

10^a

CLASSE



REPÚBLICA DE ANGOLA | MINISTÉRIO DA EDUCAÇÃO

TÉCNICAS E LINGUAGENS DE PROGRAMAÇÃO 10

TEXTOS DE APOIO AO ALUNO

reDítep
EDIÇÕES REDITEP

RETEP | REFORMA DO ENSINO TÉCNICO-PROFISSIONAL

Índice

CAPÍTULO 1 Conceitos sobre linguagens de programação 11

1.1 Conceitos básicos sobre <i>software</i>	13
1.2 Tipos de linguagens de programação	13
1.2.1 Linguagem de Baixo nível	13
1.2.2 Linguagens de alto nível.....	13
1.3 Conceitos sobre compilador e interpretador	15
1.4 Noção de programa fonte, programa objecto e programa tradutor	16

CAPÍTULO 2 Introdução à algoritmia e algoritmos fundamentais..... 17

2.1 Noções gerais de programação	19
2.1.1 Algoritmos.....	19
2.1.2 Representação de algoritmos	19
2.1.3 Concepção de um programa	21
2.2 Tipos de dados e declarações	25
2.2.1 Variáveis	25
2.2.2 Constantes	26
2.2.3 Tipos de dados	27
2.3 Operadores aritméticos	29
2.4 Operadores lógicos e relacionais	30
2.4.1 Expressões relacionais.....	31
2.5 Entrada e saída de dados	32
2.5.1 Funções pré-definidas	33
2.6 Instruções de controlo de fluxo	33
2.6.1 Instruções condicionais	33
2.6.2 Instruções cíclicas	42
2.7 Variáveis indexadas (vectores e matrizes).....	49
2.8 O Português estruturado e o C	51
2.9 Algoritmos básicos	55
PROPOSTAS DE TRABALHO	56

CAPÍTULO 3 Programação estruturada numa linguagem de

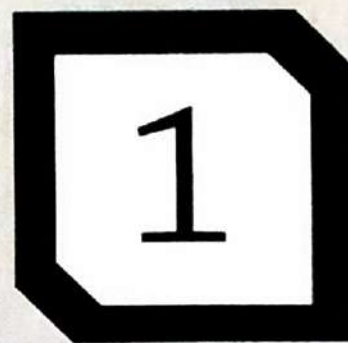
alto nível.....	61
3.1 A estrutura de um programa em C.....	63
3.1.1 Variáveis.....	64
3.1.2 Operações aritméticas.....	66
3.1.3 Operadores de comparação.....	67
3.1.4 Operadores lógicos.....	68
3.1.5 Precedências.....	68
3.2 Execução condicional.....	69
3.2.1 A instrução <i>if</i>	69
3.2.2 O operador <i>?:</i>	70
3.2.3 A instrução <i>switch</i>	70
3.3 Ciclos e iteração.....	71
3.3.1 A instrução <i>while</i>	71
3.3.2 A instrução <i>do... while</i>	72
3.3.3 A instrução <i>for</i>	73
3.3.4 <i>Break</i> e <i>continue</i>	74
3.4 Arrays.....	75
3.4.1 Arrays unidimensionais e multidimensionais.....	75
3.4.2 <i>Strings</i>	75
3.5 Funções.....	76
3.5.1 Definição de função.....	76
3.5.2 Funções <i>void</i>	77
3.5.3 Funções e arrays.....	78
3.5.4 Protótipos.....	79
3.6 Outros tipos de dados.....	79
3.6.1 Estruturas.....	79
3.6.2 Uso de <i>typedef</i>	80
3.6.3 Unões.....	81
3.6.4 Conversão entre tipos (<i>type casting</i>).....	82
3.6.5 Tipos enumerados.....	83
3.6.6 Variáveis estáticas.....	84
3.7 Apointadores.....	85
3.7.1 O que são apontadores?.....	85
3.7.2 Apointadores e funções.....	87
3.7.3 Apointadores e arrays.....	89

3.7.4 Arrays de apontadores.....	90
3.7.5 Arrays multidimensionais e apontadores.....	91
3.7.6 Inicialização de arrays de apontadores.....	92
3.7.7 Apointadores e estruturas.....	93
3.7.8 Erros comuns na utilização de apontadores.....	94
3.8 Alocação dinâmica de memória.....	95
3.8.1 <i>Malloc</i>	95
3.8.2 Listas ligadas.....	95
3.9 Entrada e saída.....	99
3.9.1 <i>Streams</i>	99
3.9.2 Operações básicas de entrada/saída.....	100
3.9.3 Entrada/saída formatadas.....	101
3.9.4 Ficheiros.....	103
3.9.5 As funções <i>printf</i> e <i>scanf</i>	105
PROPOSTAS DE TRABALHO.....	107

BIBLIOGRAFIA.....

Capítulo

CONCEITOS SOBRE LINGUAGENS DE PROGRAMAÇÃO



CONTEÚDO

- 1.1 Conceitos básicos sobre *software*.
- 1.2 Tipos de linguagens de programação.
- 1.3 Conceitos sobre compilador e interpretador.
- 1.4 Noção de programa fonte, programa objecto e programa tradutor.

OBJECTIVOS

- Dar uma noção da evolução dos sistemas informáticos ao longo dos tempos.
- Distinguir diferentes tipos de suportes de armazenamento de informação.
- Apresentar diferentes Sistemas Operativos.
- Dar uma noção das diferentes linguagens de programação existentes.

1.1 CONCEITOS BÁSICOS SOBRE SOFTWARE

O conceito de software refere-se à unidade lógica do computador e aos seus componentes imateriais. Assim, é considerado o conjunto de ordens e instruções que tornam possível ao computador realizar determinadas tarefas.

Sem software, o computador é como uma cabeça sem ideias. As informações e instruções codificadas em linguagem compreensível pelo computador constituem programas que, tal como os nossos pensamentos e ideias, orientam o nosso comportamento, controlam e regulam o funcionamento do computador.

