Programming for Engineers
Fortran: Loop-Based Control Structures

Abu Hasan Abdullah

January 7, 2009

Overview

1. Previously we have seen these conditional statements and constructs—`IF` statement, block `IF` construct, `IF...ELSE` construct and `SELECT CASE` construct.

2. In this session we take a look at another type of control structure—*loop*. There are two types

   (a) counted loop
   (b) conditional loop

Course Text:
MAYO W. E. AND CWIAKALA M. (1995): *Programming with Fortran 77*, ISBN 0-07-041155-7, McGraw-Hill

Counted Loop

- Executes a predetermined number of times and the variables controlling the loop *cannot* be altered during the loop execution
- Most widely used by engineers and scientists
- Known as `DO` loop in Fortran

Counted Loop

- General form of DO loop construct

```
          DO sl LCV=start,stop[,step]
             ...
             series of instructions
             ...
    sl    CONTINUE
```

  where

sl	statement label
LCV	loop control variable

- In this form, DO statement marks beginning of loop and *sl* CONTINUE statement marks end of loop

Counted Loop

- Alternative form of `DO` loop construct

```
DO LCV=start,stop[,step]
   ...
   series of instructions
   ...
END DO
```

  where

  LCV loop control variable

- In this form, `DO` statement marks beginning of loop and `END DO` statement marks end of loop

Counted Loop

- In both forms of the counted loop the computer has complete control of the loop and handles all tasks, including

  1. *Initialize* the `LCV` to the `start` value
  2. *Increment* the `LCV` by the `step` value each time through the loop
  3. *Test* the `LCV` to see if it exceeds the `stop` value
  4. *Decide* when to terminate the loop

Counted Loop

- Rules and guidelines for setting up the loop and *LCV*

  1. *LCV* should be integer. Avoid using real value
  2. *start*, *stop*, and *step* values used to establish *LCV* can be variables
  3. *LCV* cannot be changed inside the body of loop
  4. The *step* size can be omitted. If it is, the computer assumes a *step* size of 1
  5. It is permissible to leave the body of loop. But you may not enter a loop body from outside

Counted Loop

- Code snippet

```
      DO 5 I=1,10,1
         PRINT *, 2*I
5     CONTINUE
      PRINT *, I
```

or

```
      DO I=1,10,1
         PRINT *, 2*I
      END DO
      PRINT *, I
```

Counted Loop

- Study the `DO` loop flowcharts of Examples 5.1–5.4
- Turn Examples 5.1–5.4 into complete programs, compile and run them to study various `DO` loop behaviours and how *`LCV`* affects them

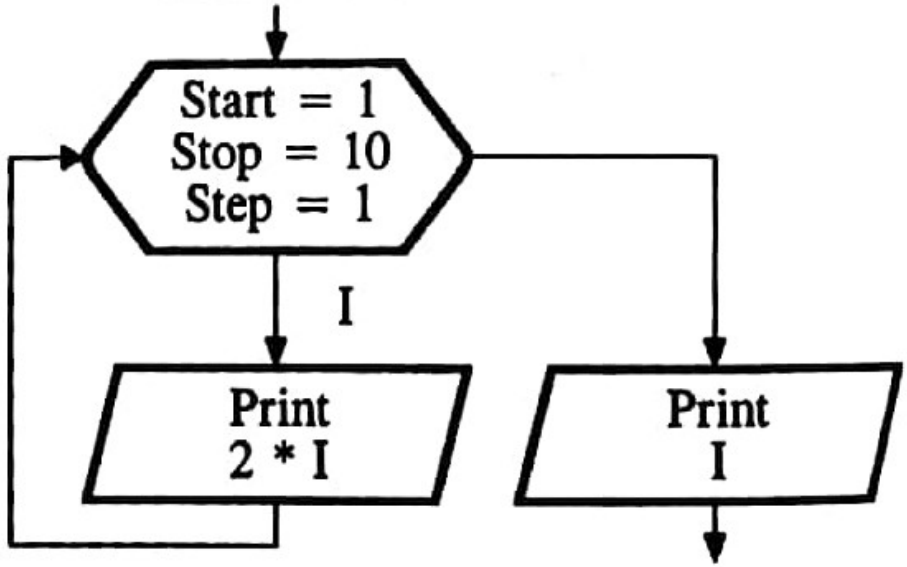
EXAMPLE 5.1

Here is an example of a DO loop in action.

