

# 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

北陸先端科学技術大学院大学  
情報科学研究科 教授

池田 心 [kokolo@jaist.ac.jp](mailto:kokolo@jaist.ac.jp)

2024-04-15

講義資料, 過去問等はLMS上にあります  
2~3日に1回はお知らせ等確認してください

# 講義概要

- I111アルゴリズムとデータ構造
- 担当: 池田 心
- 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ありうる. 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したい.

単に従来のアルゴリズムを記憶するだけではなく, その設計の考え方を身につけることが重要. 例題を用いて, アルゴリズムの正当性を確認し, 効率の改善の余地があるかを調べることの重要性を認識する.

# 教科書・評価方法

- 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.
  - 教科書通りの順番や内容でやるわけではない
  - 予習復習, 深く広く知りたい場合にどうぞ
- 評価の観点 : 基礎理論の理解度と応用力
- 評価の方法 : ~~レポート問題~~・小テスト・期末試験
- 配点 :
  - 小テスト = 35点くらい
  - 期末試験 = 65点くらい
  - 全てLMS上で提出, 実施, 返却予定
  - LMS上の確認クイズ・過去問をぜひ参考に

# 講義の予定について

- 試験は 6/5 (2限, 100分) の予定
- 当面のスケジュール
  - 4/15月 第1回(プログラミングの基礎)+TH(演習. ノートPC!)
  - 4/17水 第2回(アルゴリズムの基礎)
  - 4/22月 第3回(探索問題1)+TH(過去問解説, 演習)
  - ...

TH は主に, 演習・過去問解説です.

2023年度の録画と内容はほとんど同じです.

録画学習, 対面参加, 合っていると思うものを選んでください.

# 受講条件

- 日本語が「読む」「聴く」できること
- 「書け」なくともよい。試験は英語で良い
- アルゴリズムとプログラミングは「一応」別物だが、プログラミングを全くやったことがなければ、それなりの自習によって練習することを強く勧める。そのほうが理解できる。
- 逆に、それなりにできる人には物足りない内容
- 何らかのプログラミング環境は準備しておくこと
  - 例えば Visual Studio (C#など)は無料で本格的
  - お手軽なのは paiza.io など <https://paiza.io/ja/projects/new>
- 言語は、自習用にはなんでも良いが、~~レポートや試験解答のためには、C, C++, Java, Python, C#, Perl のどれかにしてほしい~~
  - ~~Ruby, Basic も読めないことはないです~~

# 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

- どうなればうれしい？
  - どんな入力に対しても正しい解が得られる
  - 妥当なコストで解が得られる
    - 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
- 逆に困るのは？
  - 入力によっては正しい解が得られない
  - 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

