Sebenta

de

Programação de Computadores

2001/2002

por

Miguel Abrunhosa de Brito

(Departamento de Sistemas de Informação)

1.º ano de Eng.ª Electrónica Industrial

(Departamento de Electrónica Industrial)

[1.- Introdução 3](#_Toc496515247)

[2.- “Nós” somos o programador! 5](#_Toc496515248)

[3.- Escrever no ecrã 5](#_Toc496515249)

[4.- Fazer Cálculos e usar Variáveis 6](#_Toc496515250)

[5.- Ler do teclado 10](#_Toc496515251)

[6.- Estrutura Condicional “se-então-senão” 13](#_Toc496515252)

[7.- Estrutura Condicional “caso-seja” 16](#_Toc496515253)

[8.- Estrutura de Controlo de Ciclos “repetir-até” 19](#_Toc496515254)

[9.- Estrutura de Controlo de Ciclos “enquanto-fazer” 20](#_Toc496515255)

[10.- Estrutura de Controlo de Ciclos “para-fazer” 22](#_Toc496515256)

[*11.-* *Salada* de Ciclos 24](#_Toc496515257)

[12.- Sub-rotinas 27](#_Toc496515258)

[13.- Sequências de caracteres (*strings*) 27](#_Toc496515259)

[14.- Vectores unidimensionais (*Arrays*) 28](#_Toc496515260)

[15.- Vectores multidimensionais (Matrizes) 29](#_Toc496515261)

[16.- Tuplos (Registos) 31](#_Toc496515262)

[17.- Ficheiros Sequenciais 33](#_Toc496515263)

[18.- Pilhas (*Stacks*) 33](#_Toc496515264)

[19.- Filas (*Queues*) 33](#_Toc496515265)

[Anexo I. Traçagem (*dry-running*) de programas 34](#_Toc496515266)

[Anexo II. Cábula com Sintaxe do Qbasic 36](#_Toc496515267)

[Anexo III. Tabela ASCII 39](#_Toc496515268)

[Anexo IV. C: Operadores, Palavras Reservadas e Opções de Compilação 40](#_Toc496515269)

[Anexo V. Ambiente de Desenvolvimento Integrado e Editor de Turbo C 41](#_Toc496515270)

[Anexo VI. C: Funções de Entrada e de Saída (E/S) 42](#_Toc496515271)

[Anexo VII. C: Funções de Caracteres e de Sequências de Caracteres (Strings) 43](#_Toc496515272)

[Anexo VIII. C: Funções Matemáticas 44](#_Toc496515273)

[Anexo IX. C: Funções Diversas 45](#_Toc496515274)

[Anexo X. Significado e Utilização do Binário nos Computadores 46](#_Toc496515275)

[Anexo XI. Vectores em C 47](#_Toc496515276)

[Anexo XII. Tuplos (structs) em C 48](#_Toc496515277)

[Anexo XIII. Pilhas (*Stacks*) 49](#_Toc496515278)

[Anexo XIV. Recursividade 50](#_Toc496515279)

[Anexo XV. Filas (*Queues*) 51](#_Toc496515280)

[Anexo XVI. Apontadores em C 52](#_Toc496515281)

# Introdução

Pré-requisitos: Os alunos devem ter noções básicas sobre diversas matérias que, não sendo directamente necessárias ao programa do curso, são importantes para o desempenho nas aulas.

|  |  |
| --- | --- |
| **Computadores** | Principais componentes de um computador (CPU, dispositivos E/S, memórias etc.) |
|  | Familiarização com o teclado e o rato. |
| **Sistemas Operativos** | Alguma experiência básica de utilização de sistemas operativos (DOS, Windows, Unix etc.) principalmente com linhas de comando. |
|  | Perceber o conceito de estrutura de directorias e a noção de caminho (“path”). |
|  | Comandos básicos de manipulação de directorias (Criar, navegar, remover etc.). |
|  | Comandos básicos de manipulação de ficheiros (Copiar, mudar o nome, apagar etc.). |
| **Internet** | Ter uma conta de email[[1]](#footnote-1) (e saber utilizá-la!) |
|  | Saber consultar (navegar) páginas http |
| **Matemática** | Números primos, Factorial, conversão de unidades (p. ex. Centígrados - Fahrenheit) |
|  | Menor múltiplo comum e máximo divisor comum |
|  | Divisão inteira (sim, aquela com quociente e resto!) |
|  | Lógica booleana (negação, conjunção, disjunção, tabelas de verdade etc.) |
|  | Conversão entre bases de numeração (binário[[2]](#footnote-2), decimal, hexadecimal etc.) |
| **Ambiente de trabalho** | Garantir que no(s) computador(es) de estudo (em casa, no CI etc.) está disponível um ambiente de desenvolvimento para linguagem C. Nas aulas práticas é usada uma versão antiga do “Turbo C”. Existe um ambiente de trabalho “freeware” (o “DJGPP”) que pode ser disponibilizado para cópia a quem o solicitar. |

**Contactar os docentes:** A forma preferencial de contacto com os docentes, fora do horário das aulas, é durante o respectivo horário de atendimento depois de previamente marcado por email. À entrada do Departamento de Sistemas de Informação (DSI) existe um telefone interno e uma relação dos números de telefone dos docentes. Usar esse telefone para, à hora marcada, avisar o docente da presença do aluno no departamento.

Página *http* da disciplina: A informação relevante sobre a disciplina (avisos, notas, contactos, bibliografia, enunciados de exames de anos anteriores, sugestões de exercícios a resolver etc.) encontra-se disponível em <http://www.dsi.uminho.pt/disciplinas/EIPCI>. A sua consulta regular é considerada obrigatória e a informação nela contida pressupõe-se distribuída aos alunos.

Ambiente de trabalho e linguagens: A principal componente da disciplina é a resolução de problemas de programação de computadores. Toda a resolução de problemas é feita usando linguagem algorítmica. Para testar os programas no computador, é usada a linguagem C. Nas aulas práticas é usada uma versão antiga do “turbo C[[3]](#footnote-3)” (ambiente integrado da então *Borland, inc.*) para *ms-dos*. Quem não tiver o *turbo C* no computador onde estuda pode fazer o *download* de um ambiente *freeware* nele inspirado, o DJGPP, da página da disciplina, ou do respectivo *site* na *Net* (ver o endereço exacto na página da disciplina, em “ferramentas”).

**Sugestões para concluir a disciplina com sucesso:** O tipo de matéria desta disciplina exige uma abordagem e estudo muito diferentes das da generalidade das disciplinas. De um modo geral, é de alertar para o facto de a quantidade de matéria *estudável* ser mínima. O que se pretende que o aluno aprenda é a desenvolver o tipo de raciocínio necessário à resolução de problemas de programação de computadores. Este raciocínio tem que ser *construído* metodicamente, semana a semana. A dificuldade da disciplina é precisamente essa ⎯ um aluno que, por qualquer motivo, faltou a uma aula ou não praticou a matéria dessa semana começa imediatamente a deixar de conseguir acompanhar a matéria e é frequentemente tentado a pensar que depois *estuda para exame*! Esse aluno está praticamente *chumbado* sem o saber! Para que o aluno possa *ultrapassar* com desenvoltura a disciplina de Programação de Computadores, sugerem-se os seguintes cuidados e metodologia:

|  |  |
| --- | --- |
| **1.ª aula** | A primeira aula da disciplina é a mais importante! Infelizmente é a menos frequentada... |
| **Ciclo semanal de estudo** | Cada *semana* começa com a **aula teórica**. A matéria dessa semana é exposta e explicada. Sempre que possível é exemplificada com a resolução de problemas apropriados. |
|  | **Todas as dúvidas** ou dificuldades devem ser esclarecidas imediatamente durante a aula. O aluno que sinta que perdeu o *fio à meada* deve atacar a questão com a urgência e a preocupação de quem ataca um incêndio ⎯ cada minuto pode ser fatal! ⎯. |
|  | **Antes da aula prática** seguinte, devem ser feitos os exercícios recomendados. Um erro típico, é pensar que como se percebeu a matéria não é necessário praticar ⎯ o erro está em pensar que basta perceber para saber fazer... ⎯. |
|  | **Na aula prática** são codificados e introduzidos no computador, os algoritmos feitos *em casa*. Quem não resolveu problemas em casa não tem nada para fazer na aula prática... |
|  | **Antes da aula teórica** devem ser relembradas a matéria da semana anterior e as dificuldades encontradas. Se possível, *folhear* esta sebenta para ficar com o enquadramento da matéria que vai ser dada. |
| **Como perder tempo, julgando que se está a estudar** | Como em tudo na vida, não basta fazer muita força: é necessário fazê-la na direcção certa! Para isso, vale a pena investir algum tempo a pensar na melhor estratégia para cumprir com os objectivos de aprendizagem da disciplina. Dada a particularidade do tipo de matéria, é frequente haver alunos que não conseguem cumprir os objectivos mínimos, apesar de despenderem esforço em quantidade que, correctamente orientado, seria suficiente para o conseguirem! Apresentam-se seguir alguns erros típicos. No entanto o mote é sempre o mesmo: o esforço tem que ser individual, e só se consegue aprender, fazendo! |
|  | Resolvendo os problemas de que **já se viu a resolução**, não se aprende a analisar o enunciado e a idealizar a solução, que é precisamente a lacuna mais frequente e que mais insucessos causa. |
|  | Pela mesma razão, **estudar com um colega** só funciona se servir para confrontar as ideias e soluções a que cada um chegou. Há um grande risco de que o mais expedito a conseguir idealizar uma solução exponha a sua ideia ao outro e continuem a partir daí, fazendo com que um dos dois salte sistematicamente o processo de análise, sem se aperceber que na realidade não está a aprender a fazê-lo! |
|  | Há casos ainda piores, em que um faz tudo enquanto o outro **vai dando *palpites***! Fica com a sensação de que participou activamente no estudo, quando na realidade pouco ou nada aprendeu! |
|  | **Resolver 10 vezes**, da mesma maneira, o mesmo exercício na tentativa de decorar a resolução. Muito frequente em alunos que já não estão a perceber a matéria, é começarem a tentar resolver de cor exercícios resolvidos que já viram resolvidos por outros. O problema com esta abordagem é que, para além de um problema ter inúmeras soluções, a solução nunca se constrói sequencialmente da primeira para última linha, ou seja, mesmo que decorem a solução em causa, estão a fazê-lo por uma ordem que não serve para aprender nada! |
|  | Perder 3 dias e 3 noites a **colocar efeitos gráficos num programa** de batalha naval. É instrutivo, é motivante, mas não pode ser feito em detrimento da análise e resolução de enunciados de outros exercícios. |
| **Cábulas** | Uma das actividades mais recompensadoras, é manter uma *cábula* com aquela informação que por não estar a jeito, nos faz perder o raciocínio que estávamos a seguir. De facto, a resolução de problemas de programação de computadores é muito exigente em termos de esforço de concentração, e uma das capacidades mais importantes a desenvolver, é precisamente a de conseguir organizar todo o raciocínio sem nunca perder o contexto do que se está a fazer. Uma boa cábula é um auxiliar precioso! |
|  | Tipicamente, deve conter pormenores de sintaxe em que costumamos hesitar, lembretes de coisas de que nos costumamos esquecer, e um ou dois excertos de algoritmos de que estamos sempre a precisar[[4]](#footnote-4). |

**Caixas de texto**

Ao longo do texto são introduzidas caixas de texto que, à semelhança das notas de rodapé, contêm informação importante, mas que foi retirada do miolo do texto para não perturbar a fluência da leitura.

Além das caixas que delimitam programas codificados em diversas linguagens de programação, há ainda diversos tipos de caixas, que são identificadas pelo símbolo no canto superior esquerdo. A cada símbolo corresponde um objectivo diferente, conforme se discrimina a seguir.

☝ ­‑ Observações complementares à explicação. De um modo geral correspondem a dúvidas típicas que podem surgir sobre a matéria em causa.

🗹 ­‑ Geralmente são sugestões de alteração a exemplos apresentados, de modo a que o aluno possa verificar se o compreendeu integralmente a matéria em questão.

🕮 - Conselhos de estudo (quanto à forma).

# “Nós” somos o programador!

A primeira noção que, quem quer aprender a programar, tem que ter presente, é que existem pelo menos três entidades envolvidas num programa: i) o Computador (que vai executar as instruções introduzidas através de uma linguagem de programação) ; ii) o Programa (escrito por “nós) e iii) o Utilizador (que vai usar o programa sem ter que saber como ele é feito).

O nosso objectivo é aprender a programar. Para isso, nunca podemos perder de vista que, ao escrever um programa, o que estamos a escrever são instruções que o computador vai executar mais tarde, quando alguém o mandar executá-las. Também não podemos perder de vista, que o programa é feito para *alguém* (o utilizador) o mandar executar. O utilizador não sabe que instruções tem o programa, apenas se apercebe do que o computador está a fazer, pelos efeitos visíveis dessas acções (p. ex. o que o programa manda o computador escrever no ecrã). Quando estamos a fazer um programa, nunca nos podemos esquecer de que o estamos a fazer para um utilizador ⎯ se o esquecermos, rapidamente seremos insultados por uma data de gente que nem sequer conhecemos... ☺ ⎯.

Resumindo estas duas interacções do programa (com o computador e com o utilizador) através de um exemplo, se quisermos que determinada informação seja dada ao utilizador, temos que colocar no programa instruções que indiquem exactamente ao computador aquilo que queremos que o utilizador veja ⎯ nem o utilizador adivinha o que o programa está a fazer, nem o computador toma a iniciativa de explicar seja o que for ao utilizador: a única coisa que o utilizador vai ver, é o resultado das instruções que o programador colocou no programa! ⎯.

O meio mais frequente de comunicação entre o computador e o utilizador é a *consola*. A consola é composta por um teclado e por um monitor. Os primeiros capítulos pressupõem essa realidade: pedem-se os dados ao utilizador através do teclado e enviam-se-lhe os resultados através do ecrã do monitor.

# Escrever no ecrã

Definição: Para enviar informação ao utilizador (p. ex. o resultado de um cálculo) podemos *escrever* essa informação no ecrã para que o utilizador a veja. Em linguagem algorítmica, a instrução que se usa é precisamente a palavra “escrever” seguida do que se quer escrever no ecrã.

**Por exemplo** para escrever no ecrã “Eu gosto de programar!”, a instrução é:

*escrever “Eu gosto de programar!”*

Assim, ao longo de um programa, sempre que se quiser escrever algo no ecrã, basta acrescentar uma linha com a respectiva instrução. P. ex.

*escrever “Eu gosto de programar!”*

*escrever “Mas vou gostar mais quando souber o que é...”*

Já temos um programa com duas linhas! Agora para *sentir* o que acabámos de fazer, não há nada como codificar numa linguagem de programação e mandar o computador executar. O mais fácil e compreensível, para este primeiro exemplo, é usar o interpretador de *basic* distribuído com o *ms-dos* (o “Qbasic”[[5]](#footnote-5)). Mas o mais prático mesmo, é usar o que estiver disponível no computador em que estivermos!

