Fortran プログラミング入門

Fortran プログラムの作成と実行　その２

１．キーボードからの入力

今度のプログラムはキーボードから入力した文字列をそのまま画面に表示するプログラムである。今度のプログラムは、宣言文、入力文、出力文からなっていて、だいぶプログラムらしい形になってきた。それぞれの文の意味についてプログラムの後にある解説を参照すること。特に入力文である read に注意しよう。

プログラムを変更する前にプログラムをコピーする

```
cp hello.f90 echo.f90
```

ここで、cp はコピー(copy の略)のコマンドである。

問題 F3

別紙のプログラムの変更箇所を入力し、コンパイル、実行せよ。

２．ファイルへの出力

今度のプログラムでは、画面に出していた出力をファイルへ保存する。ファイルへ保存するためには write 文の中の装置番号と実際のファイルの名前を関係づける必要がある。この関係を結ぶのが open 文である。

プログラムの名前を変更しよう。

```
mv echo.f90 echofile.f90
```

mv はファイルの移動(move)を行うコマンドである。２つめはファイルの名前であっても良いし、別のディレクトリの名前でも良い。別のディレクトリの名前とすると、ファイルの名前を変えずに別のディレクトリ(フォルダ)へそのファイルを移す。さらに、元のプログラムを残すように元の名前のファイルにコピーしておく。

```
cp echofile.f90 echo.f90
```

問題 F4

echofile.f90 を入力し、コンパイル、実行せよ。

このプログラムをファイルから文字を読んで画面に出力するプログラムに変更せよ。

３．計算と代入文 (楕円体の体積を求めるプログラム)

今度のプログラムでは、キーボードから地球の赤道半径と極半径を入力し、体積を求める。このプログラムでは、宣言文、入力文、実行文、出力文があり、やっとすべての要素を含んだプログラムとなった。このプログラムのポイントは宣言文と代入文である。Fortran は歴史が古く、メモリの少なかった時代に出来たものである。このため、メモリをなるべく節約するため、数値をメモリに格納するときに整数や実数などを異なるフォーマットで記憶する。このため、変数にどの種類の数値や文字を格納するのかあらかじめ宣言する必要があり、これをするのが宣言文である。

問題 F5

ellips.f90 を入力し、コンパイル、実行せよ。
このプログラムを改造し、惑星の平均密度を求められるようにせよ。
プログラムを実行し、太陽系の惑星の平均密度を求めよ。必要なパラメータはインターネットや本などで調べること(出典を示すこと)。

４．変数の型

このプログラムは変数の型について理解するためのものである。π の値を計算し、その結果を倍精度実数、単精度実数、整数変数に代入し、最後は実数変数に型を変換している。int や float などは型変換関数と呼ばれる。

問題 F6

type.f90 を入力し、コンパイル、実行せよ。

５．変数の型と四則演算

このプログラムは実数型の変数、または整数型の変数、実数と整数型の変数の演算がどのように行われるのか理解するためのプログラムである。間違いを減らすためには、自動変換機能を使わず、型変換関数は省略しないで書いた方がよいことが分かるだろう。

問題 F7

type2.f90 を入力し、コンパイル、実行せよ。結果として出てくる値が、このようになる理由について考えよ。

問題 F3 プログラム

```
!
!    Echo key input
!
      program echo

      character(80):: A

      read  (5,*) A
      write (6,*) A

      stop
      end program echo
```

プログラムの説明

chracter 文：変数 A を文字型に指定している。文字型変数とは文字をそのまま文字として記憶変数。数字を入れても数ではなく文字である。ここでは文字数は 80 文字記憶する。1 文字が 1byte である。

read 文：変数 A にキーボードから入力を読み込む。5 は装置番号で、標準入力(キーボード)を表す。

問題 F4 プログラム

```
!
!    Echo key input to file
!
      program echo2file

      character(80):: A

      open (10,file='outfile.dat')
      read  (5,*) A
      write (10,*) A

      stop
      end program echo2file
```