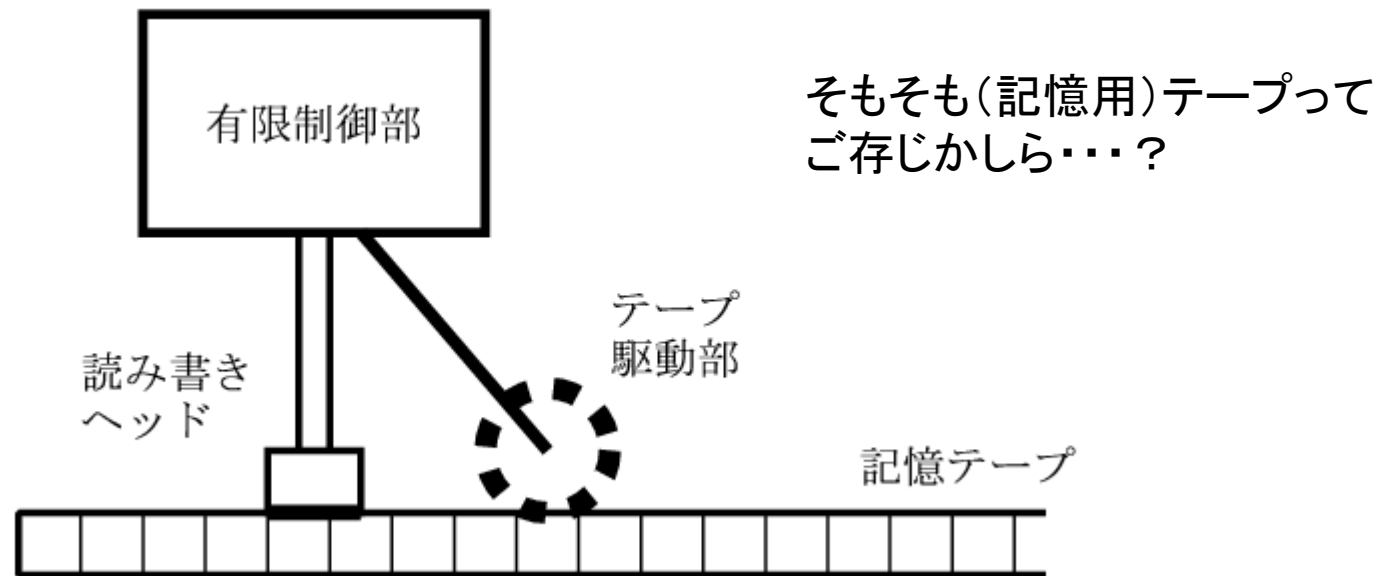
# 計算モデルの話

時間やメモリ量ってどう評価するの？

→ 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どう違ってくる ... 自動車にとっていい道と、自転車にとっていい道は違う
  - 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: 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今これが標準.
  - そのうち, GPUや量子コンピュータを前提にした議論が行われるようになるかも

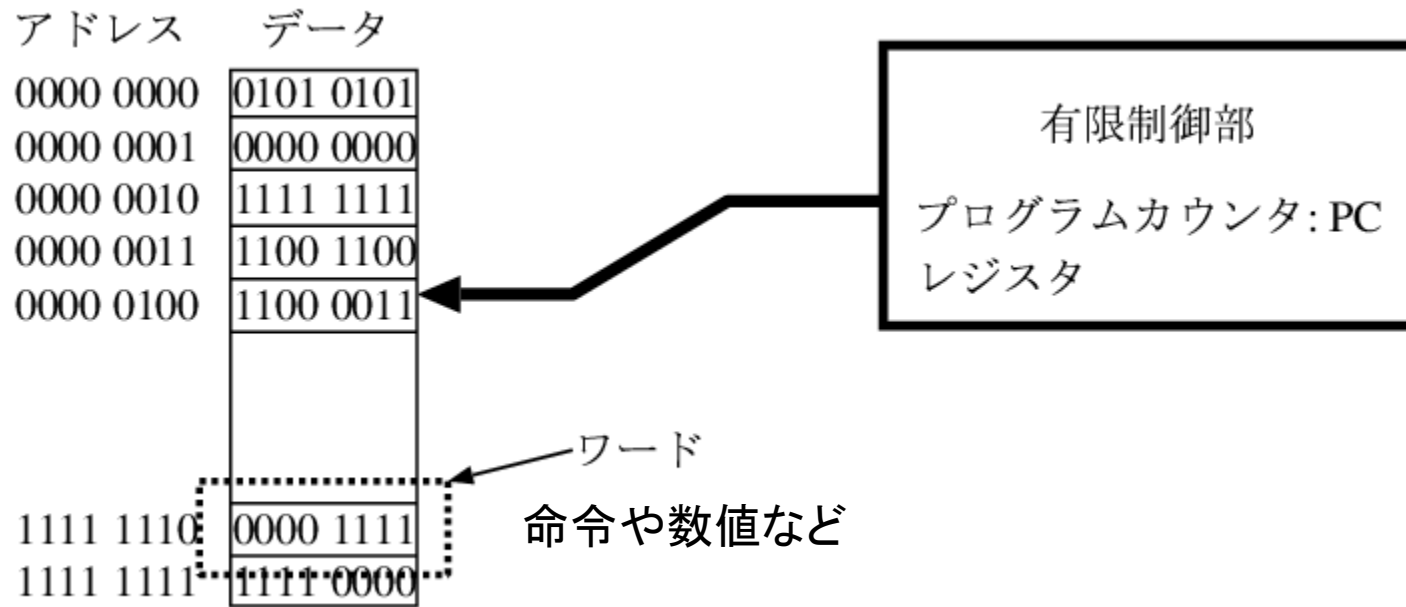
# チューリング機械



- 理論的な扱いが楽な, 非常に単純なモデル
- 原理的に解ける問題は, チューリング機械でも解ける
- 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言語など古めの言語では, 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).