Um interpretador de QBasic é o mais fácil de usar...

|  |  |
| --- | --- |
| PRINT “Eu gosto de programar!”  PRINT “Mas vou gostar mais...” | ***BASIC*** |

Em Pascal já é preciso definir o início e o fim do programa...

|  |  |
| --- | --- |
| Program bomgosto;  begin  writeln(‘Eu gosto de programar!’);  writeln(‘Mas vou gostar mais...’)  end. | ***PASCAL***  *início do programa*  *normalmente as instruções terminam com ponto-e-vírgula*  *esta não leva o ponto-e-vírgula, por vir um “end” a seguir*  *fim do programa* |

Em C é um bocado mais complicado, mas também se faz[[6]](#footnote-6). Cuidado para não trocar minúsculas com maiúsculas.

|  |  |
| --- | --- |
| *#include* <stdio.h>  main() {  printf(“Eu gosto de programar!\n”);  printf(“Mas vou gostar mais...\n”);  } | ***C***  *para usar o “printf()” temos que incluir esta linha*  *início do programa (o programa é tudo o que ficar entre as chavetas)*  *o “\n” é um caractere especial usado no C, para indicar ao computador que mude de linha (experimente sem o “\n” para ver a diferença)*  *o programa termina com o fechar de chaveta* |

# Fazer Cálculos e usar Variáveis

Se se quiser escrever um número ou o resultado de um cálculo, já não serão precisos os delimitadores de sequência de caracteres (aspas ou pelicas), mas a utilização da instrução *escrever* é semelhante:

*escrever 15*

Escreve um 15 no ecrã. Se se quiser escrever o resultado de uma operação, pode-se usar:

*escrever 3\*7 (no ecrã aparece 21[[7]](#footnote-7))*

Ou ainda, usar funções matemáticas, ou outras, que estejam disponíveis:

*escrever sen(¶/2) (aparece 1 no ecrã)*

Para escrever uma linha composta por vários elementos, basta separá-los por vírgulas:

☝ Se se pusesse *3\*7* entre aspas, no ecrã apareceria *3\*7* em vez de *21*, o que claramente não é a intenção...

*escrever “3 x 7 = ”, 3\*7, “!”*

Para que não haja dúvidas (ou para que apareçam as que possa haver), o melhor é codificar e mandar o computador executar o programa...

|  |  |
| --- | --- |
| PRINT “3 x 7 = ”; 3\*7; “!” | ***BASIC***  *em Basic, o separador é o ponto-e-vírgula* |

Agora que sabemos a estrutura de um programa em Pascal, a codificação também é imediata...

|  |  |
| --- | --- |
| Program vinteeum;  begin  writeln(‘3 x 7 = ’, 3\*7, ‘!’)  end. | ***PASCAL***  *Em Pascal usam-se pelicas em vez de aspas* |

O *printf* do C usa apenas uma *string*. Para incluir valores numéricos marca-se o seu lugar na *string* e só no fim é que coloca a lista dos valores a escrever o ecrã.

|  |  |
| --- | --- |
| *#include* <stdio.h>  main() {  printf(“3 x 7 = %d!\n”, 3\*7);  } | ***C***  *para o “printf()”*  *o “%d” indica o lugar onde vai aparecer um n.º inteiro*[[8]](#footnote-8)  *o programa termina com o fechar de chaveta* |

**Variáveis:** À medida que os cálculos começarem a ser mais complexos, vamos precisar de armazenar valores. Para isso existem as variáveis. As variáveis representam um espaço temporário de armazenamento de valores e são identificadas por nomes (para evitar confusões e para manter coerência com a generalidade das linguagens de programação, os nomes das variáveis são constituídos apenas por letras[[9]](#footnote-9) e dígitos, e começam sempre por uma letra).

Para atribuir um valor a uma variável, em linguagem algorítmica, escreve-se o nome da variável seguido do símbolo de atribuição (‘←’) e depois o valor ou a expressão cujo resultado lhe queremos atribuir.

Tome-se como exemplo o cálculo da superfície de um cilindro (ou, para os mais terra-a-terra, o cálculo aproximado da superfície de uma lata de refrigerante!) com 4cm de raio e 12cm de altura.

*pi ← 3.1415927*  (mete-se já o valor de¶ numa variável)

*raio ← 4* (sempre que possível usam-se variáveis com)

*altura ← 12* (nomes que denunciem o seu conteúdo)

*superficiedabase ← pi \* raio ↑ 2* (a superfície de um círculo é ¶·r²)

*perimetrodabase ← 2 \* pi \* raio* (o perímetro de uma circunferência é 2¶·r)

*superficielateral* ← *perimetrodabase \* altura*

*superficietotal* ← *superficielateral + 2 \**  *perimetrodabase*

*escrever “A superfície de um cilindro com ”, altura, “cm de altura e um raio de “, raio, “cm é “, superficietotal, “cm.”*

**Tipo:** Antes da codificação deste algoritmo interessa introduzir o conceito de *tipo*. Como espaço de armazenamento que é, uma variável implica a reserva de alguma memória. Por esse motivo as linguagens obrigam o programador a **declarar** o *tipo* de dados a que se destina a variável.[[10]](#footnote-10)

Seguem-se algumas codificações deste algoritmo. Os principais aspectos a reter são i) a sintaxe da atribuição, ii) o modo como é feita a declaração do tipo, iii) o exemplo de utilização de uma função para calcular o quadrado do raio (que, se necessário, poderia ser sempre efectuado como “raio x raio”) e iv) a introdução de linhas de comentário nos programas. Além de um cabeçalho que identifique o programa, o autor e a data em que foi feito, é essencial comentar todo o programa, de modo a facilitar a sua leitura mais tarde (nem sempre o que nos parece óbvio hoje, o será amanhã...)

🕮 Mesmo que não tenha dúvidas sobre o algoritmo nem sobre as codificações, é essencial que introduza e execute o programa em pelo menos uma linguagem, para que as dúvidas possam aparecer progressivamente, em vez de aparecerem mais tarde em quantidades mais difíceis de lidar.

|  |
| --- |
| ***BASIC***  ‘Cálculo da Superfície de um Cilindro, versão 1.0  ‘Autor: Miguel Abrunhosa de Brito  ‘Data: 17.Ago.2000  ‘Uma linha começada por uma pelica é um comentário (sem interferência no programa).  PI = 3.1415927  RAIO = 4  ALTURA = 12  ‘Em vez de SQR(RAIO) podia ser RAIO^2 ou RAIO\*RAIO  SUPERFICIEDABASE = PI \* SQR(RAIO)  PERIMETRODABASE = 2 \* PI \* RAIO  SUPERFICIELATERAL = PERIMETRODABASE \* ALTURA  SUPERFICIETOTAL = SUPERFICIELATERAL + 2 \* PERIMETRODABASE  PRINT “A superfície de um cilindro com ”; ALTURA; “cm de altura e um raio de “;  PRINT RAIO; “cm é “; SUPERFICIETOTAL; “cm.” |

|  |
| --- |
| ***PASCAL***  {Cálculo da Superfície de um Cilindro, versão 1.0  Autor: Miguel Abrunhosa de Brito  Data: 17.Ago.2000}  // Tudo o que estiver entre chavetas, ou entre “//” e o fim da linha é apenas um comentário.  Program SuperficieCilindro;  Var // Esta palavra marca o início das declarações das variáveis.  PI, Raio, Altura: Real; // Uma linha de declaração é sempre uma ou mais variáveis  SuperfícieDaBase, PerimetroDaBase: Real; // seguidas de dois pontos e do respectivo tipo.  SuperficieLateral, SuperficieTotal: Real; // Esta divisão em três linhas foi apenas estética!  begin  PI := 3.1415927; // Aninhar blocos de código (pondo uns espaços no início  Raio := 4; // da linha) não é obrigatório, mas torna o programa mais legível.  Altura := 12;  SuperficieDaBase := PI \* Power(Raio, 2); // Ou Raio\*Raio em vez de Power(Raio, 2)  PerimetroDaBase := 2 \* PI \* Raio;  SuperficieLateral := PerimetroDaBase \* Altura;  SuperficieTotal := SuperficieLateral + 2 \* PerimetroDaBase;  Write(‘A superfície de um cilindro com ‘, Altura); // Esta linha foi dividida em três para exempli-  Write(‘cm de altura e um raio de ‘, Raio, ‘cm é ‘); // ficar a utilização do “write” (ao contrário do  Writeln(SuperficieTotal, ‘cm.’) // “writeln”, no fim da linha não passa para a seguinte)  end. |

|  |
| --- |
| ***C***  /\* Cálculo da Superfície de um Cilindro, versão 1.0  Autor: Miguel Abrunhosa de Brito  Data: 17.Ago.2000 \*/  /\* Os blocos de comentários colocam-se entre barra-asterisco e asterisco-barra \*/  /\* Sempre que se usar uma função é necessário verificar se a biblioteca a que pertence consta na  lista de “includes” do programa (juntamente com a sintaxe de uma função, os manuais indicam  sempre a biblioteca a que pertence) \*/  #include <stdio.h> /\* para se poder usar o printf \*/  #include <math.h> /\* biblioteca que disponibiliza a função pow() \*/  main() {  float pi, raio, altura; /\* Uma linha de declaração é sempre o nome de \*/  float superfíciedabase, perimetrodabase; /\* um tipo seguido de uma lista de variáveis. \*/  float superficielateral, superficietotal; /\* Esta divisão em três linhas foi apenas estética! \*/  /\* Não há uma divisão explícita entre as declarações de variáveis e o bloco de instruções \*/  pi = 3.1415927; /\* Aninhar blocos de código (pondo uns espaços no início \*/  raio = 4; /\* da linha) não é obrigatório, mas torna o programa mais legível. \*/  altura = 12;  superficiedabase = pi \* pow(raio, 2); /\* Ou raio\*raio em vez de pow(raio, 2) \*/  perimetrodabase = 2 \* pi \* raio;  superficielateral = perimetrodabase \* altura;  superficietotal = superficielateral + 2 \* perimetrodabase;  printf(“A superfície de um cilindro com %f cm de altura e um raio de %f cm é %f cm.”, altura, raio, superficietotal);  } |

**Operadores (Aritméticos, Lógicos, Relacionais e Atribuição):** segue-se uma relação comparativa de operadores em várias linguagens de programação. Para pormenores ou dúvidas de utilização consultar os manuais respectivos (p. ex. o tipo do resultado da divisão em C, depende do tipo dos operadores, ou seja uma divisão entre reais dá um n.º real, uma divisão entre inteiros dá o quociente da divisão inteira).

| **Operação** | **L.A.[[11]](#footnote-11)** | **Basic** | **Pascal** | **C** |
| --- | --- | --- | --- | --- |
| Adição | + | + | + | + |
| Multiplicação | \* | \* | \* | \* |
| Menos ou subtracção | - | - | - | - |
| Divisão | / | / | / | / |
| Divisão inteira | \ | \ | div | / |
| Resto da divisão inteira | % | MOD | mod | % |
| Potenciação | ↑ | ^ | power[[12]](#footnote-12) | pow |
| Igualdade | = | = | = | == |
| Menor que | < | < | < | < |
| Maior que | > | > | > | > |
| Menor ou igual que | ≤ | <= | <= | <= |
| Maior ou igual que | ≥ | >= | >= | >= |
| Diferente | ≠ | <> | <> | != |
| Negação | ⎤ | NOT | not | ! |
| Conjunção | ∧ | AND | and | && |
| Disjunção | ∨ | OR | or | | | |
| Disjunção Exclusiva | p ∨ q | XOR | ([[13]](#footnote-13)) |  |
| Implicação | ⇒ | IMP | ([[14]](#footnote-14)) |  |
| Atribuição | ← | = | := | = |

**Exercícios:**

1. Elabore um algoritmo para converter 7 horas e 35 minutos, num valor apenas em minutos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um algoritmo que apresente, no ecrã, 5000 minutos num formato com dias, horas e minutos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
3. Elabore um algoritmo para converter e apresentar no ecrã 6 radianos num formato com graus, minutos e segundos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
4. Elabore um algoritmo que calcule a hipotenusa de um triângulo rectângulo com lados de 3 e 4 cm (). Codifique em *Basic*, *Pascal* e *'C'*. Procure nos manuais das linguagens a função necessária ao cálculo da raiz quadrada. Corra os programas num computador.

# Ler do teclado

Definição: Para pedir informação ao utilizador (por exemplo dados necessários para um cálculo) precisamos de uma instrução que interrompa o programa e fique à espera que o utilizador introduza[[15]](#footnote-15) a informação necessária. Em linguagem algorítmica usa-se a instrução *ler* seguida do nome da variável onde se pretende que a informação fique armazenada.

**Por exemplo** vejamos um programa que *gosta* de cumprimentar as pessoas pelo nome ☺.

*escrever “Como te chamas?”*

*ler nome* (o programa fica aqui parado à espera que o utilizador introduza o nome).

*escrever “Olá ”, nome, “!”* (agora a variável *nome* passou a ter um valor que pode ser usado).

Como se pode ver, foi usada uma variável para armazenar uma sequência de caracteres. As operações efectuadas sobre essa variável foram apenas *ler* e *escrever*. No entanto, a manipulação de *strings* tem bastante mais que se lhe diga, nomeadamente quando é necessário concatenar *strings* ou aceder a *substrings*.

|  |
| --- |
| ***C*** Em C a atribuição de uma *string* a uma variável pode ser feita através de uma das funções de manipulação de *strings* disponibilizada pela biblioteca “string”, denominada “strcpy()”. Para o efeito teria que ser feito o respectivo *include*,  #include <string.h>  e a atribuição teria o aspecto  strcpy(nome, “Joaquim”); |

Neste momento fica apenas a chamada de atenção para o facto de que a manipulação de sequências de caracteres só será abordada mais adiante. Inclusivamente fica aqui uma chamada de atenção especial para o facto de que (ao contrário do Basic e do Pascal) o C não permite fazer a atribuição directa de uma *string* a uma variável, como por exemplo,

*nome ← “Joaquim”*

Voltando ao algoritmo apresentado, apresentam-se a seguir as respectivas codificações.

