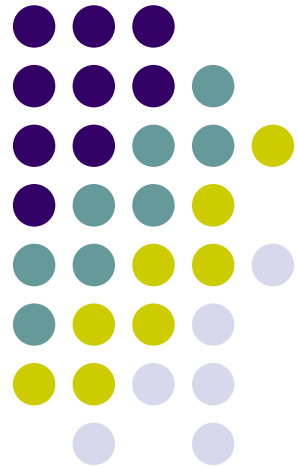
情報処理概論

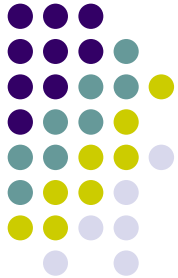
工学部 物質科学工学科
応用化学コース
機能物質化学クラス



第8回

2005年　6月　9日

前回の演習の解答例



- 多項式の計算(前半):

```
program poly
  implicit none
  integer, parameter :: number = 5
  real(8), dimension(0:number) :: a
  real(8) :: x, total
  integer :: i

  do i = 0, number
    write(*, *) 'a(', i,') = '
    read(*, *) a(i)
  end do
  write(*, *) 'x = '
  read(*, *) x
```

前回の演習の解答例

- 多項式の計算(後半):

```
  total = a(0)
  do i = 1, number
    total = total + a(i) * x**i
  end do

  write(*, *) 'total = ', total
stop
end program
```

演算回数（乗算） $= \sum_{i=1}^{number} i = \frac{number(number+1)}{2}$

演算回数（足し算） $= number$

number = 5 の時： 乗算15回　和算5回

前回の演習の解答例

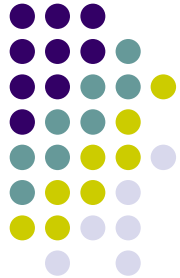
- 多項式の計算(後半)　計算量を削減:

$$a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$$
$$\rightarrow ((((a_5x + a_4)x + a_3)x + a_2)x + a_1)x + a_0$$

```
  total = a(number)
  do i = number - 1, 0, -1
    total = total * x + a(i)
  end do

  write(*, *) 'total = ', total
stop
end program
```

演算回数（乗算）　＝　***number***
演算回数（足し算）　＝　***number***

number = 5 の時:　乗算5回　和算5回



○　多次元配列の利用例

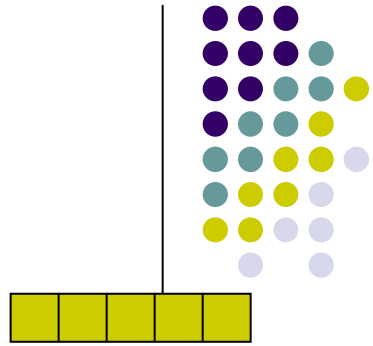
- 実行時に大きさの決まる配列
- ファイルからのデータ入力
- ファイルへのデータ出力

配列の次元

- １次元配列

```
integer, dimension(5) :: a
```

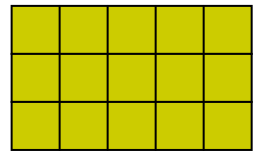
```
a(1) = 1
```



- ２次元配列

```
real(8), dimension(3, 0:4) :: b
```

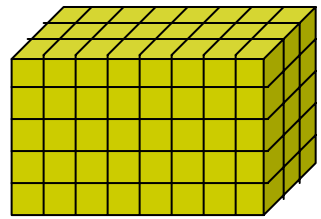
```
b(3, 2) = 1.0D0d
```



- ３次元配列

```
real(8), dimension(0:4, 8, 3) :: c
```

```
c(4, 8, 2) = 2.0D0
```

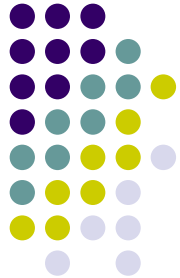


- ４次元配列

```
real(8), dimension(100, 50, 50, 3) :: d
```

```
d(i, j, k, l) = d(i, j, k, l) + 1.0D0
```