# C#の基礎: Hello World

- とりあえずC#で記述しますが, 他の似たような言語 (C++, Javaなど)でも基本は同じ
- 今日は, (プログラミングの授業ではないが) プログラミングの基礎を学ぶ
- Hello World とディスプレイに表示

```
using static System.Console;  
public class Hello{  
    public static void Main(){  
        WriteLine("Hello World");  
    }  
}
```

命令文の終わりにセミコロン

命令文

- ★ 1行目using...を削除し, 4行目System.Console.WriteLine でも動作します
- ★ Cの場合は, WriteLine の代わりにprintfを使います.

# C#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#の基礎:算術式の**注意点**

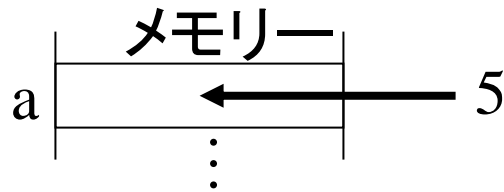
- 分数はない:  $1/3$ は1を3で割った結果になる
- 整数/整数は小数部分が切り捨てられる
  - 例:  $1/3$ は0になる。 $1.0/3$ は0.3333...
- `double av = (int)sum/(int)num` 失敗例
- コンマで区切らない
  - 例: 10,000はダメ。10000と書く。
- 計算順序を制御する括弧: 小括弧を重ねる
  - 算術式の中で中括弧`{}`や大括弧`[]`は使えない
  - 例: `{(3+4)*3+4}*6` はダメ。 `((3+4)*3+4)*6` と書く。
- べき乗の演算子はない(言語によっては `**`)

# C#の基礎: 変数

- 変数: 計算結果を蓄える名前付きの“場所” ハコとも
- 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
    - それ以外の記号は使えない
  - 大文字と小文字は区別される (けど, これで区別しないこと! )
    - `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
  - 正しい例: `x`, `orz`, `T_T`, `IE9`, `projectX`, `ff4`, `y2k`, `JAIST`
  - (ただし, 読みやすいかどうかは別問題)
  - 悪い例: 7`th`, `kokolo`@`jaist`, `ac`.`jp`, `tel`#

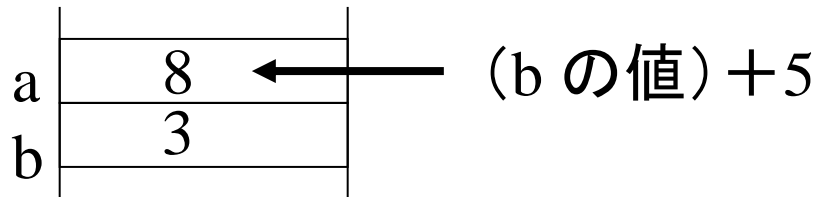
# C#の基礎: 代入文

- $a=5$



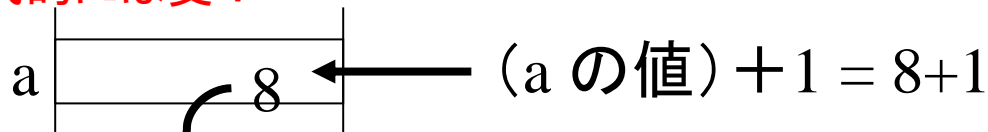
–  $a$ という名前の場所に数値5を格納

- $a=b+5$



–  $b$ 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

- $a=a+1$  数式的には変?



– 変数 $a$ の値に $1$ を加えたものを $a$ の値とする

# C#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

変数a, bを宣言  
(型はint: 整数)

```
using static System.Console;
class Program{
    static void Main(string[] args){
        int a, b;
        a = 5; b = 3;
        WriteLine(a+" "+b+"="+ (a+b));
        ReadLine();
    }
}
```

悪い例

変数を宣言せずに使っている！

```
using static System.Console;
class Program{
    static void Main(string[] args){
        a = 5;
        WriteLine("value="+a);
    }
}
```

Perl, Python などのスクリプト言語は、宣言なしでも大丈夫な場合が多い。勝手に判断してくれる。読むほうにとっては、スッキリする場合と間違える場合と...