1.2 TIPOS DE LINGUAGENS DE PROGRAMAÇÃO

As várias linguagens de programação podem-se classificar em três tipos distintos:

- Linguagens Máquina;
- Linguagens Assembly (simbólicas);
- Linguagens de Alto Nível.

1.2.1 LINGUAGEM DE BAIXO NÍVEL

1.2.1.1 LINGUAGEM MÁQUINA

Esta linguagem consiste num conjunto de números binários (sequências de 0 e 1) que são entendidos pela Unidade Central de Processamento e têm a ver com características intrínsecas do processador do computador.

1.2.1.2 LINGUAGEM ASSEMBLY

Consiste numa linguagem muito próxima da linguagem máquina, em que a cada **opcode** (código de operação) se associou uma mnemónica (palavra que nos ajuda a lembrar a acção realizada pelo comando). Esta linguagem está também ligada ao processador do computador.

1.2.2 LINGUAGENS DE ALTO NÍVEL

Linguagens que permitem a especificação de instruções para o processador numa forma abreviada, onde cada instrução representa várias instruções em linguagem máquina. Outras vantagens:

- Mais fáceis de aprender;
- Requerem menos tempo a escrever;
- Melhor documentação;

- Mais fáceis de alterar e manter;
- Não está ligado a uma só máquina.

1.2.2.1 FORTRAN

- Linguagem sequencial;
- Grandes bibliotecas de programas científicos e de engenharia;
- Difícil de definir a lógica do programa;
- Não apropriada para processar ficheiros com grande volume de dados;
- Orientada ao processamento numérico.

1.2.2.2 BASIC

- Muito parecida com o FORTRAN;
- Fácil definição da lógica do programa;
- I/O de dados de fácil acesso;
- Não estruturada.

1.2.2.3 COBOL

- Recursos que a tornam especialmente útil em aplicações comerciais;
- Usa termos muito frequentes em inglês.

1.2.2.4 ALGOL

- Linguagem modular e estruturada por blocos, sendo muito apropriada para programação por blocos.

1.2.2.5 PASCAL

- Primeira linguagem de programação a ser criada depois de os conceitos de programação estruturada serem largamente aceites.

1.2.2.6 LINGUAGEM C

- Produz um código que se aproxima da linguagem máquina em densidade e eficiência, mas que oferece também algumas características das linguagens de alto nível.

1.2.2.7 LINGUAGENS DE 4ª GERAÇÃO

- Geradores de aplicação que permitem às pessoas especificar os resultados desejados sem que os utilizadores tenham que definir a lógica de programação necessária (exemplo: LISP, LOGO, MODULA-2, PROLOG e PILLIT).

1.2.2.8 LINGUAGENS ORIENTADAS AO OBJECTO – OO

- Linguagens de programação onde o utilizador trabalha com classes de objectos da mesma família ou de famílias diferentes.
- Recorre à programação de scripts de objectos pré-definidos ou definidos pelo programador (exemplo: C++, TOOLBOK, Visual Basic, Java, etc.).

1.3 CONCEITOS SOBRE COMPILADOR E INTERPRETADOR

Um programa escrito em linguagem de alto nível é, para o computador, um simples texto. Este texto é escrito com um programa do tipo “Editor de Texto” e guardado em memória (central ou auxiliar). Posteriormente, esse mesmo programa em linguagem evolvida será lido como texto e decodificado nas correspondentes instruções em linguagem máquina. Uma só instrução de alto nível pode dar origem a um grande número de instruções em linguagem máquina.



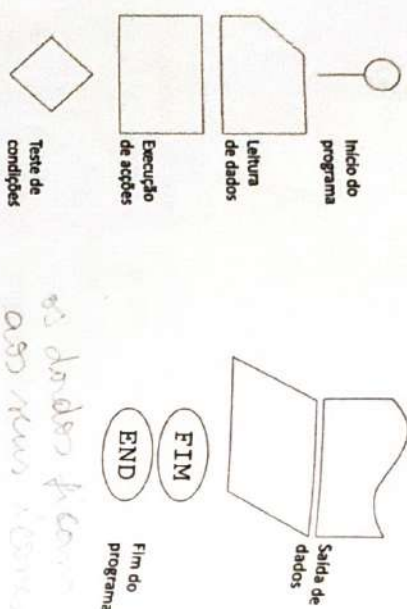
A figura anterior, mostra os dois percursos possíveis para a execução de um programa escrito em linguagem de alto nível: **compiladores** e **interpretadores**.

- **Interpretador** – Lê no programa inicial uma instrução de cada vez. Por cada instrução lida, verifica quais as acções a tomar em linguagem máquina e executa-as. Um interpretador não cria nenhum outro programa.
- **Compilador** – Lê um certo número de vezes (mas ao programador pode parecer só uma vez) o programa inicial em linguagem de alto nível (o texto) e constrói um segundo programa equivalente ao primeiro mas escrito em linguagem de máquina.

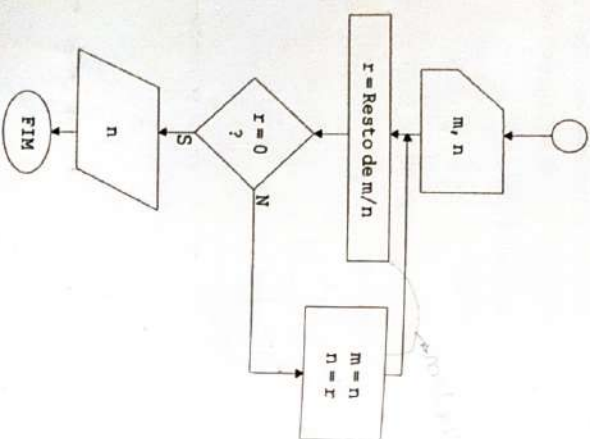
2.1.2.2 FLUXOGRAMAS

Um fluxograma ou diagrama de fluxo consiste na representação gráfica de um algoritmo, sendo constituído por um conjunto de figuras geométricas ligadas através de setas.

