Logical IF Statement

Used when we wish to execute a single statement or NOT execute the statement based on some condition being true or false.

Examples:

Form 1:
IF(hrs .GT. 40.0) otrate = rate * 1.5
IF (index .EQ. 0) STOP
IF (age .LT. cutoff) WRITE(..)…

Form 2:
IF(hrs > 40.0) otrate = rate * 1.5
IF (index == 0) STOP
IF (age < cutoff) WRITE(..)…

If the condition is true then execute the statement and continue with the rest of the program. Otherwise, skip statement and continue with the rest of the program.

Logical Variable

Two Fortran logical constants: .TRUE. and .FALSE.

A logical variable can have a value either .TRUE. or .FALSE. and can be declared as:

```
LOGICAL :: variable_name1, variable_name2, variable_name3,……
```

Example:

```
INTEGER :: b = 8

LOGICAL :: a
a = (b > 5)
IF(a) …..
```

A logical variable can be used as a condition in an IF statement. In the above example, the logical variable a will have the value .TRUE.
Logical Expressions & Conditions

Relation expressions are the most common type of Logical Expressions.

Logical Expressions are also called Boolean Expressions.

Relational Operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Form 1</th>
<th>Form 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>==</td>
<td>.EQ.</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
<td>.GT.</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
<td>.LT.</td>
</tr>
<tr>
<td>Not equal to</td>
<td>/=</td>
<td>.NE.</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>.GE.</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>.LE.</td>
</tr>
</tbody>
</table>

Relational Expressions Syntax:

<arithmetic expression> <relational operator> <arith. exp>

Examples:

(a > b)
((hours - 40.0) > 0.0)
(b**2 < 4*a*c)
(b**2 == 4*a*c)
(4*a + c /= b*c + 4)
(ABS(error) <= epsilon)
Logical IF Example

Problem:

Input: cols 1 - 4 account number, cols 6 - 8 test score
Determine: the average score, the number of A’s
(i.e. score is ≥ 90) and the highest score.

Sample Program

PROGRAM test

IMPLICIT NONE
INTEGER :: actnum, test_score, count=0, a_count=0, high_score=0, total=0, ios
REAL :: average_score
OPEN(...)
OPEN(...)
READ (9, 100, IOSTAT=ios)actnum, test_score
DO
  IF (ios < 0) EXIT
  count = count + 1
  total = total + test_score
  IF(test_score >= 90) a_count = a_count + 1
  IF(test_score > high_score) high_score = test_score
  READ (9, 100, IOSTAT=ios) actnum, test_score
END DO
average_score = REAL(total) / count
WRITE(10,201) count
WRITE(10,202) average_score
WRITE(10,203) a_count
WRITE(10,204) high_score
STOP

100 FORMAT (...)
201 FORMAT (...)
202 FORMAT (...)
203 FORMAT (...)
204 FORMAT (...)
END PROGRAM test

Note: comments and program headers have been omitted from the program examples in the class notes due to lack of space on the overheads.
Hand Checking

Desk tracing (a.k.a. hand checking) involves making up sample data and executing/tracing the program by hand manually performing the actions that the machine would carry out.

Example Trace/Check of the previous program

Sample input data file contents:

```
1073 50
2975 90
2074 75
3398 95
```

Primary Memory Storage Variables

<table>
<thead>
<tr>
<th>count</th>
<th>total</th>
<th>a_count</th>
<th>high_score</th>
<th>actnum</th>
<th>test_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>1073</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>140</td>
<td>1</td>
<td>90</td>
<td>2975</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>215</td>
<td>1</td>
<td>90</td>
<td>2074</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>310</td>
<td>2</td>
<td>95</td>
<td>3398</td>
<td>95</td>
</tr>
</tbody>
</table>

average_score
77.5
Multiple Statement Selection

Usually it is desired to execute several statements based on some condition or boolean expression (evaluates to either true or false).

Syntax:

```
IF (condition) THEN
  statement
  statement
  ...
END IF
```

Boolean Expression Considerations

Any variable used in the condition of an IF statement must have already been given a value! (initialized?)