Program

```
C The following loop will
C execute 10 times and use
C I as the loop control
C variable. Note that we
C can look at the value of I
C inside or outside the loop.
      DO 5 I=1,10,1
         PRINT *, 2*I
5     CONTINUE
      PRINT *, I
```

Flowchart



EXAMPLE 5.2

In the following program segment, we attempt to change the value of the LCV inside the loop. But since this is not allowed, we would receive an error message from the compiler.

```
      DO 5 I=1, 10, 1
         I = I + 1
5     CONTINUE
```

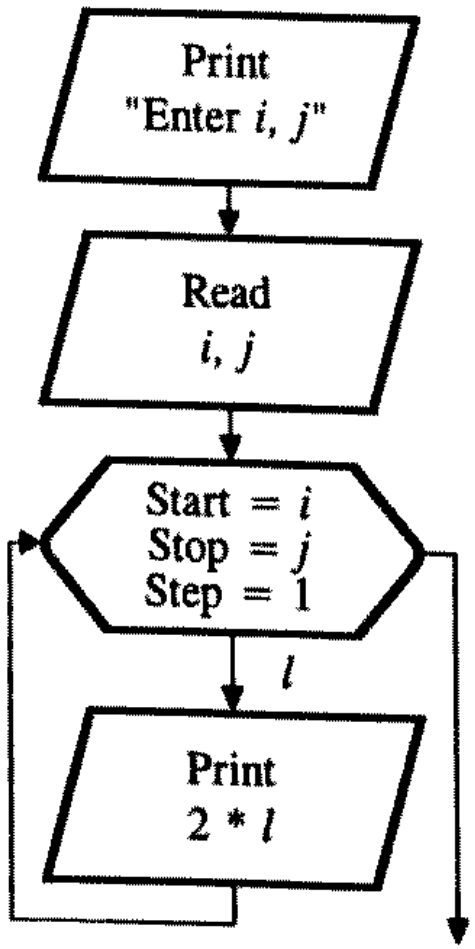
EXAMPLE 5.3

The loop control variables themselves can be either variables that are read in at execution time or the results of a computation. In this example, we will read in the variables i and j, and use these as the start and stop variables in the DO loop to compute all the even integers between and including $2i$ and $2j$.

Program

```
C We will read in I and J
C for use as the loop
C control variables.
      PRINT *, 'ENTER I, J'
      READ *, I, J
      DO 10 L = I, J, 1
         PRINT *, 2*L
10    CONTINUE
```

