LEARN FORTRAN

formula translating system

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Fortran was originally developed by a team at IBM in 1957 for scientific calculations. Later developments made it into a high level programming language. In this tutorial, we will learn the basic concepts of Fortran and its programming code.

Audience

This tutorial is designed for the readers who wish to learn the basics of Fortran.

Prerequisites

This tutorial is designed for beginners. A general awareness of computer programming languages is the only prerequisite to make the most of this tutorial.

Copyright & Disclaimer

Table of Contents

1. FORTRAN – OVERVIEW

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

Fortran was originally developed by IBM in the 1950s for scientific and engineering applications. Fortran ruled this programming area for a long time and became very popular for high performance computing, because.

It supports:

- Numerical analysis and scientific computation
- Structured programming
- Array programming
- Modular programming
- Generic programming
- High performance computing on supercomputers
- Object oriented programming
- Concurrent programming
- Reasonable degree of portability between computer systems

Facts about Fortran

- Fortran was created by a team, led by John Backus at IBM in 1957.
- Initially the name used to be written in all capital, but current standards and implementations only require the first letter to be capital.
- Fortran stands for FORmula TRANslator.
- Originally developed for scientific calculations, it had very limited support for character strings and other structures needed for general purpose programming.
- Later extensions and developments made it into a high level programming language with good degree of portability.
- Original versions, Fortran I, II and III are considered obsolete now.
- Oldest version still in use is Fortran IV, and Fortran 66.
- Most commonly used versions today are : Fortran 77, Fortran 90, and Fortran 95.

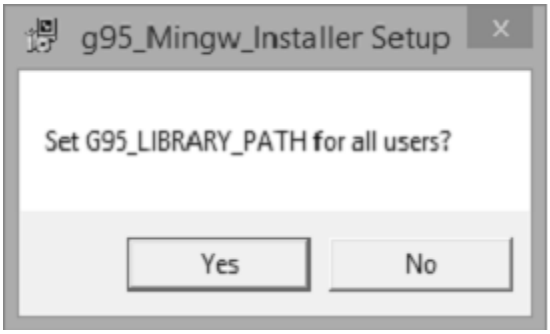
- Fortran 77 added strings as a distinct type.
- Fortran 90 added various sorts of threading, and direct array processing.

2. FORTRAN – ENVIRONMENT SETUP

Setting up Fortran in Windows

G95 is the GNU Fortran multi-architechtural compiler, used for setting up Fortran in Windows. The windows version emulates a unix environment using MingW under windows. The installer takes care of this and automatically adds g95 to the windows PATH variable.

You can get the stable version of G95 from here :



How to Use G95

During installation, **g95** is automatically added to your PATH variable if you select the option "RECOMMENDED". This means that you can simply open a new Command Prompt window and type "g95" to bring up the compiler. Find some basic commands below to get you started.

Command	Description
g95 –c hello.f90	Compiles hello.f90 to an object file named hello.o
g95 hello.f90	Compiles hello.f90 and links it to produce an executable a.out
g95 -c h1.f90 h2.f90 h3.f90	Compiles multiple source files. If all goes well, object files h1.o, h2.o and h3.o are created
g95 -o hello h1.f90 h2.f90 h3.f90	Compiles multiple source files and links them together to an executable file named 'hello'

Command line options for G95:

```
-c Compile only, do not run the linker.
-o Specify the name of the output file, either an object file or the executable.
```

Multiple source and object files can be specified at once. Fortran files are indicated by names ending in ".f", ".F", ".for", ".FOR", ".f90", ".F90", ".f95", ".F95", ".f03" and ".F03". Multiple source files can be specified. Object files can be specified as well and will be linked to form an executable file.

3. FORTRAN – BASIC SYNTAX

A Fortran program is made of a collection of program units like a main program, modules, and external subprograms or procedures.

