

# Basic Building Blocks of Programming

# Variables and Assignment


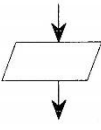
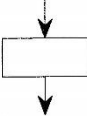
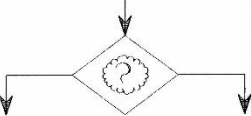
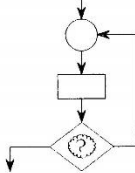
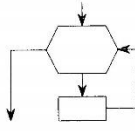
- Think of a variable as an empty container
- Assignment symbol (=) means putting a value into a variable (container)
  - This is not the same as ‘equal’
- Initialization
  - Before doing repetitive computations, usually needs initialization value (usually 0 or 1)
- Swapping
  - To swap values between two variables, we need a third dummy variable to temporarily store the value.

# Assignment (=)

- In mathematics,  $x = 1 - x$  is a statement of fact, which can be solved to get  $x = \frac{1}{2}$
- In programming  $x = 1 - x$  is a command to assign  $x$  with the value of the right hand side of the  $=$  symbol.
  - This will be read as  $x$  is now assigned with the value of 1 minus the *current value* of  $x$
- In programming, the LHS is always a single variable because a value is being assigned to it.

# Algorithm structures

- Sequential execution
- Branching
- Loops
- Nested structure
  
- Subroutines

Block	Function	Flowchart Symbol
Sequential Execution	Unconditional Transfer	
	Input or Output	
	Processing	
Branching	Conditional Transfer	
Loops	Conditional Loop	
	Counted Loop	

**Fig. 1-2 Basic building blocks of programming**

# Sequential execution

- Proper sequence is important.
- Following instructions in different sequences will give different results (or no result)

**EXAMPLE 1.1**

Construct an algorithm and a flowchart to compute the weight  $w$  of a hollow sphere of diameter  $d$ , wall thickness  $t$ , and density  $\rho$ , using the following equations:

$$r_o = \frac{d}{2} \qquad r_i = \frac{d}{2} - t$$

$$v = \frac{4}{3} \pi (r_o^3 - r_i^3) \qquad w = \rho v$$

We start with the calculation of the outside and inside radii ( $r_o$  and  $r_i$ ), based on the values of the diameter  $d$  and thickness  $t$ . From these, we next calculate the volume  $v$  of the hollow sphere. Finally, we calculate the weight, which is just the volume times the density, or  $\rho v$ . Here are the algorithm and flowchart to perform these instructions:

**Algorithm**

Compute inner and outer radii by

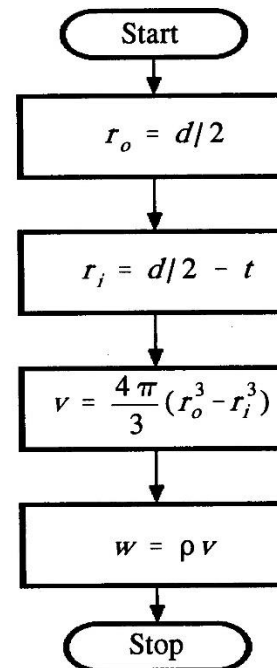
$$r_o = \frac{d}{2} \qquad r_i = \frac{d}{2} - t$$

Compute volume of sphere by

$$v = \frac{4}{3} \pi (r_o^3 - r_i^3)$$

Compute weight of sphere by

$$w = \rho v$$

**Flowchart**




### EXAMPLE 1.3

Rewrite Example 1.1 and Example 1.2 to allow for repeated calculation of the weight of a hollow sphere of diameter  $d$ , wall thickness  $t$ , and density  $\rho$ . This time, however, allow for input of different values of  $d$ ,  $t$ , and  $\rho$  before each calculation of  $w$  and  $v$ .

#### Algorithm

Enter values for  $d$ ,  $t$  and  $\rho$   
Compute inner and outer radii by

$$r_o = \frac{d}{2} \quad r_i = \frac{d}{2} - t$$

Compute volume of sphere by

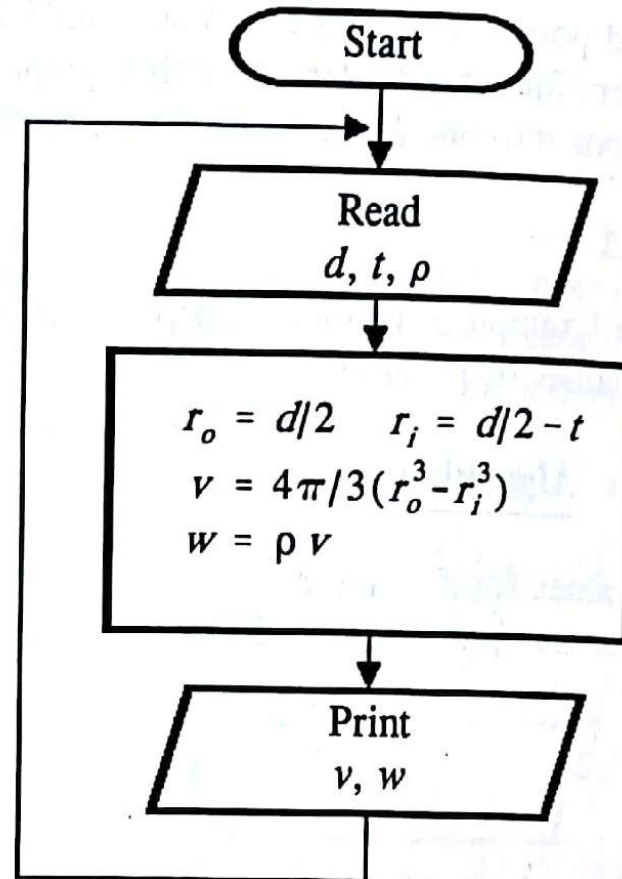
$$v = \frac{4}{3} \pi (r_o^3 - r_i^3)$$

Compute weight of sphere by

$$w = \rho v$$

Print out values of  $v$  and  $w$   
Go back to beginning

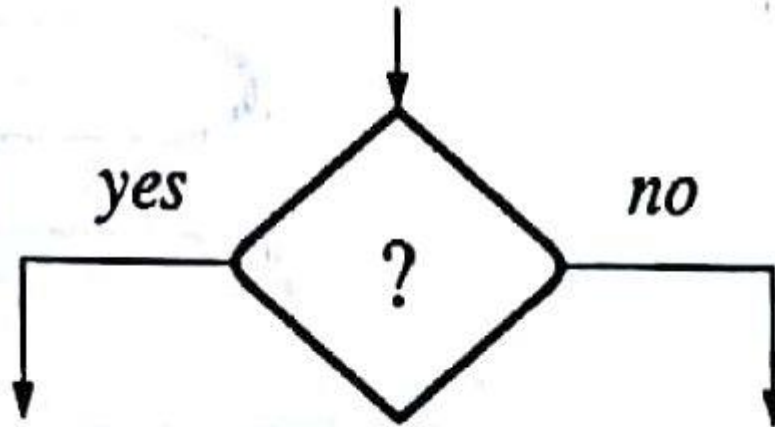
#### Flowchart





# Branching

- Also called
  - Conditional structure
  - Decision
  - IF statement
  - Selection

