ME 444
MATLAB® FOR ENGINEERS

Lecturer:

Dr. Nurettin Furkan DOĞAN

Mechanical Engineering Department

4 th Floor Office No: 303

nfdogan@gantep.edu.tr

0342 317 1200- 2569

Office Hours:

Tuesday: 13.30- 15:00

Friday : 10.00- 11.30

CHAPTER 2

MAKING ALGORITHM AND FLOW CHART

Algorithm Concept, Classification, Development

Algorithm Concept

- The algorithm was put forward in the work of the mathematician Al-Harizmi, who lived in the 800s.
- As can be understood from its history, the algorithm was used to solve problems in the field of mathematics before it entered the computer world.
- Later, with the development of computers, it began to be used in solving problems in this field.

Algorithm Concept

- In simplest terms, an algorithm is a solution path consisting of a finite number of steps to be followed to solve a problem. In other words, *an algorithm is a verbal expression of how to perform the logical solution of a problem step by step*.
- Since the solutions created by the algorithm are expressed verbally, flow diagrams are used to make it more standard that everyone can draw the same result when they see it. Flow charts consist of symbols. Each symbol has a specific function.
- A computerized version of a problem whose algorithm has been created is called a program.
- The program is the code equivalent of the whole process that needs to be done to solve the problem.
- Programming languages are used to turn algorithms into programs.
- Software is developed using programming languages.

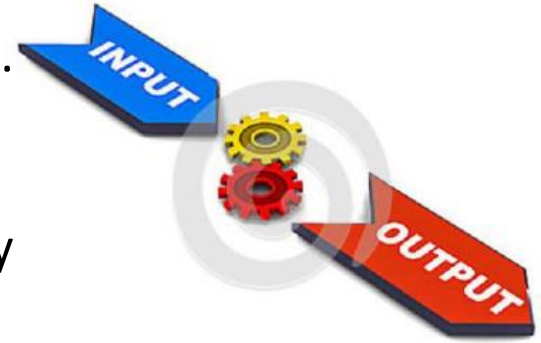
Algorithm Concept

- The main features of the algorithm are:
  - ✓ Input/Output information,
  - ✓ Finiteness ,
  - ✓ Precision,
  - ✓ Efficiency,
  - ✓ Achievement and performance.

Algorithm Concept

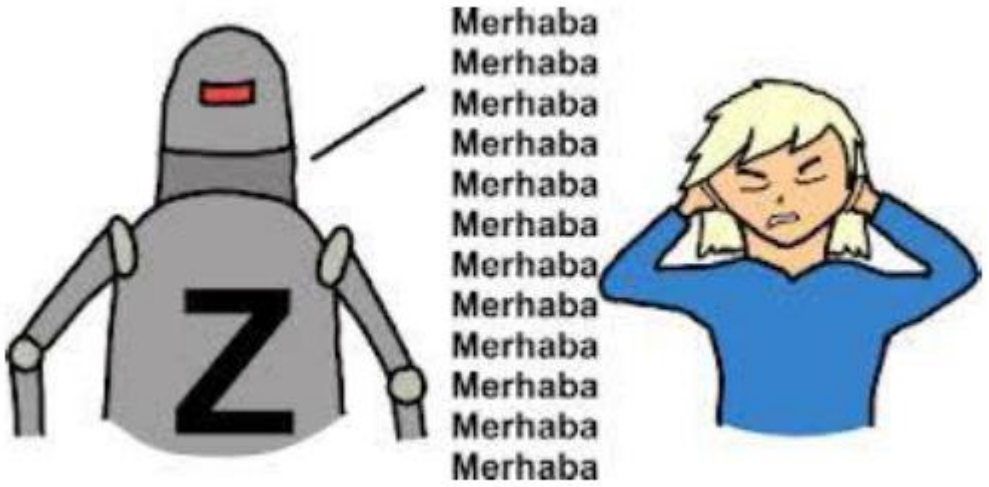
- **Input/Output information**

  - Algorithms must have input and output information.
  - The data coming from outside is called **input** information.
  - This data is processed in the algorithm and creates the **output** information. Output information is necessarily present in every algorithm.
  - The main purpose of the algorithms is to generate output information by processing the input information.
  - However, the output information of an algorithm cannot fully meet the requirements in all cases. In such cases, the output information produced by the first algorithm is sent to another algorithm as input information, so that the user has the desired information.



