Numerical Methods I
Introduction to Programming in Fortran

Instructor : Dr. Kishalay Mitra

Department Of Chemical Engineering

Indian Institute of Technology Hyderabad

kishalay@iith.ac.in

(https://sites.google.com/site/kishalaymitra/)

Introduction to Fortran Programming

- Fortran, as derived from Formula Translating System, is a general-purpose, imperative programming language.
- It is used for numeric and scientific computing.

Writing first Fortran program

- Launch Eclipse
- Select a workspace folder
- Once inside Eclipse:
  - ✓ Click File – New – Fortran Project
  - ✓ Project name: Numerical Methods
  - ✓ Project type: Executable (Gnu Fortran on Windows)
- Click Next, and Finish.
- IDE (Eclipse)
  - ✓ Project Explorer
- In Project Explorer, right click on Numerical Methods (project), click on New – Fortran Source File.
- Source file name: mainProgram.f95
- Click Finish.

Fortran - Basic Syntax

Auto-generated by Eclipse -

```
program mainProgram
   implicit none
end program mainProgram
```

- All Fortran programs start with the keyword program and end with the keyword end program, followed by the name of the program.
- The compiler reads each line within the program-end program block, and translates it to machine language.
- Words in **purple** are keywords. They are part of the language of Fortran. The compiler understands these words, without us having to explicitly define them.
- Fortran is not case-sensitive. **Program, program, and pRoGrAm** are read and understood the same way by the compiler.
- The **implicit none** statement allows the compiler to check that all your variable types are declared properly.

Implicit Typing

- Older versions of Fortran allowed a feature called implicit typing, i.e., you do not have to declare the variables before use.
- If a variable is not declared, then the first letter of its name will determine its type.
- Variable names starting with i, j, k, l, m, or n, are considered to be for integer variable and others are real variables.
- However, you must declare all the variables as it is good programming practice. For that you start your program with the statement −
    - implicit none
- This statement turns off implicit typing.

A Simple Program in Fortran

Let’s write a program that prints “Hello World” **(Example 1)**–

```
program welcome
implicit none
print *, “Hello world”
end program welcome
```

To run the code (perform these steps every time you run the code after making changes to it):

- File – Save [Cltr + S]
- Click on the hammer button to Build. This calls the compiler, which generates machine code, and creates a .exe file [Cltr + B].
- Click on the green play button to Run the executable file [Cltr + F11].
- In the Run As dialogue box, select Local Fortran Application. Click OK (this needs to be done only the first time a project is run, or after an error.)
- View output in the Console.

Output: Hello world

Character and String Declaration

- Declaring a character/string type data – character(len =) :: variable_name
- The length of the string can be specified by len specifier.
- If no length is specified, it is 1.

For example,

character :: reply, gender
We can assign a value like,
reply = ‘N’
gender = ‘F’

Example 2: Print your name as follows: Here is Ms/Mr. xxxxx using character declaration.

```
program hello
implicit none
   character(len = 13) :: surname, firstname
   character(len = 5) :: title
   title = 'Mr.  '
   firstname = 'Rohan  '
   surname = ‘Sharma'
   print *, 'Here is ', title, firstname, surname
end program hello
```

Output -
Here is Mr. Rohan Sharma

- The concatenation operator //, concatenates characters.

name = title//firstname//surname

Extracting Substrings

- In Fortran, we can extract a substring from a string by indexing the string, giving the start and the end index of the substring in a pair of brackets.
- This is called extent specifier.

- **Example 3: Extract the substring ‘world’ from the string ‘hello world’ −**

```
program subString

   character(len = 11)::hello
   hello = "Hello World"
   print*, hello(7:11)

end program subString
```

Output- World

Fortran - Numbers

Numbers in Fortran are represented by three intrinsic data types −

- Integer type – holds only integer values
- Real type – stores floating point numbers
- Complex type – stores complex numbers,

  the generic function **cmplx()** creates a complex number

Fortran - Data Types

Fortran provides five intrinsic data types -

- Integer type: The integer types can hold only integer values.
- Real type: It stores the floating point numbers, such as 2.0, 3.1415, -100.876, etc.
- Complex type: It is used for storing complex numbers.
- Logical type: It stores logical Boolean values.
- Character type: It stores characters or strings.

