Anotações de Algoritmia (versão 4)

Problema: pretendo executar uma tarefa no meu computador (por exemplo, fazer um gráfico ou registar as características e propriedades de determinados materiais ou ir à conta do Facebook) – em grupos de 2, os estudantes pensam e registam exemplos de tarefas que regularmente executam no computador. De um modo geral, que componentes da máquina são regularmente envolvidos na execução dessas tarefas? Como interagem esses componentes, uns com os outros, durante a execução das mesmas?

1. Hardware
    a) Unidade Central de Processamento (UCP)
    b) Periféricos
2. Software
    a) Software do sistema
    b) Software de aplicação (ou aplicações)

1. Hardware

Trata-se do suporte físico do computador (ou da unidade de sistema) ou o conjunto de equipamentos e componentes físicos do computador. De entre estes componentes destacam-se a **UCP** (também designada por *motherboard*) e os periféricos.

a) Unidade Central de Processamento (UCP)

A *motherboard* é a placa que existe no interior da unidade de sistema e "contém um conjunto de circuitos eletrónicos responsáveis pelo processamento de dados e pelas trocas de informação, no interior do computador e entre este e o exterior" [2]). Os principais componentes da *motherboard* são:

O processador é:

- O centro de comando cujas funções são controladas por um programa (i.e., software do sistema);
- Responsável pela gestão do computador, por exemplo, por sincronizar o funcionamento de outros componentes físicos (como memórias e barramentos, por exemplo) do computador;
- Caracterizado por uma velocidade de processamento, determinada por um relógio interno; a velocidade de processamento é a medida da rapidez (em milhões de ciclos por segundo) com que o processador recebe e trata os dados e transmite informações; por exemplo, 533MHz equivalem a 533 milhões de ciclos/segundo; 2,26 GHz equivalem a 2260 milhões de ciclos/segundo.

A memória:

- Armazena um conjunto restrito de informação que o processador necessita de utilizar e aceder a velocidades bastante elevadas;
- *Random Access Memory* (RAM) ou memória central ou principal é de acesso direto ou aleatório; permite ao processador gravar, ler e apagar informação; o processador pode aceder a qualquer informação nela armazenada de imediato; é uma memória volátil ou temporária (por exemplo, se não tivermos gravado uma sessão de trabalho e falhar a energia elétrica perdemos essa sessão de trabalho…);
- *Read Only Memory* (ROM) é uma memória apenas de leitura, onde não é possível gravar, alterar ou apagar conteúdo; contém um conjunto de programas responsáveis pelo arranque do computador.

Os barramentos:

- São as "autoestradas" por onde circulam os dados, por exemplo, entre os diferentes componentes da *motherboard*;
- Quanto maior for o número de bits (grosso modo, 1 bit equivale a uma "faixa da autoestrada de dados") neles existentes, mais rápida será a circulação de dados nos barramentos. Então, barramentos de 64-bits serão mais rápidos que os de 32-bits, por exemplo.

b) Periféricos

A entrada de dados para a *motherboard* faz-se através dos periféricos de entrada (como, por exemplo, teclado, rato, leitor ótico, microfone, webcam, CR-ROM, DVD); e a saída de informação da UCP faz-se através de periféricos de saída (como, por exemplo, monitor, impressora, *plotter*).

Noções de …

Dados – podem ser encarados como "informação em bruto" ou sem qualquer significado (por exemplo, 56010 é um dado que depois de ser "processado" pode transformar-se num nº de um aluno, num rendimento anual de uma pessoa, etc).

Processamento – é o conjunto de operações lógicas e/ou aritméticas que são aplicadas, de forma automática, sobre os dados, com auxílio de equipamentos informáticos. O processamento confere um significado aos dados.

Informação – é o resultado do processamento de dados, ou são dados aos quais foram atribuídos significados, ou ainda, são dados contextualizados. Por exemplo, na aplicação que regista novas inscrições de alunos, nos SAUM, o valor 56011 pode tratar-se de um novo nº de aluno que resultou da operação de soma de 56010 com 1.

2. Software

Também denominado de suporte lógico do computador, é o conjunto de programas responsáveis pelo seu funcionamento e pela execução de tarefas computacionais (como, por exemplo, as de tratamento de texto e dados):

- Possibilita o funcionamento do próprio computador e a sua gestão ao mais baixo nível (como, por exemplo, os programas de arranque da ROM e os de gestão do processador, também designados por software do sistema);
- Permite construir gráficos, tratar texto, fazer a contabilidade da empresa, jogar (isto é, software de aplicação ou aplicações).

a) Software do sistema

É um conjunto de programas responsáveis pelo funcionamento do computador e pela gestão de todo o hardware. De entre o software do sistema destacam-se os sistemas operativos e os programas que (como o Windows Explorer) gerem a informação guardada nas unidades de armazenamento permanentes (como, por exemplo, disco rígido, *pen*, CD e DVD) do computador.

O sistema operativo (como por exemplo, Windows, Apple Macintosh, Linux, Unix, Android):

- Controla e gere o hardware da máquina;
- Funciona como elo de ligação (disponibilizando mecanismos de comunicação) entre o utilizador e o computador;
- Gere as trocas de dados e informação entre o processador, restantes componentes da *motherboard* e periféricos;
- Otimiza a instalação e configurações de periféricos;
- Gere a memória do computador (por exemplo, quando trabalhamos com uma aplicação, apenas parte desta é colocada na memória RAM; ou seja, as instruções da aplicação a que o processador acede com mais frequência vão para memória central e as restantes ficam no disco rígido; cabe ao sistema operativo definir o que passa para a RAM de modo a melhorar o desempenho do computador);
- Define regras de funcionamento das aplicações (como Internet Explorer, MS Word, MS Excel, outras);
- Envia mensagens informativas e de erro (como “cópia terminada”, “ficheiro a ser impresso”, “ficheiro em modo de leitura”; “disco sem espaço disponível”);
- Disponibiliza programas (como o Windows Explorer) que facilitam a gestão da unidade de sistema e da informação (como copiar ficheiros, listar o conteúdo do disco ou da *pen*, procurar/organizar ficheiros em pastas ou diretorias).

De entre as funcionalidades de um programa como o Windows Explorer destacam-se:

- Organização de informação por drives (como C:, D:, E:, Z:, scanners e máquinas de filmar ou fotográficas), diretorias (ou pastas) e ficheiros – que tornam o acesso à informação, por parte do sistema operativo, mais rápido e mais fácil, o que contribui para uma maior eficiência do computador;
- Criação, eliminação e alteração do nome de diretorias e ficheiros;
- Seleção, transferência e cópia de diretorias e ficheiros;
- Pesquisa/procura de ficheiros em, por exemplo, disco e respetivas diretorias;
- Visualização de, por exemplo, propriedades de diretorias e ficheiros.

O bit e seus múltiplos

Dados (recebidos e processados pelo computador), informação (resultante de uma operação de processamento) e/ou uma variável (utilizada para armazenar/guardar um valor) ocupam espaço nas memórias. Este espaço é medido em bit e múltiplos do bit (a "linguagem" do computador).

1 Bit (ou ***b**inary dig**it***) representa 2 estados máquina diferentes: 0 ou 1

1 Byte (ou um conjunto de 8 bits) representa $2^8 = 256$ estados máquina diferentes

1 KiloByte (ou 1 KB, que equivale a 1024 bytes) representa $2^{10} \times$ byte

1 MegaByte (ou 1 MB, que equivale a 1024 KB bytes) representa $2^{10} \times KB = 2^{10} \times 2^{10} \times$ byte

1 GigaByte (ou 1 GB, que equivale a 1024 MB bytes) representa $2^{10} \times MB = 2^{10} \times 2^{10} \times KB = 2^{10} \times 2^{10} \times 2^{10} \times$ byte

(…e assim sucessivamente vai aumentando a quantidade de informação que pode ser representada e armazenada/guardada; ou seja, quanto maior for o número de bytes, maior será o número de diferentes estados máquina, ou combinações de zeros e uns (isto é, dados e informação), que podem ser representados e armazenados/guardados.)

Problema: Que tipo de dados (e de informação) pode ser representado e armazenado/guardado (e.g., numa variável)?

