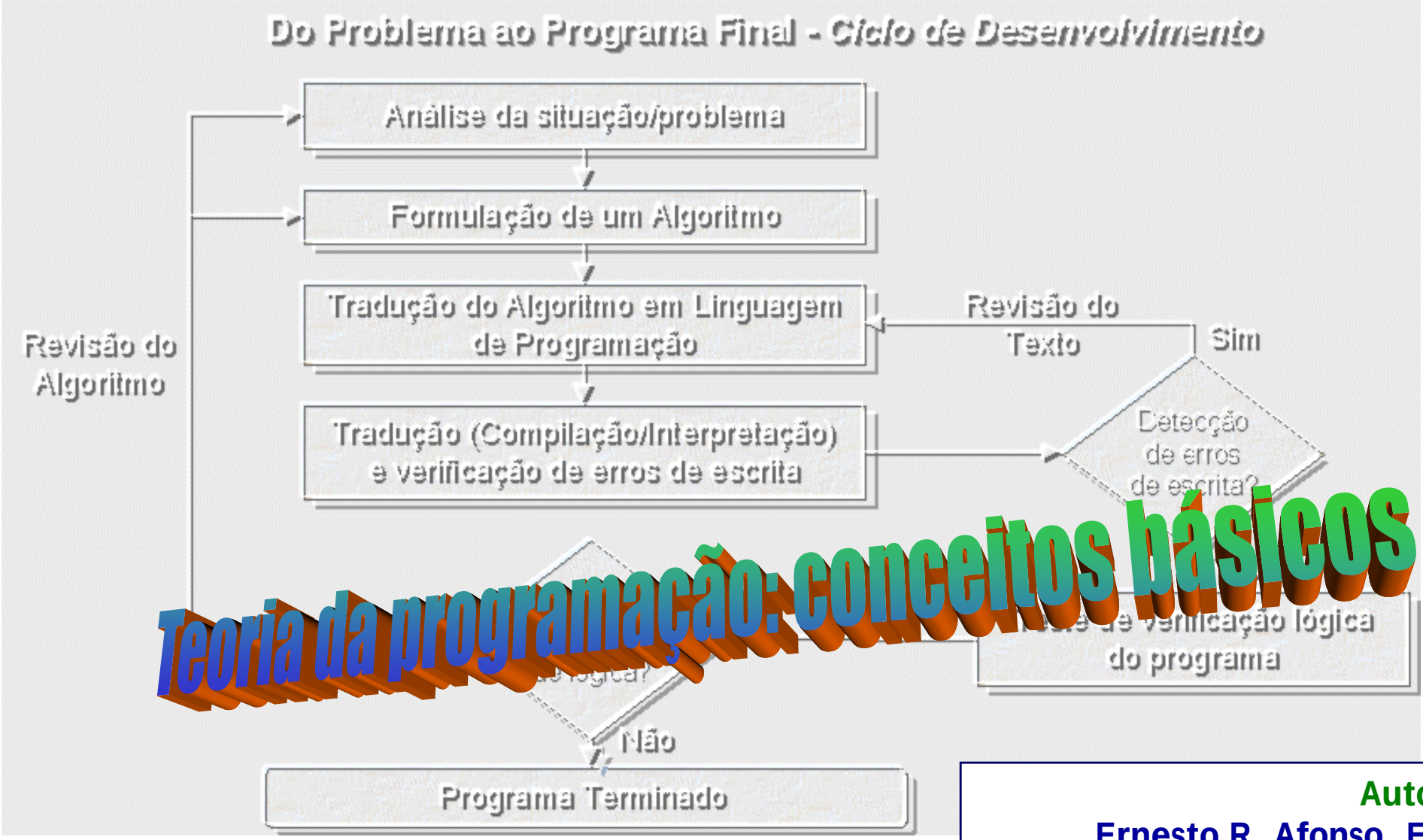


Teoria da programação: conceitos básicos

Autoria:
Ernesto R. Afonso, Eng.º
Manuel A. E. Baptista, Eng.º

Teoria da programação: conceitos básicos

2.1. Introdução à programação e seus objectivos

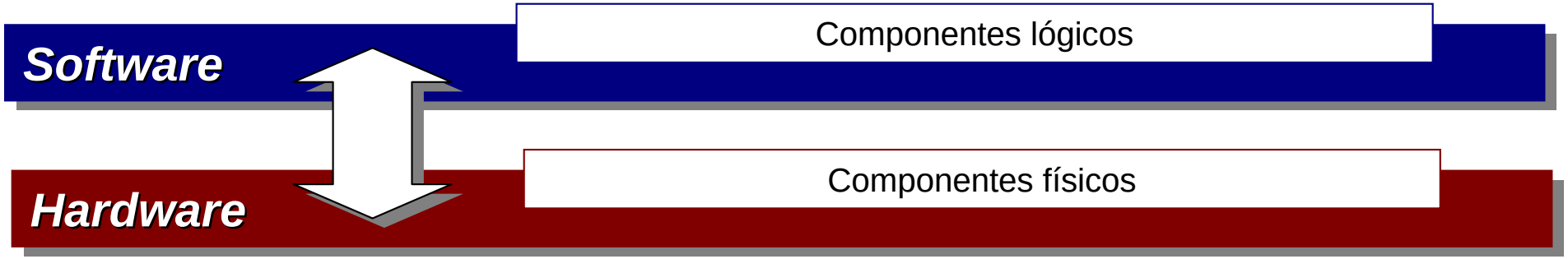
2.2. Linguagens de programação

2.3. Metodologia de programação

2.4. Construção dum algoritmo

2.1. Introdução à programação e seus objectivos

2.1.1. Dando vida ao *Hardware*: O *Software*



O *Software* permite que o *Hardware*, realize as tarefas que conduzem à resolução de problemas. Este não se trata dum elemento único, mas duma estrutura em camadas.

