McGill

COMP 208 – Computers in Engineering

COMP 208
Computers in Engineering

Lecture 05

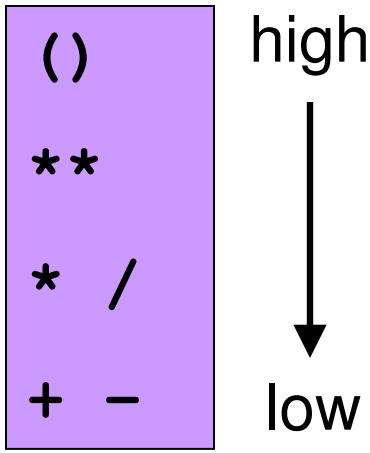
Jun Wang
School of Computer Science
McGill University
Fall 2007

Review

- Arithmetic expression
  - arithmetic operations on operands

```
+   -   *   /   **
```

  - Precedence of arithmetic operators
    - ** is right associative



- Assignment statement

```
variable = expression
```

Review

- Data types
  - FORTRAN has 5 intrinsic (built-in) data types:
    - `INTEGER`, `REAL`, `COMPLEX`, `LOGICAL`, `CHARACTER`

- REAL
  - decimal form: 1.23, .123, 123.
  - exponential form: 1.0E-3

Back to The Speed of Light

- How long does it take light to travel from the sun to earth?
- The result we got was a decimal number of minutes
- We'd rather have the number of minutes and seconds

Minutes and Seconds

```
PROGRAM light_travel
  IMPLICIT NONE
  REAL :: light_minute, distance, time
  REAL :: light_year = 9.46 * 10.0**12
  INTEGER :: minutes, seconds

  light_minute = light_year/(365.25 * 24.0 * 60.0)
  distance = 150.0 * 10**6
  time = distance / light_minute

  minutes = time
  seconds = (time - minutes) * 60

  write (*,*) "Light from the sun takes ", minutes, &
         " minutes and ", seconds, " seconds to reach earth."

END PROGRAM light_travel
```

Integer Numbers (literals)

- Integers are whole numbers represented using 32 bits (or sometimes 16 or even 64 bits)
- For 32 bit numbers whole numbers with up to 10 digits can be represented ($-2^{31} \sim 2^{31}-1$, or -2147483648 ~ 2147483647)

```
0
-987
+17
1234567890
```

Integer Arithmetic

- The result of performing an operation on two integers is an integer
- This may result in some unexpected results since the decimal part is truncated

```
12/4 → ? 3
13/4 → ? 3
 1/2 → ? 0
 2/3 → ? 0
```

Some Simple Examples

```
1 + 3 → 4

1.23 - 0.45 → 0.78

3 * 8 → 24

8.4/4.2 → 2.0 (not 2)

-5**2 → -25 (not 25 -- precedence)

3/5 → 0 (not 0.6)

3./5. → 0.6
```

Another Example

```
2 * 4 * 5 / 3 ** 2
    --> 2 * 4 * 5 / 3 ** 2
    --> 2 * 4 * 5 / 9
    --> 2 * 4 * 5 / 9
    --> 8 * 5 / 9
    --> 8 * 5 / 9
    --> 40 / 9
    --> 4
```

The result is 4 rather than 4.444444 since the operands are all integers.

Mixed Mode Assignment

In the speed of light example, we assigned an real value to an integer variable

```
minutes = time
```

The value being assigned is converted to an integer by truncating (not rounding)

When assigning an integer to a real variable, the integer is first converted to a real (the internal representation changes)

Mixed Mode Expressions

In the speed of light example, we subtracted the integer value, minutes, from the real value, time

```
seconds = (time – minutes) * 60
```

If an operation involves an integer and a real, the integer is first converted to a real and then the operation is done.

The result is real.

