

Algoritmos e Programação

1º Ano - 1º Semestre

1. Teoria de programação: conceitos básicos

Escola Superior de Tecnologia e Gestão de Viseu 2020-21

Agradecimentos a Francisco Morgado e Carlos Simões

Copyright

Os *slides* desta unidade curricular foram desenvolvidos pelo Eng.^º Manuel Baptista e pelo Eng.^º Ernesto Afonso em 1998.

A presente versão é uma adaptação elaborada por Miguel Vilaça, desses *slides*.

Novas versões foram melhoradas por Carlos Simões, Francisco Morgado e Jorge Loureiro.

1. Teoria de programação: conceitos básicos

1.1 Introdução à programação e seus objectivos

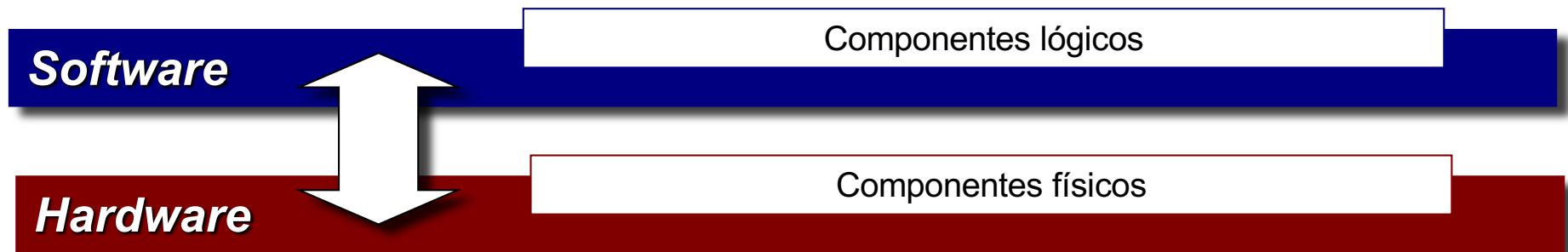
1.2 Linguagens de programação

1.3 Metodologia de programação

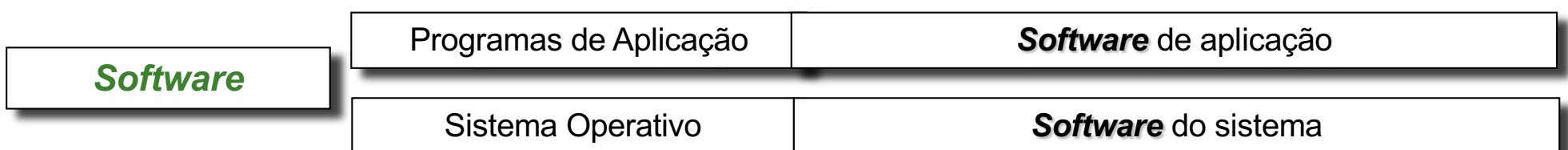
1.4 Construção de um algoritmo

1.1 Introdução à programação e seus objectivos

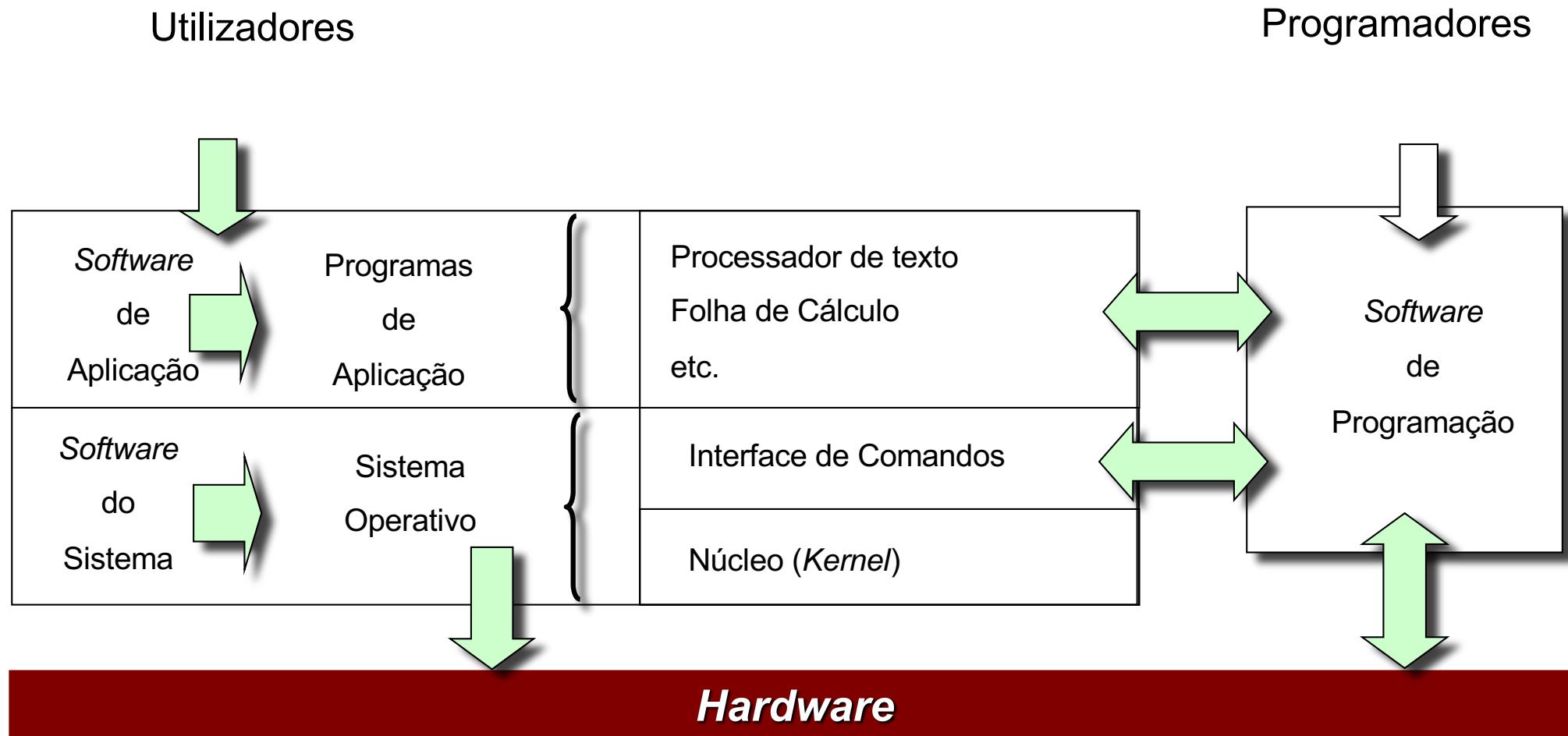
Dando vida ao Hardware: o Software



O *Software* permite que o *Hardware* realize as tarefas que conduzem à resolução de problemas.
Não se trata de um elemento único, mas de uma estrutura em camadas.



Interacção do Utilizador/Programador com o *Hardware*



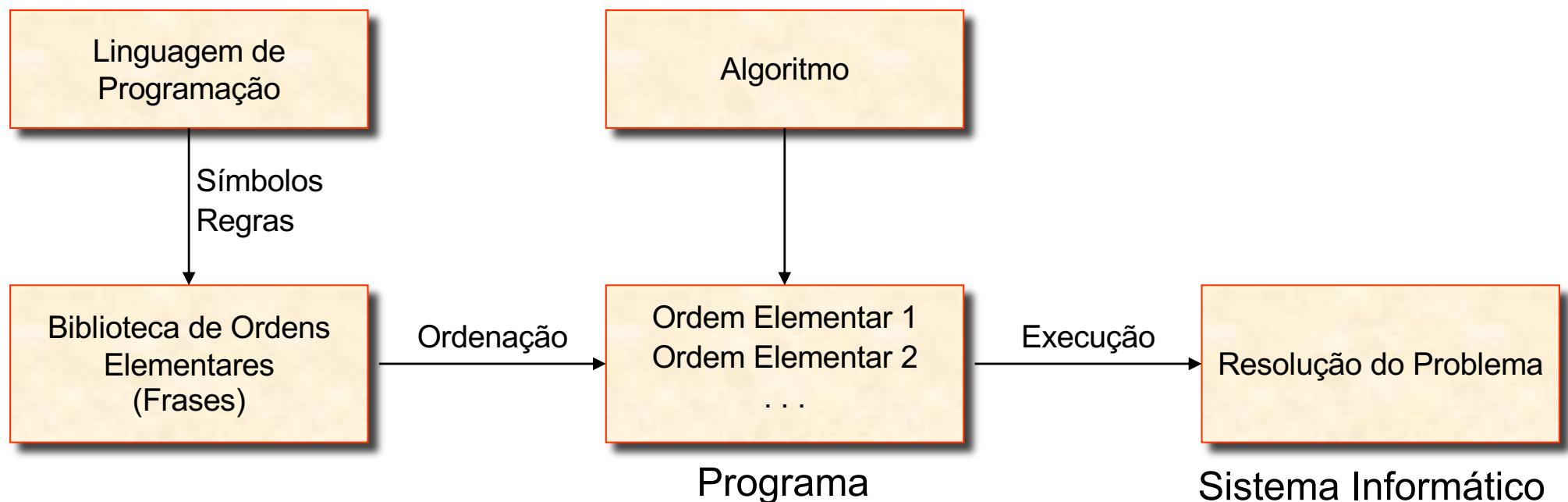
Linguagem de programação

Linguagem de programação - trata-se dum sistema de escrita que, através de um conjunto de símbolos convencionados, permite explicitar as operações a executar por um sistema informático. É constituída por duas componentes:

