作成：小宅恭史　2007/04/05

Fortran の書き方

目次

1. 変数

12.3g のりんごを 3 個買った。りんご 3 つの合計の重さは何 g でしょう?

りんご 3 つの重さは下式のように求めます。

12.3g×3 個=36.9g

よって、りんご 3 つの重さは 36.9g となります。

このとき、12.3、36.9 を実数、3 を整数といいます。

整数：数をかぞえたりする数字

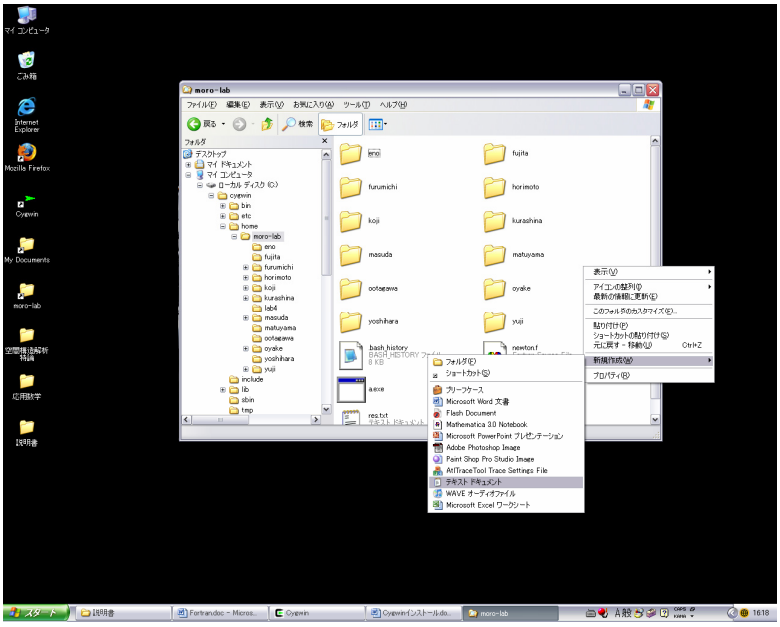
では、りんご 3 つの重さを計算するプログラムを組んでみましょう。

まず、cygwin フォルダのなかの、home フォルダの中の、パソコンのユーザー名のフォルダの中に、テキストを新規作成します。

テキストの新規作成の仕方は、パソコンのユーザー名のフォルダの中で、右クリックをして、新規作成→テキストドキュメントをクリックします。

新規テキスト ドキュメント.txt の名前を example.f と変更します。

このファイルを開きます。使っているパソコンに VisualFortran が入っていない場合は、Notepad(メモ帳)で開きましょう。



開いたファイルに次のように記述します。ここで、すべての行の前 6 マス空けることに注意して下さい。

```
      implicit double precision(a-h,o-z)
      omosa=12.3
      kazu=3
      goukei=omosa*kazu
      write(6,*)goukei
      stop
      end
```

implicit double precision(a-h,o-z)を宣言文といいます。この宣言文の意味は、『a～h と o～z から始まる名前の変数を実数として、それ以外(i～n)までを整数とします。』という意味です。

※パソコンのなかで、値を一時的に記憶しておく入れ物のことを変数といいます

stop end を END 文といいます。

プログラムを作るときは、とりあえず、

```
      implicit double precision(a-h,o-z)

      stop
      end
```

と書いておきましょう。

omosa は、o から始まる名前の変数なので実数、kazu は、k から始まる名前の変数なので整数となります。

omosa=12.3 は、omosa という入れ物に、12.3 という値を代入しています。ここで、=の右側の計算結果を=の左側の入れ物に入れるということに注意して下さい。
プログラムは常に、右側の計算結果を、左側の入れ物に入れます。

a=b　　←b を a に代入

goukei=omosa*kazu は、omosa×kazu の計算結果を、goukei という入れ物に入れています。

Fortran で、足し算や掛け算などは以下のようにします。Fortran の計算は、通常の計算式と同じように計算されます。

通常の計算式	fortran での書き方
a＋b	a+b
a－b	a-b
a×b	a*b
a÷b	a/b
a^2	a**2
a－b×c	a-b*c
(a－b)×c	(a-b)*c

