Programming and Algorithms

ISEP / SWitCH_QA / DESOFT I

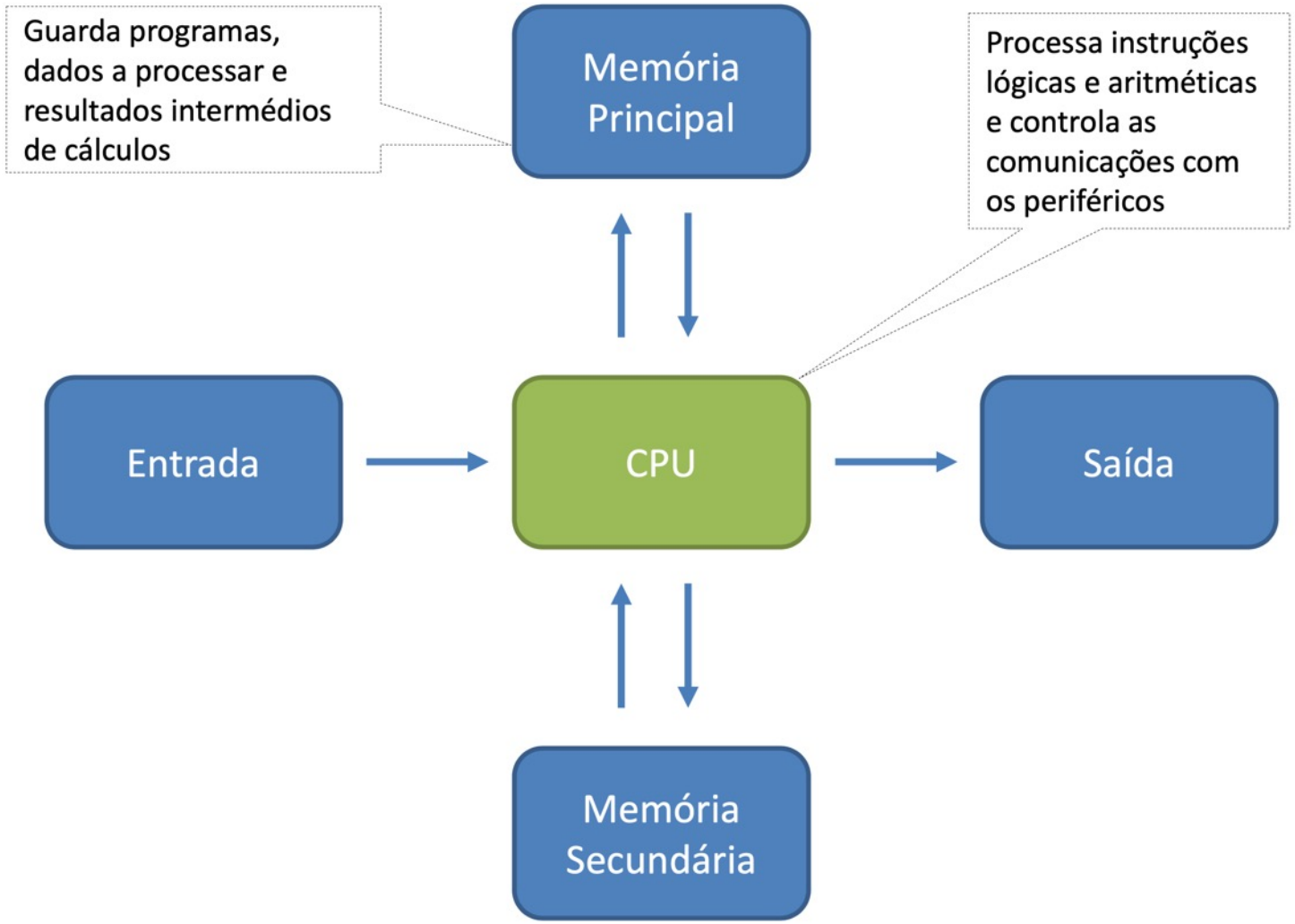
Some Definitions

- Informatics
  - Information + Automatic
  - Information + Mathematics
- Computer
  - Electronic-digital machine that executes algorithms taking into account the provided input

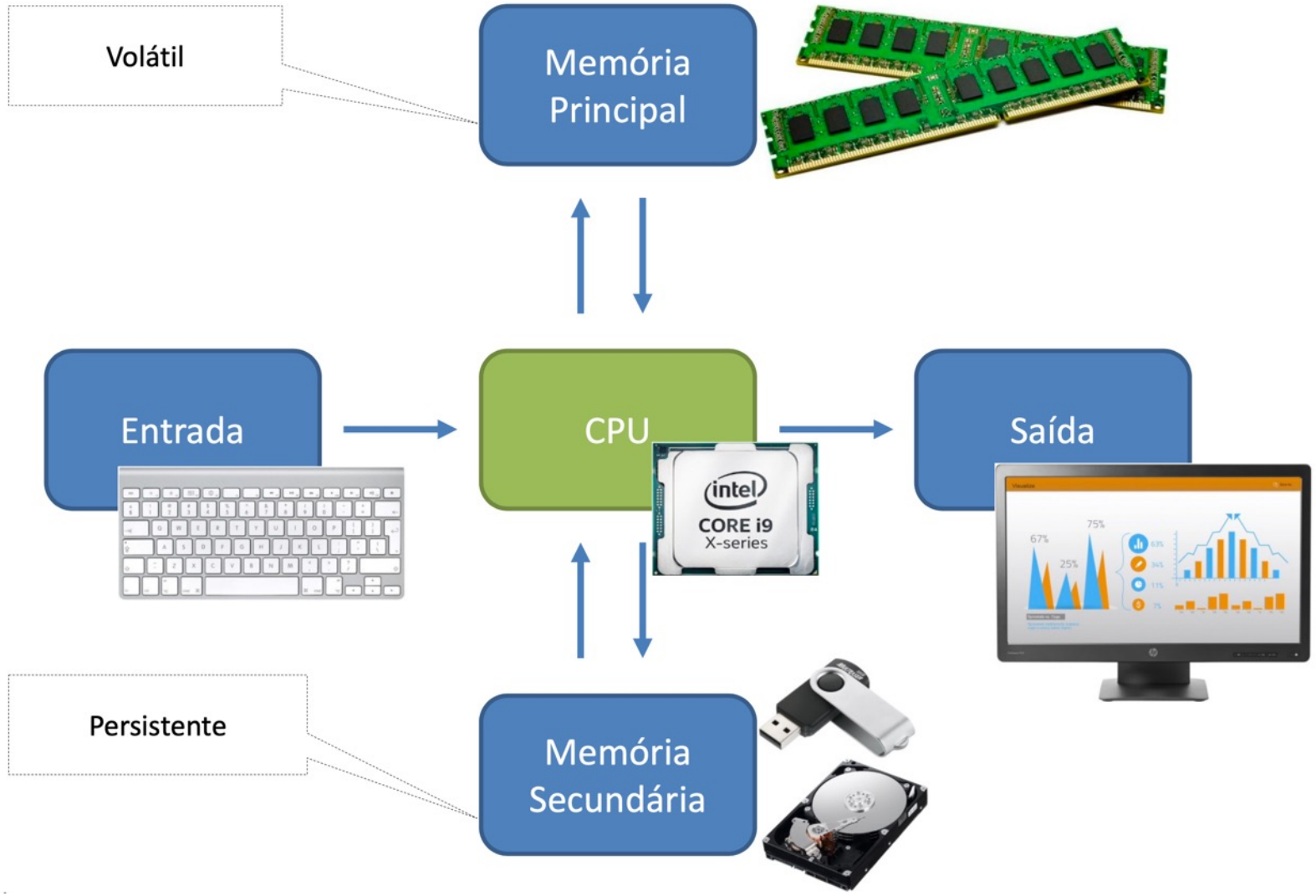
Hardware

Physical component of the computer: mechanical, electrical, electronic, magnetic components, …

Computer Main Components (1/2)



Computer Main Components (2/2)



Software

Computer logic component: Operating system, development tools, applications, …

Computer File

- Set of data grouped under a certain name that identifies it
- The OS is able to recognize and handle it
- E.g.
  - Relatorio.docx
  - Readme.md
  - MyFirstScript.ts

Operating System (OS)

- Allows the management of computer resources: memory, peripherals
- Perform Tasks
- It is a Special kind of program
- E.g.
  - Windows, Ubuntu, MacOS

Structured Programming

- Allows phasing the process of building a program by describing the computational process in an unambiguous way
- Relies on modular design and gradual refinement from top to bottom
- Defines a set of rules for creating programs

Program

- Based on Computer Files
- A Computer File that describes a specific task in a language known to the OS
- The OS is able to recognize the instructions and perform the task described in its contents
- According to this paradigm:
  - **Program = Data Structure + Algorithm**

Data Structures

- Represent how data is organized, accessed and changed
- Primitive types
  - Integers, real numbers, characters, Boolean (True or False)
  - Non-primitive (complex) types
- Complex Types (non-primitive)
  - Single and multidimensional indexed types
  - Lists, queues
  - Trees, Graphs

Algorithm

- Finite and unambiguous sequence of instructions that
- Describes the logical steps necessary to perform a given task or solve a problem
- Set of symbolic expressions that
  - represent actions (choose, assign, etc.)
  - test conditions (conditional structures)
  - control structures (cycles in the sequential structure of the algorithm)
- Specifies the problem and its solution
- It is represented through a language with associated syntax and semantics

Algorithm Representation (1/2)

- Natural language
  - Structured Portuguese, English, etc.
- Intermediate Language
  - Pseudocode
  - Applies natural language in conjunction with constructors used in programming languages
  - More accurate than natural language
- Graphical notation
  - Flowcharts

Algorithm Representation: Instructions (2/2)

Pseudocode	Flowchart	Natural Language
INÍCIO ou FIM		Início ou fim
LER()		Entrada de dados
ESCREVER()		Saída de resultados
SE...ENTÃO...SENÃO		Processamento
PARA...ATÉ...FAZER		Tomada de decisão
ENQUANTO...FAZER		módulo
FAZER...ENQUANTO		
PROCEDIMENTO/FUNÇÃO		