Tipos de dados primitivos

	Sua representação em linguagem…		
Descrição	**Algorítmica**	**C (ANSI)**	**Exemplo**
Números inteiros (positivos e negativos)	inteiro	`int`, `long int`, …	12
Números inteiros e decimais (positivos e negativos)	real	`float` e *`double`*	45,6
Valores lógicos	lógico	Não se Aplica (N/A)	verdadeiro / falso / N/A
Caracter	caracter	`char`	"H" / 'H'
Conjunto de caracteres	texto	`string`	"esta frase"

Os números (para os humanos) são representados pelo sistema de numeração decimal e as letras/caracteres por um alfabeto. Os computadores representam esta informação em conjuntos de bits (ou conjuntos de zeros e uns) ou no sistema de numeração binário. Então, para representarmos dados e informação inteligível para nós humanos em computador, temos de ter uma forma de conversão entre os dois sistemas de numeração, por exemplo.

Sistema de Numeração Decimal: que conjunto de elementos diferentes utilizamos para representar um valor numérico?
Utilizamos os elementos do conjunto {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; isto é, um total de 10 elementos/algarismos diferentes para representarmos um valor numérico; trata-se então de um sistema decimal ou de base 10. Porquê "de base dez"? Vejamos o seguinte exemplo:

O número $4133_d = 4000 + 100 + 30 + 3 = 4 \times 1000 + 1 \times 100 + 3 \times 10 + 3 \times 1 = 4 \times 10^3 + 1 \times 10^2 + 3 \times 10^1 + 3 \times 10^0$

Sistema de Numeração Binário: que conjunto de elementos diferentes utiliza o computador para representar um valor?
O computador utiliza os elementos do conjunto {0, 1}; isto é, um total de 2 elementos diferentes para representar um valor; trata-se então de um sistema binário ou de base 2. Porquê "de base dois"? Vejamos o seguinte exemplo:

O número $1011_b = 1 \times$ 3 (terceiro) bit $+ 0 \times$ 2 (segundo) bit $+ 1 \times$ 1 (primeiro) bit $+ 1 \times$ bit das unidades =

[Com 3 bits conseguimos representar 8 combinações de zeros e uns (ou estados máquina) diferentes.
Com 2 bits conseguimos representar 4 combinações de zeros e uns (ou estados máquina) diferentes.
Com 1 bit conseguimos representar 2 combinações de zeros e uns (ou estados máquina) diferentes.]

$= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

Conversão do sistema binário para o decimal, exemplo:
$10110_b = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 16 + 0 + 4 + 2 + 0 = 22_d$

Conversão do sistema decimal para o binário, exemplo:
11_d / 2 = 5 (e resto **1**); 5 / 2 = 2 (e resto **1**); 2 /2 = **1** (e resto **0**)
$11_d = 1011_b$ [= (verificação) $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11_d$]

Com 8 bits (ou uma palavra de memória de 8 bits), os números inteiros que se seguem seriam representados e/ou guardados desta forma:

Nº inteiro	Nº binário	Capacidade de Armazenamento [a]
53	0 0 1 1 0 1 0 1	2^8 ϵ {0 .. 255}
22	0 0 0 1 0 1 1 0	2^8 ϵ {0 .. 255}
-22	**1** 0 0 1 0 1 1 0	2^7 ϵ {-128 .. 127} [b]

[a] Número de bits (ou o tamanho) da palavra de memória. Este determina o intervalo de valores que pode ser representado e/ou armazenado. Ou seja, o tamanho da palavra de memória determina a capacidade de representação e armazenamento de dados, informação e/ou variáveis.

[b] O bit mais à esquerda é reservado para representar o sinal: negativo (**1**) ou positivo (**0**).

Como se representam números reais em computador?

Qualquer número real pode escrever-se na forma $\mathbf{0.m \times 10^e}$. No mínimo, o computador utiliza duas palavras de memória: uma para guardar **m** (ou mantissa) e outra para guardar **e** (ou expoente); **m** e **e** são números inteiros. Vejamos o exemplo:

$3.5 = 0.35 \times 10^1$, em memória o computador vai guardar os números inteiros **m** = 35 e **e** = 1

Como se representam letras/caracteres em computador?

Cada letra é associada a um número inteiro (ou código) e é este que é representado e guardado em memória. Uma das tabelas de códigos mais utilizadas é a ASCII. Por exemplo, nesta tabela a letra 'A' corresponde ao nº 65, a 'B' ao nº 66, …, a 'a' ao nº 97, a 'b' ao nº 98, …

Pesquisa Internet: ver os formatos utilizados para guardar sons e imagens.

Algoritmia

Nesta Unidade Curricular (UC) vamos aprender a olhar para o computador como uma ferramenta de trabalho. Assim sendo, o computador deve ajudar-nos a resolver problemas. Mas para que isto aconteça, temos de ser nós a "ensinar" ao computador como é que ele poderá resolver-nos um determinado problema. (Quando bem instruído, o computador é mais rápido a resolver problemas do que nós, humanos, e a solução que apresenta tem maior probabilidade de estar correta do que uma solução conseguida manualmente).

Ao "ensinarmos" o computador a resolver um problema, vamos procurar indicar-lhe diferentes soluções. Mas temos de ser capazes de:

- Fazer um esboço, de cada uma dessas soluções, utilizando um algoritmo;
- Traduzir esses algoritmos numa linguagem de programação – C (ANSI), por exemplo (uma vez que os computadores de que dispomos ainda não "falam" muitas das linguagens utilizadas por nós humanos) e
- Avaliar as vantagens e desvantagens de representar em computador cada uma dessas soluções.

De notar que os computadores fazem aquilo que nós lhes pedimos para fazerem e não, necessariamente, o que nós queremos que eles façam!

Problema 1: Pretende-se um programa (ou procedimento) que escreva no ecrã do computador a seguinte mensagem – O meu 1º programa!

Solução algorítmica	Procedimento em C

inicio
  escrever "O meu 1º programa!"
fim
	`#include <stdio.h>`
	`main() {`
	`  printf("O meu 1° programa! \n");`
	`}`

Problema 2: Pretende-se um programa (ou procedimento) que (1) guarde, na memória do computador (isto é, numa variável de um tipo de dados adequado), a letra C, e (2) escreva, no ecrã, o conteúdo dessa variável.

Solução algorítmica	Anotações	Procedimento em C

inicio
  caracter letra
	Declaração da variável 'letra'	`#include <stdio.h>`
		`main() {`
		`  char letra;`

letra <- "C"	Atribuição à variável 'letra' do valor pretendido	`  letra = 'C';`
escrever letra **fim**	Apresentação de resultados	`  printf("%c", letra);` `}`

Problema 3: Pretende-se um programa (ou procedimento) que (1) guarde, na memória do computador (isto é, numa variável de um tipo de dados adequado), o valor inteiro 100 e (2) escreva, no ecrã, o conteúdo dessa variável.

<table>
<tr><th>Solução algorítmica</th><th>Anotações</th><th>Procedimento em C</th></tr>
<tr><td rowspan="3">Inicio
  inteiro numero

  numero <- 100

  escrever numero
fim</td><td>Declaração da variável ‘numero’</td><td rowspan="3">#include <stdio.h>
main() {
  int numero;

  numero = 100;

  printf("%d", numero);
}</td></tr>
<tr><td>Atribuição à variável ‘numero’ do valor pretendido</td></tr>
<tr><td>Apresentação de resultados</td></tr>
</table>

Problema 4: Pretende-se um programa (ou procedimento) que (1) peça um valor real ao utilizador, (2) guarde este valor na memória do computador (isto é, numa variável de um tipo de dados adequado) e (3) escreva, no ecrã, o valor inserido pelo utilizador.

<table>
<tr><th>Solução algorítmica</th><th>Anotações</th><th>Procedimento em C</th></tr>
<tr><td rowspan="3">inicio
  real numero

  ler numero

  escrever numero
fim</td><td>Declaração da variável ‘numero’</td><td rowspan="3">#include <stdio.h>
main() {
  float numero;

  scanf(“%f”, &numero);

  printf("%f", numero);
}</td></tr>
<tr><td>Atribuição à variável ‘numero’ do valor pretendido</td></tr>
<tr><td>Apresentação de resultados</td></tr>
</table>

Algumas definições

Algoritmo – sequência de ações/tarefas/instruções simples, não ambíguas, finitas e levadas a cabo numa ordem pré-definida para que o computador execute a solução de um problema.

