McGill

COMP 208 – Computers in Engineering

COMP 208
Computers in Engineering

Lecture 06

Jun Wang
School of Computer Science
McGill University
Fall 2007

Review

- Mixed-mode arithmetic expression
  - if an operation involves 1 real and 1 integer, the integer is converted to real
- Mixed-mode assignment
  - when a real is assigned to an integer variable, the fractional part is discarded

- IF statement
  - Logical IF

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

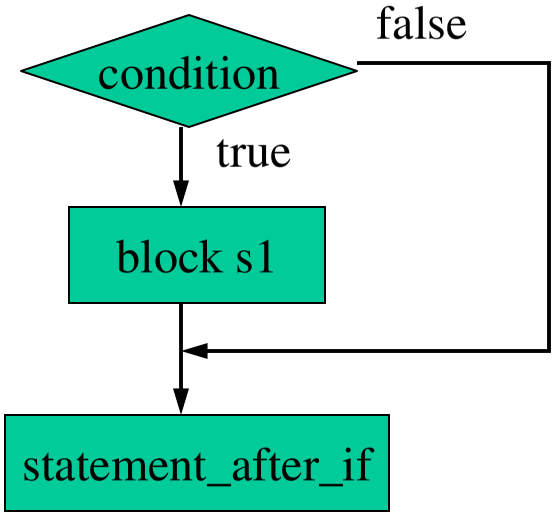
equivalent to

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

Review

- IF-THEN-END IF

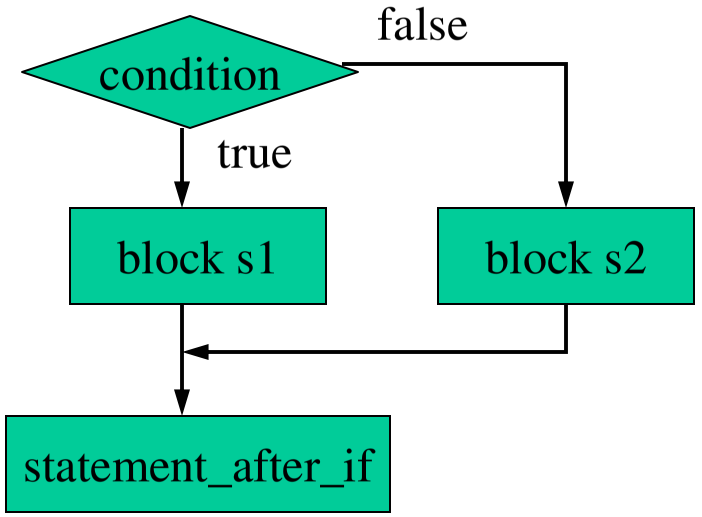
```
IF (logical-expression) THEN
    first statement block s1
END IF
```



Review

- IF-THEN-ELSE-END IF

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



Review

- Logical expression: any thing that evaluates to a LOGICAL value
  - logical literal: `.TRUE.` or `.FALSE.`
  - logical variable
  - relational expression

- Relational operators:

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

Your turn: an exercise on IF

- Task: get 2 integers from user; print out the bigger number, and the difference of the 2.
  - 1. print a message to ask user to enter a number
  - 2. get the number from user
  - 3. print a message to ask user to enter a number
  - 4. get the number from user
  - 5. use an if-else statement to print 2 lines of message like the following (assuming the user entered 5 and 1):

```
The bigger number is: 5
The difference of the 2 is: 4
```

Nested-IF

- Both the THEN part and the ELSE part of an IF statement can have another IF statement -- a nested IF statement

```
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
```

Nested-IF

```
IF (n == 1) THEN
    WRITE(*,*) "One"
ELSE
    IF (n == 2) THEN
        WRITE(*,*) "Two"
    ELSE
        IF (n == 3) THEN
            WRITE(*,*) "Three"
        ELSE
            WRITE (*,*) "Unknown"
        END IF
    END IF
END IF
```

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

This can be simplified

Syntax of IF-THEN-ELSE IF-END IF

```
IF (logical-exp, e1) THEN
  statement block, s1
ELSE IF (logical-exp, e2) THEN
  statement block, s2
ELSE IF (logical-exp, e3) THEN
  statement block, s3
  . . . . . .
ELSE
  statement block, se
END IF
```

The last ELSE part is not required.