Flowchart



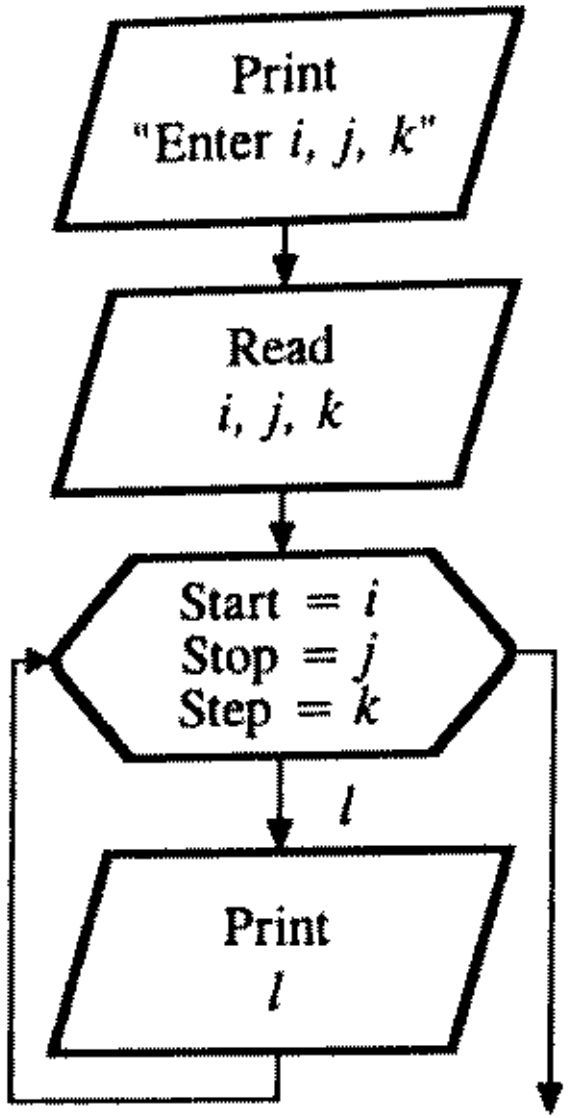
EXAMPLE 5.4

The step size of a DO loop may be a variable that is read in at execution time.

Program

```
C We will read in two integers
C I and J, along with a step
C size K. The program then
C prints out all integers
C between I and J in steps
C of K.
      PRINT *, 'ENTER I, J, K'
      READ *, I, J, K
      DO 10 L = I, J, K
         PRINT *, L
10    CONTINUE
```

Flowchart



EXAMPLE 5.5

Here are some examples of correct and incorrect usage of the DO statements:

Correct	Incorrect	Comments
DO 10 I = 1, 10		*(Step is optional (assumed=1))*
DO 20 I = J, 10		*(Mixing variables, constants OK)*
DO 30 I = 10, 1, −1		*(Decreasing index OK)*
	DO 40 I = 1.0, 5.0, 0.1	*(Mixed mode)*
	DO 50 I = 1, 10, I	*(Subtle attempt to modify LCV)*
	DO 60 I = 10, 1	*(Loop does not converge)*
	DO 70 I = 1, 10, 0	*(Zero step size is not allowed)*

Counted Loop
Nesting

- One of the most common structures in Fortran is to *nest* one loop inside another, most frequently with *arrays* and *complex I/O*

- When nesting loops, inner loop must lie completely within the outer loop and the two loops *must* use different *LCV*

Counted Loop
Nesting

- General form of nesting loops

```
            DO sl1 LCV1=start1,stop1[,step1]
               DO sl2 LCV2=start2,stop2[,step2]
               ...
               series of instructions
               ...
sl2               CONTINUE
sl1            CONTINUE
```

where

sl1 sl2	statement label of outer and inner loop, respectively
LCV1 LCV2	loop control variable of outer and inner loop, respectively

Counted Loop
Nesting

- Alternative form of nesting loops

```
DO LCV1=start1,stop1[,step1]
   DO LCV2=start2,stop2[,step2]
   ...
   series of instructions
   ...
   END DO
END DO
```

where

LCV1	loop control variable of outer loop
LCV2	loop control variable of inner loop

Counted Loop
Nesting

- Code snippet

```
      READ *, I, J
      DO 10 OUTER=1, I
         DO 20 INNER = 1, J
            PRINT *, OUTER*INNER
20       CONTINUE
10    CONTINUE
      ...
      ...
```

Counted Loop
Nesting

- Alternatively

```
READ *, I, J
DO OUTER=1, I
   DO INNER = 1, J
      PRINT *, OUTER*INNER
   END DO
END DO
...
...
```

Counted Loop

- Study the flowchart of Examples 5.7. It is a combination of block `IF` and `DO` loop control structures
- Edit Example 5.7, compile and run it
- Have a close look at Example 5.8 on examples of properly and improperly nested `DO` loops
- Turn Example 5.9 into a complete program, compile and run it to study nested `DO` loop

EXAMPLE 5.7

The Fibonacci series is a famous sequence that dates back to the thirteenth century:

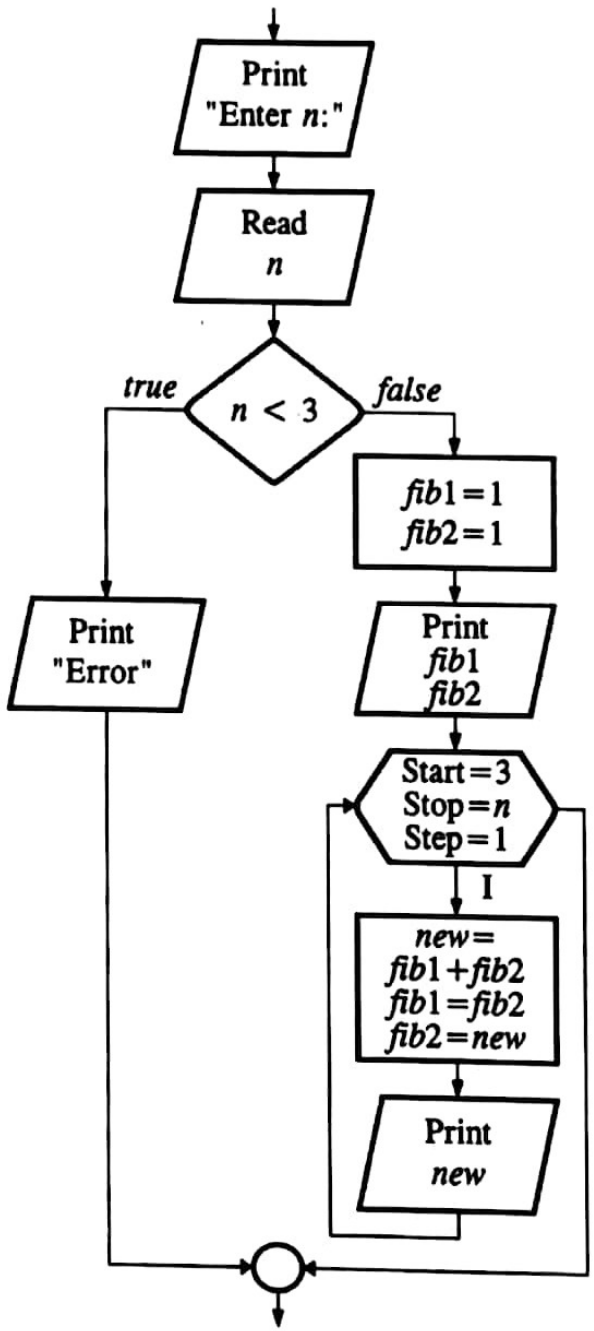
1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

The first two terms in the series are 1 and 1, but every term after that is the sum of the two previous terms. In this problem, we are going to calculate the series up to the *n*th term, where *n* is a number entered at execution time. In cases like this we have to be careful that the value of *n* is a valid number. Thus, in the program below, we will first see if *n* is a number less than 3 (with a block IF). If it is, then we will go ahead and compute *n* terms in the series (with a loop). Note that we will nest the DO loop within the block IF structure.

Program

```
C First we read in N
      INTEGER FIB1, FIB2
      PRINT *, 'Enter N:'
      READ *, N
C Now check to see if N is
C less than 3. If it is, then
C use the DO loop to compute
C N terms of the series.
      IF(N.LT.3) THEN
         PRINT *,'ERROR'
      ELSE
         FIB1=1
         FIB2=1
         PRINT *, FIB1, FIB2
         DO 10 I=3,N
            NEW=FIB1+FIB2
            FIB1 = FIB2
            FIB2 = NEW
            PRINT *, NEW
10       CONTINUE
      ENDIF
      END
```

