Topic 2
Programming Basics for Problem Solving

What is computer solving problem?

- It refers to the entire process of taking the statement of a problem and developing a computer program that solves problem.
  - Programming = process of problem solving

Steps in the Program Development Process

- Four main steps are involved in the program development process:

1. **Analysis**
2. **Design**
3. **Implementation**
4. **Test**

Analysis

- Analyse precisely what the program is supposed to accomplish (requirements gathering).
- Specify Inputs/Outputs of the desired solution.

If you don't know exactly where are you going, how will you know when you get there?

Quote by Steve Maraboli

Design

- Produce an ordered sequence of steps that describe the solution of the problem.
- This sequence of steps is called an **algorithm**.
- **Flowchart** is another tool to describe the solution.

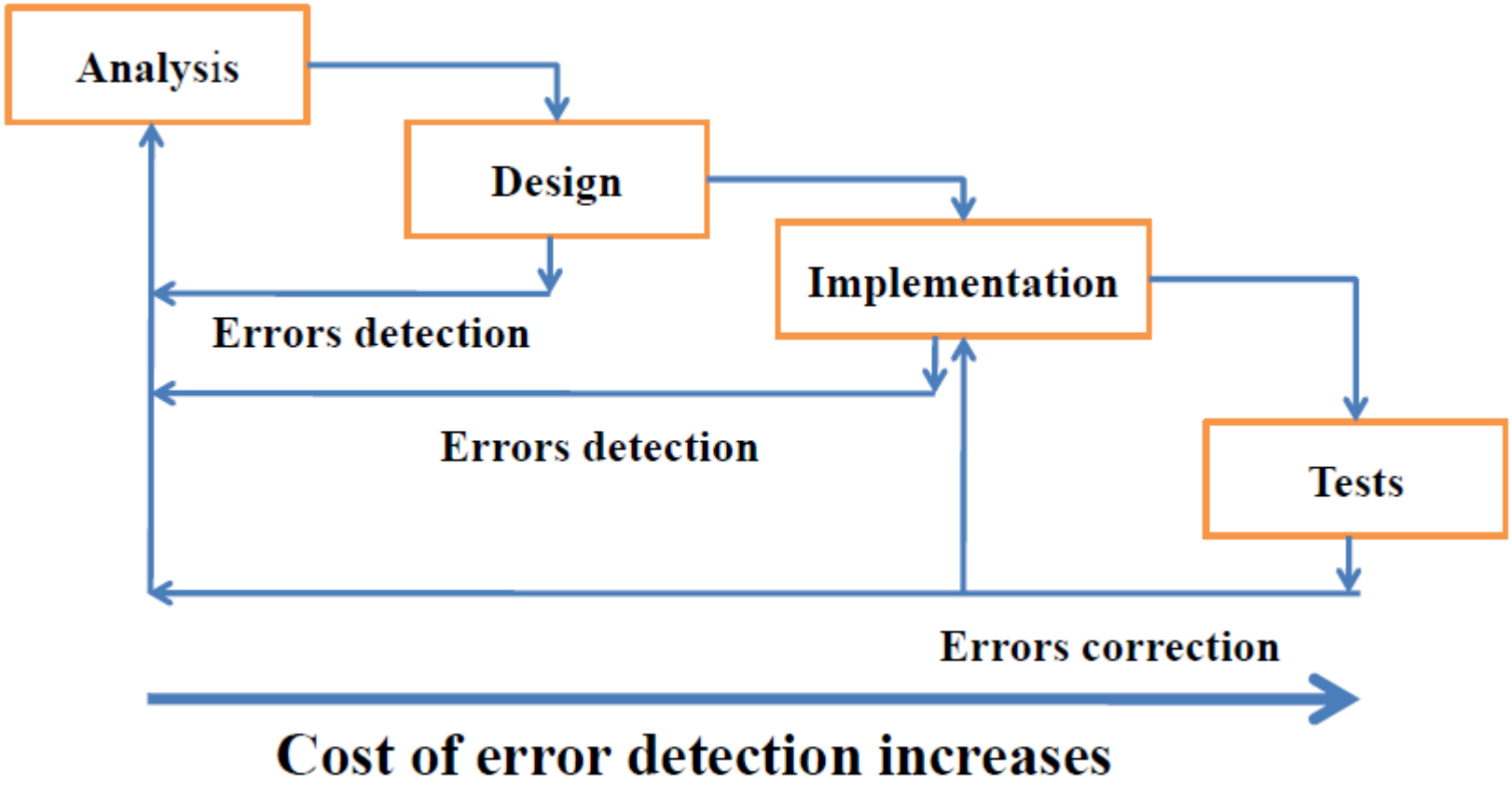
Implementation(Coding)

- Implement the program in some programming language.
- Translation of the design into **source code**.
  - Source code: Set of statements written in a certain programming languages such as C.

Test and Debug

- It is for making sure that your program is running correctly once all existing **bugs** (errors) are located and removed.

Waterfall Model



Algorithms and Flowcharts

- Algorithm and flowchart are the powerful tools for learning programming. An algorithm is a step-by-step design of the process, while a flowchart explains the steps of a program in a graphical way.
- Algorithm and flowcharts helps to clarify all the steps for solving the problem.

Algorithm

- Algorithmic is a term of Arab origin. It is derived from the name of the famous Persian mathematician **"Al Khawarizmi"** (780-850) author of his famous book **"Al Djabr"**, in which he introduced the fundamental algebraic methods and techniques for solving equations.
- Algorithms are the building blocks of computer programs. They are as important to programming as **recipes** are to cooking.
- An algorithm is a well-defined procedure that takes input and produces output. The analogy to a recipe is good here, since a cook will take ingredients (the **input**), mix things together and cook it (the **procedure**), and produce a dish (the **output**).

Definition of an Algorithm

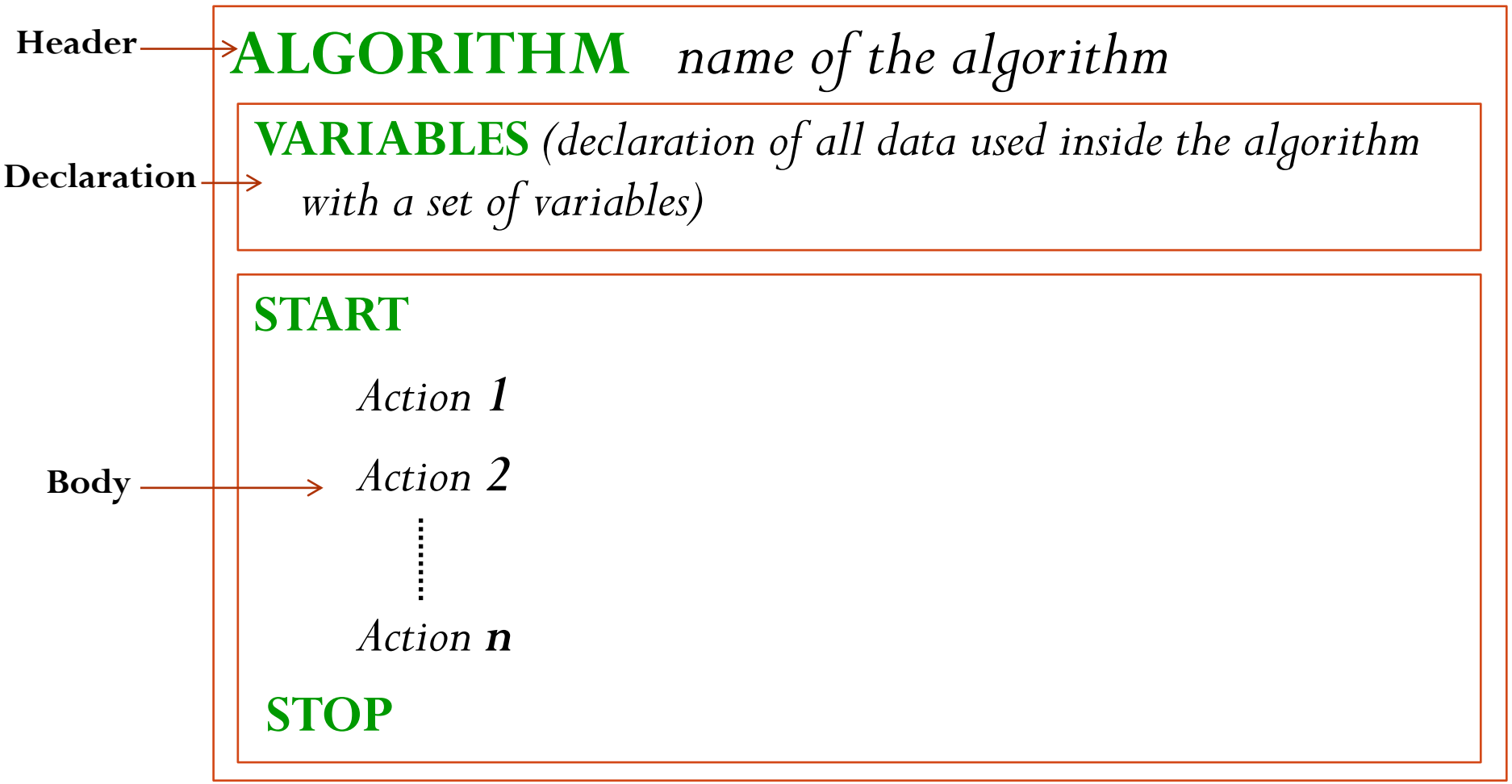
- An algorithm is a procedure (a well-ordered collection of instructions) for solving a problem in finite number of steps.

Why Algorithm?

An algorithm uses a formalism (set of words, structures, rules) whose purpose is:

- Offer a common language understood by everyone and which does not depend on a programming language.
- Facilitates the communication of the solution of a problem.
- Ensure a better design of a solution before its translation to computer program.
- The translation of the design to different programming languages is more simpler than translating a computer program from one programming language.

General Structure of an Algorithm



Variables

- Almost every algorithm contains data and the data is contained in what is called a **variable**.
- It is called variable because its value may vary during the execution of the algorithm.
- A variable may refer to an **input** data, **output** data, or **intermediate** data.
- To make it clear, it is preferably to choose meaningful names for variables. Choose variable names in relation with the given problem.
- Examples of variable names: *Nbr1, Nbr2, Value1, Value2, Sum, Average, Surface, Total_price, …etc*

Variables (cont.)

- A variable name should followed with its **type**.
- Commonly used data types are: **integer**, **real**, **character**, and **boolean**.
- Examples:
  - Integer: 5, 23, 0, -1, -52, …etc
  - Real: 2.5, 46.125, -7.09, …etc
  - Character: ‘A’, ‘r’, ‘?’, ‘&’, ‘1’, …etc
  - Boolean: true, false.

Features Used in Making Algorithms

Three main features are used in making algorithms:

- **Sequence** (stepping)
- **Decision** (choosing)
- **Repetition** (looping).

1. Sequence

- It is made by a sequence of statements placed one after the other.
- The instruction above or before gets executed first.
- The main instructions used here are:
  - Read instruction
  - Write instruction
  - Expressions to manipulate data

Read/Write instructions

- To accept data from the user, we use ***Input*** or ***Read*** statement.
- To display data for the user, we use ***Print*** or ***Write*** statement.
- Examples:
  - Read A
  - Read (V1, V2)
  - Write Sum
  - Write (Circumference, Surface)

Expressions

- An **expression** is a **mathematical** phrase that can contain ordinary numbers, variables (like x or y) and operators (like add,subtract,multiply, and divide).
  - Examples: **a+1**, **a-b**, **(3*x-y)/2**.

- Operators used in writing expressions in algorithms:
  1. Assignment operator: The assignment operator (←) is used for assigning the result of an expression to a variable.

     Example: **Z←(3*X-Y)/2**
  2. Arithmetic operators: **+, -, *, /, %**
  3. Relational operators: **>, <, >=, <=, =, !=**
  4. Logical operators: **AND**, **OR**, **NOT**

Example1

- Write an algorithm for finding the sum of two inputted integer numbers.

Example1- Solution

Algorithm Sum_of_two_numbers

Variables x,y,sum: integers

Start

Read (x,y)

sum←x+y

Write sum

Stop

Example2

- Write an algorithm for finding the average value of four inputted integer numbers.

Example2- Solution1

Algorithm Average_of_four_numbers

Variables x1,x2,x3,x4: integers

avg: real

Start

Read (x1,x2,x3,x4)

avg←(x1+x2+x3+x4)/4

Write avg

Stop

Example2- Solution2

Algorithm Average_of_four_numbers

Variables x1,x2,x3,x4,sum: integers

avg: real

Start

Read (x1,x2,x3,x4)

sum←x1+x2+x3+x4

avg←sum/4

Write avg

Stop

Example3

- Write an algorithm for finding the volume of a cone by given its diameter and its height.

Hint: the volume of cone having a radius ***r*** and height ***h*** is given by the following formula:

$$V = \frac{\pi r^2 h}{3}$$

Example3- Solution1

Algorithm Volume_of_Cone

Variables d,h,v,r: reals

Start

Read (d,h)

r←d/2

v ←(3.14*r*r*h)/3

Write v

Stop

Example3- Solution2

Algorithm Volume_of_Cone

Variables d,h,v,r,pi,r2: reals

Start

Read (d,h)

pi←3.14

r←d/2

r2←r*r

v ←(pi*r2*h)/3

Write v

Stop

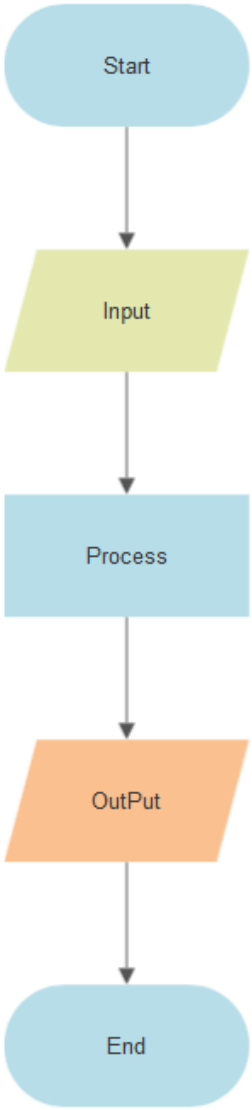
Flowchart

- Flowchart is a type of **diagram** that represents an algorithm , showing the steps (flow of data) in boxes of various kinds.
- It is very useful when the problem is complex.

Symbols used in making flowcharts

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

General form of a flowchart



Example1 – Flowchart?

Algorithm Sum_of_two_numbers

Variables x,y,sum: integers

Start

Read (x,y)

sum←x+y

Write sum

Stop

Example1 - Flowchart

Algorithm Sum_of_two_numbers

Variables x,y,sum: integers

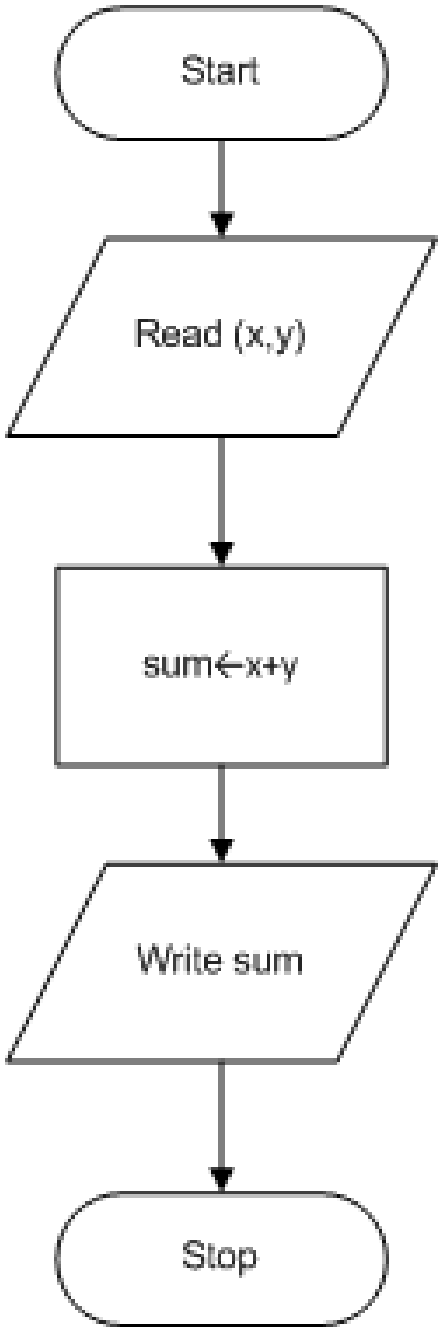
Start

Read (x,y)

sum←x+y

Write sum

Stop



Example2 – Flowchart1?

Algorithm Average_of_four_numbers
Variables x1,x2,x3,x4: integers
avg: real
Start
Read (x1,x2,x3,x4)
avg←(x1+x2+x3+x4)/4
Write avg
Stop

Example2 – Flowchart1

Algorithm Average_of_four_numbers

Variables x1,x2,x3,x4: integers

avg: real

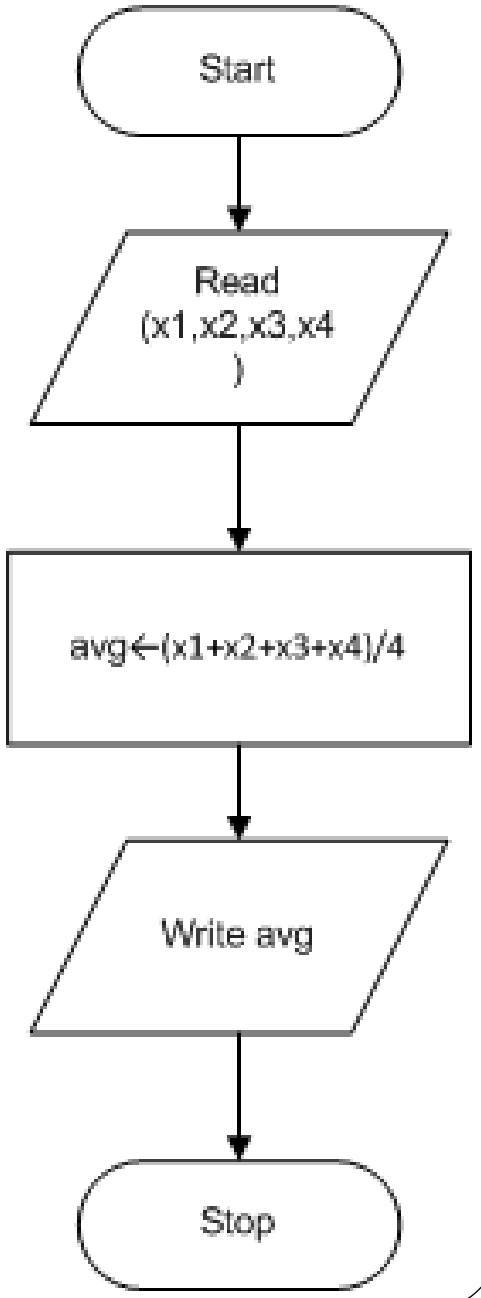
Start

Read (x1,x2,x3,x4)

avg←(x1+x2+x3+x4)/4

Write avg

Stop



Example2 – Flowchart2?

Algorithm Average_of_four_numbers

Variables x1,x2,x3,x4,sum: integers

avg: real

Start

Read (x1,x2,x3,x4)

sum←x1+x2+x3+x4

avg←sum/4

Write avg

Stop

Example2 – Flowchart2

Algorithm Average_of_four_numbers

Variables x1,x2,x3,x4,sum: integers

avg: real

Start

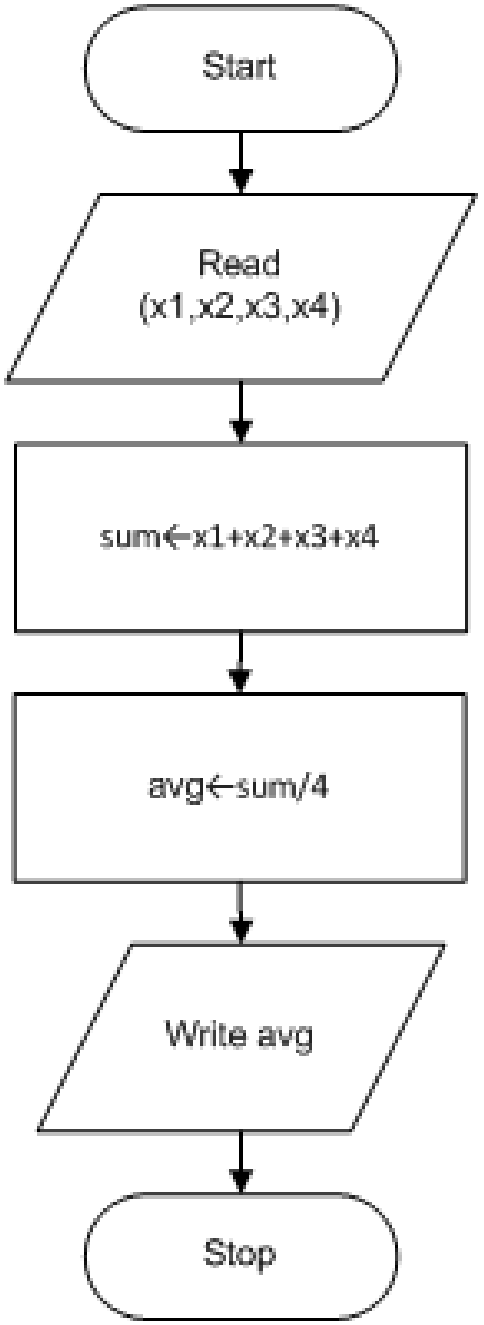
Read (x1,x2,x3,x4)

sum←x1+x2+x3+x4

avg←sum/4

Write avg

Stop



Example3 – Flowchart1?

Algorithm Volume_of_Cone

Variables d,h,v,r: reals

Start

Read (d,h)

r←d/2

v ←(3.14*r*r*h)/3

Write v

Stop

Example3 – Flowchart1

Algorithm Volume_of_Cone

Variables d,h,v,r: reals

Start

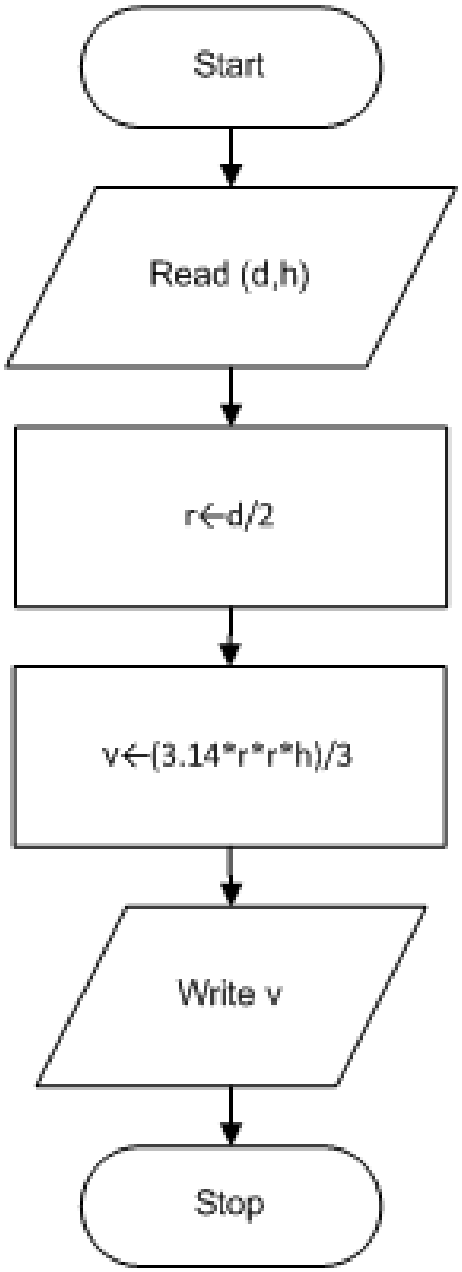
Read (d,h)

r←d/2

v ←(3.14*r*r*h)/3

Write v

Stop



Example3 – Flowchart2?

Algorithm Volume_of_Cone

Variables d,h,v,r,pi,r2: reals

Start

Read (d,h)

pi←3.14

r←d/2

r2←r*r

v ←(pi*r2*h)/3

Write v

Stop

Example3 – Flowchart2

Algorithm Volume_of_Cone

Variables d,h,v,r,pi,r2: reals

Start

Read (d,h)

pi←3.14

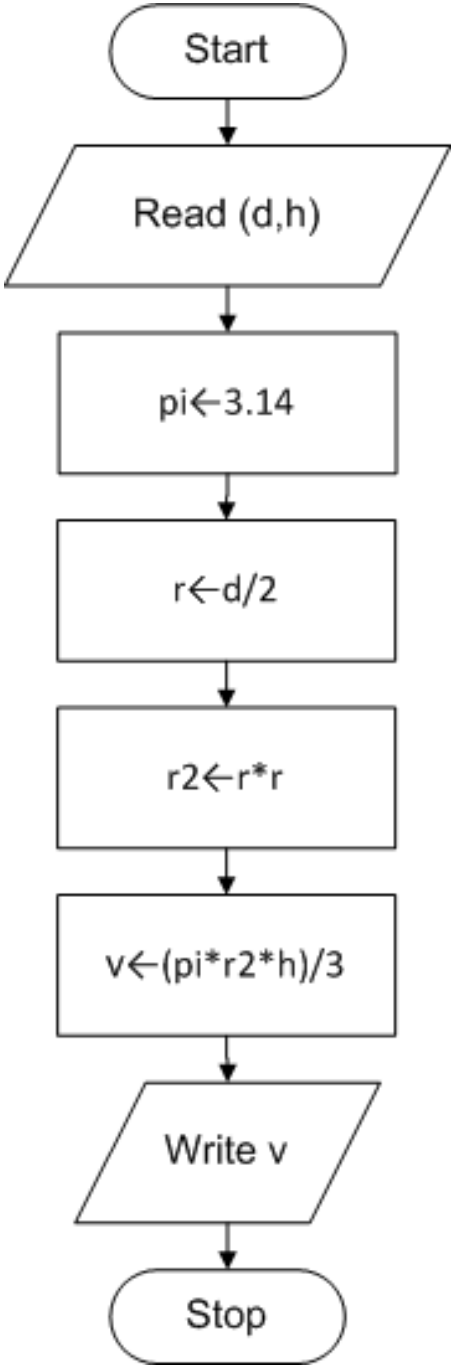
r←d/2

r2←r*r

v ←(pi*r2*h)/3

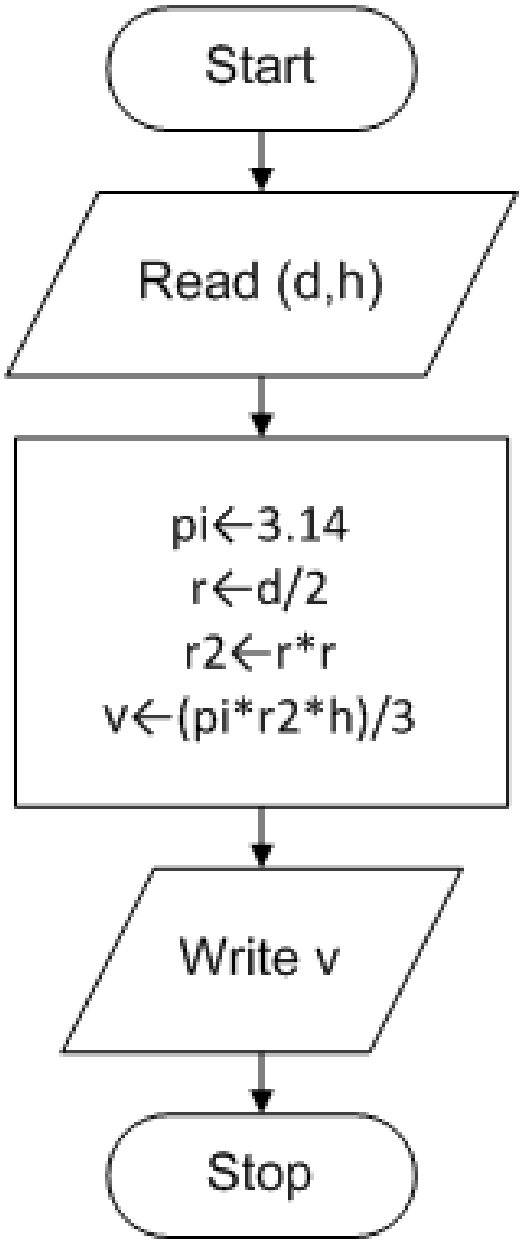
Write v

Stop



Remark

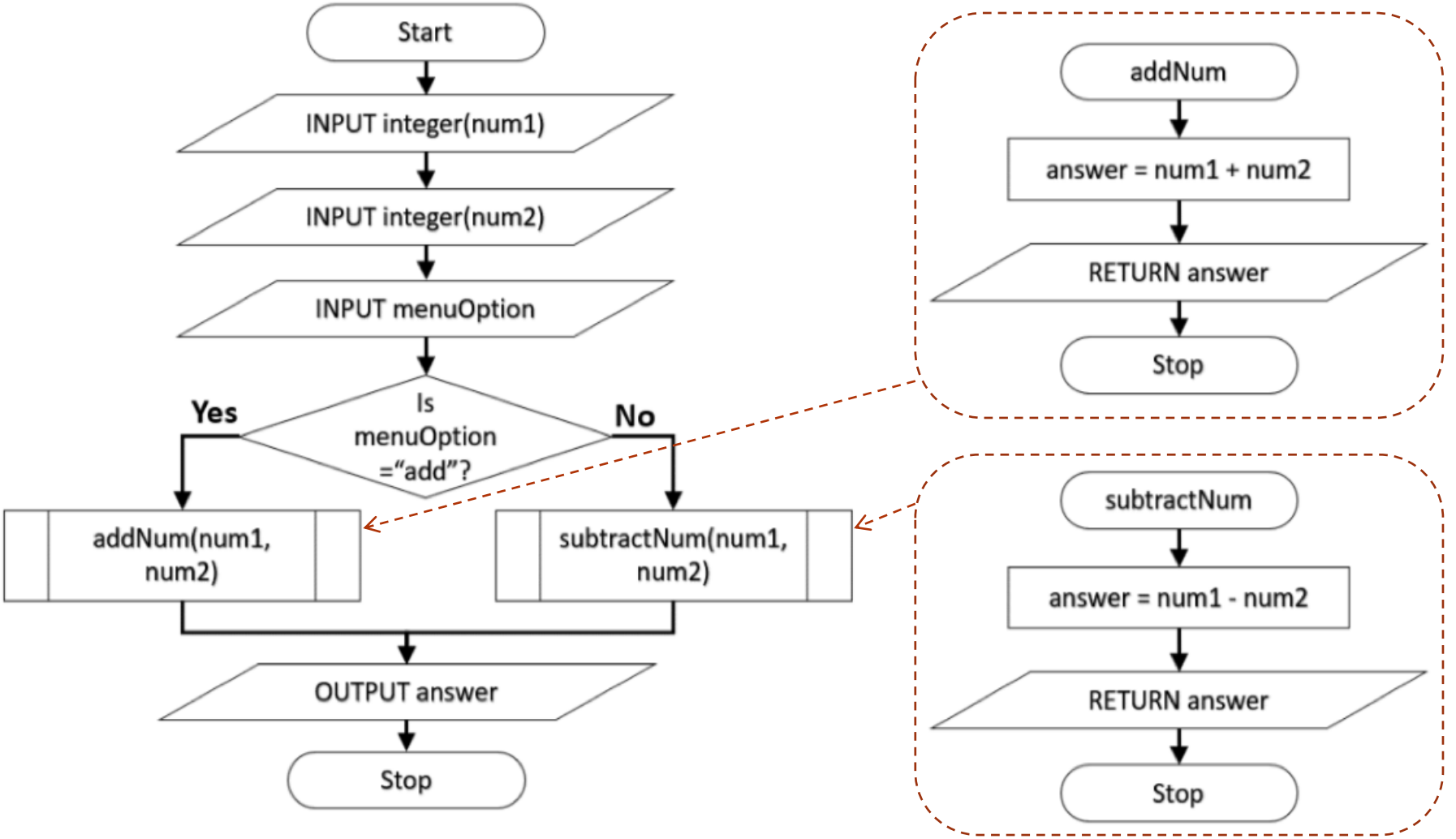
- **Successive** instructions of the **same type** can be grouped in one symbol box.



Additional Useful Shape - Subroutine

- **Subroutine Symbol:** Indicates a sequence of actions that perform a specific task embedded within a larger process. This sequence of actions could be described in more detail on a separate flowchart.

Subroutine Symbol - Example



2. Decision (Selection)

- It is used for making a choice or a decision.
- Two types of decisions:
  1. Single **IF** statement:
  2. **IF-ELSE** statement

Single **IF** statement

- Syntax:

If (condition) **Then**

action 1

action 2

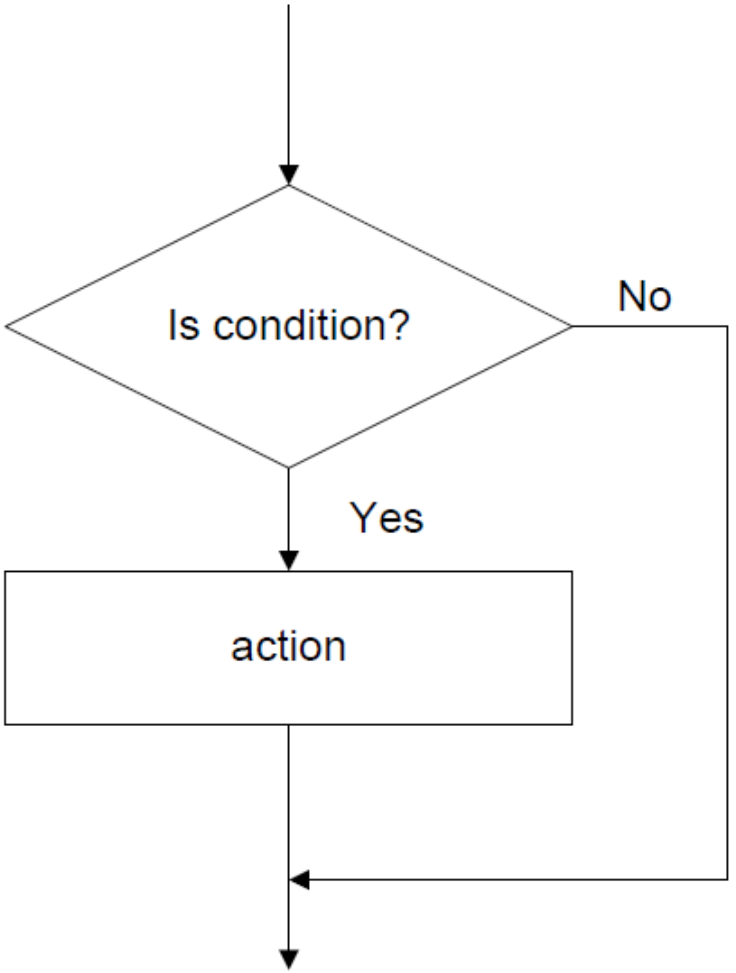
.

.

.

action n

End If



IF-ELSE Statement

- Syntax:

If (condition) **Then**

action1

action 2

.

.

.

action n

Else

action 1

action 2

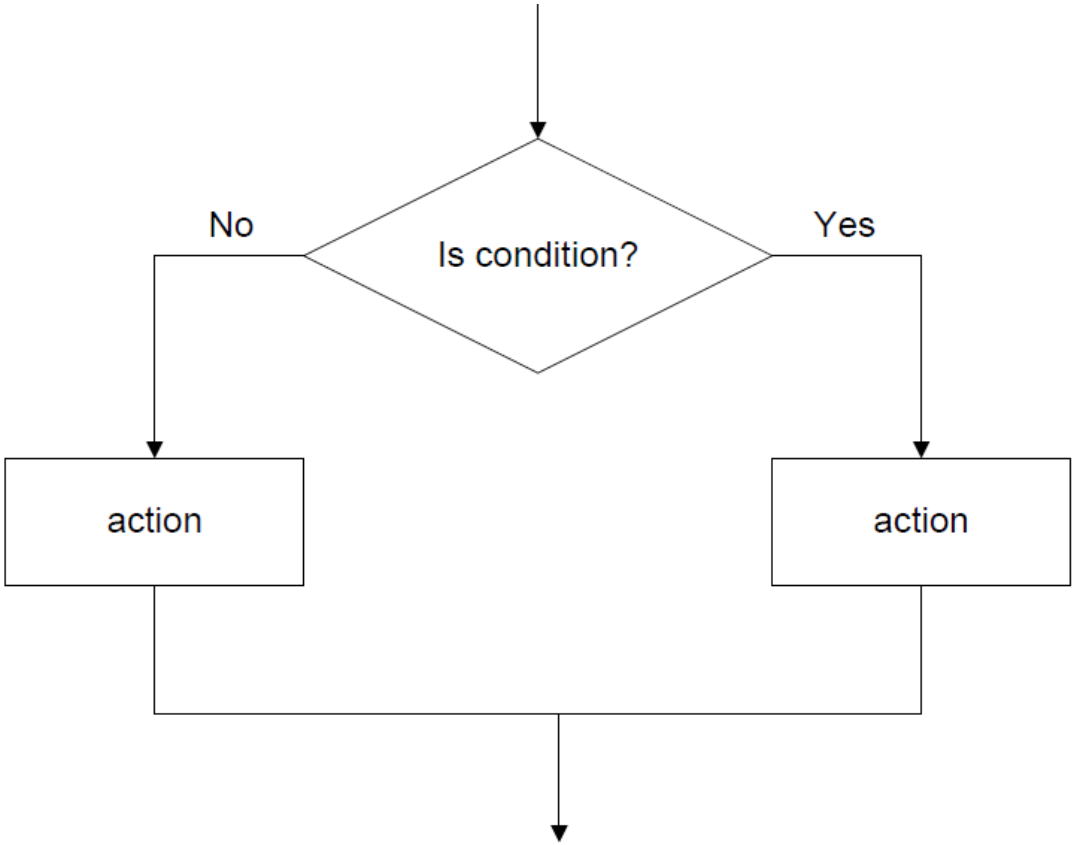
.

.

.

action n

End If

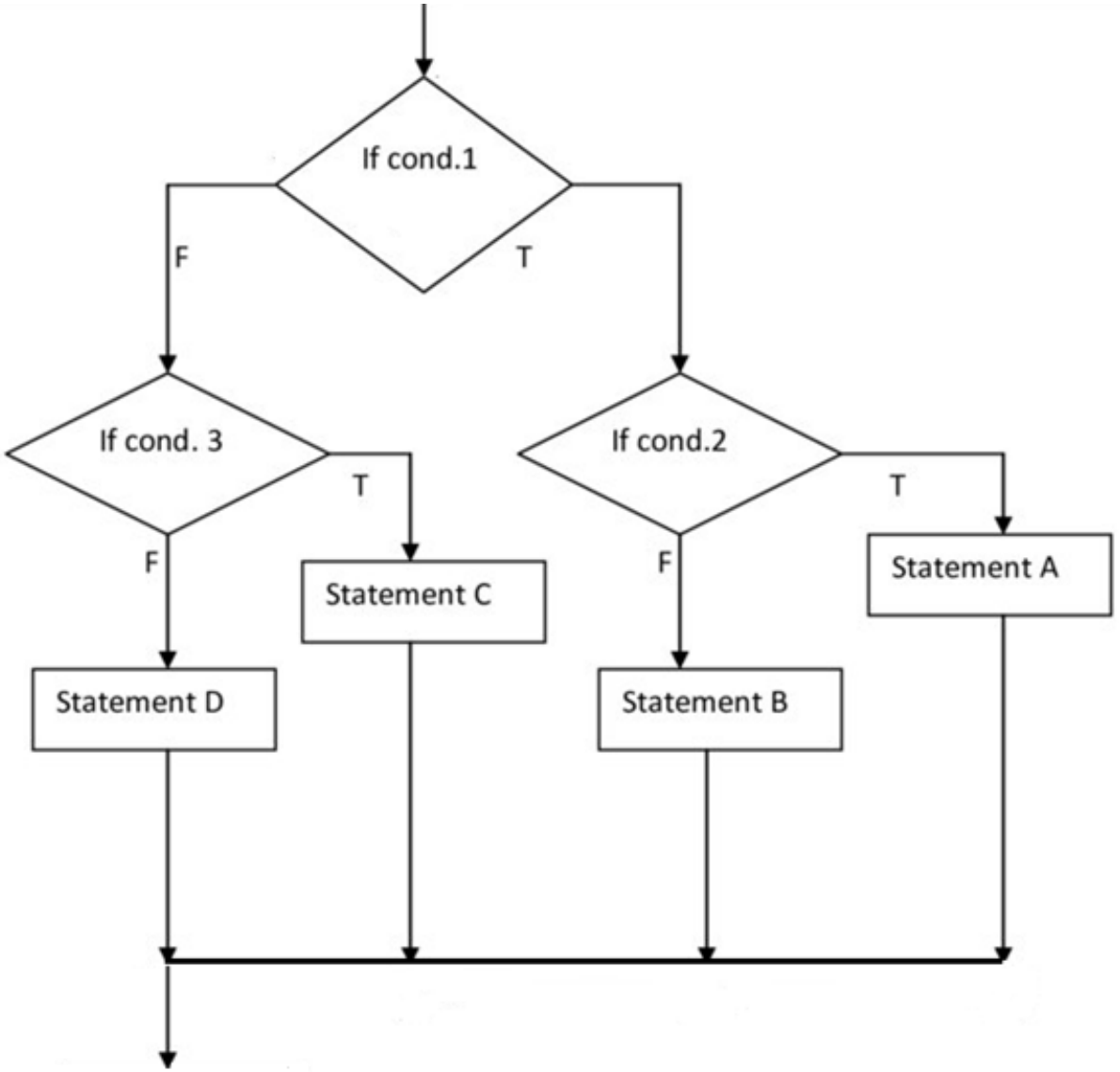


Nested IF-ELSE Statement

- A **nested If** statement is an if statement placed inside another if statement.
- Nested if statements are often used when you must test a combination of conditions before deciding on the proper action.

Nested IF-ELSE Statement

If (cond. 1) **Then**
    If (cond. 2) **Then**
        Statement A
    Else
        Statement B
    End If
Else
    If cond. 3 **Then**
        Statement C
    Else
        Statement D
    End If
End If



Example1

- Write an algorithm and draw its corresponding flowchart that reads two integer numbers and prints out “OK” if they are equal.

Example1 - Solution

```
Algorithm Check_Equality
Variables A,B: integers
Start
   Read (A, B)
   If (A=B) Then
      Write ("OK")
   End If
Stop
```

Example1 - Solution

Algorithm Check_Equality

Variables A,B: integers

Start

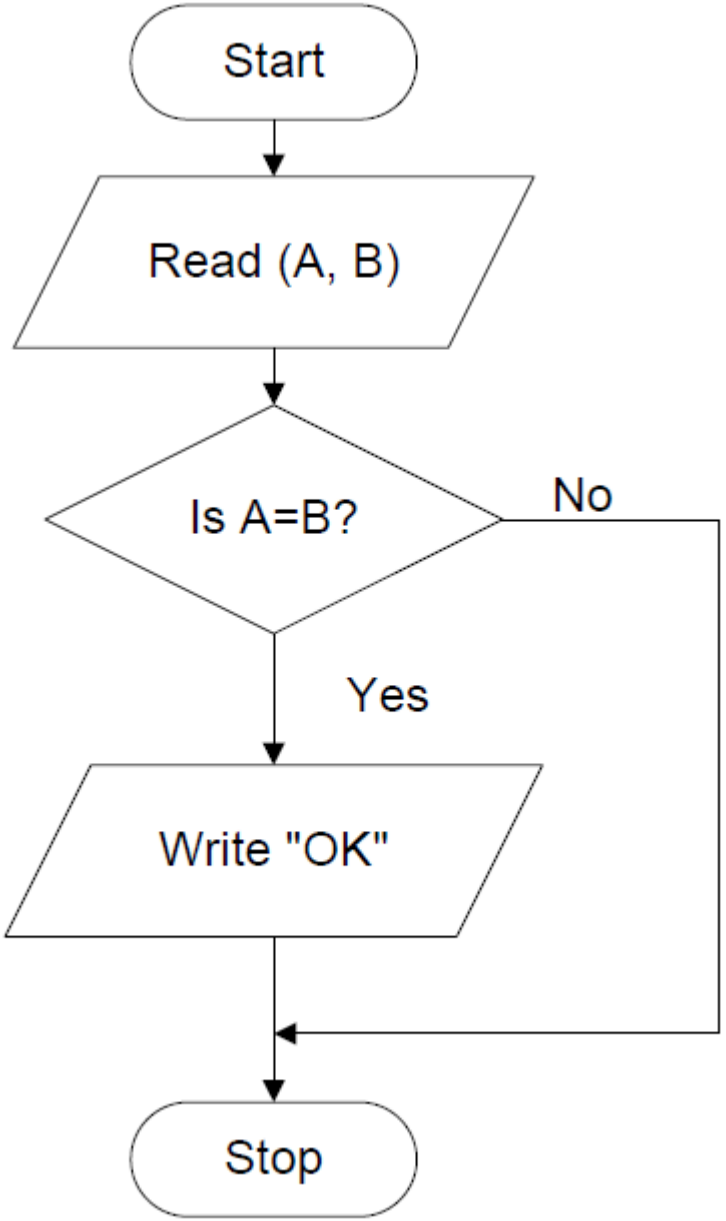
Read (A, B)

If (A=B) **Then**

Write (“OK”)

End If

Stop



Example2

- Write an algorithm that prints out the biggest of two inputted integers. Draw its corresponding flowchart.

Example2 - Solution

```
Algorithm biggest_of_two_nbrs
Variables A,B: integers
Start
    Read (A, B)
    If (A>B) Then
       Write A
    Else
       Write B
    End If
Stop
```

Example2 - Solution

Algorithm biggest_of_two_nbrs

Variables A,B: integers

Start

Read (A, B)

If (A>B) **Then**

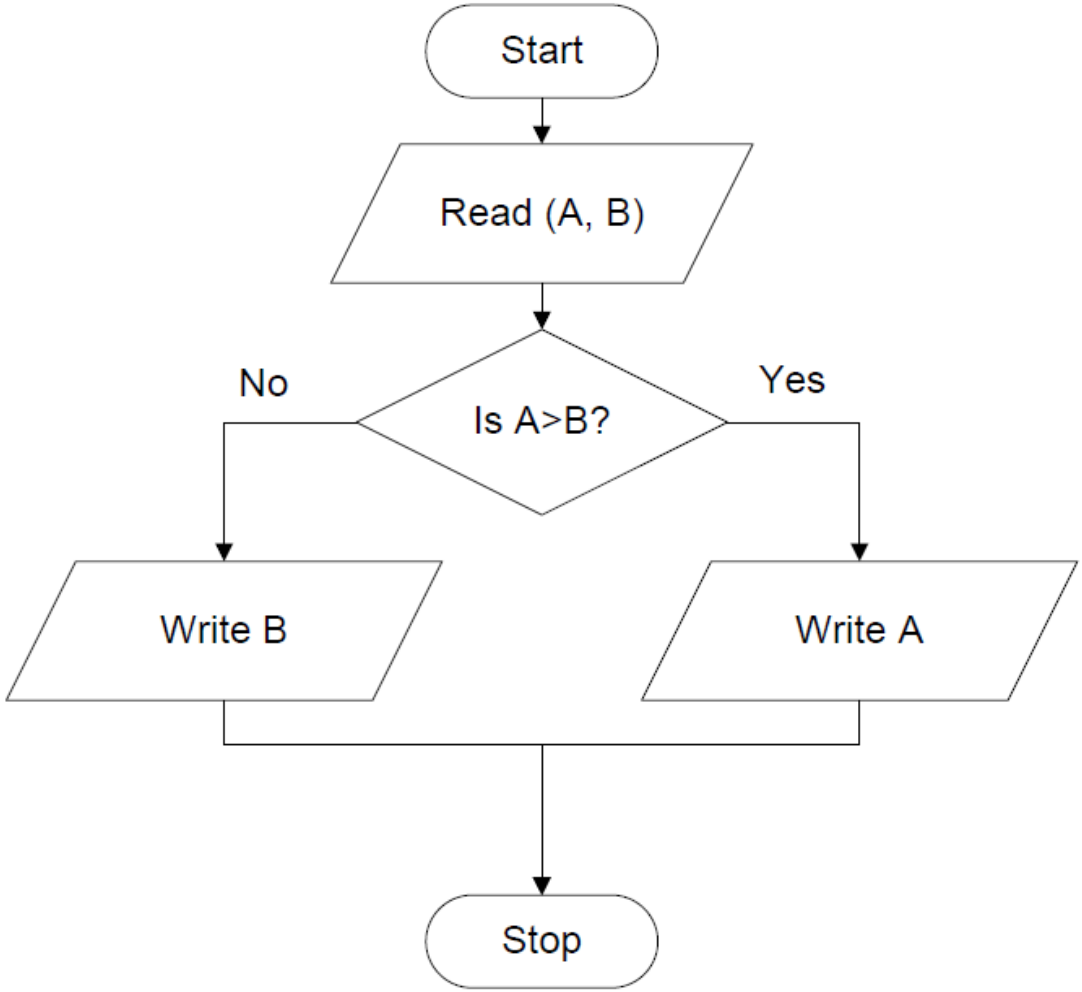
Write A

Else

Write B

End If

Stop



Example2 - Solution

Algorithm biggest_of_two_nbrs

Variables A,B: integers

Start

  Read (A, B)

  If (A>B) **Then**

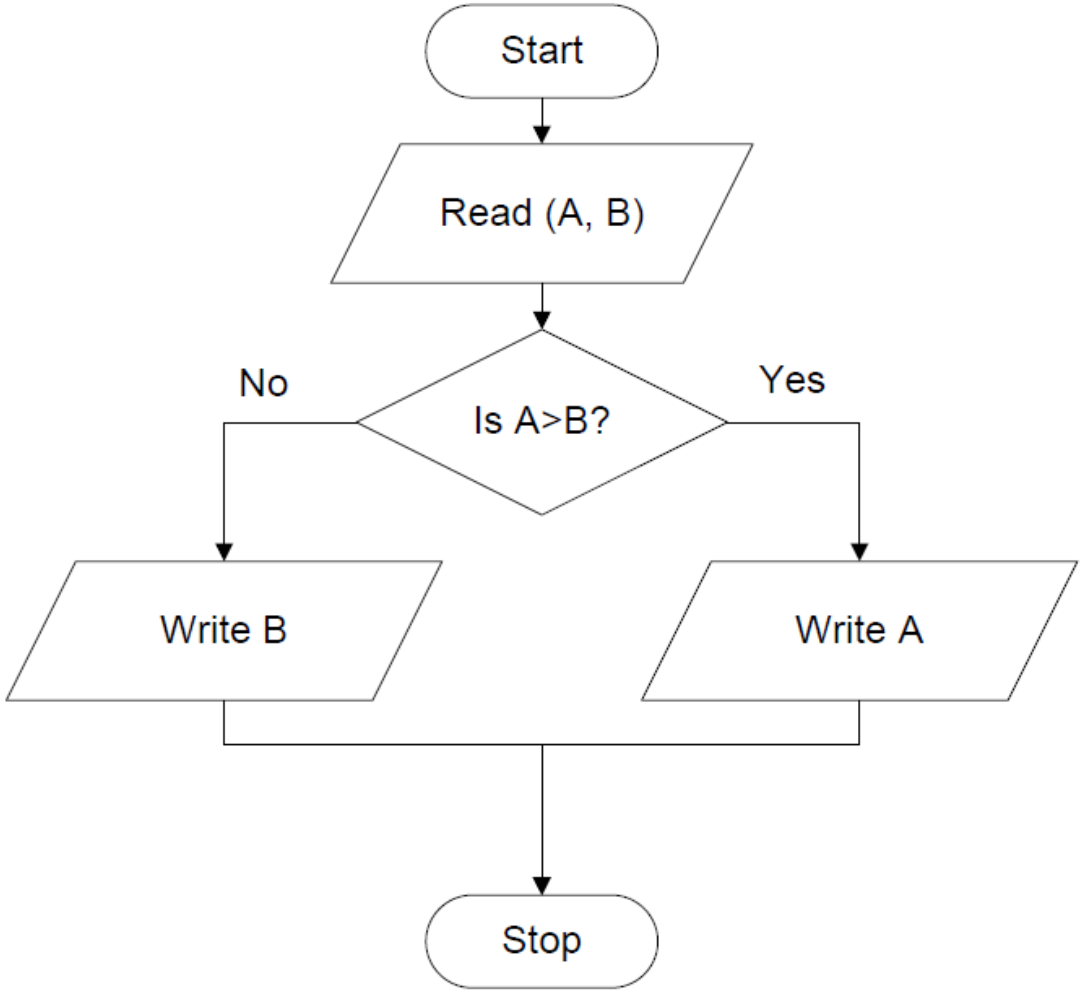
    Write A

  Else

    Write B

  End If

Stop



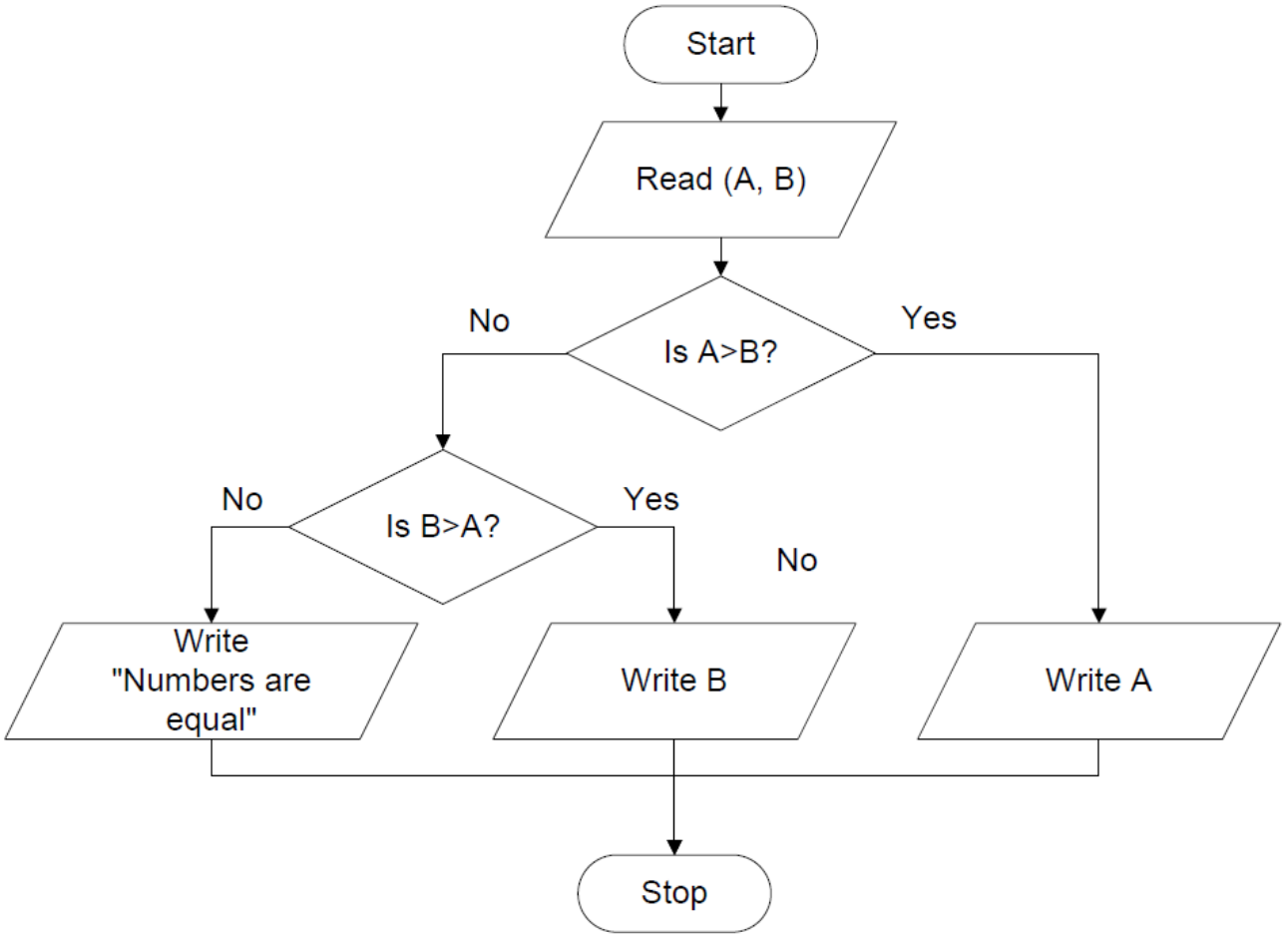
What if A=B?

Example2- Solution2

```
Algorithm biggest_of_two_nbrs
Variables A,B: integers
Start
    Read (A, B)
    If (A>B) Then
       Write A
    Else
       If (B>A) Then
            Write B
       Else
            Write "Numbers are equal"
       End If
    End If
Stop
```

Example2- Solution2

```
Algorithm biggest_of_two_nbrs
Variables A,B: integers
Start
    Read (A, B)
    If (A>B) Then
       Write A
    Else
       If (B>A) Then
          Write B
       Else
          Write "Numl
       End If
    End If
Stop
```



Types of Test Expressions

Two main types of test expressions can be used inside an IF-clause:

- **Simple**:

  Examples:

  if(*x=0*), if(*x>=10*), if(*x%2=0*), if (*x=y*), if(*x!=y*)

- **Compound:** (with logical operators)

  Examples:

  if(*x>=0 AND x<=10*), if(**age<13 OR age>65**),

  if(***NOT(a=0) AND (b=0)***), if(*a=b AND a=c AND b=c*),

  if(***(a=b AND a=c)OR(a=b and b=c) OR (a=c and b=c)***)

Truth table for AND, OR, and NOT

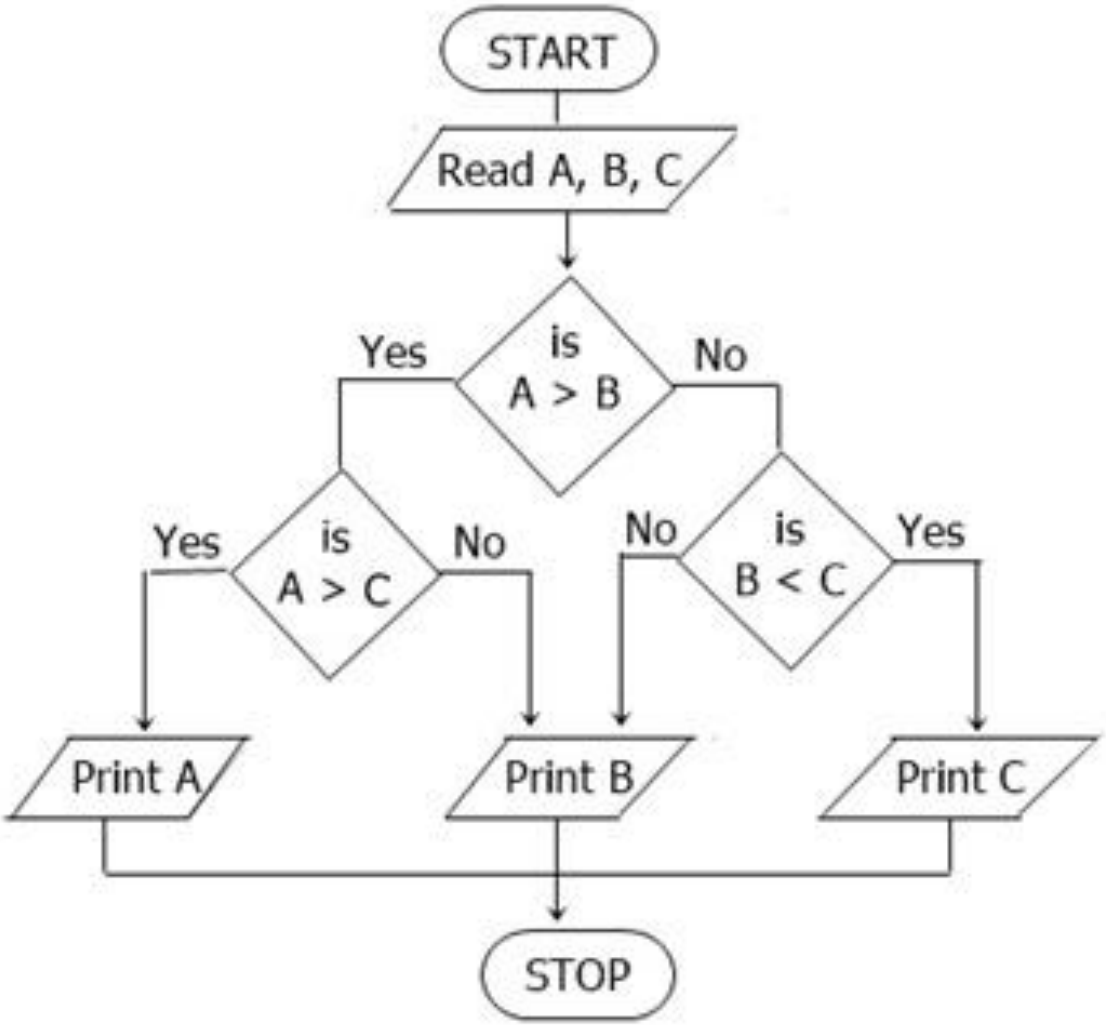
A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Example

- Write an algorithm and flowchart to print the biggest of three input integers using:
    1. Simple test expressions only
    2. Compound test expressions.

Solution1

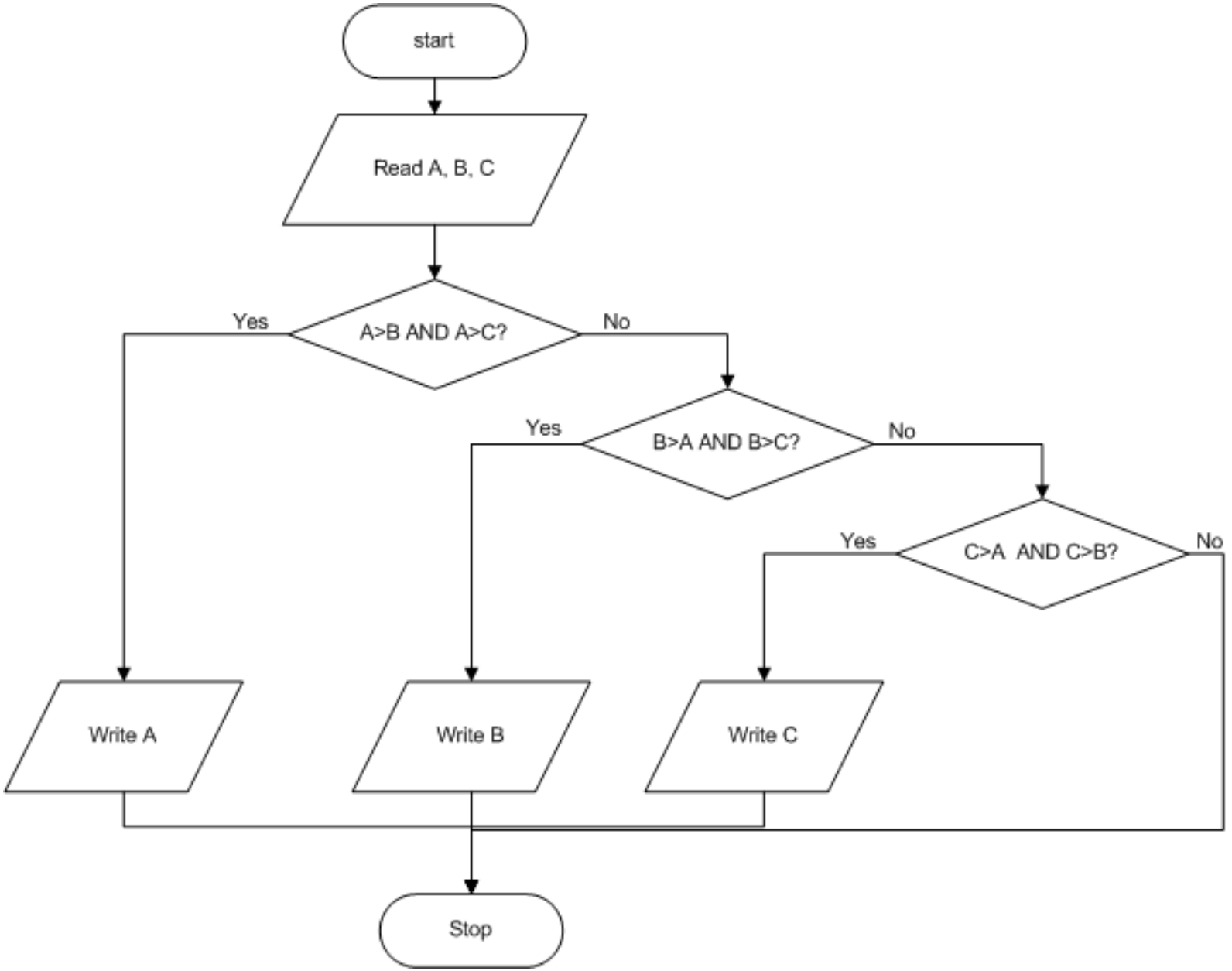
```
Algorithm biggest_of_three_nbrs
Variables A,B: integers
Start
    Read (A, B,C)
    If (A>B) Then
        If (A>C) Then
            Write A
        Else
            Write C
        End If
    Else
        If (B>C) Then
            Write B
        Else
            Write C
        End If
    End If
Stop
```



Solution2

```
Algorithm biggest_of_three_nbrs
Variables A,B: integers
Start
    Read (A, B,C)
    If (A>B AND A>C) Then
        Write A
    Else
        If (B>A AND B>C) Then
            Write B
        Else
            If (C>A AND C>B) Then
                    Write C
            End If
        End If
    End If
Stop
```

Solution2


start
Read A, B, C
A>B AND A>C?
Yes
No
B>A AND B>C?
Yes
No
C>A AND C>B?
Yes
No
Write A
Write B
Write C
Stop

Repetition (Loops)

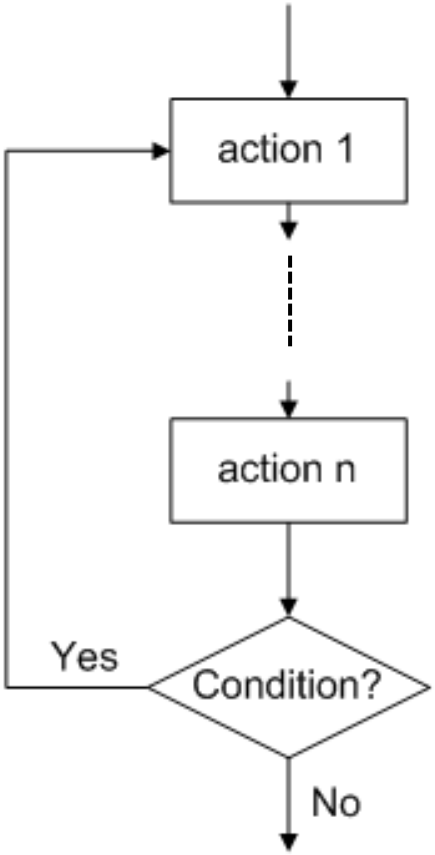
- Loops are used when we need to repeat an instruction or a set of instructions a number of time to find the solution.
- Loops can be implemented using different constructs:
  - IF-THEN-GOTO
  - FOR
  - WHILE
  - REPEAT-UNTIL

IF-THEN-GOTO

Syntax:

- **If** (condition) **Then Goto** ***label***

  Label: a given instruction in the algorithm.



Example1

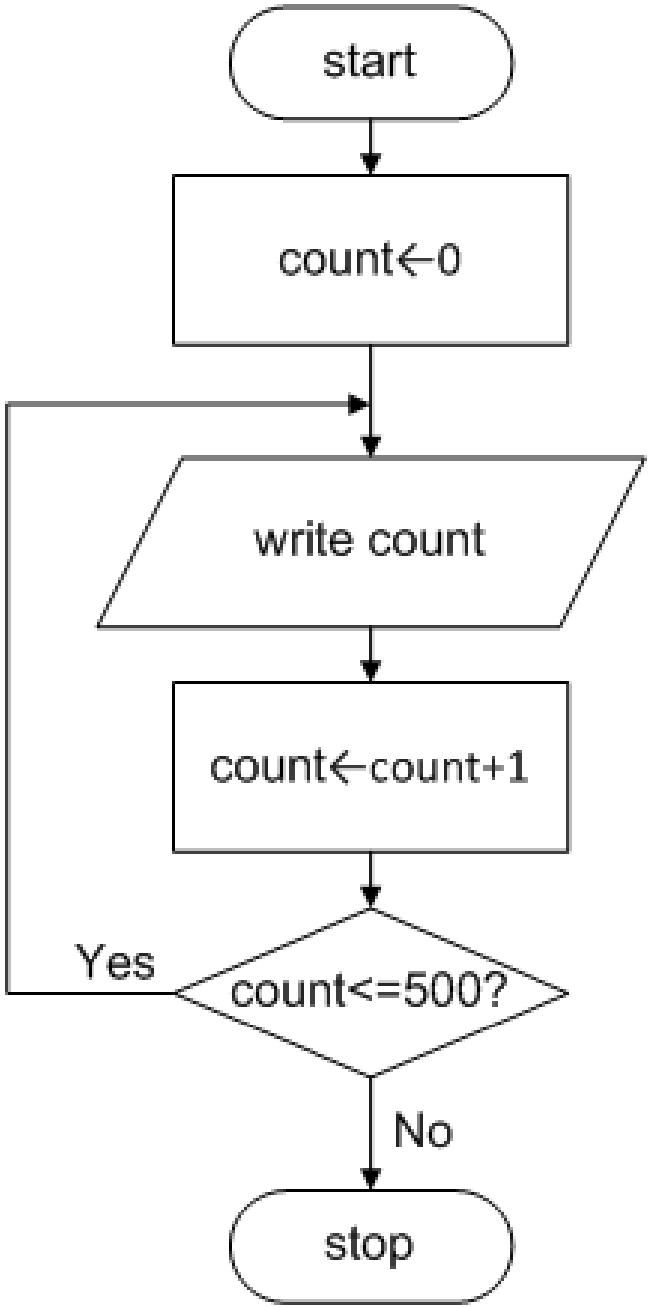
- Write an algorithm and draw flowchart for printing all integer numbers from zero to five-hundreds [0,500].

Example1 - Solution

Algorithm printing_numbers

Variables count: integer

1. **Start**
2. count←0
3. **Write** count
4. count ←count+1
5. **If** (count<=500) **Then**
6. **Goto 3**
7. **End if**
8. **Stop**



Example2

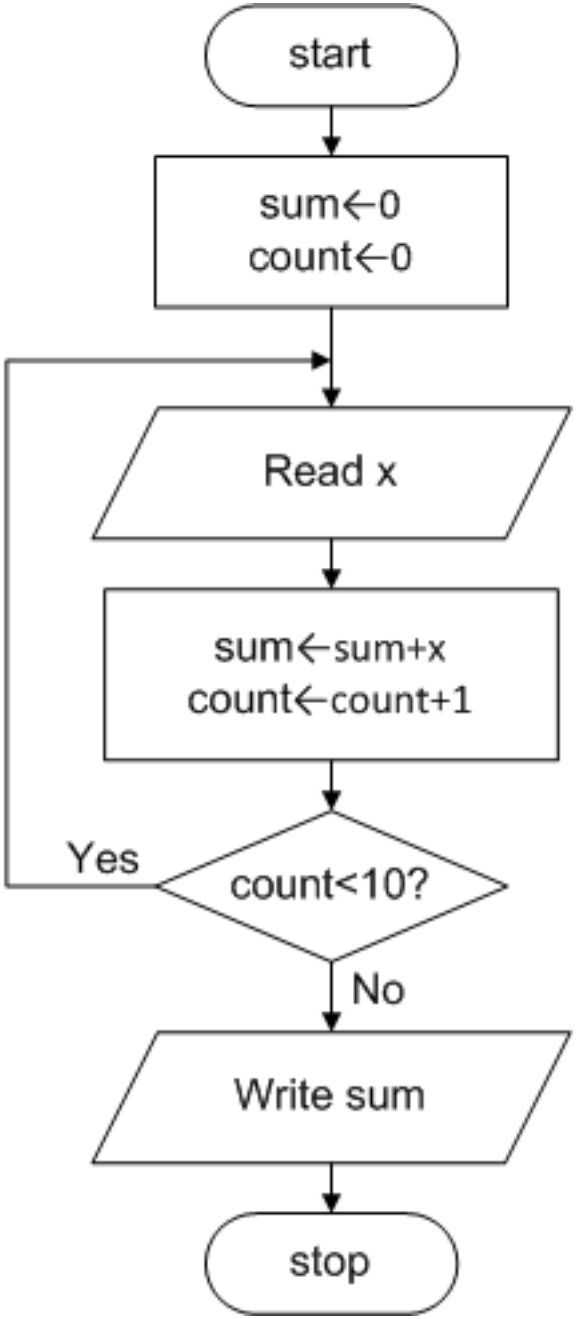
- Using **IF-THEN-GOTO** statement, Write an algorithm and draw flowchart for calculating and displaying the sum of ten random inputted integer numbers given by the user.

Example2 - Solution

Algorithm sum_of_ten_numbers

Variables x,count,sum

1. **Start**
2. sum ← 0
3. count ← 0
4. **Read** x
5. sum ← sum+x
6. count ← count+1
7. **If** (count<10) **Then**
8. **Goto 4**
9. **Else**
10. **Write** sum
11. **End if**
12. **Stop**



FOR Loop

Syntax:

For count_value=*initial_value* **To** *final_value*

action1

action 2

⋮

action n

End For

FOR Loop

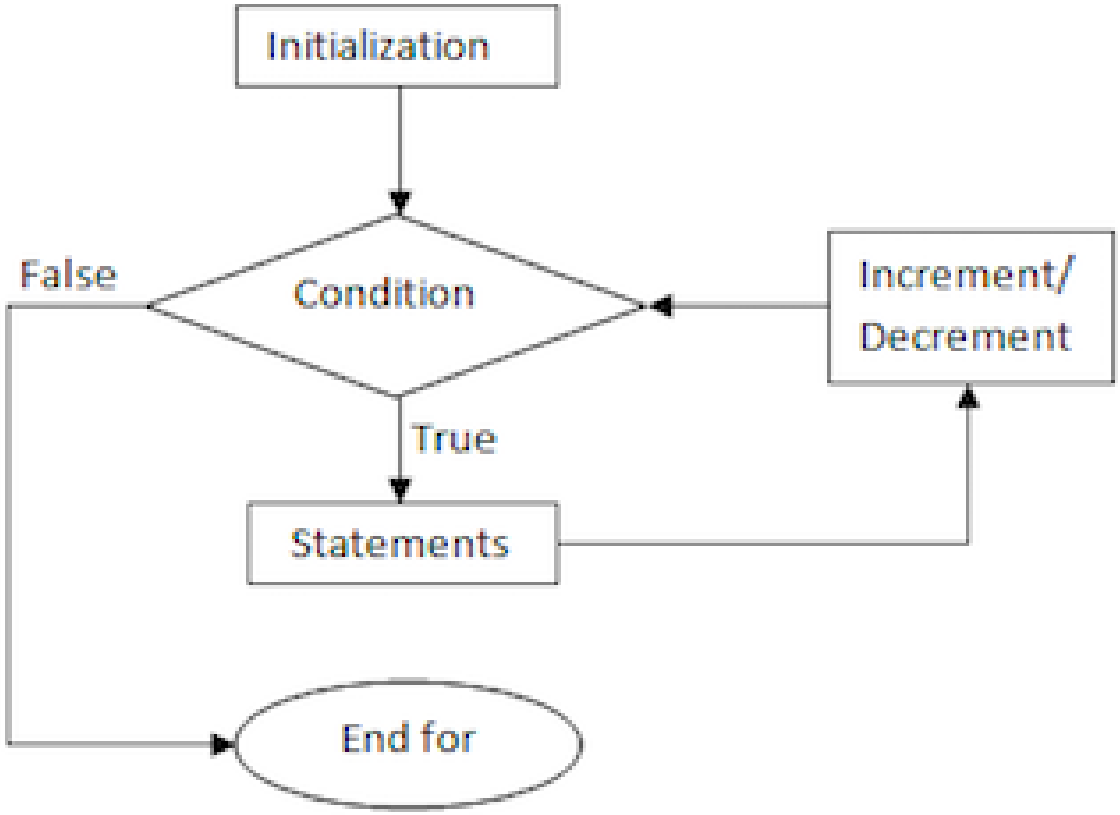


fig: Flowchart for for loop

Example1

- Repeat example1 using **FOR** loop instead of **IF-THEN-GOTO**.

Example1 - Solution

Algorithm printing_numbers

Variables count: integer

Start

  For count=0 **To** 500

    Write count

  End for

Stop

Example2

- Again, repeat example2 using **FOR** loop instead of **IF-THEN-GOTO**.

Example2 - Solution

Algorithm sum_of_ten_numbers

Variables x,count,sum

Start

sum ← 0

For count=0 **To** 9 (we can also write **For** count=1 **To** 10)

Read x

sum ← sum+x

End For

Write sum

Stop

Example3

- Write an algorithm and draw flowchart to print the average grade of four taught courses for all students in your class.

Example3 - Solution

Algorithm average_grade_of_students

Variables N, i: integers

g1, g2, g3, g4, Avg: Reals

Start

Read N

For i=1 **To** N

Read (g1, g2, g3,g4)

Avg ←(g1+g2+g3+g4)/4

Write (i, Avg)

End For

Stop

FOR Loop - Remark

- The **FOR** loop is often requested when the number iterations is known. If not, other loop types are involved.

WHILE Loop

Syntax:

While (***condition***)

action 1

action 2

⋮

action n

Body of while loop

End While

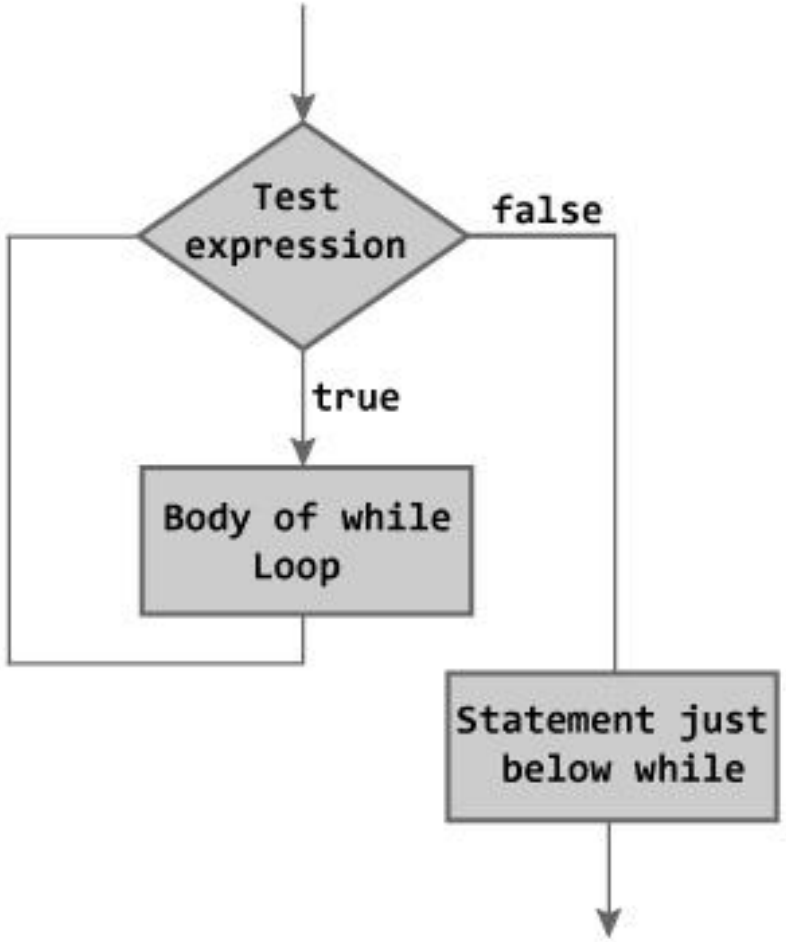


Figure: Flowchart of while Loop

Example

- Write an algorithm and draw flowchart to calculate the sum of a set of positive integer numbers given by the user. The algorithm stops when the user enters "-1" and the sum of all previously entered numbers will be displayed.

Solution

Algorithm sum_set_of_numbers

Variables x,sum:integers

Start

sum←0

x←0

While (x!=-1)

Read x

sum ←sum+x

End while

Write sum

Stop

WHILE Loop - Remark

- The **While** loop is involved when the number of iterations is unknown. However it can be applied when that number is known.

REPEAT-UNTIL Loop

Syntax:

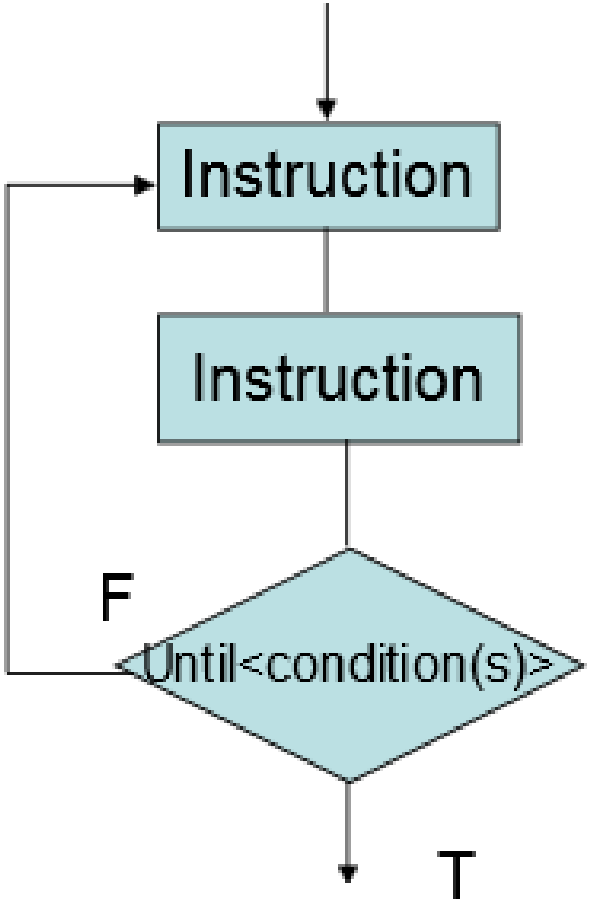
Repeat

action1

action2

action n

Until (condition)



Example1

- Repeat the example of the biggest of two integer numbers with a condition that the user should input two different ones. If not, the user is asked again repeatedly to enter other ones until they become different and then printing their biggest.

solution

```
Algorithm biggest_two_different_numbers
Variables A,B:integers
Start
   Repeat
      Read (A,B)
   Until (A!=B)
   If (A>B) Then
      Write A
   Else
      Write B
   End If
Stop
```

Example2

- Repeat the previous example of the while loop (sum of a set of unknown number of integers) using Repeat-Until.

Solution

Algorithm sum_set_of_numbers

Variables x,sum:integers

Start

sum←0

x←0

Repeat

sum ←sum+x

Read x

Until (x=-1)

Write sum

Stop

Example3

- Write an algorithm to find the average age of all students in your class. Use Repeat-Until statement with zero age denotes end-of-list for students.

Solution

Algorithm Average_age
Variables count,sum,avg:integers
Start
    sum←0
    count←0
    Read age
    Repeat
        sum ←sum+age
        count ←count+1
        Read age
    Until (age=0)
    avg ← sum/count
    Write avg
Stop

WHILE Vs REPEAT-UNTIL

- **While** loop first evaluates the condition and if it holds, one loop is done then repeat, otherwise the looping ends.
- **Repeat-Until** does one loop and then evaluates the condition. If it doesn't hold, repeat. If it holds, the looping ends.
- This means, for example, that you'll always do at least one loop with **Repeat-Until**, unlike with **While** which may not do any loops if the condition doesn't hold initially.

Infinite Loop

- An infinite loop is a loop that repeats indefinitely (**endless loop**).
- This happens when the terminating condition cannot occur.
- Most of the time we create infinite loops by mistake.
- **Example:**

```
Algorithm infinite_loop
Variables i:integer
Start
    i←1
    While (i<10)
        Write i
    End While
Stop
```

Which repetition construct to use in solving a given problem?

- Any problem containing repetition can be solved using either If-Then-Goto, For, While, or Repeat-Until loop. However, it is advisable to use the most appropriate one for that given problem.
- **Example:**

  Write an algorithm for calculating the factorial of an integer number using the four loop types (If-Then-Goto, For, While, and Repeat-Until).

  Which algorithm is better?

Solution with IF-THEN-GOTO

```
Algorithm factorial
Variables n,count,F: integer
1.  Start
2.  Read n
3.  F←1
4.  count←1
5.  F←F*count
6.  count←count+1
7.  If (count<=n) Then
8.      Goto 5
9.  Else
10.     Write F
11. End if
12. Stop
```

Solution with FOR

```
Algorithm factorial
Variables   n,count,F: integer
Start
   Read n
   F←1
   For  count=1   To  n
        F←F*count
   End for
   Write F
Stop
```

Solution with WHILE

```
Algorithm factorial
Variables   n,count,F: integer
Start
   Read n
   F←1
   count←1
   While (count<=n)
         F←F*count
         count ←count+1
   End While
   Write F
Stop
```

Solution with REPEAT-UNTIL

```
Algorithm factorial
Variables   n,count,F: integer
Start
   Read n
   F←1
   count←1
   Repeat
         F←F*count
         count ←count+1
   Until (count>n)
   Write F
Stop
```

Which solution is better?