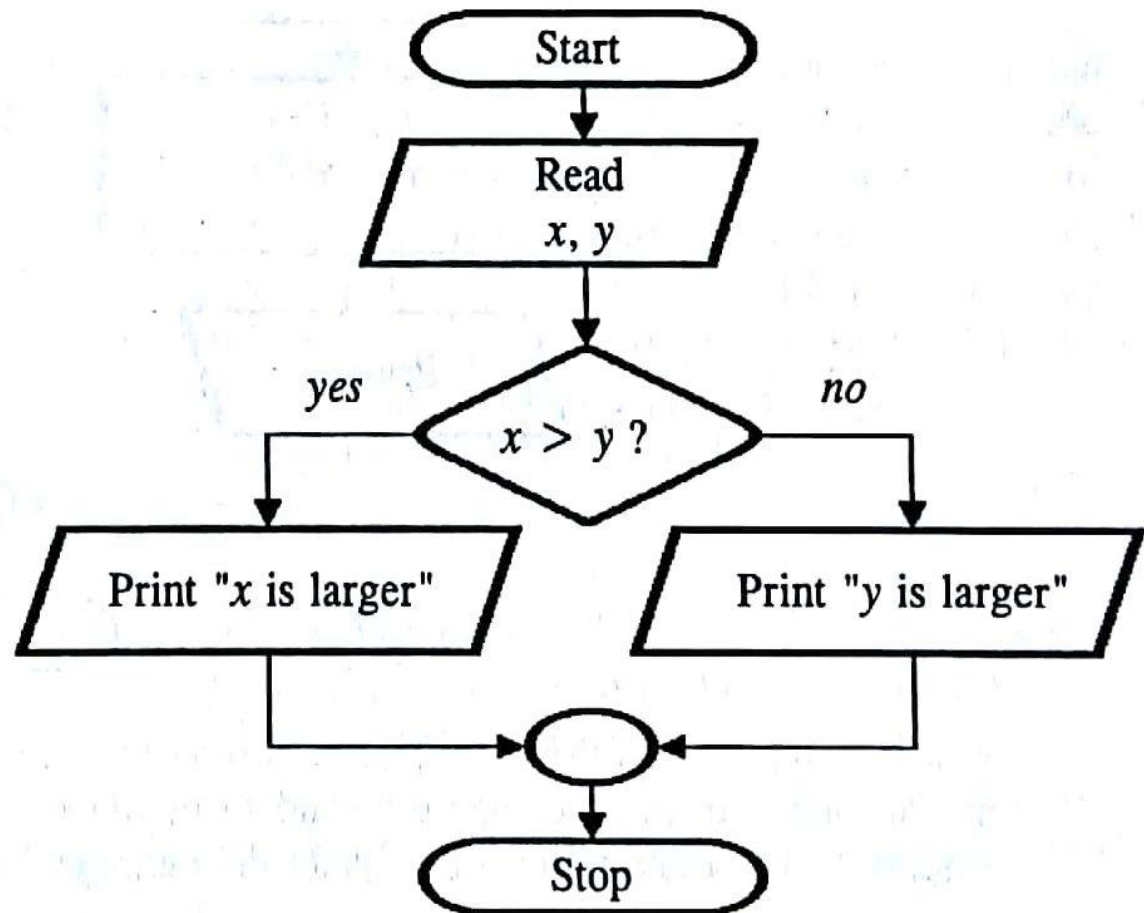


**EXAMPLE 1.4**

Construct an algorithm and flowchart to read two numbers and determine which is larger.

**Algorithm**

Enter values of  $x$  and  $y$   
Is  $x$  larger than  $y$ ?  
    If yes, print "x is larger"  
    If no, print "y is larger"  
End Branch

