

Algoritmia e Programação

Introdução à algoritmia

Marta Martinho 2025

Organização da apresentação

- Introdução à algoritmia;
- Notações para representação de algoritmos;
- Tipos de instruções;
- Tipos de dados;
- Operadores matemáticos;

Definição de algoritmo

“Um algoritmo representa uma *sequência finita e não ambígua* de instruções de modo a obter a resolução de um problema sob a forma de resultado (*saída de dados*) tendo por base uma prévia *entrada de dados*.

Um algoritmo é representado através de uma linguagem com uma determinada *sintaxe e semântica* associada.

Por fim, um algoritmo deve ser *eficaz* na resolução do problema subjacente assim como *eficiente* para resolver o problema com o melhor desempenho (performance) possível.”

in “Algoritmia e Estruturas de Dados”

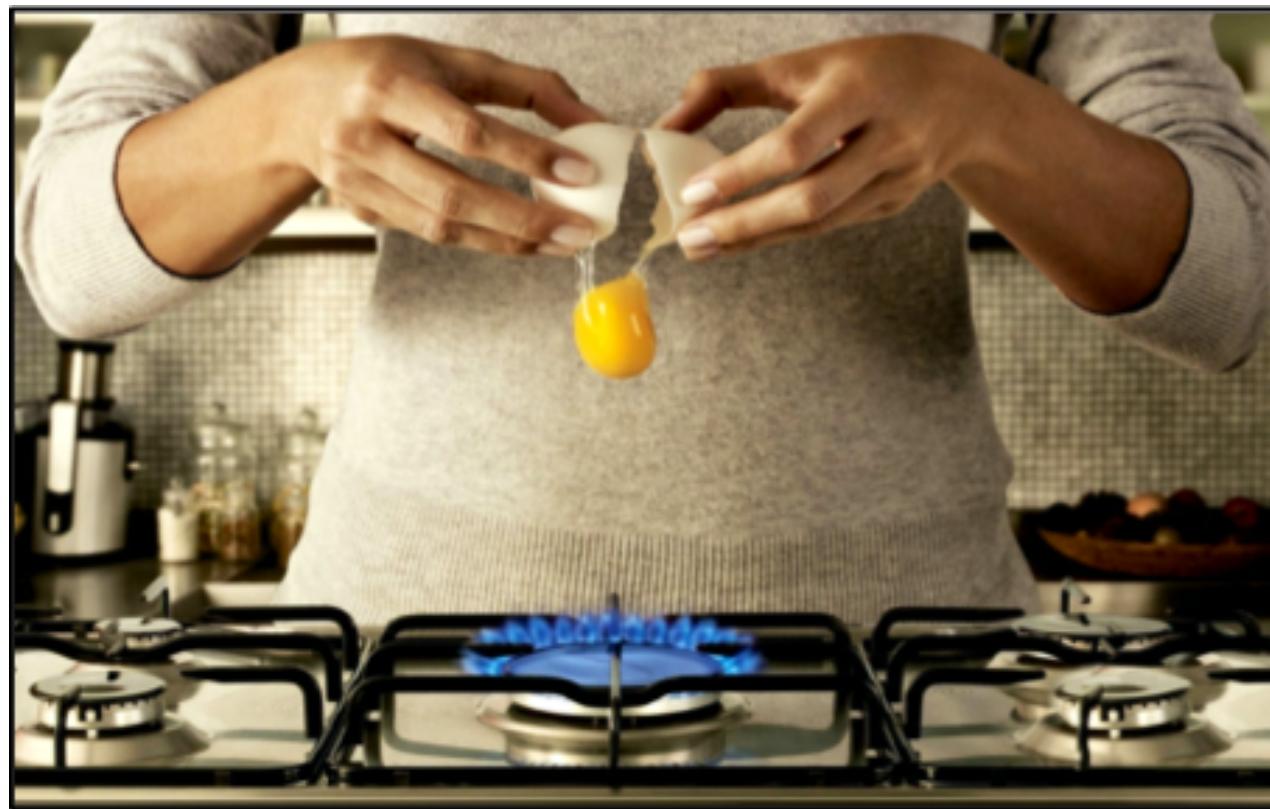
(Vasconcelos, J. & Carvalho, J. – Ed. Centro Atlântico)

Título

- Exemplo 1

- ✓ Fritar um ovo

1. Retirar um ovo do frigorífico
2. Colocar a frigideira no fogão (ligado)
3. Colocar óleo na frigideira
4. Deixar o óleo aquecer
5. Partir o ovo, separando a casca
6. Colocar o conteúdo do ovo na frigideira
7. Esperar um minuto
8. Retirar ovo da frigideira
9. Desligar o fogão



Estruturas de dados

Assume-se que cada constante, variável, expressão ou função é de um certo tipo de dados. Este tipo refere-se, essencialmente, ao conjunto de valores que uma constante, ou variável, ou expressão possa assumir, ou então a um conjunto de valores que possam ser gerados por uma função.

OMELETA DE QUEIJO FRESCO

Ingredientes:

- 170 gr de queijo fresco
- 6 ovos grandes
- 30 gr de manteiga ou margarina
- Sal q.b.

Modo de Preparação:

Ponha o queijo fresco numa tigela e esmague-o com uma colher de pau, até formar um puré espesso e cremoso. Bata os ovos e misture-os com o queijo, adicionando um pouco de água fria. Tempere a gosto. Derreta um pouco de gordura numa frigideira de base larga e adicione a mistura de ovos e queijo. Cozinhe em lume brando até que a omeleta fique pronta mas não demasiado cozida.

