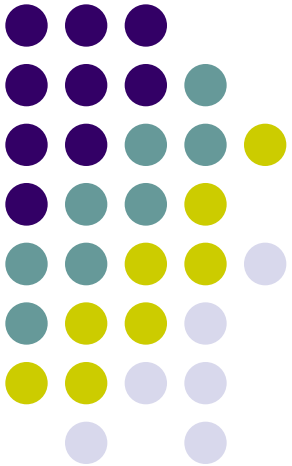
情報システム基盤学基礎1

Elements of Information Systems Fundamentals1

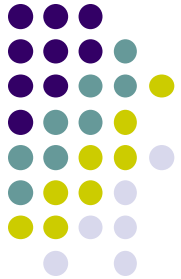
アルゴリズムとデータ構造　第1回



目次

- アルゴリズム　とは
- C言語によるアルゴリズム記述(付録)
- 擬似コード　によるアルゴリズム記述
- 計算量
- 開発環境の整備について

アルゴリズム　Algorithm



入力:　a　value　or　a set of value　//　値, または　値の集合

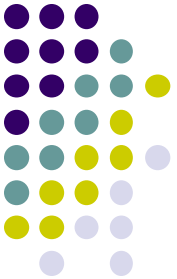
を　受けて

出力:　a value or a set of value

に変換する計算手続き　(well-defined procedure).

例:　入力　＝　n個の数の列　　a1, a2, …, an
　　出力　＝　a1’ <= a2’ <= … <= an’ の並べ替え

正しいアルゴリズム　Correct Algorithm



任意の入力　を　受けて　必ず停止し，その出力が正しいことを保証する計算手続き.

以下はこの場合のみ

例：　入力　＝　n個の数の列　　a1, a2, …, an
　　出力　＝　a1’ <= a2’ <= … <= an’ の並べ替え

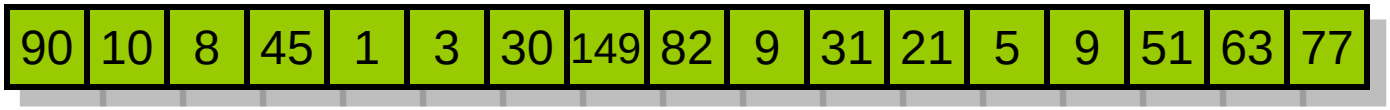
停止が保証されない計算手続でも　アルゴリズムと呼ぶことがある．例えば，正しい入力なら必ず停止して正しい結果を返すが，正しくない入力なら停止を保証しない関数（帰納可算関数）など．定理証明問題では良くある．（計算量理論を参考に．）

アルゴリズムの簡単な例

問題

n 個の整数から最大値と最小値を求める

入力: 整数の列



アルゴリズム

出力: 最大値／最小値

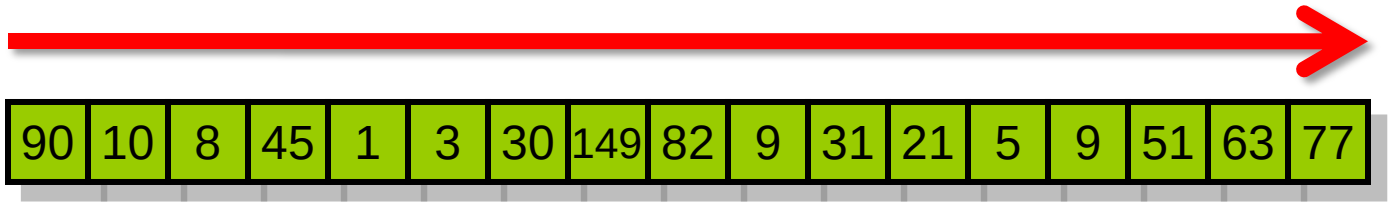
149　　1

単純なアルゴリズム（Naive Algorithm）

問題

n 個の整数から最大値と最小値を求める

処理の流れ



単純なアルゴリズム

左端から右端へ1つずつ整数を見ていく

各整数について

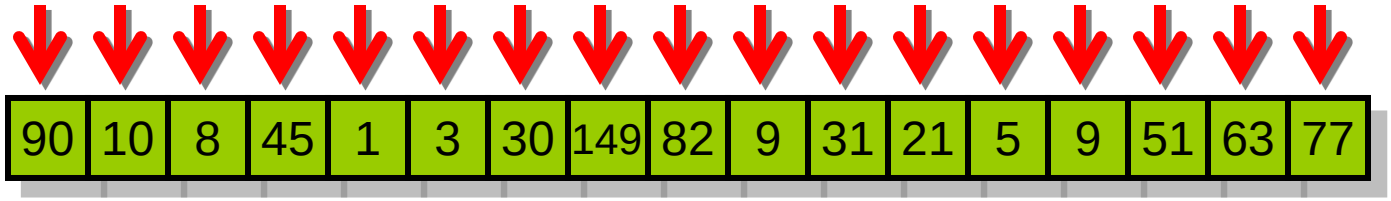
- 現在までの最大値と比較 → 大きければ最大値を更新
- 現在までの最小値と比較 → 小さければ最小値を更新

単純なアルゴリズムの動作例

問題

n 個の整数から最大値と最小値を求める

現在までの最大値: 149　　現在までの最小値: 1



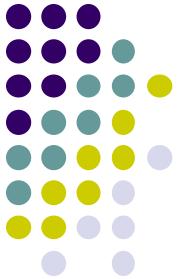
単純なアルゴリズム

左端から右端へ1つずつ整数を見ていく

各整数について

- 現在までの最大値と比較 → 大きければ最大値を更新
- 現在までの最小値と比較 → 小さければ最小値を更新

C言語のプログラム例



```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max= %d  Min= %d¥n", max,min);
}
```

配列a[100]にデータ入力

データ入力

最大値／最小値計算

Preprocessor

変数宣言部

計算部分

main という名前の関数

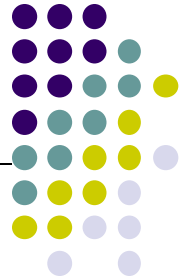
```
#include<stdio.h>
main() {
  ・・・
}
```

自分で調べる

目次

- アルゴリズム　とは
- C言語によるアルゴリズム記述(付録)
- 擬似コード　によるアルゴリズム記述
- 計算量
- 開発環境の整備について

擬似コード



擬似コード: プログラムを“簡単に”書いたもの

```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max=%d,Min=%d¥n", max,min);
}
```

左のCコードを擬似コードで書くと

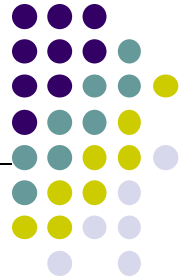
A: 入力配列
max: 現在の最大値(初期値は小さな値)
min: 現在の最小値(初期値は大きな値)

MAXMIN(A)

```
1   for i = 1 to n
2     do   if (A[i] > max) then max ← A[i]
3          if (A[i] < min) then min ← A[i]
4   解を出力
```

- { と } がない代わりに字下げ(indent)した文の列を1ブロックと見なす

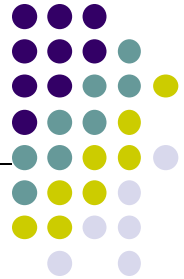
アルゴリズムとプログラム



アルゴリズムを実現するプログラムはたくさんの種類がある

```
MAXMIN(A)
1   for i = 1 to n
2       do  if (A[i] > max) then max ← A[i]
3           if (A[i] < min) then min ← A[i]
4   解を出力
```

