

1. Selected Execution using IF's, stopping with STOP

Allows execution of a series of statements (AKA a *block*) only if a set of conditions is true.

- Conditional execution begins at if/else if or else and continues until block ends with else if, else or endif
- Code inside IF indented 3 spaces
- Can nest if/else if, etc.
- END IF same as ENDIF
- Single line IF has no then, cannot have an ELSE (example later)
- STOP: Halts whole program no matter where it is found

```
PROGRAM find_max
  IMPLICIT NONE
  REAL :: num1, num2, max

  PRINT*, 'Enter first number'
  READ *, num1
  PRINT *, 'Enter second number'
  READ *, num2
  IF (num1 .GT. num2) THEN
    max = num1
  ELSE IF (num1 .LT. num2) THEN
    max = num2
  ELSE
    max = num1
    PRINT*, 'Numbers are the same!'
  END IF
  print *, 'The maximum number is ', max
END PROGRAM find_max
```

3. Single-line IFs

Single line IFs have no THEN clause, and the IF body is exactly one Fortran execution statement. Form:

```
IF (condition) execution statement
```

Examples:

```
....
if (i .GT. 50) PRINT *, 'I is big!'
....
if (i .GT. max) &
  max = i
....
```

2. IF/STOP examples

Unconditional execution resumes at END IF:

```
....
IF (i/2*2 == i) THEN
  PRINT*, i, ' is even'
ELSE
  PRINT *, i, ' is odd'
END IF
PRINT*, 'I'm executed always!'
....
```

Can string together or not have ELSE:

```
IF (ncars .GT. 4) THEN
  ....
END IF

IF (i .EQ. 1) THEN
  ....
ELSE IF (i .EQ. 2) THEN
  ....
ELSE IF (i .EQ. 3) THEN
  ....
ELSE
  ....
END IF
```

If statements can be *nested*:

```
....
IF (i .GT. j .AND. k .NE. 0) THEN
  if (k .LT. 0) then
    PRINT*, 'Illegal K val = ', k
    STOP
  ELSE
    k = k + j - i
  END IF
END IF
....
```

Same as:

```
....
IF (i .GT. j) THEN
  IF (k .NE. 0) THEN
    IF (k .LT. 0) THEN
      PRINT*, 'Illegal K val = ', k
      STOP
    END IF
    k = k + j - i
  END IF
END IF
....
```

4. Repeating instructions with logical loops

Repeats all the instructions in the *loop body* any number of times.

Repeat forever:

```
[name:] DO
  ...
  block of code
  ....
END DO [name]
```

Repeat until condition no longer true:

```
[name:] DO WHILE (condition)
END DO [name]
```

Unnamed loop:

```
DO
  READ *, temp_c
  temp_f = 9.0*temp_c/5.0 + 32.0
  PRINT *, temp_c, 'C = ', temp_f, 'F'
END DO
```

Repeat until invalid temperature entered:

```
temp_c = 0.0
DO WHILE (temp_c .GE. -273.15)
  READ *, temp_c
  temp_f = 9.0*temp_c/5.0 + 32.0
  PRINT *, temp_c, 'C = ', temp_f, 'F'
END DO
```

Named loop:

```
TCONV: DO
  READ *, temp_c
  temp_f = 9.0*temp_c/5.0 + 32.0
  PRINT *, temp_c, 'C = ', temp_f, 'F'
END DO TCONV
```

DO WHILE rarely used:

- Not performance friendly
- Not always good to exit at top
- Can use DO/EXIT instead

5. Repeating instructions with counter loops

Counter loops most common loop, and they repeat the loop body $\max((\text{stop}-\text{start}+\text{inc})/\text{inc}, 0)$ times.

General form:

```
[name:] DO variable = start, stop [, inc]
....
LOOP BODY
....
END DO [name]
```

- Most widely used loop
- Induction variable cannot be modified in loop body
- If inc not provided, assumed 1
- If start > stop, body never executed
- Loops & ifs can be nested
 - Naming is handy for deeply nested loops/ifs
- Never do IF *inside* a loop if you can get do it *outside*

Print 1-5 Example:

```
DO i = 1, 5
  PRINT*, i
END DO
Output:
1
2
3
4
5
```

Print even numbers:

```
DO i = 2, 6, 2
  PRINT *, i
END DO
Output:
2
4
6
```

6. Controlling loop execution with CYCLE and EXIT

CYCLE skips to the next iteration of the loop body:

```
DO I = 1, 4
  PRINT *, I
  IF (I .LE. 2)
    CYCLE
  END IF
  PRINT *, 'Greater than 2!'
```

END DO

Output:

```
1
2
3
Greater than 2!
4
Greater than 2!
```

EXIT breaks out of the loop completely:

```
N = 3
DO I = 1, 2000
  PRINT *, I
  IF (I .GT. N) EXIT
END DO
```

Output:

```
1
2
3
4
```

7. All these loops do the same thing

Loops:

```
loop1: DO i = 1, 9, 2
  PRINT *, i
END DO loop1
```

```
i = 1
loop2: DO
  IF (i .GT. 9) EXIT
  PRINT *, i
  i = i + 2
END DO loop2
```

```
I = 1
loop3: DO WHILE (i .LE. 9)
  PRINT *, i
  i = i + 2
END DO loop3
```

Output of any of these loops:

```
1
3
5
7
9
```

8. Nested loops, naming and CYCLE/EXIT

- By default EXIT/CYCLE apply to innermost loop
 - Can be made to affect *outer* loops by supplying name
 - : Remember these inst affect loops, not IFS!

```
jloop: DO J = 1, N
  iloop: DO I = 1, M
    ....
    if (i .LT. j) CYCLE          ! skip to next I iter
    ....
    if (i*j > 3200) EXIT jloop ! quit both loops!
    ....
    if (i .GT. J .AND. I .GT. 512) &
      CYCLE jloop              ! skip to next j iter
  END DO iloop
END DO jloop
```

9. Coding style for this class

- All programs begin with header comment giving programmer name, assignment number/name, a list of any collaborators, and any comments that you wish to pass on to the grader
- Maximum line length is 80 characters
- Comments about one line can be explained with a ! to the right of the line
 - Line these up when possible
- Bodies of loops, programs, subprograms are all indented by 3 spaces from begin/end statements
- No use of the tab character
- Uppercase keywords, lowercase everything else good, but not required
 - Whatever you do, do it consistently
- One statement/line unless they are extremely simple and similar
 - 'i = 0 ; j = 0 ; k = 0 ; M = 3' prob OK
- Comments for blocks of code are of form:
 - !
 - ! Comment for block indented to same level as block
 - !

10. Example

```
!
! CS 1073 Assignment#1 written by R. Clint Whaley. No collaborators.
! I discussed basic ideas with C. Wenk.
!
PROGRAM dumb
!
! This comment describes what the program does.
!
! .....
! This comment describes what the loop will do
!
DO I = 1, N
!
! This comment describes what this IF clause accomplishes
!
IF (I.GT.K) THEN
    PRINT *, I           ! comment describing this line only
!
! This comment describes what the ELSE clause does
!
ELSE
    PRINT *, K           ! notice indented precisely 3 spaces
END IF
END DO
END PROGRAM dumb
```