A codificação em Basic é quase literal. Basta ter em conta o sufixo certo para a variável nome...

|  |
| --- |
| ***BASIC***  ‘Cumprimento personalizado, versão 1.0  ‘Autor: Miguel Abrunhosa de Brito  ‘Data: 20.Ago.2000  ‘O ponto-e-vírgula no fim, evita que o cursor passe para a linha seguinte (experimente com e sem)  PRINT “Como te chamas?”;  INPUT NOME$  PRINT “Olá “; NOME$; “!” |

Em Pascal, usa-se o tipo *string*...

|  |
| --- |
| ***PASCAL***  {Cumprimento personalizado, versão 1.0  Autor: Miguel Abrunhosa de Brito  Data: 20.Ago.2000}  Program Cumprimento;  Var  Nome: String;  begin  // Para esclarecer pequenas dúvidas do género “porquê write e não writeln?” não há nada como fazer  // algumas experiências ao teclado do computador!  Write(‘Como te chamas?‘);  Readln(Nome); // [[16]](#footnote-16)  Writeln(‘Olá ’, Nome, ‘!’)  end. |

|  |
| --- |
| ***C***  /\* Cumprimento Personalizado, versão 1.0  Autor: Miguel Abrunhosa de Brito  Data: 20.Ago.2000 \*/  #include <stdio.h> /\* para se poder usar o printf() e o scanf() \*/  main() {  char nome[20]; /\* Define uma *string* com 19 caracteres (20-1) \*/  printf(“Como te chamas? “);  /\* Em C há várias funções de leitura. Nesta fase vamos usar o *scanf()* apesar de sofrer da restrição  de não aceitar espaços (p. ex. em “Rui João”), mas aqui, fica apenas a chamada de atenção \*/  scanf(“%s”, nome);  printf(“Olá %s!”, nome);  } |

|  |
| --- |
| ***C*** Há ainda um aspecto importante a ter em conta na utilização do scanf(), que é a necessidade de preceder os nomes das variáveis pelo símbolo *&*, sempre que a variável não é do tipo *string*. P. ex.,  (...)  int batatas;  (...)  scanf(“%d”, &batatas);  (...)  Na realidade, *&* é um operador que devolve, não o valor da variável, mas o endereço de memória em que ele se encontra. O facto de termos de o usar na função *scanf()* é apenas porque foi assim que a função *scanf()* foi construída, ou seja, em vez de pedir como argumento uma variável, pede o respectivo **apontador**.  No caso da leitura de *strings* (com o *%s*) não se usa o operador *&* porque na realidade não existe o tipo *string* em C! A variável que *contém* a *string* já é um apontador.  Para uma explicação aprofundada dos apontadores em C, aconselha-se a leitura do Anexo XVI. |

Como regra para evitar aborrecimentos desnecessários, aconselha-se vivamente o aluno a usar sempre uma instrução de leitura por cada variável. Apesar de as linguagens permitirem a leitura de uma lista de variáveis na mesma instrução, isto tem dois inconvenientes: i) obriga ao domínio exacto da instrução de leitura e ii) normalmente baralha o utilizador, que tem tendência a terminar a introdução de um valor com “Enter”, o que termina a instrução sem que tenham sido lidas as restantes variáveis.

**Exercícios:**

1. Elabore um algoritmo que peça três números ao utilizador, e apresente o resultado da sua adição. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um algoritmo para converter um valor em horas e minutos, num formato apenas em minutos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
3. Elabore um algoritmo para converter um valor em minutos num formato com dias, horas e minutos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
4. Elabore um algoritmo para converter um valor em radianos num formato com graus, minutos e segundos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
5. Elabore um algoritmo para converter uma temperatura em graus Celsius para Fahrenheit (F=32+1.8C). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
6. Elabore um algoritmo que dados dois lados de um triângulo rectângulo calcule a respectiva hipotenusa ().Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
7. Elabore um algoritmo que dado o preço de um litro de combustível e ainda, os litros gastos e os km percorridos por um automóvel, calcule os gastos de combustível em $/km e em l/100km. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
8. Elabore um algoritmo que converta as coordenadas polares (ρ,θ) de um ponto, em coordenadas cartesianas (x,y); (x=ρcos(θ), y=ρsen(θ)).Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.

# Estrutura Condicional “se-então-senão”

**Definição:** Para permitir a execução de blocos de código diferentes em função de determinada condição (por exemplo um valor introduzido pelo utilizador) é necessário introduzir uma instrução que

*se* condição *então*

{ bloco-se

*senão*

{ bloco-senão

em que a *condição* é uma expressão booleana. Se for verdadeira, é executado o *bloco-se*, se for falsa, é executado o *bloco-senão*.

**Por exemplo** vamos supor que é necessário simular o lançamento de uma moeda ao ar. Se dispusermos de uma função matemática de geração de números pseudo-aleatórios, podemos facilmente derivar daí um valor binário que represente as duas faces de uma moeda. Resumindo precisamos de um programa que converta um número real ∈[0, 1[, numa mensagem para o utilizador de “cara” ou “coroa”.

*Moeda ← Random()* (a tal função matemática que dá um valor entre 0.000000... e 0.999999...)

*se Moeda<0.5 então*

{ *escrever “Cara”*

*senão*

{ *escrever “Coroa”*

**Outro exemplo:** um exemplo cuja compreensão é extremamente relevante na aprendizagem da programação de computadores é o da ordenação de valores. Além de ser uma introdução aos *truques* da resolução de problemas, ilustra bem o facto de que em programação não há uma solução para um problema: é preciso analisar cuidadosamente o enunciado, traçar uma estratégia de resolução e, finalmente, encontrar uma sequência de instruções que implemente essa estratégia. Passe-se então à ordenação de três números por ordem crescente (para este exemplo suponham-se pedidos ao utilizador).

Como estratégia de resolução, vamos testar individualmente as 6 hipóteses possíveis. O único cuidado a ter é o de agrupar os testes, de modo a optimizar o n.º de condições necessárias, evitando uma coisa do género

*ler A*

*ler B*

*ler C*

☝ Testar que A, B e C estão por essa ordem tem que ser feito pela conjunção de duas condições, uma vez que os operadores relacionais só aceitam dois argumentos.

Uma expressão do género A < B < C seria calculada pelo computador em duas partes: i) A < B, o que daria um resultado booleano (Verdadeiro ou Falso); ii) esse resultado seria comparado com C (p. ex. *Verdadeiro < 37*) o que obviamente não faz sentido!

*se A < B ∧ B < C então*

{ *escrever A, B, C*

*se A < C ∧ C < B então*

{ *escrever A, C, B*

*se B < A ∧ A < C então*

{ *escrever B, A, C*

*se B < C ∧ C < A então*

{ *escrever B, C, A*

*se C < A ∧ A < B então*

{ *escrever C, A, B*

*se C < B ∧ B < A então*

{ *escrever C, B, A*

solução que necessita de 12 condições. Se se tiver o cuidado de agrupar os testes comparativos entre valores, pode-se chegar a uma solução como a que se segue, em que apenas são necessárias 6 condições!

*ler A*

*ler B*

*ler C*

*se A < B ∧ A < C então* (Se o valor introduzido em *A* for o menor de todos)

*se B < C então* (O segundo pode ser o B ou o C)

{ *escrever A, B, C*

*senão*

{ *escrever A, C, B*

*senão* (Se o *A* não for o menor de todos...)

*se B < C então* (O menor de todos pode ser o *B* ou o *C*)

*se A < C então* (mais uma vez testam-se os dois restantes)

{ *escrever B, A, C*

*senão*

{ *escrever B, C, A*

*senão* (O menor de todos é o *C*)

*se A < B então* (mais uma vez testam-se os dois restantes)

{ *escrever C, A, B*

*senão*

{ *escrever C, B, A*

Quem resolveu o problema sugerido, apercebeu-se de como a complexidade aumenta pela simples introdução de mais um número a ordenar. Esta complexidade, justifica que se invista algum tempo a procurar uma estratégia melhor para abordar a questão.

🗹 Para cimentar o agrupamento de condições e para melhor compreender o exemplo que se segue, antes de continuar, resolva o mesmo problema, mas para ordenar 4 números por ordem decrescente.

Se em vez de nos limitarmos a comparar os números entre eles, os começarmos a arrumar pela ordem que nos interessa, poupam-se muitos testes à medida que nos aproximamos das últimas hipóteses. Por outras palavras, se em vez de nos limitarmos a apresentar os valores ao utilizador pela ordem correcta, os ordenarmos, de facto, nas variáveis poupamos muito trabalho. Para a compreensão desta estratégia, aconselha-se o fazer de um ou dois exemplos à mão[[17]](#footnote-17).

*ler A*

*ler B*

*se A < B então* (Se o valor introduzido em *A* for o menor que o introduzido em *B*)

*X ← A* (trocam-se os dois, garantindo assim, que o maior fica sempre em *A*)

*A ← B* (Repare-se na utilização de uma variável auxiliar, *X*, de modo a ser pos-

*B ← X*  sível armazenar o valor de *B* em *A*, sem perder o valor original de *A*)

*ler C* (Note-se que o utilizador não se apercebe se os números foram todos

*se B < C então* lidos no início do programa, ou apenas à medida que foram sendo

*X ← B* necessários.)

*B ← C*

*C ← X*

*se A < B então*

*X ← A*

*A ← B*

*B ← X*

*ler D*

*se C < D então*

*X ← C*

*C ← B*

*B ← X*

*se B < C então*

*X ← B*

*B ← C*

*C ← X*

*se A < B então*

*X ← A*

*A ← B*

*B ← X*

*escrever A, B, C, D*

**Exercícios:**

1. Elabore um Algoritmo que peça dois números ao utilizador, e que indique qual dos dois é o maior. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um Algoritmo para converter um valor em horas e minutos, num formato apenas em minutos. Valide os dados de entrada. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
3. Elabore um Algoritmo que peça três números ao utilizador, e que indique qual dos três é o maior. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
4. Elabore um Algoritmo que peça três números ao utilizador, e que os apresente por ordem crescente. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
5. Elabore um Algoritmo que peça dois números ao utilizador, e apresente o resultado da sua divisão (não se esqueça de verificar os casos de indeterminação, e divisão por zero). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
6. Elabore um Algoritmo que simule a extracção de três bolas do totoloto. Não se esqueça que uma bola só pode ser tirada uma vez! ⎯ use a sua imaginação para evitar as bolas repetidas. Em último caso, use-a para dar uma desculpa *credível* ao utilizador... ⎯. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
7. Elabore um Algoritmo que implemente a função potência (xy ⇔ eyln(x), para x≠0). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
8. Elabore um Algoritmo que dada a nota de um aluno, entre 0 e 20, a classifique de *mau*, *medíocre, suficiente, bom* ou *muito bom*, conforme a nota for inferior a 5, 10, 14, 18, ou 20, respectivamente; valide os casos fora de limites. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
9. Elabore um Algoritmo que implemente uma calculadora com as funções de somar, subtrair, multiplicar e dividir. O programa deverá pedir ao utilizador os dois operandos, e perguntar qual a operação pretendida. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
10. Elabore um Algoritmo que dadas três temperaturas tiradas numa cidade ao longo do dia, indique qual a máxima e qual a mínima. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
11. Numa empresa, os funcionários são pagos a 250$/hora. Pretende-se um algoritmo que a partir da leitura do número de horas de laboração de um empregado em cada um dos 5 dias da semana, e tendo em conta que os descontos de 15% para a segurança social e 10% de IRS, calcule os vencimentos bruto e líquido do funcionário, bem como os respectivos descontos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
12. Dada uma sequência de 4 números introduzidos pelo utilizador, elabore um Algoritmo que calcule a média dos valores positivos (ou nulos), ou seja, ignorando os valores negativos. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
13. Elabore um Algoritmo que dado um número, indique se ele é divisível por 3 ou não. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
14. Elabore um Algoritmo que dados dois números, indique se algum deles é múltiplo do outro. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
15. Elabore um Algoritmo que calcule o vencimento de um funcionário, sabendo que este é calculado adicionando a um dado vencimento base, as seguintes parcelas:
16. 1% por cada ano de idade superior a 25 anos.
17. 3% por cada ano de casa.
18. 5% por cada filho ou outro dependente.

Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.

1. Elabore um algoritmo que dado um valor determinado valor em escudos, indique quantas moedas de 100$, 50$, 10$, 5$ e 1$, serão necessárias para perfazer essa quantia (com o mínimo de moedas possível). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um Algoritmo que converta uma classificação de 1 a 5 (inteiros), em "mau", "medíocre", "suficiente", "bom" e "muito bom", respectivamente. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
3. Elabore um Algoritmo que pergunte o nome ao utilizador, e responda aleatoriamente «"Olá ", *nome*, "!"», «"Como está ", *nome*, "?"» ou «"Prazer em vê-lo ", *nome*, "!"». Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
4. Elabore um Algoritmo que dada a nota de um aluno, entre 0.0 e 20.0, indique se está Aprovado (nota ≥ 9.5), Reprovado (nota < 7.5), ou para Exame (7.5 ≥ nota > 9.5); valide os casos fora de limites. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
5. Elabore um Algoritmo que peça quatro números ao utilizador, e que os apresente por ordem decrescente. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.

# Estrutura Condicional “caso-seja”

**Definição:** Há casos de algoritmos em que acontece ser necessária uma bateria de instruções *se-então* em que todas as opções são mutuamente exclusivas. Nesse caso, torna-se mais legível a utilização de uma estrutura *caso-seja* como vamos ver a seguir.

**Como exemplo** tome-se o caso de seleccionar aleatoriamente uma carta de um baralho. Neste caso, torna-se pouco prático estar a dividir o intervalo como fizemos no exemplo do lançamento de uma moeda, no capítulo anterior. O ideal é arranjar uma expressão matemática que converta imediatamente o resultado de Random() num valor inteiro entre 1 e 13 para o valor, e entre 1 e quatro para o naipe...

int(Random()\*13)+1 (*para o valor*)

☝ A quem quiser perceber melhor esta expressão (o que se aconselha) sugere-se a realização manual das operações, para os casos limite (0.000000 e 0.999999) e mais um ou dois valores do meio do intervalo.

int(Random()\*4)+1 (*para o naipe*)

em que *int()* é uma função que dá a parte inteira de um número real.

Então de posse destas expressões, resolva-se o problema normalmente à custa de instruções *se-então*.

*Valor ← int(Random()\*13)+1*

*Naipe ← int(Random()\*4)+1*

*se Valor = 1 então*

{ *escrever “Ás”*

*senão*

*se Valor = 13 então*

{ *escrever “Rei”*

*senão*

*se Valor = 12 então*

{ *escrever “Dama”*

*senão*

*se Valor = 11 então*

{ *escrever “Valete”*

*senão*

{ *escrever Valor*

*escrever “ de ”*

*se Naipe = 1 então*

{ *escrever “Copas”*

*senão*

*se Naipe = 2 então*

{ *escrever “Ouros”*

*senão*

*se Naipe = 3 então*

{ *escrever “Espadas”*

*senão*

{ *escrever “Paus”*

Como já foi dito, repare-se em como as expressões são mutuamente exclusivas, ou seja, de todas as hipóteses possíveis só pode ocorrer uma, em cada execução do programa.