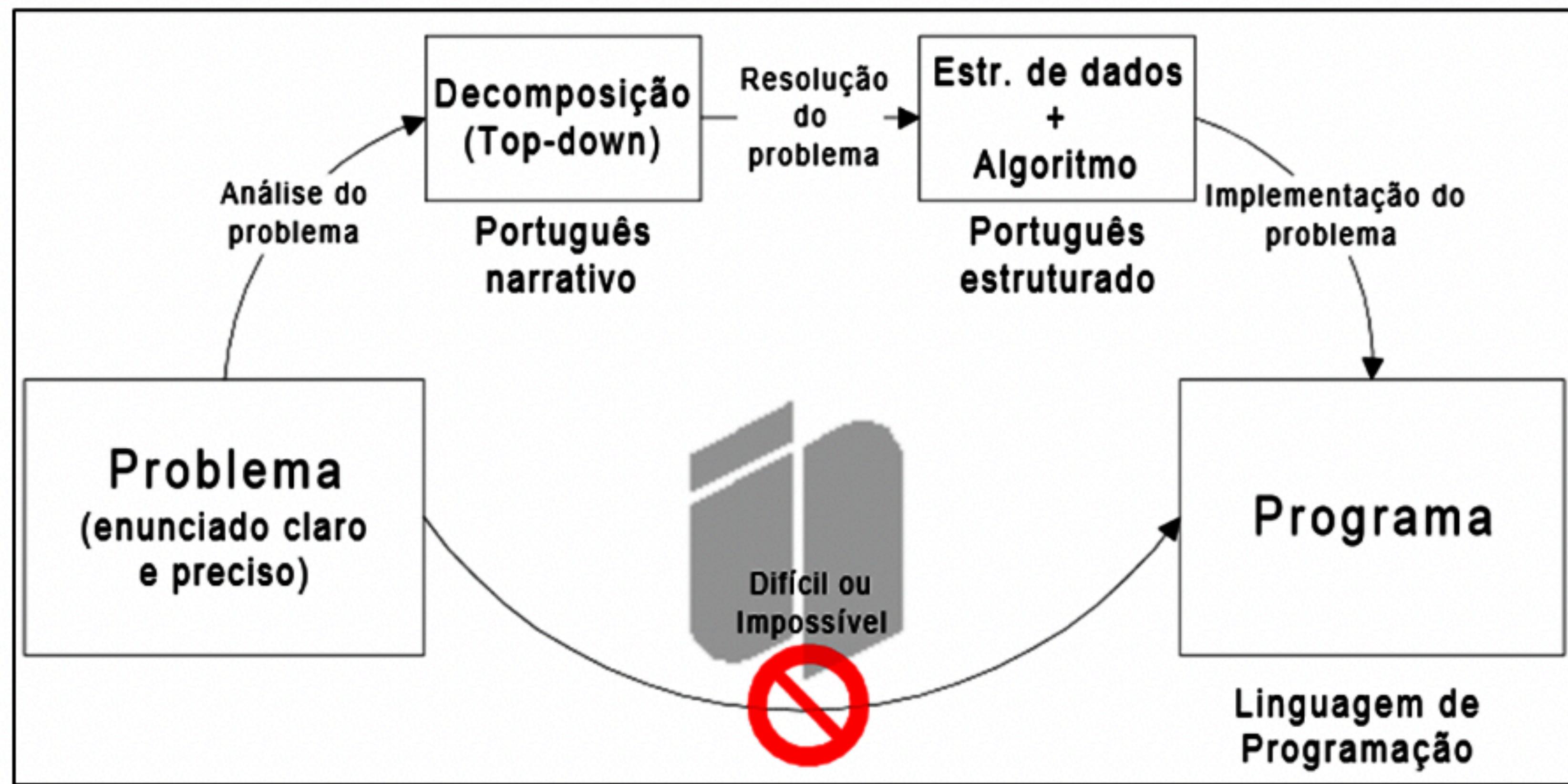
Importancia da Algoritmia

- A algoritmia é fundamental no desenvolvimento de software, pois não só facilita a resolução de problemas complexos, mas também influencia a eficiência, a escalabilidade e a manutenção do código.
- Programadores que entendem os princípios da algoritmia conseguem enfrentar os desafios do desenvolvimento de software com mais facilidade.

Importância da Algoritmia

- A sua importância pode ser compreendida através de vários aspectos:
 - **Solução de Problemas** (estruturar e dividir o problema em pedaços menores e mais fáceis de gerir)
 - **Eficiência** (Algoritmos otimizados melhoram a eficiência do software)
 - **Estruturas de Dados** (usar as estruturas de dados mais indicadas para o problema a resolver)
 - **Escalabilidade** (Requisitos em constante mudança levam à necessidade de implementação de algoritmos que facilitem a escalabilidade)
 - **Reutilização e Manutenção** (Promoção de reutilização de códigos e mais fáceis de manter)
 - **Fundamento Teórico** (base teórica sólida promove as capacidades de resolução de problemas).
 - **Inovação e Criatividade** (Programadores que dominem algoritmos são mais capazes de inovar)

Processo de implementação de um problema



Informática

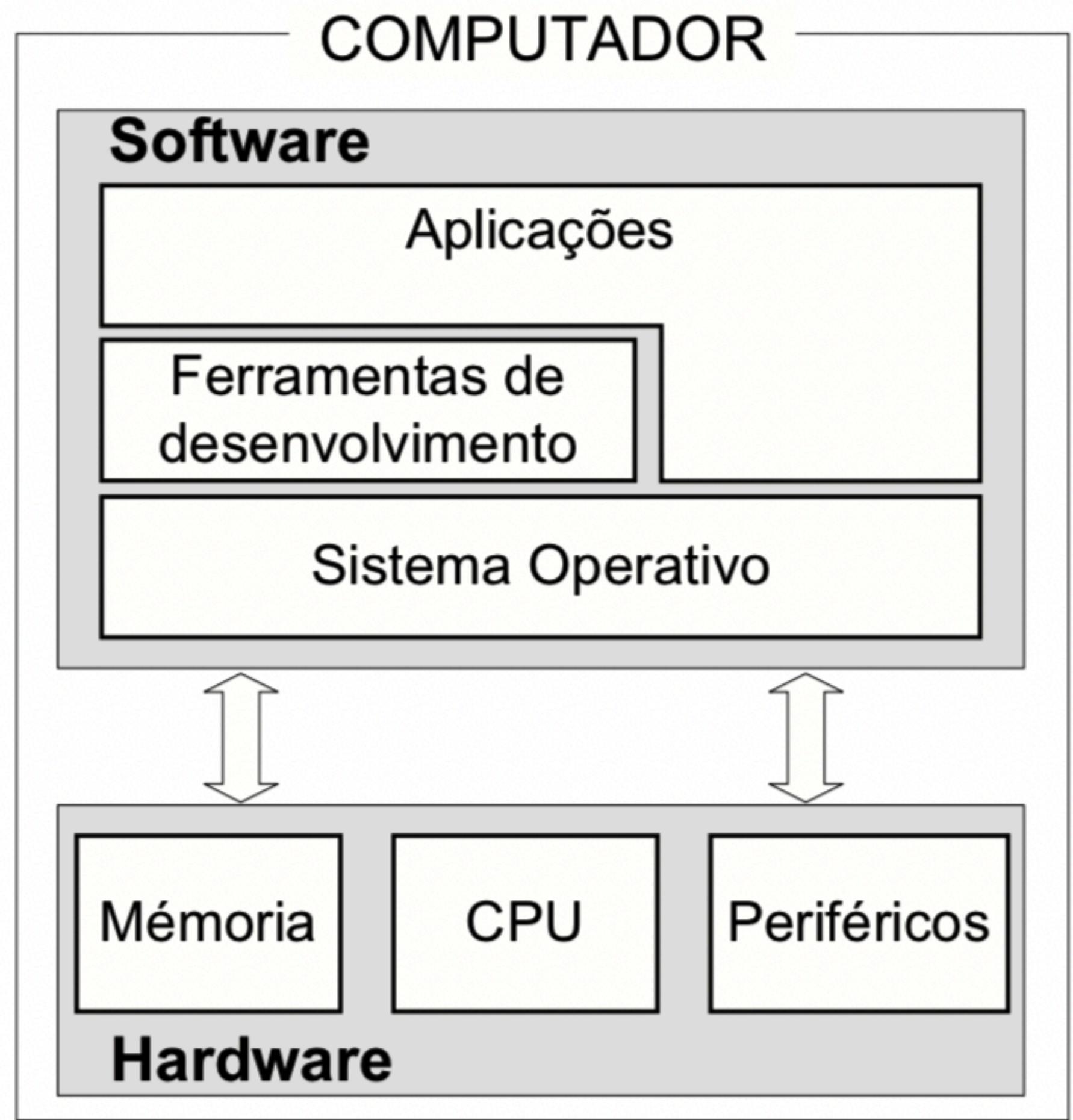
- A informática é a disciplina que trata a informação de forma automatizada.
- Representa o tratamento automático da informação.

INFOR = INFORMAÇÃO

MÁTICA = AUTOMÁTICA

Computador

- **Hardware:** componente física do computador que permite executar instruções, armazenar dados e comunicar com periféricos.
- **Software:** componente lógica do computador, ou seja, os programas que controlam o computador. O SO é o *software* que comunica diretamente com o hardware.



