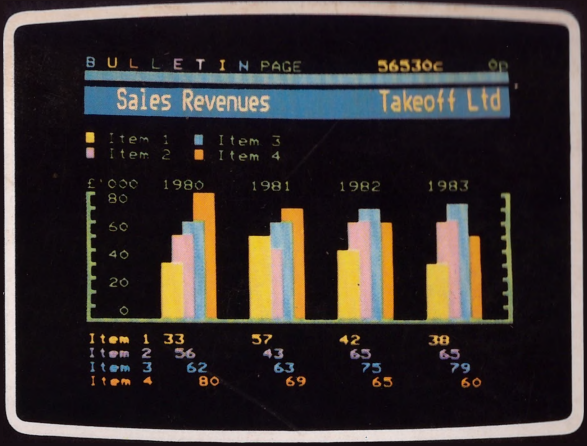
JOHN SHELLEY



ABC DA PROGRAMAÇÃO DE COMPUTADORES

TEMPOS LIVRES

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, por *Petar Trifunovitch e Sava Vukovitch*
2. Fisher / Spasski — Campeonato Mundial de Xadrez de 1972, por *Petar Trifunovitch*
4. ABC do Bridge, por *Pierre Jais e H. Lahana*
5. Guia Prático de Fotografia, por *W. D. Emanuel*
6. ABC do Judo, por *E. J. Harrison*
7. Como Fazer Cinema, por *Paul Petzold*
8. Bridge Moderno, por *Pierre Jais e H. Lahana*
9. Fotografia — Técnicas e Truques I, por *Edwin Smith*
10. Estilos de Mobiliário, por *A. Aussel*
11. Fotografia — Técnicas e Truques II, por *Edwin Smith*
12. A Pesca Submarina, por *António Ribera*
13. Teoria dos Finais de Partida, por *Yuri Averbach*
14. Aprenda Rádio, por *B. Fighiera*
15. Guia do Cão, por *Louise Laliberté-Robert e Jean-Pierre Robert*
16. ABC do Aquário, por *Anthony Evans*
17. Iniciação à Electricidade e à Electrónica, por *Fernand Huré*
18. Os Transístores, por *Fernand Huré*
19. Karaté I, por *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, por *C. Péricone*
21. Construa o seu Receptor, por *B. Fighiera*
22. Montagens Electrónicas, por *B. Fighiera*
23. O Berbequim Eléctrico, por *Villy Dreier*
24. Cactos, por *J. Nilaus Jensen*
25. Iniciação à Alta Fidelidade, por *Peter Turner*
26. O Aquário de Água Doce, por *Paulo de Oliveira*
27. ABC do Ténis, por *Fonseca Vaz*
28. Karaté II, por *A. Pflüger*
29. ABC da Criação de Canários, por *Curt Af Enehjelm*
30. Ginástica Feminina, por *Sonja Helmer Jensen*
31. Cartomancia, por *Rhea Koch*
32. Calculadoras Electrónicas de Bolso, por *E. Dam Ravn*
33. O Pastor Alemão, por *Gilles Legrand*
34. Xadrez — Teoria do Meio Jogo, por *I. Bondarevsky*
35. Manual do Super 8 - I, por *Myron A. Matzkin*
36. ABC da Criação de Periquitos, por *Cyril H. Rogers*
37. O Livro dos Gatos, por *Bärbel Gerber e Horst Bielfeld*
38. Manual do Super 8 - II, por *Myron A. Matzkin*
39. ABC do Mergulho Desportivo, por *Walter Mattes*
40. Circuitos Integrados / Aplicações Práticas, por *F. Bergtold*
41. A Apicultura, por *H. R. C. Riches*
42. ABC do Cultivo das Plantas, por *H. G. Witham Fogg*
43. ABC da Criação de Pombos, por *Kai R. Dahl*
44. Construção de Caixas Acústicas de Alta Fidelidade, por *R. Brault*
45. Raças de Canários, por *Klaus Speicher*
46. Jogos de Cartas, por *Graciano Dolma*

47. Cocker Spaniels, por *H. S. Lloyd*
48. ABC da Caça, por *Fabián Abril*
49. Aprenda Televisão, por *Gordon J. King*
50. Iniciação à Pesca, por *Juan Nadal*
51. Basquetebol, por *Marius Norregard*
52. Cães de Caça, por *Santiago Pons*
53. Aprenda Electrónica, por *T. L. Squires e C. M. Deason*
54. A Avicultura, por *Jim Worthington*
55. A Produção de Coelhos, por *Surdeau e R. Henaff*
56. ABC dos Computadores, por *T. F. Fry*
57. Natação para Crianças, por *John Idorn*
58. O Boxer, por *Anni Mortensen*
59. Voleibol, por *Ole Hansen e Per-Göran Persson*
60. Iniciação à Vela, por *Donald Law*
61. ABC da Filatelia, por *Jacqueline Caurat*
62. A Pesca à Beira-Mar, por *J.-M. Moëlle e B. Doyen*
63. Enxerto das Árvores de Fruto, por *Alejo Rigau*
64. A Cultura do Morangueiro, por *Luís Alsina Grau*
65. Emissores-Receptores (Walkies-Talkies), por *P. Duranton*
66. Iniciação à Fotoelectrónica, por *Heinz Richter*
67. Doces e Conservas de Fruta, por *Robin Howe*
68. A Criação de Hamsters, por *C. F. Snow*
69. A Criação de Porcos, por *Roy Genders*
70. Calendário do Horticultor, por *Luís Alsina Grau*
71. Jogos Electrónicos, por *F. G. Rayer*
72. Cultivo de Cogumelos e Trufas, por *Alejo Rigau*
73. Aprenda Televisão a Cores, por *Gordon J. King*
74. Gravação em Fita Magnética, por *Ian R. Sinclair*
75. Poda de Árvores e Arbustos, por *Roy Genders*
76. Como Treinar o Seu Cão, por *E. Fitch Daglish*
77. Instrumentos de Medida e Verificação, por *Heinrich Stöckle*
78. A Criação de Caracóis, por *Matías Josa*
79. Rádio-Fundamentos e Técnica, por *Gordon J. King*
80. Como Fazer Gelados, por *Sylvie Thiébault*
81. Iniciação à Jardinagem, por *Noel Clarasó*
82. A Congelação dos Alimentos, por *Suzanne Lapointe*
83. Windsurf — Prancha à Vela, por *Ernstfried Prade*
84. Raças de Cães, por *O. Hasselfeldt*
85. Rummy e Canasta, por *Claus D. Grupp*
86. A Encadernação, *Annie Persuy*
87. Aprenda Electricidade, por *Heinz Richter*
88. Taxidermia, Embalsamamento de Aves e Mamíferos, por *Harry Hjortaa*
89. Jogging — Correr para Manter a Forma, por *Werner Sonntag*
90. ABC da Cozinha Chinesa, por *Sonya Richmond*
91. Jogos T. V., por *C. Tavernier*
92. Amplificadores de Som, por *Richard Zierl*
93. O Livro do Poker, por *Claus D. Grupp*
94. Aprenda a Desenhar, por *Rose-Marie de Prémont e Nicole Philippi*

95. O Minitrampolim na Escola, por *Sonja Helmer Jensen e Klaus Dano*
96. Jogos de Luzes e Efeitos Sonoros para Guitarras, por *B. Fighiera*
97. O Cultivo do Tomate, por *Louis N. Flawn*
98. Pilhas Solares, por *F. Juster*
99. A Criação Doméstica de Coelhos, por *C. F. Snow*
100. Iniciação ao Futebol, por *Wieland Männle e Heinz Arnold*
101. Horóscopos Chineses, por *Georg Haddenbach*
102. Guia Prático de Marcenaria, por *Charles H. Hayward*
103. Andebol, por *Fritz e Peter Hattig*
104. Dispositivos Anti-Roubo, por *H. Schreiber*
105. Perus, Pintadas e Codornizes, por *Jerome Sauze*
106. Crepes — Doces e Salgados, por *Florence Arzel*
107. Aperitivos e Entradas, por *Myrette Tiano*
108. Ténis de Mesa, por *Leslie Woollard*
109. Aprenda Surf, por *R. Abbott e M. Baker*
110. Futebol — Técnica e Táctica, por *Kurt Lavall*
111. A Vaca Leiteira, por *Colin T. Whittemore*
112. O Cubo Mágico, por *Josef Trajber*
113. O Perdigueiro Português, por *José M. Correia*
114. Pizzas e Massas à Italiana, por *Marieanne Ränk*
115. O Cubo Para Quem Já o Faz, por *Josef Trajber*
116. A Pirâmide Mágica, A Torre, O Barril do Diabo, por *M. Mrowka-W. J. Weber*
117. Gansos e Patos, por *Marie Mourthe*
118. Iniciação ao Kung Fu, por *A. P. Harrington*
120. O Livro da Fortuna, por *Douglas Hill*
121. Construção de um Alimentador de Corrente, por *Waldemar Baitingu*
122. Hóquei em Patins, por *Francisco Velasco*
123. Técnicas de Tiro, por *Anton Kovacic*
124. Aprenda a Tricotar, por *Uta Mix*
125. ABC da Patinagem, por *Christa-Maria e Richard Kerler*
126. A Pesca e os seus Segredos, por *Armand Deschamps*
127. O Osciloscópio, por *R. Rateau*
128. Guia Prático da Banda do Cidadão, por *T. M. Normand*
129. Sumos e Batidos, por *Manfred Dondersk*i.
130. Introdução à Programação de Microcomputadores, por *Peter C. Sanderson*
131. Aprenda Croché, por *Uta Mix*
132. ABC do Microprocessador, por *P. Mélusson*
133. Guia Prático de Basic, por *Roger Hunt*
134. Introdução à Electrónica Digital, por *Ian R. Sinclair*
135. ABC do Vídeo, por *David Matthewson*
136. Fotografia em Movimento, por *Don Morley*
137. Guia Prático de Cobol, por *Ray Welland*
138. Fotografia a Pequena Distância, por *Sidney F. Ray*
139. Guia Moderno da Canaricultura, por *Manuel Joaquim Gonçalves*
140. Minielectrónica para Amadores, por *Heinz Richter*
141. ABC da Programação de Computadores, por *John Shelley*

JOHN SHELLEY

A B C
DA
PROGRAMAÇÃO
DE COMPUTADORES

EDITORIAL PRESENÇA

Título original:
POCKET GUIDE TO PROGRAMMING

Tradução de Conceição Jardim e Eduardo Nogueira
Fotografia da capa gentilmente cedida
pela ICL Computadores Limitada

EDITORIAL PRESENÇA, LDA.
Rua Augusto Gil, 35-A
1000 LISBOA

I

O SISTEMA COMPUTADOR

A primeira pessoa a conceber a ideia de um computador foi Charles Babbage, em 1833. Chamou-lhe «Motor Analítico». Babbage, um professor de matemática em Cambridge (Inglaterra), queria construir uma máquina capaz de calcular *qualquer* equação. Tal máquina nunca foi construída porque faltava uma coisa no mundo do século dezanove — a electrónica... Charles Babbage estava portanto limitado pela tecnologia mecânica — rodas dentadas, alavancas. Conseguiu no entanto realizar uma parte do seu Motor Analítico, que ainda hoje pode ser vista no Museu da Ciência em Londres.

Coube a Howard Aiken apresentar ao mundo, em Maio de 1944, o primeiro computador eficaz. Conseguiu fazê-lo porque dispunha já da electrónica — a tecnologia mais apropriada neste caso — sob a forma de transístores, resistências e condensadores. O computador moderno consiste em milhares ou mesmo milhões de componentes destes tipos. Felizmente não é necessário compreender pormenorizadamente o funcionamento ou constituição destes para podermos utilizar o computador. A única característica dos componentes electrónicos que

aqui nos interessa é o facto de serem dispositivos que apenas podem assumir dois estados eléctricos diferentes.

Dispositivos de dois estados

Um dispositivo de dois estados é semelhante em princípio a uma vulgar lâmpada eléctrica, que num dado momento apenas se pode encontrar num de dois estados, apagada ou acesa. Os componentes dos computadores são dispositivos de dois estados porque em qualquer instante considerado podem permitir ou não a passagem de uma carga eléctrica. As campainhas eléctricas, os telefones (que tocam ou não...) e os interruptores vulgares são outros tantos exemplos.

Os dois estados de tensão dos componentes de um computador podem ser por exemplo de + 5 V e O V, um nível de tensão positivo e outro negativo, ou ainda dois níveis positivos. Cada fabricante de computadores escolhe o método mais conveniente para a concepção global da sua máquina.

Os profissionais de computadores, exceptuando aqueles que concebem os equipamentos, não gostam de falar de correntes eléctricas e de níveis de tensão, preferindo indicar a presença ou ausência de uma carga eléctrica de um modo diferente designado por «representação binária».

O sistema binário

A sílaba *Bi,* significando dois em latim, indica que o sistema binário é um sistema de números baseados no 2, isto é, possuindo apenas dois algarismos. Como todos os

sistemas de números incluem o zero como um dos seus algarismos, daí decorre que o sistema binário só emprega um outro algarismo além de 0, a saber, o 1.