Algorithm Structure: Pseudocode

```
INÍCIO
   ED (Especificação de Dados):
   variavelUm, variavelDois: Integer
   variavelTres, variavelQuatro: Real

   LER (variavel1, variavel2)

   variavel3 ← variavel1 + variavel2

   ESCREVER (variavel3)
FIM
```

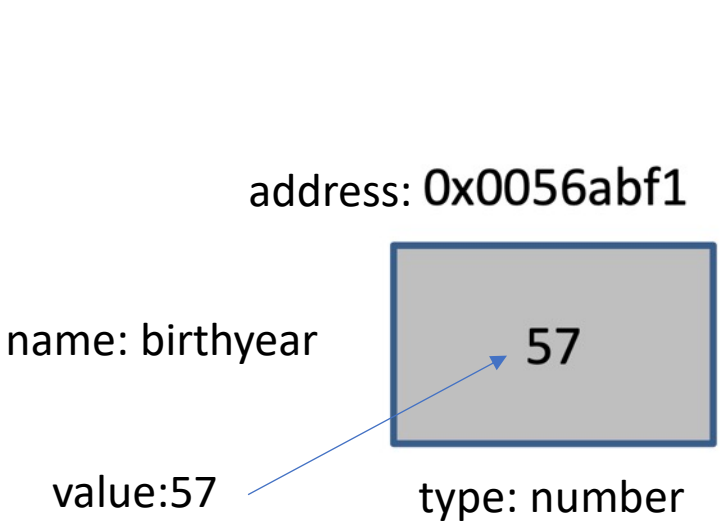
- (1) Declaring necessary data variables
- (2) reading data
- (3) processing
- (4) outputting results

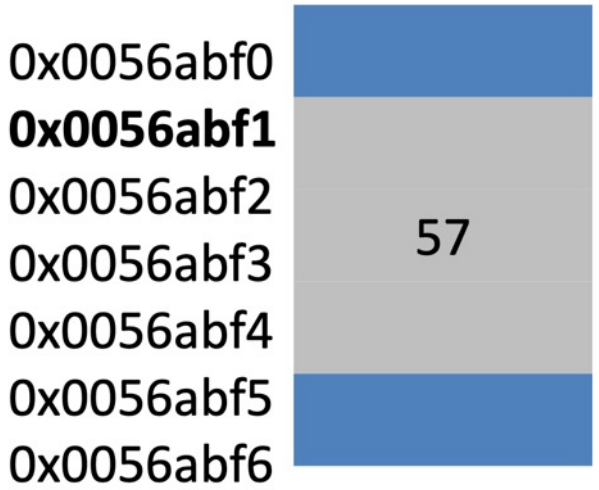
Problem Solving Technique

- Analyse the problem
  - Know the problem well
  - Describe the problem: subdivide, detail
- Solve the problem step-by-step
  - Check that there is no ambiguity in the presented solution
- Implement the solution
  - In a programming language

Variable

- Memory space which information of a given type can be accessed
- During the execution of the algorithm the value of a variable can be modified
- When a variable is declared and associated with a value/content, the following fundamental attributes are considered:
  - Name (suggestive)
  - Type
  - Value (content)
  - Address





Operator

- **Arithmetic Operators**
  - Multiplication, Division
    - *, /
  - Exponential
    - ^
  - Integer Division
    - DIV
  - Integer Division Remainder
    - MOD
  - Integer Division Remainder
    - %
  - Sum, Subtraction
    - +, -

- **Logical Operators**
  - Conjunction, Disjunction, Negation
    - AND, OR, NOT

- **Relational Operators**
  - Less, Than, Less of Equal Than
    - <, <=
  - Equal, not equal
    - =, !=
  - Higher Than, Equal or Higher Than
    - >, >=

- **Assignment Operator**
  - ←

Algorithm Control Structure

Sequence / Condition / Repetition

Algorithm Control Structure: Overview

- Sequence
  - Execute instructions sequentially, in the order they appear
- Condition
  - Choose whether or not to execute a set of instructions
- Repetition
  - Repeatedly execute a set of instructions according to a condition

Sequence Control Structure

Sequence Control Structure (1/3)

Demo #l01.01 [Integer Sum]: Given two integers, `a` and `b`, find the sum of `a` and `b`.

Sequence: (1) Variable Declaration; (2) Read Numbers; (3) Sum Numbers; (4) Show Sum

INICIO

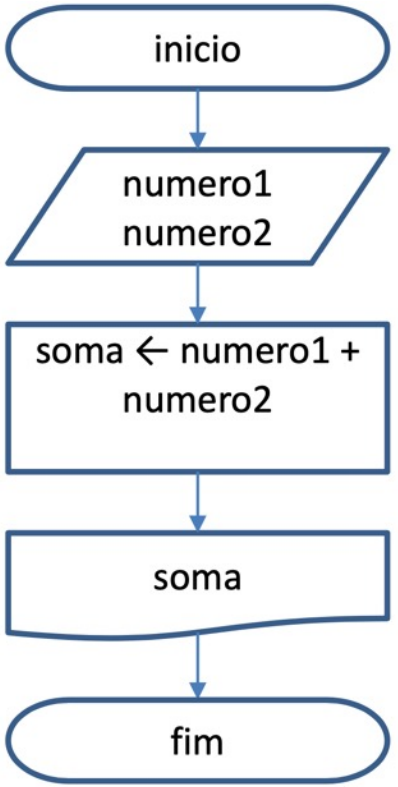
ED: numero1, numero2, soma INTEIRO

LER (numero1, numero2)

soma ← numero1 + numero2

ESCREVER (soma)

FIM



Sequence Control Structure (2/3)

Demo #l01.01 [Integer Sum]: Given two integers, `a` and `b`, find the sum of `a` and `b`.

Sequence: (1) Variable Declaration; (2) Read Numbers; (3) Sum Numbers; (4) Show Sum

```
INICIO
   ED:  numero1, numero2
        soma INTEIRO

   LER (numero1, numero2)

   soma ← numero1 + numero2

   ESCREVER (soma)
FIM
```

```
INICIO
   ED:  numero1, numero2 INTEIRO
        soma INTEIRO

   LER (numero1)

   LER (numero2)

   ESCREVER (numero1 + numero2)
FIM
```

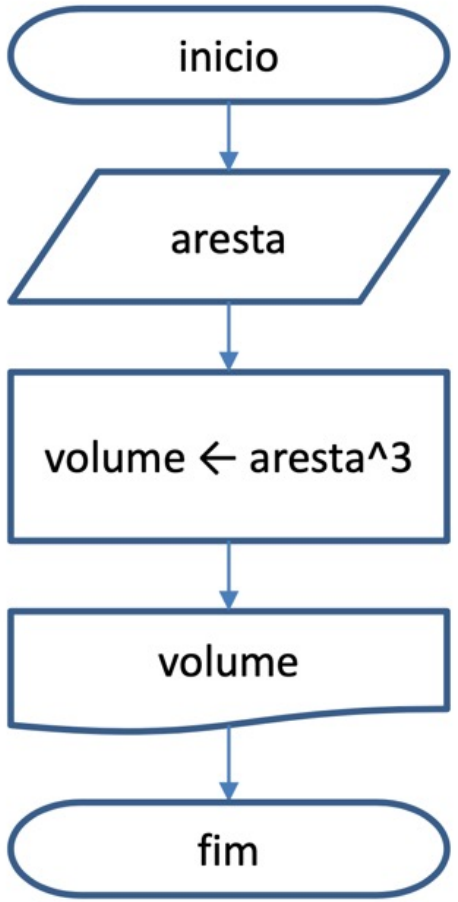
Sequence Control Structure (3/3)

Demo #l01.02 [Cube Volume]: Calculate a cube volume given its side.

Pseudocódigo

```
INICIO
    ED: aresta, volume INTEIRO
    LER (aresta)
    volume ← (aresta ^ 3)
    ESCREVER (volume)
FIM
```

