

Livro *bestseller* de programação!



Luís Damas

# Linguagem



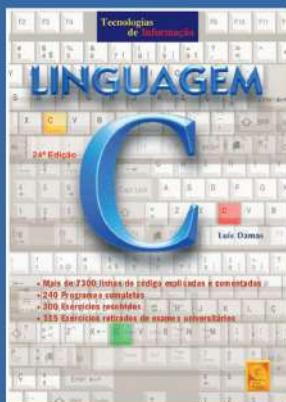
25.<sup>a</sup> Edição *Atualizada e Aumentada*  
Com 12 novos capítulos



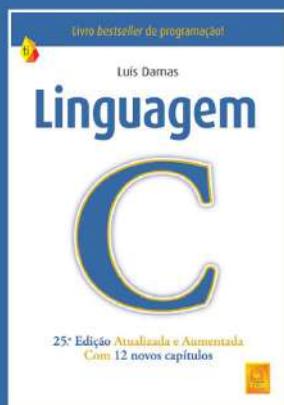


Tecnologias de Informação

O livro *bestseller*  
que acompanhou gerações  
na aprendizagem da programação



Apresenta-se  
**AGORA**  
na sua  
**25.ª edição**  
atualizada e aumentada

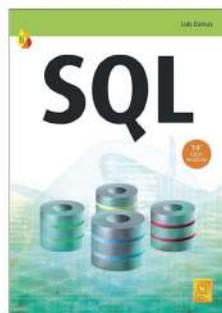


[www.fca.pt](http://www.fca.pt)

### Luís Damas

Licenciado em Informática pela Faculdade de Ciências da Universidade de Lisboa e mestre em Gestão de Informação pela Universidade Católica Portuguesa. Além do desenvolvimento de aplicações e consultadoria na área de Informática, tem lecionado disciplinas de programação, bases de dados, algoritmos e estruturas de dados e sistemas de informação em diversas instituições de ensino superior.

Do mesmo autor:



[www.fca.pt](http://www.fca.pt)

# ÍNDICE GERAL

<b>SOBRE O LIVRO</b>	<b>XIX</b>
<b>NOTA INTRODUTÓRIA</b>	<b>XXI</b>
<b>SIGLAS E ACRÓNIMOS</b>	<b>XXVIII</b>
<b>PARTE I – LINGUAGEM C</b>	<b>1</b>
<b>1 O MEU PRIMEIRO PROGRAMA</b>	<b>3</b>
1.1 INTRODUÇÃO	3
1.2 CARÁTER ESPECIAL \	9
1.3 COMENTÁRIOS	10
EXERCÍCIOS RESOLVIDOS	12
EXERCÍCIOS PROPOSTOS	13
<b>2 TIPOS DE DADOS BÁSICOS</b>	<b>17</b>
2.1 INTRODUÇÃO	17
2.2 VARIÁVEIS	18
2.2.1 NOMES DE VARIÁVEIS – INTRODUÇÃO	19
2.2.2 NOMES DE VARIÁVEIS – CUIDADOS A TER	20
2.2.3 ATRIBUIÇÃO	20
2.2.4 INTEIROS – <i>int</i>	23
2.3 OPERADORES ARITMÉTICOS	23
2.4 FUNÇÃO <i>printf</i>	24
2.5 FUNÇÃO <i>scanf</i>	25
2.6 INTEIROS E VARIAÇÕES	28
2.7 MODIFICADORES DE TIPOS	29
2.7.1 MODIFICADORES <i>short</i> , <i>long</i> E <i>long long</i>	29
2.7.2 MODIFICADORES <i>signed</i> E <i>unsigned</i>	31
2.8 REAIS – <i>float</i> E <i>double</i>	32
2.9 OPERAÇÕES SOBRE REAIS	35
2.10 CARATERES – <i>char</i>	36
2.11 CARATERES ESPECIAIS	37
2.12 FUNÇÃO <i>getchar</i>	38
2.13 <i>getchar</i> VERSUS <i>scanf</i>	39
2.14 CARATERES E INTEIROS	41
2.15 <i>CASTING</i>	42
2.16 SITUAÇÕES EM QUE INTEIROS E CARATERES NÃO SE DEVEM MISTURAR	43
2.17 CARATERES E VARIAÇÕES	46
2.18 FORMATOS DE LEITURA E ESCRITA (RESUMO)	46

EXERCÍCIOS RESOLVIDOS .....	47
EXERCÍCIOS PROPOSTOS .....	48
<b>3 INSTRUÇÕES CONDICIONAIS <i>if</i> E <i>switch</i></b> .....	<b>53</b>
3.1 INTRODUÇÃO .....	53
3.2 VALORES LÓGICOS – VERDADE E FALSO .....	54
3.3 OPERADORES RELACIONAIS .....	55
3.4 INSTRUÇÃO <i>if-else</i> .....	56
3.5 BLOCO DE INSTRUÇÕES .....	59
3.6 INDENTAÇÃO .....	60
3.7 INSTRUÇÕES <i>if-else</i> ENCADEADAS .....	62
3.8 OPERADORES LÓGICOS .....	65
3.9 PRECEDÊNCIA DOS OPERADORES LÓGICOS E RELACIONAIS .....	68
3.10 OPERADOR TERNÁRIO ? .....	69
3.11 INSTRUÇÃO <i>switch</i> .....	70
3.12 INSTRUÇÃO <i>break</i> .....	73
EXERCÍCIOS RESOLVIDOS .....	78
EXERCÍCIOS PROPOSTOS .....	81
<b>4 CICLOS</b> .....	<b>85</b>
4.1 INTRODUÇÃO .....	85
4.2 CICLO <i>while</i> .....	85
4.3 CICLO <i>for</i> .....	90
4.4 CICLO <i>do ... while</i> .....	94
4.5 CICLOS (RESUMO) .....	97
4.6 INSTRUÇÃO <i>break</i> .....	97
4.7 INSTRUÇÃO <i>continue</i> .....	98
4.8 CICLOS ENCADEADOS .....	99
4.9 CICLOS INFINITOS .....	101
4.10 OPERADORES ++ E -- .....	102
4.11 ++x VERSUS x++ .....	103
4.12 ATRIBUIÇÕES COMPOSTAS .....	105
EXERCÍCIOS RESOLVIDOS .....	106
EXERCÍCIOS PROPOSTOS .....	109
<b>5 FUNÇÕES</b> .....	<b>113</b>
5.1 INTRODUÇÃO .....	113
5.2 CARACTERÍSTICAS DE UMA FUNÇÃO .....	116
5.3 NOME DE UMA FUNÇÃO .....	116
5.4 COMO FUNCIONA UMA FUNÇÃO .....	117
5.5 ARGUMENTOS E PARÂMETROS .....	120
5.6 CORPO DA FUNÇÃO .....	122
5.7 INSTRUÇÃO <i>return</i> .....	122
5.8 FUNÇÕES QUE RETORNAM UM RESULTADO .....	122
5.9 FUNÇÕES E PROCEDIMENTOS .....	125
5.10 O “TIPO” <i>void</i> .....	126