Linguagem de programação – “conjunto de regras sintáticas, gramaticais e de pontuação sob a forma de código que permite a comunicação entre o ser humano e o computador através de instruções (frases); é o idioma que o computador entende.” [1]

Variável – é uma localização/posição na memória (do computador) onde são armazenados dados durante a execução de um programa. Cada variável tem um nome (ou designação, pelo qual é referida no programa), um tipo de dados (e.g., inteiro, real, caracter) e um conteúdo (ou o seu valor efetivo em qualquer ponto de execução do programa); este conteúdo pode variar ao longo da execução do programa.

Constante – apresenta as mesmas características de uma variável, contudo, é definida de forma diferente e o seu conteúdo não é alterado durante a execução de um programa.

Declaração de Variáveis – uma variável deve ser definida antes de ser utilizada no programa (uma das boas práticas de programação). Esta definição (1) indica ao computador qual o tipo de dado que fica associado ao nome que indicamos para a variável e (2) reserva espaço em memória para armazenar o conteúdo (ou valor) que lhe irá sendo atribuído ao longo do programa.

Input – a forma que quem utiliza o programa tem de fornecer-lhe dados e/ou informações.

Output – forma de o programa apresentar dados e/ou informações a quem o está a utilizar.

Atribuição de um Valor a uma Variável – consiste na alteração do conteúdo de uma variável pelo programa.

Operadores	Sua representação em linguagem…	
	Algorítmica	**C (ANSI)**
Aritméticos	+ (adição) - (subtração) / (divisão) * (multiplicação) % (resto da divisão inteira) ^ (potenciação)	+ (adição) - (subtração) / (divisão) * (multiplicação) % (resto da divisão inteira) N/A
Lógicos	E (conjunção) OU (disjunção) NAO (negação)	&& (conjunção) || (disjunção) ! (negação)
Relacionais	= (igual) =/= (diferente) > (maior) < (menor) >= (maior ou igual) <= (menor ou igual)	== (igual) != (diferente) > (maior) < (menor) >= (maior ou igual) <= (menor ou igual)
De Atribuição	<- (seta)	= (toma o valor de)

Utilização de constantes

Problema 5: Dado o preço (14,5 €) de um produto sem IVA incluído, pretende-se um programa (ou procedimento) que calcule o preço (deste produto) após IVA e apresente, no ecrã, o preço do mesmo sem IVA e com IVA incluído.

- Definição de constantes através da palavra reservada **const ;**

Solução algorítmica 5.0	Procedimento em C (versão 5.0)

inicio
  constante real IVA <- 0.28
  real preco <- 14.5, preco_iva, valor_iva
 	
  valor_iva <- preco * IVA
  preço_iva <- preco + valor_iva
 	
  escrever "O preço sem iva é " , preco
  escrever "O preço com iva é " , preco_iva
fim
	`#include <stdio.h>`
	
	`main() {`
	`  const float iva = 0.28;`
	`  float preço = 14.5, preço_iva, valor_iva;`
	
	`  valor_iva = preço * iva;`
	`  preço_iva = preço + valor_iva;`
	
	`  printf("O preço sem iva é %.2f\n", preço);`
	`  printf("O preço com iva é %.2f\n", preço_iva);`
	`}`

- Definição de constantes através da diretiva **#define**

Solução algorítmica 5.0	Procedimento em C (versão 5.1)

inicio
 	
  constante real IVA <- 0.28
 	
  real preco <- 14.5, preco_iva, valor_iva
 	
  valor_iva <- preco * IVA
  preço_iva <- preco + valor_iva
 	
  escrever "O preço sem iva é " , preco
  escrever "O preço com iva é " , preco_iva
fim
	`#include <stdio.h>`
	
	`#define IVA 0.28`
	
	`main() {`
	`  float preço = 14.5, preço_iva, valor_iva;`
	
	`  valor_iva = preço * IVA;`
	`  preço_iva = preço + valor_iva;`
	
	`  printf("O preço sem iva é %.2f\n", preço);`
	`  printf("O preço com iva é %.2f\n", preço_iva);`
	`}`

De notar que a solução algorítmica apresentada para o **problema 5** é a mesma nas duas tabelas. O que mudou foi a maneira como implementamos (ou traduzimos) essa solução em linguagem C (ANSI).

Problema 6: Pretende-se um programa (ou procedimento) que peça o valor do raio de um círculo ao utilizador, calcule e apresente (no ecrã) a superfície do mesmo.

Solução algorítmica	Procedimento em C

inicio
  constante real PI <- 3.14
  real raio, superfície
 	
  escrever "Qual o raio do círculo? "
  ler raio
 	
  superfície <- PI * raio ^ 2
 	
  escrever "A superfície do círculo é " , superfície
fim
	`#include <stdio.h>`
	
	`const float pi = 3.14;`
	
	`main() {`
	`  float raio, superfície;`
	`  printf("Qual o raio do círculo? ");`
	`  scanf("%f", &raio);`
	
	`  superfície = pi * raio * raio;`
	
	`  printf("A superfície do círculo é %.2f\n", superfície);`
	`}`

Referências

[1] "**Excel Macros & VBA Curso Completo**" de Henrique Loureiro, Lisboa: FCA, 2005

[2] "**Tecnologias de Informação: O que são? Para que servem?**" de Sérgio Sousa, Lisboa: FCA, 2003

Estruturas condicionais ou instruções de seleção – a instrução 'se'

Neste momento devemos conseguir ler algoritmos de execução sequencial (isto é, que começam na 1ª instrução e acabam na última) e que:

- ✓ Declaram e definem variáveis e constantes
- ✓ Atribuem valores a variáveis e efetuam cálculos com estas e com constantes
- ✓ Apresentam os resultados dos cálculos efetuados

Por vezes, durante a resolução de um problema, precisamos de fazer escolhas – isto é, selecionar de entre um conjunto de ações e/ou tarefas alternativas a mais adequada. Teremos então de utilizar as denominadas estruturas/instruções condicionais ou de seleção como, por exemplo, a instrução **SE**. A instrução **SE** escreve-se e lê-se das seguintes formas:

Instrução A: sintaxe	*Como se lê a instrução A?*	**Instrução B: sintaxe**	*Como se lê a instrução B?*
se *[condição]* **então** *[instruções 1]* **senão** *[instruções 2]* **fimSe**	**Se** a *[condição]* for verdadeira **então** é executado o conjunto de *[instruções 1]*; caso contrário – **senão**, (isto é, se a *[condição]* for falsa) é executado o conjunto de *[instruções 2]*.	**se** *[condição]* **então** *[instruções]* **fimSe**	**Se** a *[condição]* for verdadeira **então** é executado o conjunto de *[instruções]*.

A *[condição]* terá de ser formada por uma:

- ✓ (ou mais) Variável lógica (isto é, uma variável que guarde o valor 'verdadeiro' ou o valor 'falso'); ou
- ✓ Expressão que, utilizando os operadores lógicos e/ou relacionais (ver tabela de **Operadores**, pág. 9), dê como resultado 'verdadeiro' ou 'falso'.

******* Codifique os problemas que são apresentados a seguir em linguagem C (ANSI). *******

Problema 7: Pretende-se um programa (ou procedimento) que (1) calcule a média final (pesada) de um aluno a partir das notas que este obteve no teste 1 (que vale 40% da média final) e no teste 2 (que vale 60% da média final); e que (2) em função desta média final, indique se o aluno passou ou reprovou na UC em questão.

Solução Algorítmica 7.0

```
inicio
   real teste1, teste2, média
   constante real PESO1 <- 0.4, PESO2 <- 0.6

   escrever "Insira as notas dos testes 1 e 2: "
   ler teste1, teste2
   média <- PESO1 * teste1 + PESO2 * teste2
```

```
   se média >= 9.5 então
      escrever "Passou com média final de ", média
   senão
      escrever "Reprovou"
   fimSe
fim
```

Problema 8: Pretende-se um programa (ou procedimento) que (1) calcule a média final (pesada) de um aluno a partir das notas que este obteve no teste 1 (que vale 35% da média final) e no teste 2 (que vale 65% da média final); e que (2) em função desta média final, indique se o aluno passou, vai a oral, ou reprovou na UC em questão.