Fluxograma



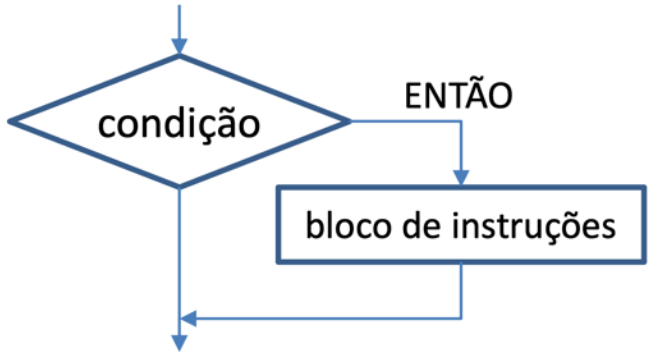
Condition Control Structure

Condition Control Structure: Overview

SE (condição) ENTÃO

<bloco de instruções>

FIMSE



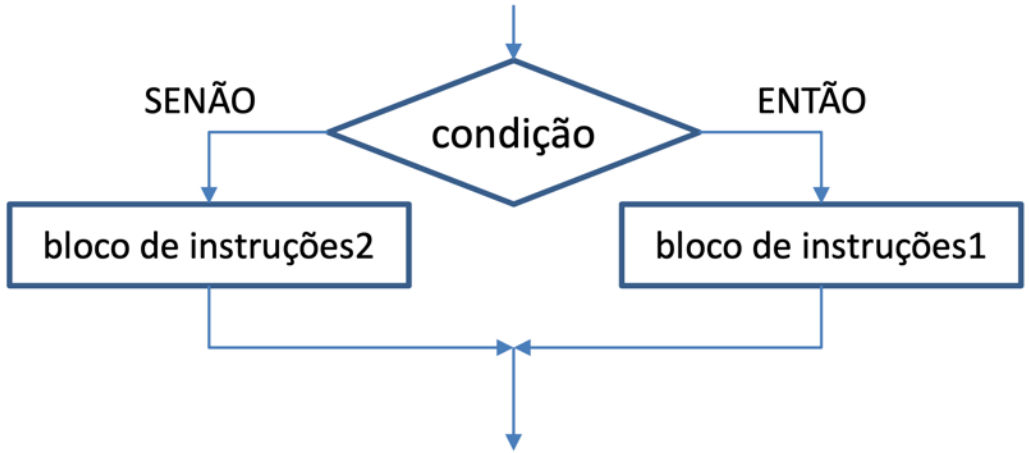
SE (condição) ENTÃO

<bloco de instruções1>

SENÃO

<bloco de instruções2>

FIMSE



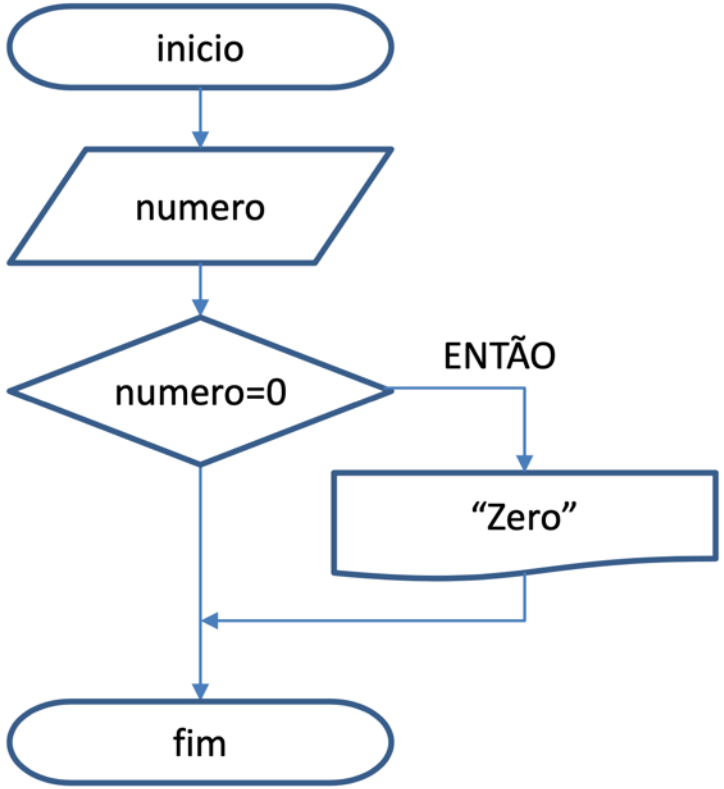
Condition Control Structure (1/2)

Demo #l01.03 [Display Zero]: Display “Zero” when a given number is Zero.

Pseudocódigo

```
INICIO
        ED: numero INTEIRO
        LER (numero)
        SE (numero = 0) ENTÃO
                ESCREVER (“Zero”)
        FIMSE
FIM
```

Fluxograma



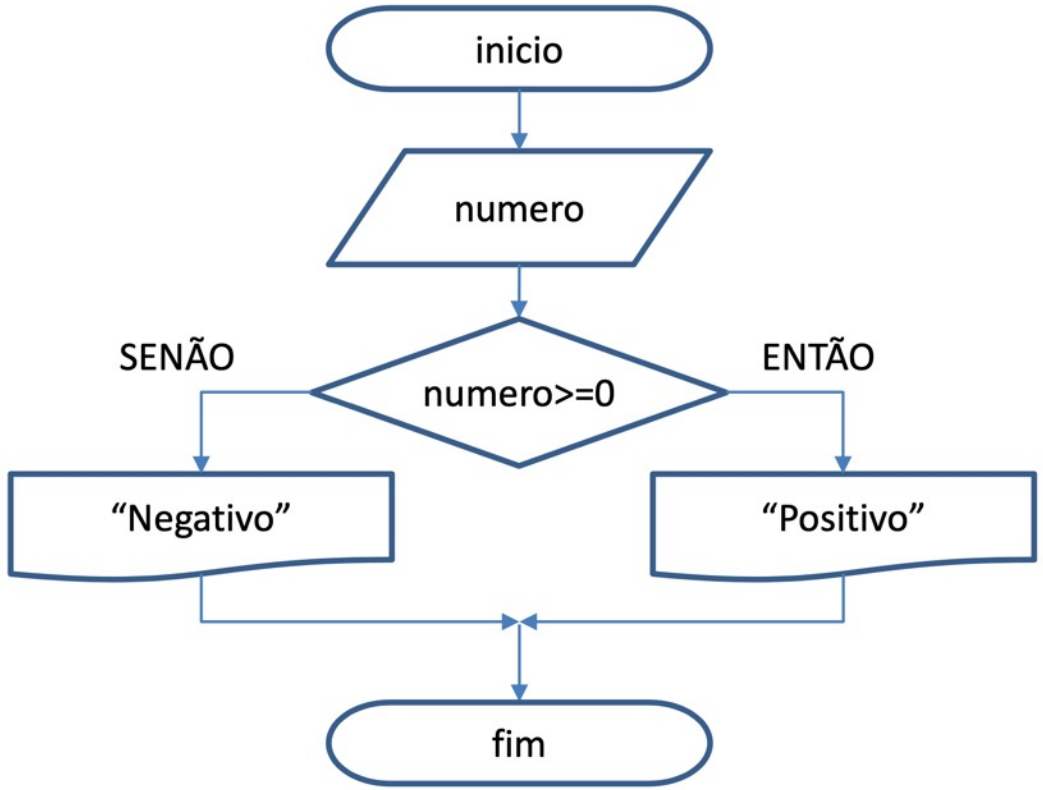
Condition Control Structure (2/2)

Demo #l01.04 [Display Positive or Negative]: Display “Positivo” or “Negativo” according to the given number.

Pseudocódigo

```
INICIO
    ED: numero INTEIRO
    LER (numero)
    SE (numero >= 0) ENTÃO
        ESCREVER (“Positivo”)
    SENÃO
        ESCREVER (“Negativo”)
    FIMSE
FIM
```

Fluxograma

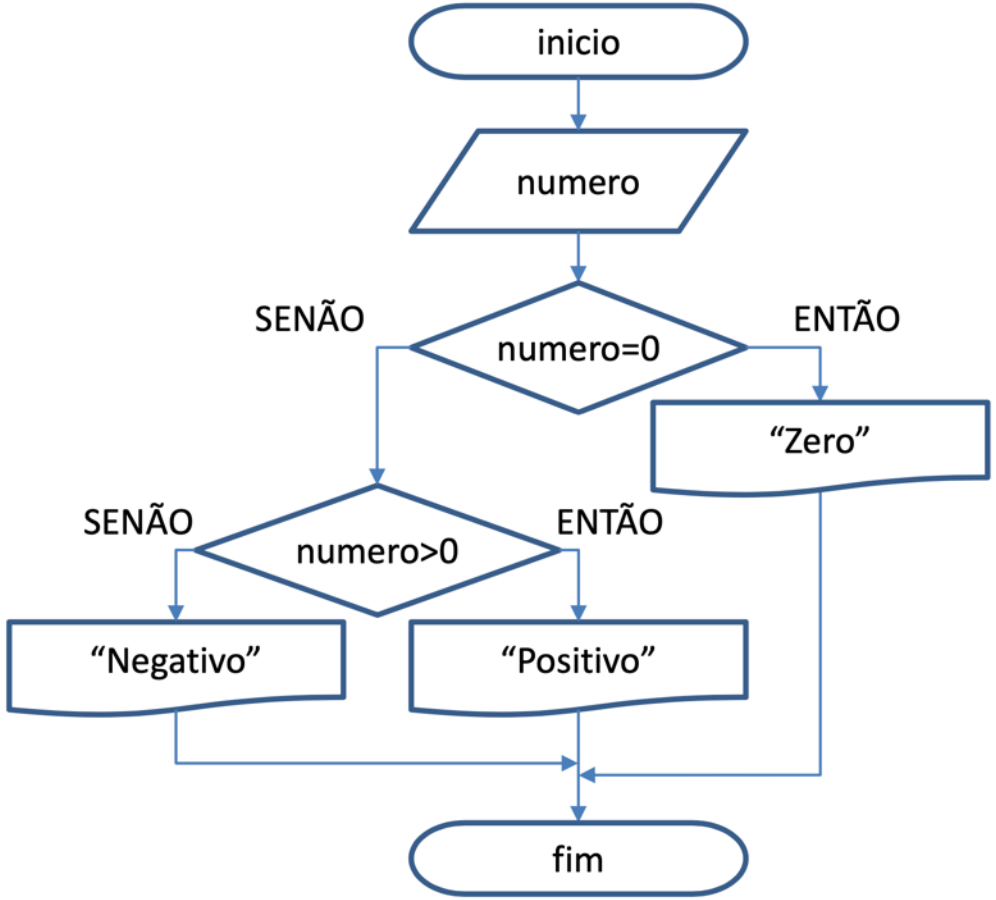


Nested Condition Control Structure

Nested Condition Control Structure

Demo #l01.05 [Display Positive, Negative or Zero]: Display “Positivo” or “Negativo” or “Zero” according to the given number.

```
INICIO
    ED: numero INTEIRO
    LER (numero)
    SE (numero = 0) ENTÃO
        ESCREVER (“Zero”)
    SENÃO
        SE (numero > 0) ENTÃO
            ESCREVER (“Positivo”)
        SENÃO
            ESCREVER (“Negativo”)
        FIMSE
    FIMSE
FIM
```



