Fortran - Control in Fortran IF and DO

Prof. Dr. Narayan Prasad Adhikari
Central Department of Physics
Tribhuvan University Kirtipur, Kathmandu, Nepal

July 5, 2021

■Warm up!!!

- Recall our lectures - Algorithm, what is it? What is flow chart? Significance of Algorithm & flowchart in programming etc ...
- Recall simple programs and recall those what you write ...

■If () statement

- CONTROL CONSTRUCTS: BRANCHES
  Branches are Fortran statements that permit us to select and execute specific sections of code (called blocks) while skipping other sections of code. They are variations of the IF statement, plus the SELECT CASE.

- The commonest form of the IF statement is the IF () statement

- In IF() inside parenthesis there will be logical expressions, hence if it is true it moves forward and executes the statement and if it is not true then it simply goes to another line of the program i.e. it skips statement ....

```
....
if (x>y) write(*,*) "x is greater than y"
....
....
```

■Block If () statement

- However IF statement like above are not so popular in FORTRAN. Generally we use "BLOCK IF () Statements like

```
IF (logical_expr) THEN
   Statement 1          }
   Statement 2          }    Block 1
   ...                  }
END IF
```

- If the logical expression is true, the program executes the statements in the block between the IF and END IF statements. If the logical expression is false, then the program skips all of the statements in the block between the IF and END IF statements, and executes the next statement after the END IF. The flowchart for a block IF construct is shown in Figure 1.

■Block If () statement

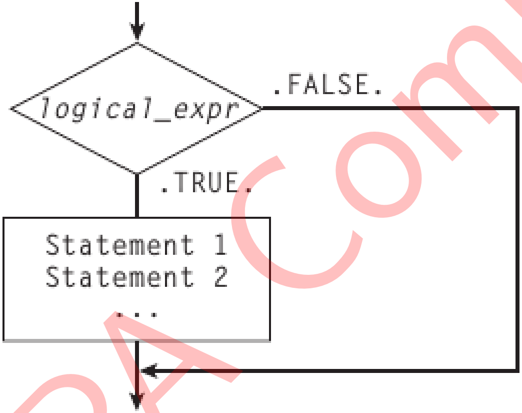


Figure: Flowchart of Block IF statement in fortran

■Block If () statement

- The IF (...) THEN is a single Fortran statement that must be written to- gether on the same line, and the statements to be executed must occupy separate lines below the IF (...) THEN statement.

- An END IF statement must follow them on a separate line. There should not be a statement number on the line con- taining the END IF statement. For readability, the block of code between the IF and END IF statements is usually indented by two or three spaces, but this is not actually required.

■Block If () statement

- As an example of BLOCK IF construct let us consider solution of a quadratic equation.

$$a\ x^2 + b\ x + c = 0 \tag{1}$$

  It has solution

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

  with discriminant

$$d = b^2 - 4ac.$$

- We know that nature of roots i.e. whether they are equal, real or complex depend on discriminant (d).

■Block If () statement

- Suppose that we wanted to examine the discriminant of the quadratic equation and tell a user if the equation has complex roots.
- In pseudocode, the block IF construct to do this would take the form:

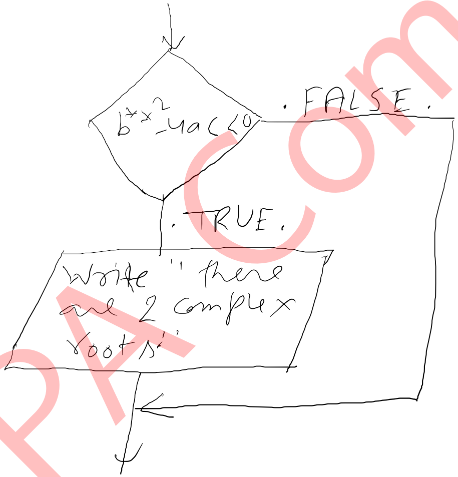
```
IF $(b^2 - 4. * a * c) < 0.$ THEN
Write message that equation has two complex roots.
END of IF
```

  In Fortran, the block IF construct is

```
IF ( $(b^2 - 4. * a * c) < 0.$) THEN
WRITE (*,*) 'There are two complex roots to this equation.'
END IF
```

- The flowchart is shown in figure 2

■Block If () statement



■IF () Then .. ELSE IF .. ELSE .. Clauses

- In the simple block IF construct, a block of code is executed if the controlling logical expression is true. If the controlling logical expression is false, all of the statements in the construct are skipped.

