Fortran 文法のまとめ

ver. 2019/05/10

Fortran90 の基本事項

大文字と小文字は区別しない

自由形式：一行 132 文字まで記述(コンパイルオプションで撤廃できる)

(固定形式：FORTRAN77 では何文字目から書くかが指定されていた)

一行に一文を記述

プログラムは上から順番に実行される

;を使うことで一行に複数文を書くことも可能。

```
a=0; b=0                    ! a にゼロを代入、b にゼロを代入
```

! 以降は行末まではコメント文としてコンパイル時には無視される。

一行が長くなって複数行にまたがる場合は一行目最後に &

次の行の頭にも&を置いても良い

```
a = 1.0d0 + 2.0d0 + 3.0d0 + 4.0d0 + &
&  5.0d0 + 6.0d0 + 7.0d0 + 8.0d0 + &
&  9.0d0 + 10.0d0
```

○プログラムの基本構造

```
PROGRAM プログラム名称
宣言文（IMPLICIT NONE 宣言、変数宣言）
実行文
END PROGRAM プログラム名称
```

IMPLICIT NONE

は、暗黙の変数型を使用せず、全ての変数の型を明確に指定する文。(使うことを推奨)

暗黙の型宣言とは、IMPLICIT NONE を使わない場合 Fortran では

```
IMPLICIT REAL*8(a-h, o-z)
```

となり a～h、o～z で始まる変数・定数は倍精度実数型、その他の i～n で始まる変数・定数はデフォルト型の整数型を使う方法が過去に広く使われていた(非推奨)

型

変数型	記述方法	推奨されない記述方法

整数型	INTEGER	
単精度実数型	REAL	REAL(4), REAL*4
倍精度実数型	DOUBLE PRECISION	REAL(8), REAL*8
複素数型	COMPLEX	COMPLEX*8
倍精度複素数型	COMPLEX(KIND(0d0))	COMPLEX*16, DOUBLE COMPLEX
論理型	LOGICAL	
文字型	CHARACTER	

REAL*8 と COMPLEX*16 はよく見かける。DOUBLE PRECISION は推奨されるがあまり見かけない。

変数名：英数字とアンダーラインが使える。英字から始まる 31 文字までの名前

変数の宣言例

```
INTEGER :: a
REAL*8 :: b, c
DOUBLE PRECISION :: d=0.0D0              ! 初期値入力
COMPLEX(kind(0d0)) :: dt1
COMPLEX(kind(0d0)), PARAMETER :: iunit = (0.0d0, 1.0d0)  ! プログラム中で値を変更しない変数(定数)にはパラメター属性をつける。
```

文字型変数の宣言例

```
CHARACTER :: a          ! 長さ 1 文字の変数
CHARACTER(LEN=4) :: b   ! 長さ 4 文字の変数
CHARACTER(4) :: c       ! 長さ 4 文字の変数をこのようにも書ける
CHARACTER*4 :: d        ! 長さ 4 文字の変数をこのようにも書ける
```

変数型について

整数型：4 バイト(32 ビット)，2 進数でメモリに格納。最大値 2^{31}-1 まで
単精度実数型：4 バイト、10^{-38}~10^{38}，約 7 桁の精度(符号、指数と仮数部に格納)
3.5，1.0e0，-1.0e5 (-10000)などと表示(e のあとは指数部)
倍精度実数型：8 バイト、10^{-308}~10^{308} 約 16 桁の精度
3.5d0，1.0d0，1.0d-5 (0.00001) などと表示(d のあとは指数部)
単精度複素数型：8 バイト、実部と虚部が単精度実数型
倍精度複素数型：16 バイト、実部と虚部が倍精度実数型
論理型： .TRUE. または .FALSE. (T,F でも OK)

物理の計算では実数と複素数型では基本的に倍精度のものを使う。

○四則演算

(+, -,*, /)、累乗は**で表す。

同じ型の演算は同じ型となる。(整数型/整数型は整数型)

```
a = 5 / 2          ! a は 2 となる。 2.5 にしたい場合は 5.0/2.0 とする
```

違う形の演算は型変換が行われる(整数と実数の演算は実数に、実数と複素数の演算は複素数に)

```
a=4**3   ! 4 の三乗、a は 64 となる
```