アルゴリズムとプログラム



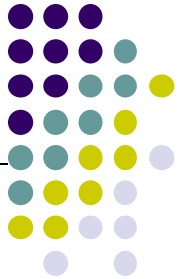
プログラム1

```
#include<stdio.h>

main()
{
 int i, n = 0, max, min, a[100];

 while (n<100) {
   scanf("%d", &a[n]);
   if (a[n] <= 0) break;
   n++;
 }
 max = a[0]; min = a[0];
 for (i=1; i<n; i++) {
   if (a[i] > max) max = a[i];
   if (a[i] < min) min = a[i];
 }
 printf("Max=%d,Min=%d¥n", max,min);
}
```

プログラム2

```
#include<stdio.h>

main()
{
 int i, max, min, a[100];

 for(n=0; n<100; n++) {
   scanf("%d", &a[n]);
   if (a[n] <= 0) break;
 }

 max = a[0]; min = a[0]; i=1;
 while(1) {
   if (a[i] > max) max = a[i];
   if (a[i] < min) min = a[i];
   if (i == n) break;
   i++;
 }
 printf("Max=%d,Min=%d¥n", max,min);
}
```

擬似コードでアルゴリズムを考える

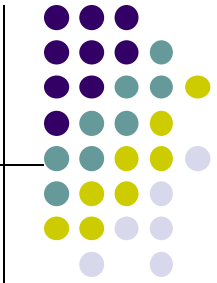


```
MaxMin(A)
1   for i = 1 to n
2       do  if (A[i] > max) then max ← A[i]
3           if (A[i] < min) then min ← A[i]
4    解を出力
```

記法 (notation)

関数: function_name (argument). Call by value が基本.
変数: 型宣言なし. オブジェクト変数あり.
配列: A[100] など. 配列要素は A[i] で 表記.
代入: a = b, a ← b など. 今年からは a=b を多用.
等号判定: if (a == b) then ….

擬似コードの規則



関数: function_name (引数)

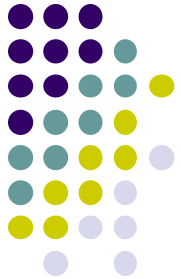
1. 文1
2. 文2
3. 文3
4. return (値)

Call by value （値を渡して，値を受け取る．）

変数: 型宣言なし．いちいちint x; とか書かない．

配列: A[100] など．配列要素は A[i] で 表記．
A[0..3] は A[0]からA[3]の要素列を示す

オブジェクト変数: struct {int a,b,c;} x と おいて
x の要素を x.a とか x.b とか 書く．

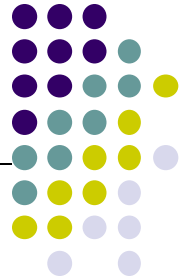


変数への代入:
a = b　　と書いたり
a ← b　　と書く.

等号判定:　if (a == b) then ….

オブジェクト変数の代入: オブジェクト変数x,y について　x = y　や x ← y で　書く.

擬似コードの規則



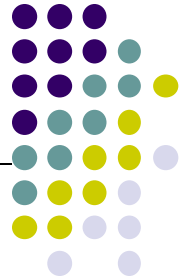
	擬似コード	C言語

If-else文	if (x == 3) then
	　　f(x);
	else if (x>3)
	　　g(x);
	　　x=x+1;
	else
	　　g(x);
		If (x == 3)
		　　f(x);
		Else if (x > 3) {
		　　g(x);
		　　x = x+1;
		}
		Else {
		　　g(x);
		}

字下げ(indent)を1回揃えた文の列を { } や begin … end で囲んだ
1ブロックと解釈する擬似コードが増えた.
論文でも多用される. (tex)

のつもり

擬似コードの規則



For ループ

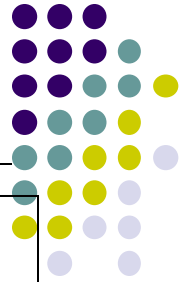
擬似コード

```
main()
1.      N = 4
2.      A[0..3]の値を設定
3.      for  j = 0 to N-1
4.              swap(A[j], A[N-j]) // A[j]とA[N-j]を交換
5.      return   //  実はAは同じ値に戻っている
```

C言語

```
main ()
{       int N = 4;
        for ( j = 0 ; j < N; j++ )
                swap(A[j], A[N-j]);
}
```

擬似コードの規則

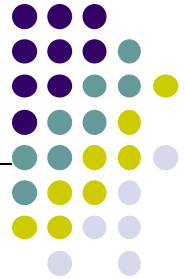


For ループ

擬似コードの書き方は　いろいろ　あるが
1ブロックを表す規則に注意すれば良い.

```
main()
1.      N = 4
2.      A[0..3]の値を設定
3.      for  j = 0 to N-1  do
4.              swap(A[j], A[N-j]) // A[j]とA[N-j]を交換
5.      return   //  実はAは同じ値に戻っている

main()
1.      N=4;   set A[0..3];
2.      for each j = 0  to  N-1
3.          do     swap(A[j], A[N-j]);
4.      return
```

擬似コードの規則



while ループ

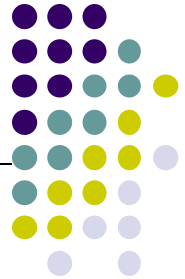
擬似コード

```
main()
1.      N = 4; j = 0;
2.      A[0..3]の値を設定
3.      while ( j < N )
4.          swap(A[j], A[N-j]) // A[j]とA[N-j]を交換
5.          j = j + 1;
5.      return
```

C言語

```
main ()
{       int N = 4;
        while ( j < N ) {
                swap(A[j], A[N-j]); j = j + 1;
        }
}
```

擬似コードの規則



while ループ

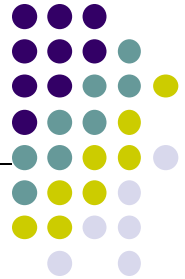
擬似コード

```
main()
1.      N = 4; j = 0;
2.      A[0..3]の値を設定
3.      while ( j < N )
4.          do
5.              swap(A[j], A[N-j]) // A[j]とA[N-j]を交換
6.              j = j + 1;
7.      return
```

ブロック(block)　を表す

擬似コードは　教科書ごとに記法が違うが
C言語に準じる.　MIT の教科書の結果, Pythonなど　ほぼ擬似コードと同じ記法の言語が増えた.

擬似コードの規則



while ループ

擬似コード

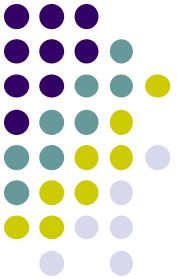
```
main()
1.      N = 4; j = 0;
2.      A[0..3]の値を設定
3.      while ( j < N )
4.          begin
5.              swap(A[j], A[N-j]) // A[j]とA[N-j]を交換
6.              j = j + 1;
7.          end
8.      return
```

擬似コードは　教科書ごとに記法が違うが
C言語に準じる.　MIT の教科書の結果, Pythonなど　ほぼ擬似コードと同じ記法の言語が増えた.