5.11	ONDE COLOCAR AS FUNÇÕES .....	127
5.12	VARIÁVEIS LOCAIS .....	129
5.13	FUNÇÕES QUE RETORNAM UM VALOR LÓGICO .....	131
5.14	FUNÇÃO <i>exit</i> .....	132
	EXERCÍCIOS RESOLVIDOS .....	134
	EXERCÍCIOS PROPOSTOS .....	138
<b>6</b>	<b>ARRAYS</b> .....	<b>143</b>
6.1	INTRODUÇÃO .....	143
6.1.1	ARRAYS – DECLARAÇÃO .....	144
6.1.2	ARRAYS – INICIALIZAÇÃO AUTOMÁTICA .....	146
6.1.3	ARRAYS – PASSAGEM DE ARRAYS PARA FUNÇÕES .....	149
6.2	CONSTANTES .....	151
6.2.1	<i>const</i> – QUALIFICADOR DE TIPO .....	153
6.2.2	<i>#define</i> – CONSTANTES SIMBÓLICAS .....	153
6.2.3	<i>const</i> VERSUS <i>#define</i> .....	154
6.3	ARRAYS MULTIDIMENSIONAIS .....	154
6.3.1	ARRAYS MULTIDIMENSIONAIS – INICIALIZAÇÃO AUTOMÁTICA .....	156
6.3.2	ARRAYS MULTIDIMENSIONAIS – PASSAGEM PARA FUNÇÕES .....	158
	EXERCÍCIOS RESOLVIDOS .....	159
	EXERCÍCIOS PROPOSTOS .....	165
<b>7</b>	<b>STRINGS</b> .....	<b>169</b>
7.1	INTRODUÇÃO .....	169
7.2	STRINGS .....	169
7.2.1	STRINGS – INICIALIZAÇÃO AUTOMÁTICA .....	172
7.2.2	STRING VERSUS ARRAY DE CARACTERES .....	172
7.3	LEITURA E ESCRITA DE STRINGS .....	173
7.3.1	FUNÇÃO <i>printf</i> .....	173
7.3.2	FUNÇÃO <i>puts</i> – (PUT STRING).....	174
7.3.3	FUNÇÃO <i>scanf</i> .....	174
7.3.4	FUNÇÃO <i>gets</i> (GET STRING) .....	175
7.4	PASSAGEM DE STRINGS PARA FUNÇÕES.....	178
7.5	PRINCIPAIS FUNÇÕES DE MANIPULAÇÃO DE STRINGS.....	178
7.6	CONCLUSÃO .....	192
	EXERCÍCIOS RESOLVIDOS .....	193
	EXERCÍCIOS PROPOSTOS .....	194
<b>8</b>	<b>APONTADORES (POINTERS)</b> .....	<b>199</b>
8.1	INTRODUÇÃO .....	199
8.2	HISTÓRIA NO PAÍS DOS TELEFONES .....	200
8.2.1	O PAÍS DOS TELEFONES – A VINGANÇA CONTINUA.....	202
8.2.2	O PAÍS DOS TELEFONES – O ATAQUE FINAL.....	204
8.2.3	CONCLUSÃO .....	204
8.3	APONTADORES .....	204
8.4	DECLARAÇÃO DE APONTADORES .....	206

8.5	INICIALIZAÇÃO AUTOMÁTICA DE APONTADORES.....	207
8.6	APONTADORES EM AÇÃO.....	207
8.7	APONTADORES E TIPOS DE DADOS.....	209
8.8	APONTADORES E <i>ARRAYS</i> .....	211
8.9	ARITMÉTICA DE APONTADORES.....	213
8.9.1	INCREMENTO.....	213
8.9.2	DECREMENTO.....	214
8.9.3	SUBTRAÇÃO (DIFERENÇA).....	215
8.9.4	COMPARAÇÃO.....	216
8.10	RESUMO DAS OPERAÇÕES SOBRE APONTADORES.....	216
8.11	APONTADORES E <i>ARRAYS</i> – ACESSO AOS ELEMENTOS.....	217
8.12	PASSAGEM DE <i>ARRAYS</i> PARA FUNÇÕES.....	218
8.13	APONTADORES DE APONTADORES.....	221
8.14	NOTAS FINAIS.....	223
	EXERCÍCIOS RESOLVIDOS.....	224
	EXERCÍCIOS PROPOSTOS.....	228
<b>9</b>	<b>PASSAGEM DE PARÂMETROS</b> .....	<b>231</b>
9.1	INTRODUÇÃO.....	231
9.2	TIPO DE RETORNO.....	233
9.3	FUNÇÃO <i>troca</i> .....	233
9.4	TIPOS DE PASSAGEM DE PARÂMETROS.....	235
9.4.1	PASSAGEM DE PARÂMETROS POR VALOR.....	237
9.4.2	PASSAGEM DE PARÂMETROS POR REFERÊNCIA.....	238
9.5	PASSAGEM DE <i>ARRAYS</i> PARA FUNÇÕES.....	241
9.6	PASSAGEM DE ARGUMENTOS NA LINHA DE COMANDO.....	244
9.7	PARÂMETRO <i>argv</i> EM PORMENOR.....	247
	EXERCÍCIOS RESOLVIDOS.....	249
	EXERCÍCIOS PROPOSTOS.....	251
<b>10</b>	<b>FICHEIROS</b> .....	<b>253</b>
10.1	INTRODUÇÃO.....	253
10.1.1	TIPOS DE PERIFÉRICOS.....	254
10.1.2	<i>STREAMS</i> .....	254
10.2	OPERAÇÕES BÁSICAS SOBRE FICHEIROS.....	254
10.2.1	FUNÇÃO <i>fopen</i> – ABERTURA DE UM FICHEIRO.....	255
10.2.2	NOME DO FICHEIRO.....	256
10.2.3	MODO DE ABERTURA.....	257
10.2.4	ABERTURA EM MODO DE TEXTO E MODO BINÁRIO.....	257
10.2.5	FUNÇÃO <i>fclose</i> – FECHO DE UM FICHEIRO.....	259
10.2.6	FUNÇÃO <i>fflush</i> .....	260
10.2.7	FUNÇÃO <i>fcloseall</i> .....	260
10.2.8	LEITURA DE CARATERES DE UM FICHEIRO.....	261
10.3	ESCRITA DE CARATERES EM FICHEIRO.....	264
10.3.1	<i>INPUT</i> E <i>OUTPUT</i> FORMATADO.....	266
10.4	<i>STREAMS STANDARD: stdin, stdout</i> E <i>stderr</i> .....	267

10.5	REDIRECIONAMENTO DE <i>INPUT/OUTPUT</i> .....	268
10.6	<i>PIPES</i> .....	270
10.7	FILTROS .....	273
10.8	PROCESSAMENTO DE FICHEIROS BINÁRIOS .....	279
10.8.1	FUNÇÃO <i>fwrite</i> – ESCRITA DE BLOCOS EM FICHEIROS BINÁRIOS .....	280
10.8.2	FUNÇÃO <i>fread</i> – LEITURA DE BLOCOS DE FICHEIROS BINÁRIOS .....	282
10.8.3	FUNÇÃO <i>feof</i> – DETEÇÃO DO FINAL DE FICHEIRO ( <i>END-OF-FILE</i> ) .....	283
10.8.4	ACESSO SEQUENCIAL <i>VERSUS</i> ACESSO DIRETO .....	287
10.8.5	FUNÇÃO <i>ftell</i> – POSICIONAMENTO NUM FICHEIRO .....	287
10.8.6	FUNÇÃO <i>rewind</i> .....	288
10.8.7	FUNÇÃO <i>fseek</i> – POSICIONAMENTO NUM FICHEIRO .....	288
	EXERCÍCIOS RESOLVIDOS .....	293
	EXERCÍCIOS PROPOSTOS .....	295
<b>II</b>	<b>ESTRUTURAS</b> .....	<b>299</b>
11.1	INTRODUÇÃO .....	299
11.2	DEFINIÇÃO DE ESTRUTURAS .....	300
11.3	DECLARAÇÃO DE VARIÁVEIS DO TIPO ESTRUTURA .....	301
11.4	ACESSO AOS MEMBROS DE UMA ESTRUTURA .....	301
11.5	INICIALIZAÇÃO AUTOMÁTICA DE ESTRUTURAS .....	302
11.6	DEFINIÇÃO DE NOVOS TIPOS – <i>typedef</i> .....	304
11.7	ONDE DEFINIR ESTRUTURAS E <i>typedef</i> .....	305
11.8	ESTRUTURAS DENTRO DE ESTRUTURAS .....	306
11.9	PASSAGEM DE ESTRUTURAS PARA FUNÇÕES .....	307
11.10	OPERAÇÕES SOBRE ESTRUTURAS .....	312
11.11	FICHEIROS DE ESTRUTURAS .....	313
11.11.1	FUNÇÃO <i>fread</i> .....	313
11.11.2	FUNÇÃO <i>fwrite</i> .....	313
11.11.3	FUNÇÃO <i>rewind</i> .....	314
11.11.4	FUNÇÃO <i>fseek</i> .....	314
11.11.5	FUNÇÃO <i>ftell</i> .....	315
	EXERCÍCIOS RESOLVIDOS .....	315
	EXERCÍCIOS PROPOSTOS .....	315
<b>12</b>	<b>MEMÓRIA DINÂMICA</b> .....	<b>317</b>
12.1	INTRODUÇÃO .....	317
12.2	ALOCAÇÃO DE MEMÓRIA .....	318
12.3	FUNÇÃO <i>malloc</i> .....	318
12.4	FUNÇÃO <i>free</i> .....	320
12.5	FUNÇÃO <i>calloc</i> .....	321
12.6	FUNÇÃO <i>realloc</i> .....	321
12.7	<i>STACK VERSUS HEAP</i> .....	326
12.8	<i>MEMORY LEAKS</i> .....	327
12.8.1	UTILITÁRIO – <i>valgrind</i> .....	328
12.9	IMPLEMENTAÇÃO DE ESTRUTURAS DE DADOS DINÂMICAS .....	332
12.10	FILA .....	332

