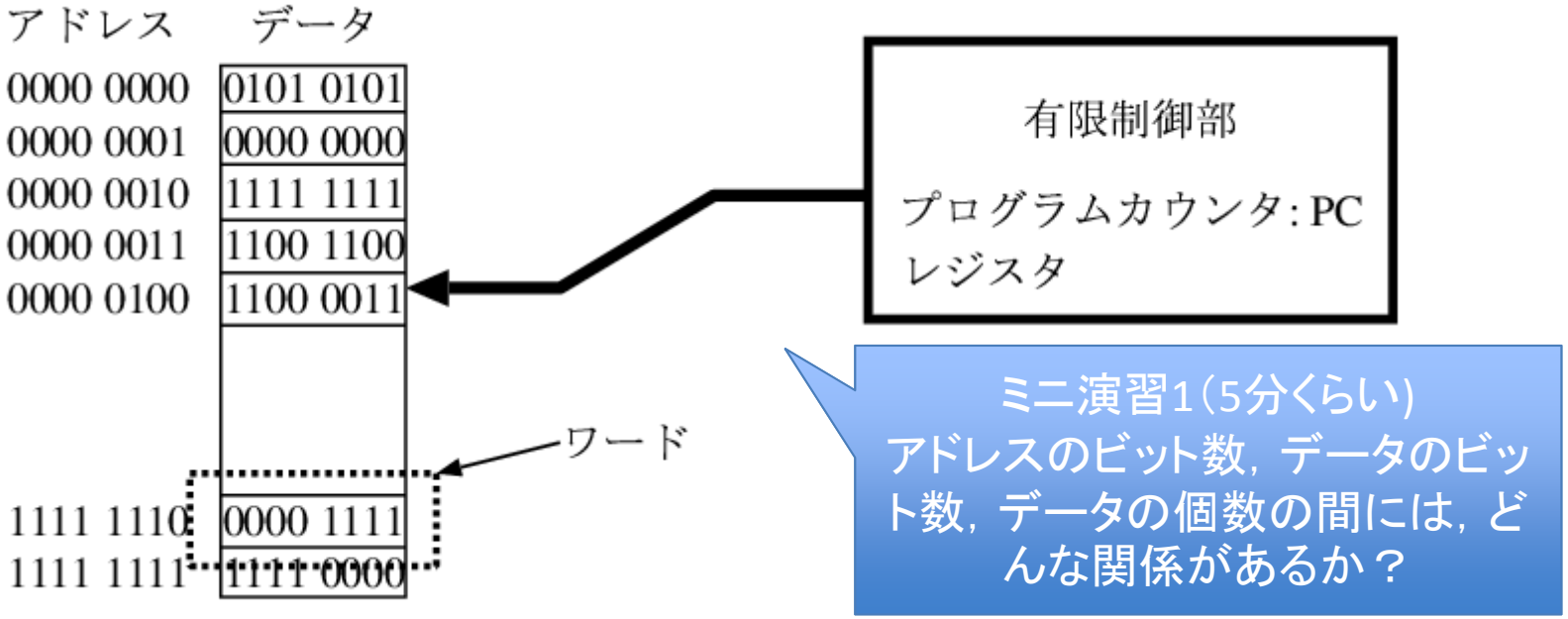


チューリング機械



- 理論的な扱いが楽な，非常に単純なモデル
- 単純すぎて実際にプログラムを作るのは苦行
  - 四則演算もない
  - アルゴリズムの本質が議論しにくい

RAMモデル



- 記憶装置とCPUからなる．(入出力は無視)
- 実際のCPUやメモリと本質的には同じ
- ランダムにデータをアクセスできる（Random Access Memory)
- C言語などでは，こうしたRAMモデルがなんとなく透けて見える体系になっている（ポインタ，配列など）

※詳しくはプログラミング演習Iを受講して下さい

C言語の基礎: main関数の宣言

- 最も簡単な(何もしない)プログラム:

```
main()   /*関数名mainを宣言*/
{        /*mainの始まり*/
}        /*mainの終わり*/
```

  - main関数は必ず定義

C言語の基礎: Hello World

- Hello World とディスプレイに表示

```
#include <stdio.h> /*printfを提供*/

main(){
    printf(“Hello World”);
}
```