Blocos usados em fluxogramas:



Exemplo: O algoritmo para calcular o m.d.c. de dois números será o seguinte



2.1.2.3 LANGUAGE ALGORITMICA

Utiliza uma linguagem estruturada próxima da linguagem natural do programador, extraindo as melhores características das duas abordagens anteriores. Assim, da representação em linguagem corrente retirou-se a descrição textual em linguagem natural (Português), e adicionou-se a representação sintética e não ambígua dos fluxogramas. Este tipo de representação é designado por pseudocódigo.

```
// Algoritmo para determinar o m.d.c. entre m e n
mdc()
{
    // Declaração das variáveis
    inteiro m, n, r;
    ler(m);
    ler(n);
    r = 1;
    enquanto (r != 0) faça
    {
        r = m % n;
        se (r != 0)
        {
            m = n;
            n = r;
        }
    }
    escrever("Mensagem", n);
}

/* Já neste pequeno exemplo se nota a diferença do C para o Pascal, pois não existem as "etiquetas" de "início" e "fim". Estas "etiquetas" foram substituídas por ((início) e por ((fim). */
```

2.1.3 CONCEPÇÃO DE UM PROGRAMA

Obedece a uma sequência de quatro fases:

1ª Fase – Formulação do problema

Estudo mais aprofundado do problema, de forma a encontrar um modelo que o represente correctamente.

2ª Fase – Resolução do problema

Obter um raciocínio estruturado, isto é, um algoritmo que conduza à solução do problema.

2.1 NOÇÕES GERAIS DE PROGRAMAÇÃO

2.1.1 ALGORITMOS

Um dos aspectos cruciais no desenvolvimento de um programa é a idealização de um algoritmo, que deve ser capaz de conduzir à solução do problema em estudo, de uma forma coerente e eficaz.

Um algoritmo não é mais do que uma sequência de operações necessárias para realizar uma dada tarefa.

Características:

- Deve ser determinístico;
- Deve atingir a solução em tempo finito;
- Deve ser genérico, de forma a considerar todas as situações particulares do problema, não contendo instruções ambíguas;
- Deve ter capacidade de comunicar com o exterior.

2.1.2 REPRESENTAÇÃO DE ALGORITMOS

2.1.2.1 LINGUAGEM CORRENTE

Esta linguagem é considerada como a forma mais simples para representar um dado algoritmo, indicando os passos a seguir na linguagem do dia-a-dia:

- Tipo receita narrativa em linguagem natural;
- Acessível à maioria das pessoas.

Exemplo: Programa para cálculo do máximo divisor comum de dois números.

1. Função principal do programa.
2. Ler o valor de m e de n .
3. Dividir m por n e atribuir o resto a r .
4. Se r for igual a 0, então ir para a linha 7.
5. Atribuir a m o valor de n , e a n o valor de r .
6. Voltar para a linha 3.
7. Apresentar n (resultado).
8. Fim do cálculo.

*escreve-se muito
para transmi-
tir pouca
coisa*

1.4 NOÇÃO DE PROGRAMA FONTE, PROGRAMA OBJECTO E PROGRAMA TRADUTOR

Podemos separar, e organizar, os programas que geramos da seguinte forma:

Programa Fonte - É o texto que nós escrevemos, utilizando um editor de texto e seguindo as regras sintácticas utilizadas na linguagem de programação, com que vamos programar.

Programa Objecto - É o programa que se obtém após ter termos feito a compilação do programa fonte, utilizando um compilador, ou seja, após ter sido traduzido o programa fonte.

Programas Tradutores - São programas que traduzem o programa fonte em linguagem de máquina, utilizando para tal o programa objecto criado a partir do programa fonte. Com a utilização dos programas tradutores obtemos um programa que pode ser executado directamente pela máquina (computador).

Capítulo

2

INTRODUÇÃO À ALGORTMIA E ALGORITMOS FUNDAMENTAIS

CONTEÚDO

- 2.1 Noções gerais de programação.
- 2.2 Tipos de dados e declarações.
- 2.3 Operadores aritméticos.
- 2.4 Operadores lógicos e relacionais.
- 2.5 Entrada e saída de dados.
- 2.6 Instruções de controlo de fluxo.
- 2.7 Variáveis indexadas (vectores e matrizes).
- 2.8 O Português estruturado e o C.
- 2.9 Algoritmos básicos.
- Propostas de trabalho.

OBJECTIVOS

- Introduzir conceitos básicos sobre técnicas de programação.
- Dar a conhecer os elementos base utilizados na programação estruturada.
- Caracterizar os elementos que constituem o Português Estruturado utilizado.
- Classificar as instruções de entrada de dados.
- Classificar as instruções de saída de dados.
- Classificar as instruções de processamento.
- Definir instruções de decisão.
- Definir instruções cíclicas.
- Abordar técnicas de trabalho com variáveis indexadas.
- Construir algoritmos básicos de programação.
- Conhecer técnicas de conversão de algoritmos em Português Estruturado para Linguagem C.

As variáveis devem ter nomes elucidativos das suas funções no programa. Assim, definem-se algumas regras relativas à escolha do nome das variáveis:

1. O primeiro carácter deve ser uma letra (maiúscula ou minúscula);
2. Os nomes são constituídos pelas letras do alfabeto, números, carácter disponível (por exemplo: '-'); main, scanf, sqrt, etc;
3. Não são permitidos espaços no meio dos nomes das variáveis;
4. Não é permitido usar como identificadores de variáveis as palavras reservadas da linguagem de programação em uso, como por exemplo: main, scanf, sqrt, etc;
5. Ter em atenção que o nome de uma variável escrita com letras maiúsculas é diferente de uma variável escrita com letras minúsculas. Exemplo: VALOR ≠ valor.