Each program contains one main program and may or may not contain other program units. The syntax of the main program is as follows:

```
program program_name
implicit none

! type declaration statements
! executable statements

end program program_name
```

A Simple Program in Fortran

Let's write a program that adds two numbers and prints the result:

```
program addNumbers

! This simple program adds two numbers
   implicit none

! Type declarations
   real :: a, b, result

! Executable statements
   a = 12.0
   b = 15.0
   result = a + b
   print *, 'The total is ', result
```

```
end program addNumbers
```

When you compile and execute the above program, it produces the following result:

```
The total is 27.0000000
```

Please note that:

- All Fortran programs start with the keyword **program** and end with the keyword**end program,** followed by the name of the program.
- The **implicit none** statement allows the compiler to check that all your variable types are declared properly. You must always use **implicit none** at the start of every program.
- Comments in Fortran are started with the exclamation mark (!), as all characters after this (except in a character string) are ignored by the compiler.
- The **print *** command displays data on the screen.
- Indentation of code lines is a good practice for keeping a program readable.
- Fortran allows both uppercase and lowercase letters. Fortran is case-insensitive, except for string literals.

Basics

The **basic character set** of Fortran contains:

- the letters A ... Z and a ... z
- the digits 0 ... 9
- the underscore (_) character
- the special characters = : + blank - * / ( ) [ ] , . $ ' ! " % & ; < > ?

Tokens are made of characters in the basic character set. A token could be a keyword, an identifier, a constant, a string literal, or a symbol.

Program statements are made of tokens.

Identifier

An identifier is a name used to identify a variable, procedure, or any other user-defined item. 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

Keywords

Keywords are special words, reserved for the language. These reserved words cannot be used as identifiers or names.

The following table, lists the Fortran keywords:

Non-I/O keywords				
allocatable	allocate	assign	assignment	block data
call	case	character	common	complex
contains	continue	cycle	data	deallocate
default	do	double precision	else	else if
elsewhere	end block data	end do	end function	end if
end interface	end module	end program	end select	end subroutine
end type	end where	entry	equivalence	exit
external	function	go to	if	implicit
in	inout	integer	intent	interface
intrinsic	kind	len	logical	module
namelist	nullify	only	operator	optional
out	parameter	pause	pointer	private

program	public	real	recursive	result
return	save	select case	stop	subroutine
target	then	type	type()	use
Where	While			
I/O related keywords				
backspace	close	endfile	format	inquire
pen	print	read	rewind	Write

4. FORTRAN – DATA TYPES

Fortran provides five intrinsic data types, however, you can derive your own data types as well. The five intrinsic types are:

- Integer type
- Real type
- Complex type
- Logical type
- Character type

Integer Type

The integer types can hold only integer values. The following example extracts the largest value that can be held in a usual four byte integer:

```
program testingInt
implicit none

   integer :: largeval
   print *, huge(largeval)

end program testingInt
```

When you compile and execute the above program it produces the following result:

```
2147483647
```

Note that the **huge()** function gives the largest number that can be held by the specific integer data type. You can also specify the number of bytes using the **kind** specifier. The following example demonstrates this:

```
program testingInt
implicit none

   !two byte integer
```

```
   integer(kind=2) :: shortval

   !four byte integer
   integer(kind=4) :: longval

   !eight byte integer
   integer(kind=8) :: verylongval

   !sixteen byte integer
   integer(kind=16) :: veryverylongval

   !default integer
   integer :: defval

   print *, huge(shortval)
   print *, huge(longval)
   print *, huge(verylongval)
   print *, huge(veryverylongval)
   print *, huge(defval)

end program testingInt
```

When you compile and execute the above program, it produces the following result:

```
32767
2147483647
9223372036854775807
170141183460469231731687303715884105727
2147483647
```

Real Type

It stores the floating point numbers, such as 2.0, 3.1415, -100.876, etc.

Traditionally there are two different real types, the default **real** type and **double precision**type.

However, Fortran 90/95 provides more control over the precision of real and integer data types through the**kind**specifier, which we will study in the chapter on Numbers.

The following example shows the use of real data type:

```
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
```

When you compile and execute the above program it produces the following result:

```
0.666666687
0
```

Complex Type

This is used for storing complex numbers. A complex number has two parts, the real part and the imaginary part. Two consecutive numeric storage units store these two parts.

For example, the complex number (3.0, -5.0) is equal to 3.0 – 5.0i

We will discuss Complex types in more detail, in the Numbers chapter.

Logical Type

There are only two logical values: **.true.** and **.false.**

Character Type

The character type stores characters and strings. The length of the string can be specified by len specifier. If no length is specified, it is 1.

For example,

```
character (len=40) :: name
name = "Zara Ali"
```

The expression, **name(1:4)** would give the substring "Zara".

