COMP 208
Computers in Engineering

Lecture 09

Jun Wang
School of Computer Science
McGill University

Fall 2007

Review

- Arrays
  - an aggregate data type
  - all elements have the same type
  - all elements share a common name
  - elements are identified by the name of the array and an integer index, e.g. `A(4)`
    - index starts at 1
  - each element is an individual variable:

```
A(4) = 12
X = A(4) * 2
```

- Array declaration

```
type :: name(bound)
```

Review

- 3 ways to input data to an array:

```
REAL :: A(1000)
...
DO I = 1, SIZE
    READ (*,*) A(I)
END DO
```

- accepts only 1 value per line
- accepts values for part of the array, or entire array

```
REAL :: A(1000)
...
READ (*,*) (A(I), I=1,SIZE)
```

- accepts multiple values per line
- accepts values for part of the array, or entire array

```
REAL :: A(1000)
...
READ (*,*) A
```

- accepts multiple values per line
- accepts values for entire array only

Review

- Indefinite iterator (expression-controlled loop)
  - loop is terminated with the `EXIT` statement
  - loop body must contain `EXIT` statement (usually inside an IF construct)

```
DO
  statement-block, s
END DO
```

- Indefinite iterator is very flexible: the `EXIT` statement can appear anywhere in the loop body

Definite iterator -> indefinite iterator

- Any counter-controlled loop can be converted to an expression-controlled loop.

```
DO I = init, final, step-size
  statement-block, s
END DO
```

→

```
I = init
DO
  IF (I > final) EXIT
  statement-block, s
  I = I + step-size
END DO
```

- `I = init` is implied right before loop
- `IF (I > final) EXIT` is implied at the beginning of loop
- `I = I + step-size` is implied at the end of loop

In the above, we assume step-size is positive.

Example

```
!================================
! calculates 1 + 2 + 3 + 4 + 5
!================================
PROGRAM SumOf5
  IMPLICIT NONE
  INTEGER :: sum, i

  sum = 0
  DO i = 1, 5
    sum = sum + i
  END DO

  WRITE (*,*), "Result = ", sum

END PROGRAM SumOf5
```

```
!================================
! calculates 1 + 2 + 3 + 4 + 5
!================================
PROGRAM SumOf5
  IMPLICIT NONE
  INTEGER :: sum, i

  sum = 0
  i = 1
  DO
    IF (i > 5) EXIT
    sum = sum + i
    i = i + 1
  END DO

  WRITE (*,*), "Result = ", sum

END PROGRAM SumOf5
```

An exercise on loops

- Task: write a program to get an integer from user, and then calculate and print the factorial of the number.
  - 1. print a message to ask user to enter an integer number
  - 2. get the number from user
  - 3. [optional]: verify the number is greater than 0
  - 4. use a while-loop to calculate the factorial: **`1*2*3*...*n`**, where n is the input number
  - 5. print the factorial.

  - The program output should look something like this:

```
Enter a positive integer:
6
The factorial of 6 is: 720
```

Another exercise on loops

- Task: write a counter-controlled loop to calculate the number of all possible combinations of Lotto 6/49 numbers, which is given by:

$$\frac{49 * 48 * 47 * 46 * 45 * 44}{1 * 2 * 3 * 4 * 5 * 6}$$

$$= \frac{49}{1} * \frac{48}{2} * \frac{47}{3} * \frac{46}{4} * \frac{45}{5} * \frac{44}{6}$$

Solution

```
!================================
! calculates number of possible
! combos for Lotto 649
!================================
PROGRAM Lotto649
  IMPLICIT NONE
  INTEGER :: total, i, j

  total = 1
  j = 49
  DO i = 1, 6
    total = total * j / i
    j = j - 1
  END DO

  WRITE (*,*) "Result = ", total

END PROGRAM Lotto649
```

```
!================================
! calculates number of possible
! combos for Lotto 649
!================================
PROGRAM Lotto649
  IMPLICIT NONE
  INTEGER :: total, i

  total = 1
  DO i = 1, 6
    total = total * (49-i+1)/i
  END DO

  WRITE (*,*) "Result = ", total

END PROGRAM Lotto649
```

Terminating a Loop

The general DO loop will go on forever without terminating

How do we get out of it?