プログラムは、左の式を右の変数に代入するので、以下の書き方がよく使われます。

```
implicit double precision(a-h,o-z)
x=0
x=x+1
stop
end
```

2 行目の x は 0 ですが、3 行目の x は 1 です。3 行目では、右側で 0+1 の計算がされて、左側の x に代入されます。

2. 出力（write 文）

```
implicit double precision(a-h,o-z)
omosa=12.3
kazu=3
goukei=omosa*kazu
write(6,*)goukei
stop
end
```

write(6,*)goukei は、goukei という入れ物の値を表示するという意味です。

write(6,*)a　　a を表示する

exampl.f を上書き保存(Ctrl+S)して、Cygwin で g77　exampl.f と入力して、a.exe と入力してみましょう。コンパイルする前に必ず上書き保存することに注意してください。

36.9000006 と表示されました。

これで、12.3g のりんご 3 つの重さを求めるプログラムの完成です。

write 文にりんごの重さと、個数を一緒に表示させると次のようになります。

```
write(6,*)omosa.kazu,,goukei
```

12.3000002　3　36.9000006 と表示されました。

複数の変数を 1 行に表示させるときは、『write(6,*)変数,変数』と書きます。変数と変数の間にカンマを入れることを忘れないようにしてください。

write 文を次のように変えてみます。

```
write(6,*)'omosa=',omosa
write(6,*)'kazu=',kazu
write(6,*)'goukei=',goukei
```

```
omosa=    12.3000002
kazu= 3
goukei=    36.9000006 と表示されました。
```

文字を表示したい場合は『write(6,*)’moji’』とシングルコーテーションで前と後ろをはさみます。文字の後ろに続けて変数を表示したい場合も、シングルコーテーションの後にカンマを入れることを忘れないようにしてください。

※　‘　←シングルコーテーション　Shift+7 で入力する。

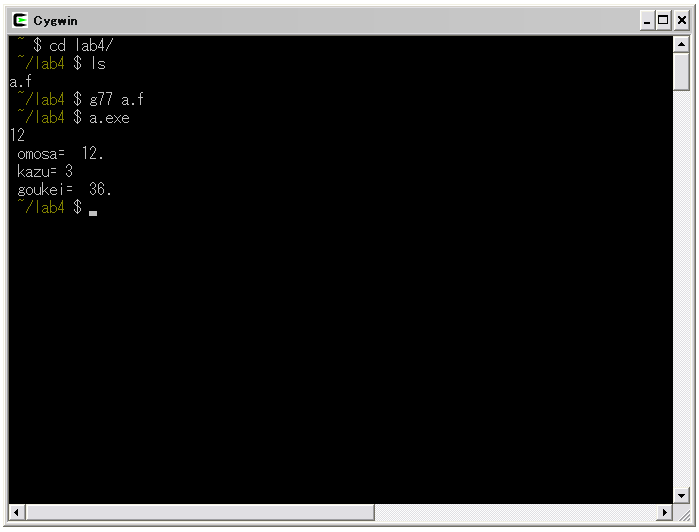
3. 入力（read 文）

```
implicit double precision(a-h,o-z)
omosa=12.3
kazu=3
goukei=omosa*kazu
write(6,*)'omosa=',omosa
write(6,*)'kazu=',kazu
write(6,*)'goukei=',goukei
stop
end
```

このプログラムを次のように変えてコンパイルして実行してみましょう。

```
implicit double precision(a-h,o-z)
read(5,*)omosa
kazu=3
goukei=omosa*kazu
write(6,*)'omosa=',omosa
write(6,*)'kazu=',kazu
write(6,*)'goukei=',goukei
stop
end
```

すると、Cygwin は入力待ちの状態になります。このときに、12 と入力し、Enter を押してみましょう。



omosa= 12.
kazu= 3
goukei= 36.と表示されます。omosa という変数には 12 が入っていることがわかります。変数の値を入力したいときは、『read(5,*)変数名』と書きます。

read(5,*)a　　a を読み込む

ついでに、数も入力できるように変更してみましょう。

omosa=12.3
kazu=3 を次のように変更します。

read(5,*)omosa
read(5,*)kazu と変更します。

read(5,*)omosa,kazu と一行で書いてもかまいません。write 文でも言ったように、変数と変数の間にカンマを入れることを忘れないようにしてください。

