Fortran Control Structures

LOGICAL Variables

- A LOGIAL variable can only hold either .TRUE. or .FALSE. , and cannot hold values of any other type.
- Use T or F for LOGICAL variable READ(*,*)
- WRITE(*,*) prints T or F for .TRUE. and .FALSE., respectively.

```
LOGICAL, PARAMETER :: Test = .TRUE.
LOGICAL            :: C1, C2

C1 = .true.      ! correct
C2 = 123         ! Wrong
READ(*,*)  C1, C2
C2 = .false.
WRITE(*,*) C1, C2
```

Relational Operators:

- **Fortran has six relational operators: `<`, `<=`, `>`, `>=`, `==`, `/=`.**
- **Each of these six relational operators takes two expressions, compares their values, and yields `.TRUE.` or `.FALSE.`**
- **Thus, `a < b < c` is wrong, because `a < b` is `LOGICAL` and `c` is `REAL` or `INTEGER`.**
- **`COMPLEX` values can only use `==` and `/=`**
- **`LOGICAL` values should use `.EQV.` or `.NEQV.` for equal and not-equal comparison.**

Relational Operators:

- **Relational operators have *lower* priority than arithmetic operators, and `//`.**
- **Thus, `3 + 5 > 10` is `.FALSE.` and `"a" // "b" == "ab"` is `.TRUE.`**
- **Character values are encoded. Different standards (*e.g.*, BCD, EBCDIC, ANSI) have different encoding sequences.**
- **These encoding sequences may not be compatible with each other.**

`IF-THEN-ELSE` Statement:

- **Fortran has three if-then-else forms.**
- **The most complete one is the `IF-THEN-ELSE-IF-END IF`**
- **An old logical `IF` statement may be very handy when it is needed.**
- **There is an old and obsolete arithmetic `IF` that you are not encouraged to use. We won't talk about it at all.**
- **Details are in the next few slides.**

IF-THEN-ELSE Statement:

- **IF-THEN-ELSE-IF-END IF is the following.**
- **Logical expressions are evaluated sequentially (*i.e.*, top-down). The statement sequence that corresponds to the expression evaluated to .TRUE. will be executed.**
- **Otherwise, the ELSE sequence is executed.**

```
IF (logical-expression-1) THEN
    statement sequence 1
ELSE IF (logical-expression-2) THEN
    statement seqence 2
ELSE IF (logical-expression-3) THEN
    statement sequence 3
ELSE IF (.....) THEN
    ...........
ELSE
    statement sequence ELSE
END IF
```

IF-THEN-ELSE Statement:

- Two Examples:

Find the minimum of **a**, **b** *and* **c** *and saves the result to* **Result**

```
IF (a < b .AND. a < c) THEN
   Result = a
ELSE IF (b < a .AND. b < c) THEN
   Result = b
ELSE
   Result = c
END IF
```

Letter grade for **x**

```
INTEGER :: x
CHARACTER(LEN=1) :: Grade

IF (x < 50) THEN
   Grade = 'F'
ELSE IF (x < 60) THEN
   Grade = 'D'
ELSE IF (x < 70) THEN
   Grade = 'C'
ELSE IF (x < 80) THEN
   Grade = 'B'
ELSE
   Grade = 'A'
END IF
```

`IF-THEN-ELSE` Statement:

- The `ELSE-IF` part and `ELSE` part are optional.
- If the `ELSE` part is missing and none of the logical expressions is `.TRUE.`, the `IF-THEN-ELSE` has no effect.

no `ELSE-IF`

```
IF (logical-expression-1) THEN
     statement sequence 1
ELSE
     statement sequence ELSE
END IF
```

no `ELSE`

```
IF (logical-expression-1) THEN
     statement sequence 1
ELSE IF (logical-expression-2) THEN
     statement sequence 2
ELSE IF (logical-expression-3) THEN
     statement sequence 3
ELSE IF (.....) THEN
     ...........
END IF
```

IF-THEN-ELSE Can be Nested:

- Another look at the quadratic equation solver.

```
IF (a == 0.0) THEN                        ! could be a linear equation
   IF (b == 0.0) THEN                     ! the input becomes c = 0
      IF (c == 0.0) THEN                  ! all numbers are roots
         WRITE(*,*) 'All numbers are roots'
      ELSE                                ! unsolvable
         WRITE(*,*) 'Unsolvable equation'
      END IF
   ELSE                                   ! linear equation bx + c = 0
      WRITE(*,*) 'This is linear equation, root = ', -c/b
   END IF
ELSE                                ! ok, we have a quadratic equation
   ...... solve the equation here ......
END IF
```

IF-THEN-ELSE Can be Nested:

- **Here is the big ELSE part:**

```
d = b*b - 4.0*a*c
IF (d > 0.0) THEN                    ! distinct roots?
   d = SQRT(d)
   root1 = (-b + d)/(2.0*a)          ! first root
   root2 = (-b - d)/(2.0*a)          ! second root
   WRITE(*,*) 'Roots are ', root1, ' and ', root2
ELSE IF (d == 0.0) THEN              ! repeated roots?
   WRITE(*,*) 'The repeated root is ', -b/(2.0*a)
ELSE                                 ! complex roots
   WRITE(*,*) 'There is no real roots!'
   WRITE(*,*) 'Discriminant = ', d
END IF
```

Logical `IF`

- The logical `IF` is from Fortran 66, which is an improvement over the Fortran I arithmetic `IF`.
- If logical-expression is `.TRUE.` , *statement* is executed. Otherwise, execution goes though.
- The statement can be assignment and input/output.

```
IF (logical-expression) statement
```

```
Smallest = b
IF (a < b)  Smallest = a
```

```
Cnt = Cnt + 1
IF (MOD(Cnt,10) == 0) WRITE(*,*) Cnt
```

