Chapter 3
Fortran Control statements

- Control statement
  - Branching statements
    - The IF .... ENDIF statement
    - The SELECT CASE statement
  - Looping statements
    - The DO...END DO statement
    - The counting DO .... END DO statement
    - The implied DO loop
  - The CYCLE and EXIT statement

Control statement

- Statements are usually part of a program that can be executed or provide basic information for the compiler.
- In this chapter, we focus only on those statements that are under the execution part of the program structure
- Statement usually specifies an action.
- FORTRAN categorizes statement into four groups
  - Expression valid expression
  - Jumping goto
  - Branching if, blocked if, and select case
  - looping while, for and do-while (also called looping)

Control statement

- Selection/branching, iteration/looping and jump are called control statement
- Control statements are statements that alter the sequential execution of instruction as necessary depending on conditions to be considered
- In this chapter, all the branching and looping control statements and related issues will be addressed

GOTO statement

- The simplest way to interrupt the linear flow of a program is to transfer control to another statement, using the GOTO statement

- GOTO statement can be
  - **conditional** or
  - **unconditional**

Unconditional GOTO statement

- Unconditional GOTO statement will force the program control to jump into the specified instruction without checking any condition

- The syntax of the unconditional GOTO statement is:

  GOTO label

- Corresponding to this jump instruction, there should be a statement in the same main or sub program that carries this label with syntax

  label [statement]

Example of Unconditional GOTO statement

```
PROGRAM UncoditionalJump
IMPLICIT NONE
100      print *, "This is the first statement "
goto 300
200      print *, "This is the second statement "
stop
300      print *, "This is the third statement "
end program UncoditionalJump
```

Conditional GOTO statement

- Conditional GOTO statement first checks a condition before forcing the program to jump into the specified instruction

- The syntax of the conditional GOTO statement is:
  - ***GOTO (label1, labe2, label3,…), integer_expression***
  - *Evaluate the integer expression and*
    - *if the value is 1,* *goto* ***label1***
    - *if the value is 2,* *goto* ***label2***
    - *if the value is 3,* *goto* ***label3***
    - *Etc*
    - Where $Label_i$ is statement number or it can be assigned GOTO identifier

GOTO statement label

- Label is a maximum of five sequence of digit characters
- This constraint is because of the meaning of older FORTRAN column concept.
- In the jump statement the label has to be written explicitly
- All statement labels are also local (we will define what local means on chapter four)

GOTO statement label

```
PROGRAM GOTO_TESTING
      IMPLICIT NONE
      INTEGER :: X;

      PRINT *, "ENTER THE VALUE OF X"
      READ *, X
      GOTO (100, 200, 300), X

      PRINT *, "THE INTEGER IS NOT IN THE LIST 1,2 OR 3"
      READ *, X
      STOP

      100 PRINT *, "THE INTEGER IS 1"
      READ *, X
      STOP

      200 PRINT *, "THE INTEGER IS 2“
      READ *, X
      STOP

      300 PRINT *, "THE INTEGER IS 3“
      READ *, X
      STOP
END PROGRAM GOTO_TESTING
```

GOTO statement summary

- Jumps are rarely needed.
- In Fortran66 they were essential for coding conditionals, but in Fortran77 and above, you can do every thing without them.

- Their most common application is for premature termination of a DO loop.

Branching statements

- The branching statement is a mechanism to implement selection of the best statement based on a given condition

- It can be implemented using
  - the if statement (logical if),
  - blocked if statement or
  - select case statement

Logical if statement

- The ***logical IF statement*** is the simplest conditional statement
- It states "if some condition holds, execute single statement"
- ***IF (condition) statement***
  - The condition is any logical variable or expression whose value is logical.

Logical if statement

```
program if_simple

    real :: x

    print *, "enter the value of x"
    read *, x

    if(x < 0)  print *, "We didn't process negative number"

    print *, "Enter another number to exit"

    read *, x
    stop
end program if_simple
```

The Blocked if statement

- The above logical IF statement is rather limited.
- To choose between two blocks of instructions one can use the block IF statement.
- Its syntax is

```
IF (condition) THEN
    statements
ELSE
    statements
ENDIF
```

- The ELSE part is optional and the list of statements in the THEN part can be empty or complex.

