Programming for Engineers
Fortran: Decision-Based Control Structures

Abu Hasan Abdullah

January 7, 2009

Preview

1. Unconditional Transfer

2. Conditional Statements and Constructs

   (a) `IF` Statement
   (b) Block `IF` Construct
   (c) The `IF...ELSE` Construct
   (d) `SELECT CASE` Construct

MAYO W. E. AND CWIAKALA M. (1995): *Programming with Fortran 77*
ISBN 0-07-041155-7, McGraw-Hill

Unconditional Transfer

- The simplest transfer operation. Also known as `GO TO` statement. Purposes
  - skip over a set of instructions
  - repeat a set of instructions
- **Avoid** using this statement in your program if you could! It's bad.

Unconditional Transfer

- It transfer control to another line in the program and the line to receive control must be labeled using a statement label.

- General form of a `GO TO` statement

```
GO TO statement label
```

- Example

```
        GO TO 20
        ...
20      PRINT *, AREA
```

- Turn **Example 4.1** and **Example 4.2** into complete programs, compile and run them to study the effect of `GO TO` statement

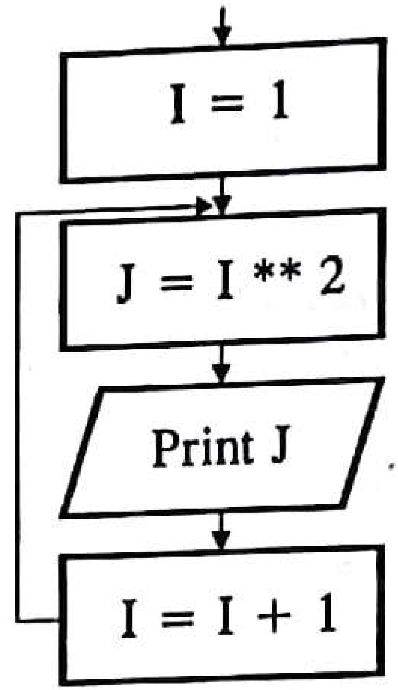
EXAMPLE 4.1

Here is a program that produces a list of the squares of positive integers

Program

```
C Use of Go To Statement to
C construct a loop. The
C variable I is a "counter"
        I = 1
20      J = I ** 2
        PRINT *, J
C After I squared is computed
C and printed, we increment I
C by 1 and loop back to sl=20
        I = I + 1
        GO TO 20
```

Flowchart



When we first start this program, I has the value of 1. Its square is computed and printed, after which I increases by 1 and the whole process repeats. While this program works and produces the desired result, it is a very poor way to accomplish this. Note for example, that the process presented is an *infinite loop*, and there is no way to get out.

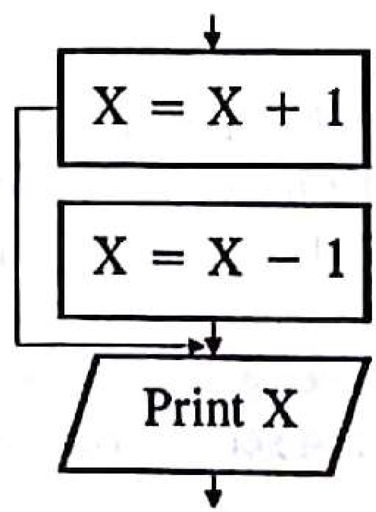
EXAMPLE 4.2

In the simple example program below, we use the GO TO statement to skip over another line within the program.

Program

```
C Demonstration of GO TO as a
C means of skipping over a set
C of instructions.
       X = X + 1
       GO TO 40
C By executing the previous
C instruction, the next line
C is skipped.
30     X = X - 1
40     PRINT *, X
```

Flowchart



Conditional Statements and Constructs

- Built upon `IF` statement to construct conditional tests. Based on this test it will be able to branch to other lines of the code for other operations.
- `IF` statement provides a way to test a condition and execute a single command if the test is true.
- General form of a `IF` statement

```
IF (test condition) statement-to-execute-if-true
```