命令文

命令文の終わりにセミコロン

C言語の基礎: 数値の表示

- 基本的なやり方: printf(“%d”,数値)
  - 3とディスプレイに出力: printf(“%d”,3)

```
#include <stdio.h> /*printfを提供*/

main(){
  printf(“%d”,3);
}
```

  - %dの意味: 与えられた数値を10進数整数として表示
    - 小数を表示するときは%fを使う

C言語の基礎: 算術式

- 四則演算: 加+ 減- 乗* 除/ 余%

算術式	意味
3+4	3と4を足す
3-1	3から1を引く
3*3	3と3をかける
4/2	4を2で割る
3%2	3を2で割った余り

  - 剰余%以外は整数型(`int`, etc.)でも浮動小数点数型(`float`, `double`, etc.)でも使える

C言語の基礎:算術式の注意点

- 分数はなし: 1/3は1を3で割った結果になる
- 整数/整数は小数部分が切り捨てられる
  - 例: 1/3は0になる。1.0/3は0.3333…
- コンマで区切らない
  - 例: 10,000はダメ。10000と書く。
- 計算順序を制御する括弧: 小括弧のみ
  - 算術式の中で中括弧{}や大括弧[]は使えない
  - 例: {(3+4)*3+4}*6 はダメ。((3+4)*3+4)*6 と書く。
- べき乗の演算子はない

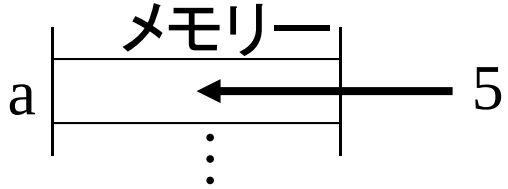
C言語の基礎: 変数

- 変数: 計算結果を蓄える名前付きの“場所”
- 名前のルール
  - アルファベットで始まる(大文字，小文字の他に _ もOK)
  - 2文字目以降にはアルファベットの他に数字が使える
    - それ以外は使えない
  - 大文字と小文字は区別される
    - FFとffとfFとFfは全部別物
  - C言語の予約語(e.g., main, include, return)と一致なし

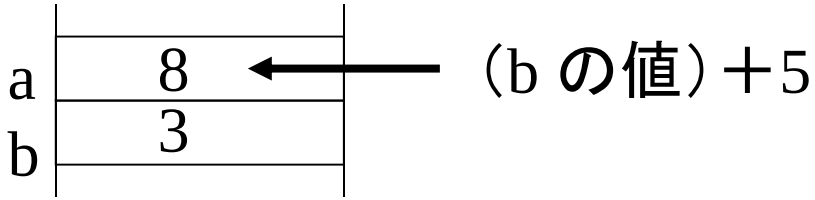
  - 良い例: x, orz, T_T, IE9, projectX, ff4, y2k, JAIST
  - 悪い例: 7th, otachi@jaist, ac.jp, tel#

C言語の基礎: 代入文

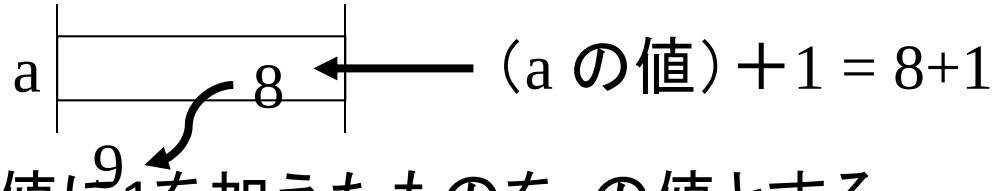
- a=5



  - aという名前の場所に数値5を格納
- a=b+5



  - bという名前の場所に格納されている値(変数bの値)に5を加えた数値をaという名前の場所に格納
- a=a+1



  - 変数aの値に1を加えたものをaの値とする

C言語の基礎: 変数の宣言

- 変数は格納する値の型を指定して予め宣言しておかなければ使えない

良い例

```
main(){
  int a,b;
  a = 5; b = 3;
  printf(“a+b=%d”,a+b);
}
```

変数a, bを宣言(型はint: 整数)