CAMADAS		
	Programas de Aplicação	***Software*** de aplicação
	Sistema Operativo	***Software*** do sistema

2.1.2. Interacção do Utilizador/Programador com o Hardware

Utilizadores

"O diálogo entre o Homem e a Máquina"

Programadores

Software de Aplicação	Programas de Aplicação	Processador de texto Folha de Cálculo etc.	Software de Programação
Software do Sistema	Sistema Operativo	Interface de Comandos Núcleo (*Kernel*)	

Hardware

2.1.3. Linguagem de programação

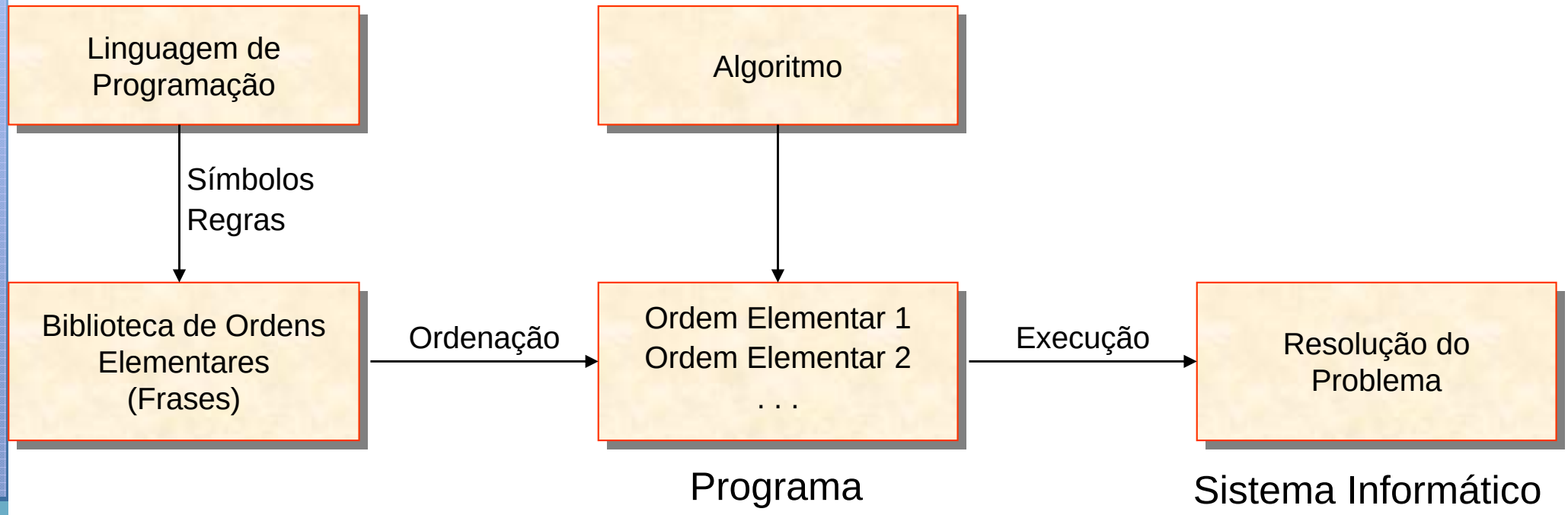
Linguagem de programação - trata-se dum sistema de escrita, que através dum um conjunto de símbolos convencionados, permite explicitar as operações a executar por um sistema informático. Esta é constituída por uma:

- **Componente Semântica** - utilização dum conjunto de termos, palavras ou sinais, que transmitem um determinado significado interpretável 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 para o CPU;

2.1.4. O Programa

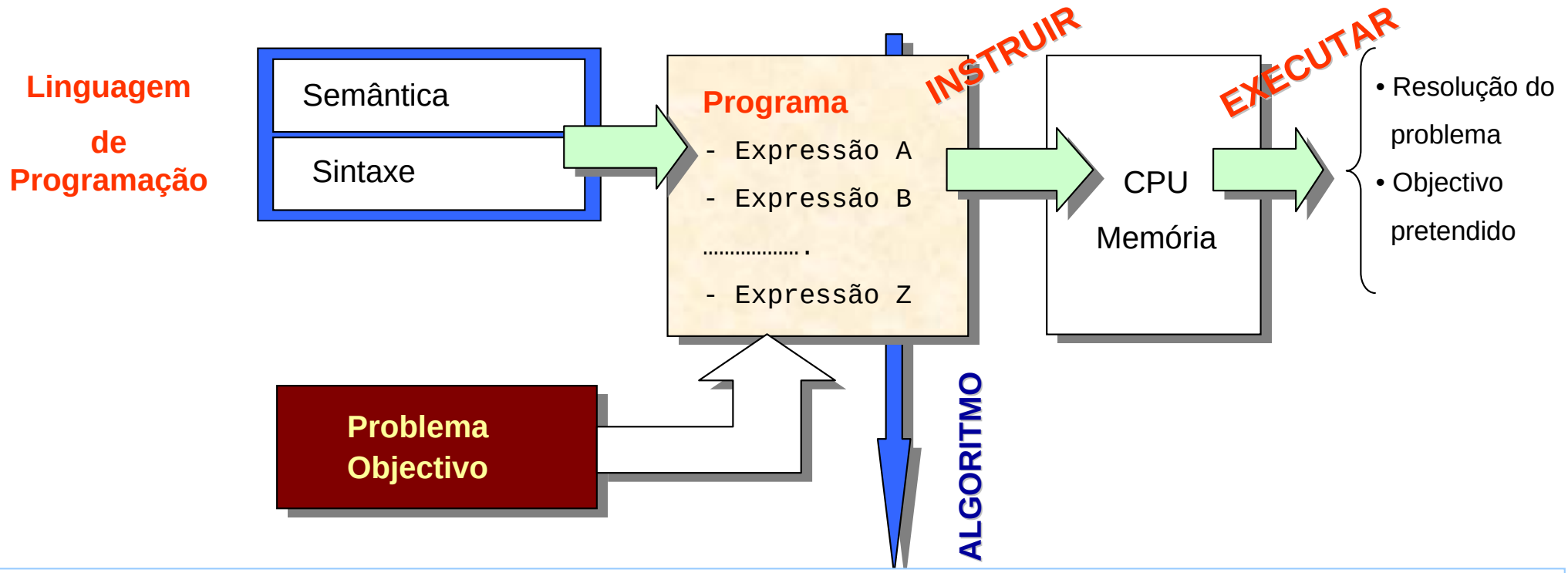
Programa - trata-se dum conjunto de expressões, baseadas nas componentes semânticas e sintácticas da linguagem de programação, que representam as instruções que o CPU deve executar para concretizar os objectivos pretendidos para a resolução dum dado problema.



2.1.5. O Algoritmo

Algoritmo - trata-se duma forma ou 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**, são normalmente a tradução dum problema ou conjunto de problemas, seguindo um determinado **Algoritmo**, com vista à sua resolução pelo sistema informático;

2.2. Linguagens de programação

2.2.1. Classificação das linguagens: níveis

Níveis das Linguagens de Programação - expressão que pretende traduzir uma maior ou menor proximidade 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 palavra.

