

Computers in Engineering COMP 208

Selection
Michael A. Hawker



Selection

- * As we have seen:
 - Every programming language must provide a selection mechanism that allows us to control whether or not a statement should be executed
 - * This will depend on whether or not some condition is satisfied (such as the discriminant being positive)



FORTRAN Selection

- Used to select an alternative sequence of statements
- The keywords separate the block statements
- * Has Additional Forms to Provide More Control

IF-THEN-END IF

* Syntax:

```
IF (logical expression) THEN block of statements, s_1 END IF
```

* Semantics:

- Evaluate the logical expression
- If it evaluates to .TRUE. execute s₁ and then continue with the statement following the END IF
- ★ If the result is .FALSE. skip s₁ and continue with the statement following the END IF



Examples of IF-THEN-END IF

 $absolute_x = x$

IF (x < 0.0) **THEN** absolute_x = -x

END IF

Examples of IF-THEN-END IF

```
INTEGER :: a, b, min
READ(*,*) a, b
min = a
```

```
IF (a > b) THEN min = b
```

END IF

WRITE(*,*) "The smaller of ", & a, " and ", b, " is ", min

Logical IF

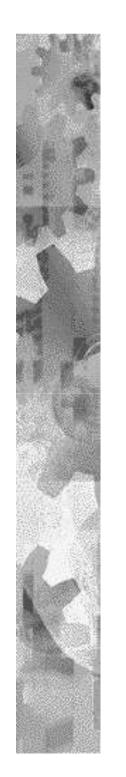
- * An even simpler form is sometimes useful.
- * Syntax:

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

- * Semantics:
 - This statement is equivalent to

```
IF (logical expression) THEN
    single-statement
END IF
```

The single-statement cannot be an IF or we might end up with an ambiguous statement



Examples of Logical IF

 $absolute_x = x$

IF (x < 0.0) absolute_x = -x

Examples of Logical IF

```
INTEGER :: a, b, min
```

$$READ(*,*)$$
 a, b

$$min = a$$

IF
$$(a > b) \min = b$$



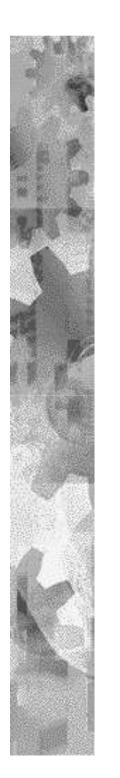
What's Going On?

- What is a "logical expression"?
- *Where do the values .TRUE. and .FALSE. come from?
- * What are those periods around the words true and false?



Logical Data Type

- FORTRAN has a LOGICAL data type, just like it has INTEGER and REAL types
- Each type has its associated values
- * There are only two values in the type LOGICAL, .TRUE. and .FALSE.
- * To enable the compiler to distinguish these values from variables, we represent them with periods around the words



Logical Data Type

* We can declare variables of type LOGICAL

```
LOGICAL :: positive x, condition
```

* We can assign values to them

```
condition = .TRUE.
positive x = x > 0
```

* These variables can only take on one of the two values of type logical



Logical Expressions

- * Logical expressions, such as those that appear in IF statements, return a logical value
- * That is, they are expressions which evaluate to .TRUE. or .FALSE.
- * We have operators that return logical values.



Relational Operators

 Relational operators compare two values and return the result .TRUE. or .FALSE.

 Relational operators are of lower precedence than all arithmetic operators

$$2 + 7 >= 3 * 3 \rightarrow .TRUE.$$

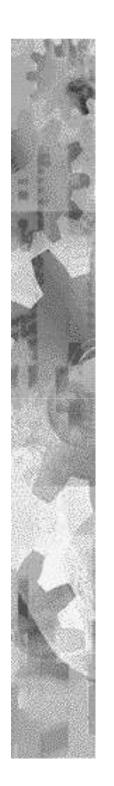
There is no associativity

$$a < b < c \rightarrow illegal$$



== or = ?