目次

- アルゴリズム　とは
- C言語によるアルゴリズム記述(付録)
- 擬似コード　によるアルゴリズム記述
- 計算量 --　より良いアルゴリズム(解き方)とは？
- 開発環境の整備について

計算量 (Computational Complexity)



アルゴリズムAが終了するまでに実行する基本命令の回数
（※最悪の場合で考える）

- 比較
- 交換
- 代入 etc.

入力の大きさ(size, サイズ) n の関数
によってAの効率を　表現

MinMax() の場合

入力

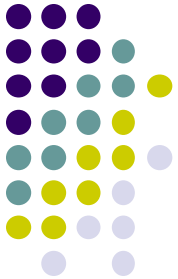
最大/最小
アルゴリズム

出力

最大値
最小値

基本命令の回数を数えて,
n の式で表す

基本命令の回数を見積もる



```
MaxMin(A)
1   for i = 1 to n
2       do  if (A[i] > max) then max ← A[i]
3               if (A[i] < min) then min ← A[i]
4   解を出力
```

比較，代入を単位操作と考えた場合：

- *n+1*回
- *2n* 回
- *2n* 回
- 1 回

} 5*n*+2 回

T(*n*): 最大値/最小値アルゴリズムの基本命令回数

$$T(n) = 5n + 2$$

（入力サイズの関数によって表現しています）

Big-O表記

基本命令の計算回数を細かく計算するのは面倒

（複雑なアルゴリズムだと見積もりが困難…）

ではどうする？

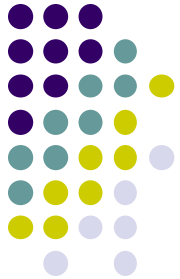
簡単に見積もりができて，かつ，
正しくアルゴリズムを評価できる尺度が欲しい

どんなもの？

基本命令の回数を big-O表記で考える

$T(n) = 5n + 2$ ⇒ $O(n)$

Big-Oの決め方（厳密ではない）



Big-O表記の作り方

1. 式中で, 一番“効いてくる”項のみを考える

$T(n) = 5n + 2 \Rightarrow 5n$

2. 係数を無視して、$O(\ )$ をつける

$5n \Rightarrow n \Rightarrow O(n)$

極端な話…

$n + 2{,}000{,}000 \lg n + 5{,}000{,}000 \Rightarrow O(n)$

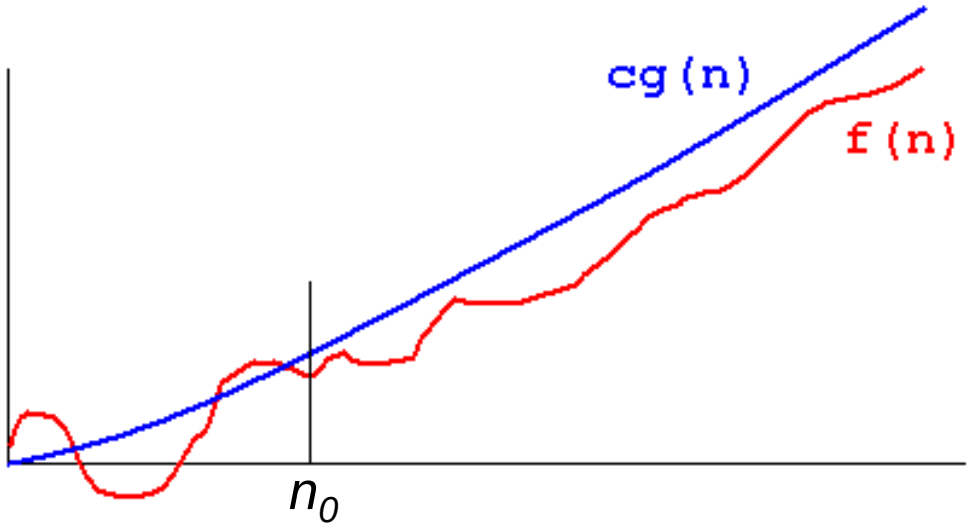
Big-O表記の正確な定義

$O(g(n))$ の定義

$O(g(n)) = \{ f(n) : n \geqq n_0$ に対して, $0 \leqq f(n) \leqq cg(n)$
を満たす定数 c と n_0 が存在する $\}$

$O(g(n))$ は“関数の集合”です

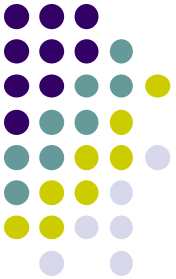
例えば、$O(n^2) = \{ f(n) = 5n, f(n) = \sqrt{n},$
$f(n) = \lg n, f(n) = n \lg n, \ldots\}$



他にも…

$\Omega(g(n))$: 下界
$\Theta(g(n))$: 上界＆下界
$o(g(n))$: 真の上界
$\omega(g(n))$: 真の下界

Big-O表記で見積もる



```
MAXMIN(A)
1   for i = 1 to n                                  n+1回
2       do  if (A[i] > max) then max ← A[i]         2n 回
3           if (A[i] < min) then min ← A[i]         2n 回
4   解を出力                                        1 回
```

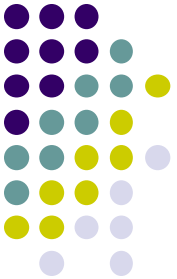
} O(*n*) 回

$$T(n) = O(n)$$

‘=’ は誤った書き方だが, 慣例的にこう書く

($T(n)$: 最大値/最小値アルゴリズムの基本命令回数)

計算量 (時間計算量)という尺度　の意味

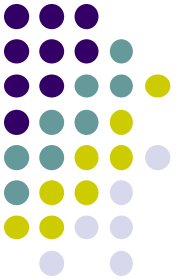


-- アルゴリズムを考える意義とは？

「直面した未解決の問題を正しく，かつ，
効率良く解く手続きを設計し実現する問題」
＋　問題の大きさn が巨大化することに対処
できる限界を明らかにする.

C，Javaのプログラム開発経験とは異なる問題です.