Linguagens de programação

- Uma **linguagem de programação** pode ser vista como um meio de comunicação entre os computadores os programadores;
- Uma **linguagem de programação** permite ao programador especificar o que deve ser executado e com que regras, respeitando as suas regras de semântica e de sintaxe;
- Uma linguagem de programação é constituída:
 - ✓ **Regras de semântica**: conjunto de termos, palavras e símbolos que são usados na linguagem;
 - ✓ **Regras de sintaxe**: ordem correta de como esses termos, palavras e símbolos são usados.
- Um **programa** é um conjunto de instruções que respeitam essas regras de semântica e sintaxe definidas por uma linguagem de programação.

Tipos de linguagens de programação

- Linguagens de programação quanto ao nível de proximidade com o computador:
 - ✓ **alto nível**: o programador escreve as instruções a serem processadas sem se preocupar com a forma como o processador as vai executar. Normalmente, estas linguagens têm um vocabulário mais próximo da linguagem natural;
 - ✓ **médio nível**: o programador tem que se preocupar com pormenores de gestão de memória, no entanto as instruções usadas são próximas da linguagem natural;
 - ✓ **baixo nível**: o programador além de definir o que pretende ver processado, tem que se preocupar com pormenores de execução do processador, e.g. gestão de memória. Exemplos deste tipo de linguagem são a linguagem máquina - binário, e a linguagem assembly.
- As linguagens de alto nível não são reconhecidas directamente pelo processador, tendo de ser traduzidas para linguagem máquina, num processo chamado **compilação**.

Paradigmas linguagens de programação

- Um **paradigma de programação** consiste na forma como um programa é estruturado:
 - ✓ **funcional** - a parte lógica das soluções é implementada através da definição prévia de funções e é muito usada em desenvolvimento de *software* que envolve risco (e.g. Haskell).
 - ✓ **imperativo** - em que o código é implementado sob a forma de comandos (ordens) que vão alterando o estado (variáveis) do programa (e.g. C, Cobol, Pascal).
 - ✓ **declarativo** - é definido o que precisa de ser feito, ficando sob a responsabilidade da máquina (linguagem de programação) encontrar a melhor solução para essa solicitação (e.g. Prolog).
 - ✓ **orientado a objetos** - uma forma de programar com um raciocínio mais próximo de como definimos algo na vida real (e.g. C#, java).

Tipos linguagens de programação - Modo de execução

- Uma linguagem de programação também pode ser definida quanto ao **modo de execução**:
 - ✓ **compiladas** - em que o código digitado pelo programador é transformado (“traduzido”) pelo compilador para uma linguagem máquina de forma a poder ser reconhecido (executado) pelo processador (e.g. C).
 - ✓ **interpretadas** - não recorrem ao processo de compilação para que o processador consiga entender e executar as instruções. Em substituição deste passo, a própria linguagem inclui mecanismos que são responsáveis, em momento de execução, pela tradução instantânea de cada instrução para que o processador a entenda.
 - ✓ **híbridas** - são simultaneamente do tipo compiladas e interpretadas. A compilação é também um processo de “tradução” mas não para uma linguagem a ser entendida diretamente pelo processador. Em vez disso, a compilação traduz o código para uma linguagem mais próxima da linguagem máquina que, posteriormente e em momento de execução, é interpretada por uma plataforma criada para o efeito que faz a mediação de execução com o processador.

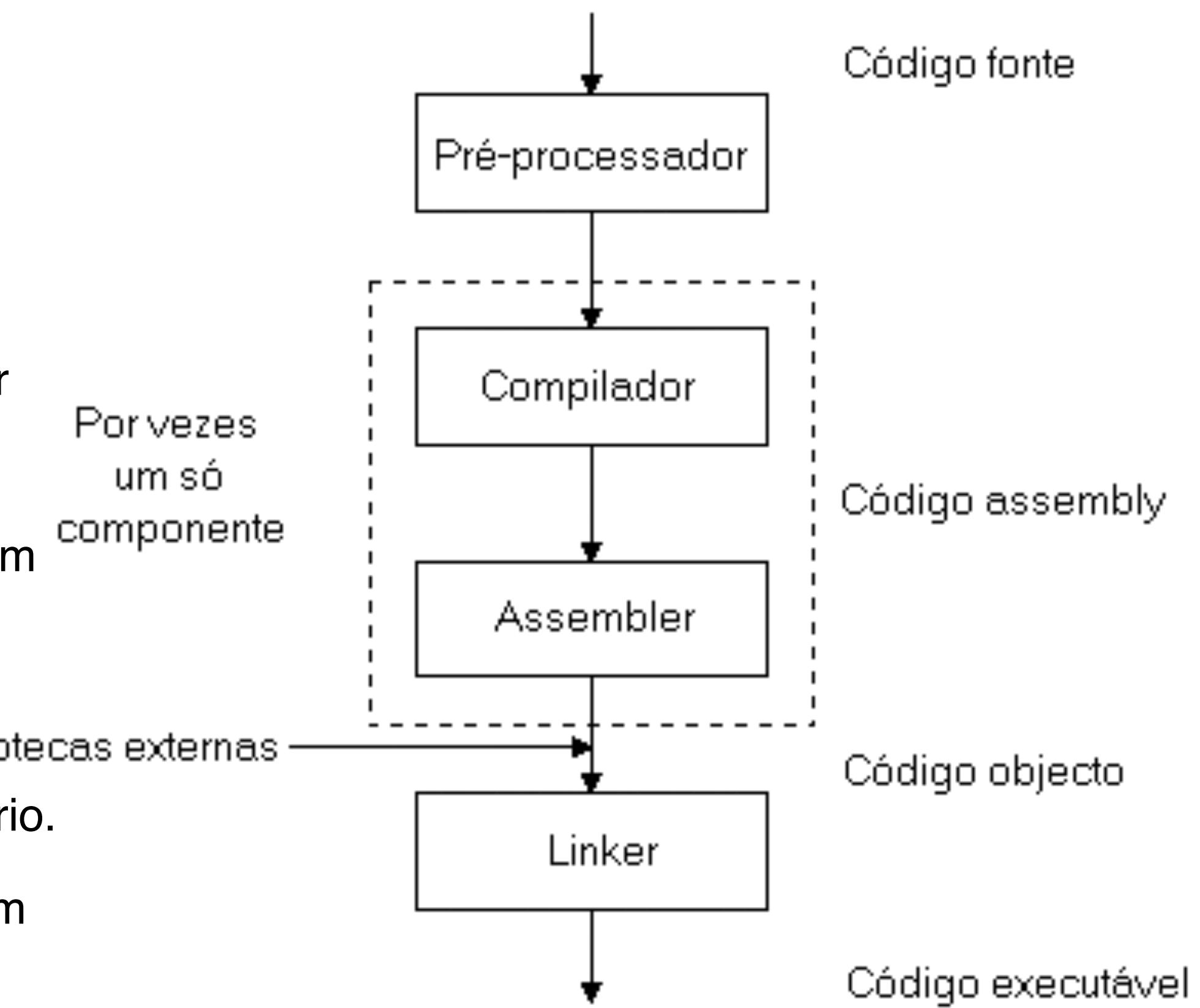
Compilação de um programa

- Um compilador tem a função de traduzir aquilo que os programadores escrevem numa linguagem de programação mais entendível para o ser humano, na linguagem que o processador pode executar (depende da sua arquitetura). A complexidade da linguagem máquina é abstraída pelo compilador.