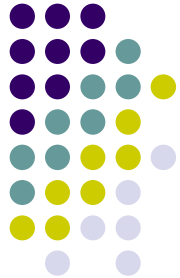
2次元配列の利用例



- 英語，数学，国語の3教科の点数を
  人数分入力し，合計点の総平均を求める
  - 人数を5人固定とし，
    点数をキーボードから入力

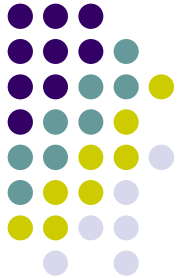
```
% ./ave
No. 1
Kamoku 1:
30
Kamoku 2:
60
Kamoku 3:
40

...

No. 5
Kamoku 1:
90
Kamoku 2:
85
Kamoku 3:
95
```

プログラム例1 (1/2)



```
program score1
  implicit none
  integer, parameter :: number = 5
  integer, parameter :: kamoku = 3
  integer :: i, j, total
  integer,dimension(number, kamoku) :: score
  real(8) :: ave
  intrinsic dble

  ! Input score data from keyboard
  do j = 1, number
    write(*, *) 'No.',j
    do i = 1, kamoku
      write(*, *) 'Kamoku ', i, ':'
      read(*, *) score(j, i)
    end do
  end do
```

プログラム例1 (2/2)

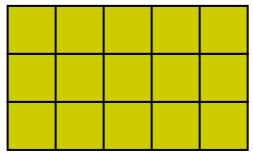


```
  ! Calculate total and average
  total = 0
  do i = 1, kamoku
    do j = 1, number
      total = total + score(j, i)
    end do
  end do
  ave = dble(total) / dble(number*kamoku)

  write(*, *) 'Average = ', ave

stop
end program
```

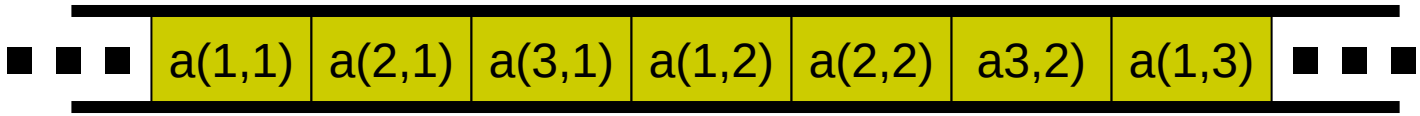
計算機内部の配列の姿とアクセス速度

- 多次元配列は一番左の添え字が先に増えるように記憶される
  - プログラムでの表記

```
real(8), dimension(3, 3) :: b
```

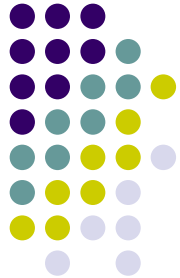


  - 計算機内部（メモリ）



- 計算機は連続したデータを高速にアクセスするようにできている.
  - アクセスの順番が重要

多次元配列の推奨参照順序



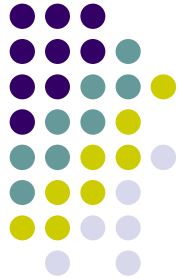
- できるだけ一番左の添え字に沿って参照するように繰り返しを構成する.

良い例

```
do j = 1, number
  do i = 1, kamoku
    total = total + score(j, i)
  end do
end do
```

悪い例

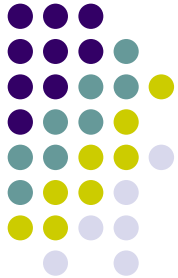
```
do i = 1, kamoku
  do j = 1, number
    total = total + score(j, i)
  end do
end do
```



・　多次元配列の利用例
○　実行時に大きさの決まる配列
・　ファイルからのデータ入力
・　ファイルへのデータ出力

実行時に大きさが決まる配列

- 同じプログラムで10人分のデータや100人分のデータを扱いたい



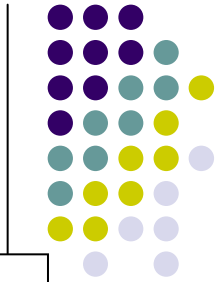
- 方法1： 毎回 emacs でプログラムを修正し，f90 で翻訳して実行ファイルを作成

```
program poly
  implicit none
  integer, parameter :: number = 5
  integer, parameter :: kamoku = 3
  integer :: i, j, total
  integer,dimension(number, kamoku) :: score
```

人数が変わるたびに変更

- 方法2： 実行時に配列の大きさを指定
  ⇒ 入力データを書き換えるだけでよい．

プログラム例2 (1/2)