De seguida apresentam-se alguns nomes de variáveis válidos:

1. AL2
2. Nome
3. X
4. nome_do_aluno

Na Linguagem Algoritmica que vamos utilizar (Português Estruturado), a atribuição de um valor a uma variável é feita através do operador ←, cuja sintaxe é a seguinte:

 <variável> ← <expressão>;


Exemplos:

numero ← 0.5;
soma ← 5 + 2;
nome ← 'João';

2.2.2 CONSTANTES

Consideram-se como valores constantes atribuídos a um "símbolo" que não se altera ao longo da execução do programa.

A sintaxe da declaração de constantes é a seguinte:

 Constante <constante> = <valor>;

Exemplos:

dias = 30;
Constante meses = 12;
Constante PI = 3.1415;
Constante Taxa_iva = 0.20;
Constante

Ou de outro modo:

Constante dias = 30; meses = 12; PI = 3.1415; Taxa_iva = 0.20;

Assim, o uso de constantes num programa justifica-se quando:

- Existem valores inalteráveis usados com frequência;
- Existem valores possíveis de alteração em futuras versões do programa;
- Há conveniência de tornar um programa mais legível.

2.2.3 TIPOS DE DADOS

Cada variável é classificada segundo um determinado tipo de dados, de acordo com o(s) valor(es) que vai armazenar.

Na elaboração dos algoritmos em Português Estruturado, consideram-se cinco tipos de dados primários: Inteiro, Real, Carácter, String, Booleano.

A razão desta separação em tipos de dados, reside na fixação de aspectos importantes como:

- O conjunto de valores que uma constante pode tomar;
- O conjunto de valores obtidos através de uma expressão;
- O conjunto de valores por uma função ou operador;
- O conjunto de valores assumidos por uma variável é essencial para a compreensão de qualquer programa ou algoritmo;
- O espaço em memória necessário para representar uma variável depende do tipo de dados associado a essa variável.

2.2.3.1 DECLARAÇÃO DE VARIÁVEIS

Em Português Estruturado a declaração de variáveis é feita pelo tipo de dados da variável seguido dos nomes das variáveis.

Linguagem Algorítmica:

// Programa de cálculo de Raízes:

Nome_da_função()

{

Inteiro a, b, c, d;

Real x1, x2;

ler(a);

ler(b);

ler(c);

d = $\text{SQRT}(b) - 4 \cdot a \cdot c$;

se (d > 0)

{

x1 = $(-b - \text{SQRT}(d)) / (2 \cdot a)$;x2 = $(-b + \text{SQRT}(d)) / (2 \cdot a)$;

escrever("x1=", x1, "x2=", x2);

}

senão

se (d = 0)

{

x1 = $-b / (2 \cdot a)$;

x2 = x1;

escrever("x1=", x1, "x2=", x2);

}

senão

escrever("Impossível!");

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

4ª Fase (Teste)

 $x2 - 5x + 6 = 0$

A	B	C	D	D > 0	D = 0	x1	x2	total
1	-	-	-	-	-	-	-	-
1	5	-	-	-	-	-	-	-
1	5	6	-	-	-	-	-	-
1	5	6	1	-	-	-	-	-
1	5	6	1	V	-	-	-	-
1	5	6	1	-	-	-3	-	-
1	5	6	1	-	-	-3	-2	-
1	5	6	1	-	-	-3	-2	-
1	5	6	1	-	-	-3	-2	-

 $x2 - 2x + 1 = 0$

A	B	C	D	D > 0	D = 0	x1	x2	total
1	-	-	-	-	-	-	-	-
1	-2	-	-	-	-	-	-	-
1	-2	1	-	-	-	-	-	-
1	-2	1	0	-	-	-	-	-
1	-2	1	0	F	-	-	-	-
1	-2	1	0	-	V	-	-	-
1	-2	1	0	-	-	1	-	-
1	-2	1	0	-	-	1	1	-
1	-2	1	0	-	-	1	1	-

2.2 TIPOS DE DADOS E DECLARAÇÕES

2.2.1 VARIÁVEIS

Uma variável pode ser entendida como um espaço reservado na memória do computador no qual podemos guardar determinado tipo de informação.

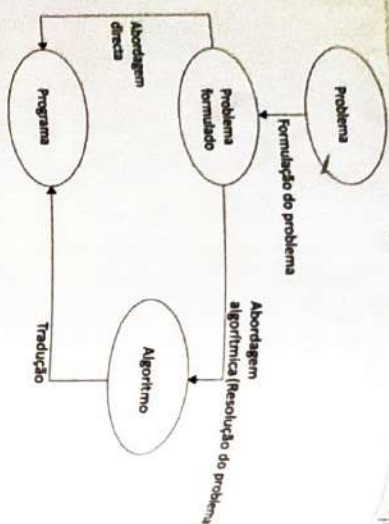
O valor de uma variável pode ser lido (actualizado) sempre que necessário ou ser modificado, quando lhe atribuirmos um novo valor. Uma nova atribuição implica a perda do valor anterior.

3ª Fase - Tradução

Tradução do algoritmo para a linguagem de programação escolhida

4ª Fase - Teste

Testes com o objetivo de verificar se o programa cumpre os objetivos propostos



Exemplo de aplicação:

Elaborar um programa que permita o cálculo das raízes de equações do 2º grau ($ax^2 + bx + c = 0$). O programa deve ler os coeficientes a, b e c .