Algorithm Concept

- **Finiteness :**
  - For all possibilities, the algorithm must finish in finite steps,
  - The algorithm should not go into an infinite loop.



Algorithm Concept

- **Precision:**
  - Each command should be simple enough that one can execute it with pen and paper.
  - Each step of the algorithm should be clearly, simply and precisely stated.
  - It should not require comments and should not have ambiguous statements.

Algorithm Concept

- **Efficiency:**
  - Written algorithms should be created effectively and thus away from unnecessary repetitions. This is one of the main features of the algorithm.
  - In addition, algorithms should be written for general purposes and should be composed of a structural main algorithm and sub-algorithms. Thus, an algorithm written before can be used for other operations later on.
  - To give an example, if we have an algorithm that we use to find the average of the given n numbers, this algorithm should also be able to be used for an algorithm that finds the average age of students in a class.

Algorithm Concept

- **Achievement and performance:**
  - The goal should be to write high-performance programs, taking into account performance criteria such as hardware requirement (such as memory usage), uptime. Unnecessary repetitions should be eliminated. The following criteria are considered in the performance evaluation of an algorithm.
  - Unit processing time
  - Data search and fetch time
  - Comparison time
  - Transfer time.

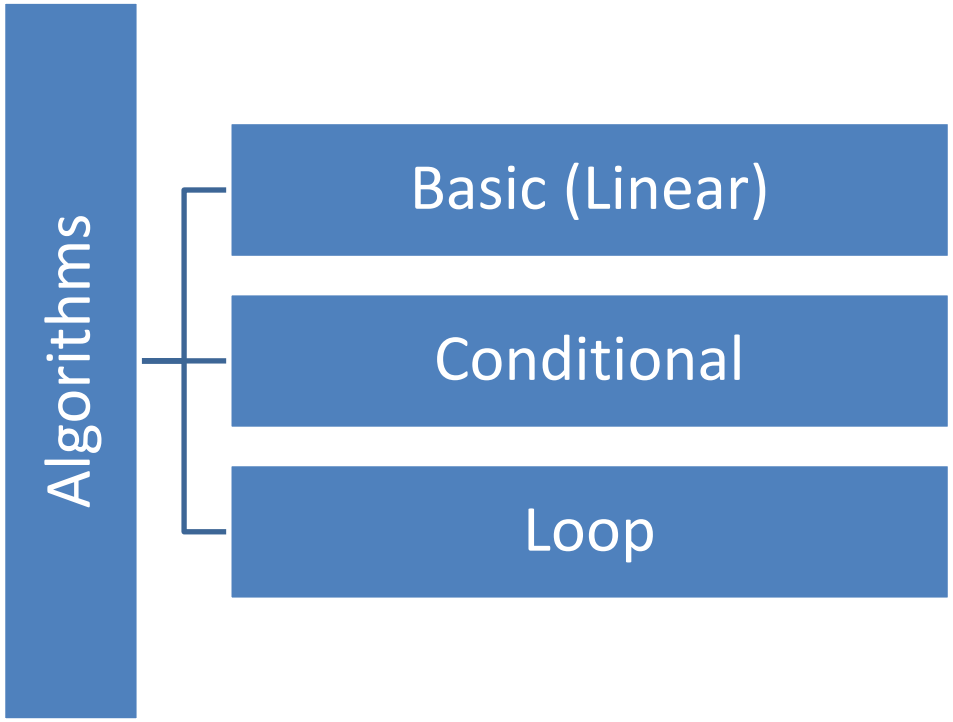
Algorithm- Example

- The solutions to be created with the algorithm are expressed verbally. For example, if a breakfast preparation algorithm is created when breakfast will be made when we wake up in the morning:

  - ✓ Get out of bed
  - ✓ Go to the kitchen
  - ✓ Take bread
  - ✓ Prepare the tea
  - ✓ Take breakfast out of the fridge
  - ✓ Fill your cup of tea
  - ✓ Get up from the table when you're full
  - ✓ Put the breakfasts in the fridge
  - ✓ Clean the table

Classification of the Algorithm

- Algorithms are examined in 3 groups according to their complexity.



Basic (Linear) Algorithms

- They are algorithms that do not contain logical expressions and do not have program flow branches.
- In these algorithms, the flow will be straight from start to finish.
- They are mostly used to perform small calculations. According to the previous algorithm example, it is seen that there is no decision structure.

