

# Fortran について

行方大輔

理化学研究所 計算科学研究センター  
粒子系シミュレータ研究チーム

# 概要

- Fortran 77 (以下, F77) ユーザを対象に、FDPS Fortran APIで使っているFortran 2003の見慣れない機能、文法について概説する。
- F77は知っていることを想定。

# 使っている新機能一覧

大きなもの

1. module, use

2. type (構造型、C++でいう構造体/クラス)

3. C言語と相互運用利用可能性 (組み込みモジュール iso\_c\_binding)

# 使っている新機能一覧 (続き)

細々としたもの

1. 変数宣言、サブルーチンの引数宣言の形式、値渡し、パラメータ文の形式
2. do ... end do 文
3. コメントの形式
4. 比較演算子

# 参考書？

## Fortran 2008入門

(<http://www.cutt.co.jp/book/978-4-87783-399-2.html>)

- module, use についての解説あり。
- 構造体についての記述も全くないわけではない。

# module, use

## モジュール宣言文法

```
module モジュール名  
    [宣言部]  
    [contains  
        モジュール副プログラム部]  
end [module [モジュール名]]
```

## モジュール使用文法

```
use モジュール名
```

モジュールが定義されているソースファイルを先にコンパイルすると、何か中間形式のファイルができる。使っているほうのコンパイルではコンパイラがそれを参照する。

# module, use (続き)

- 複数のプログラム単位で使う様々なものをまとめられる。
- パラメータ宣言、データ (common block の代わりになる)、ユーザー定義型、ユーザー定義の関数やサブルーチン等。
- 構造型はモジュール内で定義して、使うサブルーチンでモジュールを **use** するのが基本。
- ちなみに C/C++ には相変わらずモジュールにあたるものはない。

# module, use (簡単な例)

```
module sample
  integer n
  parameter (n=10)
end
program main
use sample
  write(*,*) n
end
```

コンパイル、実行:

```
% gfortran module.F90; ./a.out
```

# type (derived type, 構造型)

## 型宣言

```
type student
  character(32) name
  integer age
end
```

## 変数宣言、使用方法

```
type(student) a
a%name="Sato"
a%age=18
```

# type (derived type, 構造型)

- いわゆる構造体。
- FDPS では、3次元ベクトル型、ユーザーが定義する「粒子型」等を使用。
- プログラミングスタイル云々という話もあるが、キャッシュに確実に載るようにするとかにも有用。

# type (型束縛手続き)

- いつのまにか Fortran も「オブジェクト志向」に。
- 雑にいうと、ある構造型の変数を第一引数にする手続きを `foo(x)` の代わりに `x%foo` と書けるというだけ。こういうのを言語によってメッセージとかメンバー関数とかいう。
- 但し、同じ名前でも別の構造体のメンバー関数なら別の関数になる。演算子も関数にできるので、構造体同士の演算を定義できる。
- 以下では「メンバー関数(手続き)」と呼ぶことに。

# type (メンバー関数の文法)

モジュールstudentmoduleの定義

```
module studentmodule
  type, public:: student
    character(32) name
    integer age
    contains メンバー関数の宣言部
      procedure :: print
    end type
    contains メンバー関数の実装部
      subroutine print(self)
        class(student) self
        write(*,*) self%age
      end
    end
end
```

メイン関数の定義

```
program main
  use studentmodule
  type (student) a
  a%name="Sato"
  a%age=18
  call a%print
end
```

Fortran でもオブジェクト志向

関数のオーバーロード、演算子のオーバーロードができる (ベクトル型を定義して、ベクトル同士の加算とかする演算子を定義できる) (FDPS 側で提供してます)

# iso\_c\_binding

- FortranとCの相互運用性を保証する仕掛け
- 処理系とかOS依存ではなく言語定義として公式に
- Fortran側で、C側で使える変数型とか関数の宣言のしかたを用意
- 文法はなんか面倒だけど、とにかくそれに従っておけばCから(従ってC++からも)Fortranで宣言した構造型や関数が見える
- FDPSのFortran APIは全面的にこの仕掛けを利用

# iso\_c\_binding (例)

```
type, public, bind(c) :: full_particle
    integer(kind=c_long_long) :: id
    real(kind=c_double) mass !$fdps charge
    ....
    type(fdps_f64vec) :: pos !$fdps position
end type full_particle
```

- `bind(c)`で構造体をCからもアクセスできるようになる (C側では別に同等の構造体を宣言/用意する必要あり)
- Cと互換性のあるデータ型の選択は、`(kind=c_double)`等と`kind`値を指定する。
- `fdps_f64vec`はFDPSで提供している倍精度3次元ベクトル型。

# 細々としたこと

1. 変数宣言、サブルーチンの引数宣言の形式、値渡し、パラメータ文の形式
2. do ... end do 文
3. コメントの形式
4. 比較演算子

# 変数宣言

古代 (F77)

```
real a(50)
real c
parameter (c=1.0)
```

現代

```
real, dimension :: a(50)
real, parameter :: c=1.0
```

- dimension, parameter の他に色々属性をつけられる。つける時には変数名の前に "::" を。
- 古代語でもコンパイラは文句いわない(他の新機能も基本的にそう)

# do ... end do文

古代 (F77)

```
do 10 i=1, 50  
  ...  
50 continue
```

現代

```
do i=1,50  
  ...  
end do
```

# コメントの形式

古代 (F77)

```
c この行はコメントです  
  x = x + 1
```

現代

```
! この行はコメントです  
  x = x + 1
```

# 比較演算子

古代 (F77)

```
if (a .lt. b) then  
  ...  
end if
```

現代

```
if (a < b) then  
  ...  
end if
```

==, /=, <, <=, >, >= がある。

# まとめ

- FDPS Fortran APIで使っているFortran 77にない機能を概説した。
- `module`, 構造体, `iso_c_binding`が主。
- 他にも配列演算等の便利そうな機能があるが省略。