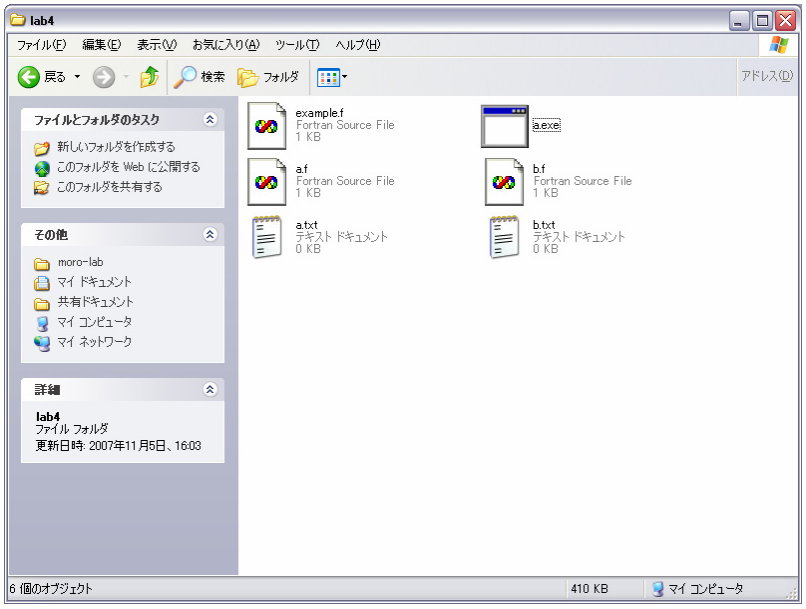
ここで、2 行に分けて read 文を書いた場合と、1 行に read 文を書いた場合の違いについて説明します。
2 行に分けて read 文を書いた場合は、そのままですが、1 行目で omosa を読み込み、2 行目で kazu を読み込んでいます。
1 行で書いた場合は、1 行目で、まず omosa を読み込んでから、kazu を読み込んでいます。

実際にどう違うか確認してみましょう。

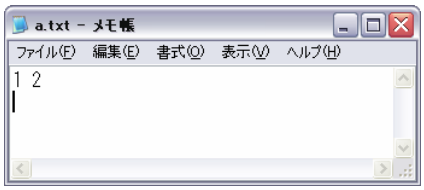
example.f ファイルをコピーして、2 つ貼り付けましょう。貼り付けたファイルの名前を a.f と b.f に変更します。

a.f には read 文を 1 行で書いたプログラム、b.f には read 文を 2 行で分けたプログラムを作成します。

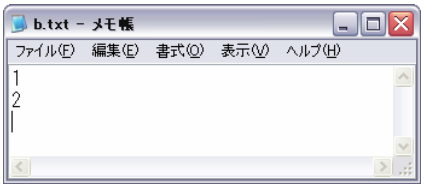
次に a.txt と b.txt を新規作成します。



a.txt には、1　2



b.txt には、1
　　2



と入力してください。最後に Enter（改行）を必ず入力してください。

では、a.f を Cygwin で、g77 a.f と入力してコンパイルしてみましょう。

次に、a.exe < a.txt と入力してみましょう。

omosa=　1.
kazu= 2
goukei=　2.と表示されました。a.txt のデータを読み込んで、実行してくれました。

次に、a.exe < b.txt と入力してみまよう。

list in: end of file
apparent state: unit 5 (unnamed)
last format: list io
lately reading direct formatted external IO
　　0 [sig] a 1896 open_stackdumpfile: Dumping stack trace to a.exe.stackdump
Aborted (core dumped)