# C#の基礎: 算術関数の利用

	関数名	数学的表現	関数の戻り値
平方根	<code>System.Math.Sqrt(x)</code>	$\sqrt{x}$	double
べき乗	<code>System.Math.Pow(x, y)</code>	$x^y$	double
自然対数	<code>System.Math.Log(x)</code>	$\log_e x$	double
常用対数	<code>System.Math.Log10(x)</code>	$\log_{10} x$	double
指数関数	<code>System.Math.exp(x)</code>	$e^x$	double



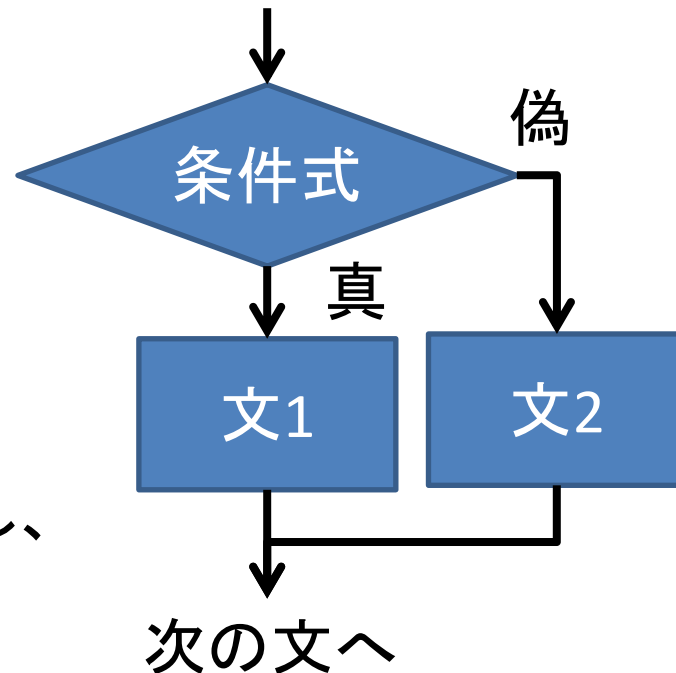
# C#の基礎: 制御構造

## 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) {  
    WriteLine("EVEN");  
} else {  
    WriteLine("ODD");  
}
```

等しいという条件は  
== 二重のイコール記号

# C#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;
```

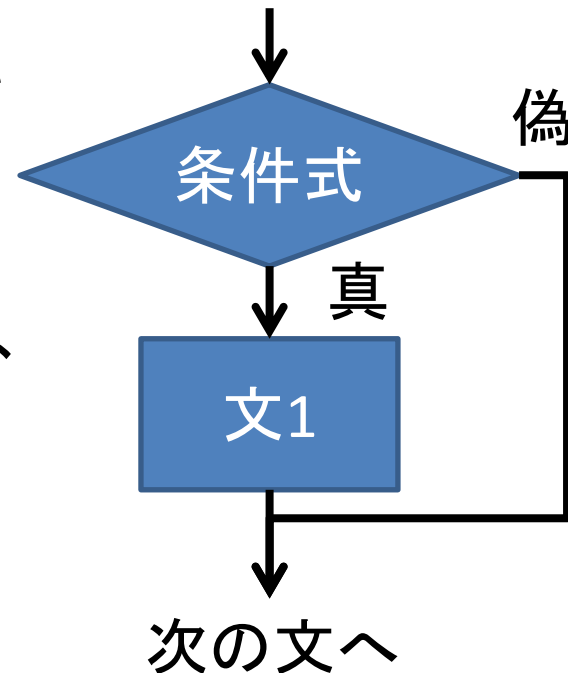
条件式が真なら文1を実行し、  
偽なら何もしない

これは何が起こる？

```
if(条件式) 文1; 文2;
```

こう書くクセをつけとく

```
if(条件式) {  
    文1;  
    文2;  
}
```



# C#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#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

$0 < x < 5$        $\rightarrow$     $0 < x \ \&\& \ x < 5$

$a == b == c$      $\rightarrow$     $a == b \ \&\& \ b == c$

- 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

```
if (year%400 ... ) 閏年      どう書く?
```