2.2.2. 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 ← C+D
2	011001010101	R, SUB B,S	- Fazer: R ← B-S
3	001010101110	E, ADD A,R	- Fazer: E ← A+R
4	010000000000	HLT	- Finalizar

2.2.3. Linguagens de Alto Nível

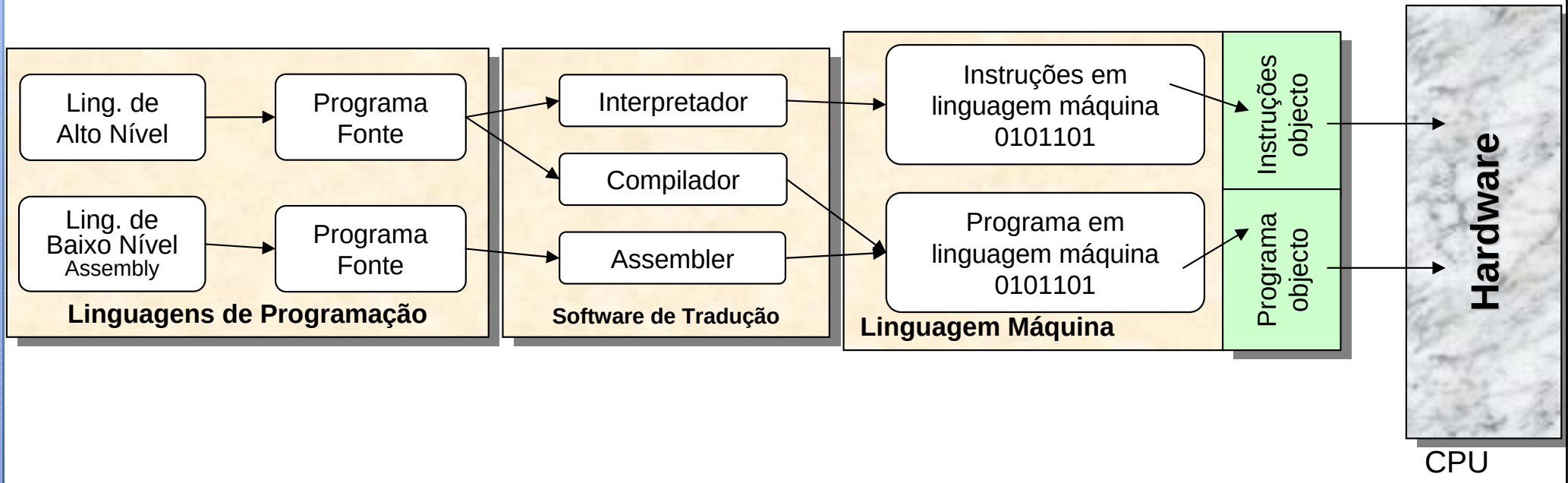
Numa **Linguagem de Alto Nível**, o programador utiliza um conjunto mais abrangente de instruções que escreve nos seus programas, tendo apenas que saber escrevê-las correctamente e dar-lhe uma ordenação lógica, com vista a obter o resultado. Surgiram em meados dos anos 50 com o ***FORTRAN***, e no início 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 "**Smaltalk**")*

2.2.4. Software de Tradução

Software de Tradução - Para converter um programa escrito em qualquer uma das linguagens de alto nível para a linguagem máquina não basta pedir ao computador que leia o programa, é necessário um *software* específico para fazer a tradução para os "0" e "1" inteligíveis ao computador.



2.2.4. 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 e independentemente do *software* que efectua a tradução.

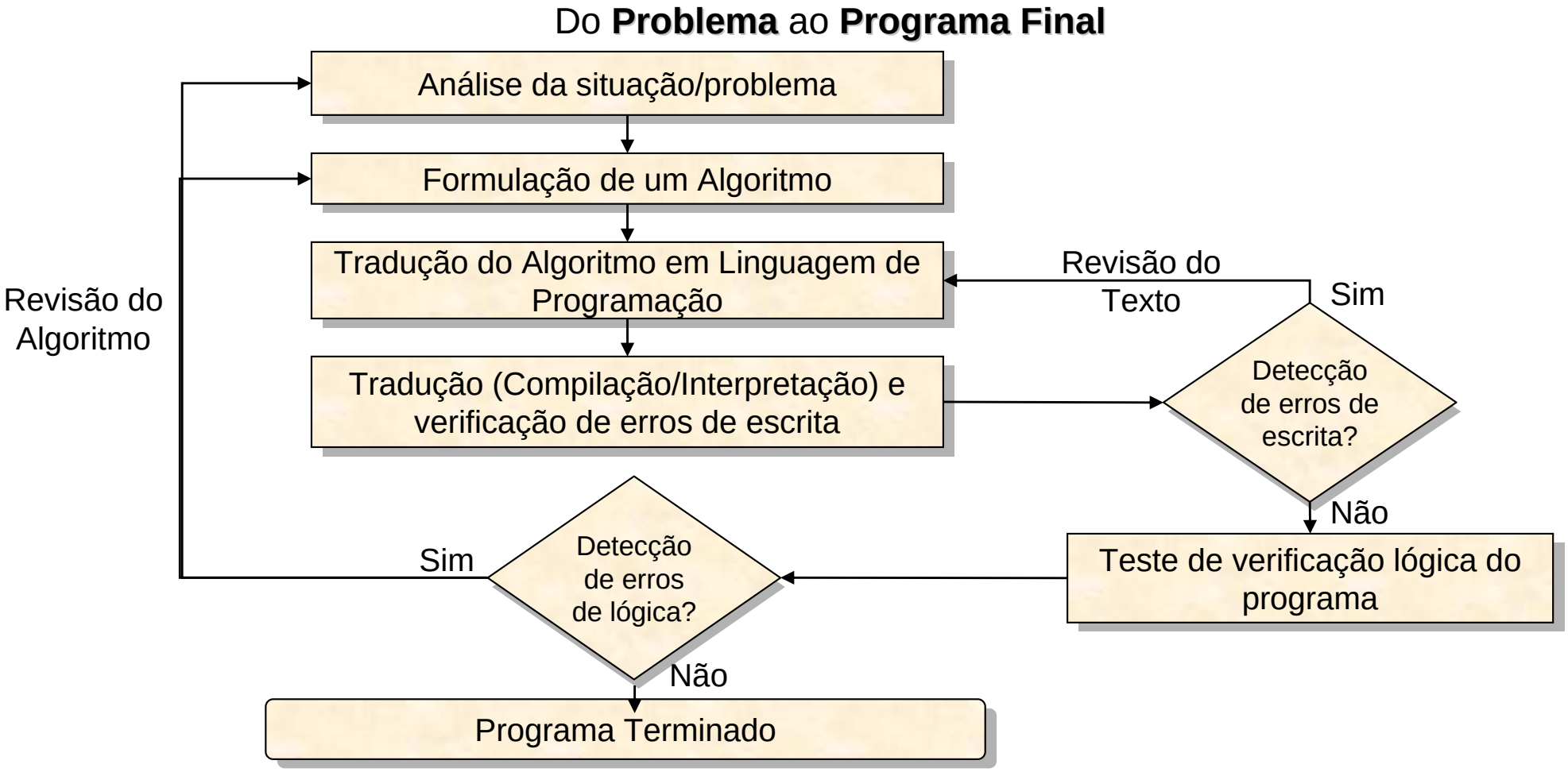
Tipos de códigos

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

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

2.3. Metodologia de programação

2.3.1. Ciclo de desenvolvimento estruturado



2.3.2. 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:	Com que dados vamos trabalhar.
Saídas:	Que dados deveremos obter como resultado.
Definição do problema:	Que processos deveremos utilizar para produzir os resultados.