The **`EXIT`** statement causes execution to leave the loop and continue with the statement following the `END DO`

Sum Positive Input Values

Read real values and sum them. Stop when the input value becomes negative.

```
REAL :: x, Sum
Sum = 0.0
DO
   READ(*,*) x
   IF (x < 0) EXIT
   Sum = Sum + x
END DO
```

- Special values like this are called sentinels. (e.g. C strings)
- They are used to signal the end of something.

Exp(x)

The exponential function can be expressed as an infinite sum:

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^i}{i!} + \ldots$$

A program to approximate the value can compute a finite portion of this sum

We can sum terms until the final term is very small, say less then 0.00001 (or any other tolerance we might choose)

Compute Exp(x) (preamble)

```
! ----------------------------------------------
! Compute exp(x) for an input x using the infinite
! series of exp(x).
! ---------------------------------------------------------

PROGRAM  Exponential
   IMPLICIT  NONE

   INTEGER         :: Count                   ! # of terms used
   REAL            :: Term
   REAL            :: Sum
   REAL            :: X
   REAL            :: Tolerance = 0.00001     ! Tolerance

   WRITE (*,*) "Enter a number: "
   READ(*,*)  X

! *** rest of program is on next slide
```

Compute Exp(x) (main part of program)

```
! *** the preamble is on the previous slide

   Count = 1
   Sum   = 1.0
   Term  = X                                ! the second term is x
   DO
      IF (ABS(Term) < Tolerance)  EXIT
      Sum   = Sum + Term
      Count = Count + 1
      Term  = Term * (X / Count)            ! compute the value of next term
   END DO

   WRITE(*,*)  "After ", Count, " iterations:"
   WRITE(*,*)  "  Exp(", X, ") = ", Sum
   WRITE(*,*)  "  From EXP()   = ", EXP(X)
   WRITE(*,*)  "  Abs(Error)   = ", ABS(Sum - EXP(X))

END PROGRAM  Exponential
```

DO WHILE

DO ... WHILE loops are a special case used when a condition is to be tested at the top of a loop

This is a looping structure provided in many different programming languages

Syntax:

```
DO WHILE (logical-expression)
  statement-block, s
END DO
```

This loop is equivalent to the `while` loop in C. The `do-while` loop in C is different: it evaluates the condition at the end of the loop.

DO WHILE

Semantics:

1. Test the logical expression
2. If it evaluates to .TRUE., execute the statement block and go back to step 1.
3. If it evaluates to .FALSE., go to the statement after the END DO

DO-WHILE

DO-WHILE loops are equivalent to

```
DO
   IF .NOT.(logical expression) EXIT
   statement block s
END DO
```

Example

The DO loop of the program to compute exp(x) can be rewritten using a DO-WHILE

```
DO
   IF (ABS(Term) < Tolerance) EXIT
   Sum = Sum + Term
   Count = Count + 1
   Term = Term * (X / Count)
END DO
```

```
DO WHILE (ABS(Term) >= Tolerance)
   Sum = Sum + Term
   Count = Count + 1
   Term = Term * (X / Count)
END DO
```

Warning!

- The loop only executes if the logical expression evaluates to .TRUE.
- *If the value of this expression doesn't change, we will get an infinite loop*
- The values of variables that the logical expression depends on must be modified within the loop
- (It still might not terminate, but at least we have a chance)

Example

Using DO-WHILE loop to print 3 lines of “hello”

```
I = 0
DO WHILE (I < 3)
  WRITE (*,*) "hello"
END DO
```

infinite loop!

```
I = 0
DO WHILE (I < 3)
  WRITE (*,*) "hello"
  I = I + 1
END DO
```

Something that might affect the loop-control expression must occur inside the loop

Nested DO-Loops

A DO-loop can contain other DO-loops in its body.

This nested DO-loop, must be completely inside the containing DO-loop.

Note that an EXIT statement transfers control out of the inner-most DO-loop that contains the EXIT statement.

Nested DO-Loop Example

The outer loop has i going from 1 to 3 with step size 1.