Esta situação é tão típica que, nestes casos, se usa uma apresentação diferente dos alinhamentos, de modo a tornar mais legível o código e a evitar que as linhas eventualmente acabem por ultrapassar a margem direita da folha, à medida que vão sendo *empurradas* para a direita! Num caso destes, a apresentação seria a seguinte:

*Valor ← int(Random()\*13)+1*

*Naipe ← int(Random()\*4)+1*

*se Valor = 1 então*

{ *escrever “Ás”*

🕮 É muito importante que o aluno faça um esforço sério de respeitar os aninhamentos e os alinhamentos que têm sido apresentados, nos exercícios que for efectuando, sob pena de adquirir vícios que potenciarão o aparecimento de erros e dificultarão a legibilidade do código produzido.

*senão se Valor = 13 então*

{ *escrever “Rei”*

*senão se Valor = 12 então*

{ *escrever “Dama”*

*senão se Valor = 11 então*

{ *escrever “Valete”*

*senão*

{ *escrever Valor*

*escrever “ de ”*

*se Naipe = 1 então*

{ *escrever “Copas”*

*senão se Naipe = 2 então*

{ *escrever “Ouros”*

*senão se Naipe = 3 então*

{ *escrever “Espadas”*

*senão*

{ *escrever “Paus”*

Repare-se como esta apresentação torna o código mais legível e mais compacto ao mesmo tempo!

Finalmente, há outra instrução que disponibiliza mais uma apresentação possível para este tipo de situações, desde que os valores envolvidos sejam discretos: a instrução *caso-seja*.

O aspecto geral para uma *expressão* que tenha *N* resultados diferentes que interesse tratar é,

*caso* expressão *seja*

valor1*:*

{ bloco1

valor2*:*

{ bloco2

(...)

valorN*:*

{ blocoN

*OutroCaso:*

{ bloco

Em que o bloco *OutroCaso* é opcional.

Aplicado ao exemplo que está a ser apresentado, resulta no seguinte algoritmo:

*Valor ← int(Random()\*13)+1*

*Naipe ← int(Random()\*4)+1*

*caso Valor seja*

*1:*

{ *escrever “Ás”*

*13:*

{ *escrever “Rei”*

*12:*

{ *escrever “Dama”*

☝ A tradução literal em relação às soluções que usaram o *se-senão* seria usar *OutroCaso* em vez de explicitar a opção *4*.

Como, neste caso, se sabe que de facto o único caso que resta é o *4*, as duas hipóteses são equivalentes. Se houvesse a possibilidade de aparecerem outros valores (situação de erro) as soluções anteriores deveriam explicitar a hipótese *4*; Se se quisesse tratar o erro, por exemplo enviando uma mensagem ao utilizador, usar-se-iam então os blocos *senão* e *OutroCaso*, respectivamente.

*11:*

{ *escrever “Valete”*

*OutroCaso:*

{ *escrever Valor*

*escrever “ de ”*

*caso Naipe seja*

*1:*

{ *escrever “Copas”*

*2:*

{ *escrever “Ouros”*

*3:*

{ *escrever “Espadas”*

*4:*

{ *escrever “Paus”*

**Exercícios:** Dos exercícios propostos no fim do capítulo anterior, indique quais são aqueles em que poderia usar a estrutura *caso-seja*. Resolva-os, usando a estrutura *caso-seja*.

# Estrutura de Controlo de Ciclos “repetir-até”

**Definição:** Para permitir a execução de um bloco de código várias vezes, é necessária uma instrução que permita repetir esse bloco até que determinada condição se verifique. É o caso da instrução

*repetir*

{ bloco-de-instruções

*até* condição

em que as instruções contidas no *bloco-de-instruções* são executadas uma vez até ao fim; nesse momento, é calculada a *condição*: se o resultado for *Falso*, o bloco é executado de novo desde o início; findo o bloco, a *condição* é calculada de novo e repete-se sucessivamente o ciclo até que o resultado da *condição* seja *Verdade*.

**Por exemplo** para repetir o lançamento de um dado 100 vezes, sem uma instrução deste género, teríamos de repetir 100 vezes

*Dado ← int(Random()\*6)+1*

*escrever Dado*

o que seria desagradável mas, apesar de tudo, possível! É claro que o mais prático seria mesmo usar a instrução *repetir-até*. Apenas temos que arranjar maneira de controlar o número de repetições do ciclo. A maneira típica de o fazer, é usar uma variável para contar o número de vezes que o ciclo é executado, incrementando-a de cada vez de o ciclo é executado, até que atinja o valor pretendido. Quando uma variável é usada desta maneira chama-se contador.

☝ A instrução *i ← i +* 1 tem que ser compreendida em duas fases: i) a expressão à direita é calculada usando o valor original de *i*; ii) o resultado é armazenado na variável à esquerda, gerando o novo valor de *i*.

*i ← 0*  (inicializar o contador)

*repetir*

*i ← i + 1* (incrementar o contador)

*Dado ← int(Random()\*6)+1*

*escrever Dado*

*até i = 100* (só pára quando o contador atingir 100)

Como já foi dito, neste caso até era possível resolver este problema sem o recurso a um ciclo. No entanto, se não soubéssemos, na altura em que estamos a fazer o programa, quantas iterações iam ser necessárias, precisaríamos mesmo de um ciclo. É o caso, p. ex., se for o utilizador a definir, quando executa o programa, quantos dados quer que sejam lançados.

☝ Em termos algorítmicos não é relevante, mas quando se codificar o programa, é necessário preceder a instrução de leitura de uma de escrita, que explique ao utilizador o que se pretende que ele introduza através do teclado.

🗹 Resolva o mesmo exercício mas usando o próprio *N* para controlar o ciclo (decrementando-o até zero).

*ler N*

*i ← 0*

*repetir*

*i ← i + 1*

*Dado ← int(Random()\*6)+1*

*escrever Dado*

*até i = N* (pára quando o contador *i* atingir o valor de *N*)

**Outro exemplo** relevante é resolver o mesmo problema, mas tendo como condição de paragem saírem dois números iguais consecutivos (e em que, nesse caso, o número repetido só aparece uma vez).

*Dado ← int(Random()\*6)+1*

*repetir*

*escrever Dado*

🕮 O aluno deve aproveitar estes exemplos resolvidos para praticar a traçagem de algoritmos. A traçagem de uma versão melhorada deste algoritmo está disponível no Anexo I

*DadoAnterior ← Dado*

*Dado ← int(Random()\*6)+1*

*até Dado = DadoAnterior*

🗹 Resolva o mesmo exercício mas usando como condição de paragem saírem três números iguais consecutivos.

**Exercícios:**

1. Elabore um Algoritmo que implemente o jogo "hi-lo". A finalidade do jogo é que o utilizador acerte num número entre 1 e 100, escolhido aleatoriamente pelo programa. A cada palpite do utilizador, o programa deve responder indicando se este é inferior ou superior ao número escondido. Quando o utilizador acertar, o programa deve dar-lhe os parabéns e indicar em quantas jogadas acertou. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um Algoritmo que apresente uma sequência de *N* números aleatórios, em que cada n.º não pode ser igual a nenhum dos dois imediatamente anteriores. *N* deve ser pedido ao utilizador. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
3. Elabore um Algoritmo que dadas as notas de um conjunto de alunos, através de uma sequência terminada por um n.º negativo, calcule os seguintes valores estatísticos: nota máxima; nota mínima; n.º de positivas (≥ 10); percentagem de negativas; média das notas; média das notas positivas. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.

# Estrutura de Controlo de Ciclos “enquanto-fazer”

**Definição:** outra instrução que permite a repetição de blocos de código, é a instrução

*enquanto* condição *fazer*

{ bloco-de-instruções

que repete o *bloco-de-instruções* enquanto *condição* for *Verdade*. Assim que o seu resultado for *Falso*, o programa salta para a primeira instrução a seguir ao fim do respectivo bloco.

Repare-se que a condição de permanência no ciclo é a negação da condição de saída do ciclo usada na instrução *repetir-até*.

Outra **diferença entre os dois ciclos** é o facto de a condição ser testada à cabeça (*enquanto-fazer*) ou no fim do bloco (*repetir-até*). Apesar de no caso geral ser indiferente ⎯ o teste da condição fica sempre entre o fim e o início do bloco ⎯ para a primeira execução a diferença é fundamental: repare-se que, no ciclo *repetir-até*, o *bloco-de-instruções* é sempre executado incondicionalmente uma vez, e só então testada a condição de paragem. Já no caso do *enquanto-fazer*, o *bloco-de-instruções* pode nunca chegar a ser executado, se a condição de entrada for falsa logo à partida.

**Exemplificando** com o cálculo de uma soma de parcelas dadas pelo utilizador, façamos então, um algoritmo que calcule o somatório de uma sequência de números inteiros positivos introduzida pelo utilizador, até que seja introduzido um n.º negativo. Este será interpretado como sinal de paragem e não conta como parcela.

Para o efeito usa-se uma variável onde se vai acumulando o total, à medida que os números vão sendo introduzidos. Quando uma variável é usada desta maneira chama-se-lhe acumulador.

*S ← 0*  (inicializar o acumulador)

*ler N* (pedir o 1.º número ao utilizador)

*enquanto N >= 0 fazer* (repare-se que se o 1.º n.º for negativo, o programa nem sequer entra no ciclo)

*S ← S + N*  (actualizar o acumulador com o valor de *N*)

*ler N* (pedir os restantes números ao utilizador)

*escrever S*

🗹 Resolva o mesmo exercício mas usando um ciclo *repetir-até*.

Repare-se também como os números da sequência não têm que ser todos lidos na mesma instrução.

Com **outro exemplo** pretende-se agora sistematizar e ilustrar melhor o processo típico de construção de algoritmos[[18]](#footnote-18).

Enunciado do problema: Construir um algoritmo para o cálculo do mínimo múltiplo comum de dois números, introduzidos pelo utilizador.

1. Análise do enunciado e compreensão do problema[[19]](#footnote-19).

Neste caso, o enunciado é muito simples e a sua interpretação não oferece dúvidas. De qualquer modo, convém sempre resolver pelo menos um exemplo *à mão* para se ter a questão mais presente, e para dar oportunidade às dúvidas de aparecerem...

Como teste, suponha-se que o utilizador escolheu os números 3 e 4.

Múltiplos de 3: 3, 6, 9, **12**, 15,...

Múltiplos de 4: 4, 8, **12**, 16,...

Menor múltiplo comum: 12

1. Definição da estratégia de resolução do problema.

☝ Fazer um programa, é resolver um problema para o qual se tem que inventar uma solução. O enunciado tem que ser bem analisado, o problema tem que ser estudado, e só então, se pode construir um algoritmo que o resolva. A frase-chave é *não tentem fazer um programa para resolver um problema que não sabem como se resolve*! Muito tempo perdido se pouparia se os alunos, antes de começarem a semear instruções, se perguntassem: “independentemente do facto de querer construir um algoritmo, será que eu sei resolver isto *à mão*?”.

Como sempre, há inúmeras estratégias possíveis para abordar e resolver um problema. Como exercício, idealizemos algumas:

1. Tentar implementar *à letra* a solução manual de ir calculando os múltiplos dos dois números até que coincidam.
2. Optar por uma solução tipo *informática-à-bruta*, e testar um a um todos os inteiros à procura do primeiro que é divisível (a divisão dá resto 0) pelos dois números. Um aperfeiçoamento, seria começar a testar a partir do maior dos dois.
3. Testar apenas os múltiplos de um dos números, verificando se é divisível pelo outro. Uma optimização é testar os múltiplos do maior.

Admitindo que esta última é a estratégia mais eficiente, o passo seguinte é, mais uma vez, arranjar pelo menos um exemplo, que permita ganhar sensibilidade e confiança na estratégia definida. Mais tarde, os exemplos seleccionados vão também servir para testar o programa (seja através de traçagem ou de execução usando o computador).

|  |  |  |
| --- | --- | --- |
| Múltiplos de 4 | Resto da divisão por 3 | É múltiplo de 3? |
| 4 | 1 | não |
| 8 | 2 | não |
| **12** | 0 | sim |

1. Elaborar o algoritmo.

Neste caso o algoritmo resume-se a i) pedir os números ao utilizador, ii) trocá-los de variável se não estiverem pela ordem certa e iii) testar os múltiplos do maior número, até encontrar um que seja também múltiplo do menor.

☝ Como se verá mais adiante, os algoritmos mais complexos não são escritos assim tão sequen­cialmente! No entanto para já fica apenas o reparo, mais adiante voltar-se-á ao assunto.

*ler A* (pedir o 1.º número ao utilizador)

*ler B* (pedir o 2.º número ao utilizador)

*se A < B então* (forçar que o maior esteja em *A*)

*X ← A*

*A ← B*

*B ← X*

*MultiploDeA ← A*

*enquanto MultiploDeA % B ≠ 0 fazer* (o símbolo “%” representa o resto da divisão inteira)

*MultiploDeA ← MultiploDeA + A* (calcular o múltiplo seguinte)

*escrever “O menor múltiplo comum entre ”,A , “ e ”, B, “ é “, MultiploDeA*

1. Codificar o algoritmo.

🕮 Um bom exercício, para quem tem a possibilidade de estudar com um colega, é, no fim de cada um dos pontos descritos, trocarem o que cada um fez, para detecção de eventuais erros.

1. Testar o programa.

Para testar o programa no computador, temos que nos colocar no papel do utilizador. Isso significa executar o programa e fornecer os dados que forem pedidos, verificando se as respostas são as esperadas. Claro que os primeiros testes a fazer são com os exemplos usados na análise do problema, porque desses já sabemos o que se espera...

**Exercícios:**

1. Elabore um Algoritmo que apresente os números de Fibonacci inferiores a um dado n.º. Fn+2 = Fn+1 + Fn, F0=0 e F1=1 (n > 0). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um Algoritmo que dada uma sequência de números inteiros positivos terminada por 0, calcule a percentagem correspondente aos valores superiores a 10 (o 0 não conta para efeitos de cálculo!). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
3. Elabore um Algoritmo que calcule o máximo divisor comum de dois números. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
4. Elabore um Algoritmo que peça uma sequência de 30 números ao utilizador, e que no fim apresente o somatório dos valores aceites. Se o utilizador introduzir dois n.os consecutivos iguais, o programa deve rejeitar o segundo e pedir outro n.º em vez desse. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
5. Elabore um Algoritmo que dadas as notas de um conjunto de alunos, através de uma sequência terminada por um n.º negativo, calcule os seguintes valores estatísticos: nota máxima; nota mínima; n.º de positivas (≥ 10); percentagem de negativas; média das notas; média das notas positivas. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.

# Estrutura de Controlo de Ciclos “para-fazer”

**Definição:** Finalmente, uma última estrutura de controlo de ciclos — o *para-fazer* —. Não faz nada que o *enquanto-fazer* não fizesse já, mas simplifica a construção de ciclos em que já se sabe à partida o n.º de vezes que vai ser executado.

