Computer Science 1510

Lecture 8

Lecture Outline

- Repetition:
  - DO
  - DO WHILE
  - DO EXIT

Repetition

- With IF and SELECT-CASE statements we saw how to execute portions of code only if some condition is satisfied.

- In many other instances, we may want to repeatedly execute a particular section of code.

- The Fortran construct that can accomplish this repetition or iteration of statements is called a DO loop.

- The DO loop has 3 different forms:

  1. The counter-controlled DO loop which iterates some specified number of times.
  2. The DO WHILE loop which iterates while some condition is true.
  3. The DO EXIT loop which is a generalized version of a DO WHILE loop.

Counter-controlled DO loop

- Syntax:

```
DO control-variable=initial,limit,step-size
   statements-to-repeat
END DO
```

- Statements in between `DO` and `END DO`, referred to as the *body of the loop*, are repeatedly executed.

- The integer `control-variable` is referred to as the *loop counter*.

- Execution proceeds as follows:

  1. Set the value of `control-variable` to `initial`.
  2. Check to see if `control-variable` is
     $\leq$ `limit` if `step-size` > 0, or
     $\geq$ `limit` if `step-size` < 0.
  3. If so, the body of the loop is executed, `step-size` is added to `control-variable` and step 2 is repeated. Otherwise, repetition terminates.

- If not specified, the value of `step-size` is one.

Example 1: Counter-controlled DO

```
PROGRAM Squares
   INTEGER::i
   DO i=1,10
      WRITE(*,'(2I6)') i, i*i
   END DO
END PROGRAM Squares
```

- This program prints a list of squares from 1 to 10.

- Execution proceeds as follows:

  1. The loop counter `i` is initialized to `1`.
  2. Since this value is less than the `limit` of 10, the WRITE statement is executed.
  3. We go back to the DO statement where `i` is increased by one (the default value).
  4. This value is less than 10 so the WRITE statement is executed.
  5. This iteration continues up to `i`=10, when the WRITE statement is executed for the last time since `i` is then increased to 11 which is greater than 10, ending the loop.
  6. Execution moves to the statement following the END DO.

Example 2: DO

```
PROGRAM Odd
   INTEGER::i,n
   READ(*,*) n
   DO i=1,n,2
      WRITE(*,*) i
   END DO
END PROGRAM Odd
```

- Prints out the odd numbers less than or equal to `n`.
- The loop counter `i` is initialized to 1, and is increased by 2 on each iteration of the loop. The loop terminates when `i` is greater than `n`.
- If `n` is odd, say 7, then `i` will have the value 1 on the first iteration of the loop, 3 on the second, 5 on the third, and 7 on the fourth, after which the loop terminates. Thus, the WRITE statement was executed 4 times.
- If `n` is even, say 8, then the loop is again executed 4 times, with the same output as above since 7+2=9 (which is >8).

After the loop

- What is the value of the loop counter at the end of the loop execution?

- For example, if we were to print the value of `i` after the loop in the first example:

```
PROGRAM Squares
   INTEGER::i
   DO i=1,10
      WRITE(*,'(2I6)') i, i*i
   END DO
END PROGRAM Squares
```

  we would obtain `i`=11 since the counter is increased prior to checking if it has exceeded the `limit`.

Example 3: DO

- The only restriction on the value of `step-size` is that it be nonzero, therefore, we can have a negative `step-size` such that the value of `control-variable` is decremented.

- Repetition continues as long as the value of `control-variable` is greater than or equal to `limit`.

- Example:

```
PROGRAM Hello
   INTEGER::i
   DO i=10,1,-1
      WRITE(*,*) 'Hello'
   END DO
   WRITE(*,*) 'i = ',i
END PROGRAM Hello
```

- `Hello` would be printed 10 times and the value of `i` printed after the loop would be 0.

Counter-controlled DO loop

- The number of repetitions of a counter-controlled `DO` loop is determined prior to the start of repetition.

- This number depends on the values of `init`, `limit`, and `step-size`.

- Although the values of variables `init`, `limit`, and `step-size` can be changed within the body of the loop, such a change does not affect the number of repetitions. This is generally poor programming practice.

- Attempting to change the value of the `control-variable` within the body of the loop will result in a compile-time error.

Example 4: DO

- The initial value, the limit, and the step-size can be variables or expressions in addition to constants.