悪い例

```
main(){
  a = 5;
  printf(“%d”,a);
}
```

変数を宣言せずに使っている！

C言語の基礎: まとめ
プログラムの構成

- 最低限main関数がある
- 関数の本体は{と}でくくる
- 命令文(e.g., `printf("%d",3)`)の終わりには;を置く
- 1行に複数の命令が書ける
  - `a=3; y=4; printf("%d",3+4);`
- /*から*/まではコメントで、実行されない
  - 注意: 入れ子にすると・・・ /* … /* … */ … */

C言語の基礎: まとめ
変数名

- アルファベットで始まる: a-z, A-Z, _
- 2文字目以降にはアルファベットの他に数字が使える
- 大文字と小文字は区別される
  - FF と ff と fF と Ff は全部別物
- C言語の予約語と一致なし
  - 予約語: `main`, `include`, `return`など

C言語の基礎: 算術関数の利用

	関数名	数学的表現	関数の型	引数の型
平方根	**sqrt(x)**	$\sqrt{x}$	**double**	**double**
べき乗	**pow(x, y)**	x^y	**double**	**double**
自然対数	**log(x)**	$\log_e x$	**double**	**double**
常用対数	**log10(x)**	$\log_{10} x$	**double**	**double**
指数関数	**exp(x)**	e^x	**double**	**double**

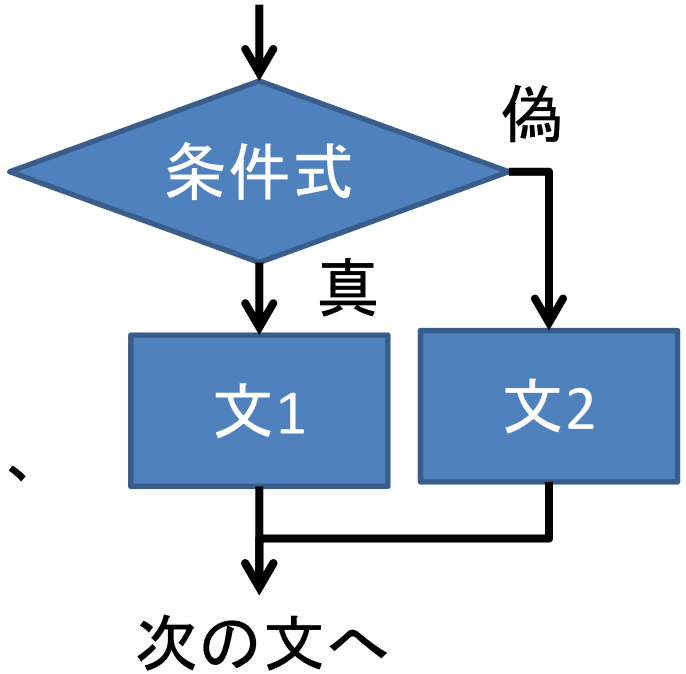
- ソースコード: プログラムの先頭に以下の文を置く
  `#include <math.h>`
- コンパイル: -lm をオプションとしてつける
  - gcc main.c –lm

C言語の基礎: 制御構造
if文 – 条件分岐(1/2)

- 文法

```
if(条件式) 文1;
else 文2;
```

条件式が真なら文1を実行し、
偽なら文2を実行する



– 例: 整数nが偶数ならEVEN、奇数ならODDと出力

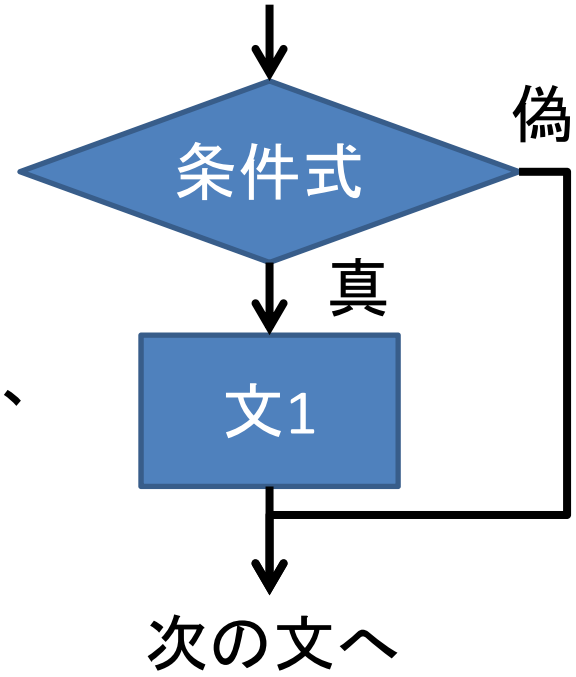
```
if(n%2==0) printf(“EVEN”);
else printf(“ODD”);
```

