Lab3
Computational Physics I - Phys381

Guilherme Contesini , 10140201
Gerswin Magat , 00325287

February 13, 2014

Abstract

Arrays are data constructs consisting of: rank, extent, shape and size. It is in this order respectively, that FORTRAN describes such an array. They can vary between data types from characters to real numbers. Arrays are conformable if parameters: rank, shape and size are equal. Matrices for example can be declared as followed:" REAL, DIMENSION (Y,X) :: M" , where Y is the column and X is the row. Subroutines and functions are operations on variables within a code, either changing the value or returning a new one. A subroutine has the property of returning a void, whereas a function can return one or more values. Modules are like packages and are more enticing to programmers that are more familiar with object-oriented programming such as Java or python. Modules are a collection of subroutines and functions and can be used as long as the correct parameters are passed into the respective subroutine or function.

1 Introduction

In part A,

We will be exploring the functions of arrays and how to handle them accordingly. An array is a systematic arrangement of objects, or a general data structure. It can be created in a number of different dimensions. For example, a 1-D array could be a string of characters, a 2-D array could be the basic multiplication table, a 3-D array could examine temperature vs wind-chill vs humidity to calculate what the temperature feels like for us. A matrix is a more specific type of an array. A matrix is a two-dimensional array containing number values, and is usually used in mathematical systems. We will be exploiting the uses of "dynamic allocation" in steps: declarations, allocations and deallocations and the implications of what all of them mean in code.

In part B,

we learn about the uses of functions, subroutines and modules. Using functions give you the advantage of using a single code for multiple calculation instances, and the ability to check each function to make sure that it works correctly as a part of a larger program. A subroutine is very similar to a function, with the exception of a value in return. A subroutine can handle already given values for variables and modify them. Modules are used when programs get too large and become difficult to use a single file for the source code. Modules enable the program to be divided into a number of different files, making it easier to re-use the code in separate programs.

2 Part A : Handling arrays and matrices

2.1 Explaining the code:

```
program phys381arrays
implicit none
integer :: N,M,L
integer :: i,j,k
real , allocatable ,
 dimension(:,:,:) :: x,y,z

!! defines 3 arrays named x,y,z and the
  dimension is going to be defines later.!!

open(12, file="input.dat"
read(12,*) N
read(12,*) M
read(12,*) L
allocate(x(N,M,L))
!! create a 3d matrix x{N}{M}{L} !!
allocate(y(N,M,L))
allocate(z(N,M,L))
do k=1, L
  do j=1, M
    do i=1, N
      x(i,j,k)=float(i)* float(j)* float(k)

      !! In x matrix element i,j,k
         receives i * j * k !!

      y(i,j,k)=float(i)* float(j)* float(k)
      z(i,j,k)=float(i)* float(j)* float(k)
      write(6,"(3F10.6,1x)")
      x(i,j,k), y(i,j,k), z(i,j,k)

      !! write on the screen for
        the 3 next values a float
        the value for x{i}{j}{k},
        y{i}{j}{k}, z{i}{j}{k} !!

    end do
  end do
end do
deallocate(x)

!! free the allocated matrix with
   the name x !!

deallocate(y)
deallocate(z)
close(12)
```

```
end program phys381arrays
```

2.1.1

Using the "read()" function like " read(12,*) N,M,L" will permit that you read all the value from the same line , and associating with N,M,L respectably

2.1.2

I would use the follow command "write(6,"(f8.5,2x,f10.6,4x,f9.5,1x)") x(i,j,k), y(i,j,k), z(i,j,k) ", this means that the first value will appear as a float with 8 digits and with precision 5 (f8.5), give 2 white spaces (2x), the second value will appear as a float with 10 digits and with precision 6 (f10,6), give 4 white spaces (4x), the third value will appear as a float with 9 digits and with precision 5(f9,5), give 1 white space (1x).

2.2 Passing arrays through the argument list

```
program phys381passarray
  integer, parameter :: N=3,M=3

  !! Is an declaring an argument
     which its attribute will be
     declare as a integer number
     in the same line !!

  real, dimension(N,M) :: x, y, z
  real, dimension(N) :: W
  x(1:,1:) = 2.0
  y(1:,1:) = 3.0
  z(1:,1:) = 4.0
  W(1:) = 5.0
  call PassInfo(N,M,x,y,z,W)
  contains
  subroutine PassInfo(ismax,jsmax
  ,xi,yi,zi,Wi)
    integer, intent(in) :: ismax,
    jsmax

    !! The intent(in) attribute
       of argument ismax and
       jsmax means that both
       cannot be changed inside
       the function !!

    real, dimension(1:ismax,1:
    jsmax) :: xi, yi, zi
    real, dimension(1:ismax):: Wi
    write(6,*) xi(1:2,1:1)
    write(6,*) yi(1:2,1:3)
    do i=1, ismax
      write(14,*) xi(i,:)

      !! Write where tag 14 is
         define, since tag 14
         is not defined on the
         screen by default the
         program will write on
         the screen by default
         the elements of the x
         array !!

    end do
    close(14)
  end subroutine PassInfo
end program phys381passarray
```

3 Part B : Modules, Functions and Subroutines

3.1 General Questions :

3.1.1

Functions and subroutines are Fortrans subprograms. Fortran functions are like mathematical functions, they take a set of input parameters and return a value. Essentially functions can return one, two or more and sometimes none. A subroutine has no value associated with its name. All outputs are defined in terms of arguments. A subroutine is not called into action simply by writing its name, we need to write a CALL statement to bring it into operation; this specifies the arguments and results in storing all

the output values.