The example has two arithmetic operations, an assignment and forces two type conversions

Mixed mode expressions - summary

- In an arithmetic expression, if both operands are INTEGER, the operation is INTEGER and result is INTEGER
- If one operand is REAL, the other is INTEGER, the INTEGER is converted to REAL, and the result is REAL
- When an INTEGER is assigned to a REAL variable, it is converted to REAL
- When a REAL variable is assigned to an INTEGER variable, the fractional part is discarded

Mixed Mode Examples

```
1 + 2.5 → 1.0 + 2.5 → 3.5

1/2.0 → 1.0/2.0 → 0.5

2.0/8 → 2.0/8.0 → 0.25

-3**2.0 → -3.0**2.0 → -9.0

4.0**(1/2) → 4.0**0 → 4.0**0.0 →
      1.0 (since 1/2 → 0)
```

Example of mixed mode arithmetic operations

```
25.0 ** 1 / 2 * 3.5 ** (1 / 3)
   → 25.0 ** 1 / 2 * 3.5 ** (1 / 3)
   → 25.0 ** 1 / 2 * 3.5 ** 0
      (25.0 ** 1.0)
   → 25.0 / 2 * 3.5 ** 0
   → 25.0 / 2 * 1.0
   → 12.5 * 1.0
   → 12.5
```

Your turn: midterm 05 question

What does the following program output?

```
PROGRAM midterm
  REAL :: A, B, C
  INTEGER :: I, J, K

  A = 3.5
  I = A
  J = 5.25
  K = I*2
  B = A*I
  C = J/3

  WRITE (*,*) A,B,C,I,J,K

END PROGRAM midterm
```

A. 3.5 10.5 1. 3 5 7
B. 3.5 12.25 1. 3 5 6
C. 3.5 10.5 1. 3 5 6
D. 3.5 10.5 1.75 3 5 6
E. None of the above

Something's Not Right Here

```
program light_travel
   implicit none
   real :: light_minute, distance, time
   real :: light_year = 9.46 * 10 ** 12

   light_minute = light_year/(365.25 * 24.0 * 60.0)
   distance = 150.0 * 10**6
   time = distance / light_minute

   write (*,*) "Light from the sun takes ", time, &
               "minutes to reach earth."

end program light_travel
```

What Happened?

Look at this assignment:

```
light_year = 9.46 * 10**12
```

Precedent rules tell us that `10**12` is evaluated first

Type rules tell us that the result is an integer

Integers can only have about 10 digits, not 13

How do we fix it?

Let's change

```
light_year = 9.46 * 10**12
```

to

```
light_year = 9.46 * 10.0**12
```

That works, but why?

Selection in Fortran

conditional actions

Why selections?

- Programs so far all have 1 sequence of statements, which are executed unconditionally
- Real problems often require actions taken based on certain conditions
- A simple program:
  - 1. Ask user to input an integer
  - 2. If it is odd, print “odd”, otherwise, print “even”

Back to roots of a quadratic

- Let's have another look at our program for finding the roots of a quadratic equation
- We used the classic formula that involved finding the square root of the discriminant

- What if the discriminant is negative?

```
! ------------------------------------------------
! Solve Ax^2 + Bx + C = 0
! ------------------------------------------------
PROGRAM QuadraticEquation
  IMPLICIT NONE

  REAL :: a, b, c
  REAL :: d
  REAL :: root1, root2

! read in the coefficients a, b and c
  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the square root of discriminant d

  d = SQRT(b*b - 4.0*a*c)

! solve the equation

  root1 = (-b + d)/(2.0*a) ! first root
  root2 = (-b - d)/(2.0*a) ! second root

! display the results

  WRITE(*,*) 'Roots are ', root1, ' and ', root2

END PROGRAM QuadraticEquation
```

A Run Time Error

If the discriminant is negative, attempting to take the square root would cause an error during execution