- Example

```
IF (VELOCITY.LE.0.0) PRINT *, 'MASS NOT MOVING'
```

Example Relational Operator	Description
IF (DENO .EQ. 0) STOP	Halt the program if the value of DENO = 0
IF (TEMP .LT. 0) PRINT *, TEMP	If TEMP < 0 then print value of TEMP
IF (X .LE. XMIN) XMIN = X	If X ≤ XMIN, set value of XMIN to X
IF (S .GT. 1E6) S = 1E6	Set S to 1×10^6 if S $>1\times10^6$
IF (A .GE. 0) GO TO 10	Permissible to transfer to a statement label
IF (SQRT(X*Y) .NE. 4) X = Y	You can use expressions for comparison
IF (ABS(X) .EQ. Y*Z) A=SQRT(X)	You can compare an expression to an expression
IF (I/2*2 .EQ. I) PRINT *,'even'	How to determine if an integer I is even or odd

Conditional Statements and Constructs: Relational Operators

- Relational operators are used in the *test-condition* of an `IF` statement by comparing two quantities and return and answer of *TRUE* or *FALSE*

Operator	Description	test-condition	Result
`.LT.`	Less than	`(1.LT.2)`	true
`.LE.`	Less than or equals	`(5.2.LE.12.1)`	true
`.EQ.`	Equals	`(3.EQ.10)`	false
`.NE.`	Not equals	`(5.NE.9)`	true
`.GT.`	Greater than	`(1.GT.23)`	false
`.GE.`	Greater than or equals	`(6.GE.3)`	true

Conditional Statements and Constructs: Logical Operators

- You may wish to check more than one test-conditions before carrying out an instruction, i.e. a *compound test*. For example, two test-conditions may need to be true simultaneously before a calculation can proceed.

Operator	Description	Number of arguments
`.NOT.`	Negation	1 argument
`.AND.`	Both simultaneously	2 arguments
`.OR.`	Either/or	2 arguments

Conditional Statements and Constructs: Logical Operators

- .AND. truth table

A	B	(A).AND.(B)
T	T	T
T	F	F
F	T	F
F	F	F

- **Example:** `(stress .GT. 0.0) .AND. (stress .LT. 100.0)`

Conditional Statements and Constructs: Logical Operators

- `.OR.` truth table

A	B	(A).OR.(B)
T	T	T
T	F	T
F	T	T
F	F	F

- **Example:** `(radius .GT. 0.0) .OR. (radius .LT. 10.25)`

Conditional Statements and Constructs: Logical Operators

- `.NOT.` truth table

A	.NOT.(A)
T	F
F	T

- **Example:** `.NOT. (icount .LT. 0)`

Priority	Math Symbol	Fortran Symbol	Meaning
1	(...)	(...)	Parentheses
2	A^b	**	Exponentiation
3	x ÷	*, /	Multiplication & division
4	+ −	+, −	Addition & subtraction
5	= ≠ <	.EQ., .NE., .LT.	Relational operators
	≤ > ≥	.LE., .GT., .GE.	
6	$\bar{x}$	.NOT.	Logical negation
7	⊙	.AND.	Logical AND
8	⊕	.OR.	Logical OR

Conditional Statements and Constructs

- Turn **Example 4.4** and **Example 4.5** into complete programs, compile and run them to study roles of relational and logical operators in constructing test-conditions

EXAMPLE 4.4

Construct a logical operator to see if a number x is within the range $1.0 < x < 10.0$. This test actually consists of two separate tests, both of which must be true simultaneously:

$$1.0 < x \quad \textit{and} \quad x < 10.0$$

We construct the two tests and connect them with the .AND. logical operator:

```
READ *, X
IF(1.0.LT.X.AND.X.LT.10.0) PRINT *,X,'is between 1 and 10'
```

When you first look at this, you might have been tempted to write, as we do in mathematics:

1.0 .LT. X .LT. 10.0

But this statement is incorrect. The reason is that the operators can only compare data of the same type. They cannot compare *true* or *false* values with numerical data for example. Let's assume X = 5.0 and trace through our hypothetical solution:

1.0 .LT. X .LT. 10.0 → 1.0 .LT. 5.0 .LT. 10.0 → *true* .LT .10.0

An error occurs at this point since the .LT. operator attempts to compare two things that are incompatible (logical data with a real number in this instance).