Example:

1. Enter the mileage to be calculated; km
2. Multiply the entered value by 1000; m=km*1000
3. Write the calculated value on the screen; m

Basic (Linear) Algorithms

Example: Algorithm that calculates the sum, product and average of three externally entered numbers

1. Enter three numbers; A B C
2. Calculate the sum of the numbers; total=A+B+C
3. Calculate the product of the numbers; product=A*B*C
4. Calculate the average of the numbers; avg=total/3
5. Print the sum, product and average of the numbers on the screen; sum, product, avg

Conditional (Logical) Algorithms

- They are structures that contain logical comparisons within the algorithm.
- According to logical comparisons, the flow of the algorithm will move to different steps.
- Algorithms created in this way are called Conditional Algorithms.
- If the first created algorithm example is a little more detailed, it is seen that decision structures emerge.

Conditional (Logical) Algorithms

1. Get out of bed
2. Go to the kitchen
3. **IF** there is no bread, buy bread
4. Prepare the tea
5. Take breakfast out of the fridge
6. Fill your cup of tea
7. Get up from the table when you're full
8. **IF** there are no breakfasts in plate put in the dishwasher
9. **IF** there are breakfasts in plate put in the fridge
10. Clean up the table.

Conditional (Logical) Algorithms

Example: Let's write an algorithm that finds the largest number among the three numbers entered.

1. Enter three numbers; A B C
2. Let A be the largest number; largest=A
3. **IF** B is greater than the largest (B>largest), let B be the largest; largest=B
4. **IF** C is greater than (C>largest), let C be the largest; largest=C
5. Print the largest number to the screen; largest

Conditional (Logical) Algorithms

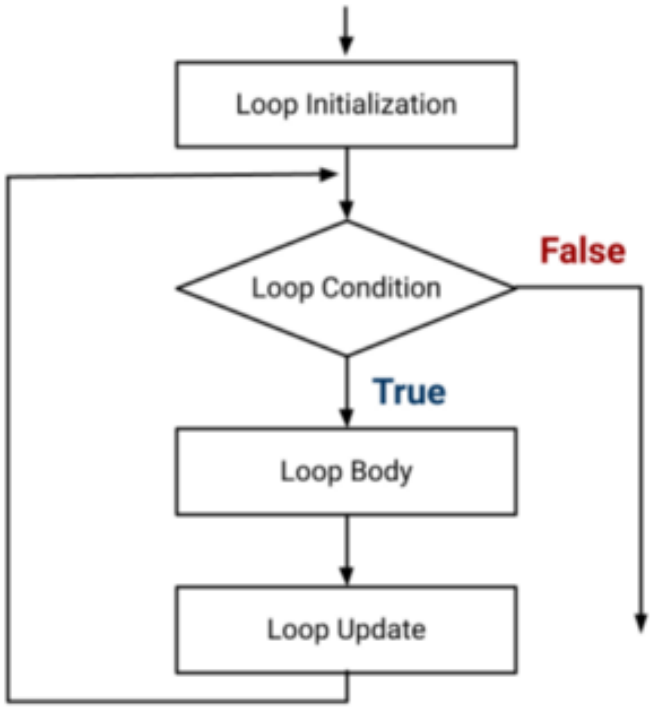
Example: Let's write an algorithm that finds whether the number is positive, negative, or zero.

1. Enter the number; a
2. IF the number a is greater than zero, write 'positive' on the screen and go to step 5;
3. IF the number a is less than zero, write 'negative' on the screen and go to step 5;
4. IF the number a is equal to zero, write 'zero' on the screen and go to step 5;
5. The program is finished.