ALGORITMO	MÁQUINA	ASSEMBLY	C, JAVA, C#...
$s \leftarrow c + d$	01101001001....	s, ADD c, d	$s = c + d;$
$r \leftarrow b - s$	01111001010....	r, SUB b, s	$r = b - s;$
$e \leftarrow a + r$	01000010010....	e, ADD a ,r	$e = a + r;$
fim	00011010101....	HLP	return

Compilação de um programa

- O **pré-processador** atua apenas ao nível do código fonte, modificando-o:
 - ✓ remove os comentários de um programa;
 - ✓ interpreta directivas especiais a si dirigidas, que começam pelo carácter `#`.
- O **compilador** traduz o código fonte (texto) para linguagem *assembly* (também texto). No entanto, são também comuns os compiladores que geram directamente código binário
- O **assembler** traduz código em linguagem *assembly* (texto) para código binário.
- O **linker** combina todos as funções contidas em bibliotecas referenciadas num único arquivo com código executável. As referências a variáveis globais externas também são resolvidas pelo *linker*.



Fluxo simplificado de execução de um programa JavaScript

1. Código fonte JS

- O ficheiro .js é carregado pelo motor

2. Parser (Analizador Sintáctico)

- Analisa o texto
- Cria a AST (Abstract Syntax Tree)
- Verifica erros de sintaxe

3. Interpreter (Ignition, no V8)

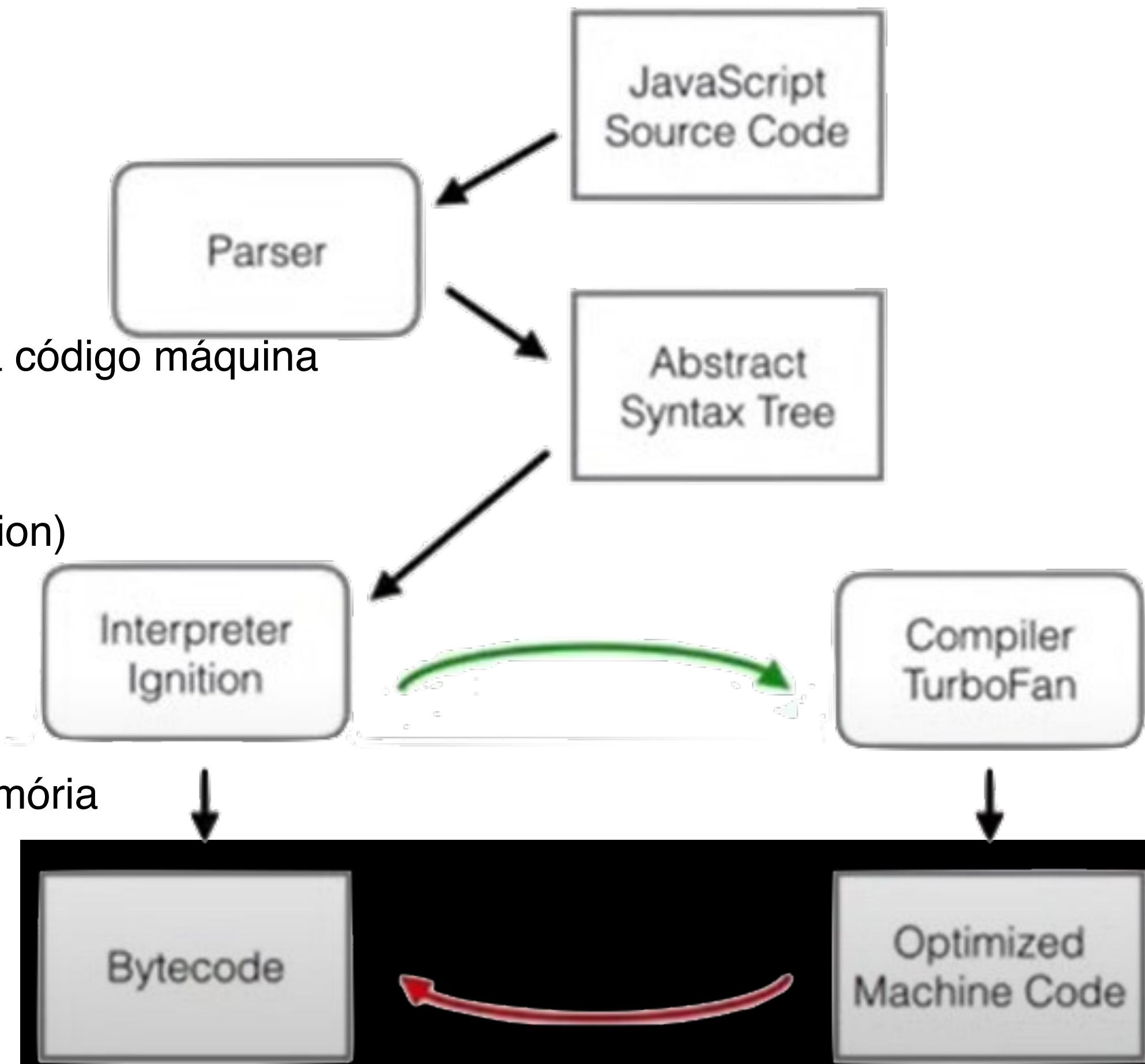
- Executa diretamente a AST
- Produz bytecode intermédio
- Otimiza execuções repetidas

4. JIT Compiler (TurboFan, no V8)

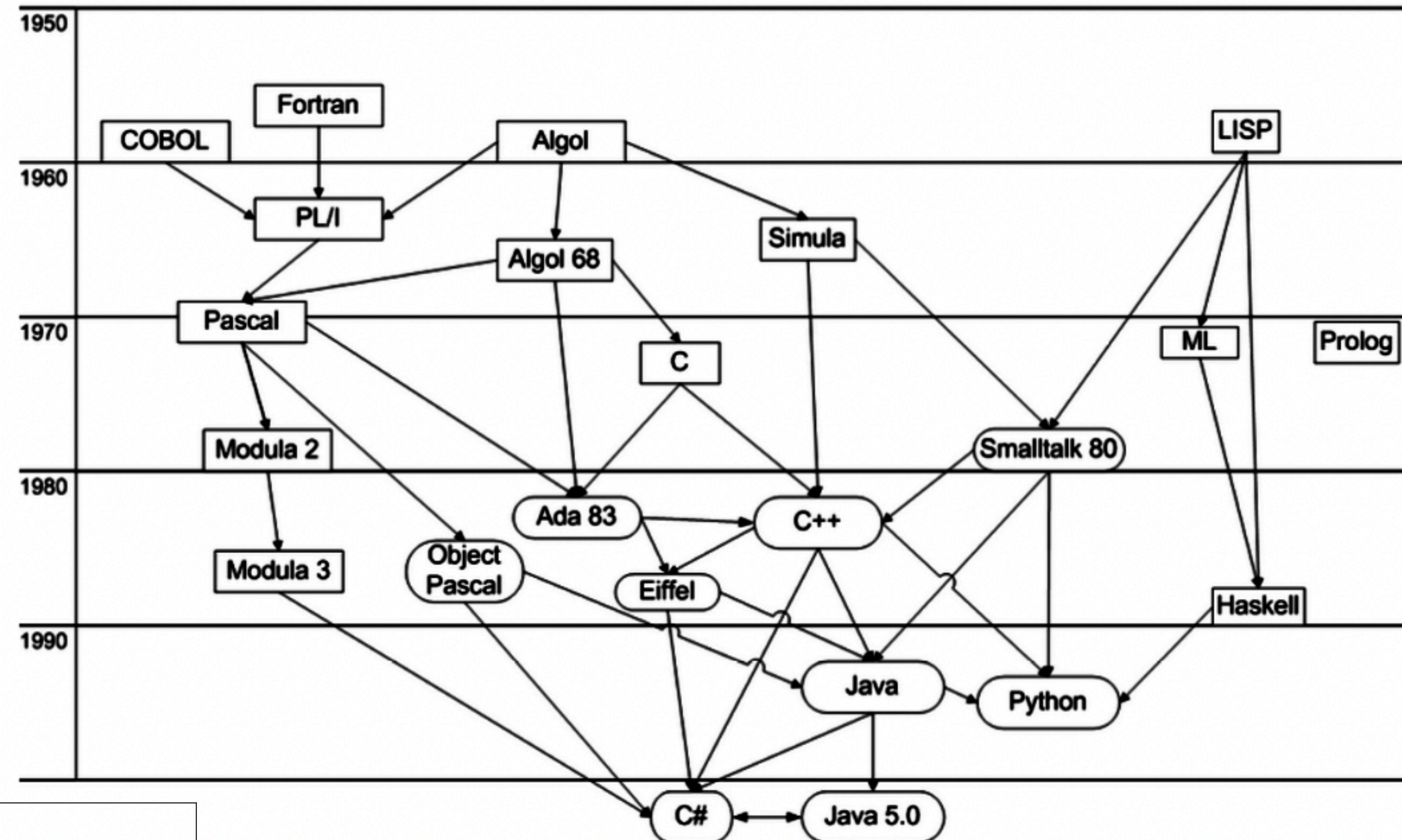
- Compila partes “quentes” do código para código máquina
- Faz otimizações agressivas
- Se o padrão muda, reverte (de-optimization)