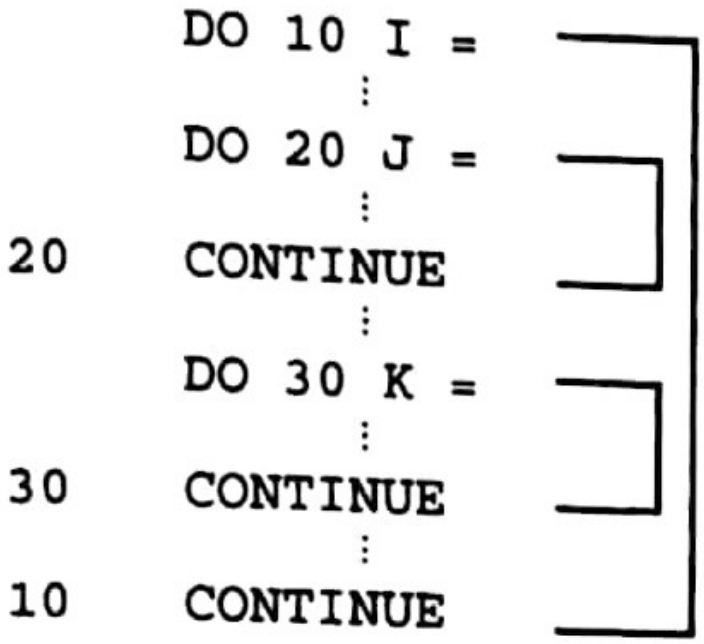
Flowchart



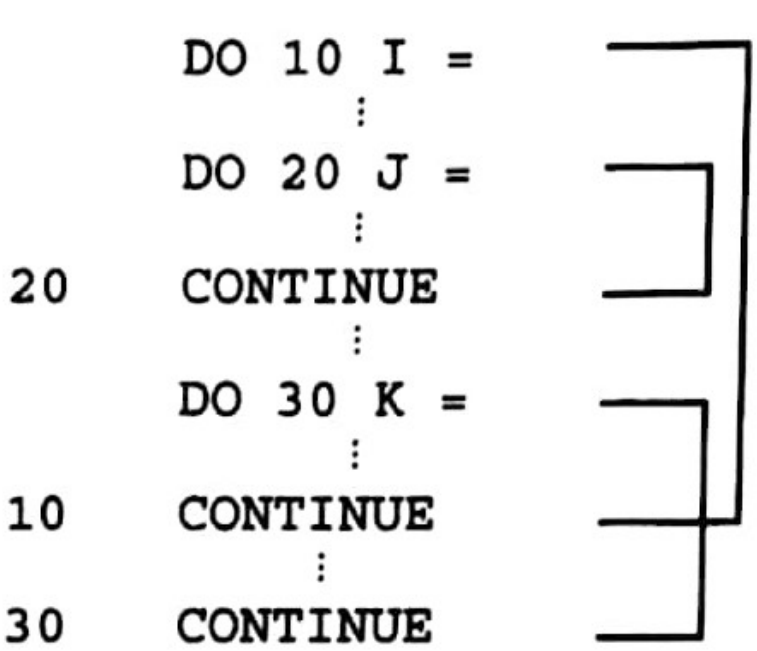
EXAMPLE 5.8

Here are examples of properly and improperly nested DO loops:

Properly Nested



Improperly Nested



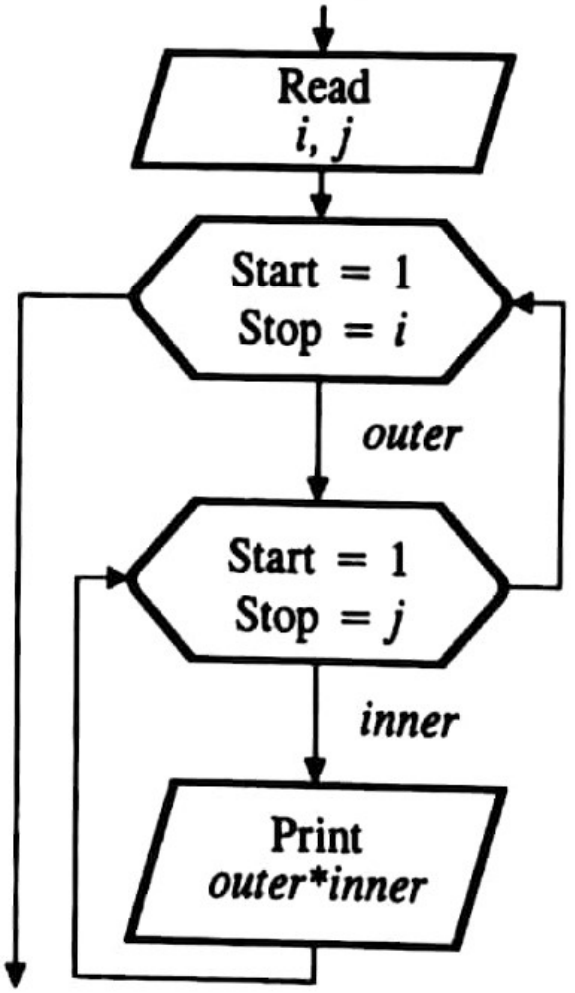
EXAMPLE 5.9

Here is an example of how to use nested loops to generate a simple multiplication table. At this point, we don't yet have the means to produce a nice square table. But at least this program will generate the values.