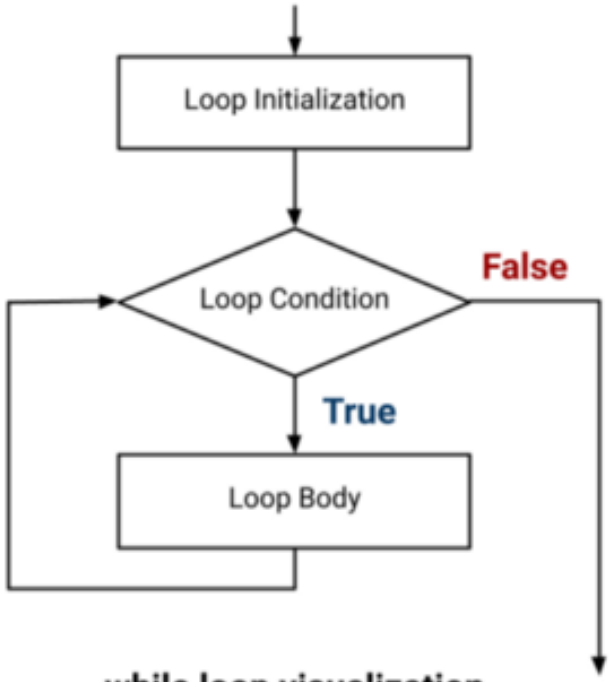
Loop Algorithms

- In the algorithm developed for the program, if a process repeats more than once, the loop algorithm structure is used.
- The logical comparison structure is specially used in loop algorithms.
- If, as a result of the logical comparison process used in the algorithm, the flow of the program goes to the previous step, not to a further step from the comparison place, the algorithms created in this way are called loop algorithms.
- In other words, as a result of logical comparison in loop algorithms, the program goes to the previous steps.

Loop Algorithms



for loop visualization



while loop visualization

Loop Algorithms

Example: Let's write an algorithm that finds the factorial of a number.

1. Enter the factorial number to be calculated; n
2. Initialize the factorial to 1; f=1
3. Set the index value to 1; i=1
4. Multiply the factorial value by the index and write the calculated value into the factorial; f = f x i
5. Increase the index value by 1; i=i+1
6. IF the index value is less than or equal to the number entered, go to Step 4;
7. Write the factorial value to the screen; f

Loop Algorithms

Example: Let's write an algorithm that finds the greatest common divisor of the two numbers entered

1. Initialize the factorial to 1; f=1
2. Set the index value to 1; i=1
3. Multiply the factorial value by the index and write the calculated value into the factorial; f = f x i
4. Increase the index value by 1; i=i+1
5. IF the index value is less than or equal to the number entered, go to Step 4;
6. Write the factorial value to the screen; f

Making Algorithms

- The algorithm developed to solve a problem can be written in three ways:
  - **Line algorithm:** The solution steps of the problem are written in plain text in clear sentences.
  - **Flowchart:** The solution steps of the problem are shown with geometric figures.
  - **Pseudo-code:** The solution steps of the problem are expressed with clear command-like texts or abbreviations.

Making Algorithms

- **Example:** Let's show the algorithm of the program that adds two numbers entered from the keyboard and displays it on the screen with 3 different methods.

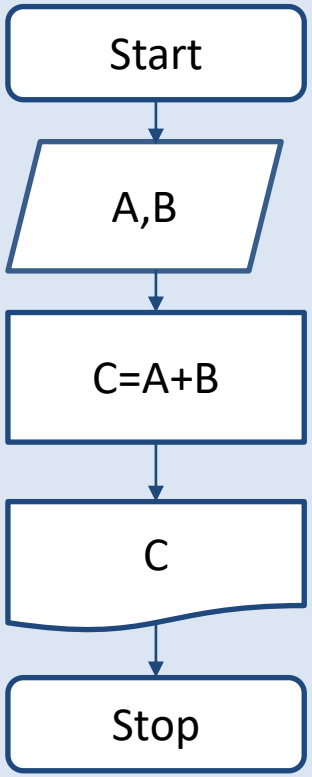
With the line algorithm:

1. Start
2. Read the first number (A) from the keyboard
3. Read the second number (B) from the keyboard
4. Create the result by adding the entered numbers (C=A+B)
5. Print the result (C) to the screen
6. Stop

With pseudocode:

1. Start
2. Read A
3. Read B
4. C=A+B
5. Write C
6. Stop

With the flowchart:



Making Algorithms

- While developing algorithms, some elements such as variable, constant, assignment, loop, decision structure, subroutine are used.
  - Data: All information processed by computers is called data. Data are basically divided into two as *numeric* and *alphanumeric*.
  - Identifier: These are the names given by the software developer to the programming units such as variables, constants, subroutines, and fields.
  - Variable: It is the part of memory allocated to hold different values in the flow of the program. For example, in an expression like C=A+B, the identifiers A, B, and C are variables.
  - Constant: An identifier that always returns the same value every time the program runs and at any time within the program is called constant. After an expression like PI=3.14, the PI constant refers to the value 3.14 throughout the program.