Exercise: Classify Student Grade

Exercise #l01.01

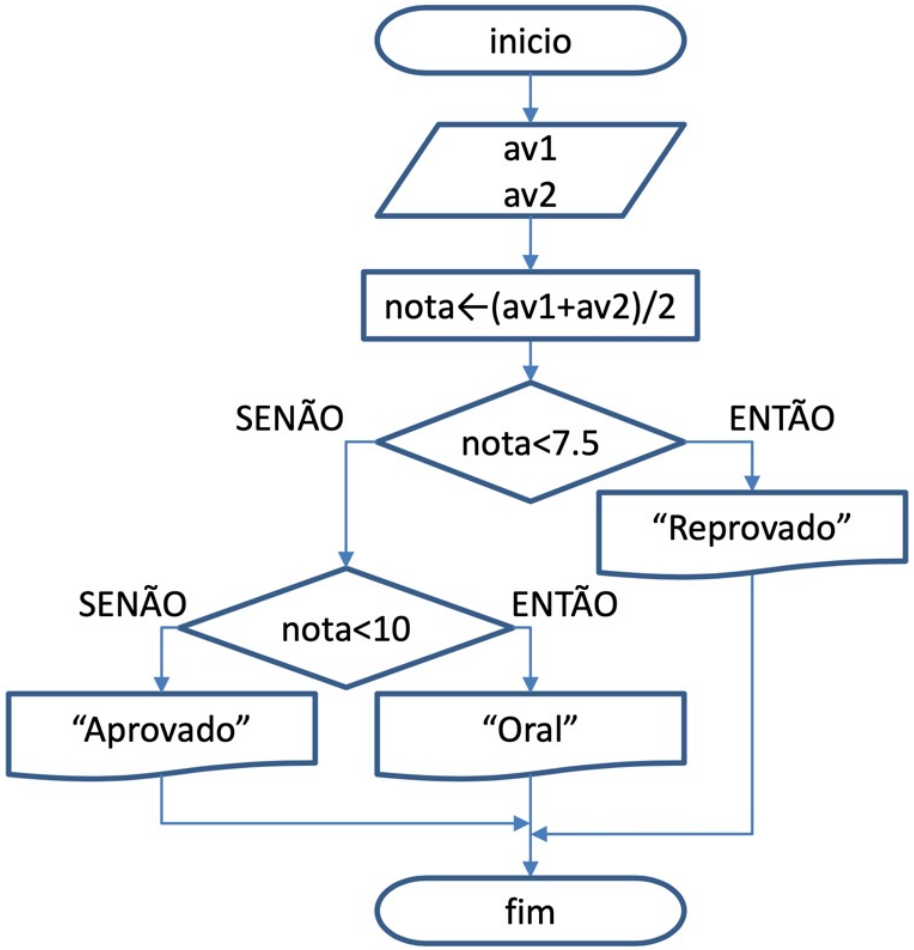
Nested Condition Control Structure: Exercise (1/3)

- Exercise: Write an algorithm capable of reading two values between 0 and 20 (related to the two assessment tests that a student has taken), classify and display the current situation of that student based on the following acceptance criteria:
  - The student's grade is obtained by averaging the two assessments' grades
  - According to the grade, the student's situation is:
    - "Reprovado" – if grade is below 7.5
    - "Oral" – if grade is equal or greater than 7.5 but less than 10
    - "Approved" – if grade is equal or greater than 10

Nested Condition Control Structure: Exercise (2/3)

Exercise #l01.01: Classify student grade (approach #1)

```
INICIO
    ED: av1, av2 INTEIRO
    ED: nota REAL
    LER (av1, av2)
    nota ← (av1 + av2) / 2
    SE (nota < 7.5) ENTÃO
        ESCREVER (“Reprovado”)
    SENÃO
        SE (nota < 10) ENTÃO
            ESCREVER (“Oral”)
        SENÃO
            ESCREVER (“Aprovado”)
        FIMSE
    FIMSE
FIM
```



Boolean Algebra

- Developed by English mathematician George Boole
- Used to describe circuits that can be built by combining logic gates, where variables can only have values 0 and 1 (False and True)
- There are only three operators AND, OR and NOT

E	*V*	*F*
V	V	F
F	F	F

OU	*V*	*F*
V	V	V
F	V	F

NAO	*V*	*F*
	F	V

X ← 8

X ← 8

X ← 8

(X > 5) E (X < 10) V
(X > 5) E (X < 7) F

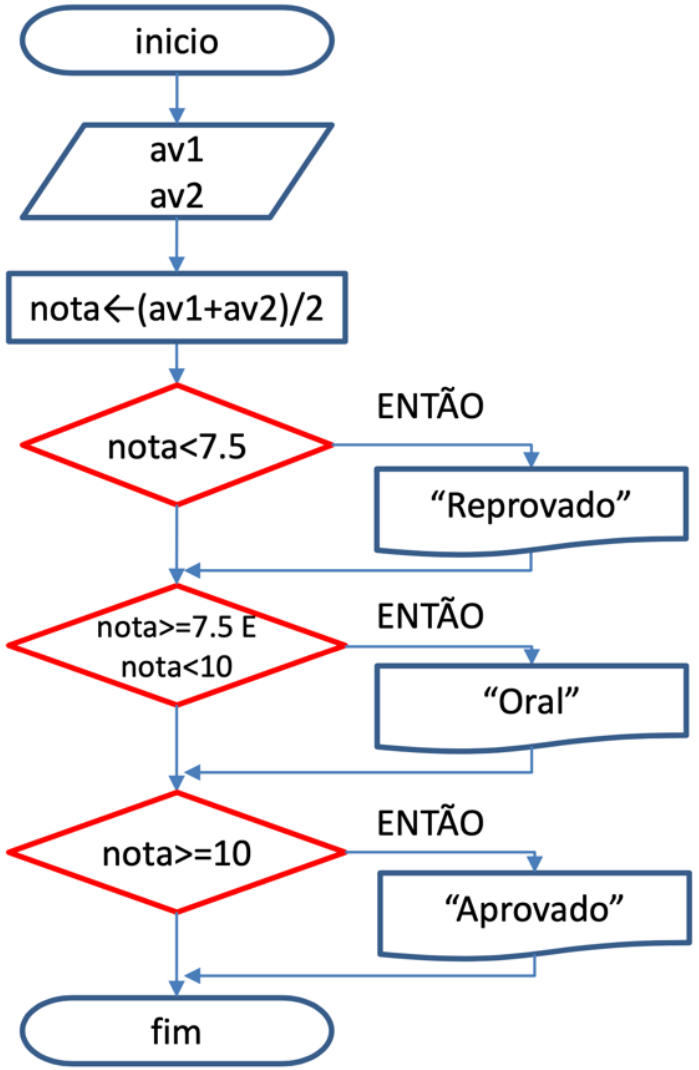
(X > 5) OU (X < 10) V
(X > 5) OU (X < 7) V
(X > 15) OU (X < 7) F

NÃO (X > 5) F
NÃO ((X > 15) OU (X < 7)) V

Nested Condition Control Structure: Exercise (3/3)

Exercise #l01.01: Classify student grade (approach #2)

```
INICIO
    ED: av1, av2 INTEIRO
    ED: nota REAL
    LER (av1, av2)
    nota ← (av1 + av2) / 2
    SE (nota < 7.5) ENTÃO
        ESCREVER (“Reprovado”)
    FIMSE
    SE (nota >= 7.5 E nota < 10) ENTÃO
        ESCREVER (“Oral”)
    FIMSE
    SE (nota >= 10) ENTÃO
        ESCREVER (“Aprovado”)
    FIMSE
FIM
```



Algorithm Tracing / Debugging

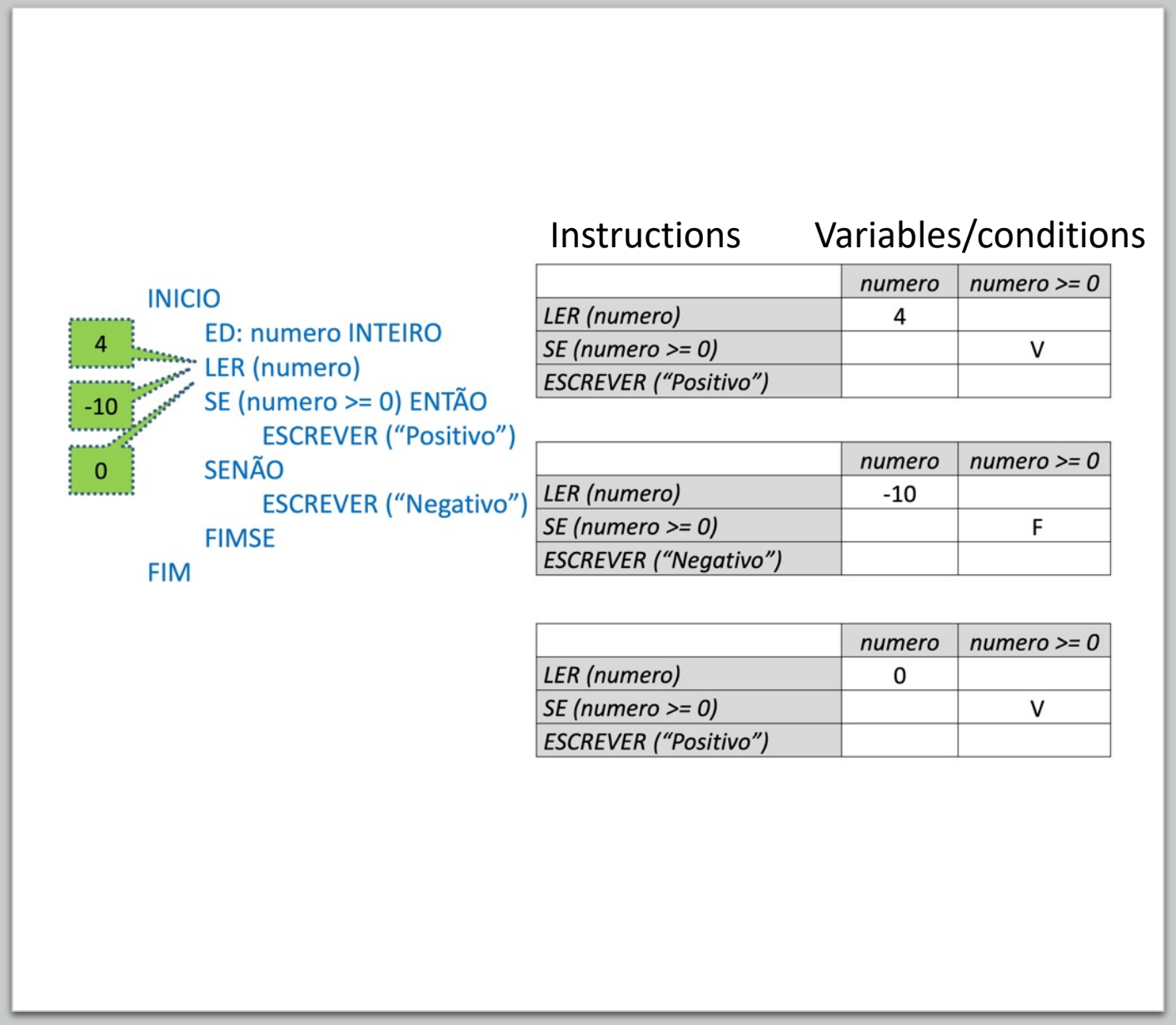
Performing manual tracing

Algorithm Tracing

- Check if the algorithm does what you is required
- Run it manually, simulating the steps
- Register the values of the variables and follow their evolution and changes

Algorithm Tracing

- Demo #l01.04 [Display Positive or Negative]: Display “Positivo” or “Negativo” according to the given number.