Fortran (unlike English) does not allow implied subjects.

Example:

The mathematical expression: $a < b < c$

cannot be expressed as it is;

it must be stated as: $(a < b) \land (b < c)$
Block IF Example

Problem
Consider the previous student/grades problem. Suppose that there are two classes of students (morning 0, and afternoon 1). Suppose that we want statistics for each class.

Sample Program

DO

IF (class == 0) then
  morn_count = morn_count + 1
  morn_total = morn_total + test_score
  IF (test_score >= 90) morn_a = morn_a + 1
END IF

IF (class == 1) THEN
  aft_count = aft_count + 1
  aft_total = aft_total + test_score
  IF (test_score >= 90) aft_a = aft_a + 1
END IF

IF (test_score > high_score) THEN
  high_score = test_score
  high_actnum = actnum
END IF

END DO

200 WRITE(....)...

NOTE indentation used and initialization of variable(s).
Block IF THEN ELSE

adds an ELSE Clause:

Syntax:

**IF** (condition) **THEN**

  statement

  • executed when (condition) is true

  •

**ELSE**

  statement

  • executed when (condition) is false

  •

**END IF**

Example

The first two if's of the previous example can be combined:

```fortran
IF (class == 0) THEN
  morn_count = morn_count + 1
  morn_total = morn_total + test_score
  IF (test_score >= 90) morn_a = morn_a + 1
ELSE
  aft_count = aft_count + 1
  aft_total = aft_total + test_score
  IF (test_score >= 90) aft_a = aft_a + 1
END IF
```

Conditions that are "mutually exclusive", (one condition being true excludes all others from being true), should be tested for with nested ifs, (as opposed to disjoint ifs), for efficiency.
Nested IF Statements

IF Statements Inside of IF Statements

Syntax:

IF (cond1) THEN
    statement A
    IF (cond2) THEN
        statement B
        statement C
    ELSE
        statement D
    END IF
ELSE
    statement E
    statement F
END IF
statement G

Under what (true) conditions, do each of the statements A..G get executed?

<table>
<thead>
<tr>
<th>Cond1</th>
<th>Cond2</th>
<th>Executed statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>A B C G</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>A D G</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>E F G</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>E F G</td>
</tr>
</tbody>
</table>
Nested IFs Example

Given three integer vars (a, b, c), having unique values, output the values in order.

```
IF (a > b) THEN
  IF (a > c) THEN
    IF (b > c) THEN
      WRITE(*,*) a, b, c
    ELSE
      WRITE(*,*) a, c, b
    END IF
  ELSE
    WRITE(*,*) c, a, b
  END IF
ELSE
  WRITE(*,*) c, a, b
END IF
ELSE
  IF (b > c) THEN
    IF (a > c) THEN
      WRITE(*,*) b, a, c
    ELSE
      WRITE(*,*) b, c, a
    END IF
  ELSE
    WRITE(*,*) c, b, a
  END IF
END IF
```

{ a is largest }
{ a > b & c > a }
{ b is largest }
{ b > a & c > b }
Equivalent IFs

Given:

IF ( x  <= y) THEN
  M = 1
ELSE
  M = 2
END IF

Which of the following IFs are logically equivalent?

IF ( x > y) THEN IF ( y >= x) THEN
  M = 2 M = 1
ELSE ELSE
  M = 1 M = 2
END IF END IF

IF ( y <= x) THEN IF ( y <= x) THEN
  M = 1 M = 2
ELSE ELSE
  M = 2 M = 1
END IF END IF

...
Trace all possible orderings of \( x \) & \( y \).

Table of all possible ordering sample values.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( X \leq Y )</th>
<th>( X &gt; Y )</th>
<th>( Y \geq X )</th>
<th>( Y &lt; X )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Y &lt; X</td>
<td>Y &gt; X</td>
<td>1</td>
<td>Y &lt; X</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Y &lt; X</td>
<td>Y &gt; X</td>
<td>2</td>
<td>Y &lt; X</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Y &lt; X</td>
<td>Y &gt; X</td>
<td>2</td>
<td>Y &lt; X</td>
</tr>
</tbody>
</table>

To form an equivalent IF, negate the condition and switch the ordering of the THEN & ELSE statements.