The SELECT CASE Statement:

- **Fortran has the `SELECT CASE` statement for selective execution if the selection criteria are based on simple values in `INTEGER`, `LOGICAL` and `CHARACTER`. No, `REAL` is not applicable.**

```
SELECT CASE (selector)
    CASE (label-list-1)
        statements-1
    CASE (label-list-2)
        statements-2
    CASE (label-list-3)
        statements-3
    ...... other cases ......
    CASE (label-list-n)
        statements-n
    CASE DEFAULT
        statements-DEFAULT
END SELECT
```

selector **is an expression evaluated to an `INTEGER`, `LOGICAL` or `CHARACTER` value**

label-list **is a set of constants or `PARAMETERS` of the same type as the *selector***

statements **is one or more executable statements**

The SELECT CASE Statement:

- The **SELECT CASE** statement is executed as follows:
  - Compare the value of *selector* with the labels in each case. If a match is found, execute the corresponding *statements*.
  - If no match is found and if **CASE DEFAULT** is there, execute the *statements-***DEFAULT**.
  - Execute the next statement following the **SELECT CASE**.

```
SELECT CASE (selector)
    CASE (label-list-1)
        statements-1
    CASE (label-list-2)
        statements-2
    CASE (label-list-3)
        statements-3
    ...... other cases ......
    CASE (label-list-n)
        statements-n
    CASE DEFAULT
        statements-DEFAULT
END SELECT
```

optional

The SELECT CASE Statement:

- Two examples of **SELECT CASE**:

```
CHARACTER(LEN=4) :: Title
INTEGER :: DrMD = 0, PhD = 0
INTEGER :: MS = 0, BS = 0
INTEGER ::Others = 0

SELECT CASE (Title)
  CASE ("DrMD")
    DrMD = DrMD + 1
  CASE ("PhD")
    PhD = PhD + 1
  CASE ("MS")
    MS = MS + 1
  CASE ("BS")
    BS = BS + 1
  CASE DEFAULT
    Others = Others + 1
END SELECT
```

```
CHARACTER(LEN=1) :: c

SELECT CASE (c)
  CASE ('a' : 'j')
    WRITE(*,*) 'First ten letters'
  CASE ('l' : 'p', 'u' : 'y')
    WRITE(*,*)                              &
        'One of l,m,n,o,p,u,v,w,x,y'
  CASE ('z', 'q' : 't')
    WRITE(*,*) 'One of z,q,r,s,t'
  CASE DEFAULT
    WRITE(*,*) 'Other characters'
END SELECT
```

The Counting DO Loop:

- Fortran has two forms of DO loop: the counting DO and the general DO.
- The counting DO has the following form:

```
DO control-var = initial, final [, step]
    statements
END DO
```

- `control-var` is an `INTEGER` variable, `initial`, `final` and `step` are `INTEGER` expressions; however, `step` *cannot be zero*.
- If `step` is omitted, its default value is `1`.
- *statements* are executable statements of the DO.

The Counting `DO` Loop:

- **Before a `DO`-loop starts, expressions `initial`, `final` and `step` are evaluated *exactly once*. When executing the `DO`-loop, these values will *not* be re-evaluated.**
- **Note again, the value of `step` *cannot be zero*.**
- **If `step` is positive, this `DO` counts up; if `step` is negative, this `DO` counts down**

```
DO control-var = initial, final [, step]
     statements
END DO
```

The Counting DO Loop:

- Two simple examples:

```
INTEGER :: N, k

READ(*,*)  N
WRITE(*,*) "Odd number between 1 and ", N
DO k = 1, N, 2
   WRITE(*,*) k
END DO
```

odd integers between 1 & N

```
INTEGER, PARAMETER :: LONG = SELECTED_INT_KIND(15)
INTEGER(KIND=LONG) :: Factorial, i, N

READ(*,*)  N
Factorial = 1_LONG
DO i = 1, N
   Factorial = Factorial * i
END DO
WRITE(*,*) N, "! = ", Factorial
```

factorial of N

The Counting `DO` Loop:

- Important Notes:
  - The step size `step` *cannot be zero*
  - Never change the value of any variable in `control-var` and `initial`, `final`, and `step`.
  - For a count-down `DO`-loop, `step` must be negative. Thus, "`do i = 10, -10`" is not a count-down `DO`-loop, and the *statements* portion is not executed.
  - Fortran 77 allows `REAL` variables in `DO`; but, don't use it as it is not safe.

General DO-Loop with EXIT:

- The general DO-loop has the following form:

```
DO
      statements
END DO
```

- *statements* will be executed repeatedly.
- To exit the DO-loop, use the EXIT or CYCLE statement.
- The EXIT statement brings the flow of control to the statement following (*i.e.*, exiting) the END DO.
- The CYCLE statement starts the next iteration (*i.e.*, executing *statements* again).

General DO-Loop with EXIT:

```
REAL, PARAMETER :: Lower = -1.0, Upper = 1.0, Step = 0.25
REAL :: x

x = Lower                    ! initialize the control variable
DO
   IF (x > Upper) EXIT       ! is it > final-value?
   WRITE(*,*) x              ! no, do the loop body
   x = x + Step              ! increase by step-size
END DO
```

```
INTEGER :: Input

DO
   WRITE(*,*) 'Type in an integer in [0, 10] please --> '
   READ(*,*) Input
   IF (0 <= Input .AND. Input <= 10) EXIT
   WRITE(*,*) 'Your input is out of range. Try again'
END DO
```