Esta simplificação, deve-se ao facto de não ser necessário inicializar, incrementar e testar uma variável que permita controlar o n.º de vezes que o *bloco-de-instruções* é repetido. A instrução faz isto tudo automaticamente, sendo apenas necessário indicar entre que valores se pretende que a variável de controlo se *movimente* e de quanto deve ser incrementada cada vez que o ciclo é repetido.

A sua sintaxe é

*para* variável-de-controlo ← valor-inicial *até* valor-final *passo* incremento *fazer*

{ bloco-de-instruções

em que no caso de se omitir o *passo*, pressupõem-se incrementos de uma unidade. Como a principal finalidade deste tipo de ciclo é repetir um bloco de instruções um determinado n.º de vezes, a sua utilização mais frequente é mesmo sem o *passo*, e para valores entre 1 e o n.º de vezes que se pretende repetir o ciclo.

**Por exemplo** para repetir o lançamento de um dado, *N* vezes

*ler N* (Perguntar ao utilizador quantos dados quer lançar)

*para i ← 1 até N fazer* (A primeira vez que o programa entra no ciclo *i[[20]](#footnote-20)* tem o valor)

*Dado ← int(Random()\*6)+1* (1, da 2.ª vez 2, e assim sucessivamente até que entra uma)

*escrever Dado* (última vez com o valor *N*)

O facto de a variável estar a controlar o ciclo não impede que possa ser utilizada. Não se pode (não se deve) é *mexer* no seu valor. Mesmo que nada o impeça, considera-se que não é *boa programação*. Como exemplo de utilização da variável de controlo, suponha-se que, além do valor do dado, se quer indicar o n.º do lançamento. Se já existe uma variável a contar os lançamentos, não é necessário usar outra!

☝ Nunca mexer na variável de controlo. Se sentir uma necessidade absoluta de mexer na variável de controlo é porque está a usar o tipo errado de ciclo!

*ler N*

*para i ← 1 até N fazer*

*Dado ← int(Random()\*6)+1*

*escrever “Lançamento n.º ”, i, “: “, Dado*

pode-se inclusivamente usar a variável de controlo em expressões, como, por exemplo para calcular os primeiros 10 múltiplos de 3:

*para i ← 1 até 10 fazer*

*{ escrever 3\*i*

ou simplesmente usar o *passo* para que a variável tome valores que, por qualquer motivo, sejam preferíveis:

*para i ← 1 até 30 passo 3 fazer*

🗹 Implemente este exemplo com um ciclo *enquanto-fazer* (além da condição de paragem do ciclo, tem que explicitar as instruções de inicialização e incremento da variável de controlo *i*).

*{ escrever i*

**Exercícios:**

1. Elabore um Algoritmo que simule a extracção de 6 números do Totoloto (os n.os podem sair repetidos). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um Algoritmo que Determine o máximo de uma sequência de 10 números. Codifique em *'C'*. ➂
3. Elabore um Algoritmo que calcule os múltiplos de 3, compreendidos no intervalo entre 6 e um dado limite superior. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
4. Elabore um Algoritmo que calcule o Factorial de um n.º. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
5. Elabore um Algoritmo que apresente os primeiros *N* números de Fibonacci (*N* fornecido pelo utilizador). Fn+2 = Fn+1 + Fn, F0=0 e F1=1 (n > 0). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
6. Elabore um Algoritmo que calcule os primeiros M múltiplos de um dado n.º N, sendo M e N fornecidos pelo utilizador. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
7. Elabore um Algoritmo que calcule o somatório de uma sequência de *N* números. *N* também é pedido ao utilizador. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
8. Elabore um Algoritmo que calcule os divisores de um n.º (o resto da divisão do n.º por um seu divisor é 0). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
9. Elabore um Algoritmo que calcule a média de uma sequência de números. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
10. Elabore um Algoritmo que dada a equação de uma recta na forma Y = mX + b, calcule os valores de *Y* para valores de *X* entre 0 e 50, de 5 em 5. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
11. Elabore um Algoritmo que depois de ler uma sequência de *N* números (*N* também pedido ao utilizador), apresente os seguintes resultados: máximo, mínimo, somatório, a quantidade de números superiores a 10, a percentagem de valores superiores a 10, a média, a média dos valores superiores a 10. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.

# *Salada* de Ciclos

**Definição:** “Salada de ciclos” não é uma expressão técnica! É apenas um nome resumido para este capítulo, em que vai ser abordada a “utilização, no mesmo algoritmo, das três estruturas de controlo de ciclo estudadas e, nomeadamente, aninhadas umas nas outras”.

Mais uma vez, vamos recorrer a um exemplo para explicar a construção do algoritmo. Como em todos os algoritmos que são apresentados para ilustrar a resolução de um problema, é muito importante que o aluno perceba que o que tem que extrair da explicação é o raciocínio e não a solução que é apresentada. Soluções há muitas, e problemas para resolver também! O objectivo não é saber as soluções dos problemas apresentados (que são apenas exemplos). O objectivo é aprender a resolver qualquer problema que seja apresentado. Ora a única maneira de aprender a resolver problemas é resolvendo problemas! Daí, que nunca seja de mais insistir na necessidade de resolver problemas de que ainda não se tenha visto nenhuma solução.

🕮 Nunca caia no erro de julgar que saberia resolver um problema, só porque percebeu a solução que alguém encontrou. Os únicos problemas que pode achar que sabe resolver são aqueles que já consegui resolver (sozinho).

**Exemplo:** Gerar aleatoriamente seis números de totoloto. O programa deve gerar chaves até que o utilizador diga que já não quer mais. Os números que compõem uma chave não podem ser repetidos.

Análise: Vamos simular no *papel* o que pretendemos que seja o efeito do programa em termos de interacção com o utilizador (em itálico o que o utilizador introduz através do teclado).

Chave 1: 23 12 7 48 2 35

Deseja continuar (S/N)? *S*

Chave 2: 12 5 8 27 35 10

Deseja continuar (S/N)? *S*

Chave 3: 1 45 48 17 13 25

Deseja continuar (S/N)? *N*

Boa sorte!

Além do aspecto essencial que é definir com precisão o que pretendemos que o programa faça, a análise também ajuda a encontrar pormenores que podem não estar explícitos no enunciado do problema. P. ex. mesmo que o enunciado não especificasse que numa chave os números não podem aparecer repetidos, haveria uma grande probabilidade de nos termos apercebido disso ao construir o exemplo.

Estratégia: A solução deste problema seria trivial se não fosse o facto de não poderem sair números repetidos. Para se impedir que volte a aparecer, na mesma chave, um número que já saiu, vai ser necessário ter uma variável para cada um dos números que já saíram e, à medida que vão sendo gerados testá-los com todos os anteriores. Sempre que o número gerado já tiver saído, gera-se outro para o substituir.

À primeira vista isto implica muitas linhas de código, em que cada *bola* é gerada numa linha de código diferente, de modo a ser armazenada numa variável distinta das anteriores, e testada contra as que já saíram. No entanto, pensando mais um bocado, consegue-se arranjar uma estratégia de manipulação das variáveis de modo a generalizar o processo e permitir a utilização de um ciclo que repita seis vezes (são seis bolas) o mesmo bocado de código.

Para conseguir repetir o mesmo bocado de código para as seis bolas, tem que se armazenar a bola gerada sempre na mesma variável! Para que a *bola* anterior não se perca quando é gerada uma nova *bola*, tem que se copiar todos os números que já saíram para outras variáveis. E quando a estratégia começa a implicar muito *movimento* entre variáveis, não há nada como representar as variáveis com quadrados e fazer uma representação gráfica do processo...

Vamos arbitrar 6 variáveis, e inicializá-las com zeros.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| B1 | B2 | B3 | B4 | B5 | B6 |

O *truque* vai consistir em armazenar o último número gerado sempre na variável B6, e antes de gerar o seguinte, *correr* os números que já saíram todos para a esquerda, ou seja, B1←B2, B2←B3, B3←B4, B4←B5 e B5←B6. Exemplificando com a chave 1 que usámos na fase de análise do problema (para já sem nos preocuparmos com a questão dos repetidos),

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| B1 | B2 | B3 | B4 | B5 | B6 |
| 0 | 0 | 0 | 0 | 0 | 23 |
| 0 | 0 | 0 | 0 | 23 | 12 |
| 0 | 0 | 0 | 23 | 12 | 7 |
| 0 | 0 | 23 | 12 | 7 | 48 |
| 0 | 23 | 12 | 7 | 48 | 2 |
| 23 | 12 | 7 | 48 | 2 | 35 |

Em que cada linha representa o estado final de uma iteração do ciclo que pretendemos construir.

Finalmente, é necessário ter ainda em conta a estratégia de construção do próprio algoritmo. Sucintamente, a estratégia de construção de um algoritmo pode ser definida como subdividir o problema até que cada parte tenha uma dimensão que seja de construção imediata (o que obviamente depende de quem o está a desenvolver). Neste caso, podemos dividir o problema em três:

1. Desenhar um ciclo que gere seis números entre 1 e 49. Faz a cópia entre variáveis conforme foi definido, gera o novo número e armazena-o em *B6*.
2. Dentro desse ciclo, introduzir um teste que obrigue a repetir a geração do último número, se for repetido.
3. Colocar tudo dentro de um ciclo que repita todo o processo até que o utilizador responda que já não quer mais chaves.

Algoritmo: Façamos então a primeira parte, de geração de uma chave sem nos preocuparmos com os repetidos.

*B2 ← 0* (*B1* não precisa de ser inicializada, porque não é usada no lado direito de nenhuma atribuição)

*B3 ← 0*

*B4 ← 0*

*B5 ← 0*

*B6 ← 0*

*Para i ← 1 até 6 fazer*

*B1 ← B2*

*B2 ← B3*

*B3 ← B4*

*B4 ← B5*

🗹 Resolva o mesmo exercício mas de modo que os números da chave apareçam ordenados.

*B5 ← B6*

*B6 ← int(Random()\*49)+1*

*escrever “Chave “, Chave, “: “, B1, “ “, B2, “ “, B3, “ “, B4, “ “, B5, “ “, B6*

Concluída a primeira parte, já se pode passar à segunda, que visa testar e eliminar números repetidos. Para isso pode-se encapsular a atribuição do valor à variável *B6* num ciclo *repetir-até* que obrigue o programa a gerar números até que o número gerado seja diferente de todos os outros.

*B2 ← 0*

*B3 ← 0*

*B4 ← 0*

*B5 ← 0*

*B6 ← int(Random()\*49)+1* (como o primeiro número não precisa de ser comparado com os)

*para i ← 1 até 5 fazer* (anteriores, pode-se *poupar* uma iteração do ciclo, se se inicializar)

*B1 ← B2* (*B6* logo com o valor da primeira bola)

*B2 ← B3*

*B3 ← B4*

*B4 ← B5*

*B5 ← B6*

*repetir*

{ *B6 ← int(Random()\*49)+1*

*até B6≠B1 ∧ B6≠B2 ∧ B6≠B3 ∧ B6≠B4 ∧ B6≠B5*

*escrever “Chave “, Chave, “: “, B1, “ “, B2, “ “, B3, “ “, B4, “ “, B5, “ “, B6*

Agora que o problema de gerar uma chave está resolvido, basta encapsular este código dentro de um ciclo que o repita enquanto o utilizador quiser.

*repetir*

*B2 ← 0*

*B3 ← 0*

*B4 ← 0*

*B5 ← 0*

*B6 ← int(Random()\*49)+1*

*para i ← 1 até 5 fazer*

*B1 ← B2*

*B2 ← B3*

*B3 ← B4*

*B4 ← B5*

*B5 ← B6*

*repetir*

{ *B6 ← int(Random()\*49)+1*

*até B6≠B1 ∧ B6≠B2 ∧ B6≠B3 ∧ B6≠B4 ∧ B6≠B5*

*escrever “Chave “, Chave, “: “, B1, “ “, B2, “ “, B3, “ “, B4, “ “, B5, “ “, B6*

*escrever “Deseja continuar (S/N)?”* (Esta linha só é necessária na fase de codificação,)

*ler Resposta* (mas também não tem contra-indicações!)

*até Resposta ≠ “S”*

*escrever “Boa sorte!”*

**Exercícios:**

1. Elabore um Algoritmo que simule o lançamento de séries de 5 dados; no fim de cada série, deve perguntar ao utilizador se quer outra série, e proceder de acordo com a resposta. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
2. Elabore um Algoritmo que apresente todos factoriais que cujo resultado seja inferior a um dado valor. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
3. Elabore um Algoritmo que apresente por ordem decrescente, os factoriais desde um dado valor, até ao factorial de 1. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
4. Elabore um Algoritmo que calcule todos os números perfeitos inferiores a um dado valor (um n.º perfeito é aquele que é igual à soma dos seus divisores, excluindo ele próprio). Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador.
5. Elabore um Algoritmo que calcule o sub-factorial de n. Codifique em *Basic*, *Pascal* e *'C'*. Corra os programas num computador. 

# Sub-rotinas

**Definição:** uma sub-rotina é um bloco de instruções colocado à parte e identificado por um nome. A primeira vantagem óbvia em dar um nome a um bloco de instruções é poder chamá-lo em vários locais diferentes ao longo do programa, evitando a duplicação de código. Além disso, permite melhorar a legibilidade do programa e facilitar a sua manutenção.

A seguir são apresentados alguns conceitos associados ao uso de sub-rotinas.

**Variáveis locais**, que apenas são reconhecidas dentro de uma sub-rotina.

**Variáveis globais**, que são reconhecidas em todo o programa.

**Argumentos; passagem de parâmetros** **por valor e por referência**.

**Procedimentos e Funções**.

# Sequências de caracteres (*strings*)

**Exercícios:**

1. Procurar um caractere numa dada *string* sem recorrer às funções de "string.h". Codifique e corra o programa num computador.
2. Procurar uma sequência de dois caracteres numa dada *string* sem recorrer às funções de "string.h". Codifique e corra o programa num computador.
3. Dadas duas strings, ordená-las alfabeticamente sem recorrer às funções de "string.h". Codifique e corra o programa num computador.
4. Elabore um Algoritmo que pergunte o nome ao utilizador, e responda aleatoriamente com uma de uma série de saudações que tenha em memória (num vector) terminando com o respectivo nome. P. ex.: «"Olá ", *nome*, "!"», «"Como está ", *nome*, "?"» ou «"Prazer em vê-lo ", *nome*, "!"». Arranje uma solução para o facto de a expressão poder ser uma exclamação ou uma questão. Codifique e corra o programa num computador.
5. Numerar os parágrafos de um texto (terminado por EOF). Codifique e corra o programa num computador.
6. Procurar uma sequência de caracteres numa linha de texto sem recorrer às funções de "string.h". Codifique e corra o programa num computador.
7. Procurar uma palavra numa linha de texto sem recorrer às funções de "string.h". Codifique e corra o programa num computador.
8. Dada uma sequência de nomes de pessoas, eliminar todos os "de" "do" "da" e "e" dos nomes. Codifique e corra o programa num computador.
9. Dada uma sequência de nomes de pessoas, converter todas as palavras em minúsculas, e em maiúsculas todas as primeiras letras de cada palavra. Codifique e corra o programa num computador.
10. Converter um n.º numa dada base de numeração à escolha do utilizador, convertê-lo para qualquer outra base também à escolha do utilizador. Codifique e corra o programa num computador.
11. Dada uma sequência de nomes de pessoas, apresentá-los ordenados por ordem alfabética. Codifique e corra o programa num computador.
12. Considerando que um *tab* corresponde a tabulações de oito caracteres, substituir todos os espaços que for possível por *tabs*. Codifique e corra o programa num computador.

