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

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.

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 – **datatype-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 (!), and 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 2: 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 3: Rewrite ex. 2, 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
```

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.

Fortran - Decisions

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

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

Example 4: 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
```

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
```

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
```

Example 5:

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 5: 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

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 - 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 6: 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 7: Repeat Ex. 6 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) 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

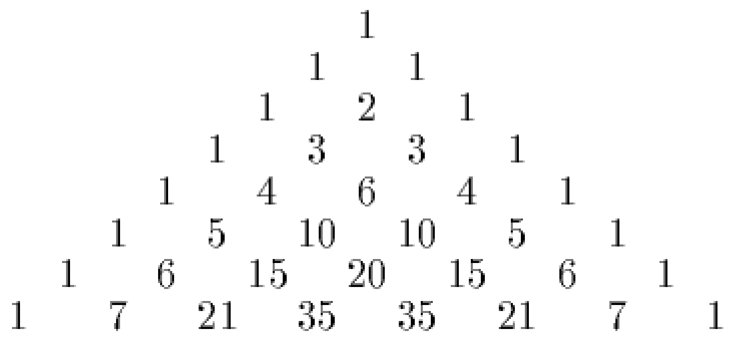
FEW MORE…

6) Write a program for printing the Fibonacci series.

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

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

9) 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
```