○代入文

変数に値を代入するには=を使う(数学の等号ではない)

```
a = 5 / 2.0      ! 倍精度実数型の a に 5/2.0 の計算結果(2.5)を代入
i = i + 1        ! i を 1 増やす。(再帰代入)
```

○DO ループ文の使い方

基本例１（sum（初期値 0）に N 回 1 を足す）

```
sum = 0                !DO ループで足し算をする場合は初期値にゼロを代入する
DO i = 1, N
   sum = sum + 1
END DO
```

基本例２（sum（初期値 0）に i が 1～N まで増える間、3 回ごとに i を足す）

```
sum = 0
DO i = 1, N, 3
   sum = sum + i
END DO
```

補足説明： インクリメント

DO ループを１つずつ変数が増えながら行うのではなく、この場合は 3 つずつ増えて行く。つまり、i=1, 4, 7, 10, …の様に増えて行く事に注意。

○標準入出力(READ/WRITE/PRINT 文の使い方)

基本例（コマンドラインから読み込んだ値を hoge に代入し、hoge をディスプレーに出力）

```
READ(*,*) hoge
WRITE(*,*) hoge   または   PRINT *, hoge
```

補足説明： WRITE 文の２つの＊（星印）の意味

１つ目の＊は IO（ファイル装置番号）を指定する為に使う（後述）

２つ目の＊は FORMAT（データの型）を指定する為に使う

何も考えずにインタラクティブに作業するのであれば、WRITE(*,*)あるいは PRINT *, を使う

○配列（ベクトル）の扱い方

型宣言

基本例

```
DOUBLE PRECISION :: a(1:5),B(1:4,1:3) ! a は 5 つの要素を持つ配列、B は 4x3 の行列
DOUBLE PRECISION :: a(5), B(4,3)  !としてもよい
DOUBLE PRECISION, DIMENSION(1:10) :: p,q,r,s  ! p,q,r,s は 10 つの要素を持つ配列
DOUBLE PRECISION :: p(1:10), q(1:10), r(1:10), s(1:10)  ! としてもよい
```

その他の例

```
DOUBLE PRECISION :: a(-2:2)
```

a(-2), a(-1), a(0), a(1), a(2)の 5 つの要素を持つ倍精度実数型配列を定義

初期値設定

```
DOUBLE PRECISION :: L(1:5)=0.0D0          !配列 L の全ての成分が 0
INTEGER :: L2(1:5) = (/1, 2, 3, 4, 5/)   !配列 L2 の各要素の初期値を設定
```

基本例（N 次元配列 a_iの全要素に 0 を代入）

```
DO i = 1, N
   a(i)=0.0D0
END DO
```

補足説明：当然、0 以外の要素の代入・演算も可能

DO ループを使わずに

```
a(1:N) = 0.0D0
```

としてもよい。

また、異なる配列の違う添字を持つ要素の代入・演算も可能

```
DO i = 1, N
   a(i)=b(i+3)
END DO
```

この時、b の配列の大きさは N+3 より大きくないと不可

DO ループを使わずに

```
a(1:N) = b(4:N+3)
```

としてもよい。

配列関係の組み込み関数

SUM 配列の和を計算

```
c = SUM(a(1:10))   ! a(1)から a(10)までの和を c に代入
```

MAXVAL/MINVAL 配列の要素の最大値あるいは最小値を返す

```
c = MAXVAL(a(1:N))  ! a(1)から a(N)の中での最大値を c に代入
d = MINVAL(a(1:N))   !a(1)から a(N)の中での最小値を d に代入
```

組み込み関数 DOT_PRODUCT

２つの 1 次元配列の内積を計算

```
c = DOT_PRODUCT(a(1:N),b(1:N))  ! a と b の内積を c に代入
```

整数型、実数型、複素数型いずれの場合も内積を計算するため**複素数型の場合は CONJG(a(1:N))と b(1:N)の積が計算される。**

宣言時に配列の大きさに変数を使いたい場合は PARAMETER 属性あるいはサブルーチンなら INTENT(IN)属性が必要(プログラム中で値が変更できなくなる)

```
INTEGER, PARAMETER :: N = 100
DOUBLE PRECISION :: A(1:N), B(1:N), C(1:2*N)
```