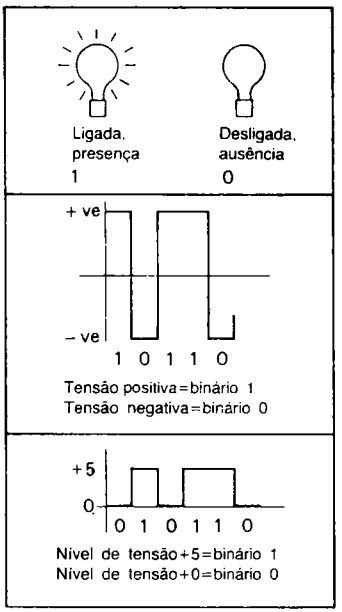


Figura 1.1

Este sistema constitui um modo muito conveniente de representar os dois estados que qualquer componente electrónico pode assumir. O sistema decimal (base dez) a que estamos mais habituados possui dez algarismos diferentes, 0, 1, ..., 9. É possível representar qualquer quantidade usando apenas estes algarismos. O sistema binário permite do mesmo modo representar qualquer número, usando no entanto apenas os dois algarismos 0 e 1.

253_{10} número decimal
11111101_2 número equivalente em binário

Figura 1.2

Concepção de um computador

Uma maneira de estudar a estrutura de um computador consiste em observar o modo como este resolve um problema. O problema que iremos considerar é muito simples — descobrir o maior número positivo numa dada lista de números (ver a figura 1.3).

Primeiramente imaginemos que este problema é dado a um ser humano, por exemplo a uma criança, a fim de verificar a sua aptidão em aritmética. A criança deve conhecer dois elementos para poder resolver o problema em causa: uma *instrução* (descobrir o número maior) e a *lista* dos números. Utilizando métodos subtis de funcionamento do cérebro e dos olhos, o ser humano pode muito rapidamente descobrir o número mais elevado. Parece prolongar-se um pouco mais para a esquerda do que os outros.

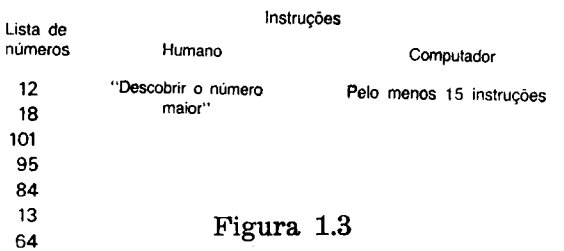


Figura 1.3

O computador requer igualmente dois elementos para poder resolver este problema: o conjunto de instruções (chamado *programa)* e a lista de números (chamados *dados*). A diferença entre a instrução dada à criança e as

que são introduzidas no computador consiste no facto de este último necessitar de mais instruções, e de estas serem muito mais pormenorizadas. Para resolver este problema, por exemplo, necessita de cerca de quinze instruções...

O programa é portanto constituído por um grupo de instruções que quando são obedecidas (executadas) pelo computador actuarão sobre dados (neste caso a lista de números).

O ser humano necessita de uma memória para poder guardar e recordar a instrução. O mesmo acontece com o computador. Assim, um dos componentes essenciais de um computador é a sua *memória*. Mas a memória é apenas um local onde são guardadas informações, como por exemplo as instruções do programa e os dados; não é capaz de realizar quaisquer cálculos. É necessário um outro componente para realizar operações aritméticas, e nos computadores modernos esta unidade é designada por *unidade aritmética*. Esta é ainda capaz de comparar dois números e descobrir qual é o maior ou se ambos são iguais. Como também sabe executar funções lógicas como OR («ou»), AND («e») e NOT («negação lógica»), o seu nome completo é *unidade aritmética/lógica* e é representada muitas vezes pelas iniciais do seu nome em inglês — ALU.

Mas como penetram nesta unidade os números vindos da memória? É esta a função de uma terceira unidade (aliás, a última), responsável por *comandar* ou enviar os números correctos para a ALU, por decidir qual a operação aritmética ou a comparação que deve ser realizada por esta, e por colocar novamente na unidade de memória o resultado obtido. Esta terceira unidade é designada

por «unidade de comando» (CU). Estas são as únicas partes integrantes de qualquer computador, e são conhecidas colectivamente pela designação geral de *unidade de processamento central* (CPU). A figura 1.4 mostra esquematicamente estes três componentes. As linhas a tracejado mostram os circuitos de comando vindos da CU, e as linhas a cheio mostram a passagem de informações entre os vários componentes.

Dispositivos de entrada/saída

O objectivo de qualquer *dispositivo de entrada* consiste em enviar informações (instruções de programa e dados) para a unidade de memória, processo que se encontra também sob o comando da CU. O dispositivo de entrada converte os caracteres que usamos quotidianamente (letras do alfabeto, algarismos 0 a 9, símbolos especiais para as operações matemáticas, e símbolos de pontuação) em padrões binários, a fim de poderem ser armazenados no interior da memória do computador. São ainda necessários *dispositivos de saída,* que realizam a função oposta à dos de entrada; isto é, traduzem a informação binária contida na memória para uma forma que o ser humano possa compreender facilmente.

Existem vários tipos de dispositivos de entrada/saída (dispositivos I/O, de «input/output»): leitores de cartões perfurados; teclados semelhantes a máquinas de escrever; impressoras capazes de fornecerem uma cópia em papel; unidades de apresentação visual (VDU's ou «video display units»); etc. Como estas unidades não fazem parte da CPU, são muitas vezes designadas por *dispositivos periféricos* do computador.

Memória limitada

Como a unidade de memória de um computador é bastante cara, é frequentemente limitada em tamanho a fim de permitir a redução dos custos globais do equipamento.

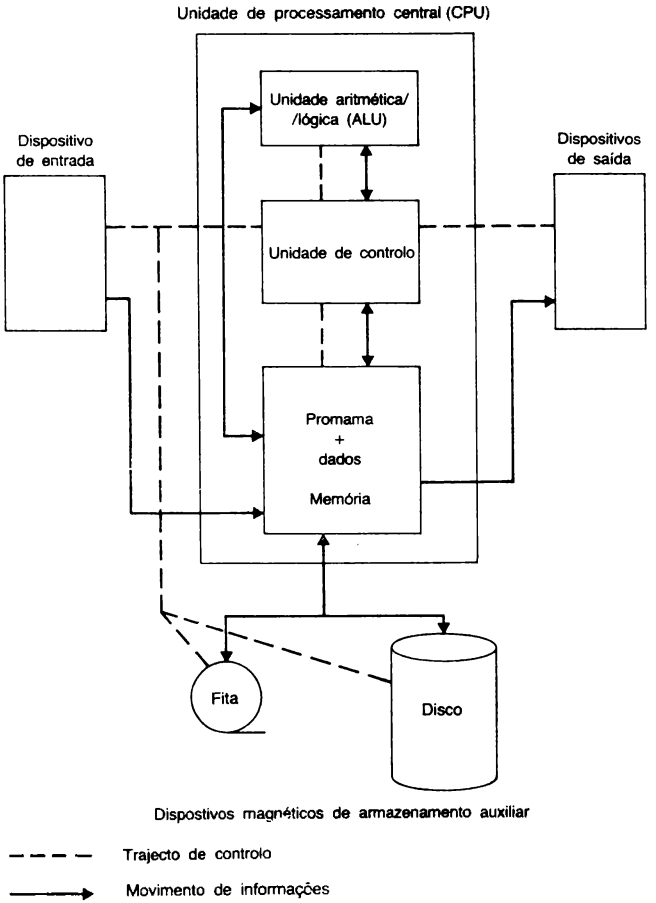


Fig. 1.4

Torna-se assim necessário recorrer a unidades de armazenamento auxiliares que suplementam a memória principal ou interna. É portanto em materiais magnetizáveis, como as fitas ou discos magnéticos, que todos os programas e dados usados pelos computadores são guardados permanentemente. É porém necessário que todas estas informações sejam enviadas destes dispositivos para a memória central antes de a CPU as poder utilizar.

Um sistema computador

Um sistema computador (ou processador) consiste portanto nos componentes seguintes — a CPU, dispositivos I/O e dispositivos auxiliares de armazenamento ou registo de informações. Chama-se ao conjunto de tudo isto o *hardware* do sistema, o qual respeita portanto àquilo que se pode ver e tocar quando se observa uma instalação computadorizada. No entanto, são também necessários programas de computador (*software*) para transformar as unidades «hardware» num sistema funcional. Talvez seja aqui útil fazer uma analogia. Um táxi que se encontre parado numa rua assemelha-se ao hardware de um computador. Por si mesmo nada faz; para poder funcionar é necessário um condutor que leve o seu «hardware» a funcionar como uma unidade completa. É este o objectivo dos programas de computador, ou do seu «software». São concebidos para fazer funcionar todas as unidades em conjunto de um modo harmonioso. No entanto, é necessário ao táxi e ao seu condutor disporem de um passageiro para se tornarem úteis. Em processamento, o «passageiro» é o programa de apli-

cação, por exemplo a execução de uma lista de pagamentos, que é escrito pelo utilizador. Nestas condições poderemos considerar três aspectos diferentes num sistema computador: o hardware (o táxi), os programas de «condução» do sistema (o condutor do táxi) e os programas de aplicação (o passageiro).

Todos os computadores, sejam grandes ou pequenos e qualquer que seja o seu custo, são conceptualmente semelhantes ao esquema da figura 1.4. Todas as disparidades em custo são reflectidas nas dimensões da unidade de memória e na velocidade a que a ALU pode funcionar.

Em síntese, existem quatro operações básicas que qualquer computador é capaz de realizar:

1) operações de entrada e saída;
2) operações aritméticas;
3) operações lógicas e comparações (ver capítulo 2);
4) movimento de dados entre os vários componentes.

A arte de programar consiste em dividir qualquer problema considerado em muitos (talvez milhares) pequenos passos, usando uma ou mais destas quatro operações básicas.

Codificação

A memória de um computador consiste em muitas células, um pouco como acontece numa colmeia, onde é possível guardar uma instrução ou um dado. O termo formal usado para designar estas células é *posição* ou *palavra*. O número de células é frequentemente um múl-

tiplo do «quilo» binário, como se usa noutras unidades (quilograma, quilómetro); mas neste caso o quilo, em vez de indicar a multiplicação da unidade básica por mil, multiplica-a de facto por 1024: 1 K indica portanto 1024 posições de memória, 4K indica 4×1024 (4096), etc. Cada palavra conterá um número específico de componentes electrónicos, cada um deles capaz de armazenar em qualquer instante o binário um ou o binário zero (como uma fiada de lâmpadas, onde cada uma pode estar acesa ou apagada num dado momento). O computador é concebido de tal modo que o número de algarismos binários (ou *bits*, de *bi*nary digi*ts* em inglês) em cada posição pode representar uma instrução, um valor ou um caracter de texto (em alguns casos é até possível guardar numa palavra mais do que um caracter).

Porque são tão úteis os computadores?

Em Dezembro de 1979, Stan Barret foi o primeiro a ultrapassar a barreira do som em terra a conduzir a cerca de 1190 km por hora, o que equivale a 330,6 m/s. Mas a velocidade a que os sinais se deslocam no interior da CPU é quase um milhão de vezes maior: viajam aproximadamente à velocidade da luz, isto é, a 300 000 km/s!

É esta velocidade que permite ao computador realizar muitos milhares (e até milhões, em máquinas maiores) de cálculos por segundo. Para quantificar as suas velocidades temos de falar em termos de microssegundos (milionésimos de segundo) ou até de nanossegundos (bilionésimos de segundo).

Quando recordamos que essencialmente os computadores apenas realizam operações aritméticas, de comparação, lógicas e de movimento de dados, apercebemo-nos da importância desta grande velocidade de execução.

Como terceira característica, os computadores, ao contrário dos frágeis seres humanos, não se aborrecem nem perdem concentração quando executam tarefas repetitivas. Se um computador deve processar um milhão de números, calcula o primeiro e o último com igual «vontade».

É certo que os *mass media* gostam de pôr em relevo todos os «erros» produzidos pelos computadores, do tipo dos avisos de «Pagamento de 0$00 no prazo de trinta dias, sob pena de execução fiscal», etc. No entanto, o computador é apenas capaz de fazer aquilo que lhe dizem. Se os seres humanos que conceberam uma dada aplicação cometeram um erro, por exemplo, nas entradas de clientes, é fácil lançar as culpas sobre o computador. Quando correctamente programados, os computadores são no entanto bastante mais rigorosos do que nós. Estas características de velocidade, armazenamento e recuperação de informações, rapidez e rigor são as virtudes do computador que nos levam a depender deles.

II

UMA INTRODUÇÃO À PROGRAMAÇÃO

Uma criança média com sete anos de idade possui um vocabulário de cerca de 2000 palavras, enquanto que uma linguagem de programação como a FORTRAN possui apenas um vocabulário standardizado de 17 palavras. Não é necessário pensarmos muito para descobrirmos a razão disto; já no capítulo anterior se disse que o computador executa apenas quatro operações básicas. Como consequência, uma linguagem de programação requer apenas um vocabulário limitado.

Baseando-nos nestas quatro funções fundamentais, observemos agora algumas das palavras que provavelmente serão encontradas em qualquer linguagem de programação de computadores.

