OBのFortran90　事始め*

山　岸　米二郎**

1　はじめに

Fortran90は新しい機能が追加され，FORTRAN77とは大きく変わった．筆者はFORTRAN66でコーデングシートにプログラムを書き，カードにパンチしていた時代のold boyである．ところが最近必要に迫られ，Fortran90やプログラムの並列化を少しかじった．この経験をもとに，Fortran90の新しい機能の独断的解釈を記し，関連して数値シミュレーションモデル作成と利用の観点から，ソフトのありかたについて考えを述べる．

FORTRAN66の文法も忘れかけていた筆者にとってはFortran90の新しい機能は驚きであった．こういうold boyが書けば，今更Fortran90なんて煩わしいと考えておられるであろうシニアの方々の大局的理解にかえって役立つかと考え，新しい機能の説明を試みた．これは文法の解説ではなく気象モデル作成に即した解釈である．細部にわたる厳密な説明よりも，文法的に不十分でも大局が理解できるような記述を試みた．また重要と考える機能の基礎のみ説明し，例示する場合も完結したプログラムを記す努力はしない．以下の説明では格子点モデルとしてArakawa-Cタイプを想定し，並列は2次元領域分割を前提とする．

はじめにFortran90の新しい機能について説明し，その後でソフト作成のあり方についての考えを記す．ソフト作成の1つの側面は標準化であり，もう1つは気象専門家と計算技術側との協力の観点である．

2　Fortran90の新しい機能

2.1　重要な観点は何か

物理過程の扱いの高度化などにより数値シミュレーションプログラムが複雑になりかつその分量も多くなっている．作成や修正でのミスの可能性が当然大きくなるし，維持・管理に要する労力も増大する．これは気象モデルに限らない．Fortran90ではミスの発見を容易にする機能，維持・管理の労力を減らすための機能が新たに導入されている．また新しい機能は並列ソフトの作成にも便利である．維持・管理を容易にするための機能の本質は，「相互に関連しているものあるいは関連づけたいものは，1つのまとまりとして扱う」ようにする工夫といえるであろう．これによりプログラムが見やすく，管理が容易になりミスも防げる．

以下では主として動的割付，構造型，moduleの3つを説明する．この他にも自由形式でプログラムが書けるとか，変数名に31文字まで使用可能とか，行の途中から注釈文が書けるなどの便利な機能がある．また暗黙の型宣言無効化（implicit none）やsubroutineでの引数の入出力特性指定（intent（IN），intent（OUT），intent（INOUT））などはミス発見に大変有効である．暗黙の型宣言無効化を指定すると，すべての変数，定数にrealとかintegerなどの型宣言をしなければならないので，プログラム作成途中のミスタイプがあると未定義としてコンパイル時に発見できる．subroutineの仮引数に，subroutineに値を引き渡すために用いる変数（IN），結果の値を返すための変数（OUT），両方に用いるもの（INOUT）を指定できることの便利さは，プログラム作成経験者ならすぐ理解できる．これらは利用に特段の問題がないのでここで説明するまでもない．

2.2　配列の動的割付

配列の寸法を実行時に指定し（割付け），また不要になったら割付を解除してメモリを解放するのが動的割付である．最初から配列の寸法を指定しておくのを静的割付と呼ぶ．動的割付には2つの方法があるが，ここではポインタ（pointer）属性指定による方法を述べ

* Old boy's experience of learning Fortrran90.