コンパイル時に配列の大きさが決まらない場合(実行時配列)

```
INTEGER :: N
REAL*8, DIMENSION(:), ALLOCATABLE :: a  ! サイズを指定せずに宣言

READ(*,*) N      ! 例えば実行時に配列のサイズとなる整数型を読み込む
ALLOCATE( a(1:N) )  ! a の配列を割り当てる
! ここで計算をする
DEALLOCATE (a)   ! a の配列をメモリから開放する
```

○行列の扱い方

基本例（N×N 行列 $B_{i,j}$の全要素に 0 を代入）

```
DO i = 1, N
   DO j = 1, N
      B(j,i)=0.0D0
   END DO
END DO
```

補足説明： 2 重ループの構造上、内側のループが優先的に変化する。

Fortran では多次元配列は
B(1,1), B(2,1), B(3,1), ..., B(N,1), B(1,2), B(2,2), ...
の順番にメモリに格納される (C 言語では逆)。

この順に演算をしたほうがキャッシュミスが少なくなる。

DO ループを使わずに

```
B(1:N,1:N) = 0.0D0
```

としてもよい。

行列の組み込み関数

MATMUL 行列の積を計算

```
c(1:N,1:K)= MATMUL(a(1:N,1:M), b(1:M,1:K)) ! a と b の積を c に代入
```

TRANSPOSE 行列の転置を計算

```
b(1:N,1:N) = TRANSPOSE(a(1:N,1:N))  !  b に a の転置行列を代入
```

行列の出力

DO 文を使う (5 x 5 行列 A の場合)

```
DO i = 1, 5
        WRITE(*,*) (A(i,j), j = 1, 5)
END DO
```

(A(i,j),j=1,5)の j(整数型変数)は展開されて A(i,1), A(i,2), A(i,3), A(i,4), A(i,5) となる。

行列のサイズが大きい場合は標準出力に出しても読めないので、一部だけ出力したり、ファイルに出力して必要な処理をする。

○型変換

整数の実数化、実数の整数化

基本例（i の値を ai と実数化、a の値を ia と整数化）

```
ai=DBLE(i)      ! i を倍精度実数型に変換
```

```
ia=INT(a)                    ! a を整数型に変換
```

補足説明：数値関数参照

複素数の実部、虚部取り出し

基本例

```
rez = DBLE(z)  ! 複素数 z の実部を(倍精度)実数型 rez に代入
imz = AIMAG(z) ! 複素数 z の虚部を(倍精度)実数型 imz に代入
```

２つの実数を実部と虚部とする複素数を作る

```
z = (2.0d0, 1.0d0)     ! 複素数に特定の数値を代入( z に 2.0+i を代入)

iunit = (0.0d0, 1.0d0)  ! 予めプログラムの最初で虚数単位を定義
z = a + b * iunit                ! 複素数 z に a+ bi (a と b は実数型)を代入
```

あるいは

```
z = DCMPLX(a,b)          ! a と b が倍精度実数型の場合
```

○組み込み関数

数学関数

FORTRAN式	名称	数学的	FORTRAN式	名称	数学的
LOG(x)	自然対数	$\log_e(x)$	LOG10(x)	常用対数	$\log_{10}(x)$
EXP(x)	指数	e^x	SQRT(x)	平方根	$\sqrt{x}$
SIN(x)	正弦	$\sin(x)$	COS(x)	余弦	$\cos(x)$
TAN(x)	正接	$\tan(x)$	ASIN(x)	逆正弦	$\arcsin(x)$
ACOS(x)	逆余弦	$\arccos(x)$	ATAN(x)	逆正接	$\arctan(x)$
ATAN2(y,x)	逆正接2	$\arctan(y/x)$	SINH(x)	双曲線正弦	$\sinh(x)$
COSH(x)	双曲線余弦	$\cosh(x)$	TANH(x)	双曲線正接	$\tanh(x)$
RAND()	[0,1]の乱数	random()	x**i	累乗	x^i

数値関数

FORTRAN式	名称	引数数	FORTRAN式	名称	引数数
INT(x)	整数化	1	REAL(x)	実数化	1
DBLE(x)	倍精度実数化	1	DCMPLX(x,y)	倍精度複素数化	2
MOD(x,y)	x/yの余り	2	MAX(x,y,..)	最大値	≧2

MIN(x,y..)	最小値	≧2	ABS(x)	絶対値	1
AIMAG(x)	虚部	1	CONJG(x)	共役複素数	1
SIGN(x,y)	Xの符号替え y/|y| |x|	2	nint(x)	四捨五入後整数化	1