Exemplo: Calcular o preço total de um terreno com forma rectangular.

Entradas:	`Preço por metro quadrado do terreno` `Medida do lado A, Medida do lado B (em metros).`
Saídas:	`Preço total do terreno.`
Definição do problema:	`Área = Lado A x Lado B` `Preço total do terreno = Área x Preço por metro quadrado`

2.3.3. Formulação do Algoritmo

Algoritmo - além da forma ou formula de resolução do problema, este é também 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 a soma, sub., mult. e divisão.
Resolver uma raiz quadrada ou uma equação de 2º grau.

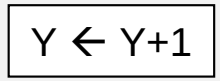
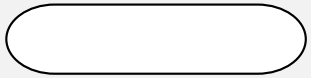
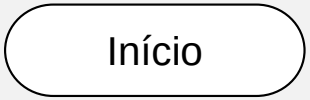
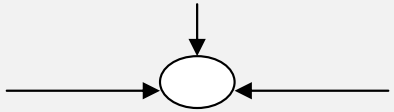
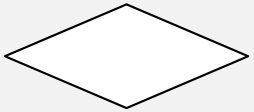
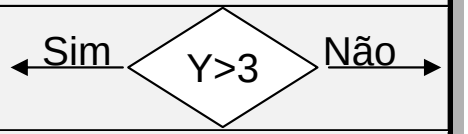
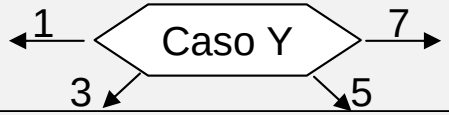
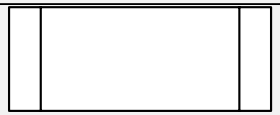
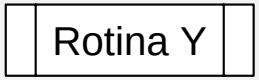
2.3.4. Formas de representação de algoritmos

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.

2.3.5. Fluxogramas: simbologia

Símbolos	Significado	Exemplos
	Processamento em geral	
	Leitura/Escrita de dados	
	Inicio/Fim de processamento	
	Linha de fluxo	
	Conector de fluxos	
	Decisão Condicional	
	Escolha Múltipla	
	Subprograma	

2.3.6. 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.

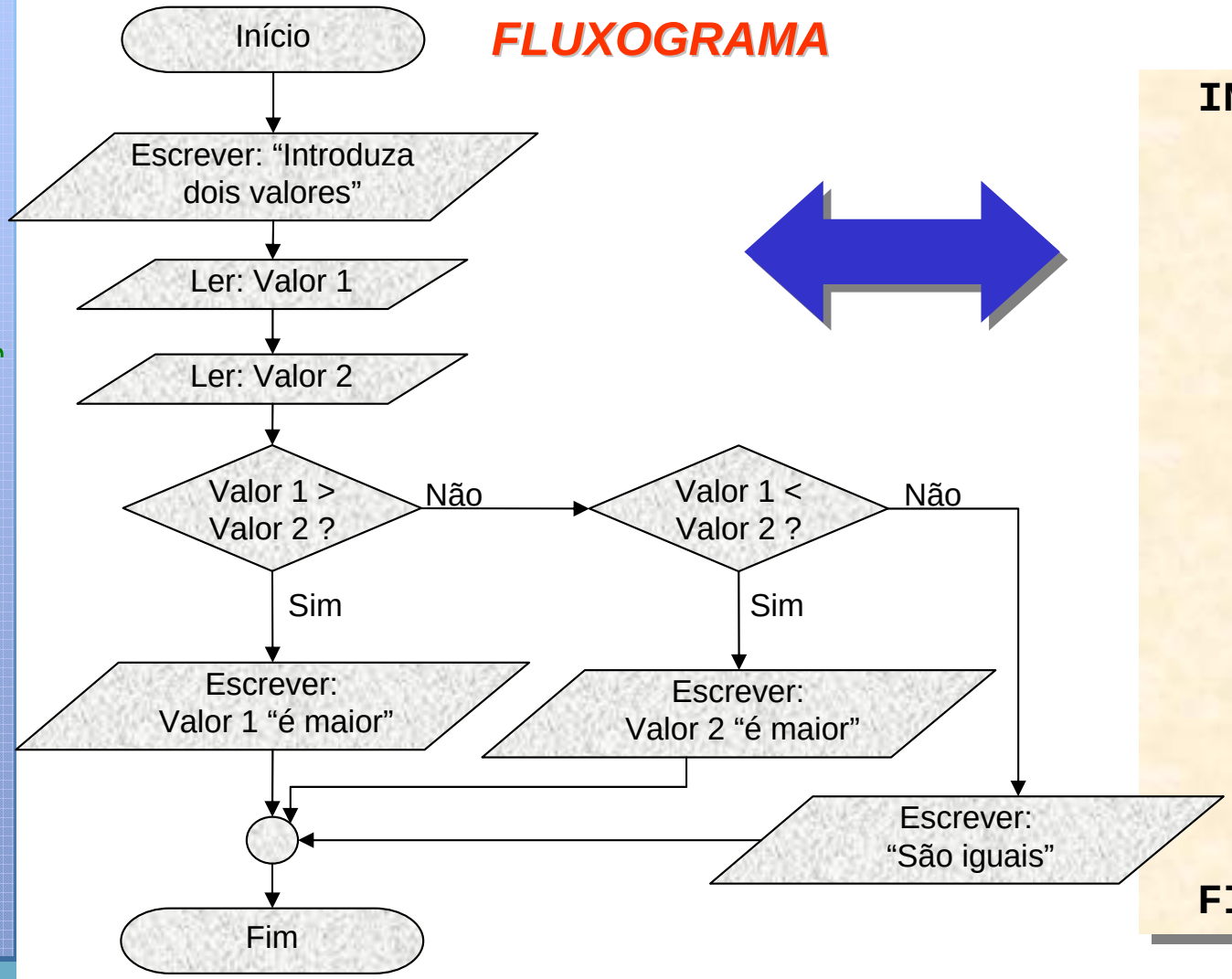
Informal - deixa de parte o formalismo rígido de uma linguagem de programação simplificando a realização do algoritmo.

