Checkpoints and Documentation

Davide Maniscalco

October 28, 2019

Abstract

This paper consists of two exercises, the first about writing a subroutine for checkpoints, the second about re-writing the last exercise of the first week with proper documentation, comments, pre and post conditions, error handling, checkpoints.

NOTE ABOUT THE COMPILATION: The delivered file 'Ex3-Maniscalco-CHECKPOINTMODULE.f90' *is not a program*, but just a module. Its purpose is of being used as an external module for other programs, and it should be compiled the following way:

`gfortran -c Ex3-Maniscalco-CHECKPOINTMODULE.f90`

If it is wanted to make some trials about the behaviour of the module, uncomment the small example program written below the module and compile normally. Furthermore, the delivered file 'Ex3-Maniscalco-CODE.f90' is the code for the second exercise only, and it *uses the checkpointmodule*. Therefore it *must* be compiled the following way:

`gfortran Ex3-Maniscalco-CODE.f90 Ex3-Maniscalco-CHECKPOINTMODULE.o`

after the compilation of the checkpoint module.

Exercise 1: Checkpoints

Code development

The full code is reported in appendix A. The module starts with an interface operator containing ten procedures, each of which is a subroutine that is defined in below.

```
INTERFACE CHECKPOINT
     MODULE PROCEDURE LINE, INTEGER_CHECK, REAL_CHECK, COMPLEX_CHECK
    ,integer_ARRAY_CHECK
     MODULE PROCEDURE
    real_ARRAY_CHECK,doublecomplex_ARRAY_CHECK,integer_MATRIX_CHECK,
    real_MATRIX_CHECK, doublecomplex_MATRIX_CHECK
  END INTERFACE CHECKPOINT
```

The purpose of the checkpoints is to print variables that one may want to check, calling the subroutine written for that variable type, in addition to a message string. The code is developed for the following types:

- integer
- real
- complex double
- integer vector

- real vector
- complex double vector
- integer matrix
- real matrix
- complex double matrix

.

The first subroutine is instead just meant for string printing (to understand, for example, where a code crashes through target printings). A do-loop was implemented for subroutines concerning matrices in order to print them in a readable way. Here is reported as an example the subroutine for real arrays:

```
SUBROUTINE real_ARRAY_CHECK(debug,string,arrayvar,dim)
  INTEGER dim
  LOGICAL debug
  CHARACTER(LEN=*) :: string
  REAL, DIMENSION(dim) :: arrayvar

   IF(debug.eqv..false.) THEN
     RETURN
  ELSE
    print*, string, arrayvar
  END IF
END SUBROUTINE real_ARRAY_CHECK
```

Each subroutine takes as inputs:

- **A logical 'debug' variable.** If false, it exits immediately from the subroutine, if true the subroutine works and prints what is required.
- **A string.** To permit to print a message from time to time

In addition, each subroutine takes simply as inputs the variable to be printed and, for matrices and vectors, one or two integers containing the dimension(s) also.

Results

Here is shown just the result of the test of the checkpoint module made with the program written after it, still reported here:

```
PROGRAM TEST
USE CHECKPOINTS

 IMPLICIT NONE
  LOGICAL  debug
  INTEGER :: a
  REAL, DIMENSION(5) :: vector

   a = 2
   vector(1) = 1
   vector(5) = 5
  debug=.true.

  CALL CHECKPOINT(debug,'line test')
  CALL CHECKPOINT(debug,'variable a value is: ',a)
```

```
  CALL CHECKPOINT(debug,'array vector is: ',vector,5)

  STOP
END PROGRAM TEST
```

The first 'printing' subroutine, the 'integer', and the 'real vector' subroutines were tested. The output is the following:

```
line test
variable a value is:    2
array vector is:   1.00000000      0.00000000      0.00000000
   0.00000000      5.00000000
```

The `debug` variable is set to `true`, and all the subroutines correctly print what they have to.

