付録 I　ローカル・コーディングルール

ATM のソースコードは Fortran で書いている。またバッチ処理はシェルスクリプトで書いている。ここでは、ローカルで定めている Fortran のコーディングルールと、シェルスクリプトを書くときに気をつけていることを記述する。

I.1　ソースコードのローカル・コーディングルール

ソースコードをコーディングルールに則って記述することは、可読性を高める意味がある。ATM 固有のコーディングルールは、Fortran 標準コーディングルール（室井・他, 2002）[1]、NHM（藤田, 2003）および asuca（石田・他, 2014）のコーディングの規則を規範として定めている。

スタイルルール

全般的なルール

- プログラムは原則小文字で書く。ただし組込み関数を除く外部ライブラリ（NuSDaS, NWP など）およびライブラリで宣言されている変数名や、`open`, `close`, `read`, `write` 文の制御項目などは大文字で書く。
- 1 行は 132 文字以内とする。
- 継続行は「`&`」を行末と継続行の始めに入れ、桁位置は前行に揃える。行末で「`,`」を伴う場合は「`, &`」をセットとして前行に揃える。
- 引数内などの文字変数は「`'`」、`write` 文などの文字出力は「`"`」で囲む。

（例）

```
call NUSDAS_INQ_DEF( type1_atm, type2_atm, type3_atm, & !! IN
                   & N_PLANE_NUM                     , & !! IN
                   & nz_atm                          , & !! OUT
                   & 1                               , & !! IN
                   & irtn                              ) !! OUT

open(n_io, FILE='NAMELIST', STATUS='OLD', ACTION='READ')

write(6, '(a, i4, a26)') "RANK, LOG_FILE = ", mpi_myrank, log_mpi
```

宣言文

use 宣言文

- 原則として `use` 文はサブルーチンの中で利用する。モジュール全体での `use` 文は次項の例外を参照。
- `use` 文のうち、`contains` 文の前で宣言するのは基本的に各サブルーチンで共用されるもの：

[1] https://www.mri-jma.go.jp/Project/mrinpd/coderule.html

```
  use nrtype     , only: sp, dp, rp, rrtype
  use mpi_type   , only: mpi_i4type, mpi_rktype
  use mytype     , only: tag, flag
  use variable   , only: mpi_myrank, mpi_rank_num, io_mpi_log, log_mpi
  use const      , only: pi, rad_unit, deg_unit
  use parm       , only: time_epsilon, space_epsilon                       , &
                       & large_null                                        , &
                       & io_rank
  use func       , only: tracer_size_phi, exponential_decay
  use filetool   , only: filetool_searchfile, filetool_searchft
  use timetool   , only: ctime_run
  use mpi_control, only: mpi_ini, mpi_final, mpi_particle_num_set          , &
                       & tracer_allgather_const, tracer_allgather_variable, &
                       & atm_grid_reduce

  implicit none

  include "nusdas_fort.h"

contains
```

の類とする。include 文についても同様とする。

基本的な方針としては、

- あるモジュール内の多数のサブルーチンのうち、大半で呼ばれる一般的な変数は、モジュール全体で use してもよい。
- モジュール内の特定のサブルーチンでのみ呼ぶ場合は、サブルーチン内で use する。
- 変数によっては、引用が多くてもサブルーチンごとに use する方が見易い場合もあるので、その都度検討する。

参照許可宣言文

- モジュールでは private 文を書く（サブルーチンや変数を無条件に公開しない）。
- モジュール間で公開するサブルーチンや変数のみ、public 文で公開する。

（例）サブルーチン make_random_uniform のみ公開

```
module random

  implicit none

  private
  public :: make_random_uniform

contains
! ######################################################################
  subroutine make_random_uniform

    call xorshift32_rng
    ...
    return
  end subroutine make_random_uniform
! ######################################################################
  subroutine xorshift32_rng
    ...
    return
  end subroutine xorshift32_rng
! ######################################################################
end module random
```

ただし型定義（nrtype.f90, mytype.f90）、変数（variable.f90）、定数（const.f90）、パラメータ（parm.f90）のモジュールについては、全体で public 宣言してよい。

save 宣言文

- 静的変数（グローバル変数）をまとめた variable.f90 については、全体で save 宣言してよい：

```
module variable

  use nrtype, only: sp, dp, rp
  use mytype, only: tag, flag

  implicit none

  save

  public
  ...
end module variable
```

配列の宣言文

- 配列データで引数を明示する場合は 1 行に 1 変数とする。継続行「&」の使用は、同じ型であっても引数のありなしで分ける。

（例）

```
  real   (sp):: r2_wrk(1:4), &
              & r4_wrk(1:4)
  integer(sp):: n2_wrk(1:2)
  integer(sp):: irtn
```

