

Control Constructs

Control Constructs

These will change the **sequential execution** order
Will cover the main **constructs** in some detail
We will cover **procedure call** later

The main ones are:

Conditionals (**IF** etc.)

Loop (**DO** etc.)

Switches (**SELECT/CASE** etc.)

Loops are by far the most complicated.

Single Statement IF (1)

The oldest and the simplest is the single statement **IF**

IF (logical expression) simple statement

If the logical expression is **.True.** then the simple statement is executed.

If the logical expression is **.False.** then the whole statement has no effect.

Single Statement IF (2)

Some examples:

```
IF (X < A) X = A
```

```
IF (INT(a*b-c) <= 47) mytest = .true.
```

```
IF (MOD(Cnt, 10) == 0) WRITE(*,*) CNT
```

Unsuitable for anything complicated.

Only **action statements** (assignment, input/output) can be used. Nothing complicated like another **IF** statement or anything containing **blocks**.

Block IF Statement

A block IF statement is much more flexible

Here is the most traditional form of it

```
IF (logical expression) THEN
    then block of statements
ELSE
    else block of statements
ENDIF
```

If the expr is `.TRUE.` then the first block is executed
If not, the second block is executed.

`ENDIF` or `END IF` can be used.

Example

```
LOGICAL :: flip
```

```
IF (flip .AND. X /= 0.0) THEN
```

```
    PRINT *, 'Using the inverted form'
```

```
    X = 1.0/A
```

```
    Y = EXP(-A)
```

```
ELSE
```

```
    X = A
```

```
    Y = EXP(-A)
```

```
ENDIF
```

Omitting the ELSE

The **ELSE** and its block can also be omitted.

```
IF (X > Maximum) THEN  
    X = Maximum  
ENDIF
```

```
IF (name(1:4) == "Miss" .OR. &  
    name(1:4) == "Mrs.") THEN  
    name(1:3) = "Ms."  
    name(4:) = name(5:)  
ENDIF
```

Including ELSE IF Blocks (1)

ELSE IF functions much like ELSE and IF

IF ($X < 0.0$) THEN ! This is tried first

$X = A$

ELSE IF ($X < 2.0$) THEN ! This second

$X = A + (B-A)*(X-1.0)$

ELSE IF ($X < 3.0$) THEN ! This third

$X = B + (C-B)*(X-2.0)$

ELSE ! This is used if none succeed

$X = C$

ENDIF

Including ELSE IF Blocks (2)

- You can have as many **ELSE IFs** as you wish
- There is only one **ENDIF** for the whole block
- All **ELSE IFs** must come before any **ELSE**
- They are checked in order and the first **success** is taken
- You can omit the **ELSE** in these constructs
- **ELSE IF** can also be spelled **ELSEIF**

Named IF Statements (1)

The **IF** can be preceded by <name>:

And the **END IF** followed by <name>:

And any **ELSE IF / THEN** and **ELSE** may be

```
myifblock: IF (X < 0.0) THEN
```

```
    X = A
```

```
ELSE IF (X < 2.0) THEN myifblock
```

```
    X = A + (B-A)*(X-1.0)
```

```
ELSE myifblock
```

```
    X = C
```

```
ENDIF myifblock
```

Named IF Statements (2)

The **IF construct name** must match and be distinct
Can be a great help for checking and clarity
You should name at least all long **IFs**

If you don't nest **IFs** that much this style is fine:

```
myifblock: IF (X < 0.0) THEN
    X = A
ELSE IF (X < 2.0) THEN
    X = A + (B-A)*(X-1.0)
ELSE
    X = C
ENDIF myifblock
```

Block Contents

- Almost any **executable statements** are okay
Both kinds of **IF**, complete **loops**, etc.
You may never notice the few restrictions
- This applies to all of the **block statements**
IF, DO, SELECT, etc.
- Avoid deep levels and very long blocks
Purely because they will **confuse** human readers

Example

```
phasetest: IF (state == 1) THEN
    IF (phase < pi_by_2) THEN
        ...
    ELSE
        ...
    ENDIF
ELSE IF (state == 2) THEN phasetest
    IF (phase > pi) PRINT *, 'A bit odd here'
ELSE phasetest
    IF (phase < pi) THEN
        ...
    ENDIF
ENDIF
ENDIF
```

SELECT CASE (1)

An alternative to the **IF** block for selective execution is the **SELECT CASE** statement. Can be used if the selection criteria are based on simple values in **INTEGER**, **LOGICAL** and **CHARACTER**.