と表示されました。これは、読み込みできませんという表示です。a.f は、read(5,*)omosa,kazu と一行で記述しているので、読み込むデータもこの形式にあわせる必要があります。

a.f と a.txt は形式が同じなので読み込むことが出来ましたが、b.txt とは形式が違うので読み込むことが出来なかったわけです。

b.f でもどのようになるか、確認してみてください。

※　　Cygwin 上で値を直接入力する場合は、気にしなくてもいいです。

4. 条件判定（if 文）

12.4g のりんごを 3 個は、18.5g のみかん 2 個よりも、重い？軽い？

りんご 3 つの重さと、みかん 2 つの重さを計算して、その重さを比べます。

りんご 3 つの重さは、
　　12.4g×3 個=37.2g
みかん 2 つの重さは、
　　18.5g×2 個=37.0g

りんご 3 つ 37.2g ＞ みかん 2 つ 37.0g

よって、りんご 3 つのほうが重い。

では、りんご 3 つの重さとみかん 2 つの重さを比べるプログラムを組んでみましょう。

```
implicit double precision(a-h,o-z)
omosa1=12.4
kazu1=3
omosa2=18.5
kazu2=2
goukei1=omosa1*kazu1
goukei2=omosa2*kazu2
if(goukei1.lt.goukei2)then
  write(6,*)'karui'
else
  write(6,*)'omoi'
end if
stop
end
```

何かを比べたりするとき、if(goukei1.lt.goukei2)then と書きます。この文のことを IF 文といいます。
goukei1.lt.goukei2 は goukei1.<.goukei2 なので、if(goukei1.lt.goukei2)then は、『もし合計 1 が合計 2 より小さいとき、write(6,*)’karui’をします！』という意味になります。
else はそれ以外のときを意味するので、『それ以外のとき、write(6,*)’omoi’をします！』という意味になります。

if 文は必ず、if(OO.??.XX)then で始まり、end if で終わります。IF 文を書くときは、とりあえず

```
if(OO.??.XX)then

end if
```

と書きましょう。プログラムを見やすくするために、if と endif の間の行は必ず 2 マス空けて記述してください。

??のところには、＝、≠、＞、≧、＜、≦･･･などを意味する文字が入ります。この文字の書き方は以下のようにします。

通常の数式	fortran での書き方
a＝b	a.eq.b
a≠b	a.ne.b
a＞b	a.gt.b
a≧b	a.ge.b
a＜b	a.lt.b
a≦b	a.le.b

プログラムの omosa2=18.5 を omosa2=18.6 に変更して、コンパイル、実行してみましょう。すると、omoi と表示されます。
これは、『もし合計 1 が合計 2 より小さいとき、write(6,*)'karui'をします！それ以外のとき、write(6,*)'omoi'をします！』というプログラムなので、omoi と表示されるわけです。

合計 1 と合計 2 が等しいときは、onaji と表示されるプログラムに変更してみましょう。

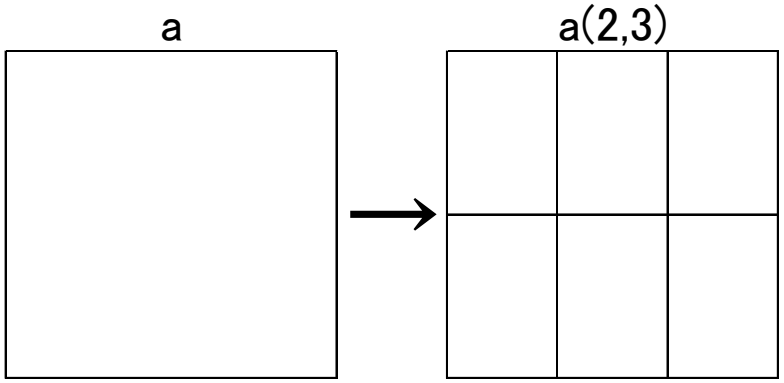
```
if(goukei1.lt.goukei2)then
   write(6,*)'karui'
elseif(goukei1.eq.goukei2)then
   write(6,*)'onaji'
else
   write(6,*)'omoi'
end if
```

条件を追加する場合は、elseif(OO.??.XX)then と書きます。

```
======== if 文========
if(OO.??.XX)then
   □□□の処理
elseif(OO.??.XX)then
   ■■■の処理
else
   △△△の処理
end if
```

5. 配列（dimension）

1 つの変数には、1 つの値しか代入することは出来ませんでした。行列の計算をするときなど、1 つの変数に 4 ついれることが出来たら便利です。fortran では、1 つの入れ物(変数)をいくつかの部屋にわけることが出来ます。