Self-evaluation

The purpose of writing a (more) subroutine for checkpoints was achieved. Furthermore, the implemented subroutines were written into a checkpoint module, allowing their usage in other programs, as it will be seen in the next session.

Exercise 2: Documentation

Code development

The previous structure of the exercise wasn't changed; there were only added the new requested features. Many new comments were added, also for the purpose of improving the design of the code. The full code is reported in appendix B.
First of all, it was added a documentation at the beginning of the code, containing information about:

- Title, description
- Author, date
- Compilation
- Variables declaration and initialization
- Matrix product
- Checkpoints
- Time, printing

It refers to the appendix for the full documentation.

With regard to pre-conditions, only one was added to check the equivalence of the number of columns of the first matrix with the number of rows of the second matrix; if not fulfilled, an error string is printed and the program stops. Furthermore, about post-conditions, one for each of the three final products (by row, by column, matmul) was added, in order to check the equivalence of the number of rows and columns with the number of rows and columns of the first and the second matrix respectively.

Instead, as regards checkpoints, all but one were handled via the `CHECKPOINTS` module defined in the previous section. This was used many times in order to check the correctness of the matrices processed by the program. One example is the following:

```
!---------------------------------------------------
      ! Input matrices checkpoint
!----------------------------------------------------
 CALL real_MATRIX_CHECK(debug,'1st matrix is: ',mat1,dim1(1),dim1(2))
 CALL real_MATRIX_CHECK(debug,'2nd matrix is: ',mat2,dim2(1),dim2(2))
```

All `CALL` takes as input the `debug` variable (that is defined and set to true at the beginning of the code), a string, the matrix to be printed and its dimensions. The same is made in the end of the code with the three output matrices.
Furthermore, a 'new' checkpoint about matrices dimensions, not available in the previous module, was added:

```
!--------------------------------------------------------------
      !checkpoint, matrices dimensions
!--------------------------------------------------------------

      IF(debug.eqv..true.) THEN
         print*, '1st matrix dimensions: ',dim1
         print*, '2nd matrix dimensions: ',dim2
         print*, 'by-column matrix dimensions: ',dimp1
         print*, 'by-row matrix dimensions: ',dimp2
         print*, 'matmul matrix dimensions: ',dimp3
      END IF
```

All the main features of the old program remained instead the same.

Results

The result for a matrices product of dimensions $(5 \times 3)(3 \times 2) = (5 \times 2)$ and with the debug variable set to true (i.e. with all the checkpoints subroutines working) is the following:

```
 1st matrix dimensions: 5 3
 2nd matrix dimensions: 3 2
 by-column matrix dimensions: 5 2
 by-row matrix dimensions: 5 2
 matmul matrix dimensions: 5 2

 1st matrix is:
   1.00000000        1.00000000        1.00000000
   2.00000000        2.00000000        2.00000000
   3.00000000        3.00000000        3.00000000
   4.00000000        4.00000000        4.00000000
   5.00000000        5.00000000        5.00000000

 2nd matrix is:
   1.00000000        2.00000000
   1.00000000        2.00000000
   1.00000000        2.00000000

 by-column matrix is:
   3.00000000        6.00000000
```

```
   6.00000000        12.0000000
   9.00000000        18.0000000
   12.0000000        24.0000000
   15.0000000        30.0000000

 by-row matrix is:
   3.00000000        6.00000000
   6.00000000        12.0000000
   9.00000000        18.0000000
   12.0000000        24.0000000
   15.0000000        30.0000000

 matmul matrix is:
   3.00000000        6.00000000
   6.00000000        12.0000000
   9.00000000        18.0000000
   12.0000000        24.0000000
   15.0000000        30.0000000

 Final matrix dimensions: 5 2
 By row time:    0.00000000
 By col time:    0.00000000
 Matmul time:    0.00000000
```

The debug variable is set to true, and all the subroutines correctly print what they have to.