Fortran - Variables

- A variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable should have a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- The name of a variable can be composed of letters, digits, and the underscore character.
- A name in Fortran must follow the following rules −
    - It cannot be longer than 31 characters.
    - It must be composed of alphanumeric characters (all the letters of the alphabet, and the digits 0 to 9) and underscores (_).
    - First character of a name must be a letter.
    - Names are case-insensitive.

Variable Declaration

- Variables are declared at the beginning of a program (or subprogram) in a type declaration statement.
- Syntax for variable declaration is as follows − **type-specifier :: variable_name**
- For example:

```
integer :: total
real :: average
complex :: cx
logical :: done
character(len = 80) :: message ! a string of 80 characters
```

- Later we can assign values to these variables, like,

```
total = 20000
average = 1666.67
done = .true.
message = “Welcome to Numerical Methods”
cx = (3.0, 5.0) ! cx = 3.0 + 5.0i
```

✓ Comments in Fortran are started with the exclamation mark (!), as all characters after this (except in a character string) are ignored by the compiler.

Arithmetic Operators

- Following table shows all the arithmetic operators supported by Fortran.
- Assume variable A holds 5 and variable B holds 3 then −

Operator	Description	Example
+	Addition Operator, adds two operands.	A + B will give 8
-	Subtraction Operator, subtracts second operand from the first.	A - B will give 2
*	Multiplication Operator, multiplies both operands.	A * B will give 15
/	Division Operator, divides numerator by de-numerator.	A / B will give 1
**	Exponentiation Operator, raises one operand to the power of the other.	A ** B will give 125

Example 4: Add two integers.

```
program add
implicit none
integer::x,y,z
x = 2
y = 3
z = x + y
write(*,*) z
end program add
```

Output: 5

- Variables are declared, all in one place, immediately after the implicit none statement.
- Multiple variables of the same type can be declared on the same line by separating names by commas.
- Variables are assigned values using the = operator.
- Variables can also be assigned values using standard mathematical operations like +, -, *, /, etc.
- ‘write’ prints the variable value.

Example 5: Rewrite ex. 4, but this time let the user enter the two numbers to be processed.

```
program mainProgram
   implicit none
   integer :: integer1, integer2, sumOfIntegers, productOfIntegers
   write(*,*) "Sum of two integers (with user input)"
   write(*,*) "Enter the first integer:"
   read(*,*) integer1
   write(*,*) "Enter the second integer:"
   read(*,*) integer2
   sumOfIntegers = integer1 + integer2
   write(*,*) "Sum of ", integer1, " and ", integer2, " is equal to ",  sumOfIntegers
end program mainProgram
```

Output:

```
Sum of two integers
Enter the first integer:   2
Enter the second integer:  3
Integer  1  = 2
Integer  2  = 3
Sum  of  2  and  3  is  equal  to 5
```

Ex: 6 – Divide 2 real numbers and print the output in real and integer format

```
program division
implicit none
   ! Define real variables
   real :: p, q, realRes
   ! Define integer variables
   integer :: i, j, intRes
   ! Assigning  values
   p = 2.0
   q = 3.0
   i = 2
   j = 3
   ! floating point division
   realRes = p/q
   intRes = i/j
   print *, realRes
   print *, intRes
end program division
```

Output

```
0.666666687
0
```

Ex: 7 – Generate a complex number from integer and real numbers

```
program createComplex
implicit none

   integer :: p = 10
   real :: x = 5.17
   print *, cmplx(p, x)

end program createComplex
```

Output:

```
(10.0000000, 5.17000008)
```

FEW MORE…

1) Perform the following operation on two real numbers: x+y/x*y. Identify the sequence in which arithmetic operations are performed.