# Vectores unidimensionais (*Arrays*)

Definição: Sequência de variáveis de um mesmo tipo, agregadas sob um único nome e identificadas por um índice.

Exemplo: Simulação do lançamento de bolas do totoloto sem repetidos.

**Exercícios:**

1. Elabore um Algoritmo que leia uma sequência de números, e os mostre por ordem inversa. Codifique e corra o programa num computador.
2. Elabore um Algoritmo que dada uma sequência de números, indique qual a percentagem que cada um representa em relação ao total. Codifique e corra o programa num computador.
3. Elabore um Algoritmo que dado um conjunto de temperaturas de cidades portuguesas, identificadas por um n.º de ordem, indique quais as que têm temperatura superior à média do país. Codifique e corra o programa num computador.
4. Elabore um Algoritmo que dado um conjunto de temperaturas de cidades portuguesas, identificadas por um n.º de ordem, indique quais as que têm temperatura superior à média do país mas ordenadas por ordem decrescente de temperatura. Codifique e corra o programa num computador.
5. Elabore um Algoritmo que simule a extracção dos 6 números do totoloto, garantindo que não há repetidos. Codifique e corra o programa num computador.
6. Elabore um Algoritmo que dadas as notas de um conjunto de alunos, apresente o n.º de ocorrências de cada uma das notas; em vez de apresentar o n.º de ocorrências de todas as notas entre 0 e 20 apresente apenas as que se situem entre a nota mínima e a nota máxima. No fim, apresente os seguintes valores estatísticos: nota máxima; nota mínima; n.º de negativas (< 10); percentagem de positivas; média das notas; média das notas positivas; n.º de notas superiores à média. Codifique e corra o programa num computador.
7. Elabore um algoritmo que dada uma sequência de nomes, os apresente por ordem alfabética. Codifique e corra o programa num computador.
8. Elabore um algoritmo que permita ao utilizador fazer a manutenção dos dados referentes aos artigos que tem em "stock". O programa deverá apresentar ao utilizador um *menu* no qual este poderá escolher a opção que pretende. As opções disponíveis, deverão ser as de inserir, alterar, remover e consultar um artigo, e ainda listar todos os artigos existentes, cujo stock seja superior a um dado valor; no fim da listagem, deverá ser apresentado o valor total do "stock" (Σ(preço \* quantidade)) correspondente aos artigos listados e o correspondente à totalidade dos artigos. Cada artigo é identificado por um código, que permite aceder à quantidade existente e ao valor unitário. Codifique e corra o programa num computador.
9. Dada uma sequência de números, determine a percentagem daqueles que são: superiores à média dos elementos que a constituem; inferiores ao antepenúltimo elemento da sequência. Codifique e corra o programa num computador.
10. Elabore um algoritmo que baseando-se nos valores de produção mensal de uma fábrica de automóveis num dado ano, calcule e apresente: os valores mensais acumulados de produção ao longo do ano; o mês em que foi atingida metade da produção anual. Codifique e corra o programa num computador.
11. Elabore um Algoritmo que permita ao utilizador jogar “Master Mind”; o programa deve gerar aleatoriamente a combinação a descobrir, e depois responder a cada *palpite* do utilizador, indicando o n.º de elementos certos na casa certa, e o n.º de elementos certos na casa errada. O número de elementos e de *cores* a usar, deve ser pedido ao utilizador no início do programa (os valores típicos são 4 e 6 respectivamente). O jogador deve ter a possibilidade de desistir! Quando o utilizador acertar na combinação gerada, o programa deve indicar o n.º de jogadas em que o fez. Codifique e corra o programa num computador.
12. Considere uma repartição pública de requisição de instalação de serviços. O atendimento está dividido em três *guichets* cujas filas de espera, por limitações físicas do próprio edifício, estão bastante distantes umas das outras. Cada pessoa que entra no edifício retira uma senha de atendimento e segue para a fila de um dos três *guichets* disponíveis. Dada a distância física entre os *guichets* não é possível respeitar a ordem de atendimento em relação à de chegada. No entanto é possível, através do n.º da senha que se tira à entrada, respeitar essa ordem para efeitos de instalação do serviço. Assim, pretende-se construir um programa em que, dados três pares de *arrays* (vectores) cuja informação representa a pessoa (pelo B.I.) e o n.º da senha, fazer a junção dos três vectores num único *array* de B.I.s ordenado pelo n.º de senha. As senhas não aparecem todas, porque normalmente há desistências. Como cada uma das três filas já está ordenada, basta ir retirando de cada uma o elemento com n.º de senha mais baixo e colocar o respectivo n.º de B.I. no *array* de requisição de serviços. Codifique e corra o programa num computador.
13. Elabore um algoritmo que permita registar as notas dos alunos de *Engenharia Electrónica Industrial* à disciplina de *Programação de Computadores I*. O programa deve ter informação sobre o número, nome, e notas dos 3 exames. Deve permitir ao utilizador introduzir, consultar e alterar as notas de um aluno, ter uma opção para alterar as notas de todos os alunos existentes, e ainda tirar listagens por ordem de número de aluno e por ordem alfabética, conforme pedido do utilizador; para cada aluno, as listagens deverão indicar se está aprovado ou não, sabendo que para isso deverá ter uma das três notas > 10. Codifique e corra o programa num computador.
14. Elabore um algoritmo que pergunte o nome ao utilizador, e responda aleatoriamente «"Olá ", *nome*, "!"», «"Como está ", *nome*, "?"» ou «"Prazer em vê-lo ", *nome*, "!"»; depois de cumprimentar o utilizador, deve pedir outro nome, e assim sucessivamente até que a resposta seja “ninguém”! Se o nome introduzido for igual a um dos últimos 10 nomes anteriormente introduzidos, em vez de cumprimentar, responder «”Outra vez, “, nome, “?”». Codifique e corra o programa num computador.
15. Converter uma data no formato "*dd*-*mm*-*aaaa*" no formato "*dd* de *mês* de *aaaa*". P. ex.: "01-03-1998" => "01 de Março de 1998". Codifique e corra o programa num computador.
16. Converter um valor numérico no seu extenso. P. ex.: 1 234 => "mil duzentos e trinta e quatro". Codifique e corra o programa num computador.
17. Converter uma data no formato "*dd*-*mm*-*aaaa*" no formato "*dia* de *mês* de *ano*". P. ex.: "01-03-1998" => "um de Março de mil novecentos e noventa e oito". Codifique e corra o programa num computador.

# Vectores multidimensionais (Matrizes)

Definição: Em vez de uma sequência de variáveis (de um mesmo tipo) agregadas, nos vectores multidimensionais, os seus elementos distribuem­‑se ao longo de várias *dimensões*. Para identificar os seus elementos, utiliza­‑se a seguir ao nome da matriz, um índice para cada dimensão.

☝ Por convenção, quando um vector tem representação gráfica, representa­‑se primeiro a linha, depois a coluna e finalmente a profundidade. No entanto, desde que manipulado coerentemente, é absolutamente irrelevante a eventual representação gráfica que possamos fazer de um vector.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M | 1 | 2 | 3 | 4 |
| 1 | 23 | 64 | 0 | ­‑4 |
| 2 | ­‑76 | 123 | 64 | ­‑12 |
| 3 | 45 | 0 | 64 | 12 |

Atribuição: à semelhança dos vectores unidimensionais, podemos manipular cada elemento de uma matriz do mesmo modo que o fazemos com uma qualquer outra variável do tipo desse elemento. Por exemplo, para atribuir o valor ­–12 à célula (2,4) da matriz **M**:

*M[2,4]← ­‑12*

Percorrer todos os elementos de uma matriz: Em vez de uma sequência simples de variáveis (de um mesmo tipo) agregadas, nos vectores multidimensionais, os seus elementos espalham­‑se ao longo das diversas dimensões. Para os percorrer a todos, a maneira mais simples é encadear sucessivos ciclos aninhados, um por cada dimensão. Por exemplo, percorrer todos os elementos da matriz **M**, de modo a pedir o seu conteúdo ao utilizador:

☝ À semelhança dos vectores unidimensionais, os índices (aqui apresentados a contar a partir de *1*) podem, conforme a linguagem de codificação, começar em *0* (caso do 'C'), noutro inteiro qualquer, ou simplesmente adoptar a sequência de um tipo ordinal que não os números inteiros positivos.

*para i ← 1 até 3 fazer*

*para j ← 1 até 4 fazer*

*ler M[i, j]*

Se o vector tivesse quatro dimensões (p. ex. 3, 4, 4, 2) em vez de duas (não tem que ter equivalente físico...):

*para i ← 1 até 3 fazer*

*para j ← 1 até 4 fazer*

*para k ← 1 até 4 fazer*

*para l ← 1 até 2 fazer*

*ler M[i, j, k, l]*

Exemplo 1: Uma função para procurar um valor numa matriz bidimensional. Devolve o número de ocorrências do valor na matriz. Escreve no ecrã as coordenadas dos valores encontrados.

*função ProcuraValor(Valor, Matriz, NumLinhas, NumColunas)*

*c ← 0*

*para i ← 1 até NumLinhas fazer*

*para j ← 1 até NumColunas fazer*

*se Matriz[i, j]=Valor então*

*c ← c + 1*

*escrever i, j*

🗹 Resolva o mesmo exercício mas para um vector pentadimensional.

*ProcuraValor ← c*

**Exercícios:**

1. Dada uma matriz de inteiros e um valor pedido ao utilizador, **procurá-lo** na matriz e indicar a primeira posição onde for encontrado. Codifique e corra o programa num computador.
2. Calcular a **soma** de duas matrizes dadas. Codifique e corra o programa num computador.
3. Calcular a **transposta** de uma matriz. Codifique e corra o programa num computador.
4. Calcular a matriz resultante da **multiplicação** de duas matrizes dadas. Codifique e corra o programa num computador.
5. Considere uma matriz A de 7 por 5. Desenvolva um algoritmo que: i) Inicialize a matriz com valores aleatórios entre 1 e 100; ii) Identifique qual o maior elemento da matriz, e indique a sua posição (apenas uma mesmo que haja vários valores iguais); iii) Efectue as trocas de linhas e colunas necessárias por forma a que aquele elemento ocupe o centro da matriz. Codifique e corra o programa num computador.
6. **Rodar uma matriz** quadrada de dimensão definida pelo utilizador, 90º para a direita, sem usar matrizes auxiliares ou seja, rodar a matriz através de trocas directa entre os seus elementos. Codifique e corra o programa num computador.
7. Dada uma matriz de inteiros com pentadimensional e um valor pedido ao utilizador, **procurá-lo** na matriz e indicar as posições onde for encontrado. Codifique e corra o programa num computador.
8. Considere uma matriz que regista as temperaturas obtidas num determinado instante, num determinado espaço tridimensional. Desenvolva um algoritmo que determine quais os pontos interiores em que a temperatura é idêntica à média dos 26 que com ele confinam. Codifique e corra o programa num computador.
9. Considere uma matriz A de 8 por 8. Desenvolva um algoritmo que: i) Inicialize a matriz com valores aleatórios entre 2 e 9, com excepção da última linha e da última coluna; ii) Preencha as últimas linha e coluna, sabendo que os seus elementos resultam da soma dos restante elementos da mesma coluna ou da mesma linha, respectivamente; iii) O último elemento da matriz resulta da soma dos elementos da diagonal principal. Codifique e corra o programa num computador.
10. Faça o algoritmo de uma função para procurar um valor numa matriz bidimensional. Devolve o número de ocorrências do valor, na matriz. Escreve no ecrã as coordenadas da primeira ocorrência encontrada. Codifique e corra o programa num computador.
11. Faça o algoritmo de uma função para procurar um valor numa matriz bidimensional a partir de uma dada coordenada até ao fim. Devolve as coordenadas da primeira ocorrência encontrada. Codifique e corra o programa num computador.
12. Implementar o jogo da **batalha naval** mas apenas com submarinos! A dimensão do tabuleiro e o n.º de submarinos em jogo é definido pelo utilizador no início do jogo. Computador e utilizador devem alternar entre si sequências de três tiros, até que um dos dois acerte em todos os submarinos do adversário. Codifique e corra o programa num computador.
13. Elabore um algoritmo que permita ao utilizador jogar "batalha naval" com o computador, mas apenas com submarinos! O programa deverá escolher aleatoriamente a posição dos seus *submarinos* tendo o cuidado de não os colocar em *casas* adjacentes. O primeiro a jogar (utilizador ou computador) deverá ser escolhido aleatoriamente. Computador e utilizador deverão dar como palpites, séries de três *tiros*, até que um dos dois *afunde* a esquadra inimiga. Os palpites do computador também deverão ter em conta a impossibilidade dos submarinos serem colocados em *casas* adjacentes. Codifique e corra o programa num computador.
14. Considere uma matriz onde são registados os espectadores dos jogos de campeonato de futebol da primeira divisão, dividindo-os em sócios, não sócios, crianças , convidados, imprensa e policias. Considere um total de 16 estádios e 30 jornadas. Desenvolva um algoritmo que determine e indique: i) Em que jornada se verificou um número maior de espectadores; ii) Para que estádio, ao longo do ano, a percentagem de polícias, em relação ao numero total de espectadores, foi superior; iii) Em que jogos não estiveram crianças. Codifique e corra o programa num computador.
15. Implementar o **jogo do galo**. O utilizador joga contra o computador. O programa deve ser suficientemente "inteligente" para nunca perder o jogo. Obviamente, deverá detectar automaticamente o fim do jogo (vitória ou empate). Codifique e corra o programa num computador.

# Tuplos (Registos)

Definição: Estrutura de dados que permite agrupar um conjunto de variáveis de tipos diferentes. Estas variáveis designam­‑se por "campos". A estrutura é identificada por um nome idêntico aos usados para as variáveis simples. Um campo também é identificado por um nome, que se acrescenta ao nome da estrutura; os dois nomes são separados por um ponto.