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 á complexidade da implementação, sendo ideal para a representação de algoritmos mais complexos.

2.3.7. Fluxograma vs Pseudo-código (exemplo)



PSEUDOCÓDIGO

```
INÍCIO
   Escrever ("Introduza dois valores")
   Ler (Valor 1)
   Ler (Valor 2)
   SE Valor 1 > Valor 2 ENTÃO
      Escrever ( Valor 1,  "é maior")
   SENÃO
      SE Valor 1 < Valor 2 ENTÃO
         Escrever ( Valor 2,  "é maior")
      SENÃO
         Escrever ("São iguais")
      FIM SE
   FIM SE
FIM
```

2.4. Construção dum algoritmo

2.4.1. 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.

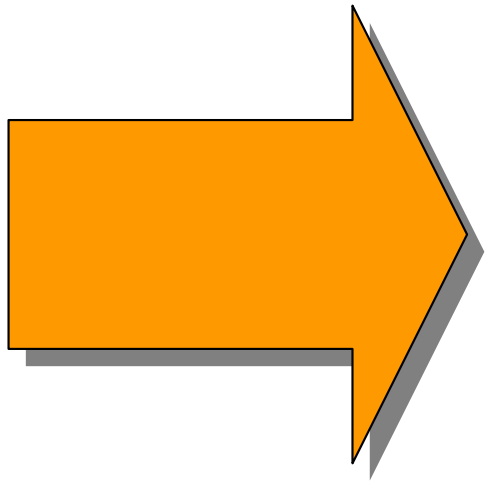
***Top-down* ou Descendente** - é um método de abordagem dos problemas em que primeiro se procuram identificar os pontos essenciais e mais gerais e só depois as componentes mais particulares, em níveis sucessivos e mais concretos, até ao nível do pormenor.

(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.

2.4.2. Componentes fundamentais



Dados

Instruções Básicas

Expressões

Estruturas de Controlo

Sub-programas

Algumas considerações...

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

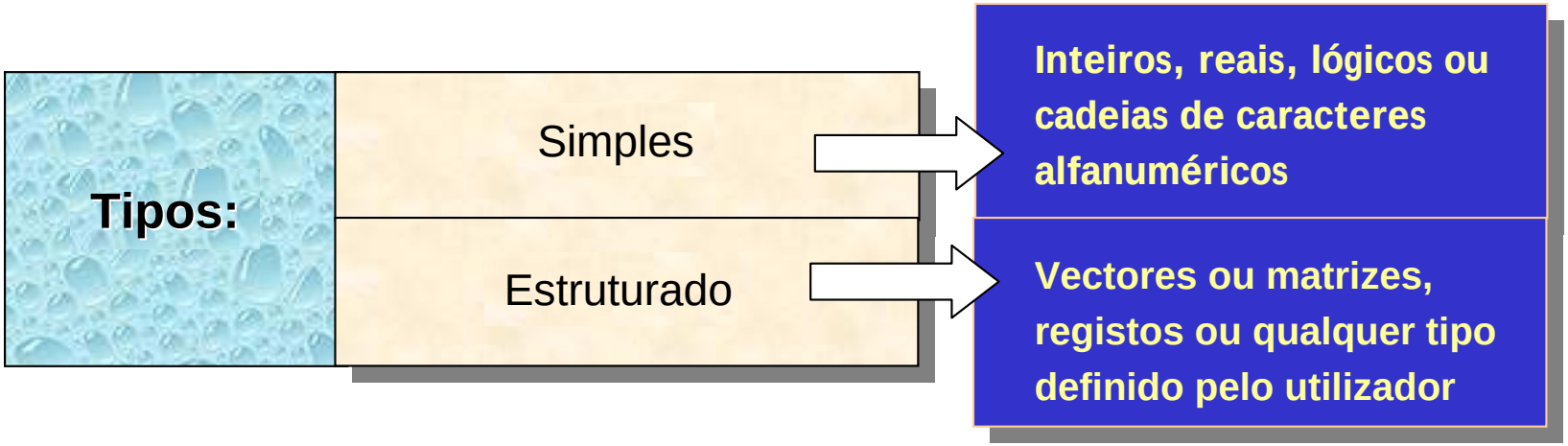
2.4.2.1. Dados

Dados - constituem a matéria prima para qualquer aplicação ou programa, quer estes tenham 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.

Quanto à utilização de identificadores:	Dados directos
	Dados indirectos
Quanto ao tipo:	Simples
	Estruturado

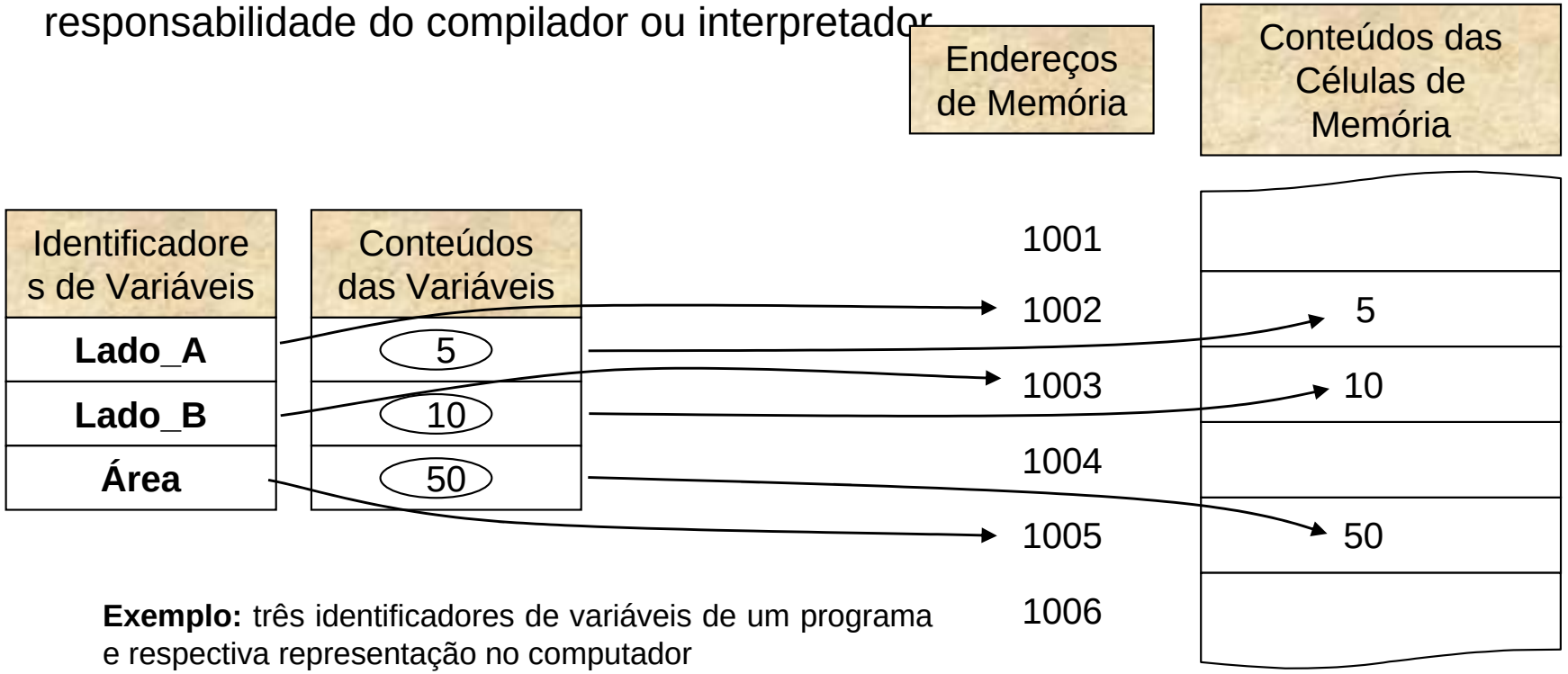
Variáveis - são entidades que estão sempre associadas a identificadores e que podem assumir diferentes valores ao longo do algoritmo.