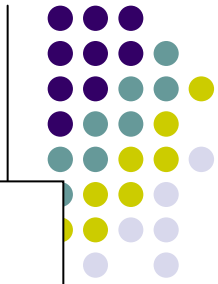
```
program score2
  implicit none
  integer, parameter :: kamoku = 3
  integer :: i, j, total, number
  integer, dimension(:,:), allocatable :: score
  real(8) :: ave
  intrinsic dble

  ! Input number from keyboard
  write(*, *) 'Number: '
  read(*, *) number

  ! Set size of array A
  allocate(score(number, kamoku))
```

大きさがまだ決まってない配列を宣言

まだ大きさ(範囲)を指定しない

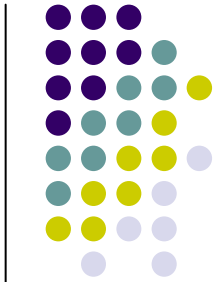
大きさ(範囲)の指定

プログラム例2 (2/2)



```
  ! Input score data from keyboard
  do j = 1, number
    write(*, *) 'No.',j
    do i = 1, kamoku
      write(*, *) 'Kamoku ', i, ':'
      read(*, *) score(j, i)
    end do
  end do

  ! Calculate total and average
  total = 0
  do i = 1, kamoku
    do j = 1, number
      total = total + score(j, i)
    end do
  end do
  ave = dble(total) / dble(number*kamoku)

  write(*, *) 'Average = ', ave
stop
end program
```

実行時に大きさが決まる配列



- 利用法： 宣言して大きさを指定する
  - 宣言

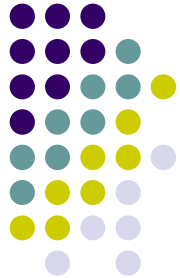
    ```
    型, dimension(:), allocatable :: 配列変数名
    ```

    - 範囲は次元一つにつき ： を一つ記述
      例） 2次元の場合 (: , :)