- Note that == is the FORTRAN (and C) syntax for a relational operator meaning "is equal to"
- The expression x == y has the value .TRUE. if x and y are equal and .FALSE. if x and y are not equal
- A single equal sign (=) is the FORTRAN (and C) syntax for assignment
- The statement x = y means assign the value of y to the variable x



- == or = ?
- In FORTRAN you will get an error message if you use either operator incorrectly
- * When we study C, we will see a program can still work but give incorrect results if you confuse these operators



The Missing ELSE

- * There is another more complex selection mechanism we can use
- ★ The IF-THEN-ELSE-END IF form allows us to choose between two alternatives
- * It allows us to choose whether or not to perform a one set of actions or another
- * We either perform one action or another before we continue

```
! Solve Ax^2 + Bx + C = 0
PROGRAM QuadraticEquation
  IMPLICIT NONE
! **** Same old declarations and set up ****
! compute the square root of discriminant d
 d = b*b - 4.0*a*c
                              ! is it solvable?
  IF (d \ge 0.0) THEN
    d = SORT(d)
    root1 = (-b + d) / (2.0*a)
    root2 = (-b - d)/(2.0*a)
    WRITE(*,*) "Roots are ", root1, " and ", root2
 ELSE
                                 ! complex roots
    WRITE(*,*) "There is no real root!"
    WRITE (*, *) "Discriminant = ", d
 END IF
END PROGRAM QuadraticEquation
```

IF-THEN-ELSE-END IF * Syntax:

```
IF (logical expression) THEN block of statements, \mathbf{s}_1 ELSE block of statements, \mathbf{s}_2 END IF
```

Semantics:

- Evaluate the logical expression
- If it evaluates to .TRUE. execute s₁ and then continue with the statement following the END IF
- If it evaluates to .FALSE. execute s₂ and continue with the statement following the END IF



Is a Number Even or Odd?

```
IF (MOD(number, 2) == 0) THEN
    WRITE(*,*) number, " is even"
ELSE
    WRITE(*,*) number, " is odd"
END IF
```



Is A Number Even or Odd? (alternate)

```
IF (number/2*2 == number) THEN
    WRITE(*,*) number, " is even"
ELSE
    WRITE(*,*) number, " is odd"
END IF
```

Find Absolute Value

```
REAL :: x, absolute_x
x = ...

IF (x >= 0.0) THEN
  absolute_x = x

ELSE
```

 $absolute_x = -x$

END IF

WRITE(*,*) "The absolute value of ",&x, " is ", absolute x

Which value is smaller?

```
INTEGER :: a, b, min
READ(*,*) a, b

IF (a <= b) THEN
    min = a

ELSE
    min = b</pre>
```

END IF

WRITE(*,*) "The smaller of ", a, & " and ", b, " is ", min



Quadratic Roots Revisited

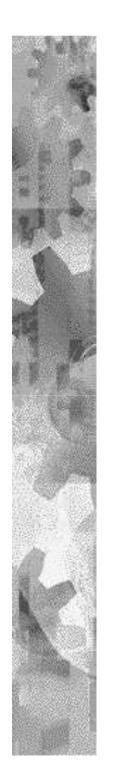
- The problem of finding the roots of a quadratic is a bit more complicated than we have been assuming
- If the discriminant is zero there is only a single root

```
! Solve Ax^2 + Bx + C = 0
 Detect complex roots and repeated roots.
PROGRAM QuadraticEquation
  IMPLICIT NONE
! **** same old declarations and setup statements omitted ****
  d = b*b - 4.0*a*c
  IF (d > 0.0) THEN! distinct roots?
     d = SORT(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)
                                 ! complex roots
     ELSE
        WRITE(*,*) 'There is no real root!'
        WRITE(*,*) 'Discriminant = ', d
     END IF
  END IF
END PROGRAM QuadraticEquation
```



IF-THEN-ELSE IF-END IF

- The nested IF statements in the last example are a bit complicated
- * When we use IF to select between several (not just two) alternatives, we end up with more than a single END IF, one for each of the branches
- Let's simplify this



Syntax of IF-THEN-ELSE IF-END IF

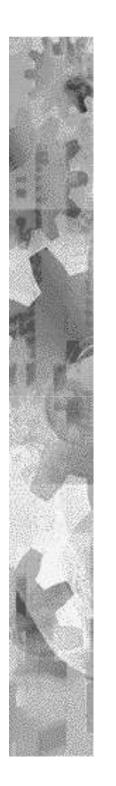
```
IF (logical-exp, e<sub>1</sub>) THEN
  statement block, s<sub>1</sub>
ELSE IF (logical-exp, e2) THEN
  statement block, s<sub>2</sub>
ELSE IF (logical-exp, e3) THEN
  statement block, s3
ELSE
  statement block, s
END IF
```