Forming Equivalent IFs

Given:

\[
\begin{align*}
\text{IF} \ (\text{cond}) & \quad \text{THEN} & & \text{IF} \ (\neg \text{cond}) & \quad \text{THEN} \\
S1 & & & S3 \\
S2 & & & S4 \\
\text{ELSE} & & \text{ELSE} & \\
S3 & & S1 & \\
S4 & & S2 & \\
\text{END IF} & & \text{END IF} & \\
\end{align*}
\]
Logical (Boolean) Operators

.NOT.  .AND.  .OR.  .EQV.  .NEQV.

Logical operators allow relational expressions to be combined to form complex compound conditions.

Generally, logical expressions syntax:

( relational expression) .logical operator. ( relational expression)

Examples:

((class == 0) .AND. (test_score >= 90))
((time == 930) .OR. (time == 1400))
((time == 930) .OR. (.NOT. (time ==1400)))

Complex Condition Evaluation

Truth tables for: ( A .logical operator. B )
where A and B are relational expressions.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>.NOT. X</th>
<th>X .AND. Y</th>
<th>X .OR. Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

The logical operators .EQV. and .NEQV. will be discussed in class
Complex Condition Evaluation

Operator Hierarchy

0) ( ... ) do what’s in ( ) first
1) .NOT.  2) .AND.  3) .OR.
4) .EQV. and .NEQV.

Complex Cond Evaluation Example

Given:
X = 2.0  Y = 3.0  Z = 1.0

Evaluate
(Y == X + Z) .OR. (Y >= Z) .AND. .NOT. (X < Z)

All solutions, (for any values of X, Y or Z), can be determined by developing the truth table for the complex condition.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>B.AND. .NOT.C</th>
<th>A.OR.B.AND..NOT.C</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Problem: Determine what day of the week any given date falls upon.

Given: the following formula which computes the day of the week for any date, \((m, d, y;\) where \(y > 1752\))

\[
\text{calc\_day} = d + 2m + \text{INT}(3(m+1)/5) + y + \text{INT}(y/4) - \text{INT}(y/100) + \text{INT}(y/400) + 1
\]

\[\text{weekday} = \text{MOD} (\text{calc\_day}, 7)\]

where:

- \(\text{weekday} = 0\) \(\rightarrow\) Sunday
- \(\text{weekday} = 1\) \(\rightarrow\) Monday
- \(\ldots\)
- \(\ldots\)
- \(\ldots\)
- \(\text{weekday} = 6\) \(\rightarrow\) Saturday

Note: the formula requires that Jan & Feb be treated as the 13th & 14th months, respectively, of the preceding year.

The INT function returns only the integer result.

The Fortran function MOD returns the remainder.
PROGRAM day_of_week
IMPLICIT NONE
INTEGER :: month, day, year, calc_day, weekday, ios
OPEN ( 9, FILE = 'inday.dat')
OPEN (10, FILE = 'outday.dat')
WRITE(10,10)
10 FORMAT('This program determines the day of the', &
' week for any given date')
READ (9,40, IOSTAT=ios) month, day, year
40 FORMAT( I2, I3, I5)
DO
  IF (ios < 0) EXIT
  IF (month < 3) THEN
    month = month + 12
    year = year -1
  END IF
  calc_day = day + 2*month + (3*(month+1)/5) + year &
  weekday = MOD(calc_day, 7 )
  IF (month > 12) THEN
    month = month - 12
    year = year +1
  END IF
  WRITE (10, *)
  IF (weekday == 0 ) THEN
    WRITE( 10, 100) month, day, year
  ELSE IF (weekday == 1 ) THEN
    WRITE(10, 101) month, day, year
  ELSE IF (weekday == 2 ) THEN
    WRITE(10, 102) month, day, year
  ELSE IF (weekday == 3 ) THEN
    WRITE(10, 103) month, day, year
  ELSE IF (weekday == 4 ) THEN
    WRITE(10, 104) month, day, year
  ELSE IF (weekday == 5 ) THEN
    WRITE(10, 105) month, day, year
  ELSE IF (weekday == 6 ) THEN
    WRITE(10, 106) month, day, year
  END IF
  READ (9,40, IOSTAT=ios) month, day, year
END DO
END DO

