

Assignment 2

Documentation, debugging and derived types (FORTRAN)

31st October 2023

Exercise 1: **checkpoint** subroutine for debugging.

The checkpoint subroutine will print a custom message 'msg'

- (a) Include a control on a logical variable (Debug=.TRUE. or .FALSE.)
- (b) Include an additional (optional) string to be printed.
- (c) Include additional (optional) variables to be printed.

Modules compilation

```
gfortran -c debug.f90
gfortran -c mod_matrix_c8.f90

gfortran -o main.x debugger.o mod_matrix_c8.o main.f90
```

```
module debugger

=====
!
!   This module implements a checkpoint function with
!   levels of verbosity.
!   -----
!
! SUBROUTINES:
!
! checkpoint(debug, verb, msg)
!
!           Inputs  | debug  (logical)  If true, the checkpoints
!                   |                   are printed in output
!                   |
!                   | msg      (string) (optional)
!
program main
  use debugger
  implicit none

  ! test messages
  call checkpoint(debug = .TRUE., msg = 'your message')

  ! this is not executed because DEBUG is false
  call checkpoint(debug = .FALSE., msg = 'your message')

end program
```

Exercise 2: Documentation: Rewrite Exercise 3 from Assignment 1 including

(a) **Documentation** (i.e. **Doxygen** https://en.wikibooks.org/wiki/Fortran/Documenting_Fortran)

(b) **Comments.**

(c) Pre- and post- conditions.

(d) Error handling.

(e) Checkpoints.

Modules compilation

```
gfortran -c debug.f90
gfortran -c mod_matrix_c8.f90
```

```
gfortran -o main.x debugger.o mod_matrix_c8.o main.f90
```

Documentation

```
module debugger

=====
!
!   This module implements a checkpoint function with
!   levels of verbosity.
!   -----
!
! SUBROUTINES:
!
! checkpoint(debug, verb, msg)
!
!           Inputs | debug  (logical) If true, the checkpoints
!                   |                   are printed in output
!                   |
!                   | msg      (string) (optional)
!
!
!
!
! test messages
call checkpoint(debug = .TRUE., msg = 'your message')

! this is not executed because DEBUG is false
call checkpoint(debug = .FALSE., msg = 'your message')

end program
```

Exercise 2: Documentation: Rewrite Exercise 3 from Assignment 1 including

- (a) Documentation
- (b) Comments.
- (c) Pre- and post- conditions.
- (d) Error handling.
- (e) Checkpoints.

Modules compilation

```
gfortran -c debug.f90
gfortran -c mod_matrix_c8.f90
```

```
gfortran -o main.x debugger.o mod_matrix_c8.o main.f90
```

```
! -----
! SUBROUTINE IJK loop order
! -----
subroutine matmul_loop_ijk(m1,m2,m3) ! this is by row
  implicit none
  real, dimension(:, :) :: m1, m2, m3
  integer :: I, K, J
  integer :: ii, kk, jj

  I = size(m1, 1); K = size(m1, 2); J = size(m2, 2);

  ! check everything
  if ( K .ne. size(m2,1)) then ! 1) check m1 and m2 sizes
    call checkpoint(debug = .TRUE., msg = 'wrong input shapes for matrix
product')
    stop
  end if
  if ( I .ne. size(m3,1)) then ! 2) check dim 1 of target matrix
    call checkpoint(debug = .TRUE., msg = 'wrong target shape (1) for
matrix product')
    stop
  end if
  if ( J .ne. size(m3,2)) then ! 3) check dim 2 of target matrix
    call checkpoint(debug = .TRUE., msg = 'wrong target shape (2) for
matrix product')
    stop
  end if
```

Exercise 3: define a **module** which contains a complex matrix **derived type** for operating with matrices in double precision.

Data structure: collection of data variables, relationship among them, functions and subroutine applicable to the data:

- Default datatype: FORTRAN intrinsic data types are INTEGER, REAL, COMPLEX, LOGICAL, CHARACTER
- Create your DERIVED TYPE

Modules compilation

```
gfortran -c debug.f90
gfortran -c mod_matrix_c8.f90
```

```
gfortran -o main.x debugger.o mod_matrix_c8.o main.f90
```

Type Definition

Interface blocks

Function(s) / Subroutine(s)

```
module mod_matrix_c8

type complex8_matrix
    ! store the dimension of the matrix
    integer, dimension(2) :: size
    ! to store the elements of matrix
    complex*8, dimension(:,:), allocatable :: elem
end type
```

```
interface operator(.Adj.)
    module procedure CMatAdjoint
end interface

interface operator(.Tr.)
    module procedure CMatTrace
end interface
```

contains

```
function CMatAdjoint(cmx) result(cmxadj)
    type(complex8_matrix), intent(in) :: cmx
    type(complex8_matrix) :: cmxadj

    cmxadj%size(1) = cmx%size(2);    cmxadj%size(2) = cmx%size(1);
    allocate( cmxadj%val(cmxadj%size(1),cmxadj%size(2)) )
    cmxadj%elem = conjg(transpose(cmx%elem))
end function
```

```
function CMatTrace(cmx) result(tr)
    type(complex8_matrix), intent(in) :: cmx
    complex*8 :: tr
    integer :: ii

    tr = complex(0d0,0d0) ! init to zero before loop
    do ii = 1, cmx%size(1)
        tr = tr + cmx%elem(ii,ii)
    end do

end function
```

```
end module mod_matrix_c8
```

Exercise 3: define a **module** which contains a complex matrix **derived type** for operating with matrices in double precision.

Data structure: collection of data variables, relationship among them, functions and subroutine applicable to the data:

- Default datatype: FORTRAN intrinsic data types are INTEGER, REAL, COMPLEX, LOGICAL, CHARACTER
- Create your DERIVED TYPE

Why?

- Improve readability;
- Faster debugging & code writing

```
program main
  use mod_matrix_c8 ! matrix of complex*8 numbers
  type(complex8_matrix) :: A, B
  complex*8 :: x
```

Manual
initialization

```
! -----
!   INIT
! -----
A%elem = complex(1d0,0d0)
B = .randInit.(/100, 100/) ! square matrix
```

Using
function
for random
initialization

```
! -----
!   MATH OPERATIONS
! -----
! testing Trace (square matrix only)
x = .Tr.B

! testing Adjoint
C = .Adj.B

! -----
!   I/O
! -----
```

```
! writing matrix to file
call CMatDumpTXT(C, 'data/matrix.txt')
print *, "The matrix written to file"
```

```
end program
```