- **Componente Semântica** - utilização dum conjunto de termos, palavras ou sinais que transmitem um determinado significado interpretável, directa ou indirectamente, pelo CPU
- **Componente Sintáctica** - utilização dum conjunto de regras (tal como uma gramática) que definem a forma correcta de utilização dos termos da linguagem, na construção de instruções válidas.

O Programa

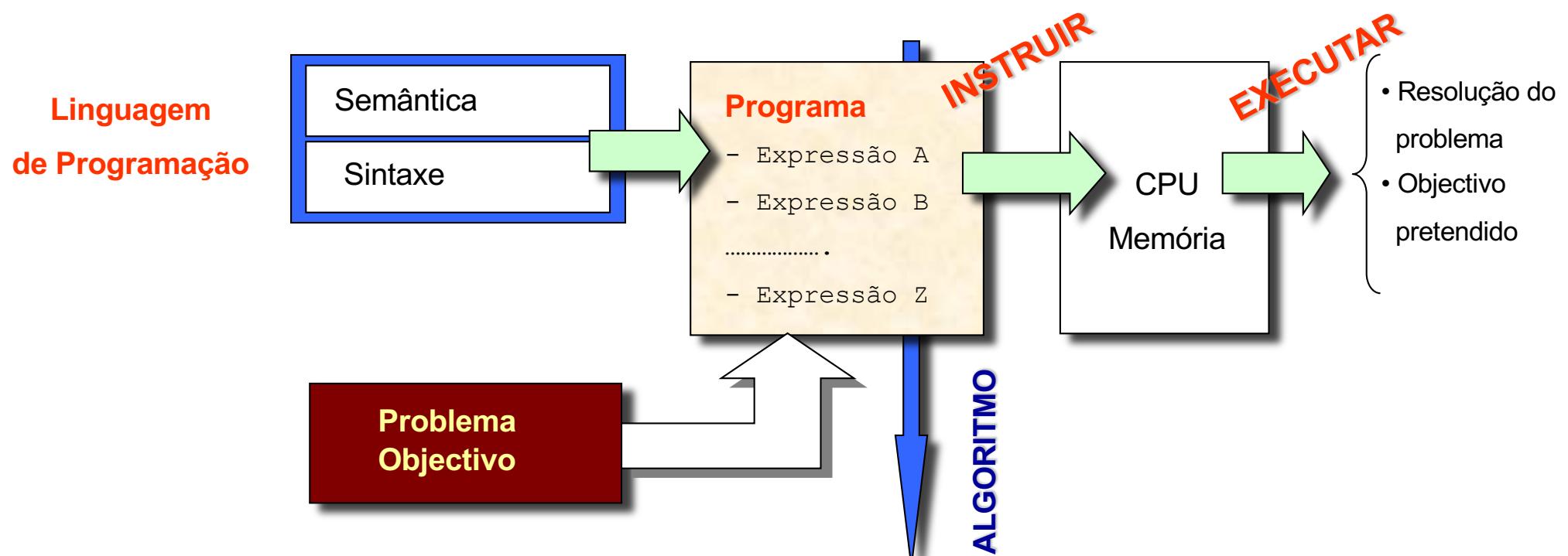
Programa - trata-se dum conjunto de expressões, baseadas nas componentes semânticas e sintácticas da linguagem de programação, que representam as instruções que o CPU deve executar para concretizar os objectivos pretendidos para a resolução dum dado problema.



O Algoritmo

Algoritmo - trata-se duma fórmula para a resolução dum determinado problema, através do estabelecimento de determinadas regras ou procedimentos.

(A sua especificação será detalhada mais em concreto, na secção seguinte)



Os **Programas** consistem normalmente na tradução dum problema ou conjunto de problemas, seguindo um determinado **Algoritmo**, com vista à sua resolução pelo sistema informático;

1.2 Linguagens de programação

Classificação das linguagens: níveis

Níveis das Linguagens de Programação - exprimem uma maior ou menor proximidade do programador relativamente à linguagem do computador.

Linguagens de programação	Alto Nível	O programador não tem que saber nem preocupar-se com a forma como a máquina resolve as operações que ele manda executar.
	Baixo Nível	Cada instrução (ordem para o CPU) está associada a uma sequência de “0” e “1”, designada por <i>palavra</i> .

Linguagens de Baixo Nível

- **Linguagem máquina (formato binário)** - codificação das instruções e dados para formato binário do computador, de acordo com a estrutura ou arquitectura do CPU. Corresponde a sequências codificadas de *bits*.
- **Linguagem assembly (mnemónicas)** - nível de codificação ainda muito próximo da linguagem máquina, mas baseada em mnemónicas (abreviaturas ou caracteres sugestivos das palavras), representando as operações.

Implementação da expressão: $Z = A + B - (C + D)$

linguagem máquina vs linguagem assembly

Instrução	Linguagem máquina	Linguagem assembly	Significado
1	011011010100...	S, ADD C,D	- Fazer: $S \leftarrow C+D$
2	011001010101...	R, SUB B,S	- Fazer: $R \leftarrow B-S$
3	001010101110...	E, ADD A,R	- Fazer: $E \leftarrow A+R$
4	010000000000...	HLT	- Finalizar

Linguagens de Alto Nível

Numa **Linguagem de Alto Nível** o programador utiliza um conjunto mais abrangente de instruções que escreve nos seus programas, tendo apenas que saber escrevê-las correctamente e dar-lhes uma ordenação lógica, com vista a obter o resultado. Surgiram em meados dos anos 50 com o **FORTRAN**, e no início da década de 60 já se contavam cerca de 100, atingindo várias centenas ao longo dessa década e seguintes.

Algumas linguagens

- **FORTRAN** (*FORmula TRANslation System*)
- **COBOL** (*Common Business Oriented Language*)
- **BASIC** (*Beginner's All purpose Symbolic Instruction Code*)
- **PASCAL** (*Homenagem a Blaise Pascal*)
- **C** (Kernighan / Ritchie)
- **Linguagens orientadas por objectos (OOP)** (*Não são na maioria dos casos linguagens criadas de raiz, mas a aplicação do conceito de programação associado à evolução da programação estruturada e modular. Entre as linguagens que aderiram a este conceito, temos o C++ e o Turbo Pascal da BORLAND. Contudo, existem linguagens criadas de raiz, de acordo com esta filosofia, como é o caso do "Smalltalk". Mais recentemente, existe o JAVA e o C#.*)