2) Write a program for calculating the area of a circle.

3) Convert a character to an integer and vice versa.

4) Find the ceil and floor of any real number using Fortran numerical functions (that is by using the commands Ceiling and Floor).

5) Compute the horizontal and vertical position x and y respectively of a projectile after a time, t –
where, x = u t cos(a) and y = u t sin (a) - g t^2 / 2

6) Search for the substring ‘test’ in the string ‘My test is on Monday’ (Refer: Index function)

FORMAT STATEMENT

- The format statement allows you to mix and match character, integer and real output in one statement.
- **write** (*,*). The first * tells the compiler to write the output to the console and the second * tells the compiler the format in which we want to view the output.
- The **format** statement.
  - Must have a label (number to identify the line on which it appears).
  - Replace the second * in **write** (*,*) by the label to use the format specified.
  - A single **format** statement can be used by multiple **write** statements (that use the same format).

Examples:

- To format characters/strings, use a. a5: string of five characters
- To format integers, use i. i4: integer of up to 4 digits (for smaller integers, the left portion of the output is blank, for larger ones **** are displayed instead)
- To format real numbers, use f. f8.2: real numbers with 5 digits before, and 2 digits after the decimal point (8 columns total, including the decimal point).

▪ Play around with the format statement till you get a satisfactory output. For example, what happens when i1 is used in the format statement, and the integer has two or more digits? What happens if you use i4 to write an integer with a single digit?

▪ While printing a pre-defined string using **write** ("text written within quotes"), count the number of characters in the string (including spaces at the ends, and elsewhere) to arrive at the a- number.

▪ The format statement can appear anywhere in the code (not necessarily after or before the write statement that uses it.)

Example 8: Take 2 real numbers as inputs and find their product with proper formatting of the output.

```
program mainProgram
   implicit none
   real :: number1, number2, productOfNumbers
   write(*,*) " Product of two numbers”
   write(*,*) "Enter the first number:"
   read(*,*) number1
   write(*,*) "Enter the second number:"
   read(*,*) number2
   productOfNumbers = number1 * number2

   write(*,10) "Number 1 = ", number1
   write(*,10) "Number 2 = ", number2
   10 format(a12, f5.2)
   write(*,20) "Product of ", number1, " and ", number2, " is equal to ",
   productOfNumbers

   20 format(a11, f5.2, a5, f5.2, a13, f5.2)

end program mainProgram
```

Output -

```
Product of two numbers

Enter the first number:  2.5
Enter the second number: 4.1
Number 1 = 2.50
Number 2 = 4.10
Product of 2.50 and 4.10 is equal
to 10.25
```

Fortran - Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators

Relational Operators

Assume A = 10 and B = 20, then −

Operator	Equivalent	Description	Example
==	.eq.	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
/=	.ne.	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	.gt.	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	.lt.	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	.ge.	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	.le.	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

- Logical operators in Fortran work only on logical values .true. and .false.
- Assume variable A holds .true. and variable B holds .false. , then –

Operator	Description	Example
.and.	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A .and. B) is false.
.or.	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A .or. B) is true.
.not.	Called Logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A .and. B) is true.
.eqv.	Called Logical EQUIVALENT Operator. Used to check equivalence of two logical values.	(A .eqv. B) is false.
.neqv.	Called Logical NON-EQUIVALENT Operator. Used to check non-equivalence of two logical values.	(A .neqv. B) is true.

Intrinsic Functions

- FORTRAN is especially useful for mathematical computation because of its rich library of inbuilt functions (***intrinsic functions***).

function name	type of argument	type of result	Definition
sin(x)	real	real	sine
cos(x)	real	real	cosine
tan(x)	real	real	tangent
atan(x)	real	real	arctangent
abs(x)	real/integer	real/integer	absolute value
sqrt(x)	real	real	square root
exp(x)	real	real	e^x
log10(x)	real	real	$\log_{10}x$

- Trigonometric functions are calculated in radians (1 radian = 180/Pi degrees).