  - 大きさ(範囲)の指定

    ```
    allocate(配列変数(範囲))
    ```

    - 配列変数： allocatable付きで宣言した配列
    - 範囲の指定方法は配列を宣言するときと同様
    - 配列を利用する前に指定する



- 多次元配列の利用例
- 実行時に大きさの決まる配列
- ○ ファイルからのデータ入力
- ファイルへのデータ出力

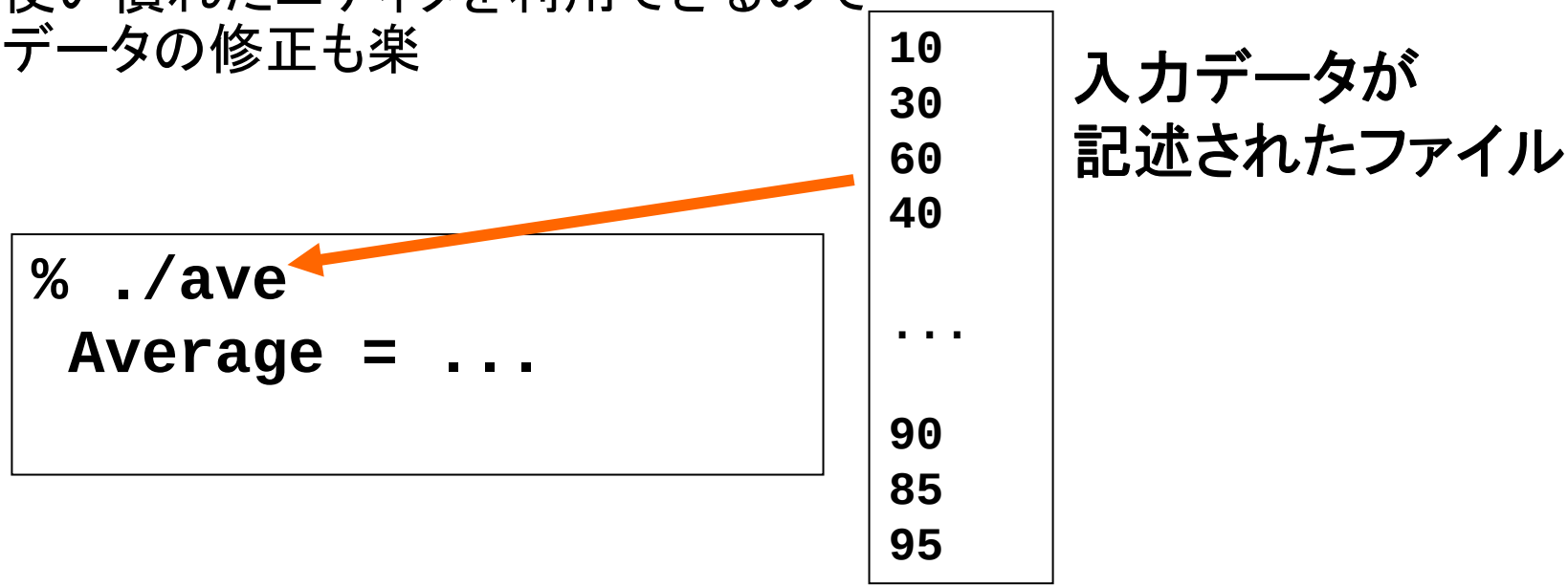
キーボードからの入力

- 大量のデータを入力する場合非効率
  - 人間が打つので間違える可能性大
    間違えた場合の修正も困難
  - 速度も遅い
  - 何度も実行するプログラムで
    毎回データを入力するのは無駄

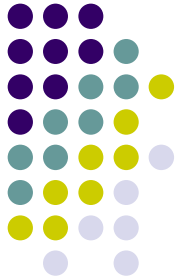
```
% ./ave
Number:
10
No. 1
Kamoku 1:
30
Kamoku 2:
60
Kamoku 3:
40

...

No. 10
Kamoku 1:
90
Kamoku 2:
85
Kamoku 3:
95
Average = 65.3
```

ファイルからの入力

- ファイルに記述されたデータを
  プログラムの入力データとして利用
  - 一度入力すれば何度でも利用できる
  - 様々なデータを利用できる
    - インターネットからダウンロードしたファイル
    - メールに添付されたファイル
    - スキャナや実験装置から得られたデータ
  - 使い慣れたエディタを利用できるので
    データの修正も楽



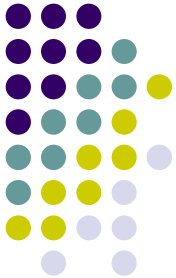
ファイルから入力する方法



- 方法1： UNIXの”リダイレクション”機能を利用
  - キーボードから直接データを入力するプログラムをそのまま利用できる
    - 実行時に選択可能
  - データの数が少ない場合に利用

- 方法2： Fortranのファイル操作命令を利用
  - 必ずファイルからデータを入力
  - データの数が多い場合に利用

方法1: UNIXの”リダイレクション”機能を用いたデータ入力



- コマンドを実行する際
  キーボードからの入力の代わりに
  ファイルに記述したデータを入力する

- 利用法:

```
実行コマンド  <  入力データのファイル
```

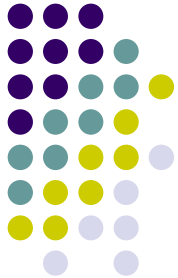
  - 例)

```
% ./ave < score.dat
```

```
10
30
60
40

...

90
85
95
```

score.dat
（Emacs等で
別途作成）

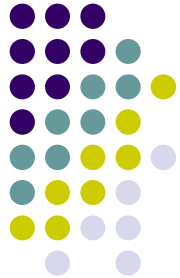
リダイレクション利用時の画面表示



- 入力データ以外は全て表示される.
  - Kamoku1: のような表示も残る

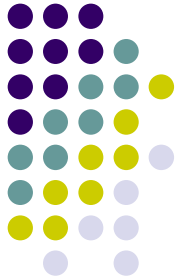
```
% ./ave < score.dat
Number:
No. 1
Kamoku 1:
...
Kamoku 3:
Average = 65.3
%
```

方法2： Fortranのファイル操作命令を用いたデータ入力



- プログラムの中で
  - ファイルを開いて (open)
  - データを読んで (read)
  - ファイルを閉じる(close)