12.10.1	PROGRAMAÇÃO MODULAR.....	332
12.10.2	FILA – IMPLEMENTAÇÃO DE PRIMITIVAS.....	333
12.10.3	FILA – IMPLEMENTAÇÃO DINÂMICA COM UM ARRAY.....	333
12.10.4	FILA – IMPLEMENTAÇÃO EM LISTA SIMPLEMENTE LIGADA.....	341
	EXERCÍCIOS RESOLVIDOS .....	348
	EXERCÍCIOS PROPOSTOS .....	352
<b>13</b>	<b>MACROS E PRÉ-PROCESSADOR</b> .....	<b>355</b>
13.1	INTRODUÇÃO .....	355
13.2	MACROS.....	356
13.2.1	REGRAS PARA A COLOCAÇÃO DE PARÊNTESES.....	362
13.2.2	OPERADOR DE CONTINUIDADE \.....	363
13.2.3	OPERADOR # .....	363
13.2.4	OPERADOR ##.....	364
13.3	PRÉ-PROCESSADOR.....	365
13.3.1	<i>#define</i> .....	365
13.3.2	<i>#undef</i> .....	367
13.3.3	<i>#include</i> .....	368
13.4	COMPILAÇÃO CONDICIONAL.....	370
13.4.1	<i>#if ... #endif</i> .....	370
13.4.2	<i>#ifdef, #ifndef E #undef</i> .....	371
13.4.3	OPERADOR <i>defined()</i> .....	372
13.5	MACROS PREDEFINIDAS.....	373
13.5.1	<i>#line</i> .....	374
13.5.2	<i>#error</i> .....	374
13.5.3	<i>#pragma</i> .....	375
13.5.4	<i>#pragma startup E #pragma exit</i> .....	375
13.5.5	<i>#PRAGMA ONCE</i> .....	376
13.5.6	MACRO <i>assert()</i> .....	377
13.5.7	MACRO <i>offsetof()</i> .....	378
13.6	MACROS <i>VERSUS</i> FUNÇÕES.....	379
	EXERCÍCIOS RESOLVIDOS .....	380
	EXERCÍCIOS PROPOSTOS .....	382
<b>14</b>	<b>PROGRAMAÇÃO MODULAR</b> .....	<b>385</b>
14.1	INTRODUÇÃO .....	385
14.2	PROGRAMAÇÃO MODULAR.....	386
14.2.1	DIVISÃO DOS PROJETOS POR VÁRIOS FICHEIROS.....	386
14.2.2	PARTILHA DE VARIÁVEIS GLOBAIS ENTRE MÓDULOS .....	388
14.2.3	FUNÇÕES <i>static</i> .....	390
14.2.4	VARIÁVEIS <i>static</i> .....	392
14.3	UTILITÁRIO <i>nm</i> .....	393
14.3.1	<i>nm</i> – SÍMBOLOS <i>t   T (TEXT)</i> .....	396
14.3.2	<i>nm</i> – SÍMBOLOS <i>d   D (DATA)</i> .....	397
14.3.3	<i>nm</i> – SÍMBOLO <i>U (UNDEFINED)</i> .....	397
14.3.4	<i>nm</i> – SÍMBOLO <i>C (COMMON)</i> .....	398

14.3.5	<i>nm</i> – SÍMBOLOS <i>b</i> I <i>B</i> (BSS).....	398
14.4	PROTEÇÃO CONTRA INCLUSÃO MÚLTIPLA .....	398
14.4.1	<i>#pragma once</i> .....	400
14.4.2	<i>#ifndef</i> .....	401
<b>15</b>	<b>APONTADORES PARA FUNÇÕES</b> .....	<b>403</b>
15.1	INTRODUÇÃO .....	403
15.2	APONTADORES PARA FUNÇÕES .....	404
15.3	DECLARAÇÃO DE UM APONTADOR PARA UMA FUNÇÃO.....	405
15.4	ARRAY DE APONTADORES PARA FUNÇÕES .....	408
	EXERCÍCIOS RESOLVIDOS .....	409
<b>16</b>	<b>SISTEMAS DIGITAIS E OPERAÇÕES <i>bit-a-bit</i></b> .....	<b>413</b>
16.1	INTRODUÇÃO .....	413
16.2	DECIMAL, OCTAL, HEXADECIMAL E BINÁRIO .....	415
16.2.1	<i>BYTE A BYTE</i> .....	418
16.2.2	DECIMAL, OCTAL E HEXADECIMAL .....	419
16.2.2.1	BASE OCTAL.....	419
16.2.2.2	BASE HEXADECIMAL.....	420
16.3	CONVERSÕES ENTRE BASES .....	421
16.3.1	CONVERSÃO – DECIMAL PARA BINÁRIO .....	421
16.3.2	CONVERSÃO – BINÁRIO PARA OCTAL .....	422
16.3.3	CONVERSÃO – BINÁRIO PARA HEXADECIMAL .....	423
16.3.4	CONVERSÃO – BINÁRIO PARA DECIMAL .....	424
16.3.5	CONVERSÃO – OCTAL PARA DECIMAL .....	424
16.3.6	CONVERSÃO – HEXADECIMAL PARA DECIMAL.....	424
16.4	OPERAÇÕES <i>bit-a-bit</i> .....	424
16.4.1	OPERADORES DE MANIPULAÇÃO DE <i>BITS</i> ( <i>BITWISE OPERATORS</i> ) .....	425
16.5	<i>BIT FIELDS</i> .....	427
	EXERCÍCIOS RESOLVIDOS .....	430
	EXERCÍCIOS PROPOSTOS .....	435
<b>17</b>	<b>ELEMENTOS ADICIONAIS</b> .....	<b>437</b>
17.1	INTRODUÇÃO .....	437
17.2	<i>enum</i> .....	437
17.3	<i>register</i> .....	439
17.4	<i>union</i> .....	440
17.5	<i>goto</i> .....	442
17.6	FUNÇÃO <i>strtok</i> .....	443
	<b>PARTE II – ALGORITMOS E ESTRUTURAS DE DADOS</b> .....	<b>447</b>
<b>18</b>	<b>RECURSIVIDADE</b> .....	<b>449</b>
18.1	INTRODUÇÃO .....	449
18.2	ITERATIVIDADE <i>VERSUS</i> RECURSIVIDADE .....	450

18.3	TIPOS DE RECURSIVIDADE .....	452
18.4	FUNÇÃO <i>fatorial</i> .....	453
18.5	COMO FUNCIONA UMA CHAMADA RECURSIVA .....	456
18.6	REGRAS PARA A ESCRITA DE FUNÇÕES RECURSIVAS .....	457
18.7	RECURSIVIDADE USANDO <i>STRINGS</i> .....	457
18.8	OUTROS EXEMPLOS DE RECURSIVIDADE .....	460
18.9	TORRES DE HANOI.....	464
<b>19</b>	<b>NOTAÇÃO <i>Big-O</i></b> .....	<b>467</b>
19.1	INTRODUÇÃO .....	467
19.2	ANÁLISE DE EFICIÊNCIA.....	468
19.2.1	NOTAÇÃO <i>Big-O</i> .....	469
19.2.2	TIPOS DE CRESCIMENTO.....	469
19.2.3	ANÁLISE DE CASOS .....	471
19.2.4	ANÁLISE DO MELHOR CASO ( <i>BEST CASE</i> ).....	471
19.2.5	ANÁLISE DO PIOR CASO ( <i>WORST CASE</i> ).....	472
19.2.6	SITUAÇÃO MÉDIA ( <i>AVERAGE CASE</i> ).....	472
19.3	ALGUMAS REGRAS A LEVAR EM CONTA.....	472
19.4	EXEMPLOS .....	473
19.5	NOTAÇÃO <i>Big-O</i> – POTENCIAIS PROBLEMAS .....	473
<b>20</b>	<b>ORDENAÇÃO (<i>SORTING</i>)</b> .....	<b>477</b>
20.1	INTRODUÇÃO .....	477
20.2	PORQUÊ ORDENAR?.....	478
20.3	ORDENAÇÃO – EM QUE CONSISTE? .....	479
20.3.1	DADOS .....	479
20.3.2	CHAVE DE ORDENAÇÃO.....	479
20.4	TIPOS DE ALGORITMOS DE ORDENAÇÃO .....	481
20.4.1	EFICIÊNCIA .....	481
20.4.2	ORDENAÇÃO SIMPLES <i>VERSUS</i> ORDENAÇÃO COMPLEXA.....	482
20.4.3	ORDENAÇÃO ESTÁVEL <i>VERSUS</i> ORDENAÇÃO INSTÁVEL.....	483
20.4.3.1	DEFINIÇÃO .....	483
20.4.4	ORDENAÇÃO INTERNA <i>VERSUS</i> ORDENAÇÃO EXTERNA .....	484
20.4.4.1	DEFINIÇÃO .....	484
20.4.5	ORDENAÇÃO DIRETA <i>VERSUS</i> ORDENAÇÃO INDIRETA .....	485
20.4.6	ORDENAÇÃO ADAPTATIVA <i>VERSUS</i> ORDENAÇÃO NÃO ADAPTATIVA.....	486
20.4.7	ORDENAÇÃO COM OU SEM ESTRUTURAS AUXILIARES.....	487
20.5	ALGORITMOS DE ORDENAÇÃO ELEMENTARES.....	487
20.5.1	<i>BUBBLE SORT</i> .....	487
20.5.2	<i>SELECTION SORT</i> .....	495
20.5.3	<i>INSERTION SORT</i> .....	501
20.6	ALGORITMOS DE ORDENAÇÃO AVANÇADOS .....	506
20.6.1	DIVISÃO E CONQUISTA.....	506
20.6.2	<i>MERGE SORT</i> .....	507
20.6.3	<i>QUICK SORT</i> .....	513

