REGCM_4.3

Developer Guide

Stefano Cozzini

CNR/IOM Democritos, Trieste, Italy

and

Graziano Giuliani

ICTP Earth System Physics Section, Trieste, Italy

Contents

Chapter 1
Introduction

This guide is intended for people interested in developing and enhancing the REGCM_4.3 package. It describes the FORTRAN90 programming style used in REGCM_4.3 and explain the underlying principles of *object-based* and module-oriented programming followed throughout our implementation. The aim is to help the developer to understand the general philosophy followed during our project and easily improve upon the existing code.

1.1 About FORTRAN90 in REGCM_4.3

Release REGCM_4.3 differs significantly from the previous ones (versions 3 and 4.0). The most important change is that the package is undergoing a complete refactoring to be written in FORTRAN90 . This will require in the next future a radical change from the previous FORTRAN77 , a procedural programming language, to an *object-based/oriented* programming protocol.

It is our opinion that FORTRAN77 is no longer the right tool to deal with such a large project, especially when large type of data sets need to be managed across the whole package, and the model itself needs a clean and convenient infrastructure to be coupled with a broad spectrum of complementary physics models to become part of more complete earth climate modelling framework. The main weakness of FORTRAN77 is the lack of any explicit mechanism to express the dependencies among different data structures and procedures. This makes the

development process hard and difficult: any new change in the code requires an analysis of the involved data across the whole code and, in the case of FORTRAN77 , this is done necessarily by analyzing many source files. This procedure could be very error prone, especially for developers not mastering all the data of the entire software project.

On the contrary FORTRAN90 is a modern programming language with features that support new important programming concepts, including some of those used in object oriented (OO) programming. FORTRAN90 is moreover backward compatible with FORTRAN77 : it was therefore possible to introduce FORTRAN90 features into the code in a incremental fashion, and a lot of work already done could be easily incorporated.

REGCM_4.3 represents now a first release towards a full development along the guidelines of modular and OO programming style. The transition from procedural to the modular paradigm and later object -base paradigm is not yet completed in the present release. At this stage, after having analyzed the old data structures and their interdependencies, we have built a first hierarchy of modules containing the relevant data structures. In this way the relationships amongst data are made explicit. Furthermore in some cases we have included directly in the module itself the subroutines acting on the data : this is the concept of classes of methods on OO programming and now data and routines acting on them are tightly linked together. Therefore any modification of the code becomes extremely localized, and adding new features needs a much simpler intervention than in the past. For example, implementing a new algorithm for dust requires just to change a single FORTRAN90 module.

To summarize it has to be stressed that now in FORTRAN90 the building blocks of the program are modules, not subroutines, and to modify the code one has to understand the modules hierarchy build up. The general description of the

latter is presented in the following sections.

1.2 Object-oriented programming in FORTRAN90

Learning FORTRAN90 is easy for FORTRAN77 programmers *W. S. Brainerd* [1990]; *M. Metcalf* [1990]; *Kerrigan* [1993]. A more difficult task is "thinking" in terms of *objects* and *methods Stroustrup* [1991], even though FORTRAN90 is not really an object-oriented, but only an object-based, language.

There is an excellent web site *C. D. Norton* [1996] where OO techniques used in the REGCM_4.3 project are clearly explained and examples are provided. We strongly recommend it to interested people.

Chapter 2

The REGCM_4.3 programming style

The programming style of REGCM_4.3 is intended to be as uniform as possible. Potential contributors to the code are kindly requested to follow the stylistic convention.

2.1 Programming Language

The general programming language is FORTRAN90 . Some FORTRAN77 routines are still present in the package and they should be regarded as temporary, and due to migration from version 3.0. C routines are permissible, provided they are Fortran compatible. File names extensions are:

- `.F90` : for f90 free format code, containing preprocessing directives to be analyzed by fortran preprocessor
- `.f90` : for f90 free format code not to be preprocessed.
- `.c` : for C code

  Compilation is achieved with a hierarchy of Makefiles

- `Makefile` : the Makefile with general and architecture-independent rules
- `Makefile.`*arch* : the architecture dependent Makefiles

Two distinct rules are given in the Makefile:

- `.F90.o` using the macros **$(F90)** and **$(FCFLAGS)** for related flags. The files are preprocessed.
- `.f90.o` rule, using the macros **$(F90)** for compiler and **$(FCFLAGS)** for related flags.

Modules have always the extension `.[fF]90.` and should start with `\mod_`. Avoid uppercase letters, especially for modules, since module names are converted to lowercase by the compiler.

2.1.1 Banned FORTRAN77 language features:

- `common`, instead use modules;
- `include`, instead use modules and `#include`;
- `implicit double precision`, instead use `implicit none`);
- Obsolescent FORTRAN77 features, e.g. see Chapt.1 of *Kerrigan* [1993].;
- FORTRAN77 notation, such as `real*8`, etc.;
- Non-standard FORTRAN77 integer pointers, instead use allocatable arrays, or FORTRAN90 pointers;
- Avoid vendor extensions that diminish portability.

2.2 Target Computers

REGCM_4.3 is primarily developed on Linux Cluster in both its serial and parallel versions using the MPI communication library.

During its developing phase the code is compiled using three different suite of compilers:

- Intel suite compiler (from 10.1 to 12.0)
- gfortran compiler version 4.4 on
- Pgi suite of compiler (version 9.03 on)

The code is, however, developed having in mind portability (see later for a detailed discussion about this issue) and it has also been carefully tested (parallel version) on AIX SP6 IBM platform with xlf compiler.

2.3 Module Template

A `module` is the basic unit of the code, and any function or subroutine is wrapped in a `module`. A `module` contains data to be shared with other REGCM_4.3 modules and data internal to the `module`. The FORTRAN90 attribute `private` should be placed right after the `use` lines in the `module` code. Any items, be data, function or subroutine to be accessed from other modules or the Main program is to be explicitly given the `public` attribute.