Self-evaluation

The purpose of rewriting the old code with documentation, proper comments, checkpoints and pre/post conditions was achieved. The usage of an external module was learned also. The unique part not be dealt exhaustively is the one about error handling. In particular, it was found a way to deal with bad allocation/deallocation of vectors, that would have worked as in the following example with mat1:

```
ALLOCATE(mat1(nn,mm),stat=my_stat,errmsg=my_msg)
  IF(my_stat /= 0) THEN
  PRINT*, 'ERROR: allocation of first matrix failed: ' //trim(my_msg)
  END IF
```

The ALLOCATE command can take as input an integer, that is 0 if everything is correct, and an eventual error message, to be printed in case of errors. While the code made up with also these commands compiled on the author's personal laptop, it didn't on the server of spiro.fisica. Because there was not the possibility to try it directly on the lab pc, it wasn't inserted in the final code.

A Full code, exercise 1

```
!  Checkpoint subroutines for scalars, arrays, and 2d matrices, for all
!  integers, real, complex double. All the subroutines print a string and
! the variable to be checked.
! ############################################################
MODULE CHECKPOINTS

INTERFACE CHECKPOINT
     MODULE PROCEDURE LINE, INTEGER_CHECK, REAL_CHECK, COMPLEX_CHECK
    ,integer_ARRAY_CHECK
     MODULE PROCEDURE
    real_ARRAY_CHECK,doublecomplex_ARRAY_CHECK,integer_MATRIX_CHECK,real_MATRIX_CHECK,
     doublecomplex_MATRIX_CHECK
  END INTERFACE CHECKPOINT

CONTAINS     !this must be put AFTER the interface definition

!----------------------------------------------------------
SUBROUTINE LINE(debug,string)
    LOGICAL  debug
    CHARACTER(LEN=*) :: string
    IF(debug.eqv..false.) THEN
       STOP
    ELSE
       print*,string
    END IF
  END SUBROUTINE LINE

!----------------------------------------------------------
SUBROUTINE INTEGER_CHECK(debug,string,intvar)
    LOGICAL debug
    CHARACTER(LEN=*) :: string
    INTEGER :: intvar
  IF(debug.eqv..false.) THEN
     RETURN
  ELSE
     print*, string, intvar
  END IF
END SUBROUTINE INTEGER_CHECK

!-------------------------------------------------------------
SUBROUTINE REAL_CHECK(debug,string,realvar)
    LOGICAL debug
    CHARACTER(LEN=*) :: string
    REAL :: realvar
  IF(debug.eqv..false.) THEN
     RETURN
  ELSE
     print*, string, realvar
  END IF
END SUBROUTINE REAL_CHECK

!--------------------------------------------------------------
SUBROUTINE COMPLEX_CHECK(debug,string,complexvar)
    LOGICAL debug
    CHARACTER(LEN=*) :: string
    DOUBLE COMPLEX :: complexvar
  IF(debug.eqv..false.) THEN
     RETURN
  ELSE
     print*, string, complexvar
  END IF
END SUBROUTINE COMPLEX_CHECK
```

```
!-------------------------------------------------------------