補足説明：角度の単位はラジアン単位なので注意（180°=πラジアン）

○IF 文の使い方

基本例１（i の値が 0 の時は 0 を、１以上の時は 1 を、それ以外（つまり負の値）の時は-1 を加える）

```
IF (i .EQ. 0) THEN
   i = i
ELSE IF (i .GE. 1) THEN
   i = i + 1
ELSE
   i = i - 1
END IF
```

補足説明：　関係演算子

```
A .EQ. B        A==B    :A が B と等しい時（A=B）に真
A .NE. B        A/=B    :A が B と等くない時（A≠B）に真
A .GE. B        A>=B    :A が B 以上の時（A≥B）に真
A .GT. B        A>B     :A が B より大きい時（A>B）に真
A .LE. B        A<=B    :A が B 以下の時（A≤B）に真
A .LT. B        A<B     :A が B より小さい時（A<B）に真
```

基本例２　実行する文が短い場合は Then も省略して一行で書くことも可能

```
delta = 0
IF (i .EQ. j) delta = 1          ! i=j であれば delta に 1 を代入
```

○IF 文による DO ループの制御

基本例 1　DO 文で終了条件を書かない (EXIT 文でループを抜ける)

```
x = 0.0d0
DO
        x = x + 0.10d0
        y = x**2
        PRINT *, x, y
        IF( x .GT. 1.0d0) EXIT     ! x>1 なら DO ループから出る
```

```
END DO
```

条件が満たされないとプログラムは終了しないので端末から Ctrl-C で強制終了する。

基本例 2 特定の条件のときは DO ループを実行しない(CYCLE で DO ループの頭に戻る)

```
DO i = 1, N

        IF ( i .EQ. 7 ) CYCLE    ! i が 7 のときは戻って i=8 の処理に移る。
!ここに i が 7 以外のときの計算を書く。

END DO
```

基本例 3 入れ子(ネスト)になる場合は DO ループに名前をつける

```
LOOP1: DO i = 1, N
        LOOP2: DO j = 1, N
                IF( i .EQ. j ) CYCLE LOOP2
        END DO LOOP2
END DO LOOP1
```

IF-END IF 文にも同様に名前をつけることができる。

○論理演算

If 文の中で、比較演算子や論理型の変数と共に用いられる。

論理演算子	意味 & 意味	
.NOT.	以外	.NOT. a==b
.AND.	かつ	a==b .AND. b==c
.OR.	もしくは	a==b .OR. b==c
.EQV.	論理値が等しい	a<0 .EQV. b<0 （aとbの符号が同じという比較例）
.NEQV.	論理値が等しくない	a<0 .NEQV. b<0 (aとbの符号が一致しないという比較例)

例：

```
IF(A.EQ.B.AND.B.GT.C) THEN      : A と B が等しく、B が C より大きい時真
IF(.NOT.A.EQ.B) THEN            : A と B が等し無い時真(A.NE.B と同じ)
IF(A.EQ.B.EQV.B.GT.C) THEN      : A と B が等しく、B が C より大きい時真、また
                                  A と B が等しく無く、B が C 以下の時真
```

○関数副プログラム

関数を定義することで組み込み関数と同じように使うことができる。関数がどの型の変数を返すかは関数の中で定義する。(多次元)配列を返すこともできる。

基本例： $f(x) = 4x^3 - 5x$ を関数副プログラムとして定義、主プログラムの中に関数副プログラムを置く場合(内部副プログラム)

```
PROGRAM EXAMPLE2
        IMPLICIT NONE
        DOUBLE PRECISION :: x
        x = 0.0d0
        DO

                PRINT *, x, Func(x)
                x = x + 0.10d0
                IF( x .GT. 5.0d0) EXIT

        END DO

        CONTAINS        !CONTAINS 以下に副プログラムを列挙

        FUNCTION Func(x)                        ! 関数見出し
                DOUBLE PRECISION, INTENT(IN) :: x  ! 宣言部
                DOUBLE PRECISION :: Func  ! 関数の型を宣言する

                Func = 4.0 * x**3 - 5.0 * x     ! 実行部

                RETURN

        END FUNCTION Func
END PROGRAM EXAMPLE2
```

関数に受け渡される引数に INTENT(IN)属性をつけることで関数内でこの値が変更できないようにできる。(推奨)