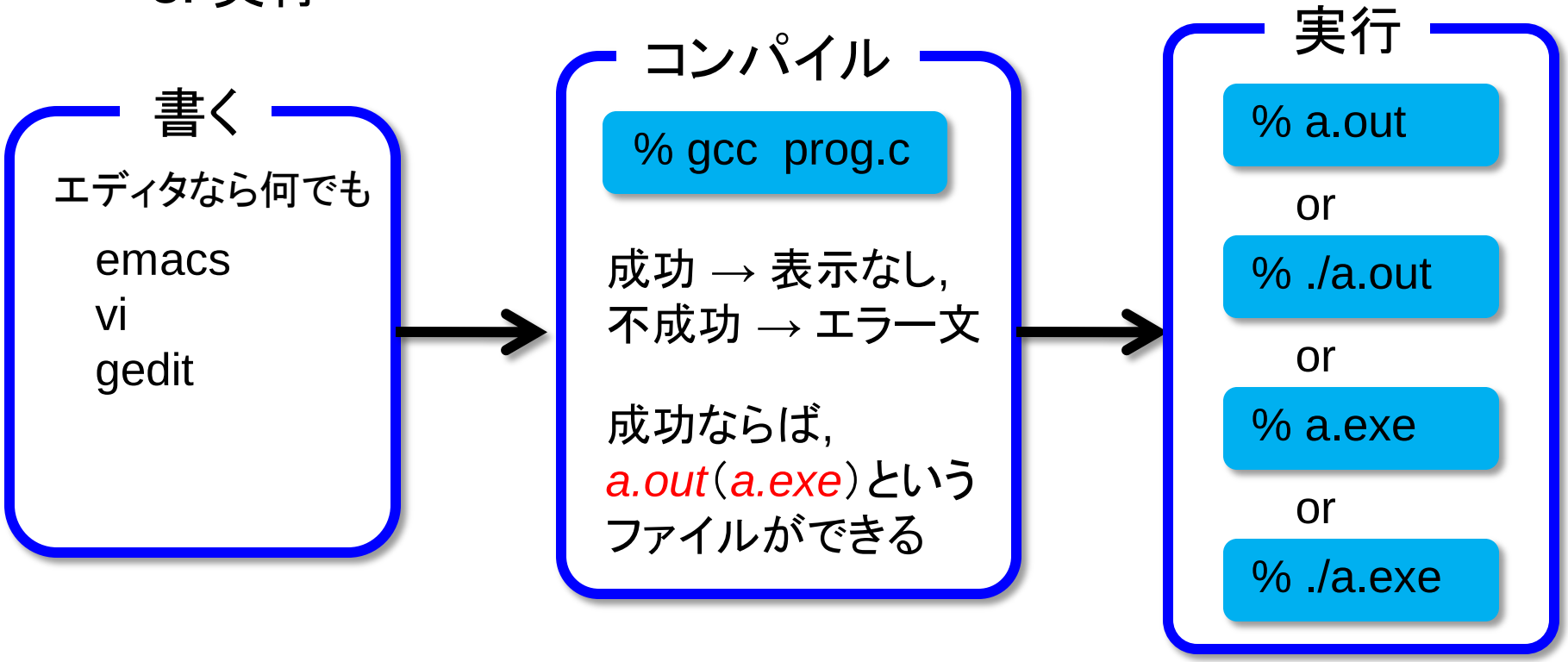
目次



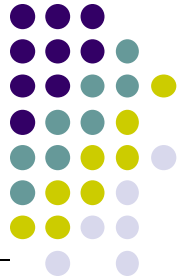
- アルゴリズムとは?
- アルゴリズムのさわり
- C言語のさわりだけ解説
- 擬似コード
- 計算量の見積もり
- 開発環境の整備　--　次回までに必須.

プログラムを動かしたい

1. エディタでプログラムを書く
2. コンパイルする
3. 実行



開発環境の紹介



- VMware + Ubuntu　（FS2受講者）

  Windows上でLinuxを仮想的に動かす

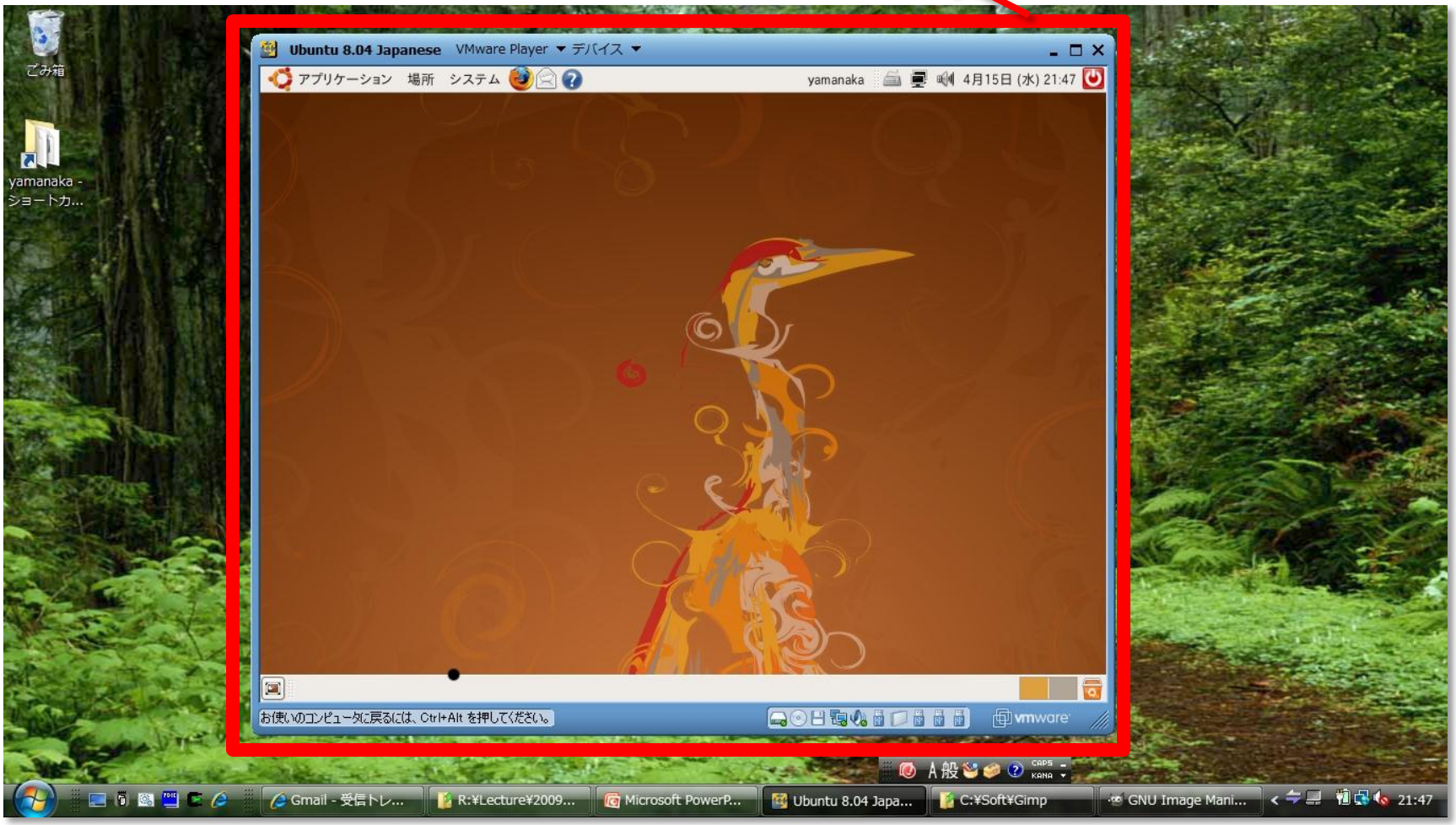
  （FS基礎2 でインストールの説明（演習？）があります
  受講している人は、そちらの説明を優先すること！）

2年間 情報工学の修士を務めるなら:

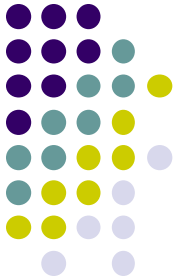
1. 最初から Unix/ Linux を使う (大学なら当然です)
2. 主義者の研究室なら MacOS (宗教)
3. Windows/java でアプリや開発業務の経験，
   ライブラリによる穴埋めソフト開発の経験，などは
   アルゴリズムと問題解決の練習にならない.
   （未経験な方が むしろ 伸びる）.