上の図では、a という入れ物を、2 行、3 列の入れ物に変えました。

fortran で入れ物を作るときは、下に書いたプログラムのように、dimension a(2,3)と implicit 文の下に書きます。この文も宣言文と言います。

```
implicit double precision(a-h,o-z)          ←宣言文
dimension a(2,3)                            ←宣言文
a(1,1)=1
a(1,2)=2
a(1,3)=3
a(2,1)=4
a(2,2)=5
a(2,3)=6
stop                                        ←エンド文
end
```

このプログラムでは、a の中身は下図のようになっています。

a(2,3)

1	2	3
4	5	6

6. 繰り返し（do ループ）

九九の 3 の段を計算するプログラムを作って見ましょう。

```
implicit double precision(a-h,o-z)
dimension igoukei(9)
igoukei(1)=3*1
igoukei(2)=3*2
igoukei(3)=3*3
igoukei(4)=3*4
igoukei(5)=3*5
igoukei(6)=3*6
igoukei(7)=3*7
igoukei(8)=3*8
igoukei(9)=3*9
write(6,*)igoukei(1)
write(6,*)igoukei(1)
write(6,*)igoukei(2)
write(6,*)igoukei(3)
write(6,*)igoukei(4)
write(6,*)igoukei(5)
write(6,*)igoukei(6)
write(6,*)igoukei(7)
write(6,*)igoukei(8)
write(6,*)igoukei(9)
stop
end
```

このようにプログラム書くのは、めんどうです。
そこで、下のようなプログラムにします。

```
implicit double precision(a-h,o-z)
dimension igoukei(9)
do i=1,9
   igoukei(i)=3*i
end do
do i=1,9
   write(6,*)igoukei(i)
end do
stop
end
```

do と enddo の間のことを do ループと言います。do ループを見やすくするために、do と end do の間の行は、必ず 2 マス空けてください。
do i=1,9 というのは、i が 1,2,3,･･･,9 と変化しながら do と enddo の間を 9 回繰り返しますという意味です。i が 10 になったとき、do ループから出ます。
なので、この 2 つのプログラムは同じ結果になります。

次に、縦に書き出されていた 3 の段を横に書き出してみましょう。

```
implicit double precision(a-h,o-z)
dimension igoukei(9)
do i=1,9
   igoukei(i)=3*i
end do
write(6,*)igoukei(1), igoukei(2), igoukei(3), igoukei(4),‥‥, igoukei(9),
stop
end
```

これもめんどうです。
そこで、下のようなプログラムにします。

```
implicit double precision(a-h,o-z)
dimension igoukei(9)
do i=1,9
   igoukei(i)=3*i
end do
write(6,*)(igoukei(i),i=1,9)
stop
end
```

これは、i が 1,2,3,･･･,9 と変化しながら write の行を 9 回繰り返しますという意味です。なので、この 2 つのプログラムも同じ結果になります。

では、九九を表示するプログラムを組んでみましょう。

```
implicit double precision(a-h,o-z)
dimension igoukei(9,9)
do i=1,9
   do j=1,9
      igoukei(i,j)=i*j
   end do
end do
do i=1,9
   write(6,*)(igoukei(i,j),j=1,9)
end do
stop
end
```

上のプログラムは、i が 1 のとき、j が 1,2,3,･･･,9 と変わり 9 回繰り返したら、i が 2 に変わり、j が 1,2,3,･･･,9 と変わり 9 回繰り返したら、i が 3 に変わり、この作業を i が 9 になるまで繰り返します。プログラムは上から下に読んでいくので、プログラムがどのように読まれているか、注意してプログラムを書いてください。

7. 組み込み関数

fortran では、あらかじめ用意されている関数があります。

組み込み関数	組み込み関数の意味
mod(a,b)	a を b で割った余りを求める
abs(a)	a の絶対値を求める
sqrt(a)	a の平方根を求める
log(a)	a の自然対数を求める
sin(a)	Sin a を求める(a はラジアン)
cos(a)	Cos a を求める(a はラジアン)
tan(a)	Tan a を求める(a はラジアン)
asin(a)	Sin^{-1} a を求める(a はラジアン)
acos(a)	Cos^{-1} a を求める(a はラジアン)
atan(a)	Tan^{-1} a を求める(a はラジアン)
dble(i)	i を倍精度実数に変える
int(a)	a を整数に変える