Operações de entrada

Um aspecto que é importante referir quanto à escrita de programas é que se parte sempre do princípio de que o programa já se encontra no interior da memória central e que os dados estão fora dela, prontos a serem in-

troduzidos por qualquer instrução de entrada. É este obviamente o caso quando o programa está a ser executado. Portanto, todas as linguagens devem conter um modo de introduzir dados na memória interna, visto que até os dados se encontrarem na CPU o programa não pode ser executado. As palavras típicas para realizar esta tarefa são READ e INPUT, as quais constituem veículos linguísticos para o transporte dos dados para a memória central.

Dados

Existem essencialmente dois tipos de dados: o texto (ou caracteres) e os números. Os computadores devem ser capazes de fazer uma distinção entre ambos e, consequentemente, quando usamos instruções de leitura (READ) ou entrada (INPUT) devemos fornecer uma informação adicional que indique à máquina qual o tipo de dados que está a «ler». Cada linguagem possui o seu próprio método de realizar esta distinção. O programador noviço deve familiarizar-se com estas regras. Por exemplo, a linguagem de programação BASIC distingue entre ambas utilizando um sinal «$» (designado na gíria «dólar») para indicar as informações textuais (por vezes designadas ainda por *literais*). Assim, INPUT A$ lê texto para a posição A$, enquanto que INPUT A lê um valor numérico para a posição A.

Certas linguagens reconhecem também dois tipos de números, os que possuem vírgula decimal (por exemplo 10,309 ou 1,00) e os que não a possuem (por exemplo 10). Os primeiros são designados números *reais* ou de

vírgula flutuante, enquanto os outros são designados por números *inteiros* (note que apesar de 1,0 e 1 exprimirem o mesmo valor, o primeiro é real e o segundo inteiro). As linguagens simples, como a BASIC, não fazem esta distinção. No entanto, nas linguagens que a fazem, como a FORTRAN e a ALGOL, o programador deve aprender as regras a que obedece esta distinção. A razão disto é que o computador armazena o número inteiro de uma maneira e o real de outra; como resultado, as operações aritméticas só podem ser realizadas sobre um tipo de números de cada vez.

Operações de saída

Depois de executadas operações sobre dados no interior da CPU, de acordo com as instruções do programa, podemos querer ver os resultados dessas operações. Como tais resultados se encontram no interior da CPU num formato binário, é necessário que a linguagem de programação possua termos que permitam a transferência de informações para um dispositivo de saída. As instruções típicas para este fim são WRITE (escrever) e PRINT (imprimir). Tal como acontece no caso das instruções de entrada, é necessário fornecer informações adicionais, juntamente com estas instruções, a fim de indicar o tipo de caracteres que serão apresentados na saída: textos ou números.

Movimento de dados e criação de listagens

Como os números devem ser processados na unidade aritmética/lógica, existem métodos para transferir os

números para a ALU e os resultados para a memória central. Esta actividade é deixada principalmente à unidade de comando, mas deve ser especificada na própria instrução aritmética. Veremos mais adiante como se faz isto.

A possibilidade de agrupar elementos do mesmo tipo em listas de dados é muito importante. Todos fazemos listagens deste tipo na nossa vida quotidiana. Por exemplo, as listas de produtos a comprar na mercearia; a lista de ingredientes necessários para fazer um bolo; a lista dos alunos de uma classe; a lista de pagamentos; a lista de clientes; a lista de valores a somar para obter um total, etc. Em todos estes casos as listas agrupam conjuntos de elementos relacionados entre si — alunos, clientes, ingredientes, etc. É possível criar listagens semelhantes em linguagens de programação, se bem que estas listas recebam a designação mais formal de *arrays*.

Operações aritméticas

No interior da unidade aritmética são realizadas quatro operações aritméticas básicas: adição, subtracção, multiplicação e divisão. É igualmente possível elevar a uma potência (exponenciação, por exemplo 5^4). Algumas linguagens, como a COBOL, obrigam a escrever por extenso as funções aritméticas a executar, por exemplo MULTIPLY, ADD (multiplicar, somar). A maior parte das outras linguagens utilizam no entanto símbolos para indicar as funções aritméticas. Estes símbolos são frequentemente designados por *operadores aritméticos* e variam de uma linguagem para outra. Mas a adição e a

subtracção são normalmente representadas como em álgebra, pelos símbolos + e —. A multiplicação é por vezes indicada por um asterisco (*) e a divisão por um traço oblíquo (/), como acontece em FORTRAN.

Comparações e operações lógicas

Os computadores podem comparar dois números e decidir qual deles é maior ou menor, ou se ambos são iguais. Isto aplica-se igualmente às letras do alfabeto. «B» pode ser considerada como «menor» do que «D» porque se encontra mais próxima do início do alfabeto.

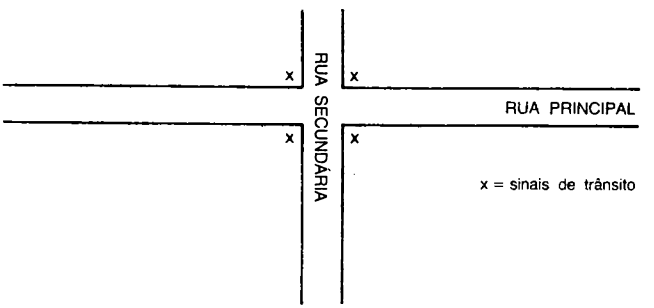


Fig. 2. — Cruzamento de ruas

No entanto, a importância da comparação de dois valores reside em poder decidir entre duas possíveis acções qual deve ser executada. Tomemos como exemplo os sinais luminosos nos cruzamentos de ruas comandados por computador. Uma rua principal é cruzada por uma secundária, e a temporização dos sinais luminosos

comanda a passagem do tráfego. É possível introduzir no sistema uma sequência temporizada, capaz de resolver o problema do tráfego em condições normais. No entanto, se a quantidade de veículos na rua principal exceder 16, por exemplo, é utilizada outra sequência de tempos. É possível fazê-lo facilmente usando comparações, e na prática esta instrução é de facto incluída na maior parte do programa.

Em linguagem normal, o problema pode ser apresentado do seguinte modo:

> Se (IF) o número de veículos é maior do que 16, então (THEN) passar para outra sequência de temporização; se não, manter a sequência normal.

Duas palavras-chave ilustram neste caso a instrução usada: IF (se) e THEN (então). A instrução de comparação é de facto frequentemente chamada «instrução IF... THEN». O problema pode ser resolvido do seguinte modo: à medida que os carros passam sobre um sensor montado a toda a largura da rua, é efectuada uma contagem pelo computador. A intervalos frequentes, por exemplo todos os segundos, esta contagem é comparada com o programa, sendo executada a seguinte instrução:

> IF contagem maior do que 16 THEN mudar sequência de temporização; senão manter a sequência normal.

Vemos aqui como é possível realizar uma comparação entre dois números, a contagem feita e o valor constante de 16, para «decidir» qual de duas acções se irá realizar em seguida.

As instruções INPUT observam o número de carros que passam na rua principal através de sensores apropriados;

As instruções OUTPUT determinam qual das duas sequências de temporização é obedecida pelos sinais de trânsito;

As instruções aritméticas podem acrescentar 1 ao total da contagem sempre que passa um carro sobre o sensor;

As instruções de comparação determinam qual de duas acções se irá executar em seguida.

Consideremos um outro exemplo que ilustre o uso destas palavras.

Problema 1

É necessário dar entrada a dois números, somá-los e observar o resultado.

Para resolver este problema, talvez agarremos imediatamente na nossa máquina de calcular ou numa folha de papel e num lápis em vez de pensarmos em usar um computador... Isto ilustra um aspecto importante do uso de um computador. Estas máquinas só devem ser utilizadas quando o esforço humano a que obrigam é menor do que o exigido por qualquer outro método. No entanto, temos de começar por algum lado e nada nos impede de tentarmos escrever um «programa» para resolver este problema.

Recordando que o programa é sempre escrito partindo do princípio de que virá a encontrar-se *dentro* do computador, estando de fora apenas os dados que irá

utilizar, recorremos a instruções INPUT/READ para introduzir dois números na memória. Temos ainda de indicar os nomes das posições de memória onde esses valores serão guardados (ver adiante). Assim:

READ o primeiro número e chame-lhe PRIMEIRO
READ o segundo número e chame-lhe SEGUNDO
ADD PRIMEIRO e SEGUNDO e chame-lhe RESULTADO
PRINT RESULTADO na saída
END (fim) do programa.

Usando apenas operações de entrada/saída e uma simples operação aritmética, resolvemos o problema através de um conjunto de instruções semelhantes a um programa. Dentro em pouco veremos como os programadores apresentam as suas soluções de um modo esquemático, mas nesta fase começamos a compreender como é possível usar simples palavras-chave (infelizmente inglesas...) para formular o problema. Note por outro lado que o programa em causa pode ser usado repetidamente para quaisquer valores diferentes, porque o uso de nomes permite representar quaisquer valores presentes nas respectivas posições de memória.

Problema 2

Num trabalho de recenseamento, é necessário descobrir o número de jovens (com idade inferior a 18 anos) em toda a população de uma dada área, representando este número sob a forma de uma percentagem da população total.

Para simplificar este problema, consideraremos que o impresso de recenseamento contém todos os detalhes importantes sobre cada pessoa, e que a idade é sempre um número inteiro escrito numa caixa intitulada «idade da pessoa antes de d/m/a» (ou seja, uma certa data). Num sistema manual, quando cada impresso é verificado, é somado um ao total de população e se a idade for inferior a 18 acrescenta-se um ao total de jovens. Depois de ter sido processado o último impresso, é determinada a percentagem de jovens e os três números — total de população, total de jovens e valor percentual destes — são escritos pela máquina num outro impresso.

Um modo de programar este problema de maneira a obter para ele uma solução computadorizada poderá ser:

Pôr a zero o total de população e o total de jovens (ou seja, garantir que estas posições de memória comecem pelo valor zero).

READ idade no primeiro impresso:
ADD 1 ao total da população;
Será a idade inferior (<) a 18?

IF sim: ADD 1 ao total de jovens
em caso contrário, passar ao impresso seguinte.

READ idade no segundo impresso
ADD 1 ao total da população
Será a idade inferior (<) a 18?

IF sim: ADD 1 ao total de jovens
em caso contrário, passar ao impresso seguinte.

Etc.

Como é óbvio, este método simples resulta num programa muito longo, sendo necessário descobrir um outro. Se observarmos as duas séries de instruções indicadas por parêntesis rectos, verificamos que a única diferença entre elas consiste no uso das palavras «primeiro» e «segundo» relativas aos impressos. Como uma das grandes qualidades do computador é a sua capacidade para repetir continuamente séries de instruções, podemos tentar aproveitá-la para este programa. Se alterarmos as palavras «primeiro» e «segundo» para «seguinte» (NEXT), como se mostra adiante, e introduzirmos um mecanismo que repita o mesmo bloco de instruções, bastará este bloco para processar *todos* os impressos.

Pôr os totais de população e de jovens a zero; ou seja, garantir que estas posições de memória começam pelo valor zero.

READ idade no impresso seguinte:
ADD 1 ao total da população
Será a idade inferior (<) a 18?

IF sim: ADD 1 ao total de jovens
de outro modo, continuar o programa.

Repetir a série anterior para todos os impressos.
Determinar percentagem.
PRINT «Total de população — Total de jovens — Percentagem».
Fim do programa.

Usando a palavra «NEXT» («seguinte»), todos os impressos, incluindo o primeiro, são processados do

mesmo modo e pelas mesmas instruções existentes no bloco referido.

A arte da programação consiste em subdividir um problema de uma forma suficientemente rigorosa para permitir obter uma solução por computador. Fazê-lo requer um exercício mental a um nível provavelmente pouco comum de pormenor. Não é difícil nem complicado; apenas pouco habitual, porque na comunicação humana abunda a ambiguidade. Mas não há lugar para esta no trabalho com computadores.

Tudo aquilo que queremos levar o computador a fazer deve ser-lhe ensinado e expresso sob a forma de uma sequência ordenada de acontecimentos. Esta representação de um problema é conhecida pelo nome de *algoritmo*. Só quando temos a certeza de que o algoritmo está correcto é que podemos começar a converter os vários passos da nossa solução em instruções convenientemente expressas em função das regras gramaticais de uma dada linguagem de programação.

Os problemas concebidos para uso em aprendizagem de programação tendem a ser simples e razoavelmente directos, podendo-se dizer que de facto não são problemas no verdadeiro sentido da palavra; quase não necessitam portanto de qualquer planificação prévia. Mas quanto mais complexo for o problema maior é o tempo de que necessitamos para planificar a solução e mais importante se torna fazê-lo correctamente.

Por agora, as «soluções» para os nossos problemas são apenas parciais, não tendo o rigor suficiente para poderem ser traduzidas em instruções de programação. Por exemplo, a instrução «Repetir a série anterior para todos os impressos» é demasiado vaga. De facto, não

corresponde sequer a qualquer das quatro operações básicas. Como poderemos exprimi-la de modo a concordar com uma delas?