基本例 2: END PROGRAM の後に関数副プログラムを置く場合(外部副プログラム)

```
PROGRAM EXAMPLE2
        IMPLICIT NONE
```

```
        DOUBLE PRECISION :: x
        DOUBLE PRECISION :: Func        !関数副プログラムの型宣言が必要
        x = 0.0d0
        DO
                PRINT *, x, Func(x)
                x = x + 0.10d0
                IF( x .GT. 5.0d0) EXIT
        END DO
END PROGRAM EXAMPLE2
FUNCTION Func(x)                        ! 関数見出し
        DOUBLE PRECISION, INTENT(IN) :: x  ! 宣言部
        DOUBLE PRECISION :: Func        ! 関数の型を宣言する
        Func = 4.0 * x**3 - 5.0 * x     ! 実行部
        RETURN
END FUNCTION Func
```

RETURN 文は関数やサブルーチンの処理を中断してその段階での値が関数やサブルーチンを呼び出したプログラムに返される。プログラムの末尾の場合は省略してもよい。

○サブルーチン副プログラム

関数副プログラムとほぼおなじ。違う点は

・値を返さない(型がない)

・引数として(複数の)値を返すことができる

・CALL 文で呼び出す

・サブルーチン副プログラムは 1 つの実行文だが関数副プログラムは実行文にはならない)

基本例

```
PROGRAM EXAMPLE2
        IMPLICIT NONE
        DOUBLE PRECISION :: x, y
        x = 0.0d0
        DO
                CALL Func(x,y)    ! CALL 文で Func サブルーチンを呼び出し
                PRINT *, x, y
                x = x + 0.10d0
                IF( x .GT. 5.0d0) EXIT

        END DO

        CONTAINS        !CONTAINS 以下に副プログラムを列挙
```

```
        SUBROUTINE Func(x,y)                    ! サブルーチン見出し
                IMPLCIIT NONE
                DOUBLE PRECISION, INTENT(IN) :: x  ! 宣言部
                DOUBLE PRECISION, INTENT(OUT) :: y

                y = 4.0 * x**3 - 5.0 * x        ! 実行部

        END SUBROUTINE Func
END PROGRAM EXAMPLE2
```

サブルーチンでは IMPLICIT NONE を改めて宣言(推奨)

サブルーチンの引数には INTENT 属性をつけることを推奨

INTENT(IN) : サブルーチンに値が受け渡され、サブルーチン内で値の変更ができない変数(入力引数)

INTENT(OUT): サブルーチンで値がセットされる変数(出力引数)

INTENT(INOUT): サブルーチンに値が受け渡され、サブルーチンで値を変更できる変数(入出力引数)

○再帰

関数副プログラムやサブルーチンの中で自分自身(関数副プログラムやサブルーチン)を引用することを再帰という。漸化式の計算などに使える。

関数副プログラムを再帰的に呼び出すためには RECURSIVE をつける

基本例(フィボナッチ数列)

```
        RECURSIVE FUNCTION Fibonacci(N) RESULT(fibo)
                INTEGER, INTENT(IN) :: N
                INTEGER :: fibo  !結果を代入する変数

                IF(N .LT. 0) THEN
                        fibo = -1
                        RETURN
                END IF

                IF(N .EQ. 0) THEN
                        fibo = 0
                ELSE IF(N .EQ. 1) THEN
                        fibo = 1
```

```
        ELSE
                fibo = Fibonacci(N-1) + Fibonacci(N-2)
        END IF

        RETURN

    END FUNCTION Fibonacci
```

関数の実行結果は Fibonacci ではなく RESULT 句で指定された変数 fibo に代入される
サブルーチンの場合も RECURSIVE キーワードをつければよい。

