

# 情報処理概論

第8回 ファイル入出力

情報基盤研究開発センター 谷本 輝夫

# 今日の予習プログラム (1/2)

```
program msum
implicit none
integer :: i, j, m, n
real(8),dimension(:,:),allocatable :: a, b, c

open(10, file='a.dat')
open(11, file='b.dat')
read(10, *) m
read(11, *) n

if (m /= n) then
  write(*, *) "Error: Sizes of matrices are different."
  stop
end if

allocate(a(n,n))
allocate(b(n,n))
allocate(c(n,n))
```

# 今日の予習プログラム (2/2)

```
do i = 1, n
  read(10, *) a(i,1:n)
  read(11, *) b(i,1:n)
end do
close(10)
close(11)

c = a + b

open(12, file='c.out')
do i = 1, n
  write(12, *) c(i, 1:n)
end do
close(12)

stop
end program
```

さらに、次ページの二つのファイルも入力する

# 今日の予習プログラム

- ▶ データファイルを2個作成

ファイル名 a.dat

```
3
1 2 3
4 5 6
7 8 9
```

ファイル名 b.dat

```
3
1 1 1
2 2 2
3 3 3
```

- ▶ ファイルの作り方： プログラムの作成と同じ

```
$ emacs ファイル名
```

- ▶ プログラムを実行すると c.out というファイルが出来るので、内容を less か emacs で確認

# 今回の内容

- ▶ ファイルからの read
- ▶ ファイルへの write

# キーボードからの入力の問題点

- ▶ 大量のデータを入力する場合に非効率

なぜ非効率か？

```
% ./msum
Enter n:
2
A( 1, 1):
1
A( 1, 2):
2
A( 2, 1):
3
A( 2, 2):
4
B( 1, 1):
1
B( 1, 2):
1
B( 2, 1):
1
B( 2, 2):
1
```

# データファイルの利用

- ▶ データを別途ファイルとして入力し保存  
⇒ プログラムの入カデータとして利用

プログラム

```
% ./score4  
Average = ...
```

データファイル

```
10  
30  
60  
40  
...  
90  
85  
95
```

読み込み

# ファイルによる入力の利点

- ▶ 一度入力すれば**何度でも利用可能**
- ▶ 入力や修正が簡単
  - ▶ emacs など、使い慣れたエディタを利用
  - ▶ **途中で間違っても修正が容易**
- ▶ 様々なデータを利用可能
  - ▶ 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



# プログラム例 1 (1/2)

```
program score4
  implicit none
  integer, parameter :: kamoku = 3
  integer :: i, j, total, number
  integer, dimension(:, :), allocatable :: score
  real(8) :: ave
  intrinsic dble
```

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

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

! Read number from data file

```
read(10, *) number
```

ファイル番号 10 からデータ入力

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

このプログラムとは別に、score.dat という名前でデータファイルを (emacsで) 作成する

# プログラム例 1 (2/2)

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

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

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

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

stop
end program
```

# ファイルからのデータ入力の手順

- ▶ ファイルを開く (open)
  - ▶ ファイルに番号が付く
- ▶ データを読む (read)
  - ▶ 指定した番号のファイルからデータを読む
- ▶ ファイルを閉じる(close)
  - ▶ 番号とファイルの対応付けが無くなる  
= 同じ番号を別の open に利用できるようになる

# open ファイルを開く

- ▶ プログラムから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
  - ▶ 番号のうち5と6は既に別の用途で使われているので避ける
- ▶ ファイル名
  - ▶ と言うより、ファイルの場所
  - ▶ 別のディレクトリにあるときはディレクトリ名を付けて書く

```
open(10, file="data/score.dat")
```

# read ファイルからデータを読む

## ▶ 利用法

**read(番号, 書式) データを格納する変数**

- ▶ 番号 : open で指定したファイルの番号
  - ▶ \* を指定すると, キーボードから入力
- ▶ 書式 : 通常は\* のままで良い
  - ▶ \* を指定すると書式指定無し
  - ▶ Write と同様の書式を指定することもできるが、データもその書式の通りに並んでいなければならない

# close ファイルを閉じる

▶ ファイルへのアクセス終了

▶ 利用法