Semantics of IF-THEN-ELSE IF-END IF

- Evaluate e1
- ★ If the result is .TRUE., execute s₁ and go on to the statement that follows the END IF
- ★ If the result is .FALSE., evaluate e₂. If it is .TRUE., execute s₂ and go on to the statement that follows the END IF
- * If the result of e_2 is false, repeat this process.
- If none of the expressions e_i evaluate to .TRUE., execute s_e and then go on to the statement that follows the END IF

```
! Solve Ax^2 + Bx + C = 0
 Detect complex roots and repeated roots.
PROGRAM QuadraticEquation
  IMPLICIT NONE
! **** same old declarations and setup statements omitted ****
  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)
                             ! complex roots
  ELSE
     WRITE(*,*) 'There is no real root!'
     WRITE(*,*) 'Discriminant = ', d
  END IF
END PROGRAM QuadraticEquation
```



Quadratic Roots Final Version

- The problem of finding the roots of a quadratic has some more complications
- ★ What if a is zero. Dividing by 2.0*a would cause an error.
- # If a is zero, the equation is linear, not quadratic
- # If a and b are zero but c isn't there is no solution

```
Solve Ax^2 + Bx + C = 0
   Now, we are able to detect the following:
   (1) unsolvable equation
     (2) linear equation
     (3) quadratic equation
        (a) distinct real roots
       (b) repeated root
        (c) no real roots
PROGRAM QuadraticEquation
   IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d
  REAL :: root1, root2
! read in the coefficients a, b and c
  READ(*,*) a, b, c
```

```
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
                                 ! linear equation
  ELSE
     WRITE(*,*) 'This is linear equation, root = ', -c/b
  END IF
                                 ! ok, we have a quadratic equation
ELSE
  d = b*b - 4.0*a*c
  IF (d > 0.0) THEN
                    ! distinct root
     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)
                                 ! complex roots
  ELSE
     WRITE(*,*) 'There is no real root!'
    WRITE(*,*) 'Discriminant = ', d
  END IF
END IF
END PROGRAM QuadraticEquation
```



What Day is Tomorrow?

- * Here is a new problem to solve.
 - Given today's date (day,month,year)
 - * Compute and output tomorrow's date
- * What's the problem?
- If the date is the last day of the month, we have to update the day and month
- If it is the last day of the year, we also have to update the year

First Validate the Data

```
PROGRAM nextday
    IMPLICIT NONE
    INTEGER :: day, month, year
    INTEGER :: lastday
    WRITE (*,*) "Please enter the date, day month and year:"
    READ (*,*) day, month, year

! validate month

IF (month < 1 .OR. month > 12) THEN
    WRITE (*,*) "Invalid month"
    STOP
END IF

! Validation of year and day omitted to save space
```

Compute the last day of the month

Compute Tomorrow's Date

```
! The usual case
  day = day + 1
   ! Handling the end of the month or year
   IF (day > lastday) THEN
    day = 1
    month = month + 1
    IF (month > 12) THEN
      month = 1
      year = year + 1
    END IF
  END IF
  WRITE (*,*) day, month, year
END PROGRAM nextday
```

Logical Operators

More complex logical expressions can be formed using logical operators

The Logical Operators listed in order of decreasing precedence are:

```
.NOT.
.AND. (or &&)
.OR. (or ||)
.EQV. (or ==), .NEQV. (or /=)
```

The precedence of all logical operators is lower than all relational operators

They all associate from left to right



Area of a Triangle

Heron's formula gives the area of a triangle in terms of the lengths of its sides.

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

Where a, b, and c are the lengths of the sides and

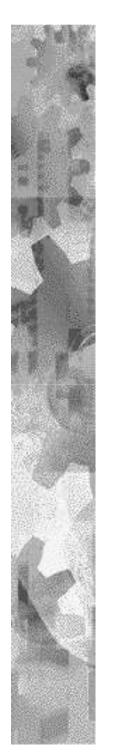
$$s = \frac{a+b+c}{2}$$



Area of a Triangle

★ To use it, we must make sure that the sides form a triangle.

- * There are two necessary and sufficient conditions:
 - * All side lengths must be positive
 - The sum of any two sides must be greater than the third



Area of a Triangle (program preamble)



Area of a Triangle (main body of program)

END PROGRAM HeronFormula