プログラム例3 (1/2)



```
program score3
  implicit none
  integer, parameter :: kamoku = 3
  integer :: i, j, total, number
  integer, dimension(:,:), allocatable :: score
  real(8) :: ave
  intrinsic dble

  open(10, file="score.dat")

  ! Read number from data file
  read(10, *) number

  allocate(score(number, kamoku) )
```

score.dat という名前のファイルを開いて
ファイル番号 10 を割り当てる

ファイル番号 10 からデータ入力

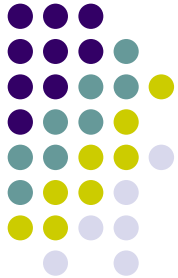
プログラム例3 (2/2)

```
  do j = 1, number
      read(10, *) score(j, 1:kamoku)
  end do
  close(10)

  total = 0
  do i = 1, kamoku
    do j = 1, number
      total = total + score(j, i)
    end do
  end do
  ave = dble(total) / dble(number*kamoku)

  write(*, *) 'Average = ', ave

stop
end program
```

ファイル番号 10 からデータ入力（一人分ずつ）

ファイル番号 10 を閉じる

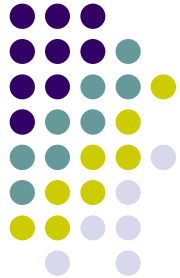
ファイルを開く: open



- ファイルにアクセスするための番号を付ける
- 利用法

```
open(番号，file = "ファイル名")
```

  - 番号：正の整数.
    慣習的に10以上の番号を付けることが多い
  - ファイル名：別のディレクトリにあるときは場所も書く



ファイルを閉じる: close

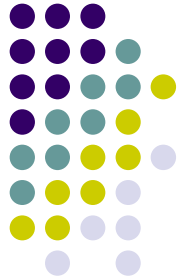
- ファイルへのアクセス終了

- 利用法

```
close(番号)
```

  - 番号： open で開いたファイルの番号

ファイルからデータを読む: read



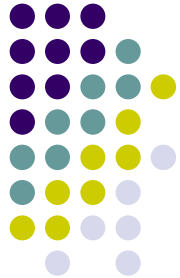
- 利用法

read(番号, 書式) データを格納する変数

  - 番号: ファイルの番号を指定
    ＊を指定するとキーボードから入力

  - 書式: 今まで read や write で指定したものと同じ
    ＊を指定すると書式指定無し(おまかせ)

read による配列データの入力



- 方法A： 1要素ずつ入力

```
do j = 1, number
  do i = 1, kamoku
    read(10, *) score(j, i)
  end do
end do
```

- 方法B： まとめて入力

```
do j = 1, number
    read(10, *) score(j, 1:kamoku)
end do
```

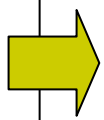
データを読み込む場合の注意

- 1行分のデータを1回のread文で全て読み込む
  - 1回の read で1行文のデータがプログラムに取り込まれるので，全て格納しないと読み落としが発生する．
    - 例） 以下のデータを読み込み

```
10 20 30
40 50 60
```

方法A（1要素ずつ）では読み落としが発生

```
do i = 1, 2
  do j = 1, 3
    read(10, *) score(i, j)
  end do
end do
```



score(1,1)に1行目
score(1,2)に2行目

score(1,3)以降に入力するデータが不足

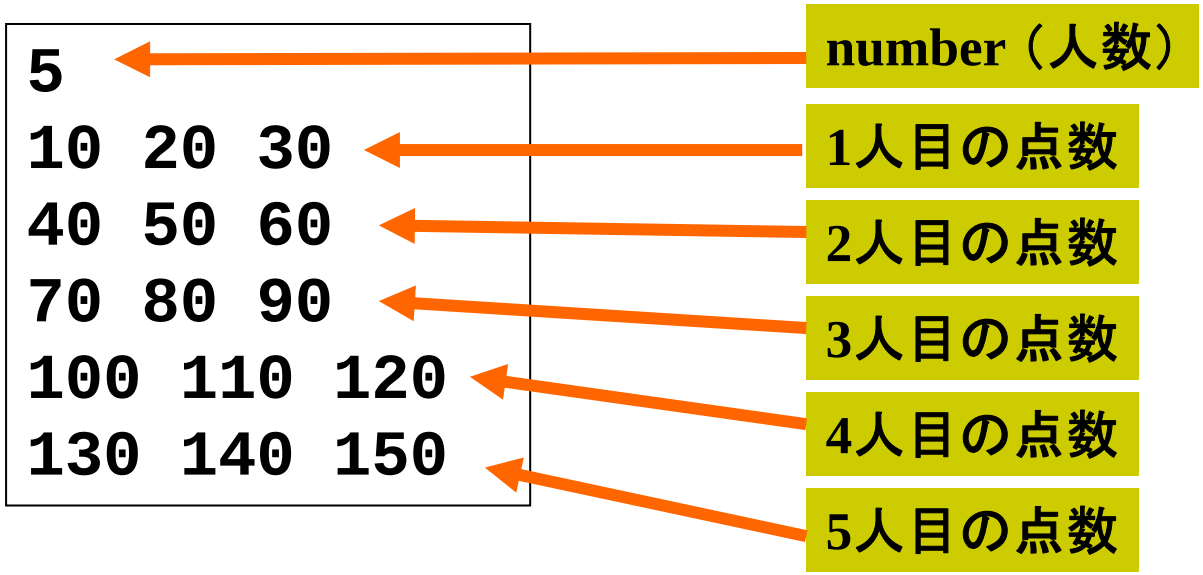
方法B（1行ずつ）で読み込む

```
do i = 1, 2
  read(10, *) score(i, 1:3)
end do
```