EXAMPLE 4.5

Evaluate the following expressions, assuming that X = 10.0, Y = −2.0, and Z = 5.0:

(X*Y .LT. Z/X .OR. X/Y .GT. Z*X .AND. Z*Y .LT. X)

First, substitute the values for X, Y, and Z, and perform the mathematical operations:

(10.0*−2.0 .LT. 5.0/10.0 .OR. 10.0/−2.0 .GT. 5.0*10.0 .AND. 5.0*−2.0 .LT. 10.0)

Next, perform the relational comparisons (.LT., .GT., .LT. left to right):

(*true* .OR. *false* .AND. *true*)

From the hierarchy table, we see that .AND. takes precedence over .OR.. Thus, this reduces to

(*true* .OR. *false*) → (*true*)

IF Statement

- General form of an IF statement

```
IF (test condition) statement-to-execute-if-true
```

- Example

```
IF (VELOCITY.LE.0.0) PRINT *, 'MASS NOT MOVING'
```

Block IF Construct

- Useful when you have a single instruction to execute after the test condition is evaluated
- Not suitable in situation where more than a single instruction is needed
- General form

```
IF (test-condition) THEN
    Block of statements if test-condition is TRUE
ELSE
    Block of statements if test-condition is FALSE
END IF
```

Block IF Construct

- Code snippet

```
IF (X.LT.0.0) THEN
   PRINT *,'Error!'
ELSE
   PRINT *,'Valid'
END IF
```

- It is a good practice to indent the block of instructions, see example above, when writing an IF-THEN-ELSE-ENDIF block

- Turn **Example 4.6** and **Example 4.7** into complete programs, compile and run them to study IF-THEN-ELSE-ENDIF block usage

Program Blocks

- Block `IF` construct is one of the many *program blocks* available in Fortran
- Rules for *program blocks* include
  - From *inside* the block, control can be transferred to statement *outside* of the block
  - It is valid to transfer control from one statement of a block to another statement *within* the same block
  - Control cannot be transferred from *outside* of a block to *inside* of a block except by way of the controlling structure
  - It is possible to nest constructs as long as the inner construct is completely with the outer block i.e. NO crossing of block boundaries is permitted!!
  - It is valid for a `GO TO` to send control to the closing statement of a construct

Program Blocks

- Turn **Examples 4.8–4.11** into complete programs, compile and run them to study some of these program block rules

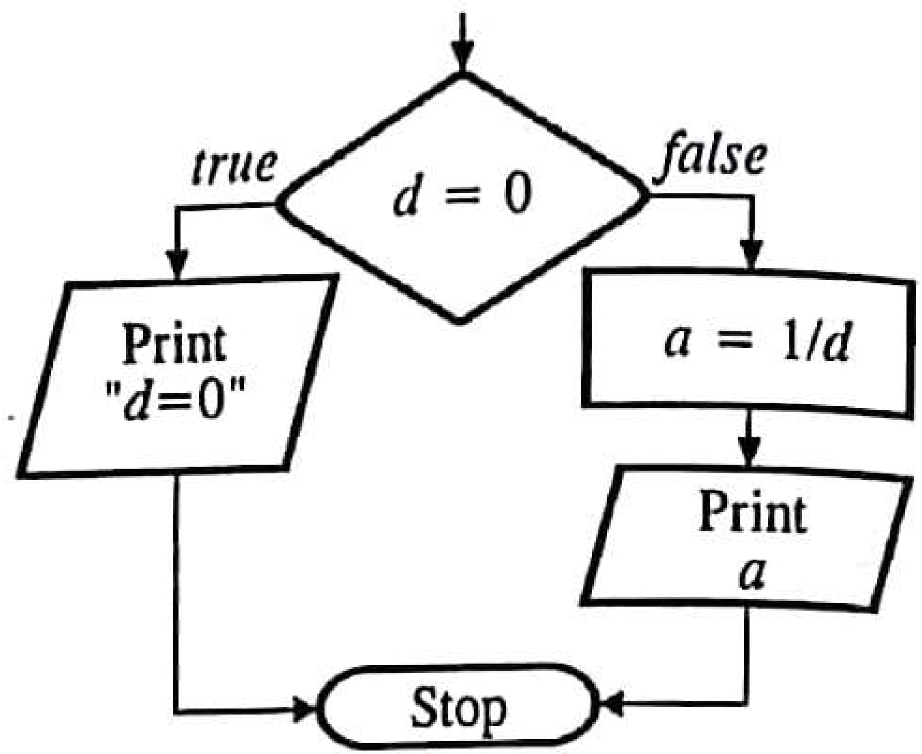
EXAMPLE 4.8

The following example demonstrates that it is permissible to transfer *out* of a block IF construct. We will see shortly that the reverse operation (transferring *into* the body of a block) is never permitted.