組み込み関数を使うとこのようになります。

```
implicit double precision(a-h,o-z)
a=3.14
b=sin(a)
write(6,*)b
stop
end
```

他にも色々な組み込み関数があるので、本やインターネットで探してみてください。

※注意

1.変数、2.出力、3.入力で説明したプログラムで、goukei=omosa*kazu としています。このように、実数×整数となる計算をするのは予期しない計算結果を招く恐れがあるのでやめましょう。

実数×整数をする場合は、整数を実数に変換してから計算します。先ほどのプログラムでは、goukei=omosa*dble(kazu)のようにします。

8. 変数に文字を代入（character）

キャラクターは、文字を入れる入れ物を用意するときに使う宣言文です。
今までは、変数に値を代入していました。キャラクターを使うことで、変数に文字を代入することが出来ます。

```
implicit double precision(a-h,o-z)
character ntmp(3,3)
do i=1,3
   do j=1,3
      ntmp(i,j)='X'
   end do
end do
write(6,*)ntmp(1,3)
stop
end
```

文字を代入する際に、シングルコーテーションで前後をはさむことを忘れないで下さい。
上のプログラムでは、ntmp には、1 つの部屋に、半角 1 文字分しか文字を入れることが出来きないので、ntmp(i,j)='■'のように、全角 1 文字を入れようとしても入れることが出来ません。

そこで、宣言文を次のようにします。

```
implicit double precision(a-h,o-z)
character ntmp(3,3)*2
do i=1,3
   do j=1,3
      ntmp(i,j)='■'
   end do
end do
write(6,*)ntmp(1,3)
stop
end
```

character ntmp(3,3)*2 の*2 は、ntmp の 1 つの部屋の大きさは半角 2 文字分という意味です。

9. ファイルの操作（open 文 close 文）

```
implicit double precision(a-h,o-z)
a=1
b=2
c=a+b
d=a-b
write(6,*)c
write(6,*)d
stop
end
```

このプログラムを実行すると、Cygwin の画面に 3 と-1 が表示されます。この結果を、res.txt に書いてほしいときは、a.exe > res.txt と入力すれば出来ます。

c は Cygwin の画面に表示して、d は res1.txt に書きたいときは、このようにします。

```
implicit double precision(a-h,o-z)
open(unit=7,file='res1.txt',form='formatted',status='unknown')
a=1
b=2
c=a+b
d=a-b
write(6,*)c
write(7,*)d
close(7)
stop
end
```

open(unit=7,file='res1.txt',form='formatted',status='unknown')を open 文といいます。

write(7,*)なら、res1.txt に書き出す
read(7,*)なら、res1.txt から読み込む

くらいに覚えておいて下さい。

unit=7 の 7 は、5 と 6 以外の数字に変えることが出来ます。

10. write 文の書式（format 文）

```
      implicit double precision(a-h,o-z)
      dimension goukei(9,9)
      x=0
      do i=1,9
        x=x+1
        y=0
        do j=1,9
        y=y+1
          goukei(i,j)=x*y
        end do
      end do
      do i=1,9
        write(6,9999)(goukei(i,j),j=1,9)
      end do
 9999 format(9(1x,f3.0))
      stop
      end
```

掛け算のプログラムを上のように書くとキレイに出力されます。write(6,9999)は文番号9999 の書式で書き出しますという意味です。write(6,*)の 6 はつなげる装置の番号、*は書式を表しています。format(9(1x,f3.0))は、1 マス空けて、f3.0 で記述するのを 9 回繰り返しますという意味です。
write 文に format を書く場合は write(6,'(9(1x,f3.0))')のように、シングルコーテーションとカッコの間に書きます。

f3.0　3 マス分で小数点以下 0 桁になるように書き出す
e9.3　9 マス分で小数点以下 3 桁になるように 10^?のように書き出す
1x　1 マス空ける
a4　4 マス分の文字を書き出す