Paradigmas de Programação

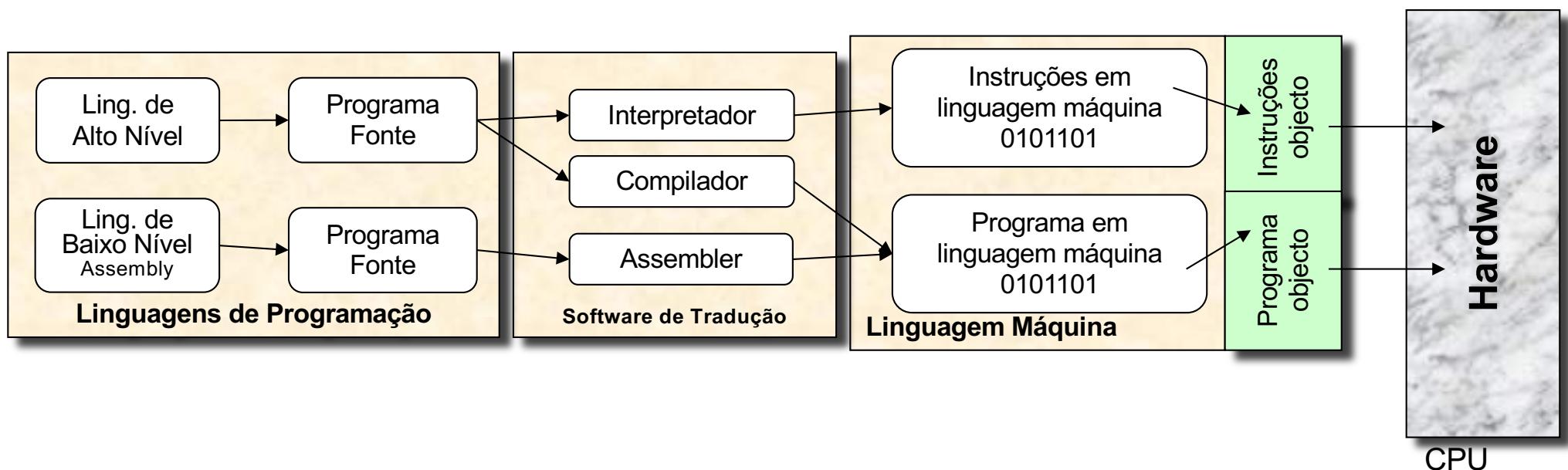
Existem diferentes paradigmas, isto é filosofias ou perspectivas/visões, de programação:

- Imperativo (C, Basic, Cobol)
- Declarativo
 - Funcional (Haskell, Scheme)
 - Lógico (Prolog)
- Orientado a Objectos (C++, JAVA, C#)
- (outros)

Nesta unidade curricular restringimo-nos ao paradigma imperativo.

Software de Tradução

Software de Tradução - Para converter um programa escrito numa linguagem de alto nível para linguagem máquina, não basta indicar ao computador que leia o programa, é necessário um *software* específico para fazer a tradução para a linguagem máquina inteligível ao computador.



Software de Tradução (continuação)

Tipos de Tradutores

Interpretador - traduz instrução a instrução à medida que o programa vai sendo lido e executado, sendo a tradução feita simultaneamente. O programa necessitará sempre de recorrer ao interpretador sempre que é executada uma instrução.

Compilador - traduz a totalidade das instruções para um programa em linguagem máquina, o qual poderá ser executado directamente pelo computador, independentemente do *software* que efectua a tradução.

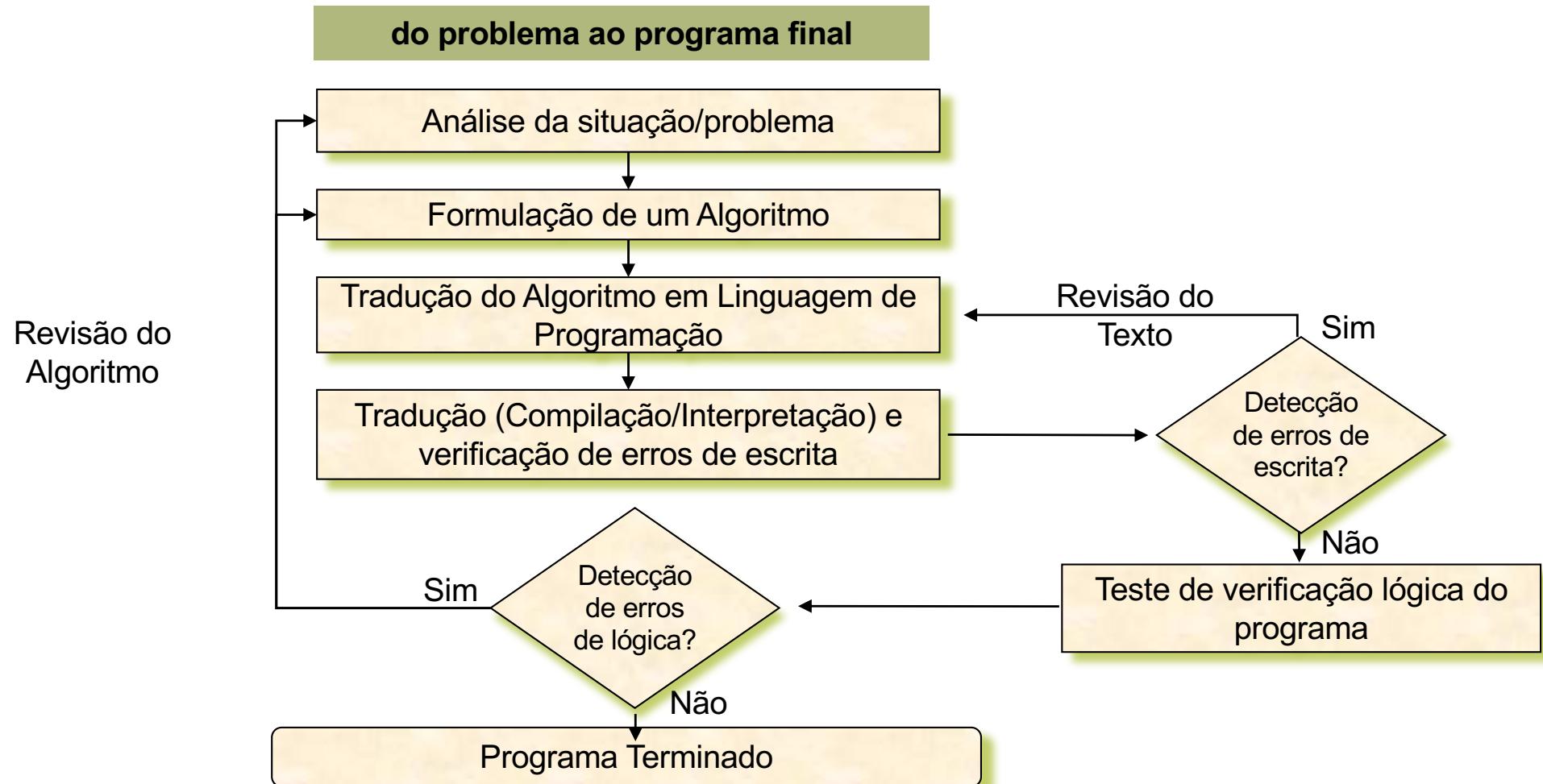
Tipos de códigos

Linguagem/código fonte - trata-se do programa escrito numa linguagem, antes de ser traduzido.

Linguagem/código objecto - trata-se do programa traduzido para formato binário, já no formato executável ou código máquina.

1.3 Metodologia de programação

Ciclo de desenvolvimento estruturado



Análise do problema

Análise do Problema - consiste na compreensão e descrição do problema a resolver, para que possa ser estudado em toda a dimensão e profundidade. A análise de um problema deve responder às seguintes questões:

Entradas: dados com que vamos trabalhar.

Saídas: dados que deveremos obter como resultado.

Definição do problema: processos que deveremos utilizar para produzir os resultados.

Exemplo: Calcular o efeito da redução do IVA em 1% num determinado carregamento de telemóvel.

Entradas: Custo anterior

Saídas: Novo custo

Definição do problema: Novo custo = $1.2 / 1.21 \times$ Custo anterior

(Novo custo= $1.2 \times$ Valor; Custo anterior= $1.21 \times$ Valor. Portanto, Novo custo = $1.2 / 1.21 \times$ Custo anterior)

Exemplo: em vez de um carregamento de 25€, podemos carregar 24,793388 pelo mesmo serviço (poupamos cerca de 20 céntimos ...)

Formulação do Algoritmo

Algoritmo – podemos entendê-lo como uma fórmula de resolução do problema, que sintetiza uma descrição da sequência ordenada e precisa de passos, acções ou operações, que ao realizarem-se levam à resolução do problema.

Exemplos:

- Descrição da execução de uma receita de culinária.
- Processos utilizados para realizar as operações aritméticas elementares: soma, subtração, multiplicação e divisão.
- Extracção de uma raiz quadrada
- Resolução de uma equação de 2º grau.

Formas de representação de algoritmos

Fluxogramas - são diagramas representativos do fluxo das acções de um programa através de símbolos (tipos de acções) e ligações (encadeamento das acções).

Pseudocódigo - é um código de escrita em que se utilizam termos convencionais para indicar as instruções do programa. Usualmente utiliza-se um misto de palavras da nossa linguagem natural com palavras e notações típicas das linguagens de programação.

COMENTÁRIO: o algoritmo não é mais que a descrição da forma ordenada, com clareza e rigor, das operações que se pretendem realizar no sistema informático, para resolver problemas ou atingir determinados objectivos.