** Yonejiro Yamagishi，財団法人高度情報科学技術研究機構．

る．動的割り付けは以下のようにして行う．

```
real, pointer::PT (:,:,:) ！ポインタ属性の3次元配列
                          ！宣言
integer::l, m, n
read (5,*) l, m, n
allocate(PT(l, m, n)) ！大きさl，m，nの配列pT
                      ！の領域指定
```

最初の行はポインタ属性の3次元実数配列PTを宣言しているが，この時点では当然ながらコアにはPTの領域はない．故に配列の寸法は明示されない．なお記号::は変数や定数の宣言の前につける符号，(:,:,:)は配列の寸法が指定されていないことを示す記号である．1次元配列なら(:)となる．allocate文により寸法l, m, nの配列の領域が指定されたので，PTに値を代入するなど使用可能となる．必要なくなったときdeallocate (PT) と宣言すると（割付解除），PTのメモリが解放される．

動的割付けは実行時の領域の大きさの変更が容易であり，コアの節約も可能である．並列の場合の利点も大きい．領域の大きさが等しくない分割をする場合とか，動的バランスを確保するために実行途中で分割方法を変更したいというような場合，動的割付の機能がないと頭を抱えることになる．

ポインタは文字通り他の領域を指し示すことで，それによりequivalence文と類似の機能を持っている．動的割付けの機能もそこから導かれるが，詳しい説明は省略する．

2.3 構造型と構造体

a） 構造体とは

実数型，整数型，複素数型等のおなじみの型の他に，Fortran90では構造型（derived data type）が導入された．構造型で定義された変数を構造体あるいは構造型変数と呼ぶ．構造型は利用者が定義する型で，相互に関連している，あるいは関連づけたい変数や定数等を構造型の成分として定義し，ひとかたまりにして管理したり移動させたりするのに用いる．例えていうならば構造型変数は枠組みで，中に入っている構成物が構造型の成分である．従って構造型変数名は入れ物の識別符と思いばよい．

以下に3次元空間における三角形の例を示す．三角形は3点で定義され，各点はx，y，zの座標で定義されるので，それぞれ3つの成分を持つ構造体を3つ定義する．構造型を使用するには，まず下の例のように構造型名を定義し，構造体にいれるべき成分を宣言する．

```
type zahyo      ！構造型名zahyo定義のtype構文
real::x, y, z   ！構造型成分x，y，zを定義
end type zahyo ！type構文の終わり
```

この3行で，zahyoという構造型名の構造型変数は，3つの成分x，y，zをもち，それらはいずれも実数型変数であることを宣言している．この後必要に応じ構造型変数をいくつでも定義することができる．構造型名zahyoの構造型変数a，bを定義する時は

```
type(zahyo)::a, b ！zahyoという構造型名の構造
                  ！型変数a，bの定義
```

とする．構造型変数a，bの成分はともにx，y，zという同じ名前であるが，中身は一般に異なるので，成分をあわすときはa%x，b%x，a%z等と記して区別する．a%x，b%x等に値を代入する命令の説明は省く．

構造型の成分には，整数型，実数型，複素数型，文字型等任意の変数或いは配列を指定することができる．また構造型変数を成分とすることもできる．成分の数に制限もない．構造型変数名を指定すればいつも成分全体が含まれる．従ってこれを上手に使いこなすと効率的なプログラムができる．三角形を表すためにもう1つの構造型を定義する．

```
type triangle
type(zahyo)::p1, p2, p3 ！構造型triangleの成分
                        ！は構造型名zahyoの構
                        ！造型変数p1，p2，p3
end type triangle
type(triangle)::t       ！p1，p2，p3を成分に持つ
                        ！構造体t
```

はじめの例とあわせるとtriangleという構造型変数tはp1，p2，p3という3つの成分をもち，p1，p2，p3は構造型名zahyoで定義される構造型変数で，それぞれx，y，zという3つの成分を持つ．x，y，zが3次元空間の座標とすると，tはp1，p2，p3を頂点とする三角形と考えることができる．ここで三角形の重心を求める演算を考える．すでに述べたように構造型変数は識別名に過ぎないので，それに加減乗除などの演算を施すことは通常できない．演算は各成分に対して行う．三角形の重心の座標を，type(zahyo)::gで定義すると，g%x＝(t%p1%x＋t%p2%x＋t%p3%x)/

