itpass 実習数値計算編　2016 年 9 月 16 日
2016 年 9 月 23 日
2016 年 9 月 30 日

# Fortran(90) を使ってみよう

# 目次

- 準備
- とりあえずプログラムを書いて実行しよう
  - Hello World
- 変数
- 簡単な入出力
- 簡単な演算
- 条件分岐
- 繰り返し
- グラフを描いてみよう（本来, Fortran とは無関係）
  - gnuplot
- 配列
- サブルーチン・関数

# 準備

- 使用する計算機に Fortran コンパイラがインストールされていますか?
  - 例えば, 下のように打ってみましょう.

    $ gfortran
- インストールされていなければ, 適宜インストールしてください.
  - 例えば,

    $ sudo apt-get install gfortran

# とりあえずプログラムを書いて実行しよう

# とりあえずプログラムを書いて動かしてみよう (1)

- 好きなエディタを使って, 下のようなプログラムを hello.f90 というファイル名で作成しよう.

```
program hello
  ! Hello world
  implicit none

  write( 6, * ) 'Hello world'

end program hello
```

# とりあえずプログラムを書いて実行してみよう (2)

- コンパイルしよう.

  $ gfortran hello.f90

- 実行しよう.

  $ ./a.out

- なお, 下のようにすることで, 実行ファイル名を指定することができる.

  $ gfortran –o hello hello.f90
  $ ./hello

# とりあえずプログラムを書いて実行してみよう (3)

- 注意
  - “!” 以降はコメント文.
  - Fortran では大文字と小文字は区別されない.

# 変数

# 変数 (0)

- 下のようなプログラムを variables1.f90 というファイル名で作成しよう.

```
program variables1
  implicit none
  number = 3
  value = 5
  write( 6, * ) number, value, number * value
end program variables1
```

# 変数 (1)

- コンパイルして実行しよう.
  - しかし, このプログラムはコンパイルできません.

- 理由は, 変数を宣言していないためです.

# 変数 (2)

- variables1.f90 を変更して実行しよう.

```
program variables1
  implicit none
  integer :: number
  real(4) :: value
  number = 3
  value = 5
  write( 6, * ) number, value, number * value
end program variables1
```

# 変数 (3)

- Fortran では, 変数は宣言してから使用することが望ましい.
  - 「Fortran では, 変数は宣言しなければ使用できない」と書きたいのだが, Fortran の仕様としては, 宣言しなくても使える.
  - “implicit none” と書くことで, 宣言が必須になって便利.
- Fortran で使える変数
  - 整数 (integer)
  - 単/倍精度実数 (real(4), float / real(8), double)
  - 文字 (character)
  - 論理型 (logical)
  - 複素数 (complex(4), complex(8))

# 変数 (4)

- variables1.f90 を変更して実行しよう.

```
program variables1
  implicit none
  integer :: number
  real(4) :: value4
  real(8) :: value8
  character(20) :: line
  number = 3
  value4 = 5
  value8 = 5
  line = “test program”
  write( 6, * ) number, value4, value8
  write( 6, * ) line
end program variables1
```

# 変数 (5)

- 下の内容の variables2.f90 を作成して実行しよう.

```
program variables2
  implicit none
  integer :: num1, num2
  real(4) :: val41, val42
  real(8) :: val81, val82
  num1 = 3
  num2 = 5
  val41 = 3.0
  val42 = 5.0
  val81 = 3.0d0
  val82 = 5.0d0
  write( 6, * ) num1, num2, num1/num2
  write( 6, * ) val41, val42, val41/val42
  write( 6, * ) val81, val82, val81/val82
end program variables2
```

# 変数 (6)

- 注意
  - 整数同士の演算の結果は整数になる.
  - 整数, 単(倍)精度実数を含む演算の結果は単(倍)精度実数になる.

