The National School of Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي

Data Structure & Algorithms 1

CHAPTER1: INTRODUCTION TO ALGORITHMIC THINKING & PROBLEM SOLVING

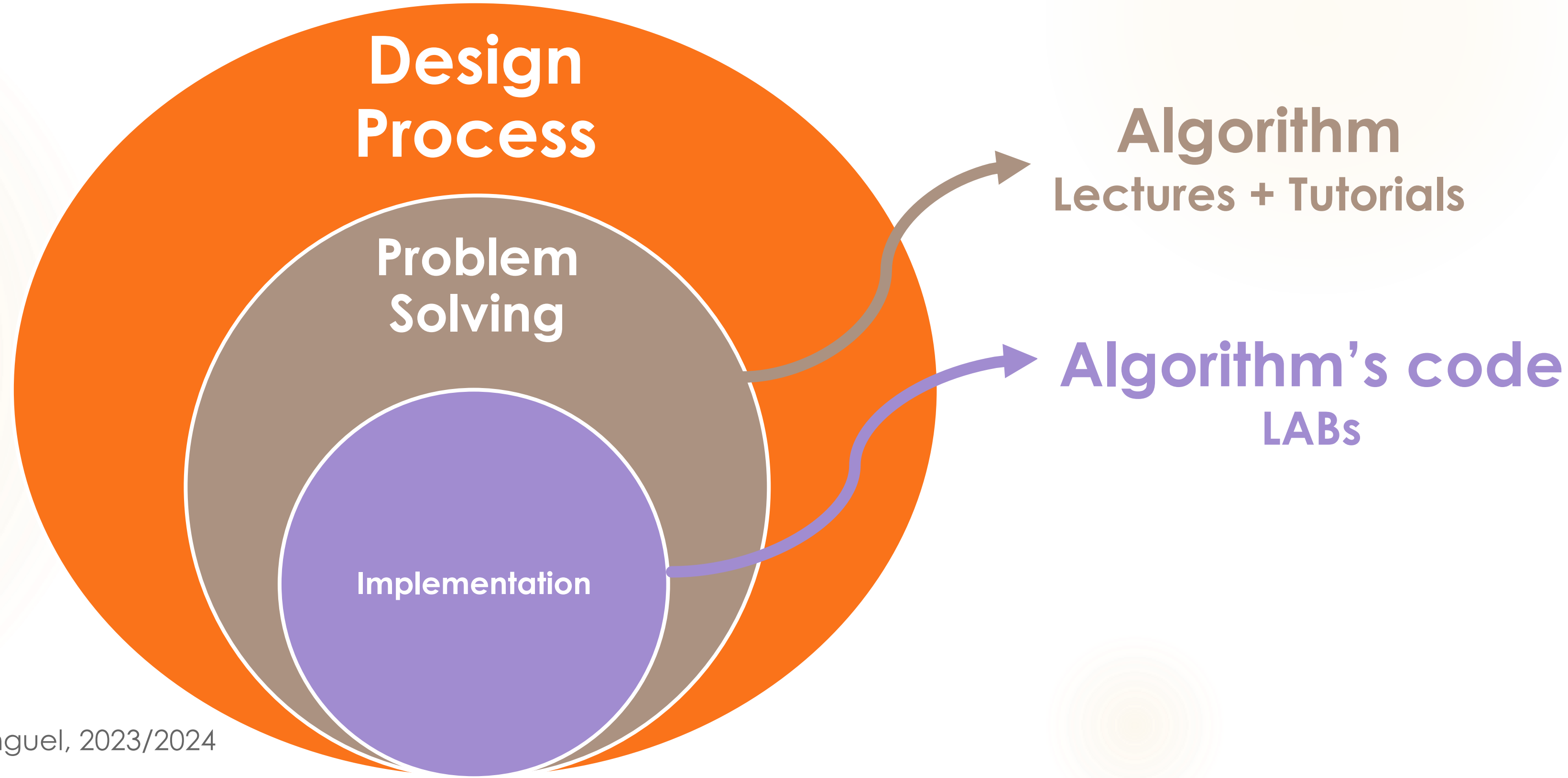
Dr. Fouzia Anguel & Dr. Noureddine Lasla