3.1.2

The use of common blocks on this code means that, the two variables "a" and "b" have a static memory address so all the operation with the variable will not overwrite the initial value, the variable's value cannot be changed during the program, so you can only read the value. Common block is a robust way for what its call a static pointer on high level languages. Comparing with a file is like a "only read" file, instead of a "write and read" file.

3.1.3

!!(FIX)!! Fortran 90 has a capability called modules. Its a way to provide the programmers, on a compact way, functions commonly used.It very similar to a main program, except for something that are specific to modules. The differences between a program and a module are the following:

- A module starts with the keyword MODULE and ends with END MODULE rather than PROGRAM and END PROGRAM.
- A module has specification part and could contain internal function,but it does not have any statements between the specification part and the keyword CONTAINS.
- A module does not contains any statements to be executed as in a program.
- A module cannot exist alone; it must be used with other modules and a main program

3.1.4

→By saying implicit none at the beginning of the module, you dont have to say it at the beginning of the subroutines within the module. (TRUE!)

→Modules, unlike program blocks, cannot have code that is not contained within a procedure. (TRUE!)

→Modules can hold data, declared, as in a common block (see example above), before the contains statement.(TRUE!)

3.2 Subroutines and Functions

3.2.1

```
program main
  implicit none
  real :: a,b
  print*, "Enter a real number:"
  read(*,*) a
  print*, "Enter a second
  real number:"
  read(*,*) b
  call sumtworeal(a, b)
end program main

subroutine sumtworeal(a,b)
  real, intent(in) :: a,b
  write(6,*),"The sum of the
  two real numbers is:",a+b
end subroutine sumtworeal
```

3.2.2

```
program main
  implicit none
  real :: a,b
  real :: sum_of_two_real_2
  print*, "Enter a real number:"
  read(*,*) a
  print*, "Enter a second
  real number:"
  read(*,*) b
  call sum_of_two_real_1(a,b)
  print*,sum_of_two_real_2(a,b)
end program main

function sum_of_two_real_2(x,y)
  real :: x,y
  print*,"The sum of the two real
```

```
   numbers is:",x+y
end function sum_of_two_real_2

subroutine sum_of_two_real_1(a,b)
  real, intent(in) :: a,b
  print*,"The sum of the two real
   numbers is:",a+b
end subroutine sum_of_two_real_1
```

3.3 Modules

3.3.1

```
program main
  implicit none
  real :: a , b
  real :: sum_2

  ! Declare the the type of the function !

  print*, "Enter a real number:"
  read(*,*) a
  print*, "Enter a second real number:"
  read(*,*) b
  call sum_1( a, b )

  ! Call the subroutine !

  print*,sum_2( a, b )

  ! print and call the internal function !

end program main

function sum_2( x, y )

  ! Declare the name of the functions and
    the arguments  !

  real :: x , y

  ! Declare two variables received
    as arguments !

  print*,"The sum of the two real numbers
   is:",x+y

  ! Print the sum of the two variables !

end function

subroutine sum_1( a, b )

  ! Declare the name of the subroutine and
    the arguments !

  real, intent(in) :: a , b

  ! Declare two variables received as
    arguments and use the values from
    the main program that cannot be
    changed inside this subroutine !

  print*,"The sum of the two real
   numbers is:",a+b

  ! Print the sum of the two variables !

end subroutine
```

3.3.2

```
!!!!!!!!!!!!!!!!!!!!!!!!!
!On a different file named!
!"functions.f90 write     !
!!!!!!!!!!!!!!!!!!!!!!!!!
module functions

  contains

  function sum_2( x, y )

    real :: x , y
    print*,"The sum of the two real
    numbers is:",x+y

  end function

  subroutine sum_1( a, b)
```

```
    real, intent(in) :: a , b
    print*,"The sum of the two real
    numbers is:",a+b

  end subroutine

end module functions

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Now compile the code above using the !
!  following command on the terminal:  !
!  " gfortran -c functions.f90 "       !
!  Now on a new file write the code    !
!  named "main.f90":                   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

program main

  use functions
  real :: a , b

  print*, "Enter a real number:"
  read(*,*) a
  print*, "Enter a second real number:"
  read(*,*) b
  call sum_1( a, b )
  print*, sum_2( a, b )

end program main

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! now compile the code above using the       !
! following command on the terminal:         !
! "gfortran -o main.exe main.f90 functions.o"!
! now run the program " ./main.exe " and     !
! it should work                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

3.3.3

If we do the "Call sum2(x,y,z)" without declaring z on the subroutine we will receive the following error: " Warning: More actual than formal arguments in procedure call at (1)"

3.3.4

The same thing happened if we "Call sum2(x,y,z)" without declaring z on the subroutine on the module version, the error appear: " Error: More actual than formal arguments in procedure call at (1)"

We can conclude that an error was expected since the subroutine was not suppose to receive an extra argument and the was not declaration either.

4 Conclusion :

We have examined thoroughly the basic uses of arrays/matrices, functions, subroutines and modules. These tools should help us efficiently write code for the upcoming labs and assignments. All of these Fortran functions help us to store data in specific places, and call on them from within the program or from outside of the program.