# 簡単な入出力

# 簡単な入出力 (1)

- 下の内容の io1.f90 を作成して実行しよう.

```
program io1
  implicit none
  integer :: number
  write( 6, * ) “Input a number:”
  read( 5, * ) number
  write( 6, * ) number
end program io1
```

# 簡単な入出力 (2)

- 下の内容の io2.f90 を作成して実行しよう.

```
program io2
  implicit none
  integer :: number
  write( 6, * ) “Input a number:”
  read( 5, * ) number
  open( 50, file=‘output.txt’, status=‘unknown’ )
  write( 50, * ) number
  close( 50 )
end program io2
```

# 簡単な入出力 (3)

- 準備として, 下の内容のデータファイル, input.data, を作成しよう.

```
5
10
15
```

# 簡単な入出力 (4)

- 下の内容の io3.f90 を作成して実行しよう.

```
program io3
  implicit none
  integer :: number
  open( 50, file='input.data', status='unknown' )
  read( 50, * ) number
  write( 6, * ) number
  read( 50, * ) number
  write( 6, * ) number
  read( 50, * ) number
  write( 6, * ) number
  close( 50 )
end program io3
```

# 練習問題 1

- input.data に保存されている数字の和を求めるプログラムを作りなさい.

# 簡単な演算

# 簡単な演算 (1)

- 下の内容の calc.f90 を作成して実行しよう.

```
program calc
  implicit none
  real(8), parameter :: pi = 3.141592d0
  write( 6, * ) sin( 0.0d0 ), sin(pi/2.0d0)
  write( 6, * ) 2**5, 2**0.5, int( 2.5 )
  write( 6, * ) sqrt( 2.0 ), exp( 2.0 ), &
                log( 100.0 ), log10( 100.0 )
  write( 6, * ) mod( 10, 3 )
end program calc
```

# 簡単な演算 (2)

- 注意
  - 長い行は, 行の最後に “&” を打つことで継続される.
  - Fortran では, 1 行の長さには上限がある（Fortran90 では 132 文字?）. この上限にひっかかるのを避けるためにも “&” を使うことができる.
    - 実際の長さの上限は, コンパイラ依存かもしれない.

# 練習問題 2

- 下は, 半径 2 の円の円周を計算するプログラムである. これを, 円筒の体積を計算するプログラムに変更しなさい. ただし, 円筒の半径と高さはキーボードから入力できるようにすること.

```
program calc2
  implicit none
  real(8), parameter :: pi = 3.141592d0
  real(8) :: radius
  radius = 2.0d0
  write( 6, * ) 2.0d0 * pi * radius
end program calc2
```

# 条件分岐

# 条件分岐 (1)

- 下の内容の ifelse.f90 を作成して実行しよう.

```
program ifelse
  implicit none
  real(8) :: value
  write( 6, * ) “Input a number:”
  read( 5, * ) value
  if ( value < -10.0d0 ) then
    write( 6, * ) “Less than -10”
  else if ( value < 0.0d0 ) then
    write( 6, * ) “Larger than -10 and less than 0”
  else
    write( 6, * ) “Positive”
  end if
end program ifelse
```

# 条件分岐 (2)

- 条件分岐で使える記号
  - A と B が等しい　　A == B　　( A .eq. B )
  - A と B が等しくない　　A /= B　　( A .ne. B )
  - A は B より大きい　　A > B　　( A .gt. B )
  - A は B 以上　　A >= B　　( A .ge. B )
  - A は B より小さい　　A < B　　( A .lt. B )
  - A は B 以下　　A <= B　　( A .le. B )
  - A は B 以下, かつ C は D より大きい
    ( A <= B ) .and. ( C > D )
  - A は B 以下, または A は D 以上
    ( A <= B ) .or. ( A >= D )

# 練習問題 3

- 二つの整数をキーボードから入力し, 一つ目の整数が二つ目の整数で割り切れるかどうかを調べるプログラムを作りなさい.
  - 例えば, 割り切れるときには “OK” と表示し, 割り切れないときには “NG” と表示しなさい.