Fluxogramas: simbologia

Símbolos	Significado	Exemplos
(Retângulo)	Processamento em geral	$Y \leftarrow Y+1$
(Trapecio)	Leitura/Escrita de dados	Escreve Y
	Início/Fim de processamento	Início
→	Linha de fluxo	→
○	Conector de fluxos	
◇	Decisão Condicional	
◇	Escolha Múltipla	
(Retângulo)	Subprograma	

Pseudo-código/Pseudo-linguagem: características

Genérico - a sintaxe e a semântica não são específicas duma linguagem de programação, pelo que qualquer algoritmo em pseudo-código pode ser implementado em qualquer linguagem de programação (imperativa).

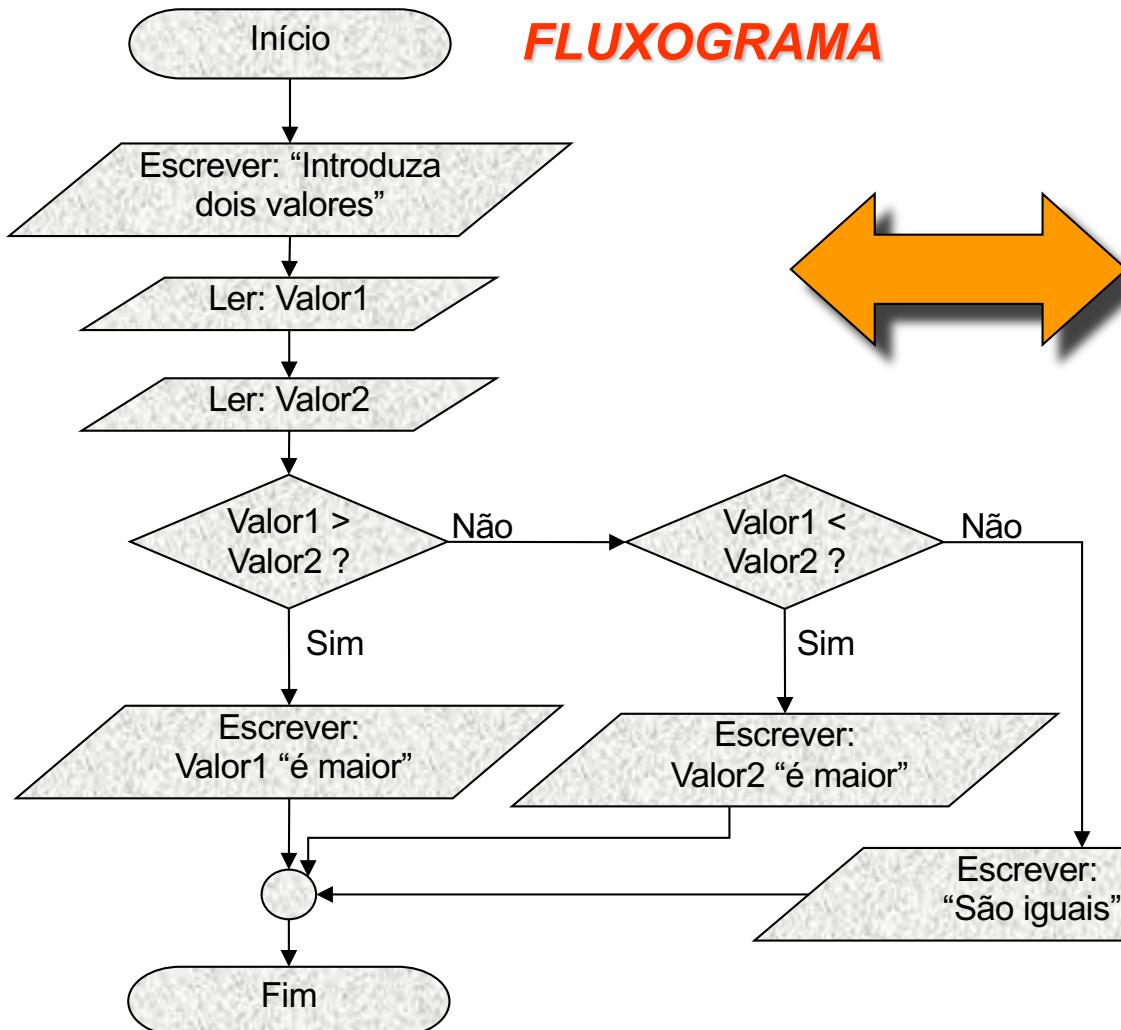
Informal - deixa de parte o formalismo rígido de uma linguagem de programação simplificando a realização do algoritmo.

Abrangente - permite resolver qualquer tipo de problema de programação, desde os mais simples aos mais complexos.

Estruturado - inclui as estruturas de controlo da programação estruturada, implementadas também nas linguagens de programação, de modo a facilitar a interligação entre a pseudo-linguagem e a implementação.

Compacto - permite um maior nível de abstracção relativamente à complexidade da implementação, sendo ideal para a representação de algoritmos mais complexos.

Fluxograma versus Pseudo-código (exemplo)



PSEUDOCÓDIGO

INÍCIO

Escrever ("Introduza dois valores")

Ler (Valor1)

Ler (Valor2)

SE Valor1 > Valor2 ENTÃO

 Escrever (Valor1, "é maior")

SENÃO

SE Valor1 < Valor2 ENTÃO

 Escrever (Valor2, "é maior")

SENÃO

 Escrever ("São iguais")

FIM-SE

FIM-SE

FIM

1.4 Construção de um algoritmo

Abordagens

Estruturada - Separação entre a **Parte Declarativa** e a **Parte Operativa** dum algoritmo.

Parte Declarativa - onde se declaram os tipos de dados, as estruturas, as constantes e as variáveis que se pretende utilizar na parte operativa do algoritmo.

Parte Operativa - o corpo de instruções com que se pretende concretizar determinado objectivo num algoritmo.

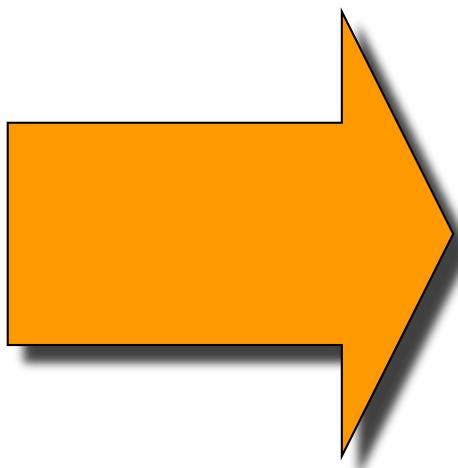
Top-down ou Descendente - é um método de abordagem dos problemas em que primeiro se procuram identificar os pontos essenciais e mais gerais e só depois as componentes mais particulares, em níveis sucessivos e mais concretos, até ao nível do pormenor.

(Na abordagem **Bottom-up ou Ascendente** faz-se o percurso inverso)

Refinamento progressivo - é um complemento lógico da abordagem descendente, e consiste em concretizar, cada vez com mais detalhe, exactidão e perfeição, os passos sucessivos do algoritmo.

Modular - trata-se também dum complemento à abordagem descendente, consistindo na decomposição do problema complexo em **módulos** independentes entre si e bem diferenciados logicamente, mas relacionáveis, tornando possível a sua reutilização noutros contextos ou programas.

Componentes fundamentais



Dados

Instruções Básicas

Expressões

Estruturas de Controlo

Subprogramas

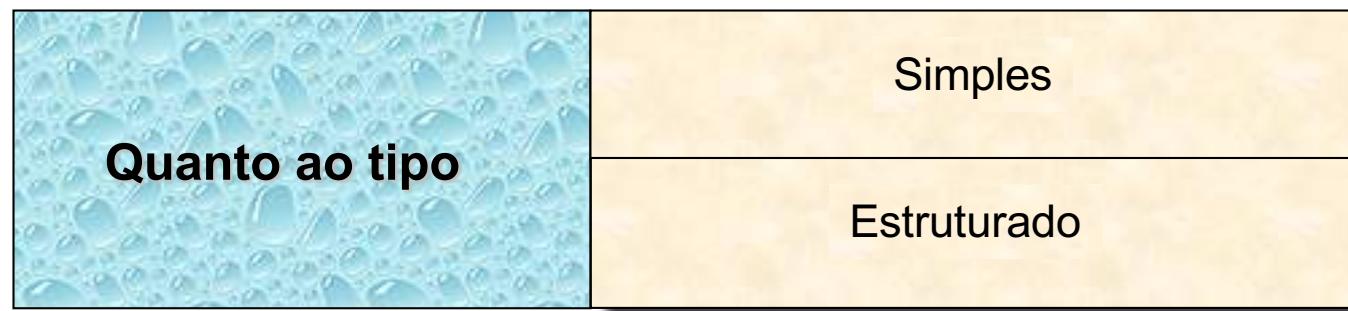
Algumas considerações...