Comentário: Embora não deva ser uma preocupação primordial a ter em conta na elaboração do algoritmo, na definição de um 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.

2.4.2.2. 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

2.4.2.3. 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 algoritmos 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 do teclado, rato ou disco.

Num algoritmos 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 (←) indicando o sentido em que circulam os dados, isto é, **identificador de variável** ← **valor a atribuir**.

2.4.2.4. Expressões

Expressão - trata-se 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:

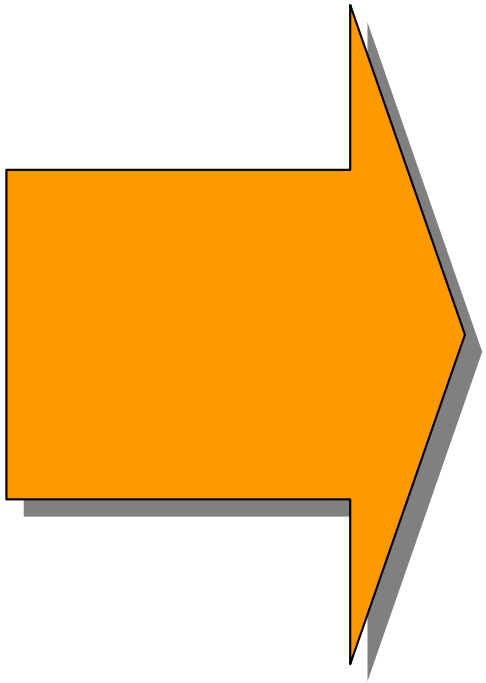
Numéricas - expressões onde se utilizam **apenas operadores aritméticos**, sendo os **operandos do tipo numérico** (inteiros ou reais).

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

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

Ex.: `prev > custo*1.15`

2.4.2.5. Estruturas de Controlo

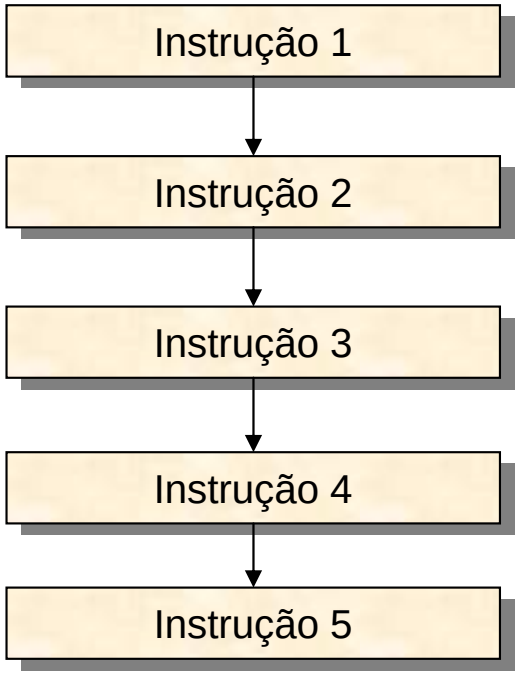


- Sequências de Instruções Simples.
- Alteração da Sequência por Instrução de Salto.
- Estruturas de Decisão Condicional.
- Estruturas de Repetição com Contador.
- Estruturas de Repetição com Condição.

2.4.3. Estruturas de Controlo

2.4.3.1. 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 mesma 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)

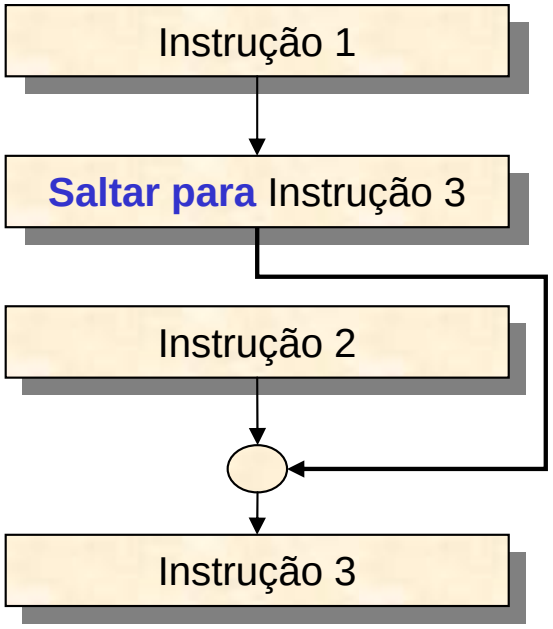
Inst.4: **`area ← base * altura`**

Inst.5: **`Escrever`** (“A área é:” area)

2.4.3.2. 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, remete a continuação da execução de um programa para um outro ponto.

Nota: Embora a maioria das linguagens suporte esta instrução, a sua utilização é fortemente desaconselhada nas linguagens estruturadas.