Fortran - Decisions

- ✓ The basic syntax of an if… then statement is −

```
if (logical expression) then
   statement(s)
end if
```

Example 9: Take user input for CGPA (use 8.5) and print distinction if it greater than 7.5.

```
program ifprog
    implicit none
   ! local variable declaration
   real :: cgpa
   write(*,*) "Enter cgpa"
   read(*,*) cgpa
   ! check the logical condition using if
statement
   if (cgpa > 7.5 ) then

   !if condition is true then print the following
   print *, "distinction"
   end if

end program ifprog
```

Output: distinction

Syntaxes of decision making constructs

If…then...else

```
if (logical expression) then
   statement(s)
else
   other_statement(s)
end if
```

If…elseif…else

```
if (logical expression 1) then
   ! statement 1
else if (logical expression 2) then
   ! statement 2
else if (logical expression 3) then
   ! statement 3
else
   ! statement 4
end if
```

Nested if

```
if ( logical_expression 1) then
   !Executes when the boolean expression 1 is true
   …

   if(logical_expression 2)then
      ! Executes when the boolean expression 2 is true
      …
   end if
end if
```

Example 10:

Attendance	Score	Output /Print
> 80	> 85	Grade A
> 65	> 75	Grade B
> 50	> 60	Grade C

Take user input as: Attendance = 70 and Score = 78

Example 10: Contd…

Attendance	Score	Output /Print
> 80	> 85	Grade A
> 65	> 75	Grade B
> 50	> 60	Grade C

Take user input as:
Attendance = 70 and Score = 78

```
program grade
   implicit none
   real :: attendance,score
   write(*,*) "Enter attendance"
   read(*,*) attendance
   write(*,*) "Enter score"
   read(*,*) score
   if (attendance > 80 .and. score > 85 ) then
      print *, "Grade A"
   elseif (attendance > 65 .and. score > 75 ) then
      print *, "Grade B"
   elseif (attendance > 50 .and. score > 60 ) then
      print *, "Grade C"
  end if
end program grade
```

Output: Grade B

Fortran - select case

Syntax for select case

```
select case (expression)
   case (selector1)
   ! some statements
   ... case (selector2)
   ! other statements
   ...
   case default
   ! more statements
   ...
end select
```

Syntax for nested select case

```
select case(expression1)

   case (selector1)
   ! some statements (Outer case)
   select case(expression2)
      case (selector2)
      ! some statements (Inner case)
   end select

end select
```

Specifying a Range for the Selector

```
case (low:high)
```

Example 11:

Age	Output /Print
1 to 30	young
31 to 65	old
66 to 100	Very old

Example 11: Contd…

Age	Output /Print
1 to 30	young
31 to 65	old
66 to 100	Very old

Take user input as:
Age = 32

```
program switchcase
   implicit none
   integer :: age
   write(*,*) "Enter age"
   read(*,*) age

   select case (age)
     case (1:30)
     print *, "Young"
     case (31:65)
     print *, "Old"
     case (66:100)
     print *, "Very Old"
   end select

end program switchcase
```

Output: old

Fortran - Loops

Syntax of do loop is −

```
do var = start, stop, step
   ! statement(s)
   …
end do
```

where,

- the loop variable var should be an integer
- start is initial value
- stop is the final value
- step is the increment, if this is omitted, then the variable var is increased by unity

Example 12: Calculate the factorials of numbers 1 to 5.

```
program factorial
implicit none
   integer :: nfact = 1
   integer :: n
   ! compute factorials
   do n = 1, 5
      nfact = nfact * n
      ! print values
      print*,  n, " ", nfact
   end do
end program factorial
```