Program

```
      READ *, I , J
      DO 10 OUTER = 1, I
         DO 20 INNER = 1, J
            PRINT *, OUTER*INNER
20       CONTINUE
10    CONTINUE
```

Flowchart



Conditional Loop

- Available on most Fortran compiler, but a few may still NOT support it
- Should be used where we do not know in advance how often to execute the loop
- Built upon `DO WHILE` structure
- Those without `DO WHILE` capability may replace it by an equivalent structure using `IF-THEN-ELSE`s and `GO TO`s

Conditional Loop

- General form of `DO WHILE` structure

```
DO WHILE (condition is true)
   ...
   block of instructions
   ...
END DO
```

- If test condition is *true*, the *block of instructions* is executed

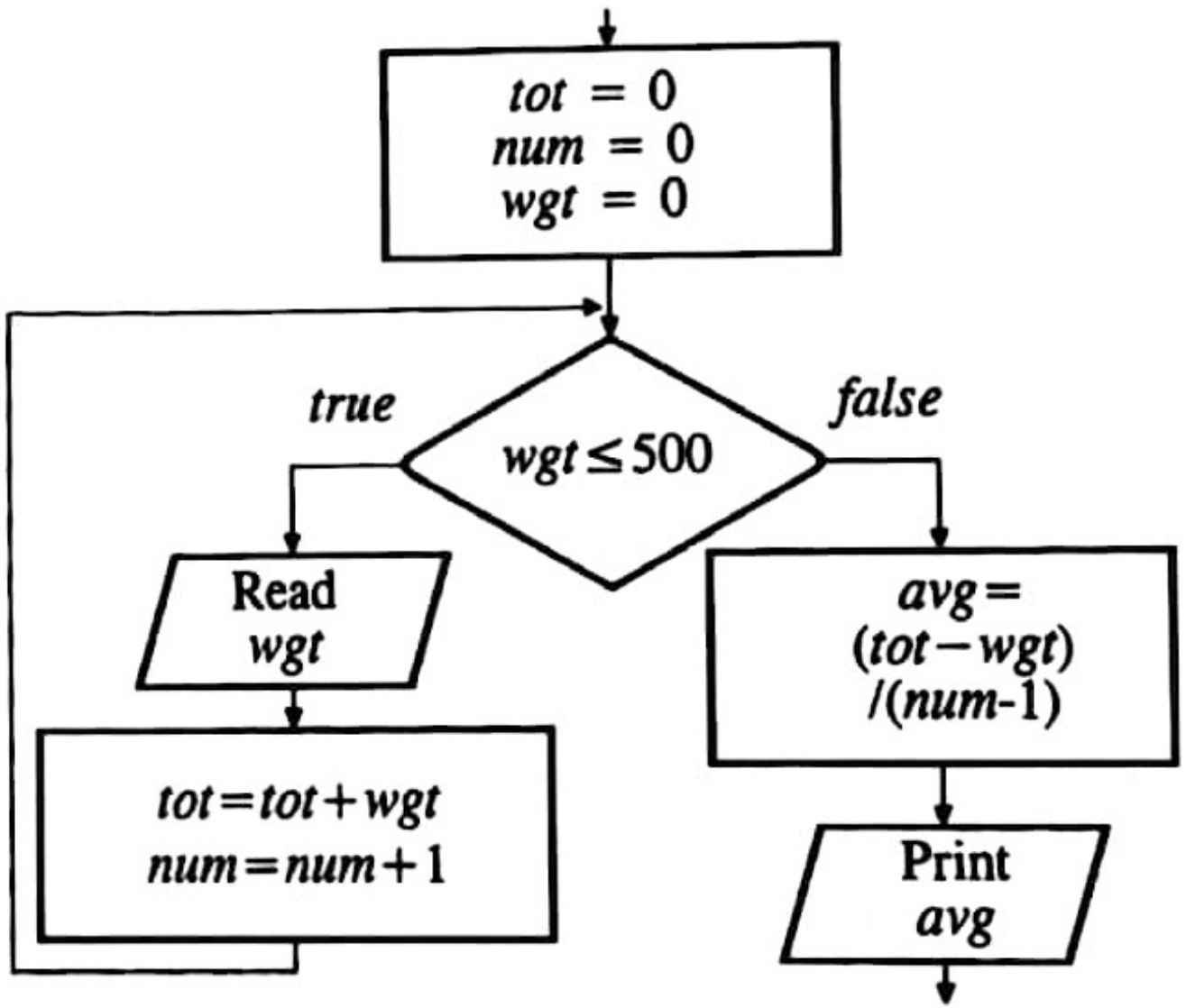
- If test condition is *false*, the loop terminates and control jumps to statement after end of loop

