Fortranプログラミング入門

-内部副プログラム(1)-

副プログラム

副プログラムとは…

- プログラムを小さいプログラムに分割して使う機能
- 単独では実行出来ない
- Fortranには関数とサブルーチンがある

プログラムにバグが出にくい!
バグが出ても，エラーになる部分が限定される!

プログラム単位

Fortranのプログラムを構成する基本的な構成要素を
プログラム単位と呼ぶ.

・主プログラム

　program文で始まる. 一つのプログラムに必ず一つ.

・外部副プログラム

　現在はモジュールで代替. 様々な主プログラムから
　呼び出せる副プログラム.

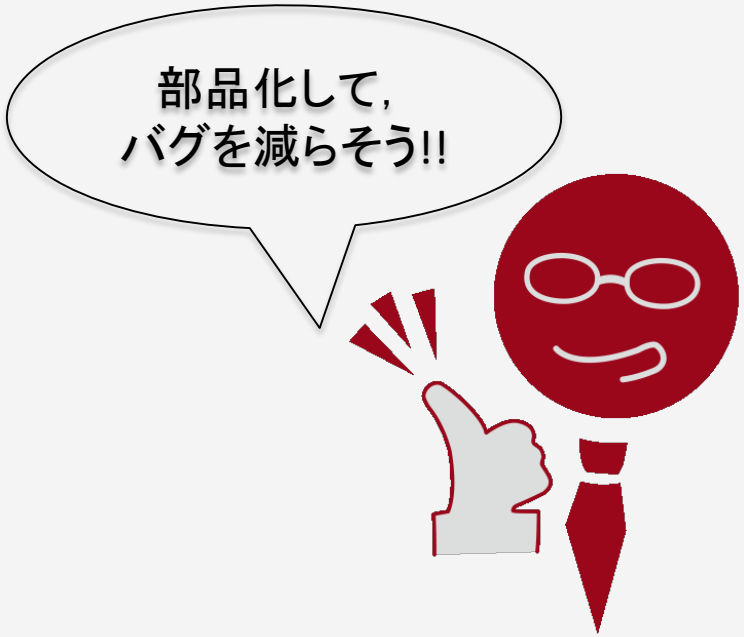
・モジュール(Fortran90から追加)

　変数や副プログラムなどをまとめたもの.

内部副プログラム

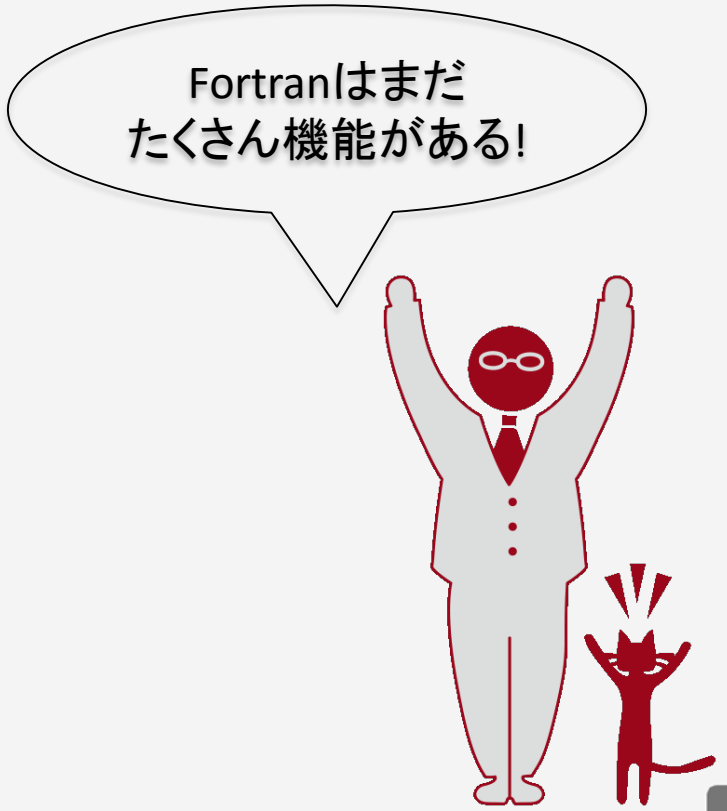
内部副プログラムとは…

- 主プログラムの中に記述する副プログラム
- 内部副プログラムが記述してある主プログラム以外からは利用できない.