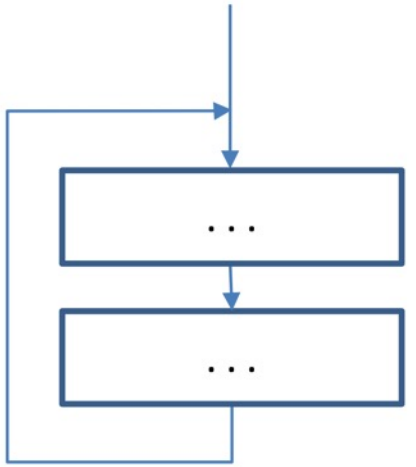
Repetition Control Structure

Repetition Control Structure: Overview

REPETIR ENQUANTO (*<condição>*)

    <bloco de instruções>

FIMREPETIR



REPETIR

    <bloco de instruções>

ENQUANTO (*<condição>*)

REPETIR PARA *<v>* ← *<vi>* ATE *<vf>* *PASSO* *<p>*

    <bloco de instruções>

FIMREPETIR

<v> : variável de controlo
<vi> : valor inicial
<vf> : valor final
<p> : valor do incremento de *<v>*

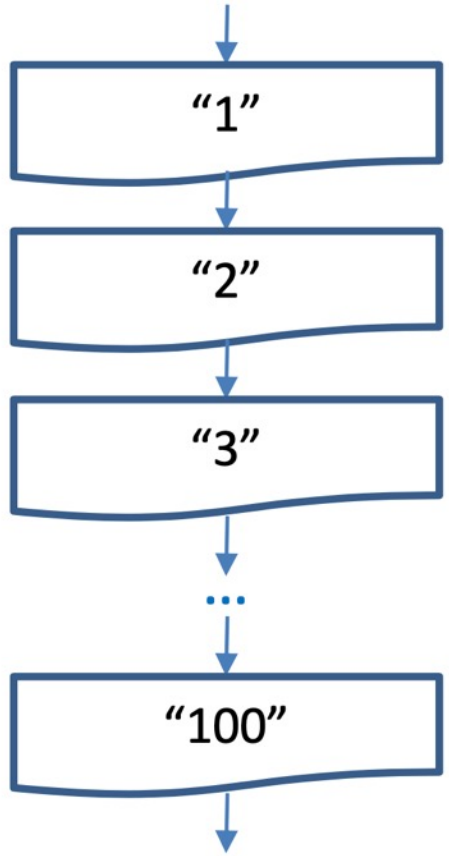
Repetition Control Structure: Example (1/5)

Display the numbers from 1 to 100

Repetition Control Structure: Example (2/5)

Display the numbers from 1 to 100

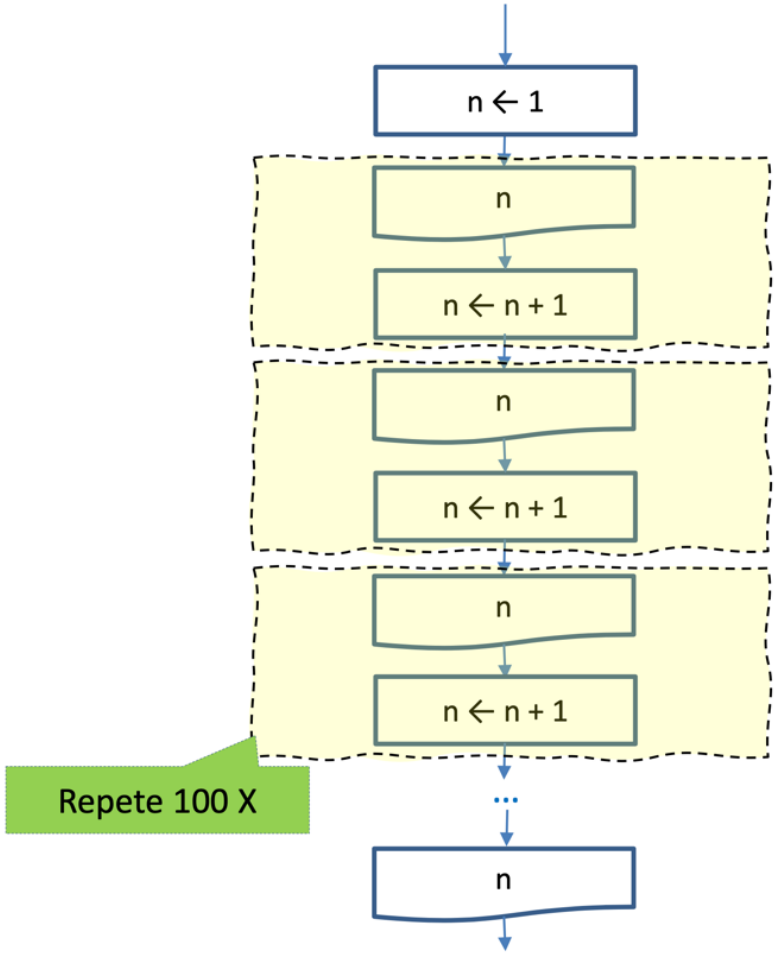
```
INICIO
    ESCREVER (“1”)
    ESCREVER (“2”)
    ESCREVER (“3”)
    ...
    ESCREVER (“100”)
FIM
```



Repetition Control Structure: Example (3/5)

Display the numbers from 1 to 100

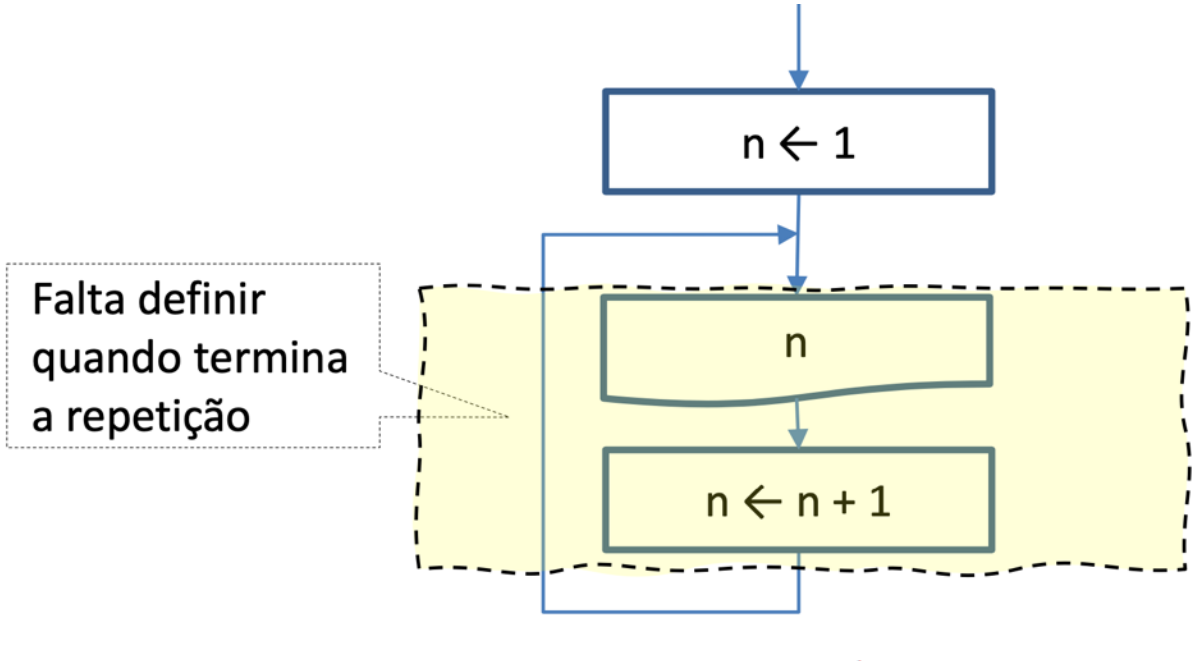
```
INICIO
    ED: n INTEIRO
    n ← 1
    ESCREVER (n)
    n ← n + 1
    ESCREVER (n)
    n ← n + 1
    ESCREVER (n)
    ...
    n ← n + 1
    ESCREVER (n)
FIM
```



Repetition Control Structure: Example (4/5)

Demo #l01.06 [Display numbers to 100]: Display the numbers from 1 to 100 (approach #01)