The Blocked if statement

- If a number of conditions have to be checked, one can write nested conditionals:

```
IF (condition1) THEN
    IF (condition2) THEN
        statements_TT
    ELSE
        statements_TF
    ENDIF
ENDIF
```

=

```
IF (condition1 && condition2) THEN
    statements_TT
ELSE IF (condition1) THEN
    statements_TF
ENDIF
```

OR

```
IF (condition1) THEN
    statements_T
ELSE
    IF (condition2) THEN
        statements_FT
    ELSE
        statements_FF
    ENDIF
ENDIF
```

=

```
IF (condition1) THEN
    statements_T
ELSE IF (condition2) THEN
    statements_FT
ELSE
    statements_FF
ENDIF
```

Blocked if example

```
program if_blocked
      real :: x
      print *, "enter the value of x"
      read *, x

      if(x < 0) then
            print *, "your number is negative and its sqrt couldn't be computed"
      else
            print *, "the square root of the number is ", sqrt(x);
      end if
      print *, "Enter another number to exit"
      read *, x
      stop
end program if_blocked
```

Blocked if example

```
program max_of_3_num
      real num1, num2, num3
      print *, "Enter 3 real numbers "
      print *, "enter  num1:"
      read *, num1
      print *, "enter  num2:"
      read *, num2
      print *, "enter  num3:"
      read *, num3
      if(num1 > num2 .and. num1 > num3) then
            print *, num1, "  is the maximum "
      else if(num2 > num3) then
            print *, num2, "  is the maximum "
      else
            print *, num3, "  is the maximum "
      end if
end program max_of_3_num
```

Blocked if example

- Write a program that compute the roots of a quadratic equation ax2 + bx + c = 0 given the coefficient a, b, c

- Could you modify your program to generate the real coefficient

The SELECT CASE statement

- The CASE construct is a convenient (and often more readable and/or efficient) alternative to an IF ... ELSE IF ... ELSE construct.
- It allows different actions to be performed depending on the set of outcomes (selector) of a particular expression.
- The general form is:

```
SELECT CASE ( expression )
CASE ( selector-1 )
    block-1
CASE ( selector-2 )
    block-2
    :
    :
[CASE DEFAULT
default block ]
END SELECT
```

The SELECT CASE statement

- expression should be evaluated into an integer, character or logical value.
- It is often just a simple variable.
- selector-n is a set of values from which expression might take.
- Selectors are lists of non-overlapping integer or character outcomes, separated by commas.
  - It can be a single value as CASE (1)
  - A list of values separated by comma as CASE (1,2,3,4,5)
  - A range of value as **start:end** as CASE (1:5)
  - Combination of any of the above separated by comma as CASE ('A':'Z', 1,2,3,4:15)
- block-n is the set of statements to be executed if expression lies in selector-n.
- CASE DEFAULT is used if expression does not lie in any other category. It is optional.

The SELECT CASE statement

➢*Example*. What type of key am I pressing?

```
PROGRAM KEYPRESS
    IMPLICIT NONE
    CHARACTER LETTER
    PRINT *, 'Press a key'
    READ *, LETTER
    select case (LETTER)
        CASE ( 'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U' )
            PRINT *, 'Vowel'
        CASE ( 'b':'d', 'f':'h', 'j':'n', 'p':'t', 'v':'z', &
            'B':'D', 'F':'H', 'J':'N', 'P':'T', 'V':'Z' )
            PRINT *, 'Consonant'
        CASE ( '0':'9' )
            PRINT *, 'Number'
        CASE DEFAULT
            PRINT *, 'Something else'
    END SELECT
    print *, "enter key to exit"
    read *, LETTER
END PROGRAM KEYPRESS
```

Looping statements

- Looping statements are statement that execute the same block of instructions again and again iteratively

- One advantage of computers is that they never get bored by repeating the same action many times.

- There are a number of possible implementations of such control structure in Fortran. They are collectively called the DO Loops

The DO Loops

- There are two basic types of DO loops:
  - ***Deterministic* DO loops** – the number of times the section is repeated is stated explicitly;

    In this case the number of iteration can be known immediately before starting executing the loop

  - ***Non-deterministic* DO loops** – the number of repetitions is not stated in advance.

    In this case the number of iteration can not be known before/during execution of the loop

Deterministic DO loops

- The general form of the DO statement in this case is:

DO variable = value1, value2 [, value3]

repeated section

END DO

- Note that:
  - the loop will execute for each value of the variable from value1 to value2 in steps of value3.
  - value3 may be negative or positive; if it is omitted (which is the usual case) then it is assumed to be 1;
  - the counter variable must be of INTEGER type; (there could be round-off errors if using REAL variables);
  - value1, value2 and value3 may be constants (e.g. 100) or expressions evaluating to integers (e.g.6 * (2 + J))

Deterministic DO loops

Example1

```
PROGRAM LINES
      ! Illustration of DO-loops
      IMPLICIT NONE
      INTEGER L ! a counter
      DO L = 1, 100      ! start of repeated section
            PRINT *, 'I must not talk in class'
      END DO ! end of repeated section
END PROGRAM LINES
```

Deterministic DO loops

Example2:

```
PROGRAM DOLOOPS
    IMPLICIT NONE
    INTEGER I
    DO I = 1, 20
        PRINT *, I, I * I
    END DO
END PROGRAM DOLOOPS
```

Non-deterministic DO loops

- The enclosed section is repeated until some condition meet or failed to meet.