3.0, g%y=(t%p1%y+t%p2%y+t%p3%y)/3.0などとして求める．この演算表記は大変煩わしい．これの回避策は気象モデルの例で検討する．

b) 気象モデルの階層構造と構造体

気象の数値シミュレーションモデルのプログラムは2つの階層とそれをつなぐ中間層に分け得る．領域の大きさを決めたり，境界条件の方式を設定したり，並列の領域分割を指定したり，初期条件を与えたりという，いわば時間積分の準備が第1の階層である．移流の計算や放射の計算等の力学，物理過程の計算プログラムは，領域分割や初期条件等に関係なく存在しうる．これを第2の階層としよう．第1の階層では構造型の機能が大変有効である．第2の層は構造体を用いると便利な面もあるが，演算式に%を用いなければならないという煩わしさが発生する．

時間積分の制御をする部分は第1層と第2層を結びつけるところでいわば中間層である．力学や物理過程を計算するsubroutine（第2層）が繰り返し使われる一方，時間ステップ終了毎に並列の袖領域の交換のために第1層のsubroutineとも関係する．気象モデルでは中間層をどのように設定するかで構造体の扱いが変わりうる．これを具体例で考える

気象の専門家に興味があるのは第2層のsubroutineであるが，これの理解に必要な範囲で第1層と中間層をまず検討する．気象モデルでは座標系を決め，各座標点で速度の3成分，温度，水蒸気量等の時間変化を計算する．従って各変数と座標系の特性をひとまとめにして構造型で定義すると便利である．Arakawa-Cタイプではベクトル量とスカラー量で格子点の数が異なるので，配列の寸法や計算領域の大きさと変数をひとまとめにしておくとミスの防止にも役立つ．例えば構造型を次のように定義したとする．

```
type 3_dimension
integer::ims, ime, jms, jme, kms, kme
real, pointer::var (:,:,:)
type (arakawa_C)::grid
end type 3_dimension
```

この構造型は速度や温位，比湿などの3次元変数を扱うためのものとする．varは3次元変数を動的に割り付けるための配列である．配列の寸法はgridの成分に含まれていて，grid%idsとかgrid%ide，grid%jds等で表されるものとする．またimsやimeは変数varを計算するときのdo loopの制御変数の初期値，終値であると考える．境界条件を考慮すればimsとgrid%idsの値は一般に同じではない．またimeやgrid%ideの数値はベクトル量とスカラー量で異なる．

構造型を定義した後，構造型変数を

```
type(3_dimension)::du, dv, dw, dpt, dq
```

のように宣言する．構造型変数du, dv, dw, dpt, dqはそれぞれ速度のx，y，z成分，温位，比湿の識別名のつもりである．そうすると例えば速度のx成分の値はdu%varで，温位はdpt%varであらわされる．配列varの割付，du%varやdpt%varへの実際の値の代入，ims, jms, grid%ids, grid%jds等への数値の代入やそれに先立つ構造型の定義，構造型変数名の定義などの作業は第1層の仕事で，ここでの説明の対象ではないので省略する．これらの作業がすべて終了しているとして次のような第2層のsubroutineを考える．

```
subroutine example (dummy)
use datagata
implicit none
type (3_dimension), intent (INOUT)::dummy
integer::i, j, k

do i=dummy%ims, dummy%ime
do j=dummy%jms, dummy%jme
do k=dummy%kms, dummy%kme
dummy%var (i, j, k)=0.1* dummy%var (i, j, k)
end do
end do
end do
end subroutine example
```

上の例のuse datagataは後で説明する．この事例は意味のない演算であるが，実引き数をdu, dv, dw, dpt, dqとして，call example (du)，call example (dq)等とすればベクトルかスカラーかなどに煩わされずに使用できて便利である．記号の説明はすでに済んでいるので，この例の意図は理解いただけるだろう．ここで1つ注意する．dummyは構造型変数だが，dummyがどんな成分を持つかは明示されないし，成分の配列の寸法の宣言もない．これではコンパイラが困るので，構造型3_dimensionが定義されている場所をコンパイラに教える必要がある．use datagataという宣言は3_dimensionという構造型が定義されている場所はmodule datagataであることを示すための宣言であ