Solução Algorítmica 8.0

```
inicio
   real teste1, teste2, média
   constante real PESO1 <- 0.35, PESO2 <- 0.65

   escrever "Insira as notas dos testes 1 e 2: "
   ler teste1, teste2
   média <- PESO1 * teste1 + PESO2 * teste2

   se média >= 8 E média < 9.5 então
      escrever "Vai a oral"
   senão

      se média < 8 então
         escrever "Reprovou"
      senão
         escrever "Passou com média final de ", média
      fimse

   fimSe
fim
```

Problema 9: Pretende-se um programa (ou procedimento) que (1) peça as notas que um aluno obteve nos testes 1 e 2; (2) verifique se as notas inseridas estão na escala de 0 a 20 valores; (3) calcule, apenas com notas desta escala, a média final (pesada) desse aluno (sabendo que a nota do teste 1 vale 40% da média final e a do teste 2 vale 60% da mesma média); e que (4) indique, apenas em função da média final, se o aluno passou, vai a oral, ou reprovou na UC em questão.

Solução Algorítmica 9.0

```
inicio
   real teste1, teste2, média
   logico tst1 <- verdadeiro, tst2 <- verdadeiro
   constante real PESO1 <- 0.4, PESO2 <- 0.6

   escrever "Insira as notas dos testes 1 e 2: "
```

```
  ler teste1, teste2

  se teste1 < 0 OU teste1 > 20 entao
    tst1 <- falso
  fimse
  se teste2 < 0 OU teste2 > 20 entao
    tst2 <- falso
  fimse

  se tst1 E tst2 então
    média <- PESO1 * teste1 + PESO2 * teste2

    se média >= 9.5 então
      escrever "Passou com média final de ", média
    senão

      se média >= 8 então
        escrever "Vai a oral"
      senão
        escrever "Reprovou"
      fimse

    fimSe

  senão
    escrever "Não foi possível calcular a média."
  fimse
fim
```

Exercício: depois de termos compreendido a **solução algorítmica 9.0** (que resolve o **Problema 9**) vamos simplificá-la. Por exemplo, vamos escrever uma **solução algorítmica**, versão **9.1**, que utilize: (1) apenas uma variável do tipo lógico, 'escala', e (2) a condição 'teste1 < 0 OU teste1 > 20 OU teste2 < 0 OU teste2 > 20'.

Solução Algorítmica 9.1

```
inicio
  real teste1, teste2, média
  logico escala <- verdadeiro
  constante real PESO1 <- 0.4, PESO2 <- 0.6

  escrever "Insira as notas dos testes 1 e 2: "
  ler teste1, teste2

  se teste1 < 0 OU teste1 > 20 OU teste2 < 0 OU teste2 > 20 entao
    escala <- falso
  fimse

  se escala então
    média <- PESO1 * teste1 + PESO2 * teste2
```

```
    se média >= 9.5 então
      escrever "Passou com média final de ", média
    senão

      se média >= 8 então
        escrever "Vai a oral"
      senão
        escrever "Reprovou"
      fimse

    fimSe

  senão
    escrever "Não foi possível calcular a média."
  fimse
fim
```

Estruturas de controlo de ciclo – a instrução iterativa 'ENQUANTO'

Todos os algoritmos (ou procedimentos) que temos visto (e escrito) até agora executam cada uma das suas instruções (ou linhas) uma única vez, no máximo. No caso de o algoritmo (ou procedimento) ter estruturas condicionais **SE**, algumas das suas instruções podem nem chegar a ser executadas. Por exemplo, na **solução algorítmica 9.1**, se o aluno tem notas para passar à UC, (grosso modo) apenas são executadas as instruções até à linha 'escrever "Passou com média final de ", média', acabando o programa logo a seguir.

No entanto, para resolvermos um determinado problema, podemos ter necessidade de fazer com que uma dada instrução (ou um conjunto de instruções) seja executada várias vezes. Isto é, podemos ter necessidade de executar repetidamente uma, ou várias instruções. Neste caso, teremos de utilizar as denominadas estruturas/instruções de controlo de ciclo, cíclicas ou iterativas como, por exemplo, a instrução **ENQUANTO**.

A instrução **ENQUANTO** escreve-se e lê-se da seguinte forma:

Instrução ENQUANTO: sintaxe	*Como se lê a instrução ENQUANTO?*
enquanto *[condição]* **faz** *[instruções]* **fimEnquanto**	**Enquanto** a *[condição]* for verdadeira, é executado o conjunto de *[instruções]*. Ou seja, quando a instrução **enquanto** é encontrada, a *[condição]* é avaliada. Se o valor desta *[condição]* for 'verdadeiro', é executado o conjunto de *[instruções]* e a *[condição]* volta a ser avaliada. Este processo é repetido até o valor da *[condição]* ser 'falso'. Sendo a *[condição]* falsa, o programa continua a sua execução na instrução a seguir à **fimEnquanto**.

- ✓ A *[condição]* terá de ser formada por uma:
  - (ou mais) Variável lógica (isto é, uma variável que guarde o valor 'verdadeiro' ou o valor 'falso'); ou
  - Expressão que, utilizando os operadores lógicos e/ou relacionais (ver tabela de **Operadores**, pág. 9), dê como resultado 'verdadeiro' ou 'falso'.
- ✓ Quando a instrução **enquanto** é encontrada pela 1ª vez e o resultado da avaliação da *[condição]* é 'falso', as *[instruções]*, que compõem o '**corpo do ciclo**', não são executadas (ou são executadas zero vezes).
- ✓ Temos de ser muito cuidadosos a escolher as variáveis que farão parte da *[condição]* e a escrever a *[condição]*, pois devemos garantir que esta última acabará por ser falsa. Se isto nunca acontecer, as *[instruções]* (que compõem o '**corpo do ciclo**') serão executadas indefinidamente. Neste caso, estaremos em presença de um 'ciclo infinito' e de uma solução (ou programa) que nunca acaba!

*** Codifique os problemas que são apresentados a seguir em linguagem C (ANSI). ***

Problema 10: Pretende-se um programa (ou procedimento) que apresente os 10 primeiros números inteiros positivos por ordem crescente. (Nota: o número zero não é positivo nem negativo.)

Solução Algorítmica 10.0

```
inicio
   inteiro num <- 1

   enquanto num <= 10 faz
      escrever num, "\t"
      num <- num + 1
   fimenquanto
fim
```

Solução Algorítmica 10.1

```
inicio
   inteiro num <- 0

   enquanto num < 10 faz
      num <- num + 1
      escrever num, ";"
   fimenquanto
fim
```

Problema 11: Pretende-se um programa (ou procedimento) que apresente os 10 primeiros números inteiros positivos por ordem decrescente. (Nota: o número zero não é positivo nem negativo.)

Solução Algorítmica 11.0

```
inicio
   inteiro num <- 10

   enquanto num > 0 faz
      escrever num, " ; "
      num <- num – 1
   fimenquanto
fim
```

Solução Algorítmica 11.1

```
inicio
   inteiro num <- 11

   enquanto num > 1 faz
      num <- num – 1
      escrever num, "\n"
   fimenquanto
fim
```

Problema 12: Pretende-se um programa (ou procedimento) que (1) peça as notas que um aluno obteve nos testes 1 e 2; (2) volte a pedir que esta(s) seja(m) inserida(s) quando deteta uma nota fora da escala de 0 a 20 valores; (3) calcule, apenas com notas desta escala, a média final (pesada) desse aluno (sabendo que a nota do teste 1 vale 30% da média final e a do teste 2 vale 70% da mesma média); e que (4) indique, apenas em função da média final, se o aluno passou, vai a oral, ou reprovou na UC em questão.