Nesta e em muitas outras situações semelhantes devemos começar pelos próprios dados. Em cada impresso de recenseamento existe uma informação importante, a idade da pessoa. Pode-se arranjar as coisas de tal modo que seja acrescentado aos impressos um outro impresso especial contendo uma idade negativa, por exemplo — 99; o computador pode depois, através de uma comparação, verificar se encontra esta idade negativa sempre que lê um impresso. Se não a encontra, parte do princípio de que o impresso em causa é válido e processa o seu conteúdo. Quando a encontra, «sabe» que atingiu o final da sua tarefa, devendo imprimir os totais e calcular a percentagem.

```
7  READ impresso seguinte (NEXT)
   Idade igual a — 99?
      IF sim: GO TO (ir para) a parte de impressão
         do programa (indicada pelo número 10)
      Em caso contrário:
         ADD 1 ao total de população
            Idade < 18?
               IF sim: ADD 1 ao total de jovens
               Caso contrário: GO TO NEXT im-
                presso (número 7)

10 PRINT totais de população e jovens
   Determinar percentagem
```

PRINT percentagem
Fim do programa

Notas

1. Através de um truque deliberado baseado no conhecimento da natureza dos dados e do uso de uma comparação, tornou-se possível levar a máquina a descobrir o momento em que atinge o final do seu trabalho (o último impresso). Note ainda que era necessário ao programador conhecer a natureza dos dados de entrada antes de poder descobrir este método e resolver o problema.
2. Todas as instruções agora incluídas no programa correspondem de facto a uma das quatro operações básicas do computador.
3. Note bem a posição do impresso especial, em último lugar, e o facto de estar colocado (em termos de programa) antes da instrução «ADD 1 ao total de população». É evidentemente esta a sua posição lógica, pois de outro modo a máquina acrescentaria erradamente um último 1 ao total. Este pormenor serve para sublinhar a importância que deve ser atribuída em todos os casos à verificação da sequência lógica das operações a realizar pelo computador.

Se bem que os aspectos acabados de citar possam parecer banais, neles assenta de facto a programação de um computador. Convirá agora concentrar a nossa

atenção num outro aspecto importante: como se torna possível a um programador assegurar que todas as instruções se encontram na ordem adequada, recorrendo a uma representação esquemática designada por fluxograma.

III

OS FLUXOGRAMAS — UM MODO DE RESOLVER PROBLEMAS

Um grave erro dos programadores noviços, e até de muitos profissionais, consiste em estarem tão ansiosos por escreverem um programa que não se preocupam em realizar previamente uma especificação detalhada do que querem fazer. A escrita de um programa deve com efeito ser realizada em três fases:

1. Primeiramente é necessário compreender bem o problema a resolver (o que inclui conhecer a natureza exacta e a estrutura dos dados de entrada);
2. É então necessário especificar as fases do programa, normalmente recorrendo a uma representação esquemática;
3. Depois de o programador ter verificado o diagrama assim construído, pode traduzir (codificar) a representação nele contida para instruções de programa.

Desprezar qualquer das duas fases iniciais apenas serve para atrair desgraças... Infelizmente, no entanto,

o único modo de os principiantes se aperceberem disso é passarem por elas.

Um programa completo que envolve meia dúzia de instruções, como por exemplo a soma de dois números e a impressão do resultado, não obriga à construção de um fluxograma porque dificilmente esquecemos meia dúzia de factos. Mas quando se pede à mente humana que construa um programa com muitas instruções torna-se de facto necessária alguma ajuda. Entre os vários auxiliares a que podemos recorrer encontra-se o fluxograma.

Um fluxograma é uma representação esquemática em duas dimensões do problema que desejamos resolver num computador. Como os computadores podem realizar apenas quatro operações básicas, os símbolos usados nos fluxogramas devem corresponder a essas quatro operações. Muitos departamentos de programação preferem adoptar símbolos próprios, mas os que se seguem são os aceites por instituições como o BSI ou a ECMA (British Standards Institute, European Computing Manufactures Association).

Um exemplo simples ajudará a compreender o modo como estes símbolos são ligados entre si para fornecerem a solução de um problema, por exemplo o do uso de um telefone público. Não se trata de um problema que verdadeiramente deva ser resolvido por um computador, mas é útil para introduzir os símbolos em causa e o uso dos fluxogramas. A figura 3.2 mostra as fases a considerar. Note como os símbolos START (princípio) e STOP (parar) são úteis para nos ajudarem a descobrir o início e o fim da solução concebida. As setas dirigem o leitor do fluxograma para alguns outros símbolos que especificam a operação que deve ser realizada em seguida.

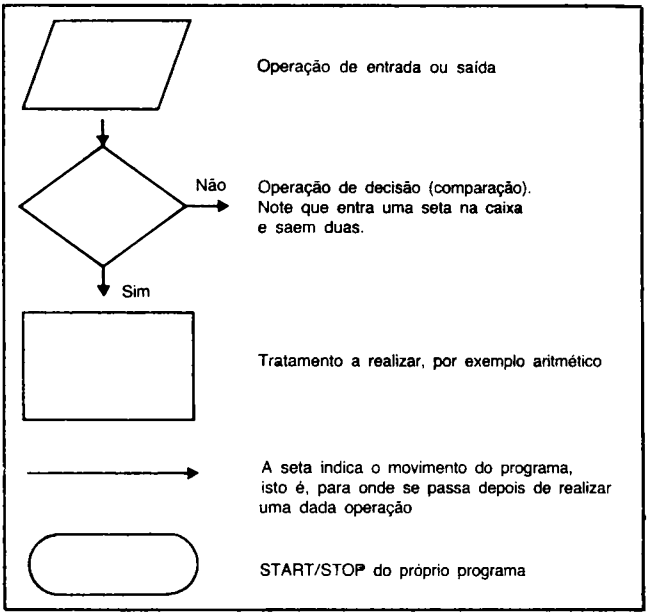


Fig. 3.1

Dentro das formas são usadas palavras da linguagem normal, geralmente de um modo bastante conciso dada a exiguidade do espaço nelas disponível.

Na maior parte dos casos sai de cada símbolo uma única seta, excepto quando está envolvida uma decisão (caso dos losangos). No entanto, note que em qualquer momento só um dos dois possíveis trajectos de saída de um losango é de facto válido. Por exemplo, à pergunta «Sinal de chamada?» só uma resposta pode ser válida em qualquer momento, só podendo portanto ser seguido um trajecto.

Ao resolver um problema, o fluxograma deve ter em conta todas as possibilidades. Se tal não acontecer,

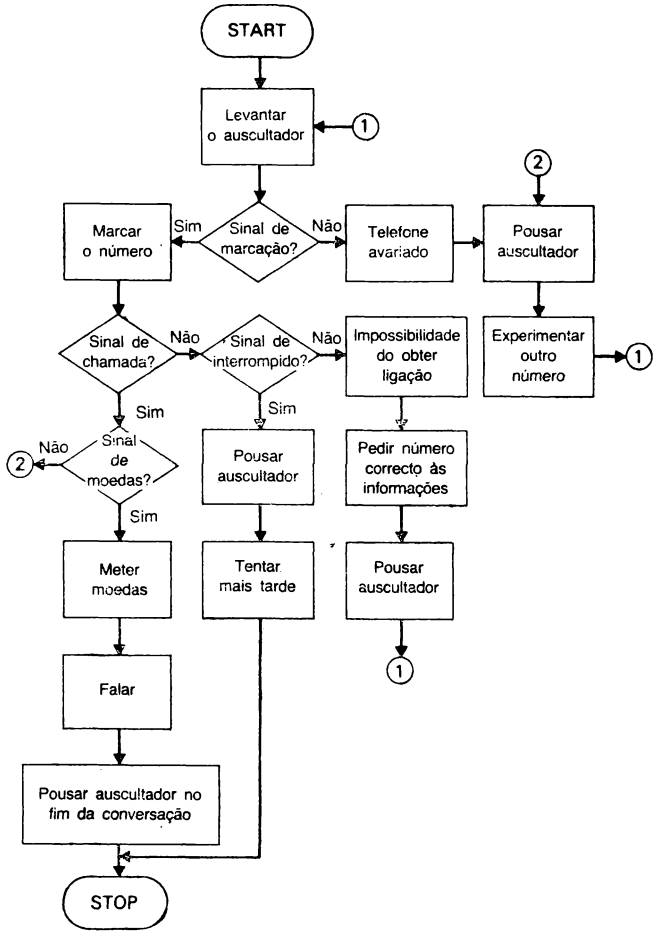


Fig. 3.2

obtém-se um programa tão incompleto como o próprio fluxograma. Depois de o fluxograma estar correcto, é relativamente fácil escrever um programa, em qualquer linguagem escolhida, capaz de realizar os objectivos iniciais.

Como segundo exemplo, neste caso mais apropriado a uma solução por computador, poderemos considerar o problema do controlo de tráfego. A figura 3.3 ilustra a solução.

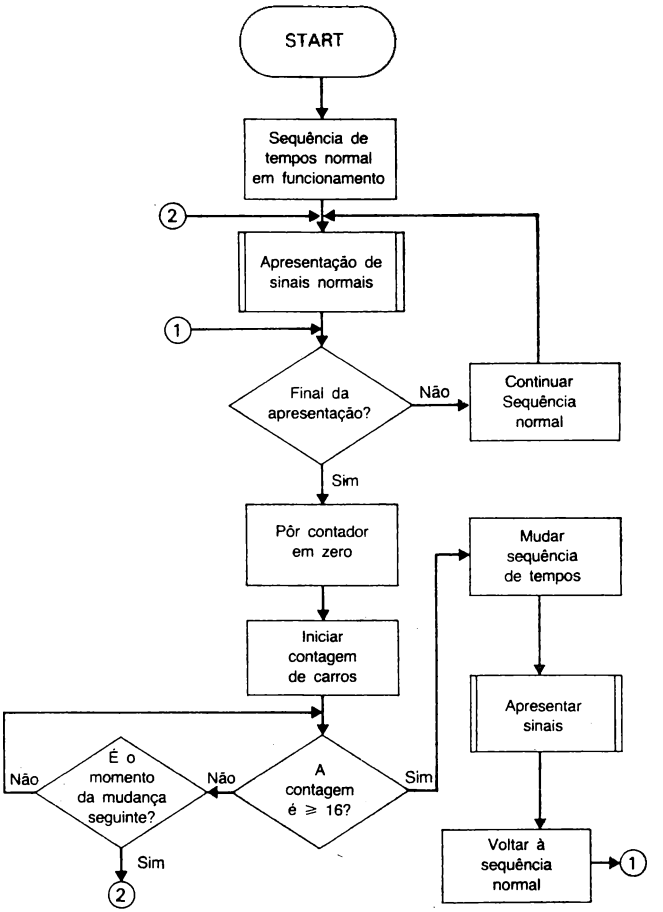
É necessário considerar em separado alguns aspectos da construção de um fluxograma. Em primeiro lugar, nenhuma pessoa conceberá um fluxograma exactamente do mesmo modo que outra. O que é importante, é que o fluxograma construído atinja de facto os objectivos originais. Em segundo lugar, introduzimos algumas caixas com linhas duplas. Este símbolo é frequentemente usado para indicar certos processos que em si mesmos requerem vários passos. Assim, a caixa que representa o processo «Acender sinais» envolverá todos os passos necessários para alterar o padrão dos sinais de trânsito em todas as entradas do cruzamento em função da sequência de tempos escolhida. Numa zona comercial pode ser necessário ordenar uma série de registos de um determinado modo. Neste caso, utiliza-se um outro programa que necessitará de um fluxograma próprio para descrever os vários processos envolvidos no processo de ordenação.

Em terceiro lugar, dir-se-ia que omitimos aqui a entrada dos sensores que informam a máquina sobre o número de veículos em trânsito na rua principal. Esta actividade é estudada na figura 3.4. Este pequeno subprograma será executado paralelamente ao programa

principal, sendo responsável pela actualização da contagem de veículos. Esta «contagem» será verificada pelo programa principal de cada vez que atinge as instruções correspondentes à caixa de decisão que contém a pergunta «N.º de veículos maior que 16?». O leitor poderá ler mais alguma coisa sobre programas e subprogramas na parte do livro que se refere a cada uma das linguagens de programação, mas neste momento bastará saber que o programa principal comanda este subprograma e passa a contagem a zero depois de cada mudança dos sinais.

Se fosse escrito um subprograma mais sofisticado, permitindo obter um sistema de controlo mais completo, os carros que não conseguissem «passar» a tempo quando a luz verde estivesse acesa seriam retidos na contagem, em vez de esta ser passada totalmente a zero. Este capítulo não é dedicado, no entanto, a obter este grau de sofisticação, devendo o problema ser deixado àqueles que se interessam de facto pela concepção de sistemas práticos de controlo de trânsito.

O nosso terceiro exemplo é o já mencionado no capítulo I, isto é, a descoberta do maior número entre vários. Se bem que este pareça ser o problema mais trivial dos três já discutidos, levanta de facto grande parte dos problemas mais importantes a resolver quando se escrevem programas «a sério». Uma verdadeira compreensão de todos estes problemas ajudará o leitor a enfrentar a tarefa da concepção de outros programas. Recordemos a definição do problema. Foi-nos fornecida uma colecção de números positivos desordenados, sendo-nos pedido simultaneamente que construíssemos um fluxograma capaz de descobrir o maior desses números. A figura 3.5 mostra a solução.