る．moduleについては次節で説明する．

第2層に構造型変数をそのまま用いると上の例のように演算式に%を使う煩わしさが生じ，FORTRAN77で記されたsubroutineを利用するのに大変不便である．これを避ける1つの方法は，subroutineの仮引数には普通の変数，実引数には構造型の対応する成分を使用し，subroutine内では普通の変数の演算式を使うことである．今の場合について最も単純な1つの案を次に示す．

```
subroutine example2 (F, ims, ime, jms, jme, &
kms, kme, ids, ide, jds, jde, kds, kde)
implicit none
real, intent (INOUT)::F (ids:ide, jds:jde, kds:kde)
integer, intent (IN)::ims, ime, jms, jme, &
                      kms, kme, ids, ide, &
                      jds, jde, kds, kde
integer::i, j, k

do i=ims, ime
do j=jms, jme
do k=kms, kme
F (i, j, k)=0.1* F (i, j, k)
end do
end do
end do
end subroutine example2
```

この場合のcall文は風速のx成分を想定すると次のようになる．

```
call example2 (du%var, du%ims, du%ime, &
du%jms, du%jme, du%kms, du%kme, &
du%grid%ids, du%grid%ide, du%grid%jds, &
du%grid%jde, du%grid%kds, du%grid%kde)
```

FORTRAN77で記された演算部分をほとんど修正することなく，構造型を用いたプログラムで使用できることが御理解いただけたと思う．subroutineの引数で対応づけておくと，subroutine内で変数の値が変われば，対応する構造型変数の成分も自動的に変わるので，ミスが起こる余地が少なく便利である．この例では配列の寸法が構造型変数，gridで定義されているとしたため，du%grid%idsのように%が2つ必要になった．こういう例を示したいがためであって，このようにする必然性はない．上の例ではsubroutineの変数はすべて引数で渡すという前提で記した．commonを使うよりミスを防ぎやすいと思う．

アメリカでNCARとNCEP等が協力して開発している非静力学モデルWRF（Michalakes *et al.*, 1999）では，時間積分ステップが始まるところで，上の例に示したのと類似の方法で構造型変数から普通の変数へ変換するためのsubroutineを1つ配置し，以後の力学や物理過程を計算するsubroutineでは普通の変数を使用している．袖領域のデータ交換などでは構造型変数を用いるので，時間積分のステップ毎に1回，構造型変数から普通の変数への変換が行われる．WRFはこの方法を採用した理由として，過去の資産の再利用が容易であること，構造型の定義が異なる場合でも力学，物理部分のsubroutineを変更する必要がないこと等を理由として挙げている．この方法だと第2層のsubroutineを記述するときに構造形を意識する必要がない．

筆者も気象モデルでは力学，物理過程を計算するsubroutineで構造型変数を用いるのは望ましくないと考えている．理由はWRFの説明でつきている．モデル依存（マシン依存も含む）の第1層と，領域分割や袖領域の通信に関係しない第2層の計算は切り離しておく方が，モデルの作成，変更に便利である．

ここでは単純な例で構造型の基礎的説明を行った．WRFはモデルのすべての変数と格子構造情報等を成分として含む構造型，domainを定義し，これの構造型変数を複数宣言して，それらを（指し示す機能の）ポインタで管理することによりモデルのネスト構造をエレガントに管理している．筆者が紹介するには高度過ぎるので詳しい説明は省く．

2.4 module

最後にmoduleを説明する．moduleにはいろいろな機能がある．単純な機能としてはcommon機能の実現としてモデル全体で使用されるglobal変数や定数の宣言がある．FORTRAN77ではcommonをinclude文で使うことが多いから，この意味ではinclude文の代わりといってもよい．この機能は，これまでに存在していた他の機能の代替で簡明なのでここでは説明しない．

Fortran90で新しく導入されたmodule機能は次のようにいえるだろう．構造型が相互に関連する変数等を1つにまとめて管理するのに対し，moduleは関連するいくつかの操作を1つのmoduleにまとめて，ある役割を実現し，プログラムの記述と管理の便利さを