20.7	<i>qsort</i> .....	520
20.7.1	<i>qsort</i> – ORDENAÇÃO DESCENDENTE .....	522
20.7.2	<i>qsort</i> – ORDENAÇÃO DE <i>STRINGS</i> .....	523
20.7.3	<i>qsort</i> – ORDENAÇÃO DE ESTRUTURAS COMPLEXAS .....	525
20.7.4	<i>qsort</i> – ORDENAÇÃO INDIRETA DE UM <i>ARRAY</i> .....	527
20.8	CONCLUSÃO .....	531
<b>21</b>	<b>PESQUISA (<i>SEARCHING</i>)</b> .....	<b>533</b>
21.1	INTRODUÇÃO.....	533
21.2	TIPOS DE PESQUISA .....	537
21.3	PESQUISA SEQUENCIAL.....	538
21.3.1	PESQUISA SEQUENCIAL EM CONJUNTOS NÃO ORDENADOS.....	539
21.3.1.1	ANÁLISE DE EFICIÊNCIA .....	541
21.3.2	PESQUISA SEQUENCIAL EM CONJUNTOS ORDENADOS.....	542
21.3.2.1	ANÁLISE DE EFICIÊNCIA .....	543
21.4	PESQUISA BINÁRIA .....	543
21.4.1	IMPLEMENTAÇÃO ITERATIVA.....	544
21.4.2	IMPLEMENTAÇÃO RECURSIVA.....	546
21.4.3	ANÁLISE DE EFICIÊNCIA.....	548
21.5	COMPARAÇÃO DE RESULTADOS .....	549
21.6	FUNÇÃO <i>bsearch</i> .....	549
	EXERCÍCIOS RESOLVIDOS .....	555
<b>22</b>	<b>PILHA (<i>STACK</i>)</b> .....	<b>561</b>
22.1	INTRODUÇÃO.....	561
22.2	PILHA – IMPLEMENTAÇÃO ESTÁTICA .....	562
22.2.1	ESTRUTURA DE DADOS .....	562
22.2.2	INICIALIZAÇÃO DA PILHA .....	563
22.2.3	FUNÇÕES QUE INDICAM O ESTADO DA PILHA .....	563
22.2.4	INSERÇÃO DE ELEMENTOS .....	564
22.2.5	REMOÇÃO DE ELEMENTOS .....	565
22.2.6	LISTAGEM DE ELEMENTOS.....	565
22.2.7	ANÁLISE DE EFICIÊNCIA.....	566
22.3	PILHA – IMPLEMENTAÇÃO DINÂMICA .....	566
22.3.1	ESTRUTURA DE DADOS .....	566
22.3.2	INICIALIZAÇÃO DA PILHA .....	567
22.3.3	FUNÇÕES QUE INDICAM O ESTADO DA PILHA .....	567
22.3.4	INSERÇÃO DE ELEMENTOS .....	568
22.3.5	REMOÇÃO DE ELEMENTOS .....	568
22.3.6	LISTAGEM DE ELEMENTOS.....	569
22.3.7	ANÁLISE DE EFICIÊNCIA.....	569
22.4	PILHA DINÂMICA – IMPLEMENTAÇÃO COM UMA LISTA SIMPLEMENTE LIGADA .....	570
22.4.1	ESTRUTURA DE DADOS .....	570
22.4.2	INICIALIZAÇÃO DA PILHA .....	571
22.4.3	FUNÇÕES QUE INDICAM O ESTADO DA PILHA .....	572

22.4.4	INSERÇÃO DE UM ELEMENTO .....	573
22.4.5	REMOÇÃO DE UM ELEMENTO .....	574
22.4.6	LISTAGEM DOS ELEMENTOS .....	576
22.4.7	ANÁLISE DE EFICIÊNCIA .....	576
	EXERCÍCIOS RESOLVIDOS .....	576
<b>23</b>	<b>TIPOS ABSTRATOS DE DADOS</b> .....	<b>579</b>
23.1	INTRODUÇÃO .....	579
23.2	TIPOS DE DADOS BÁSICOS .....	579
23.3	TIPOS DE DADOS DO UTILIZADOR .....	581
23.4	TAD – TIPOS ABSTRATOS DE DADOS .....	584
23.4.1	DIVISÃO DA APLICAÇÃO EM MÓDULOS .....	585
23.4.2	TIPOS INCOMPLETOS DE DADOS .....	588
23.5	TAD – UMA DECLARAÇÃO E MÚLTIPLAS DEFINIÇÕES .....	591
	EXERCÍCIOS RESOLVIDOS .....	594
<b>24</b>	<b>FILA (QUEUE)</b> .....	<b>601</b>
24.1	INTRODUÇÃO .....	601
24.2	COMO FUNCIONA UMA FILA .....	602
24.3	FILA – IMPLEMENTAÇÃO ESTÁTICA .....	602
24.3.1	INICIALIZAÇÃO .....	603
24.3.2	FUNÇÕES QUE INDICAM O ESTADO DA FILA .....	604
24.3.3	INSERÇÃO DE ELEMENTOS .....	605
24.3.4	REMOÇÃO DE ELEMENTOS .....	605
24.3.5	ANÁLISE DE EFICIÊNCIA .....	608
24.4	FILA CIRCULAR .....	609
24.4.1	INICIALIZAÇÃO .....	612
24.4.2	FUNÇÕES QUE INDICAM O ESTADO DA FILA .....	612
24.4.3	INSERÇÃO DE ELEMENTOS .....	613
24.4.4	REMOÇÃO DE ELEMENTOS .....	614
24.4.5	IMPRESSÃO DA FILA .....	615
24.4.6	ANÁLISE DE EFICIÊNCIA .....	617
24.5	FILA DINÂMICA – SIMPLEMENTE LIGADA .....	617
24.5.1	IMPLEMENTAÇÃO .....	617
24.5.2	ANÁLISE DE EFICIÊNCIA .....	626
24.6	FILA DINÂMICA – DUPLAMENTE LIGADA .....	626
24.6.1	INICIALIZAÇÃO DA FILA .....	627
24.6.2	ESTUDO DO ESTADO DA FILA .....	628
24.6.3	INSERÇÃO DE ELEMENTOS .....	628
24.6.4	REMOÇÃO DE ELEMENTOS .....	630
24.6.5	ANÁLISE DE EFICIÊNCIA .....	632
<b>25</b>	<b>LISTAS</b> .....	<b>635</b>
25.1	INTRODUÇÃO .....	635
25.2	LISTAS ESTÁTICAS <i>VERSUS</i> LISTAS DINÂMICAS .....	636