VMware + Ubuntu（詳細はFS基礎2にて）

Windows上でLinuxが動いている



使えるようにするまで



1. VMware player のインストール

以下からダウンロードしてインストール
http://www.vmware.com/jp/products/player/
（メンバー登録してダウンロードしてください）

2. Ubuntu 仮想マシンのダウンロード

以下よりVMware用仮想マシンをダウンロード
http://www.ubuntulinux.jp/products/JA-Localized/vmware
（基礎2の受講者は，指示のあった仮想マシンをダウンロードすること）

3. VMware の実行 + Ubuntu の読み込み

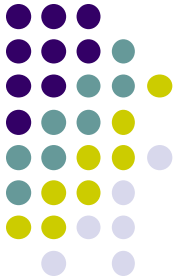
2. でダウンロードしたファイルの中に ”ubuntu.vmx” があります
それをVMware から開いてください
　VMware を実行 ⇒ ubuntu.vmx を開く ⇒ いろいろ設定

注意事項

- プログラム初心者の方へ
  - プログラムを練習しておくことを強くお勧めします
    - 教科書のプログラムを書いて動かしてみる
    - 教科書の演習課題をやってみる, etc

付録

C言語のプログラム例



```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max= %d  Min= %d¥n", max,min);
}
```

データ入力

最大値／最小値計算

プリプロセッサ（おまじない）

変数宣言部

計算部分

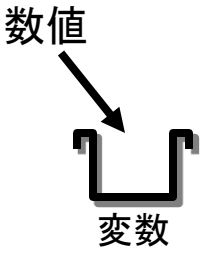
main という名前の関数

はじめての人は…

```
#include<stdio.h>
main() {
  ・・・
}
```

を“おまじない”と思ってよい

変数




```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max= %d  Min= %d¥n", max,min);
}
```

変数宣言部

データ入力

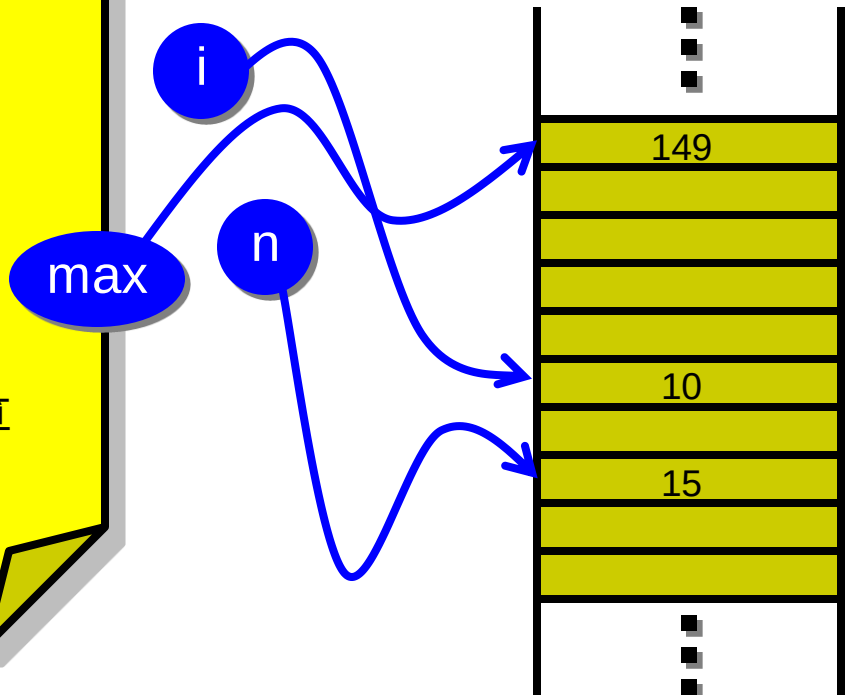
最大値／最小値計算

変数

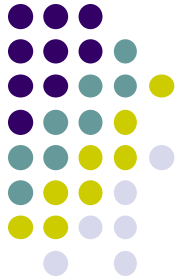
数値をしまっておくための箱

（メモリ領域の一部を，数値を保存するために割り当てたもの）

メインメモリのイメージ

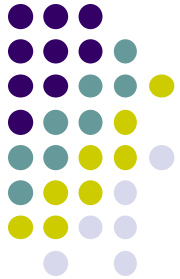


いろいろな変数



- int, float, char
- 配列

いろいろな変数: int, float, char



- *int*： 整数型
  整数を保存する変数
  （4 byte（または 2 byte）, -2147483648～2147483647）
- *float*： 実数型
  実数を保存する変数
  （4 byte, 約 -10^{38}～10^{38}）
- *char*： 文字型
  文字を保存する変数
  （1 byte, 約 -128～127）

（他にも
long: 倍長整数型
double: 倍長実数型）

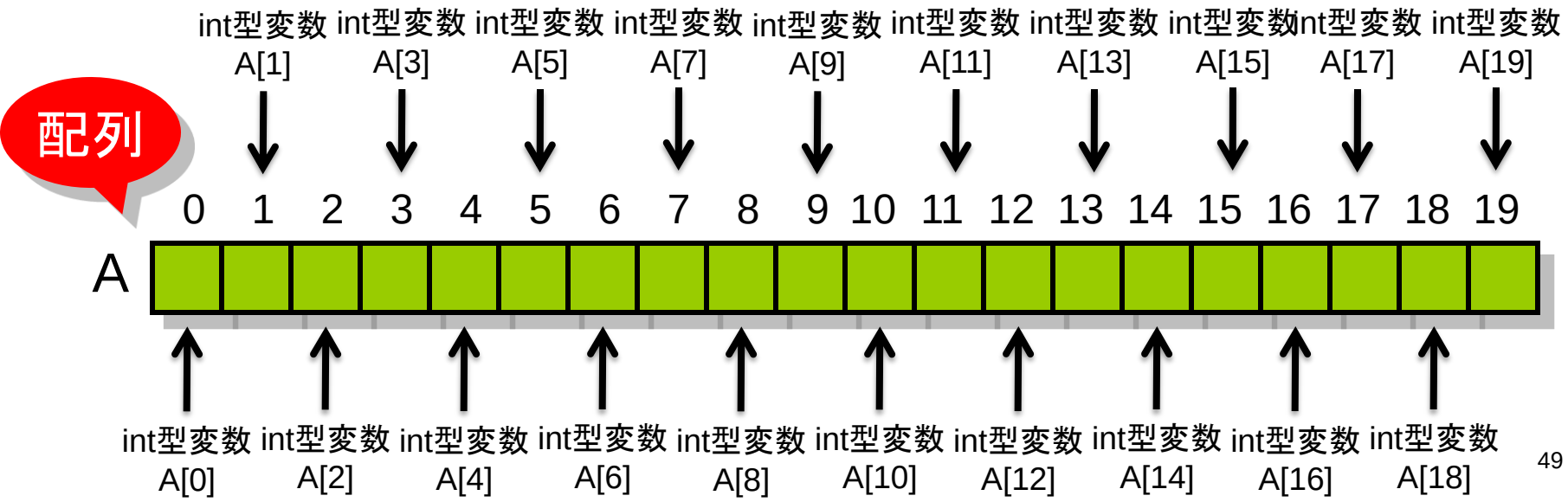
いろいろな変数: 配列

- 基本の変数（例えばint型）を複数個だけ1列に並べたもの

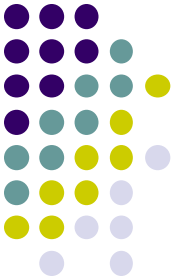
  例：int A[20];