C言語の基礎: 制御構造
if文 – 条件分岐(2/2)

- else部がなくてもよい

```
if(条件式) 文1;
```

条件式が真なら文1を実行し、
偽なら何もしない



C言語の基礎: 条件式の表現方法 (1/2)

記号	意味	例	例の意味
==	等しい	n == 2	nは2に等しい
!=	等しくない	n != 0	nは0に等しくない
>	より大きい	n > 3	nは3より大きい
>=	～以上	n >= 3	nは3以上
<	より小さい	n < 0.01	nは0.01より小さい
<=	～以下	n <= 0.01	nは0.01以下
&&	～かつ～	0 < n && n <= 10	nは0より大きく10以下
||	～または～	n < 0 || 0 < n	nは0より小さいか、または0より大きい
!	～でない	!(n <= 0.01)	nは0.01以下でない

C言語の基礎: 条件式の表現方法 (2/2)

- 3個以上の数は一度に比較できない
  - `0<x<5` ➔ `0 < x && x < 5`
  - `a==b==c` ➔ `a == b && b == c`

- 例: 閏年かどうかの判定
  - 400で割り切れる, または
  - 100で割り切れないが4で割り切れる

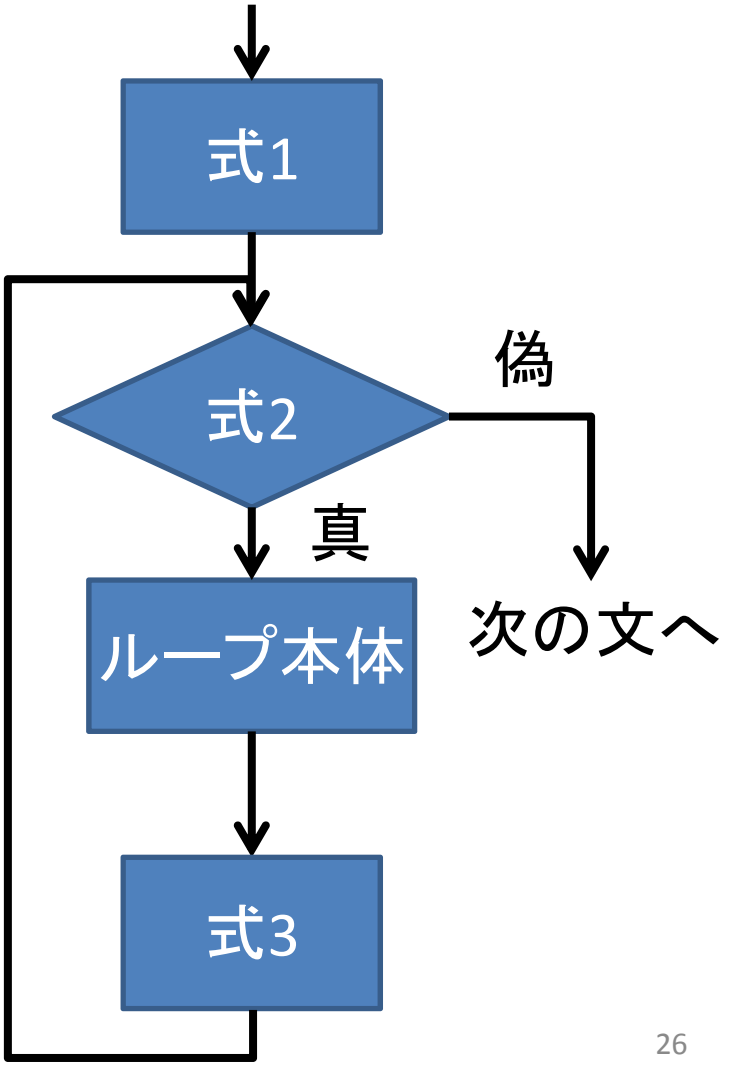
```
year%400==0 || (year%100!=0 && year%4==0)
```