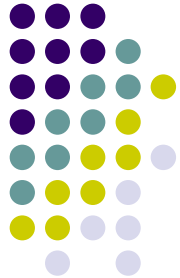
score.dat の例

```
do j = 1, number
    read(10, *) score(j, 1:kamoku)
end do
```

方法B で入力するので, 一行に一人分のデータを全て記述しておく

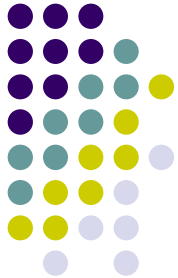


プログラム実行前に用意（Emacs 等で作成）



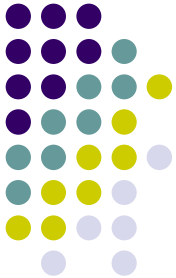
- 多次元配列の利用例
- 実行時に大きさの決まる配列
- ファイルからのデータ入力
- ファイルへのデータ出力

出力データ



- 画面に表示されたデータは（原則として）保存されない.
  - ログアウトすると消えてしまう.
- 保存したいデータは画面ではなくファイルに出力
  - 画面に収まりきれない量のデータ
  - 他のプログラムで利用したいデータ
  - 成果として公表したいデータ
    etc.
- 入力と同様，２通りの方法
  - 方法1： UNIXのリダイレクション機能を利用
  - 方法2： Fortranのファイル操作命令を利用

方法1: UNIXの”リダイレクション”機能を用いたデータ出力



- コマンドを実行する際
  画面に出力する代わりに
  ファイルにデータを格納する

- 利用法:

```
実行コマンド ＞ 出力ファイル
```

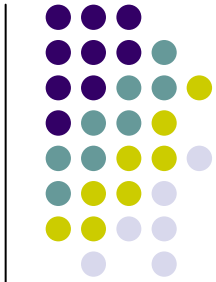
  - 例)

```
% ./ave > score.out
```

  - 例） 入出力の組み合わせ

```
% ./ave < score.dat > score.out
```

方法2: Fortran のファイル操作命令を利用したデータ出力

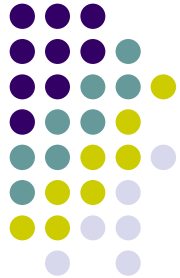


- プログラムの中で
  - ファイルを開き (open)
  - データを書いて (write)
  - ファイルを閉じる (close)

- open と close は入力のときと同じ
  - 通常 read するファイルと write するファイルは別なので, それぞれ open, close を行う.