1ª Fase (Formulação do problema)

Equações/condições:

$$d = b^2 - 4 \cdot a \cdot c$$

se $d > 0$:

$$x1 = \frac{-b - \sqrt{d}}{2 \cdot a}$$

$$x2 = \frac{-b + \sqrt{d}}{2 \cdot a}$$

se $d = 0$:

$$x1 = -b / (2 \cdot a)$$

$$x2 = -b / (2 \cdot a)$$

se $d < 0$:

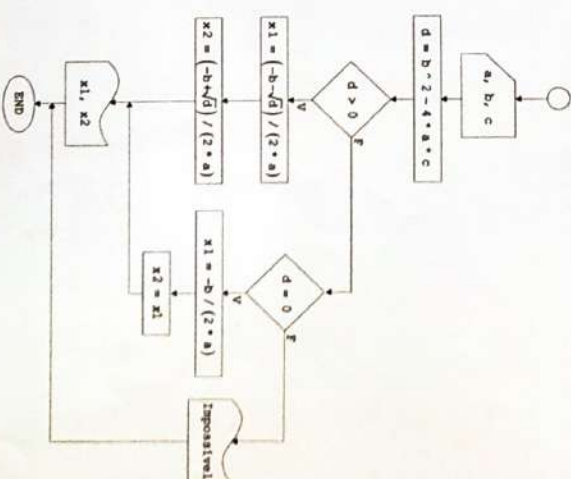
Enviar uma mensagem de erro.

2ª Fase (Resolução do problema)

Linguagem corrente:

1. Início do cálculo.
2. Ler o valor de a, b e c .
3. Atribuir a d o valor do quadrado de b menos o produto de 4 por a e por c .
4. Se $d < 0$, então ir para a linha 14.
5. Se $d = 0$, então ir para a linha 9.
6. Atribuir a $x1$ o valor de $((-b - \sqrt{d}) / (2 \cdot a))$.
7. Atribuir a $x2$ o valor de $((-b + \sqrt{d}) / (2 \cdot a))$.
8. Ir para a linha 11.
9. Atribuir a $x1$ o valor de $(-b / (2 \cdot a))$.
10. Atribuir a $x2$ o valor de $x1$.
11. Escrever o valor de $x1$.
12. Escrever o valor de $x2$.
13. Ir para a linha 15.
14. Escrever a mensagem de erro.
15. Fim do cálculo.

Fluxograma:



2.5 ENTRADA E SAÍDA DE DADOS

A leitura de dados e a apresentação de resultados são dois aspectos de grande importância em programação.

De facto, para que um programa seja útil, isto é, para que possa resolver o problema real que lhe deu origem, é necessário fornecer um conjunto de dados (entrada) para que o programa possa apresentar os resultados respectivos (saída).



Na linguagem Português Estruturado existem dois procedimentos pré-definidos que permitem esta interacção com o exterior:

- ler (< lista de variáveis de entrada >);
- escrever (< lista de variáveis de saída >);

Exemplo:

// Programa para a leitura e escrita de um número inteiro

```
Principal()
{
    inteiro x,y;
    ler(x);
    y ← x * 5 + 7;
    escrever(y);
}
```

A função **ler()** guarda o valor que for introduzido, através do teclado do computador, na variável que tem como argumento (**x**).

A função **escrever()** escreve, no ecrã do computador, o valor que contém a variável que está no seu argumento.

2.5.1 FUNÇÕES PRÉ-DEFINIDAS

função	descrição
ABS(x)	Valor absoluto de x
SQR(x)	Potência quadrática de x
SQRT(x)	Raiz quadrada de x
TRUNC(x)	Trunca o valor de x
ROUND(x)	Arredonda o valor de x
LOG(x)	Logaritmo de x na base e
EXP(x)	Exponencial de x
SIN(x)	Seno de um ângulo (em x)
COS(x)	Co-seno de um ângulo
TAN(x)	Tangente de um ângulo

ler, escrever, truncar, arredondar

Exemplos:

```
p ← - 5;
y ← ABS ( p );
h ← SQR ( SQR ( c1 ) + SQR ( c2 ) );
v ← 4.5;
v1 ← TRUNC ( v );
va ← ROUND ( v );
```

2.6 INSTRUÇÕES DE CONTROLO DE FLUXO

2.6.1 INSTRUÇÕES CONDICIONAIS

Estas instruções permitem a selecção de uma opção a partir de um conjunto de opções alternativas.

Quando é seleccionada a opção pretendida, é executada uma instrução ou um bloco de instruções que correspondem a essa opção. A escolha baseia-se no resultado da avaliação de uma expressão lógica.

2.6.1.1 SE... ENTÃO...

Esta instrução consiste na avaliação de uma expressão lógica. Se o resultado da expressão avaliada for verdadeiro, então é executada uma instrução ou bloco de instruções.

A sintaxe desta instrução é a seguinte:

Estes operadores são os que aparecem no seguinte quadro:

OPERADOR	OPERAÇÃO	SINTAXE
+	Adição	$a + b$
-	Subtração	$a - b$
*	Multiplicação	$a * b$
/	Divisão	a / b
%	Resto da divisão inteira	$a \% b$
-	Sinal	$-a$

Obedecem às seguintes prioridades:

OPERADOR	ASSOCIATIVIDADE
Sinal -	Da direita para a esquerda
*, /, %	Da esquerda para a direita
+ -	Da esquerda para a direita

2.4 OPERADORES LÓGICOS E RELACIONAIS

Os operadores lógicos definidos na linguagem Português Estruturado são os seguintes:

A	B	A AND B	A OR B	NOT A
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Tal como os operadores aritméticos, os operadores lógicos também têm uma determinada prioridade associada a cada um deles, como se pode ver no seguinte quadro:

OPERADOR	ASSOCIATIVIDADE
NOT	Da direita para a esquerda
AND	Da esquerda para a direita
OR	Da esquerda para a direita

Os operadores relacionais são utilizados em expressões que condicionam acções, normalmente designadas por controlo de fluxo.

Essas acções de controlo de fluxo determinam se um dado bloco de instruções deve ser ou não executado, avaliando para tal uma expressão lógica.