# 繰り返し

# 繰り返し (1)

- 下の内容の loop.f90 を作成して実行しよう.

```
program loop
  implicit none
  integer :: i
  do i = 1, 5
    write( 6, * ) i
  end do
end program loop
```

# 繰り返し (2)

- loop.f90 を変更して実行しよう.

```
program loop
  implicit none
  integer :: i
  do i = 1, 50, 10
    write( 6, * ) i
  end do
end program loop
```

# 繰り返し (3)

- loop.f90 を変更して実行しよう.

```
program loop
  implicit none
  integer :: i
  do i = 1, 50, 10
    if ( i > 20 ) then
      exit
    end if
    write( 6, * ) i
  end do
  write( 6, * ) ‘finish’
end program loop
```

# 練習問題 4

1. 1 から 40 までの整数を順に表示するプログラムを作りなさい. ただし, 3 の倍数および 3 がつく数字の時だけ「アホになる」こと.
   - 「アホになる」＝「”FOOL!” と出力する」
2. 1 から 10 までの整数の和を求めるプログラムを作りなさい.
3. 5! を計算するプログラムを作りなさい.
4. $\frac{1.5}{1!}, \frac{1.5^2}{2!}, \frac{1.5^3}{3!}, \ldots, \frac{1.5^{10}}{10!}$ の和を求めるプログラムを作りなさい.

# グラフを描いてみよう

# グラフを描いてみよう (0-1)

- 使用する計算機に gnuplot がインストールされていますか?
  - 例えば, 下のように打ってみましょう.

    $ gnuplot
- インストールされていなければ, 適宜インストールしてください.
  - 例えば,

    $ sudo apt-get install gnuplot-x11

# グラフを描いてみよう (0-2)

- gnuplot が起動したら下のように打ってみましょう.

  gnuplot> plot sin(x)

- グラフのウィンドウが現れたら成功.
- 現れなければ対策が必要です.
  - gnuplot が適切にインストールされていますか?
    - debian では, gnuplot パッケージではなく, gnuplot-x11 パッケージが必要です.
  - Xming などをインストールする必要があるでしょう（Windows の場合）.
  - TeraTerm などからリモートアクセスするときは, XForwarding 機能を有効にする必要があるでしょう.
- gnuplot を終了するには下のようにします.

  gnuplot> quit

# グラフを描いてみよう (0-3)

- 下の URL にあるファイルをダウンロードして, ファイル内のデータを確認しよう.

  https://itpass.scitec.kobe-u.ac.jp/seminar/lecture/fy2016/160916/pub/wangara.tgz

  – このファイルには, Wangara での境界層観測で得られた温度と圧力の分布が保存されている.
  – 例えば,

```
$ wget http://itpass.scitec.... (上の URL)
$ tar xvf wangara.tgz
$ lv wangara/Data_Aug17LT12H.data
```

# グラフを描いてみよう (1)

- ダウンロードしてデータのグラフを描いてみよう.

  $ gnuplot

  …

  gnuplot> plot “wangara/Data_Aug17LT12H.data” w l

# グラフを描いてみよう (2)

- グラフの x 軸, y 軸を入れ替えてみよう.

```
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 2:1 w l
```

- 温度-圧力分布を描いてみよう.

```
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w l
```

- y 軸の向きと範囲を変えてみよう.

```
gnuplot> set yrange [102000:90000]
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w l
```

# グラフを描いてみよう (3)

- 点グラフにしよう.

```
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w p
```

- 線と点を両方描こう.

```
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w lp
```

- y 軸を対数軸にしてみよう
  - この場合にはあまり意味はないが.

```
gnuplot> set logscale y
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w lp
```

# グラフを描いてみよう (4)

- x 軸, y 軸の名前を書こう.

```
gnuplot> set xlabel “Temperature (K)”
gnuplot> set ylabel “Pressure (Pa)”
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w p
```

- 二本の線を一つの図に描こう.

```
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w lp,
        “wangara/Data_Aug17LT15H.data” u 3:2 w lp
```

  - 注意: 改行せずに一行に書くこと.