Output

1	1
2	2
3	6
4	24
5	120

Fortran - Loops

Syntax of do while loop –

```
do while (logical expr)
   statements
end do
```

Example 13: Repeat Ex. 12 using do while loop

	1	1
	2	2
Output	3	6
	4	24
	5	120

Syntax of nested do loop –

```
iloop: do i = 1, 3
   print*, "i: ", i

   jloop: do j = 1, 3
      print*, "j: ", j

      kloop: do k = 1, 3
         print*, "k: ", k

      end do kloop
   end do jloop
end do iloop
```

```
program factorial
implicit none

   ! define variables
   integer :: nfact = 1
   integer :: n = 1

   ! compute factorials
   do while (n <= 5)
      nfact = nfact * n
      print*,  n, " ", nfact
      n = n + 1
   end do
end program factorial
```

FEW MORE…

1) Write a program for printing the Fibonacci series.

2) Given a user-defined number, identify whether it is a prime number or not.

3) Find whether a given number is even or odd.

4) Create the 8 rows of Pascal triangle

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

Fortran - Arrays

- An array is used to store a collection of variables of the same type.
- Arrays can be one-dimensional (like vectors), two-dimensional (like matrices) and Fortran allows you to create up to 7-dimensional arrays.

Declaring Arrays

- Arrays are declared with the **dimension** attribute.
- Syntax of 1-D arrays - **real, dimension( ) :: numbers**
- Syntax of two-dimensional array of integers named matrix –

  integer, dimension (size1,size2) :: matrix

Operations on Arrays

Operation	Example	Meaning
+ (array addition)	Real, Dimension(3,3) :: A1 Real, Dimension(3,3) :: A2 Real, Dimension(3,3) :: B B = A2 + A1	for each i, j: B(i,j) = A2(i,j) + A1(i,j)
- (array subtraction)	B = A2 - A1	for each i, j: B(i,j) = A2(i,j) - A1(i,j)
* (array multiplication) Not the same as matrix multiplication	B = A2 * A1	for each i, j: B(i,j) = A2(i,j) * A1(i,j)
/ (array division)	B = A2 / A1	for each i, j: B(i,j) = A2(i,j) / A1(i,j)
** (array exponentiation)	B = A2 ** A1	for each i, j: B(i,j) = A2(i,j) ** A1(i,j)

Some Intrinsic Array-type Functions

Shape	Number of elements (the extent) in each dimension.
Size	Number of elements an array contains.
Dot_product(A, B)	Returns the dot product of A and B
Maxval(A)	Returns the maximum value in array A
Maxloc(A)	Returns a one-element 1D array whose value is the location of the first occurrence of the maximum value in A
Product(A)	Returns the product of the elements of A
Sum(A)	Returns the sum of the elements of A
Transpose (A)	Transpose of A

Example 14: Add any two user-defined arrays. Say, A = [2.4 3 6] and B = [4 2 9.3]

```
program addition
    implicit none
    real,dimension(3) :: A,B,C
    write(*,*) "Enter array A"
    read(*,*) A
    write(*,*) "Enter array B"
    read(*,*) B
    C = A + B
    print *,C
end program addition
```

Output – 6.4
5
15.3

Example 15: Find the dot product of any two arrays.

[Dot product = sum(A(i) * B(i))]

Say, A = [2.4 3 6] and B = [4 2 9.3]

```
program dotprod
    implicit none
    real,dimension(3) :: A,B
    real,dimension(1) :: C
    write(*,*) "Enter array A"
    read(*,*) A
    write(*,*) "Enter array B"
    read(*,*) B
    C = dot_product(A,B)
    print *,C
end program dotprod
```

Output – 71.4

Dynamic arrays

- A **dynamic array** is an array, the size of which is not known at compile time, but will be known at execution time.
- Dynamic arrays are declared with the attribute **allocatable**.
- To declare a real allocatable array A,

  real, dimension(:,:), allocatable :: A
- The rank of the array, i.e., the dimensions has to be mentioned however, to allocate memory to such an array, you use the **allocate** function. Ex: **allocate (A(size1,size2))**
- After the array is used, in the program, the memory created should be freed using the **deallocate** function. Ex: **deallocate (A)**

Example 16: Take a 2 dimensional allocatable array (matrix) and obtain the elements of the matrix as follows:

A(1,1) = 1*1 = 1
A(1,2) = 1*2 = 2
A(2,2) = 2*2 = 4
and so on….