C言語の基礎: 制御構造
forループ – 繰り返し実行(1/4)

- 文法

```
for(式1;式2;式3){
   ループ本体
}
```

- 実行:
  A) 式1を実行
  B) 式2が真ならばCへ、偽ならばDへ
  C) ループ本体を実行、式3を実行してBへ
  D) 次の命令へ



C言語の基礎: 制御構造
forループ – 繰り返し実行(2/4)

- 例: 1からnまでの和 $\sum_{i=1}^{n} i$ を計算して出力

```
int i,n,sum;
n=/*initialized somehow*/;
sum=0;
for(i=1;i<=n;i=i+1){
  sum=sum+i;
}
printf("1+…+%d=%d",n,sum);
```

C言語の基礎: 制御構造
forループ – 繰り返し実行(3/4)

- 例: 1からnまでの2乗和$\sum_{i=1}^{n} i^2$を計算

```
int i,n,sum;
n=/*initialized somehow*/;
sum=0;
for(i=1;i<=n;i=i+1){
  sum=sum+i*i;
}
```

C言語の基礎: 制御構造
forループ – 繰り返し実行(4/4)

- 例: $\sum_{i=1}^{n}(2i-1)^2$ を計算

```
int i,n,sum;
n=/*initialized somehow*/;
sum=0;
for(i=1;i<=2n-1;i=i+2){
  sum=sum+i*i;
}
```