5. Garbage Collector

- Gera automaticamente libertação de memória



Linguagens de programação no tempo



Quadro resumo

LINGUAGEM	NÍVEL	PARADIGMA	EXECUÇÃO
Assembly	Baixo	Imperativa Simples	Compilada
C	Médio	Imperativa	Compilada
C++	Médio	Imperativa, OO	Compilada
C#	Médio / Alto	Imperativa, OO	Híbrida
Haskell	Alto	Funcional	Interp. / Comp.
Java	Médio / Alto	Imperativa, OO	Híbrida
Perl	Alto	Imperativa	Interpretada
Prolog	Alto	Declarativa	Interpretada
Python	Alto	Imperativa	Interpretada
Visual Basic .NET	Alto	Imperativa, OO	Híbrida

Representação de algoritmo - linguagem natural

- Cálculo da área de um retângulo:
 - ✓ Passo 1: Procurar a fórmula (conhecer o domínio)
 - ✓ Passo 2: Analisar sobre as estruturas de dados necessárias para resolver o problema
 - ✓ Passo 3: Recolher o altura e a largura;
 - ✓ Passo 4: Calcular a área (altura x largura);
 - ✓ Passo 5: Mostrar resultado.

Representação de algoritmo - pseudocódigo

- O **pseudocódigo** consiste na representação de um algoritmos numa linguagem próxima da linguagem natural, mas de forma estruturada.

```
Algoritmo "Área do Retângulo"
/*
    autor: Marta Martinho
    data: 22-09-2017
    funcionalidade: Calcula a área de um retângulo
*/

Variáveis
    altura: real
    largura: real
    area: real

Início
    //atribuição de valores
    altura <- 2,5
    largura <- 4

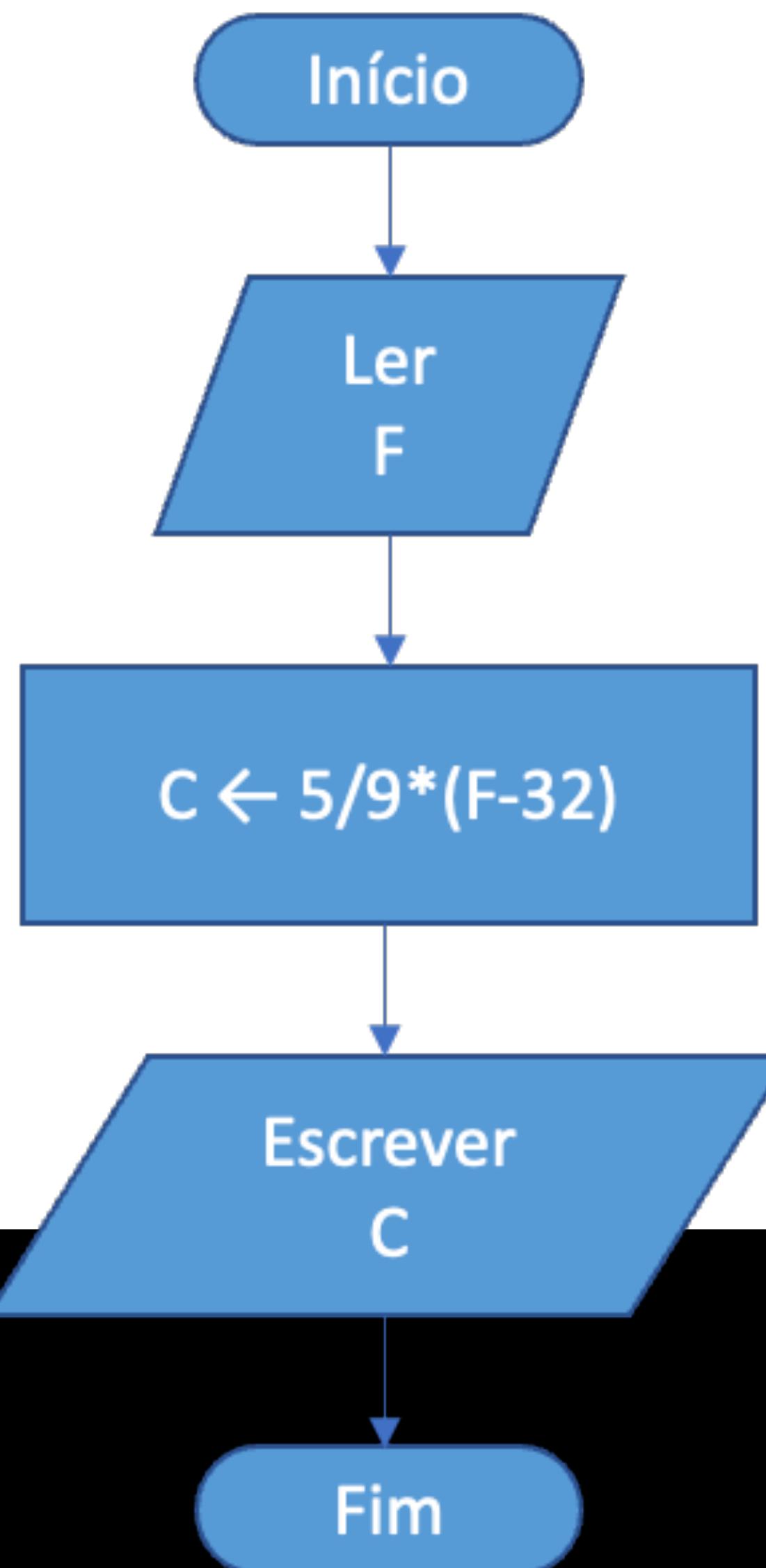
    //calculo da área
    area <- altura * largura

    //mostrar resultado
    escrever("A área é: ", area)

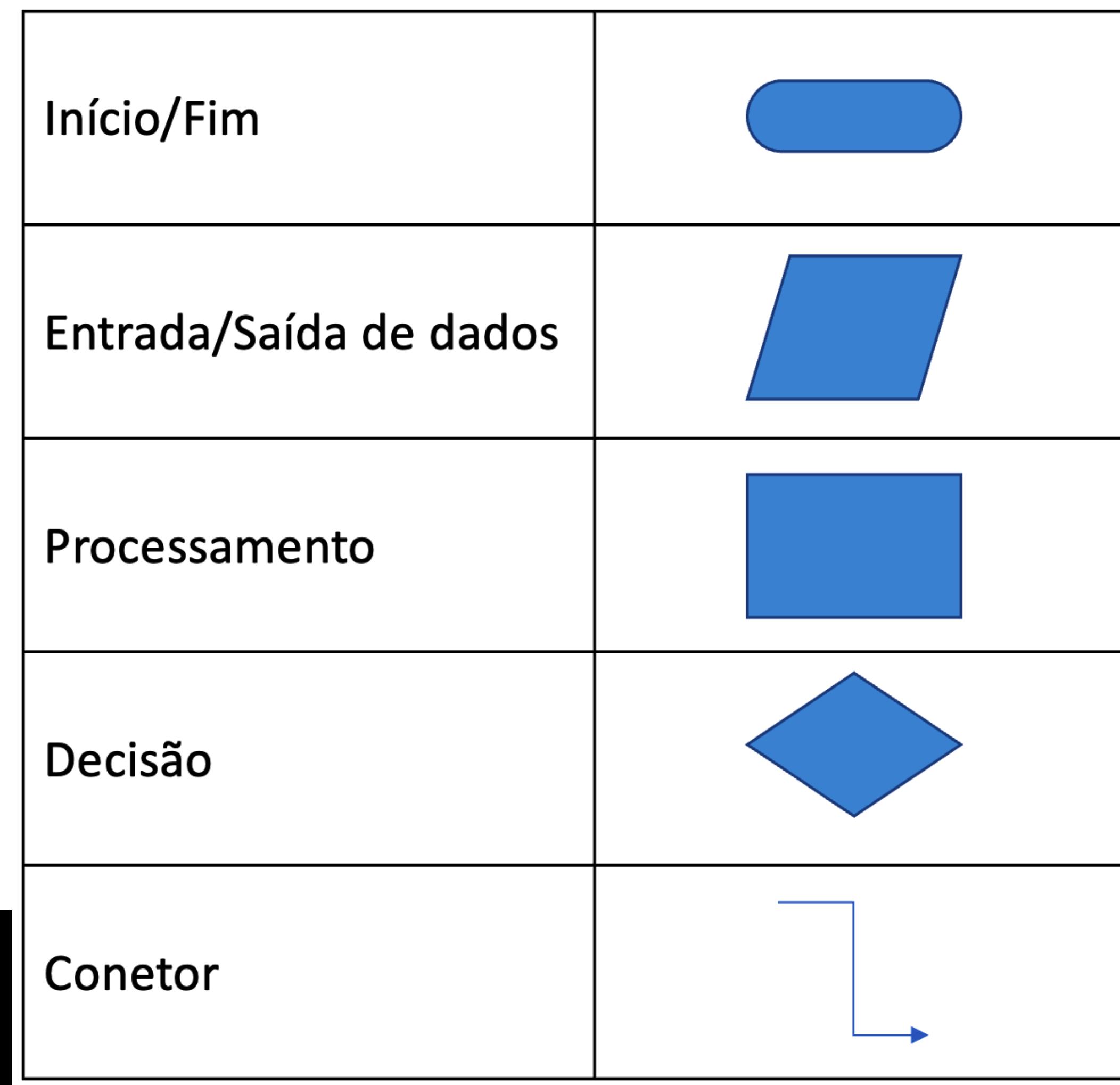
Fim
```