FLOWCHART

Start/End Symbol

The terminator symbol marks the starting or ending point of the system. It usually contains the word "Start" or "End."

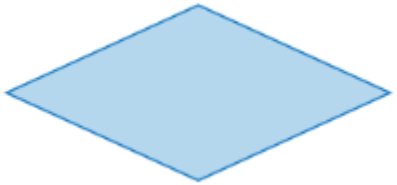
Action or Process Symbol

A box can represent a single step ("add two cups of flour"), or an entire sub-process ("make bread") within a larger process.

Document Symbol

A printed document or report.

FLOWCHART



Decision Symbol

A decision or branching point. Lines representing different decisions emerge from different points of the diamond.

Input/Output Symbol

Represents material or information entering or leaving the system, such as customer order (input) or a product (output).

Subroutine Symbol

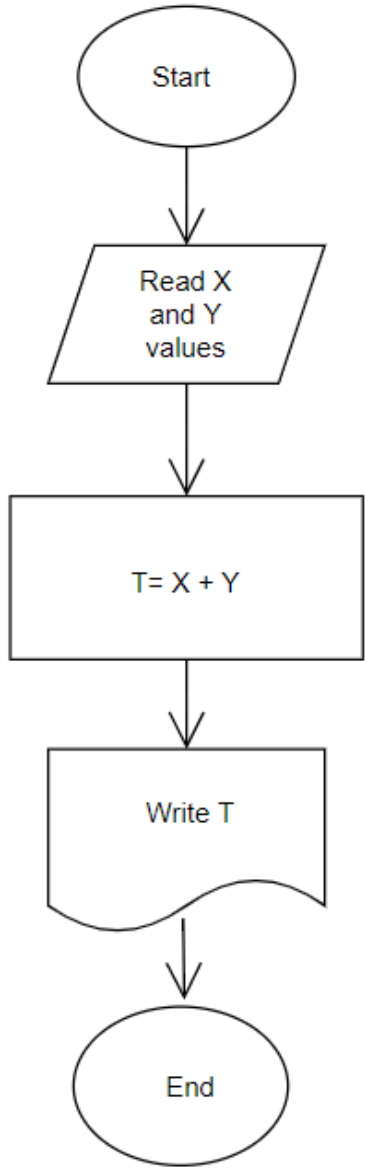
Indicates a sequence of actions that perform a specific task embedded within a larger process. This sequence of actions could be described in more detail on a separate flowchart.

Example: Summation of two numbers

Using pseudocode

Assign T for the sum, X for the first number, Y for the second number

1. START
2. READ X value
3. READ Y value
4. T = X + Y
5. WRITE T value
6. FINISH



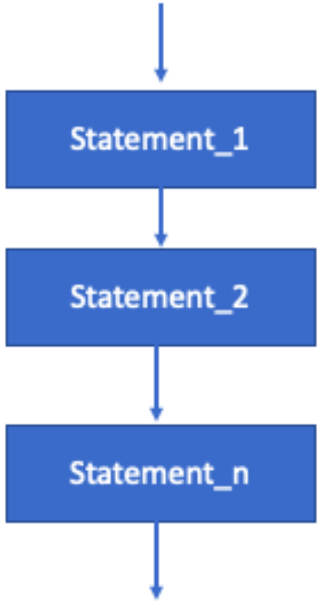
Logical Structures

Regardless of the programming language used in the development of a computer program, three simple logical structures are generally used in the flowcharts of this program:

1. Sequential Structure
2. Decision Making Structure
3. Repetitive Structure

Logical Structures: Sequential structure

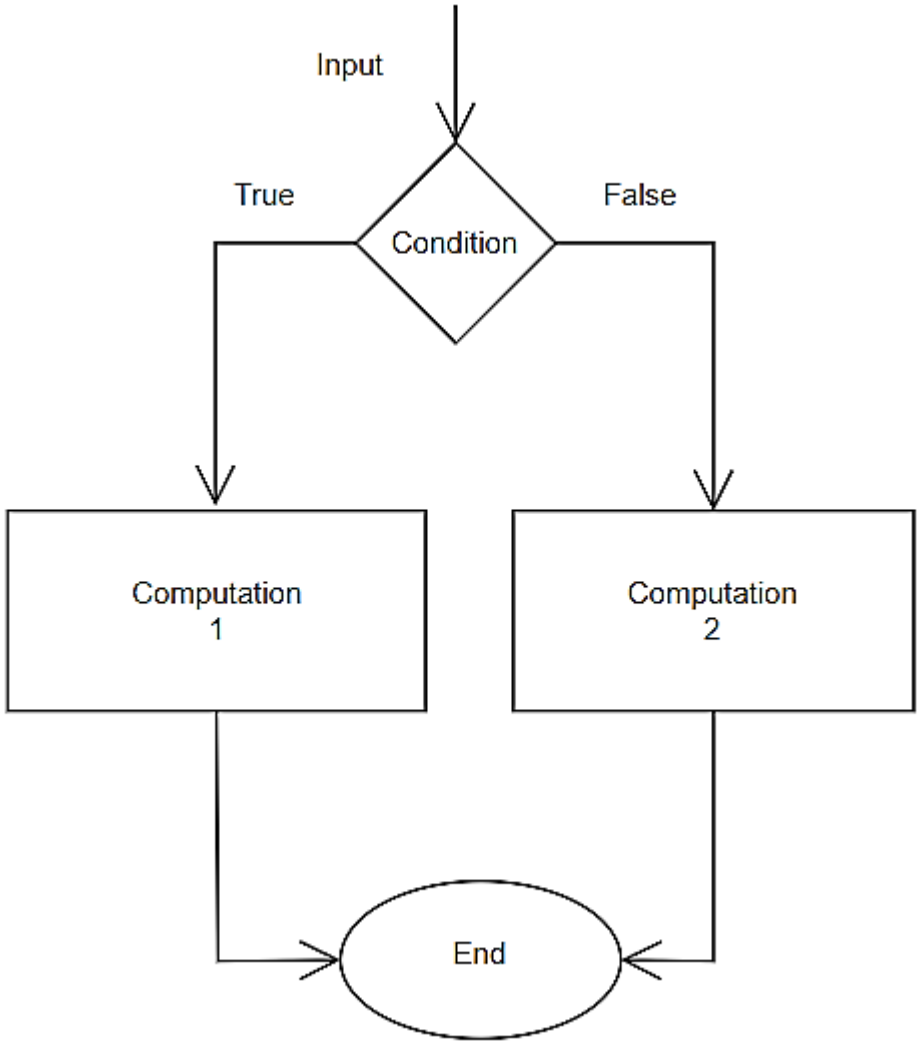
The sequential structure emphasizes where each operation in the program to be prepared should take place in logical order. A second process cannot start until this build is over.



Sequence Control Structure

Logical Structures: Decision Making Structure

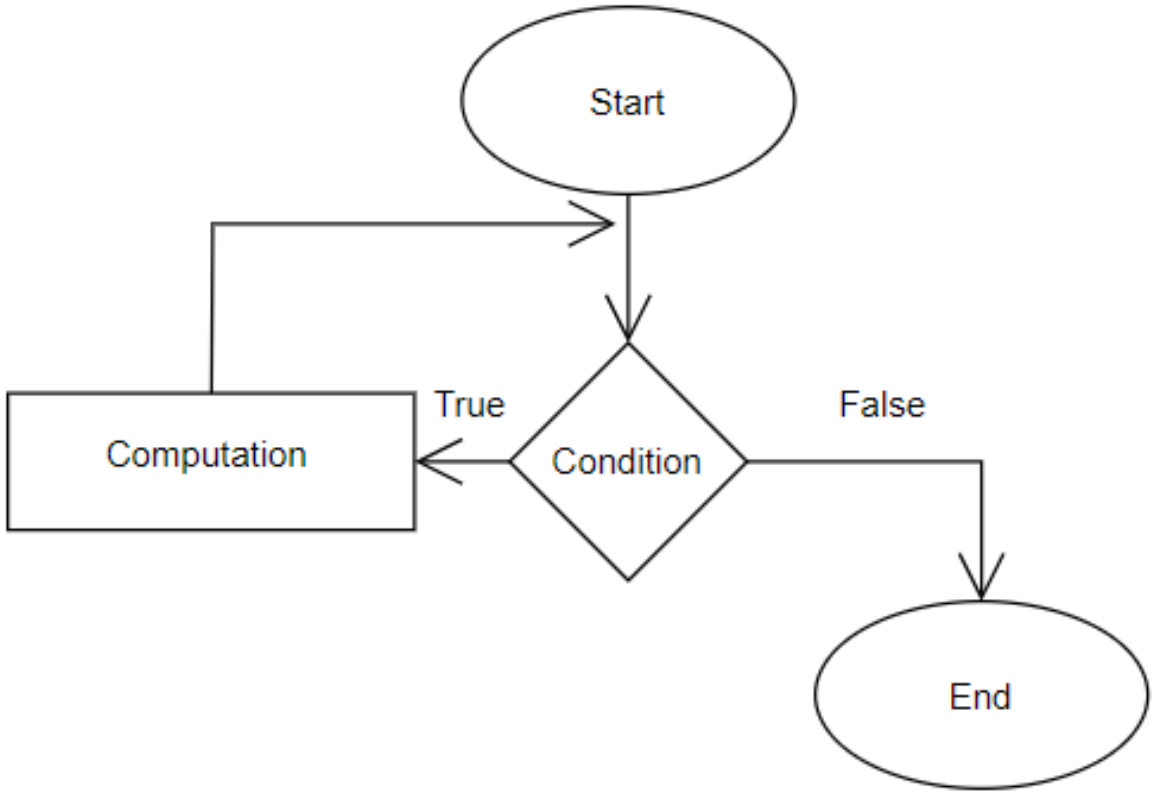
In modules that include more than one sequential structure option, this structure determines which sequential structure will be selected under which conditions.