**Flowchart**

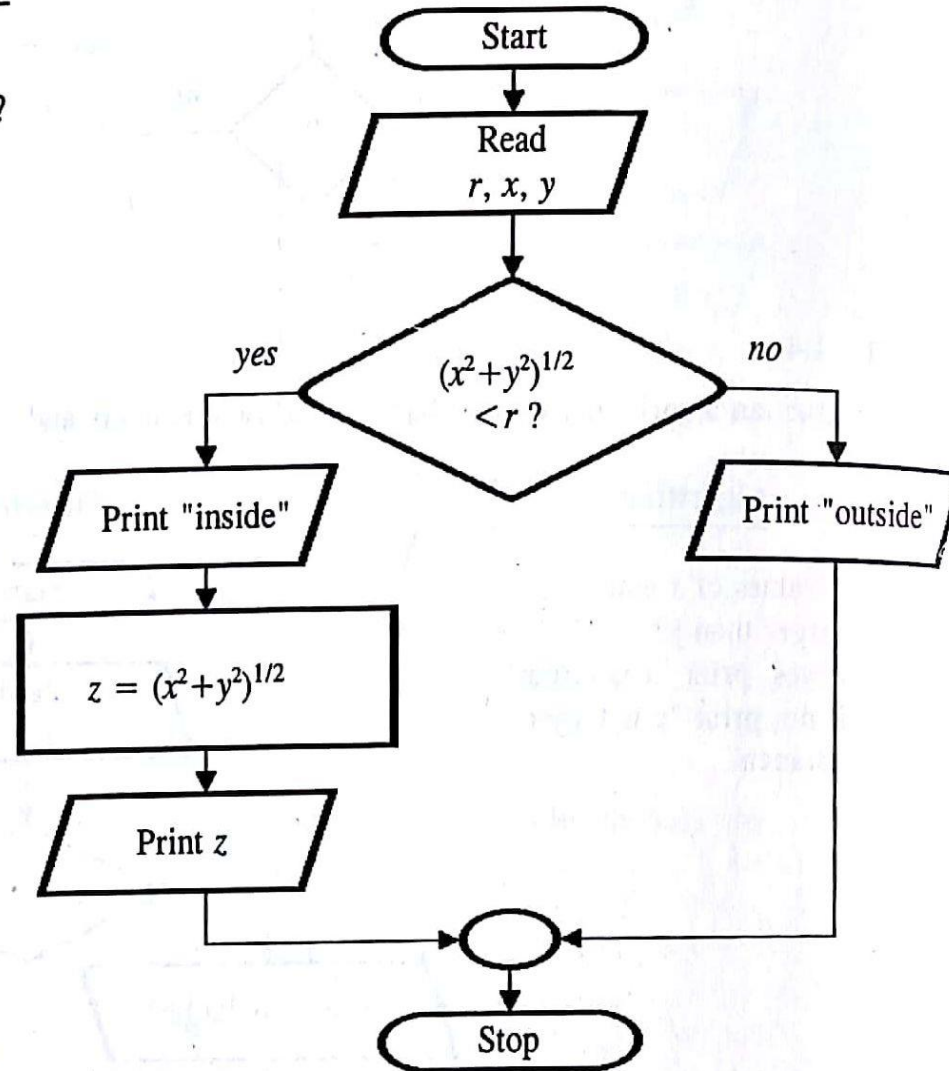
### EXAMPLE 1.5

Construct an algorithm and flowchart to determine if a point  $(x, y)$  lies within a circle of radius,  $r$ , centered at the origin. Use the condition that if  $(x^2 + y^2)^{1/2} < r$ , then the point is within the circle. If the point lies within the circle, print out a message and the distance,  $z$ , of that point from the center of the circle.

#### Algorithm

Read in  $r$  and  $(x, y)$   
 Is  $(x^2 + y^2)^{1/2} < r$ ?  
 If yes, then  
 print "inside"  
 compute  $z$   
 print  $z$   
 If no, then  
 print "outside"  
 End Branch

#### Flowchart



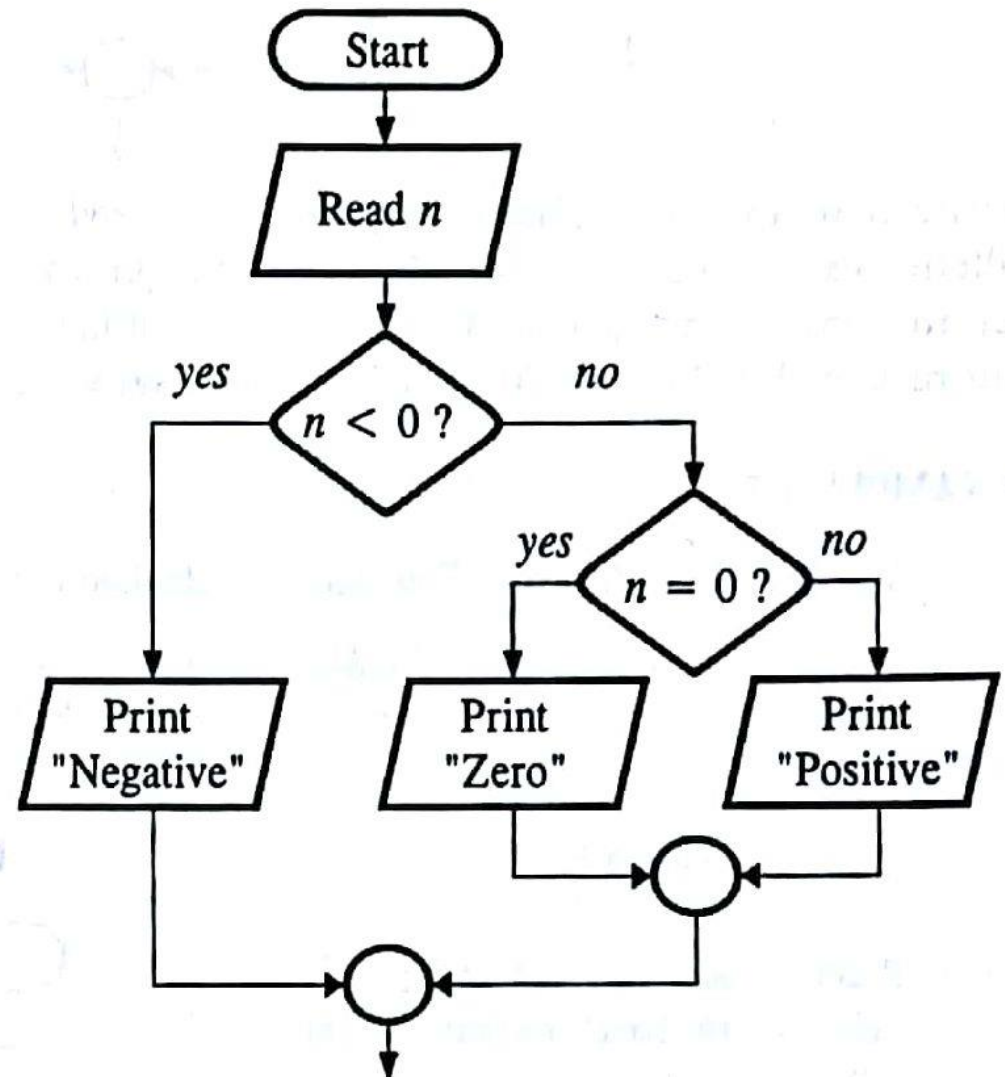
## EXAMPLE 1.6 Nested IF Statement

Construct an algorithm and flowchart to see if a number  $n$  is negative, positive, or zero.

### Algorithm

Read in  $n$   
Is  $n < 0$ ?  
    If yes,  $n$  is negative  
    print "Negative"  
    If no, is  $n = 0$ ?  
    If yes,  $n$  is zero  
    print "Zero"  
    If no,  $n$  is positive  
    print "Positive"  
    End Branch  
End Branch