- Sometimes we may want to execute one set of statements if some condition is true, and different sets of statements if other conditions are true. In fact, there might be many different options to consider. An ELSE clause and one or more ELSE IF clauses may be added to the block IF construct for this purpose. The block IF construct with an ELSE clause and an ELSE IF clause has the form

■Block If () statement

```
IF (logical_expr_1) THEN
   Statement 1                     }
   Statement 2                     }    Block 1
   ...                             }
ELSE IF (logical_expr_2) THEN
   Statement 1                     }
   Statement 2                     }    Block 2
   ...                             }
ELSE
   Statement 1                     }
   Statement 2                     }    Block 3
   ...                             }
END IF
```

■IF () Then .. ELSE IF .. ELSE .. Clauses

- If logical-expr-1 is true, then the program executes the statements in Block 1, and skips to the first executable statement following the END IF. Otherwise, the program checks for the status of logical-expr-2. If logical-expr-2 is true, then the program executes the statements in Block 2, and skips to the first executable statement following the END IF. If both logical expressions are false, then the program executes the statements in Block 3.The ELSE and ELSE IF statements must occupy lines by themselves. There should not be a statement number on a line containing an ELSE or ELSE IF statement. There can be any number of ELSE IF clauses in a block IF construct. The logical expression in each clause will be tested only if the logical expressions in every clause above it are false. Once one of the expressions proves to be true and the corresponding code block is executed, the program skips to the first executable statement following the END IF.

■IF () Then .. ELSE IF .. ELSE .. Clauses

- To illustrate the use of the ELSE and ELSE IF clauses, let's reconsider the quadratic equation once more. Suppose that we wanted to examine the discriminant of a quadratic equation and to tell a user whether the equation has two complex roots, two identical real roots, or two distinct real roots. In pseudocode, this construct would take the form

```
IF $(b**2 - 4.*a*c) < 0.0$ THEN
Write message that equation has two complex roots.
ELSE IF $(b**2 - 4.*a*c) > 0.0$ THEN
Write message that equation has two distinct real roots.
ELSE
Write message that equation has two identical real roots.
END IF
```

■Block If () statement

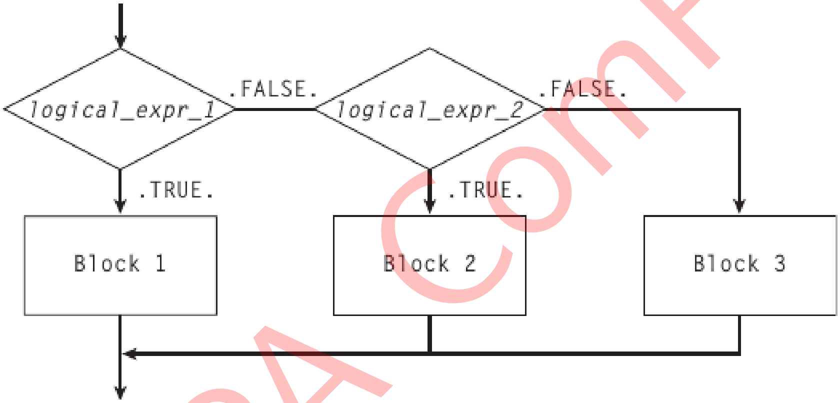


Figure: Flowchart for a block IF construct with an ELSE IF (...) THEN clause and an ELSE clause.

■Block If () statement

The Fortran statements to do this are

IF $((b * *2 - 4. * a * c) < 0.0)$ THEN
WRITE (*,*) 'This equation has two complex roots.'
ELSE IF $((b * *2 - 4. * a * c) > 0.0)$
WRITE (*,*) 'This equation has two distinct real roots.'
ELSE
WRITE (*,*) 'This equation has two identical real roots.'
END IF
The flowchart for this construct is as shown in figure 4