Logical Structures: Repetitive Structure

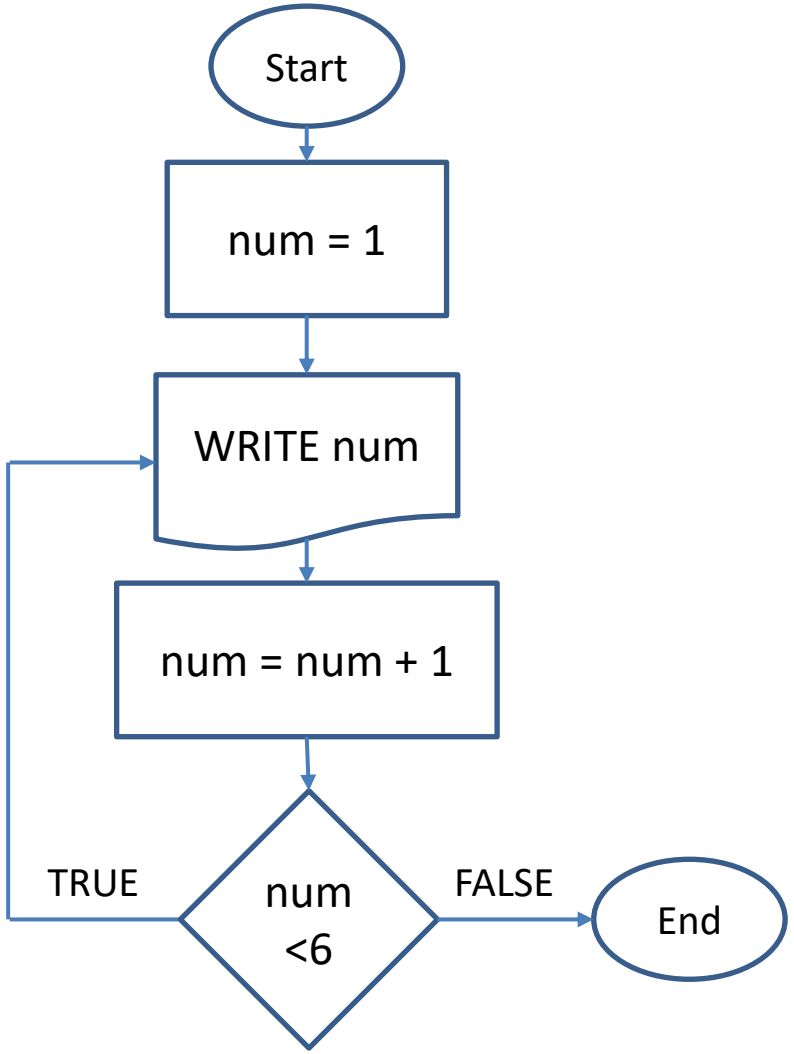
A loop exists within the algorithm if some rows are being processed repeatedly.