Semantics of IF-THEN-ELSE IF-END IF

Evaluate e_1

If the result is .TRUE., execute s_1 and go on to the statement that follows the END IF

If the result is .FALSE., evaluate e_2. If it is .TRUE., execute s_2 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)
   ELSE                               ! complex roots
      WRITE(*,*)  'There is no real root!`
      WRITE(*,*)  'Discriminant = ', d
   END IF
END PROGRAM  QuadraticEquation
```

IF-THEN-ELSE IF-END IF

```
IF (n == 1) THEN
    WRITE(*,*) "One"
ELSE IF (n == 2) THEN
    WRITE(*,*) "Two"
ELSE IF (n == 3) THEN
    WRITE(*,*) "Three"
ELSE
    WRITE (*,*) "Unknown"
END IF
```

only one END IF

Quadratic Roots Final Version

- The problem of finding the roots of a quadratic has some more complications
- What if a is zero. Dividing by 2a 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 (a,b==0, c/=0)
!    (2) linear equation (a==0, b/=0)
!    (3) quadratic equation (a/=0)
!        (a) distinct real roots (d > 0)
!        (b) repeated root (d == 0)
!        (c) no real roots (d < 0)
! ------------------------------------------------

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
   ELSE                                 ! linear equation
      WRITE(*,*)  'This is linear equation, root = ', -c/b
   END IF
ELSE                                    ! ok, we have a quadratic equation
   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)
   ELSE                                 ! complex roots
      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

1. 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
```

The STOP statement terminates the program.

2. Compute the last day of the month

```
IF (month == 1 .OR. month == 3 .OR. month == 5 .or.
       month == 7 .OR. month == 8 .OR. month == 10 .or.
       month == 12) THEN
  lastday =31
ELSE IF (month == 4 .OR. month == 6 .OR. month == 9 .OR.
            month == 12) THEN
  lastday =30
ELSE IF ((mod(year,4) == 0 .AND. mod(year,100) /= 0) .OR.
             mod(year,400) == 0) THEN
  lastday = 29  ! February of leap year
ELSE
  lastday = 28
END IF
```

3. 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.
.EQV., .NEQV.
```

- The precedence of all logical operators is lower than all relational operators
- They all associate from left to right

Logical NOT

- also called logical negation
- unary operator
- If a Logical expression `exp` is `.TRUE.` , then `.NOT. exp` is `.FALSE.` ; if `exp` is `.FALSE.` , then `.NOT. exp` is `.TRUE.` .

exp	.NOT. Exp
.TRUE.	.FALSE.
.FALSE.	.TRUE.

truth table

```
LOGICAL :: success
...
IF (.NOT. success) THEN
  WRITE (*,*) "Houston, we have a problem!"
END IF
```

```
IF (.NOT. (x >= 0)) THEN
  WRITE (*,*) "x is negative!"
END IF
```

Logical AND

- binary operator
- evaluates to `.TRUE.` if and only if both operands are `.TRUE.`

exp1	exp2	exp1 .AND. exp2
.FALSE.	.FALSE.	.FALSE.
.FALSE.	.TRUE.	.FALSE.
.TRUE.	.FALSE.	.FALSE.
.TRUE.	.TRUE.	.TRUE.

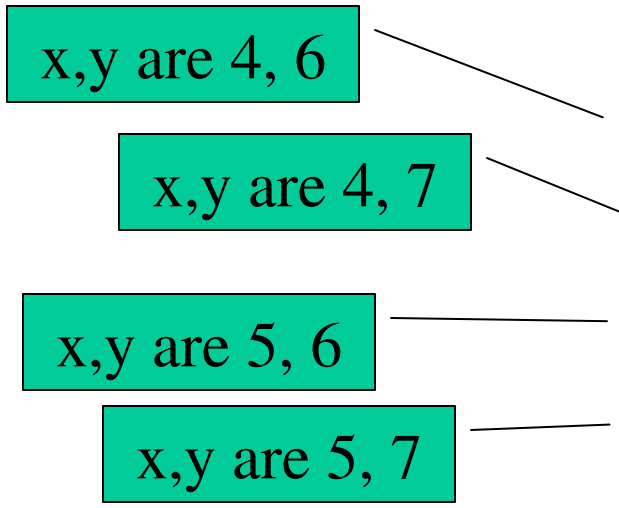
truth table