プログラムの説明

open 文：ファイルを開いて、ファイル名を装置番号[*1] と関連付ける。ここでは、outfile.dat というファイルを開いて、それを装置番号 10 に指定している[*2]。ファイルがないときは新しいファイルを作り、あるときにはそのファイルを開く。上書きしないように注意が必要である[*3]。

write 文：変数 A の値(ここでは文字列)を open 文で関係づけられた装置番号 10 のファイル(outfile.dat)へ書き込む。

*1: 大型汎用計算機時代の名残である。出力装置に固有の番号が振られていた。

*2: ファイルへ入出力の場合は、5, 6(標準入出力)以外の値を付けておく。

*3: 上書きを防ぐには、status オプションを使って、開くファイルが新規ファイルであることを指定する。

```
      open (10,file='outfile.dat',status='new')
```

と書けば上書きを防げる。既存ファイルがある場合はエラーになって終了する。

問題 F5 プログラム

```
!
!    volume of rotating ellipsoid
!
      program ellips

      implicit none
      real(4):: a, b, pi, v

      write (6,*) 'input equatorial radius a (km)'
      read  (5,*) a

      write (6,*) 'input polar radius b (km)'
      read  (5,*) b

      pi = 4.0e0 * atan(1.0e0)

      a = a * 1.0e3
      write (6,*) 'a =',a,'m'
      b = b * 1.0e3
      write (6,*) 'b =',b,'m'

      v = 4.0 / 3.0 * pi * a**2 * b
      write (6,*) 'The volume is',v,'m^3'

      stop
      end program ellips
```

プログラムの説明

implicit none 文：プログラム中に出てくる変数の宣言文を一切省略しないで書くことを宣言する。宣言文が書いていない変数が出てくるとコンパイルエラーになる。変数名を書き間違えるバグは多いので、バグの予防になる。implicit を何も宣言しないと、

```
implicit real*4 (a-h,o-z)
implicit integer (i-n)
```

と宣言しているのと同じになる (暗黙の型宣言)。

`Real(4)::`文：後ろに続く変数を単精度実数に型宣言する。`(4)`はメモリ使用量が１つの変数につき 4 byte (=32bit)であることを表す。だいたい６桁位の精度である。`(4)`を省略して単に `real` と書くことが出来る。

`pi=4.0e0*atan(1.0e0)`：代入文。π の値を計算している。イコールで結ばれている式は方程式ではなく、右辺で計算した値を左辺の変数に代入することを表す。`4.0e0` は関数電卓と同じで単精度の浮動小数点実数 0 .400000×10^{1} を表す。`atan` は数学の関数 $\tan^{-1}$(arc tan)である。関数は関数名(引数)のように書く。引数は変数でもよい。

`a = a * 1.0e3`：代入文。a の値を 1000 倍して単位をキロメートルからメートルに直している。右辺で a×1000 を計算し、その結果を変数 a に上書きする。

`v = 4.0e0 / 3.0e0 * pi * a**2 * b`：代入文。楕円体の体積(4/3πabc、ここでは a=c)を計算して v に代入する。`a**2` は a^2 を表す。

問題 F6 プログラム

```
!
!    type of numerical variables
!
      program type1

      implicit none
      real(4):: a1,a2
      real(8):: da1,da2
      integer:: i

      a1  = 4.0e0*atan(1.0e0)
      da1 = 4.0d0*atan(1.0d0)
      da2 = dble(a1)
      i   = int(a1)
      a2  = float(i)

      write (6,*) 'a1   =', a1
      write (6,*) 'da1 =', da1
      write (6,*) 'da2 =', da2
      write (6,*) 'i   =', i
      write (6,*) 'a2  =', a2

      stop
      end program type1
```