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.

5. 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.

Based on the basic types explained in previous chapter, following are the variable types:

Type	Description
Integer	It can hold only integer values.
Real	It stores the floating point numbers.
Complex	It is used for storing complex numbers.
Logical	It stores logical Boolean values.
Character	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:

```
type-specifier :: variable_name
```

For example,

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

Later you can assign values to these variables, like,

```
total = 20000
average = 1666.67
done = .true.
message = "A big Hello from Tutorials Point"
cx = (3.0, 5.0) ! cx = 3.0 + 5.0i
```

You can also use the intrinsic function **cmplx,** to assign values to a complex variable:

```
cx = cmplx (1.0/2.0, -7.0) ! cx = 0.5 - 7.0i
cx = cmplx (x, y) ! cx = x + yi
```

Example

The following example demonstrates variable declaration, assignment and display on screen:

```
program variableTesting
implicit none

   ! declaring variables
   integer :: total
   real :: average
   complex :: cx
   logical :: done
```

```
   character(len=80) :: message ! a string of 80 characters

   !assigning values
   total = 20000
   average = 1666.67
   done = .true.
   message = "A big Hello from Tutorials Point"
   cx = (3.0, 5.0) ! cx = 3.0 + 5.0i

   Print *, total
   Print *, average
   Print *, cx
   Print *, done
   Print *, message

end program variableTesting
```

When the above code is compiled and executed, it produces the following result:

```
20000
1666.67004
(3.00000000, 5.00000000 )
T
A big Hello from Tutorials Point
```

6. FORTRAN – CONSTANTS

The constants refer to the fixed values that the program cannot alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, a complex constant, or a string literal. There are only two logical constants : **.true.** and **.false.**

The constants are treated just like regular variables, except that their values cannot be modified after their definition.

Named Constants and Literals

There are two types of constants:

- Literal constants
- Named constants

A literal constant have a value, but no name.

For example, following are the literal constants:

Type	Example
Integer constants	0 1 -1 300 123456789
Real constants	0.0 1.0 -1.0 123.456 7.1E+10 -52.715E-30
Complex constants	(0.0, 0.0) (-123.456E+30, 987.654E-29)
Logical constants	.true. .false.
Character constants	"PQR" "a" "123'abc$%#@!" " a quote "" " 'PQR' 'a' '123"abc$%#@!' ' an apostrophe '' '

A named constant has a value as well as a name.

Named constants should be declared at the beginning of a program or procedure, just like a variable type declaration, indicating its name and type. Named constants are declared with the parameter attribute. For example,

```
real, parameter :: pi = 3.1415927
```

Example

The following program calculates the displacement due to vertical motion under gravity.

```
program gravitationalDisp

! this program calculates vertical motion under gravity
implicit none

   ! gravitational acceleration
   real, parameter :: g = 9.81

   ! variable declaration
   real :: s ! displacement
   real :: t ! time
   real :: u ! initial speed

   ! assigning values
   t = 5.0
   u = 50

   ! displacement
   s = u * t - g * (t**2) / 2

   ! output
   print *, "Time = ", t
   print *, 'Displacement = ',s

end program gravitationalDisp
```

When the above code is compiled and executed, it produces the following result:

```
Time = 5.00000000
Displacement = 127.374992
```

7. FORTRAN – OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Fortran provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators

Let us look at all these types of operators one by one.

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

Try the following example to understand all the arithmetic operators available in Fortran:

```
program arithmeticOp
```

```
! this program performs arithmetic calculation
implicit none

   ! variable declaration
   integer :: a, b, c

   ! assigning values
   a = 5
   b = 3

   ! Exponentiation
   c = a ** b

   ! output
   print *, "c = ", c

   ! Multiplication
   c = a * b

   ! output
   print *, "c = ", c

   ! Division
   c = a / b

   ! output
   print *, "c = ", c

   ! Addition
   c = a + b

   ! output
   print *, "c = ", c
```

```
   ! Subtraction
   c = a - b

   ! output
   print *, "c = ", c

end program arithmeticOp
```

When you compile and execute the above program, it produces the following result:

```
c = 125
c = 15
c = 1
c = 8
c = 2
```

Relational Operators

Following table shows all the relational operators supported by Fortran. Assume variable **A**holds 10 and variable **B** holds 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.