NB: Não é necessário STOP porque o processo é contínuo
≥ símbolo de "maior ou igual a"

Fig. 3.3

A fim de resolver o problema, torna-se necessário fazer entrar dois números no computador e compará-los

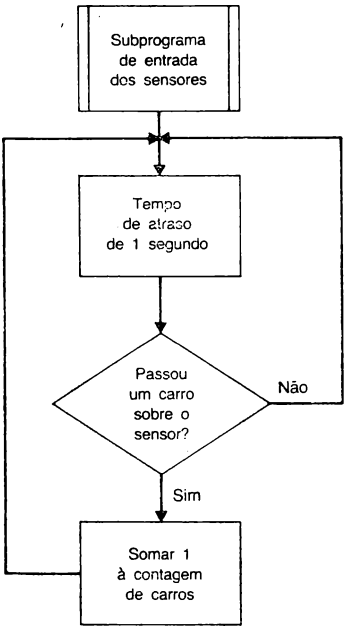


Fig. 3.4

entre si. É então mantido o maior deles, comparando-o com o número comunicado em seguida ao computador. É novamente retido o maior deles, e admitido um novo número. Este processo é repetido até todos os números terem entrado no computador e terem sido comparados. Numa situação real este tratamento pode fazer parte de um programa maior destinado a descobrir a pessoa mais idosa numa dada listagem (um censo), a pessoa que deve ser reformada em seguida por uma dada em-

presa, o maior volume de vendas num conjunto de vendedores, etc.

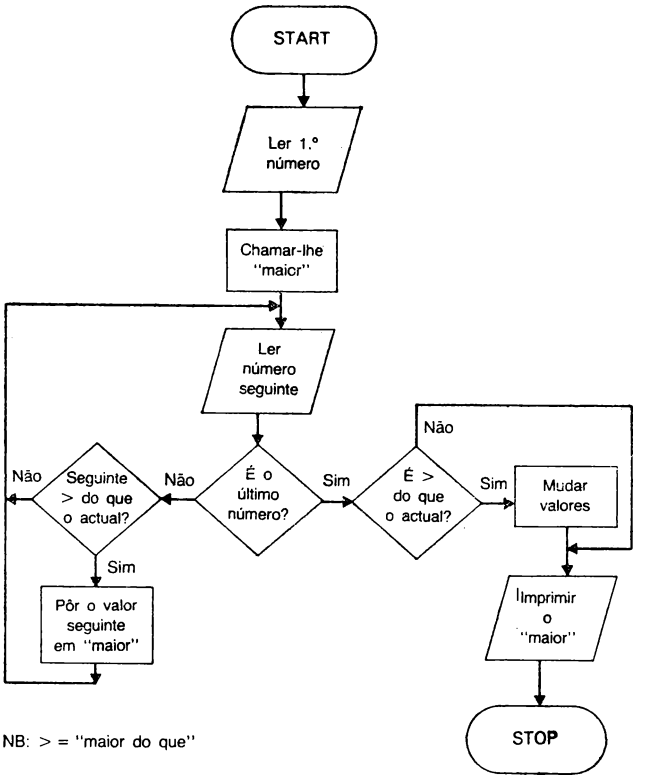


Figura 3.5

A diferença básica entre um ser humano e um computador ao resolver um problema deste tipo é que o ser humano utiliza de modo complexo os seus olhos e o cérebro para descobrir o número maior. Observa rapidamente

a lista de números e nota aqueles que possuem algarismos mais significativos em certas posições, fixando assim os valores maiores. Os computadores ainda não são capazes de fazer isto. Apenas sabem comparar dois números que lhes são apresentados e decidir qual deles é o maior (ou o menor, conforme o caso). Este número é retido e comparado em seguida com outro, como já se disse. O computador só «sabe» qual é o maior número depois de ter lido o último.

Compreender isto é compreender as quatro operações básicas dos computadores, e conduzirá a um melhor conhecimento do que é de facto a programação. Só é possível adquirir porém este conhecimento experimentando alguns problemas reais. Aprende-se a fazer um programa programando.

Observemos agora o fluxograma e discutamos alguns dos seus aspectos mais importantes. Não se preocupe se não compreender tudo à primeira leitura. Poucas pessoas o conseguem. É a partir da experiência prática (ou seja, dos erros feitos) que se compreende a verdadeira importância destes aspectos. Portanto, esta secção deverá ser relida em ocasiões futuras.

1 — Note que existem apenas duas instruções READ/ /INPUT em todo o fluxograma. A primeira instrução INPUT lê o primeiro dado válido (número), e a segunda é responsável pela leitura de todos os números subsequentes, independentemente da sua quantidade. Seria inaceitável utilizar uma nova instrução de entrada para cada número. Como seria possível, nesse caso, prever a quantidade de números que se desejaria comparar, por exemplo, num recenseamento?

No entanto, usando uma instrução de entrada e um teste de controlo, é possível comunicar ao computador qualquer número de dados. É esta a essência do processamento em computador. Trata-se de uma técnica muito simples, comparável à invenção da roda ou ao conceito de zero no sistema de números decimais, mas sem ela não seria possível utilizar os computadores. A capacidade para repetir uma instrução ou uma série delas é de facto o «segredo» da programação.

2 — Suponha que a lista de dados envolve muitas centenas de números, como acontece num recenseamento. Não será desperdício ter uma instrução que repetidamente verifica centenas ou milhares de vezes se um número é o último, mas que de facto só será necessária uma vez? A resposta óbvia é «sim» — parece ser um desperdício. Mas qual é a alternativa permitida pelas quatro operações de um computador? Nenhuma. A resposta para este problema reside na incrível velocidade a que estas operações básicas são executadas. Estas comparações podem ser realizadas muitos milhões de vezes por segundo. Assim, mesmo que seja necessário dar entrada a milhares de números e verificar em todos os casos se se trata ou não do último número, apenas são perdidos alguns milissegundos...

3 — A *lógica* de um programa nada tem a ver com a disciplina chamada lógica. Refere-se ao processo de tomada de decisões pela máquina, assegurando o cumprimento em cada instante da instrução correcta. Normalmente, as instruções

são executadas pela CU uma a seguir à outra segundo uma sequência estrita. Se for necessária uma dada ordem de execução, é a «lógica» do programa que assegura o seu cumprimento. Um bom exemplo disto são as duas respostas possíveis à pergunta «foi lido o último número?». Se «sim», depois de uma última comparação é impresso o número maior; se «não», este número é simplesmente comparado com o anterior. Um fluxograma ajuda o programador a especificar a lógica e a verificar se funciona do modo pretendido.

4 — Como «sabe» o computador se existe ou não um outro número para ser lido? Os seres humanos podem resolver este problema usando os olhos e o cérebro. Sabemos qual é o último número porque o vemos no fim da lista. Mas os computadores não têm esta vantagem, não podendo ver do mesmo modo que nós. Uma máquina pode comparar dois números e decidir entre dois tipos de acção; mas os programadores devem incluir um número especial no fim da lista, e quando este é detectado o programa «sabe» que já leu todos os números válidos.

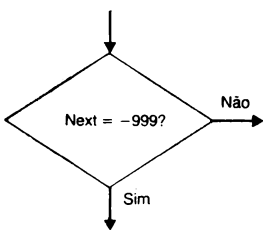


Figura 3.6

Como definem os programadores este número especial? Depende da natureza dos números válidos. No nosso caso, os dados são sempre números positivos. Uma boa forma de resolver a situação consistirá portanto no uso de um número negativo, por exemplo — 999. Podemos agora emendar a caixa de decisão «É o último número?» para a forma que tem na figura 3.6, isto é, «Número igual a — 999?». Se a resposta é «sim», o programa «sabe» que leu todos os dados pertinentes e que pode imprimir o maior dos números lidos. Se a resposta é «não», o programa «sabe» que deve ler ainda um outro dado (ver a figura 3.7).

É necessário compreender que dado o número limitado de funções que o computador é capaz de realizar, este é apenas um truque a que o programador se vê forçado a recorrer a fim de resolver o problema. O aspecto aqui mais importante é, uma vez mais, que o programador só pode escrever um programa depois de compreender perfeitamente a natureza dos dados que irá usar.

5 — Finalmente, um fluxograma deve ser capaz de ter em conta todas as situações possíveis. Suponhamos, no nosso exemplo, que o primeiro número a ler não era válido, sendo de facto o número — 999 que indica o final da lista de números. A figura 3.5 não permitiria enfrentar esta situação anormal e, no entanto, como os seres humanos são os responsáveis pela entrada de dados num computador, ela poderia facil-

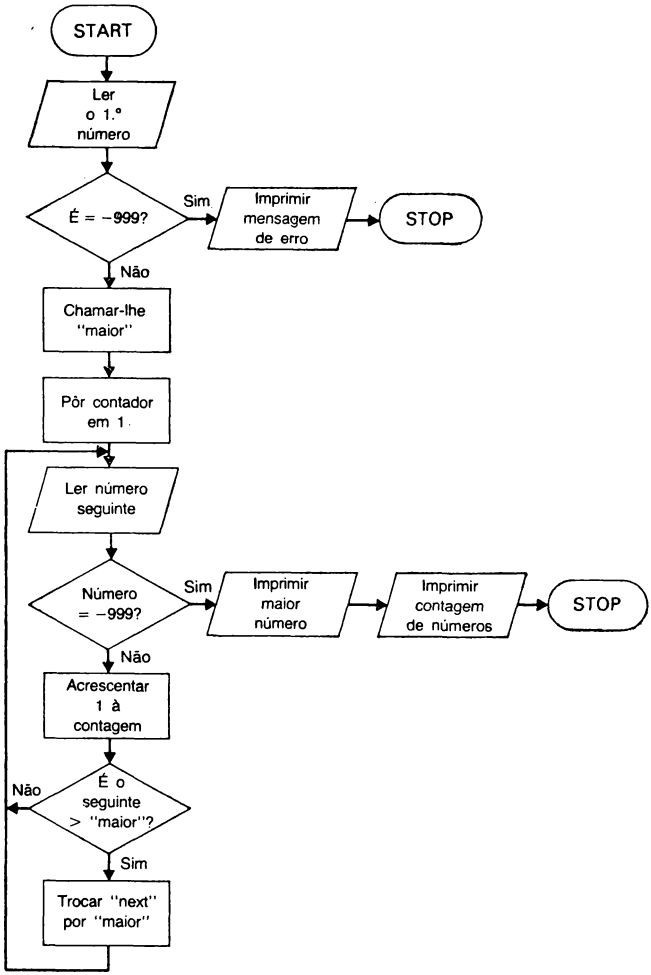

START
Ler o 1.º número
É = −999?
Sim
Imprimir mensagem de erro
STOP
Não
Chamar-lhe "maior"
Pôr contador em 1
Ler número seguinte
Número = −999?
Sim
Imprimir maior número
Imprimir contagem de números
STOP
Não
Acrescentar 1 à contagem
É o seguinte > "maior"?
Não
Sim
Trocar "next" por "maior"

Figura 3.7

mente ocorrer. O que impede um funcionário de enganar-se e colocar o número especial no início da lista e não no fim? Quanto mais nos envolvemos em problemas de programação, melhor compreendemos a facilidade com que estes acontecimentos anormais se transformam em lugar comum.

Tivemos portanto este problema em consideração ao construir o fluxograma da figura 3.7, e além disso alterámos nele as palavras utilizadas em algumas caixas. Observe cuidadosamente a figura. Se bem que talvez ainda não se aperceba do facto, este fluxograma envolve grande parte das técnicas gerais da arte da programação.

IV

MAIS PRINCÍPIOS DE PROGRAMAÇÃO

É a partir do fluxograma que o programador começa a escrever as instruções de programação na linguagem que lhe interessa. Tentemos fazê-lo a partir do fluxograma da figura 3.7. Como não estamos a discutir nenhuma linguagem em particular, utilizaremos aqui uma pseudolinguagem, ilustrada na tabela 4.1, e depois traduzi-la-emos para linguagem Basic com os comentários apropriados.

Tabela 4.1

Número de linha	Instrução	Tipo
1	READ primeiro número chamando-lhe «maior» (até agora)	Entrada
2	IF número = — 999 THEN GOTO (então ir para) a instrução 11 . . .	Comparação
3	LET (fazer) contagem de números = 1	Aritmética
4	READ NEXT (seguinte) número . .	Entrada
5	IF NEXT = — 999 THEN GOTO instrução 9	Comparação
6	LET contagem = contagem + 1 . . .	Aritmética
7	IF NEXT maior do que «maior», THEN LET NEXT passar a «maior»; e GOTO 4	
8	Senão, GOTO instrução 4	Comparação
9	PRINT «O maior número é», contagem	Saída
10	STOP (parar) a execução	
11	PRINT mensagem «Erro no primeiro número»	Saída
12	STOP a execução	

Convirá notar os seguintes pontos:

Instrução 1 — É lido o primeiro número, que pode ser de facto o número especial (o que significará que não existe qualquer número válido).

Instrução 2 — O programa verifica este número, comparando com o número especial que indica o fim da lista. Se é o mesmo, a máquina passa a executar a linha 11, imprimindo na saída uma mensagem de erro.

Se não é o mesmo, o programa considera que se trata de um número válido e continua para o seguinte.

Instrução 3 — Como não foi lido o número — 999, a contagem passa para o valor 1.

Instrução 4 — É lido o número seguinte a fim de ser comparado com o anterior.