- ◆ Num algoritmo o fluxo de controlo é sequencial
- ◆ Apenas uma instrução é executada de cada vez
- ◆ Cada accão pode ser constituída por um passo ou por um conjunto sequencial de pequenos passos (bloco)
- ◆ Cada passo corresponde sempre a uma operacão elementar

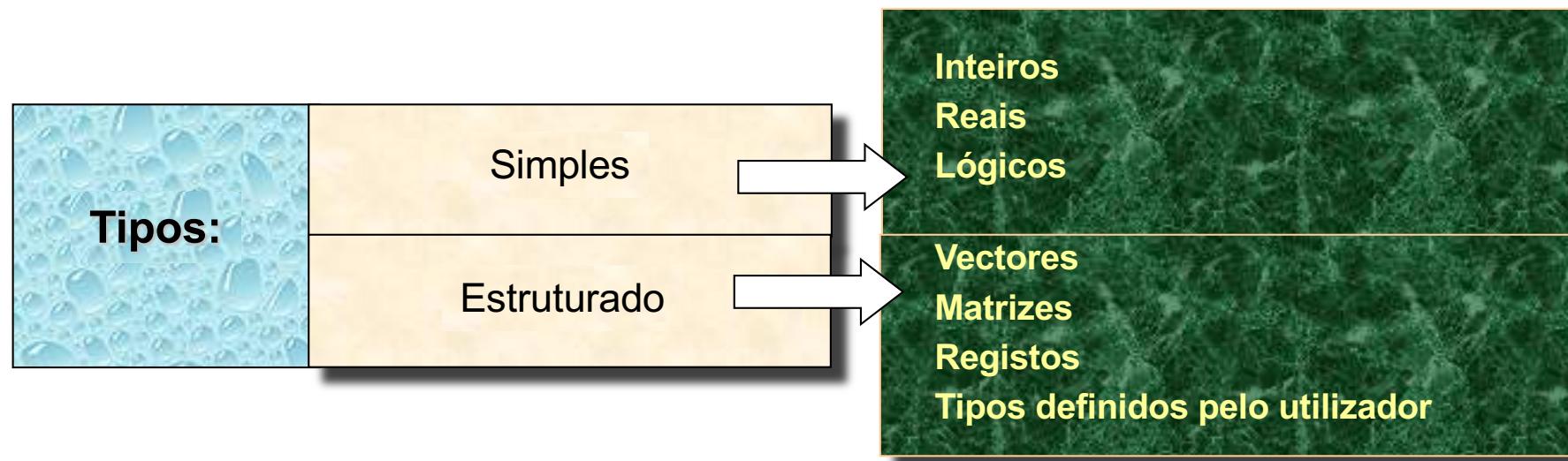
Dados

Dados - constituem a matéria prima para qualquer aplicação ou programa, quer estes venham de origem externa ou interna ao sistema informático. Relativamente aos tipos de dados, estes podem ser:

Constantes - são valores que se mantêm inalterados dentro do programa.



Variáveis - são entidades que estão sempre associadas a identificadores e que podem assumir diferentes valores ao longo do algoritmo.

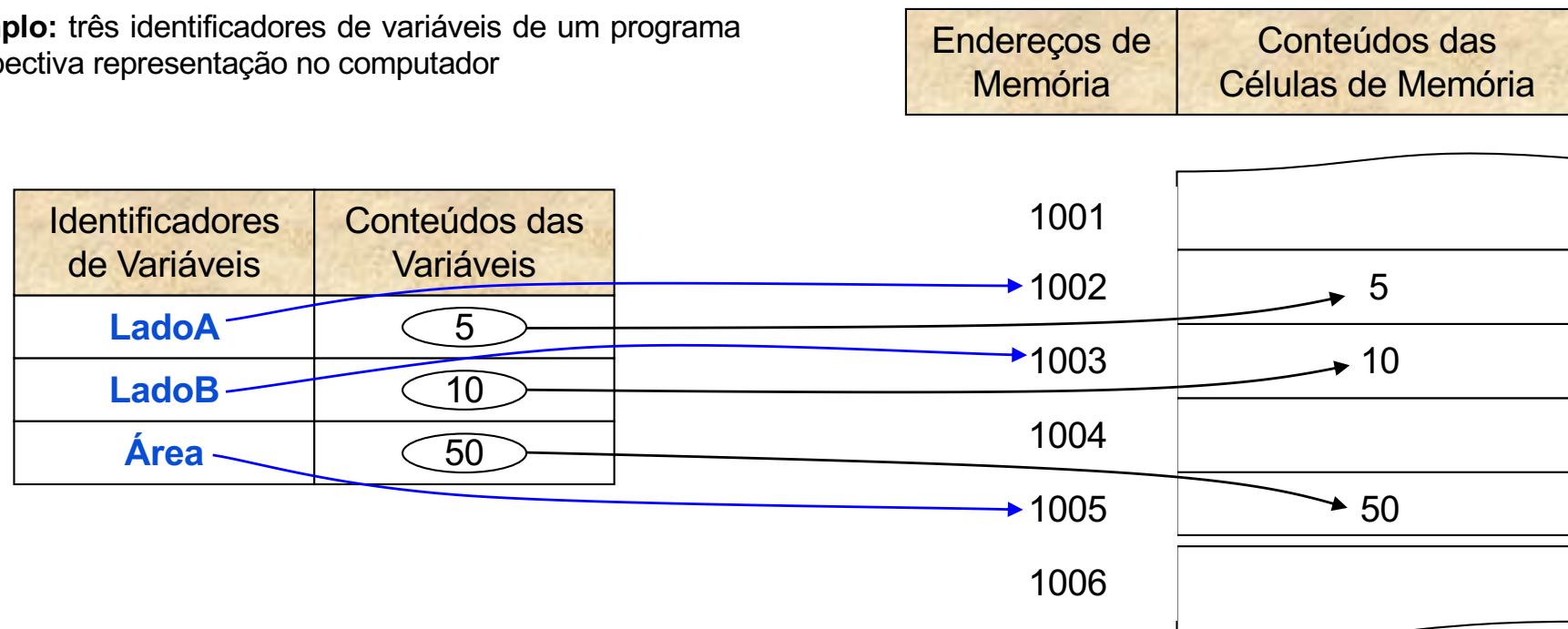


COMENTÁRIO: embora não deva ser uma preocupação primordial a ter em conta na elaboração do algoritmo, na definição de uma variável está subjacente o dimensionamento do espaço físico (tamanho) que esta vai ocupar na memória (1, 2, 4, 8, ... bytes), condicionando os valores máximo e mínimo que pode tomar.

Identificadores

Identificador (de variáveis ou constantes) - são designações simbólicas através das quais passam a ser identificados os dados (tal como uma incógnita numa equação matemática). Para o computador um identificador corresponde ao endereço de memória, onde serão armazenados os valores assumidos pelas variáveis ou constantes. A transposição da designação simbólica para endereço de memória é da responsabilidade do compilador ou interpretador.

Exemplo: três identificadores de variáveis de um programa e respectiva representação no computador



Identificadores (cont.)

- **Regras:**

- Não podem conter espaços;
- Apenas podem conter caracteres alfanuméricos (letras e números) e ...

- **Boas Práticas:**

- nomes sugestivos do seu conteúdo;
- nomes completos (cuidado com as abreviaturas!!!);
- nomes que devem permitir que qualquer pessoa, entenda a sua semântica.

Instruções Básicas

Instruções Básicas - são “frases” que enunciam ou indicam as acções ou operações simples que se pretendem realizar com o algoritmo. As instruções básicas mais frequentes e comuns a qualquer linguagem são:

Escrita ou **Output** - serve para enviar dados (ex.: mensagens ou valores) para um dispositivo de saída (ex.: monitor de vídeo, impressora ou disco).

Num algoritmo esta instrução toma a forma **Escrever (...)** ou **Escreve (...)**.

Leitura ou **Input** - utiliza-se para solicitar uma entrada de dados para a aplicação, normalmente através do teclado, rato ou disco.

Num algoritmo esta instrução toma a forma **Ler (...)** ou **Lê (...)**.

Atribuição - enquanto que numa instrução de leitura a variável recebe um valor proveniente de uma entrada externa, numa atribuição, o valor é proveniente de uma operação interna de processamento. Em termos de escrita de algoritmos, esta instrução é representada por uma seta (\leftarrow), indicando o sentido em que circulam os dados, isto é, identificadorDeVariável \leftarrow valorAAtribuir.