Output -

```
Enter the size of the array:
2
1
 A(1,1) =  1.00000000
 A(2,1) =  2.00000000
```

```
program alloc
implicit none

  real, dimension (:,:), allocatable :: A
  integer :: s1, s2
  integer :: i , j
  print *, "Enter the size of the array:"
  read (*,*)  s1, s2
  ! allocate memory
  allocate (A(s1,s2))
  do i = 1, s1
    do j = 1, s2
      A(i,j) = i * j
      print *, "A(" , i,",", j,") = ", A(i,j)
    end do
  end do
  ! deallocate memory
  deallocate (A)

end program alloc
```

Functions

- In Fortran, functions are subprograms. You can reuse them in other functions, programs or projects.
- They make the code simpler. Identifying and rectifying the bugs becomes easier.
- Ideally, a function should do only one thing and do it well.
- **intent**(in) : This statement tells the compiler that the variables are inputs to the function.
- Specify the type for the function output – (i) Real (ii) Logical
- The name of the function is treated as a variable (an output variable), and should be assigned a value (either directly, or by calculation) that is consistent with the output type of the function itself.

Declaring Real Functions

Create a Fortran source file with name sumOf

```
program sumOf
    implicit none
end program sumOf
```

⬇

Replace the word **program** by **function and specify the type of function in the beginning.**

```
real function sumOf
    implicit none
end function sumOf
```

- Declaring the newly defined functions in the main program: **real, external**
- Call the functions from the main program using function name, wherever appropriate.

Example 17: Write a program from adding any two real numbers with the help of functions

```
real function sumOf(a, b)
    implicit none

    real, intent(in) :: a, b

    sumOf = a + b
end function sumOf
```

```
program main
    implicit none
    real :: a, b, c
    real, external :: sumOf
    print *,"Enter a , b"
    read(*,*) a, b

    c = sumOf(a,b)

    10 format(a7, f5.2, a5, f5.2, a4, f7.3)
    write(*,10)"Sum of ", a," and ", b," is ",c

end program main
```

Output

```
Enter a , b
2
3
Sum of 2.00 and 3.00 is 5.000
```

- Before running the code, use **File** – **Save All** (instead of Save) and **Project** – **Build Project** (instead of Build), since we now have multiple source files in your project.

Logical Functions

- Logical functions return either True or false when their arguments are evaluated.

Declaring Logical Functions

```
logical function checkSigns
    implicit none
end function checkSigns
```

Declaring in other programs

```
logical, external
```

Example 18: Write a program to check if any two real numbers have same signs or not with the help of logical functions

```fortran
logical function checkSigns(a,b)
  implicit none

  real, intent(in) :: a, b

  checkSigns = .true.

  if ((a * b) < 0) then
    checkSigns = .false.
  end if

end function checkSigns
```

```fortran
program main
  implicit none
  real :: a, b
  logical, external :: checkSigns

  print *,"Enter a , b"
  read(*,*) a,b

  if (checkSigns(a,b) .eqv. .true.) then
    write(*,*) “The 2 numbers have same sign”
      else
    write(*,*) “The 2 numbers have different signs”
  end if

end program main
```

Output

```
Enter a,b
-3.4
9.1
The 2 numbers have different signs
```

Example 18: Check if two real numbers have same sign or not

Calling one function from another function

- We can call one function in another function (similar to calling in main program).