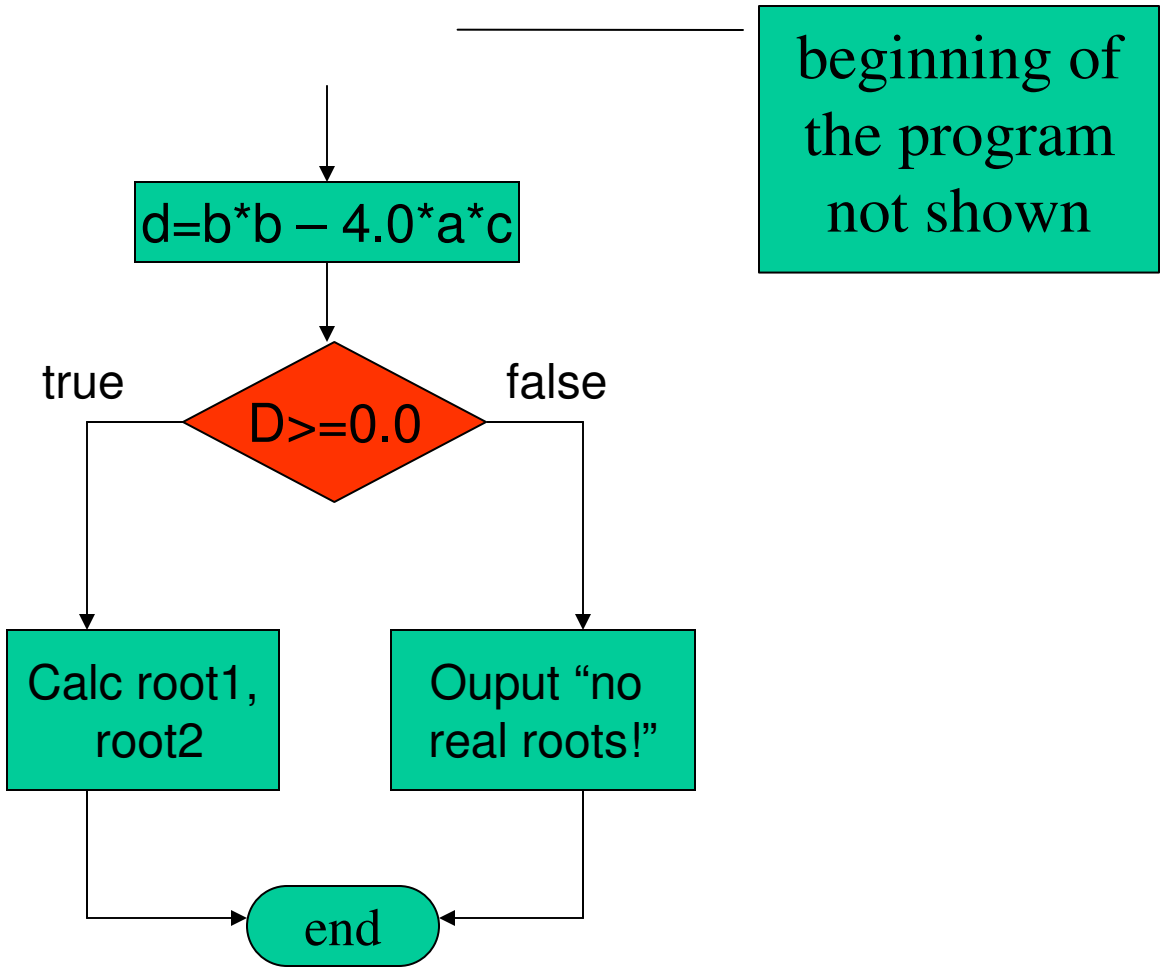
This is called a run time error and the program would either abort or return a meaningless result

Selection

What can we do?

- 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)

Quadratics example: flow-chart



```
! ------------------------------------------------
!   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
  IF (d >= 0.0) THEN                  ! is it solvable?
     d     = SQRT(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
```

FORTRAN Selection

Used to select between two alternative sequences of statements.

The keywords delineate the statement blocks.

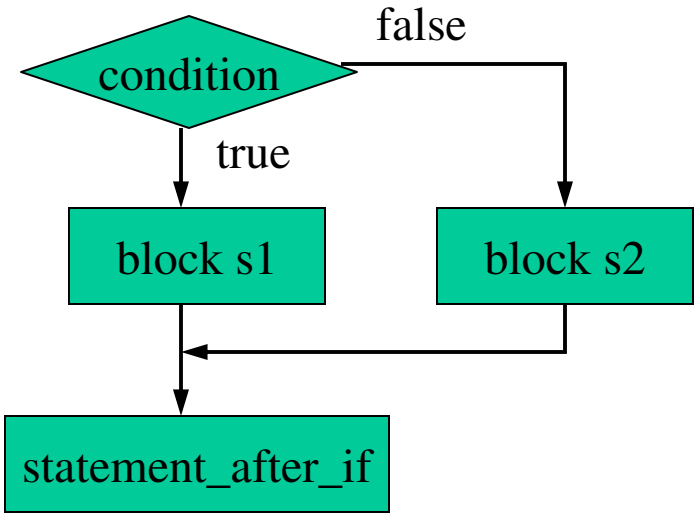
Syntax:

```
IF (logical-expression) THEN
    first statement block s1
ELSE
    second statement block s2
END IF
```

Semantics of IF…THEN…ELSE…END IF

- Evaluate the logical expression. It can have value `.TRUE.` or value `.FALSE.`
- If the value is `.TRUE.`, evaluate s_1, the first block of statements
- If the value is `.FALSE.`, evaluate s_2, the second block of statements
- After finishing whichever of s_1 or s_2 that was chosen, execute the next statement following the END IF

Flow-chart for selection



What's Going On?