注意: 同時に同じ番号でファイルを開くことは不可

ファイルにデータを書き出す



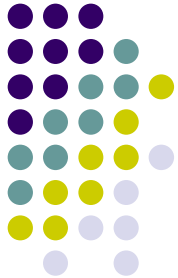
- 利用法

```
write(番号，書式) 出力データ
```

  - 番号：　ファイルの番号を指定する
    ＊を指定すると画面に出力

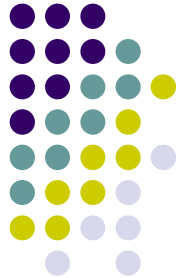
  - 書式：　今まで read や write で指定したものと同じ
    ＊を指定すると書式指定無し (おまかせ)

プログラム例4 (1/2)



```
program score3
  implicit none
  integer, parameter :: kamoku = 3
  integer :: i, j, total, number
  integer, dimension(:,:), allocatable :: score
  real(8) :: ave
  intrinsic dble

  open(10, file="score.dat")

  ! Read number from data file
  read(10, *) number

  allocate(score(number, kamoku) )
```

プログラム例4 (2/2)



```
  do j = 1, number
      read(10, *) score(j, 1:kamoku)
  end do
  close(10)

  total = 0
  do i = 1, kamoku
    do j = 1, number
      total = total + score(j, i)
    end do
  end do
  ave = dble(total) / dble(number*kamoku)

  open(11, file = "score.out")
  write(*, *) 'Average = ', ave
  close(11)

stop
end program
```

演習1

1. 次のプログラムを入力し,コンパイル

```
program ex1
  implicit none
  integer, parameter :: number = 5
  integer :: i, total
  integer,dimension(number) :: score
  intrinsic dble
  total = 0
  do i = 1, number
    read(*, *) score(i)
    total = total + score(i)
  end do
  write(*, *) 'Average = ', dble(total)/dble(number)
stop
end program
```

演習1

2. 次のファイルを作成する (emacs コマンド)

```
98
23
17
100
45
```

ファイル名は任意. 例えば test.dat

```
% emacs test.dat
```

3. 実行

a) キーボードから点数を直接入力

b) 2.で作成したデータファイル test.dat をリダイレクション機能を用いて入力

演習2：2次元配列を用いたプログラム

- 学生の人数と各学生の3教科分の点数を入力すると，各教科の平均とすべての平均を表示する
  - データはファイルから入力する
  - 入力データ例

```
5
10 20 30
40 50 60
70 80 90
100 110 120
130 140 150
```

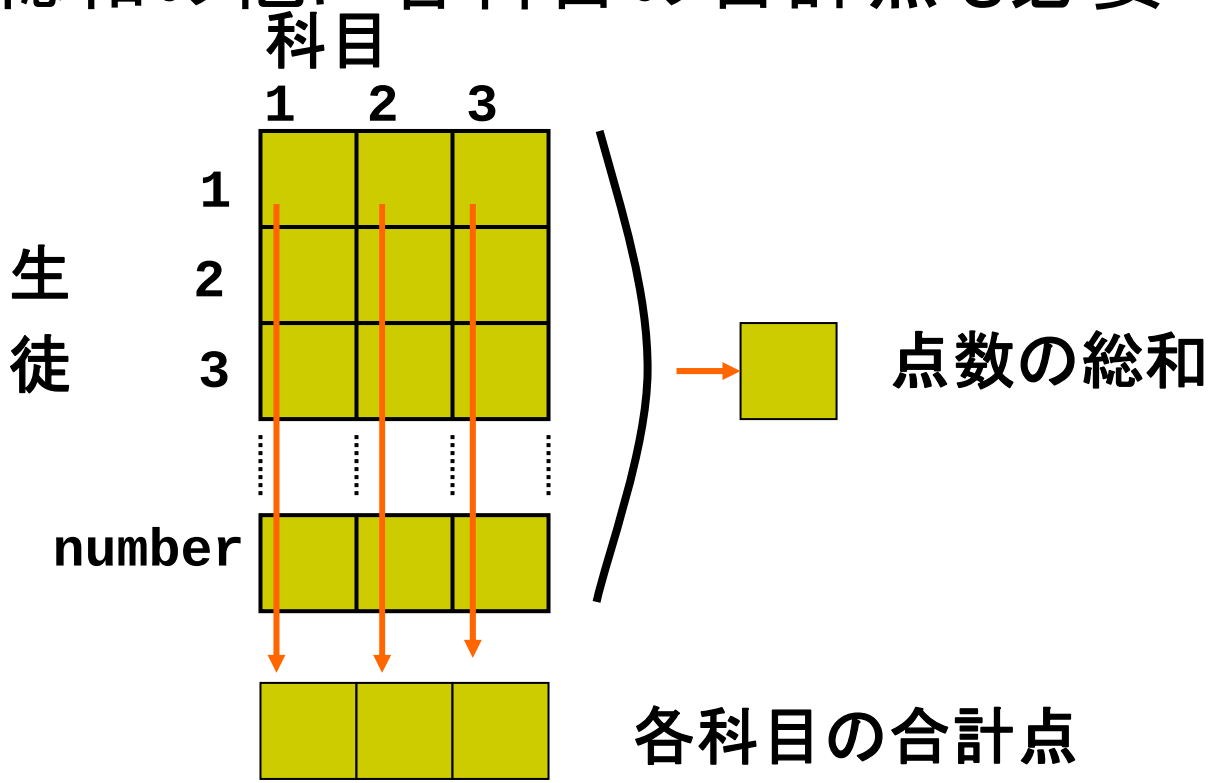
  - 実行例

```
% ./ave3
Average=80.00
 Kamoku 1. 70.00
 Kamoku 2. 80.00
 Kamoku 3. 90.00
%
```