Estes operadores são os que se apresentam no quadro seguinte:

OPERADOR	OPERAÇÃO	SINTAXE
==	Igual a	$a == b$
<	Menor que	$a < b$
<=	Menor ou igual a	$a <= b$
>	Maior que	$a > b$
>=	Maior ou igual a	$a >= b$
!=	Diferente de	$a != b$

2.4.1 EXPRESSÕES RELACIONAIS

Leis comutativas:

- $a \text{ OR } b = b \text{ OR } a$
- $a \text{ AND } b = b \text{ AND } a$

Leis associativas:

- $(a \text{ OR } b) \text{ OR } c = a \text{ OR } (b \text{ OR } c)$
- $(a \text{ AND } b) \text{ AND } c = a \text{ AND } (b \text{ AND } c)$

Leis distributivas:

- $(a \text{ AND } b) \text{ OR } c = (a \text{ OR } c) \text{ AND } (b \text{ OR } c)$
- $(a \text{ OR } b) \text{ AND } c = (a \text{ AND } c) \text{ OR } (b \text{ AND } c)$

Leis de DeMorgan:


- $\text{NOT } (a \text{ OR } b) = \text{NOT } a \text{ AND NOT } b$
- $\text{NOT } (a \text{ AND } b) = \text{NOT } a \text{ OR NOT } b$

Outras simplificações:

- $a <= b$ é equivalente a $(a < b) \text{ OR } (a = b)$
- $a >= b$ é equivalente a $\text{NOT } (a < b)$
- $a > b$ é equivalente a $\text{NOT } (a < b) \text{ AND NOT } (a = b)$

Handwritten notes:
 $a >= b$ é equivalente a $\text{not}(a < b) \text{ and not}(a = b)$
 $a > b$ é equivalente a $\text{not}(a < b)$

A sintaxe da declaração de variáveis é a seguinte:

 <tipo da variável> <nome da variável>;

2.2.3.2 TIPO INTEIRO

Este tipo de dados refere-se a números inteiros, positivos e negativos, e centes a determinada gama dependendo do computador em questão.

Em Português Estruturado, a declaração de variáveis do tipo inteiro seguinte forma:

- inteiro x;
- inteiro numero;
- inteiro vari;

Ou de outra forma:

- inteiro x, numero, vari;

Após estas declarações, as variáveis x, numero e vari, estão em condições de serem inicializadas e utilizadas nos programas.

2.2.3.3 TIPO REAL

O tipo real define o conjunto dos números reais, respectivamente dependente do computador onde se está a executar o programa.

Existem duas formas de representar números reais:

Notação decimal:

1634.17
- 173.176
0.0243

Notação científica:

1.63417 E + 3
- 1.73176 E - 2
2.43 E - 2

Podem ser declarados variáveis do tipo real da seguinte forma:

real x, y;
103.46 E + 8
123.46 E + 10

2.2.3.4 TIPO CARACTER

Os elementos deste tipo de dados são todos os caracteres individuais que podem ser representados por um computador.

Actualmente, os fabricantes de computadores recorrem a um conjunto de caracteres definidos no código ASCII (*American Standard Code for Information Interchange*) - A...Z (maiúsculas e minúsculas), 0...9 e o caracter correspondente ao espaço vazio.

A declaração de variáveis é do seguinte tipo:

- caracter letra, simbolo;

A atribuição de valores pode ser feita do seguinte modo:

- letra ← "A";
- simbolo ← "5";

2.2.3.5 TIPO STRING

Este tipo consiste num valor que não é mais do que um conjunto de caracteres.

A declaração de variáveis é feita do seguinte modo:

- caracter nome_var[amanho];

A atribuição de valores é feita como se apresenta a seguir:

- palavra ← "Programação e Computadores I";

2.2.3.6 TIPO BOOLEANO

O tipo Booleano representa o conjunto de valores verdadeiro e falso. A declaração de uma variável deste tipo é feita do seguinte modo:

- booleano bol;

Esta variável pode tomar valores lógicos V (verdadeiro) ou F (falso), e são da seguinte forma:

- bol ← V;

2.3 OPERADORES ARITMÉTICOS

Os operadores aritméticos definidos na linguagem Português Estruturado são os que correspondem às operações aritméticas simples, mais um operador que fornece como resultado o resto de uma divisão inteira.

O passo seguinte seria a determinação da menor distância, isto é, o menor de d_1 , d_2 e d_3 . Esta determinação será feita através da seguinte selecção:

```

Se ( $d_1 < d_2$ )
   $m \leftarrow d_1$ ;
senão
   $m \leftarrow d_2$ ;
Se ( $d_3 < m$ )
   $m \leftarrow d_3$ ;

```

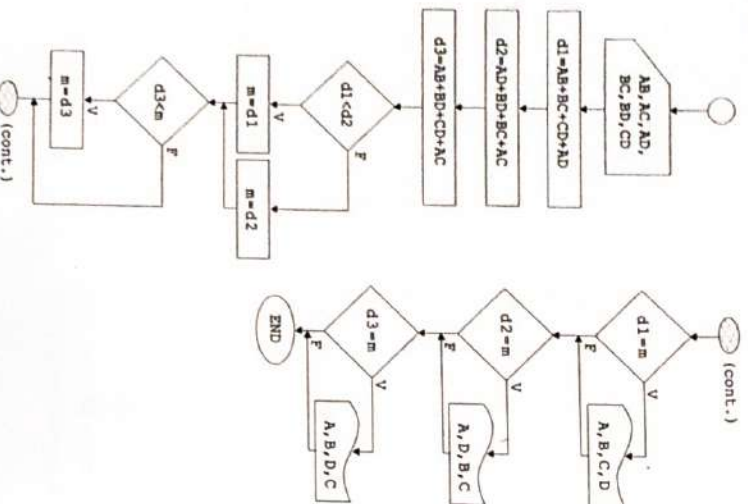
Após colocado em m o valor do menor caminho, só falta identificar a qual dos percursos corresponde o valor. Assim:

Se $d_1 = m$ então Caminho 1 é o mais curto

Se $d_2 = m$ então Caminho 2 é o mais curto

Se $d_3 = m$ então Caminho 3 é o mais curto

Fluxograma:



Linguagem algorítmica:

```

// Programa que calcula a distância de caminhos principal()
{
  inteiro AB, AC, AD, BC, BD, CD, m, d1, d2, d3;
  escrever("distância entre A e B:");
  ler(AB);
  escrever("distância entre A e C:");
  ler(AC);
  escrever("distância entre A e D:");
  ler(AD);
  escrever("distância entre B e C:");
  ler(BC);
  escrever("distância entre B e D:");
  ler(BD);
  escrever("distância entre C e D:");
  ler(CD);

  d1 ← AB + BC + CD + AD;
  d2 ← AD + BD + BC + AC;
  d3 ← AB + BD + CD + AC;

  se (d1 < d2) então
    m ← d1;
  senão
    m ← d2;
  se (d3 < m) então
    m ← d3;
  se (d1 == m) então escrever("Menor caminho: A-B-C-D");
  se (d2 == m) então escrever("Menor caminho: A-D-B-C");
  se (d3 == m) então escrever("Menor caminho: A-B-D-C");
}

```

2.6.1.3 CASO... SEJA...

Considera-se como uma instrução condicional que permite a selecção a partir de mais do que duas condições. A sua sintaxe é a seguinte:

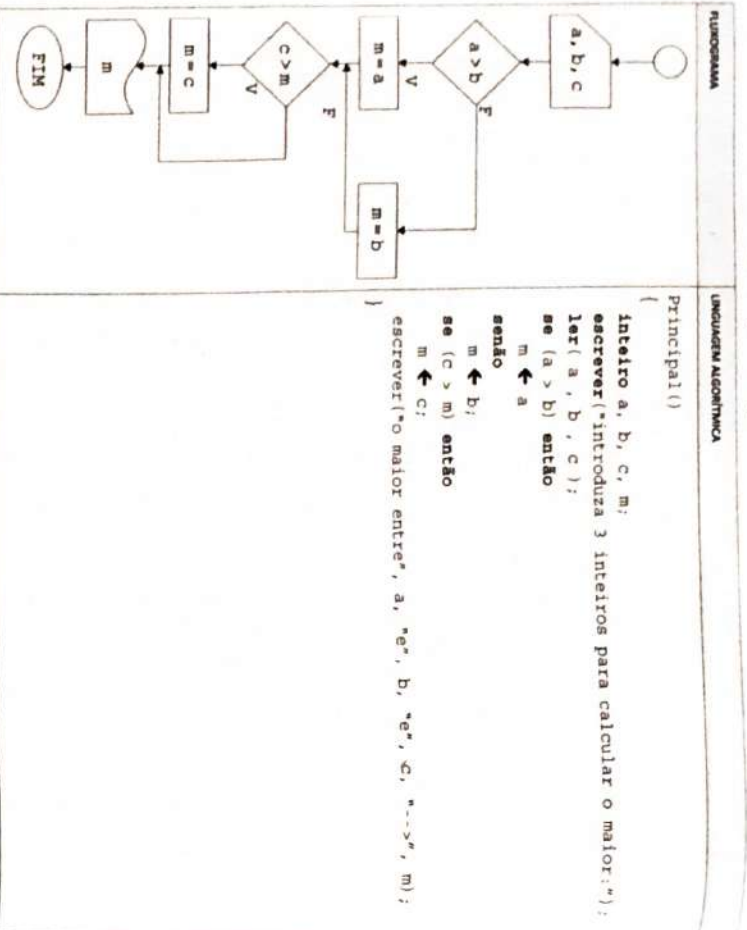
caso (variável)

```

{
  Seja <constante_1>;
    <instruções_1>;
  Seja <constante_2>;
    <instruções_2>;
  ...
  Seja <constante_n>;
    <instruções_n>;
}

```


Construção do Algoritmo:



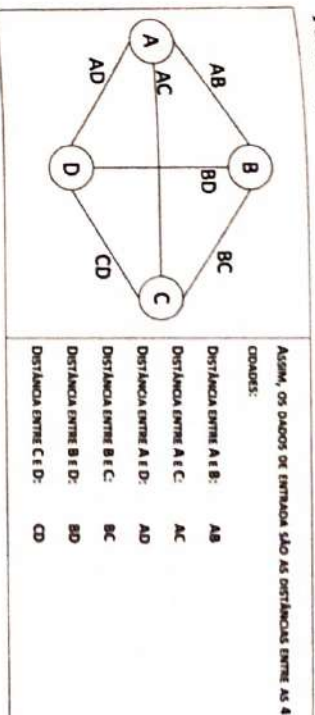
Teste:

A	B	C	M	A > B	C > M	cmA
2	7	5	-	-	-	-
2	7	5	-	f	-	-
2	7	5	7	-	-	-
2	7	5	7	-	f	-
2	7	5	7	-	-	O maior entre 2 e 7 e 5 --> 7

Exemplo de aplicação II

Considere o caso do caixeiro-viajante e escreva um programa que o auxilie na tomada de decisão do caminho mais curto que o leva da cidade A até as cidades B, C e D, retornando à cidade A, após a visita a todas elas, sem repetir os caminhos entre as cidades.

Formulação do problema:



Por outro lado, o dado de saída é o caminho mais curto para percorrer as 4 cidades, começando em A e acabando em A, sem repetir caminhos. Para tal vamos analisar os caminhos possíveis:

- 1º caminho: A → B → C → D → A
 2º caminho: A → D → C → B → A
 3º caminho: A → D → B → C → A
 4º caminho: A → C → B → D → A
 5º caminho: A → B → D → C → A
 6º caminho: A → C → D → B → A

Após este estudo podemos retirar algumas conclusões, dos 6 caminhos descritos, 3 deles estão repetidos:

- 2º que se repete no 1º;
- 4º que se repete no 3º;
- 6º que se repete no 5º.