Sep – Dec 2023

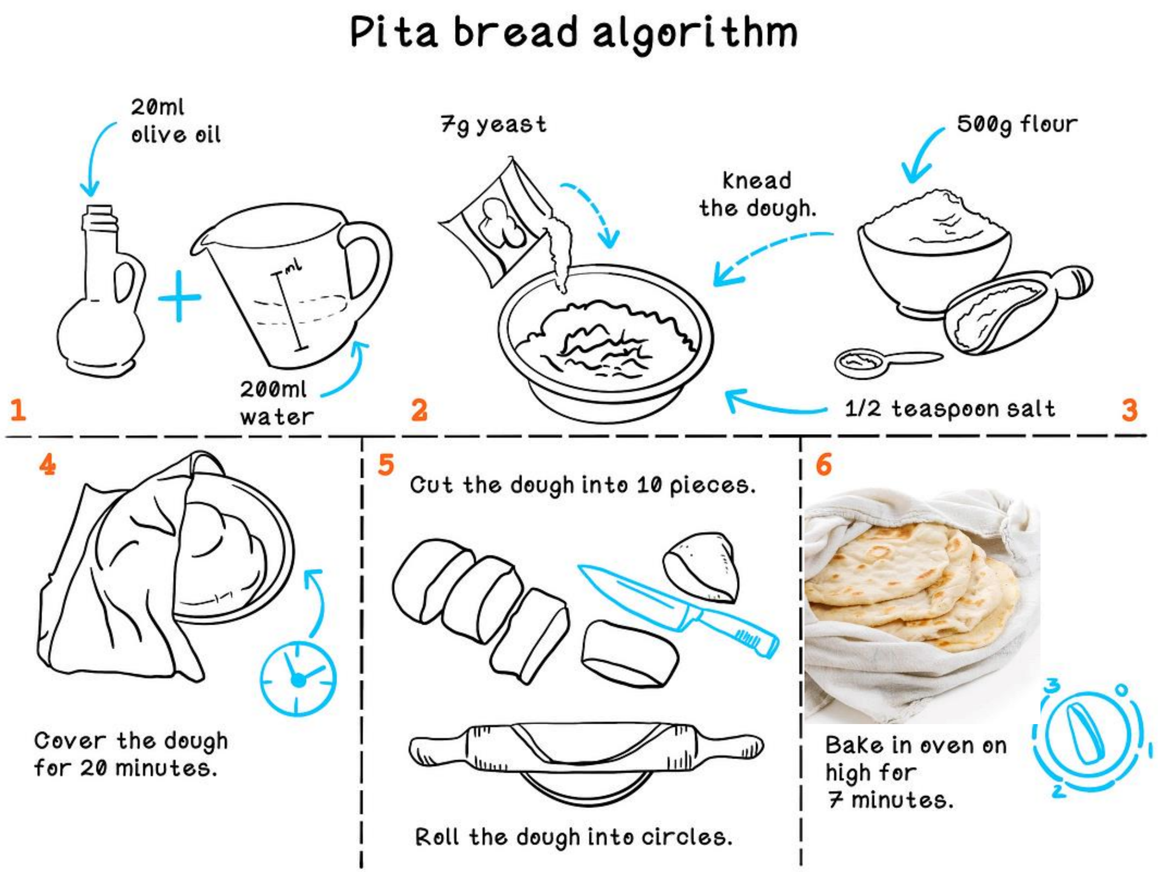
Program Design

Programming is a creative process

No complete set of rules for creating a program



What is an Algorithm?



- An algorithm is a **plan of actions** described using a certain language.

OR

- A description of **a method** to be implemented in order to **solve a problem**.