# グラフを描いてみよう (5)

- 図をファイルに出力しよう.

```
gnuplot> set term png
gnuplot> set output “gnuplot.png”
gnuplot> plot “wangara/Data_Aug17LT12H.data” u 3:2 w p
```

- 出力した png ファイルは下のように表示できる.
  $ eog gnuplot.png
- 他にも, 下のような terminal を選べる（”png” の部分）.
  gif, postscript, pdfcairo, …
- それぞれに合わせて output を指定すること.

# グラフを描いてみよう (6)

- 下の内容の plot.gp を作ろう.

```
set xlabel “Temperature (K)”
set ylabel “Pressure (Pa)”
set yrange [102000:90000]
set logscale y
plot “wangara/Data_Aug17LT12H.data” u 3:2 w lp,
        “wangara/Data_Aug17LT15H.data” u 3:2 w lp
```

- これを使って gnuplot で図を描こう.

```
$ gnuplot
…
gnuplot> load”plot.gp”
```

# グラフを描いてみよう (7)

- gnuplot の help を活用しよう.
  - 例えば, plot のオプションなどを調べたいとき.

    gnuplot> ? plot
  - さらに続けて, plot につける with について調べたいとき.

    Subtopic of plot: with

# 練習問題 5-1

- 0 次元モデルでの地球の熱平衡状態を, 図を描いて考察しよう.
  - 0 次元モデルでは, 地球の熱収支は下の式によって与えられる. (記号の意味はよくあるやつです.)

$$\sigma T^4 = \frac{1}{4}(1 - A)F_s$$

  - アルベドが 0.3 のとき, 200 K から 300 K の間での左辺(射出量)と右辺(吸収量)の値を計算してファイルに出力するプログラムを作り, 横軸を射出量/吸収量, 縦軸を熱フラックスとしたグラフを描くことで, 平衡状態があることを示しなさい.

# 練習問題 5-2

- 0 次元モデルでの地球の熱平衡状態を, 図を描いて考察しよう.
  - 先の問題において, アルベドが下のように温度に依存するとき, 200 K から 300 K の間での左辺(射出量)と右辺(吸収量)の値を計算してファイルに出力するプログラムを作り, 横軸を射出量/吸収量, 縦軸を熱フラックスとしたグラフを描くことで, 平衡状態が複数あることを示しなさい.

$$A = \begin{cases} 0.7 & (T < 230\ K) \\ \text{線形で変化} & (230\ K \leq T \leq 263\ K) \\ 0.1 & (T > 263\ K) \end{cases}$$

# 配列

# 配列 (1)

- 下の内容の array.f90 を作成して実行しよう.

```
program array
  implicit none
  real(8) :: arr(3)
  arr(1) = 2.0d0
  arr(2) = 15.0d0
  arr(3) = 25.0d0
  write( 6, * ) arr(1) * arr(2) + arr(3)
end program array
```

# 配列 (2)

- array.f90 を変更して実行しよう.

```
program array
  implicit none
  integer, parameter :: nx = 3
  real(8) :: arr(nx)
  integer :: i
  do i = 1, nx
    arr(i) = i * 10
  end do
  do i = 1, nx
    write( 6, * ) i, arr(i)
  end do
end program array
```

# 配列 (3)

- array.f90 を変更して実行しよう.

```
program array
  implicit none
  integer, parameter :: nx = 2, ny = 3
  real(8) :: arr(nx, ny)
  integer :: i, j
  do j = 1, ny
    do i = 1, nx
      arr(i,j) = i  + j * 10
    end do
  end do
...
```

右に続く

```
...
  do j = 1, ny
    do i = 1, nx
      write( 6, * ) i, j, arr(i,j)
    end do
  end do
end program array
```