Instrução 5 — É verificado o número seguinte, para descobrir se se trata do último. Se for, o programa passa para a instrução 9 imprimindo na saída o maior número lido; se não é, o programa passa para a instrução que se segue.

Instrução 6 — Tendo lido um outro número válido, a contagem é aumentada de 1.

Instrução 7 — O programa verifica aqui o último número lido para a variável NEXT a fim de verificar se é maior que o contido na variável «maior». Se sim, o programa substitui o valor em «maior» pelo de

NEXT, passando à instrução 4. Se não, o programa passa simplesmente para a instrução que se segue, que é apenas...

Instrução 8 — ... uma instrução que envia de novo para a instrução 4.

Instrução 9 — Esta instrução imprime a mensagem «O maior número é» juntamente com o valor do maior número contido no nome «Maior», assim como o total de números lidos.

Instrução 10 — Pára a execução do programa.

Instrução 11 — Imprime uma mensagem de erro no caso de o número — 999 ter sido introduzido no início da lista de números.

Instrução 12 — Pára a execução do programa.

A instrução 3 coloca o contador de números em 1. Esta simples instrução é por vezes designada «de atribuição» porque atribui um valor a uma variável — neste caso «contagem». O aspecto aqui importante é que o valor de «1» é lido do exterior, sendo pelo contrário definido internamente sob a forma de uma constante especificada no programa. Podemos considerar isto como uma instrução aritmética primitiva.

A instrução 6 soma «1» a esta «contagem». Deste modo simples, o programa pode manter um registo do número de valores dados de tal modo que possa imprimir esse número ao mesmo tempo que imprime o valor «maior».

A instrução 8 é uma instrução lógica bastante simples. Como não está aqui envolvida qualquer comparação, é normalmente designada instrução de *salto incon-*

dicional, ao contrário das instruções de *salto condicionado* n.os 2, 5 e 7.

No caso destas últimas instruções, caracterizadas pela construção IF... THEN, a parte THEN é executada apenas quando a comparação ou condição é verdadeira. Assim, na instrução 2 (e na 5), a parte THEN é executada (obedecida) apenas quando o número acabado de ler pela instrução 1 (ou 4) é de facto igual a — 999. De outro modo, a parte THEN é ignorada e é executada a instrução que se segue normalmente. No caso da instrução 7, a parte THEN só é executada nas ocasiões em que o valor contido em NEXT é maior do que o contido em «maior». Depois da parte THEN ser executada, o programa segue normalmente. Por fim encontram-se duas instruções STOP, uma na linha 10 e outra na 12. Logicamente, ou seja de acordo com o trajecto de facto seguido pela máquina na execução do programa, só uma destas instruções será de facto encontrada por ela de cada vez que o executa. Note igualmente que apesar de as linguagens de programação requererem alguma forma de instrução STOP/END, não requerem uma instrução START (início) correspondente.

O programa levanta alguns outros problemas, mas convém discutir certas questões prévias antes de voltarmos a eles.

Nomes de dados

Em álgebra, a expressão x = a + b significa que o valor representado pelo símbolo *a* e o valor representado pelo símbolo *b* são somados e produzem um resultado

por sua vez representado pelo símbolo x. Esta expressão pode designar um número infinito de valores a e b. Este mesmo conceito aplica-se a todas as linguagens de programação. No programa que determina o maior de um conjunto de números não somos obrigados a usar sempre os mesmos números. É possível usar repetidamente esse programa para diferentes grupos de dados, independentemente da sua dimensão. Isto torna-se possível porque o programa não especifica dados mas sim nomes de dados.

A ideia de dar nomes como a, b e x é um lugar comum em matemática. Também damos normalmente nomes aos nossos filhos ou até aos animais domésticos... Isto ajuda-nos a identificá-los. É necessário fazer o mesmo em programação. Uma razão disto é que ajuda o computador a descobrir os valores correctos que se encontram em muitas posições no interior da memória. Uma outra razão é que, com algumas excepções, os programas são destinados a ser usados muitas vezes. Como são caros, tendem a ser usados repetidamente. Por exemplo, um programa de salários é usado todos os meses. Apenas os valores dos dados se podem alterar de um mês para o outro.

O programa correspondente ao fluxograma da figura 3.7 poderá ser usado para um número infinito de conjuntos de dados porque usámos nomes (em vez dos números propriamente ditos) como NEXT, «maior», «contagem», etc.

Os nomes de dados constituem portanto uma parte integral de um programa. Cada linguagem possui regras próprias para a criação destes nomes. Em algumas linguagens, os nomes são restringidos a dois caracteres,

noutras a seis ou mesmo a mais. Cada programador deve descobrir por si próprio estas regras. Quaisquer que sejam as regras, é sempre útil criar nomes que tenham algum significado mnemónico, indicando o tipo de dados a que se referem. Se queremos multiplicar o «número de horas de trabalho» pelo «preço do trabalho à hora» a fim de obter um «salário total», poderemos por exemplo usar nomes mnemónicos como TOT=HRS * PRÇ.

Armazenamento de valores pelos respectivos nomes

A «instrução» acima constitui um bom exemplo de como se deve armazenar informações no interior da CPU. A memória central consiste num certo número de posições de registo. Algumas destas posições são usadas para guardar instruções de programa, estando o restante disponível para dados, cada um deles referenciado pelo nome escolhido pelo programador. Quando este utiliza um nome como HRS, MAIOR ou NEXT, o nome em causa é associado às posições de memória disponíveis. O exemplo seguinte poderá ilustrar isto.

```
1  INPUT HRS, PRÇ
2  TOT=HRS * PRÇ
3  PRINT TOT
4  STOP
```

Notar que em BASIC seriam usados nomes muito mais curtos, por exemplo H para horas, P para preço, T para total. Se este curto programa for introduzido na máquina, a primeira instrução levará esta a ler valores colocando-os nas posições de memória associadas aos nomes HRS e PRÇ definidos pelo programador. A instru-

ção seguinte transfere, sob o comando da CU, *cópias* destes valores das posições de memória em causa para a ALU, provocando simultaneamente uma operação de multiplicação. Esta mesma instrução indica que o resultado do cálculo feito deve ser colocado numa posição de memória chamada TOT. No final desta instrução, o estado da memória assemelha-se à figura 4.1. A instrução 3 limita-se a imprimir o valor presente na posição TOT no dispositivo de saída. O equivalente binário de 120 £, no caso da figura 4.1, é traduzido em algarismos decimais comuns. No capítulo seguinte discutiremos brevemente o modo como se atribuem nomes às posições de memória.

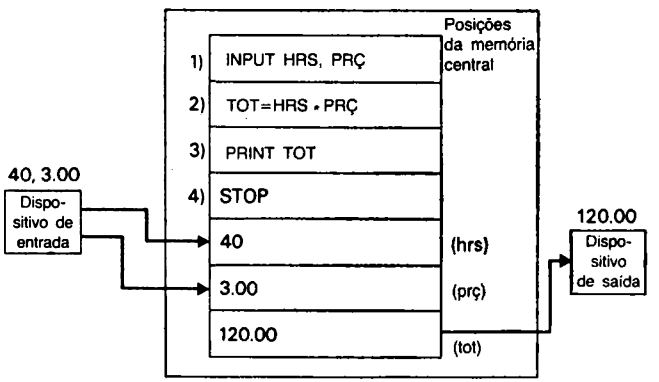


Figura 4.1 — Depois do processamento o valor TOT é transferido para o dispositivo de saída

Conjuntos de caracteres

Num texto em português encontram-se certamente muitos caracteres diversos, por exemplo caracteres alfa-

béticos em maiúsculas e minúsculas, algarismos decimais (0-9), símbolos de pontuação, talvez mesmo símbolos matemáticos ou usados na gíria de qualquer profissão. Talvez sejam nele usadas centenas de caracteres. As linguagens de programação são muito mais limitadas. A FORTRAN, como já vimos, possui apenas 47 caracteres, usando apenas letras maiúsculas. Na figura 4.2 mostra-se o conjunto de caracteres das linguagens BASIC e FORTRAN. Note a pequena quantidade de símbolos especiais em ambas, apenas 11 em FORTRAN.

FORTRAN	BASIC
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 ( ) * + − / £ , = . "espaço"	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 ( ) * + − / £ , = . "espaço" ; > < ↑ ↓ " :

Figura 4.2 — Conjuntos de caracteres alfanuméricos usados em FORTRAN e BASIC. Note que alguns sistemas computadores permitem o uso de um maior número de caracteres nestas linguagens

Talvez pareça à primeira vista que as linguagens de programação são demasiado restringidas pelo pequeno conjunto de caracteres que usam, mas na prática este facto não constitui um problema. No entanto, o programador deve de facto saber quais os caracteres que pode usar. O uso de quaisquer outros símbolos provoca problemas, impedindo a execução do programa. Considere-

mos um exemplo. Em aritmética normal, um símbolo como o x minúsculo simboliza a multiplicação. Mas como a FORTRAN, a BASIC e a maior partes das outras linguagens não permitem estes símbolos, é necessário inventar outro. Neste caso é normalmente usado o asterisco «*», como já fizemos na «frase» TOT=HRS * PRÇ.

Exemplo 1 — Voltemos agora ao fluxograma da figura 3.7 e ao correspondente pseudoprograma em frases normais então usado. Em seguida apresentamos este programa na linguagem de programação BASIC. Antes de a estudar, devemos porém notar alguns pormenores.

(1) Os nomes dos valores numéricos consistem em dois caracteres na maior parte das versões da BASIC. O primeiro pode ser alfabético, e o segundo um algarismo opcional no sistema decimal (ou seja, 0, 1, 2, ..., 9). Assim, A1, B e B0 são válidos, mas 4R, 3 e AD4 não o são.

(2) Como a BASIC foi concebida para utilizadores em «Time-sharing» (ver o capítulo 6), todas as instruções recebem um *número de linha,* em ordem ascendente, com um máximo de cinco algarismos (ou seja, numa gama entre 00001 e 99999). Este número de linha deve ser usado quando se obriga a máquina a executar um salto para uma instrução fora da ordem. Normalmente incluem-se zeros à esquerda quando os algarismos significativos não chegam a cinco, a fim de tornar o programa mais legível.

(3) A última instrução de um programa BASIC deve ser END (fim), para informar o computador de que não se seguem mais instruções e parar a exe-

cução do programa. Só é permitida uma instrução END e, portanto, note que a instrução da linha 230 contém um GOTO 900, indicando 900 a linha onde se encontra a instrução END.

```
00100 REM PROGRAMA PARA LER UMA LINHA DE
      NÚMEROS
00110 REM E DESCOBRIR O MAIOR
00120 INPUT L1
00130 IF L1=—999 THEN 240
00140 LET C1=1
00150 INPUT N
00160 IF N=—999 THEN 220
00170 LET C1=C1+1
00180 IF N > L1 THEN 200
00190 GOTO 150
00200 LET L1=N
00210 GOTO 150
00220 PRINT «O MAIOR NÚMERO É», L1, «TOTAL
      DE NÚMEROS LIDOS=», C1
00230 GOTO 900
00240 PRINT «ERRO NO PRIMEIRO NÚMERO»
00900 END
```

onde L1 = contém o maior número lido até ao momento
N = número a ler em seguida
C1 = contagem de números lidos

As instruções REM usadas em BASIC permitem a inclusão de observações (REMarks em inglês) não constituindo de facto instruções de programação, isto é, não sendo executáveis pela máquina.

Figura 4.3

Exemplo 2 — Considere o problema do censo já discutido anteriormente.

Inclui-se em seguida um fluxograma do problema (figura 4.4) assim como o programa final em BASIC.

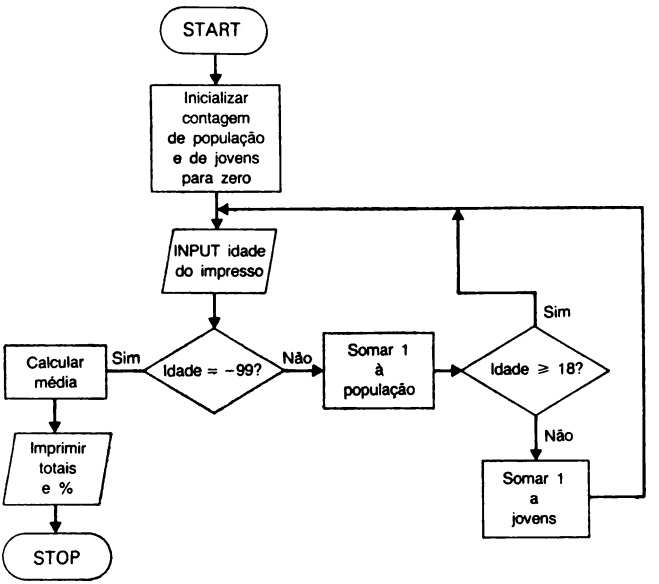


Figura 4.4

```
00100 REM PROGRAMA PARA LER QUALQUER
          NÚMERO DE IMPRESSOS
00110 REM PARA DESCOBRIR A POPULAÇÃO TO-
          TAL DE UMA ÁREA, O
00120 REM NÚMERO TOTAL DE JOVENS E A PER-
          CENTAGEM DESTES
```

```
00130 REM EM RELAÇÃO À POPULAÇÃO TOTAL
00140 REM
00150 LET P1=0
00160 LET C1=0
00170 INPUT A1
00180 IF A1=-99 THEN 230
00190 LET P1=P1+1
00200 IF A1 >= 18 THEN 170
00210 LET C1=C1+1
00220 GOTO 170
00230 LET P2=C1/P1 * 100
00240 PRINT «POPULAÇÃO TOTAL=», P1
00250 PRINT «NÚMERO TOTAL DE JOVENS=», C1
00260 PRINT «PERCENTAGEM DE JOVENS=», P2
00270 END
```

onde P1=Contagem total da população
C1=Número total de jovens
A1=Idade lida no impresso
P2=Percentagem de jovens determinada depois do processamento.