Atribuição: A atribuição pode ser feita a um campo do mesmo modo que se faz com qualquer outra variável, ou directamente entre dois tuplos do mesmo tipo. Por exemplo, se definirmos um tipo "Nota" como sendo um tuplo com os campos como a seguir se indicam,

*Tuplo Nota*

*Num: inteiro*

*Aluno: string*

*Valor: real*

e considerarmos duas variáveis *A* e *B* do tipo *tuplo Nota*, podemos proceder a operações de atribuição como estas:

*A.Num ← 33256*

*A.Aluno ← "João Silva Mendes"*

*A.Valor ← 17,6*

*B ← A*

|  |  |
| --- | --- |
| *#include* <stdio.h>  *#include* <string.h>  main() {  *struct* nota{  *int* num;  *char* aluno[50];  *float* valor;  };  *struct* nota A, B;  A.num = 33256;  strcpy(A.aluno, "João Silva Mendes");  A.valor =17.6;  B = A;  } /\* fim de main() \*/ | ***C***  *para "strcpy()"*  *define o tipo "nota"*  *declara as variáveis A e B como sendo do tipo "struct nota"* |

Exercícios:

1. Dado um vector em que cada elemento é uma estrutura composta por número e nome de um aluno, elaborar um algoritmo que dado o número, indique o respectivo nome. Codifique e corra o programa num computador.
2. Ler três vectores, respectivamente com a informação referente ao número, nome e nota dos alunos de uma turma. Depois de carregados os três vectores, copiar toda a informação para um único vector em que cada registo é compostos por uma estrutura com os campos *número*, *nome* e *nota*. Codifique e corra o programa num computador.
3. Calcular a área e o comprimento da diagonal de um rectângulo dado. O rectângulo deve ser definido através de uma estrutura composta por dois pontos. Por sua vez, cada ponto é definido através de uma estrutura composta por duas coordenadas *x* e *y*. Codifique e corra o programa num computador.
4. Fazer um algoritmo para a inserção ordenada (por número) de um vector de estruturas do tipo {número, nome, nota} para armazenar as notas finais dos alunos da disciplina de programação de computadores. A procura do local de inserção deve ser feita através de uma pesquisa binária (divisão sucessiva do vector a meio). Codifique e corra o programa num computador.
5. Aceitar reservas para uma sala de espectáculos, atribuindo automaticamente os respectivos lugares (dispostos numa matriz de *F* Filas por *C* Cadeiras). O cliente pode pedir lugares para a parte da frente ou de trás da sala. No fim imprimir um mapa da ocupação da sala e uma lista das reservas, ordenada pelo último nome do cliente, e com a indicação se está pago ou não. Refine o algoritmo de modo a aceitar desistências e a evitar que os lugares de uma reserva fiquem separados (seja intercalados com outros, seja em filas diferentes). Codifique e corra o programa num computador.
6. Fazer um algoritmo que deduza um resultado através de sucessivas perguntas de resposta tipo Sim/Não a que o utilizador vai respondendo. No fim, se não acertar, deve pedir ao utilizador a resposta correcta e a pergunta que deveria ter feito para conseguir distinguir entre a resposta que deu e a que deveria ter dado. Na vez seguinte, o programa já "aprendeu" a dar mais uma resposta correcta! Sugestão: usar um vector em que cada elemento é uma estrutura composta por: i) uma *string* que contém uma pergunta ou resposta, ii) um valor que *aponta* para a pergunta seguinte a fazer no caso da resposta do utilizador ser afirmativa, e iii) um valor que *aponta* para a pergunta seguinte a fazer no caso da resposta ser negativa; o programa pode saber se está a fazer uma pergunta ou se já está a dar a resposta pelo valor dos *apontadores* (p. ex. se forem 0  já não há mais perguntas  é a resposta). Exemplo para adivinhar o nome de um elemento da turma: "É do sexo masculino?"  *sim*  "É loiro?"  *não*  "É o Paulo?"  *não*  "Então quem era?"  *Nuno*  "Indique uma característica que o distinga do Paulo"  *É alto*  ; Na próxima vez, se não for loiro, o computador em vez de sugerir "Paulo" deve perguntar se é alto: se for, é o Nuno se não é o Paulo. Codifique e corra o programa num computador.

# Ficheiros Sequenciais

Definição: Até agora, todos os dados com que temos trabalhado são armazenados na memória central do computador e consequentemente libertados assim que acaba a execução do programa. Não é, portanto, possível guardar dados de uma execução para a outra. A manipulação de ficheiros sequenciais, vai permitir armazenar dados em ficheiros na memória secundária (disco), bem como ler ficheiros já existentes.

☝ Para manipular grandes quantidades de informação, usam­‑se preferencialmente Bases de Dados, mas isso constitui só por si outra disciplina.

Abrir um ficheiro: Para começar a trabalhar com um ficheiro, é necessário começar por associar o nome *físico* do ficheiro no sistema operativo a uma variável através da qual o vamos manipular. Além disso, há outras operações implícitas ao acto de *abrir* um ficheiro (por exemplo para evitar que dois utilizadores tentem escrever nele ao mesmo tempo) mas que escapam ao âmbito deste documento. Por exemplo, para criar um ficheiro chamado "texto.txt", que vamos manipular no nosso programa através da variável *fich1*,

*fich1 ← abrir("texto.txt","w")*

em que o "*w*" indica que estamos a abrir o ficheiro para escrita. No entanto também se pode­ abrir um ficheiro já existente, seja para ler seja para acrescentar mais caracteres no fim. Os valores permitidos para o modo de abertura são os seguintes:

***w****: para escrever. Se o ficheiro já existir o seu conteúdo é apagado.*

***r****: para ler.*

***a****: para escrever, mas acrescentando no fim do conteúdo existente.*

Usar o ficheiro: Algoritmicamente, ler-de ou escrever-para um ficheiro é o mesmo que para o ecrã. Trata-se apenas de transferir o conteúdo de constantes ou variáveis para o canal de entrada ou saída pretendido. A única diferença, é o facto de ser necessário indicar o ficheiro de onde queremos *ler* os dados, ou para onde queremos *escrevê-los*. Usamos portanto uma notação ligeiramente adaptada:

*ler(fich1, var)*

*escrever(fich1, var)*

Além disso, é de boa prática *fechar* os ficheiros abertos, quando deixam de ser usados. Se se tratar do fim do programa, na prática nem sequer é necessário fazê-lo (apenas boa prática). Se for necessário reabrir o ficheiro ─ p. ex. para escrever num ficheiro que estava aberto para leitura ─ nesse caso é mesmo obrigatório *fechar* e voltar a *abrir* no modo de abertura adequado.

*fechar(fich1)*

Testar o fim do ficheiro: Finalmente, é frequente ler dados de ficheiros de que se desconhece o tamanho. Nesses casos é útil uma função que nos indique se já chegámos ao fim do ficheiro (*end-of-file*):

*eof(fich1)*

Exemplo de utilização em C++:

|  |  |
| --- | --- |
| *#include* <stdio.h>  *#include* <string.h>  main() {  *struct* nota{  *int* num;  *char* aluno[50];  *float* valor;  };  *struct* nota A, B;  A.num = 33256;  strcpy(A.aluno, "João Silva Mendes");  A.valor =17.6;  B = A;  } /\* fim de main() \*/ | ***C++***  *para "strcpy()"*  *define o tipo "nota"*  *declara as variáveis A e B como sendo do tipo "struct nota"* |

Exercícios:

1. Ler um texto até *EOF*. No final indicar o n.º de letras, de palavras e de linhas desse texto. Codifique e corra o programa num computador.
2. Dado um texto (ler até *EOF*) apresentá-lo com as letras de cada palavra escritas de trás para a frente (p. ex.: "de tras para a frente" => "ed sart arap a etnerf"). Codifique e corra o programa num computador.

# Pilhas (*Stacks*)

Definição: Uma *Stack* é uma estrutura de dados bastante simples, que permite armazenar dados sequencialmente. As duas principais operações são colocar um dado na pilha (*Push*) e retirar um dado da pilha (*Pop*). O dado que é retirado é sempre o último que foi introduzido; este mecanismo de colocar/retirar dados designa­‑se por "LIFO" (de "*Last In, First Out*" ou seja, o primeiro a sair é o que entrou em último lugar).

Operações: As operações típicas sobre *Stacks* são cinco:

1. **Inicializar** a pilha, que consiste em preparar uma pilha vazia pronta a ser utilizada;
2. **Push** ─ Acrescentar um elemento no topo da pilha;
3. **Pop** ─ Retirar o elemento do topo da pilha;
4. **Ver** o elemento que se encontra no topo da pilha sem o retirar;
5. Testar se a pilha está **vazia**.

Implementação:

# Filas (*Queues*)

Implementação:

1. Traçagem (*dry-running*) de programas

**Introdução:** Como o próprio nome indica, o *dry-running* é um processo para *correr a seco* um programa, ou parte de um. O princípio geral consiste em percorrer o programa linha a linha e agir, no papel, exactamente como faria o computador.

O domínio desta técnica é muito útil na detecção de erros. No entanto, tem que ser praticada desde o início em exemplos simples, de modo a adquirir a prática necessária à sua aplicação em partes de programas complexos da *vida real*.

**Exemplo 1:** Ordenação de 4 números por ordem decrescente (Conhecimentos necessários: Estrutura condicional *se-então*).

A primeira coisa a fazer é numerar as linhas de modo a identificá-las durante a traçagem.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | O segundo passo é escolher a(s) sequência(s) de valores com que se vai testar o programa, simulando o que o utilizador responderia quando os valores lhe fossem pedidos. Neste caso vamos arbitrar 12, 7, 10 e 15. Finalmente, para cada simulação, faz-se uma grelha como a abaixo apresentada em que, para cada linha que é executada, se altera o estado das variáveis envolvidas. | | | | | | | | | |
| 1 | *ler A* |  |
| 2 | *ler B* |  |
| 3 | *se A < B então* |  |
| 4 | *X ← A* |  |
| 5 | *A ← B* |  |
| 6 | *B ← X* |  | linha | A | B | C | D | X | A<B | B<C | C<D | ecrã |
| 7 | *ler C* |  | 1 | 12 |  |  |  |  |  |  |  |  |
| 8 | *se B < C então* |  | 2 |  | 7 |  |  |  |  |  |  |  |
| 9 | *X ← B* |  | 3 |  |  |  |  |  | F |  |  |  |
| 10 | *B ← C* |  | 7 |  |  | 10 |  |  |  |  |  |  |
| 11 | *C ← X* |  | 8 |  |  |  |  |  |  | V |  |  |
| 12 | *se A < B então* |  | 9 |  |  |  |  | 7 |  |  |  |  |
| 13 | *X ← A* |  | 10 |  | 10 |  |  |  |  |  |  |  |
| 14 | *A ← B* |  | 11 |  |  | 7 |  |  |  |  |  |  |
| 15 | *B ← X* |  | 12 |  |  |  |  |  | F |  |  |  |
| 16 | *ler D* |  | 16 |  |  |  | 15 |  |  |  |  |  |
| 17 | *se C < D então* |  | 17 |  |  |  |  |  |  |  | V |  |
| 18 | *X ← C* |  | 18 |  |  |  |  | 7 |  |  |  |  |
| 19 | *C ← D* |  | 19 |  |  | 15 |  |  |  |  |  |  |
| 20 | *D ← X* |  | 20 |  |  |  | 7 |  |  |  |  |  |
| 21 | *se B < C então* |  | 21 |  |  |  |  |  |  | V |  |  |
| 22 | *X ← B* |  | 22 |  |  |  |  | 10 |  |  |  |  |
| 23 | *B ← C* |  | 23 |  | 15 |  |  |  |  |  |  |  |
| 24 | *C ← X* |  | 24 |  |  | 10 |  |  |  |  |  |  |
| 25 | *se A < B então* |  | 25 |  |  |  |  |  | V |  |  |  |
| 26 | *X ← A* |  | 26 |  |  |  |  | 12 |  |  |  |  |
| 27 | *A ← B* |  | 27 | 15 |  |  |  |  |  |  |  |  |
| 28 | *B ← X* |  | 28 |  | 12 |  |  |  |  |  |  |  |
| 29 | *escrever A, B, C, D* |  | 29 |  |  |  |  |  |  |  |  | 15 12 10 7 |

**Exemplo 2:** Repetir o lançamento de um dado, até que ocorra um lançamento igual ao imediatamente anterior. Ignorar o lançamento repetido. No fim, indicar quantos dados foram *lançados* (Conhecimentos necessários: Estrutura de controlo de ciclos *repetir-até*).

|  |  |
| --- | --- |
| 1 | *Dado ← int(Random()\*6)+1* |
| 2 | *i ← 0* |
| 3 | *repetir* |
| 4 | *i ← i + 1* |
| 5 | *escrever Dado* |
| 6 | *DadoAnterior ← Dado* |
| 7 | *Dado ← int(Random()\*6)+1* |
| 8 | *até Dado = DadoAnterior* |
| 9 | *escrever “Foram efectuados ”, i. “ lançamentos”* |

Faça-se a traçagem para a sequência de lançamentos {5, 3, 4, 3, 3}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| linha | i | Dado | DadoAnterior | Dado = DadoAnterior | ecrã |
| 1 |  | 5 |  |  |  |
| 2 | 0 |  |  |  |  |
| 4 | 1 |  |  |  |  |
| 5 |  |  |  |  | 5 |
| 6 |  |  | 5 |  |  |
| 7 |  | 3 |  |  |  |
| 8 |  |  |  | F |  |
| 4 | 2 |  |  |  |  |
| 5 |  |  |  |  | 3 |
| 6 |  |  | 3 |  |  |
| 7 |  | 4 |  |  |  |
| 8 |  |  |  | F |  |
| 4 | 3 |  |  |  |  |
| 5 |  |  |  |  | 4 |
| 6 |  |  | 4 |  |  |
| 7 |  | 3 |  |  |  |
| 8 |  |  |  | F |  |
| 4 | 4 |  |  |  |  |
| 5 |  |  |  |  | 3 |
| 6 |  |  | 3 |  |  |
| 7 |  | 3 |  |  |  |
| 8 |  |  |  | V |  |
| 9 |  |  |  |  | Foram efectuados 4 lançamentos |

1. Cábula com Sintaxe do Qbasic

Caracteres especiais:

' Linha de comentário (colocar a pelica no início da linha)

; Controla aspecto do PRINT e do INPUT

, Controla aspecto do PRINT e do INPUT

: Separa várias instruções na mesma linha

Operadores lógicos:

NOT Negação Lógica

AND Conjunção

OR Disjunção

XOR "Ou" exclusivo

EQV Equivalência

IMP Implicação

Tabela de verdade para cada um dos operadores. F é Falso e corresponde a um 0 (zero); V é Verdadeiro e corresponde a –1 ("menos 1"):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **p** | **q** | NOT p | p AND q | p OR q | p XOR q | p EQV q | p IMP q |
| **V** | **V** | F | V | V | F | V | V |
| **V** | **F** | F | F | V | V | F | F |
| **F** | **V** | V | F | V | V | F | V |
| **F** | **F** | V | F | F | F | V | V |

Funções Trigonométricas:

COS(ângulo) Co-seno de um ângulo em radianos

SIN(ângulo) Seno de um ângulo em radianos

TAN(ângulo) Tangente de um ângulo em radianos

ATN(expr.-numérica) Arco em radianos cuja Tangente é *expr.-numérica*

Funções de Números Aleatórios:

RANDOMIZE (seed%) ou RANDOMIZE TIMER

RND[(n#)]

Funções diversas:

ABS(expr.-numérica) Módulo da expr.-numérica

SGN(expr.-numérica) 1 (um) afectado do sinal de expr.-numérica

SQR(expr.-numérica) Raiz quadrada

EXP(expr.-numérica) Exponencial

LOG(expr.-numérica) Logaritmo Neperiano

FIX(expr.-numérica) Trunca a expressão numérica à sua parte inteira.