■Block If () statement

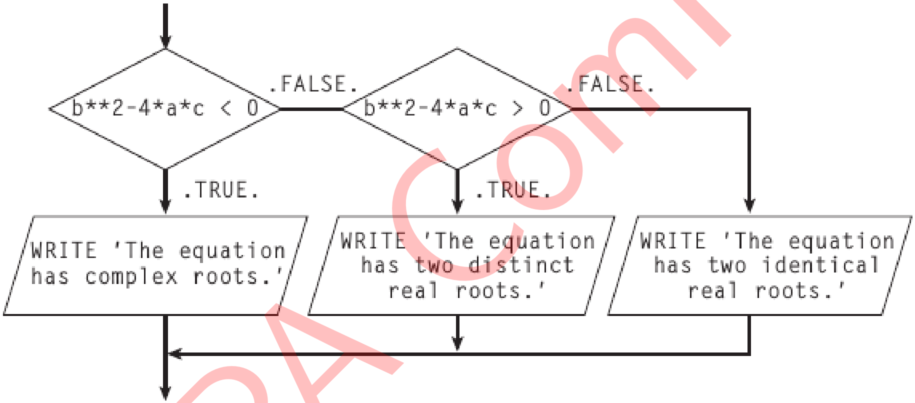

b**2-4*a*c < 0
.FALSE.
b**2-4*a*c > 0
.FALSE.
.TRUE.
.TRUE.
WRITE 'The equation has complex roots.'
WRITE 'The equation has two distinct real roots.'
WRITE 'The equation has two identical real roots.'

Figure: Flowchart showing structure to determine whether a quadratic equation has two complex roots, two identical real roots, or two distinct real roots.

■Quadratic equation - Block If

- Problem: Design and write a program to solve for the roots of a quadratic equation, regardless of type.
- **Solution:** Follow all the outlines described above.
- 1. State the problem. The problem statement for this example is very simple. We want to write a pro- gram that will solve for the roots of a quadratic equation, whether they are distinct real roots, repeated real roots, or complex roots.
- 2. Define the inputs and outputs. The inputs required by this program are the coefficients a, b, and c of the quadratic equation

$$a * x^2 + b * x + c = 0 \tag{3}$$

The output from the program will be the roots of the quadratic equation, whether they are distinct real roots, repeated real roots, or complex roots.

■Quadratic equation - Block If

- 3. Design the algorithm. This task can be broken down into three major sections, whose functions are input, processing, and output:

  Read the input data
  Calculate the roots
  Write out the roots

  We will now break each of the above major sections into smaller, more detailed pieces. There are three possible ways to calculate the roots, depending on the value of the discriminant, so it is logical to implement this algorithm with a three-branched IF statement. The resulting pseudocode is:

■Quadratic equation - Block If

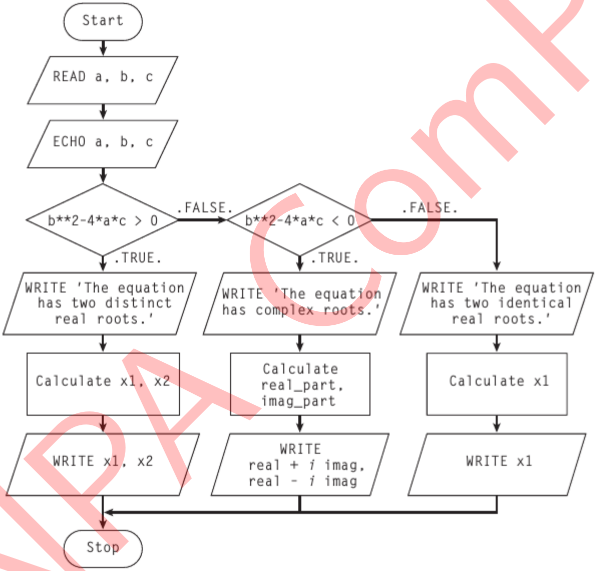
```
Prompt the user for the coefficients a, b, and c.
Read a, b, and c
Echo the input coefficients
discriminant$\leftarrow b**2-4.*a*c$
IF discriminant $> 0$ THEN
    x1 $\leftarrow (-b+sqrt(discriminant))/(2.*a)$
    x2 $\leftarrow (-b-sqrt(discriminant))/(2.*a)$
    Write message that equation has two distinct real roots.
    Write out the two roots.

ELSE IF discriminant $< 0$ THEN
    real˙part $\leftarrow -b/(2.*a)$
    imag˙part $\leftarrow sqrt(abs(discriminant))/(2.*a)$
    Write message that equation has two complex roots.
    Write out the two roots.
```