25.2.1	MEMÓRIA ESTÁTICA – VANTAGENS E DESVANTAGENS.....	636
25.2.2	MEMÓRIA DINÂMICA – VANTAGENS E DESVANTAGENS.....	637
25.3	FILAS DE PRIORIDADES .....	638
25.3.1	FILA DE PRIORIDADES – IMPLEMENTAÇÃO COM ARRAY DINÂMICO .....	639
25.3.2	FILA DE PRIORIDADES – IMPLEMENTAÇÃO DINÂMICA .....	645
<b>26</b>	<b>CONJUNTOS (SETS)</b> .....	<b>655</b>
26.1	INTRODUÇÃO.....	655
26.2	DEFINIÇÃO.....	655
26.3	OPERAÇÕES BÁSICAS SOBRE CONJUNTOS .....	657
26.3.1	PERTENÇA (EXISTS).....	657
26.3.2	CONTIDO .....	657
26.3.3	CONTÉM .....	657
26.3.4	UNIÃO.....	657
26.3.5	INTERSEÇÃO .....	658
26.3.6	CARDINALIDADE .....	659
26.3.7	IGUALDADE.....	660
26.3.8	DISJUNÇÃO.....	660
26.3.9	DIFERENÇA (MINUS).....	660
26.4	CONJUNTO (SET) DE INTEIROS – IMPLEMENTAÇÃO .....	661
26.5	CONJUNTO (SET) DE CARATERES ASCII – IMPLEMENTAÇÃO.....	669
<b>27</b>	<b>ÁRVORES (TREES)</b> .....	<b>677</b>
27.1	INTRODUÇÃO.....	677
27.2	CONCEITOS BÁSICOS.....	677
27.3	PERCURSOS.....	679
27.4	ÁRVORES BINÁRIAS .....	680
27.5	ÁRVORES BINÁRIAS DE PESQUISA .....	680
27.5.1	ÁRVORES BINÁRIAS DE PESQUISA – IMPLEMENTAÇÃO COM O TIPO BÁSICO (int)...	680
27.5.2	ÁRVORES BINÁRIAS DE PESQUISA – IMPLEMENTAÇÃO COM UM TIPO DE DADOS COMPOSTO (PERSON).....	692
27.6	BALANCEAMENTO.....	704
<b>28</b>	<b>HASHING (TABELAS DE DISPERSÃO)</b> .....	<b>709</b>
28.1	INTRODUÇÃO.....	709
28.2	FUNÇÃO DE HASH.....	712
28.3	COLISÕES .....	712
28.4	RESOLUÇÃO DE COLISÕES .....	721
28.4.1	OPEN ADDRESSING.....	723
28.4.2	LINEAR PROBING.....	723
28.4.3	QUADRATIC PROBING .....	725
28.4.4	DUPLO HASHING.....	726
28.4.5	CHAINING.....	726
28.5	OPEN ADDRESSING – EXEMPLO.....	728

Nesta 25.<sup>a</sup> edição, procedeu-se a uma atualização do texto e a uma revisão detalhada do conteúdo do livro, tendo como objetivo principal fornecer aos leitores uma abordagem ainda mais clara, prática e completa sobre a linguagem C e promovendo uma aprendizagem sólida e eficaz, o que reflete a evolução que tem existido na linguagem e nas práticas do setor.

Todos os capítulos da edição anterior foram revistos, aplicando o novo acordo ortográfico, e os conteúdos de alguns deles foram significativamente expandidos, como operações *bit-a-bit*, apontadores para funções e outros.

Com o intuito de aprofundar o conhecimento técnico dos leitores, nesta edição foi adicionada uma segunda parte que contém 12 novos capítulos inteiramente dedicados aos algoritmos e estruturas de dados e algumas técnicas de programação, fornecendo a preparação necessária para a evolução dos leitores para tópicos de programação avançada.

Os novos capítulos incluem:

- Recursividade;
- Notação *Big-O*;
- Programação modular;
- Tipos Abstratos de Dados (TAD);
- Ordenação (*Bubble Sort*, *Selection Sort*, *Insertion Sort*, *Quick Sort*, *Merge Sort*);
- Pesquisa (sequencial e binária);
- Pilhas, filas e listas (normais, circulares e de prioridades);
- Conjuntos;
- Árvores e árvores binárias de pesquisa;
- Tabelas de dispersão (*Hashing*).

Reconhecendo a importância de algumas ferramentas especiais no desenvolvimento moderno em C, foram incluídas breves referências práticas aos utilitários *valgrind* (para deteção de fugas de memória) e *nm* (para inspeção de símbolos em ficheiros binários).

O código de todos os exercícios práticos encontra-se disponível para *download* na página do livro em [www.fca.pt](http://www.fca.pt).

Alguns números a considerar:

- Mais de 370 programas completos;
- Mais de 1000 funções implementadas;
- Mais de 70 páginas com as soluções dos exercícios propostos, acedidas através de códigos QR;
- Mais de 35 000 linhas de código.

# NOTA INTRODUTÓRIA

Bem-vindos ao mundo da programação em C!

Este livro irá apresentar as características de uma linguagem de programação que se tornou um fenómeno de popularidade, a linguagem C.

Dicas e truques usados pelos programadores mais experientes serão também apresentados através de múltiplos exemplos explicados e comentados.

A primeira edição deste livro descreveu as características da linguagem. Nesta nova edição, foram acrescentados 12 novos capítulos direcionados para a área dos algoritmos e das estruturas de dados implementados em C. Existem também capítulos adicionais que cobrem tópicos genéricos, como recursividade, notação *Big-O* e tipos abstratos de dados.

No final de alguns capítulos, serão propostos exercícios práticos relacionados com a matéria dos mesmos, cuja solução se encontra disponível através de códigos QR.

## NOTA

→ As soluções vão ser acedidas através de um código QR disponibilizado no final de cada capítulo.

## BREVE HISTÓRIA DA LINGUAGEM C

Embora possua um nome estranho quando comparada com outras linguagens de terceira geração, como o FORTRAN, o PASCAL ou o COBOL, a linguagem C foi criada em 1972, nos Bell Telephone Laboratories, por Dennis Ritchie e tinha por finalidade permitir a escrita de um sistema operativo (o Unix), utilizando uma linguagem de relativo alto nível e evitando, assim, o recurso ao *Assembly*.

Devido às suas capacidades e características, e através da divulgação do sistema Unix pelas universidades dos Estados Unidos da América, a linguagem C cedo deixou as portas dos Laboratórios Bell, disseminou-se e tornou-se conhecida entre os programadores, independentemente dos projetos em que estivessem envolvidos, sendo o livro *The C Programming Language*, de Kernighan e Ritchie (1978)<sup>1</sup>, a principal fonte de informação sobre a linguagem editado por essa altura.

Esta dispersão levou a que diferentes organizações desenvolvessem e utilizassem diferentes versões da linguagem C, criando, assim, alguns problemas de portabilidade, e não só.

Em 1983, o ANSI (American National Standards Institute) criou um Comité para a definição de um padrão para a linguagem C, o que culminou no desenvolvimento do padrão ANSI C em 1989, conhecido como C89, revisto posteriormente nos padrões C99, C11 e C17.

O nome da linguagem (e a própria linguagem) resulta da evolução de uma outra linguagem de programação, desenvolvida por Ken Thompson também nos Laboratórios Bell e denominada B. Assim, é perfeitamente natural que a evolução da linguagem **B** viesse a dar origem a uma nova linguagem, denominada **C**.

<sup>1</sup> Kernighan, B. W. & Ritchie, D. M. (1978). *The C Programming Language*. Prentice-Hall.

## PORQUÊ PROGRAMAR EM C?

A linguagem C é de tal forma impactante na evolução das linguagens de programação que quase todas as linguagens de programação que lhe sucederam usam a sintaxe igual ou baseada nas instruções da linguagem C.

Qual é a área de desenvolvimento a que a linguagem se destina?

A resposta é “nenhuma em particular”. É aquilo que, habitualmente, se denomina por *general purpose*, o que representa uma das suas grandes vantagens, pois a linguagem C adapta-se ao desenvolvimento de qualquer tipo de projeto, como sistemas operativos, interfaces gráficas e processamento de registos. A linguagem C foi também utilizada para escrever o próprio compilador da linguagem (e também para escrever os compiladores de outras linguagens).