OR

- A **sequence** of precise instructions that leads to a **solution**

Algorithm Properties

- **Completeness**
  - Algorithms must consider all possible cases.
  - They address both general and specific scenarios.
- **Finiteness**
  - Algorithms always consist of a finite number of actions.
- **Repetitiveness**
  - Algorithms are often repetitive.
  - They include processes that repeat as needed.
- **Language and Hardware Independence**
  - Algorithms are not tied to programming languages or specific hardware.
  - After development, they are translated into a chosen programming language.

Algorithm Characteristics

To precisely characterize an algorithm, we propose the following rules:

- **Clear Object Definitions**
  - Every object manipulated within the algorithm should be clearly defined before its use. Avoid any ambiguity in the definitions of objects.
- **Unambiguous Objects**
  - Ensure that each object has a single, unambiguous interpretation.
  - Ambiguity can lead to errors and unintended behavior.
- **Use of Known Operations**
  - Any combination of elementary operations should be constructed using operations that are assumed to be known and well-defined.
- **Finite Number of Operations**
  - For every set of input data, the algorithm must provide a result through a finite number of operations.

Algorithm Methodology (1, 2, 3)

We present a methodology consisting of three (03) key steps for developing algorithms:

1. **Problem Definition**
   - The first step is to define the problem you intend to solve in a precise and unambiguous manner.

2. **Seek a Resolution Method**
   - After defining the problem, the next step is to explore and research various methods for solving it. Investigate different approaches and consider their pros and cons.

3. **Algorithm Implementation**
   - Once a suitable method is identified, proceed to implement the algorithm with explicit details.

Algorithm Methodology

1. Defining the Problem:

1. **Specifying Data:**
   - Define a set of data based on the problem statement, assumptions, or external information sources.
   - Determine what information is required to address the problem.
2. **Setting Goals:**
   - Specify the goals and objectives to be achieved through the algorithm.
   - Identify the desired outcomes and the sequence of operations required to reach those outcomes.
3. **Defining Constraints:**
   - Specify the constraints that must be considered when designing the algorithm.
   - Constraints are often derived from the problem statement and the broader context.

Algorithm Methodology

2. Seek a Resolution Method (1/2):

First of all:

- **Clarifying the Statement:**
  - Begin by carefully analyzing and clarifying the problem statement.
- **Simplifying the Problem:**
  - Identify the core components of the problem that need to be addressed.
- **Incremental Approach:**
  - Avoid attempting to solve the problem directly in its entirety.
  - Instead, consider breaking it down into smaller, solvable sub-problems.
- **Assessing Decidability:**
  - Make sure the problem is solvable, some problems may be undecidable, meaning there is no algorithm that can provide a definite solution for all cases.

Algorithm Methodology
2. Seek a Resolution Method (2/2):

Then, search for an algorithm construction strategy:

- **Decomposing the Problem:**
  - Break down the complex problem into smaller, more manageable sub-problems.
  - Each sub-problem should contribute to the solution of the overall problem.
- **Iterative Refinement:**
  - Begin with a basic outline of the algorithm and gradually refine it.
- **No Universal Strategy:**
  - Understand that there is no one-size-fits-all strategy for constructing algorithms.
  - The strategy should be adapted to the specific problem, its complexity, and your problem-solving skills.

Algorithm Methodology

3. Implementation:

- **Finite Operations:**
  - The algorithm must execute within a finite number of operations.
- **Clear Specification:**
  - The algorithm must be specified clearly, leaving no room for ambiguity.
- **Data Type Specification:**
  - Specify the data types used within the algorithm (e.g., real numbers, integers).
- **Result Generation:**
  - The algorithm should produce at least one result, which could be in the form of output or display.
- **Human Simulatability:**
  - Every operation within the algorithm should be such that a human can simulate them within a finite time.
  - This ensures that the algorithm is understandable and practical.