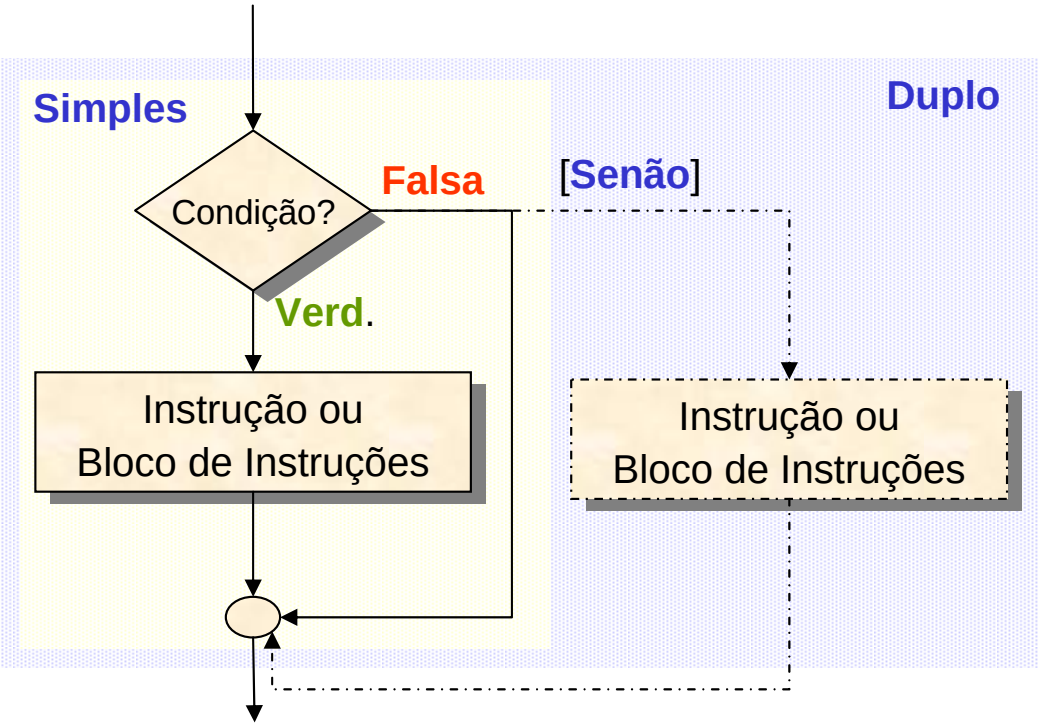


Algoritmo com Instrução de Salto

Inst.1: **Escrever** ("Cálculo da área do rectângulo)
Inst.2: **Ler** (base)
Inst.3: **Ler** (altura)
Inst.4: **Saltar para** ***Instrução 6***
Inst.5: **area ← base * altura**
Inst.6: **Escrever** ("A área é:" area)

2.4.3.3. 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

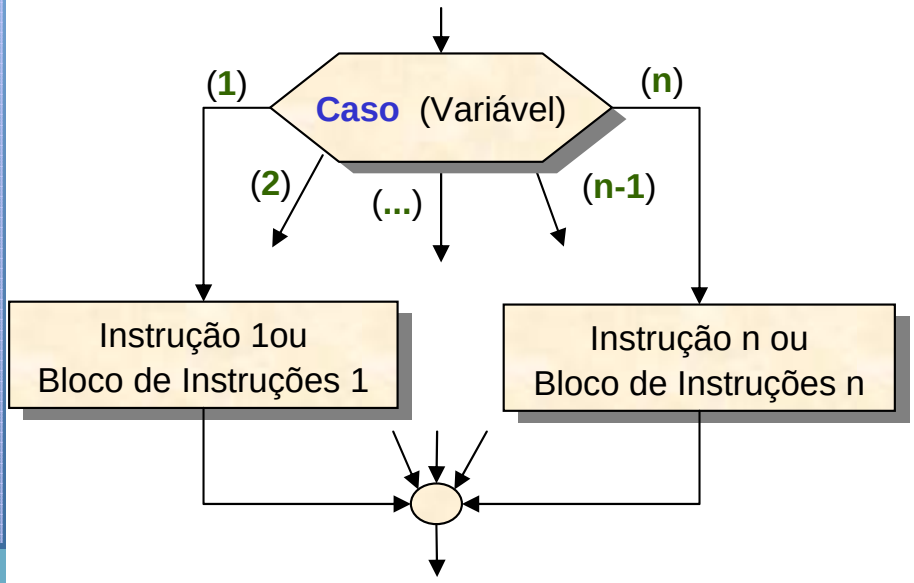
```
SE condição ENTÃO
    (Condição é Verdadeira)
     Instrução ou Bloco de Instruções
FIM SE
```

Decisão Dupla

```
SE condição ENTÃO
    (Condição é Verdadeira)
     Instrução ou Bloco de Instruções
SENÃO
    (Condição é Falsa)
     Instrução ou Bloco de Instruções
FIM SE
```

2.4.3.4. Estruturas de Decisão Condicional: Múltipla

Estruturas de Decisão Condicional Múltiplas - 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

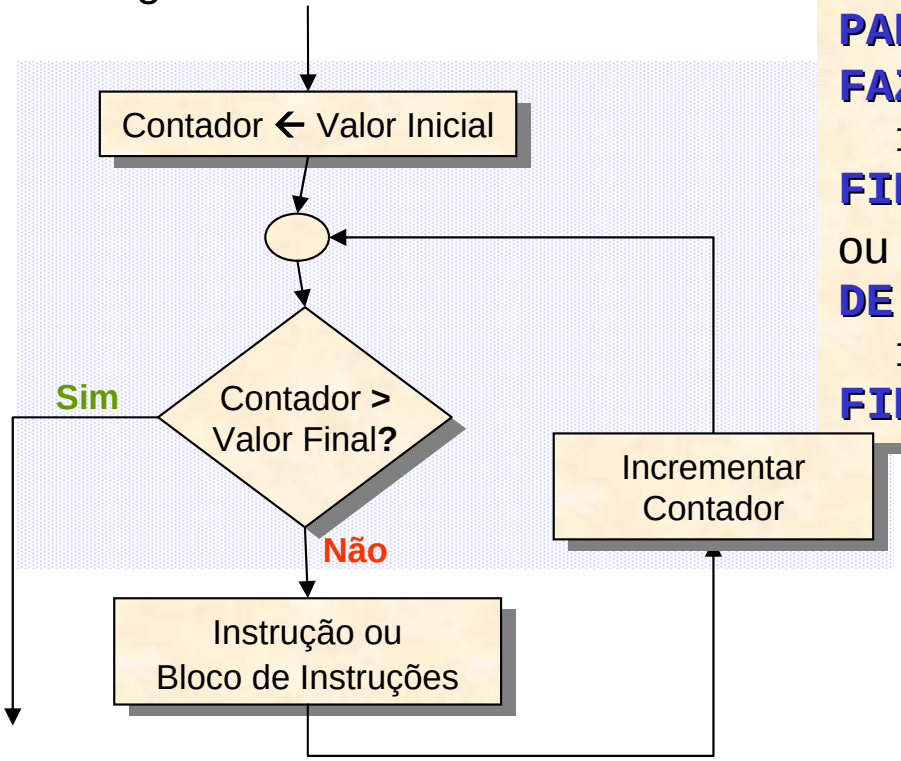
```
SE condição 1 ENTÃO
    (Condição 1 é Verdadeira)
     Instrução ou Bloco de Instruções
( … )
SENÃO SE condição n ENTÃO
    (Condição n é Verdadeira)
     Instrução ou Bloco de Instruções
SENÃO
    (Condição n é Falsa)
     Instrução ou Bloco de Instruções
FIM SE
```