■Quadratic equation - Block If

```
ELSE
    x1 $\leftarrow -b/(2. * a)$
    Write message that equation has two identical real roots.
    Write out the repeated root.
END IF
```

- **4. Turn the algorithm into Fortran statements.**

■Block If () statement



Start
READ a, b, c
ECHO a, b, c
b**2-4*a*c > 0
.FALSE.
b**2-4*a*c < 0
.FALSE.
.TRUE.
.TRUE.
WRITE 'The equation has two distinct real roots.'
WRITE 'The equation has complex roots.'
WRITE 'The equation has two identical real roots.'
Calculate x1, x2
Calculate real_part, imag_part
Calculate x1
WRITE x1, x2
WRITE real + i imag, real - i imag
WRITE x1
Stop

Figure: Flowchart of program roots.

■Block If () statement

- Evaluating a Function of Two Variables:

Write a Fortran program to evaluate a function f(x,y) for any two user-specified values x and y. The function f(x,y) is defined as follows:

$$f(x,y) = \begin{cases} x + y & x \geq 0 \text{ and } y \geq 0 \\ x + y^2 & x \geq 0 \text{ and } y < 0 \\ x^2 + y & x < 0 \text{ and } y \geq 0 \\ x^2 + y^2 & x < 0 \text{ and } y < 0 \end{cases}$$

■Block If () statement

1. State the problem. This problem statement is very simple: Evaluate the function f(x,y) for any user-supplied values of x and y.

2. Define the inputs and outputs. The inputs required by this program are the values of the independent variables x and y. The output from the program will be the value of the function f(x,y).

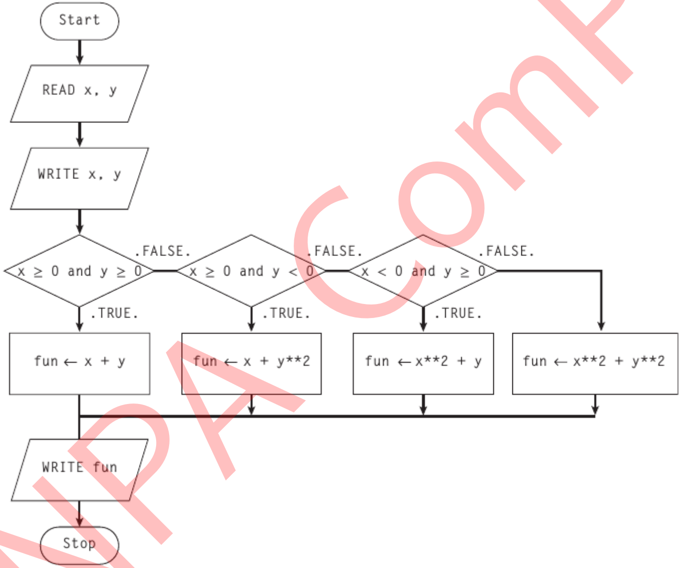
3. Design the algorithm. This task can be broken down into three major sections, whose functions are input, processing, and output:

```
Read the input values x and y
      Calculate f(x,y)
      Write out f(x,y)
```

■Pseudocode -two variabl function

```
Prompt the user for the values x and y.
Read x and y
Echo the input coefficients
IF x >= 0 and y >= 0 THEN
fun ← $x + y$
ELSE IF $x >= 0$ and $y < 0$ THEN
fun ← $x + y**2$
ELSE IF $x < 0$ and $y >= 0$ THEN
fun ← $x**2 + y$
ELSE
fun ← $x**2 + y**2$
END IF
Write out f(x,y)
```

■flowchart-two variable function



■Class work

1. CLASS WORK: WRITE CODE FOR ABOVE PROBLEM
2. Home WORK: Print out your marksheet of first SEM.
3. Home WORK: From chapman's book: 3-2 to 3-11,3-14
from 4th edition

■IF-THEN-ELSE Can be Nested: Homework

- IF-THEN-ELSE Can be Nested
- you check above examples of IF () THEN ... ELSE IF ()....ELSE ....END IF to solve quadratic equation
- HW: 1. now rewrite the code to solve quadratic equation using nested IF ().
- HW: 2. Use three different IF () THEN ... without ELSE
- Compare above 1. & 2. just think which one is easier....

■LOOPS in FORTRAN

- Loops are Fortran constructs that permit us to execute a sequence of statements more than once. There are two basic forms of loop constructs: while loops and iterative loops (or counting loops). The major difference between these two types of loops is in how the repetition is controlled.