Figura 4.5

Exemplo 3 — Uma pequena loja pode facilmente computadorizar os seus pedidos de encomenda. Um empregado anota o número de mercadorias em stock e a quantidade a encomendar. O programa inclui um ficheiro de dados incorporado (usando instruções DATA — «dados») que apresenta uma lista de códigos de diferentes mercadorias e a quantidade máxima que pode ser encomendada em qualquer momento (ver a figura 4.7). A ma-

triz ou lista bidimensional armazenará igualmente a quantidade a encomendar, de tal modo que se torna

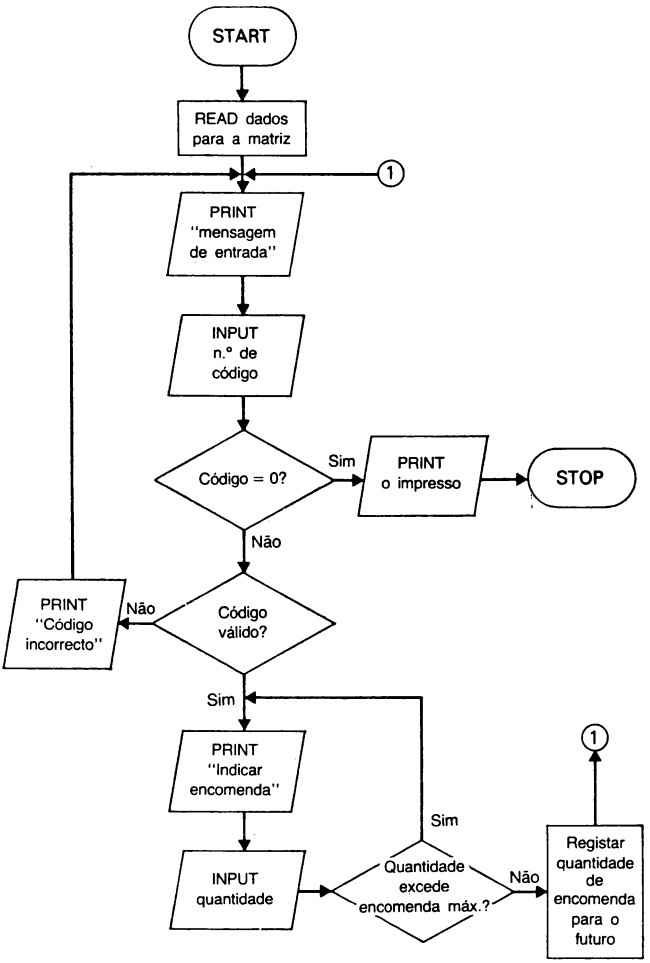


Figura 4.6 — Fluxograma para pedidos de encomenda.

possível imprimir o pedido de encomenda no final do programa. Notar que em certos sistemas pode tornar-se necessário passar esta coluna para zero de cada vez que o programa é usado.

A figura 4.6 apresenta-nos o fluxograma que soluciona este problema, e no texto apresenta-se a listagem do programa (em BASIC). Note que se trata de um programa muito curto, e de facto com alguns acrescen-

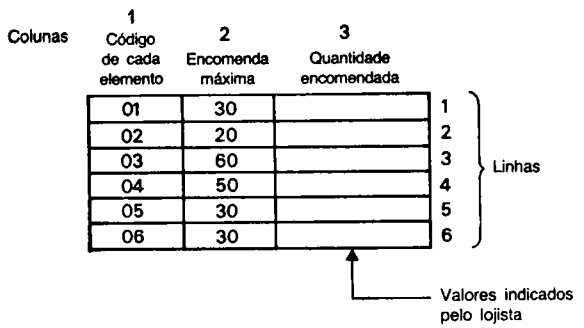


Figura 4.7 — Uma lista bidimensional (um «array»).

tos poderia transformar-se num programa muito mais útil e realista. Por exemplo, a matriz poderia ser aumentada para qualquer valor aumentando a instrução DIM (de dimensionamento). É possível fazer quaisquer alterações no ficheiro de dados alterando as instruções DATA. Neste caso os valores são aos pares — primeiro o número de código do artigo e depois a quantidade máxima a encomendar. O programa verifica igualmente se o código de cada artigo é válido.

A fim de compreender bem as listagens dos programas dos exemplos 3 e 4, convirá consultar nesta colecção o livro «Guia Prático de Basic» sobre a programação BASIC. No entanto, bastarão aqui algumas palavras para permitir já ter uma ideia do que neles se faz.

A instrução DIM

A instrução DIM é um exemplo de uma instrução que fornece informação *antes* de o programa ser executado. Indica muito simplesmente que certas posições no interior da memória central devem ser reservadas para uso durante a execução do programa. No exemplo 3,

```
DIM L(6,3)
```

reserva uma matriz (ver figura 4.7) de 18 posições sob o nome de «L». Para fazer referência a qualquer destas posições durante a execução do programa é necessário indicar entre parêntesis tanto o número da linha como o da coluna onde se encontra. Assim, L(5,2) indica a posição da linha 5, coluna 2. Dentro do parêntesis podem-se usar constantes inteiras ou variáveis, mas se se utiliza um nome de variável este nome deve receber anteriormente um qualquer valor prévio. A linha 00280 mostra um exemplo:

```
00280 IF N=L(I,1) THEN 00330
```

Esta instrução provoca a execução de um «loop» (um ciclo de instruções) no interior do programa, sendo repetidas todas as instruções BASIC que se encontram entre a instrução FOR e a correspondente instrução NEXT. O número de vezes que estas instruções são executadas é determinado pela instrução FOR. Por exemplo, no caso que estamos a estudar:

```
00260 FOR I=1 TO 6
  •
  •
  •
00290 NEXT I
```

as instruções indicadas repetirão o mesmo ciclo seis vezes. I é colocado ao valor 1, sendo executadas as instruções entre as linhas 270 e 280. A instrução «NEXT I» da linha 290 força a unidade de controlo a voltar à instrução «FOR» da linha 00260, e aumenta I de 1 (pelo que ficará agora com o valor 2). Este incremento é verificado a fim de descobrir se excede o número máximo de repetições (neste caso 6). Se não excede, o «loop» é novamente executado, e a instrução «NEXT I» força novamente a CU a voltar à linha 260. Quando I excede 6, a CU passa automaticamente para a instrução *depois* da NEXT (no nosso caso, para a instrução 300).

Pode-se usar o valor de I no interior do loop, e se estudarmos este caso verificaremos que se usa I para definir o número de linha da variável L «dimensionada». Um «loop» FOR-NEXT pode ser incluído dentro de outro, como acontece nas linhas 160 a 200. O loop inte-

rior é sempre completado *antes* do exterior. Quando o loop exterior é novamente incrementado, o interior é completamente repetido. É por este processo que se lêem neste caso os valores da matriz «L».

É igualmente possível saltar para fora de um loop em qualquer ponto da sua execução, mesmo que esta ainda não esteja terminada.

Uma melhor explicação destas duas instruções está para além dos limites deste livro, mas bastará o que se disse para permitir ao leitor compreender o que se passa nos exemplos 3 e 4.

Programa para o Exemplo 3
81/03/04 10.36.36

Programa de stocks

```
00100 REM A INSTRUÇÃO DIM RESERVA ESPAÇO
      — UMA MATRIZ DE 6 LINHAS E 3 COLUNAS
00110 DIM L(6,3)
00120 REM AS INSTRUÇÕES DATA CONTÊM OS
      VALORES A USAR PELO PROGRAMA
00130 DATA 01,30,02,20,03,60
00140 DATA 04,50,05,30,06,30
00150 REM AS INSTRUÇÕES SEGUINTES LÊEM OS
      VALORES DADOS PARA A MATRIZ
00160 FOR P=1 TO 6
00170 FOR T=1 TO 2
00180 READ L(P,T)
00190 NEXT T
00200 NEXT P
```

```
00210 REM INDICAR NÚMERO DE CÓDIGO
00220 PRINT «INDICAR NÚMERO DE CÓDIGO. ZERO
      TERMINA ENTRADAS»
00230 INPUT N
00240 IF N=0 THEN 00420
00250 REM VERIFICAR SE O CÓDIGO É VÁLIDO
00260 FOR I=1 TO 6
00270 LET K=1
00280 IF N=L(I,1) THEN 00330
00290 NEXT I
00300 PRINT «CÓDIGO INVÁLIDO —»; N
00310 GOTO 00220
00320 REM INDICAR QUANTIDADE REQUERIDA
00330 PRINT «INDICAR ENCOMENDA NECESSÁ-
      RIA»
00340 INPUT R1
00350 IF R1 < L(K,2) THEN 00390
00360 PRINT «ENCOMENDA EXCEDE O MÁXIMO»
00370 GOTO 00330
00380 REM GUARDAR QUANTIDADE NA TER-
      CEIRA COLUNA DA MATRIZ
00390 LET L(K, 3) =R1
00400 GOTO 00220
00410 REM IMPRIMIR PEDIDO DE ENCOMENDA
00420 PRINT
00430 PRINT «EMPRESA — DD/MM/AA — PEDIDO
      DE ENCOMENDA»
00440 PRINT
00450 PRINT
00460 FOR 1=1 TO 6
00470 PRINT «ARTIGO», L(I, 1), «QUANTIDADE=»,
      L(I, 3)
```

```
00480 NEXT I
00490 END
```

Vejamos a sucessão de entradas e saídas do programa durante a execução.

Carregar na tecla «run» (executar, fazer «correr» o programa).

```
81/03/04        10.39.12
```

Programa de stocks

```
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 1
INDICAR ENCOMENDA NECESSÁRIA
? 24
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 2
INDICAR ENCOMENDA NECESSÁRIA
? 12
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 3
INDICAR ENCOMENDA NECESSÁRIA
? 50
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 4
```

```
INDICAR ENCOMENDA NECESSÁRIA
? 48
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 5
INDICAR ENCOMENDA NECESSÁRIA
? 24
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 60
CÓDIGO INVÁLIDO — 60
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 6
INDICAR ENCOMENDA NECESSÁRIA
? 240
ENCOMENDA EXCEDE O MÁXIMO
? 24
INDICAR NÚMERO DE CÓDIGO. ZERO TERMINA
   ENTRADAS
? 0
LOJA — DD/MM/AA — PEDIDO DE ENCOMENDA
ARTIGO 1 QUANTIDADE=24
ARTIGO 2 QUANTIDADE=12
ARTIGO 3 QUANTIDADE=50
ARTIGO 4 QUANTIDADE=48
ARTIGO 5 QUANTIDADE=24
ARTIGO 6 QUANTIDADE=24
EXECUÇÃO COMPLETA
```

Notar que o símbolo «?» indica um pedido de dados pelo computador.

Exemplo 4 — Uma firma de aluguer de automóveis poderia usar um outro programa muito simples para calcular o custo baseado no tipo de automóvel, reflectindo o aluguer ao dia e o total de quilometragem percorrida (ver a figura 4.8). Usou-se novamente uma matriz (figura 4.9) para criar um «ficheiro» usado pelo programa. Este programa verifica igualmente o tipo de carro, de tal modo que não seja indicado à máquina qualquer tipo não válido. Imprime os dados introduzidos, de tal modo que o operador os possa verificar. O espaço disponível aqui não nos permite apresentar um programa mais extenso, e portanto omitimos as instruções REM. Mas mesmo um principiante não necessitará de muito tempo para poder escrever programas deste tipo e ampliá-los de modo a poderem realizar tarefas mais úteis.