図ったものである．Fortranでの操作は普通subroutineで記述されるから，moduleは相互に関連するsubroutineの集合体と考えてもよい．moduleはそれに含まれるsubroutineの記述に必要な変数や定数の定義も含むので，極論すれば1つのモデルが1つのmoduleから構成されることも不可能ではないが，一般的には1つのモデルは，10～30のmoduleから構成されると考えておけばよい．各moduleには名前を付ける．あるmodule，module_Aの中で別のmodule，module_Bで定義された変数あるいはsubroutineを使用するときには，module_AでuseモジュールBと宣言する必要がある．このことは構造型の事例でもすでに説明した．以下に例を示す．但しsubroutineの引数は今の説明にとって本質でないので省略してある．

```
module module_ex_12
use module_ex_1
implicit none
real::am, bm
integer::im, jm
real, pointer::array（:,:,:）

contains
subroutine ex_msub_12
---------------------------------
---------------------------------
end subroutine ex_msub_12
end module module_ex_12
```

この例は次のように説明される．module_ex_12という名前のmoduleは，module_ex_1のなかで定義された変数或いはsubroutineを利用するので，それをuseしている．implicit none以下containsまではsubroutine ex_msub_12で使用する変数の定義である．必要なら構造型もimplicit noneからcontainsの間で定義される，この例にあるcontainsの説明は省略するが，moduleでsubroutineを定義する前に必ず記す符号と理解しておいていただきたい．この例ではsubroutineは1つだが，いくつでも定義することが可能である．

moduleは目的に応じて多くのsubroutineをまとめて管理できるので便利である．最近のモデルではsubroutineの数が200～300にもなるのが多いので，機能ごとにmoduleで大きくまとめることの意義は大きい．但しこのままでは不便さもある．

この例ではmodule_ex_1の中のどのような変数やsubroutineがsubroutine ex_msub_12で使用されるのかがわからない．これはプログラムを記述したり読んだりするときに不便であるだけでなく，間違いが起こる危険性が大きい．それを防ぐ手だてが2通り用意されている．1つはonlyの使用である．

```
use module_ex_1, only：a
```

とすると，module_ex_1内で宣言されたもののうち，ここではaだけを用いることを意味する．これで使用される変数等が陽に見えるという目的が達せられる．もう1つはmodule内で変数や手続きにpublic属性，private属性を指定することである．その意味は以下の通りである．

- private属性を指定された変数や手続きは，このmoduleの外では使用できず外からアクセスできない．
- public属性を指定された変数や手続きはこのmoduleの外でも使用でき，外からアクセス可能である．

これによりプログラム作成時の安全性が増すとともに，moduleを記述するときも，変数や手続きの役割を明確に意識できるので便利である．private属性が指定してあれば，間違って外からこの変数を変更したりすることができないし，その部分の変更はmoduleの外には影響しないのだから，管理に便利である．つまり不必要な情報を隠蔽していることになる．なおここでの例のようにpublic，private属性を指定しなければpublicと見なされる．moduleはある意味で入れ物で，操作の実体はsubroutineである．publicやprivateの指定があるとそのmoduleで定義されて，外で使用されるsubroutineや構造型変数が容易に識別できて読みやすいという利点もある．

moduleは多くの操作をまとめているのでsubroutineと違って引数の概念がない．それで安全のために，onlyの使用，pubilic，private属性の指定が必須であると思う．

moduleのもう1つの大きな利点としてsubroutineの引数チェックがある．FORTRAN77まではsubroutineは別々にコンパイルされ，実行時に結合編集されるので引数の数や型の対応のチェックは行われず，実行時にミスがあると発見に苦労する．Fortran90でもmoduleは別々にコンパイルされるが，コンパイル時に結合処理が行われるのでmoduleのなかのsubroutineの引数の数や型のチェックも行われ，誤りを

未然に防止することができる.