Solução Algorítmica 12.0

```
inicio
   real teste1, teste2, média
   constante real PESO1 <- 0.3, PESO2 <- 0.7

   escrever "Insira a nota do 1º teste (de 0 a 20): "
   ler teste1
   enquanto teste1 < 0 OU teste1 > 20 faz
      escrever "Insira a nota do 1º teste (de 0 a 20): "
      ler teste1
   fimenquanto

   escrever "Insira a nota do 2º teste (de 0 a 20): "
   ler teste2
   enquanto teste2 < 0 OU teste2 > 20 faz
      escrever "Insira a nota do 2º teste (de 0 a 20): "
      ler teste2
   fimenquanto

   média <- PESO1 * teste1 + PESO2 * teste2

   se média < 8 então
      escrever "Reprovou porque a nota final é ", média
```

```
   senão

      se média < 9.5 então
         escrever "Vai a oral"
      senão
         escrever "Passou com nota final de ", média
      fimse

   fimse
fim
```

Estruturas de controlo de ciclo – a instrução iterativa 'PARA'

Outra das denominadas estruturas/instruções de controlo de ciclo, cíclicas ou iterativas que nos permite executar repetidamente uma, ou várias instruções é a instrução **PARA**. A instrução **PARA** escreve-se e lê-se da seguinte forma:

(1) **Instrução PARA: sintaxe**	
para *[variável numerica]* **de** *[valor inicial]* **ate** *[valor final]* **passo** *[valor de separação]* *[intruções]* **próximo**	
(2) **Instrução PARA: sintaxe**	
para *[variável numerica]* **de** *[valor inicial]* **ate** *[valor final]* *[intruções]* **próximo**	
Como se lê a instrução PARA?	Repete as *[instruções]* sob o controlo da *[variável numérica]*. A *[variável numérica]*: • Toma valores entre dois limites (inclusive) – o *[valor inicial]* e o *[valor final]* • Percorre (de forma crescente ou decrescente) todos os valores entre esses dois limites • É atualizada com o *[valor de separação]*, apropriado, em cada uma das repetições do ciclo. Quando o **passo** está omisso (ver a sintaxe da instrução (2)), o *[valor de separação]* (implícito) é igual a **1**. A *[variável numérica]* percorre assim o intervalo entre o *[valor inicial]* e o *[valor final]* apenas de forma crescente. Neste caso, o *[valor inicial]* terá sempre de ser inferior ao *[valor final]*. O ciclo termina quando o valor da *[variável numérica]* ultrapassa o *[valor final]*.

- ✓ Genericamente, utilizamos a instrução **PARA** quando sabemos à partida o número exato de vezes que queremos repetir as *[instruções]* do '**corpo do ciclo**'.
- ✓ Qualquer instrução iterativa **PARA** pode ser reescrita utilizando um ciclo **ENQUANTO**. O contrário nem sempre é possível! Os ciclos **ENQUANTO** utilizados na **solução algorítmica 12.0** (ver págs. 16 e 17) são um exemplo disso; isto é, dificilmente estes ciclos podem ser reescritos utilizando a instrução iterativa **PARA**.

*** Codifique os problemas que são apresentados a seguir em linguagem C (ANSI). ***

Problema 10: Pretende-se um programa (ou procedimento) que apresente os 10 primeiros números inteiros positivos por ordem crescente. (Nota: o número zero não é positivo nem negativo.)

Solução Algorítmica 10.2

```
inicio
   inteiro contador

   para contador de 1 ate 10
      escrever contador, "\n"
   proximo
fim
```

Problema 13: Pretende-se um programa (ou procedimento) que apresente os 10 primeiros números inteiros negativos por ordem crescente. (Nota: o número zero não é positivo nem negativo.)

Solução Algorítmica 13.0

```
inicio
   inteiro contador

   para contador de -10 ate -1
      escrever contador, "\t"
   proximo
fim
```

Problema 14: Pretende-se um programa (ou procedimento) que apresente os 5 primeiros números inteiros positivos por ordem decrescente. (Nota: o número zero não é positivo nem negativo.)

Solução Algorítmica 14.0

```
inicio
   inteiro contador

   para contador de 5 ate 1 passo -1
      escrever contador, ";"
   proximo
fim
```

Problema 15: Pretende-se um programa (ou procedimento) que apresente os 10 primeiros números inteiros, pares, positivos por ordem crescente. (Nota: o número zero não é positivo nem negativo.)

Solução Algorítmica 15.0

```
inicio

   inteiro contador

   para contador de 2 ate 20 passo 2
      escrever contador, ";"
   proximo
fim
```

Problema 16: Pretende-se um programa (ou procedimento) que apresente os 10 primeiros números inteiros, ímpares, positivos por ordem decrescente. (Nota: o número zero não é positivo nem negativo.)

Solução Algorítmica 16.0

```
inicio
   inteiro contador

   para contador de 19 ate 1 passo -2
      escrever contador, "\t"
   proximo
fim
```

Problema 17: Pretende-se um programa (ou procedimento) que calcule e apresente o somatório e o produto dos *n* primeiros números naturais (Nota: *n* é fornecido pelo utilizador do programa).

Solução Algorítmica 17.0

```
inicio
   inteiro n, num, soma <- 0, produto <- 1

   escrever "Quantos números naturais pretende somar e multiplicar?  "
   ler n

   para num de 1 ate n
      soma <- soma + num
      produto <- produto * num
   proximo

   escrever "Soma = ", soma, "\nProduto = ", produto
fim
```

Problema 18: Pretende-se um programa (ou procedimento) que apresente as 5 primeiras tabuadas, separadas por uma linha em branco.

Solução Algorítmica 18.0

```
inicio
   inteiro i , j
   para i de 1 ate 5

      para j de 1 ate 10
         escrever i , " * " , j , " = " , i * j , "\n"

         se j = 10 entao
            escrever "\n"
         fimse

      próximo

   próximo
fim
```

Estruturas de controlo de ciclo – a instrução iterativa 'repete'

A última das denominadas estruturas/instruções de controlo de ciclo, cíclicas ou iterativas que vamos estudar é a instrução **REPETE**. A instrução **REPETE** escreve-se e lê-se da seguinte forma:

Instrução REPETE: sintaxe	*Como se lê a instrução REPETE?*
repete *[instruções]* **até***[condição]*	**Repete** as *[instruções]* **até** a *[condição]* ser verdadeira. Ou seja, quando a instrução **repete** é encontrada são executadas as *[instruções]* e, de seguida, é avaliada a *[condição]*. Se o valor desta *[condição]* for 'falso', as *[instruções]* voltam a ser executadas e a *[condição]* avaliada. Este processo repete-se até o valor da *[condição]* ser 'verdadeiro'.

- ✓ A *[condição]* terá de ser formada por uma:
  - (ou mais) Variável lógica (isto é, uma variável que guarde o valor 'verdadeiro' ou o valor 'falso'); ou
  - Expressão que, utilizando os operadores lógicos e/ou relacionais (ver tabela de **Operadores**, pág. 9), dê como resultado 'verdadeiro' ou 'falso'.
- ✓ Quando a instrução **repete** é encontrada pela 1ª vez, as *[instruções]* que compõem o '**corpo do ciclo**' são executadas. Depois, sendo 'verdadeiro' o resultado da avaliação da *[condição]*, as *[instruções]* não voltam a ser executadas. Isto é, sempre que o resultado da 1ª avaliação da *[condição]* é 'verdadeiro', as *[instruções]* são executadas uma única vez.
- ✓ Temos de ser muito cuidadosos a escolher as variáveis que farão parte da *[condição]* e a escrever a *[condição]*, pois devemos garantir que esta última acabará por ser verdadeira. Se isto nunca acontecer, as *[instruções]* (que compõem o '**corpo do ciclo**') serão executadas indefinidamente. Neste caso, estaremos em presença de um 'ciclo infinito' e de uma solução (ou programa) que nunca acaba!
- ✓ De uma maneira geral, qualquer instrução iterativa **REPETE** pode ser reescrita utilizando um ciclo **ENQUANTO**; o contrário também é verdadeiro.
- ✓ Contudo, nem sempre é possível reescrever um ciclo **REPETE** utilizando uma instrução iterativa **PARA**. Por exemplo, os ciclos **REPETE** utilizados na versão **12.1** (ver págs. 22 e 23) do **problema 12** são um exemplo disso; isto é, dificilmente estes ciclos podem ser reescritos utilizando a instrução iterativa **PARA**. No entanto, o contrário já é possível: ver como a **solução algorítmica 17.0** (da pág. 19) pode ser reescrita nas versões **17.1** e **17.2** (da pág. 22).

*** Codifique os problemas que são apresentados a seguir em linguagem C (ANSI). ***

Problema 19: Pretende-se um programa (ou procedimento) que apresente os 5 primeiros números naturais, pares, por ordem crescente.