  イメージ：20個のint型変数を一列に並べたもの

  意味：“A”という名前で，長さ20の整数型配列



最大値／最小値を計算するプログラムの例



```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max= %d  Min= %d¥n", max,min);
}
```

ここの説明を見てきた

変数宣言部

データ入力

次にここの説明を見ていく

計算部分

最大値／最小値計算

制御文

- if 文
- for 文
- while 文
  - do-while 文
  - switch 文
  - continue 文
  - break 文

if文

指定した条件によって，処理を2パターンに分ける

Templete

```
if（条件式）{
      ・・・・・
    パターン1
    （条件式が成立した場合の処理）
      ・・・・・
} else {
      ・・・・・
    パターン2
    （条件式が成立しない場合の処理）
      ・・・・・
}
```

気持ち

もし（条件式）を満たすなら，
　→ パターン1の処理を実行，
そうでないならば，
　→ パターン2の処理を実行

ちなみに，else以下を書かずに

```
if（条件式）{
      ・・・・・
}
```

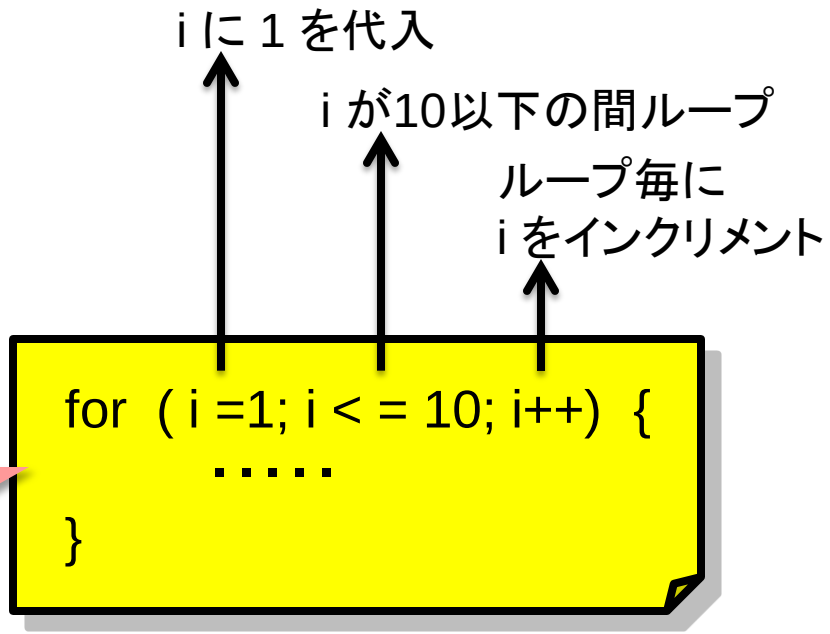
としても大丈夫（elseの処理なし）

for文

指定した回数だけ処理を繰り返す

Templete

```
for (初期値; 継続条件; 増分) {
    .....
    繰り返す処理
    .....
}
```



例：1から10までのループ

処理の流れ

- i に1 を代入
- i <= 10 かどうかチェック
- ループ内の処理
- i をインクリメント
- i <= 10 かどうかチェック
- ループ内の処理
- i をインクリメント

...

while文

➡ 条件を満たすまで処理を繰り返す

Templete

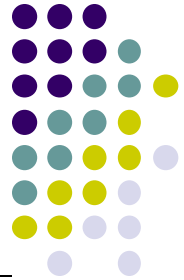
```
while (条件式) {
・・・・・
  繰り返す処理
・・・・・
}
```

処理の流れ

- 条件式を満たすかどうかチェック
- ループ内の処理
- 条件式を満たすかどうかチェック
- ループ内の処理
- 条件式を満たすかどうかチェック
- ループ内の処理

・・・

最大値／最小値を求めるプログラム



while文

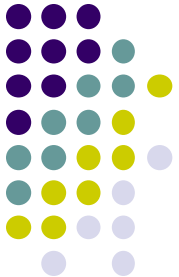
for文

```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max= %d  Min= %d¥n", max,min);
}
```

データ入力

最大値／最小値計算

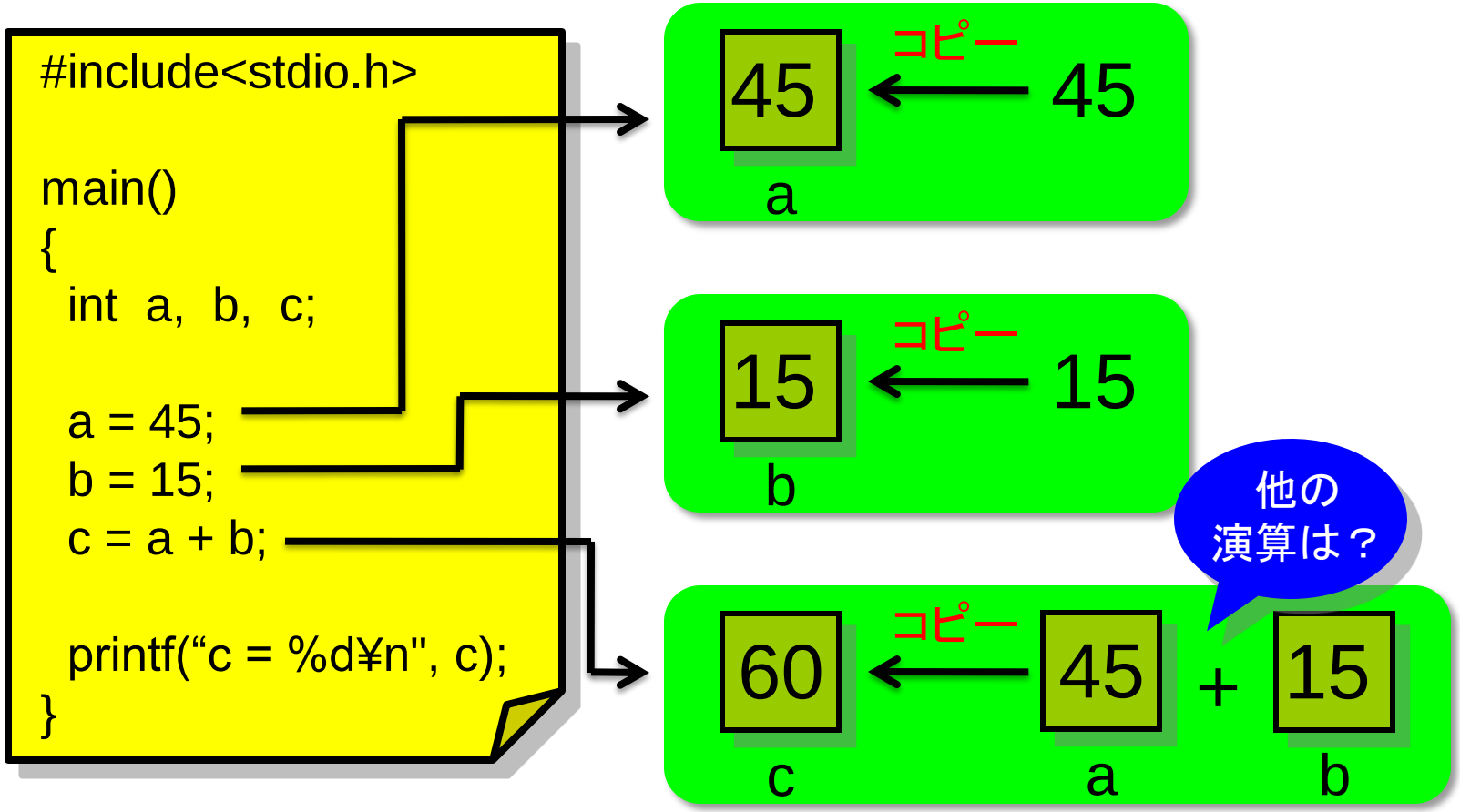
演算



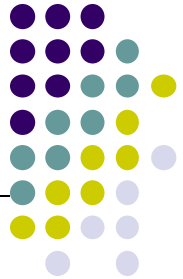
- 代入
- 算術演算
- 比較演算

代入 “=”