- What is a "logical expression" ?
- an expression that evaluates to either `.TRUE.` or `.FALSE.`
  - a LOGICAL literal: `.TRUE.`, `.FALSE.`
  - a LOGICAL variable
  - a relational expression
  - more complex expression consisting of the above

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.`
- Recall a data type defines:
  - the range of values it can assume
  - the valid operations for that type

C does not have logical type; it uses integers instead

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

Relational Operators

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

```
<   <=   >   >=    ==   /=
```

C uses != for "not equal"

- Relational operators are of lower precedence than all arithmetic operators

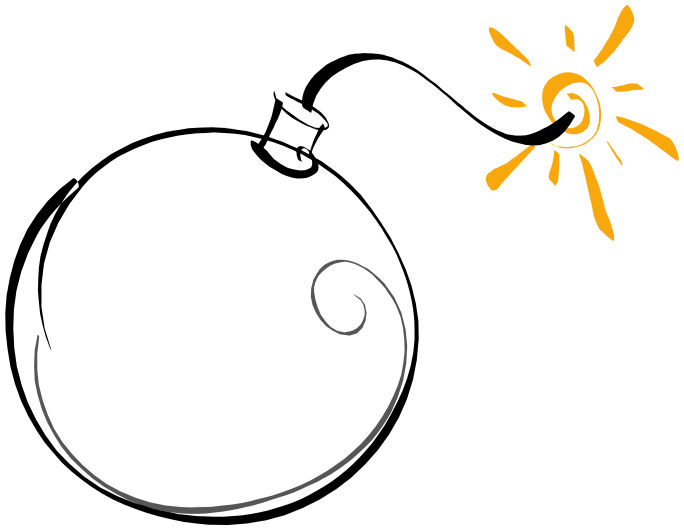
```
2 + 7 >= 3 * 3 → .TRUE.
```

Same as `(2 + 7) >= (3 * 3)`

- There is no associativity

```
a < b < c → illegal
```

Danger Lurks



== or = ?

Note that **==** is the FORTRAN (and the 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

Later on, when we study C, we will C that the program could work and give totally incorrect results if you confuse these operators

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

The Missing ELSE

The IF-THEN-ELSE-END IF form allows us to choose between two alternatives

There is another simpler selection mechanism that can sometimes be used

It allows us to choose whether or not to perform a single block of actions

We either perform the actions and go on, or skip them and go on

IF-THEN-END IF

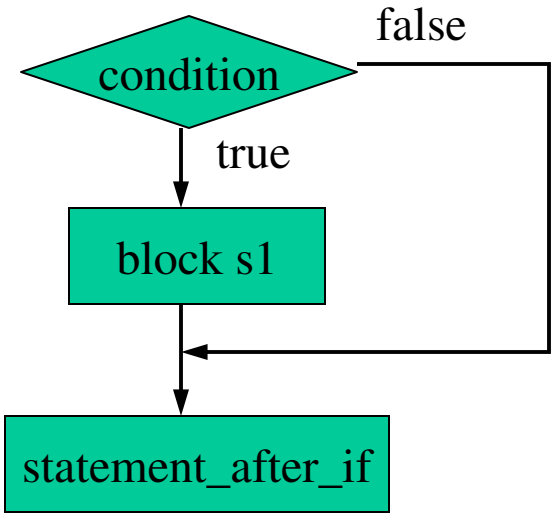
Syntax:

```
IF (logical-expression) THEN
    first statement block s1
END IF
```

Semantics:

- Evaluate the logical expression
- If it evaluates to .TRUE. execute s_1 and then continue with the statement following the END IF
- If the result is .FALSE. skip s_1 and continue with the statement following the END IF

Flow-chart for selection



Examples of IF-THEN-END IF

```
absolute_x = x

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

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

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
```

Must be logically on a single line.

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

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

Examples of Logical IF

```
INTEGER :: a, b, min

READ(*,*) a, b

min = a

IF (a > b) min = b

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     = 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 root!`
         WRITE(*,*)  'Discriminant = ', d
      END IF
   END IF
END PROGRAM  QuadraticEquation
```