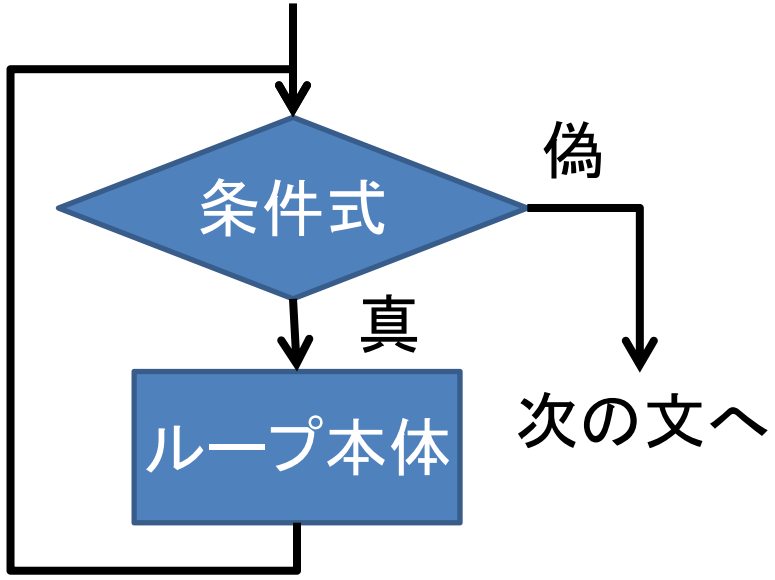
iは 2j-1 を指している

- 何故これで求まる？
  - 理由: $\sum_{i=1}^{n}(2i-1)^2 = 1^2 + 3^2 + \cdots + (2n-1)^2$

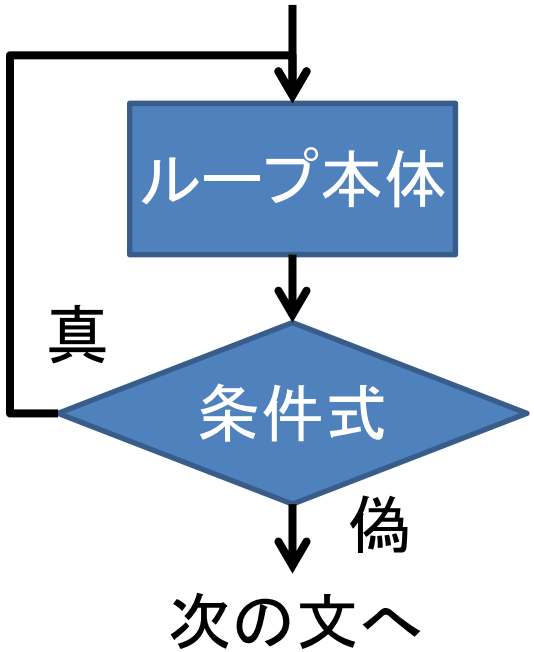
C言語の基礎: 制御構造
whileループとdo-whileループ(1/2)

- 文法

```
while(条件式){
   ループ本体
}
```




```
do{
   ループ本体
}while(条件式)
```



C言語の基礎: 制御構造 whileループとdo-whileループ(2/2)

- 例: 2つの自然数a,bの最大公約数を計算

```
int a,b,r;
a=/*some value*/;
b=/*some value*/;
do{
  r = a % b;
  a = b; b = r;
}while(r!=0);
printf("G.C.D.=%d",a);
```

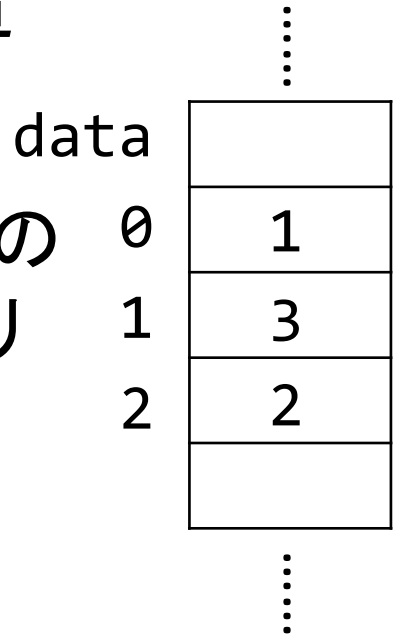
a=1848, b=630の実行

a	b	r=a%b
1848	630	588
630	588	42
588	42	0
42	0	0

この計算法(アルゴリズム)は『ユークリッドの互除法』として知られる

C言語の基礎: 配列(1/2)

- 配列とは?
  同じ型(int, float, etc.)の値をメモリ上に連続して並べるデータ構造
- 例: int data[3]
  - dataという名前でint型の値の格納場所を3つメモリ上に連続して確保




```
int data[3];
data[0]=1;
data[2]=2;
data[1]=3;
```

C言語の基礎: 配列(2/2)
最大値を取得する

- 例: int data[100]に格納された値の最大値を計算する


```
int data[100];
int i,max;
/*data is initialized somehow*/
max=0;
for(i=0;i<100;i=i+1){
  if(max<data[i]) max=data[i];
}
printf(“maximum data = %d”,max);
```

Q: このプログラムは正しい？

C言語の基礎: 配列(2/2)
最大値を取得する

- 例: int data[100]に格納された値の最大値を計算する

正しくない！

```
int data[100];
int i,max;
/*data is initialized somehow*/
max=0;
for(i=0;i<100;i=i+1){
  if(max<data[i]) max=data[i];
}
printf("maximum data = %d",max);
```

dataの値が全て負のとき0が最大値になる！

Q: このプログラムは正しい？

C言語の基礎: 配列(2/2)
最大値を取得する

- 例: int data[100]に格納された値の最大値を計算する – 正しいプログラム

```
int data[100];
int i,max;
/*data is initialized somehow*/
max=data[0];
for(i=1;i<100;i=i+1){
  if(max<data[i]) max=data[i];
}
printf("maximum data = %d",max);
```

maxの値は常にdataの値のどれか

ミニ演習

- （ほぼ）毎回，簡単な演習をする予定
  - 成績には含めない
- ミニ演習2: 次の関数は何をする？
  - collatz(5) と collatz(7) の出力を書け

```
collatz(正整数 n) {
  print(n);  // n を出力
  if (n == 1) return;
  if (n%2==0) collatz(n/2);
  else        collatz(3n+1);
}
```