配列の割付

- 配列の大きさが予め決まらない場合を含め、allocatable 属性による動的割付を基本とする。ただしサブルーチン内で、始めから終わりまで作業配列を確保するときは自動割付を基本とする。
- 配列の代入は下記のとおりとする：

（よい例）

```
  do jj = 1, ny_gpv
    do ii = 1, nx_gpv
      rain_gpv2(ii, jj) = rain_gpv1(ii, jj)
    end do
  end do

または

  rain_gpv2(1:nx_gpv, 1:ny_gpv) = rain_gpv1(1:nx_gpv, 1:ny_gpv)
```

（よくない例）

```
  rain_gpv2(:, :) = rain_gpv1(:, :)
```

ネームリスト

- 1 変数 1 行ずつ書き、行末にカンマはつけない（原則、小文字で書く）。

- 終わりは「&end」ではなく「/」を用いる。

（例）
```
&namsrcinfo
 n_tracer_esp=250000
 n_stage_esp=1
/
```

サブルーチン

- できるだけ、一つの処理を行う。開発を進めるうちに、一つのサブルーチンに複数の処理が入った場合、サブルーチンの分割を検討する。
- 名前は省略せずに、意味がはっきりと分かるようにつける。

（よい例）
```
  subroutine calc_tracer_size_distribution
```

（よくない例）頭文字だけで表現
```
  subroutine ctsd
```

- サブルーチンコール
  - 呼出し側の配列の引数は明示し、行末に「!! IN」、「!! OUT」などの入出力特性を示すことを基本とする。ただし自明なものは省略してもよい。
  - 一まとまりで意味がある引数（nx, ny や type1, type2, type3 など）については、属性が同じであれば 1 行に書いてよい。
  - 配列の場合は、引数の範囲を明記する。

（例）
```
    call swap_j( rr  (1:nx, 1:ny), & !! IN
               & data(1:nx, 1:ny), & !! OUT
               &        nx,   ny   ) !! IN

    call ctime_run(  50, "OUTPUT_ATM                    " )

    call NUSDAS_ALLFILE_CLOSE( N_FOPEN_ALL, irtn )
```

- 終了前に return 文を書く。
- end 文にはモジュール名、サブルーチン名を書く：

（よい例）
```
  end subroutine drive_atm
end module sub_atm
```

（よくない例）
```
  end subroutine
end
```

演 算

- 整数化・実数化は型を明示する。

（例）

```
int(n_tracer / n_sample, sp)

nint(elapse_time / 60._rp, sp)
```

- 混合演算は（結果が同じでも）型を合わせて演算する。

（例）

```
real( ny_surf_tl319 + 1, rp) / 2
```

ではなく

```
real( ny_surf_tl319 + 1, rp) / 2._rp
```

ただし実数の整数乗は（演算が積からベキに変わるのを防ぐため）例外とする。

（例）

```
work_rz ** 2._rp
```

ではなく

```
work_rz ** 2
```

- if 文の大小比較に「.gt.」や「.ne.」などは使わない。「>」や「/=」を使う。

コメント

- 原則として、1 桁目に「!」を示す。
- 最初：ソースファイルの名前、概要、作成日、更新日を記入し、更新日（・更新者・所属）は、いつだれが最終更新したかが分かるようにする。

（例）

```
! ========================================================================
! Name    : Atm/Fcst/main_atm.f90
! Outline: Main program of the JMA-ATM
!          (The Japan Meteorological Agency Atmospheric Transport Model)
! ========================================================================
! Release: 2019/03/01 T.Shimbori, Meteorological Research Institute/JMA
! Update : 2020/06/22 T.Shimbori, ditto
! Update : 2020/06/24 K.Ishii   , ditto
! ------------------------------------------------------------------------
```

- 仕切：それぞれ以下の書式で 1 桁目から入れる：
  - メインプログラムの宣言部と実行部の仕切

```
! @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

  - サブルーチン間の仕切

```
! ########################################################################
```

- 実行部中のコメント：必要に応じて以下のレベル分けをして、下記の書式で入れる。
  - レベル 0：プログラム全体を通して最も大きな枠組みとして利用（目安：多くてもモジュール全体で数ヶ所)。

```
! ****************************************************************
!   Level 0 comment (2-character indent)
! ****************************************************************
```

– レベル 1：サブルーチン内の構造を見易くするために利用（目安：サブルーチン内で数ヶ所）。レベル 1 の中にレベル 1 は入れず、並列に利用する（「入れ子」にしない）。

```
! ============================================================
!   Level 1 comment (2-character indent)
! ============================================================
```

– レベル 2：レベル 1 の中の各処理の説明に利用（目安：多数）。