プログラム単位

- 主プログラム(program文)
  - 内部関数 ← 今回はこれ!!
  - 内部サブルーチン
- 外部副プログラム
- モジュール



内部関数の書き方

```
program プログラム名
 宣言部
 実行文
 stop
 contains
 function 関数名(引数1, 引数2,・・・)
  ・・・
 end function 関数名
end program プログラム名
```

内部関数の書き方

☆文法　function 関数名(引数1, ･･･)

宣言文

実行文

end function 関数名

あるいは

function 関数名(引数1, ･･･) result(戻り値)

宣言文

実行文

end function 関数名

内部関数の書き方

☆文法　function 関数名(引数1, ・・・)

宣言文

実行文

end function 関数名

・宣言文に仮引数1, ・・・変数の宣言をする

・戻り値の変数名は関数名となる

・宣言文で戻り値の変数名(関数名)の宣言も書く

内部関数の書き方

☆文法　function 関数名(引数1, ・・・) result(戻り値)

宣言文

実行文

end function 関数名

・宣言文に仮引数1, ・・・変数の宣言をする

・戻り値の変数名は戻り値となる

・宣言文で戻り値の変数名(戻り値)の宣言も書く

例題1

i, jを整数型の変数で宣言し，iに適当な値を代入する．
そのとき，iに1加えて値を返す関数addoneを作成し，
addoneを呼び出して値を j に代入せよ．

例題1

```
program test1
 implicit none
 integer :: i, j
 i = 1
 write(*,*) i
 j = addone(i)
 write(*,*) j
 stop
 contains
 function addone(i)
  integer :: i
  integer :: addone
  addone = i + 1
 end function addone
end program test1
```

ここの仮引数iは上のiの宣言とは関係ない

ここで仮引数iの宣言をしている!

ここで戻り値(関数名)の変数の宣言をしている!

ここでiに1加えた値をaddoneに代入.

例題1

```
program test1
 implicit none
 integer :: i, j
 i = 1
 write(*,*) i
 j = addone(i)
 write(*,*) j
 stop
 contains
 function addone(fff)
  integer :: fff
  integer :: addone
  addone = fff + 1
 end function addone
end program test1
```

例えば仮引き数をfffとしても問題ない!

ここで仮引数fffの宣言をしている!

ここでfffに1加えた値をaddoneに代入.

例題1

```
program test1
 implicit none
 integer :: i, j
 i = 1
 write(*,*) i
 j = addone(i)
 write(*,*) j
 stop
 contains
 function addone(fff) result(ggg)
  integer :: fff
 integer :: ggg
 ggg = fff + 1
 end function addone
end program test1
```

戻り値をgggと設定

ここで戻り値の変数gggの宣言をしている!

ここでfffに1加えた値をgggに代入.