- The code in a while loop is repeated an indefinite number of times until some user-specified condition is satisfied. By contrast, the code in an iterative loop is repeated a specified number of times, and the number of repetitions is known before the loop starts.

■DO Loop

- A *do loop* is a block of statements that are repeated indefinitely as long as some condition is satisfied. The general form of a do loop (in fact it is called "infinite do loop") in Fortran is:

```
DO
...
IF (logical˙expr) EXIT }CodeBlock
...
END DO
```

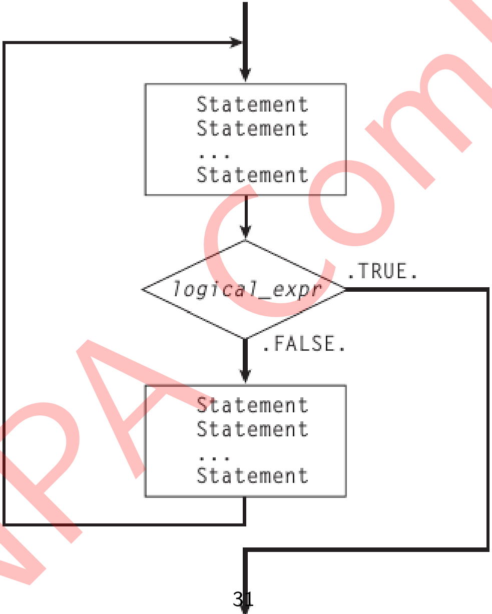
- The block of statements between the DO and END DO are repeated indefinitely until the logical˙expr becomes true and the EXIT statement is executed. After the EXIT statement is executed, control transfers to the first statement after the END DO.
- Discuss a few examples in class....
- One you try without IF in above form....

■The DO While Loop

- A while loop may contain one or more EXIT statements to terminate its execution. Each EXIT statement is usually a part of an IF statement or block of construct. If the logical˙expr in the IF is false when the statement is executed, the loop continues to execute. If the logical˙expr in the IF is true when the statement is executed, control transfers immediately to the first statement after the END DO. If the logical expression is true the first time we reach the while loop, the statements in the loop below the IF will never be executed at all!
- The pseudocode corresponding to a while loop is

```
DO WHILE (logical expression)
statement...
statement...
End of WHILE (DO)
```

■The DO While Loop - Flow Chart



■The DO While Loop - Flow Chart

Writing a code to get mean/standard deviation of given data.
For this:

- 1. State the problem.
  Since we assume that the input numbers must be positive or zero, a proper statement of this problem would be: calculate the average and the standard deviation of a set of measurements, assuming that all of the measurements are either positive or zero, and assuming that we do not know in advance how many measurements are included in the data set. A negative input value will mark the end of the set of measurements.

- 2. Define the inputs and outputs.
  The inputs required by this program are an unknown number of positive or zero real (floating-point) numbers. The outputs from this program are a printout of the mean and the standard deviation of the input data set.

■The DO While Loop - Flow Chart

- In addition, we will print out the number of data points input to the program, since this is a useful check that the input data was read correctly.
- 3. Design the algorithm.
  This program can be broken down into three major steps:

  Accumulate the input data
  Calculate the mean and standard deviation
  Write out the number of points, mean and standard deviation,

  The first major step of the program is to accumulate the input data. To do this, we will have to prompt the user to enter the desired numbers. When the numbers are entered, we will have to keep track of the number of values entered, plus the sum and the sum of the squares of those values. The pseudocode for these steps is:

■The DO While Loop - Flow Chart

```
Initialize n, sum˙x, and sum˙x2 to 0
Do WHILE
Prompt user for next number
Read in next x
IF $x < 0.$ EXIT
n $\leftarrow n + 1$
sum˙x $\leftarrow$ sum˙x $+$ x
sum˙x2 $\leftarrow$ sum˙x2 $+$ x**2
End of DO WHILE
```

Note that we have to read in the first value before the IF () EXIT test so that the while loop can have a value to test the first time it executes.

