Conditionals and Loops

Carlos Cruz
Jules Kouatchou
Bruce Van Aartsen

NASA GSFC Code 606 (ASTG)
Greenbelt Maryland 20771

October24, 2018

Basic IF-statement

The basic 1-line **IF** statement is:

IF (*logical-expression*) *exec-statement*

Where the *logical-expression* must evaluate to **.TRUE.** for the statement to execute.

For example:

```
IF ( tempA < tempB ) maxTemp = tempB
```

IF-statement Operators

A Fortran logical-expression can use 6 relational operators:

Description	F90 symbol	Pre-F90 symbol
Less than	<	.LT.
Less than or equal to	<=	.LE.
Greater than	>	.GT.
Greater than or equal to	>=	.GE.
Equal to	==	.EQ.
Not equal to	/=	.NE.

- Note that “equal to” operator uses “==” to distinguish it from the single “=” used by assignment statements

IF-statement Operators

A logical-expression can also use 5 logical operators :

Description	F90 symbol
Reverse the logic of operands	.NOT.
True if either operand is true	.OR.
True if both operands are true	.AND.
True if both operands are the same	.EQV.
True if operands are NOT the same	.NEQV.

For example, if a, b, c, d are set to 2, 3, 4, 5, respectively:

```
(a < b .AND. c <= d) .NEQV. (b == d)
(      .TRUE.      ) .NEQV. (.FALSE.)
                     .TRUE.
```

IF-block Construct

The IF-block is a more flexible version of the single-line IF:

```
IF ( logical-expression ) THEN
   then-block
ELSE IF ( logical-expression ) THEN
   elseif-block
ELSE
   else-block
END IF
```

*Note: All code inside if-blocks or loops should always be indented for readability, and clarity of flow.

IF-block Construct

For example:

```
IF (avgWind >= 74) THEN
   stormType = "Hurricane"
ELSE IF (avgWind >= 39) then
   stormType = "Tropical Storm"
ELSE
   stormType = "Depression"
END IF
```

CASE Construct

The SELECT CASE construct is an alternative to using IF statements. Code branching depends on the value of a single expression:

```
[name:] SELECT CASE ( expression )
CASE ( value1 )
   block1
CASE ( value2 )'
   block2
<etc…>
CASE DEFAULT
   default block
END SELECT [name]
```

- The ***expression*** can be integer, character, or logical.
- the ***value*** argument can be a range, minimum, maximum, a list of values, or a combination, such as:

  (min:max), (min:), (:max), (2,3,6,7), (11,14,18:20,24)

CASE Construct

Example:

```
Hurricane: SELECT CASE (sustainedWindMPH)
CASE ( 75:95 )
   Category = 1
CASE ( 96:110 )
   Category = 2
CASE ( 111:129 )
   Category = 3
CASE ( 130:156 )
   Category = 4
CASE ( 157: )
   Category = 5
CASE DEFAULT
   Category = 0
END SELECT Hurricane
```

DO Loop

DO loops contain statements that you want to repeat a number of times. The most common form is the **Indexed DO Loop**

```
[name:] DO index = start, stop, [step]
   statement1
   statement2
   etc.
END DO [name]
```

For example:

```
sumPrecip: DO i = 1, 48
   totalPrecip = totalPrecip + precip(i)
END DO sumPrecip
```

Legacy DO Loops

Please note that older Fortran 77 code used numbered CONTINUE statements to end DO loops, instead of END DO:

```
      DO 100 index = start, stop, [step]
         statement1
         statement2
         etc.
100   CONTINUE
```

In fact, the CONTINUE statement was optional, and loops could end on any statement with the given numeric label. This is bad practice and should be fixed whenever practical.

DO WHILE Loop

This loop continues executing as long as the ***logical expression*** at the top remains .TRUE., so the values in the ***logical expression*** must be altered, making the condition .FALSE. to exit the loop.

```
[name:] DO WHILE (logical expression is .TRUE. )
    statement1
    statement2
    etc.
END DO [name]
```

For example:

```
a = 0
DO WHILE (a < 10)
    ...
    a = a +1
END DO
! Jumps to here when a = 10
```

Conditional EXIT Loop

Sometimes it is useful to jump out of a DO loop in the middle of a code block. This is done with a Conditional EXIT command.

```
DO
   statement1
   ...
   IF (a > b) EXIT
   ...
END DO
! Exit command jumps to here
```

If the DO loop is nested inside a outer DO loop, control jumps to the end of the inner loop, and continues inside the outer loop.

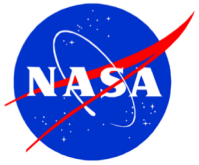
*Note: Legacy code sometimes used GOTO statements to jump out of loops, or control flow. We strongly advise against the use of ANY GOTO statements. It makes the code hard to follow and maintain!

Conditional CYCLE Loop

A CYCLE command also lets you jump out of the middle of a DO loop, but CYCLE merely ends the current pass, and begins the next iteration.

```
Foo: DO           ! 2nd cycle command jumps to here
   Bar: DO        ! 1st cycle command jumps to here
      i = i + 1
      ...
      IF (a(i) > b) CYCLE
      ...
      IF (a(i) == c) CYCLE Foo
      ...
   END DO Bar
   IF (c > d) EXIT
   ...
END DO Foo
! Exit command jumps to here
```

If a CYCLE statement is not followed by a construct name then the directive applies to the innermost loop.



Example

Given an array of **`dailyPrecip(365)`** and **`avgYearlyPrecip`**, find how many days it took to equal the normal yearly average

```
dayOfYear = 0
totalPrecip = 0.0
DO WHILE (totalPrecip < avgYearlyPrecip)
   dayOfYear = dayOfYear +1
   IF (dayOfYear > 365) THEN
      print *, “Precip was below normal this year”
      STOP
   END IF
   totalPrecip = totalPrecip + precip(dayOfYear)
END DO
print *, ”Precip surpassed yearly average on day”, dayOfYear
```

Exercise

Convert the following code to use DO with a Conditional EXIT, instead of DO WHILE

```
dayOfYear = 0
totalPrecip = 0.0
DO WHILE (totalPrecip < avgYearlyPrecip)
   dayOfYear = dayOfYear +1
   IF (dayOfYear > 365) THEN
      print *, “Precip was below normal this year”
      STOP
   END IF
   totalPrecip = totalPrecip + precip(dayOfYear)
END DO
print *, ”Precip surpassed yearly average on day”, dayOfYear
```