Expressões

Expressão - trata-se de um conjunto de **operandos**, normalmente dados na forma directa ou indirecta, relacionados entre si através de **operadores**.

As expressões, por si sós, não constituem instruções válidas. Estas, em geral, surgem integradas em:

- Instruções de escrita;
- Instruções de atribuição;
- Condições de controlo nas estruturas de decisão ou repetição.

Tipos de operadores

Aritméticos - **Adição (+)**, **Subtracção (-)**, **Multiplicação (*)** ou **Divisão (/)**.

Comparação - **igual (=)**, **menor que (<)**, **menor ou igual (<=)**, **maior que (>)**, **maior ou igual (>=)**, **diferente (<>)**.

Lógicos ou **Booleanos** - **~ Negação**, **E lógico**, **OU lógico**.

Texto - correspondem a operações com cadeias de caracteres (ex.: concatenação).

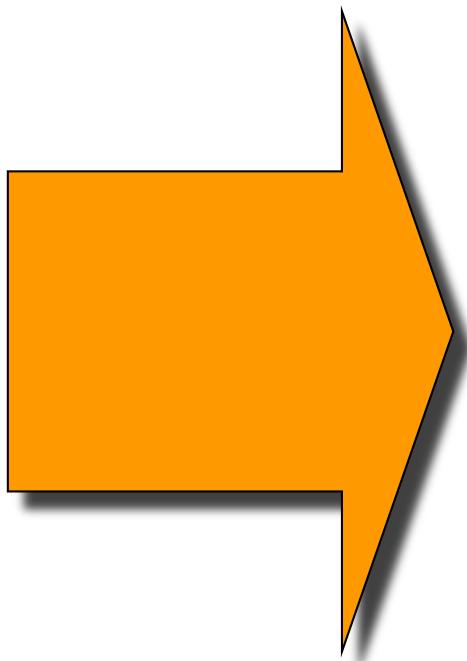
Tipos de expressões:

Numéricas - expressões onde se utilizam apenas operadores aritméticos, sendo os operandos do tipo numérico (inteiros ou reais).

Ex.: `100 * (1+0.15)` ; `quantidade*custoUnidade`; `100 * (custoUnidade - desconto) + 1000`

Booleana - são expressões onde se utilizam operadores de comparação e se espera obter um resultado do tipo lógico (Verdadeiro ou Falso).

Ex.: `x + y > 10`



Sequências de Instruções Simples

Alteração da Sequência por Instrução de Salto

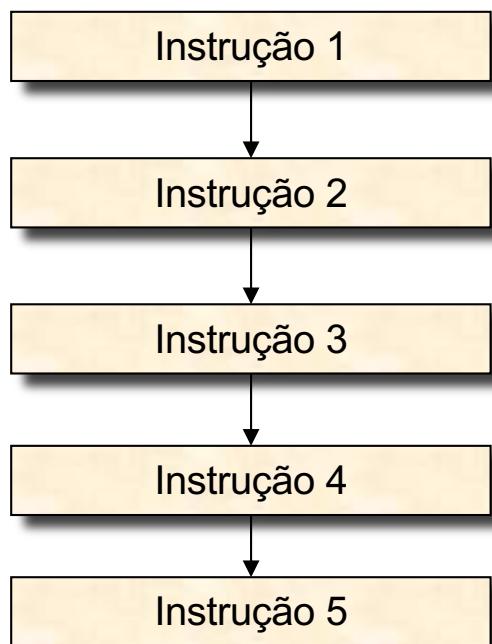
Estruturas de Decisão Condicional

Estruturas de Repetição com Contador

Estruturas de Repetição com Condição

Sequências de Instruções Simples

Sequências de Instruções Simples - nesta estrutura, as acções ou instruções são executadas uma após outra, sem possibilidade de omitir nenhuma. A ordem de execução será exactamente a definida no algoritmo, sem saltos de instruções.

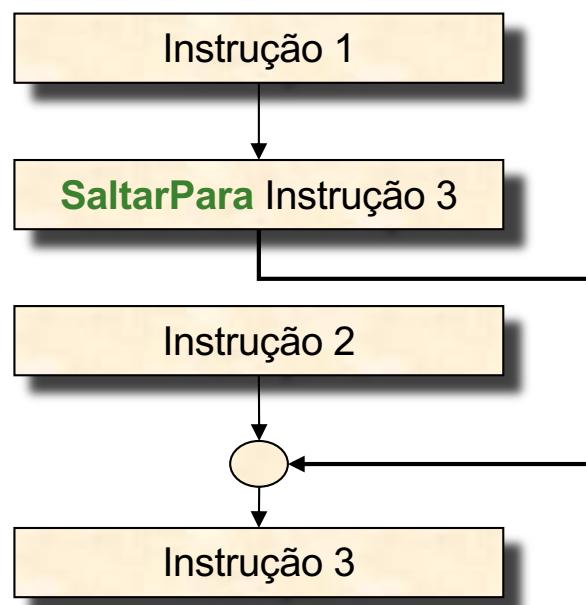


Algoritmo Sequencial Simples

Inst.1: Escrever ("Cálculo da área do rectângulo");
Inst.2: Ler (base);
Inst.3: Ler (altura);
*Inst.4: area ← base * altura;*
Inst.5: Escrever ("A área é:", area);

Alteração da Sequência por Instrução de Salto

Alteração da Sequência por Instrução de Salto - a alteração da ordem de execução das instruções, é feita utilizando a **Instrução de Salto**. Esta instrução remete a continuação da execução de um programa para um outro ponto.



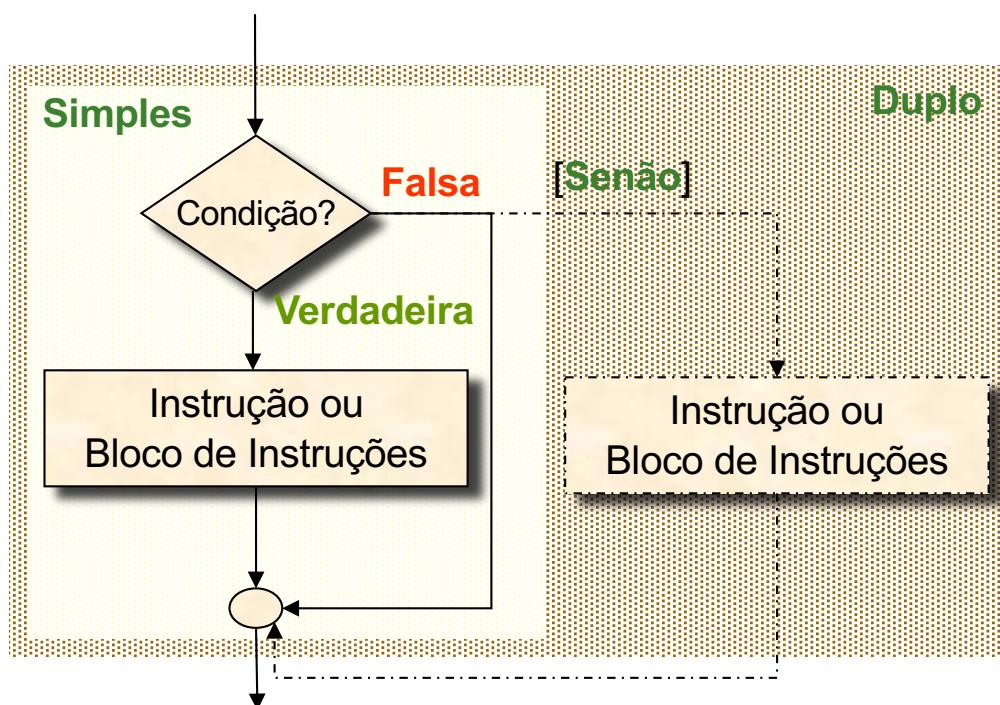
Algoritmo com Instrução de Salto

Inst.1: Escrever (“Cálculo da área do rectângulo”);
Inst.2: Ler (base);
Inst.3: Ler (altura);
Inst.4: SaltarPara Instrução6;
Inst.5: area ← base * altura;
Inst.6: Escrever (“A área é:”, area);

Nota: Embora a maioria das linguagens suporte esta instrução, o seu uso é fortemente desaconselhado nas linguagens estruturadas. E é expressamente proibido nesta UC!!!!