Program

```
C The GO TO statement in
C this example transfers
C out of the block IF
      IF(D.EQ.0) THEN
         PRINT *,'D = 0'
         GO TO 10
      ELSE
      END IF
      ANS = 1 / D
      PRINT *,ANS
10    STOP
      END
```

Flowchart



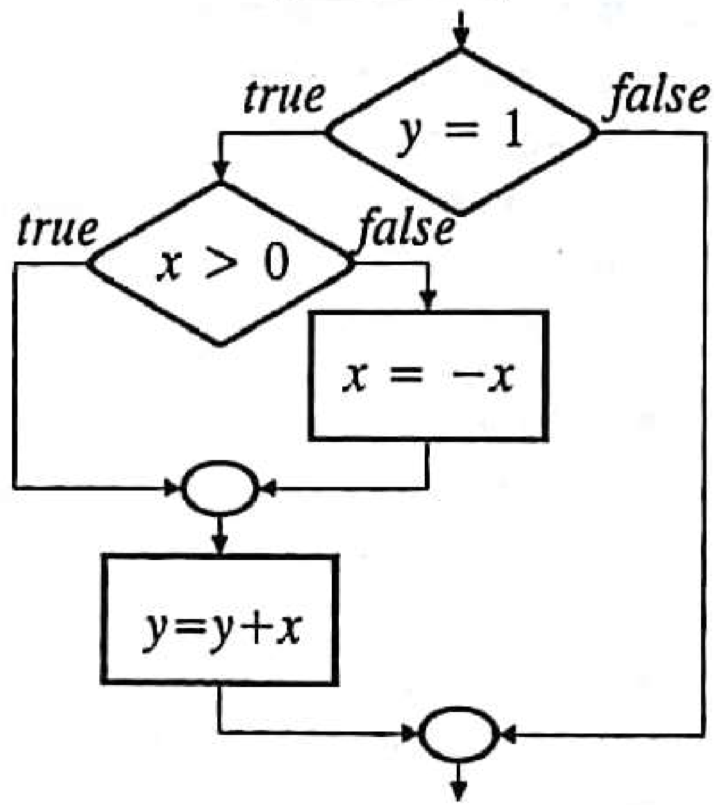
EXAMPLE 4.9

It is permissible to transfer control from one statement of a block to another statement *within* the same block.

Program

```
C The second IF statement
C will cause the program to
C jump to a position within
C the block IF
       IF(Y.EQ.1) THEN
          IF(X.GT.0) GOTO 10
          X=-X
10        Y=Y+X
       ELSE
       ENDIF
```

Flowchart



EXAMPLE 4.10

In the following example, we show how you might attempt to transfer into the middle of a block. The Fortran compiler however, will not allow you to do this.

```
      IF (X .EQ. 0) GO TO 20
      IF (Y .EQ. 0) THEN
20       X=X+1                    (This statement is inside the block IF construct)
      ELSE
      END IF
```

When the program attempts to jump to statement label 20, the statement that controls the branching operation (IF (Y .EQ. 0)) is completely bypassed.