CLOSE (9)
CLOSE (10)

STOP

100 FORMAT (’The date ’, 2(I2, ’/’),I4, &
’ falls/fell upon a Sunday’)
101 FORMAT (’The date ’, 2(I2, ’/’),I4, &
’ falls/fell upon a Monday’)
102 FORMAT (’The date ’, 2(I2, ’/’),I4, &
’ falls/fell upon a Tuesday’) &
103 FORMAT (’The date ’, 2(I2, ’/’),I4,
’ falls/fell upon a Wednesday’)
104 FORMAT (’The date ’, 2(I2, ’/’),I4, &
’ falls/fell upon a Thursday’)
105 FORMAT (’The date ’, 2(I2, ’/’),I4, &
’ falls/fell upon a Friday’)
106 FORMAT (’The date ’, 2(I2, ’/’),I4, &
’ falls/fell upon a Saturday’)

END PROGRAM day_of_week

ELSE IF statement

• may appear only between a block IF & END IF
• never has a matching END IF statement
This program determines the day of the week for any given date

The date 1/1/1900 falls/fell upon a Monday

The date 11/11/1918 falls/fell upon a Monday

The date 3/12/1924 falls/fell upon a Wednesday

The date 9/1/1939 falls/fell upon a Friday

The date 7/4/1963 falls/fell upon a Thursday

The date 2/29/1964 falls/fell upon a Saturday

The date 5/11/1974 falls/fell upon a Saturday

The date 3/14/1977 falls/fell upon a Monday

Sample input and corresponding output for the day_of_week program are shown here. This is a good example of the use of many of the Fortran features covered to this point in the course. Be sure you understand it.
CASE Construct

Like the block IF construct, the CASE construct allows to select a block of statements (or none) for execution from a set of blocks of statements depending on the value of a case selection expression. The syntax for the CASE construct is:

```
SELECT CASE (case_selection_expression)
    CASE (case_selector)
        block_of_statements
    CASE (case_selector)
        block_of_statements
    :
    :
END SELECT
```

The `case_selection_expression` must be of integer, character, or logical type but cannot be of real type. A CASE statement can take the form:

```
CASE (case_selector)
or
CASE DEFAULT
```

The `case_selector` determines which block of statements will be executed. The form of a `case_selector` can be:

- `case_value`
- `low_value:high_value`
- `low_value:high_value`
- a list of any combination of the above

The block of code following the CASE DEFAULT statement is executed, if no values or value ranges match with the value of the `case_selection_expression`.

Note: CASE DEFAULT is optional
CASE Construct: Examples

Example 1:

SELECT CASE (class_code)
CASE (1)
  WRITE(*,*) "Freshman"
CASE (2)
  WRITE(*,*) "Sophomore"
CASE (3)
  WRITE (*,*) "Junior"
CASE(4)
  WRITE (*,*) "Senior"
CASE DEFAULT
  WRITE (*,*) "Invalid Class code"
END SELECT

Example 2:

SELECT CASE (mid_term_score+final_score)
CASE(90:)
  grade_point = 4.0
  WRITE (*,*) " Your course grade is A"
CASE (80:89)
  grade_point = 3.0
  WRITE (*,*) " Your course grade is B"
CASE (70:79)
  grade_point = 2.0
  WRITE (*,*) "Your course grade is C"
CASE (60:69)
  grade_point = 1.0
  WRITE (*,*) "Your course grade is D"
CASE (:59)
  grade_point = 0.0
  WRITE (*,*) "Grade is not everything in life"
END SELECT