- Example:

```
PROGRAM Sum_of_integers
  IMPLICIT NONE
  INTEGER :: num, i, sum=0
  WRITE(*,*) 'This program prints the sum &
              & 1 + 2 + 3 +...+ num'
  WRITE(*,*) 'Enter a value for num'
  READ(*,*) num
  DO i=1,num
      sum = sum + i
  END DO
  WRITE(*,*) '1 + 2 + 3 +...+ ',num,' = ',sum
END PROGRAM Sum_of_integers
```

Example 5: DO

- DO loops can be nested.
- Example:

```
PROGRAM Mult_table
  IMPLICIT NONE
  INTEGER :: m, n, lastm, lastn, prod
  WRITE(*,*) 'Calculating m*n up to some limit'
  WRITE(*,*) 'Enter the limit of m and n'
  READ(*,*) lastm, lastn
  WRITE(*,*) 'M    N    M*N'
  WRITE(*,*) '-------------'
  DO m=1,lastm
      DO n=1,lastn
          prod = m*n
          WRITE(*,2) m,n,prod
      END DO
  END DO
2 FORMAT(I2,2X,I2,2X,I3)
END PROGRAM Mult_table
```

Fortran statements: DO-WHILE

- A DO-WHILE loop combines the iteration of a DO loop with the conditional execution of an IF statement.

- This is useful when the number of repetitions is not known in advance.

- Syntax:

```
DO WHILE (logical-expression)
   statements-to-repeat
END DO
```

- `logical-expression` is any expression that evaluates to true or false.

- The loop iterates as long as `logical-expression` is true.

Example 6: DO WHILE

```
PROGRAM While
   IMPLICIT NONE
   INTEGER::n,m
   n=0
   m=5
   DO WHILE((m-n)>0)
      n=n+1
      WRITE(*,90) m,n
   END DO
90 FORMAT('m = ',I2,' n = ',I2)
END PROGRAM While
```

Output:

```
m =  5 n =  1
m =  5 n =  2
m =  5 n =  3
m =  5 n =  4
m =  5 n =  5
```

Example 7: DO WHILE

```
PROGRAM While_odd
   INTEGER::i,n
   i=1
   READ(*,*) n
   DO WHILE (i<=n)
      WRITE(*,*) i
      i=i+2
   END DO
END PROGRAM While_odd
```

- Like `PROGRAM Odd`, the above program prints the odd numbers less than or equal to `n`, but uses a DO WHILE loop instead of a counter-controlled DO loop.

Fortran statements: DO EXIT

- Syntax:

```
DO
   statement-sequence-1
   IF (logical-expression) EXIT
   statement-sequence-2
END DO
```

- IF `logical-expression` is true, the `EXIT` command causes the execution to break out of the loop, that is, execution is immediately moved to the statement following the `END DO`.

- A DO EXIT loop behaves similar to a DO WHILE loop in the case where `statement-sequence-1` is not present.

- One has to be careful to not introduce an infinite loop (ie. a loop that never stops iterating) since termination of a DO EXIT loop requires `logical-expression` to be true at some point.

Example 8: DO EXIT

```
PROGRAM Summation
    IMPLICIT NONE
    INTEGER :: num, sum, limit

    WRITE(*,*) 'Finding smallest value of n such that &
                & 1+2+...+n exceeds limit'
    WRITE(*,*) 'Enter value for limit'
    READ(*,*) limit
    num = 0
    sum = 0
    DO
        IF (sum > limit) EXIT
        num = num + 1
        sum = sum + num
    END DO
    WRITE(*,*) '1+...+',num,'=',sum,'>',limit
END PROGRAM Summation
```

Fortran statements: CYCLE

- In some cases we may want to terminate the current iteration of a loop and return to the beginning of the loop for the next iteration.

- This can be accomplished by using a CYCLE statement.

- Syntax:

```
DO i=1,N
   statement-sequence-1
   IF (logical-expression-1) CYCLE
   statement-sequence-2
END DO
```

- If `logical-expression-1` is true on any given iteration, `statement-sequence-2` is skipped, and execution returns to the top of the loop.

- A CYCLE statement can be used in any type of DO loop.

A few more points about loops

- The statements within each loop should be indented for clarity.

- It is possible for the body of a loop to never be executed. For example, in a counter-controlled DO loop, if `initial` exceeds `limit` on the first check, then control jumps to the statement following the END DO.

- EXIT and CYCLE can also be used in counter-controlled DO and DO WHILE loops.