g14.7　f と e の間みたいな書式　どんな値でも対応するから便利

内部の値	format文	出力	内部の値	format文	出力
12345	I4	****	0.1234567*10^5	F6.3	******
12345	I5	12345	0.1234567*10^5	F7.3	*******
12345	I10	___12345	0.1234567*10^5	F8.2	12345.67
12345	I10.6	__012345	0.1234567*10^5	F10.1	__12345.7
-12345	I5	*****	-0.1234567*10^5	F8.2	-12345.7
-12345	I6	-12345	-0.1234567*10^5	F10.1	_-12345.7
-12345	I10	___-12345	0.1234567*10^5	E12.5	_0.12345E+05
-12345	I10.7	_-0012345	0.1234567*10^5	E12.3	__0.123E+05

※アンダーバーはスペースを表します

11. サブルーチン

```
      implicit double precision(a-h,o-z)
      dimension goukei(9,9)
      x=0
      do i=1,9
        x=x+1
        y=0
        do j=1,9
        y=y+1
          goukei(i,j)=x*y
        end do
      end do
      do i=1,9
        write(6,9999)(goukei(i,j),j=1,9)
      end do
 9999 format(9(1x,f3.0))
      stop
      end
```

このプログラムの九九を計算する処理を keisan という名前のサブルーチンにします。

```
      implicit double precision(a-h,o-z)
      dimension goukei(9,9)
      call keisan(goukei)
      do i=1,9
        write(6,9999)(goukei(i,j),j=1,9)
      end do
 9999 format(9(1x,f3.0))
      stop
      end
******
      subroutine keisan(goukei)
      implicit double precision(a-h,o-z)
      dimension goukei(9,9)
      x=0
      do i=1,9
        x=x+1
        y=0
        do j=1,9
          y=y+1
          goukei(i,j)=x*y
        end do
      end do
      return
      end
```

******よりも上側をメイン、それより下をサブルーチンといいます。

call keisan(goukei)でメインから、サブルーチンにいきます。このとき、カッコの中の変数をこのサブルーチンに持っていきます。サブルーチンに持っていく変数のことを引数(ひきすう)と言います。

subroutine keisan(goukei)では goukei に九九の結果を入れています。return でカッコの中の変数を持ってメインに戻ります。
subroutine で goukei を計算して、メインに持って帰ったので、メインで goukei を書き出してもちゃんと表示されるわけです。

```
      implicit double precision(a-h,o-z)
      dimension goukei(9,9)
      call keisan(goukei)
      do i=1,9
        write(6,9999)(goukei(i,j),j=1,9)
      end do
 9999 format(9(1x,f3.0))
      stop
      end
******
      subroutine keisan(xkuku)
      implicit double precision(a-h,o-z)
      dimension xkuku (9,9)
      x=0
      do i=1,9
        x=x+1
        y=0
        do j=1,9
          y=y+1
          xkuku (i,j)=x*y
        end do
      end do
      return
      end
```

上のプログラムは、call keisan(goukei)でメインから、goukei を引数としています。Subroutine では、xkuku となっています。メインの goukei の値が、subroutine では xkuku に入って計算をします。

このように、メインとサブルーチンの変数の名前が違っていても、dimension のサイズ同じであれば大丈夫です。

```
call keisan(a,b,3)
subroutine(a,g,i)
```

複数の引数を書くときは、上のように、間にカンマを入れることと、引数の数が同じになるように注意して下さい。この場合は、a には a、g には b、i には 3 が入ります。

subroutine を書くときは、引数をちゃんと指定することに注意して下さい。

12. パラメータ（parameter）

```
      implicit double precision(a-h,o-z)
      parameter(idofmax=9)
      dimension goukei(idofmax,idofmax)
      x=0
      do i=1,9
        x=x+1
        y=0
        do j=1,9
        y=y+1
          goukei(i,j)=x*y
        end do
      end do
      do i=1,9
        write(6,9999)(goukei(i,j),j=1,9)
      end do
 9999 format(9(1x,f3.0))
      stop
      end
```