```
! ------------------------------------------------
!   Level 2 comment (2-character indent)
! ------------------------------------------------
```

レベル 0, 1, 2 それぞれの仕切の長さは、見易さを考慮して例外あり。

- 行末にコメントを入れる場合は「!!」で始める。
- 文字化けを防ぐため、日本語（2 バイト言語）は使わない。

字下げ（インデント）

- 2 文字分の字下げ（「2 字字下げ」）する。
- contains 文以降は 2 字字下げする（仕切は除く）。
- if 文、do 文などの内部も 2 字字下げを基本とする。
- 条件判断が複数のとき、短い場合は複数行に分けなくてもよいが、分けるときの字下げは下記を参考にする：

（例）

```
if (       center_lat(nn) == 0._rp .and. center_lon(nn) == 0._rp &
   & .and. target_lat(nn) == 0._rp .and. target_lon(nn) == 0._rp ) then
```

- コメントの字下げ（各行の開始位置）は、コード本文の各行に合わせる。すなわちインデントを揃えることを原則とするが、見易さを考慮して例外あり。

（よい例）

```
! ============================================================
!   Output
! ============================================================
do nn = 1, n_stage

  ! ------------------------------------------------
  !   Write: data
  ! ------------------------------------------------
  do kk = 1, nz_gpv
    ...
  end do

end do
```

（よくない例）レベル 1 が入れ子、コメント開始位置がコード本文に合っていない

```
! ============================================================
!   Output
! ============================================================
  do nn = 1, n_stage

! ============================================================
!   Write: data
! ============================================================
    do kk = 1, nz_gpv
      ...
    end do

  end do
```

空白（スペース）

- 「,」、「:」、「::」の前にはスペースを入れず、後にスペースを入れる（英文タイプの句読法）。ただし配列の範囲を「:」で指定するときは後にもスペースを入れない。
- 「(」、「)」の前後には原則スペースを入れない。ただし、`if` 文や `do while` 文など、次に条件文が入る「(」、「)」の前後はスペースを入れる。またサブルーチン名や関数名と「(」の間はスペースを入れず、「(」の後は前行と桁位置を揃えるためにスペースを入れてもよい。
- 二項演算子の両脇はスペースを開ける。ただし、ループ変数の上下限値や配列の引数など、一まとまりにした方が可読性が上がる場合は省略できる。

（例）

```
allocate(grid_atm_air_concent(1:nx_atm, 1:ny_atm, 1:nz_atm))

open(11, FILE='NAMELIST_ELEM', STATUS='OLD', ACTION='READ')

read(11, NML=namverif)
write(6, NML=namverif)

close(11)

deallocate(grid_atm_air_concent)

do jj = 1, ny
  do ii = 1, nx
    data_out(ii, jj) = data_in(ii, ny-jj+1)
  end do
end do
```

変数の命名ルール

- 暗黙の型宣言は無効にする：`implicit none`
- 変数の型宣言は、付録 C で述べたように、Numerical Recipes (`nrtype.f90`, Press *et al.*, 1996) または独自（`mytype.f90`）の型定義（Tables C.4, C.5）から `use` して使用する。実数型・整数型の種別 `kind` は省略してよいが、文字型の長さ `len` は明示することを推奨する。

（よい例）
```
use nrtype, only: sp, dp, rp

  real     (rp)   :: dx, dy
  real     (dp)   :: bz
  integer  (sp)   :: nn, irtn
  character(len=4):: element
```

（よくない例）
```
  real        :: dx, dy
  real     (8):: bz
  integer  (4):: nn, irtn
  character*4 :: element
```

- トレーサーに関する変数は「tracer_」で始める。

（例）
```
  real(rp), allocatable:: tracer_size(:) !! Diameter of tracer [m]
  real(rp), allocatable:: tracer_mass(:) !! Mass     of tracer [kg]
```

- 整数の変数は原則、「i_」、「j_」、「k_」や「n_」で始める。

（例）
```
  integer(sp):: n_tracer !! Number of tracer
```

- 検索し易くするため、1 文字変数は避ける。

（よい例）
```
do nn = 1, n_max
  ...
end do
```

（よくない例）
```
do n = 1, n_max
  ...
end do
```

- 一時的な作業変数に「wrk」や「work」はできるだけ使わず、「work_名前」として意味の分かる変数名にする。

（例）
```
  work, work1, work2
```
ではなく
```
  work_angle, work_random1, work_random2
```

また気温と混同し易いため、「temp_名前」は使わない。

パラメータスイッチ・フラグのルール

スイッチ

- ネームリストなどで外から与える変数「n_switch_名前」に対し、ソースコード内で定める対応する固定パラメータは「m_switch_名前」とする。

（例）
```
integer(sp):: n_switch_coordinate_vertical !! Vertical coordinate 1:P, 2:Z