■The DO While Loop - Flow Chart

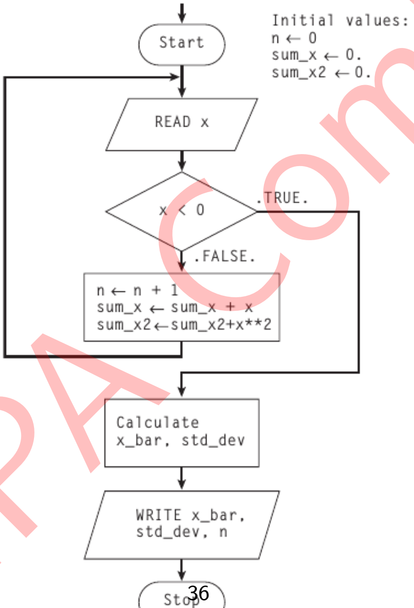
Next, we must calculate the mean and standard deviation. The pseudocode for this step is

```
x˙bar ← sum˙x / REAL(n)
std˙dev ← SQRT((REAL(n)*sum˙x2 - sum˙x**2) /
(REAL(n)*REAL(n-1)))
```

Finally, we must write out the results.

```
Write out the mean value x˙bar
Write out the standard deviation std˙dev
Write out the number of input data points n
```

■ mean/s.d. - Flow Chart



mean/s.d. - Flow Chart

- Write a code according to the flow diagram. At first you can consider n=10
- Now you think a bit different algorithm to calculate mean and s.d. of given set of data.

The Iterative or Counting Loop

- Just recall previous lecture.
- In the Fortran language, a loop that executes a block of statements a specified number of times is called an iterative DO loop or a counting loop. The counting loop construct has the form

```
DO index = istart, iend, incr
Statement 1
...
Statement n
END DO
```

  where index is an integer variable used as the loop counter (also known as the loop index). The integer quantities istart, iend, and incr are the parameters of the counting loop; they control the values of the variable index during execution. The parameter incr is optional; if it is missing, it is assumed to be 1.

The Iterative or Counting Loop

- The statements between the DO statement and the END DO statement are known as the body of the loop. They are executed repeatedly during each pass of the DO loop.
  The counting loop construct functions as follows:
- 1. Each of the three DO loop parameters istart, iend, and incr may be a constant, a variable, or an expression. If they are variables or expressions, then their values are calculated before the start of the loop, and the resulting values are used to control the loop.
- 2. At the beginning of the execution of the DO loop, the program assigns the value istart to control variable index. If index*incr $\leq$ iend*incr, the program executes the statements within the body of the loop.

The Iterative or Counting Loop

- 3. After the statements in the body of the loop have been executed, the control variable is recalculated as
  index = index + incr
  If index*incr is still $\leq$ iend*incr, the program executes the statements within the body again.
- 4. Step 2 is repeated over and over as long as index*incr $\leq$ iend*incr. When this condition is no longer true, execution skips to the first statement following the end of the DO loop.

- The number of iterations to be performed by the DO loop may be calculated using the following equation

$$iter = \frac{iend - istart + incr}{incr} \tag{4}$$

■ The Iterative or Counting Loop

- Let's look at a number of specific examples to make the operation of the counting loop clearer. First, consider the following example:

```
DO i = 1,10
Statement 1
...
Statement n
END DO
```

- In this case, statements 1 through n will be executed 10 times. The index variable i will be 1 on the first time, 2 on the second time, and so on. The index variable will be 10 on the last pass through the statements. When control is returned to the DO statement after the tenth pass, the index variable i will be increased to 11. Since 11 X 1 $\geq$10 X 1, control will transfer to the first statement after the END DO statement.

The Iterative or Counting Loop

- Second, consider the following example:
  DO i = 1, 10, 2
  Statement 1
  ...
  Statement n
  END DO
- In this case, statements 1 through n will be executed five times. The index variable i will be 1 on the first time, 3 on the second time, and so on. The index variable will be 9 on the fifth and last pass through the statements. When control is returned to the DO statement after the fifth pass, the index variable i will be increased to 11. Since 11 X 2 $>$ 10 X 2, control will transfer to the first statement after the END DO statement.

The Iterative or Counting Loop

- Third, consider the following example:

  ```
  DO i = 1, 10, -1
  Statement 1
  ...
  Statement n
  END DO
  ```

- Here, statements 1 through n will never be executed, since index*incr > iend*incr on the very first time that the DO statement is reached. Instead, control will transfer to the first statement after the END DO statement.