Solução Algorítmica 19.0

```
inicio
   inteiro par <- 2
   repete
      escrever par , "\t"
      par <- par + 2
   ate par = 12
fim
```

Solução Algorítmica 19.1

```
inicio
   inteiro par <- 0
   repete
      par <- par + 2
      escrever par , "; "
   ate par = 10
fim
```

Problema 17: Pretende-se um programa (ou procedimento) que calcule e apresente o somatório e o produto dos *n* primeiros números naturais (nota: *n* é fornecido pelo utilizador do programa).

Solução Algorítmica 17.1

```
inicio
   inteiro n, num <- 0, soma <- 0, produto <- 1

   escrever "Quantos números naturais pretende somar e multiplicar? "
   ler n

   repete
      num <- num + 1
      soma <- soma + num
      produto <- produto * num
   ate num = n

   escrever "Soma = ", soma, "\nProduto = ", produto
fim
```

Solução Algorítmica 17.2

```
inicio
   inteiro n, num <- 1, soma <- 0, produto <- 1

   escrever "Quantos números naturais pretende somar e multiplicar?  "
   ler n
```

```
   repete
      soma <- soma + num
      produto <- produto * num
      num <- num + 1
   ate num > n

   escrever "Soma = ", soma, "\nProduto = ", produto
fim
```

Problema 12: Pretende-se um programa (ou procedimento) que (1) peça as notas que um aluno obteve nos testes 1 e 2; (2) volte a pedir que esta(s) seja(m) inserida(s) quando deteta uma nota fora da escala de 0 a 20 valores; (3) calcule, apenas com notas desta escala, a média final (pesada) desse aluno (sabendo que a nota do teste 1 vale 30% da média final e a do teste 2 vale 70% da mesma média); e que (4) indique, apenas em função da média final, se o aluno passou, vai a oral, ou reprovou na UC em questão.

Solução Algorítmica 12.1

```
inicio
   real teste1, teste2, média
   constante real PESO1 <- 0.3, PESO2 <- 0.7

   repete
     escrever "Insira a nota do 1º teste (de 0 a 20): "
     ler teste1
   ate teste1 >= 0 E teste1 <= 20

   repete
      escrever "Insira a nota do 2º teste (de 0 a 20): "
      ler teste2
   ate teste2 >= 0 E teste2 <= 20

   média <- PESO1 * teste1 + PESO2 * teste2

   se média >= 8 E média < 9.5 entao
      escrever "Vai a oral"
   senao

      se média >= 9.5 então
         escrever "Passou com nota final de ", média
      senao
         escrever "Reprovou porque a nota final é ", média
      fimse

   fimse
fim
```

Problema 20: Pretende-se um programa (ou procedimento) que peça uma data (isto é, solicite um ano, um mês e um dia, separadamente) e a apresente da seguinte forma: “dd-mm-oano” (em que “dd” representa o dia – escrito com 2 algarismos, “mm” representa o mês – escrito com 2 algarismos, “oano” representa o ano – escrito com 4 algarismos). O programa verifica ainda se **(1)** o ano inserido pertence ao conjunto de 1900 a

2015, **(2)** o mês inserido é um valor entre 1 e 12 inclusive e **(3)** o dia do mês é válido. O programa **(a)** termina quando o utilizador insere o caracter 'n' (maiúsculo ou minúsculo) ou **(b)** continua a pedir e apresentar datas sempre que o utilizador insere o carater 's' (maiúsculo ou minúsculo).

Solução Algorítmica 20.0

```
inicio
   inteiro ano, mes, ultimodia, dia
   caracter tecla

   repete

      repete
         escrever "Ano (de 1900 a 2015): "
         ler ano
      ate ano >= 1900 E ano <= 2015

      repete
         escrever "Mês: "
         ler mes
      ate mes > 0 E mes < 13

      ultimodia <- 31

      se mes = 4 OU mes = 6 OU mes = 9 OU mes = 11 entao
         ultimodia <- 30
      senao

         se mes = 2 entao
            se ano % 400 = 0 OU ano % 4 = 0 E ano % 100 =/= 0 entao     //Condição que determina se 1 ano é bissexto
               ultimodia <- 29
            senao
               ultimodia <- 28
            fimse
         fimse
      fimse

      repete
         escrever "Dia: "
         ler dia
      ate dia > 0 E dia <= ultimodia

      escrever "\n Data: "
      se dia < 10 entao
         escrever "0"
      fimse
      escrever dia , "-"
```

```
    se mes < 10 entao
       escrever "0"
    fimse
    escrever mes , "-", ano

    repete
       escrever "\n Deseja continuar (s\\n) ? "
       ler tecla
    ate tecla = "s" OU tecla = "S" OU tecla = "n" OU tecla = "N"

  ate tecla = "n" OU tecla = "N"
fim
```

Dados estruturados – vetores (ou *arrays* unidimensionais) de comprimento fixo

Nos nossos algoritmos (ou procedimentos) temos utilizado variáveis simples, em que cada uma destas guarda um **único valor** em qualquer ponto da execução do programa. Isto é, até ao momento, aprendemos a associar apenas **um valor** (do tipo inteiro, real, caracter, lógico ou outro) a uma variável (ou nome). No entanto, também é possível associarmos uma sequência de valores a um único nome (ou variável). Como? Por exemplo, utilizando variáveis estruturadas (ou estruturas de dados) designadas por vetores (ou ***arrays*** unidimensionais).

- ✓ Ao declararmos um vetor, estamos a associar um nome (o nome da variável vetor) a uma sequência de valores dispostos consecutivamente na memória (ver **exemplo A**, mais abaixo, nesta página).
- ✓ Um vetor guarda uma sequência de valores todos do mesmo tipo (inteiro, real, caracter, lógico ou outro).
- ✓ A declaração de um vetor (ou *array* unidimensional) faz-se da seguinte forma (**sintaxe**):

  tipo_dados nome[*dimensão*]

  - o '*tipo_dados*' diz respeito ao tipo de dados (inteiro, real, caracter, lógico ou outro) de cada um dos elementos que compõe o vetor (ou a sequência);
  - o 'nome' é a designação pela qual o vetor é conhecido no programa;
  - o '*dimensão*' é um valor inteiro, constante (ou uma expressão cujo resultado é um valor inteiro), que indica quantos elementos tem o vetor.
- ✓ Podemos aceder a cada um dos elementos do vetor usando o seu nome e um índice, assim (**sintaxe**):

  nome[*índice*]

  - o 'nome' é a designação pela qual o vetor é conhecido no programa
  - o '*índice*' é um valor inteiro (ou uma expressão cujo resultado é um valor inteiro) que varia entre **0** e **'*dimensão*' – 1**

Exemplo A: Pretende-se definir uma estrutura de dados (ou variável) que permita guardar as classificações finais das cinco primeiras UC que o Manuel concluiu no curso.

Solução: Podemos **declarar** (ou definir) a seguinte variável (vetor ou estrutura de dados), denominada 'classificações': inteiro classificações[5]

	Memória
classificações[0]	11
classificações[1]	18
classificações[2]	15
classificações[3]	14
classificações[4]	10

✓ Evitamos, assim, declarar cinco variáveis distintas (uma por cada classificação), colocando todos estes dados numa <u>única variável</u> (vetor ou estrutura de dados) denominada 'classificações'.

✓ Para acedermos, por exemplo, à classificação final da **segunda** UC que o Manuel concluiu (e que, na tabela de cima, corresponde ao valor **18**) escrevemos: classificações[1]

*** Codifique os problemas que são apresentados a seguir em linguagem C (ANSI). ***

Problema 21: Pretende-se um programa (ou procedimento) que guarde, no vetor 'raízes', as raízes quadradas dos valores 25, 49, 144, 4 e 100, que se encontram no vetor 'nums'.

Solução Algorítmica 21.0

```
inicio
   inteiro nums[5] <- {25,49,144,4,100}          //Conjunto de valores atribuídos pelo programa ao vetor 'nums'
   inteiro raízes[5], i
   para i de 0 ate 4
      raízes[i] <- raiz(nums[i])      //A função 'raiz(x)' recebe um valor, x, e devolve a raiz quadrada deste valor
   proximo
fim
```

Solução Algorítmica 21.1

```
inicio
   inteiro nums[5] <- {25,49,144,4,100}
   inteiro raízes[5], i <- 0
   enquanto i < 5 faz
      raízes[i] <- raiz(nums[i])
      i <- i + 1
   fimenquanto
fim
```

Solução Algorítmica 21.2

```
inicio
   inteiro nums[5] <- {25,49,144,4,100}
   inteiro raízes[5], i <- 0
   repete
      raízes[i] <- raiz(nums[i])
      i <- i + 1
   ate i > 4
fim
```

Problema 22: Pretende-se um programa (ou procedimento) que calcule a média de idades dos 20 alunos de uma turma de inglês técnico (nota: cada aluno fornece a sua idade ao programa).