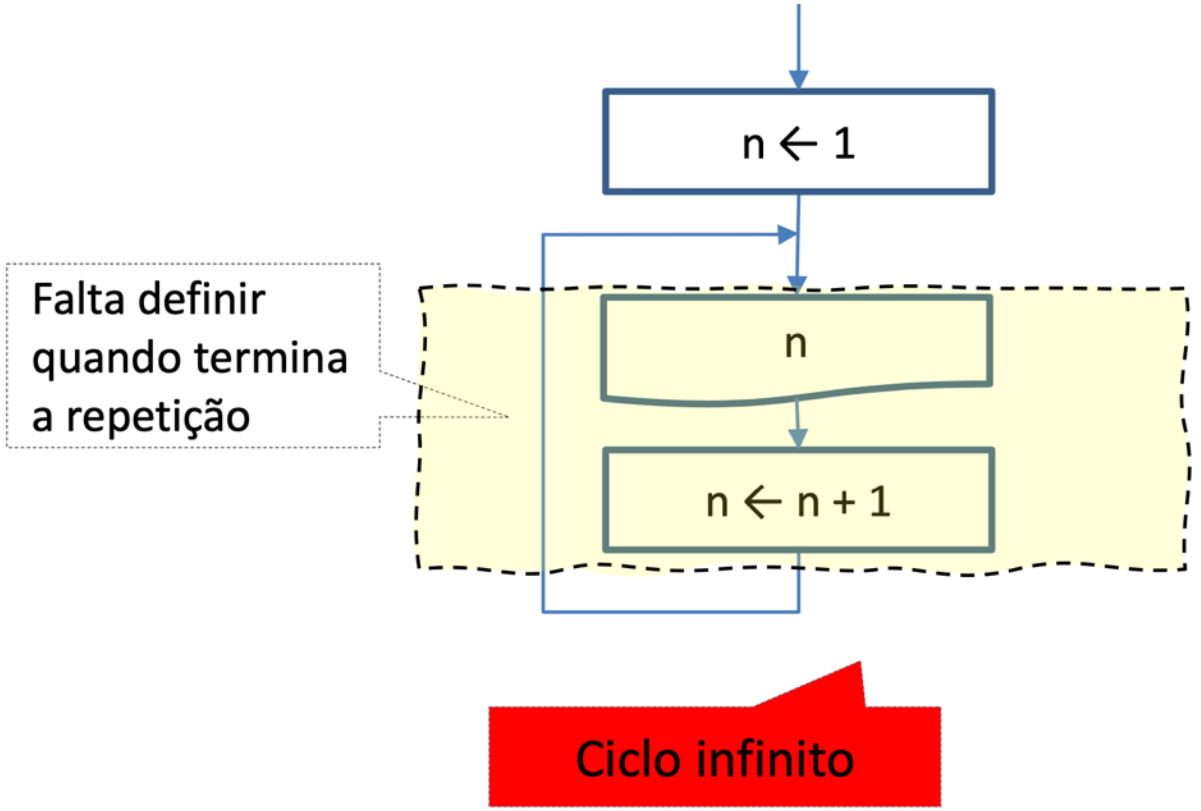
```
INICIO
    ED: n INTEIRO
    n ← 1
    REPETIR
        ESCREVER (n)
        n ← n + 1
    FIMREPETIR
FIM
```



Repetition Control Structure: Example (5/5)

Demo #l01.06 [Display numbers to 100]: Display the numbers from 1 to 100 (approach #01)

```
INICIO
    ED: n INTEIRO
    n ← 1
    REPETIR
        ESCREVER (n)
        n ← n + 1
    FIMREPETIR
FIM
```



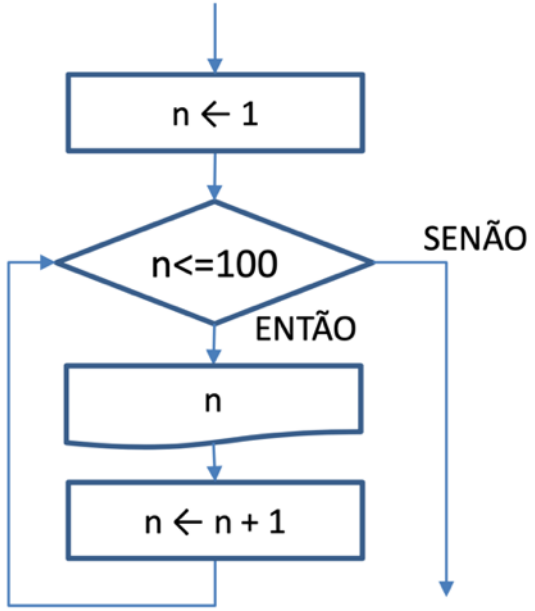
“Repetir Enquanto …” Control Structure

Demo #l01.06 [Display numbers to 100]: Display the numbers from 1 to 100 (approach #02)

```
REPETIR ENQUANTO (<condição>)
    <bloco de instruções>
FIMREPETIR
```

Problema1: Escrever os números inteiros de 1 a 100

```
INICIO
    ED: n INTEIRO
    n ← 1
    REPETIR ENQUANTO (n <= 100)
        ESCREVER (n)
        n ← n + 1
    FIMREPETIR
FIM
```



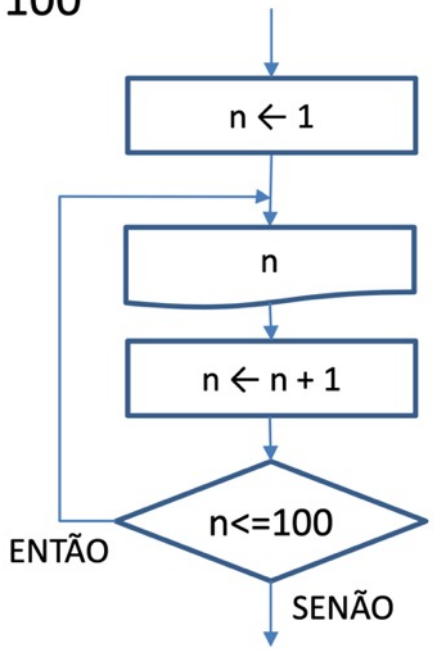
“Repetir … Enquanto” Control Structure

Demo #l01.06 [Display numbers to 100]: Display the numbers from 1 to 100 (approach #03)

```
REPETIR
    <bloco de instruções>
ENQUANTO (<condição>)
```

Problema1: Escrever os números inteiros de 1 a 100

```
INICIO
    ED: n INTEIRO
    n ← 1
    REPETIR
        ESCREVER (n)
        n ← n + 1
    ENQUANTO (n <= 100)
FIM
```



“Repetir Para …” Control Structure

Demo #l01.06 [Display numbers to 100]: Display the numbers from 1 to 100 (approach #04)

REPETIR PARA *<v>* ← *<vi>* ATE *<vf>* *PASSO* *<p>*

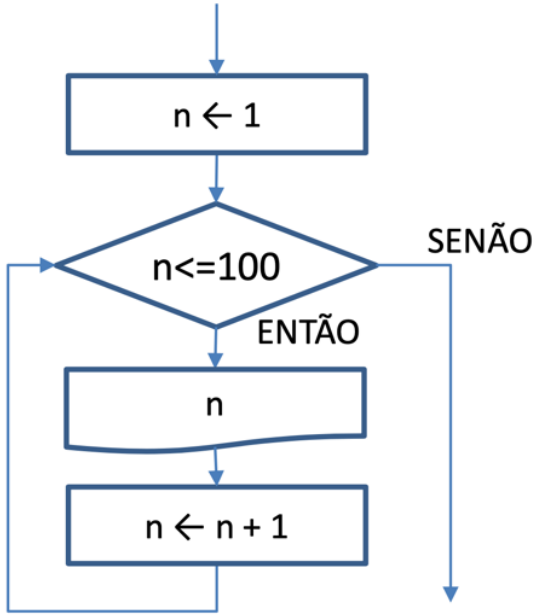
    <bloco de instruções>

FIMREPETIR

<v> : variável de controlo
<vi> : valor inicial
<vf> : valor final
<p> : valor do incremento de *<v>*

Problema1: Escrever os números inteiros de 1 a 100

```
INICIO
    ED: n INTEIRO
    REPETIR PARA n ← 1 ATE 100 PASSO 1
        ESCREVER (n)
    FIMREPETIR
FIM
```



Repetition Control Structure: Pitfalls

Repetition Control Structure Pitfalls

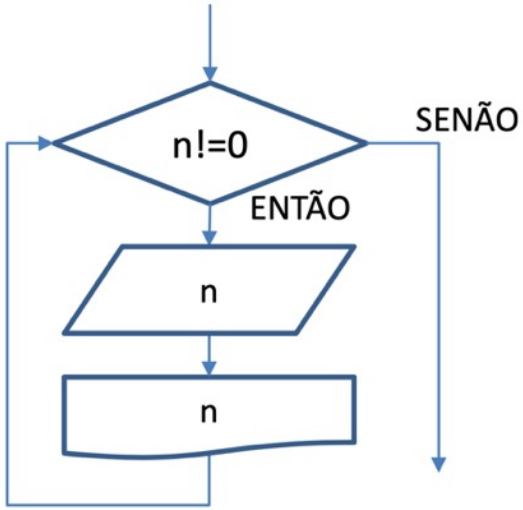
“Repetir Enquanto …” Control Structure: Pitfalls

Demo #l01.07 [Repeat While Pitfall]: Read and write given integer numbers while zero is not entered (approach #01).

```
REPETIR ENQUANTO (<condição>)
        <bloco de instruções>
FIMREPETIR
```

```
INICIO
        ED: n INTEIRO
        REPETIR ENQUANTO (n != 0)
                LER (n)
                ESCREVER (n)
        FIMREPETIR
FIM
```

Qual o valor de n ?

