UNIVERSITÀ DEGLI STUDI DI PADOVA

Quantum Information and Computing

2022 - 2023

Nguyen Xuan Tung

12/07/2022

Exercise #02

Physics of Data

Checkpoints

Theory

A Fortran subroutine is a block of code that performs some operation on the input variables, and as a result of calling the subroutine, the input variables are modified.

Code development

- We create a subroutine called *check_point* to perform if *DEBUG_ == TRUE* we are in *'debug mode'.*
- Print the line and arg.
- We run the program *test.*
- Assign 2 real values *x* and *y.*
- If DEBUG_ == TRUE, print the value.

```fortran
subroutine check_point(DEBUG, realarg, line)
    logical       :: DEBUG   ! input, if DEBUG_ == TRUE we are in 'debug mode'
    real          :: realarg ! optional generic real argument
    integer       :: line

    if (DEBUG .eqv. .TRUE.) then
        print *, 'LINE:',line   ! print file and line
        print*,'arg:', realarg
    end if
end subroutine check_point

#define check_real_(realarg) check_point(DEBUG, realarg,__LINE__)

program test
    logical :: DEBUG = .TRUE.
    real    :: x = 3.14159265359, y = 9.53562951413

! DEBUG_ is true, this value should be printed
    call check_real_(x)

    DEBUG = .FALSE.
! DEBUG_ is now false, this value should NOT be printed
    call check_real_(y)

end program test

! Error appears when compiling, in order to compile sucessful, a flag -cpp is used
! gfortran -o checkpoint checkpoint.f90 -cpp
! ./checkpoint
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\Physics-Data-Msc\Quantum-IC\Assignment-2> gfortran -o checkpoint checkpoint.f90 -cpp
PS D:\Physics-Data-Msc\Quantum-IC\Assignment-2> ./checkpoint
 LINE:          30
 arg:   3.14159274
```

Documentation

Documentation and comments

- For every function and subroutine, we create a documentation and comment in order to understand the function/subroutine better.

Result

- We write function to fill the matrix with random variables.
- The matrix dimension stored in a vector of two dimensions: N, M.
- We create a module named *debug which contains several subroutines with different functions:* print matrix in the terminal, print a matrix and its lines, check for custom implemented matrix multiplication.

```
!!
!! @param[in] N : integer, number of columns in matrix
!! @param[in] M : integer, number of rows in matrix
!! @param[in] rand_range : integer, range of random numbers
!!
!! @return a matrixA = real*4, dimension(N,M), real random matrix
!!-----------------------------------------------------------------
    function fill_no_matrix(N, M, rand_range) result(matrixA)
        integer :: N ,M, rand_range
        real*4, dimension(N,M) :: matrixA
        call random_number(matrixA)
        matrixA = rand_range * matrixA

    end function fill_no_matrix

!> @brief function perfrom matrix multiplication through a loop method
!! @param[in] matrixA = real*4, dimension(N,M), real random matrix
!! @param[in] matrixB = real*4, dimension(N,M), real random matrix
!!
!! @return a matrixC = real*4, dimension(N,M), matrixA*matrixB
!!-------------------------------------------------------------------
!Assume A is lhs matrix, B rhs second, C is the result matrix
!first index "i" runs slower in c_ij
    function matrix_multiplication(matrixA, matrixB) result(matrixC)
        integer :: ii, jj, kk
        logical :: check
        real*4, dimension(:,:)  :: matrixA, matrixB
        real*4, dimension(size(matrixA,2),size(matrixB,1)) :: matrixC
```

Derived types

Theory

- The Hermitian adjoint matrix, A, over a field C of complex numbers, is its complex-conjugate transpose matrix.
- The adjoint matrix A denoted as $A^{\dagger}$, $where$: $A^{\dagger} = A^{T*}$
- * is denoted as comjugate
- Trace given matrix A *n x n, is defined:* Tr(A)=$\sum_{i=1}^{n} a_{ii}$
- The following properties hold:
- $Tr^{*}(A) = Tr(A^{\dagger})$
- Det$(A^{\dagger}) = detA^{*}$

Results

- The result shows us the dimension, trace and the adjoint of the matrix generated.

```
PS D:\Physics-Data-Msc\Quantum-IC\Assignment-2> gfortran -o Derivedtypes Derivedtypes.f90
PS D:\Physics-Data-Msc\Quantum-IC\Assignment-2> ./Derivedtypes
 |              (1.0000000000000000,0.0000000000000000)             (2.0000000000000000,0.0000000000000000) |
 |              (3.0000000000000000,0.0000000000000000)             (4.0000000000000000,0.0000000000000000) |
 Dimension:           2           2
 Trace:               (5.0000000000000000,0.0000000000000000)
 Adjoint:
 |             (1.0000000000000000,-0.0000000000000000)            (3.0000000000000000,-0.0000000000000000) |
 |             (2.0000000000000000,-0.0000000000000000)            (4.0000000000000000,-0.0000000000000000) |
```

Derived types

Code development

- A module matrices is written, and inside it derived type data *type*, called cmatrix.
- After data *type,* we perform interfaces for the initialization of the *type,* the *adjoint matrix* and the *trace* computation.
- After that, we create functions to:
- - computes the trace.
- - initializes complex matrix randomly.
- - initializes complex matrix type given the 2d array.
- Then we create subroutines to prints matrix on terminal and writes matrix to file.

```fortran
!!-----------------------------------------------------------------------------------------
    type cmatrix
        integer, dimension(2)                      :: dim        ! dimension of the matrix
        complex*16, dimension(:,:), allocatable :: element
        complex*16                                 :: trace, det
    end type cmatrix

    interface operator(.Adj.)
        module procedure cmatrix_adjoint
    end interface

    interface operator(.Trace.)
        module procedure cmatrix_trace
    end interface

    contains

!> @brief computes the trace
!! @param[in] cmat = type(cmatrix), input complex matrix
!! @return trace = complex*16, trace of thecomplex matrix
!!-----------------------------------------------------------------------------------------
    function cmatrix_trace(cmat) result(trace)
        integer                    :: ii
        complex*16                 :: trace
        type(cmatrix), intent(IN) :: cmat

        if(cmat%dim(1) == cmat%dim(2)) then ! iff the matrix is square
            trace = 0
            do ii = 1, size(cmat%element,1), 1
                trace = trace + cmat%element(ii,ii)
            end do
        else
            trace = 0
            trace = trace/trace
        end if
    end function cmatrix_trace

!> @brief initializes complex matrix randomly
!! @param[in] nrow = integer, number of rows of matrix
!! @param[in] ncol = integer, number of columns of matrix
!! @param[in] range = real, range of numbers: 0<=x<=range
!! @return cmat = type(cmatrix), complex matrix
!!-----------------------------------------------------------------------------------------
```

800 ANNI 1222 • 2022

UNIVERSITAS STUDII PADUANI MCCXXII

UNIVERSITÀ DEGLI STUDI DI PADOVA

Thanks for the attention