Representação de algoritmo - fluxograma

- O **fluxograma** consiste na descrição de um algoritmo de forma gráfica.
-



Representação de algoritmo - fluxograma



Representação de algoritmo - linguagem de programação

C	C#	Java
//declaração de variáveis float altura, largura, area; //atribuição de valores altura = 2.5F; largura = 4F; //cálculo da área area = altura * largura; //mostrar resultado printf("A área é: %f", area);	//declaração de variáveis float altura, largura, area; //atribuição de valores altura = 2.5F; largura = 4F; //cálculo da área area = altura * largura; //mostrar resultado Console.WriteLine("A área é: {0}", area);	//declaração de variáveis float altura, largura, area; //atribuição de valores altura = 2.5F; largura = 4F; //cálculo da área area = altura * largura; //mostrar resultado System.out.println("A área é: " + area);

Javascript

```
//declaração de variáveis  
let altura, largura, area;  
  
//atribuição de valores  
altura = 2.5;  
largura = 4;  
  
//cálculo da área  
area = altura * largura;  
  
//mostrar resultado  
console.log("A área é: " + area);
```

Exercício

- Instalar node.js <https://nodejs.org/pt/download>
- Instalar visual studio code <https://code.visualstudio.com/download>
- Instalar visualg <https://sourceforge.net/projects/visualg30/>
- Instalar extensão code runner no visual studio code
- Run code runner in terminal (configurações)
- Criar “hello world” (terminal e browser)
-

Tipos de instruções

- Instruções de sequência:

✓ são ordens simples (atómicas), usadas em cálculos, atribuição de valores, leitura e escrita de dados;

- Instruções de decisão:

✓ permitem a escolha entre vários caminhos de processamento com base em avaliações lógicas de uma condição;

- Instruções de repetição:

✓ permitem a execução de forma repetida de um conjunto de instruções.

Instruções de sequência

- Atribuição:

✓ permitem guardar um valor numa variável

```
nome <- "Marta Martinho"
```

```
let nome = "Marta Martinho";
```

- Leitura de dados (input):

✓ permitem recolher a informação introduzida pelo utilizador;

```
ler(nome)
```

```
let nome = prompt("Introduz o teu nome:");
```

- Escrita de dados (output):

✓ permitem mostrar os resultados das operações com os dados ao utilizador.

```
escrever("nome:", nome)
```

```
console.log("Olá, " + nome);
```

O que é uma variável?

- Espaço reservado na memória, identificado por um nome (o tipo não é declarado).
- Pode guardar qualquer tipo de valor: números, texto, booleanos, objetos, funções...
- O tipo é dinâmico — pode mudar durante a execução.
- Sintaxe:

```
let nome = valor;  
  
const nome = valor; (valor constante)  
  
var nome = valor; (forma antiga, evita-se)
```

Regra de nomenclatura de variáveis

- O nome de uma variável deve ser sugestivo;
- Não deve ser uma palavra reservada do JavaScript (ex.: while, for, class, let, ...);
- A primeira letra deve ser sempre uma letra, _ ou \$;
- Não pode começar com número;
- Não pode conter espaços em branco;
- Não pode conter caracteres especiais (como -, +, ?, /, *, %, etc.);
- O comprimento não tem limite prático relevante (os motores modernos suportam milhares de caracteres, mas não é recomendado).

