In Windows OS

<https://gcc.gnu.org/wiki/GFortranBinaries#Windows>

Commands to start:

<https://gcc.gnu.org/wiki/GFortranGettingStarted>

General:

<https://gcc.gnu.org/wiki/GFortran>

[https://web.stanford.edu/class/me200c/tutorial\_77/](https://web.stanford.edu/class/me200c/tutorial_77/06_expressions.html)

Every variable *should* be defined in a *declaration*. This establishes the *type* of the variable. The most common declarations are:

integer *list of variables*

real *list of variables*

double precision *list of variables*

complex *list of variables*

logical *list of variables*

character *list of variables*

* The list of variables should consist of variable names separated by commas.
* Integers are usually stored as 32 bits (4 bytes) variables
* Fortran 77 has two different types for floating point variables, called real and double precision. While real is often adequate, some numerical calculations need very high precision and double precision should be used.
* Usually a real is a 4 byte variable and the double precision is 8 bytes, but this is machine dependent.

### Constants

* Some constants appear many times in a program. It is then often desirable to define them only once, in the beginning of the program. This is what the parameter statement is for.
* The syntax of the parameter statement is
* parameter (name = constant, ... , name = constant)

The fifth type is logical constants. These can only have one of two values:

.TRUE.

.FALSE.

Note that the dots enclosing the letters are required.

The last type is character constants. These are most often used as an array of characters, called a string. These consist of an arbitrary sequence of characters enclosed in apostrophes (single quotes):

'ABC'

'Anything goes!'

'It is a nice day'

Strings and character constants are case sensitive. A problem arises if you want to have an apostrophe in the string itself. In this case, you should double the apostrophe (its alls refered as escape character as in json, javascript etc…):

'It''s a nice day'

### Expressions

This raises the question of precedence: Does the last expression mean x + (2\*y) or (x+2)\*y? The precedence of arithmetic operators in Fortran 77 are (from highest to lowest):

\*\* {exponentiation}

\*,/ {multiplication, division}

+,- {addition, subtraction}

All these operators are calculated left-to-right, except the exponentiation operator \*\*, which has right-to-left precedence. If you want to change the default evaluation order, you can use parentheses.

The above operators are all binary operators. there is also the unary operator - for negation, which takes precedence over the others. Hence an expression like -x+y means what you would expect.

Extreme caution must be taken when using the division operator, which has a quite different meaning for integers and reals. If the operands are both integers, an integer division is performed, otherwise a real arithmetic division is performed. For example, 3/2 equals 1, while 3./2. equals 1.5 (note the decimal points).

**Note: returns only the integer value as espected since it a division between two integers**

### Assignment

The assignment has the form

variable\_name = expression

**Type conversion**

it is good programming practice to force the necessary type conversions explicitly

The first three have the obvious meaning. ichar takes a character and converts it to an integer, while char does exactly the opposite.

Example: How to multiply two real variables x and y using double precision and store the result in the double precision variable w:

w = dble(x)\*dble(y)

Note that this is different from

w = dble(x\*y)

**Logical expressions**

Logical expressions can only have the value .TRUE. or .FALSE.. A logical expression can be formed by comparing arithmetic expressions using the following relational operators:

.LT. meaning <.LE. <=".GT.">

.GE. >=

.EQ. =

.NE. /=

So you cannot use symbols like

Logical expressions can be combined by the logical operators .AND. .OR. .NOT. which have the obvious meaning.

### Logical variables and assignment

Truth values can be stored in logical variables. The assignment is analogous to the arithmetic assignment. Example:

logical a, b

a = .TRUE.

b = a .AND. 3 .LT. 5/2

The order of precedence is important, as the last example shows. The rule is that arithmetic expressions are evaluated first, then relational operators, and finally logical operators. Hence b will be assigned .FALSE. in the example above. Among the logical operators the precedence (in the absence of parenthesis) is that .NOT. is done first, then .AND., then .OR. is done last.

### The if statement

if (logical expression) executable statement

This has to be written on one line. This example finds the absolute value of x:

if (x .LT. 0) x = -x

If more than one statement should be executed inside the if, then the following syntax should be used:

if (logical expression) then

statements

endif

The most general form of the if statement has the following form:

if (logical expression) then

statements

elseif (logical expression) then

statements

:

:

else

statements

endif

### Loops

For repeated execution of similar things, loops are used. If you are familiar with other programming languages you have probably heard about for-loops, while-loops, and until-loops. Fortran 77 has only one loop construct, called the do-loop. The do-loop corresponds to what is known as a for-loop in other languages. Other loop constructs have to be built using the if and goto statements.

### do-loops

The do-loop is used for simple counting. **Here is a simple example that prints the cumulative sums of the integers from 1 through n** (assume n has been assigned a value elsewhere) ‘so it start at 1 and ends at the n-th cycle’:

integer i, n, sum

sum = 0

do 10 i = 1, n

sum = sum + i

write(\*,\*) 'i =', i

write(\*,\*) 'sum =', sum

10 continue

The number 10 is a statement label. Typically, there will be many loops and other statements in a single program that require a statement label. The programmer is responsible for assigning a unique number to each label in each program (or subprogram). Recall that column positions 1-5 are reserved for statement labels. The numerical value of statement labels have no significance, so any integers can be used, in any order. Typically, most programmers use consecutive multiples of 10.

The general form of the do loop is as follows:

do label var = expr1, expr2, expr3

statements

label continue

var is the loop variable (often called the loop index) which must be integer. expr1 specifies the initial value of var, expr2 is the terminating bound, and expr3 is the increment (step).

**CODE**

!

!

! Online Fortran Compiler.

! Code, Compile, Run and Debug Fortran program online.

! Write your code in this editor and press "Run" button to execute it.

!

!

!this is fortran 95 the c and \* for comment doesn't work as also the next line (+) also doesn't work it needs to be & thats fortran95

program circle

real r, area

parameter (pi = 3.14159265358979)

logical a, b

integer i, n, sumValue, ii, j, k

a = .TRUE.

b = a .AND. 3 .LT. 8/2

! This program reads a real number r and prints

! the area of a circle with radius r.

!write (\*,\*) 'Give radius r:'

!read (\*,\*) r

! The next statement goes over two physical lines

!area = pi &

!\* r \* r

!write (\*,\*) 'Area = ', area

write (\*,\*) 'so it fixed value multiplication'

write (\*,\*) 'pi^2 = ', pi\*pi

write (\*,\*) '2\*10\*10 = ', 2.0E2 !E equal to 10 elevate to number followed by E as in math nomenclature

write (\*,\*) '8/6 = ', 10/6 !returns the result as integer

write (\*,\*) '8./6. = ', 8./6. !returns the result as float

write (\*,\*) 'b is ', b !result of the condition

do 20 ii = 10, 1, -2

write(\*,\*) 'ii =', ii

20 continue

write (\*,\*) 'give a value for variable j:'

read (\*,\*) j

do 40 i = 1, j

write (\*,\*) j

j = j + 1

40 continue

write (\*,\*) j

!a while loop

write (\*,\*) 'while loop', NEW\_LINE('a'), NEW\_LINE('a')

k = 1

60 if (k .le. 10) then

k = k + 1;

write (\*,\*) k

goto 60

endif

stop

end