Estrutura CASO

```
CASO Ident. Var
  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
  SENÃO FAZ: Instrução ou Bloco de Instruções
FIM de CASO
```

2.4.3.5. Estruturas de Repetição com Contador

Estruturas de Repetição com Contador - Esta estrutura permite repetir a mesma instrução ou bloco de instruções um número fixo definível de vezes e de um modo controlado, socorrendo-se de uma **Variável de Controlo** ou **Contador**. A cada repetição é designado um **Ciclo**.



Estrutura PARA ou DE

```
PARA contador DESDE valor inicial ATÉ valor final
FAZ
   Instrução ou Bloco de Instruções
FIM PARA
```

ou

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

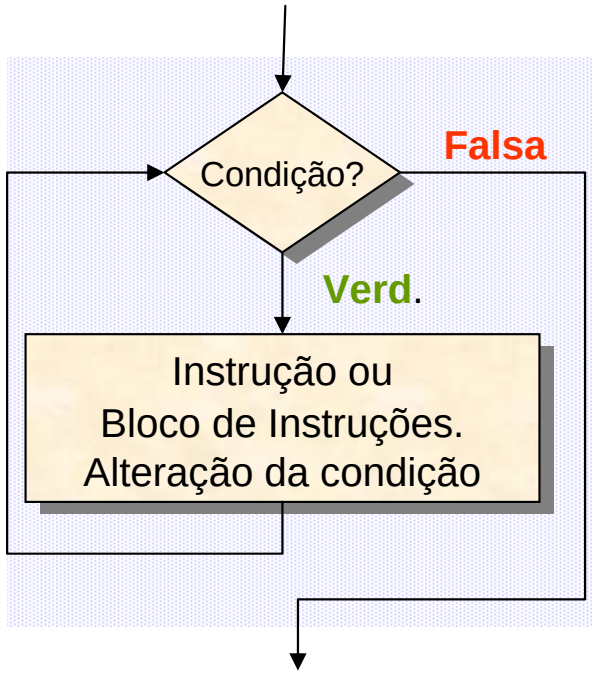
Exemplo:

```
DE contador ← 0 ATÉ 5 FAZ
   Escrever ("Estrutura de repetição")
FIM DE
```

2.4.3.6. 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, entraria em ciclo infinito.

Estrutura ENQUANTO



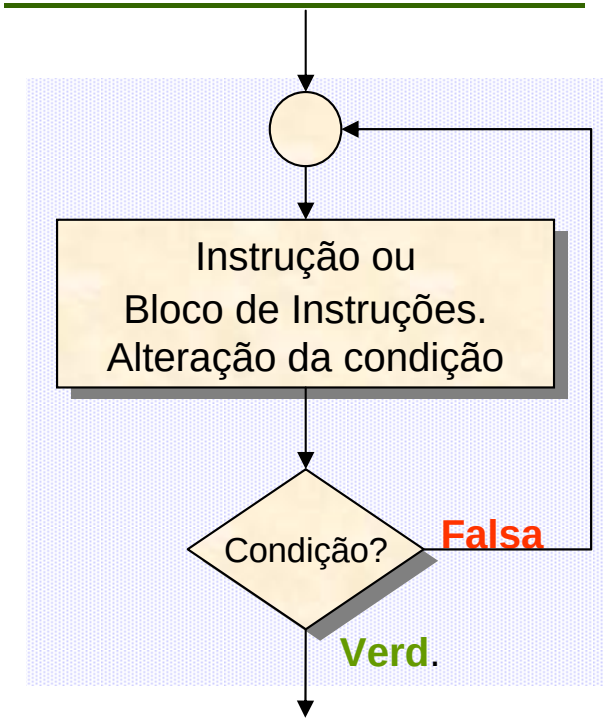
Estrutura ENQUANTO

```
ENQUANTO Condição FAZ
   (Condição Verdadeira)
     Instrução ou Bloco de Instruções
     incluindo alteração da Condição
FIM ENQUANTO
```

Estrutura REPETIR ... ATÉ

```
REPETIR
     Instrução ou Bloco de Instruções
     incluindo alteração da Condição
ATÉ Condição   (até ser verdadeira)
```

Estrutura REPETIR ... ATÉ



2.4.4. 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 e 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 uma abordagem modular dum algoritmo, dividindo em pequenos sub-programas simples um problema complexo;
- Sendo possível utilizar uma sub-rotina em diferentes locais do algoritmo pela evocação apenas do seu identificador, podemos evitar as repetições de código (Resultando em nítidas vantagens na elaboração do algoritmo e na posteriormente na sua manutenção);
- Permitem a aplicação duma mesma funcionalidade o sub-programa em contextos diferentes.
- Facilitam a leitura do algoritmo, porque estes tornam-se mais pequenos;
- Podendo o identificador indicar a funcionalidade associada a um sub-programa, este fornece-nos a abstracção do seu conteúdo;
- Possibilita a divisão das tarefas por várias pessoas, não necessitando cada uma delas, de ter conhecimento da globalidade do algoritmo.

2.4.5. Regras de estilo na escrita dum 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 e a sua legibilidade.

- Pôr em destaque as **palavras-chave** - utilização de LETRA MAIÚSCULA, sublinhar, **negrito**.
- Utilizar **tabulações** ou **avanços de linha** - deslocamento das linhas em relação á margem, formando reentrâncias, de forma a agrupar e destacar determinadas partes. (Ver os exemplos das estruturas de controlo).
- 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, é preferível sub-dividir a linha em várias.
- Inserção de **comentários explicativos** - um algoritmo ou um programa torna-se mais fácil de entender por outras pessoas ou pelo próprio programador, volvido algum tempo, se forem anexados comentários que descrevam as opções tomadas (Devem-se evitar os comentários óbvios).

2.4.6. Alguns problemas simples...

- Calcular a área de um trapézio rectângulo.
- Calcular a média final de curso de um aluno.
- 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.