The Iterative or Counting Loop

- Finally, consider the example: DO i = 3, -3, -2

  Statement 1
  ...
  Statement n
  END DO

- In this case, statements 1 through n will be executed four times. The index variable i will be 3 on the first time, 1 on the second time, -1 on the third time, and -3 on the fourth time. When control is returned to the DO statement after the fourth pass, the index variable i will be decreased to -5. Since (-5 X -2) > (-3 X -2), control will transfer to the first statement after the END DO statement.

The Iterative or Counting Loop

- The pseudocode corresponding to a counting loop is:

```
DO for index = istart to iend by incr
Statement 1
...
Statement n
End of DO
```

and the flowchart for this construct is shown in Figure 6.

The Iterative or Counting Loop

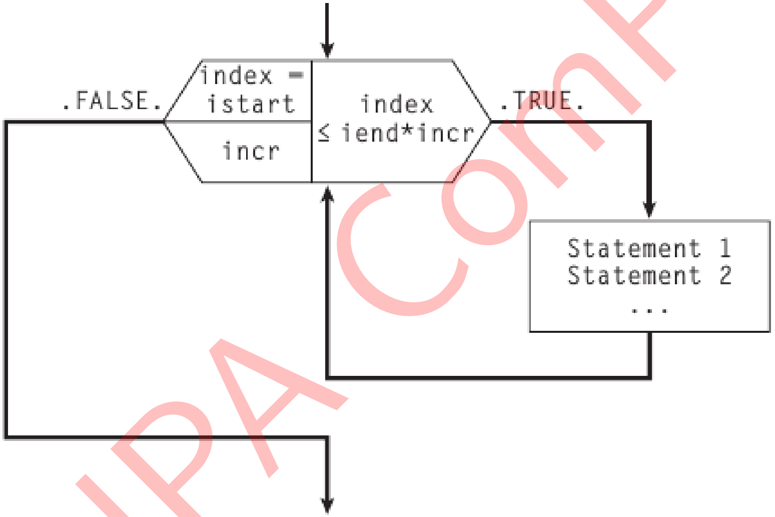


Figure: Flow chart of DO in fortran

■ The Iterative/Counting Loop -CA/HA

- Implement an algorithm that reads in a set of measurements and calculates the mean and the standard deviation of the input data set, when any value in the data set can be positive, negative, or zero.

Important TIPS

- Never modify the value of a DO loop index variable while inside the loop.
- Never depend on an index variable to retain a specific value after a DO loop completes normally.
- Always indent the body of a DO loop by two or more spaces to improve the readability of the code.

■ The CYCLE and EXIT Statements

- There are two additional statements that can be used to control the operation of while loops and counting DO loops: CYCLE and EXIT.
- If the CYCLE statement is executed in the body of a DO loop, the execution of the current iteration of the loop will stop, and control will be returned to the top of the loop. The loop index will be incremented, and execution will resume again if the index has not reached its limit. An example of the CYCLE statement

```
PROGRAM test˙cycle
INTEGER :: i
DO i = 1, 5
IF ( i == 3 ) CYCLE
WRITE (*,*) i
END DO
WRITE (*,*) 'End of loop!'
END PROGRAM test˙cycle
```

■ The CYCLE and EXIT Statements

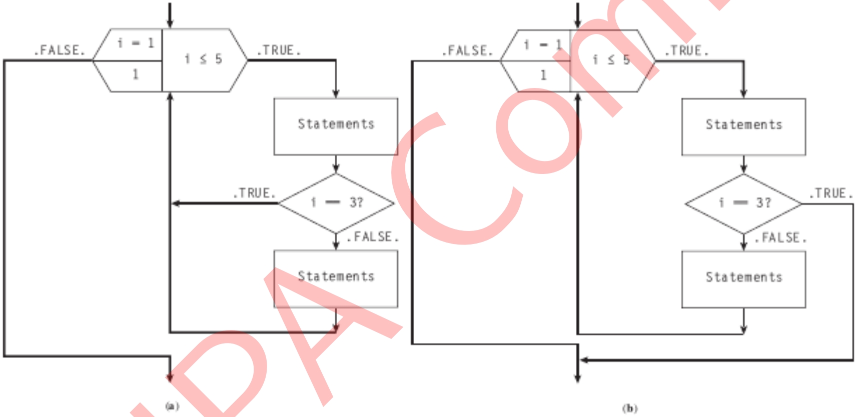


Figure: (a) Flowchart of a DO loop containing a CYCLE statement. (b) Flowchart of a DO loop containing an EXIT statement.