SUBROUTINE integer_ARRAY_CHECK(debug,string,arrayvar,dim)
  INTEGER dim
  LOGICAL debug
  CHARACTER(LEN=*) :: string
  INTEGER, DIMENSION(dim) :: arrayvar

   IF(debug.eqv..false.) THEN
     RETURN
  ELSE
    print*, string, arrayvar
  END IF
END SUBROUTINE integer_ARRAY_CHECK

!-------------------------------------------------------------
SUBROUTINE real_ARRAY_CHECK(debug,string,arrayvar,dim)
  INTEGER dim
  LOGICAL debug
  CHARACTER(LEN=*) :: string
  REAL, DIMENSION(dim) :: arrayvar

   IF(debug.eqv..false.) THEN
     RETURN
  ELSE
    print*, string, arrayvar
  END IF
END SUBROUTINE real_ARRAY_CHECK

!-------------------------------------------------------------
SUBROUTINE doublecomplex_ARRAY_CHECK(debug,string,arrayvar,dim)
  INTEGER dim
  LOGICAL debug
  CHARACTER(LEN=*) :: string
  DOUBLE COMPLEX, DIMENSION(dim) :: arrayvar

   IF(debug.eqv..false.) THEN
     RETURN
  ELSE
    print*, string, arrayvar
  END IF
END SUBROUTINE doublecomplex_ARRAY_CHECK

!------------------------------------------------------------------
SUBROUTINE integer_MATRIX_CHECK(debug,string,matrixvar,dim1,dim2)
  INTEGER dim1, dim2
  LOGICAL debug
  CHARACTER(LEN=*) :: string
  INTEGER, DIMENSION(dim1,dim2) :: matrixvar

   IF(debug.eqv..false.) THEN
     RETURN
  ELSE
    print*, string
    DO ii=1,dim1
  print*, matrixvar(ii,:)
    END DO
    print*,
  END IF
END SUBROUTINE integer_MATRIX_CHECK

!-------------------------------------------------------------
SUBROUTINE real_MATRIX_CHECK(debug,string,matrixvar,dim1,dim2)
  INTEGER dim1,dim2
  LOGICAL debug
  CHARACTER(LEN=*) :: string
```

```
  REAL, DIMENSION(dim1,dim2) :: matrixvar

   IF(debug.eqv..false.) THEN
     RETURN
  ELSE
    print*, string
    DO ii=1,dim1
  print*, matrixvar(ii,:)
    END DO
    print*,
END IF
END SUBROUTINE real_MATRIX_CHECK

!-------------------------------------------------------------------
SUBROUTINE doublecomplex_MATRIX_CHECK(debug,string,matrixvar,dim1,dim2)
  INTEGER dim1,dim2
  LOGICAL debug
  CHARACTER(LEN=*) :: string
  DOUBLE COMPLEX, DIMENSION(dim1,dim2) :: matrixvar

   IF(debug.eqv..false.) THEN
     RETURN
  ELSE
    print*, string
    DO ii=1,dim1
  print*, matrixvar(ii,:)
    END DO
    print*,
  END IF
END SUBROUTINE doublecomplex_MATRIX_CHECK


END MODULE CHECKPOINTS

!############################################################
! small testing program
! ##########################################################

!PROGRAM TEST
!USE CHECKPOINTS

 !IMPLICIT NONE
  !LOGICAL  debug
 ! INTEGER :: a
 ! REAL, DIMENSION(5) :: vector

  ! a = 2
  ! vector(1) = 1
  ! vector(5) = 5
  !debug=.true.

  !CALL CHECKPOINT(debug,'line test')
  !CALL CHECKPOINT(debug,'variable a value is: ',a)
  !CALL CHECKPOINT(debug,'array vector is: ',vector,5)

  !STOP
!END PROGRAM TEST
```