Regra de nomenclatura de variáveis

- let idade; // válido
- let altura_media; // válido
- let primeiraLetra; // válido
- let 2anos; // inválido (não pode começar por número)
- let altura-m; // inválido (não pode ter '-'; seria interpretado como subtração)
- let while; // inválido (palavra reservada da linguagem)

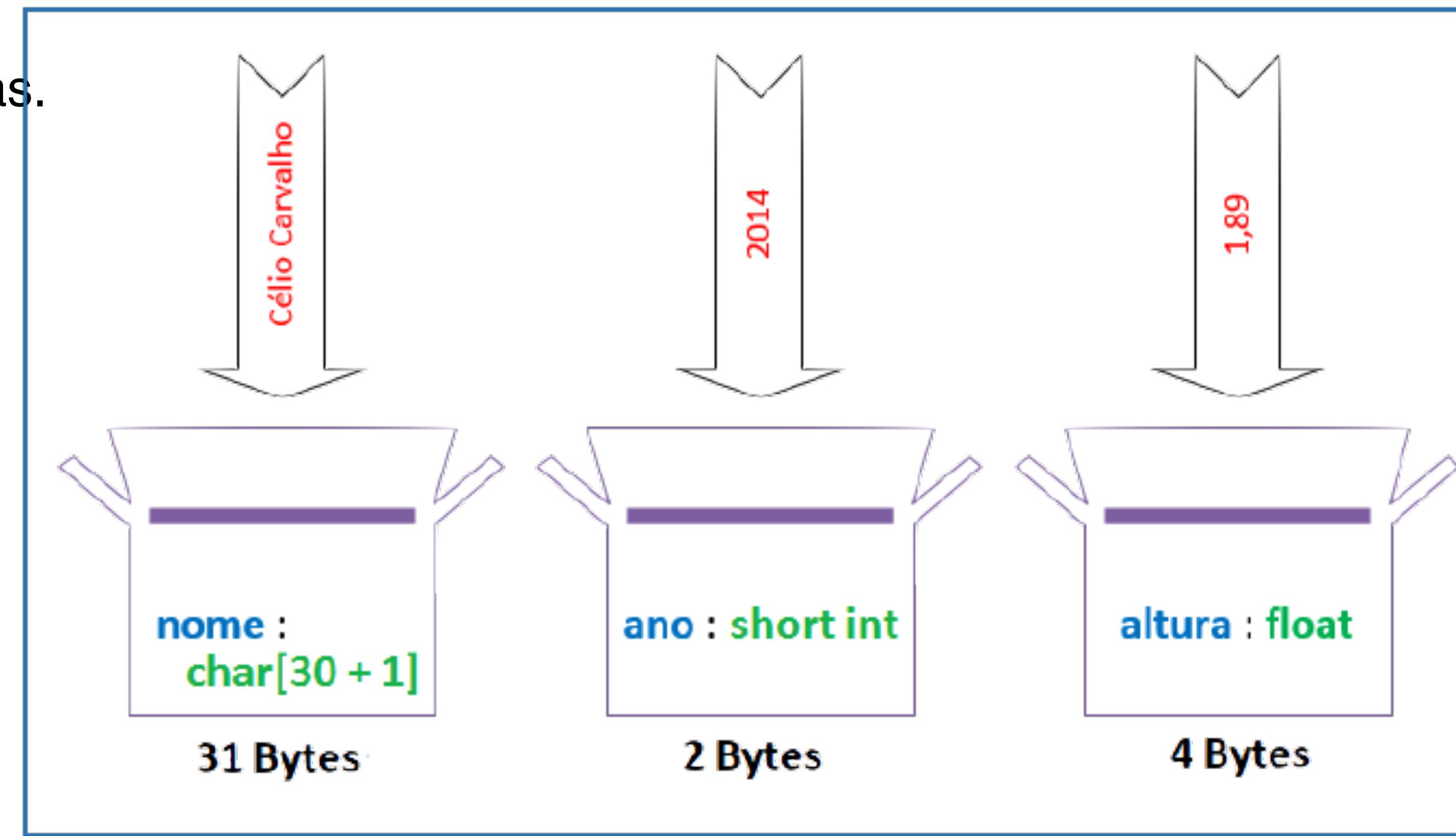
Variáveis e memória

x = 42 (int, 4 bytes)

0x1000 0x1001 0x1002 0x1003 0x1004 0x1005 0x1006 0x1007

Um int típico ocupa 4 bytes contíguos. Aqui, x usaria 0x1000–0x1003.

- A memória do computador é como uma cômoda cheia de gavetas.
- Cada variável é uma gaveta com uma etiqueta (nome).
- Ao contrário de C, não escolhemos o tamanho da gaveta
 - o motor JavaScript trata disso automaticamente.
- Não declaramos o tipo:
 - → let x = 10; (número)
 - → x = "Olá"; (passa a string)
- O motor decide quanto espaço reservar internamente.
- Valores complexos (objetos, arrays, funções) são guardados na memória heap, e a variável guarda apenas uma referência.



Termos reservados

if	else	switch	case	default
break	continue	for	while	do
return	function	class	extends	super
import	export	new	this	typeof
delete	in	instanceof	try	catch
finally	throw	let	const	var
true	false	null	undefined	await
yield	enum*	interface*	private*	public*
protected*	implements*	package*	static*	arguments*

Tipos de dados

PSEUDO-CÓDIGO	TIPO EM JAVASCRIPT	TAMANHO	VALORES POSSÍVEIS
INTEIRO (não há short, byte, long)	number (JS converte tudo para número double)	8 bytes	Inteiros seguros de -9 007 199 254 740 992 a 9 007 199 254 740 992
REAL	number	8 bytes (double)	±1.7976931348623157e308, 15–17 dígitos de precisão
INTEIRO MUITO GRANDE	bigint	variável	valores inteiros arbitrariamente grandes (ex.: 12345678901234567890n)
TEXTO / CARACTER	string	variável	Unicode (cada char pode ocupar 2 a 4 bytes)
LÓGICO	boolean	4 bytes (aprox., dependente do motor)	true ou false
NULO	null	(tag primitivo)	ausência intencional de valor
INDEFINIDO	undefined	(tag primitivo)	variável declarada mas não inicializada
SÍMBOLO	symbol	variável	identificadores únicos
OBJETO	object	variável (heap)	arrays, funções, mapas, datas, etc.

Declaração de variáveis

```
algoritmo "teste"
// Função : Declaração de variáveis
// Autor : Célio Carvalho
// Versão 1.0
var

    // variaveis
    nome : caracter
    ano : inteiro
    altura : real

inicio

    // aqui instruções (código)

fimalgoritmo
```

Figura 11 – Declaração de variáveis em linguagem algorítmica

Declaração de variáveis

```
algoritmo "teste"
// Função : Definição ambígua de tipos de dados
// Autor :
// Versão
var
    idade : real
    dia : caracte
```

Figura 18 - Definição ambígua de tipos de dados em pseudo-código

```
algoritmo "teste"
// Função : Definição clara de tipos de dados
// Autor :
// Versão
var
    idade : inteiro
    diaMes : inteiro
    diaSemana : caracte
```

Figura 21 - Definição natural de tipos de dados em pseudo-código

Declaração de variáveis

```
// variáveis sem tipo declarado (JS infere pelo valor)
let idade = 25; // número
let nome = "Patrícia"; // string
let dia = 10; // dia do mês? (número)
```

```
let diaMes = 10; // numero
let diaSemana = "segunda-feira"; // string
```

Em JavaScript não declaramos tipos (int, char, double não existem).

O tipo é determinado pelo valor atribuído.

let → variável “normal” (pode mudar).

const → constante (não pode ser reatribuída).

var → forma antiga preferencialmente evitada.

```
const nomeLicenca = "9999 - Patrícia Leite";
const PI = 3.141592653589793;
```

Operadores matemáticos

ALGORITMOS	JAVASCRIPT	NOTAS
+	+	Adição
-	-	Subtração
*	*	Multiplicação
/ ou DIV	/	Sempre divisão realEx.: $7/2 = 3.5$
% ou MOD	%	Resto da divisãoEx.: $5 \% 2 = 1$

Em JavaScript, não existe divisão inteira automática.

$7 / 2$ dá sempre 3.5, mesmo que ambos sejam inteiros.