EXAMPLE 5.10

stop.

To demonstrate how this works, let's assume that we are calculating the average weight of rabbits in a laboratory. Since rabbits multiply so fast, we never know in advance how many there will be. So we set up the loop to read in the weights, one at a time, until one of the weights is greater than 500 pounds. When this occurs, the loop will stop. We sometimes call this special value a *sentinel* value. The loop is set up so that we watch for this key value, which we have chosen so that it is unlikely to be found in the data set. So when you enter this value, the program will recognize it as the signal to stop.

Flowchart


tot = 0
num = 0
wgt = 0
wgt ≤ 500
true
false
Read
wgt
tot = tot + wgt
num = num + 1
avg =
(tot − wgt)
/(num-1)
Print
avg

Conditional Loop

- Code snippet

```
TOT = 0.0
NUM = 0
WGT = 0.0
DO WHILE (WGT .LE. 500.0)
   PRINT *,'Enter Weight'
   READ *, WGT
   TOT = TOT + WGT
   NUM = NUM + 1
END DO
AVG = (TOT-WGT)/(NUM-1)
```

Conditional Loop

- Alternative way to construct the `DO WHILE` code snippet above

```
        TOT = 0.0
        NUM = 0
        WGT = 0.0
10      IF (WGT .LE. 500.0) THEN
            PRINT *,'Enter Weight'
            READ *, WGT
            TOT = TOT + WGT
            NUM = NUM + 1
            GO TO 10
         ELSE
            AVG = (TOT-WGT)/(NUM-1)
         END IF
```

5.15 Write a program to compute the value of b given by the series shown below. Continue computing the sum of the terms until the absolute value of any individual term falls below 0.01. By doing this, we evaluate the series for all terms that are significant. We will ignore any term whose value is so small that it has little effect on the series total.

Note in this series that the terms have an alternating sign. This is best handled by defining a variable SIGN whose initial value is set at 1.0. For each successive term in the series, we will multiply SIGN by -1.0, in effect, alternating the sign.

$$b = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots$$

5.17 Write a program to read in the radius r of a circle centered at the origin. Then read in the coordinate pairs (x, y) of a point and determine if that point lies within the circle. Use the condition that if

$$(x^2 + y^2)^{0.5} < r$$

then the point is inside the circle. Terminate the program the first time that $(x^2 + y^2)^{0.5} > 2r$.

5.20 Write a program to simulate a population explosion. Start out with a single bacteria cell that can produce an offspring by division every 4 hours. The new cell must incubate for 24 hours before it can divide. The parent cell meanwhile will continue to divide every 4 hours. Assume that any new cells will follow this pattern. How many cells will you have in 1 day, 1 week, and 1 month, if none of the new cells die?

```
C H24 is the number of cells that are 24 hours old or older.
c Similarly, H20 is the number of cells that are 20 hours old,
c H16 is the number of cells 16 hours old and so forth. Every
c four hours, we move the number stored in each variable to the
c next higher level. Thus, H16 receives the value from H12. The
c number of new cells created (NEWCEL) is the value stored in
c H24.
      H24=1
      PRINT *, 'Enter Number of hours'
      READ *, HOURS
      DO 10 I = 1, HOURS/4
         NEWCEL=H24
         H24=H24+H20
         H20=H16
         H16=H12
         H12=H8
         H8=H4
         H4=H0
         H0=NEWCEL
10    CONTINUE
      TOT=H0+H4+H8+H12+H16+H20+H24
      PRINT *, 'Number of Cells=', TOT
```