B Full code, exercise 2

```
!######################################################################
! POINT 2 EXERCISE 3 INFORMATION THEORY AND COMPUTATION course,
! master degree in Physics of Data, University of Padova, October 2019
! Davide Maniscalco
!-------------------------------------------------------------------
! MATRIX MULTIPLICATION with CHECKPOINTS, COMMENTS, PRE/POST CONDS.,
   DOCUMENTATION
! ------------------------------------------------------------------
! Matrix-matrix multiplication in 3 diff ways with CPU time monitoring
!-------------------------------------------------------------------
!
! #################################################################
! IMPORTANT NOTE ABOUT COMPILATION:
! This code uses, for the checkpoints, an external module, named
   'checkpoints'.
! Therefore the external module file must be compiled in the same folder
   of this code
! with the -c flag. A .mod file and a .o file must deliver after that
   compilation.
! Moreover, this code must be compiled writing the name of the .o file at
   the end, e.g:
! gfortran Ex3-Maniscalco-CODE.f90 Ex3-Maniscalco-CHECKPOINTMODULE.o
!--------------------------------------------------------------
! VARIABLES DECLARATION:
!
! real allocatable arrays for the matrices
! integers for the do loops
! reals for the cpu_times
! integer arrays for matrices dimensions
! debugging logical variable
! ------------------------------------------------------------
! DIMENSIONS INITIALIZATION:

! Dimensions must me handled by hand in the code: nn,mm,ll, meant for the
   matrix-
!matrix product (nn x mm)(mm x ll) = (nn x ll)
! shape function returns an integer array containing the dimension of an
   input matrix
! ------------------------------------------------------------
! MATRICES INITIALIZATION:

! made in the code arbitarily
! first_matrix(ii,jj) = ii (its row number)
! second_matrix(ii,jj) = jj (its column number)
! output matrices are initialized to 0
!---------------------------------------------------------------
! PRODUCTS:

! prodmat1: output matrix product from by-column loop
! prodmat2: output matrix product from by-row loop
! prodmat3: output matrix product from matmul
!---------------------------------------------------------------
! CHECKPOINTS:
! Checkpoints are meant to print matrices and matrices dimension. Set
   debug.eqv..false.
! to deactivate checks.
! -------------------------------------------------------------
! CALL_CPU_TIME:
!
! Will be used to store istant ina real variable. Istants differences will
   be
! made in the end
```

```fortran
! ---------------------------------------------------------------
! TIMES PRINTING:
!
! The print of the final matrix dimensions is based on the matmul one
!#################################################################
!
! --------------------------------------------------------------
! Variables declaration
! ---------------------------------------------------------------
PROGRAM MATRIX_MULTIPLICATION
      USE CHECKPOINTS
      IMPLICIT NONE

      REAL,DIMENSION(:,:),ALLOCATABLE ::mat1,mat2
      REAL,DIMENSION(:,:),ALLOCATABLE ::prodmat1,prodmat2,prodmat3
      INTEGER ii,jj,kk,nn,mm,ll
      REAL T1,T2,T3,T4,T5,T6

      INTEGER, DIMENSION(2) :: dim1,dim2,dimp1,dimp2,dimp3
      LOGICAL debug
!
    -------------------------------------------------------------------------
      !set to false to ignore all debugs
!
    -------------------------------------------------------------------------
      debug = .true.
!
    --------------------------------------------------------------------------
      ! Dimension initialization
      ! Product between matrices of dimensions: (nn x mm)(mm x ll)=(nn x
    ll)
!
    ---------------------------------------------------------------------------
      nn = 5
      mm = 3
      ll = 2

      ALLOCATE(mat1(nn,mm))
      ALLOCATE(mat2(mm,ll))
      ALLOCATE(prodmat1(nn,ll))
      ALLOCATE(prodmat2(nn,ll))
      ALLOCATE(prodmat3(nn,ll))

      dim1 = SHAPE(mat1)
      dim2 = SHAPE(mat2)
      dimp1 = SHAPE(prodmat1)
      dimp2 = SHAPE(prodmat2)
      dimp3 = SHAPE(prodmat3)
!-------------------------------------------------------------