# 配列 (4)

- array.f90 を変更して実行しよう.

```
program array
  implicit none
  integer, parameter :: nx = 2, ny = 3
  real(8) :: arr(nx, ny)
  integer :: I, j
  do j = 1, ny
    do i = 1, nx
      arr(i,j) = i  + j * 10
    end do
  end do
…
```

```
…
  do j = 1, ny
    write( 6, * ) j, (arr(i,j), i = 1, nx)
  end do
end program array
```

右に続く

# 配列 (5)

- 下の内容の atm.f90 を作成して実行しよう.
  - Wangara の観測データを使用することに注意.

```
program atm
  implicit none
  integer, parameter :: nz = 31
  real(8) :: height(nz), temp(nz), press(nz)
  integer :: k
  open(50, file='wangara/Data_Aug17LT12H.data', status='unknown')
  read(50, *)
  do k = 1, nz
    read(50, *) height(k), press(k), temp(k)
  end do
  close(50)
  do k = 1, nz
    write(6, *) temp(k), height(k)
  end do
end program atm
```

# 練習問題 6

1. wangara/Data_Aug17LT12H.data を用いて, Wangara における 900 hPa 気圧面での温度を計算するプログラムを作りなさい.
   - 要するに, データを補間/内挿しなさい.
2. wangara/Data_Aug17LT12H.data を用いて, Wangara での温位の鉛直分布を計算して, ファイルに出力するプログラムを作りなさい. また, 計算した温位のグラフを描きなさい.

# サブルーチン・関数

# サブルーチン・関数 (0)

- 準備として下のプログラムを作りなさい.
  - 1 から 10 までの整数の和, 1 から 20 までの整数の和, 1 から 30 までの和を計算するプログラムを作りなさい. ただし, 一つのプログラムで三つの和を計算すること.

# サブルーチン・関数 (1)

- 例えばこのようになるだろう (sum.f90).

```
program summation
  implicit none
  integer :: i
  integer :: sum
  sum = 0
  do i = 1, 10
    sum = sum + i
  end do
  write( 6, * ) sum
…
```

右に続く

```
…
  sum = 0
  do i = 1, 20
    sum = sum + i
  end do
  write( 6, * ) sum
  sum = 0
  do i = 1, 30
    sum = sum + i
  end do
  write( 6, * ) sum
end program summation
```

# サブルーチン・関数 (2)

- 下のように変更して実行しよう.

```
program summation
  implicit none
  integer :: sum
  call calcsum(10,sum)
  write( 6, * ) sum
  call calcsum(20,sum)
  write( 6, * ) sum
  call calcsum(30,sum)
  write( 6, * ) sum
end program summation
…
```

右に続く

```
…
subroutine calcsum(n,sum)
  implicit none
  integer, intent(in) :: n
  integer, intent(out) :: sum
  integer :: i
  sum = 0
  do i = 1, n
    sum = sum + i
  end do
end subroutine calcsum
```

# サブルーチン・関数 (3)

- サブルーチンを別ファイルにしよう.

左右のプログラムは別ファイルに作成

sum.f90

```
program summation
  implicit none
  integer :: sum
  call calcsum(10,sum)
  write( 6, * ) sum
  call calcsum(20,sum)
  write( 6, * ) sum
  call calcsum(30,sum)
  write( 6, * ) sum
end program summation
```

sum_sub.f90

```
subroutine calcsum(n,sum)
  implicit none
  integer, intent(in) :: n
  integer, intent(out) :: sum
  integer :: i
  sum = 0
  do i = 1, n
    sum = sum + i
  end do
end subroutine calcsum
```

# サブルーチン・関数 (4)

- 複数のファイルに分割したプログラムは下のようにしてコンパイルすることができます.

```
$ gfortran sum.f90 sum_sub.f90
```

- 参考
  - 大規模なプログラムは多数のファイルに分割されています.
  - それらたくさんのファイルをコンパイルする便利な方法が make です.
    - コンパイル方法を Makefile に記述しておくことで, make と打つだけでコンパイルすることができます.
    - 興味があれば調べてみてください.