Assim, as distâncias dos 3 circuitos disjuntos são:

- d1 ← AB + BC + CD + AD
 d2 ← AD + BD + BC + AC
 d3 ← AB + BD + CD + AC



se (< expressão lógica >) então < instrução >;

Vejam os seguinte exemplo:

```
//Programa que compara dois números
Principal()
{
    inteiro valor = 10;
    inteiro x;
    ler(x);
    se (x > valor) então
        escrever("O número", x, "é maior que", valor);
}
```

Note que a expressão a avaliar, no programa acima, é $x > \text{valor}$, em que x é uma variável inteira lida pelo programa, e **valor** é uma constante inteira.

Se o conteúdo da variável x for superior ao valor da constante (condição verdadeira) é executada a instrução de escrita no ecrã, caso contrário o programa termina a sua execução sem apresentar qualquer resultado.

2.6.1.2 SE... ENTÃO... SENÃO...

Esta instrução é idêntica à anterior mas tem a possibilidade de executar instruções alternativas no caso da instrução lógica ser falsa.

Neste caso existe sempre uma instrução ou um bloco de instruções que é(são) executada(s) sob o controlo da instrução condicional.

Vejam os seguinte exemplo:

```
//Comparação de dois números diferentes
Principal()
{
    inteiro x, y;
    ler(x);
    ler(y);
    se (x > y) então
        escrever("O número maior é", x)
    senão
        escrever("O número maior é", y);
}
```

A sintaxe desta instrução é:

```
se (< expressão lógica >) então
    < instrução1 >
senão
    < instrução2 >;
```

Estas instruções condicionais podem também ser encadeadas, isto é, podem-se incluir instruções condicionais dentro de outras, como se pode ver no seguinte exemplo:

```
//Programa que calcula médias
Principal()
{
    real media;
    ler(media);
    se (media >= 10) então
    {
        escrever("A média é positiva");
        se (media < 14) então
            escrever("Suficiente");
        se ((media >= 14) AND (media < 18)) então
            escrever("Bom");
        se (media >= 18) então
            escrever("Muito Bom");
    }
    senão
        escrever("A média é negativa");
}
```

Exemplo de aplicação I

Elabore um programa que lê três valores inteiros, a partir do standard input, e escreva no output o maior deles.

Formulação do problema:

Valores lidos: a, b, c

Verificar qual o maior valor dos dois primeiros: se $a > b$ então guardar o valor de a, senão guardar-se o valor de b.

Verificar se o terceiro valor é maior do que o seleccionado anteriormente: se c é maior que o valor que se obteve na comparação anterior, então c é o maior dos três valores, senão o maior é o obtido anteriormente.

Handwritten notes:
 se (a > b) então
 guardar(a) como maior;
 senão
 guardar(b) como maior;
 se (c > maior) então
 guardar(c) como maior;
 escrever(maior);

Um bom exercício é realizar todos os rastreiros existentes na aplicação.

2.6.2 INSTRUÇÕES CICLICAS

Instruções que permitem a execução repetitiva de uma instrução simples ou de um conjunto de instruções, sob o controle de uma expressão lógica.

2.6.2.1 ENQUANTO... FAÇA...

Nesta instrução, enquanto for verdadeira a expressão lógica colocada a seguir a **enquanto** é executada a instrução, ou o bloco de instruções, associado a **faça**.

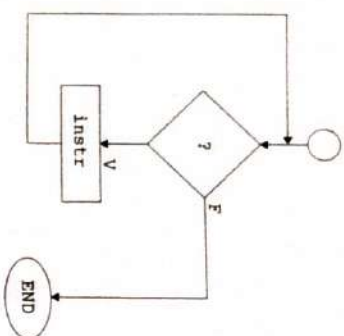
Sempre que o programa termina a execução do bloco de instruções (a seguir a **faça**), procede à avaliação da expressão lógica de controle.

Quando esta expressão lógica assumir um valor **FALSO**, a instrução cíclica termina e o programa passa à instrução seguinte.

A sintaxe desta instrução é a seguinte:

enquanto (<expressão Lógica>) **faça** <instrução> ;

O fluxograma que caracteriza esta instrução é o seguinte:



Exemplo:

```

//Programa que calcula as médias
principal()
{
  constante=num_alunos = 25;
  inteiro n1, n2, n3, n4, aluno;
  real media;
  aluno ← 1;
  enquanto (aluno <= num_alunos) faça
  {
    ler(n1, n2, n3, n4);
    media ← (n1 + n2 + n3 + n4) / 4;
    escrever("A média do aluno", aluno, " é", media);
    aluno ← aluno + 1;
  }
  escrever("Calculada a média de", num_alunos, " alunos");
}
  
```

num_alunos = 25

Exemplo de aplicação

Escreva um programa que leia uma sequência de valores inteiros (positivos e negativos), terminada com o valor 0 (zero), e determine qual foi o maior elemento dessa sequência e a sua ordem de entrada.

Formulação do problema

Como variáveis, podemos utilizar as seguintes:

- **X** – variável que vai armazenar o valor lido;
- **Maior** – variável que guarda o maior valor;
- **Pos** – variável que guarda a posição do maior elemento;
- **I** – variável que vai contar quantos elementos foram introduzidos.

A solução para o problema será ler o primeiro valor e inicializar as variáveis **maior**, **pos**, e **i** antes da instrução cíclica.

Dentro desta instrução, deve-se verificar se o valor lido é maior que o conteúdo na variável **maior** e actualizar as variáveis **pos** e **maior**.

Também dentro do ciclo, sempre que um novo valor é lido, deve-se actualizar o contador (variável **i**).

No fim do programa, depois da instrução cíclica, devemos escrever os resultados:

- **maior**;
- **pos**.