Algorithm Construction Steps:

Creating an algorithm involves a structured series of steps:

1. **Variable Declaration:**
   - In this step, we detail the elements that the algorithm will use.
2. **Preparing for Processing:**
   - This phase involves initializing or inputting the data required for solving the problem.
3. **Processing:**
   - The heart of the algorithm lies in this step, where we perform the necessary operations to address the problem.
   - We break down complex problems into manageable steps and solve them one by one if needed.
4. **Results Presentation (Output):**
   - Results can be displayed on the screen, saved in a file, etc.

Algorithm Components:
Processors, Action & Environment

Processor:

- An Algorithm is always executed by a PROCESSOR.
- A PROCESSOR is any entity capable of understanding and executing the actions that make up an algorithm.
- A Processor can be a person, an electronic device, a mechanical device (such as a vending machine), or a computer.

Algorithm Components:
Processors, Action & Environment

Action :

- An ACTION is a <u>step</u> in the algorithm. It is a finite-duration event that <u>alters</u> the environment.
- A primitive action is an action that a processor can execute <u>without</u> any additional information.

Environnement

- The set of necessary objects and elements required to carry out a task described by an algorithm is called the Environment.

Algorithm Examples

In the era of Big Data, Machine Learning, and Artificial Intelligence (AI), **algorithms** have become increasingly pervasive in our daily lives.

Nowadays, we rely on **algorithms** not only for investing in stocks but also for personalizing content recommendations on streaming platforms, optimizing supply chains, and automating various processes.

Algorithms play a vital role in our economy and society, shaping decision-making, innovation, and the way we interact with technology and information.

Algorithm Examples

Cooking recipe...

In each recipe, a specific procedure must be followed in **sequence**. The various **steps** of the recipe represent the **operations** that make up the algorithm.

Let's take a Crepe recipe, for instance. In this recipe, a specific procedure must be followed in a particular **order**. Each **step** of the recipe corresponds to an **operation** that constitutes the algorithm.

Algorithm Examples

Cooking Crepe recipe

INPUT

List of Ingredients:

- 250 grams of sifted or all-purpose flour
- 4 eggs
- 450 milliliters of slightly warm milk
- …

PROCESS

Recipe Steps:

Start

- Melt the butter in the microwave and gently warm the milk, which should be barely lukewarm (this prevents lumps).
- Mix the sifted flour, sugar, and salt in a large bowl.
- …