```
IF (x >= 0 .AND. x <= 10) THEN
  WRITE (*,*) "x is between 0 and 10"
END IF
```

Logical AND example

```
IF (mod(x,2) == 1 .AND. mod(y,2) == 1) THEN
  WRITE(*,*) "Both x, y are odd numbers"
END IF
```

- exp1: `mod(x,2) == 1`
- exp2: `mod(y,2) == 1`
- exp: `mod(x,2) == 1 .AND. mod(y,2) == 1`



exp1	exp2	exp
.FALSE.	.FALSE.	.FALSE.
.FALSE.	.TRUE.	.FALSE.
.TRUE.	.FALSE.	.FALSE.
.TRUE.	.TRUE.	.TRUE.

Logical OR

- binary operator
- evaluates to `.TRUE.` if one of the operands is `.TRUE.`

exp1	exp2	exp1 .OR. exp2
.FALSE.	.FALSE.	.FALSE.
.FALSE.	.TRUE.	.TRUE.
.TRUE.	.FALSE.	.TRUE.
.TRUE.	.TRUE.	.TRUE.

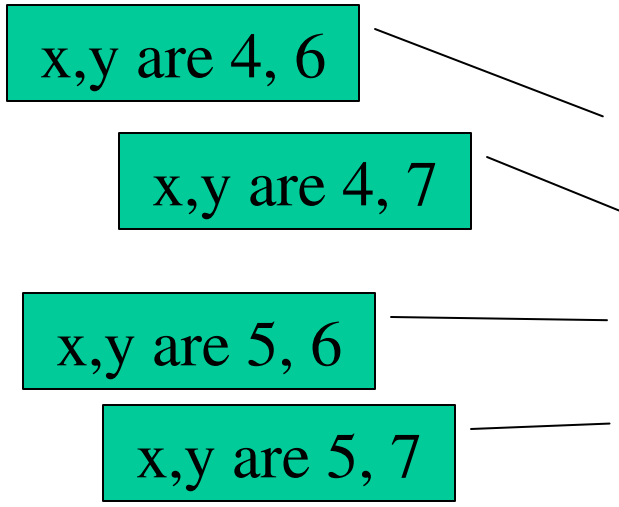
truth table

```
IF (x < 0 .OR. x > 10) THEN
  WRITE(*,*) "x is NOT between 0 and 10"
END IF
```

Logical OR example

```
IF (mod(x,2) == 1 .OR. mod(y,2) == 1) THEN
  WRITE(*,*) "either one or both are odd numbers"
END IF
```

- exp1: `mod(x,2) == 1`
- exp2: `mod(y,2) == 1`
- exp: `mod(x,2) == 1 .AND. mod(y,2) == 1`



exp1	exp2	exp
.FALSE.	.FALSE.	.FALSE.
.FALSE.	.TRUE.	.TRUE.
.TRUE.	.FALSE.	.TRUE.
.TRUE.	.TRUE.	.TRUE.

Operator precedence

High

Low

arithmetic operators (+, -, *, /, **)

relational operators (>, >=, <, <=, ==, /=)

.NOT.

.AND.

.OR.

assignment operator (=)

```
IF (.NOT. x > 0)  //x <= 0
IF (.NOT. (x > 0)) //same as above, but clearer
IF (x > 0 .AND. X < 10) //same as: IF ((x>0) .AND. (x<10))
IF (x < 0 .OR. x > 2 .AND. x ** 2 < 20) // x is less than 0
                    // OR x>2 and x square is less than 20
IF ((x < 0 .OR. x > 2) .AND. x ** 2 < 20) // x is less than 0
                    // or greater than 2 AND x square is less than 20
```

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:

1. All side lengths must be positive
2. The sum of any two sides must be greater than the third

Area of a Triangle (program preamble)

```
! ------------------------------------------------
! Compute the area of a triangle using Heron's formula
! ------------------------------------------------

