01: PROGRAMMING PROBLEM-SOLVING

Programming Technique I

(SCSJ1013)

Problem-Solving Process

The Programming Process

Programming Process

- Programming is a process of problem solving.

- Problem solving techniques
  - Outline the problem requirements
  - Analyze the problem
  - Design steps (algorithm) to solve the problem

Programming Process

This week

1. Clearly define what the program is to do.
2. Visualize the program running on the computer.
3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.
5. Type the code, save it, and compile it.
6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program. Repeat Steps 5 through 8 as many times as necessary.
9. Validate the results of the program.

Input, Process and Output

Input, Process and Output

Three steps that a program typically performs:

- Gather input
  - from keyboard
  - from files on disk drives
- Process the input
- Display the result as output
  - send it to the screen
  - write to a file

Example 1

- Identify an input, process and output to develop a program to calculate area of a rectangle.

- Input data
  - Length
  - Width
- Process the input
  - Area = Length * Width
- Output Data
  - Area

In-Class Exercise 1

Identify the input, process and output for a program to calculate employee income tax based on the following formula:

Tax = 0.25 * (monthly income * 11 – number of kids * 450)

Your program will display the name of the employee and amount of tax on the screen.

- Input data
  - Name
  - Monthly income
  - Number of kids
- Process the input
  - Tax = 0.25 * (Monthly income * 11 – Number of kids * 450)
- Output Data
  - Name
  - Tax

In-Class Exercise 2

Identify the following information:

Input data, Process the input data, and Output data

- Do Exercise 2.7, No. 7(a), pg. 56
- Do Exercise 2.7, No. 7(c), pg. 56
- Do Exercise 2.7, No. 7(d), pg. 57
- Do Exercise 2.7, No. 7(e), pg. 57
- Do Exercise 2.7, No. 8, pg. 57
- Do Exercise 2.7, No. 9, pg. 59

Procedural and Object-Oriented Programming

Procedural and Object-Oriented Programming

- Procedural programming (a.k.a structured programming) is centered on procedures or functions (a.k.a modules). Example language: C.

- Object-oriented programming (OOP), is centered on objects. An object contains data and procedures. Example language: C++.

Problem Solving Techniques

Problem Solving Methods

Three problem solving methods will be discussed in this class are:

- Develop Algorithms
  - Flowchart
  - Pseudo code
- Top-down design
  - Structured Chart

Algorithms

- Algorithm - a sequence of a finite number of steps arranged in a specific logical order to produce the solution for a problem.

- Algorithms requirements:
  - Must have input
  - Must produce output
  - Unambiguous
  - Generality
  - Correctness
  - Finiteness
  - Efficiency

(pg. 32)

Pseudo codes

Pseudo Code

- Pseudocode is a semiformal, English-like language with limited vocabulary that can be used to design & describe algorithms.
- Purpose- to define the procedural logic of an algorithm in a simple, easy-to-understand for its readers.
- Free of syntactical complications of programming language.

Example: Pseudo Code

- Execution sequence follow the steps flow.

```
Example: Algorithm for
multiplying two numbers
1. Start
2. Get A
3. Get B
4. Calculate result,
      C = A * B
5. Display result C
6. End
```

Execution sequence

In-Class Exercise 1

Develop a pseudo code for a program to calculate employee income tax based on the following formula:

Tax = 0.25 * (monthly income * 11 – number of kids * 450)

Your program will display the name of the employee and amount of tax on the screen.

```
1.Start
2.Get name
3.Get monthly_income
4.Get num_kids
5.Calculate tax:
    tax = 0.25 *
    (monthly_income *
    11 - num_kids *
    450)
6.Display name and
  tax
7.End
```

In-Class Exercise 2

Develop a pseudo code:

- Do Exercise 2.7, No. 7(a), pg. 56
- Do Exercise 2.7, No. 7(c), pg. 56
- Do Exercise 2.7, No. 7(d), pg. 57
- Do Exercise 2.7, No. 7(e), pg. 57
- Do Exercise 2.7, No. 8, pg. 57
- Do Exercise 2.7, No. 9, pg. 59

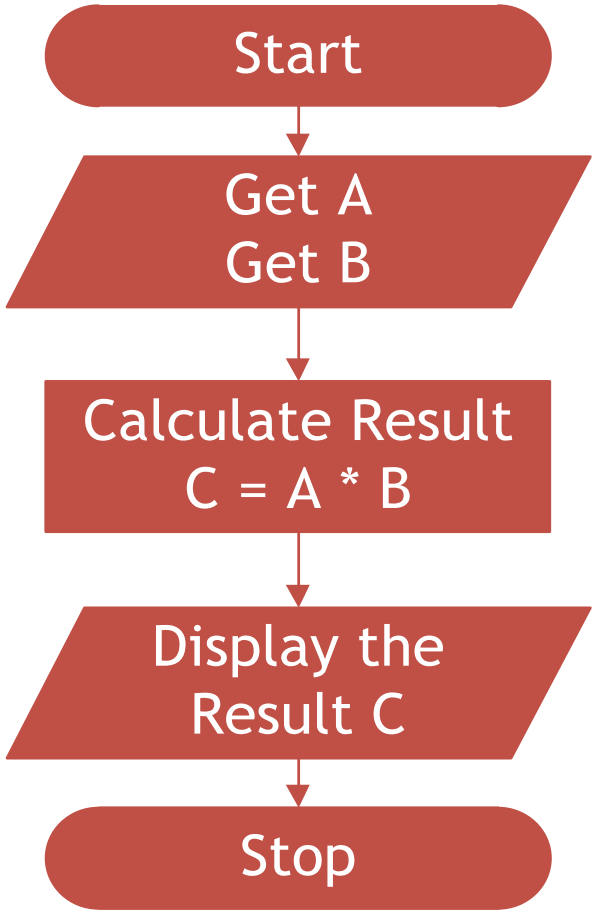
Flowcharts

Flowchart

- Flowchart - represents an algorithm in graphical symbols.
- Flowchart – a graph of geometrical shapes that are connected by lines.
- Two important element in flow chart:
  - Geometrical shapes – represent type of statements in the algorithm
  - Flow line – show the order in which the statements of an algorithm are executed.

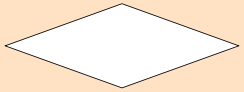
Example: Flowchart

- Algorithm for multiplying two numbers



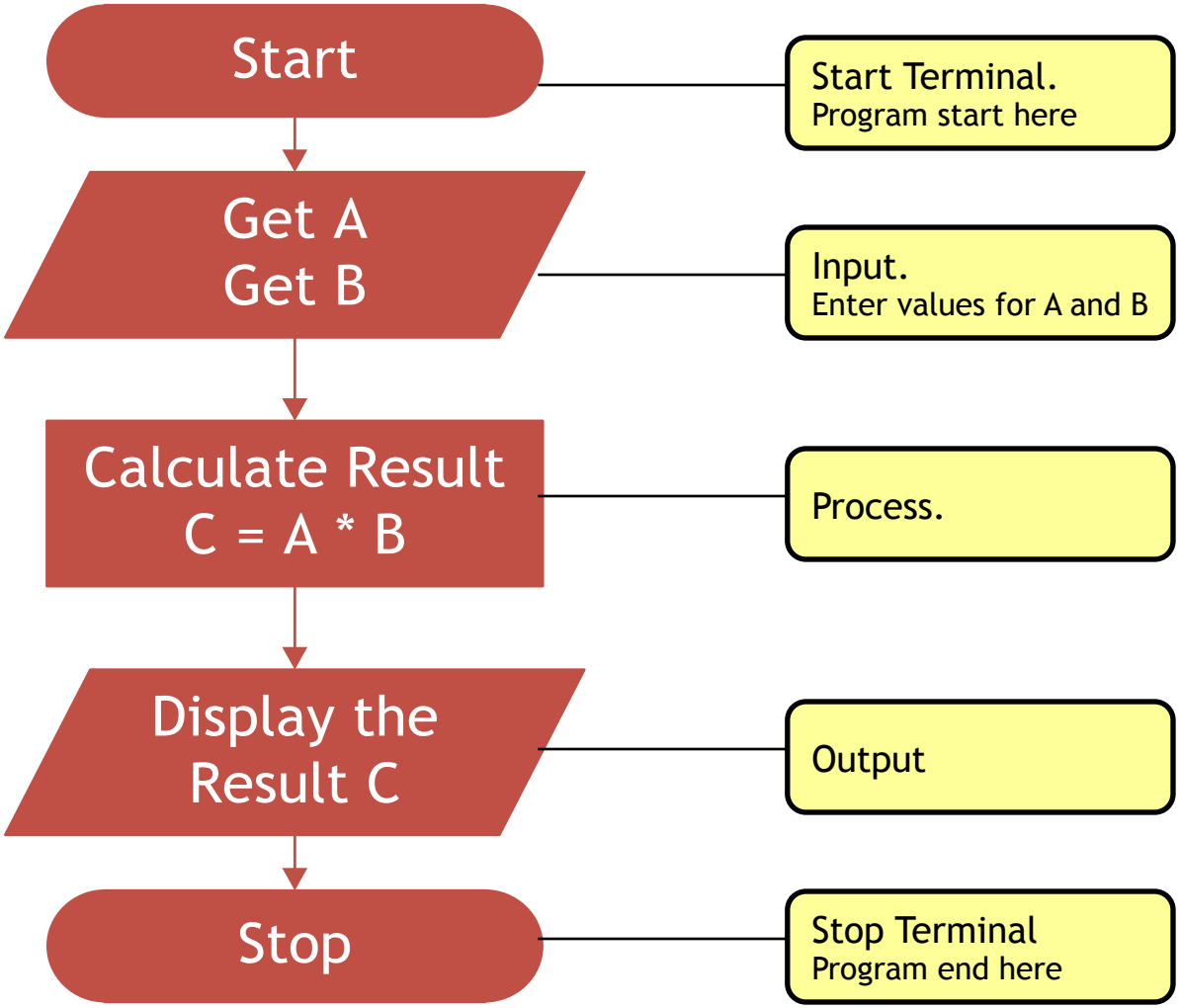
Flowchart Symbol

Flowchart Symbol

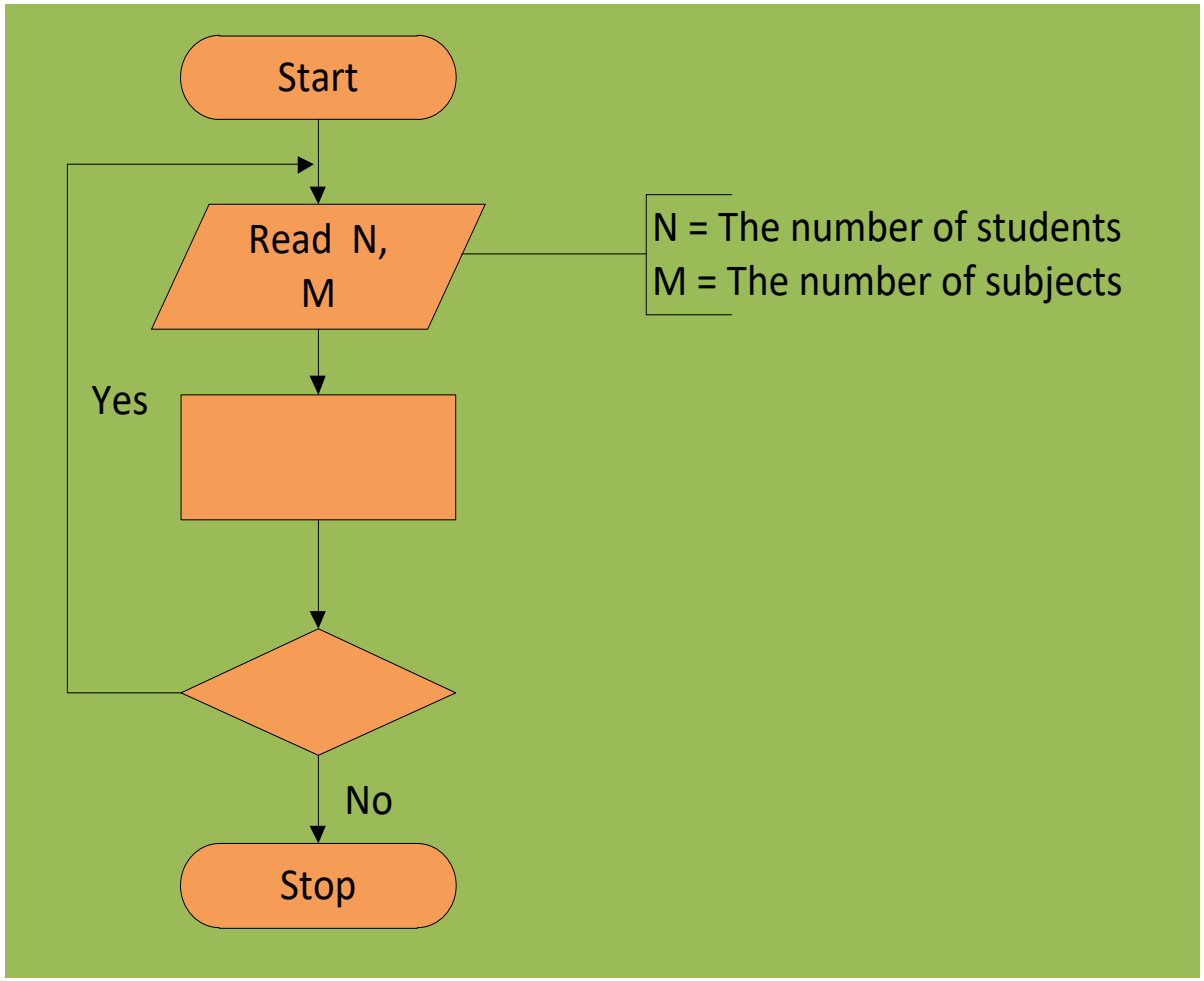
	**Terminal:** Used to indicates the start and end of a flowchart. Single flowline. Only one "Start" and "Stop" terminal for each program. The end terminal for function/subroutine must use "Return" instead of "Stop".
	**Process:** Used whenever data is being manipulated. One flowline enters and one flowline exits.
	**Input/Output:** Used whenever data is entered (input) or displayed (output). One flowline enters and one flowline exits.
	**Decision:** Used to represent operations in which there are two possible selections. One flowline enters and two flowlines (labelled as "Yes" and "No") exit.
	**Function / Subroutine:** Used to identify an operation in a separate flowchart segment (module). One flowline enters and one flowline exits.
	**On-page Connector:** Used to connect remote flowchart portion on the same page. One flowline enters and one flowline exits.
	**Off-page Connector:** Used to connect remote flowchart portion on different pages. One flowline enters and one flowline exits.
	**Comment:** Used to add descriptions or clarification.
	**Flowline:** Used to indicate the direction of flow of control.

(pg. 39)

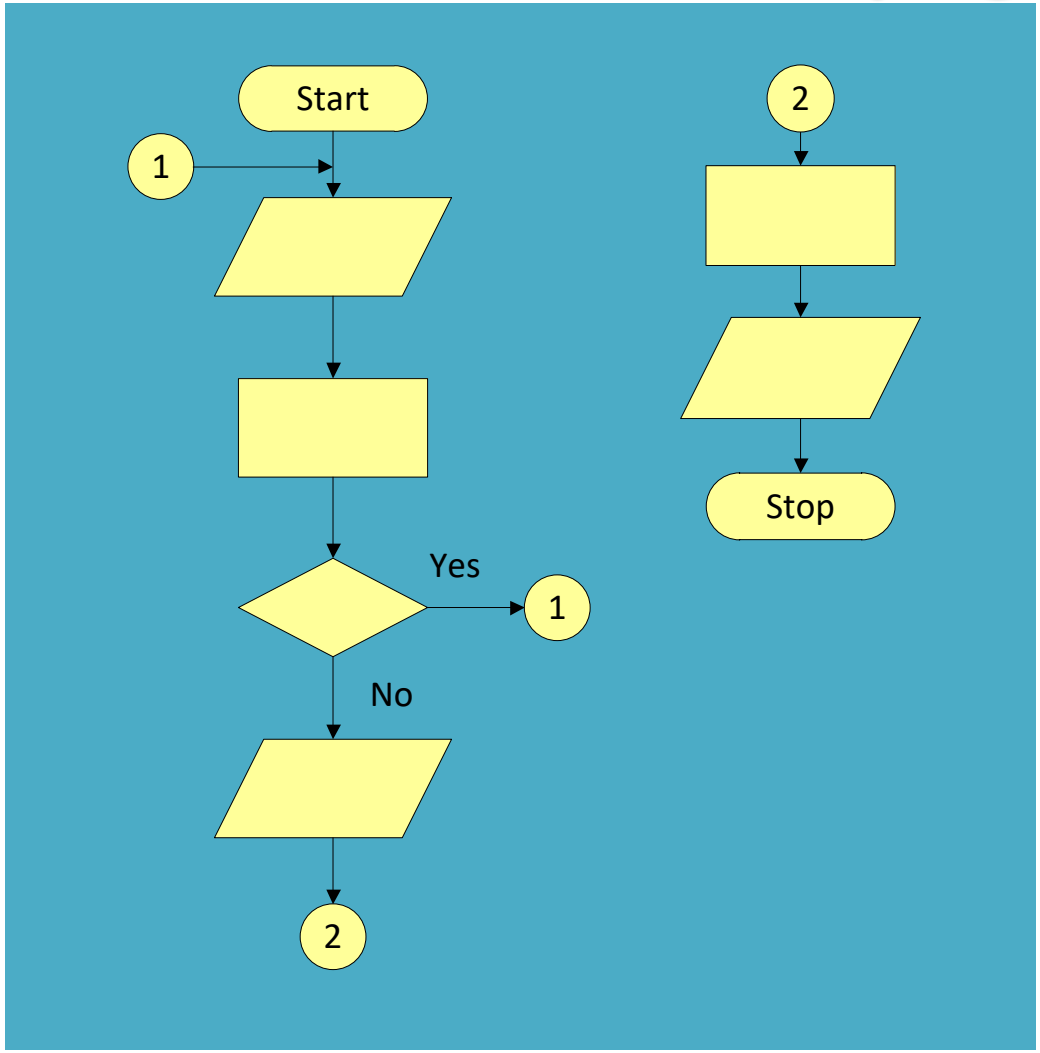
Flowchart Explanation



Example: Use of comments/ description



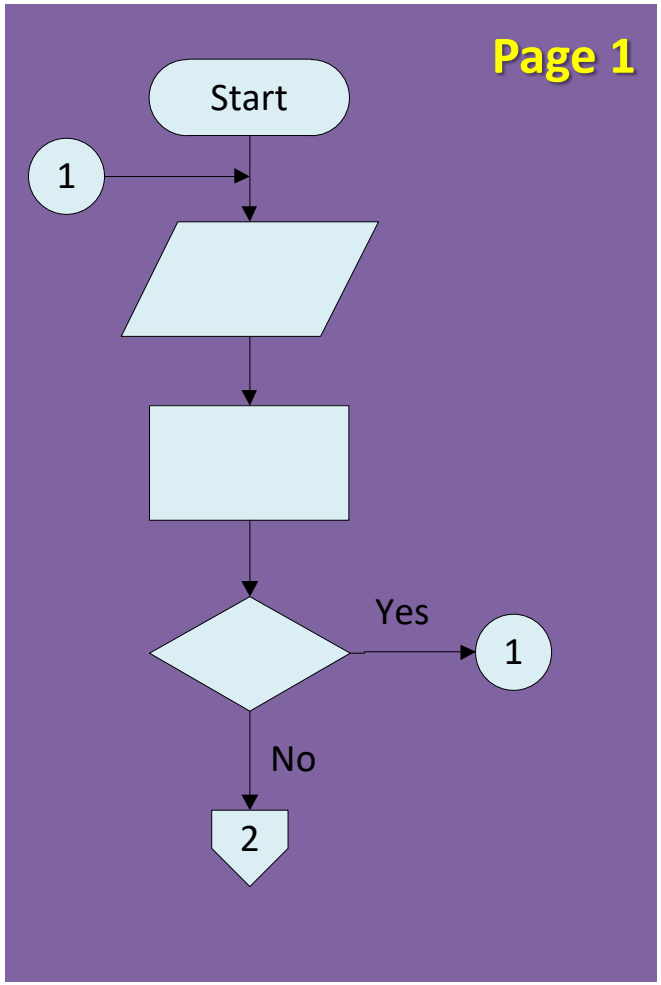
Example: Use of connectors on the same page

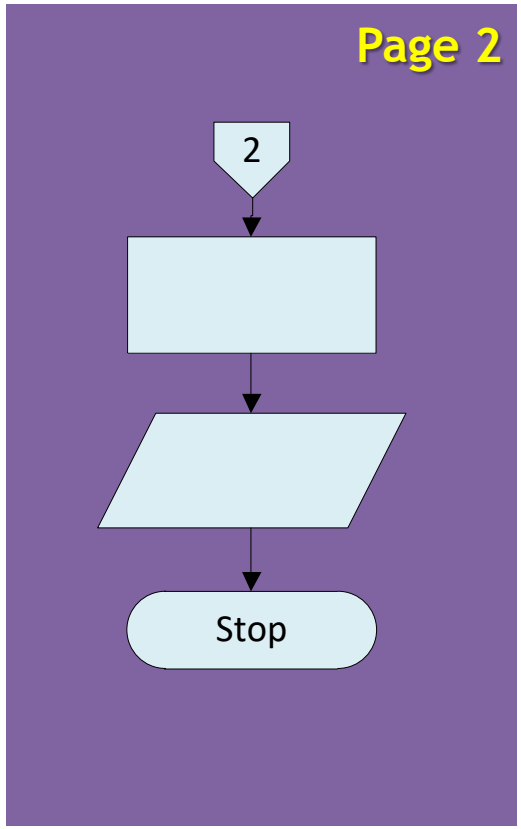


1- connection on the same flowchart portion

2- connection on the different flowchart portion

Example: Use of connectors on the different page



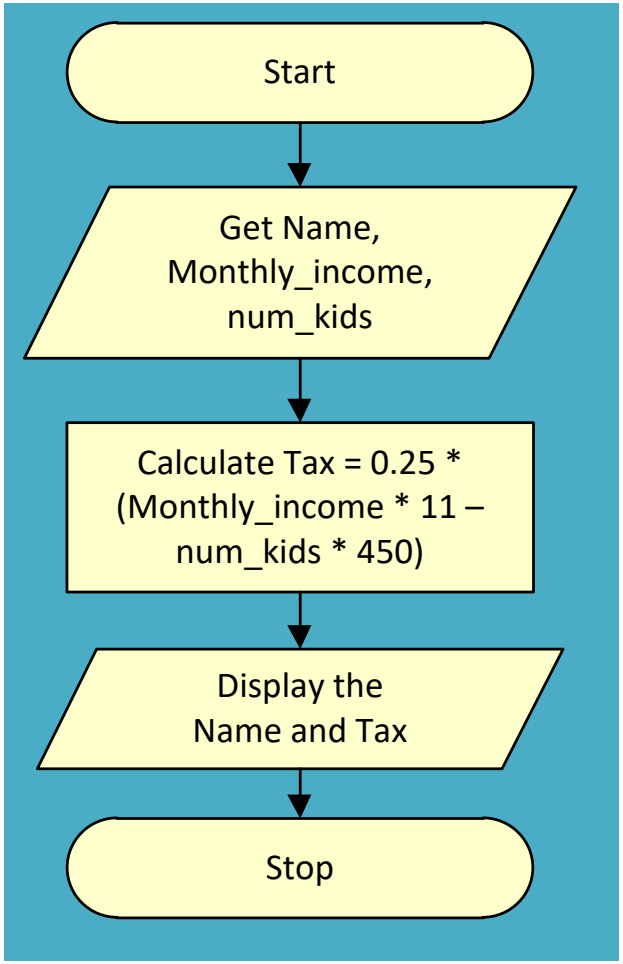


In-Class Exercise

Draw a flowchart for a program to calculate employee income tax based on the following formula:

Tax = 0.25 * (monthly income * 11 – number of kids * 450)

Your program will display the name of the employee and amount of tax on the screen.



Flowchart Structures

Control Structure

- Describe the flow of execution

- Basic types of control structure:
  - Sequential
  - Selection
  - Repetition

Flowchart Structures: Sequential

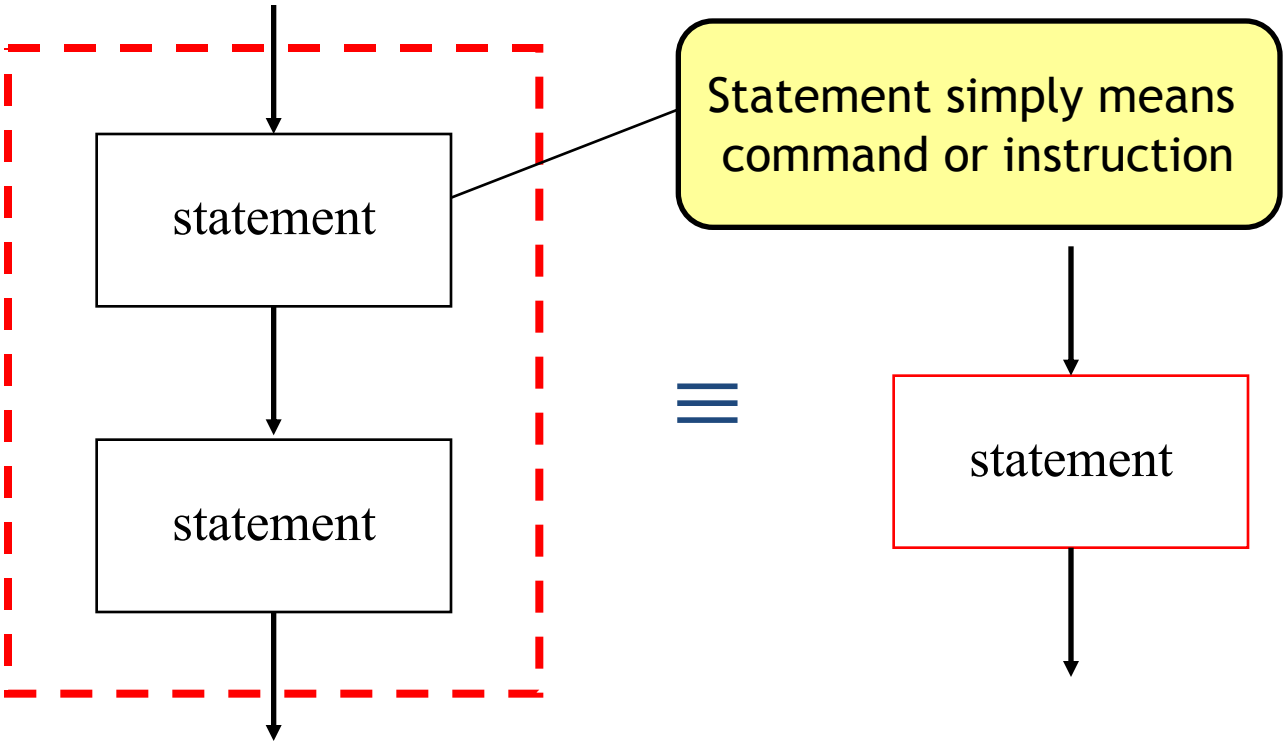
Sequential Structure

- A series of steps or statements that are executed in the order they are written in an algorithm.
- Pseudo code - Mark the beginning and end of a block of statements.

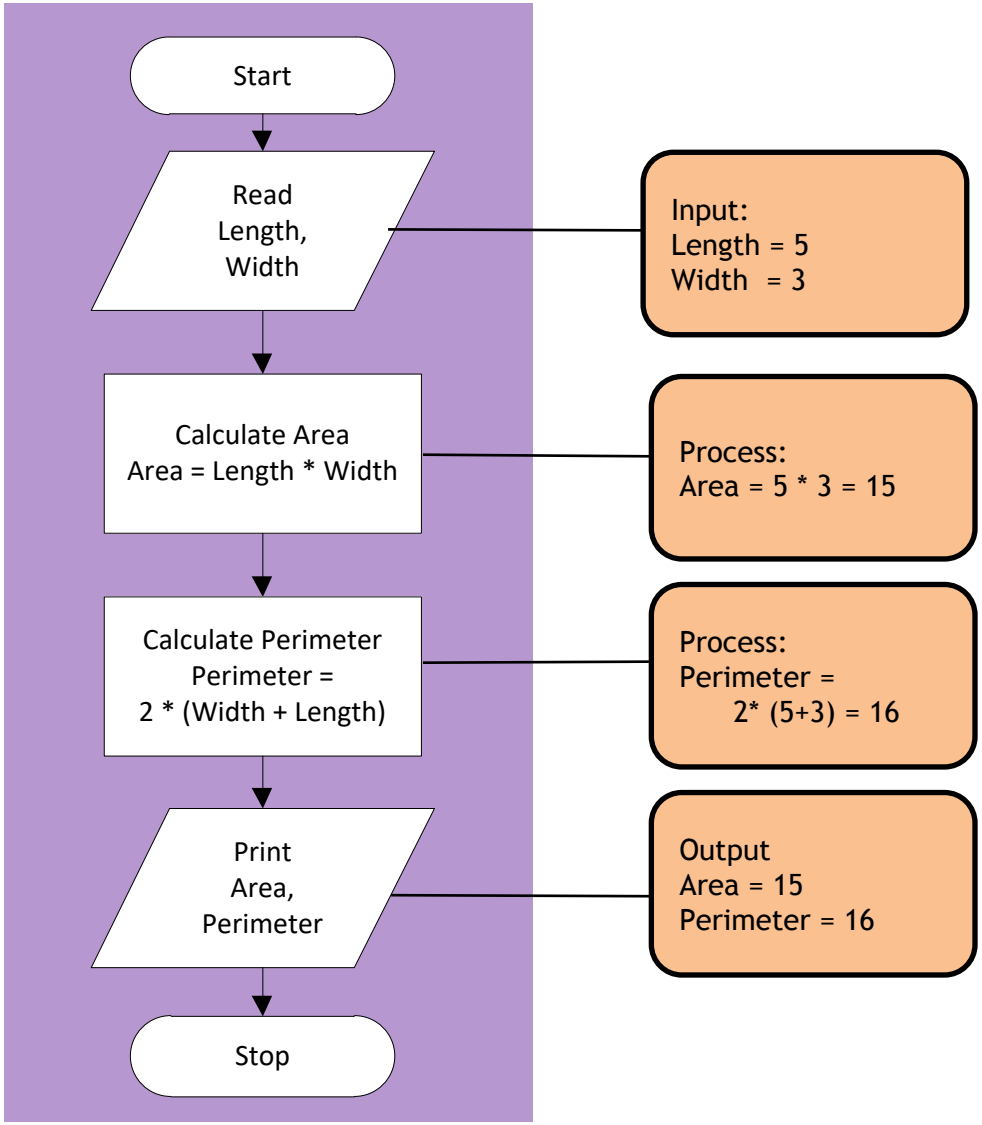
```
1.    Start
2.    Statement_1
3.    Statement_2
4.    Statement_3
:          :
n.    Statement_n
n+1.  End
```

Sequential Structure: Flowchart

- Multiple statements considered as one statement.



Flowchart Tracing



Trace Table

- Trace tables allow developers to test their algorithms in order to make sure there are no logic errors.
- Within the trace table, each variable, conditional test and output must be listed.
- Example:

Length	Width	Area	Perimeter	Output
10	15	150	50	Area = 150, Perimeter = 50
20	20	400	80	Area = 400, Perimeter = 80
30	15	450	90	Area = 450, Perimeter = 90

In-Class Exercise 1

Trace the content of the variables and determine the output of the following algorithm, if the input for Radius is:

a. 3 b. 10 c. 150

```
Algorithm 1: Compute the area of a circle
1. Start
2. Set PI = 3.14159
3. Read the Radius
4. Calculate the area of a circle using the
   formula:
       Area = Radius x Radius x PI
5. Display Area
6. End
```

In-Class Exercise 2

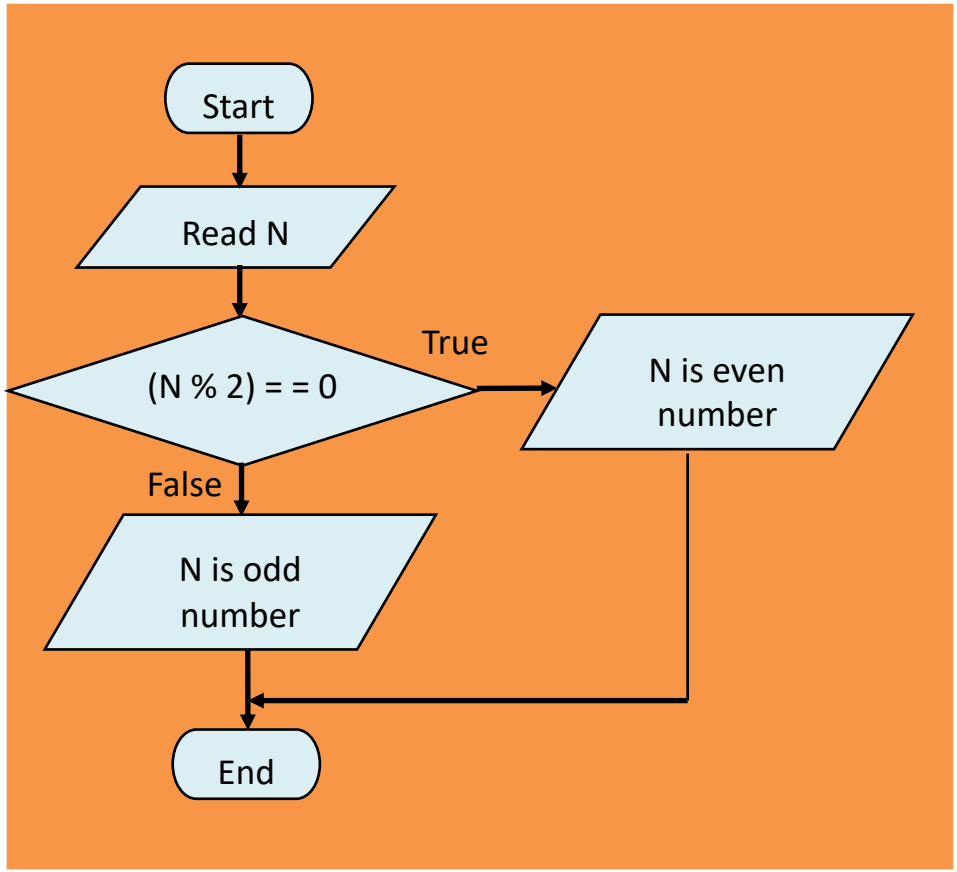
Execute the flowchart using the following input values:

a. 89

b. 26

c. 0

d. 3



Flowchart Structures: Selection

Selection Structure

- Selection allows you to choose between two or more alternatives; that is it allows you to make decision.
- Decisions made by a computer must be very simple since everything in the computer ultimately reduces to either true (1) or false (0).
- If complex decisions are required, it is the programmer's job to reduce them to a series of simple decisions that the computer can handle.

Selection Structure: Problem Examples

- Problem 1: Determine whether profit, return capital or loss.
- Problem 2: Determine whether a number is even or odd.
- Problem 3: Determine whether the marks is less than 60%. If it is less than 60, then print “fail”, otherwise print “pass”.
- Problem 4: Determine whether the speed limit exceeds 110 km per hour. If the speed exceeds 110, then fine = 300, otherwise fine = 0. Display fine.
- Problem 5: Determine whether the age is above 12 years old. If the age is above 12, then ticket = 20, otherwise ticket = 10. Display ticket.

Selection Structure (*cont..*)

- Pseudo code – requires the use of the keywords if.

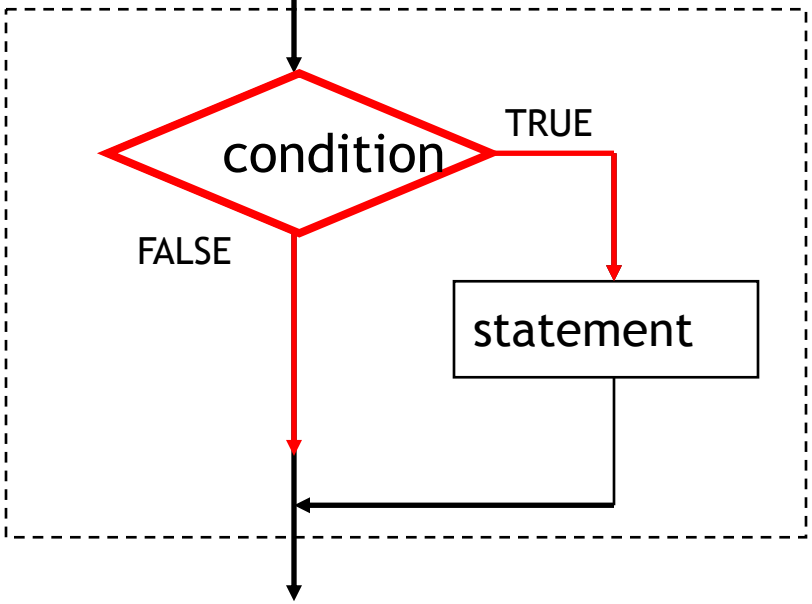
```
Algorithm: one choice selection
:
n.   if condition
          n.1 statement
n+1. end_if
:
```

Selection Structure (*cont..*)

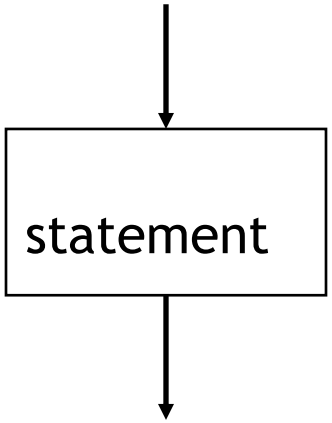
"do or don't"

If

(one-choice)



≡



If set condition is true, execute the statement, else do nothing

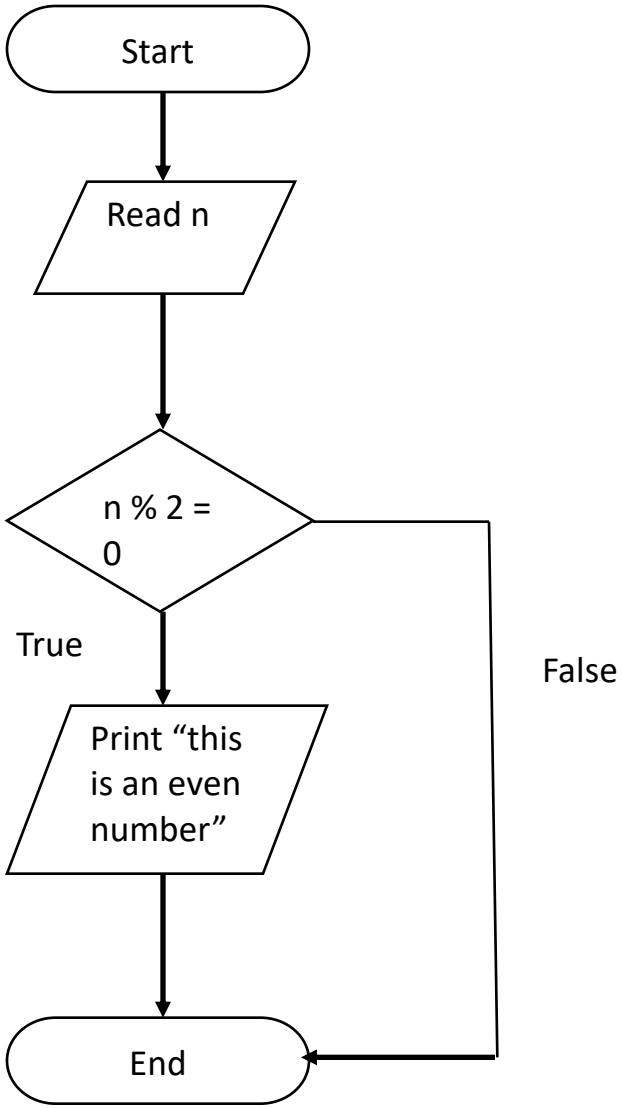
Example

- Determine whether an input number is even. If the number is even, print “This is even number”.

Pseudocode

1. Start
2. Read n
3. If n modulus 2 == 0
    1. Print “This is an even number”
4. End if
5. End

Flowchart



Selection Structure (*cont..*)

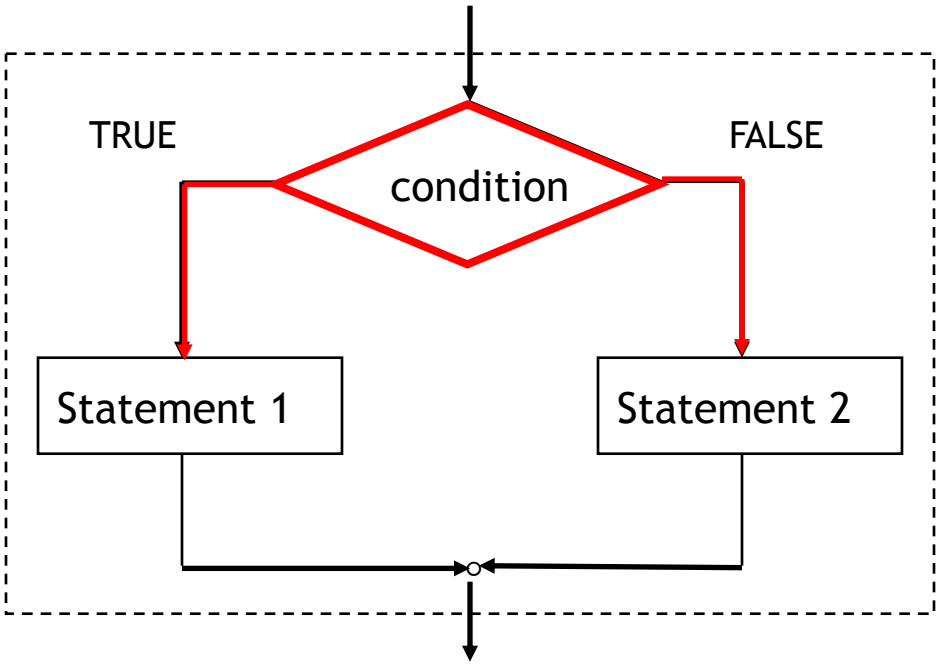
- Pseudo code – requires the use of the keywords `if` and `else`.

```
Algorithm: two choices selection
:
n.    if condition
            n.1 statement
            :
n+1.  else
      n+1.1 statement
      :
n+2.  end_if
:
```

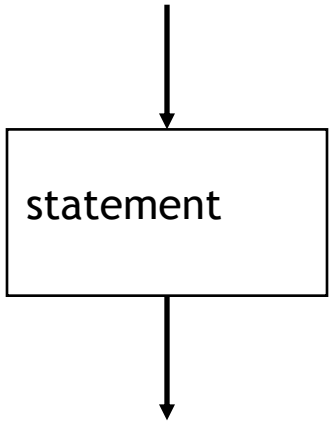
Selection Structure (*cont..*)

If-else
(two-choices)

"do this or do that"



≡



If set condition is true, execute the first statement, else execute second statement

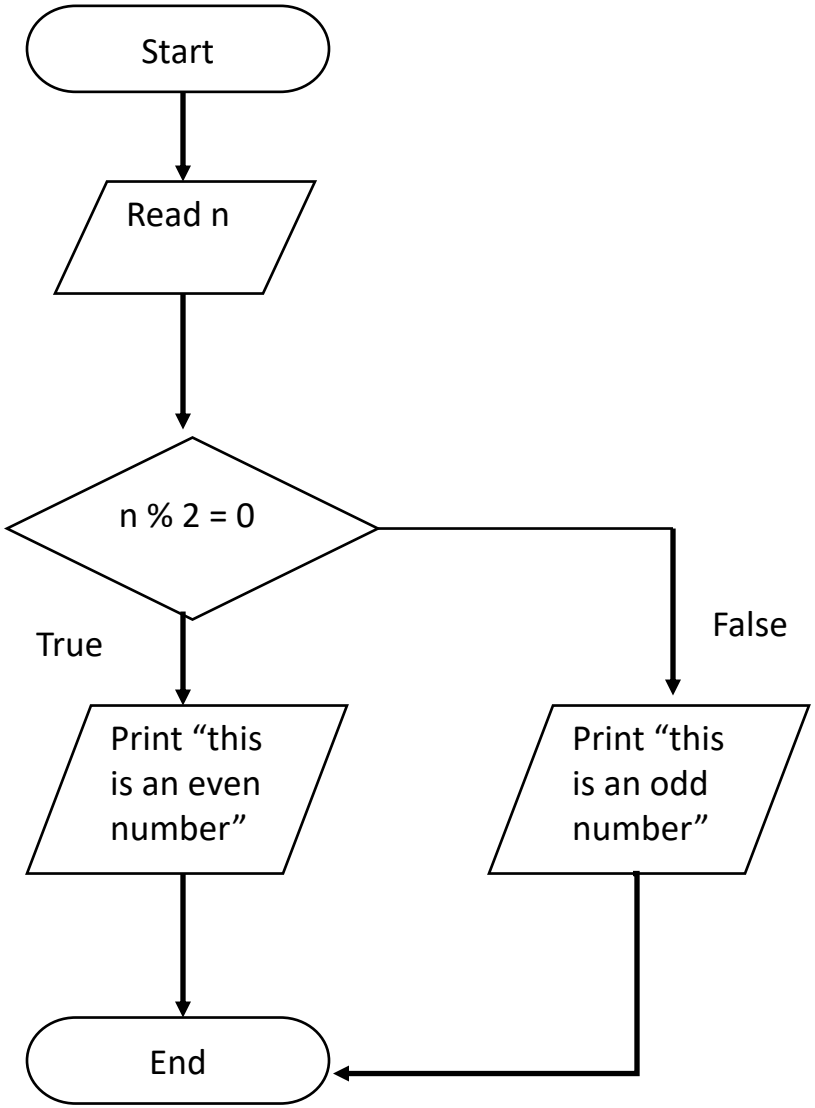
Example

- Determine whether an input number is even or odd. If the number is even, print “This is even number”. Else, print “This is odd number”.

Pseudocode

1. Start
2. Read n
3. If n modulus 2 = 0
    1. Print “This is an even number”
    2. Go step 6
4. Else
    1. Print “This is an odd number”
5. End if
6. End

Flowchart


Start
Read n
n % 2 = 0
True
False
Print “this is an even number”
Print “this is an odd number”
End

Selection Structure - Problem Examples

- Used to compare numbers to determine relative order
- Operators:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Relational Expressions

- Boolean expressions – `true` or `false`
- Examples:

12 > 5 is true
7 <= 5 is false
if x is 10, then
x == 10 is true,
x != 8 is true, and
x == 8 is false

Logical Operators

- Used to create relational expressions from other relational expressions

- Operators, meaning, and explanation:

& &	AND	New relational expression is true if both expressions are true
| |	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression – true expression becomes false, and false becomes true

Truth Table

AND (&&)		
P	Q	P && Q
T	T	T
T	F	F
F	T	F
F	F	F

OR (||)		
P	Q	P||Q
T	T	T
T	F	T
F	T	T
F	F	F

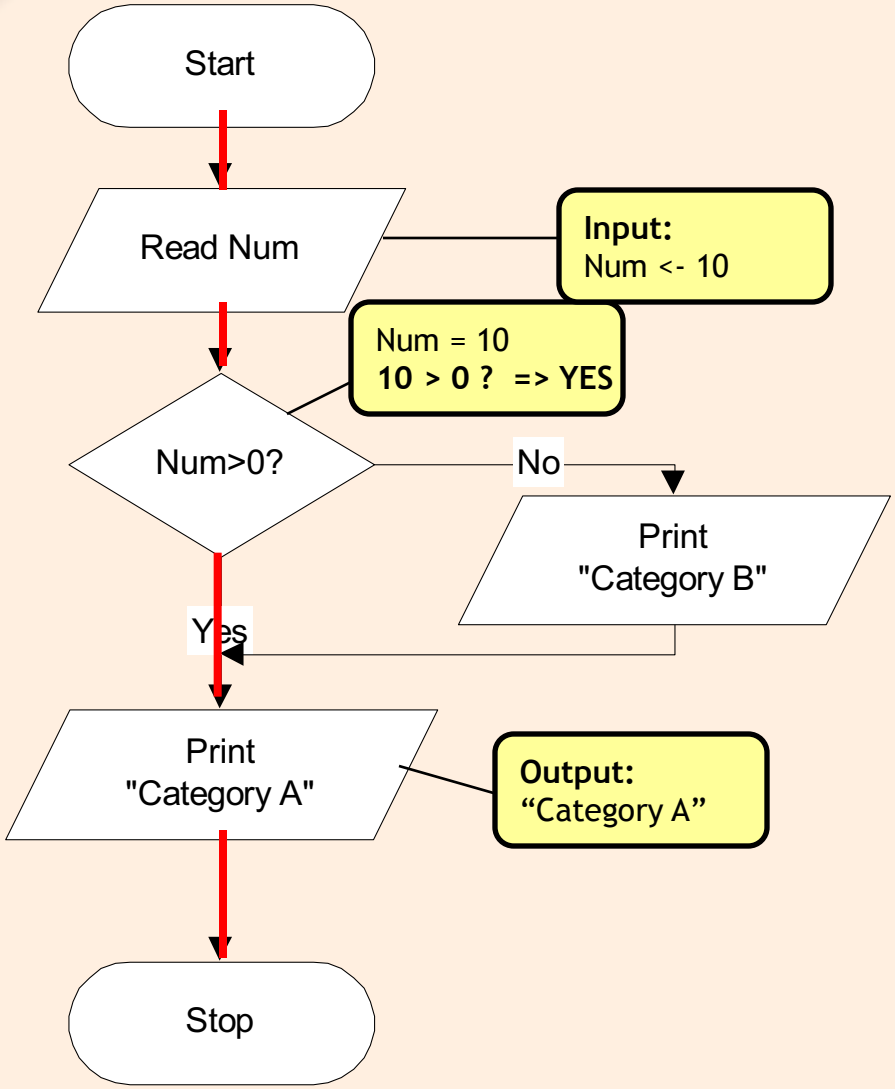
Logical Operators - examples

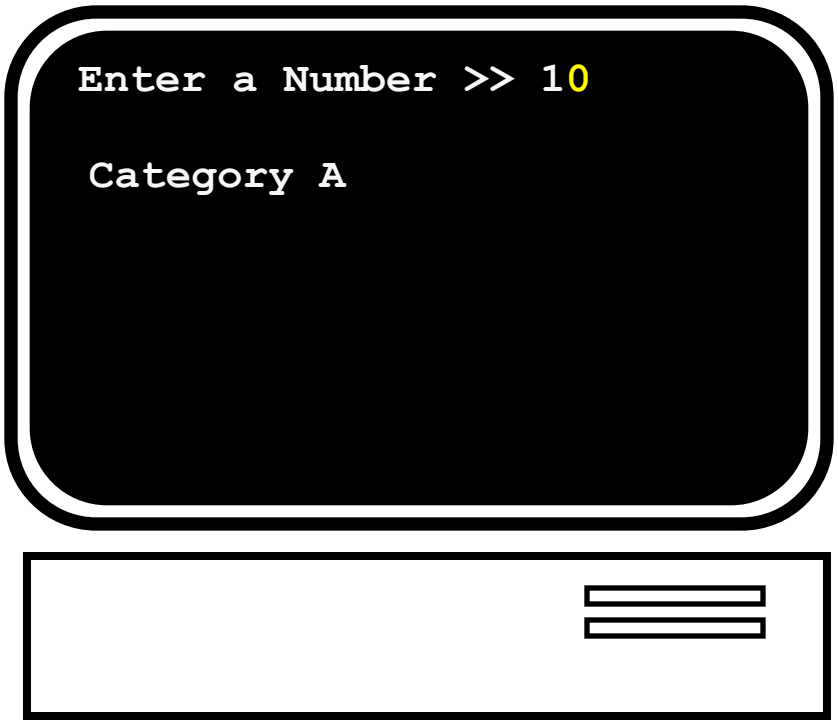
```
int x = 12, y = 5, z = -4;
```

`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

Example:

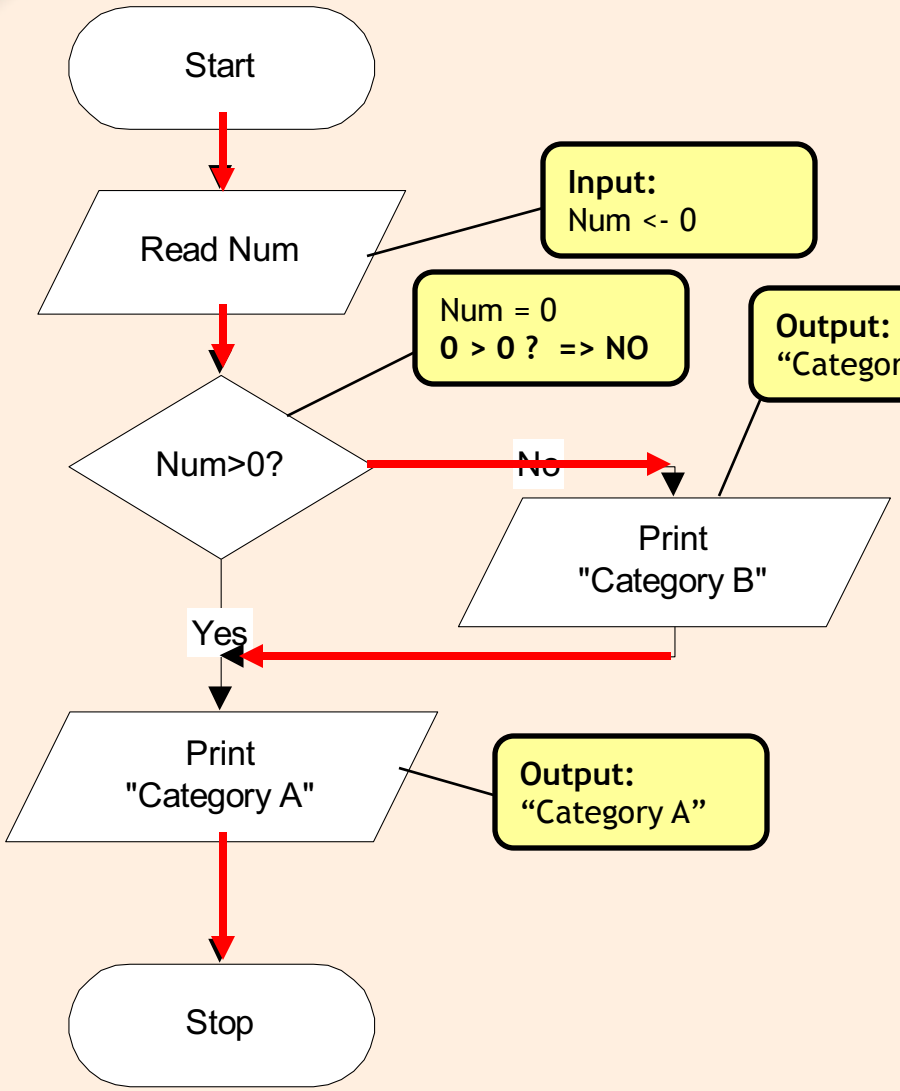
What is the output of the following flowchart when the input Num= 10

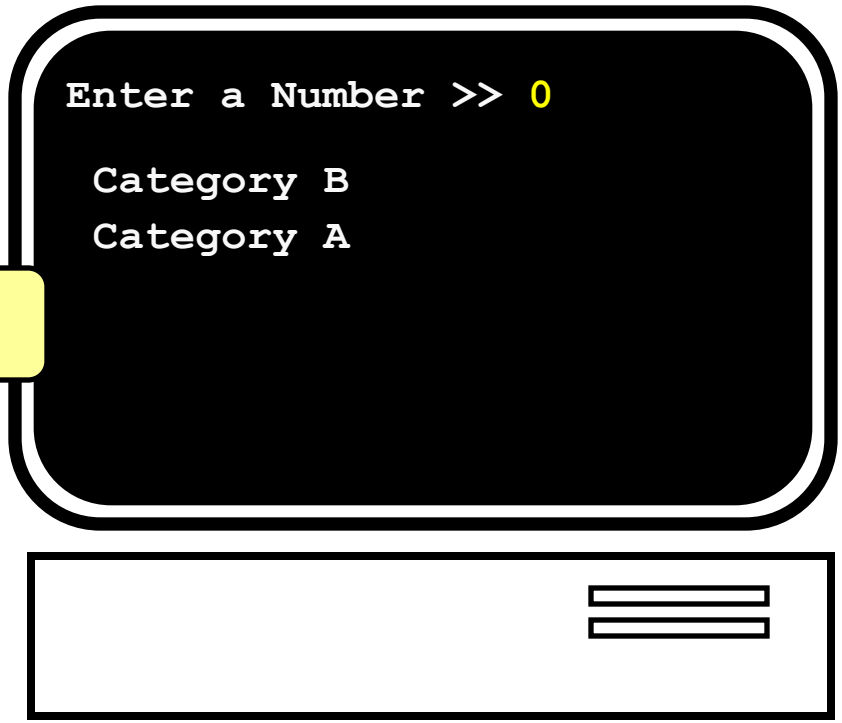




Example:

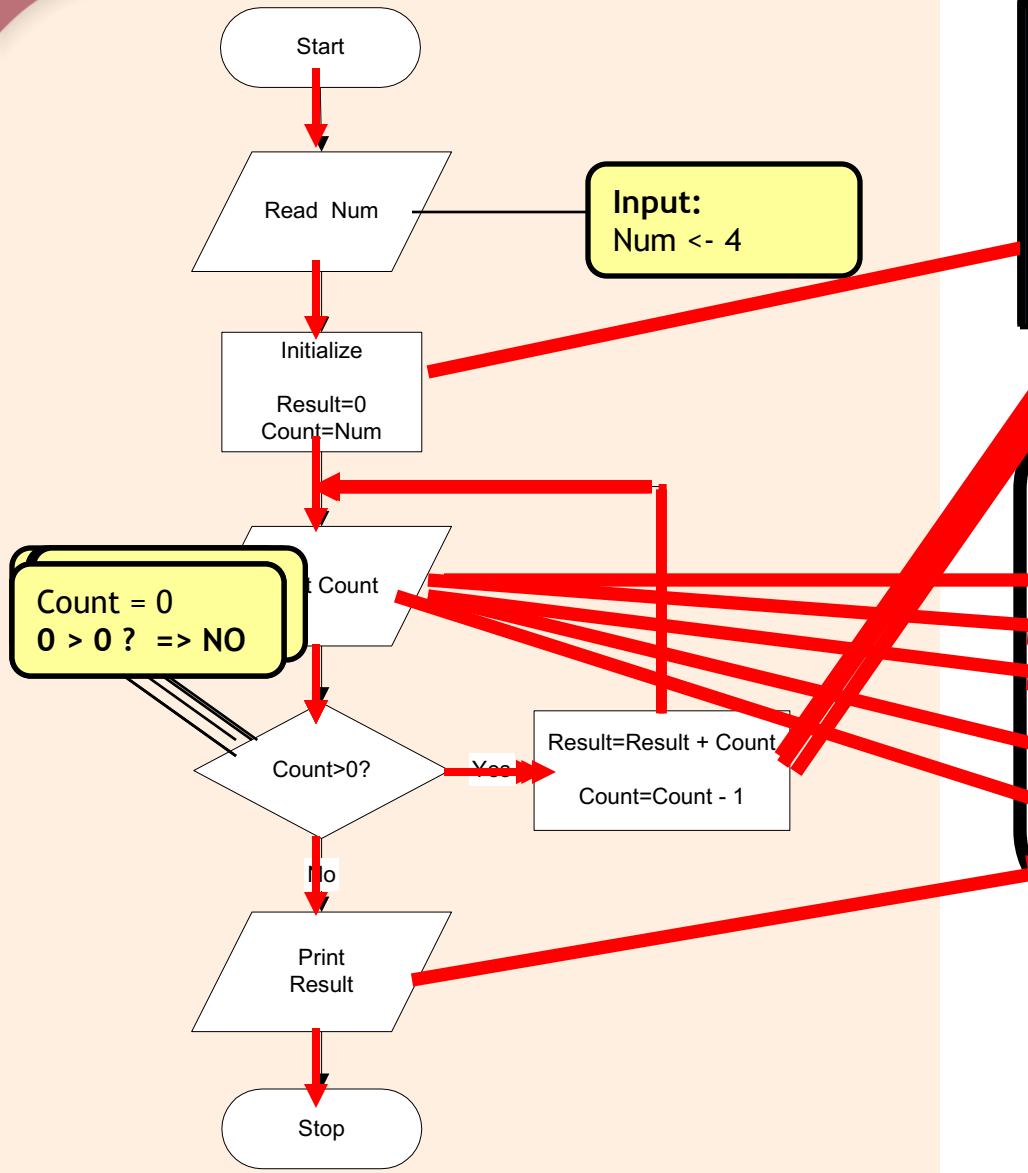
What is the output of the following flowchart when the input is Num= 0





Example:

What is the output of the following flowchart when the input Num= 4

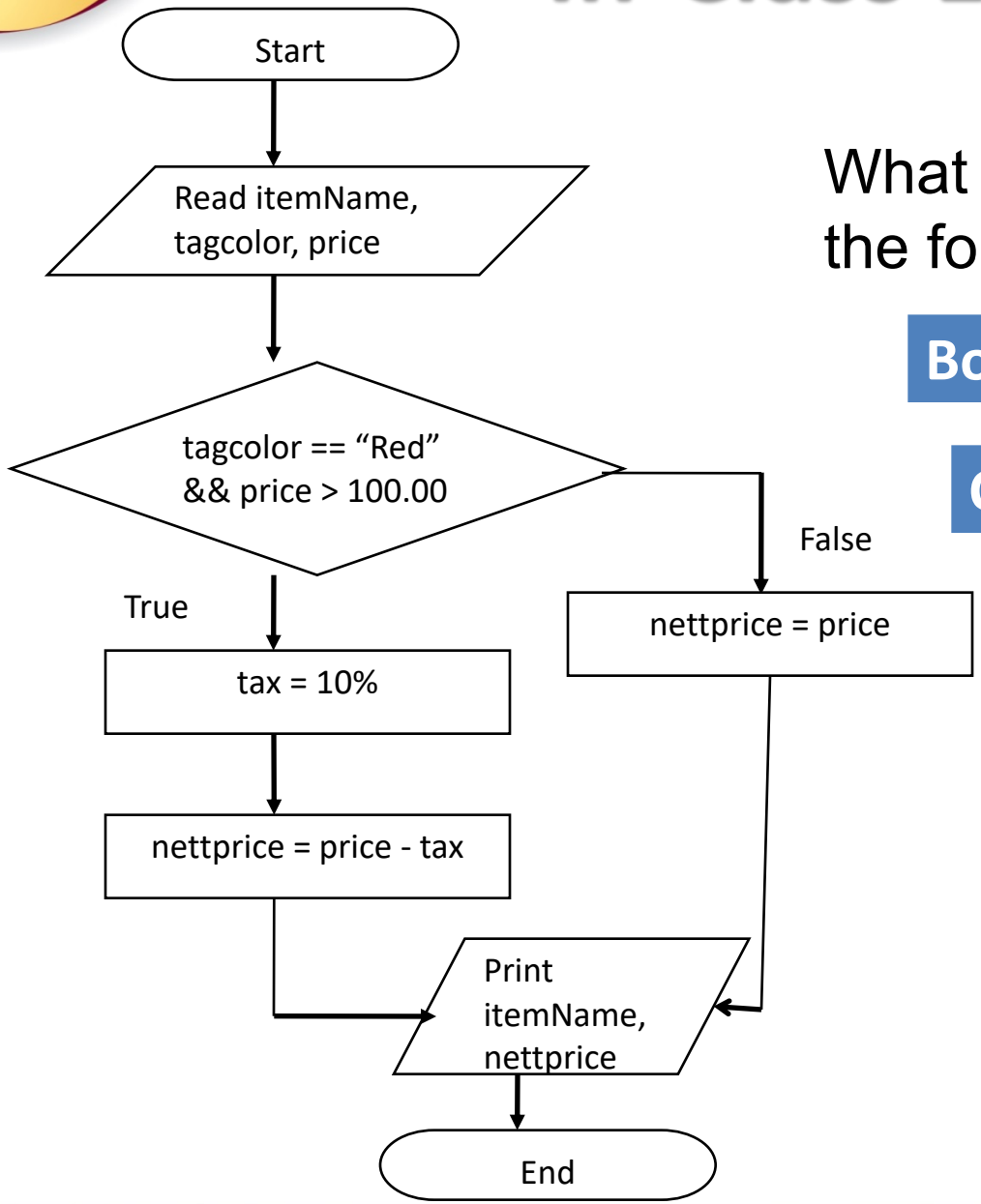


Variables *(in memory)*:

```
Num     [ 4 ]
Result  [ 10]  9 + 1
Count   [ 0 ]  1 - 1
```

```
Enter a Number => 4
Count: 4
Count: 3
Count: 2
Count: 1
Count: 0
Result: 10
```

In-Class Exercise



What will be the output for the following input?

Book Green 350.00

Curtain Red 500.00

In-Class Exercise 2

- Write a pseudo code for a program that will accept 2 numbers. If the first number is greater than the second number, find the difference between the numbers and print the numbers and difference value. If the second number is greater than the first number, find the sum of the two values and print the numbers and the sum.
- Draw the flowchart for the pseudo code.
- Trace the algorithm with the following input. Write the output:

40	50
70	30

In-Class Exercise 3

- Write down an algorithm (pseudo code) and draw a flowchart to read two numbers. If the first number is greater than the second number and it is larger than 50, find the sum and print the sum. Else, print the difference.

- Verify your result by a trace table. (Use 52, 30 as the numbers read)

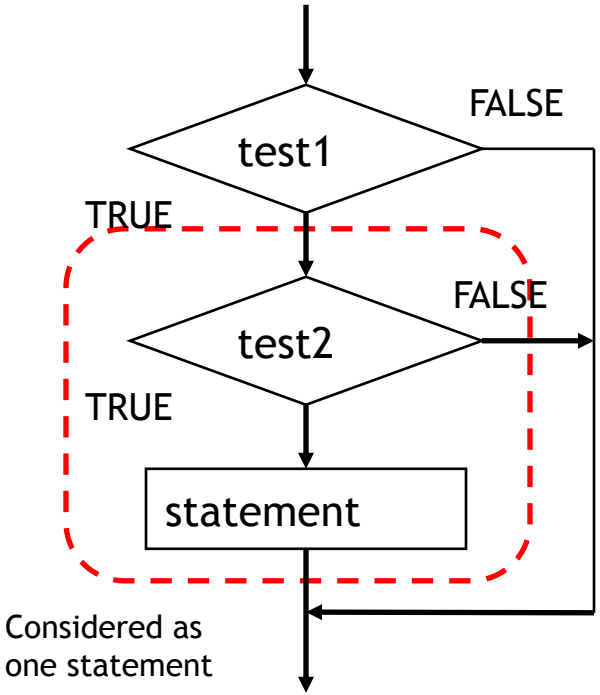
Selection Structure (*cont..*)

- Pseudo code – nested if.

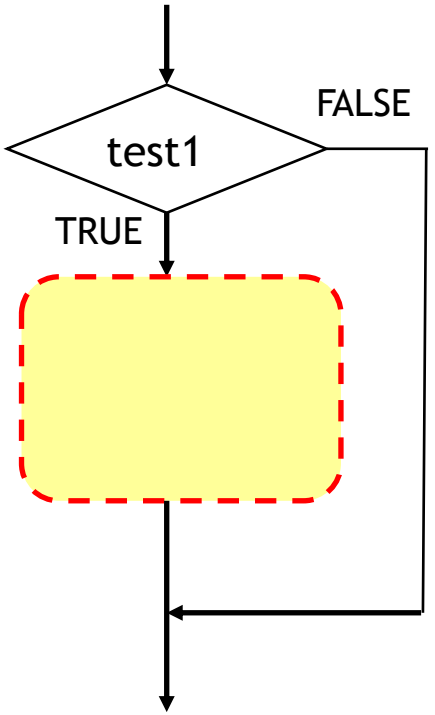
```
Algorithm: nested if
:
n. if condition
      :
      n.m if condition
            n.m.1 statement
            :
n+1. end_if
:
```

Selection Structure (*cont..*)

Nested if
(if within if)



≡



it is an “one-choice” if

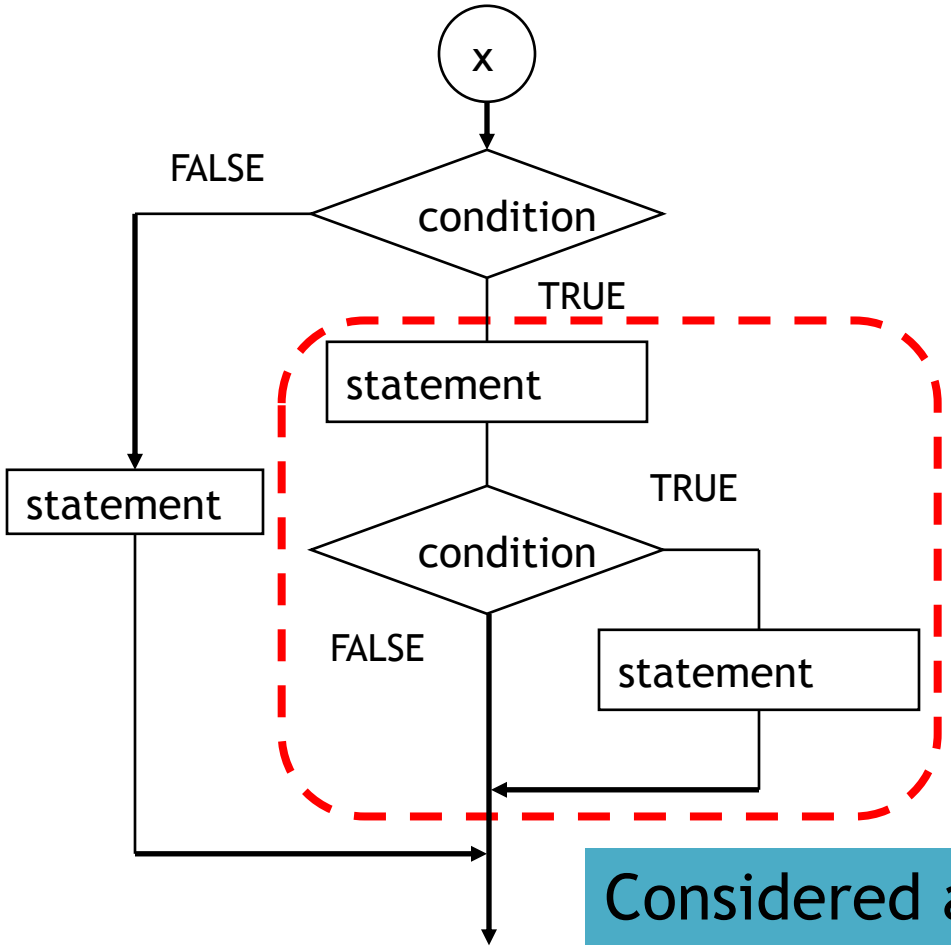
Selection Structure (*cont..*)

- Pseudo code – nested if using if-else if.

```
Algorithm: if-else if
:
n. if condition
      n.m if condition
            n.m.1 statement
            :
n+1.  else
      n+1.m.1 statement
      :
n+2.  end_if
:
```

Selection Structure (*cont..*)

Complex if-else & if Statements

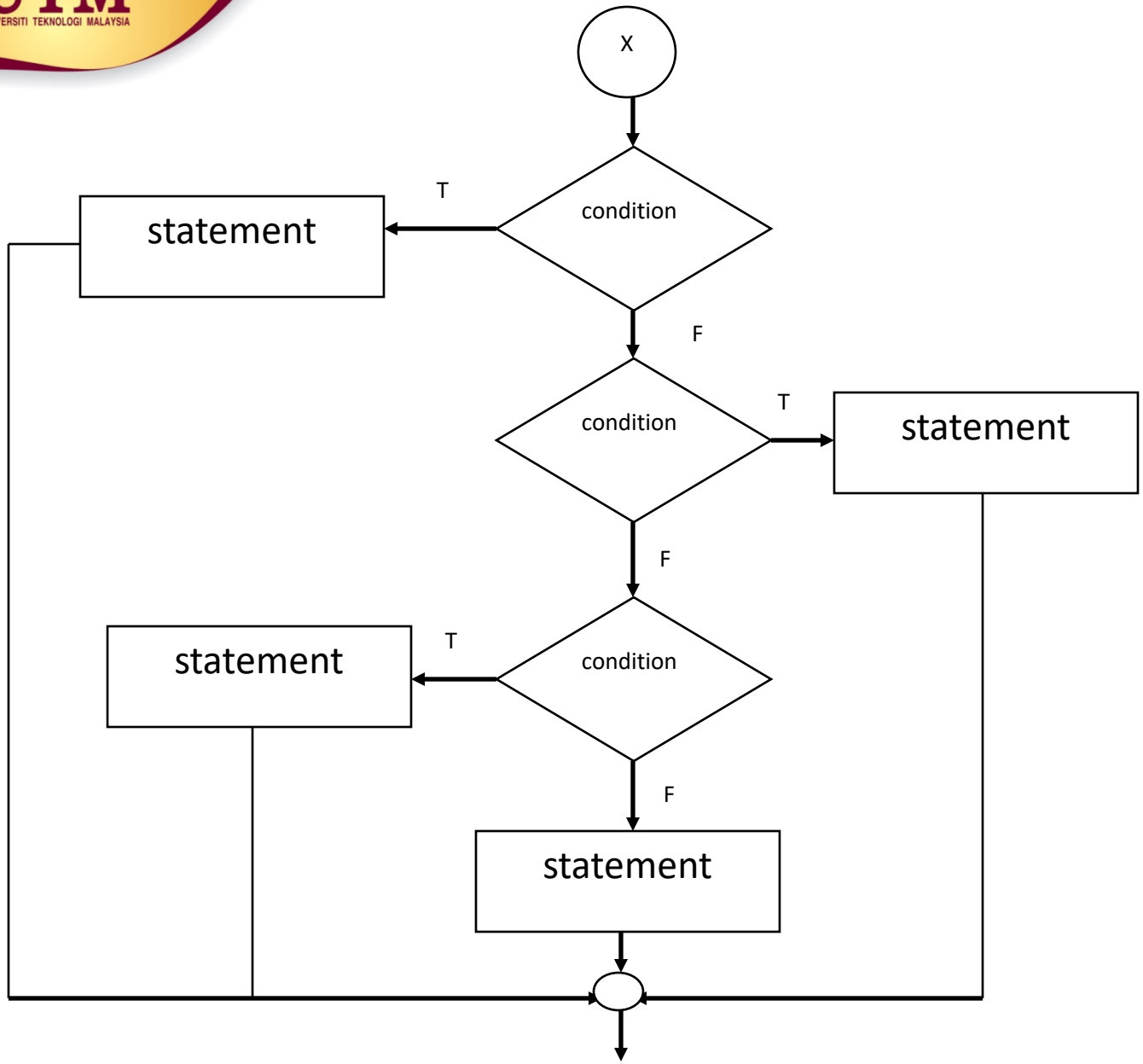


Considered as one statement

Selection Structure (*cont..*)

- Pseudo code – nested if using if-else if- else if - else.

```
Algorithm: if-else if – else if -
else
:
n. if condition
      n.m statement
n+1   else if condition
      n+1.1 statement
      :
n+2   else if condition
      n+2.1 statement
n+3   else
      n+3.1 statement
n+4 end_if
:
```


X
condition
T
statement
F
condition
T
statement
F
condition
T
statement
F
statement

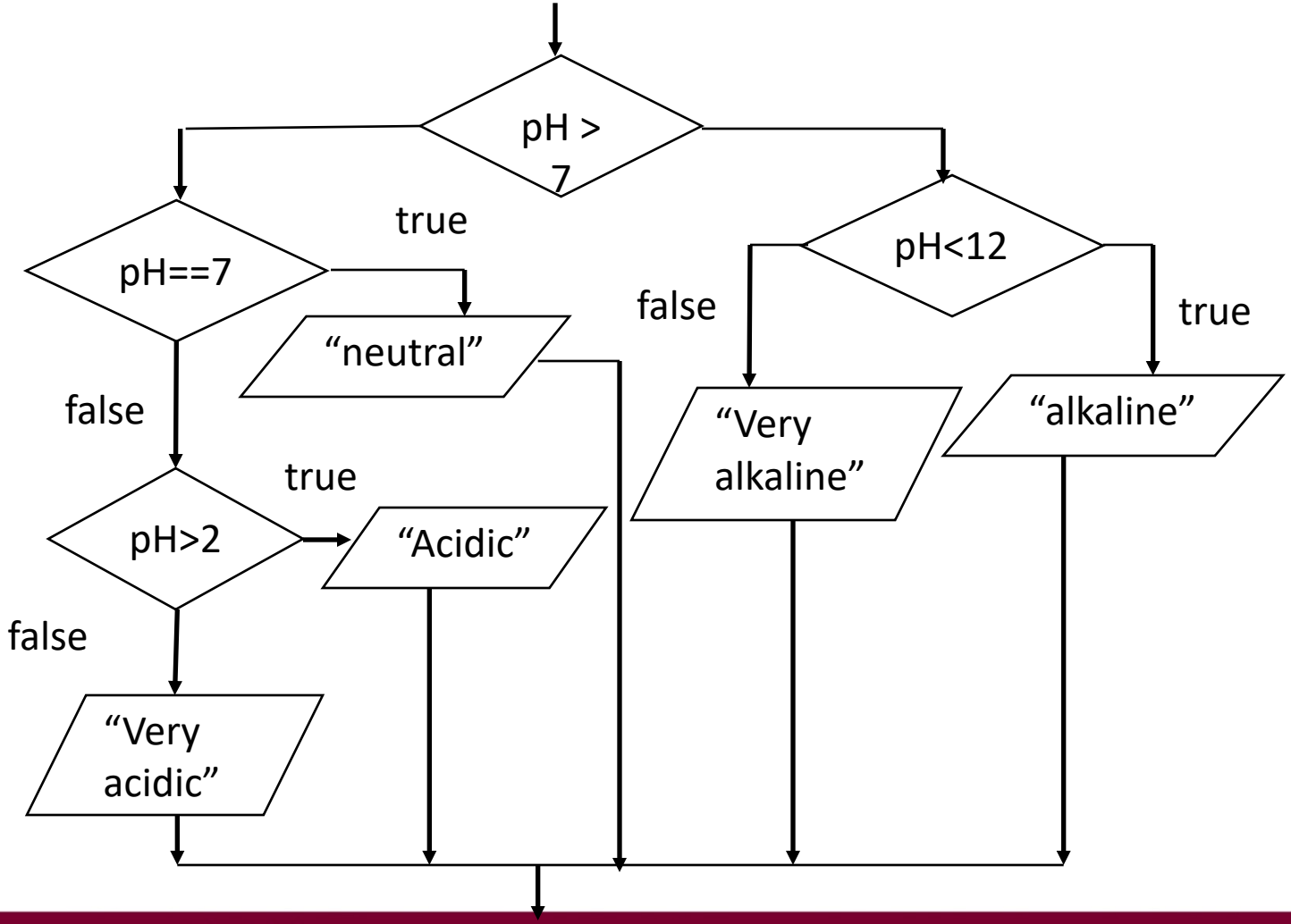
In-Class Exercise 1

- Suppose we want to associate noise loudness measured in decibels with the effect of the noise. The following table shows the relationship between noise levels and human perception of noises. Draw a flow chart.

Loudness in decibels (db)	Perception
50 or lower	Quiet
51-70	Intrusive
71 – 90	Annoying
91- 110	Very annoying
Above 110	uncomfortable

In-Class Exercise 2

- Write a pseudo code for nested if as illustrated in the flow chart below:



In-Class Exercise 3

- Write a pseudo code and draw a flow chart for a program that will implement the following decision table. The program will print the transcript message based on the input grade point.

Grade Point Average	Transcript Message
0.0 – 0.99	Failed
1.0 – 1.99	On probation
2.0 – 2.99	Average
3.0 – 3.49	Dean's List
3.5 – 4.00	Highest Honors

Flowchart Structures: Repetition

Repetition Structure

- Specifies a block of one or more statements that are repeatedly executed until a condition is satisfied.

- Usually the loop has two important parts:
  - An expression that is tested for a true/false,
  - A statement or block that is repeated as long as the expression is true

- 2 styles of repetition or loop
  - Pre-test loop
  - Post test loop

Repetition Structure - Counters

- Counter: Can be used to control execution of the loop (loop control variable)
- It will increment or decrement each time a loop repeat
- Must be initialized before entering loop

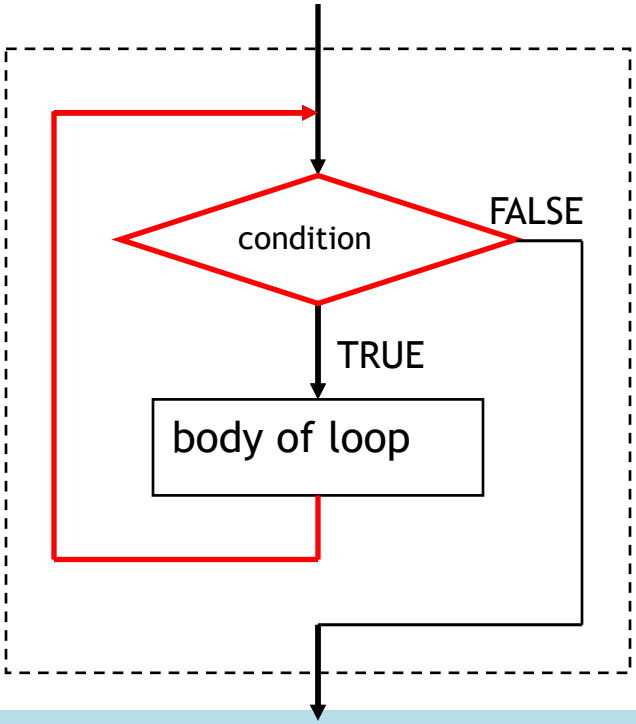
Repetition Structure: Pre-Test Loop

- Pseudo code - requires the use of the keywords `while` for pre-test loop.

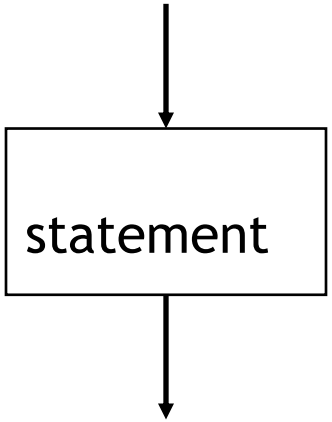
```
Algorithm: one choice selection
:
n.   while condition
          n.1 statement
                :
n+1. end_while
:
```

Repetition Structure (*cont..*)

while Loop
(pre-test loop)

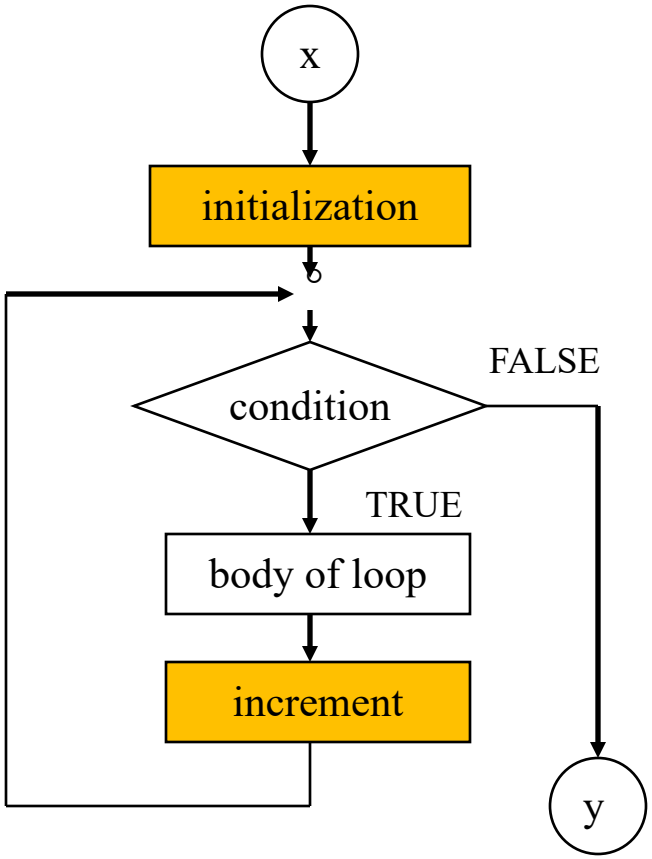


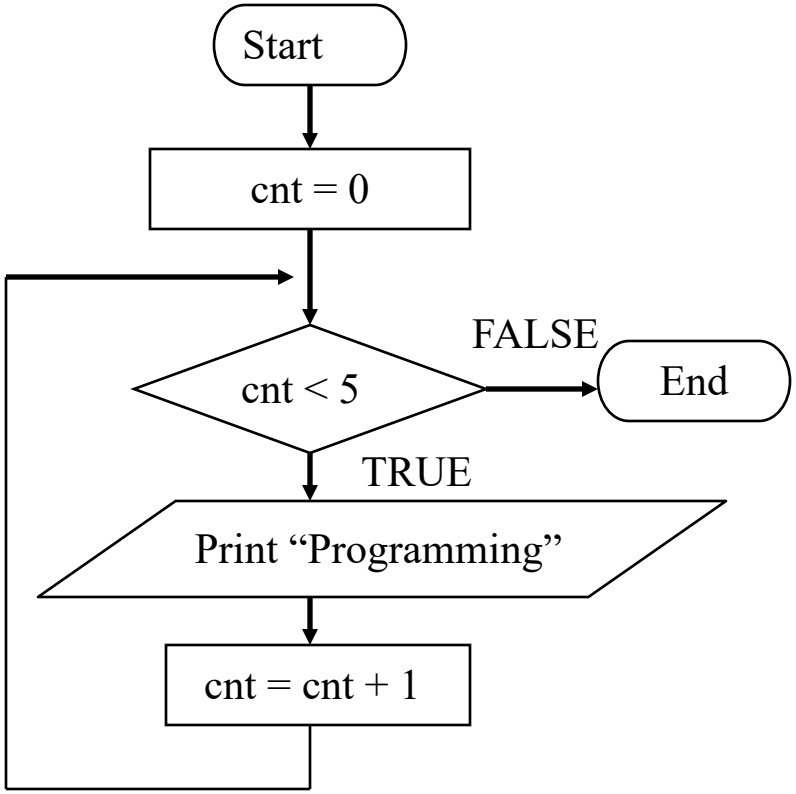
≡



While a set condition is true, repeat statement (body of loop)

Repetition Structure (*cont..*)





Pre-test loop steps summary

- Counter-controlled loop
  - Initialization of counter: counter = 0
  - Testing of counter value: counter > n
  - Updating of counter value (increase by 1) during each iteration: counter = counter + 1

Example

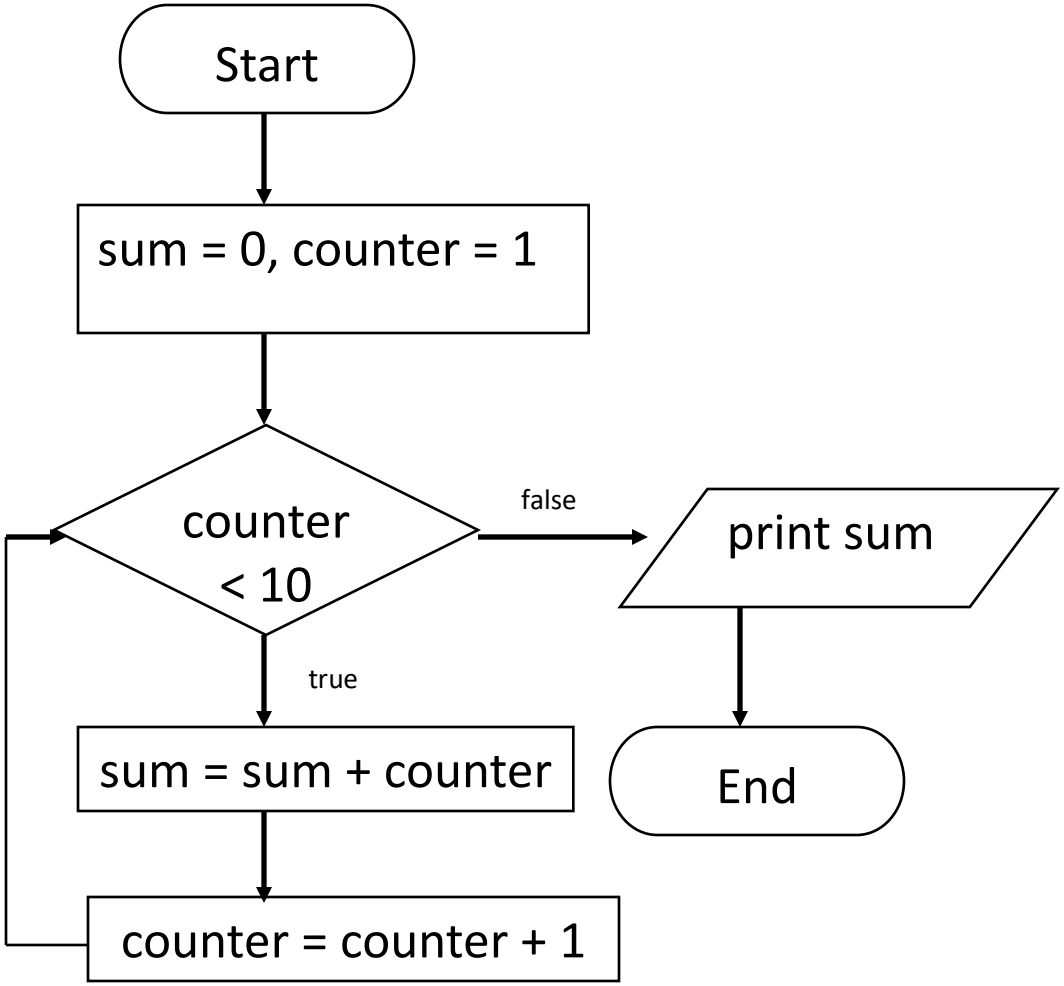
- Suppose we want to write a program to compute a sum of the first 10 positive integers.

- Steps:
  - How many repetition?
    - Initialization
    - Condition to check for the counter?
    - Update of counter

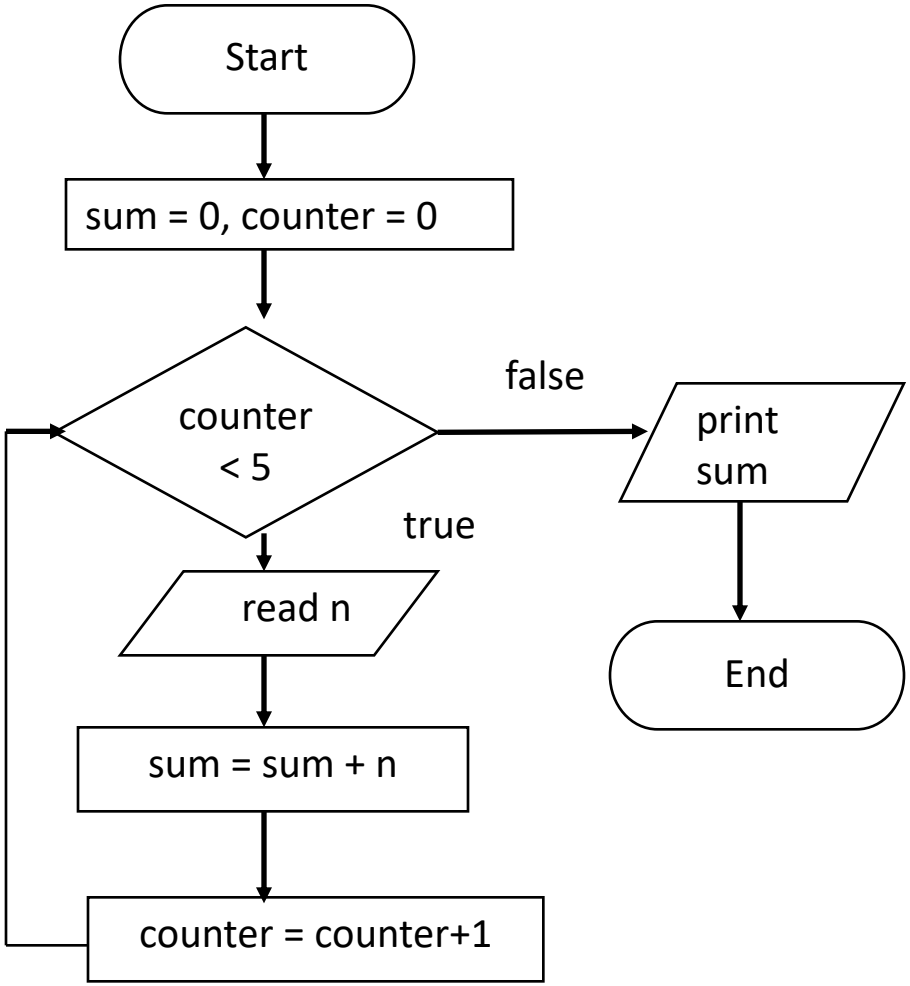
Pseudo code

1. Start
2. Set sum=0, counter = 0
3. While (counter < 10)
   - 3.1 sum = sum + counter
   - 3.2 counter = counter + 1
4. End_While
5. Display sum
6. End

Flow Chart


Start
sum = 0, counter = 1
counter
< 10
false
print sum
true
sum = sum + counter
End
counter = counter + 1

Trace the following



What is the output for the following input:

20 30 40 50 10

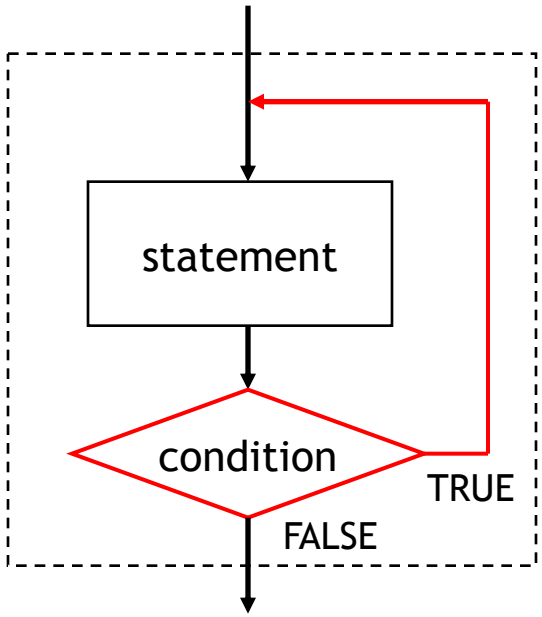
Repetition Structure: Post-Test

- Pseudo code - requires the use of the keywords `repeat..until` for post-test loop.

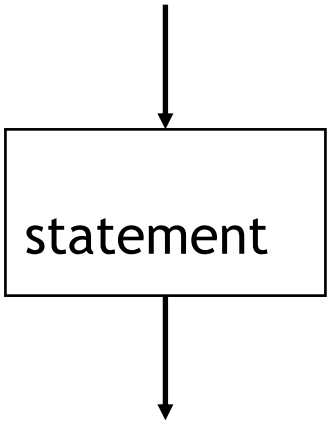
```
Algorithm: one choice selection
:
n.   Do
          n.1 statement
                :
n+1. While condition
:
```

Repetition Structure (*cont..*)

do-while Loop
(post-test loop)



≡

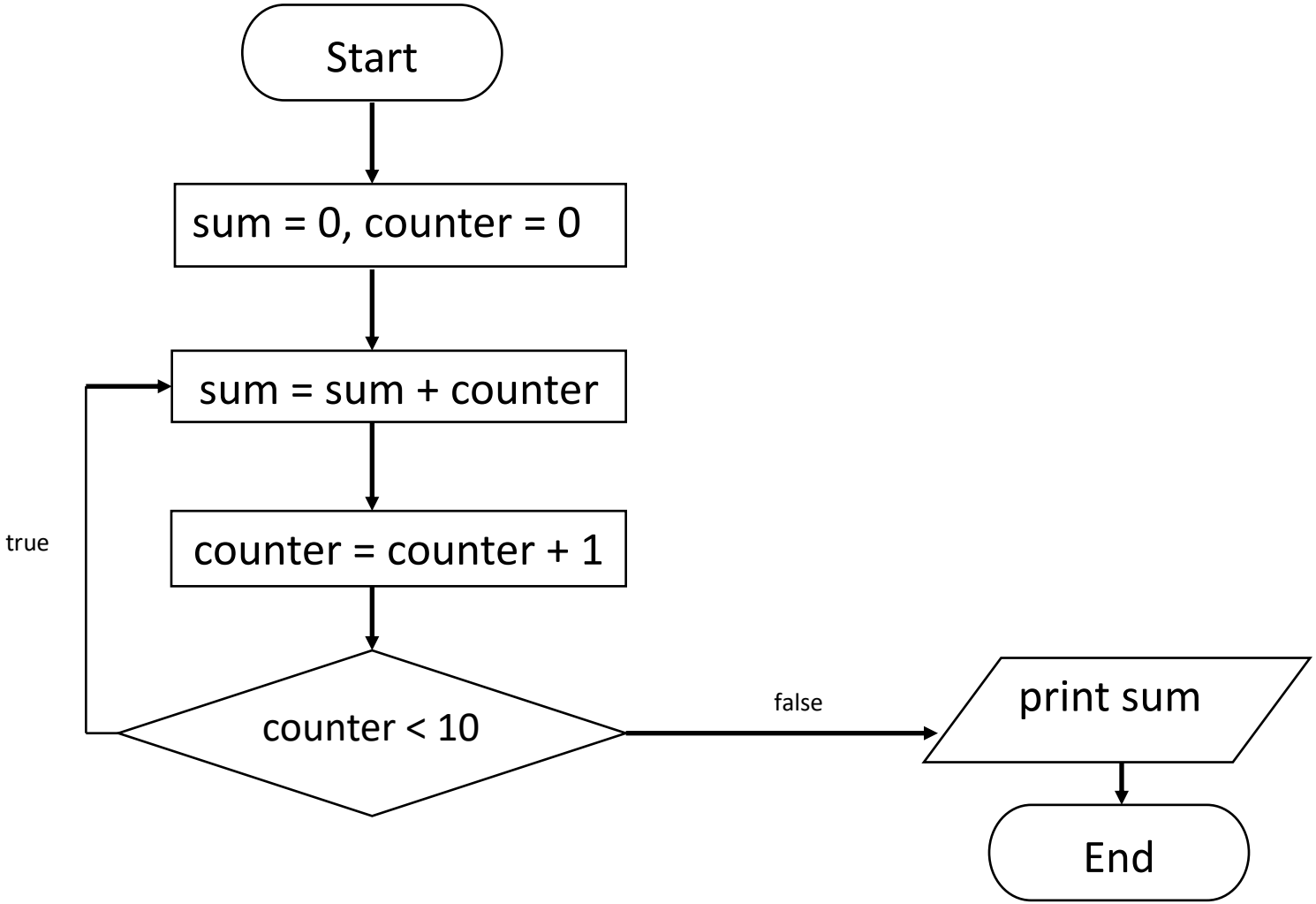


- Do the statement (body of loop) while a condition is true.
- The loop body is executed at least once.

Example

1. Start
2. Set sum = 0, counter = 0
3. do
   - 3.1 sum = sum + counter
   - 3.2 counter = counter + 1
4. while (counter < 10)
5. Display sum
6. End

Flow Chart


Start
sum = 0, counter = 0
sum = sum + counter
counter = counter + 1
counter < 10
true
false
print sum
End

In-Class Exercise 1

- Develop an algorithm (pseudo code) and flow chart for a program to calculate an average of 15 numbers input by the user. Use pre-test loop
- Modify your solution above by using the post-test loop.

In-Class Exercise 2

- Develop an algorithm and flow chart to print even numbers between 1 to 50. Use pre-test loop.
- Modify your solution by using post-test loop.

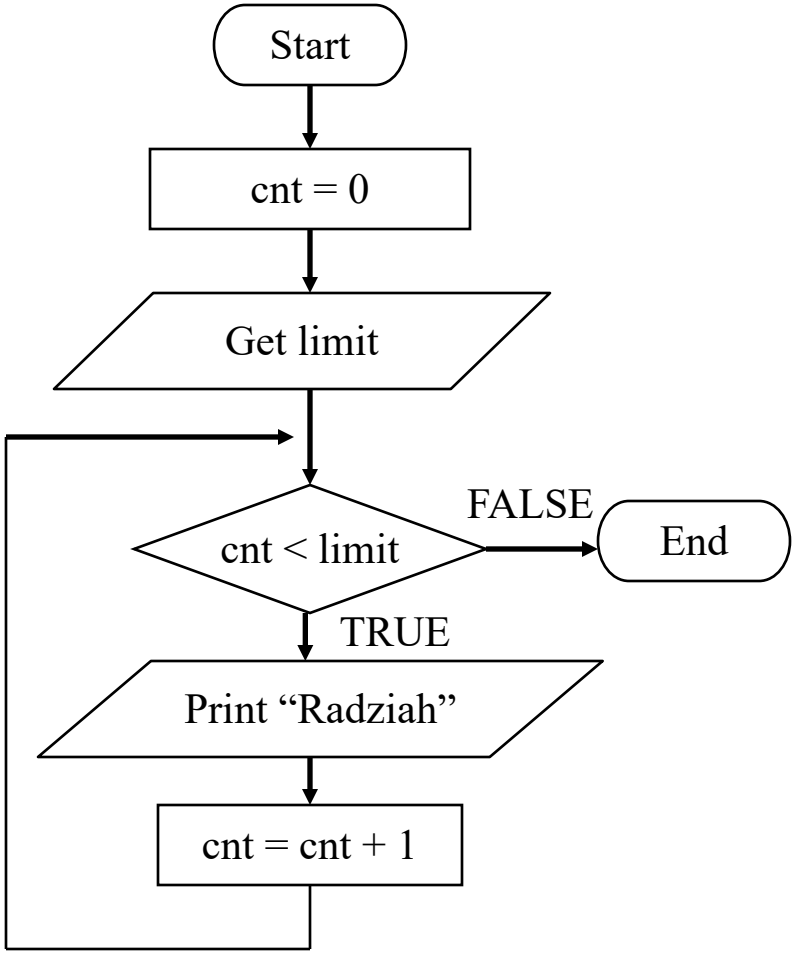
Repetition Structure - Letting the User Control a Loop

- Program can be written so that user input determines loop repetition.

- Used when program processes a list of items, and user knows the number of items

- User is prompted before loop. Their input is used to control number of repetitions

Repetition Structure (*cont..*)



Repetition Structure - Sentinels

- sentinel: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, *e.g., -999 for a test score*
- Used to terminate input when user may not know how many values will be entered

Repetition Structure - Sentinels

```
Algorithm: Loop control by sentinel value

1. Start
2. Set repeat = 1
3. while (repeat == 1)
   3.1 Read no1
   3.2 Read no2
   3.4 Print no1 + no2
   3.5 Read repeat
4. end_while
5. End
```

In-Class Exercise 1

Trace the following pseudo code:

1. Start
2. Set product = 1, number = 1, count = 20
3. Calculate: lastNumber = 2 * count – 1
4. while (number <= lastNumber)
   - 4.1 product = product * number
   - 4.2 number = number + 2
5. end_while
6. Display product

In-Class Exercise 2

- Convert the pseudo code in In-Class Exercise 1 to its flow chart.
- Convert the while loop in In-Class Exercise 1 to do..while loop. Draw its respective flow chart.

In-Class Exercise 3

- Bina Education Sdn. Bhd. wants you to develop a program for finding experience teachers for its offered course. Your program will request name and number of years teaching from the applicants. To be accepted as the teacher, the applicant must have at least 8 years of teaching experience. Your program will display list of successful applicants' names, numbers of successful applicants and average of numbers of years of teaching experience of successful applicants. Your program will terminate when the name input is "OK".

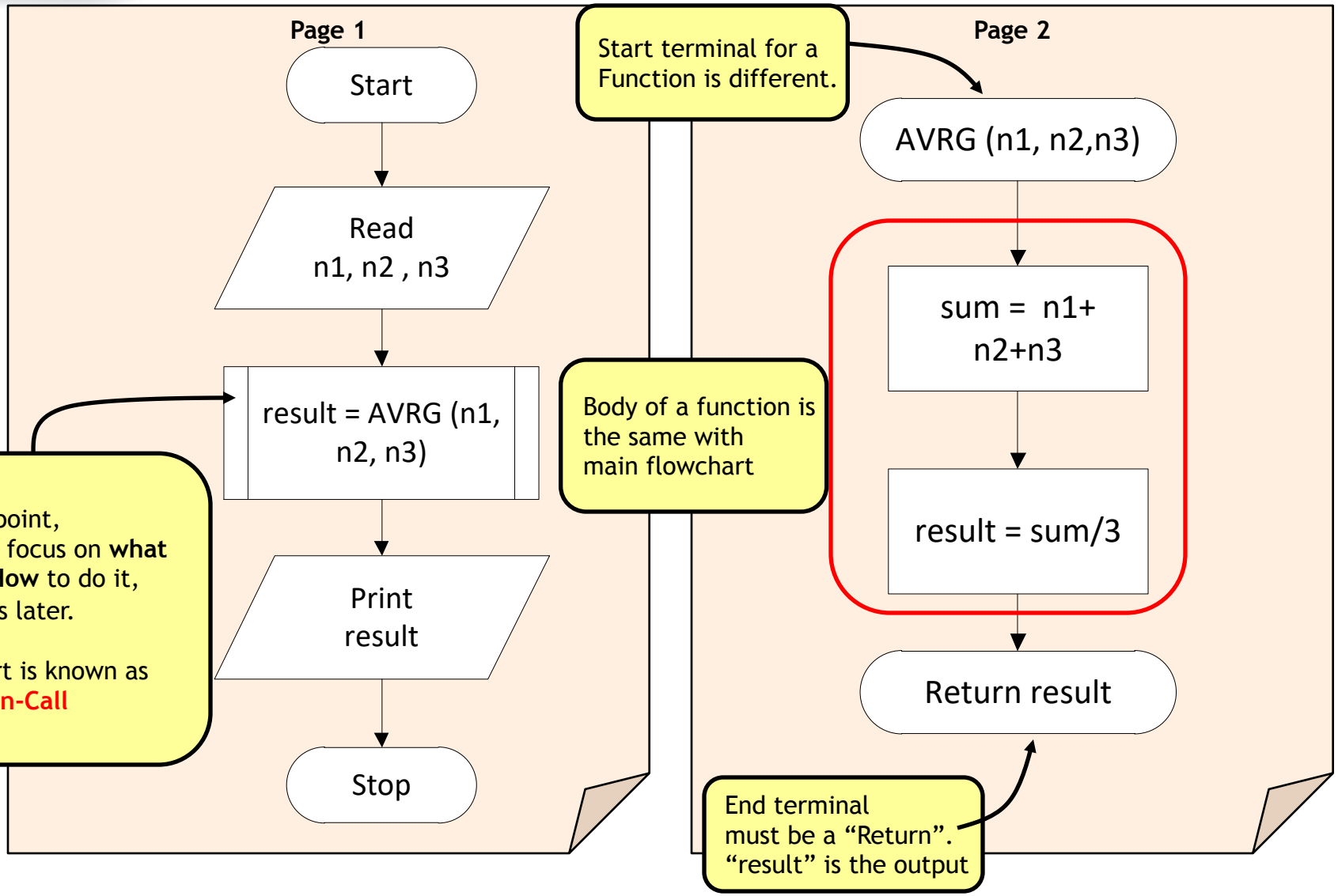
Modular Flowcharting

Function

Problem: *To average three numbers*

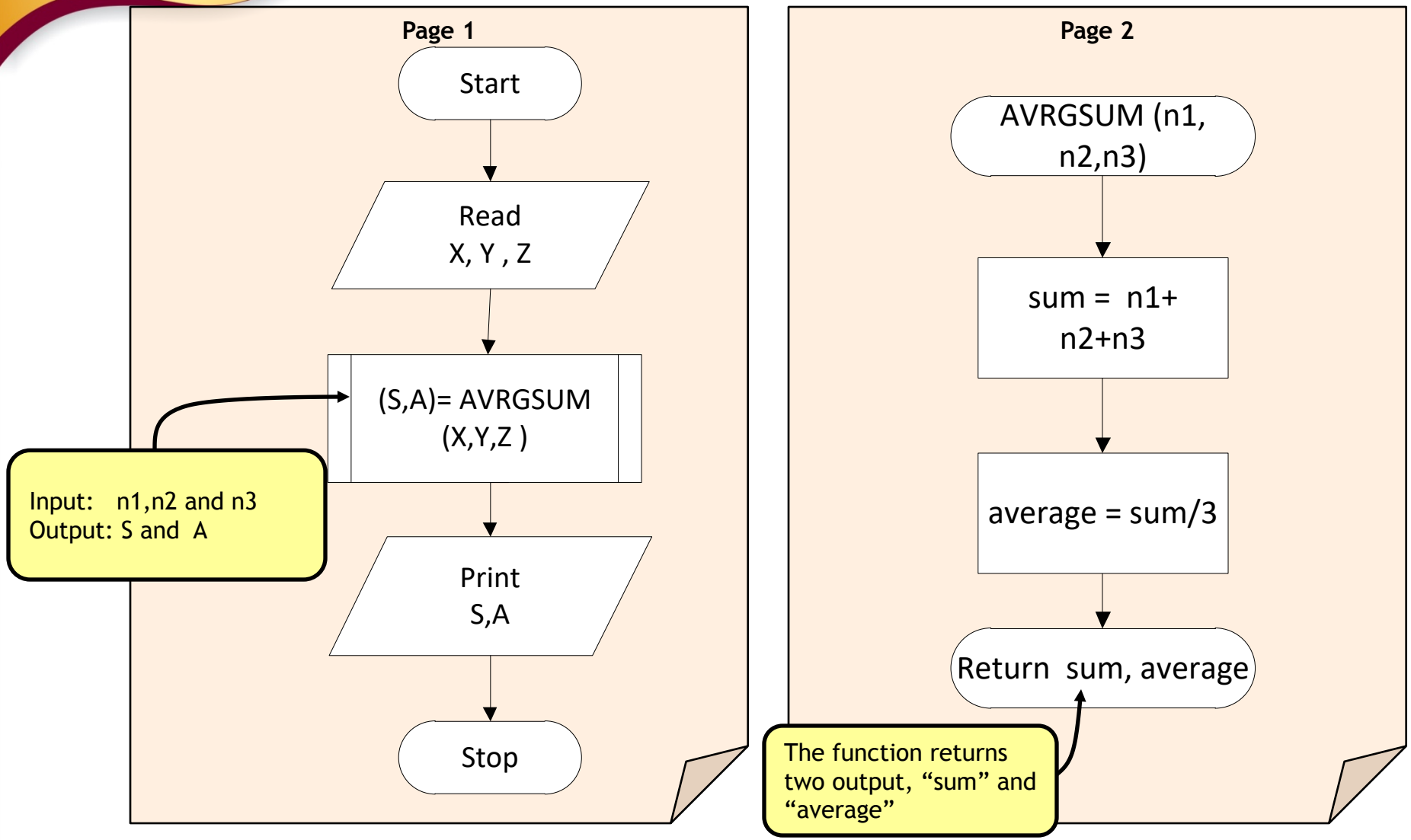
The detail of **how the function works** is put in another flowchart.

This is known as **Function-Definition**



Flowchart AVRG calculates the average of three numbers

Problem: *To sum and average three numbers*



Flowchart AVRGSUM calculates the total and average of three numbers

Related Terms and Concepts

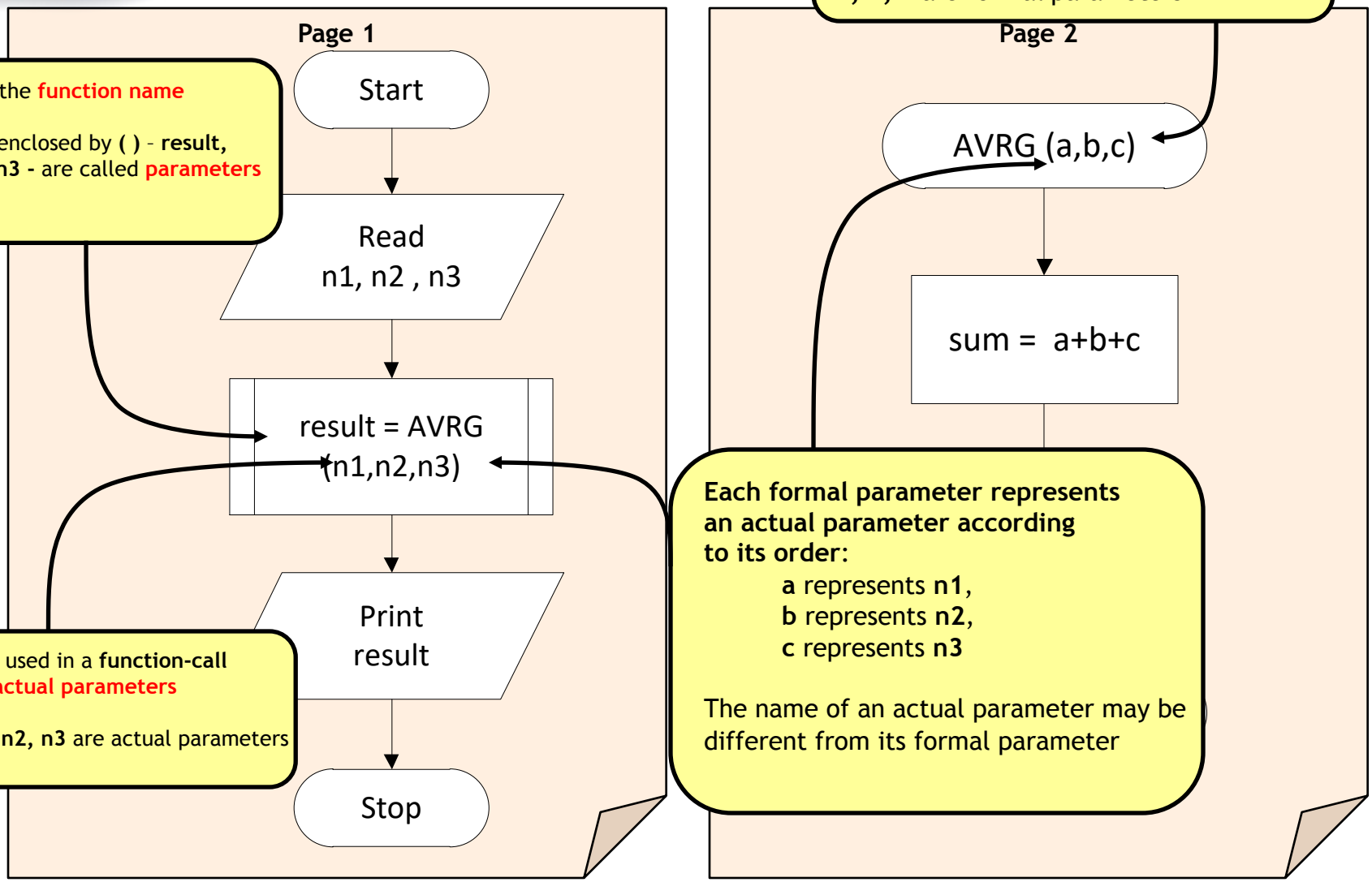
Actual and Formal Parameters

Parameters used in a **function-definition** are called **formal parameters**

a, b, c are formal parameters

AVRG is the **function name**

Objects enclosed by **( )** - **result, n1, n2, n3 -** are called **parameters**



Parameters used in a **function-call** are called **actual parameters**

result, n1, n2, n3 are actual parameters

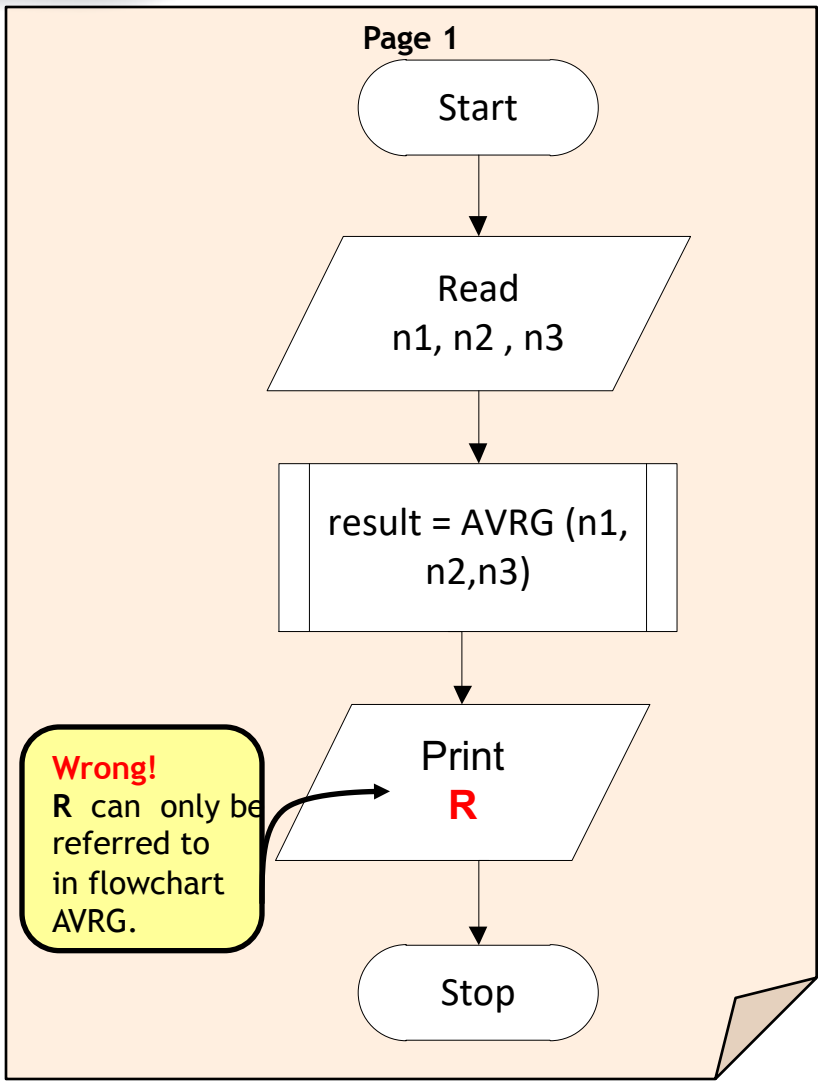
Each formal parameter represents an actual parameter according to its order:

a represents **n1**,
b represents **n2**,
c represents **n3**

The name of an actual parameter may be different from its formal parameter

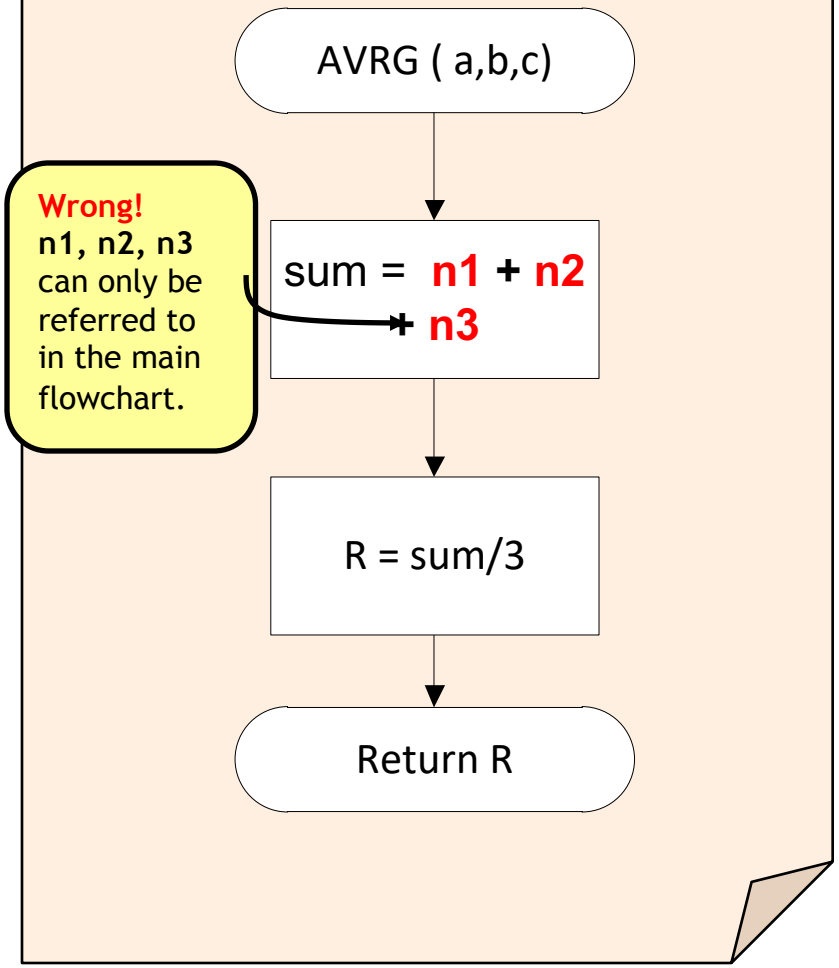
Flowchart AVRG calculates the average of three numbers

Only refer to variables that exist in the current chart

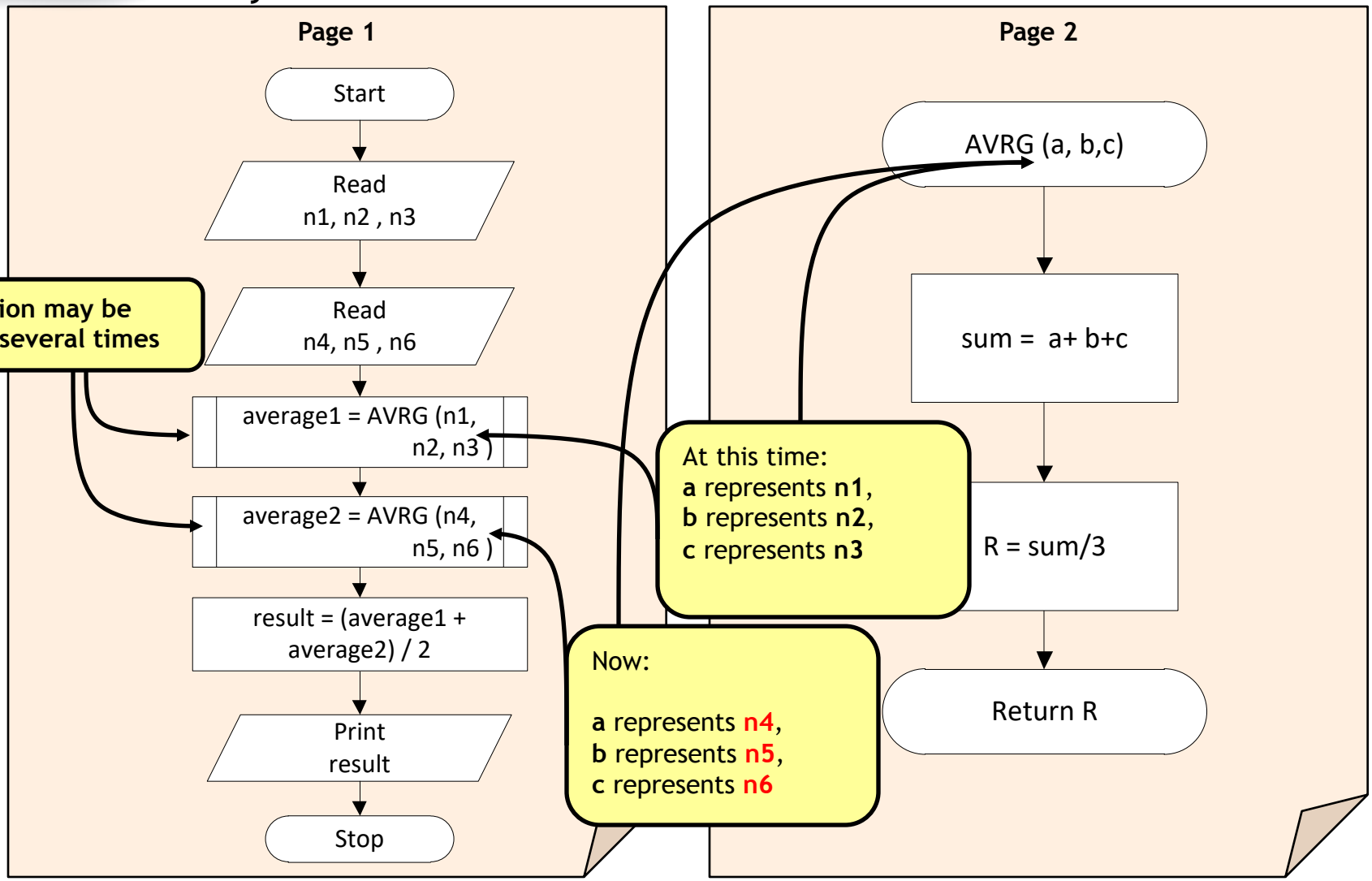


In a function-definition, you should only use formal parameters : a, b, c

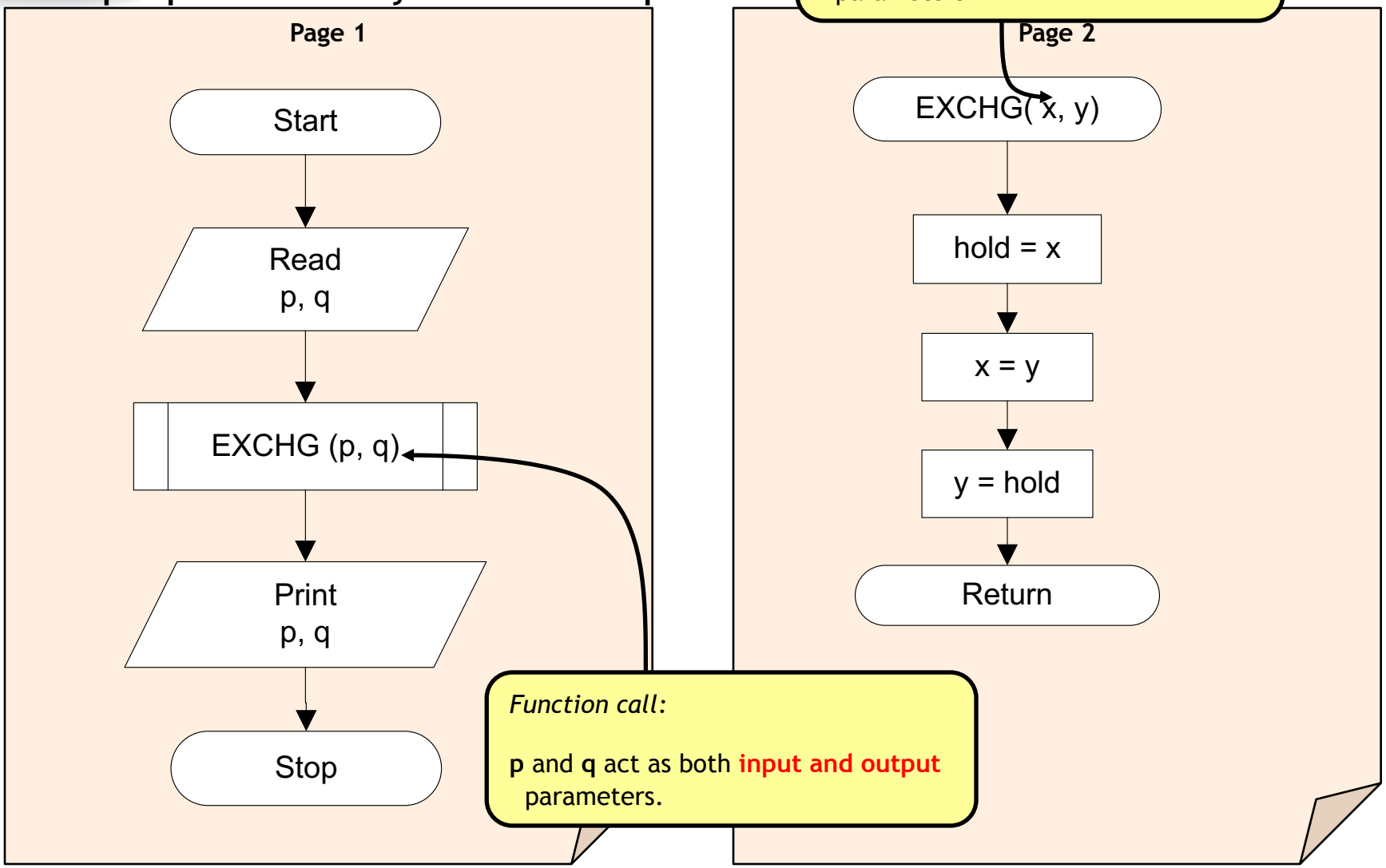
You shouldn't use actual parameters



A function may be called several times



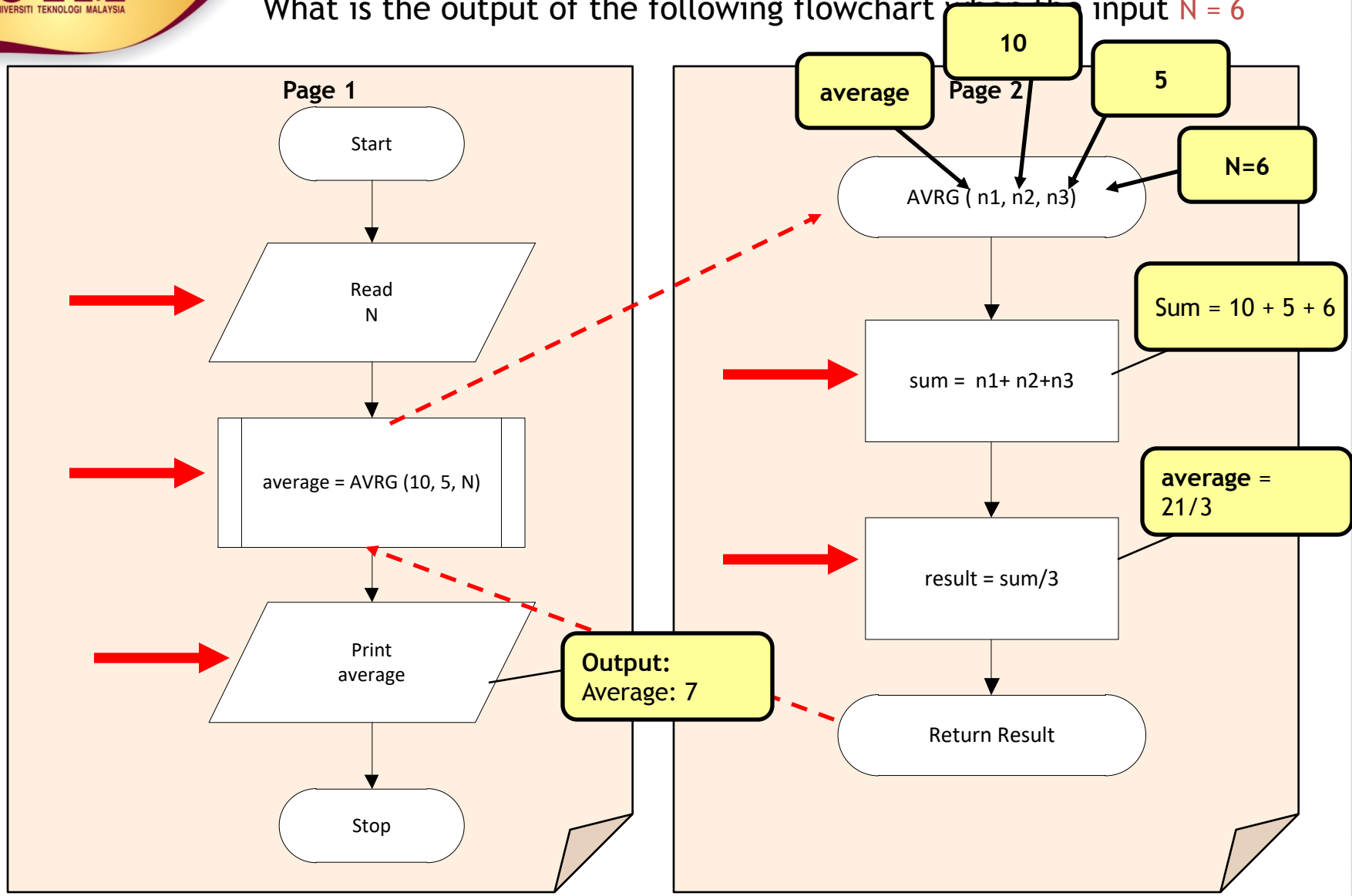
An input parameter may also be an output



Flowchart EXCHG exchanges or swaps the value of x and y each other

Example:

What is the output of the following flowchart with the input N = 6



What you have learnt so far ….

- Analysis:
  - -Understanding the problem

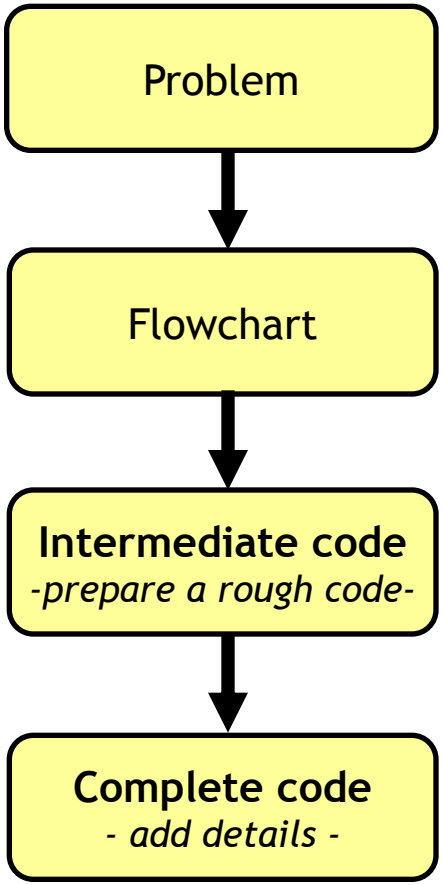
- Design:
  - - Developing algorithm
  - - Understanding how a flowchart works
  - - Constructing flowcharts

The next step is implementation (coding)

Coding is a process of converting flowchart to programming code

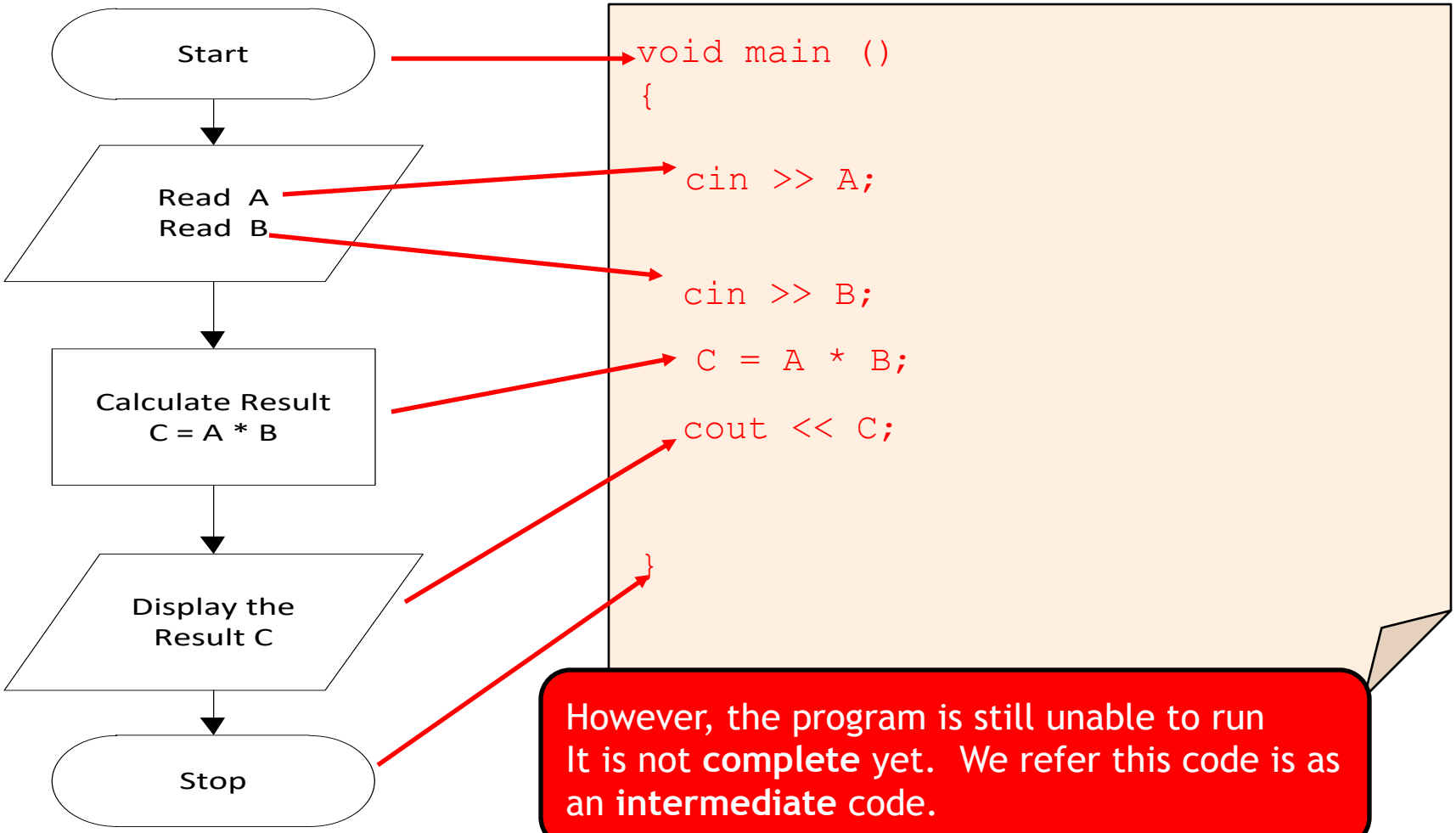
Before you can start doing this, you should learn some basics including the language itself

Writing a C++ Program is a systematic task

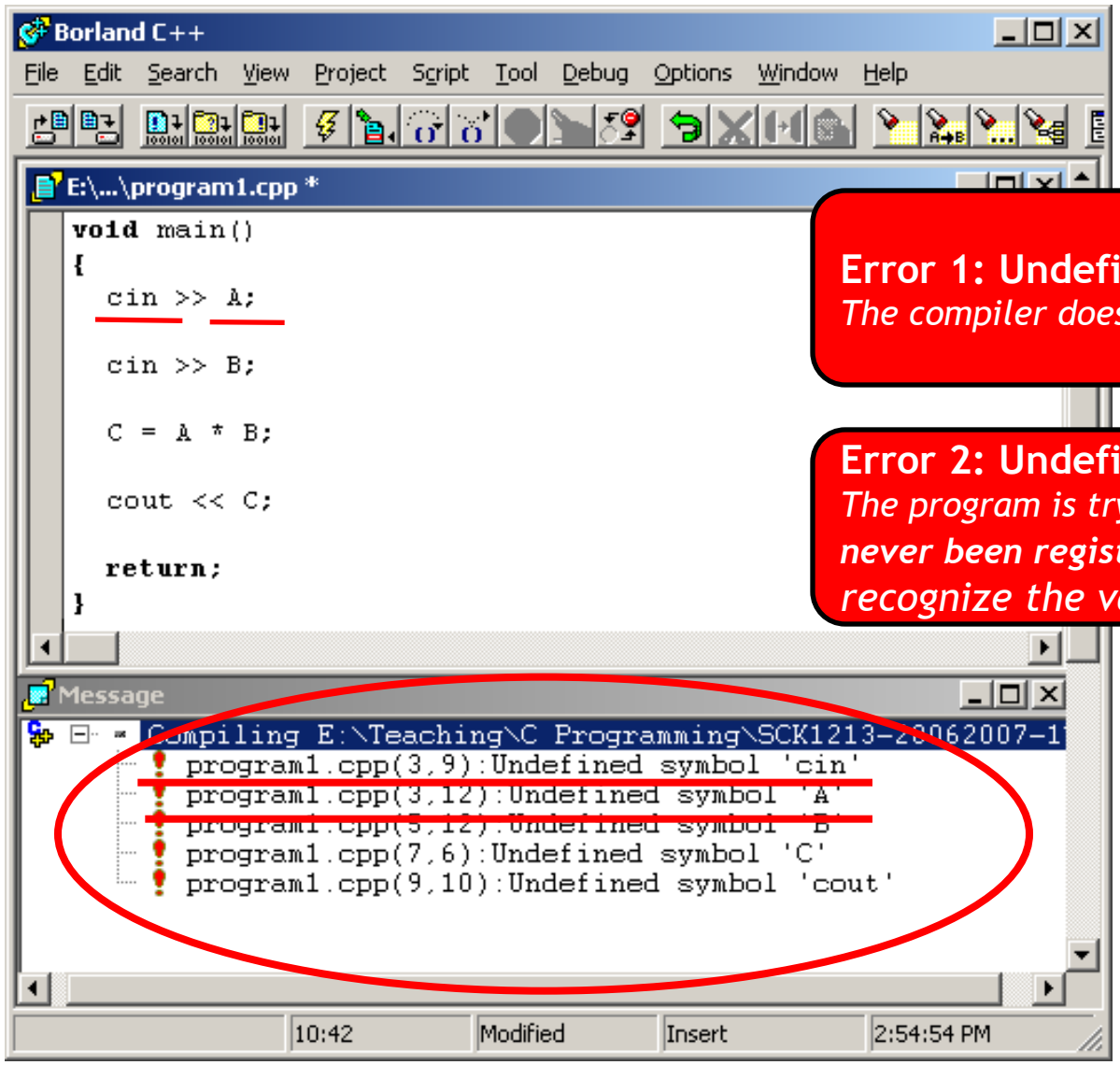


The conversion process is straight forward

Example: multiplying two numbers



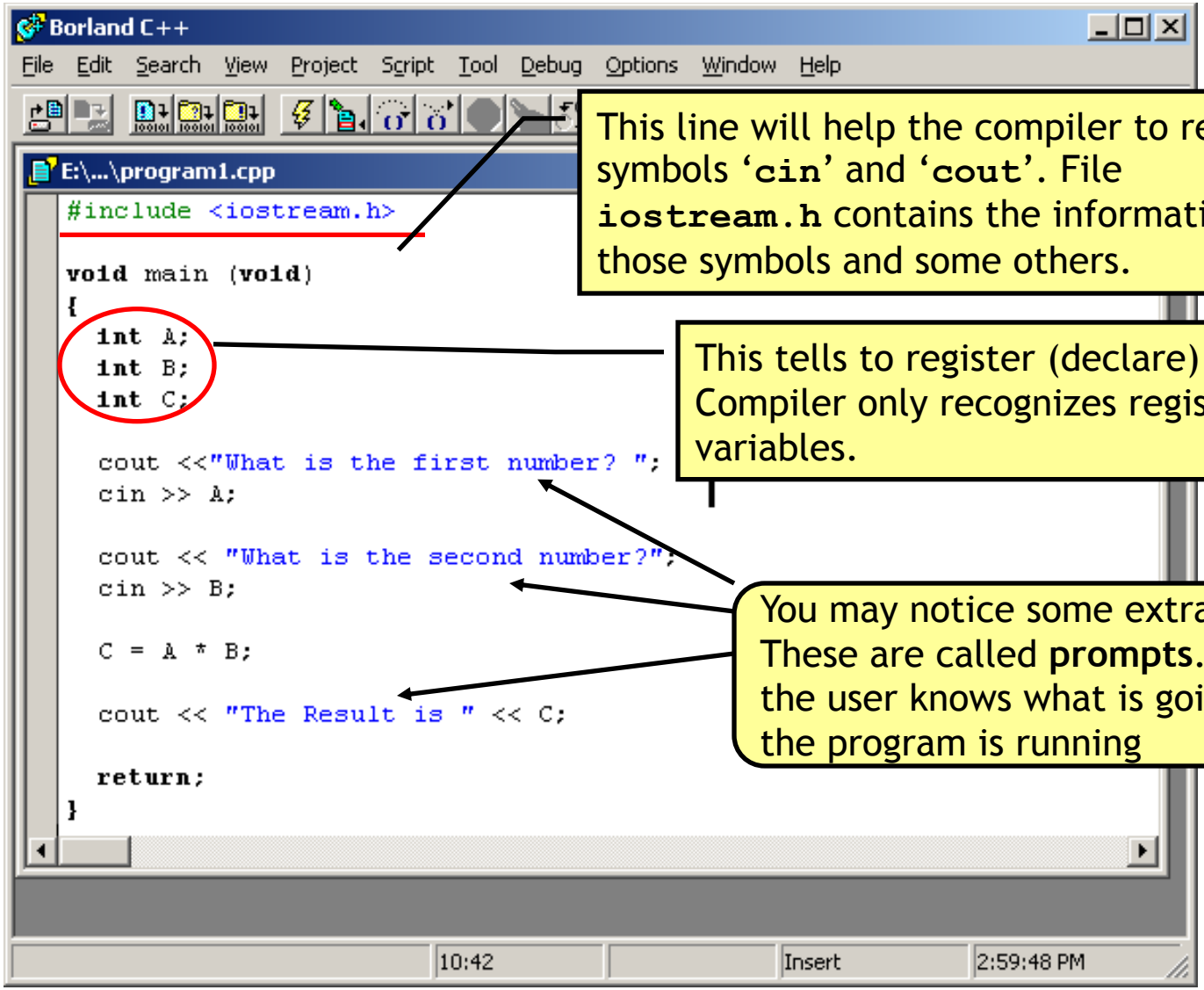
You will get these errors



Error 1: Undefined symbol 'cin'
The compiler doesn't recognize the 'cin'

Error 2: Undefined symbol 'A'
*The program is trying to **use** a variable A but **has never been registered**. Compiler doesn't recognize the variable*

Fixing the errors and completing the program



This line will help the compiler to recognize symbols '**cin**' and '**cout**'. File **iostream.h** contains the information of those symbols and some others.

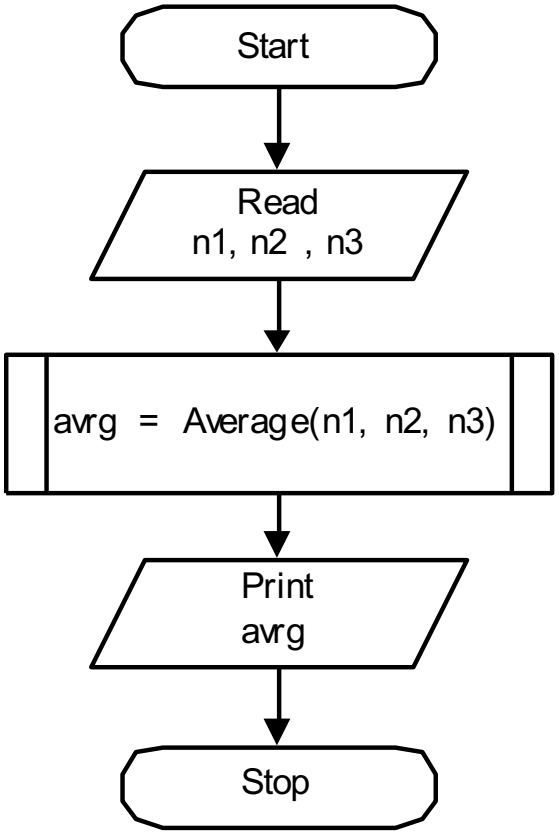
This tells to register (declare) variables. Compiler only recognizes registered variables.

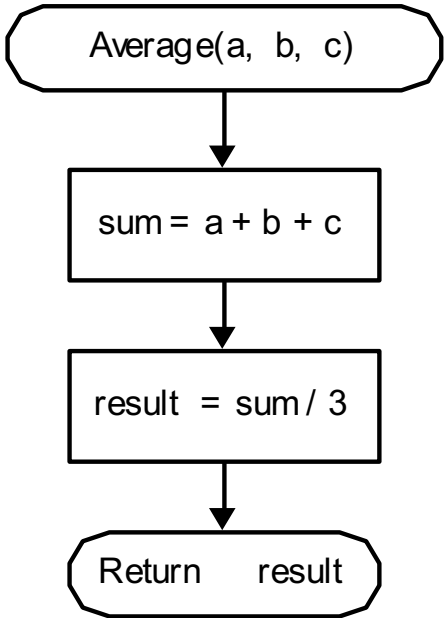
You may notice some extra things. These are called **prompts**. They used to let the user knows what is going on while the program is running

Example

Problem: Finding the average of three numbers

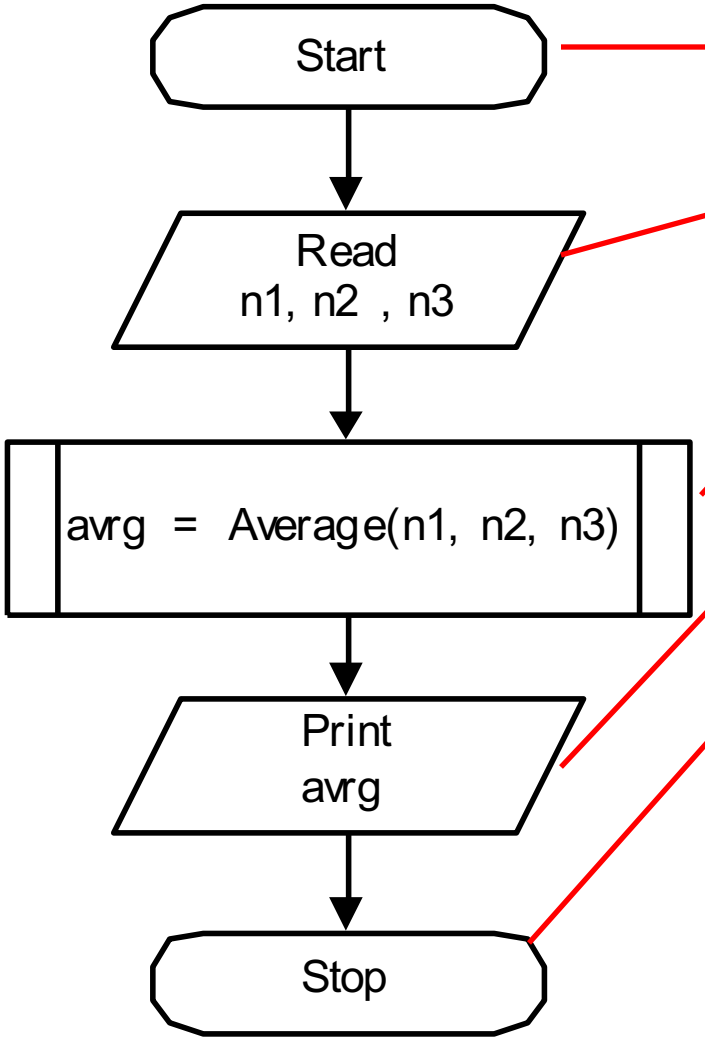
Flowcharts:





Intermediate code of the main flowchart

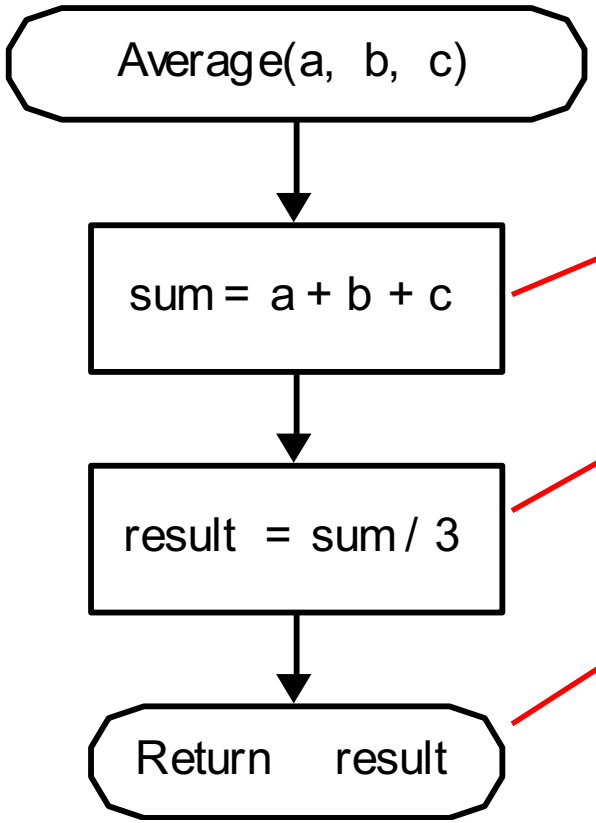
Preparing the rough code




```
void main ()
{
   cin >>n1;
   cin >>n2;
   cin >>n2;

   avrg = Average(n1, n2, n3);

   cout << avrg;

}
```

Intermediate code of the function flowchart

Preparing the rough code




```
Average(a, b, c)
{
  sum = a + b + c;

  result = sum / 3.0;

  return result;
}
```

The complete code

Adding details to the rough code. The details are shown by bold texts

```
#include <iostream>
Using namespace std;

float Average(int a, int b, int c)
{ float sum;

  sum = a + b + c;
  result = sum/3.0;
  return  result;
}

int main ()
{
  int n1;
  int n2;
  int n3;
  float avrg;

  cout <<"Enter three numbers: ";
  cin >> n1;
  cin >> n2;
  cin >> n3;

  avrg = Average(n1,n2,n3);

  cout << "The average is " << avrg;
  return 0;
}
```