End

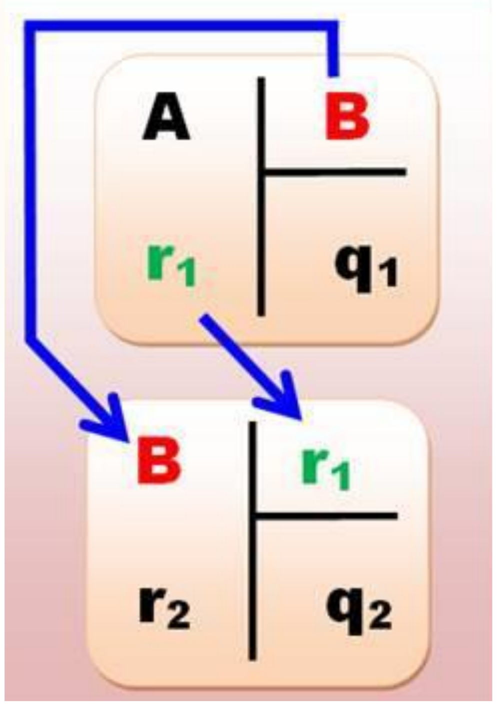
OUTPUT

Algorithm Examples

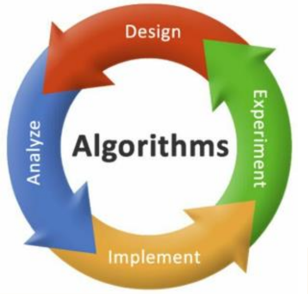
Finding the GCD

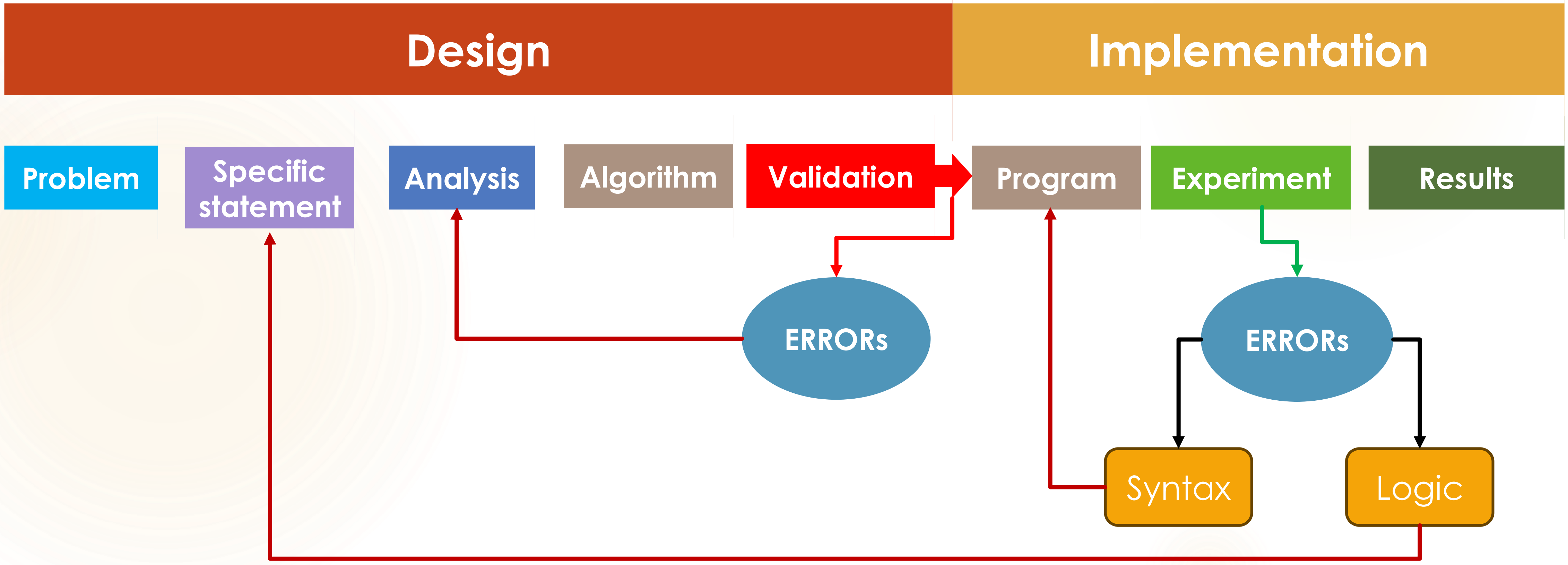
One of the oldest and most well-known algorithms is the Euclidean algorithm, which is used to compute the Greatest Common Divisor (GCD).

- This method involves a cascading series of divisions, where the results of one division are used as the basis for the next:
  - The divisor **B** becomes the new dividend.
  - The remainder **r_1** becomes the new divisor.
- The process continues until the division result is zero.
- The **remainder** obtained just before reaching zero is the **GCD** of the numbers **A** and **B.**

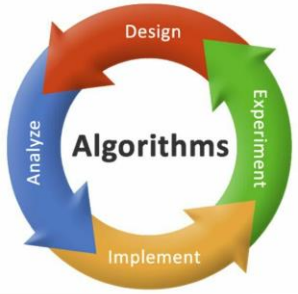


Resolution Process
From Problem to Result





Resolution Process
Analysis (1/2) 



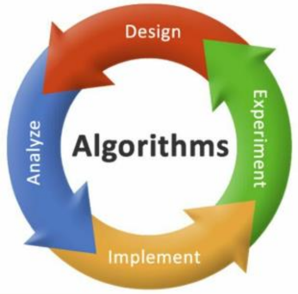
Analysis **MUST** be performed **PRIOR** to constructing the algorithm.
In this phase:

- You should outline the fundamental ideas.
- Express them with simple, concise, and clear sentences or through diagrams.
- Then, structure these ideas. Don't hesitate to apply your own conventions, such as diagrams, narratives, or colors.