Principais características da linguagem C:

- **Paradigma imperativo** – O C é uma linguagem imperativa extremamente potente e flexível na qual as instruções são executadas numa determinada sequência, alterando o estado das variáveis de um programa, começando cada instrução apenas quando a anterior termina;
- **Rapidez** – A linguagem C consegue obter *performances* semelhantes às obtidas pelo *Assembly*, usando instruções de alto nível;
- **Simples** – A sua sintaxe é extremamente simples e o número de palavras reservadas, tipos de dados básicos e operadores é diminuto, reduzindo, assim, a quantidade de tempo e esforço necessários à aprendizagem da linguagem;
- **Portável** – Existem *standards* [ANSI e ISO (International Organization for Standardization)] específicos que definem as características a que qualquer compilador da linguagem deve obedecer. Deste modo, o código escrito numa máquina pode ser transportado para outra máquina e compilado sem qualquer alteração (ou com um número reduzido de alterações);
- **Popular** – A linguagem C é universalmente conhecida e reconhecida como um marco no desenvolvimento das linguagens de programação, tendo servido de base para um grande número de linguagens de programação que evoluíram a partir dela, como o C++, o Java, o JavaScript, o C#, o Kotlin e o Python. Está muito documentada em livros, revistas da especialidade, manuais, etc. Existem compiladores para todo o tipo de arquiteturas e computadores;
- **Modular** – A linguagem C permite o desenvolvimento modular de aplicações, facilitando a separação de projetos em módulos distintos e independentes, com recurso à utilização de funções específicas dentro de cada módulo;
- **Alto nível** – A linguagem C permite também o acesso à maior parte das funcionalidades do *Assembly*, utilizando expressões e instruções de alto nível. É possível, por exemplo, aceder à memória diretamente, utilizando o endereço de qualquer objeto (seja variável ou função), e manipular diretamente a memória pertencente a esses objetos sem qualquer tipo de restrições, o que aumenta a flexibilidade da linguagem;
- **Bibliotecas muito poderosas** – A linguagem C contém um número reduzido de palavras-chave, o que indica que as capacidades da linguagem são muito limitadas, e, na realidade, são, mas esta é precisamente uma característica da linguagem. A maior parte das funcionalidades que a linguagem tem são-lhe adicionadas pela utilização de funções que existem em bibliotecas adicionais e que realizam todo o tipo de tarefas, desde a escrita de um simples carácter no ecrã, até ao processamento de ficheiros, à movimentação de dados na memória do computador, etc.;

- Criar o executável final (opção *-o*) *myprog* a partir do ficheiro-objeto *myprog.o*:

```
$ cc myprog.o -o myprog
```

- Compilar e “linkar” o ficheiro *myprog.c*, gerando o executável final (opção *-o=output*) *myprog*:

```
$ cc myprog.c -o myprog
ou
$ make myprog
```

- 4. Execução do programa** – Se o processo de “linkagem” terminar com sucesso, temos, então, disponível um ficheiro executável. Se é executável, podemos executá-lo digitando o seu nome:

```
$ myprog
ou
$ ./myprog
```

Se o programa não faz aquilo que deveria fazer, é porque o elemento mais fraco de todo este processo – o programador – se enganou. Deverá, então, editar o código fonte, alterá-lo e voltar a realizar todo o processo.

#### NOTA

- O ciclo de desenvolvimento de uma aplicação anteriormente apresentado é verdadeiro para a maioria das linguagens. No entanto, a linguagem C adiciona uma fase antes da compilação e que tem por missão expandir todas as macros e executar todas as diretivas para o pré-processador. Mais informações sobre esta fase podem ser obtidas no Capítulo 13.

Para ver o resultado do trabalho do pré-processador, pode usar a opção *-E* do compilador:

```
$ cc -E myprog.c
```

Podemos, então, afirmar que, no caso da linguagem C, o ciclo de desenvolvimento de uma aplicação decorre nas seguintes cinco fases:

1. Escrita do código-fonte (programador).
2. Pré-processamento (pré-processador).
3. Compilação (compilador).
4. “Linkagem” (*linker*).
5. Execução do programa.

## COMPOSIÇÃO DESTES LIVROS

Este livro foi escrito com o objetivo de servir como fonte de referência da linguagem para estudantes, sendo, por isso, apresentados múltiplos exemplos de programas escritos na linguagem ao longo de todo o documento e para todos os assuntos em análise.

Pretende-se que a aprendizagem seja baseada em conceitos teóricos claros e que estes sejam solidificados com a realização de um número alargado de exercícios práticos.

Sempre que possível, será apresentado o código de um ou mais programas para exemplificar a matéria descrita.

Na maior parte dos casos, cada programa é denominado seguindo a seguinte terminologia: **prog $nn$ kk.c**, em que  $nn$  representa o número do capítulo a que o programa pertence e  $kk$  o número do programa dentro desse capítulo.

Por exemplo, `prog0305.c` é o nome do quinto programa apresentado no Capítulo 3 deste livro.

Os programas possuem sempre o seguinte aspeto, sendo o nome do programa colocado por baixo da listagem do código:

```
1: #include <stdio.h>
2: main()
3: {
4:     printf("Hello World\n");
5: }
```

### PROG0103.C

Antes de cada linha de código, é colocado o número da linha, apenas por simplicidade de representação e referência. No entanto, ao escrever um programa, nunca coloque o número da linha, pois irá obter um erro de compilação.

A sintaxe das várias instruções ao longo do livro é apresentada através do seguinte formato:

**Exemplo:** A definição de variáveis faz-se utilizando a seguinte sintaxe:

```
tipo var1 [, var2 , ... , varn];
```

O uso de parênteses retos na sintaxe indica que a componente dentro dos parênteses é opcional.

Sempre que se pretende representar a execução de um programa, este é representado como sendo executado no *prompt* `$`:

```
$ prog0103
Hello World
$
```

No exemplo anterior, é executado o programa `prog0103`, que escreveu no ecrã a mensagem *Hello World*, deixando, em seguida, o cursor na linha seguinte (onde aparece novamente o *prompt*).

Embora seja possível executar os comandos no ambiente Linux tal como mostrados no exemplo anterior, normalmente, é necessário indicar que o executável se encontra na pasta atual, colocando o nome do programa precedido por um ponto e uma barra:

```
$ ./prog0103
Hello World
$
```

Por simplicidade, a forma como se mostra a execução de comandos/programas ao longo do livro é apresentada no formato `$ progxxyy`.

Contudo, muito provavelmente, para executar algo no seu sistema Linux deverá preceder o comando da pasta atual onde se encontra: `$ ./progxxyy`.

<sup>ii</sup> O *prompt* varia de sistema para sistema e é, habitualmente, `C:\>` na linha de comando do Windows e `#` ou `$` nos sistemas Linux mais comuns.

# INSTRUÇÕES CONDICIONAIS *if* E *switch*

## OBJETIVOS

- Valores lógicos *verdade* e *falso*
- Expressões e condições
- Operadores relacionais (`==`, `!=`, `>`, `<`, `>=`, `<=`)
- Operadores lógicos (`&&`, `||`, `!`)
- Instruções *if-else* e *switch*
- Instrução *break*
- Instruções *if-else* encadeadas
- Operador ternário (*cond*) ? *exp1* : *exp2*
- Precedência entre operadores
- Blocos de instruções
- Indentação

## 3.1 INTRODUÇÃO

Os programas que escrevemos até este momento estão particularmente adaptados a um mundo “perfeito”, sem erros, dúvidas ou qualquer tipo de variações. As instruções seguem-se umas às outras, seguindo sempre a mesma ordem de execução, quaisquer que sejam os valores de entrada.

Com a matéria apresentada até agora, podíamos escrever um programa que tivesse como objetivo preparar uma pessoa para o seu dia de trabalho. Seria algo parecido a:

```
Vestir camisa;  
Vestir camisola;  
Vestir calças;  
Calçar sapatos;  
Pentear  
...
```

No entanto, basta pensar num dia comum e verificar que, mal nos levantamos, uma das primeiras coisas que fazemos é verificar o estado do tempo, de forma a decidir se devemos ir mais ou menos agasalhados.

No caso de nos encontrarmos em pleno inverno, o programa anterior poderia estar mais ou menos adaptado, no entanto, caso o sol brilhasse de forma abrasadora, talvez não fosse a melhor maneira de nos equiparmos para o trabalho, pois não faz qualquer sentido vestir camisolas durante o verão.

Qual seria, então, a solução? Se retirarmos a instrução "*Vestir camisola*", o programa fica adaptado ao verão, mas não ao inverno.

A solução passa por manter a linha de código com a instrução "*Vestir camisola*" no programa, mas executá-la apenas quando a temperatura for suficientemente baixa.

Ao contrário dos programas anteriormente apresentados, em que todas as instruções eram sempre executadas, podemos escrever programas que tenham a capacidade de tomar decisões sobre que instruções executar:

```
Vestir camisa;
Se estiver um frio de rachar então
  Vestir camisola;
Vestir calças;
Calçar capatos;
Pentear;
...
```

## 3.2 VALORES LÓGICOS – VERDADE E FALSO

A linguagem C possui apenas quatro tipos de dados (*int*, *float*, *char* e *double*). Não existe, por isso, nenhum tipo que permita representar os valores lógicos *verdade* e *falso*.