プログラムの説明

`real*8` 文:後ろに続く変数を倍精度実数に型宣言する。`*8` は 8 byte (=64bit)であることを表す。だいたい 15〜16 桁の精度で計算する。`double precision` とも書くことが出来る。

`integer` 文:後ろに続く変数を単精度整数に型宣言する。メモリ使用量は 1 つの整数変数につき 4byte=32bit である。$\pm 2^{32}$-1($\pm 2^{31}$?)まで表せる。

`da1=4.0d0*atan(1.0d0)`:代入文。倍精度実数。`4.0d0` は $0.4000000000000 \times 10^{1}$ を表す。引数

が倍精度の時は、関数は倍精度になる。sin や exp, log などだいたいの数学関数が使える。

`da2 = dble(a1)`：単精度型変数を倍精度型へ変換する。`dble` は単精度実数を倍精度実数へ変換する関数で型変換関数と呼ばれる。Fortran の仕様上、後ろに 0 がくっつくのではないことに注意が必要である。逆に倍精度実数を単精度実数に変換する関数は、`sngl( )`である。

`i  = int(a1)`：実数変数を整数型へ変換する。小数点以下は切り捨てられる。

`a2  = float(i)`：整数変数を単精度実数型へ変換する。`real( )`と書くことも出来る。倍精度実数型への変換は `dfloat( )`である。

問題 F7 プログラム

```
!
!    integer devide by real or integer variable
!
      program type2

      implicit none
      real(4):: a,b,c,d,e,f,g,h
      integer:: i,j,k

      i = 5
      j = 3
      k = 2
!
!   watch out the automatic type transfomation
!
      a = float(i)/float(j)
      b = i/j
      c = float(i/j)

      d = i*j/k
      e = i/k*j
      f = i/k*float(j)
      g = float(i)/k*j

      write (6,*) 'a,b,c =', a,b,c
      write (6,*) 'd,e   =', d,e
      write (6,*) 'f,g   =', f,g

      stop
      end program type2
```

`e = i*j/k`：整数型変数の四則演算の結果を実数型変数に代入する代入文である。四則演算は数学と同じ順序で行われる。代入時に自動的に型変換される。

```
e = float(i*j/k)
```

と同じ意味。式が数学的に同じでも、また、エラーにならなくても値が異なってしまうことがあることに注意せよ。

まとめ

数学演算

+：加算　c = a+b

-：減算または負号　c = a-b　c = -a

*：積算　c = a*b

/：除算　c = a/b

：べき乗　c = a2 (c = a*a と同じ。べきは実数でも良い)

()：括弧　f = a*((b+c)*d+e) 内側の括弧が最優先される。{}[]などは使用しない。

Fortran の組み込み関数

int(x): 実数型から整数型への変換

float(i): 整数型から単精度実数型への変換

dfloat(i): 整数型から倍精度実数型への変換

sngl(x): 倍精度実数型から単精度実数型への変換

dble(x): 単精度実数型から倍精度実数型への変換

sin(x): sin x

acos(x): $\cos^{-1}$ x, arc cos x

tanh(x): tanh x

sqrt(x): $\sqrt{x}$

log(x): log x

log10(x): $\log_{10}$ x

exp(x): e^x, exp x

abs(x): 絶対値 |x|

mod(x,y): 剰余関数、x を y で割った余り(引数が実数型の時は値も実数型)

min(x,y): x と y のうち小さい方

max(x,y): x と y のうち大きい方

Linux のコマンド　その２

ファイルのコピー(file_a から file_b)　　cp *file_a file_b*

ファイル(file1, file2, …)の別ディレクトリへのコピー

cp *file1 file2 … directoryname*

ex. 別ディレクトリのファイルのカレントディレクトリへのコピー

`cp directoryname/filename .`

複数のファイルをコピーするときは *filename* のところにはワイルドカード*(*, ?)を使っても良い。

ファイルの名称変更(file_a から file_b)	`mv file_a file_b`
ファイルの別ディレクトリへの移動	`mv file1 file2 … directoryname`
ファイルの削除	`rm file1 file2 …`