- 計算結果を変数へコピー



算術演算



5つの算術演算子

演算子	説明	使用例
+	足し算	a = b + c;
-	引き算	a = b – c;
*	掛け算	a = b* c;
/	割り算	a = b / c;
%	余り（mod）	a = b % c;

注1: 割り算は int型 と float型 で値が異なる

注2: b % c ⇒ “b を c で割ったときの余り”

（例えば「22 % 4」の答えは “2”）

比較演算

```
...
while (n<100) {
  scanf("%d", &a[n]);
  if (a[n] <= 0) break;
  n++;
}
...
```

6つの比較演算子

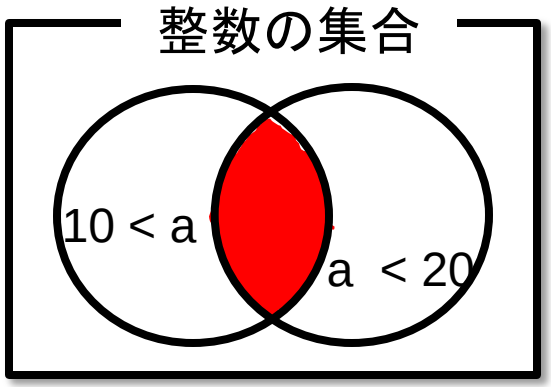
演算子	説明	使用例
<	小さい	a < 10
<=	小さいか等しい	a <= 10
>	大きい	a > 20
>=	大きいか等しい	a >= 20
==	等しい	a == 15
!=	等しくない	a != 18

論理演算

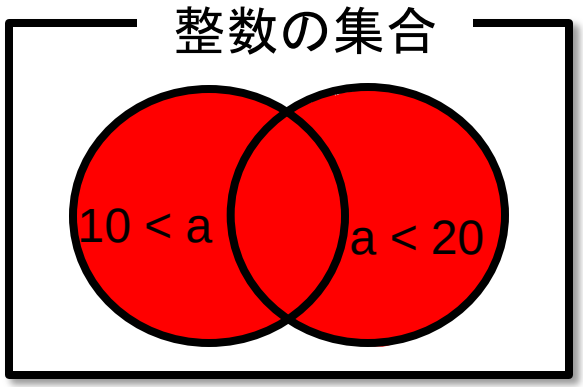
- 3つの論理演算子

演算子	説明	使用例
&&	論理積（かつ：∧）	(10 < a && a < 20)
||	論理和（または：∨）	(10 < a || a < 20)
!	否定（¬）	! (10 < a || a < 20)

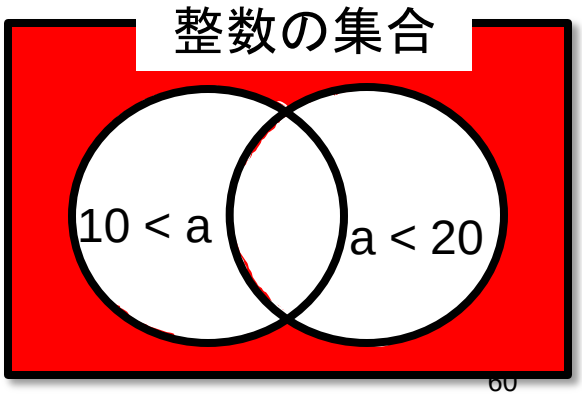
ベン図で考えると下記の通り



（10 < a && a < 20）

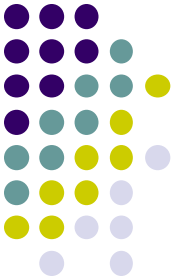


（10 < a || a < 20）



!（10 < a || a < 20）

最大値／最小値を計算するプログラムの例



以上の説明で,
だいたい理解できます

補足

break文

強制的に
ループの外へジャンプ!!

インクリメント/デクリメント

i++; ⇒ i = i + 1;

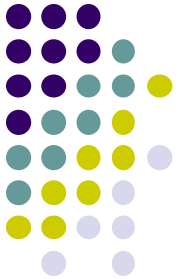
（i--; ⇒ i = i - 1;）

```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max= %d  Min= %d¥n", max,min);
}
```

データ入力

最大値／最小値計算

関数



- 関数／関数の引数と戻り値
- 再帰呼出

関数を使いたいとき

大きなプログラムを書いているとき

```
#include<stdio.h>

main()
{
  .....
  .....
  .....
  .....
  .....
  .....
  .....
  .....
  .....
  .....
  .....
}
```

長い計算

分かりずらい!!

どうする？

機能ごとに
分けて記述しよう

ここで
使えるのが…

関数

1つの関数により
1つの機能を表現

関数を使ってプログラムを機能ごとに分ける

```
#include<stdio.h>

main()
{
  int i, n = 0, max, min, a[100];

  while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++;
  }

  max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i];
  }
  printf("Max= %d  Min= %d¥n", max,min);
}
```

データ入力

最大値／最小値計算

結果の出力

```
#include<stdio.h>

 int i, n = 0, max, min, a[100];

void  input() {
 while (n<100) {
    scanf("%d", &a[n]);
    if (a[n] <= 0) break;
    n++; }
}

void  maxmin() {
 max = a[0]; min = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > max) max = a[i];
    if (a[i] < min) min = a[i]; }
}

void  print() {
 printf("Max= %d  Min= %d¥n", max,
}

main()
{
  input();
  maxmin();
  print();
}
```