PROGRAM  HeronFormula
   IMPLICIT  NONE

   REAL     :: a, b, c                ! three sides
   REAL     :: s                      ! half of perimeter
   REAL     :: Area
   LOGICAL  :: Cond_1, Cond_2
   READ(*,*)  a, b, c
```

Area of a Triangle (main body of program)

```
   Cond_1 = (a > 0.) .AND. (b > 0.) .AND. (c > 0.0)
   Cond_2 = (a+b > c) .AND. (a+c > b) .AND. (b+c > a)
   IF (Cond_1 .AND. Cond_2) THEN
      s    = (a + b + c) / 2.0
      Area = SQRT(s*(s-a)*(s-b)*(s-c))
      WRITE(*,*) "Triangle area = ", Area
   ELSE
      WRITE(*,*) "ERROR: this is not a triangle!"
   END IF

END PROGRAM  HeronFormula
```

Data Type Character (A Brief Digression)

We have seen character string constants in examples.

```
"Hello World"
```

FORTRAN also allows us to declare variables that can hold character string values

```
CHARACTER(LEN=5) :: message_1
CHARACTER(LEN=20) :: message_2
```

We can assign values to these variables.

```
message_1 = "Hello World"
message_2 = "Hello World"
```

Data Type Character (A Brief Digression)

What happens if we assign values that don't match the declared length

```
CHARACTER(LEN=5) :: message_1
CHARACTER(LEN=20) :: message_2
message_1 = "Hello World"
message_2 = "Hello World"
```

If the length is too short, the string is truncated

```
message_1 contains "Hello"
```

If it is too long it is padded with extra blanks

```
message_2 contains "Hello World          "
```

Operations on Character Strings

- Comparison using relational operators
  - The ordering for comparison is called lexicographic (or dictionary) ordering
  - A string is less than another if it would come first in the dictionary
- Concatenation (//)
  - We can join two strings together by concatenating them

```
CHARACTER(len=21) :: instructor
CHARACTER(len=8) :: surname
surname = "Friedman"
instructor = "Prof. " // "Nathan " // surname
```

McGill

COMP 208 – Computers in Engineering

Over and Over and Over Again

Repetition

Nathan Friedman

Fall, 2007

Repetition

- To fully take advantage of the speed of a computer, we must be able to instruct it to do a lot of work
- The program must be relatively short or it would take us too long to write
- To get the computer to do a lot of work, we must be able tell it to do some computations many times, perhaps with different data values each time

Integer division

1. Get 2 numbers: numerator, denominator
2. Compute and output numerator/denominator

```
WRITE (*,*) "Enter denominator:"
READ (*,*) denominator

IF (denominator == 0) THEN
  WRITE (*,*) "The denominator may not be 0”
ELSE  !perform calculation
  output = nominator / denominator
  WRITE (*,*) "The result is: ”,output
END IF
! End of program
```

What if we want to give user another chance after 0 is entered as denominator?

Nested if-statement?

```
WRITE (*,*) "Enter denominator:"
READ (*,*) denominator
IF (denominator == 0) THEN
  WRITE (*,*) "The denominator may not be 0"
  WRITE (*,*) "Enter denominator: "
  READ (*,*) denominator
  IF (denominator == 0) THEN
      WRITE (*,*) "The denominator may not be 0"
  ELSE
    output = nominator / denominator;
    WRITE (*,*) "The result is: ", output
  END IF
ELSE
  output = nominator / denominator;
  WRITE (*,*) "The result is: ", output
END IF
```

How about we give user one more chance?

Nested if-statement? (cont'd)

```
WRITE (*,*) "Enter denominator:"
READ (*,*) denominator
IF (denominator == 0) THEN
  WRITE (*,*) "The denominator may not be 0"
  WRITE (*,*) "Enter denominator:"
  READ (*,*) denominator
  IF (denominator == 0) THEN
      WRITE (*,*) "The denominator may not be 0"
      WRITE (*,*) "Enter denominator:"
      READ (*,*) denominator
      IF (denominator == 0) THEN
        WRITE (*,*) "The denominator may not be 0"
      ELSE
        ! calculate and print
      END IF
  ELSE
    ! calculate and print
  END IF
ELSE
  ! calculate and print
END IF
```

What if we only proceed when denominator is not 0?

nested if won't help!
need something new!