Solução Algorítmica 22.0

```
inicio
  real idades[20], soma <- 0, media
  inteiro i

  para i de 0 ate 19
    escrever "Insira a sua idade: "
    ler idades[i]
  proximo

  para i de 0 ate 19
     soma <- soma + idades[i]
  proximo

  media <- soma / 20
  escrever "A media de idades dos ", i, " alunos é ", media
fim
```

Solução Algorítmica 22.1

```
inicio
  real idades [20], media, soma <- 0
  inteiro i

  para i de 0 ate 19
    escrever "Insira a sua idade: "
    ler idades[i]
    soma <- idades[i] + soma
  proximo

  media <- soma / i
  escrever "A media de idades dos ", i, " alunos é ", media
fim
```

Problema 23: Pretende-se um programa (ou procedimento) que leia e guarde as precipitações (em mm) referentes a cada um dos meses ímpares do ano de 2014 e, no final, apresente a precipitação associada a cada mês sob a seguinte forma: “Em janeiro choveram 72 mm”

Solução Algorítmica 23.0

```
inicio
   real prec[13]
   inteiro i

   para i de 1 ate 12 passo 2
      escrever "Introduza a precipitação do ", i, "º mes do ano: "
      ler prec[i]
   proximo
```

```
  i <- 1
  enquanto i < 13 faz
    escolhe i
      caso 1: escrever "Em janeiro choveram ", prec[i], " mm\n"
      caso 3: escrever "Em março choveram ", prec[i], " mm\n"
      caso 5: escrever "Em maio choveram ", prec[i], " mm\n"
      caso 7: escrever "Em julho choveram ", prec[i], " mm\n"
      caso 9: escrever "Em setembro choveram ", prec[i], " mm\n"
      caso 11: escrever "Em novembro choveram ", prec[i], " mm\n"
    fimescolhe
    i <- i + 2
  fimenquanto
fim
```

NOTA: tal como a instrução 'se', **'**escolhe' é uma instrução condicional ou de seleção (ver descrição na "Ajuda da Linguagem" da aplicação **PortugolViana**). Geralmente, esta última pode ser reescrita à custa de uma série de instruções 'se' aninhadas**.**

Problema 24: Pretende-se um programa (ou procedimento) que (1) simule seis lançamentos de um dado, guardando os valores correspondentes a cada lançamento; (2) peça ao jogador que escolha um número (também de 1 a 6, inclusive); e que verifique e comunique se o número escolhido, pelo jogador, coincide com algum dos seis lançamentos simulados.

Solução Algorítmica 24.0

```
inicio
  inteiro dado[6], num, i

  para i de 0 ate 5
     dado[i] <- arred(aleatorio() * 5 + 1)  //Ver NOTAS, mais abaixo, sobre as funções 'arred(x)' e 'aleatorio()'
  proximo

  repete
    escrever "Escolha um valor de 1 a 6, inclusive: "
    ler num
  ate num > 0 E num < 7

  i <- 0
  enquanto i < 6 E dado[i] =/= num faz
     i <- i + 1
  fimenquanto

  se i > 5 entao
     escrever num, " não corresponde a nenhum lançamento do dado"
  senao
     escrever num, " corresponde ao ", i + 1, "º lançamento do dado"
  fimse

fim
```

NOTAS: A função '**aleatorio**()' gera um valor real no intervalo, fechado, de 0 a 1. A função '**arred**(*x*)' recebe um valor real, *x*, e devolve um valor inteiro (ou sem parte decimal); este valor inteiro resulta do arredondamento do valor *x*.

Funções versus procedimentos

Nos nossos algoritmos (ou procedimentos) temos utilizado **funções** pré-definidas (isto é, funções escritas por outros programadores). Seguem alguns exemplos:

Funções pré-definidas nas linguagens:	
Algorítmica	**C (ANSI)**
raiz (*x*)	sqrt (*x*)
aleatorio ()	rand ()

Para utilizarmos uma **função** pré-definida (tal como acontece com as funções matemáticas) basta sabermos o 'nome' da função, o(s) valor(es) (ou 'argumento(s)') que esta aceita e o 'resultado' que a função produz (ou devolve). Por exemplo, sobre a função – **raiz (x)** da linguagem algorítmica, sabemos tratar-se de uma função pré-definida com (1) o 'nome' de **raiz**; que (2) aceita um 'argumento' (ou valor do tipo inteiro ou real), ***x*** ; e (3) devolve a **raiz quadrada de *x*** como 'resultado'.

Mas podemos ser nós a escrever (ou definir) as nossas próprias funções. Fazemo-lo com o objetivo de simplificar um algoritmo (ou procedimento) complexo e extenso. Esta simplificação passa por dividirmos o algoritmo (ou procedimento) em partes (módulos ou **funções**) que contêm tarefas elementares bem definidas. Um módulo (ou **função**) é mais fácil de ler, entender, testar, corrigir, alterar e codificar do que um programa (ou procedimento), extenso, que contém múltiplas tarefas interligadas entre si. Cada módulo (ou **função**) deve ainda ser independente de qualquer outro. Porquê? Se o for, poderá ser reutilizado em diferentes algoritmos (e procedimentos) e só temos de o escrever uma vez!

As **funções** escritas (ou definidas) por nós (tal como as pré-definidas) podem ser utilizadas em qualquer algoritmo (ou procedimento) que designaremos de 'principal' de agora em diante. Assim, sempre que formos nós a definir (ou escrever) uma **função** e quisermos utilizá-la num algoritmo principal (também escrito por nós), procedemos da seguinte forma:

Definição de uma **função** e sua utilização num **algoritmo principal: sintaxe**

```
funcao <tipo_dados>  nome_funcao(<tipo_dados> parametro1, …, <tipo_dados> parametroN)
  inicio
    [instruções]
    retornar variavel/valor/expressão
fimfuncao

inicio                                                    //Algoritmo principal
    [1º bloco de instruções]
    variavel <- nome_funcao(argumento1, ..., argumentoN)
    [2º bloco de instruções]
fim
```

Como **se lê** esta **sintaxe**?

- ✓ Começa-se pelo '**inicio**' do algoritmo principal que chama (ou utiliza) a função designada por 'nome_funcao'
- ✓ Prossegue-se com a execução do *[1º bloco de instruções]* no algoritmo principal
- ✓ Quando é encontrada a instrução 'nome_funcao(argumento1, ..., argumentoN)' (isto é, quando a **função é chamada** para ser utilizada), o algoritmo principal é interrompido temporariamente
- ✓ Passa-se, então, para a execução da '**função**', sendo o 'argumento1' (proveniente do algoritmo principal) copiado para o 'parametro1' da função, o 'argumento2' para o 'parametro2' e assim sucessivamente. (Este processo é também conhecido por 'passagem de argumentos por valor'.)
- ✓ Posteriormente, são executadas as *[instruções]* que correspondem ao corpo da função
- ✓ Quando é encontrada a instrução '**retornar** *variavel/valor/expressão*', a função devolve um valor (ou resultado) para o algoritmo principal e termina a sua execução ('**fimfuncao**')
- ✓ Neste ponto é retomada a execução do algoritmo principal (que havia sido temporariamente interrompida) na instrução de atribuição '*variavel* <- nome_funcao(argumento1, ..., argumentoN)'. Isto significa que o resultado devolvido (ou retornado/produzido) pela função fica guardado na '*variavel*'
- ✓ Prossegue-se depois com a execução do *[2º bloco de instruções]* no algoritmo principal. Este termina quando é encontrado o '**fim**'.