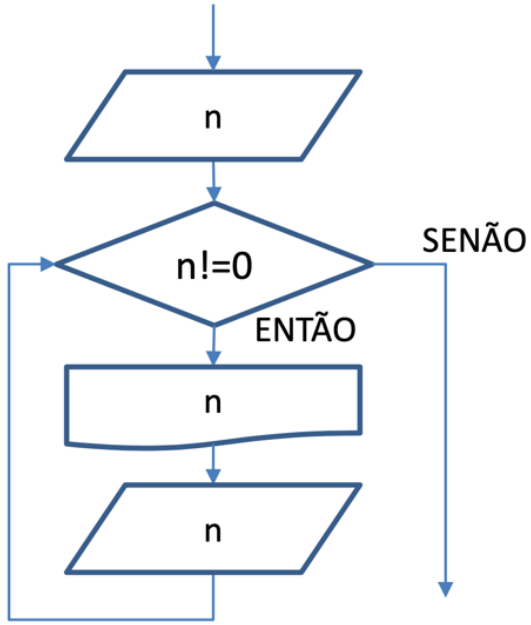


“Repetir Enquanto …” Control Structure: Pitfalls

Demo #l01.07 [Repeat While Pitfall]: Read and write given integer numbers while zero is not entered (approach #02).

```
REPETIR ENQUANTO (<condição>)
        <bloco de instruções>
FIMREPETIR
```

```
INICIO
        ED: n INTEIRO
        LER (n)
        REPETIR ENQUANTO (n != 0)
                ESCREVER (n)
                LER (n)
        FIMREPETIR
FIM
```



“Repetir ... Enquanto” Control Structure: Pitfalls

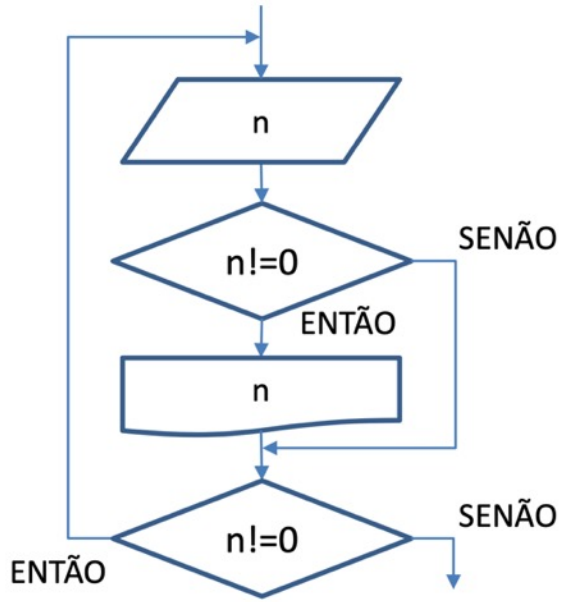
Demo #l01.08 [Repeat While Pitfall]: Read and write given integer numbers while zero is not entered (approach #01 and #02).

```
REPETIR
    <bloco de instruções>
ENQUANTO (<condição>)
```

```
INICIO
   ED: n INTEIRO
   REPETIR
      LER (n)
      ESCREVER (n)
   ENQUANTO (n != 0)
FIM
```

Não imprimir o zero

```
INICIO
   ED: n INTEIRO
   REPETIR
      LER (n)
      SE (n != 0 ) ENTAO
         ESCREVER (n)
      FIMSE
   ENQUANTO (n != 0)
FIM
```



“Repetir … Para” Control Structure Pitfalls

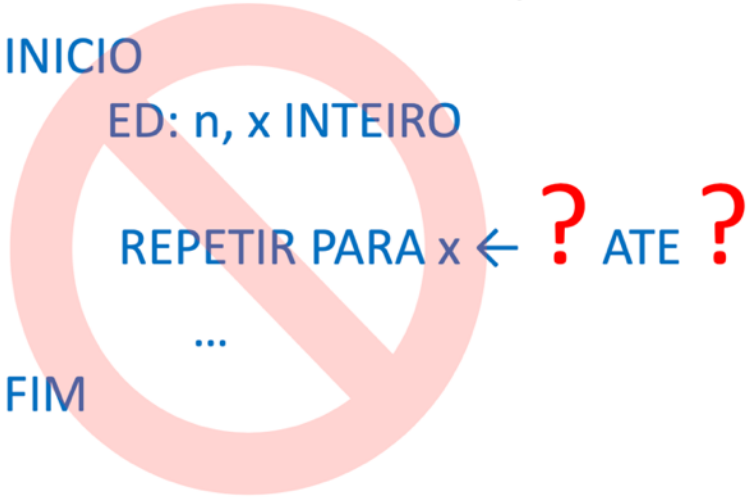
Demo #l01.09 [Repeat For Pitfall]: Read and write given integer numbers while zero is not entered (approach #01).

REPETIR PARA *<v>* ← *<vi>* ATE *<vf>* *PASSO* *<p>*

    <bloco de instruções>

FIMREPETIR

Problema2: Ler e escrever números inteiros enquanto não for inserido o número zero (não imprimir o zero).



```
INICIO
    ED: n, x INTEIRO

    REPETIR PARA x ← ? ATE ?
        ...
FIM
```

- Quantos nº vão ser introduzidos?
- Qual o valor inicial e o final?

Exercise: Daily Average Temperature

Exercise #l01.02

Nested Repetition Control Structure Example

Exercise #l01.02 [Daily Average Temperature]: Calculate the average daily temperature over the course of a week. Temperatures were recorded hourly for 7 days a week.

Nested Repetition Control Structure Example

Exercise #l01.02 [Daily Average Temperature]: Calculate the average daily temperature over the course of a week. Temperatures were recorded hourly for 7 days a week.

```
INICIO
    ED: dia, hora INTEIRO
    ED: temperatura, soma, media REAL
    REPETIR PARA dia ← 1 ATE 7 PASSO 1
        soma ← 0
        REPETIR PARA hora ← 1 ATE 24 PASSO 1
            LER (temperatura)
            soma ← soma + temperatura
        FIMREPETIR
        media ← soma / 24)
        ESCREVER (media)
    FIMREPETIR
FIM
```

Para cada dia da semana

para cada hora do dia

Recursion

Factorial using “Repetir Para ...”

Demo #l01.10 [Factorial]: Calculate the factorial of a number entered by the user. n! = 1 * 2 * 3 * ... * (n-1) * n (approach #01)

Factorial using “Repetir Para ...”

Demo #l01.10 [Factorial]: Calculate the factorial of a number entered by the user. n! = 1 * 2 * 3 * ... * (n-1) * n (approach #01)

```
INICIO
    ED: x, n, fatorial INTEIRO
    LER (n)
    fatorial ← 1
    REPETIR PARA x ← 1 ATE n PASSO 1
        fatorial ← fatorial * x
    FIMREPETIR
    ESCREVER (fatorial)
FIM
```

ou (x ← 2)

Factorial using “Repetir Enquanto …”

Demo #l01.10 [Factorial]: Calculate the factorial of a number entered by the user. n! = 1 * 2 * 3 * ... * (n-1) * n (approach #02)

```
INICIO
   ED: x, n, fatorial INTEIRO
   LER (n)
   fatorial ← 1
   x←2

   REPETIR ENQUANTO (x <= n)
      fatorial ← fatorial * x
      x←x+1
   FIMREPETIR
   ESCREVER (fatorial)
FIM
```

Tracing

Input = 4

```
INICIO
   ED: x, n, fatorial INTEIRO LER (n)
   fatorial ← 1
   x←2

   REPETIR ENQUANTO (x <= n)
      fatorial ← fatorial * x
      x←x+1
   FIMREPETIR
   ESCREVER (fatorial)
FIM
```

	n	fatorial	x	x<=n
LER (n)	4			
fatorial ← 1		1		
x ← 2			2	
REPETIR ENQUANTO (x <= n)				V
fatorial ← fatorial * x		2		
x ← x + 1			3	
REPETIR ENQUANTO (x <= n)				V
fatorial ← fatorial * x		6		
x ← x + 1			4	
REPETIR ENQUANTO (x <= n)				V
fatorial ← fatorial * x		24		
x ← x + 1			5	
REPETIR ENQUANTO (x <= n)				F
ESCREVER (fatorial)		24		

Exercise: Split Number

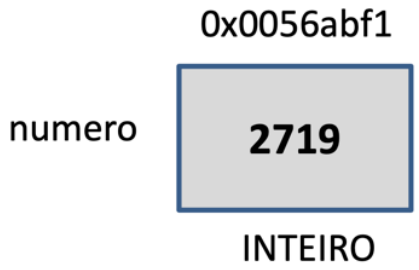
Exercise #l01.03

Split a number exercise (1/4)

Exercise #l01.03 [Split Number]: Write in sequence the digits of units, tens, ... of a whole number.

```
INICIO
    ED: numero INTEIRO
    LER(numero)
    ...
```

Como separar os algarismos?

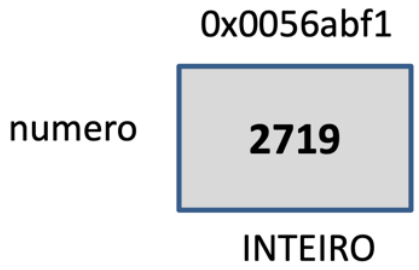


Split a number exercise (2/4)

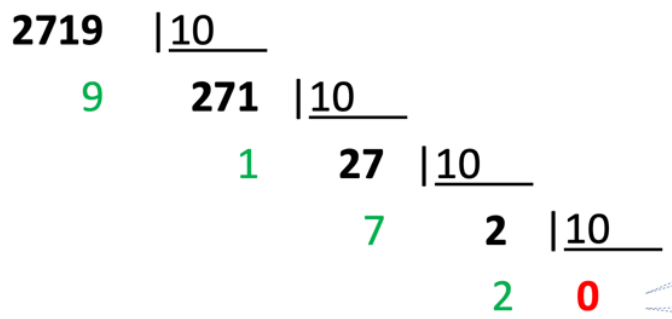
Exercise #l01.03 [Split Number]: Write in sequence the digits of units, tens, ... of a whole number.

```
INICIO
    ED: numero INTEIRO
    LER(numero)
    ...
```

Como separar os algarismos?