For each of the seven values of i, the inner loop iterates 4 times with j going from 1 to 4.

```
INTEGER :: i, j
DO i = 1, 3
  DO j = 1, 4
    WRITE(*,*) i*j
  END DO
END DO
```

There are 12 values printed in total

Table of Exp(x) (preamble)

```
! ----------------------------------------------------------------
! This program computes exp(x) for a range of  values of x using
! the infinite Series expansion of exp(x)
! The range has a beginning value, final value and step size.
! ----------------------------------------------------------------

PROGRAM  Exponential
   IMPLICIT  NONE
   INTEGER            :: Count
   REAL               :: Term
   REAL               :: Sum
   REAL               :: X
   REAL               :: ExpX
   REAL               :: Begin, End, Step
   REAL               :: Tolerance = 0.00001

   WRITE(*,*)  "Initial, Final and Step please --> "

   READ(*,*)   Begin, End, Step

! **** body of program on next slide
```

Table of Exp(x) (body)

```
! **** The preamble is on the previous slide ****

   X = Begin                    ! X starts with the beginning value
   DO
      IF (X > End)  EXIT        ! if X is > the final value, EXIT
      ExpX  = EXP(X)            ! the exp(x) from Fortran's EXP()

      ! calculate exp(x); result stored in Sum
      [redacted]

      WRITE(*,*) X, Sum, ExpX, ABS(Sum-ExpX), ABS((Sum-ExpX)/ExpX)

      ! next X
      X = X + Step
   END DO
END PROGRAM  Exponential
```

Can we use a definite interator instead?

Table of Exp(x) (body)

```
! **** The preamble is on the previous slide ****

   X = Begin                        ! X starts with the beginning value
   DO
      IF (X > End)  EXIT            ! if X is > the final value, EXIT
      ExpX  = EXP(X)                ! the exp(x) from Fortran's EXP()

      ! calculate exp(x); result stored in Sum
      Count = 1
      Sum   = 1.0
      Term  = X
      DO
         IF (ABS(Term) < Tolerance)  EXIT
         Sum   = Sum + Term
         Count = Count + 1
         Term  = Term * (X / Count)
      END DO

      WRITE(*,*) X, Sum, ExpX, ABS(Sum-ExpX), ABS((Sum-ExpX)/ExpX)

      ! next X
      X = X + Step
   END DO
END PROGRAM  Exponential
```

Example

```
!=========================================
! Calculates  3! + 4! + 5!
!=========================================
PROGRAM nestedloop
   IMPLICIT NONE
   INTEGER :: i, j, result, factorial

   result = 0

   DO i = 3, 5
      ! calculate i!; store result in factorial
      factorial = 1
      DO j = 1, i
         factorial = factorial * j
      END DO

      result = result + factorial
   END DO

   WRITE (*,*) "Result= ", result

END PROGRAM nestedloop
```

Back to Counted Do Loops

- There are a few things we have to be careful about when using counted do loops

Beware! DANGER!!!

Changing the values of the control variable or any variables involved in the controlling expressions is risky.

Some compilers will not allow this and will halt and signal an error. Others may allow it with **unpredictable** results.

Programs should be portable. That is they should run on many different systems. Using features that are handled differently in different environments is a no-no.

Beware! DANGER!!!

Do not change the value of the *control-var*.

```
DO a = b, c
  a = b + c
END DO
```

Does the loop ever terminate?

```
DO a = b, c
  READ(*,*) a
END DO
```

What does this do?

Beware! DANGER!!!

Do not change the value of any variable involved in *initial-value*, *final-value* and *step-size*.

```
DO a = b, d, e
   READ(*,*) b               ! initial-value
  changed
   d = 5                     ! final-value changed
   e = -3                    ! step-size changed
END DO
```

The results are unpredictable!

Watch Your Step

What happens if the step size is zero?

```
DO count = -3, 4, 0
...
END DO
```

It seems to be an infinite loop.

In some system this might cause a run-time error and the program is terminated.

GCD Revisited

- A more efficient way of computing the GCD of two integers is possible
- It doesn't even use division!!

Some GCD Facts

- The trivial cases:

```
gcd(k,k) = k , for nonzero k
gcd(0,k) = gcd(k,0) = k, for nonzero k
```

- The general case:

```
For i >= j, gcd(i,j) = gcd(i-j,j)
```

- Using this, we can work backwards from the general case by reducing the larger of the two arguments until we reach one of the trivial cases

A GCD Program

```
INTEGER :: I, J, G
! get I, J from user
DO WHILE (I /= 0 .and. J /= 0 .and. I /= J)
  IF (I>J) THEN
    I = I - J
  ELSE
    J = J - I
  END IF
END DO

IF (I == 0) THEN
  G = J
ELSE ! J == 0 or I == J
  G = I
END IF
```

I	J
36	84
36	48
36	12
24	12
12	12

I	J
39	25
14	25
14	11
3	11
3	8
3	5
3	2
1	2
1	1