○関数・サブルーチンでの複数の値・配列の受け渡し

出力が(多次元)配列一つであれば配列型の関数としてもよいし、出力が複数配列、変数にわたる場合はサブルーチンの引数に入れる。

基本例：外積を計算(関数を用いて)

```
    PROGRAM TEST
      IMPLICIT NONE
      DOUBLE PRECISION :: a(1:3), b(1:3), c(1:3)
      a(1) = 6.0d0; a(2) = 3.0d0; a(3) = 4.0d0;
      b(1) = 3.0d0; b(2) = -2.0d0; b(3) = -4.0d0;
      c(1:3) = OUTERPRODUCT(a(1:3), b(1:3))
      WRITE(*,*) c(1:3)
    CONTAINS

      FUNCTION OUTERPRODUCT(a, b)
        DOUBLE PRECISION, INTENT(IN) :: a(1:3), b(1:3)
        DOUBLE PRECISION :: OUTERPRODUCT(1:3) !3 つの要素を持つ配列として関数
を定義

        OUTERPRODUCT(1)=a(2)*b(3)-a(3)*b(2);
        OUTERPRODUCT(2)=a(3)*b(1)-a(1)*b(3);
        OUTERPRODUCT(3)=a(1)*b(2)-a(2)*b(1);

        RETURN

    END FUNCTION OUTERPRODUCT


END PROGRAM TEST
```

基本例：外積を計算(サブルーチンを用いて)

```
PROGRAM TEST
  IMPLICIT NONE
  DOUBLE PRECISION :: a(1:3), b(1:3), c(1:3)
  a(1) = 6.0d0; a(2) = 3.0d0; a(3) = 4.0d0;
  b(1) = 3.0d0; b(2) = -2.0d0; b(3) = -4.0d0;
  CALL OUTERPRODUCT(a(1:3), b(1:3), c(1:3))  !入力も出力も引数に入れる
  WRITE(*,*) c(1:3)
CONTAINS

  SUBROUTINE OUTERPRODUCT(a, b, c)
    DOUBLE PRECISION, INTENT(IN) :: a(1:3), b(1:3)
    DOUBLE PRECISION, INTENT(OUT) :: c(1:3)

    c(1)=a(2)*b(3)-a(3)*b(2);
    c(2)=a(3)*b(1)-a(1)*b(3);
    c(3)=a(1)*b(2)-a(2)*b(1);

    RETURN

  END SUBROUTINE OUTERPRODUCT

END PROGRAM TEST
```

○ファイル入出力

標準入力：端末上でのキーボードからの入力

READ(*,*) var などで標準入力を変数へ読み込める。

標準出力：ディスプレイ上の端末への出力

WRITE(*,*) var や PRINT *, var で変数を標準出力に表示できる。

標準エラー出力：ディスプレイ上への端末へ出力されるが標準出力とは区別される。

WRITE(0,*) var で変数を標準エラー出力に表示できる。

UNIX 上で標準入出力をファイル入出力に変更する：リダイレクト

> 標準出力への出力結果をファイルに保存。a.out が実行ファイルとすると

```
./a.out > outputfile
```

outputfile に./a.out のプログラム実行による標準出力への出力結果を上書き保存

>> すでに存在するファイルに上書きではなく追記したい場合

```
./a.out >> outputfile
```

outputfile の末尾に./a.out のプログラム実行による標準出力への出力結果を追記保存

2> 標準エラー出力への出力結果をファイルに上書き保存。

```
./a.out 2> errorfile
```

標準出力とエラー出力を異なるファイルに上書き保存する場合は

```
./a.out > outputfile 2> errorfile
```

同じファイルに保存する場合は (bash の場合)

```
./a.out >& outputfile
```
または
```
./a.out &> outputfile
```
など

ファイルの値を標準入力にわたす場合

```
./a.out < inputfile
```

inputfile の内容が標準入力に渡される。

Fortran で入出力ファイルを指定する

基本例（入力ファイルを装置番号 1 番に、出力ファイルを装置番号 2 番に割り当て、それぞれのファイルに入出力する）

```
OPEN(1,FILE='inputfile',STATUS='UNKNOWN')
OPEN(2,FILE='outputfile',STATUS='UNKNOWN')
READ(1,*) hoge
WRITE(2,*) hoge
CLOSE(1)
CLOSE(2)
```