# C#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

$0 < x < 5$        $\rightarrow$     $0 < x \ \&\& \ x < 5$

$a == b == c$      $\rightarrow$     $a == b \ \&\& \ b == c$

- 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

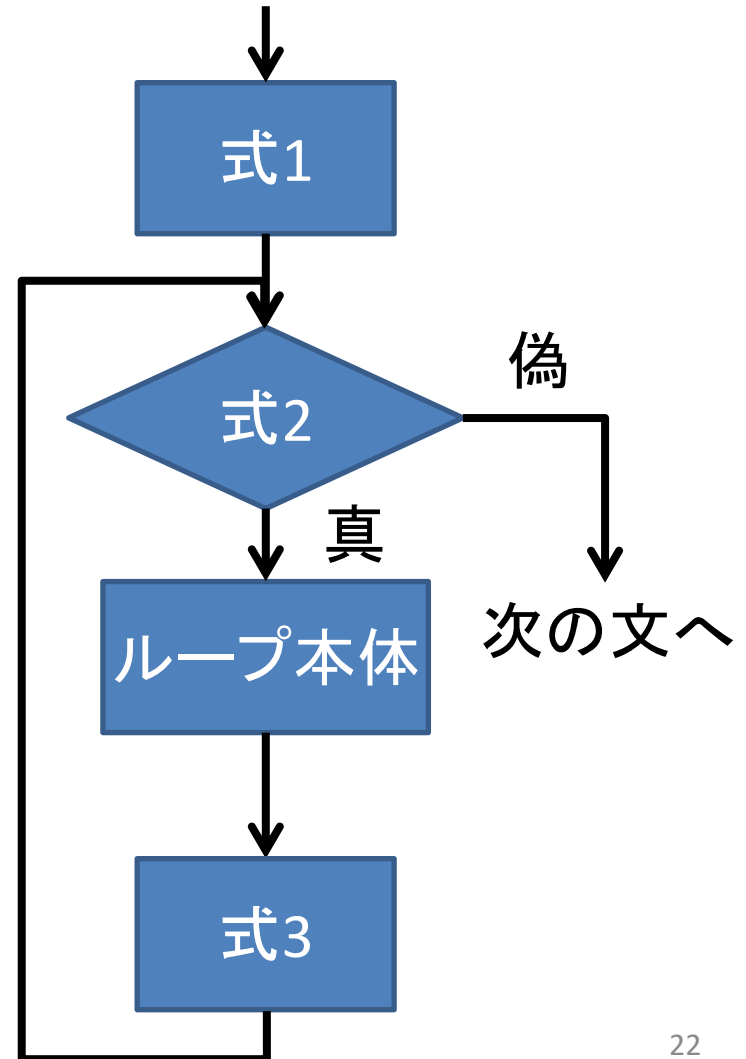
## forループ – 繰り返し実行(1/4)

- 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



一見複雑だが、まずは次ページの  
基本パターンをマスターしよう

# C#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

```
using static System.Console;
class Program{
    static void Main(string[] args){
        int sum = 0;
        int n = 10;
        for (int i = 1; i <= n; i++){
            sum = sum + i;
        }
        WriteLine("1+...+"+n+"="+sum);
        ReadLine();
    }
}
```

i=i+1の代わりに  
i++ とも書く

sum=sum+iの代わりに  
sum+=i とも書く

# C#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

```
using static System.Console;
class Program{
    static void Main(string[] args){
        int n = 10;
        int sum = 0;
        for (int i = 1; i <= n; i++){
            sum = sum + i*i;
        }
        WriteLine("square sum="+sum);
        ReadLine();
    }
}
```