Existem linguagens de programação que disponibilizam um tipo específico para representar valores lógicos (por exemplo, *Boolean*, *bool*).

### NOTA

→ Em C, não existe um tipo específico de dados para armazenar valores lógicos.

Em C, qualquer valor 0 (zero) poderá ser interpretado como o valor lógico *falso*.

Qualquer valor diferente de 0 (zero) representa o valor lógico *verdade*.

### Exemplos:

```
Falso   : 0, 0.0, '\0', NULL4
Verdade : 2, -3, 123.45, 0.000001, "hello", '@', '\n', ""
```

### NOTA

→ O valor lógico *verdade* em C não é o valor 1, mas qualquer valor diferente de 0 (zero). O valor 1 é apenas um dos valores possíveis para representar o valor lógico *verdade*.

Os valores lógicos podem resultar da avaliação de afirmações:

- A terra é quadrada. (*falso*);

<sup>4</sup> '\0' representa um *char* com o número zero (ASCII 0). *NULL* é uma constante especial em C que representa o endereço de memória número 0.

# 5

## FUNÇÕES

### OBJETIVOS

- Funções e procedimentos
- Cabeçalho e corpo de uma função
- Parâmetros e argumentos
- Passagem de parâmetros de tipos básicos
- Valor de retorno (*return*)
- O “tipo” *void*
- Protótipos de funções
- Variáveis locais
- *Scope* das variáveis

### 5.1 INTRODUÇÃO

Embora ainda não saibamos como escrever uma função, já as temos utilizado ao longo dos programas apresentados neste livro. São exemplos as funções *printf*, *scanf*, *getchar*, *putchar*, etc., que fazem parte da biblioteca de funções *standard* do C e que estão disponíveis e acompanham o compilador da linguagem.

Neste capítulo, iremos aprender como escrever funções e procedimentos, como comunicar com as funções através da passagem de parâmetros e como devolver algum valor como resultado do processamento de uma função.

É indispensável que qualquer programador de C (ou de outra qualquer linguagem de programação) domine em absoluto a escrita de programas de forma modular, através de procedimentos e funções.

**Problema:** Escreva um programa que coloque no ecrã o seguinte *output*, escrevendo cada linha de 20 asteriscos através de um ciclo *for*.

```
*****  
Números entre 1 e 5  
*****  
1  
2  
3  
4  
5  
*****
```

```

1: #include <stdio.h>
2:
3: int main(void)
4: {
5:     // Escrever o cabeçalho
6:     for (int i=1 ; i<=20 ; i++)
7:         putchar('*');
8:     putchar('\n');
9:
10:    puts("Números entre 1 e 5");
11:
12:    for (int i=1 ; i<=20 ; i++)
13:        putchar('*');
14:    putchar('\n');
15:
16:    // Escrever os valores 1..5
17:    for (int i=1; i<=5 ; i++)
18:        printf("%d\n",i);
19:
20:    // Escrever o rodapé
21:    for (int i=1 ; i<=20 ; i++)
22:        putchar('*');
23:    putchar('\n');
24:
25:    return 0;
26: }

```

**PROG0501.C**

Como se pode observar, o conjunto de código utilizado para escrever uma linha de asteriscos no ecrã aparece repetido três vezes:

```

for (int i=1 ; i<=20 ; i++)
    putchar('*');
    putchar('\n');

```

O ideal seria escrever esta porção de código uma única vez e poder fazer a sua invocação sempre que fosse necessário.

**Problema:** Escreva um programa que coloque uma linha com 20 asteriscos no ecrã.

```

1: #include <stdio.h>
2:
3: int main(void)
4: {
5:     for (int i=1 ; i<=20 ; i++)
6:         putchar('*');
7:     putchar('\n');
8:
9:     return 0;
10: }

```

**PROG0502.C**

O programa anterior coloca no ecrã uma linha com 20 asteriscos. Foi implementado no contexto da escrita de um programa e todo o código foi colocado dentro da função *main*. Como a sua função é escrever uma linha, em vez de lhe chamarmos *main*, vamos chamá-lo *linha*.

```

1: #include <stdio.h>
2:
3: int linha(void)
4: {

```

**5.5** *float getVAL(float x, int n, float t)*

Devolve o VAL (Valor Atual Líquido) para  $n$  anos, à taxa  $t$ , que é definido através da seguinte fórmula:

$$VAL = \frac{x}{(1+t)} + \frac{x}{(1+t)^2} + \frac{x}{(1+t)^3} + \dots + \frac{x}{(1+t)^n}$$

**Sugestão:** Utilize a função *my\_power* implementada anteriormente (consultar PROG0514.C).

**5.6** *long int n\_segundos(int n\_horas)*

Devolve o número de segundos que um conjunto de horas tem:

```
n_segundos(0)    → 0
n_segundos(1)    → 3600
n_segundos(2)    → 7200
```

**5.7** *long int num(int n\_horas, char tipo)*

Semelhante à função anterior, mas recebe um parâmetro adicional que indica o que se pretende obter: 'h' (horas), 'm' (minutos) e 's' (segundos):

```
num(3, 'h')      → 3
num(3, 'm')      → 180
num(3, 's')      → 10800
```

Resolva este exercício de três formas distintas, usando:

- if-else*;
- switch* com *break*;
- switch* sem *break*.

**NOTA**

→ Supõe-se que o valor do parâmetro *tipo* está sempre correto.

**5.8** *float max3(float x, float y, float w)*

Devolve o maior dos valores  $x$ ,  $y$  e  $w$ .

**5.9** *int impar(int x)*

Devolve *verdade*, se  $x$  for ímpar, e *falso*, caso contrário.

**5.10** *int Entre(int x, int lim\_inf, int lim\_sup)*

Verifica se  $x$  se encontra no intervalo  $\text{lim\_inf} \leq x \leq \text{lim\_sup}$ .

**5.11** Escreva as seguintes funções:

Função	Devolve
<i>int isdigit(int ch)</i>	<i>Verdade</i> , caso <i>ch</i> seja um dígito, e <i>falso</i> , caso contrário
<i>int isalpha(int ch)</i>	<i>Verdade</i> , caso <i>ch</i> seja uma letra do alfabeto, maiúscula ou minúscula, e <i>falso</i> , caso contrário
<i>int isalnum(int ch)</i>	<i>Verdade</i> , caso <i>ch</i> seja um dígito ou uma letra do alfabeto, e <i>falso</i> , caso contrário
<i>int islower(int ch)</i>	<i>Verdade</i> , caso <i>ch</i> seja uma letra minúscula, e <i>falso</i> , caso contrário
<i>int isupper(int ch)</i>	<i>Verdade</i> , caso <i>ch</i> seja uma letra maiúscula, e <i>falso</i> , caso contrário

(continua)

(continuação)

Função	Devolve
<b><i>int isspace(int ch)</i></b>	Verdade, caso <i>ch</i> seja um espaço ou um <i>tab</i> , e falso, caso contrário
<b><i>int tolower(int ch)</i></b>	Devolve o valor do carácter <i>ch</i> transformado em minúsculas
<b><i>int toupper(int ch)</i></b>	Devolve o valor do carácter <i>ch</i> transformado em maiúsculas

**NOTA**

→ Para ter acesso às funções apresentadas no Exercício resolvido 5.11 deste capítulo, bastará incluir no início do seu programa a seguinte linha:

```
#include <ctype.h> // Funções sobre o tipo char (ctype = char type)
```

**5.12 *int isSquare(int x, int y)***

Devolve um valor lógico que indica se *x* é ou não igual a  $y^2$ .

**5.13 *int Minus(int valor)***

Devolve o *valor* recebido sempre como número negativo:

```
Minus(10) → -10
Minus(-10) → -10
```

**5.14 *int isSpecial(int x)***

Devolve um valor lógico que indica se o dobro de *x* é igual a  $x^2$ .

**5.15 *int Cubo(int x)***

Devolve o valor de  $x^3$ .

**5.16 *int isVowel(char ch)***

Verifica se *ch* é uma das vogais do alfabeto (em minúsculas ou maiúsculas).