```
close(番号)
```

▶ 番号 : open で指定した番号

# データを読み込む場合の注意

- ▶ 1 回の read で1行分のデータが全て読み込まれる
  - ▶ 1 回の read で1行文のデータを全て格納しないと読み落としが発生
- ▶ 例) 以下のデータファイルを入力する場合

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

- ▶ 1 回のread で 1 行分の全データが読み込まれるので

```
10 20 30
```

- ▶ 1回あたり 3 個分の格納場所を指定する

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

または

```
read(10, *) a, b, c
read(10, *) d, e, f
```

# 配列データの入力方法

- ▶ 方法A： 1要素ずつ入力

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

データファイル

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

- ▶ 方法B： 1行分まとめて入力

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

データファイル

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



# 間違いの例

- ▶ 1行のデータ数と readに指定された格納場所の数が一致しない  
例)

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

データファイル

10	20	30
40	50	60

score(1,1)に1行目の最初の要素 10  
score(1,2)に2行目の最初の要素 40  
がそれぞれ格納され、20, 30, 50, 60はどこにも格納されない

- ▶ しかも、score(1,3)のデータを読み込もうとしてデータが不足するのでエラーが発生

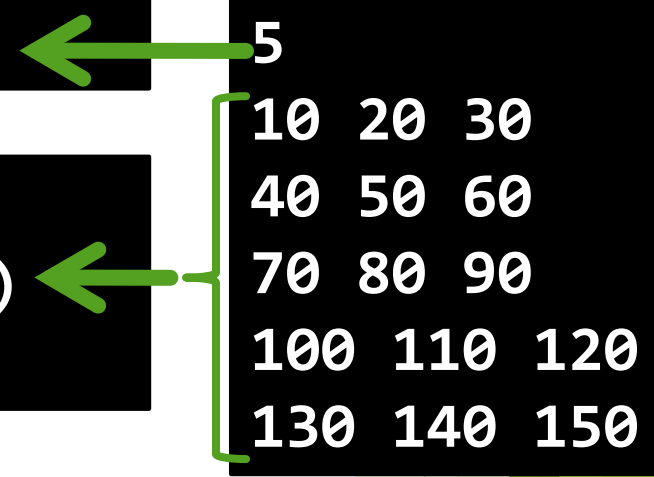
# プログラム例 1 の場合

- ▶ プログラム中の read とデータファイルの内容の対応関係

```
read(10, *) number
```

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

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



# 今回の内容

- ▶ ファイルからの read
- ▶ ファイルへの write

# ファイルへの write

- ▶ 計算結果をデータとしてファイルに保存
  - ▶ 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.



- ▶ 画面ではなくファイルに write する

# ファイルへのデータ出力

- ▶ ファイルを開く (open)
  - ▶ ファイルに番号が付く
- ▶ データを書き出す (write)
  - ▶ 指定した番号のファイルにデータを書き出す
- ▶ ファイルを閉じる (close)
  - ▶ 番号とファイルの対応付けが無くなる

# 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

## プログラム例 2 (1/2)

```
program score5
  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 2 (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(11, *) 'Average = ', ave
close(11)

stop
end program
```



# 今日の演習

- ▶ 「n人の身長と体重をファイルから読み込み、BMIを計算してファイルに書き出す」
  - ▶ 人数nもファイルから読み込む
  - ▶ BMIの計算式は、

$$BMI = \text{体重}[\text{kg}] \div (\text{身長}[\text{m}])^2$$

- ▶ 時間に余裕がある人は、以下にも挑戦
  - ▶ 「BMIが普通体重の範囲外であるものについては、マークを併せて表示する」
  - ▶ 但し、日本人の適正範囲（普通体重）は、18.5以上、25未満