Observações (sobre a **sintaxe**):

- ✓ O **cabeçalho** de cada função tem a seguinte forma:

  funcao <*tipo_dados*> nome_funcao(<*tipo_dados*> parametro1, …, <*tipo_dados*> parametroN)
  - O '*tipo_dados*' (inteiro, real, ou outro) – a seguir à palavra reservada '**função**', define/identifica o tipo de dados do resultado devolvido (ou retornado/produzido) pela função
  - 'nome_funcao' é o nome que o programador atribui à função que está e escrever (ou definir)
  - Os parâmetros ('parametro1', …, 'parametroN') são variáveis declaradas no cabeçalho da função, que recebem os argumentos ('argumento1', …, 'argumentoN') enviados pelo algoritmo principal (ou procedimento) quando este chama (ou utiliza) a função
- ✓ O 'parametro1' é a única variável que pode receber o 'argumento1' (o mesmo acontece com os restantes parâmetros e argumentos). Isto significa o seguinte:
  - O 'argumento1' tem de ocupar a mesma posição, na lista de argumentos, que ocupa o 'parametro1' na lista de parâmetros
  - Ambos ('argumento1' e 'parametro1') têm de ser do mesmo *tipo_dados*
- ✓ Um 'argumento' pode ser uma variável, um valor ou uma expressão.
- ✓ Uma função pode não aceitar 'argumentos'. É o caso da função **aleatorio()**.

- ✓ Uma função devolve um único valor (ou produz um resultado único) a partir dos argumentos que lhe são passados pelo algoritmo principal
- ✓ Uma função que não devolve qualquer valor (ou não produz qualquer resultado) tem o nome de **procedimento**. (Note que as soluções algorítmicas escritas até aqui são **procedimentos**.)

*** Codifique os problemas que são apresentados a seguir em linguagem C (ANSI). ***

Problema 17: Pretende-se reescrever a **solução algorítmica 17.1** definindo agora duas funções: Soma e Produto.

Solução Algorítmica 17.3

```
funcao inteiro Soma(inteiro valor, inteiro somatorio)
 inicio
   somatorio <- somatorio + valor

   retornar somatorio
fimfuncao

//-----------------------------------------------------------------
funcao inteiro Produto(inteiro valor, inteiro resultado)
 inicio
   resultado <- resultado * valor

   retornar resultado
fimfuncao
//-----------------------------------------------------------------
inicio                                          //Algoritmo principal
  inteiro n, num <- 0, sum <- 0, prod <- 1

  escrever "Insira o limite máximo de n.os naturais a somar e multiplicar: "
  ler n

  repete
    num <- num + 1
    sum <- Soma(num, sum)
    prod <- Produto(num, prod)
  ate num = n

  escrever "Soma = ", sum, "\nProduto = ", prod
fim
```

Problema 9: Pretende-se reescrever a **solução algorítmica 9.0** definindo agora duas funções: Verifica (que averigua se as notas dos testes estão na escala de 0 a 20) e ResultadoUC (que indica se o aluno passou, vai a oral, ou reprovou na UC em questão).

Solução Algorítmica 9.2

```
função logico Verifica(real nota)
 inicio
  logico tst <- verdadeiro

   se nota < 0 OU nota nota > 20 entao
     tst <- falso
   fimse

   retornar tst
fimfuncao
//-----------------------------------------------------------------
funcao texto ResultadoUC(real média)
 inicio
   texto frase

     se média >= 9.5 então
       frase <- "Passou à UC"
     senao
       se média >= 8 então
         frase <- "Vai a oral"
       senão
         frase <- "Reprovou"
       fimse
     fimSe

   retornar frase
fimfuncao
//-----------------------------------------------------------------
inicio                                          //Algoritmo principal
  real teste1, teste2, média
  logico tst1, tst2
  constante real PESO1 <- 0.4, PESO2 <- 0.6

  escrever "Insira as notas dos testes 1 e 2: "
  ler teste1, teste2

  tst1 <- Verifica(teste1)
  tst2 <- Verifica(teste2)

   se tst1 E tst2 então
     média <- PESO1 * teste1 + PESO2 * teste2
     escrever ResultadoUC(média), " com média de ", média
   senão
     escrever "Não foi possível calcular a média"
   fimse

fim
```

NOTA: embora tenham o mesmo nome, a variável 'média' do algoritmo principal é diferente (isto é, ocupa um espaço em memória diferente) do parâmetro da seguinte **função** – 'ResultadoUC(média)'. Isto acontece porque estamos a passar 'argumentos por valor' para as funções.

Problema 25: Pretende-se um programa (ou procedimento) que apresente os 10 primeiros números naturais por ordem crescente e decrescente à custa de duas funções: Crescente e Decrescente. (Nota: as funções nem sempre têm parâmetros.)

Solução Algorítmica 25.0

```
funcao inteiro Crescente()
 inicio
   inteiro contador, cresce <- 1

   para contador de 1 ate 10
      escrever contador, "\t "
   proximo

   retornar cresce
fimfuncao
//-------------------------------------------------------------
funcao inteiro Decrescente()
 inicio
   inteiro contador, decresce <- 1

   para contador de 10 ate 1 passo -1
      escrever contador, "\t "
   proximo

   retornar decresce
fimfuncao
//-------------------------------------------------------------
inicio                                          //Algoritmo principal
   inteiro ordem

      ordem <- Crescente()
      escrever "\n"
      ordem <- Decrescente()
fim
```

Problema 12: Pretende-se reescrever a **solução algorítmica 12.0** definindo uma função Leitura e reutilizando a função ResultadoUC anteriormente definida.

Solução Algorítmica 12.2

```
funcao real Leitura()
 inicio
   real teste
   escrever "Insira a nota do teste (de 0 a 20): "
   ler teste
   enquanto teste < 0 OU teste > 20 faz
      escrever "Insira a nota do teste (de 0 a 20): "
      ler teste
   fimenquanto
   retornar teste
fimfuncao
```

```
//------------------------------------------------------------------
funcao texto ResultadoUC(real média)
   // Reutilizar a função definida na solução algorítmica 9.2.
fimfuncao
//------------------------------------------------------------------
inicio                                                    //Algoritmo principal
   real teste1, teste2, média
   constante real PESO1 <- 0.3, PESO2 <- 0.7

   teste1 <- Leitura()
   teste2 <- Leitura()

   média <- PESO1 * teste1 + PESO2 * teste2

   escrever ResultadoUC(média) , " com média de ", média
fim
```

Problema 20: Pretende-se reescrever a **solução algorítmica 20.0** definindo quatro funções: LeValor, Ultimo_Dia, EscreveValor e Termina.

Solução Algorítmica 20.1

```
funcao inteiro LeValor(inteiro inferior, inteiro superior)
 inicio
   inteiro valor
   repete
      escrever " (insira um valor válido): "
      ler valor
   ate valor >= inferior E valor <= superior
   retornar valor
fimfuncao
//------------------------------------------------------------------
funcao inteiro Ultimo_Dia (inteiro mes, inteiro ano)
 inicio
   inteiro fimmes

     fimmes <- 31
     se mes = 4 OU mes = 6 OU mes = 9 OU mes = 11 entao
        fimmes <- 30
     senao
        se mes = 2 entao
           se ano % 400 = 0 OU ano % 4 = 0 E ano % 100 =/= 0 entao     //Condição que determina se 1 ano é
bissexto
              fimmes <- 29
           senao
              fimmes <- 28
           fimse
        fimse
     fimse

   retornar fimmes
fimfuncao
```

```
//--------------------------------------------------------------
funcao caracter EscreveValor(inteiro num)
 inicio
    caracter zero <- “”

    se num < 10 entao
        zero <- “0”
    fimse

   retornar zero
fimfuncao
//--------------------------------------------------------------
funcao caracter Termina()
 inicio
   caracter stop

   repete
      escrever "\n Deseja continuar (s\\n) ? "
      ler stop
   ate stop = "s" OU stop = "S" OU stop = "n" OU stop = "N"

   retornar stop
fimfuncao
//--------------------------------------------------------------
inicio                                                  //Algoritmo principal
  inteiro ano, mes , dia , ultimodia
  caracter stop

  repete
     escrever “Insira um ano (de 1900 a 2014) \n ”
     ano <- LeValor(1900, 2014)

     escrever “Insira um mes (1 a 12) \n ”
     mes <- LeValor(1, 12)
     ultimodia <- Ultimo_Dia(mes, ano)

     escrever “Insira um dia \n ”
     dia <- LeValor(1, ultimodia)

     escrever "Data: ", EscreveValor(dia), dia , "-", EscreveValor(mes), mes , "-", ano

     stop <- Termina()

  ate stop = "n" OU stop = "N"
fim
```