- This may be done in two alternative ways.
- The first requires a logical reason for *stopping* looping
- The syntax of the loop is

```
DO
    STATEMENTS
END DO
```

- This approach require
  - an (***if (…) exit*** form),or
  - An (***if (…) stop*** form),

  statement to terminate the loop.
- Otherwise it is an infinite step
- Use CYCLE to transfer control to the END DO part

Deterministic DO loops

Example:

```
PROGRAM nonDetrministicDo
    ! Illustration of DO-loops
    IMPLICIT NONE
    integer i;
    i = 0;
    DO        ! start of repeated section
        PRINT *, 'I must not talk in class'
        i = i + 1
        !if(i == 10) exit    OR
        if(i == 10) stop
    END DO ! end of repeated section
END PROGRAM nonDetrministicDo
```

Non-deterministic DO loops

- The second requires a logical reason for continuing looping (the DO WHILE(…)).

- The syntax of the loop is

```
DO WHILE(Condition)
    STATEMENTS
END DO
```

- The DO WHILE (...) form continues until some logical expression evaluates as .FALSE..

- Then it stops looping and continues with the code after the loop.

Deterministic DO loops

Example:

```
PROGRAM DoWhile
    ! Illustration of DO-loops
    IMPLICIT NONE
    integer i;
    i = 0;
    DO  while(i < 10) ! start of repeated section
        PRINT *, 'This is the fist sentence'

        i = i + 1
    end do
END PROGRAM DoWhile
```

Example on Non-deterministic DO loops

- Prime number checker

```
program isNPrime
IMPLICIT NONE
!check weather a given number N is prime or not
integer n, mod, starts, ends
print *, "Enter the value of N >=1";read *, n
if(n < 1) then          print *, n, " is not prime";    stop
else
        starts = 2;     ends = sqrt(n * 1.0) + 1
        !ends can be n, or n/2 or sqrt(n)+1
        print *, "check ends at ", ends
        do while(starts < ends)
                mod = n - (starts * (n / starts))
                if(mod == 0) then
                        print *, n, " is not a prime";  stop
                end if
                starts = starts + 1
        end do
        print *, n, " is a prime number"
end if
end program isNPrime
```

Nested DO Loops

- DO loops can be nested (i.e. one inside another).
- Indentation is definitely recommended to clarify the loop structure.

```
PROGRAM NESTED
    ! Illustration of nested DO-loops
    IMPLICIT NONE
    INTEGER I, J ! loop counters
    DO I = 1, 6 ! start of outer loop
        PRINT *, 'Start of loop for I = ', I
        DO J = 1, 3 ! start of inner loop
            PRINT *, 'J = ', J
        END DO
    END DO ! end of repeated section
END PROGRAM NESTED
```

Example on Non-deterministic DO loops

- Nth Prime number finder

```
!program isNPrime
IMPLICIT NONE
!Compute the Nth prime number
integer n, m, next, starts, ends, mod;
!M is to indicate how many primes found so far
!Start is the minimum number to find the next prime
logical found, NthPrime
m = 0; next = 2      ;        print *, "Enter the value of N >=1"
read *, n
if(n < 1) then                print *, "Invalid value of input";          stop
else
        do while(m < n)
                found = .false.
                !find a prime number >=next
                do while(.not. found)
                        !check if next is a prime number
                        starts = 2; ends = sqrt(next *1.0) + 1;  isPrime = .true.
```

Example on Non-deterministic DO loops

```
            do while(starts < ends .and. isPrime)
                mod = next - (starts * (next / starts))
                if(mod == 0) isPrime = .false.;   starts = starts + 1
            end do
            if(isPrime) found = .true.;                next = next + 1
        end do
        m = m + 1
    end do
    print *, "The ", n,"th prime number is ", next - 1
end if
end program NthPrime
```

The CYCLE statement

- The cycle statement is used to continue the next iteration by over passing the set on instruction in a looping structure starting from the CYCLE statement until the end do statement

```
PROGRAM cycleExample1
        ! Illustration of DO-loops
        IMPLICIT NONE
        integer i;      i = 0;
        DO      ! start of repeated section
            PRINT *, 'This is the fist sentence'
            i = i + 1
            !if(i == 10) stop    OR
            if(i == 10) stop
            cycle
            PRINT *, 'This is the fist sentence'
        END DO ! end of repeated section
END PROGRAM cycleExample1
```

The implied DO loop

- This will be discussed Later