→ The analysis should be modular and well-structured

Resolution Process

Analysis (2/2)



Take an example, well-chosen, and explain your idea through this example. Validate your analysis by systematically asking yourself the following questions:

1. Is the basic idea of my reasoning expressed correctly and simply?
2. Can I easily detect the control structures used in my analysis?
3. Are the ideas that stem from the basic idea well-structured?
4. Have I not delved into details that might make my analysis more confusing?

A well-structured analysis will prevent you from making logical errors and will enable you to construct your algorithm very easily.

Resolution Process
Manual Processing

- The process involves executing the algorithm, **step by step**, **manually**, to verify that it **produces** the desired **outcome**.
- We begin by creating a set of test cases, which are concrete and complete examples.
- To do this, we construct a table in which each column represents an object from our environment, and each row represents a step in our algorithm.
- The designer's behavior during this process can be both passive (observational) and active (engaging in the logic of operation).

Resolution Process
Example

- **Problem:**
  - Find the **list of divisors** of a number N (including N).

- **Precise Statement:**
  - Given an integer N, construct the computer solution that allows us to obtain the list of its divisors (including N).

Resolution Process
Example

Analysis

The questions one should always ask:

- What are we doing? → Answer: Finding the divisors of a number N.
- What is a divisor? → Answer: A divisor (or factor) of an integer N is a number that divides N without a remainder (the remainder is equal to zero).
- How many times? → from 1 to N/2 (to avoid unnecessary divisions), and in the end, also display N (which is a divisor of itself)

If the number of iterations is not known, find the termination condition.

Resolution Process
Example

Analysis …

Let N be an integer.

- We will successively divide N by i = 1, 2, 3, …, N/2.
  - **If the remainder of N divided by i is equal to 0** (**then i is a divisor**),
    - We print i.
- We display N (N is a divisor of itself).

Resolution Process

Example

```
// Algorithm to calculate the divisors of N
Variables N, i: INTEGER
BEGIN
     WRITE ('Enter a number N: ‘)
     READ (N)
     WRITE ('The divisors of: ', N)
     FOR i <- 1 TO (N DIV 2) DO
     BEGIN
          IF (N MOD i = 0) THEN
          BEGIN
               WRITE(i)
          END
     END
     WRITE(N, ' is also a divisor of itself')
END
```

Resolution Process
Example

```
#include <iostream>
int main() {
    int N, i;
    std::cout << "Enter a number N : ";
    std::cin >> N;
    std::cout << “The divisor of : " << N << "\n";
    for (i = 1; i <= N / 2; i++) {
        if (N % i == 0) {
            std::cout << i << "\n";
        }
    }
    std::cout << N << " is also a divisor of itself\n";
    return 0;
}
```

Resolution Process
Example

Manual Processing

N	i	Reminder of N/i	Result
21	**1**	**0**	Display i (1)
	2	1	
	3	**0**	Display i (3)
	4	1	
	5	1	
	6	3	
	7	**0**	Display i (7)
	8	5	
	9	3	
	10	1	
			Display N (21)

Welcome | divisor.cpp

divisor.cpp > main()

Program name

```cpp
#include <iostream>
int main() {
    int N, i;
    std::cout << "Enter a number N : ";
    std::cin >> N;
    std::cout << "The divisor of :" << N << "\n";
    for (i = 1; i <= N / 2; i++) {
        if (N % i == 0) {
            std::cout << i << "\n";
        }
    }
    std::cout << N << " is also a divisor of itself\n";
    return 0;
}
```

Program

```
Enter a number N : 21
The divisor of :21
1
3
7
21 is also a divisor of itself
```

Results