Estruturas de Decisão Condicional: Simples e Dupla

Estruturas de Decisão Condicional (Simples e Dupla) - permitem, com base na análise de uma condição (em geral, o resultado duma expressão booleana), decidir sobre a execução ou não de determinada instrução ou bloco de instruções, ou optar entre duas ou mais instruções alternativas. As estruturas de decisão condicional podem ser do tipo **Simples**, **Duplo** ou **Múltiplo**.



Decisão Simples

SE condição **ENTÃO**
(Condição é Verdadeira)

Instrução ou Bloco de Instruções
FIM-SE

Decisão Dupla

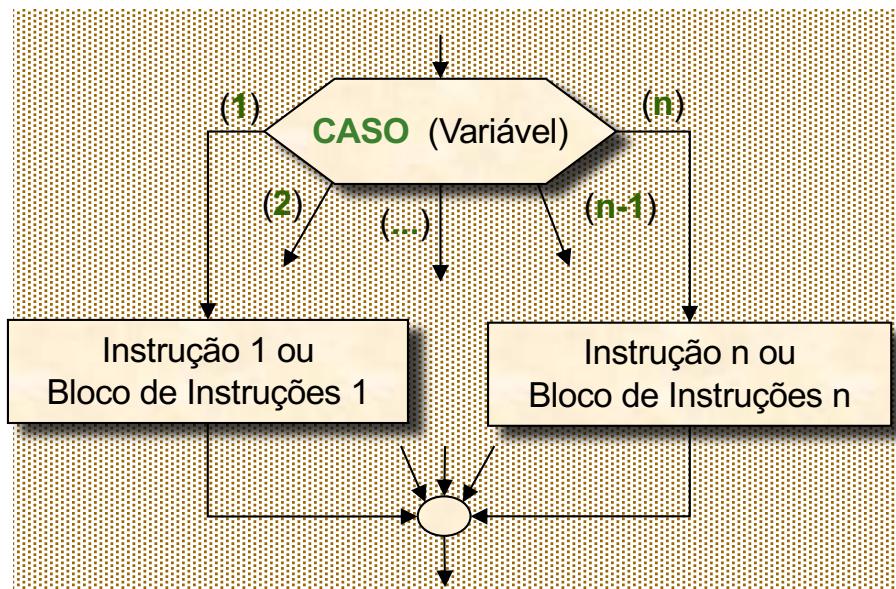
SE condição **ENTÃO**
(Condição é Verdadeira)

Instrução ou Bloco de Instruções
SENÃO

(Condição é Falsa)
Instrução ou Bloco de Instruções
FIM-SE

Estruturas de Decisão Condicional: Múltipla

Estruturas de Decisão Condicional Múltipla - quando é necessário decidir entre **múltiplas** opções possíveis para a instrução ou bloco de instruções a executar, podemos utilizar **várias estruturas condicionais duplas (SE, ENTÃO, SENÃO) encadeadas** ou a **estrutura CASO**.



Condicional Dupla Encadeada

SE condição 1 **ENTÃO**

(Condição 1 é Verdadeira)

Instrução ou Bloco de Instruções

(...)

SENÃO **SE** condição n **ENTÃO**

(Condição n é Verdadeira)

Instrução ou Bloco de Instruções

SENÃO

(Condição n é Falsa)

Instrução ou Bloco de Instruções

FIM-SE

(...)

FIM-SE

Estrutura CASO

CASO Identificador **Variável** **SEJA**

valor-1 **FAZ**: Instrução 1 ou Bloco de Instruções 1

valor-2 **FAZ**: Instrução 2 ou Bloco de Instruções 2

(...)

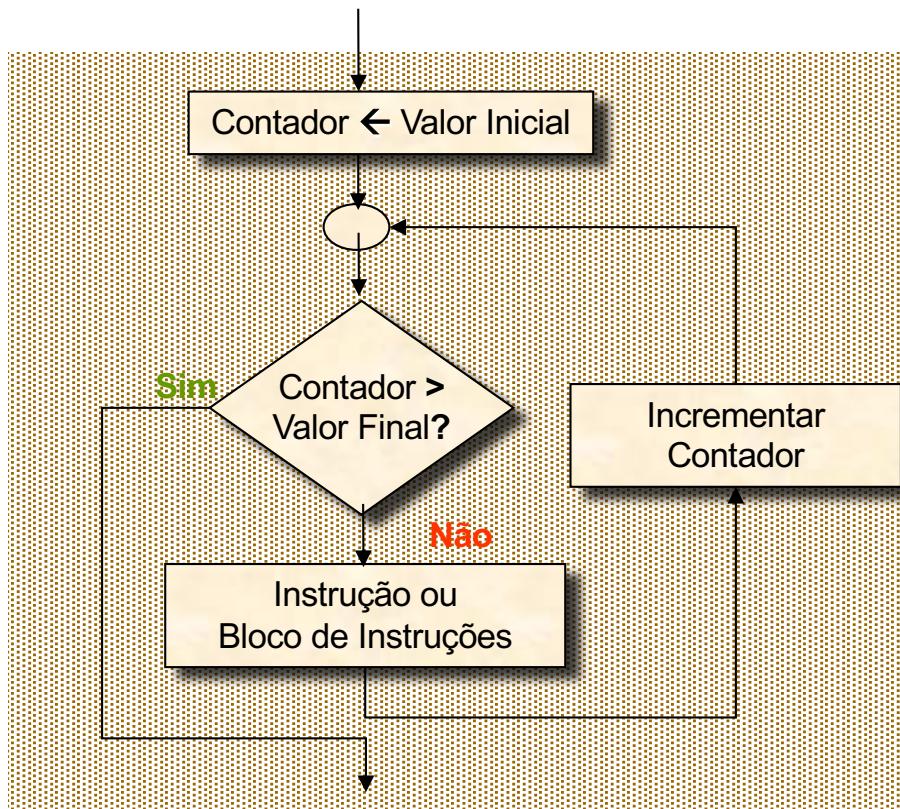
valor-n **FAZ**: Instrução n ou Bloco de Instruções n

OUTROS FAZ: Instrução ou Bloco de Instruções

FIM-CASO

Estruturas de Repetição com Contador

Estruturas de Repetição com Contador - Esta estrutura permite repetir a mesma instrução ou bloco de instruções um dado número de vezes e de um modo controlado, através de uma **Variável de Controlo** ou **Contador**. Cada repetição é designada por **Ciclo**.



Estrutura DE ... ATÉ

DE contador ← valor inicial **ATÉ** valor final **FAZ**
Instrução ou Bloco de Instruções
FIM-DE

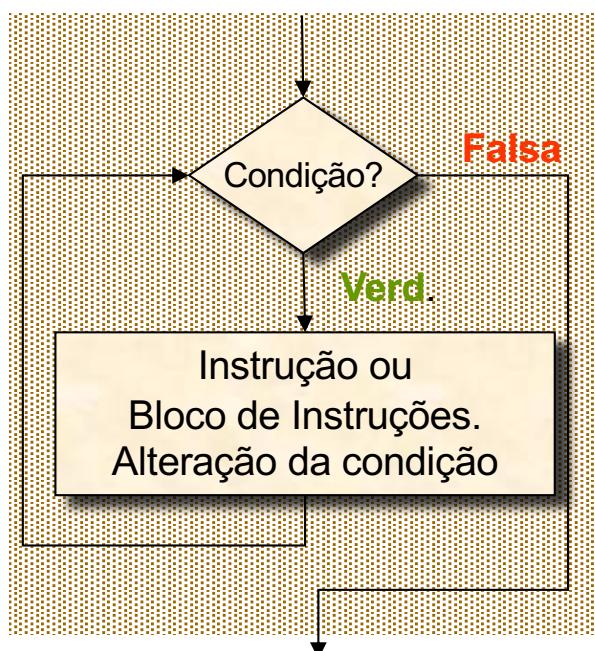
Exemplo:

DE contador ← 0 **ATÉ** 5 **FAZ**
Escrever ("Estrutura de repetição");
FIM-DE

Estruturas de Repetição com Condição

Estruturas de Repetição com Condição - estas diferem das anteriores porque utilizam uma condição (em geral, uma expressão booleana), em vez de um contador. Utilizam-se quando se desconhece o número de ciclos de repetição a executar. Dentro do bloco de instruções terá que ocorrer uma alteração que interfira com a condição, caso contrário, entrará em ciclo infinito.

Estrutura ENQUANTO



Estrutura ENQUANTO

ENQUANTO Condição **FAZ**

(Condição Verdadeira)

Instrução ou Bloco de Inst.