**5.17 *double Inverso(int x)***

Devolve o valor inverso de *x* ( $1/x$ ) ou zero:

```
Inverso(0) → 0
Inverso(5) → 0.2
```

**5.18 *void Triangulo(int n)***

Desenha no ecrã um triângulo encostado à direita, com uma base de *n* asteriscos:

```
Triangulo(3)      Triangulo(5)
*                *
**               **
***              ***
                 ****
                 *****
```

Aceda aqui às soluções



# ORDENAÇÃO (*SORTING*)

## OBJETIVOS

- Algoritmo *Bubble Sort*
- Algoritmo *Selection Sort*
- Algoritmo *Insertion Sort*
- Algoritmo *Merge Sort*
- Algoritmo *Quick Sort*
- Ordenação crescente e decrescente
- Ordenação de estruturas complexas
- Ordenação indireta de *arrays*

## 20.1 INTRODUÇÃO

Neste capítulo, iremos estudar um assunto de primordial importância na ciência da computação: a ordenação.

Poderá parecer estranho que, no século XXI, se estudem algoritmos de ordenação, mas, na verdade, a ordenação é um processo de tal maneira importante que a própria palavra “computador” existe, em algumas línguas, com o sentido de “ordenador”<sup>49</sup>, pois a tarefa de ordenar seja o que for sempre foi demasiado dispendiosa, e os computadores foram originalmente usados, de forma mais ou menos generalizada, para efetuar tarefas de ordenação, daí a associação ao nome “ordenador”.

Embora o conceito de ordenação seja simples e se consiga implementar e resolver de forma quase natural, não deixa de ser surpreendente que existam tantos algoritmos de ordenação para resolver um problema que, aparentemente, é tão simples e que consiste na colocação de um conjunto de elementos por uma determinada ordem.

Ao longo deste capítulo, iremos estudar os principais algoritmos de ordenação. O resultado final da execução de cada um deles terá de ser o mesmo, isto é, todos deverão apresentar o conjunto de dados originais mas já ordenados. Contudo, a forma como cada um dos algoritmos o faz e a estratégia adotada por cada um deles podem levar a enormes ganhos ou custos em termos de tempos de execução, memória consumida, complexidade, etc.

<sup>49</sup> Por exemplo, *ordenador*, em espanhol, e *ordinateur*, em francês.

Com base no estudo dos vários algoritmos e na análise de resultados, será perceptível que poderá ser muito vantajoso optar por usar um algoritmo em particular para resolver um problema específico. Perante a necessidade de ordenar um conjunto de valores, pode ser vantajoso ou não optar por uma determinada estratégia, em detrimento de outra.

Ao longo dos capítulos deste livro, podemos verificar que existe uma relação estreita entre o tipo de estrutura onde os dados são armazenados e a forma como os algoritmos que os manipulam são implementados.

Existe, assim, geralmente, uma estreita dependência entre a estrutura de dados e os algoritmos que a processam, por norma nos dois sentidos, isto é, a estrutura influencia o algoritmo e o algoritmo também pode influenciar a estrutura de dados.

Neste capítulo e no Capítulo 21, iremos, por momentos, esquecer-nos desta forte ligação. Vamos concentrar-nos no estudo de diferentes algoritmos que permitem a resolução de problemas específicos de programação: ordenação; e pesquisa. A componente de estruturas de dados terá uma menor importância, pois todos os casos a estudar irão basear-se num conjunto de elementos armazenados sequencialmente num *array*.

Tendo sempre como base a mesma estrutura de dados, iremos verificar que é a qualidade do algoritmo, ou seja, a qualidade da estratégia de ataque e resolução do problema, que irá acentuar a diferença entre melhores ou piores resultados nos vários casos a estudar.

Neste capítulo, iremos sempre partir de um *array* simples com dados para ordenar e obter como resultado um *array* – que, em princípio, será o mesmo – com os dados originais já ordenados.

## 20.2 PORQUÊ ORDENAR?

Ordenar corresponde ao processo de rearranjo de um conjunto de elementos colocando-os numa ordem específica.

O objetivo da ordenação de elementos é, normalmente, o de facilitar e tornar mais célere o processo posterior de pesquisa desses elementos.

Existem muitas razões pelas quais poderemos ter de ordenar dados numa aplicação, entre as quais as seguintes:

- Os dados necessários a uma aplicação podem ser obtidos do exterior e pode não existir a garantia de que estes dados estejam ordenados, tal como pode ser requerido pela aplicação;
- A maior parte das aplicações informáticas na fase de introdução de dados necessita de apresentar listas de valores para o utilizador escolher. É aconselhável que os dados presentes nessas listas estejam ordenados por algum critério lógico, de modo a facilitar a vida ao utilizador;
- Ainda que os dados de *input* de uma aplicação estejam ou não ordenados, o utilizador poderá querer que os diversos *outputs* gerados pela aplicação reflitam a aplicação de diferentes critérios e formas de ordenação dos dados de saída, sendo, por isso, necessário que a própria aplicação trate do processo de ordenação dos dados para a geração correta dos *outputs*;

```

104:
105: void hash_print(HASH_TABLE *h)
106: {
107:     printf("Imprimir a HASH TABLE (%d) registros\n",
108:           hash_count(h));
109:
110:     for (int i=0; i<HASH_SIZE; i++)
111:         item_print(i, h->arr[i], 1);
112: }

```

A aplicação de teste recebe dois argumentos na linha de comando:

- **argv[1]** – Nome do ficheiro de dados;
- **argv[2]** – Função de *probing* a usar: **(l)inear**|(q)uadratic|(p)rime].

```

1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <ctype.h>
4: #include "hash.h"
5:
6: #define LEN(x) (sizeof(x)/sizeof(x[0]))
7:

```

#### PROG2802.C

Foram definidas três constantes que irão identificar a função de *probing* a usar:

```

8: #define LINEAR    'l'
9: #define QUADRATIC 'q'
10: #define PRIME    'p'
11:
12: int load_data(ITEM arr[], char *filename)
13: {
14:     FILE *fp = fopen(filename, "rt");
15:     if (fp==NULL)
16:     {
17:         fprintf(stderr, "Impossível abrir [%s]\n", filename);
18:         exit(2);
19:     }
20:
21:     int index=0;
22:     while (fscanf(fp, "%s %s", arr[index].name,
23:                 arr[index].phone)==2)
24:         index++;
25:     fclose(fp);
26:
27:     return index; // count
28: }
29:
30: void print_array(ITEM arr[], int count)
31: {
32:     printf("Imprimir o ARRAY (%d registros)\n", count);
33:     for (int i=0; i<count; i++)
34:         item_print(i, arr+i, 1);
35: }
36:
37: void load_hash_table(HASH_TABLE *h, ITEM arr[], int count)
38: {
39:     for (int i=0; i<count; i++)
40:         hash_add(h, arr+i);
41: }
42:

```



*Linguagem C*, da autoria de Luís Damas, é um clássico na literatura técnica em português sobre programação.

Amplamente utilizado como uma obra de referência e estudo, especialmente por estudantes e profissionais em áreas como Engenharia e Ciência da Computação, este é um livro essencial que cobre os fundamentos da linguagem C de forma clara e estruturada, abordando **desde os conceitos básicos até tópicos mais avançados**, incluindo inúmeros exercícios práticos, com acesso às resoluções, e exemplos que ajudam a consolidar a aprendizagem da matéria.

Esta nova edição mantém a apresentação detalhada da linguagem C, refletindo a evolução da linguagem e a forma como deve ser usada na atualidade, e conta com **12 novos capítulos com matérias avançadas** diretamente relacionadas com técnicas de programação ou com a implementação de estruturas de dados clássicas.

Baseado na longa experiência de programação e ensino do seu autor, este livro é destinado a estudantes de programação e a programadores que pretendem reforçar o seu conhecimento e as suas competências nas técnicas de desenvolvimento de *software* e na implementação de estruturas de dados usando a linguagem que serviu de base a quase todas as linguagens modernas e que poderão aplicar estes conhecimentos às linguagens que usam atualmente.

**Aprende com o *Linguagem C*?**

**Então atualize os seus métodos e desenvolva os seus conhecimentos com a 25.ª edição do livro de programação mais vendido em Portugal!**

# Linguagem C

25.ª Edição

Atualizada e Aumentada

## O que pode encontrar neste livro:

- Apresentação e evolução da linguagem C
- Estruturas de dados (um livro completo 2 em 1!)
- 12 novos capítulos
- Mais de 370 programas completos
- Mais de 1000 funções implementadas
- Mais de 70 páginas de soluções dos exercícios propostas (com acesso via código QR)
- Mais de 35 000 linhas de código

## Porquê programar em C?