関数のイメージ（引数と戻り値）

数学で習った関数

$x = 5$ ⇒ 代入

$f(x) = x^2 + x + 1$ 関数 ⇒

$f(x) = 31$ 計算結果

C言語の関数

func(5) ⇒ 関数呼出

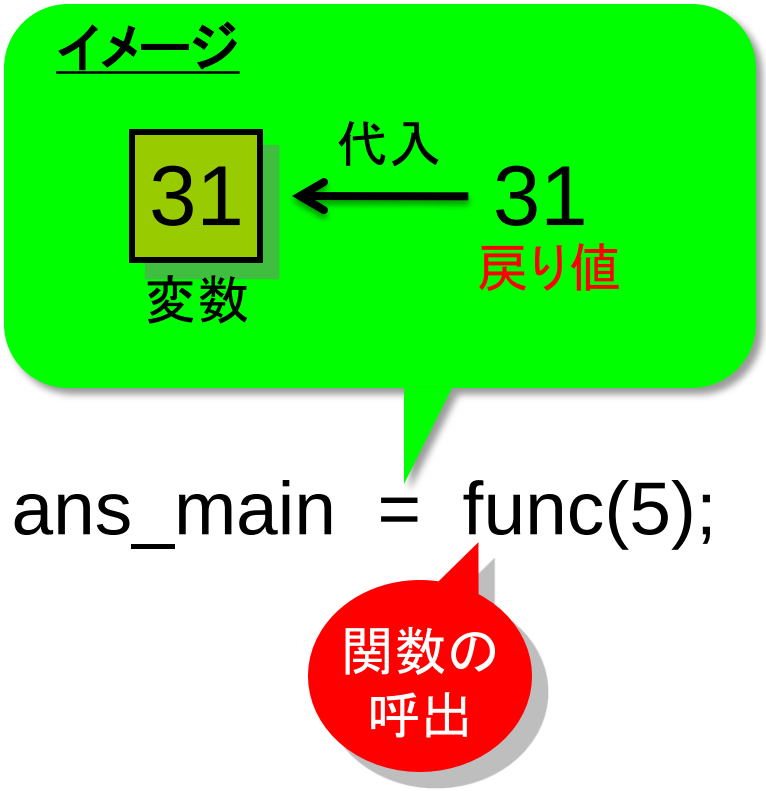
引数

```
int  func (int x)  {
   int ans;
   ans = x*x + x + 1;
   return  ans;
}
```

関数 ⇒ 31 戻り値

関数を使ってみる

```
#include<stdio.h>

int  func(int x) {
  int  ans;

  ans = x*x + x + 1;
  return ans;
}

main()
{
  int  ans_main;

  ans_main = func(5);
  printf("f(5) = %d¥n", ans_main);
}
```



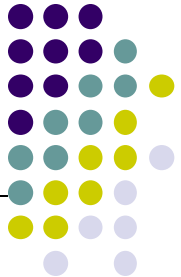
再帰呼出

自分自身を呼び出す関数

```
int  func(int  x)  {
     .....
     ans = func(x - 1);
     .....
    return  ans;
}
```

自分自身の呼出
（再帰呼出）

再帰呼出の例 1/2



- *n*!（n の階乗）の計算を考えよう

数学的に書くと… （*n* は自然数とします）

$$f(n) = \begin{cases} 1 & n = 1 \text{ のとき} \\ n \times f(n-1) & n > 1 \text{ のとき} \end{cases}$$

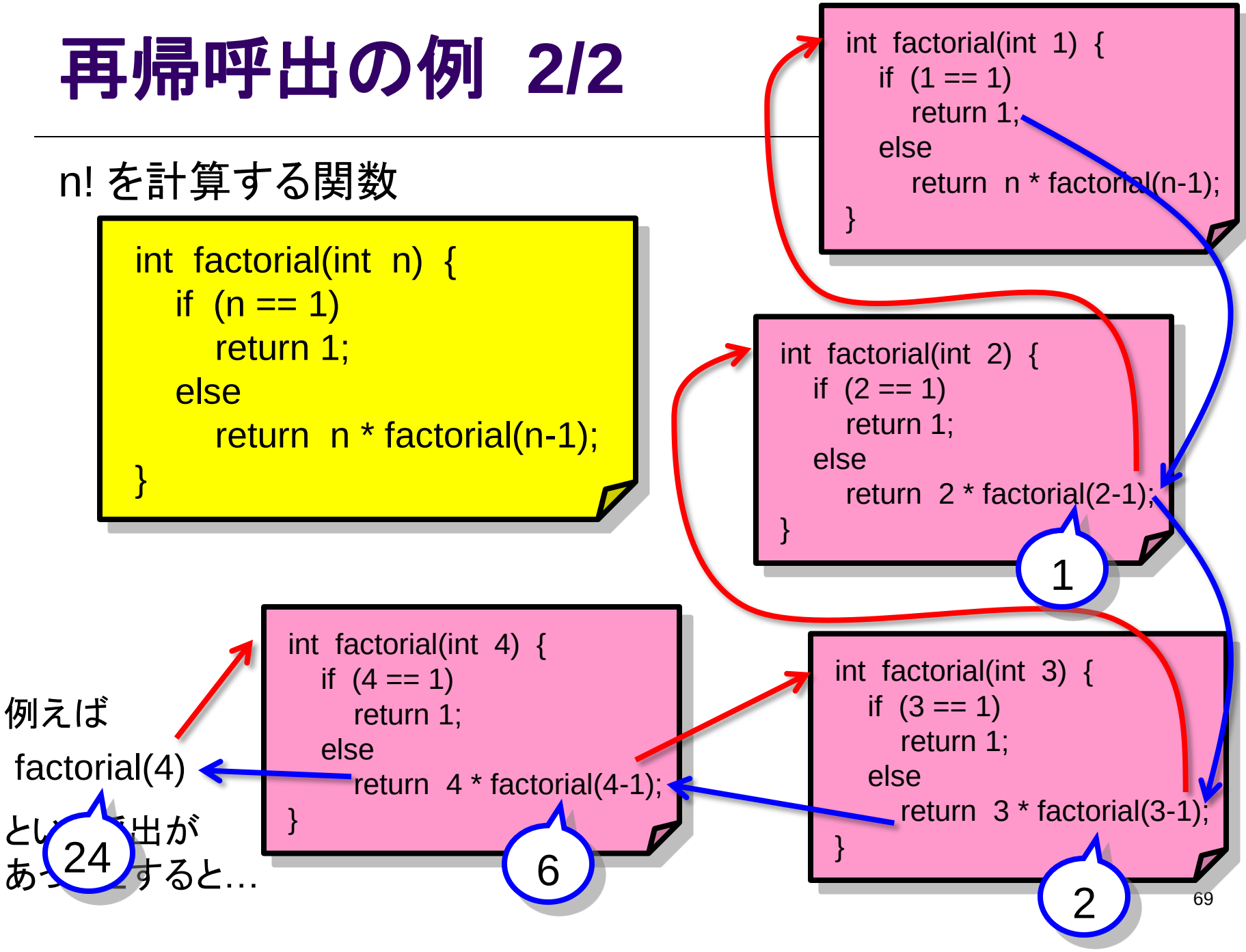
そのまま
プログラム
にすると

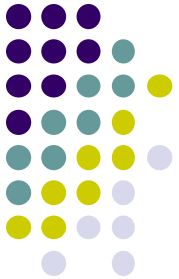
```
int  factorial(int  n)  {
    if  (n == 1)
        return 1;
    else
        return  n * factorial(n-1);
}
```

再帰呼出の例 2/2

n! を計算する関数

```
int  factorial(int  n)  {
    if  (n == 1)
        return 1;
    else
        return  n * factorial(n-1);
}
```





諸注意

　　このスライドは　2010年4月に　山中克久　博士
（FS専攻DB講座助教・当時）
がMIT教科書を参考にして
作成したオリジナル版を基にして　今回　大森が改訂したものです．優れた例題解説はオリジナル版の著者の貢献であり，もし　分かりにくいところがあれば全て改訂者の責任です．
この事実を明記して，スタッフの努力に謝意申し上げます．

2013年4月5日記．　　FS専攻DB講座　大森匡