Loops are used to describe actions that continue as long as a certain condition is true.



Example: Printing numbers between 1-5 on the screen

1. START
2. num = 1
3. WRITE num value
4. num = num + 1
5. IF num<6, GO TO STEP 3
6. END



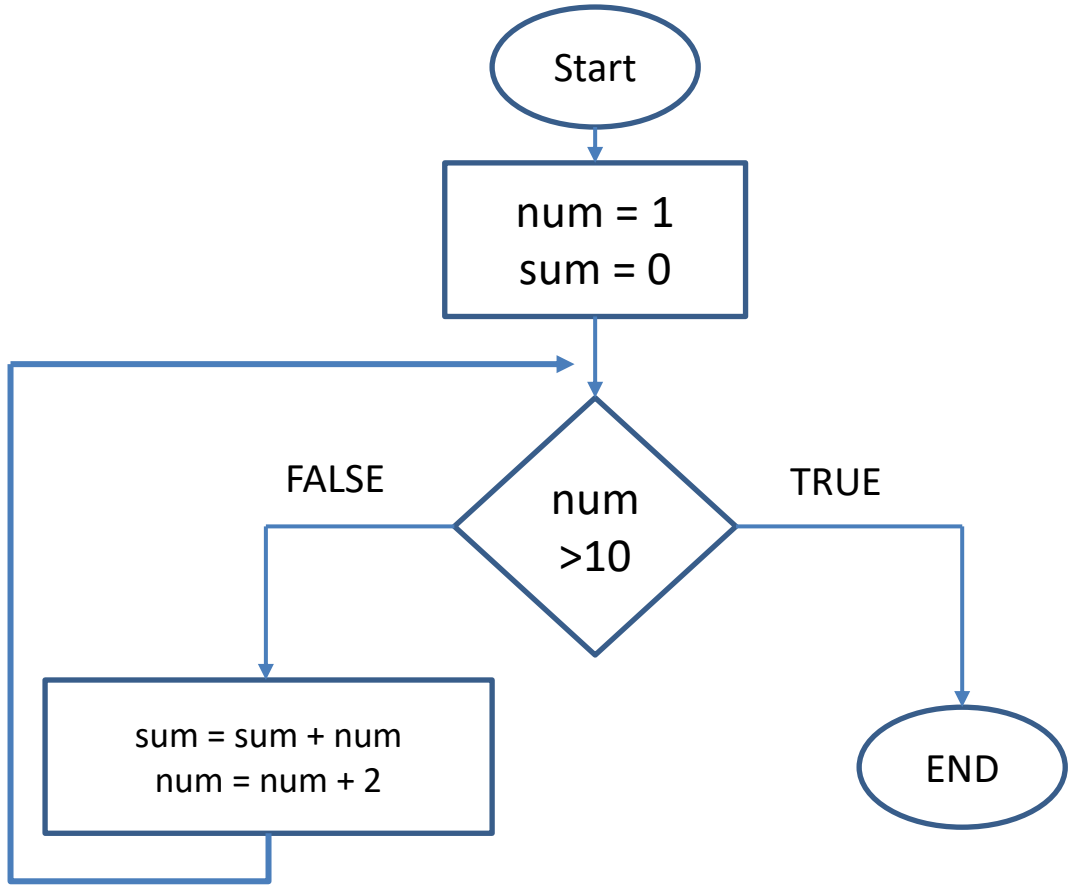
Example: Printing numbers between 1-5 on the screen

1. START
2. num = 1
3. WRITE num value
4. num = num + 1
5. IF num<6, GO TO STEP 3
6. END

OLD num	NEW num	SCREEN
1	2	1
2	3	2
3	4	3
4	5	4
5	6	5

Example: Sum of odd numbers from 1 to 10

1. START
2. num = 1
3. sum = 0
4. IF num>10, GO TO 8
5. sum = sum + num
6. num = num + 2
7. GO TO 4
8. END



Example: Sum of odd numbers from 1 to 10

1. START
2. num = 1
3. sum = 0
4. IF num>10, GO TO 8
5. sum = sum + num
6. num = num + 2
7. GO TO 4
8. END

Old num	Old sum	New sum	New num
1	0	1	3
3	1	4	5
5	4	9	7
7	9	16	9
9	16	25	11
11			

Next week

Chapter 3

MATLAB FUNDAMENTALS