Example 19: Take any 2 real numbers (a,b) -

1) Check whether they are having same signs using functions (Ex. 18).

2) Write a new function for the following:

$$f(a,b) = \begin{cases} \cos(a) * e^b, & \textit{same signs} \\ \sin(a) * b^2, & \textit{unequal signs} \end{cases}$$

3) Main program: Evaluate the function value and print the obtained value.

Step 1

```
logical function checkSigns(a,b)
   implicit none

   real, intent(in) :: a,b

   checkSigns = .true.

   if ((a * b) < 0) then

     checkSigns = .false.

   end if

end function checkSigns
```

Step 2

```
real function getFun(a,b)
   implicit none
   real, intent(in) :: a,b
   logical, external :: checkSigns
   real :: pi
   pi = 3.14
   if (checkSigns(a,b) .eqv. .false.) then
      getFun = sin(a*(pi/180))*b**2
   else
      getFun = cos(a*(pi/180))*exp(b)
   end if

end function getFun
```

Step 3

```
program main
   implicit none
   real :: a1, a2,funVal
   real, external :: getFun

   print*,"Enter a1 , a2"
   read(*,*) a1,a2

   funVal = getFun(a1,a2)

   print *,"The function value is ", funVal

end program main
```

1 degree = pi/180 radians

Output :

```
Enter a1 , a2
45
-1
 The function value is   0.706825197
```

Multiple functions in a program

- A program and function can have more than one function.

Example 20: Repeat Ex. 19 and include another function in main program which gives sum of 2 numbers.

```
real function sumOf(a, b)
    implicit none

    real, intent(in) :: a,b

    sumOf = a + b
end function sumOf
```

```
program main
    implicit none
    real :: a1, a2,funVal,addVal
    real, external :: sumOf
    real, external :: getFun

    print*,"Enter a1 , a2"
    read(*,*)a1,a2

    addVal = sumOf(a1,a2)
    funVal = getFun(a1,a2)

    print*," Sum of two given numbers is", addVal
    print*,"The function value is ",funVal

end program main
```

Subroutines

- Subprograms in Fortran.
- They can perform multiple operations at the same time unlike functions (which perform only a single operation).
- The return arguments has to be specified by **intent(out)**
- **intent(out)** : This statement tells the compiler that the variables are output to the subroutine.
- **intent(inout)** : This statement tells the compiler that the variables are input as well as output to the subroutine.
- A subroutine can be invoked only by a special calling statement.

Declaration of Subroutines

- Create a Fortran source file with name statistics **(in a new Fortran project).**

```
program statistics
   implicit none
end program statistics
```

↓

- Replace the word **program** with **subroutine**.

```
subroutine statistics
   implicit none
end subroutine statistics
```

- Call the subroutines from the main program using the following syntax:

call nameOfTheSubroutine (Ex. call statistics)

Example 21: Create a subroutine to perform both summation and multiplication of any two user-defined real numbers and print the outputs.

Step 1:

```
subroutine statistics(num1,num2,sumOF,ProdOf)
   implicit none
   real, intent(in) :: num1, num2
   real, intent(out) :: sumOf, ProdOf

   sumOF = num1 + num2
   ProdOf = num1 * num2

end subroutine statistics
```

Example 21…

Step 2:

```
program main
    implicit none
    real :: num1, num2,sumOf,ProdOf
    print*,"Enter num1 , num2"
    read(*,*)num1,num2

    call statistics(num1, num2, sumOf, ProdOf)
```

Name of the subroutine

Variables declared as intent(in) and intent(out)

```
    print*,"The sum of the two given numbers is ",sumOf
    print*,"The product the two given numbers is ",ProdOf
end program main
```

Output

Enter number1 , number2
47
98
The sum of the two given numbers is 145.000000
The product the two given numbers is 4606.00000

Calling one subroutine from another subroutine

Example 22: Create subroutines to perform the following operations:

(i) Summation and multiplication of any two user-defined real numbers.