3 数値シミュレーションモデルの作成と管理

ソフトウェアの高度化・複雑化により，開発・改良・維持に多くの研究者,組織の協力が必要になっている.またこれから分散主記憶型の並列計算機が主流になると想定すると，計算機の能力を使いこなすための労力も従来よりは大きくなる．この解決策の1つがソフト構造の標準化であり，もう1つの側面が計算技術側との協力強化である.

多くの研究者，組織の協力でソフトを開発，改良，維持してゆくためには，ソフトウエアが多様な計算機に容易に移植可能で，読みやすく間違いが少ないように，プログラムのコーデングと構造にある種の標準スタイルが必要である．ヨーロッパや米国ではFortran90で数値シミュレーションプログラムを記述するときのコーデングルールやモデル構造の標準スタイルが提案されている．例えばEuropean Standards for Writing and Documenting Exchangeable Fortran90 Code(http://nsipp.gsfc.nasa.gov/infra/eurorules.html), Common Modeling Infrastructure Working Group (http://janus.gsfc.nasa.gov/~mkistler/infra/master.html).

日本でもFortran90の導入を契機に気象シミュレーションソフトウェアの標準スタイルというようなものを確立してゆく必要があろう.

標準スタイルを導入しても，膨大なソフトの作成・改良・維持・管理を気象の専門家だけで遂行するのは極めて困難である．すでに述べたようにモデルは，計算技術あるいは機種依存の側面が強い階層と，物理や力学依存の強い階層にわけることができる．標準スタイルを作る段階から計算技術者側との協力を強く進めてゆくことが重要ではなかろうか.

科学技術振興調整費総合研究「高精度の地球変動予測のための並列ソフトウエア開発に関する研究」(研究代表者,東京大学気候システム研究センター長,住　明正教授）に関連して，しばらくモデルの並列化の仕事の渦中にいた．ここから痛切に感じたのは「気象モデル作成も手作りの独立作業から共同或いは分業形態の作業が必要な時代になっている」ということである.日本の気象モデル作りでは情報技術側との共同作業が不足し，それがモデル発展のネックになりつつあるのではないかと危惧している.

並列計算機の出現，情報ネットワーク技術の発展等により，大規模計算を実施できる環境の充実はめざましい．しかし応用ソフトウェアの開発は依然として手作りで,過去の資産の再利用も必ずしも容易ではない.大規模ソフトウェアの開発を可能な限り効率化するための手だては，プログラム構造の標準的スタイルの普及，プログラムの階層性を考慮した気象の専門家と情報技術側との分業と協力の強化であろう.

4 終わりに

Fortran90の新しい機能を解釈し，関連して気象モデル作成のあり方について意見を述べた．ここで欠けているのは実際の経験である．新しい機能の計算スピードはどうなのか．動的割付と解除を繰り返すのは計算スピードの点で望ましくないとか，動的割付は静的割付に比較して計算スピードが遅いのではないかとの懸念も聞く．ただしまだ経験が不足しているし，コンパイラの機種依存もある．これから多くの人が経験を積み，不十分な点があれば，便利な機能が便利に使えるように改善を求めていくことも必要である.

説明は基本的事項のみに限定し，module機能を使うと並列の袖領域のデータ交換プログラムの作成が大変容易になるとか，指し示すという本来の意味でのポインタ機能を利用すると並列で分割した領域の管理がエレガントにできることなどの具体的な説明は省略した．ここでの議論の目的ではないし，それらの説明は筆者の能力をこえている.

謝　辞

財団法人高度情報科学技術研究機構の志村和紀，荒川隆両氏にはFortran90について多くのご教示を頂き，この報告の草案に貴重なコメントを頂いた．お礼申し上げます．天気編集委員の新野理事から報告内容について,気象の立場から貴重なコメントをいただき,改稿に大変役立った．感謝致します.

参考文献

Michalakes, J., J. Dudhia, D. Gill, J. Klemp, W. Skamarock, 1999：Design of a next-generation wearher research and forecast model. Proceeding of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology.