Programming Techniques I SCJ1013

Problem Solving

Dr Masitah Ghazali

Software Engineering vs Problem Solving

- Software Engineering - A branch of Computer Science & provides techniques to facilitate the development of computer programs
- Problem Solving - refers to the entire process of taking the statement of a problem and developing a computer program that solves that problem.

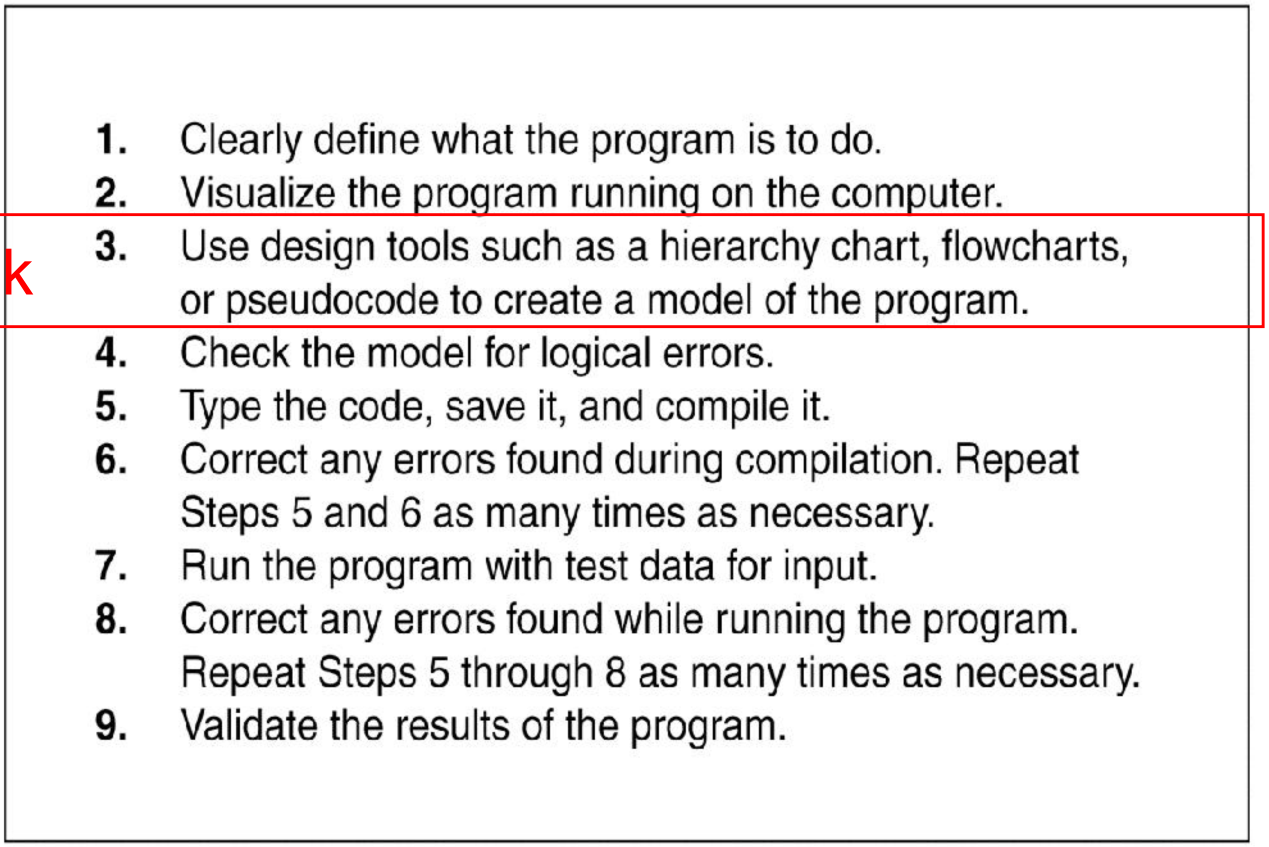
The Programming Process

The Programming Process

- SE concepts require a rigorous and systematic approach to software development called software development life cycle

- Programming process is part of the activities in the software development life cycle

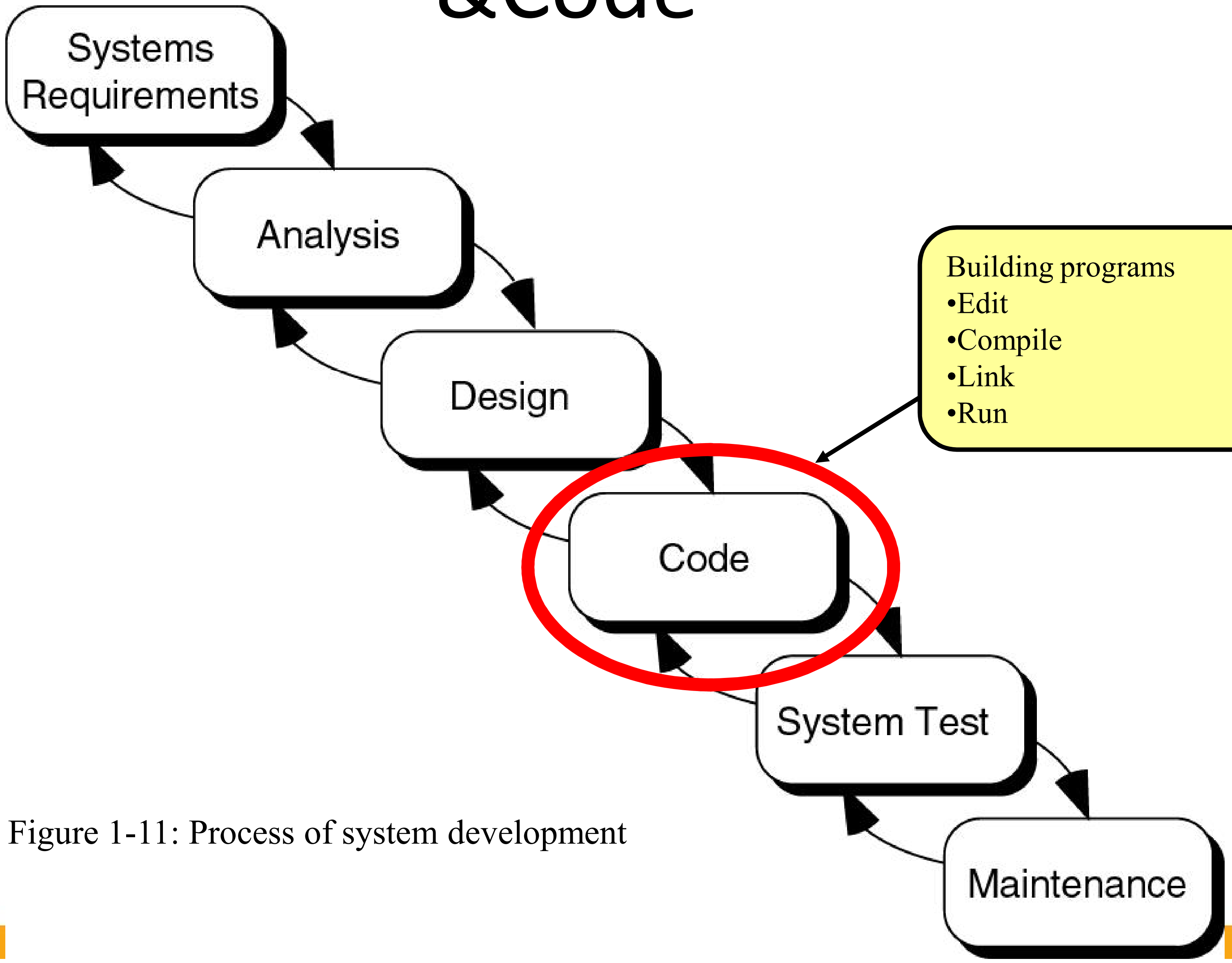
The Programming Process



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.

This week

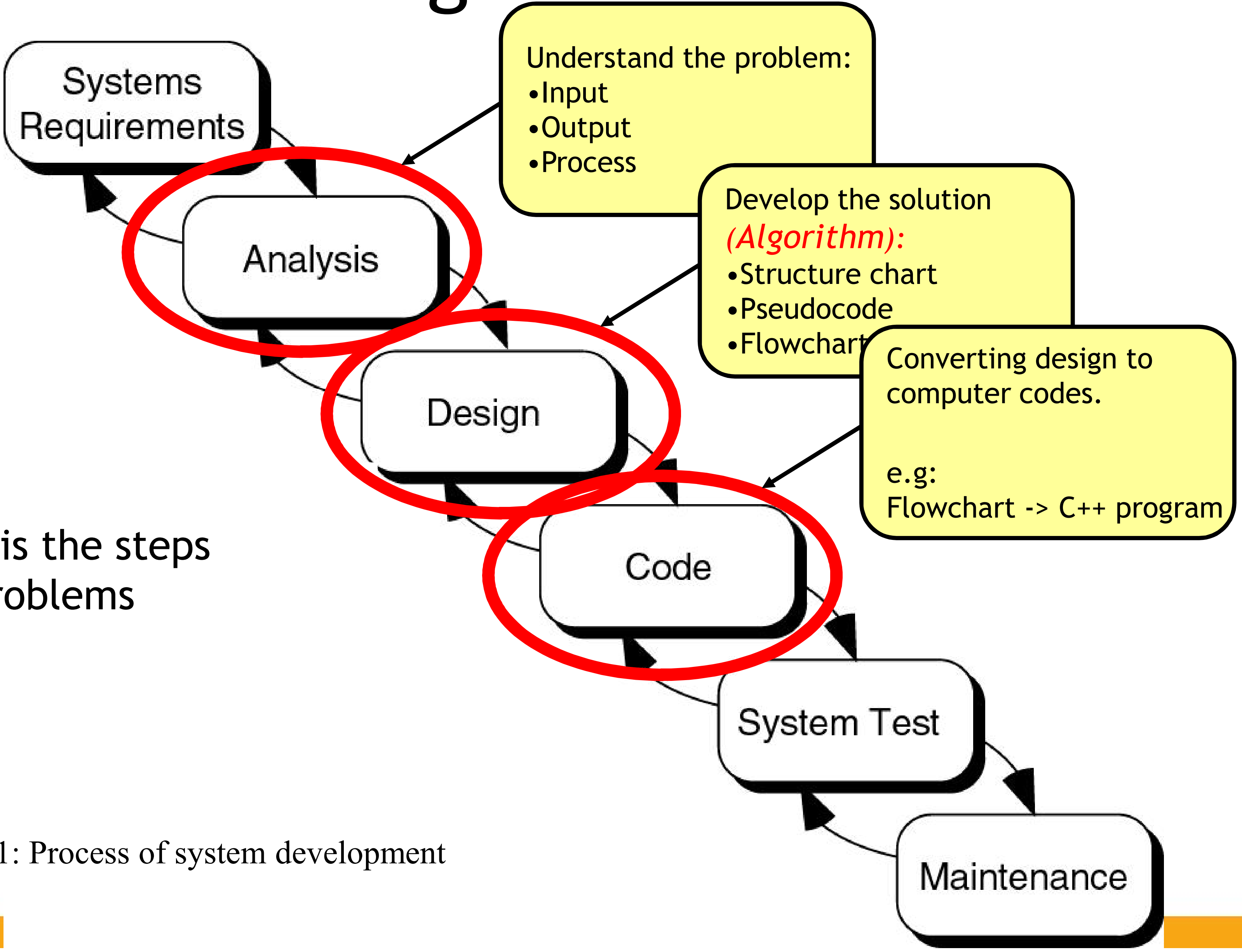
Software Development Life Cycle &Code



Building programs
- •Edit
- •Compile
- •Link
- •Run

Figure 1-11: Process of system development

Software Development Life Cycle & Algorithm



Understand the problem:
- Input
- Output
- Process

Develop the solution *(Algorithm)*:
- Structure chart
- Pseudocode
- Flowchart

Converting design to computer codes.

e.g:
Flowchart -> C++ program

Algorithm is the steps to solve problems

Figure 1-11: Process of system development

Software Development Life Cycle

- Problem Analysis
  - Identify data objects
  - Determine Input / Output data
  - Constraints on the problem
- Design
  - Decompose into smaller problems
  - Top-down design
    - Structured Chart
  - Develop Algorithm
    - Pseudocode
    - Flowchart

Software Development Life Cycle

- Implementation/coding/programming
  - Converting the algorithm into programming language
- Testing
  - Verify the program meets requirements
  - System and Unit test
- Maintenance
  - All programs undergo change over time

Software Development Life Cycle

- Case Study: Converting Miles to Kilometres

Input, Processing, and Output

Input, Processing, and Output

Three steps that a program typically performs:

1) **Gather input data:**
   - from keyboard
   - from files on disk drives
2) **Process the input data**
3) **Display the results as output:**
   - send it to the screen
   - write to a file

Exercise Week2_1

- Do Lab 2, Exercise 3, No. 1-4 in pg. 27-28.
- Identify the following information:

1. Input data
2. Process the input data
3. Output data

Representation of Algorithms

Problem solving methods in this Class

- 3 problem solving methods will be discussed in this class are:

1. Develop Algorithms
   - ❖ Flowchart
   - ❖ Pseudo code
2. 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:
  - i. Must have input
  - ii. Must produce output
  - iii. Unambiguous
  - iv. Generality
  - v. Correctness
  - vi. Finiteness
  - vii. Efficiency

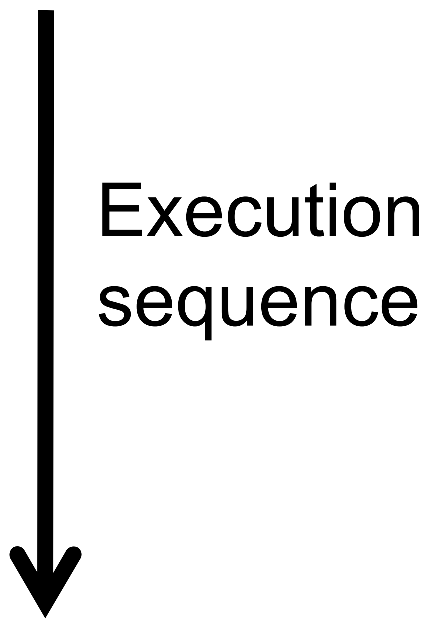
Pseudo code

- 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.

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
```



Exercise Week2_2

- Refer to Lab2, Exercise 1, No. 2.1.1-2.1.3 in pg. 17-19

1. Desk Check/Trace the algorithm
2. Complete the exercise

Flowchart

- Flowchart – a graph of geometrical shapes that are connected by lines.

- 2 important element in flow chart:

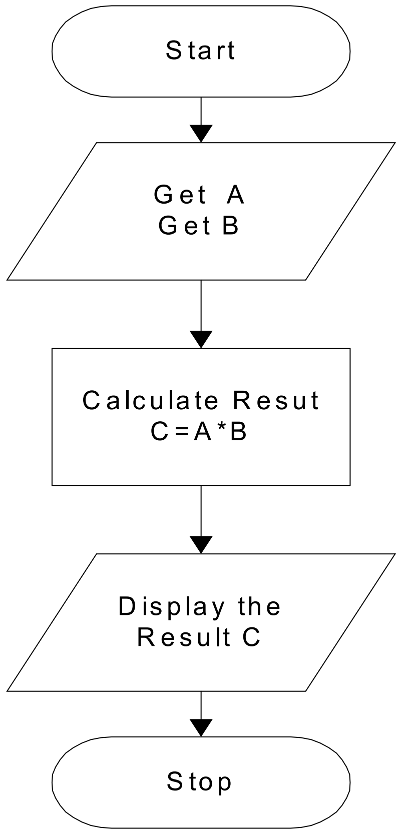
1. geometrical shapes – represent type of statements in the algorithm
2. Flow line – show the order in which the statements of an algorithm are executed.

Flowchart

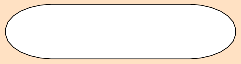
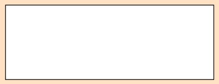
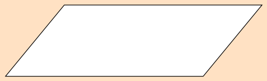
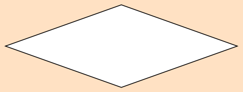
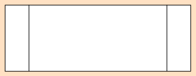
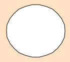
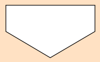
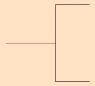
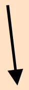
- Flowchart - Represents an algorithm in graphical symbols

```
Example: Algorithm
      for multiplying
          two numbers
```

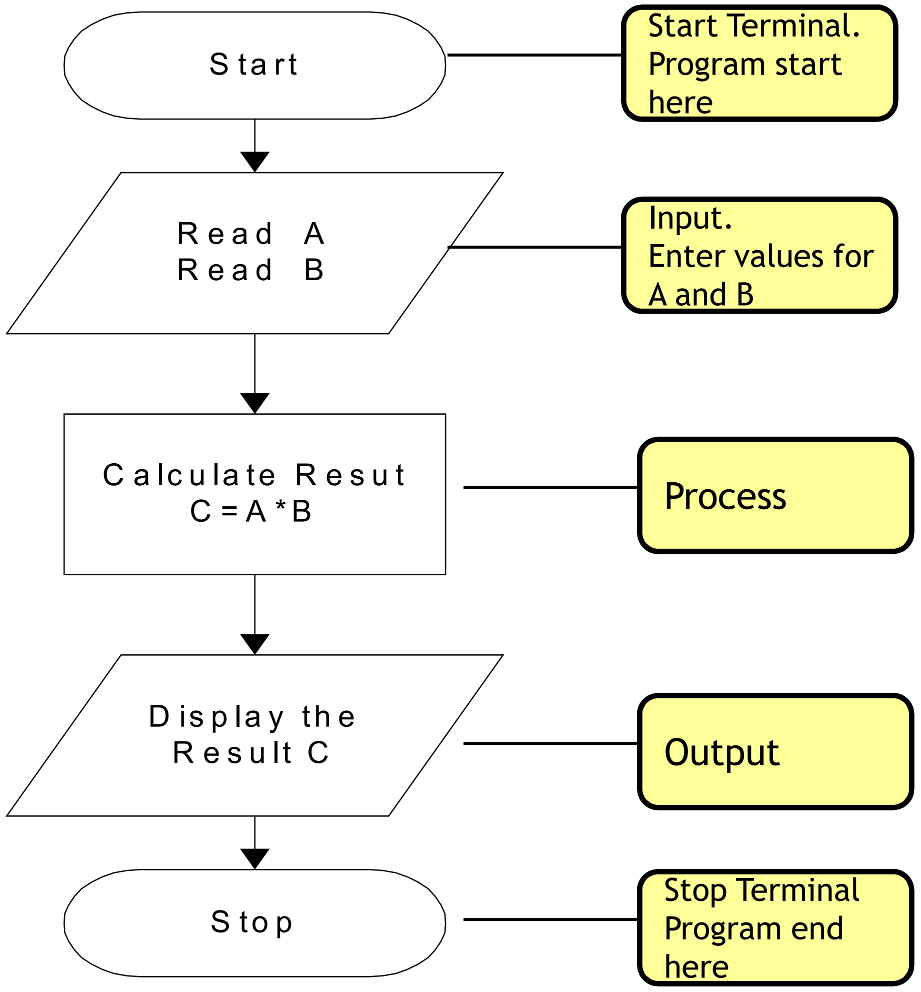
- Desk Check/Trace the algorithm!!!



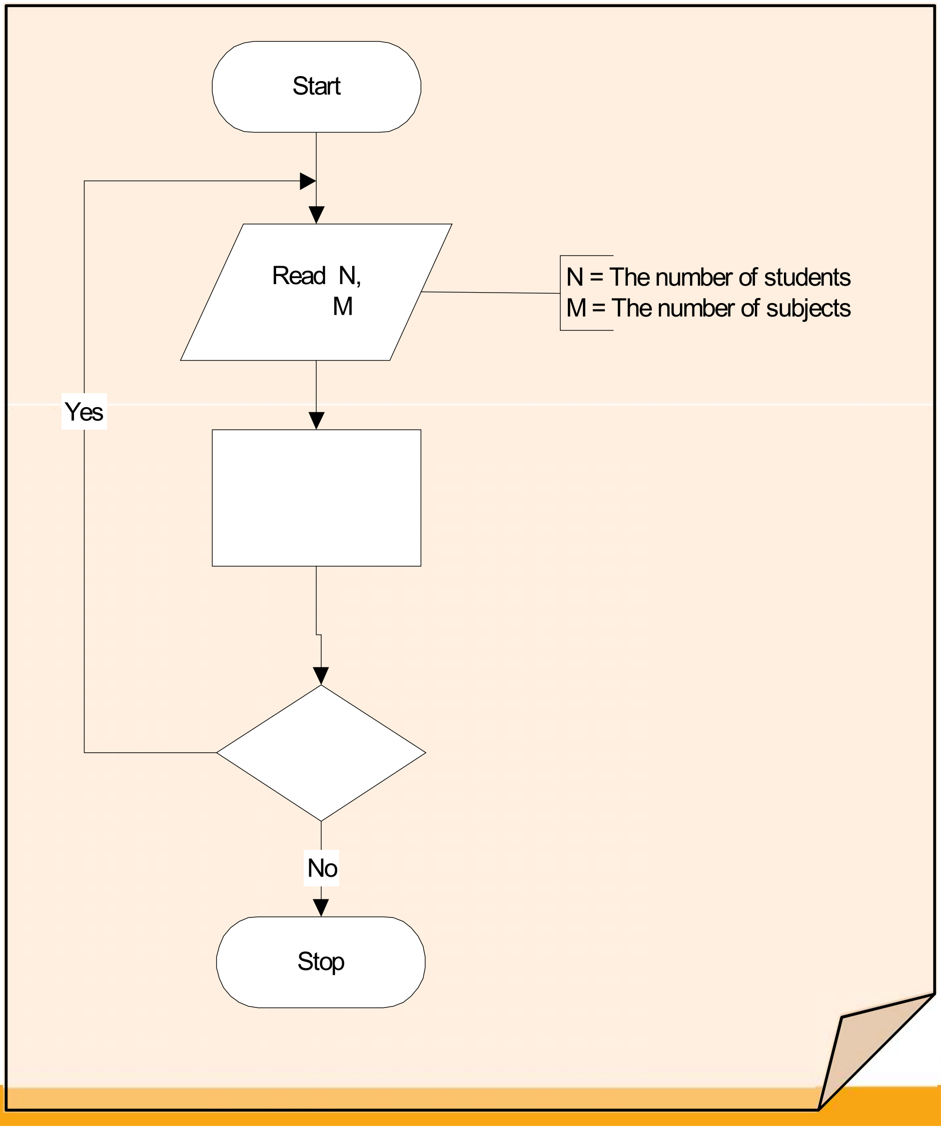
Flowchart Symbol

	**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.

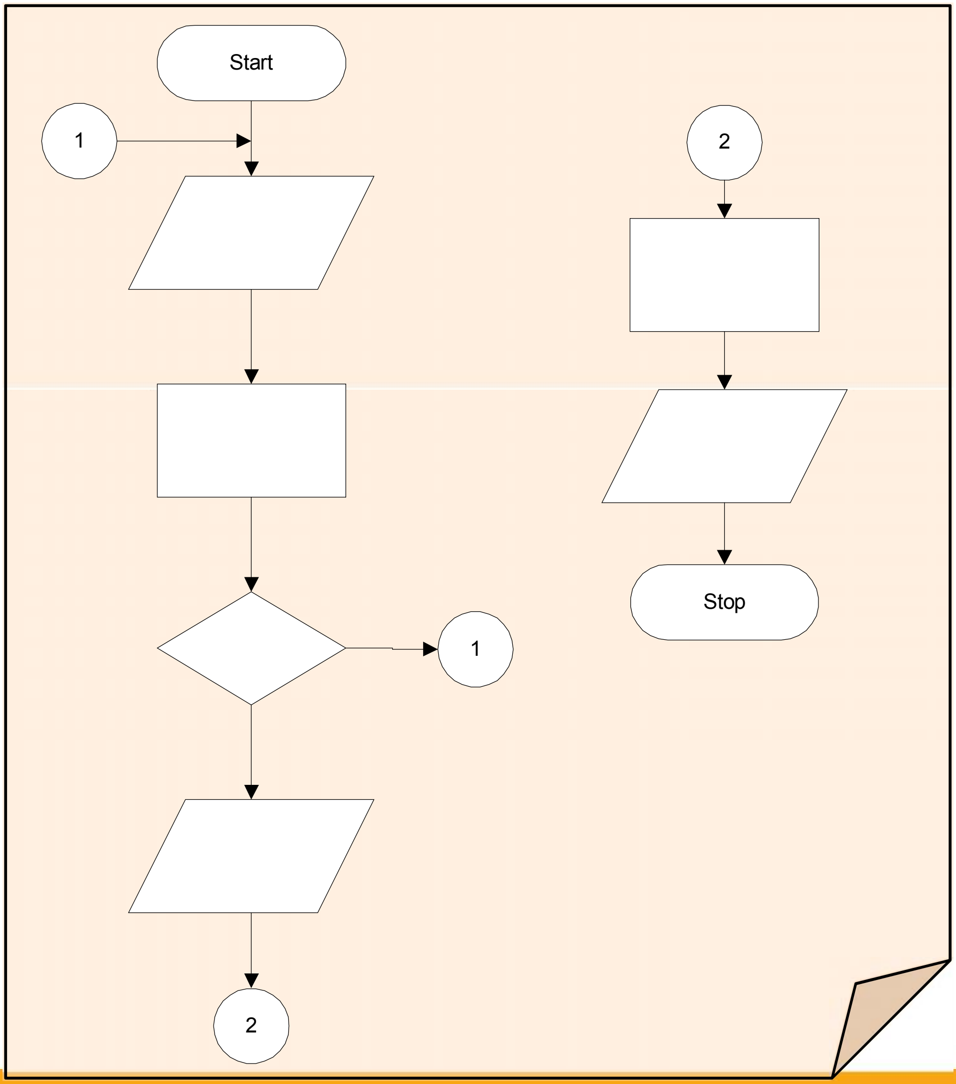
The Flowchart Explanation


Start
Start Terminal. Program start here
Read A
Read B
Input. Enter values for A and B
Calculate Resut
C = A * B
Process
Display the Result C
Output
Stop
Stop Terminal Program end here

Example: Use of comments/description



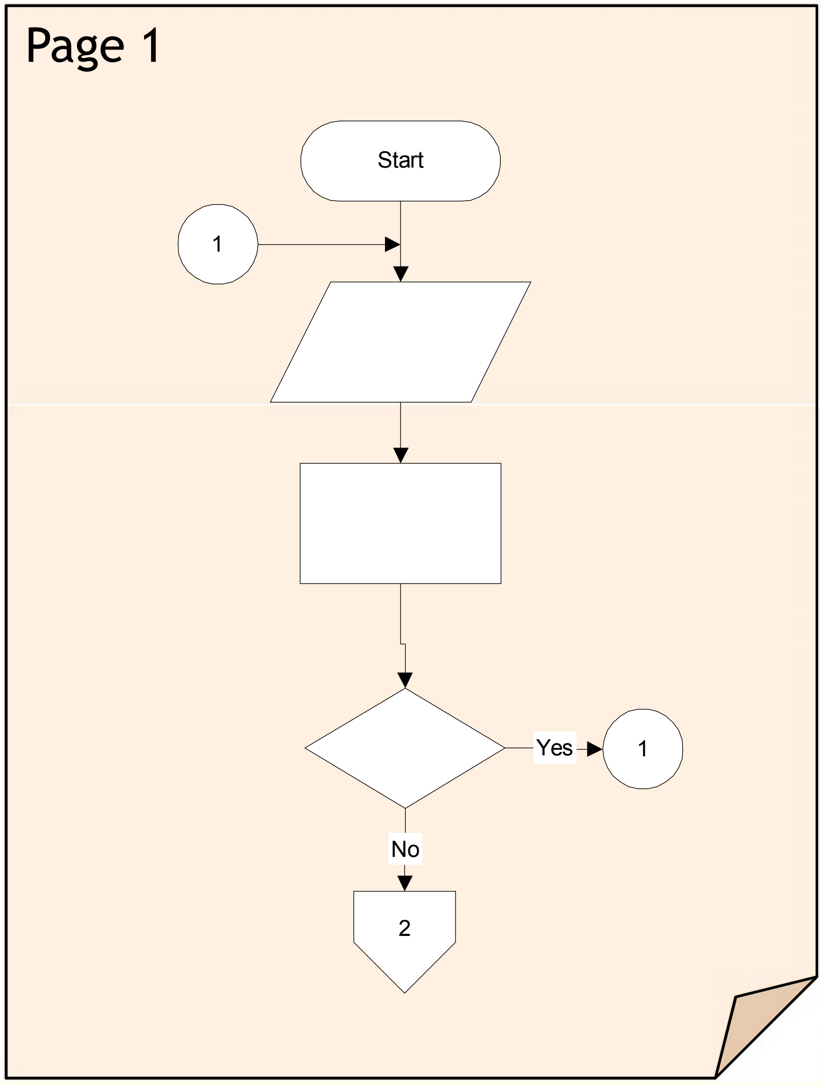
Example: Use of connectors on the same page.

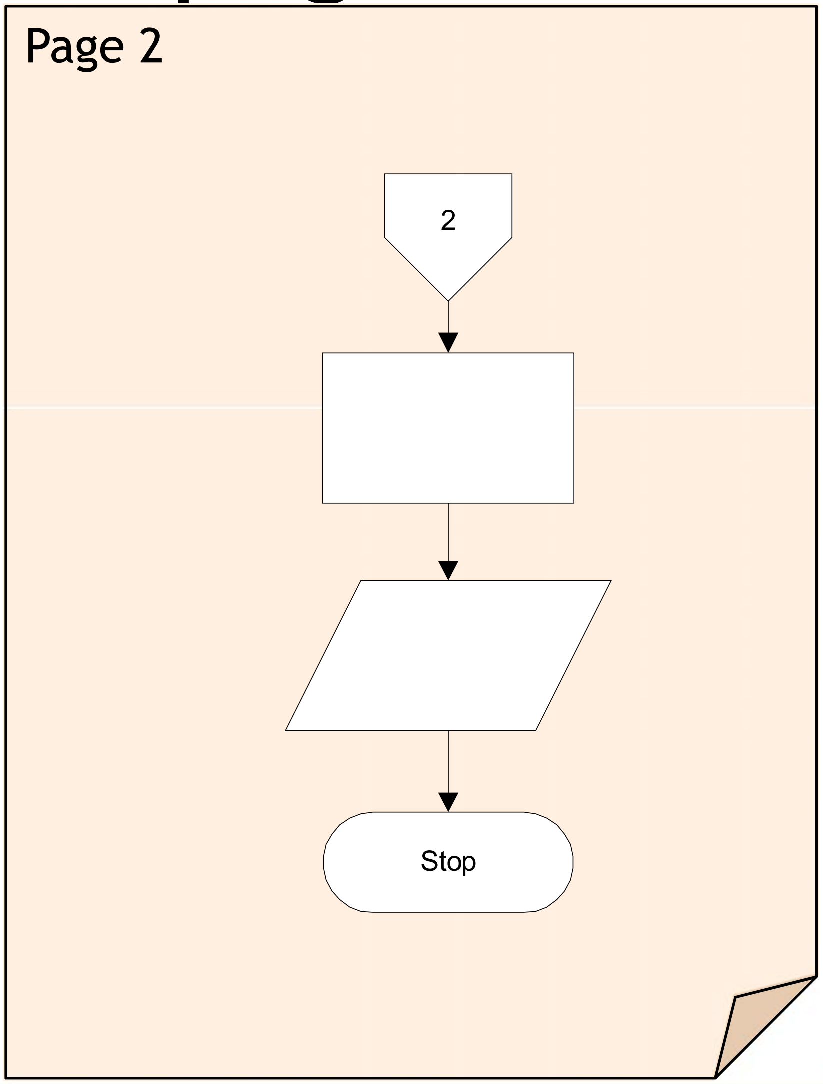


1- connection on the same flowchart portion

2- connection on the different flowchart portion

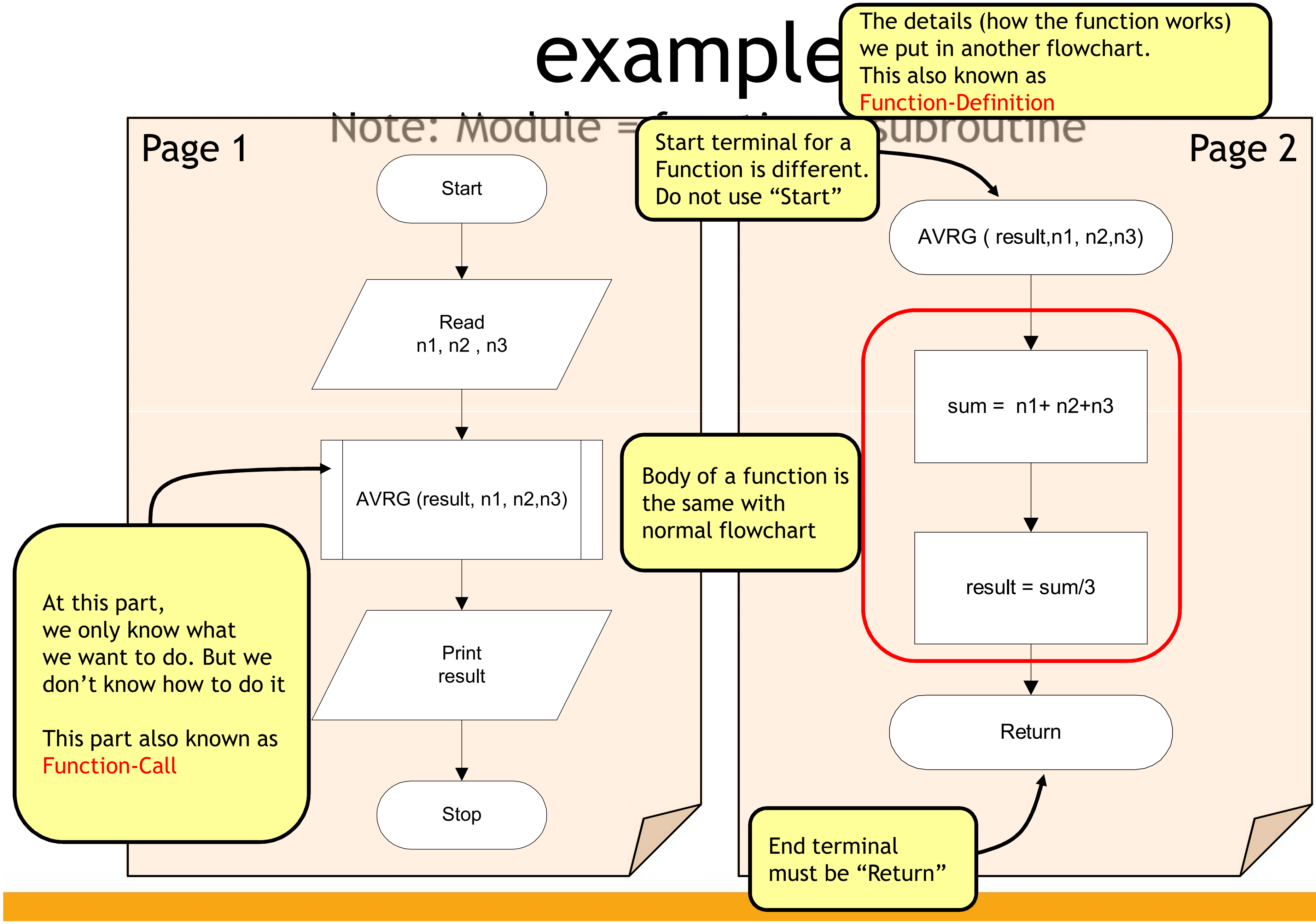
Example: Use of connectors on the different page.





Example: Function-call example

Note: Module = function = subroutine



Page 1
Start
Read n1, n2 , n3
AVRG (result, n1, n2,n3)
Print result
Stop
Page 2
AVRG (result,n1, n2,n3)
sum = n1+ n2+n3
result = sum/3
Return
The details (how the function works) we put in another flowchart. This also known as Function-Definition
Start terminal for a Function is different. Do not use "Start"
Body of a function is the same with normal flowchart
At this part, we only know what we want to do. But we don't know how to do it
This part also known as Function-Call
End terminal must be "Return"

Exercise Week2_3

- Refer to Lab 2, Exercise 1, No. 4-5 in pg. 24-25.
- Complete the exercise

Control Structure of Algorithms

Control Structures

- Describe the flow of execution

- Basic types of control structure:
  1. Sequential
  2. Selection
  3. Repetition

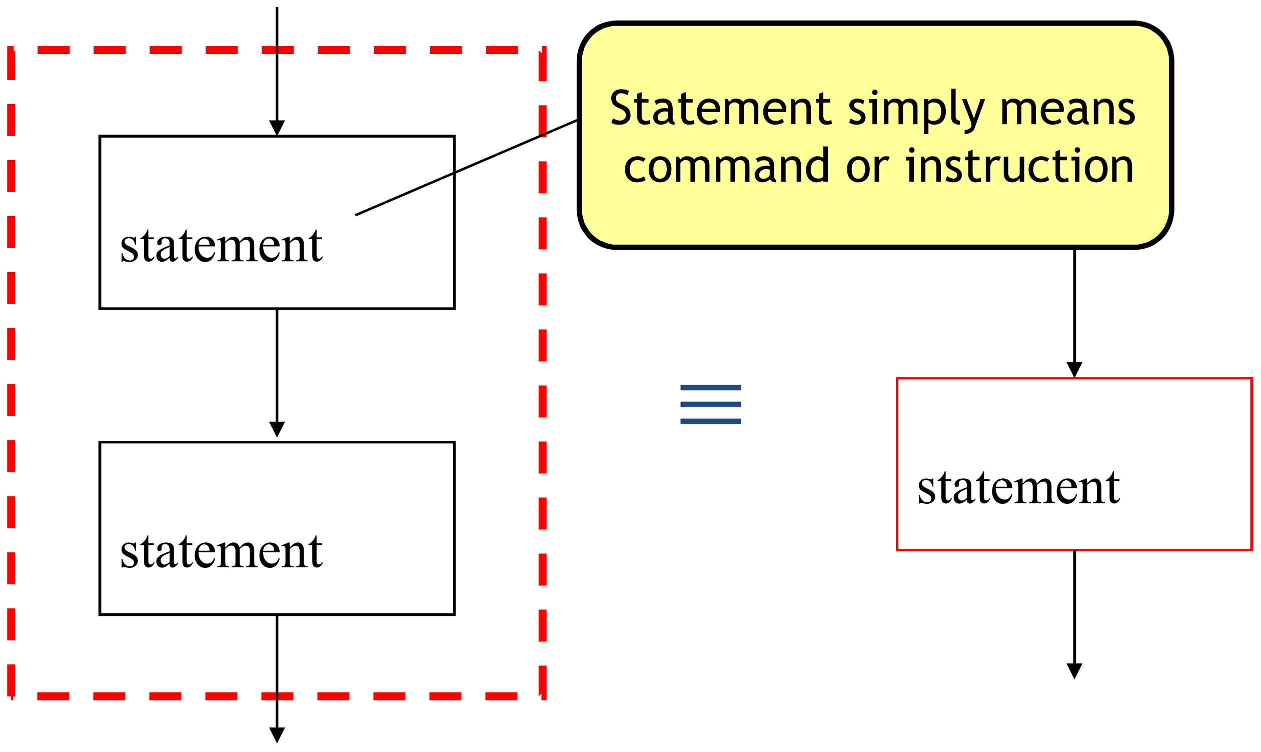
Sequential Structure

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

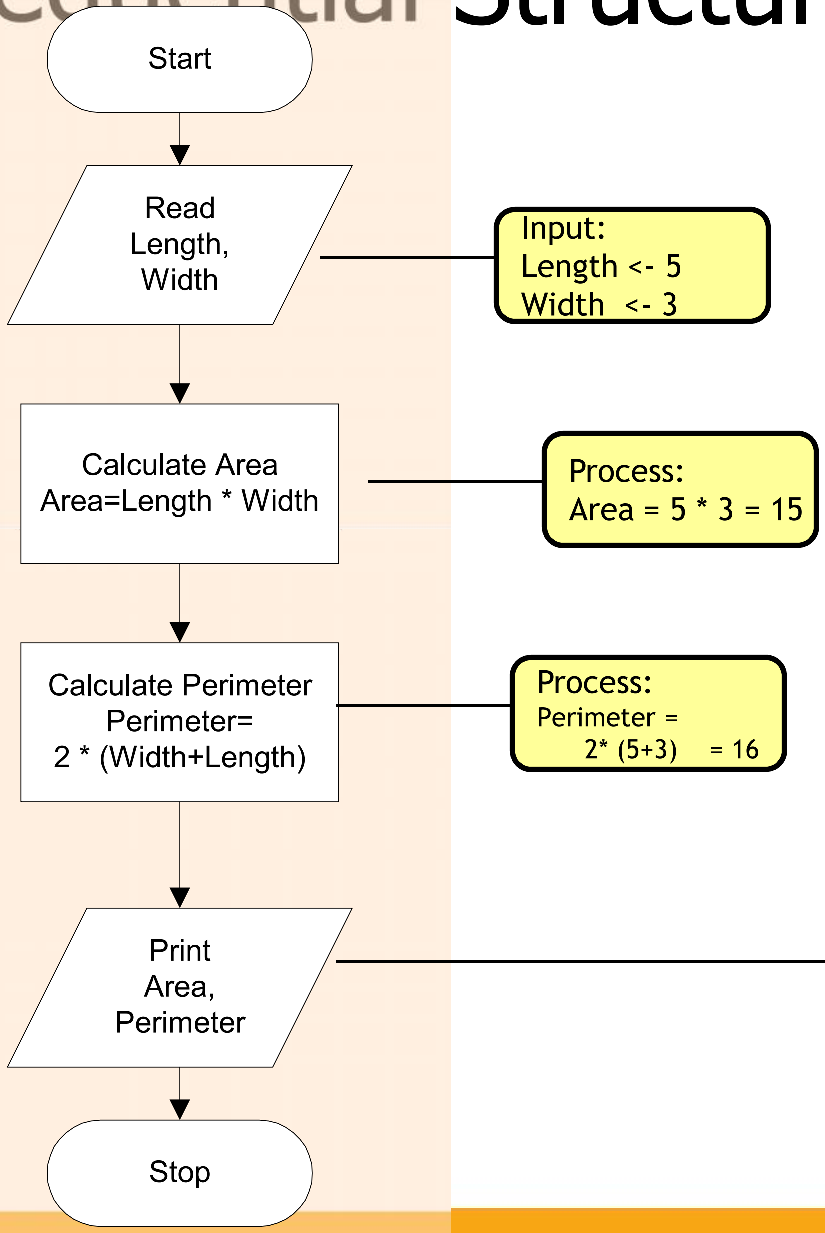
```
1.   Start
2.   Statement_1
3.   Statement_2
4.   Statement_3
n.   Statement_n+1
N+1.End
```

Sequential Structure – flow chart

- Multiple statements considered as one statement



Sequential Structure - trace


Start
Read Length, Width
Calculate Area Area=Length * Width
Calculate Perimeter Perimeter= 2 * (Width+Length)
Print Area, Perimeter
Stop
Input:
Length <- 5
Width <- 3
Process:
Area = 5 * 3 = 15
Process:
Perimeter = 2* (5+3) = 16

Output

```
Area: 15
Perimeter: 16
```

Sequential Structure – case study

- Case Study: Calculate the Payment

Exercise Week2_4

- Refer to Lab 2, Exercise 1 No. 1 & 2 in pg. 20.
- Discuss
- Convert to flow chart

- Refer to Lab 2, Exercise 1, No. 3 in pg. 21.
- Complete the exercise

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

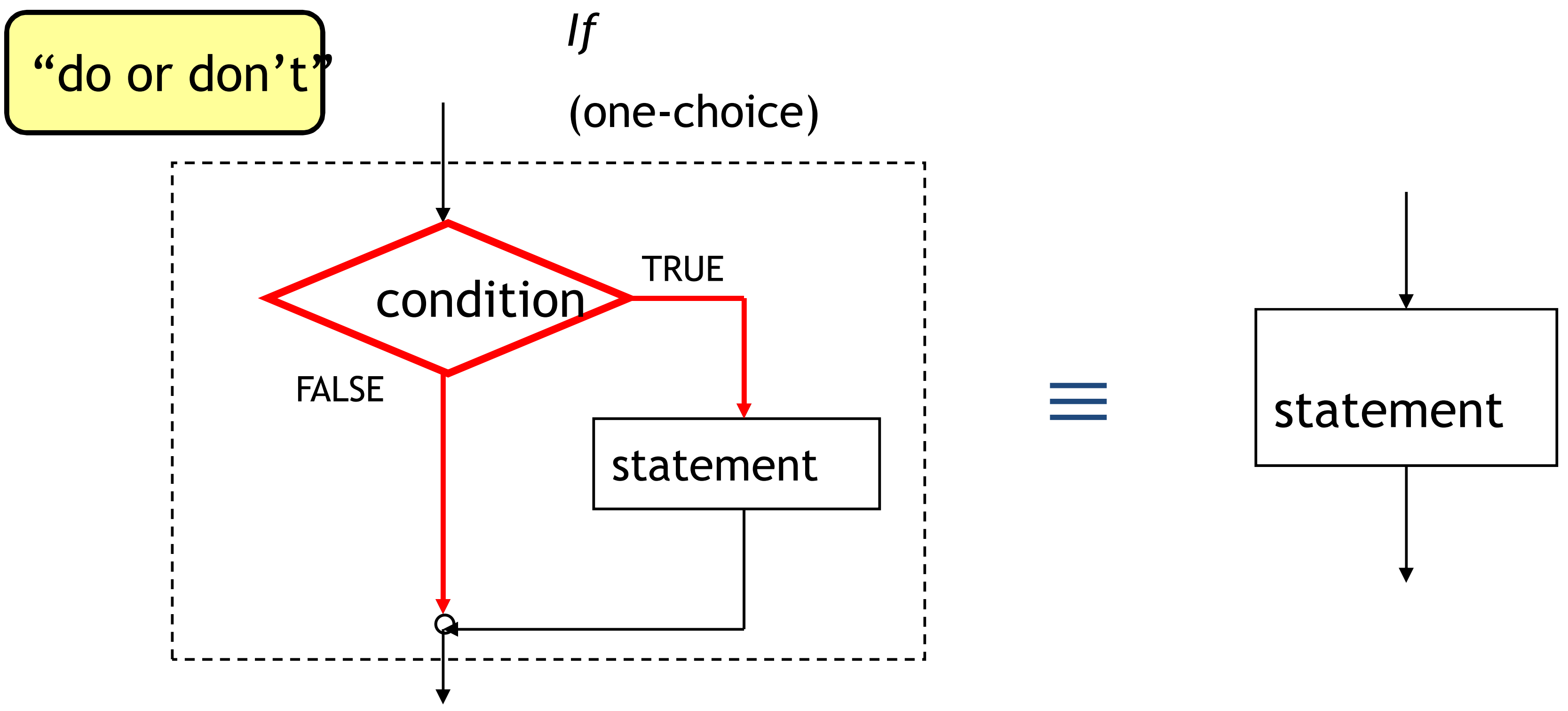
- Problem 1: Determine whether profit, return capital or loss.
- Problem 2: Determine whether a number is even or odd.
- Problem 3: Determine whether the marks is less than 60%. If it is less than 60, then print “fail”, otherwise print “pass”.
- Problem 4: Determine whether the speed limit exceeds 110 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..*)



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

Selection Structure (*cont..*)

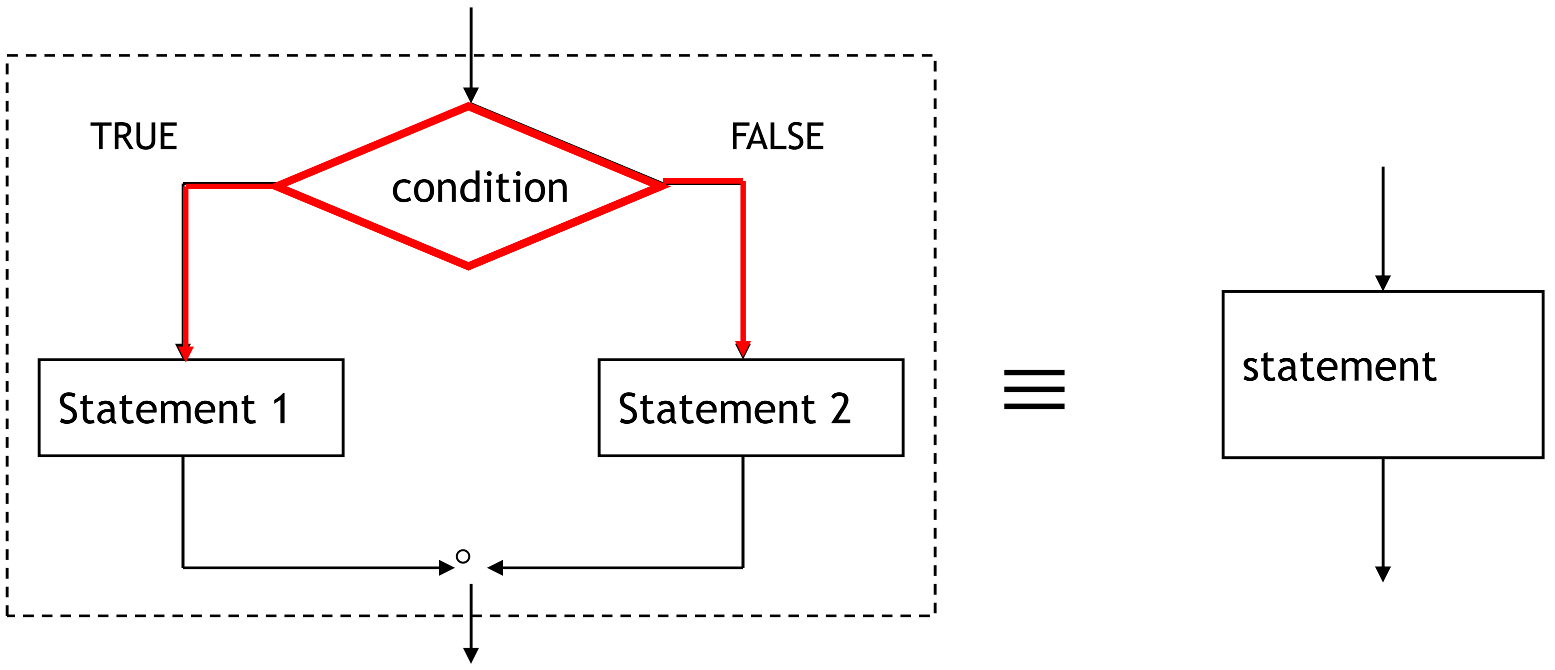
- Pseudo code - requires the use of the keywords `if` and `else`.

```
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

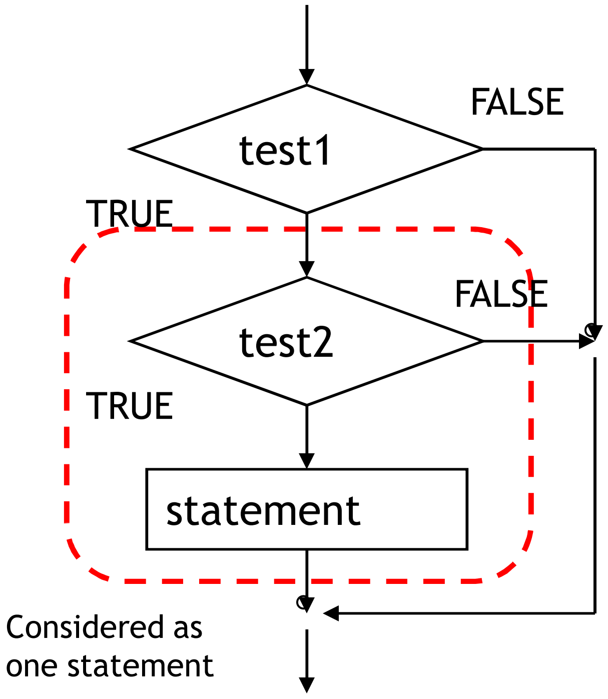
Selection Structure (*cont..*)

- Pseudo code – nested if.

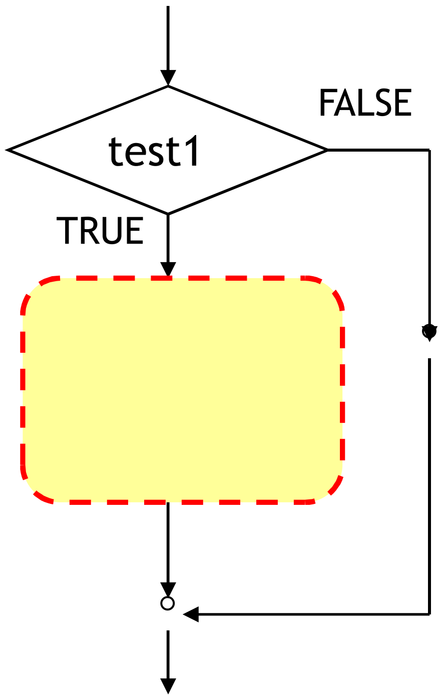
```
Algorithm: nested if
:
n.if condition
     :
     n.m if condition
          n.m.1 statement
          :
n+1. end_if
:
```

Selection Structure (*cont..*)

Nested if
(if within if)



≡



it is an “one-choice” if

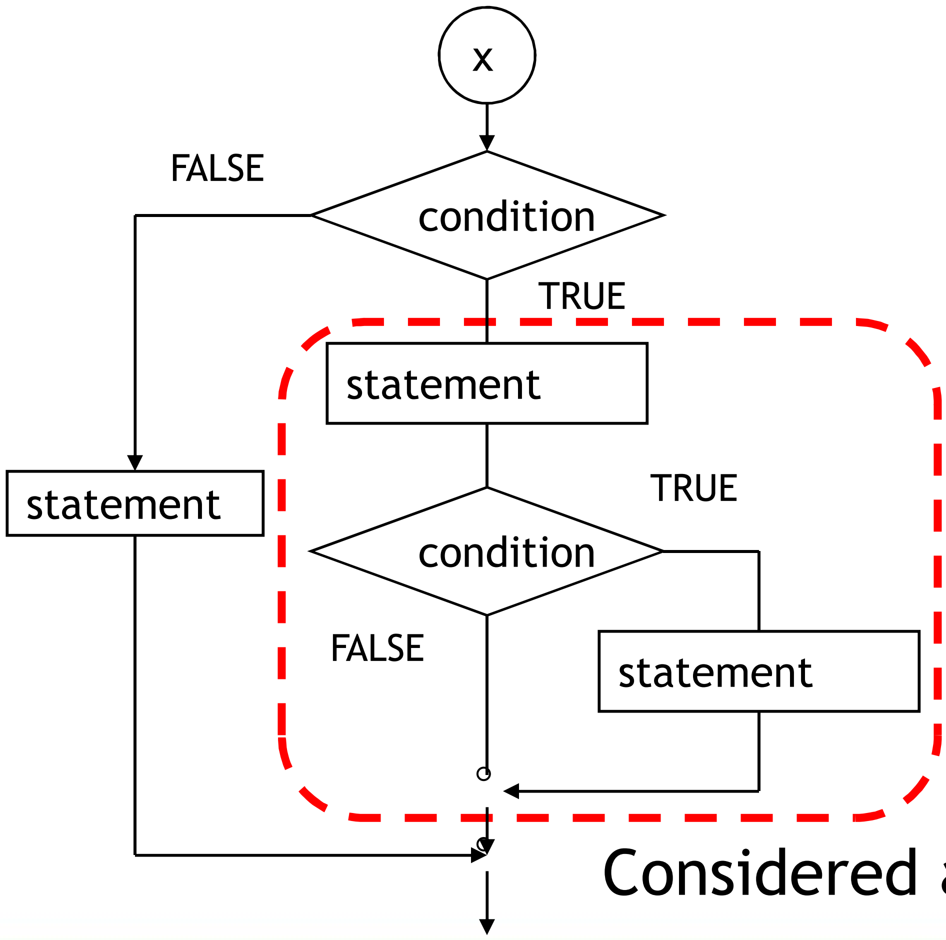
Selection Structure (*cont*..)

- 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

Relational Operators

- 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

Logical Operators - examples

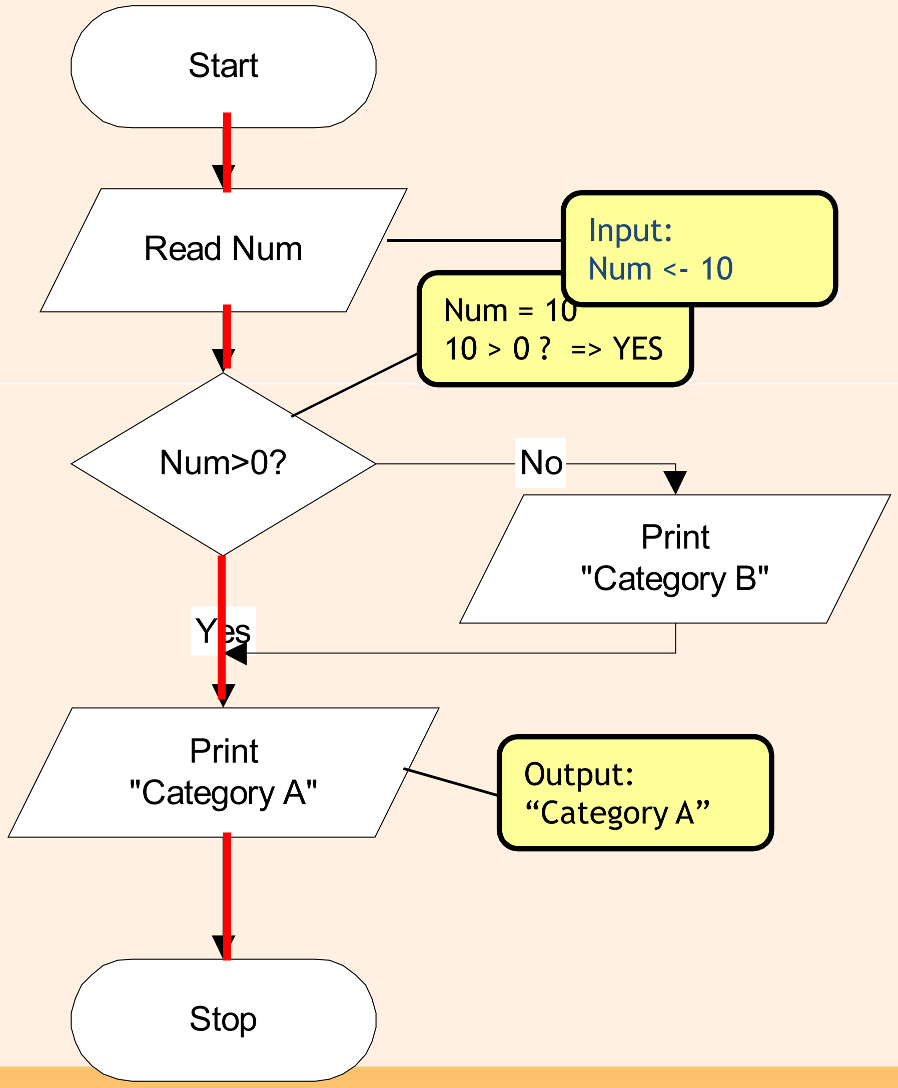
```
int x = 12, y = 5, z = -4;
```

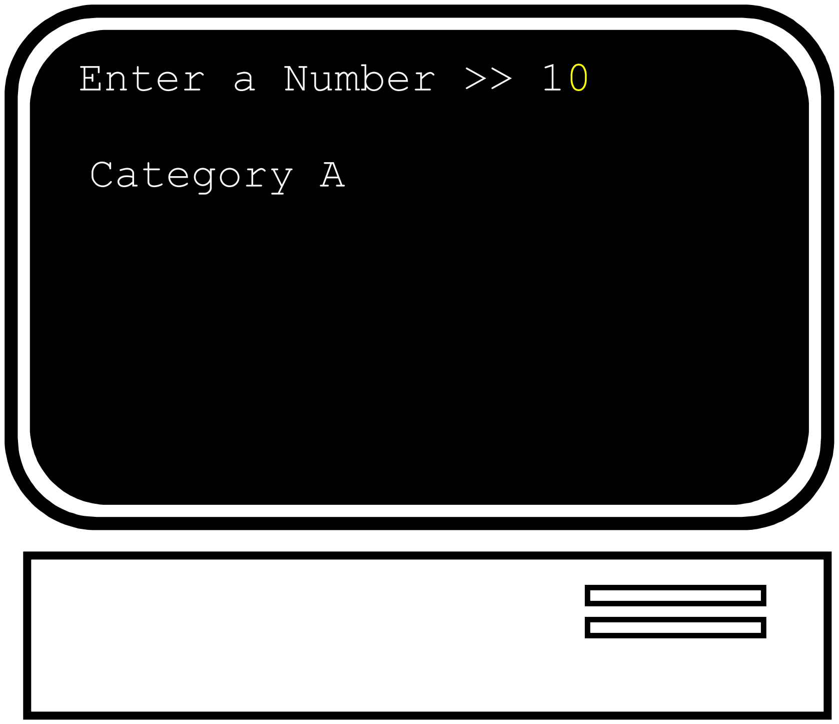
`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

Exercise Week2_5

- Refer to Lab 3, Exercise 1, No. 2 in pg. 34.
- Draw flow chart symbol for the given conditions

Selection Structure - trace

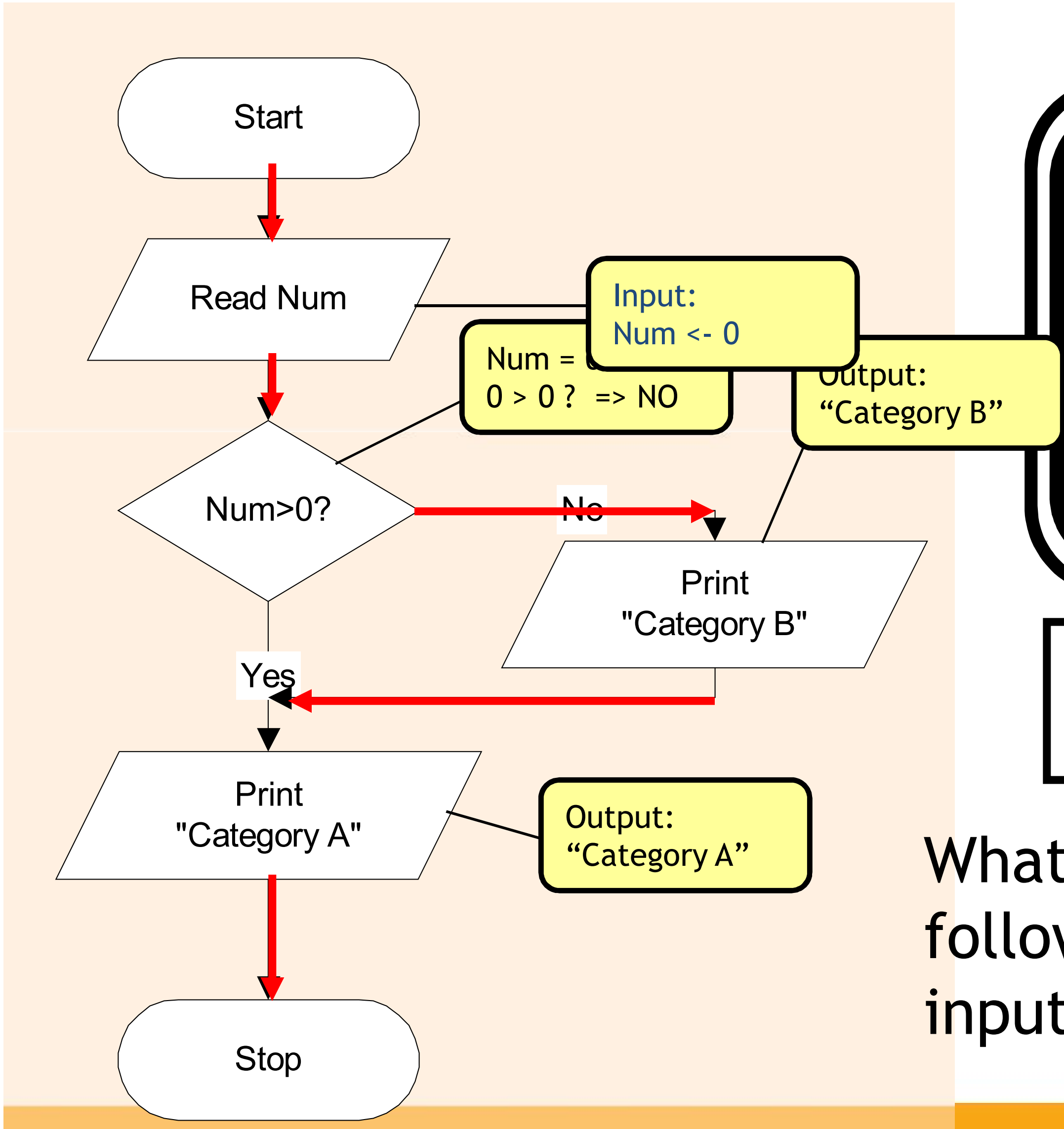


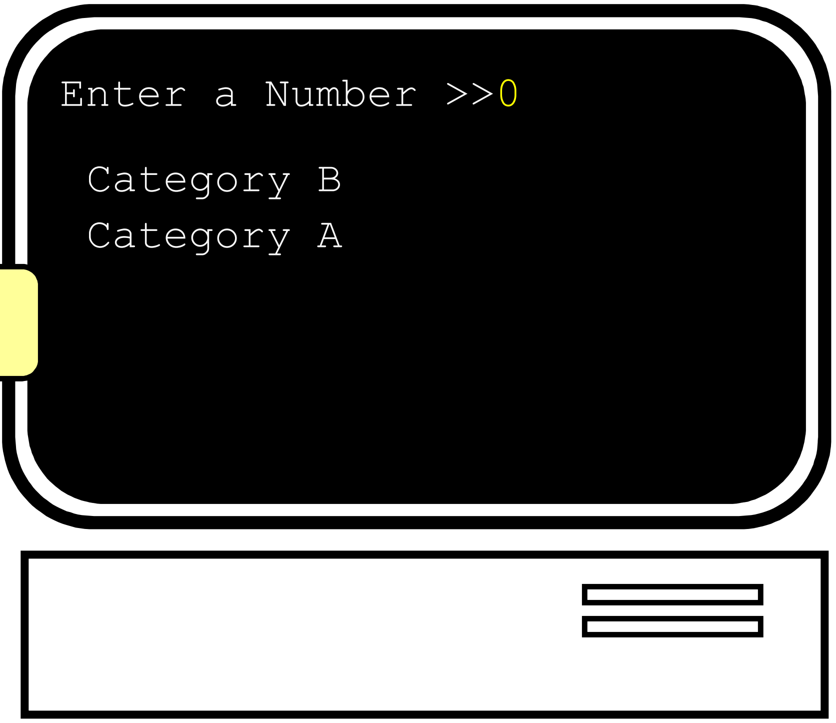


```
Enter a Number >> 10

 Category A
```

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

Selection Structure – trace (*cont..*)





```
Enter a Number >>0

 Category B
 Category A
```

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

Exercise Week2_6

- Refer to Lab 3, Exercise 2, No. 5 in pg. 40.
- Complete the exercise

Selection Structure – case study

- Case Study: Determine whether profit, return capital or loss

Exercise Week2_7

- Refer to Lab 3, Exercise 3, No. 5(i) in pg. 42.
- Complete the exercise

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:
  1. An expression that is tested for a true/false,
  2. A statement or block that is repeated as long as the expression is true
- 2 styles of repetition or loop
  1. Pre-test loop
  2. Post test loop

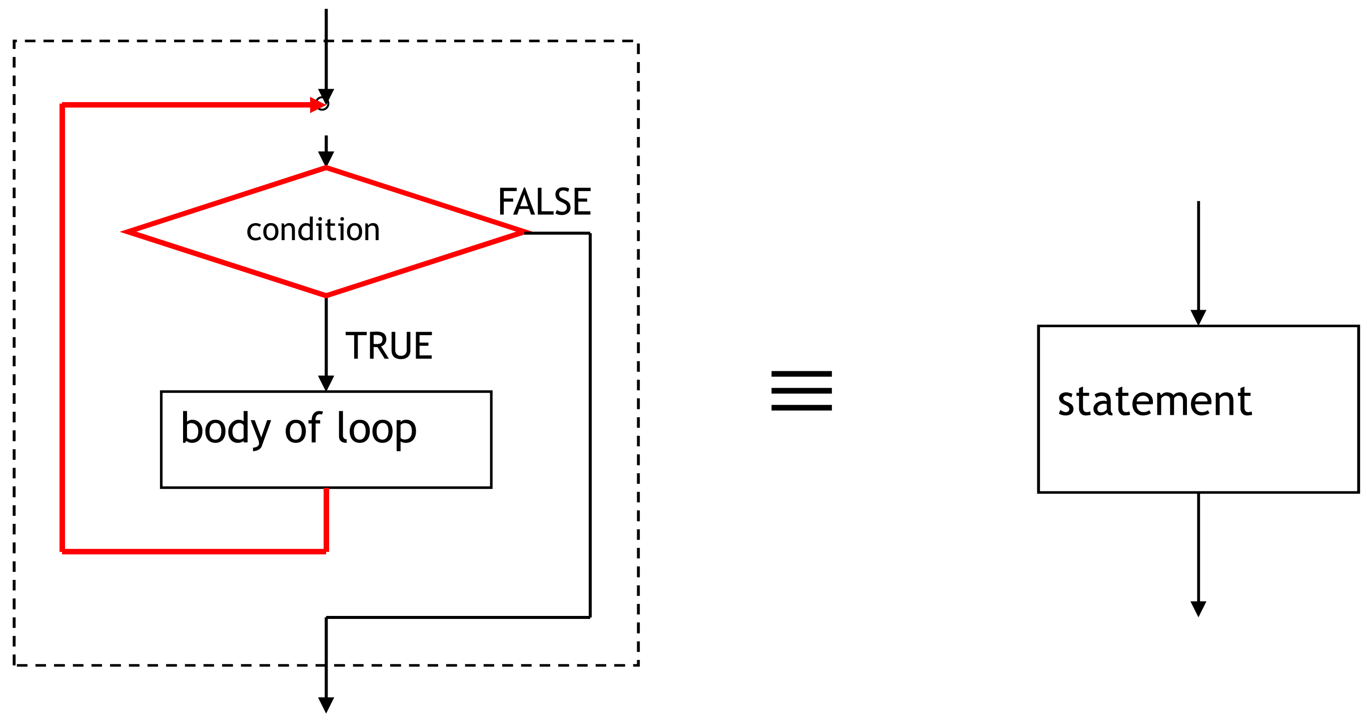
Repetition Structure (*cont..*)

- 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)

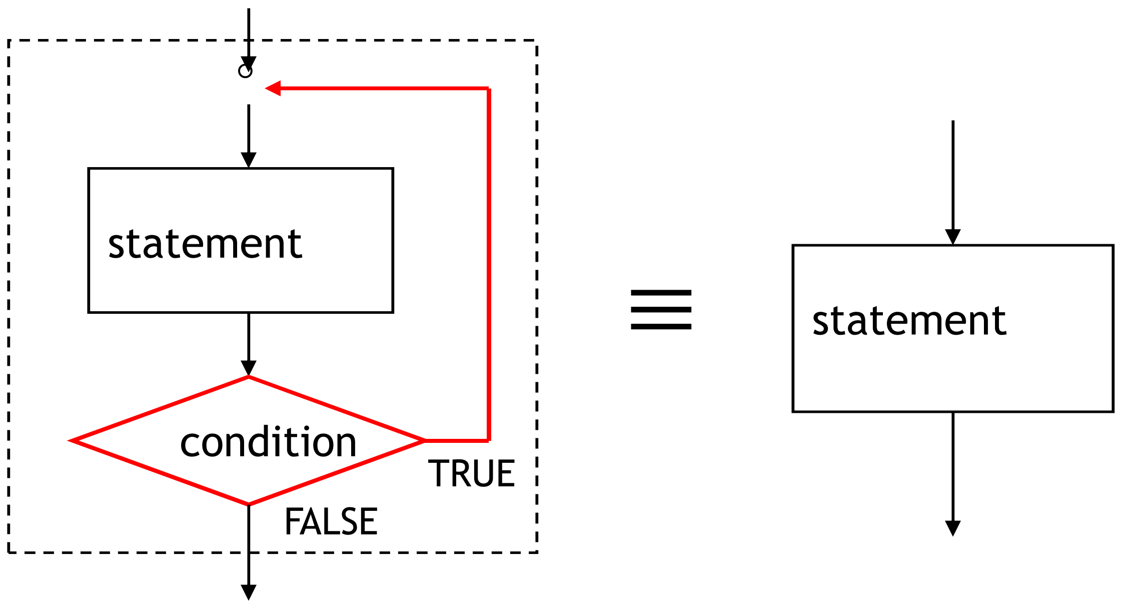
Repetition Structure (*cont..*)

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

```
Algorithm: one choice selection
:
n.   Repeat
          n.1 statement
               :
n+1. until condition
:
```

Repetition Structure (*cont..*)

do-while Loop
(post-test loop)

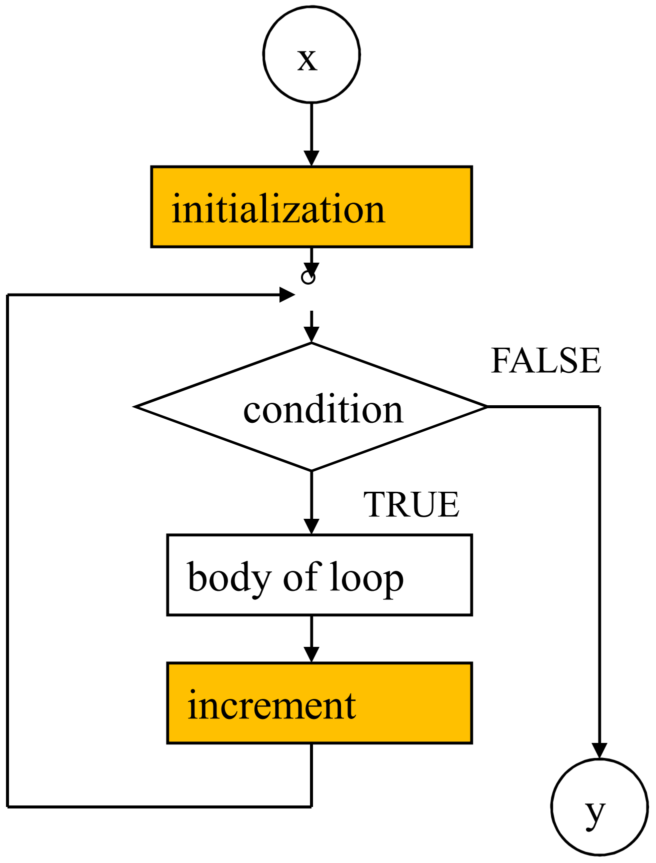


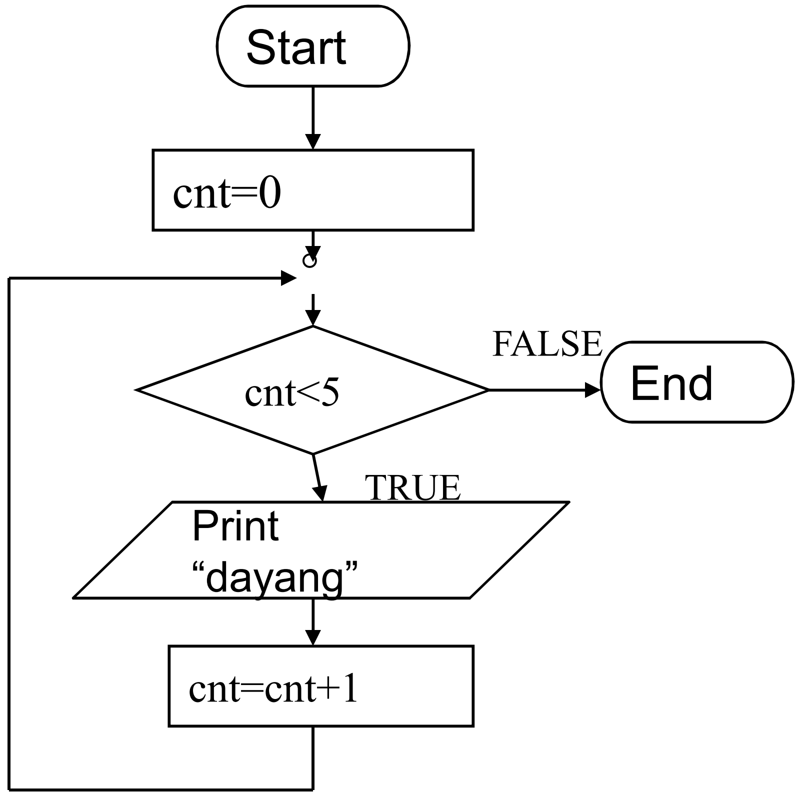
Do the statement (body of loop) while a condition is true

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 repeats
- Must be initialized before entering loop

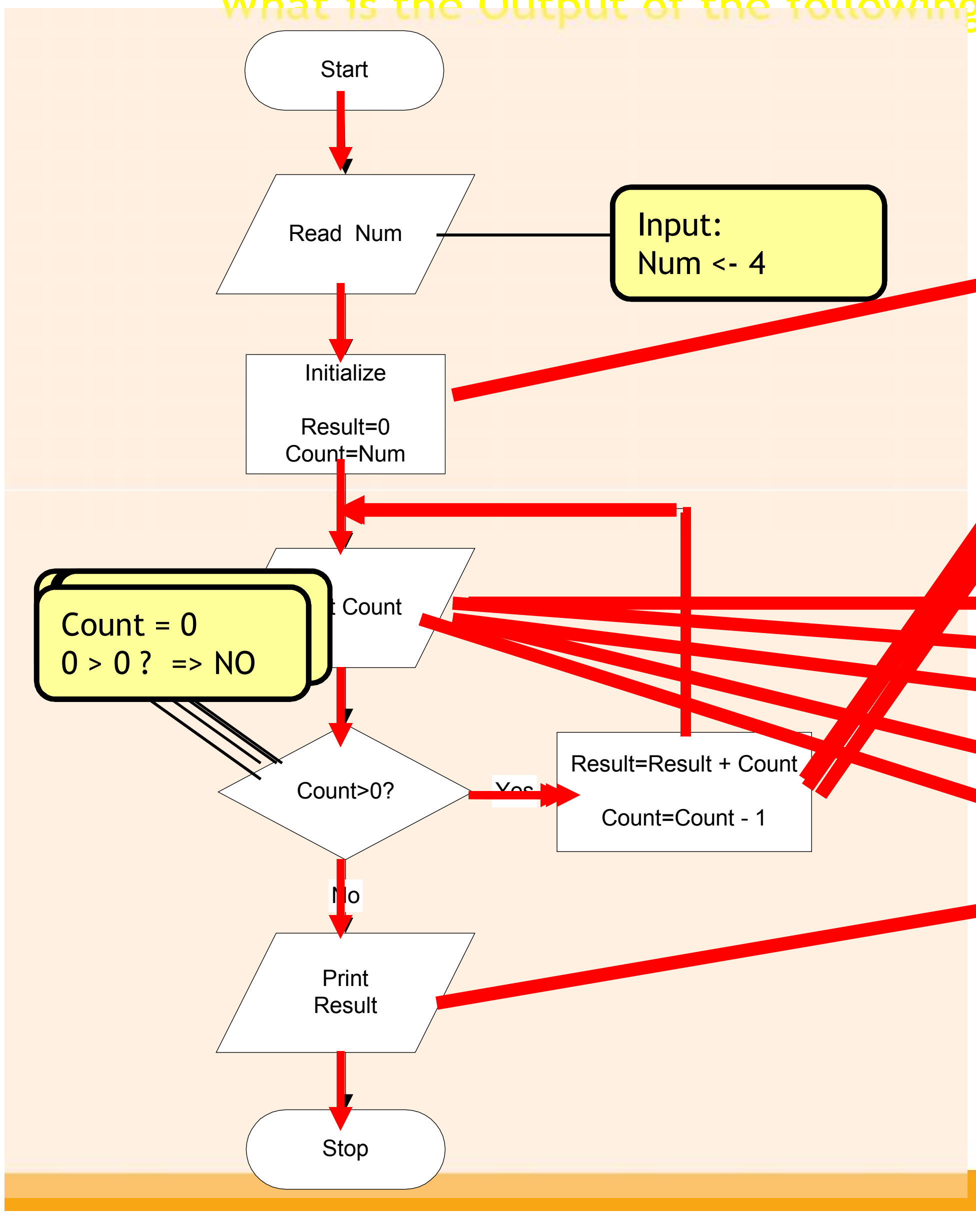
Repetition Structure (*cont..*)





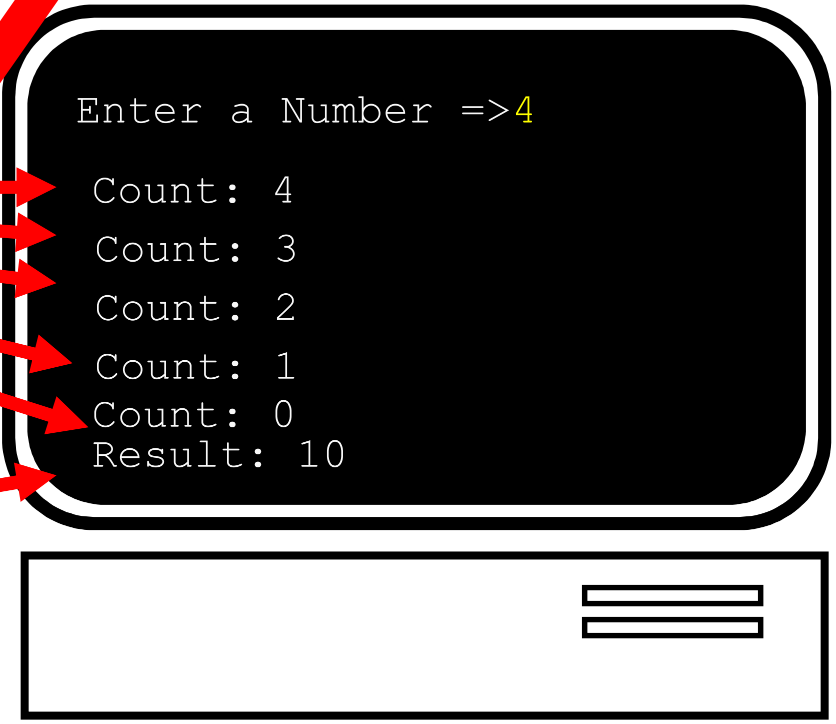
Repetition Structure (*cont..*)

What is the Output of the following flowchart when the input is 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
```

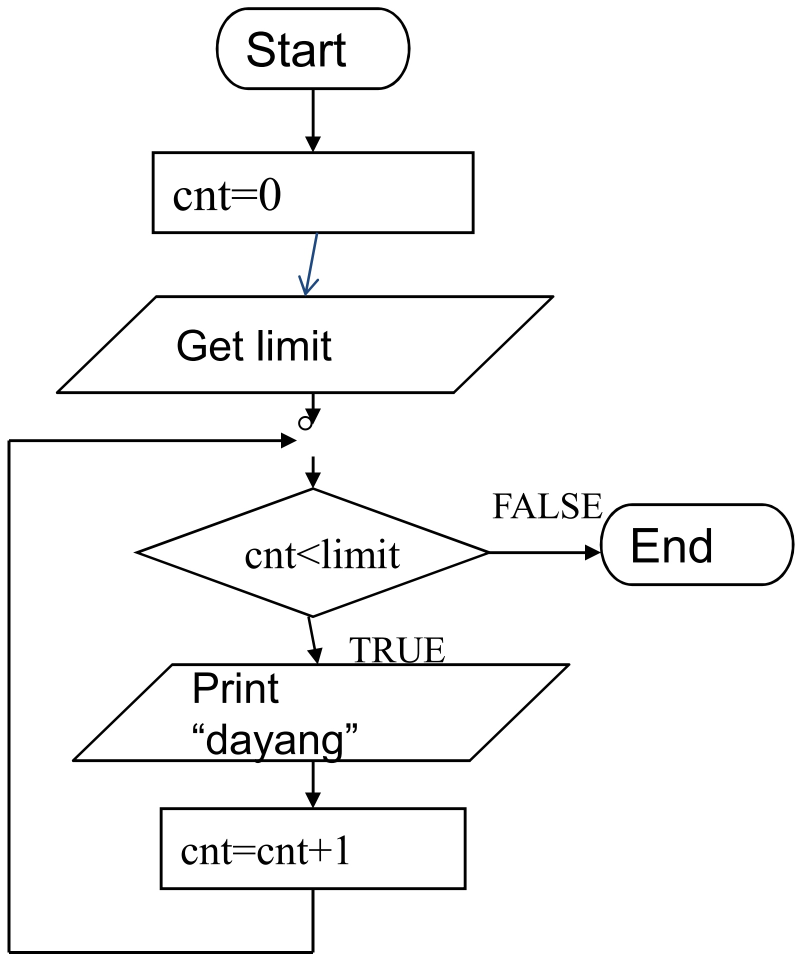
Exercise Week2_8

- Refer to Lab 3, Exercise 2, No. 1 in pg. 37.
- Complete the exercise

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

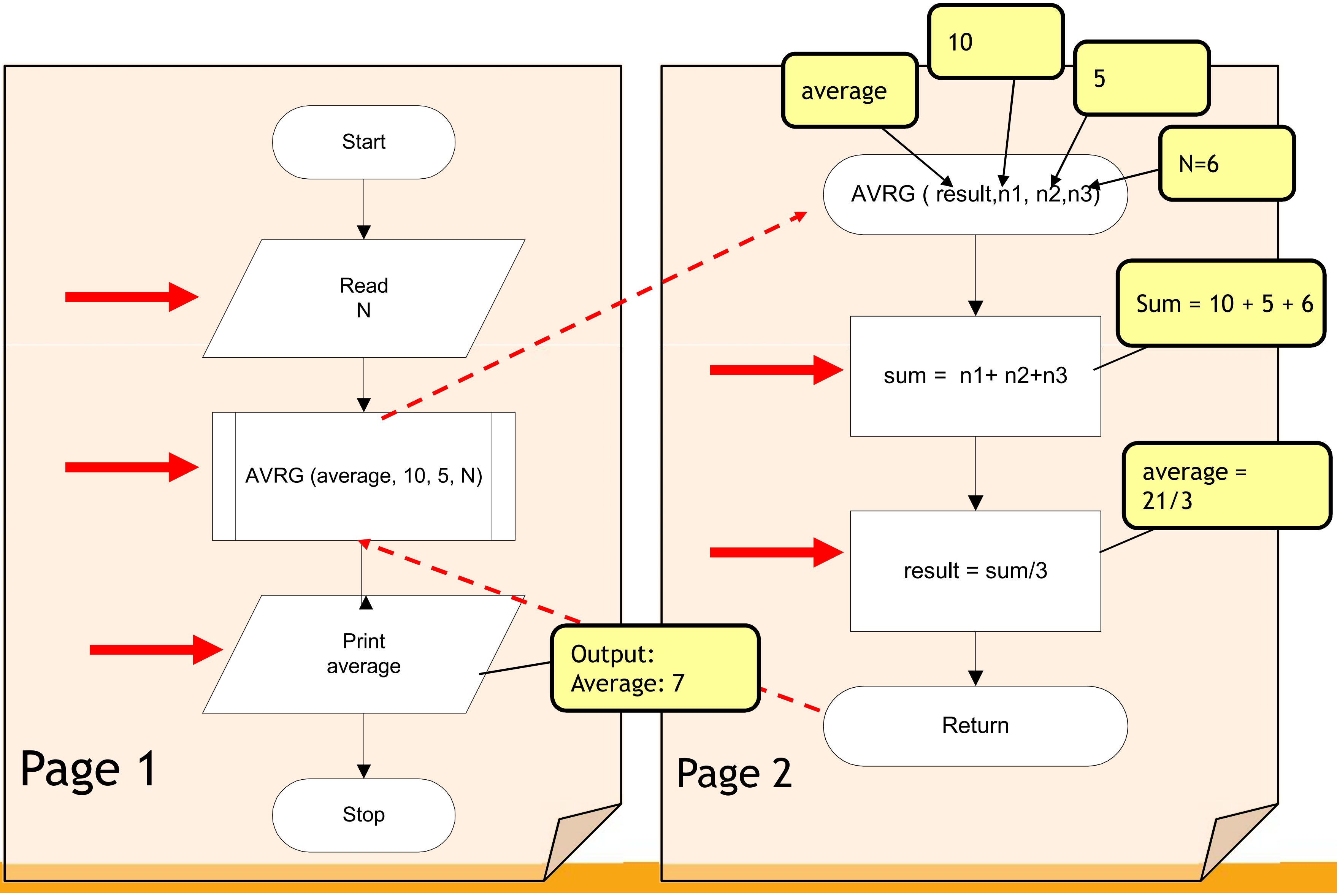
```
Algorithm 3.3: Loop control by sentinel value

1. St art
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
```

Exercise Week2_9

- Refer to Lab 3, Exercise 2 No. 3, in pg. 39.
- Identify the sentinel value
- Complete the exercise

Repetition Structure (*cont..*)

What is the Output of the following flowchart when the input is N = 6



Exercise Week2_10

- Refer to Lab 3, Exercise 3, No. 2 in pg. 41.
- Complete the exercise

Structure Chart

Control Structures

- Describe the flow of execution

- Basic types of control structure:
  1. Sequential
  2. Selection
  3. Repetition

structure chart (*cont..*)

- Also called module chart, hierarchy chart - is a graphic depiction of the decomposition of a problem.
- illustrates the partitioning of a problem into subproblems and shows the hierarchical relationships among the parts.
- It is a tool to aid in software *design* - aid the programmer in *dividing and conquering* a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain.
- The process is called *top-down design*, or *functional decomposition*.

Structure chart (*cont*..)

Structured software follows rules:

1. Modules are arranged hierarchically.
2. There is only one root (i.e., top level) module.
3. Execution begins with the root module.
4. Program control must enter a module at its entry point and leave at its exit point.
5. Control returns to the calling module when the lower level module completes execution

Structure chart (*cont..*)

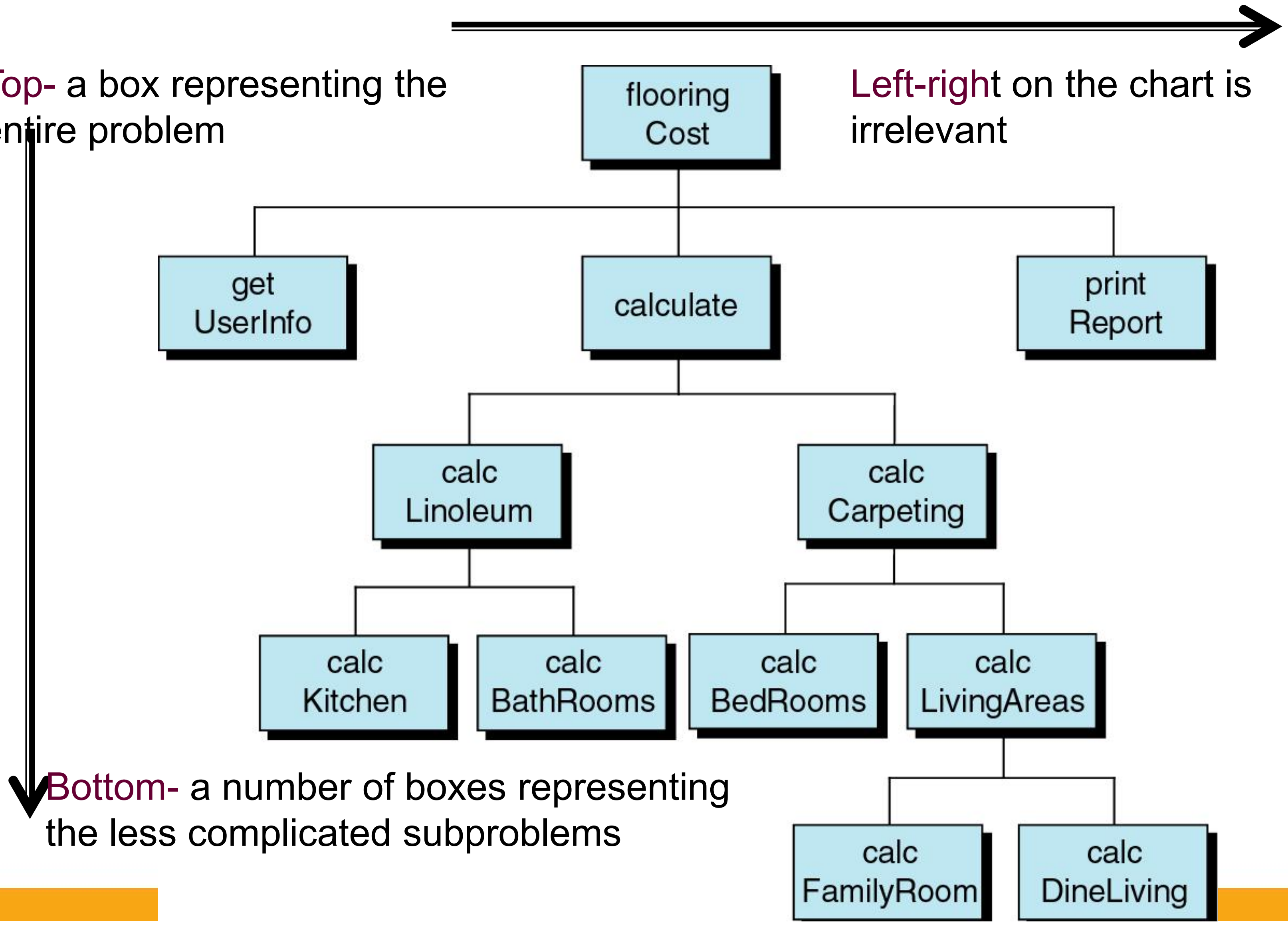
When designing structured software, three basic constructs are represented :

1. Sequence - items are executed from top to bottom. (PT1)
2. Repetition - a set of operations is repeated. (PT2)
3. Condition - a set of operations are executed only if a certain condition or CASE statement applies.(PT2)

Structure chart (*cont..*)

Top- a box representing the entire problem

Left-right on the chart is irrelevant

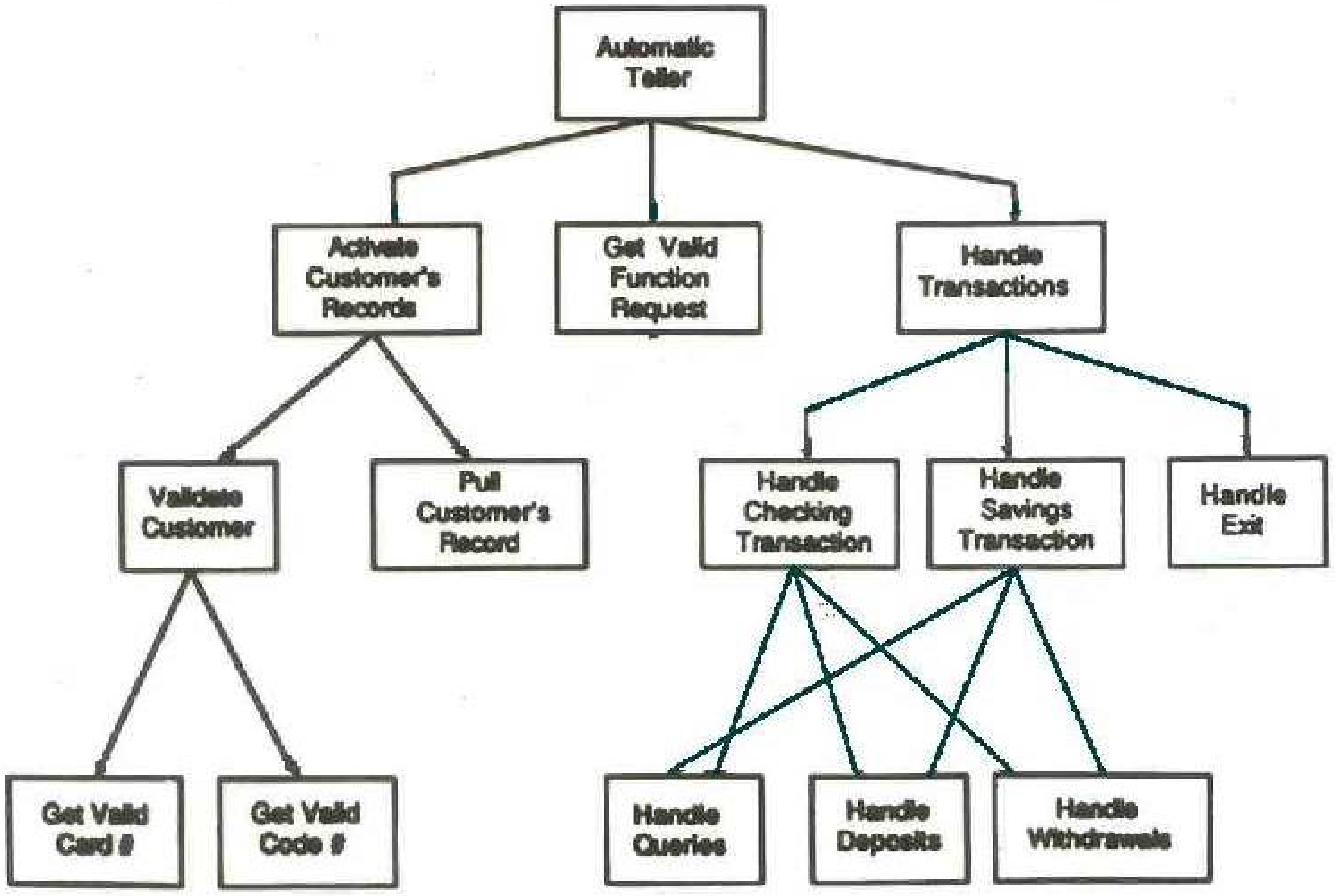


Bottom- a number of boxes representing the less complicated subproblems

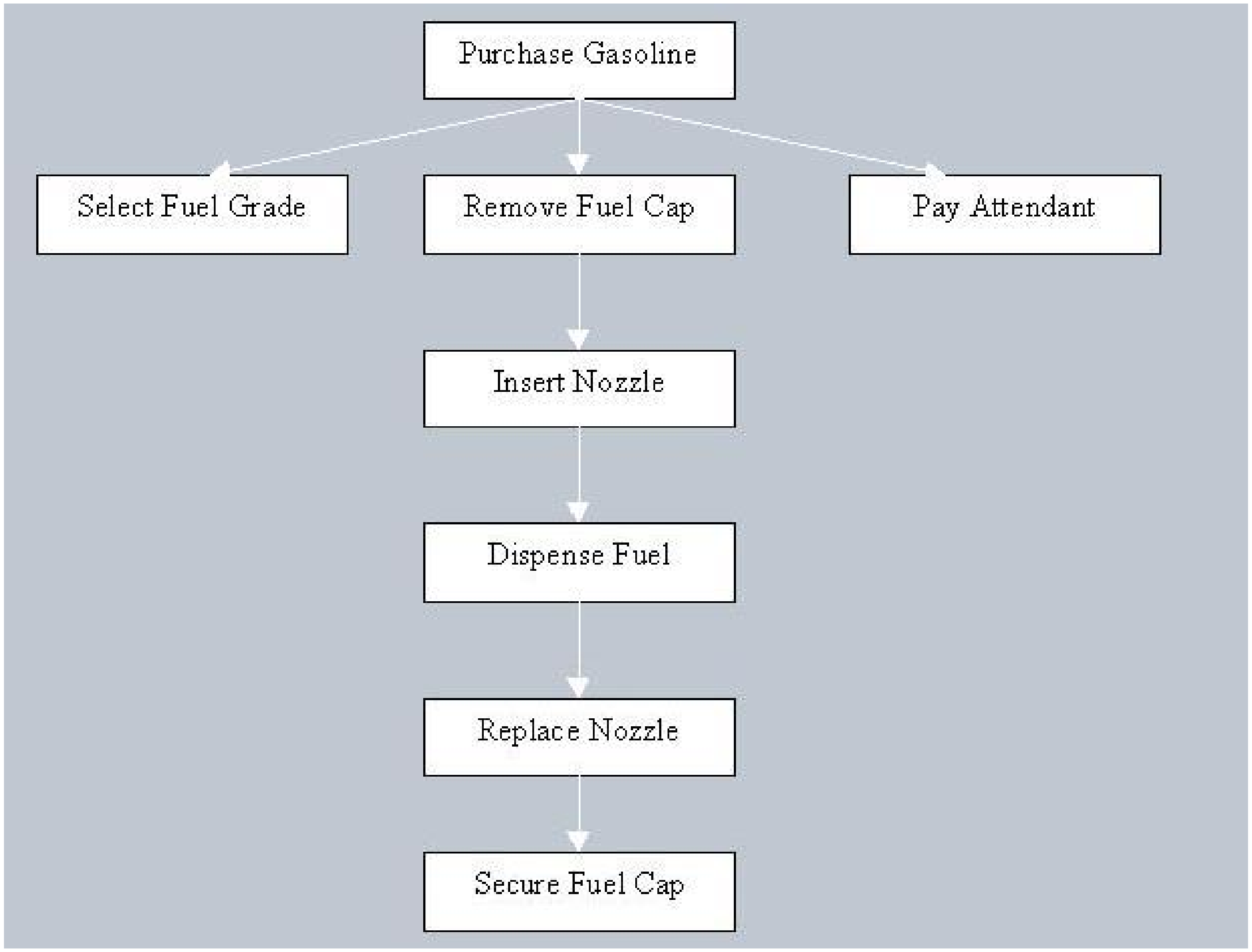
A structure chart is (*cont..*)

- NOT a flowchart.
- It has nothing to do with the logical sequence of tasks.
- It does NOT show the order in which tasks are performed.
- It does NOT illustrate an algorithm

Example ATM Machine structure chart



Common Errors in Structure Charts



Structure chart– revisit case study

- Case Study: Calculate the Payment

Exercise Week2_11

- Refer to Lab 2, Exercise 3, No. 5 in pg. 28-29.
- Complete the exercise

Thank You

Q & A