***(アスタリスク)**は文字数・文字ともに不定で、? は 1 文字に対する不定を表す。

`cp *.f directoryname`

とすると、Fortran のファイル(名前が.f で終わるファイル)をすべて directoryname のディレクトリをコピーする。

`rm *`

とするとそのディレクトリ内のファイルすべてを消去してしまうので、注意が必要である。

付録

Fortran の参考書

冨田博之・齋藤泰洋「Fortran90/95 プログラミング (改訂新版)」、培風館、2011 年
新井親夫「Fortran90 入門—基礎から再帰手続きまで」森北出版、1998 年
牛島　省「数値計算のための Fortran90/95 プログラミング入門」森北出版、2007 年

参考になるホームページ

Fortran90 プログラミング

`http://ace.phys.h.kyoto-u.ac.jp/~tomita/education/fortran90/sec0.html`

冨田博之・齋藤泰洋「Fortran90/95 プログラミング」の Web 版。サーバー負荷を掛けないようにするため、ファイルをダウンロードして使用すること。

Fortran 標準コーディングルール

`http://www.mri-jma.go.jp/Project/mrinpd/coderule.html`

気象庁による Fortran プログラミングの書式に関するガイドライン。

バグの修正

プログラムの誤りのことをバグと呼び、バグを取り除く作業をデバッグという。プログラムがある程度長くなると、ほぼ必ずバグが入ってしまう。バグにはコンパイルの時に見つかるものと実行時に見つかるものがある。

コンパイル時エラー

コンパイル時にエラーが出た場合、エラーメッセージの最初のものだけに注目して修正する。後ろに出てくるものは、最初のエラーが原因で起きている可能性もあるからである。

(1) エラーメッセージにはエラーを起こした行番号と内容が出てくるので、それに従ってスペルをチェックする。
(2) l(エル)と 1(いち)や O(オー)と 0(ゼロ)などの間違いがないかチェック。
(3) 固定形式の場合フォーマットに従っているかチェックする。文字数は？行番号や継続記号の位置は正しいか？
(4) 後ろの行の継続記号が正しく書かれているか(意図しない継続記号がないか)もチェックする。
(5) 自由形式で、行を継続させる場合行の後ろに継続記号(&&)を忘れていないかチェックする。
(6) 宣言文に入っていないとメッセージが出た変数を宣言文に正しく記述する。
(7) do 文やブロック if 文が end do, end if で正しく閉じられているかチェックする。

実行時エラー

実行時エラーにはプログラムを動かすと止まるものと、止まらないけれどもおかしい答えが返ってくる場合の２通りがある。実行時止まるエラーには、演算のエラー(０で割り算や関数が無限大に発散)と、変数を記憶している領域の外へアクセスしようとした場合の２つがある(下の(3)(4)の場合など)。これら、特に後者のエラーの原因となるバグを取り除くコツというのは余りない。１つ１つの変数に自分が意図している通りの値が入っているかチェックする。注目している(怪しいと思われる)変数の値を write 文で画面やファイルに書くのである。これは、プログラムが複雑になるほど重要である。時として、その数は膨大なものになり、見るのも大変だが、複雑なプログラムを正しく動作させる唯一の方法と言って良い。

(1) 宣言文で変数を正しく定義しているか
(2) 変数に正しい値が入力されたか (出入力文などで)
(3) サブルーチンや関数での引数の受け渡しが正しく記述されているか
(4) メインルーチンとサブルーチンで変数の型が違っていないか、配列の数が同じか。
(5) 倍精度実数に値を代入するときの浮動小数点表示で 1.0d0 のように書いているか。

プログラムは意図しているとおりでなく、書いたとおりに動くのである。正しく動作するプログラムは必ず作ることが出来るので、根気よくデバッグしよう。