# サブルーチン・関数 (5)

- サブルーチンを関数にしよう.

左右のプログラムは別ファイルに作成

sum2.f90

```
program summation2
  implicit none
  integer :: sum
  integer :: calcsum
  sum = calcsum(10)
  write( 6, * ) sum
  sum = calcsum(20)
  write( 6, * ) sum
  sum = calcsum(30)
  write( 6, * ) sum
end program summation2
```

注意

sum2_func.f90

```
function calcsum(n) result(sum)
  implicit none
  integer, intent(in) :: n
  integer :: sum
  integer :: i
  sum = 0
  do i = 1, n
    sum = sum + i
  end do
end function calcsum
```

# サブルーチン・関数 (6)

- 関数の引数に配列を与えよう.

sum3.f90

```
program summation3
  implicit none
  integer, parameter :: nn = 10
  real(8) :: array(nn)
  integer :: i
  real(8) :: calcsum
  do i = 1, nn
    array(i) = 2 * i
  end do
  write( 6, * ) calcsum(nn, array)
end program summation3
```

注意

sum3_func.f90

```
function calcsum(nn, array) result(sum)
  implicit none
  integer, intent(in) :: nn
  real(8), intent(in) :: array(nn)
  real(8) :: sum
  integer :: i
  sum = 0
  do i = 1, nn
    sum = sum + array(i)
  end do
end function calcsum
```

サブルーチンでも同様に配列を引数に与えることができる.

# 練習問題 7-1

1. wangara/Data_Aug17LT12H.data を用いて, Wangara の大気の安定度の鉛直分布を計算してファイルに出力するプログラムを作りなさい. また, 計算した安定度のグラフを描きなさい.
   - ただし, 安定度の計算の際には, 微分を下のように評価できることを用いてもよい.

$$\frac{d\theta}{dz} \sim \frac{\theta(z+\Delta z)-\theta(z)}{\Delta z}$$

2. Wangara の観測データは, 3 時間毎に 1 日分ある. すべての時刻での温位分布と安定度分布を計算し, 日変化の様子を調べなさい.
   - データには一部, 欠損値が含まれるので注意すること. なお, 欠損箇所のデータは -99999.0 となっている.

# 練習問題 7-2

2. 下の URL にあるファイルには, 神戸気象台で測定された 2015 年の気温の日平均値が保存されている. このファイルのデータを読んで, 各月の気温の平均値と標準偏差を求めるプログラムを作りなさい.

   https://itpass.scitec.kobe-u.ac.jp/seminar/lecture/fy2016/160916/pub/kobe_temp.data

   – ファイルには, 1 カラム目に 1 月 1 日からの日数, 2 カラム目には日平均気温が保存されている.
   – ファイルは wget でダウンロードすると良い.

$ wget http://itpass.scitec....（上記の URL）

# 練習問題 7-3

3. 右のプログラムは, xy 空間の $0 < x < 1, 0 < y < 1$ の範囲内の一点の座標をランダムに選んで出力する. このプログラムを基にして, $-1 < x < 1, -1 < y < 1$ の範囲の 10000 個のランダムな点の座標を出力するプログラムを作り, その点をの分布を図示しなさい.
   - 右のプログラムは, 実行するたびに同じ座標を出力します. 出力する乱数を指定するためには random_seed を使うと良いでしょう.

```
program random
  implicit none
  real(4) :: x, y
  call random_number(x)
  call random_number(y)
  write( 6, * ) x, y
end program random
```

# 練習問題 7-4

4. 先のプログラムで出力(計算)したランダムな点のうち, 原点からの距離が 1 以下の点の割合を求めるプログラムを作りなさい.
5. ランダムな点の数に対して, 上記の割合がどのように変化するか調べなさい.

# 覚書

- 本当は, format を解説しないといけないような.
- 本当は, module を解説しないといけないような.
- 本当は, allocate を解説しないといけないような.
- 本当は, gnuplot でアニメーションの方法を解説しないといけないような.