補足説明： inputfile, outputfile には操作したいファイル名を入力する

READ, WRITE 文の 1 つ目の変数に読み書きを行う入出力装置番号を指定する。

装置番号を * としたときは標準入出力から読み書きをする

READ(*,*)とすると入力は 5 番のファイル（標準入力、画面上）から読み込む

WRITE(*,*)とすると出力は 6 番のファイル（標準出力、画面上）へ書き出す

WRITE(0,*)とすると出力は 0 番のファイル(標準エラー出力、画面上)へ書き出す

OPEN 文は画面(標準入出力)以外のファイルからの入出力をする為に使う

開けたら(OPEN)、閉じる(CLOSE)のを忘れずに

STATUS 指定子：'OLD' : すでにファイルが存在する場合

'NEW': ファイルが新しく作られる場合

'REPLACE': ファイルが存在する場合は前のファイルが削除される

'UNKNOWN': デフォルト値。ファイルが存在する場合も存在しない場合もある。存在しない場合は新しく作成する。

ACTION 指定子：'READ'：読み込み専用でファイルを開く

'WRITE'：書き込み専用でファイルを開く

'READWRITE': デフォルト。読み書き可能。

POSITION 指定子： 'ASIS' : デフォルトの場所、通常ファイルの先頭位置

'REWIND': ファイルの先頭位置

'APPEND': ファイルの末尾位置

存在するファイルをオープンして出力した場合上書き保存され、前の内容は削除される。

すでに存在するファイルの内容を消さずに一番最後に結果を出力したい場合は

```
OPEN(1,FILE="ファイル名", STATUS="OLD", POSITION="APPEND")
```

などとしてファイルをオープンする。

○書式指定（I:整数、F:実数）

基本例

a) 5 桁までの整数を表示

```
WRITE(*,' (I5)') ihoge
```

b) 全桁数 15、小数点以下 8 桁の実数を 3 つ表示

```
WRITE(*,' (3F15.8)') xhoge, yhoge, zhoge
```

c) 5 桁までの整数を 2 つ表示後、全桁数 15、小数点以下 8 桁の実数を 3 つ表示

```
WRITE(*,' (2I5,3F15.8)') ihoge, jhoge, xhoge, yhoge, zhoge
```

編集記述子の種類

編集記述子	形式	意味
I	**Iw**	**整数値(幅 w)**
F	**Fw.d**	**実数(幅 w, 小数点以下 d 桁)**
E	**Ew.d**	**実数(幅 w,小数点以下 d 桁)**
A	**Aw**	**文字列（幅 w）**

X	**wX**	**空白(幅 w)**
/	/	**改行**

例

```
WRITE(*,' (3X,I5)') 123                    _ _ _ _ _ 1 2 3
WRITE(*,' (F8.2)') 123.4567                _ _ 1 2 3 . 4 5
WRITE(*,' (E8.2)') 123.4567                _ 0 . 1 2 E + 3
WRITE(*,' (A8)') “ABC”                     _ _ _ _ _ A B C
```

○プログラムの終了(エラー処理)

STOP 文でプログラムの実行を終了できる。" " で文字列をエラー出力に出せる。

例：

```
IF(n .LT. 0) STOP "n should be zero or positive"
```

これは

```
IF (n .LT. 0) THEN
        WRITE(6,*) "n should be zero or positive"
        STOP
END IF
```

と同じ。

○コンパイル

```
$ gfortran プログラムソースファイル.f90  -o 実行ファイル名
```

コンパイルオプション

-O0, -O1, -O2, -O3 : 最適化オプション

-ffree-line-length-none : デフォルトでは一行 132 文字であるがこの制限を撤廃

-Wall : 全てのコンパイル時の警告メッセージを出力

-Wuninitialized : 初期化されていない変数を検出

-pedantic : 標準外の機能利用を警告

-fbounds-check : 配列の領域外参照を検出

-ffpe-trap=invalid,zero,overflow 浮動小数点例外発生時に異常終了

-fbacktrace 異常終了時にプログラムソースコードの行番号を表示

○デバッグ

文法以前のチェック事項

① 変数の打ち間違いは無いか？

② 配列の大きさが、定義された範囲を超えていないか？

③ DO-END DO、IF THEN ELSE IF-ELSE-END IF などがきちんと閉じているか？

④ emacs では正しくコーディングされているとタブを押したときに各行が整列する。

実践的なデバッグの方法

① 色々な所に WRITE(*,*) /PRINT *, 文を挿入し、途中結果、DO ループの変数の増え方等をモニターする。

② !でコメント化することでソースコードの保存や実行を飛ばすことも可能。

③ do ループの変数がちゃんと受け渡されているかどうかチェックする。特に、配列の大きさを超える変数が受け渡されると、他のデータを破壊してしまうので要注意。

④ 配列にきちんとデータが読み込まれているかチェック。

⑤ 検索コマンドで、同じ変数がちゃんと検索されるかどうかチェック。

⑥ コンパイル時にデバッグオプションを追加