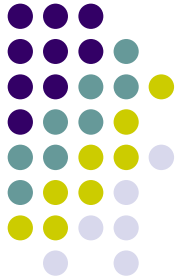
演習2: ヒント

- 講義で紹介したプログラム例3またはプログラム例4を参考にする

- 点数の総和の他に各科目の合計点も必要



補足:
複数の配列要素に対する参照,代入



Fortran90では配列の複数の要素に対する値の代入や参照 , 計算を簡単に記述できる「省略形」が利用できる

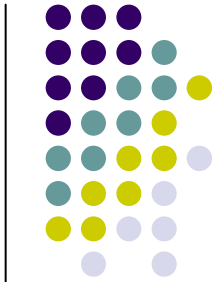
- 例えば配列　a の全要素に 0.0D0 を代入する場合
  do文等で繰り返して参照,代入してもよいが
  省略することも出来る
  - do文を利用

```
do i = 1, n
  do j = 1, n
    a(j, i) = 0.0D0
  end do
end do
```

  - 省略形

```
a = 0.0D0
```

配列全体に対する代入の省略形



- 全部に同じ値を代入

```
a = 0.0D0
```

- 一次元配列の場合：
  - 各要素に任意の値を代入

```
a = (/1.0D0, 0.2D0, -3.5D0, 0.0D0, 12.5D0/)
```

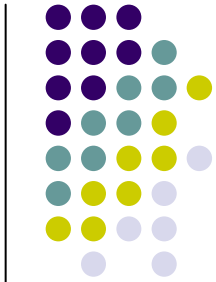
  - 規則的な値を代入
    - 省略形

```
a = (/ (2 * i, i = 1, 5) /)
```

以下の do文と同じ意味

```
do i = 1, 5
  a(i) = 2 * i
end do
```

配列の一部に対する代入の省略形



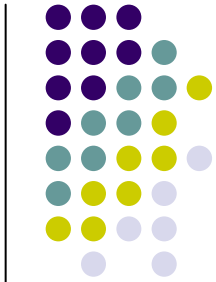
- 2列目の5行目から10行目の要素に 1.0D0 を代入

```
a(5:10,2) = 0.0D0
```

- 偶数の要素に, 1, 3, 5, 7, 9 と
  奇数を順に代入

```
a(2:n:2) = (/ (i, i = 1,n,2) /)
```

省略形の利用例： 参照と代入



- 配列 a に配列 b をコピー
  - ただし, a と b の要素数が同じ場合に限る

```
a = b
```

- 配列 a に配列 b の各要素を2倍したものを格納
  - ただし, a と b の要素数が同じ場合に限る

```
a = b * 2
```

- 配列 a の全要素の総和を計算
  - sum は intrinsic で予め宣言すること

```
total = sum(a)
```

多次元配列の一括入力，出力

- 多次元配列の値を一度に入力，出力することもできる

```
integer, dimension(5, 3) :: a
```

```
read(10, *) a
```

```
write(*, *) a
```

- この場合，「左の添え字から増える」順序で行われる
  - 上の例では，入出力の順番は以下の通り
    a(1,1) → a(2,1) → a(3,1) → a(4,1) →
    a(5,1) → a(1,2) → a(2,2) → a(3,2) → ...
- 入力データは一行に一要素ずつ記述する

```
10
40
70
...
```

でも順番を間違えやすいので
通常は do文で順序を指定することが多い