# C#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

```
using static System.Console;
class Program{
    static void Main(string[] args){
        int n = 10;
        int sum = 0;
        for (int j = 1; j <= 2*n-1; j=j+2){
            sum = sum + j*j;
        }
        WriteLine("result="+sum);
        ReadLine();
    }
}
```

jは  $2i-1$  を指している

- 何故これで求まる？
  - 理由:

$$\sum_{i=1}^n (2i-1)^2 = 1^2 + 3^2 + \dots + (2n-1)^2$$

# C#の基礎: 制御構造

## forループ – 繰り返し実行(4/4おまけ)

- 例:  $\sum_{i=1}^n (2i - 1)^2$  を計算

```
using static System.Console;
class Program{
    static void Main(string[] args){
        int n = 10;
        int sum = 0;
        for (int i = 1; i <= n; i++){
            sum = sum + (2*i-1)*(2*i-1);
        }
        WriteLine("result="+sum);
        ReadLine();
    }
}
```

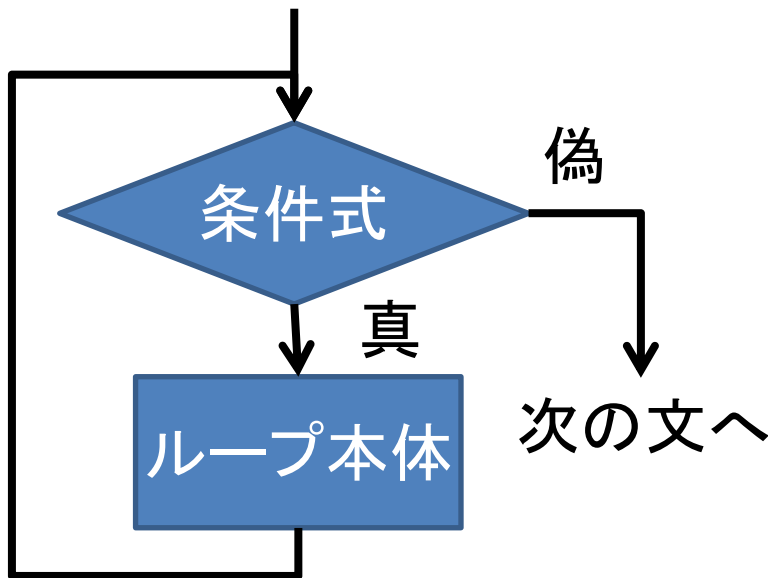
- 別にこれでも良い

# C#の基礎: 制御構造

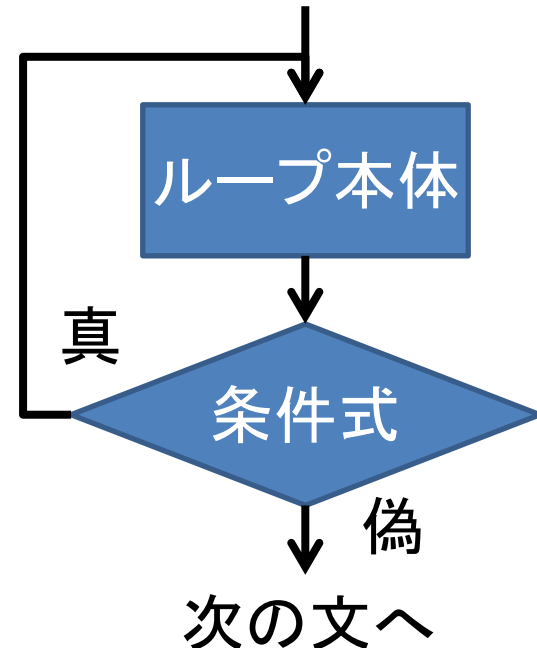
## whileループとdo-whileループ(1/2)

- 文法

```
while(条件式){  
    ループ本体  
}
```



```
do{  
    ループ本体  
}while(条件式)
```



# C#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

```
using static System.Console;
class euclid{
    static void Main(string[] args){
        int a = 1071;
        int b = 1029;
        int r;
        do{
            r = a % b;
            a = b;
            b = r;
        } while (r != 0);
        WriteLine("GCD="+a);
        ReadLine();
    }
}
```

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#の基礎: 配列(1/2)


- 配列とは?