Alteração da Condição

FIM-ENQUANTO

Estrutura REPETIR ... ENQUANTO

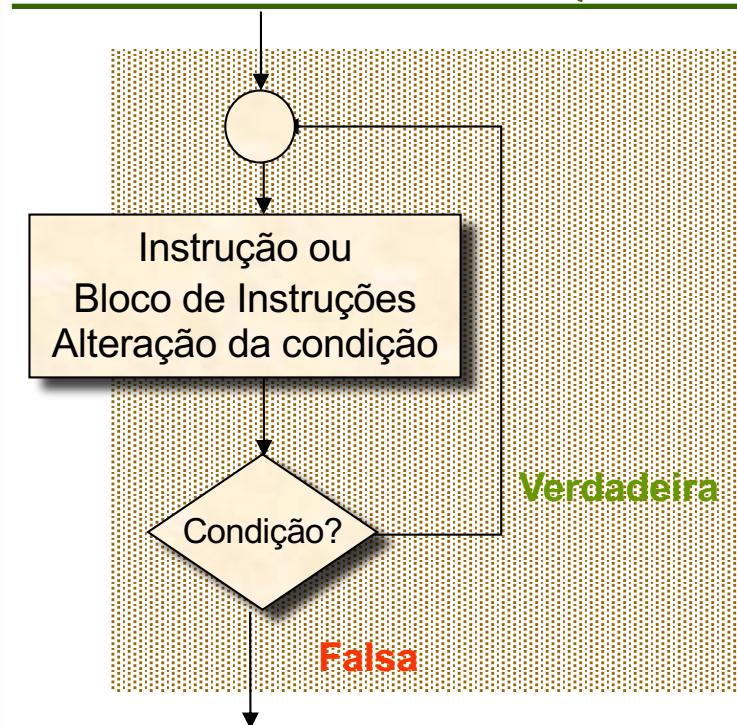
REPETIR

Instrução ou Bloco de Inst.

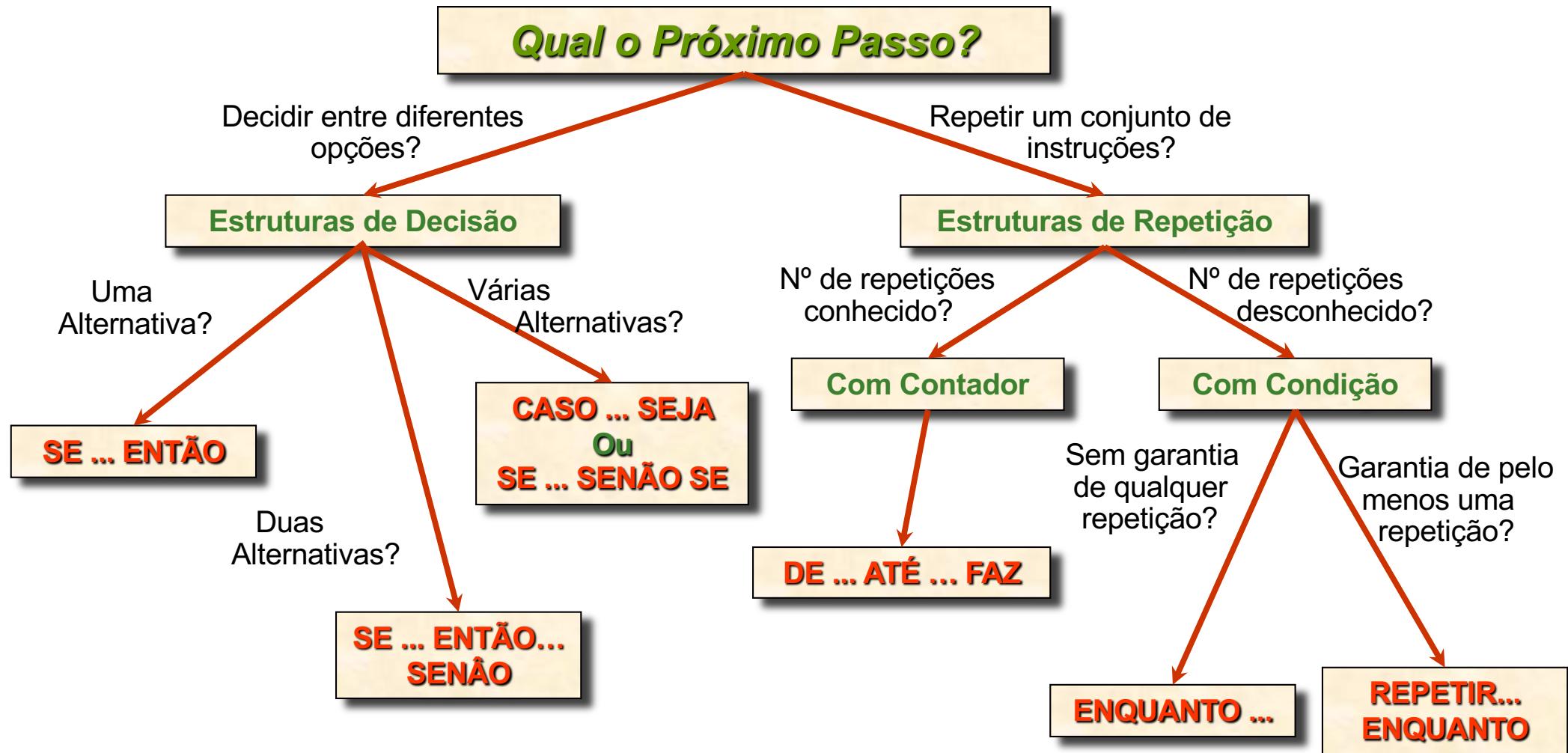
Alteração da Condição

ENQUANTO Condição *(enquanto for verdadeira)*

Estrutura REPETIR ... ENQUANTO



Qual a estrutura de controlo a utilizar



Sub-programas

Sub-programa - trata-se duma pequena parte dum algoritmo que reúne um certo número de instruções às quais é atribuído um **identificador**. Estas são normalmente escritas num local diferente da sequência normal do algoritmo, podendo ser chamadas através do respectivo **identificador**, num ou em vários locais desse algoritmo.

Vantagens

- Facilitam a abordagem modular dum algoritmo, dividindo em pequenos subprogramas simples um problema complexo;
- Ao utilizar um subprograma em diferentes locais do algoritmo, evocando apenas o seu identificador, podemos evitar as repetições de código, resultando daí nítidas vantagens na elaboração do algoritmo e posteriormente, na sua manutenção;
- Permitem a aplicação duma mesma funcionalidade do subprograma, em contextos diferentes;
- Facilitam a leitura do algoritmo, pois este torna-se mais pequeno;
- Uma vez que o identificador pode indicar a funcionalidade associada a um subprograma, pode também fornecer-nos a abstracção do seu conteúdo (conhecer o que faz, mas não conhecer como o faz);
- Possibilitam a divisão das tarefas por várias pessoas, não necessitando cada uma delas de ter conhecimento da globalidade do algoritmo.

Regras de estilo na escrita de um algoritmo

Na escrita de um **Algoritmo** ou de um **Programa** é conveniente seguir algumas regras, na distribuição do texto pela página ou ao longo de uma linha, para facilitar a sua leitura.

Desde que se respeitem as regras sintácticas estipuladas, existe liberdade total na distribuição do texto, pelo que convém utilizar só as formas de disposição do texto que beneficiem a sua legibilidade.

- Utilizar **tabulações** ou **avanços de linha** - deslocamento das linhas em relação à margem, formando, reentrâncias, de forma a agrupar e destacar determinadas partes (ver os exemplos das estruturas de controlo);
- Pôr em destaque as **palavras-chave** - utilização de **negrito**, LETRA MAIÚSCULA ou sublinhar. Utilizar **identificadores** que sejam sugestivos daquilo que pretendem designar;
- Não colocar mais do que **uma instrução por linha**;
- Não utilizar **linhas muito extensas**. Quando ocorre uma linha longa, é preferível subdividi-la em várias;
- Inserção de **comentários explicativos** - decorrido algum tempo, um algoritmo ou um programa torna-se mais fácil de entender por outras pessoas ou pelo próprio programador, se forem anexados comentários que descrevam as opções tomadas (devem evitar-se os comentários óbvios).

Alguns problemas simples...

- Calcular a **área** de um **trapézio rectângulo** $A = (a + b) / 2 * h$
- Aplicação do **Teorema de Pitágoras** ($h^2 = x^2 + y^2$)
- **Número de segundos** entre **dois valores de horas**
- **Simulação de uma calculadora** com as operações $+, -, *, /$
- Aplicação da **fórmula resolvente de equações quadráticas**

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Calcular a **classificação** na UC de **Algoritmos e Programação**