Two default subroutines should be present in a `module`:

1. `init_mod_modname` : Allocates dynamic space in heap and set up any internal control flag.
2. `release_mod_modname` : Deallocates resources and cleanup

Any number of function and subroutines can be present in a `module` The variables in the subroutine arguments are specified in the order:

1. input arguments to subroutine
2. output arguments from subroutine
3. logical flags to control internal branching

This method is sometimes used quite liberally. REGCM_4.3 enforces for a safer a cleaner programming usage of the FORTRAN90 attribute `intent(in/out/inout)` for subroutine arguments. An example of a module with a defined subroutine is given in the following example:

```
!::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
!
!    This file is part of ICTP RegCM.
!
!    ICTP RegCM is free software: you can redistribute it and/or modify
!    it under the terms of the GNU General Public License as published by
!    the Free Software Foundation, either version 3 of the License, or
!    (at your option) any later version.
!
!    ICTP RegCM is distributed in the hope that it will be useful,
!    but WITHOUT ANY WARRANTY; without even the implied warranty of
!    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!    GNU General Public License for more details.
!
!    You should have received a copy of the GNU General Public License
!    along with ICTP RegCM.  If not, see <http://www.gnu.org/licenses/>.
!
!::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
module mod_template

  use mod_basicmodule

  private

  real(dp) :: arealvar
  integer :: integervar
  real(dp) , dimension(4) :: fixed_dims_array
  real(sp) , allocatable , dimension(:,:) :: allocatable_2d_array

  public :: allocatable_2d_array     ! Data accessed from outside
  public :: init_mod_basicmodule     ! Allocate space and init data
  public :: release_mod_basicmodule ! Release allocated resources and cleanup
  public :: meaningful_subroutine    ! A subroutine doing some work

  contains

  subroutine init_mod_basicmodule(lfag)
    implicit none

    ! Dummy arguments are to be specified of theyr intent

    logical , intent(in) :: lflag    ! Flag to control behaviour

    ! Local variables

    integer :: istat

    !-----------------------------------------------------------------

    if ( lflag ) then
      allocate(allocatable_2d_array(iy,jx), stat=istat)
    end if

  end subroutine init_mod_basicmodule

  subroutine release_mod_basicmodule
    implicit none

    !-----------------------------------------------------------------

    if ( allocated(allocatable_2d_array) ) then
      deallocate(allocatable_2d_array)
    end if

  end subroutine release_mod_basicmodule

  real(dp) function internal_function(a)
    implicit none
    real , intent(in) :: a

    !-----------------------------------------------------------------
```

```
    ! Constants are to be declared in double precision
    internal_function = dlog(a) * 3.0D0 + 4.0D0
  end function internal_function
  subroutine meaningful_subroutine
    implicit none
    integer :: i , j
    !-----------------------------------------------------------------
    do i = 1 , iy
      do j = 1 , jx
        ! Cast of variables need to be specified
        allocatable_2d_array(i,j) = internal_function(dble(i))
      end do
    end do
  end subroutine meaningful_subroutine
end module mod_template
```

2.4 FORTRAN Parameters

All parameters defined by the FORTRAN90 parameter statements are specified in modules and described in one or more comment cards.

2.5 Arithmetic Precision

All real variables and parameters are specified in 64-bit precision (i.e. real(kind=8)). The future release of REGCM_4.3 will count on FORTRAN90 features in order to set its proper defined precision.

2.6 Other Issues

2.6.1 FORTRAN90 /FORTRAN77 (in)compatibility

To achieve language compatibility the developer should keep in mind that using arrays as dummy arguments can slow down the performances dramatically. Beware of passing FORTRAN90 array descriptors (e.g., `a(:,2:18:2)`) to FORTRAN77 ssubroutines. This forces the FORTRAN90 compiler to make local copies of the array in order to make it contiguous, whereas in FORTRAN90 contiguity is not required with a net loss of speed.

Chapter 3

Useful (?) Tips

3.1 How to debug the code

REGCM_4.3 provides a Debugging facility to help developers and users. In order to activate the debugging facility, the code must be compiled with the macro `DEBUG` enabled. At runtime the code produces different files containing information about different processors. File names are `DEBUG_0` for processors 0 or for a serial run, and `fort.`*nn* for parallel runs, where *nn* is the processor number, numbered from 0 to N-1.

The debug level goes from 0 (very basic information) to 6 (detailed information: beware the `DEBUG_.`*nn* files can be huge).

The default behaviour of the code is DEBUG disabled and debug_level variable equal to 0. DEBUG_* files report quite detailed information about the initialization phase and then silence during the iteration steps. We strongly recommend users/developers to run with this default, especially when they start to simulate new systems.

In the code all the debug section are inserted within

```
#ifdef DEBUG
 ! Something to be done only in debug mode
#endif
```

Bibliography

C. D. Norton, B. K. S., V. K. Decyk, High performance object-oriented programming in fortran 90, Internet Web Pages, http://www.cs.rpi.edu/ szymansk/oof90.html, 1996.

Kerrigan, J. F., *Migrating to Fortran 90*, 1st ed., O'Reilly, Sebastopol, CA, 1993.

M. Metcalf, J. R., *Fortran 90 explained*, 1st ed., Oxford University Press, Oxford, UK, 1990.

Stroustrup, B., *The C++ programming language*, 2 ed., Addison-Wesley, Reading, MA, 1991.

W. S. Brainerd, J. C. A., C. H. Goldberg, *Programmer's guide to Fortran 90*, 1st ed., McGraw-Hill, New York, NY, 1990.