It provides a streamlined syntax for an important special case of a **multiway selection**.

SELECT CASE (2)

The **basic format** is:

```
SELECT CASE ( <selector> )
    CASE (label-list-1)
        statements-1
    CASE (label-list-2)
        statements-2
    CASE (label-list-n)
        statements-n
    CASE DEFAULT
        statements-default
END SELECT
```

SELECT CASE (3)

The label-list can take one of many forms:

- `val` → a specific value
- `val1, val2, val3` → a specific set of values
- `val1 : val2` → values between `val1` and `val2` inclusive
- `val1 :` → values larger than or equal to `val1`
- `: val2` → values less than or equal to `val2`

`val`, `val1` and `val2` must be **constants** or **parameters**!

Example: `select_example1.f90` / `select_example2.f90`

SELECT CASE (4)

Some important notes:

- The values in the **label-lists** should be unique. Otherwise you will get a compilation error.
- **CASE DEFAULT** should be used if possible as it guarantees that a match will be found even if it is an error condition.
- Technically the **CASE DEFAULT** can be placed anywhere within the **SELECT CASE** statement but the preferred position is at the bottom.

DO Construct

The **loop construct** in Fortran is known as the **do loop**.
The basic syntax is:

```
[ loop name ] DO [ loop control ]  
    block of statements  
END DO [ loop name ]
```

- loop name and loop control are optional
- With no loop control it loops indefinitely
- **END DO** or **ENDDO** can be used.

Indexed DO Loop (1)

This is the most common form.

```
DO <control-var> = <initial>, <final> [, <step>]  
    block of statements  
END DO
```

- <control var> is an integer variable.
- <initial>, <final> and <step> are integer expressions
- If <step> is omitted its default value is 1.
- <step> cannot be zero.

Indexed DO Loop (2)

If **<step>** is positive:

- **<control-var>** receives the value of **<initial>**.
- If the value of **<control-var>** is less than or equal to **<final>**, the block of statements contained within the loop are executed.
- Then the value of **<control-var>** is iterated by **<step>** and compared to **<final>**.
- When the value of **<control-var>** exceeds the value of **<final>** execution moves below the **END DO**.

Indexed DO Loop (3)

If **<step>** is negative:

- **<control-var>** receives the value of **<initial>**.
- If the value of **<control-var>** is greater than or equal to **<final>**, the block of statements contained within the loop are executed.
- Then the value of **<control-var>** is iterated by **<step>** and compared to **<final>**.
- When the value of **<control-var>** is less than the value of **<final>** execution moves below the **END DO**.

Indexed DO Loop (4)

Important notes:

- `<step>` cannot be **zero**.
- Before the loop starts the values of `<initial>`, `<final>` and `<step>` are evaluated **exactly once**. i.e., these values are never re-evaluated as the loop executes.
- Never attempt to change the values of `<control-var>`, `<initial>`, `<final>` or `<step>`.
- Don't use **real** variables for the loop expressions.
- Examples: **simpleloop.f90**

Non-Indexed DO Loop

We can omit the loop control but then we need a way to exit the loop.

- The **EXIT** statement brings the flow of control to the statement following the **END DO**.
- The **CYCLE** statement starts the next iteration.
- Examples: **exitloop.f90**

WHILE Loop

The **WHILE loop control** has the following form:

```
DO WHILE ( <logical expression> )
```

```
.
```

```
END DO
```

- The **logical expression** is reevaluated for each cycle
- The loop exits as soon as it becomes **.FALSE.**
- It's actually a redundant feature as the same thing can be accomplished with an **EXIT** statement.
- Examples: **whileloop.f90**

CONTINUE Statement

CONTINUE is a statement that does nothing
Used to be fairly common particularly before **END DO**
came along but now it is rare.

It's mainly a placeholder for **labels**
This is **purely** to make the code clearer

It can be used anywhere a **statement** can.

RETURN and STOP

RETURN causes a procedure to halt execution with control given back to the calling program

STOP halts execution cleanly.

Typically used with an **IF** statement to stop the program if some error condition is encountered.