最近の言語では,  
値に限らない

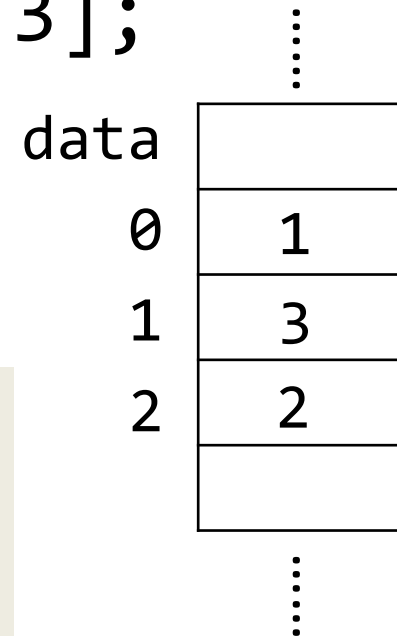
同じ型(int, double, etc.)の値をメモリ上に連続して並べるデータ構造

- 例: `int[] data = new int[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 = new int[3];  
data[0]=1;  
data[2]=2;  
data[1]=3;
```



# Cの基礎: 配列(2/2)

## 最大値を取得する

- 例: `int[] data = new int[100]`に格納された値の最大値を計算する

正しくない！

```
using static System.Console;
int[] data = new int[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];
}
WriteLine("max data =" +max);
```

Q: このプログラムは(classがないとか以外で)正しい？

埋めネタ(配布資料上に答えが見えないようにしたなごり)

僕は学生さんが「入学前配属内定制度」を使いたいと言ったとき、必ず次の質問をしています。

「良いプログラムとはどんなものですか？」

「どんな言語で、何行くらいのプログラムを書いたことがありますか？」

良い授業とは何か？ 良いプログラムとは何か？ 良い人間とは何か？  
もちろん答えはそれぞれで正解などありませんが、いろんな視点で  
プログラムを見つめられる人間は、それだけ経験とセンスがあると思っています。  
例えば・・・

- ・ 高速に動くこと.
- ・ メモリを大きく食わないこと.
- ・ 変な入力に対しても、適切に対処できること.
- ・ 関連して、セキュリティが堅固であること.
- ・ ソースコードに適切にコメントがあり、読みやすいこと.
- ・ ソースコードが適切にブロック化構造化され、読みやすいこと.
- ・ 関連して、無駄に長くも短くもないこと.
- ・ 変数名や関数名が適切な長さで、意味を誤解しないようなものであること.
- ・ 機能が独立性高く作られ、追加や移動や削除や転用が容易であること.
- ・ 言語依存性が低く、別の言語に移植しやすいこと.
- ・ バージョン依存性が低く、別のマシンでも動きやすいこと.
- ・ 各機能や新しい機能がテストしやすいように作られていること.

などなど.



# C#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

```
using static System.Console;
int[] data = new int[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];
}
WriteLine("max data = {0}",max);
```

正しくない！

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(classがないとか以外で)正しい？

# C#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

```
using System;
using static System.Console;
class Program{
    static void Main(string[] args){
        int[] data = new int[100];
        Random rand = new Random();
        for(int i=0; i<data.Length; i++){
            data[i] = rand.Next(-100,101);
        }

        int max = data[0];
        for (int i=1; i<100; i++){
            if (data[i] > max) max = data[i];
        }
        WriteLine(max);
        ReadLine();
    }
}
```

maxの値は常に  
dataの値のどれか

# ミニ演習

- 次の関数は何をする？
  - collatz(5) と collatz(7) の出力を求めよ

```
using static System.Console;
class Program{
    static void Main(string[] args){
        collatz(7);
        ReadLine();
    }
    static void collatz(int n){
        WriteLine(n);// n を出力
        if (n == 1){
            return;
        }
        if (n % 2 == 0){
            collatz(n / 2);
        }else{
            collatz(3 * n + 1);
        }
    }
}
```

関数が自分自身を  
別の引数で呼んでいる「**再帰**」  
これは重要なので  
どこかのTHで解説予定

# TH課題

1. 整数 $n$ が与えられたとき, 1から $n$ までの総和を求めてください.
2. 整数 $n$ が与えられたとき,  $n$ の階乗を求めてください.
3. 7桁の学生証番号が与えられたとき, 奇数番目の数字の最大値と, 偶数番目の数字の最小値を求めてください.
4. 金額 $n$ 円に対し, 100円玉, 50円玉, 10円玉, 5円玉, 1円玉を使って最小枚数で支払う場合の合計枚数を求めてください.
5.  $n$ までの素数を列挙してください.