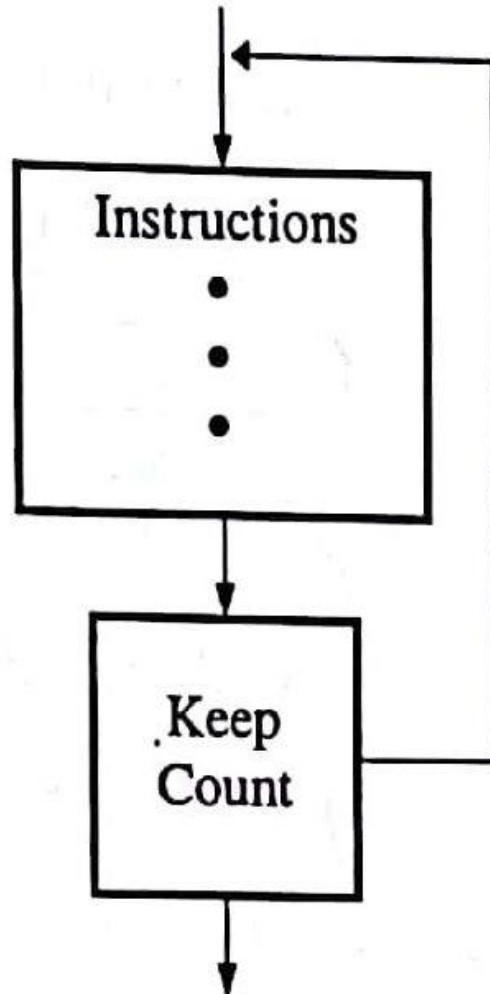
### Flowchart



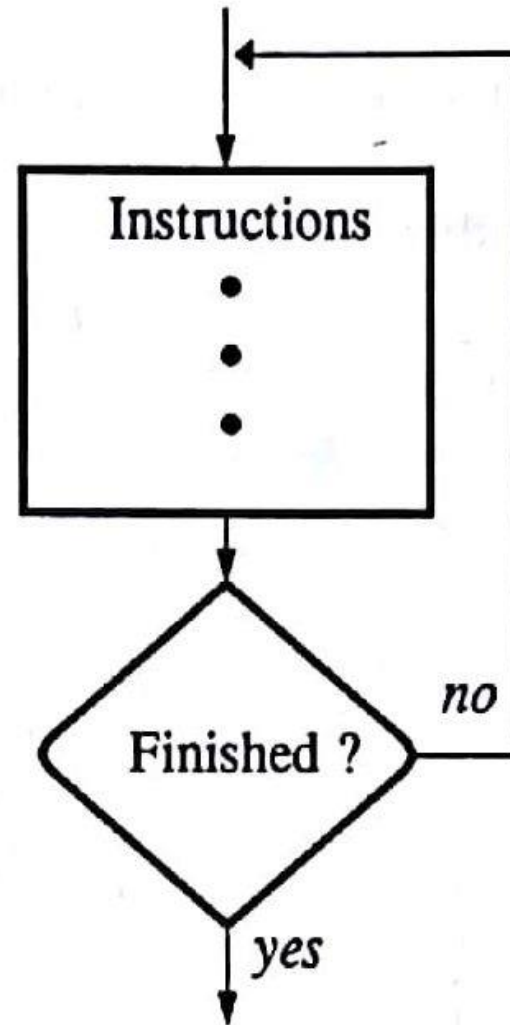
# Loops

- Also called *repetition, recursion*
- Two types
  - *Counted*
  - *Conditional*
- Avoid infinite loops!
- Loops are controlled by LCV (*Loop Control Variable*) which determines when to exit the loop

# Loops

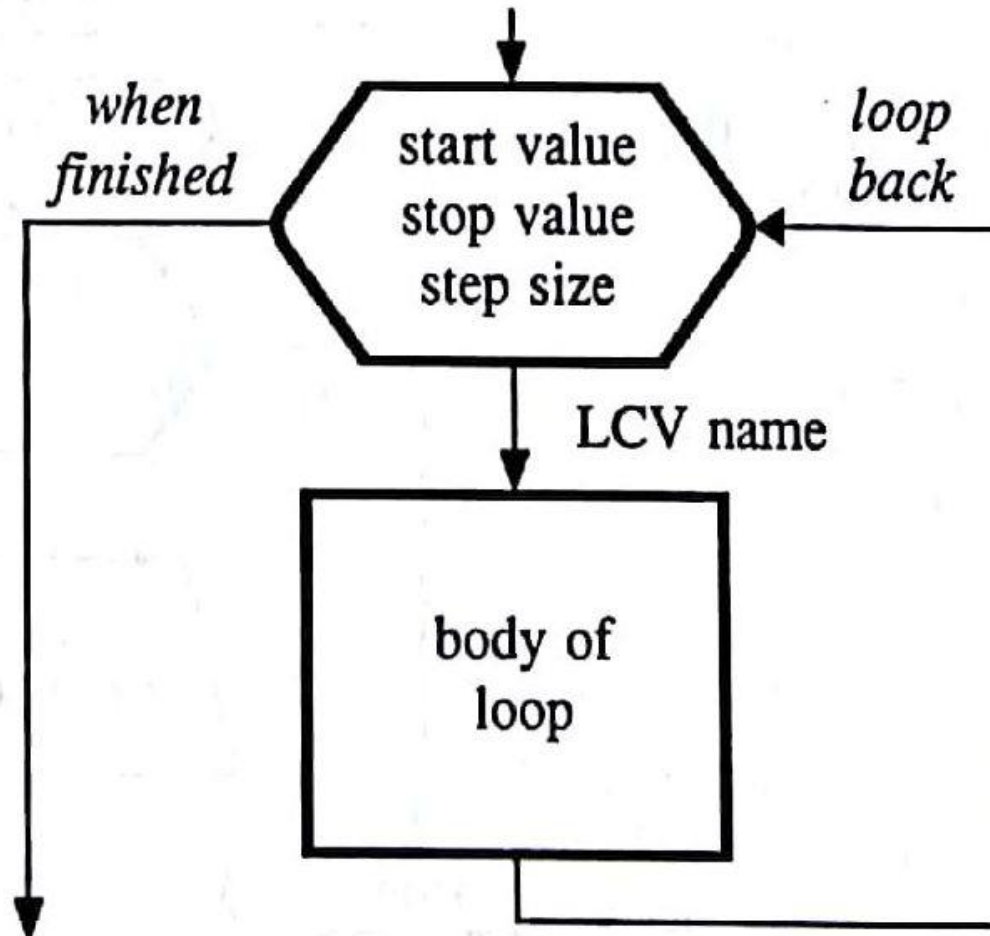


(a) Counted Loop



(b) Conditional Loop

# Counted Loop



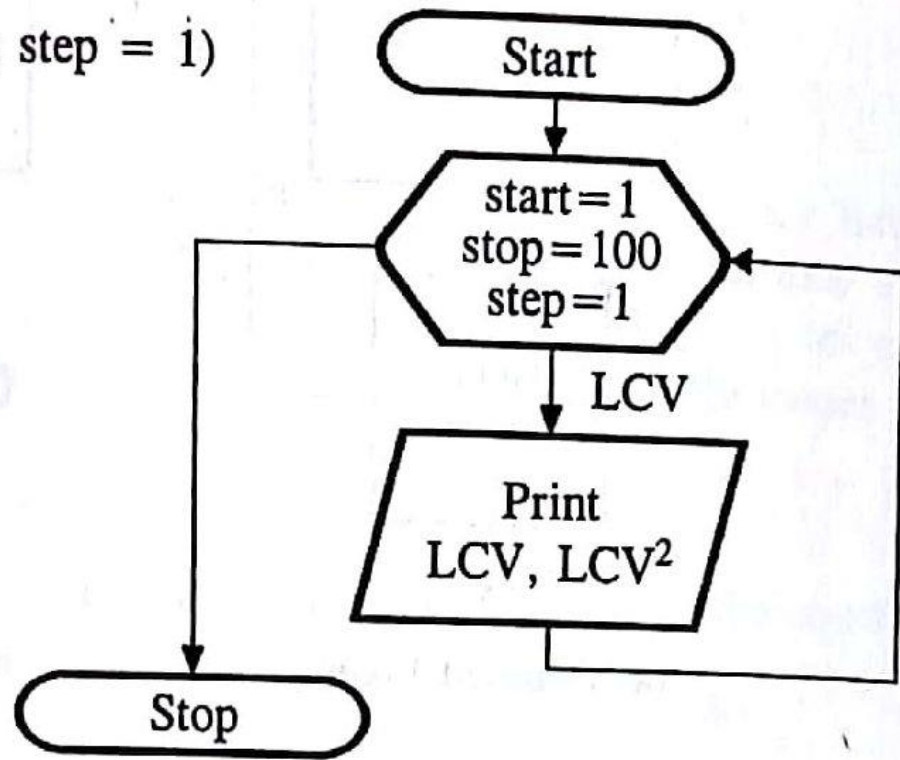


**EXAMPLE 1.8**

Write an algorithm and flowchart to print out the numbers 1 to 100 and their squares.

**Algorithm**

Loop (LCV start = 1; stop = 100; step = 1)  
  Print LCV value and  $LCV^2$   
End Loop

**Flowchart**



# Nested Loop

## EXAMPLE 1.11

Construct an algorithm and flowchart to create a 10 by 10 multiplication table such as  $1 \times 1 = 1$ ,  $1 \times 2 = 2$ , and so forth. This problem is best solved by nesting two counted loops. One loop keeps the value of the first number constant, while the second loop changes the second number from one to ten.

---

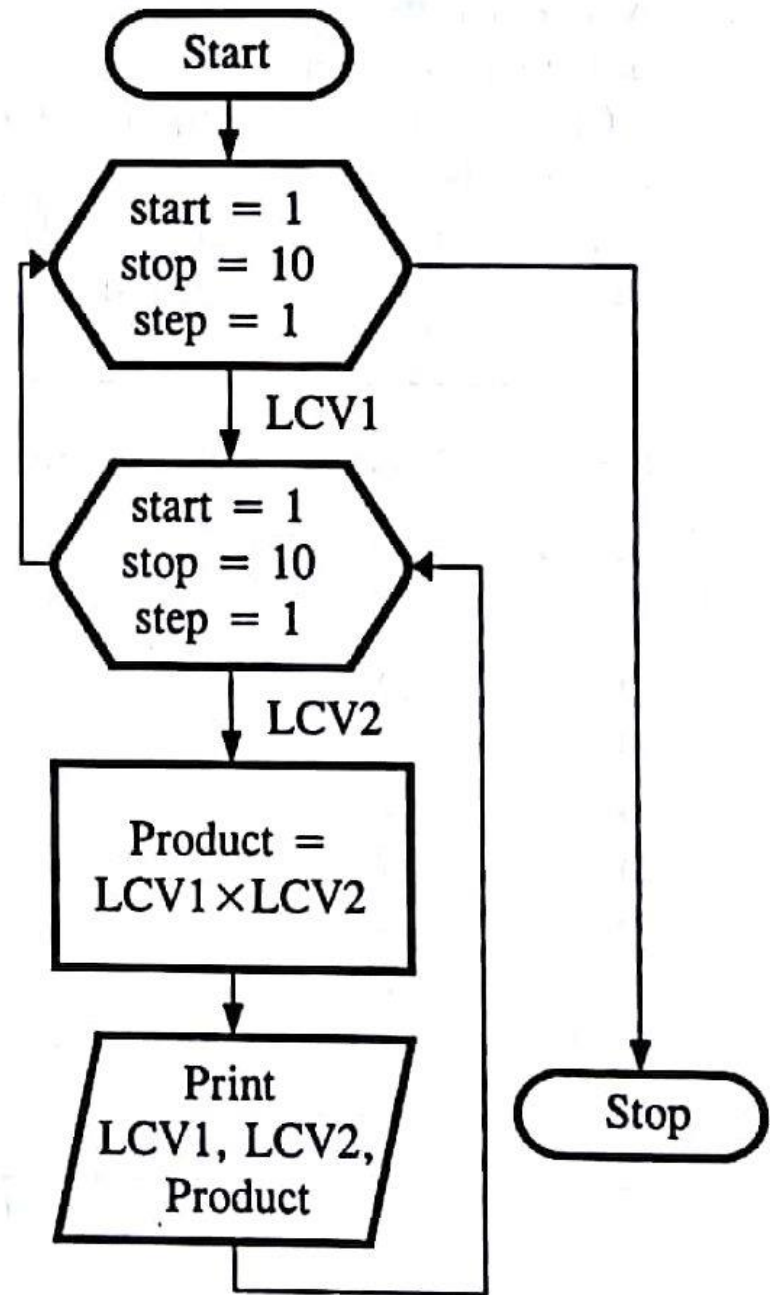
Value of LCV1 in Outer Loop	Value of LCV2 in Inner Loop	Product (LCV1 × LCV2)
1	1	1
1	2	2
:	:	:
2	1	2
2	2	4
:	:	:
10	9	90
10	10	100