Se quiseres divisão inteira, tens de usar: `Math.floor(7 / 2); // 3`

Exemplo

- Cria um programa que declare duas variáveis inteiros, lhes atribua valores e calcule a sua soma.

```
algoritmo "soma"
// Função : soma de variáveis
// Autor : Célio Carvalho
// Versão 1.0
var
    num1, num2, num3 : inteiro

inicio

    // atribuição de valores às variáveis num1 e num2
    num1 <- 10
    num2 <- 5

    // efetuar a soma e atribuir o resultado à variável num3
    num3 <- num1 + num2

    // escrever na consola o resultado
    escreval("num1 + num2 = ", num3)

fimalgoritmo
```

Figura 26 - Uso do operador soma em pseudo-código

Exemplo

- Cria um programa que declare duas variáveis inteiros, lhes atribua valores e calcule a sua soma.

```
1 // variáveis
2 let num1;
3 let num2;
4 let num3;
5
6 // atribuição de valores às variáveis num1 e num2
7 num1 = 10;
8 num2 = 5;
9
10 // efetuar a soma e atribuir o resultado à variável num3
11 num3 = num1 + num2;
12
13 // escrever na consola o resultado
14 console.log("num1 + num2 = " + num3);
15
16 // (não é necessária pausa para fechar a aplicação em JS)
17
```

```
npm install prompt-sync
```

```
const prompt = require("prompt-sync")();
```

Exemplo

- Desenvolve um programa que utilize duas variáveis do tipo real, atribua valores e calcule a divisão do primeiro número pelo segundo.
- Apresenta o resultado com três casas decimais.

```
algoritmo "divisão"
// Função : divisao de variáveis
// Autor : Célio Carvalho
// Versão 1.0
var
    num1, num2, num3 : real

início
    // atribuição de valores às variáveis num1 e num2
    num1 <- 12.5
    num2 <- 3.4

    // efetuar a divisão e atribuir o resultado à variável num3
    num3 <- num1 / num2

    // escrever na consola o resultado
    escreval("num1 / num2 = ", num3)

fimalgoritmo
```

Figura 29 - Uso do operador de divisão em pseudo-código

Exemplo

- Desenvolve um programa que utilize duas variáveis do tipo real, atribua valores e calcule a divisão do primeiro número pelo segundo.
- Apresenta o resultado com três casas decimais.

```
1 // variáveis
2 let num1;
3 let num2;
4 let num3;
5
6 // atribuição de valores às variáveis num1 e num2
7 num1 = 12.5;
8 num2 = 3.4;
9
10 // efetuar a divisão e atribuir o resultado à variável num3
11 num3 = num1 / num2;
12
13 // escrever na consola o resultado
14 console.log("num1 / num2 = " + num3.toFixed(3));
```

npm install prompt-sync

```
const prompt = require("prompt-sync")();
```

Exemplo

- Escreve um programa que determine o resto da divisão inteira entre dois números inteiros introduzidos no código.
- Mostra o resultado no ecrã indicando a operação realizada.

```
algoritmo "divisão"
// Função : divisao de variáveis
// Autor : Célio Carvalho
// Versão 1.0
var
    num1, num2, num3 : inteiro

início

    // atribuição de valores às variáveis num1 e num2
    num1 <- 7
    num2 <- 2

    // determinar o resto da divisão e atribuir o
    // resultado à variável num3
    num3 <- num1 % num2

    // escrever na consola o resultado
    escreval("num1 % num2 = ", num3)

fimalgoritmo
```

Figura 32 - Uso do operador resto em pseudo-código

Exemplo

- Escreve um programa que determine o resto da divisão inteira entre dois números inteiros introduzidos no código.
- Mostra o resultado no ecrã indicando a operação realizada.

```
1 // variáveis
2 let num1;
3 let num2;
4 let num3;
5
6 // atribuição de valores às variáveis num1 e num2
7 num1 = 7;
8 num2 = 2;
9
10 // determinar o resto da divisão e atribuir o resultado a num3
11 num3 = num1 % num2;
12
13 /*
14  * NOTA:
15  * Em JavaScript o operador % é sempre resto da divisão.
16 */
17
18 // escrever na consola o resultado
19 console.log("num1 % num2 = " + num3);
20
```

npm install prompt-sync

```
const prompt = require("prompt-sync")();
```

Exemplo

- Escreve um programa que declare uma variável contador, lhe atribua um valor inicial e a incremente várias vezes usando diferentes formas de incremento.
- Mostra no ecrã o valor atual da variável após cada operação realizada.

```
algoritmo "contador"
// Função : contador
// Autor : Célio Carvalho
// Versão 1.0
var
    contador : inteiro

início

    // atribuição do valor inicial
    contador <- 0

    // incrementar contador
    contador <- contador + 1

    // escrever na consola o valor atual do contador
    escreval("contador = ", contador)

fimalgoritmo
```

Figura 35 - Uso de operador incrementador e decrementador em pseudo-código

Exemplo

- Escreve um programa que declare uma variável contador, lhe atribua um valor inicial e a incremente várias vezes usando diferentes formas de incremento.
- Mostra no ecrã o valor atual da variável após cada operação realizada.

```
1 // variáveis
2 let contador;
3
4 // atribuição do valor inicial
5 contador = 0; // valor atual de contador: 0
6
7 // incrementar contador
8 contador += 1; // valor atual: 1
9 contador = contador + 1; // valor atual: 2
10 contador++; // valor atual: 3
11
12 // escrever na consola o valor atual do contador
13 console.log("contador = " + contador);
```

```
npm install prompt-sync
```

```
const prompt = require("prompt-sync")();
```

Exemplo com input de dados

- Elabora um programa que peça ao utilizador três números inteiros, calcule a média e apresente o resultado formatado com duas casas decimais.
- Deve ainda incluir uma mensagem inicial no ecrã e aguardar que o utilizador pressione ENTER para terminar o programa.

```
algoritmo "Calcularmedia"
// Função : Calcular média de 3 números recolhidos do teclado
// Autor : Célia Carvalho
// Versão : 1.0
var
    // Declarar variáveis e atribuir valores exemplo
    num1, num2, num3 : inteiro
    media : real

    inicio
        // recolher o primeiro valor
        escreva("insira o primeiro valor: ")
        leia(num1)

        // recolher o segundo valor
        escreval("")
        escreva("insira o segundo valor: ")
        leia(num2)

        // recolher o terceiro valor
        escreval("")
        escreva("insira o terceiro valor: ")
        leia(num3)

        // calcular media
        media <- (num1 + num2 + num3) / 3

        // mostrar resultado
        escreval("")
        escreval("o resultado é", media)
fimalgoritmo
```

Exemplo com input de dados

- Elabora um programa que peça ao utilizador três números inteiros, calcule a média e apresente o resultado formatado com duas casas decimais.
- Deve ainda incluir uma mensagem inicial no ecrã e aguardar que o utilizador pressione ENTER para terminar o programa.

```
1 const prompt = require("prompt-sync")({ sigint: true });

2
3 // declarar variáveis
4 let num1, num2, num3;
5 let media;

6
7 // limpa consola
8 console.clear();

9
10 // desenhar ecrã
11 console.log("===== Calcular a média =====");
12 console.log("                                by John Doe");
13 console.log("");

14
15 // pedir inserção do primeiro valor
16 console.log("");
17 num1 = Number(prompt("      Insira o valor 1/3: "));

18
19 // pedir inserção do segundo valor
20 console.log("");
21 num2 = Number(prompt("      Insira o valor 2/3: "));

22
23 // pedir inserção do terceiro valor
24 console.log("");
25 num3 = Number(prompt("      Insira o valor 3/3: "));

26
27 // calcular a média
28 media = (num1 + num2 + num3) / 3.0;

29
30 // mostrar média no ecrã
31 console.log("");
32 console.log("      A média é: " + media.toFixed(2)); //arredonda a duas casas decimais

33
```

Exercício

- Resolver FE01 - Instruções de sequência

AED

Introdução à algoritmia

Marta Martinho 2025