(ii) Calculate average of the given 2 numbers.

Then, print the sum, product and average of the numbers.

Step 1: (same as Ex. 21)

```
subroutine statistics(a,b,sumOF,ProdOf)
   implicit none
   real, intent(in) :: a, b
   real, intent(out) :: sumOf, ProdOf

   sumOF = num1 + num2
   ProdOf = num1 * num2

end subroutine statistics
```

Step 2:

```
subroutine average(a,b,sumOf,ProdOf,avg)
   implicit none
   real, intent(in) :: a,b, sumOf, ProdOf
   real, intent(out) :: avg

   call statistics(a,b,sumOf,ProdOf)
   avg = sumOf/2;
end subroutine average
```

Example 22…

Step 3: Calling multiple subroutines in a program

```
program main
   implicit none
   real :: a, b,sumOf,ProdOf
   real :: avg
   print*,"Enter a , b"
   read(*,*) a,b

   call statistics(a,b,sumOf,ProdOf)
   call average(a,b,sumOf,ProdOf,avg)

   print*,"The sum of the two given numbers is ",sumOf
   print*,"The product the two given numbers is ",ProdOf
   print*,"The average the two given numbers is ",avg
end program main
```

Output

```
Enter a , b
4
6
The sum of the two given
numbers is 10.0000000
The product the two given
numbers is 24.0000000
The average the two given
numbers is 5.00000000
```

Example 23: Create subroutine for swapping of any two real numbers and then write a program for printing the swapped numbers.

```
subroutine swapNumbers(num1, num2)
implicit none
real, intent(inout) :: num1, num2
real :: swapper
swapper = num1
num1 = num2
num2 = swapper
end subroutine swapNumbers
```

```
program main
   implicit none
   real :: a, b
   print*,"Enter num1 , num2"
   read(*,*)a,b
   print*,“Before swapping",a,b

   call swapNumbers(a,b)

   print*,"After swapping",a,b

end program main
```

Output

```
Before swapping  -2.00000000     -5.00000000
After swapping   -5.00000000     -2.00000000
```

Points to remember

- Except for subprograms, create a new Fortran project for every new program.
- Add the extension .f95 for every new Fortran source file.
- We can call any number of functions and(or) subroutines in one program.
- Functions need to be declared in the main program unlike subroutines.
- While calling subprogram(s) in the main program, the order of variables declared in both the programs should be equal.

subroutine

```
subroutine statistics(num1,num2,sum,Prod)
implicit none
end subroutine statistics
```

Main program

```
call statistics(a,b,sum,Prod)
call statistics(sum,Prod,a,b)
```

Few more…

1) Take a two dimensional array, say $A = \begin{bmatrix} 2 & 9 & 0 \\ 4 & 3 & -2 \end{bmatrix}$, as input and find the following:
   (i) size and shape of A
   (ii) maximum value of A
   (iii) minimum value of A
   (iv) location of values obtained in (i) and (ii)
   (v) sum and product of elements in A.

2) Take any two matrices (2-D arrays) as input from the user and perform matrix multiplication.

3) Write a function/subprogram for converting any user defined angle (in degrees) (say, x) to radians. Using this function, calculate the value of tan(x).

4) Check whether any given number is an integer or real number using logical functions.

5) Write a function for computing the sum of elements in a two dimensional user-defined array.

6) Find the roots of a quadratic equation: $ax^2 + bx + c = 0$ using the formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ (a,b,c – user defined). Print both real and imaginary parts of the root.

7) Write a subprogram for sorting any ‘n’ (user-defined) real numbers and print them in ascending order.

8) Repeat the same exercise for sorting elements of a 2-D array $A = \begin{bmatrix} 4 & 2 & -5 \\ 10 & 7 & 2 \end{bmatrix}$.

(**Note** – Use allocatable arrays wherever the size of the array is not known at the time of declaration).