---

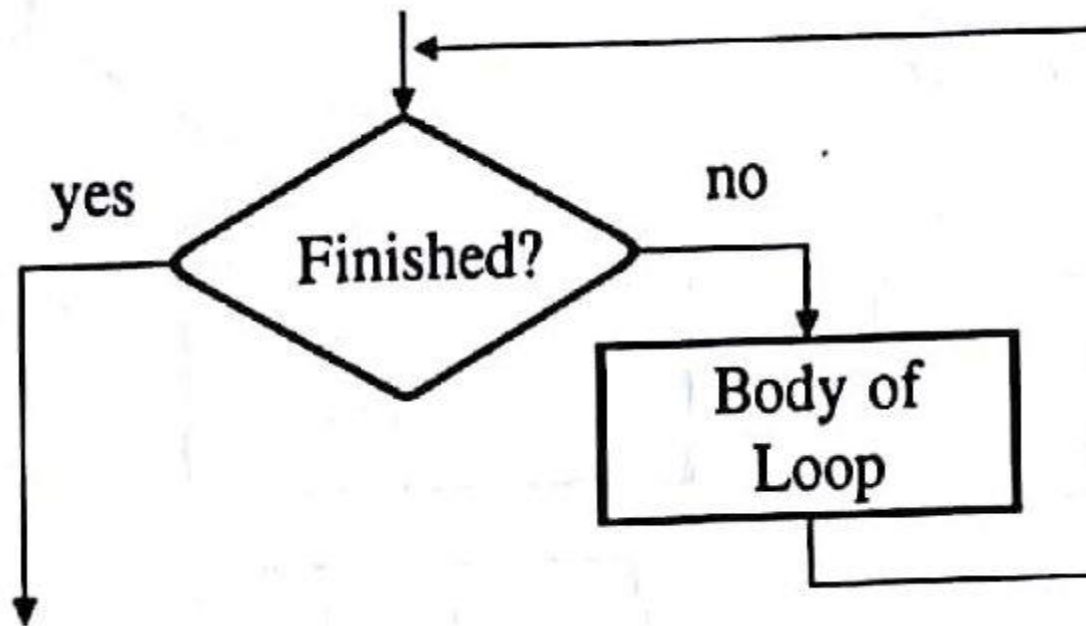
## Algorithm

Loop (LCV1 start = 1; stop = 10; step = 1)  
  Loop (LCV2 start = 1; stop = 10; step = 1)  
    Product = LCV1 × LCV2  
    Print LCV1, LCV2, and Product  
  End Loop  
End Loop

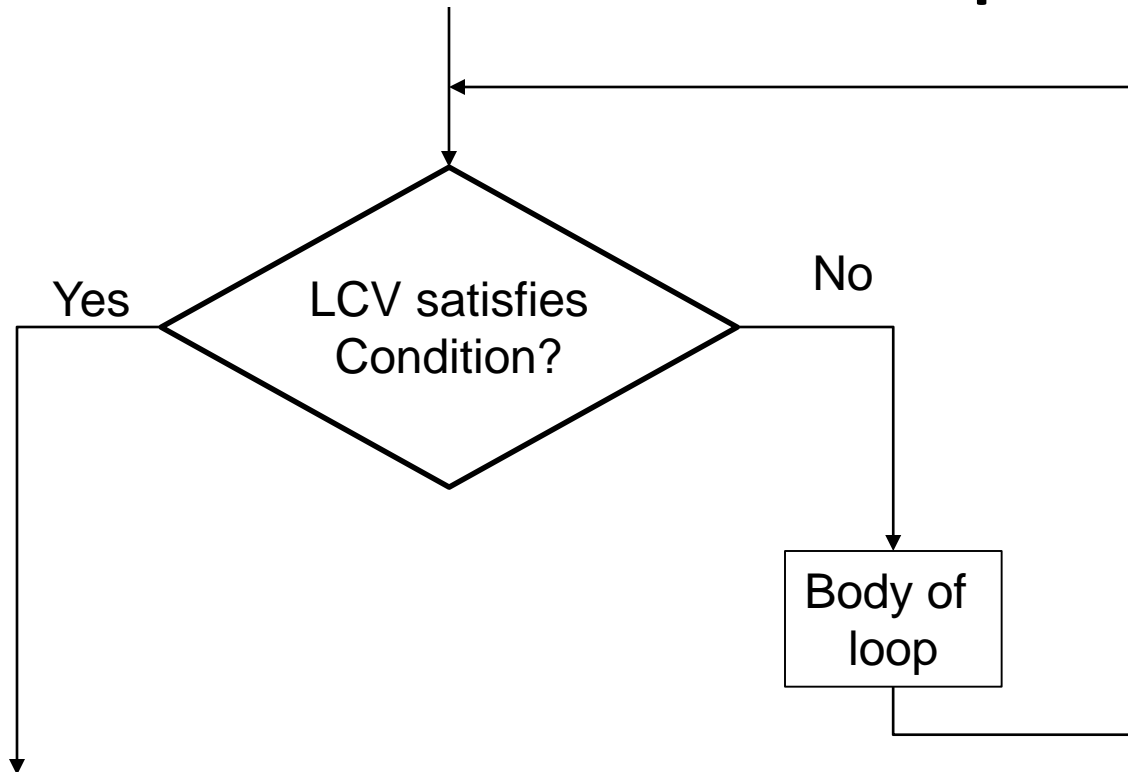
## Flowchart



# Conditional Loop



# Conditional Loop



- LCV must change inside the *body of loop* to finally satisfy *condition*.
- Otherwise it will result in an *infinite loop*.

# Example

**1.12** Construct an algorithm and a flowchart to compute an approximation to the series:

$$\frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3} + \dots$$

This series continues indefinitely and it is impossible to complete the computation. Yet we can *estimate* the series value by carrying out the computation until a term in the series adds a negligible amount to the total sum of all the previous terms. The way that we will do this is by performing the computation until any term falls below a critical value ( $\epsilon$ ) that you read in. As an example, if we read in a value of  $\epsilon=0.005$  each term will be evaluated and added to the total until any individual term becomes smaller than 0.005 as shown below.

---

Term	Sum	Comments
$1/1^3$	1.0	<i>Term (1.000) &gt; <math>\epsilon</math> (0.005), so continue series</i>
$1/2^3$	1.125	<i>Term (0.125) &gt; <math>\epsilon</math> (0.005), so continue series</i>
$1/3^3$	1.162	<i>Term (0.037) &gt; <math>\epsilon</math> (0.005), so continue series</i>
$1/4^3$	1.178	<i>Term (0.016) &gt; <math>\epsilon</math> (0.005), so continue series</i>
$1/5^3$	1.186	<i>Term (0.008) &gt; <math>\epsilon</math> (0.005), so continue series</i>
$1/6^3$	1.191	<i>Term (0.0046) &lt; <math>\epsilon</math> (0.005), so stop</i>

---



## Algorithm

Read  $\epsilon$

$term = 1$

$sum = 1$

$count = 2$

Loop: (While  $term \geq \epsilon$ )

$term = 1/count^3$

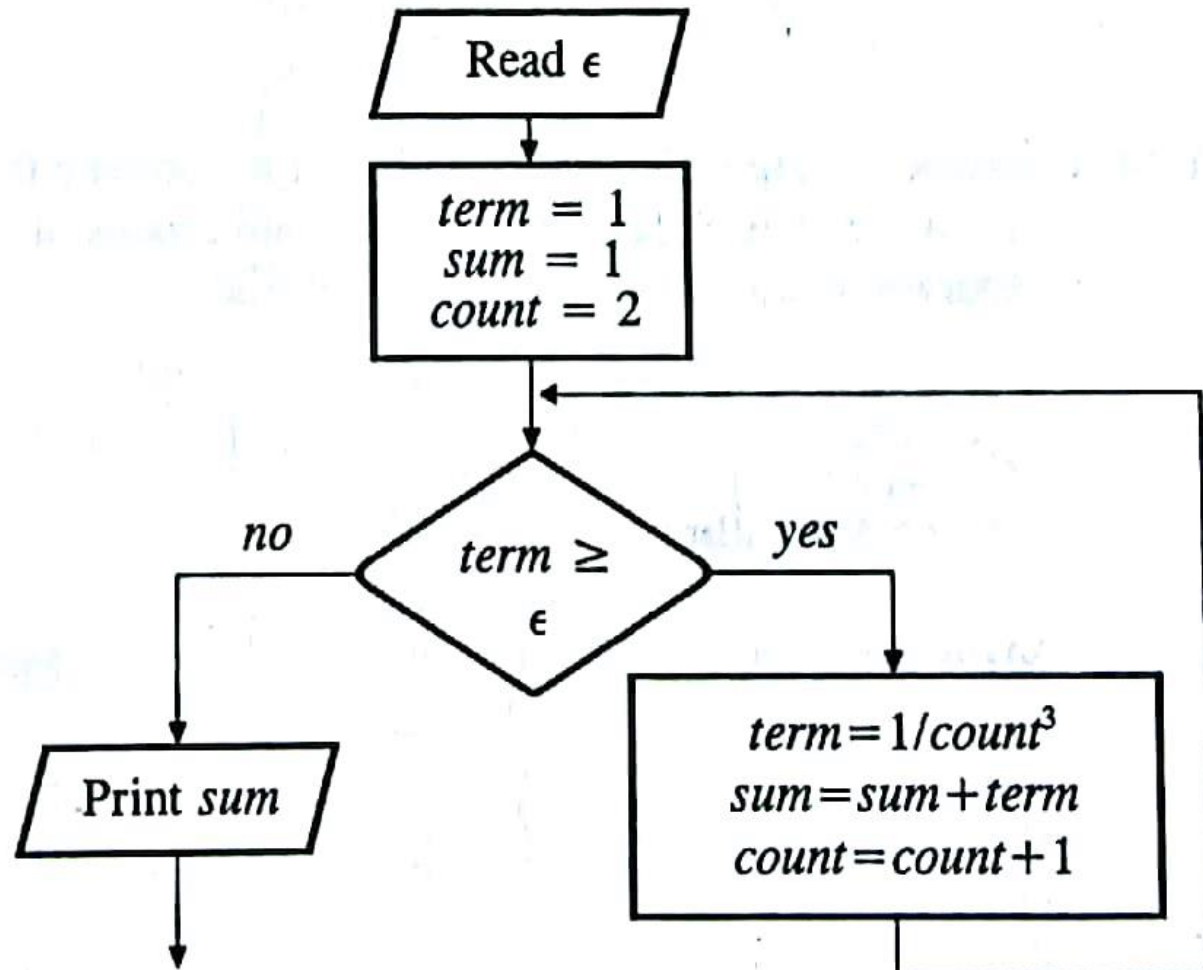
$sum = sum + term$

$count = count + 1$

End Loop

Print  $sum$

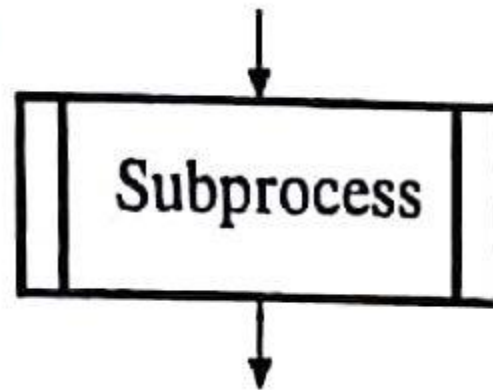
## Flowchart





# Subroutines

- Also called *functions, procedures, or modules*
- Common sub-procedures are often made into small subprograms for reuse.
- Useful for large problems when it is broken up into small sub-problems.

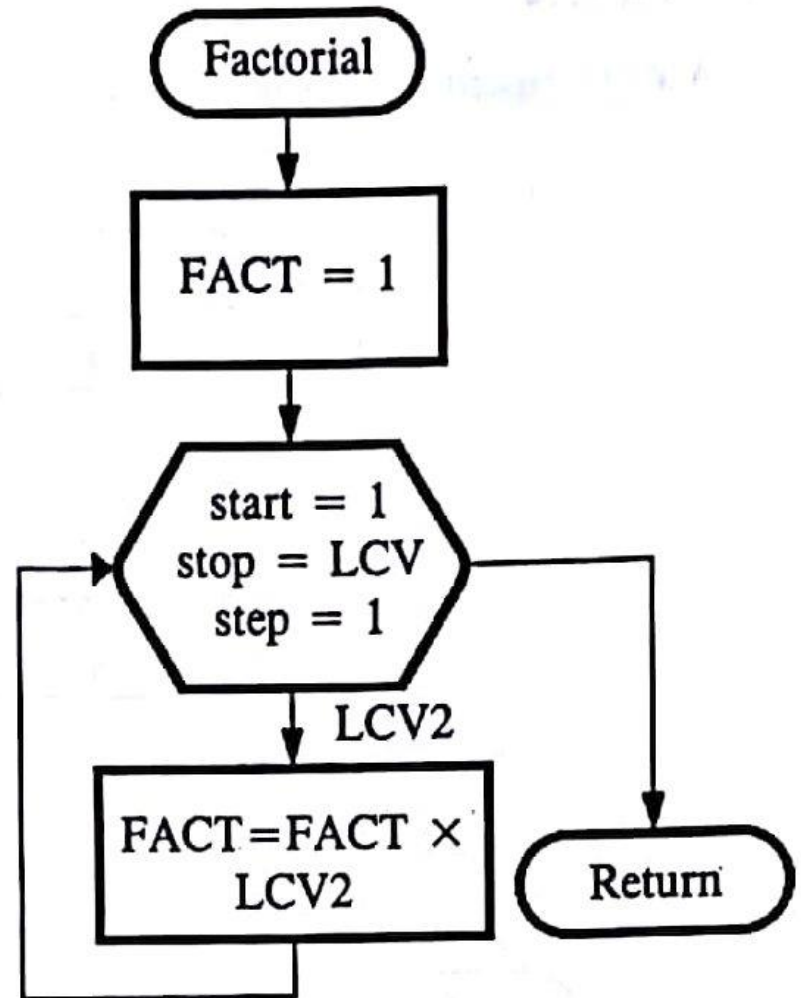
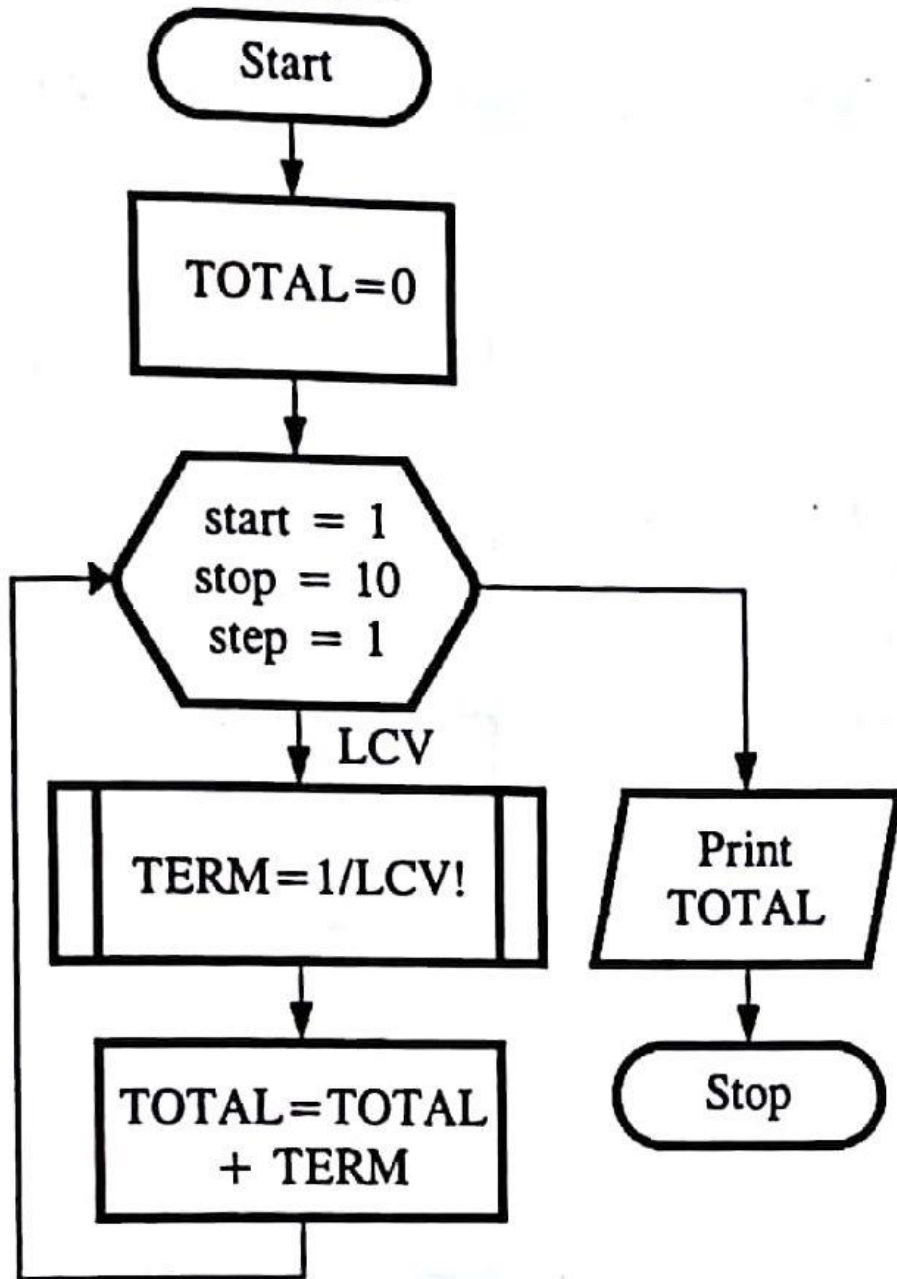


**EXAMPLE 1.13**

Construct an algorithm and a flowchart to evaluate the first ten terms of the infinite series:

$$a = \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

# Flowchart



# Patterns and Structure

- Separate data and variables
  - Look at empty variables
- Look for patterns and structure of empty variables to help in constructing algorithms
  - Loops (repetitions)
  - Conditions (branching)

# Walk-through

- Also called *tracing* an algorithm
- To check whether the algorithm would produce the expected result

1.16 Trace through the following flowchart (or algorithm) and predict the output.

### Algorithm

$f1 = 1$

$f2 = 1$

Print  $f1, f2$

Loop (LCV start = 3, stop = 10, step = 1)

$f3 = f1 + f2$

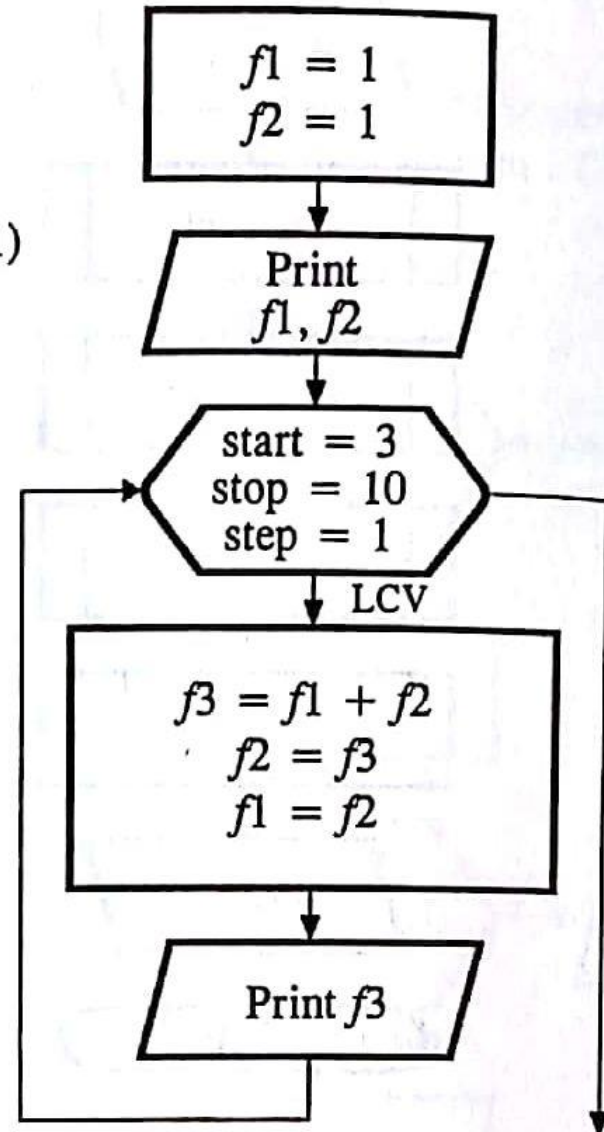
$f2 = f3$

$f1 = f2$

Print  $f3$

End Loop

### Flowchart



# Debugging Tip

- Insert *print* command into places in the program where you suspect the error is occurring and see whether the output is what it should be.