parameter で宣言した変数は、プログラム中で書き換えることが出来ません。上のプログラムの場合は idofmax は常に 9 になります。

13. ファンクション（function）

```
      implicit double precision(a-h,o-z)
      parameter(idofmax=9)
      dimension goukei(idofmax,idofmax)
      x=0
      do i=1,9
        x=x+1
        y=0
        do j=1,9
        y=y+1
          goukei(i,j)=x*y
        end do
      end do
      a=xnorm(goukei)
      write(6,*)a
      stop
      end
*****
      function xnorm(goukei)
      implicit double precision(a-h,o-z)
      parameter(idofmax=9)
      dimension goukei(idofmax,idofmax)
      xnorm=0
      do i=1,9
        do j=1,9
          xnorm=xnorm+goukei(i,j)
        end do
```

```
      end do
      return
      end
```

ファンクションで自作の関数を作ることが出来ます。ファンクション xnorm は、九九の総和を求める関数です。

14. データ文（data）

```
      implicit double precision(a-h,o-z)
      dimension a(3,3)
      data a/1,2,3,4,5,6,7,8,9/
      do i=1,3
        write(6,*)(a(i,j),j=1,3)
      end do
      stop
      end
```

このプログラムを実行すると下のような結果になります。

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

データ文は、a の中身を上のように指定することが出来ます。a(1,1)、a(2,1)、a(3,1)、a(1,2)、…の順番で値が入ることに注意してください。

15. go to 文

```
      implicit double precision(a-h,o-z)
      parameter(idofmax=9)
      dimension goukei(idofmax,idofmax)
      x=0
      do i=1,9
        x=x+1
        y=0
        do j=1,9
          y=y+1
          goukei(i,j)=x*y
          if(j.eq.4)then
            go to 100
          end if
        end do
      end do
      a=xnorm(goukei)
100   write(6,*)a
      stop
      end
```

このプログラムでは、j が 4 になったとき、文番号 100 番のところに飛びます。

fortran は、最初の 1 文字目で、その文がコメント(無視する)かどうかを判断します。2～4 文字目は文番号を表します。5 文字目は、上の文とつながっていることを表します。

16. 複数の条件判定（if 文）

```
if((a.eq.0).and.(b.eq.3))then
   x=1
elseif((a.eq.1).and.((b.eq.3).or.(c.eq.4)))then
   x=2
else
   x=3
end if
```

このプログラムでは、a=0 と b=3 のとき x=1 になり、a=1 と b=3 または 4 のときに x=2 になり、それ以外のときは x=3 になります。

条件が複数あるときは、if((a.eq.0).and.(b.eq.3))then や if((a.eq.0).or.(b.eq.3))then と書きます。カッコをつける位置に注意してください。

17. 文字の足し算（character）

キャラクターは、文字を入れる入れ物と説明しました。ここでは、キャラクター同士の足し算について説明します。

キャラクターの足し算は、＋じゃなくて、//　を使うだけです。

```
implicit double precision(a-h,o-z)
character atmp,btmp,ctmp
atmp='a'
btmp='b'
ctmp=atmp//btmp
write(6,*)ctmp
stop
end
```

このプログラムでは、atmp と btmp を足して ctmp に代入しています。ctmp には ab が入っています。

18. 整数をキャラクターに代入（character）

整数を文字列としてキャラクターに代入する方法を説明します。

```
      implicit double precision(a-h,o-z)
      character atmp*3
      do i=1,9
        write(atmp,100)i
        write(6,*)atmp
      end do
100 format('AB',i1)
      stop
      end
```

このプログラムは、AB1、AB2、･･･AB9 と表示させるプログラムです。atmp に i の整数の値を代入したいときは、write(atmp,100)i のように書きます。文字と整数を組み合わせて代入したいときは、format 文を使います。

この方法を知っていると、たくさんの解析データのファイルを作るときなどに便利になります。

```
      implicit double precision(a-h,o-z)
      character atmp*3
      do i=1,9
        write(atmp,100)i
        open(unit=7,file=atmp//'.txt',form='formatted',status='unknown')
        write(7,*)i
        close(7)
      end do
100 format('AB',i1)
      stop
      end
```

こんな書き方をすると、AB1.txt、AB2.txt、･･･AB9.txt というファイルができます。