FDPS 講習会 (FDPS で使っている)Fortran について

牧野淳一郎
理化学研究所 計算科学研究機構
コデザイン推進チーム
兼 粒子系シミュレータ開発チーム
(本務は神戸大学理学惑星学専攻)

概要

- **F77** ユーザーを対象に、**FDPS Fortran API** で使っている **Fortran 2003** のみなれない機能、文法を概説します。
- **F77** は知っていること想定。
- まず質問: **F77** 以外に使っている言語ありますか？

使っている新機能一覧

大きなもの

1. **module, use**
2. **type** (構造型、C++ でいう構造体とクラス)
3. **iso_c_binding**

使っている新機能一覧 (続き)

細々したもの

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 ... enddo**
3. コメントの形式
4. 比較演算子

まだもうちょっとあったかもしれません。

参考書？

http://www.cutt.co.jp/book/978-4-87783-399-2.html
Fortran 2008 入門

- **module, use** とかの解説もあった。構造体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
          10
```

type (derived type、構造型)

文法 型宣言

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

変数宣言、使用

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

type (derived type、構造型)

- いわゆる構造体。Fortran 90 からあるのでまあそろそろどうでしょうみたいな。
- 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 続き (型束縛手続き)

- いつのまにか **Fortran** も「オブジェクト指向」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。

メンバー関数の文法

```
module studentmodule
  type, public:: student
     character(32)  name
     integer age
   contains
     procedure :: print
  end type
contains
  subroutine print(self)
    class(student) self
    write(*,*) self%age
  end
end
```

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

Fortran でもオブジェクト指向

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

- 処理系とか OS 依存ではなく言語定義として公式に Fortran と C の相互運用性を保証する仕掛け
- Fortran の側で、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使える
- というわけで FDPS の Fortran API は全面的にこの仕掛けを利用

iso_c_binding 例(FDPSの中から)

```
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(C側では別に宣言する必要あり)
- (kind=c_double)とかで、基本データ型をCとコンパチブルに
- (fdps_f64vec は FDPS で提供している倍精度3次元ベクトル型)

細々したこと

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 ... enddo**
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 ... enddo

古代 (F77)

```
  DO 10 i=1,50
    ...
50 CONTUNUE
```

現代

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

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   ! ここにもコメントかけます
```

比較演算子

古代 (F77)

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

現代

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

==, /=, <, <=, >, >= 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。
- 他にも配列演算とか便利そうな機能があるが省略。