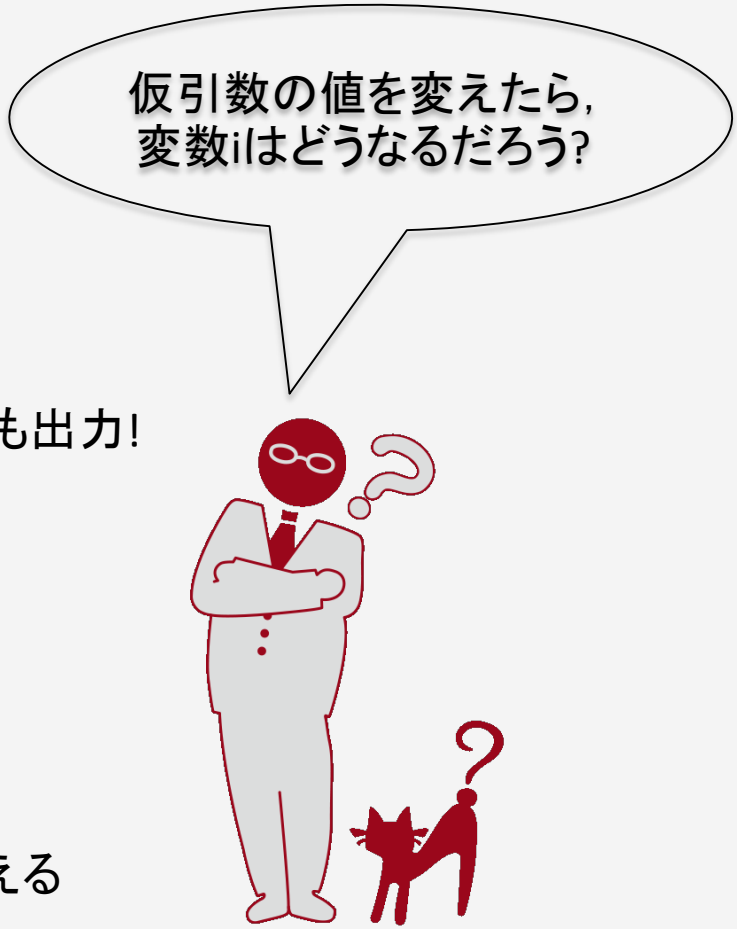
参照引き渡し

次の文を実行したらどうなるか?

```
program test1
 implicit none
 integer :: i, j
 i = 1
 write(*,*)   “i=“, i
 j = addone(i)
 write(*,*) “i=“, i, “j=“, j
 stop
 contains
 function addone(fff) result(ggg)
  integer :: fff
  integer :: ggg
  ggg = fff + 1
  fff = fff + 10
 end function addone
end program test1
```

iとjどちらも出力!

仮引数fffに10を加える



参照引き渡し

次の文を実行したらどうなるか?

```
program test1
 implicit none
 integer :: i, j
 i = 1
 write(*,*)   “i=“, i
 j = addone(i)
 write(*,*) “i=“, i, “j=“, j
 stop
 contains
 function addone(fff) result(ggg)
  integer :: fff
  integer :: ggg
  ggg = fff + 1
  fff = fff + 10
 end function addone
end program test1
```

出力結果

i = 1

i = 11 j = 2

iの値が変わっている!!

Fortranは参照渡し
(変数を共有する渡し方)
であるため注意!!

intent属性

☆文法　データ型, intent(属性) :: 変数名

・仮引数に入力のみ可能，出力のみ可能，入力出力両方可能のいずれかの属性をつける.

※戻り値には書けない.

属性:

in : 入力のみ可能

out : 出力のみ可能

inout : 入力出力両方可能

バグを減らすだけでなく，
最適化で速くなるので必ず書こう!!

例題2

i,jを整数型の変数で宣言し，iに適当な値を代入する．そのとき，iに1加えて値を返す関数addoneを作成し，addoneを呼び出して値をjに代入せよ．但し，関数の引数及び戻り値にはintent属性をつけよ．

例題2

```
program test1
 implicit none
 integer :: i, j
 i = 1
 write(*,*) i
 j = addone(i)
 write(*,*) j
 stop
 contains
 function addone(i)
  integer, intent(in) :: i
  integer :: addone
  addone = i + 1
 end function addone
end program test1
```


仮引数iは入力のみ可能!!
例えば関数内で
　i = i+10
と書いたらコンパイルエラーになる.

配列の引き渡し方

例

A, B, Cは倍精度実数型の二次元配列とし，Cの行数をAの行数，Cの列数をBの列数にする場合の書き方．

```
function 関数名(A, B) result(C)
  real(8), intent(in) :: A(:,:), B(:,:)
  real(8) :: C(size(A,1),size(B,2))
  実行文
end function 関数名
```

2次元配列なのでコロンは2つ!

size関数でサイズを指定した
変数Cを宣言