      !checkpoint, matrices dimensions
!-------------------------------------------------------------

      IF(debug.eqv..true.) THEN
         print 1, '1st matrix dimensions: ',dim1
         print 1, '2nd matrix dimensions: ',dim2
         print 1, 'by-column matrix dimensions: ',dimp1
         print 1, 'by-row matrix dimensions: ',dimp2
         print 1, 'matmul matrix dimensions: ',dimp3
      END IF

      ! matrices dimensions precondition
     IF (dim1(2) /= dim2(1)) THEN
      print*, 'Error: first input matrix has ',dim1(2), 'columns, while
    second input matrix has ',dim2(1), 'rows'
      STOP
   END IF
```

```
!----------------------------------------------------------
   ! matrices initialization
! ---------------------------------------------------------

   ! first matrix
       DO ii=1,nn
         DO jj=1,mm
            mat1(ii,jj)=ii
         ENDDO
      ENDDO

      !second matrix
      DO ii=1,mm
         DO jj=1,ll
            mat2(ii,jj)=jj
         ENDDO
      ENDDO

   ! initialization to 0 of prodmatrices
   DO ii=1,nn
         DO jj=1,ll
            prodmat1(ii,jj)=0
            prodmat2(ii,jj)=0
            prodmat3(ii,jj)=0
         ENDDO
      ENDDO
!---------------------------------------------------
      ! Input matrices checkpoint
!----------------------------------------------------
 CALL real_MATRIX_CHECK(debug,'1st matrix is: ',mat1,dim1(1),dim1(2))
 CALL real_MATRIX_CHECK(debug,'2nd matrix is: ',mat2,dim2(1),dim2(2))
!--------------------------------------------------
 ! By column loop, with post condition
!--------------------------------------------------
      CALL CPU_TIME(T3)
      DO kk=1,mm
         DO jj=1,ll
            DO ii=1,nn
              prodmat1(ii,jj)=prodmat1(ii,jj)+mat1(ii,kk)*mat2(kk,jj)
            ENDDO
         ENDDO
      ENDDO
      CALL CPU_TIME(T4)

!post condition
IF ((dim1(1) /= dimp1(1)) .or. (dim2(2) /= dimp1(2))) THEN
 print*, 'Error: by-column product matrix has not compatible dimensions
   with input ones'
 STOP
 END IF
!-------------------------------------------------
 ! By row loop, with post condition
!------------------------------------------------
      CALL CPU_TIME(T1)
      DO ii=1,nn
         DO jj=1,ll
            DO kk=1,mm
               prodmat2(ii,jj)=prodmat2(ii,jj)+mat1(ii,kk)*mat2(kk,jj)
            ENDDO
         ENDDO
      ENDDO
      CALL CPU_TIME(T2)

!post condition
IF ((dim1(1) /= dimp2(1)) .or. (dim2(2) /= dimp2(2))) THEN
 print*, 'Error: by-row product matrix has not compatible dimensions with
```

```
    input ones'
 STOP
END IF
!-----------------------------------------------
 ! Fortran function, with post condition
!----------------------------------------------
      CALL CPU_TIME(T5)
      prodmat3 = MATMUL(mat1,mat2)
      CALL CPU_TIME(T6)

!post condition
IF ((dim1(1) /= dimp3(1)) .or. (dim2(2) /= dimp3(2))) THEN
 print*, 'Error: matmul product matrix has not compatible dimensions with
    input ones'
 STOP
 END IF
!---------------------------------------------------
 ! Output matrices checkpoint
 ! ---------------------------------------------------
 CALL real_MATRIX_CHECK(debug,'by-column matrix is:
    ',prodmat1,dimp1(1),dimp1(2))
CALL real_MATRIX_CHECK(debug,'by-row matrix is:
    ',prodmat2,dimp2(1),dimp2(2))
CALL real_MATRIX_CHECK(debug,'matmul matrix is:
    ',prodmat3,dimp3(1),dimp3(2))
!---------------------------------------------------------
! Times printing
! --------------------------------------------------------
      print 1, 'Final matrix dimensions: ',dimp3(1),dimp3(2)
      print*, 'By row time: ',T2-T1
      print*, 'By col time: ',T4-T3
      print*, 'Matmul time: ',T6-T5
!----------------------------------------------------------
! Deallocations
!----------------------------------------------------------
DEALLOCATE(mat1)
DEALLOCATE(mat2)
DEALLOCATE(prodmat1)
DEALLOCATE(prodmat2)
DEALLOCATE(prodmat3)
!---------------------------------------------------------
! Printing formats
!----------------------------------------------------------
1 FORMAT(A,I0,' ',I0)
!----------------------------------------------------------
      STOP
    END PROGRAM MATRIX_MULTIPLICATION
```