INT(expr.-numérica) Devolve o maior número inteiro inferior a expr.-numérica

CINT(expr.-numérica)Arredonda expr.-numérica para o integer mais próximo.

CLNG(expr.-numérica)Arredonda expr.-numérica para o long integer mais próximo.

HEX$(expr.-numérica) Converte base decimal para Hexadecimal

OCT$(expr.-numérica) Converte base decimal para Octal

Tipos de variáveis: em *Basic* o próprio nome da variável define o seu tipo, através do último caractere. A seguir é apresentada uma tabela de tipos com os respectivos sufixos e com os valores mínimos e máximos. Se a variável não terminar com um dos sufixos existentes, pressupõe-se um *integer*.

Uma variável do tipo real será por exemplo batatas1! enquanto que um inteiro será por exemplo batatas2%.

|  | Sufixo | Mínimo | Máximo |
| --- | --- | --- | --- |
| Nomes das Variáveis |  | 1 caractere | 40 caracteres |
| String(comprimento) | $ | 0 caracteres | 32,767 caracteres |
| Integers | % | -32,768 | 32,767 |
| Long integers | & | -2,147,483,648 | 2,147,483,647 |
| Single-precision: | ! |  |  |
| Positivos |  | 2.802597E-45 | 3.402823E+38 |
| Negativos |  | -3.402823E+38 | -2.802597E-45 |
| Double-precision: | # |  |  |
| Positivos |  | 4.940656458412465D-324 | 1.79769313486231D+308 |
| Negativos |  | -1.79769313486231D+308 | -4.940656458412465D-324 |

Estruturas Condicionais:

IF condição THEN

[bloco­‑de­‑instruções]

[ELSEIF condition2 THEN

[bloco­‑de­‑instruções]]...

[ELSE

[bloco­‑de­‑instruções]]

END IF

SELECT CASE expressão em que:

CASE lista-de-expressões ¦ expressão: Qualquer expressão

[bloco­‑de­‑instruções] numérica ou alfanumérica.

[CASE lista-de-expressões ¦ lista-de-expressões:

[bloco­‑de­‑instruções]]... expressão [,expressão]...

[CASE ELSE expressão TO expressão

[bloco­‑de­‑instruções]] ISrelational-operator expressão

END SELECT

Operações com sequências de caracteres:

LEN(string) Comprimento de uma *string*

STR$(expr.-numérica) Converte uma expr.-numérica numa *string*

VAL(string) Converte uma string numa expressão numérica

LTRIM$(string) devolve string sem espaços à esquerda

RTRIM$(string) devolve string sem espaços à direita

LEFT$(string,n%) Devolve os n% primeiros caracteres de string

RIGHT$(string,n%) Devolve os n% últimos caracteres de string

MID$(string,start%[,length%]) length caracteres a começar em start

LCASE$(string) String em minúsculas

UCASE$(string) String em maiúsculas

ASC(string) Dá o código ASCII do primeiro caractere de *string*

CHR$(codigo%) Caractere ASCCI cujo código é codigo%

Estruturas de Controlo de Ciclos:

DO

bloco-de-instruções

LOOP UNTIL condição

WHILE condição

bloco-de-instruções

WEND

FOR contador = inicio TO fim [STEP incremento]

bloco-de-instruções

NEXT [contador]

1. Tabela ASCII

Tabela ASCII (caracteres 0 - 127)

000 (nul) 016 (dle) 032 sp 048 0 064 @ 080 P 096 ` 112 p

001 (soh) 017 (dc1) 033 ! 049 1 065 A 081 Q 097 a 113 q

002 (stx) 018 (dc2) 034 " 050 2 066 B 082 R 098 b 114 r

003 (etx) 019 (dc3) 035 # 051 3 067 C 083 S 099 c 115 s

004 (eot) 020 ¶ (dc4) 036 $ 052 4 068 D 084 T 100 d 116 t

005 (enq) 021 § (nak) 037 % 053 5 069 E 085 U 101 e 117 u

006 (ack) 022 (syn) 038 & 054 6 070 F 086 V 102 f 118 v

007 (bel) 023 (etb) 039 ' 055 7 071 G 087 W 103 g 119 w

008 (bs) 024 (can) 040 ( 056 8 072 H 088 X 104 h 120 x

009 (tab) 025 (em) 041 ) 057 9 073 I 089 Y 105 i 121 y

010 (lf) 026 (eof) 042 \* 058 : 074 J 090 Z 106 j 122 z

011 (vt) 027 (esc) 043 + 059 ; 075 K 091 [ 107 k 123 {

012 (np) 028 (fs) 044 , 060 < 076 L 092 \ 108 l 124 |

013 (cr) 029 (gs) 045 - 061 = 077 M 093 ] 109 m 125 }

014 (so) 030 ‑ (rs) 046 . 062 > 078 N 094 ^ 110 n 126 ~

015 ¤ (si) 031 ­ (us) 047 / 063 ? 079 O 095 \_ 111 o 127 

Tabela ASCII extendida (caracteres 128 - 255)

128 Ç 143 Å 158 × 172 ¼ 186 ¦ 200 + 214 Í 228 õ 242 =

129 ü 144 É 159 ƒ 173 ¡ 187 + 201 + 215 Î 229 Õ 243 ¾

130 é 145 æ 160 á 174 « 188 + 202 - 216 Ï 230 µ 244 ¶

131 â 146 Æ 161 í 175 » 189 ¢ 203 - 217 + 231 þ 245 §

132 ä 147 ô 162 ó 176 ¦ 190 ¥ 204 ¦ 218 + 232 Þ 246 ÷

133 à 148 ö 163 ú 177 ¦ 191 + 205 - 219 ¦ 233 Ú 247 ¸

134 å 149 ò 164 ñ 178 ¦ 192 + 206 + 220 \_ 234 Û 248 °

135 ç 150 û 165 Ñ 179 ¦ 193 - 207 ¤ 221 ¦ 235 Ù 249 ¨

136 ê 151 ù 166 ª 180 ¦ 194 - 208 ð 222 Ì 236 ý 250 ·

137 ë 152 ÿ 167 º 181 Á 195 + 209 Ð 223 ¯ 237 Ý 251 ¹

138 è 153 Ö 168 ¿ 182 Â 196 - 210 Ê 224 Ó 238 ¯ 252 ³

139 ï 154 Ü 169 ® 183 À 197 + 211 Ë 225 ß 239 ´ 253 ²

140 î 155 ø 170 ¬ 184 © 198 ã 212 È 226 Ô 240 ­ 254 ¦

141 ì 156 £ 171 ½ 185 ¦ 199 Ã 213 i 227 Ò 241 ± 255

142 Ä 157 Ø

1. C: Operadores, Palavras Reservadas e Opções de Compilação

Herbert Schildt, “Turbo C: Guia de Referência Básica”, McGraw-Hill, 1-35

1. Ambiente de Desenvolvimento Integrado e Editor de Turbo C

Herbert Schildt, “Turbo C: Guia de Referência Básica”, McGraw-Hill, 36-56

1. C: Funções de Entrada e de Saída (E/S)

Herbert Schildt, “Turbo C: Guia de Referência Básica”, McGraw-Hill, 81-118

1. C: Funções de Caracteres e de Sequências de Caracteres (Strings)

Herbert Schildt, “Turbo C: Guia de Referência Básica”, McGraw-Hill, 119-132

1. C: Funções Matemáticas

Herbert Schildt, “Turbo C: Guia de Referência Básica”, McGraw-Hill, 133-139

1. C: Funções Diversas

Herbert Schildt, “Turbo C: Guia de Referência Básica”, McGraw-Hill, 222-242

1. Significado e Utilização do Binário nos Computadores

A. Tenenbaum, “Estruturas de Dados Usando C”, Makron Books, 1-33

1. Vectores em C

A. Tenenbaum, “Estruturas de Dados Usando C”, Makron Books, 34-57

1. Tuplos (structs) em C

A. Tenenbaum, “Estruturas de Dados Usando C”, Makron Books, 57-85

1. Pilhas (*Stacks*)

A. Tenenbaum, “Estruturas de Dados Usando C”, Makron Books, 86-131

1. Recursividade

A. Tenenbaum, “Estruturas de Dados Usando C”, Makron Books, 132-206

1. Filas (*Queues*)

A. Tenenbaum, “Estruturas de Dados Usando C”, Makron Books, 207-223

1. Apontadores em C

Ted Jensen, “A Tutorial on Pointers and Arrays in C”

Este anexo apresenta uma explicação pormenorizada sobre apontadores em C.

Aconselha-se a leitura atenta deste anexo, para uma melhor compreensão sobre apontadores. Apesar de objectivamente não fazer parte da matéria que se pretende abarcar, o seu conhecimento facilita amplamente a compreensão de aspectos relevantes da referida matéria.

No âmbito dos objectivos de aprendizagem que fazem parte do programa, apenas é necessário o conhecimento mínimo à compreensão da passagem de parâmetros entre funções, e à utilização de algumas funções do C, nomeadamente o *scanf()* e algumas funções de manipulação de *strings*.

1. O centro de informática (CI) disponibiliza aos alunos serviços como uma conta unix, conta de correio electrónico (email), acesso à internet a partir dos laboratórios ou de casa etc. ⎯ sugere-se aos alunos que se dirijam ao CI no sentido de se informarem dos serviços de que podem usufruir e de subscrever aqueles de que necessitarem ⎯. [↑](#footnote-ref-1)
2. No Anexo X encontra-se uma apresentação sobre a utilização do binário nos computadores. Não é importante perceber tudo, no entanto a sua leitura pode ajudar a compreender melhor, alguns pormenores da matéria que serão dados mais adiante, tornando a sua aprendizagem mais interessante e *confortável*. [↑](#footnote-ref-2)
3. Informação mais pormenorizada sobre o ambiente integrado de desenvolvimento e o editor do Turbo C, pode ser encontrada no Anexo V. [↑](#footnote-ref-3)
4. Nos exames é a única consulta permitida, desde que não ultrapasse o tamanho de uma folha A4 (frente e verso). Como há avaliações práticas (em que se usa o C) e avaliações teóricas (em que apenas se usa linguagem algorítmica) o ideal é manter cábulas diferentes para as duas situações. [↑](#footnote-ref-4)
5. Procurar no disco do computador o ficheiro “qbasic.exe”. Se existir, executá-lo para entrar no respectivo ambiente de trabalho. Para introduzir o programa basta digitar as duas linhas apresentadas, e procurar nos menus o comando “RUN”. [↑](#footnote-ref-5)
6. No Anexo I está um guia de referencia sobre a utilização do Turbo C. [↑](#footnote-ref-6)
7. Os operadores dependem da linguagem. Em linguagem algoritmica, vamos convencionar o uso do asterisco (‘\*’) para representar a multiplicação por ser o mais frequentemente usado. Em geral, estes operadores mais elementares são comuns à maioria das linguagens, no entanto outras funções tendem a divergir no nome e no modo de utilização. Como um dos objectivos desta disciplina é que os alunos aprendam C, a linguagem algoritmica adoptada tenderá a utilizar funções identicas às do C sempre que necessário. Mais adiante, neste capítulo, é apresentada uma relação dos operadores existentes. [↑](#footnote-ref-7)
8. A sintaxe completa do *printf()* está disponível no Anexo VI. [↑](#footnote-ref-8)
9. Para os mais distraídos: “letras” são aqueles caracteres entre o ‘a’ e o ‘z’! Ou seja, não vale usar espaços, símbolos matemáticos, sinais de pontuação etc. etc. etc. [↑](#footnote-ref-9)
10. Os tipos disponíveis em cada linguagem podem ser encontrados nos respectivos manuais. No Anexo I está um resumo dos tipos do Qbasic. Sobre o C, ver o Anexo IV (pág.s 12, 14-26) e Anexo VI (101-104). [↑](#footnote-ref-10)
11. Linguagem Algorítmica [↑](#footnote-ref-11)
12. “pow” e “power” são funções matemáticas, uma vez que o Pascal e o C não têm operadores para estas operações. Assim, em vez de se escrever 2^3 como em BASIC, em Pascal escreve-se power(2,3) [↑](#footnote-ref-12)
13. A operação de disjunção exclusiva não é fornecida no padrão da linguagem. No entanto pode ser obtida sabendo que é equivalente a p ∧ ⎤ q ∨ ⎤ p ∧ q (se não sabe porquê, talvez esteja na altura certa para ir estudar um bocado de lógica — é suficiente a que faz parte do programa de matemática no ensino secundário). [↑](#footnote-ref-13)
14. A operação de implicação não é fornecida no padrão da linguagem. No entanto pode ser obtida sabendo que p ⇒ q é equivalente a ⎤ p ∨ q [↑](#footnote-ref-14)
15. Vamos pressupor que através do teclado... [↑](#footnote-ref-15)
16. Em Pascal também existe um “Read()” mas não é apropriado para ler do teclado. [↑](#footnote-ref-16)
17. Além do exercício que representa fazer um ou dois exemplos à mão, é altura de começar a aprender a verificar a correcção dos algoritmos feitos. Para isso é ensinada no Anexo I a técnica de traçagem de programas, que deve ser estudada no fim deste exemplo. [↑](#footnote-ref-17)
18. Com esta sistematização, pretende-se apenas ajudar os alunos a conseguir construir um algoritmo a partir da especificação de um problema, e não, introduzir conceitos de *processo de software*. [↑](#footnote-ref-18)
19. Ou “análise do problema”, para os amigos ☺ [↑](#footnote-ref-19)
20. Quando a variável de controlo não tem nenhum significado especial, para além de controlar o n.º de vezes que o ciclo é repetido, é costume usarem-se variáveis com os nomes *i, j, k, l etc.* [↑](#footnote-ref-20)