integer(sp), parameter:: m_switch_coordinate_vertical_p = 1 !! p-coordinate
integer(sp), parameter:: m_switch_coordinate_vertical_z = 2 !! z-coordinate
```

- スイッチ変数に関する if 文は、パラメータの値ではなく名前（Table C.2）を書く。

（よい例）
```
if ( n_switch_coordinate_vertical == m_switch_coordinate_vertical_p ) then
  ...
else if ( n_switch_coordinate_vertical == m_switch_coordinate_vertical_z ) then
  ...
end if
```

（よくない例）
```
if ( n_switch_coordinate_vertical == 1 ) then
  ...
else if ( n_switch_coordinate_vertical == 2 ) then
  ...
end if
```

フラグ

- スイッチと同様に、「n_flag_名前」（変数）、「m_flag_名前」（対応する固定パラメータ）で始める。
- フラグ変数に関する if 文の書き方は、スイッチの場合と同様にする。

その他

- 不要なゼロクリアは避ける。
- 出力については、装置番号をサブルーチン filetool_searchft で空き番号を探して与え、open 文で明示的に記述する。
- 行内の「,」、「:」や「::」などについても、できるだけ複数行に渡って揃えて書く：

（よい例）
```
integer(sp), intent(in) ::            n_max
real   (rp), intent(in) :: dx      (1:n_max), &
                         & dx12    (1:n_max), &
                         & in_data1(1:n_max), &
                         & in_data2(1:n_max)
real   (rp), intent(out):: out_data(1:n_max)
```

（よくない例）
```
integer(sp), intent(in):: n_max
real(rp), intent(in):: dx(1:n_max)
real(rp), intent(in):: dx12(1:n_max)
real(rp), intent(in):: in_data1(1:n_max)
real(rp), intent(in):: in_data2(1:n_max)
real(rp), intent(out):: out_data(1:n_max)
```

- 外部ソースの利用などで、上記ルールに適合しないコードはディレクトリを分けておく（例えば、Lib/）。
- 上記ルール化できない部分は、各ソースコードの担当者の書き方に準じる。

I.2　シェルスクリプトの書き方

シェルスクリプトについては、書き方に自由度をもたせるため、ライティングルールを特に定めていない。しかし、ソースコードのコンパイルのような確認機能がなく、一定の可読性は維持したいことから、書き方の指針を設けている。

最 初

- 先頭行は、

```
#!/bin/sh
```

で始める。

- 必要に応じて、シェルの終了・出力オプションと出力ファイルのモードを設定する：

```
set -evx
umask 022
```

シェル変数

- 原則、すべて大文字とする。
- シェル変数による代入は中括弧でくくる。ただし組込み変数にはつけない。

（例）

```
OLD_PATH=${PATH}
echo $1
```

コマンド

- シェルに組み込まれているコマンドを使用し、派生言語（Perl や Ruby など）の使用は必要最小限にする。

（例）整数の計算

```
N_TRACER_ESP=`expr ${N_TRACER_ESP} + ${N_TRACER_STAGE}`
```

（例）浮動小数点を含む計算

```
TOP=`echo "scale=4; ${TOP} * 0.3048 " | bc`
```

または

```
TOP=`echo ${TOP} | awk '{printf "%.4f", ($1 * 0.3048)}'`
```

test コマンド

- [ ] を用い、if 文との組合せは次のように書く：

（例）
```
if [ "${YYYY}" = "" ]; then
  continue
fi
```

- 文字列に関する test は=を用い、シェル変数が空か否かに関わらず、常にダブルクォートでくくる。
（[X${YYYY} = X] としない）

字下げ（インデント）

- ローカル・コーディングルールと同様、「2 字字下げ」を基本とする。
- 改行が複数の場合は\の位置を揃え、継続行はさらに 2 字以上字下げする。

（例）
```
if [    "${ELEM}" = "TDEP" \
     -o "${ELEM}" = "FOUT" \
     -o "${ELEM}" = "DDEP" \
     -o "${ELEM}" = "WDEP" \
     -o "${ELEM}" = "WOUT" \
     -o "${ELEM}" = "ROUT" \
     -o "${ELEM}" = "TCLM" ]; then
```

空白（スペース）

- 不要な空白文字は入れない。
- 改行\の前、test コマンド [ ] やパイプ|の前後は 1 字だけ入れる。
- クォート内の先頭と末尾には必要のない限り入れない。

（例）
```
HHMM_LIST="0000 0600 1200 1800"
```

ヒアドキュメント

- 閉じる文字列には EOF を使う。

（例）
```
cat << EOF > NAMELIST
&namsrcinfo
 n_tracer_esp=${N_TRACER_ESP}
 n_stage_esp=${N_STAGE_ESP}
/
EOF
```

その他

conf ファイルなどで置換する変数は、前後を@一文字以上で囲むことを原則とする：@変数名@

最 後

- 最終行は、

```
exit
```

で終わり、必要に応じて終了コードを引数にして返す。