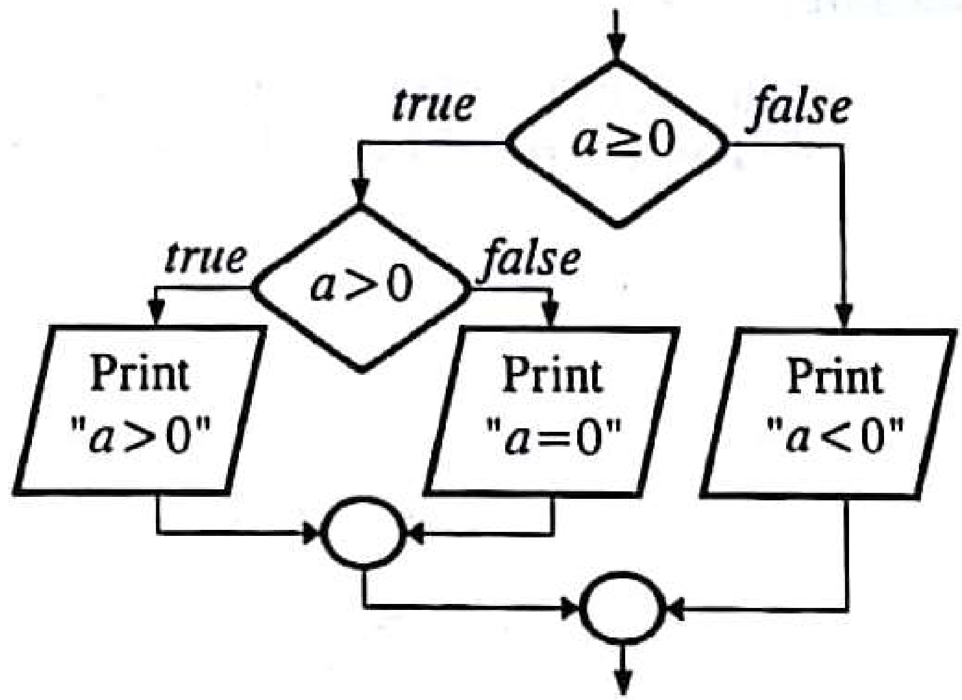
EXAMPLE 4.11

Here is a sample program to determine if *a* is positive, negative, or zero. Notice that this requires two nested block IFs, since there are three possible outcomes:

Program

```
A — IF(A .GE. 0) THEN
B —     IF(A .GT. 0) THEN
             PRINT *,'A > 0'
        ELSE
             PRINT *,'A = 0'
        END IF
    ELSE
C —     PRINT *,'A < 0'
    END  IF
```

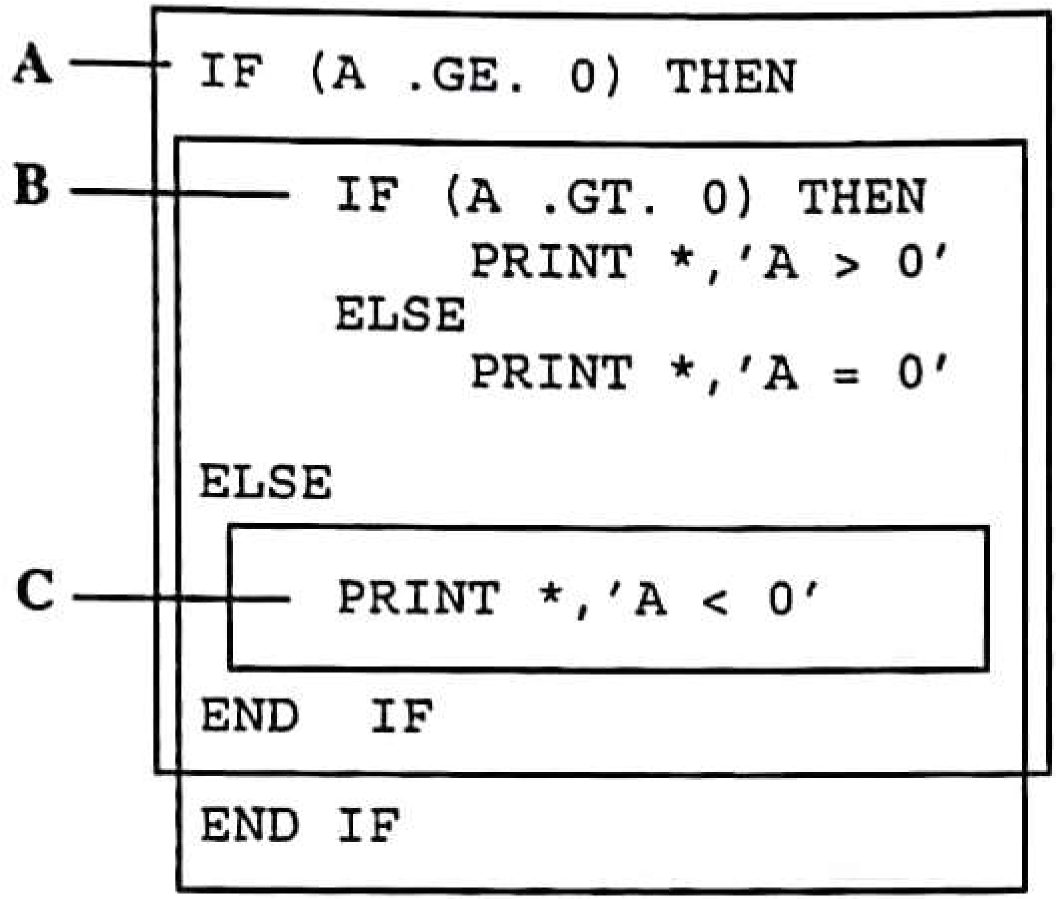
Flowchart



Here is an example of invalid nesting:

EXAMPLE 4.12

Here is the same program as in Example 4.11,



ELSE IF Construct

- ELSE IF construct is a special form of IF construct
- It is a nested block IF structure in which a block IF is placed inside the *false block* of an outer block
- ELSE IF form allows a list of conditions to be tested more precisely than with the block IF

ELSE IF **Construct**

- General form

```
IF (test-condition-1) THEN
      Block-1
ELSE IF (test-condition-2) THEN
      Block-2
...
...
ELSE IF (test-condition-N) THEN
      Block-N
ELSE
      Block-N+1
END IF
```

ELSE IF **Construct**

- Code snippet

```
IF (C .LE. 0) THEN
   PRINT *,'Frozen'
ELSE IF (C .LE. 20) THEN
   PRINT *,'Cold --> Cool'
ELSE IF (C .LE. 30) THEN
   PRINT *,'Warm'
ELSE
   PRINT *,'Hot'
END IF
```

- Turn **Example 4.14** into a complete program, compile and run it to study ELSE IF construct

EXAMPLE 4.14

The following program reads in a temperature in degrees C and prints out an appropriate message using the following criteria:

Temperature ≤ 0°C	Print "It's below freezing"
0°C < Temperature ≤ 10°C	Print "It's cold out"
10°C < Temperature ≤ 20°C	Print "It's cool out"
20°C < Temperature ≤ 30°C	Print "It's warm"
Temperature > 30°C	Print "It's hot!"

```
PRINT *,'Enter the temperature in degrees C'
READ *,C
IF (C .LE. 0) THEN
  PRINT *,'It''s below freezing'
ELSE
  IF (C .LE. 10) THEN
    PRINT *,'It''s cold out'
  ELSE
    IF (C .LE. 20) THEN
      PRINT *,'It''s cool out'
    ELSE
      IF (C .LE. 30) THEN
        PRINT *,'It''s warm'
      ELSE
        PRINT *,'It''s hot!'
      END IF
    END IF
  END IF
END IF
```

Can be written like this

This program can also be written more concisely with the ELSE IF form of the IF construct.

```
PRINT *, 'Enter the temperature in degrees C'
READ *, C
IF (C .LE. 0) THEN
   PRINT *, 'It''s below freezing'
ELSE IF (C .LE. 10) THEN
   PRINT *, 'It''s cold out'
ELSE IF (C .LE. 20) THEN
   PRINT *, 'It''s cool out'
ELSE IF (C .LE. 30) THEN
   PRINT *,'It''s warm'
ELSE
   PRINT *,'It''s hot!'
END IF
```

`SELECT CASE` Construct

- Many Fortran compilers offer the `SELECT CASE` structure as an extension of the Fortran 77 standard
- NOT all compilers offer this control structure
- If your compiler does not offer it, you have to use the nested block `IF` and/or `ELSE IF` structure to choose from among many multiple alternatives

SELECT CASE **Construct**

- General form

```
SELECT CASE (expression)
     CASE (selector list 1)
          Block-1
     CASE (selector list 2)
          Block-2
     ...
     ...
     CASE DEFAULT
          Block-N
END SELECT
```

SELECT CASE Construct

- Code snippet

```
READ *, N
SELECT CASE (N)
     CASE (1)
        PRINT *,'#1 Entered'
     CASE (2)
        PRINT *,'#2 Entered'
     CASE (3)
        PRINT *,'#3 Entered'
     CASE DEFAULT
        PRINT *,'Error!'
END SELECT
```

SELECT CASE Construct

- Have a look at **Example 4.15** and study its flowchart for SELECT CASE structure
- Turn **Examples 4.16–4.18** into complete programs, compile and run them to study SELECT CASE structure