Executar divisões inteiras por 10, ou seja, dividir sempre por 10 sem usar casas decimais

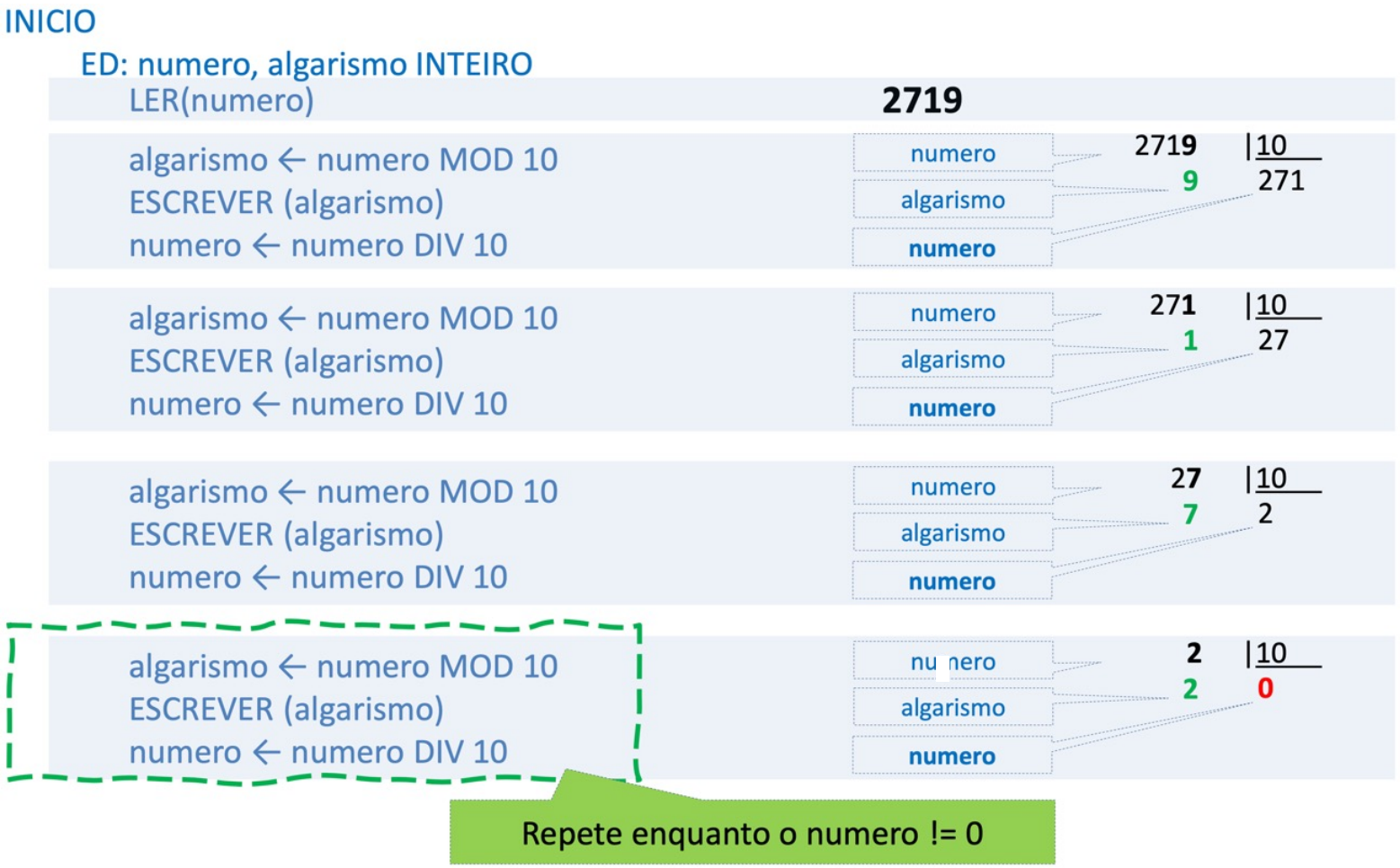


Pára de dividir quando o numero chegar a zero

Não há mais algarismos

Split a number exercise (3/4)

Exercise #l01.03 [Split Number]: Write in sequence the digits of units, tens, ... of a whole number.



Split a number exercise (4/4)

```
INICIO
   ED: numero, algarismo INTEIRO

   LER(numero)

   REPETIR
      algarismo ← numero MOD 10
      ESCREVER (algarismo)
      numero ← numero DIV 10
   ENQUANTO (numero!=0)
FIM
```

	numero	algarismo	numero!= 0	*output*
LER (n)	453			
algarismo ← numero MOD 10		3		
ESCREVER (algarismo)				3
numero ← numero DIV 10	45			
ENQUANTO (numero!=0)			V	
algarismo ← n MOD 10		5		
ESCREVER (algarismo)				5
numero ← numero DIV 10	4			
ENQUANTO (numero!=0)			V	
algarismo ← numero MOD 10		4		
ESCREVER (algarismo)				4
numero ← numero DIV 10	0			
ENQUANTO (numero!=0)			F	

Remember
DIV and MOD?

Exercise: Digit Count

Exercise #l01.04

Digit Count (1/2)

- Exercise #l01.04 [Digit Count]: Count the digits of a given number.

- Example 1:
  - Input: 453
  - Output: 3

- Example 2:
  - Input: 5
  - Output: 1

Digit Count (2/2)

Exercise #l01.04 [Digit Count]: Count the digits of a given number. (approach #01)

```
INICIO
   ED: n, qtd INTEIRO
   LER(n)
   qtd ← 0
   REPETIR
      qtd ← qtd + 1
      n ← n DIV 10
   ENQUANTO (n!=0)
   ESCREVER (qtd)
FIM
```

	n	qtd	n!= 0	*output*
LER (n)	453			
qtd ← 0		0		
qtd ← qtd + 1		1		
n ← n DIV 10	45			
ENQUANTO (n!=0)			V	
qtd ← qtd + 1		2		
n ← n DIV 10	4			
ENQUANTO (n!=0)			V	
qtd ← qtd + 1		3		
n ← n DIV 10	0			
ENQUANTO (n!=0)			F	
ESCREVER (qtd)				3

Exercise: Calculate Check Digit

Exercise #l01.05

Check Digit

- Exercise #l01.05 [Check Digit]: Calculate a given number's check digit.

- "A check digit is a form of redundancy check used for error detection on identification numbers"
  - https://en.wikipedia.org/wiki/Check_digit
  - UPC (Universal Product Code)
  - ISBN (International Standard Book Number)
  - EAN (European Article Number)
  - IMEI (International Mobile Equipment Identity)
  - …

Calculate Luhn Algorithm Definition

- The Luhn algorithm or Luhn formula, also known as the "modulus 10" or "mod 10" algorithm, named after its creator, IBM scientist Hans Peter Luhn, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, etc..”
  - https://en.wikipedia.org/wiki/Luhn_algorithm

Calculate Luhn Algorithm Sequence

1. From the rightmost digit and moving left, double the value of every digit in odd positions.
   - If the result of this doubling operation is greater than 9 (e.g., 8 × 2 = 16), then add the digits of the result (e.g., 16: 1 + 6 = 7, 18: 1 + 8 = 9) or, alternatively, the same final result can be found by subtracting 9 from that result (e.g., 16: 16 − 9 = 7, 18: 18 − 9 = 9).
2. Take the sum of all the digits.
3. The check digit (x) is obtained by computing the sum of the non-check digits then computing 9 times that value modulo 10.

Calculate Luhn Algorithm Example (1/2)

Luhn Algorithm

1. From the rightmost digit and moving left, double the value of every digit in odd positions. If the result of this doubling operation is greater than 9 (e.g., 8 × 2 = 16), then add the digits of the result (e.g., 16: 1 + 6 = 7, 18: 1 + 8 = 9) or, alternatively, the same final result can be found by subtracting 9 from that result (e.g., 16: 16 − 9 = 7, 18: 18 − 9 = 9).

2. Take the sum of all the digits.

3. The check digit (x) is obtained by computing the sum of the non-check digits then computing 9 times that value modulo 10.

Assume an example of an account number "7992739871" that will have a check digit added, making it of the form 7992739871x:

Account number	7	9	9	2	7	3	9	8	7	1	**x**
Double every other	7	**18**	9	**4**	7	**6**	9	**16**	7	**2**	**x**
Sum digits	7	**9**	9	4	7	6	9	**7**	7	2	**x**

Calculate Luhn Algorithm Example (2/2)

1. The sum of all the digits in the third row is 67 + x.
2. Compute the sum of the non-check digits (67).
3. Multiply by 9 (67 x 9 = 603).
4. The check digit (x) is obtained by computing the sum of the non-check digits then computing 9 times that value modulo 10 (603 mod 10 = 3).
5. The units digit (3) is the check digit. Thus, x=3.

Account number	7	9	9	2	7	3	9	8	7	1	**x**
Double every other	7	**18**	9	**4**	7	**6**	9	**16**	7	**2**	**x**
Sum digits	7	**9**	9	4	7	6	9	**7**	7	2	**x**

Calculate Luhn Algorithm: Pseudocode

```
INICIO
    ED: numero, posicao, soma, digito, dobro INTEIRO
    LER(numero)
    posicao ← 0
    soma ← 0
    REPETIR ENQUANTO (numero != 0)          pára quando o numero chegar a zero
        digito ← numero MOD 10
        numero ← numero DIV 10              retira o algarismo à direita no número
        posicao ← posicao + 1
        SE (posicao MOD 2 != 0) ENTAO
            dobro ← digito * 2
            SE(dobro > 9) ENTAO
                dobro ← dobro - 9
            FIMSE                           processa com base no algarismo
            soma ← soma + dobro
        SENAO
            soma ← soma + digito
        FIMSE
    FIMREPETIR
    ESCREVER("check digit=", (soma * 9) MOD 10)
FIM
```

Summary

- So far,
- We've understood what that software is a sequence of a set of instructions that solve problems.
- There are some "paper" exercises for you to do on this week
- Next week, we'll see how to translate those instructions into computer code using TypeScript.

Bibliography

- Adapted from LEI/APROG's, LEI/ESOFT's & SWITCH/DESOFT's slide decks
- "Programação, algoritmos e estruturas de dados"; João Pedro Neto; Escolar Editora