■ The CYCLE and EXIT Statements

- Note that the CYCLE statement was executed on the iteration when i was 3, and control returned to the top of the loop without executing the WRITE statement. After control returned to the top of the loop, the loop index was incremented and the loop continued to execute.
- If the EXIT statement is executed in the body of a loop, the execution of the loop will stop and control will be transferred to the first executable statement after the loop.

  An example of the EXIT statement in a DO loop is shown below.

The CYCLE and EXIT Statements

```
PROGRAM testexit
INTEGER :: i
DO i = 1, 5
IF ( i == 3 ) EXIT
WRITE (*,*) i
END DO
WRITE (*,*) 'End of loop!'
END PROGRAM testexit
```

- You can assign name to DO LOOP

■ NESTED DO LOOP

- It is possible for one loop to be completely inside another loop. If one loop is completely inside another one, the two loops are called nested loops. The following example shows two nested DO loops used to calculate and write out the product of two integers. PROGRAM nestedloops

```
INTEGER :: i, j, product
DO i = 1, 3
DO j = 1, 3
product = i * j
WRITE (*,*) i, ' * ', j, ' = ', product
END DO
END DO
END PROGRAM nestedloops
```

NESTED DO LOOP

- In this example, the outer DO loop will assign a value of 1 to index variable i, and then the inner DO loop will be executed. The inner DO loop will be executed three times with index variable j having values 1, 2, and 3. When the entire inner DO loop has been completed, the outer DO loop will assign a value of 2 to index variable i, and the inner DO loop will be executed again.
- Test above code

NAMING DO LOOP

```
PROGRAM bad˙nested˙loops˙2 INTEGER :: i, j, product
outer: DO i = 1, 3
inner: DO j = 1, 3
product = i * j
WRITE (*,*) i, ' * ', j, ' = ', product
END DO outer
END PROGRAM bad˙nested˙loops˙2
```

- Extremely useful if you are working with many nested do loops. Otherwise OR if you can remeber its not useful.

■ NESTED IF AND DO

- Nesting loops within IF constructs and vice versa It is possible to nest loops within block IF constructs or block IF constructs within loops. If a loop is nested within a block IF construct, the loop must lie entirely within a single code block of the IF construct.
  For example, the following statements are illegal since the loop stretches between the IF and the ELSE code blocks of the IF construct.

```
outer: IF ( a < b ) THEN
...
inner: DO i 1, 3
...
ELSE
...
END DO inner
```

■ NESTED IF AND DO

- In contrast, the following statements are legal, since the loop lies entirely within a single code block of the IF construct.

```
outer: IF ( a < b ) THEN
...
inner: DO i = 1, 3
...
END DO inner
...
ELSE
...
END IF outer
```

Go to and Continue

The GO TO statement has the form

GO TO label

where label is the label of an executable Fortran statement. When this statement is executed, control jumps unconditionally to the statement with the specified label. In the past, GO TO statements were often combined with IF statements to create loops and conditional branches. For example, a while loop could be implemented as

```
10 CONTINUE
...
IF ( condition ) GO TO 20
...
GO TO 10
20 ...
```

CW: Write a code using go to statement

Go to and Continue

```
program gotom
implicit none
!This program shows how to use goto command in f90.
!real::r,s,t
integer::i,j,sum1
sum1=0
i=1
1010 sum1=sum1+i
i=i+1
if(i<101) goto 1010
write(*,*) "sum is ",sum1
end program
```

```
program goto1
implicit none
integer::i,j,k,sum1
real::x,y,z
sum1=0
do i=1,100,2
sum1=sum1+i
enddo
print*,"sum1=",sum1
sum1=0
i=1
11 sum1=sum1+i
i=i+2
if(i¡100) goto 11
print*,"sum1=",sum1
end program
!Class work: Just ry to understand the code...
```

Go to and Continue

```
program continue1
implicit none
!This program shows how to use continue command in f90.Nov1, 2018
integer::i,sum1
sum1=0
do 10 i=1, 100
sum1=sum1+i
10 continue
write(*,*) "sum is ",sum1
end program
```

HW: Discuss the applications of "go to", "continue", "do" and " do while". Explain their differences and similarities (if any among any of two or more) with suitable examples.