Programa para o exemplo 4

```
81/03/04        10.28.49
FIRMA DE ALUGUER DE AUTOMÓVEIS
00100 DIM C3(3,3)
00110 DATA 1,12,10
00120 DATA 2,15,12
00130 DATA 3,20,15
00140 FOR L=1 TO 3
00150 FOR N=1 TO 3
00160 READ C3(L,N)
00170 NEXT N
00180 NEXT L
00190 PRINT «INDICAR TIPO DE CARRO, N.º DE
      DIAS, QUILOMETRAGEM TOTAL»
```

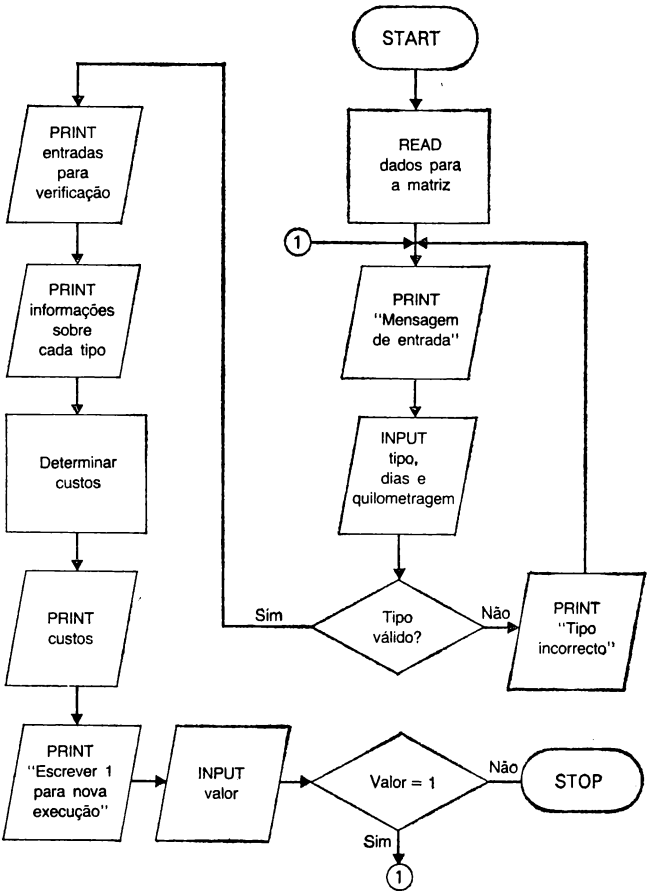


Figura 4.8 — Fluxograma para empresa de aluguer de automóveis

```
00200 PRINT «POR ESTA ORDEM E SEPARANDO
      COM VÍRGULAS»
00210 PRINT
00220 PRINT
```

Tipo de carro	Custo por dia (escudos)	Custo por quilómetro (escudos)
1	12	10
2	15	12
3	20	15

Figura 4.9

```
00230 INPUT C,D,M,
00240 FOR I=1 TO 3
00250 LET K=1
00260 IF C3(I, 1)=C THEN 00310
00270 NEXT I
00280 PRINT «TIPO NÃO VÁLIDO —»; C
00290 PRINT
00300 GOTO 00190
00310 PRINT «VERIFICAÇÃO: TIPO=»; C; «N.º DE
      DIAS=»;D; «QUILOMETRAGEM=»;M
00320 PRINT
00330 PRINT «TIPO»; C; «=»; C3 (K, 2); «ESCUDOS
      POR DIA»;
00340 PRINT «    =»; C3 (K, 3); «ESCUDOS POR
      QUILÓMETRO»
00350 PRINT
00360 LET F1=C3 (K2) * D
00370 LET F2=C3 (K3) * M/100
00380 PRINT «CUSTO DOS DIAS=» ;D; «*»; C3
      (K,2); «=» ;F1; «ESCUDOS»
00390 PRINT «CUSTO DOS QUILÓMETROS=» ;M;
      «*»; C3 (K,3); «=»; F2; «ESCUDOS»
00400 PRINT
00410 PRINT «TIPO      DIAS      QUILÓMETROS»
00420 PRINT «**********************************»
```

```
00430 PRINT
00440 PRINT « »; C; « »; D; «  »; M
00450 PRINT
00460 PRINT «TOTAL MENOS DEPÓSITO=»; F1+
      +F2 — 5000; «ESCUDOS»
00470 PRINT
00480 PRINT «AGRADECEMOS A SUA PREFERÊN-
      CIA»
00490 PRINT
00500 PRINT «ESCREVER 1 PARA NOVA EXE-
      CUÇÃO»
00510 INPUT S
00520 IF S=1 THEN 00190
00530 END
```

Vejamos as entradas e saídas do programa ao ser executado:

```
   Carregar na tecla «run» para executar.
81/03/04        10.32.22
PROGRAMA DE FIRMA DE ALUGUER DE AUTO-
   MÓVEIS
INDICAR TIPO DE AUTOMÓVEL, NÚMERO DE DIAS,
   QUILOMETRAGEM TOTAL, POR ESTA ORDEM
   E SEPARANDO COM VÍRGULAS

?  1, 6, 786
VERIFICAÇÃO: TIPO=1 NÚMERO DE DIAS=6 QUI-
   LOMETRAGEM=786

TIPO 1=1000 ESCUDOS POR DIA;
       =    5 ESCUDOS POR QUILÓMETRO
```

```
CUSTO DOS DIAS=6 * 1000=6000 ESCUDOS
CUSTO DOS QUILÓMETROS=786*5=3930 ESCUDOS
TIPO      DIAS      QUILÓMETROS
**********************************
  1         6              786
        TOTAL MENOS DEPÓSITO=4430 ESCUDOS
AGRADECEMOS A SUA PREFERÊNCIA
ESCREVER 1 PARA NOVA EXECUÇÃO
? 1
INDICAR TIPO DE AUTOMÓVEL, NÚMERO DE DIAS,
   QUILOMETRAGEM TOTAL, POR ESTA ORDEM
   E SEPARANDO COM VÍRGULAS
?  3, 3, 500
VERIFICAÇÃO: TIPO=3 NÚMERO DE DIAS=3 QUI-
   LOMETRAGEM=500
TIPO 3=1800 ESCUDOS POR DIA
        =7,5 ESCUDOS POR QUILÓMETRO
CUSTO DOS DIAS=3*1800=5400 ESCUDOS
CUSTO DOS QUILÓMETROS=500*7,5=3750 escudos
TIPO      DIAS      QUILÓMETROS
**********************************
  3         3              500
        TOTAL MENOS DEPÓSITO=4150 ESCUDOS
AGRADECEMOS A SUA PREFERÊNCIA
ESCREVER 1 PARA NOVA EXECUÇÃO
?  0
EXECUÇÃO TERMINADA.
```

Resta-nos apresentar as nossas desculpas ao leitor pela inverosimilhança dos preços indicados...

V

TRADUTORES DE LINGUAGENS E NÍVEIS DE LINGUAGEM

Níveis de linguagem

Existem fundamentalmente dois tipos ou categorias de linguagens de programação: linguagens de baixo nível e linguagens de alto nível. Até aqui temo-nos referido apenas a linguagens de alto nível, isto é, às linguagens que 80 % dos programadores utilizam para resolver os seus problemas.

Linguagens de baixo nível

É muitas vezes uma surpresa para os programadores principiantes saberem que o computador por eles usado só pode compreender uma linguagem apesar de ser possível usar neles programas escritos em linguagens diferentes. No entanto, todos os computadores são concebidos de modo a compreenderem apenas uma linguagem, a saber, a chamada *linguagem-máquina* ou *código-máquina*. É necessário ter em conta que quando

o computador é construído é igualmente especificada e concebida a sua linguagem, e que a máquina depois só responde a esta. Chama-se-lhe linguagem-máquina porque as instruções usadas nesta linguagem estão directamente relacionadas com certas características da máquina como as dimensões da sua memória, as dimensões de cada posição (ou seja, o número de algarismos binários em cada posição), etc. Estes pormenores não nos devem preocupar. Só são relevantes para certos técnicos especializados, chamados técnicos de sistema («systems engineers»), que tratam do funcionamento interno de um computador.

Linguagens de alto nível

Mas para 80 % ou mais das pessoas que trabalham em informática, apenas é relevante a solução de problemas concretos exteriores à máquina. Existem portanto outras linguagens que não estão orientadas para o aproveitamento dos recursos dos computadores mas sim para a solução de certos tipos de problemas. Para resolver um problema de um destes tipos, escolhe-se uma linguagem que permita resolvê-lo com maior facilidade. A FORTRAN, por exemplo, possui características que ajudam a resolver problemas de matemáticas ou engenharia; a COBOL facilita a resolução de problemas comerciais, envolvendo registos e ficheiros. Mas a COBOL não é muito útil para a resolução de problemas matemáticos. Estas linguagens orientadas para a resolução de problemas de tipos específicos são chamadas

linguagens de alto nível. Os termos *baixo* e *alto* nada têm portanto a ver com a inferioridade ou superioridade da linguagem em causa mas com o facto de esta se encontrar mais próxima da máquina ou de nós e dos nossos problemas.

Tradutores de linguagens

Como cada computador só pode compreender a sua própria linguagem-máquina, o que acontece quando lhe apresentamos um programa em alguma linguagem de alto nível? Como é óbvio torna-se necessária uma fase de «tradução», tal como é necessário um tradutor para passar um texto de inglês para português. Essa tradução é realizada por um programa tradutor, que converte a linguagem «desconhecida» na linguagem própria do computador. O programa traduzido é igual ao programa original, apresentando-se no entanto num «formato» diferente.

Os tradutores das linguagens de alto nível são em geral designados por *compiladores*. São programas ou «software» fornecidos pelos fabricantes e que fazem parte do sistema operativo.

Um compilador deve executar várias funções. Primeiramente deve traduzir instruções de alto nível para o correspondente código-máquina do computador. Em segundo lugar, se existirem erros de sintaxe (ou seja, utilizações incorrectas das regras da linguagem, usando por exemplo caracteres não válidos ou demasiados caracteres no nome atribuído a um dado), o compilador

envia uma mensagem de erro, por vezes chamada uma mensagem de diagnóstico. Em terceiro lugar, quando o compilador encontra o nome de um dado, atribui este nome a uma posição disponível na memória central. Se estas posições foram usadas por algum programa anterior, os nomes usados nesse outro programa são «apagados» e substituídos pelos novos. Finalmente, se o programa não possui quaisquer erros de sintaxe, o compilador «informa» a unidade de comando de que o programa está pronto a executar.

O programa escrito pelo programador é designado por *programa-fonte* ou código-fonte, sendo a versão traduzida pela máquina chamada *programa-objecto.* Só a versão em programa-objecto pode ser executada pela CU. Se se observarem erros de sintaxe numa instrução, não pode ser traduzida para código-objecto e o compilador não «informa» a CU de que pode executar o programa. O programador deve corrigir os erros e depois submeter de novo todo o programa ao compilador. Existe ainda um outro tipo de tradutores, os *interpretadores,* associados aos sitemas ditos de «time-sharing».

O estudo destes tradutores é deixado para o capítulo seguinte, onde introduziremos o conceito de «time-sharing».

Tipos de erros

Quando se escrevem programas podem ocorrer erros de sintaxe, erros de lógica ou erros na apresentação de dados.

Erros de sintaxe

Incluem-se nesta categoria os erros que consistem no uso indevido das regras a que a linguagem obedece. Alguns destes erros já foram mencionados ao longo do livro. Outros consistem por exemplo na escrita incorrecta das palavras-chave que indicam ao computador um qualquer tipo de operação, por exemplo RAED em vez de READ, no uso incorrecto de operadores aritméticos, etc.

Erros de lógica

Os erros de lógica ocorrem frequentemente quando o programador não constrói um fluxograma antes de codificar o programa ou quando o concebe de modo incorrecto. O programador pode nestas condições levar o programa a saltar para uma instrução errada; por exemplo no primeiro programa do capítulo IV, a instrução 2 poderia enviar erradamente para a linha 9 em vez de o fazer para a linha 11. É importante compreender que o compilador nunca pode detectar erros de lógica: pode «compreender» as regras usadas numa dada linguagem, mas não aquilo que o programador deseja fazer. Os erros de lógica podem, muitas vezes, ser descobertos ao observar resultados inesperados ou sem rigor produzidos pelo programa durante a execução. Se os resultados não são aquilo que deviam ser, torna-se necessário verificar a lógica do programa.

Erros de dados

Um erro muito comum consiste na apresentação ao computador de dados errados. Este é provavelmente o tipo de erros mais difícil de detectar. Se o sr. Silva trabalhou 24 horas numa dada semana, mas o funcionário encarregue de comunicar as horas de trabalho ao computador escreveu no teclado 14 ou 42, o sr. Silva talvez se queixe se receber menos dinheiro do que espera, mas é menos provável que o faça se tiver recebido dinheiro a mais...

Qualquer programa de computador é apenas tão bom como os operadores que o utilizam. Num livro de maiores dimensões ser-nos-ia possível mostrar como é possível evitar muitos tipos de erros ou pelo menos reduzir a sua quantidade ou consequências. Mas dada a exiguidade do espaço de que dispomos apenas nos resta chamar a atenção do leitor para este problema e fornecer alguns exemplos de erros, o que será feito no princípio do penúltimo capítulo.

VI

SISTEMAS «BATCH» E «TIME-SHARING»

Ao longo dos anos tornaram-se predominantes dois modos de usar os computadores, designados em inglês por «batch» e «time-sharing».

Processamento por lotes

O primeiro sistema é normalmente designado em português por «processamento por lotes». Neste tipo de processamento, que frequentemente usa cartões perfurados como suporte de entrada, ou em geral dispositivos auxiliares magnéticos, os programadores limitam-se a deixar os cartões onde gravaram os seus programas num local qualquer para isso designado. Várias vezes ao longo de cada dia, os operadores recolhem estes programas e introduzem-nos na máquina, onde são processados por «lotes». Em momentos definidos os programadores vão buscar os seus programas juntamente com os resultados do computador, por exemplo várias horas mais tarde ou no dia seguinte. O período de tempo

entre a entrega de um programa para execução e a recolha dos resultados é normalmente o mesmo para todos os casos. Se o compilador descobre erros de sintaxe no programa, o programador ver-se-á forçado a corrigi-los e a voltar a apresentar o programa. O atraso causado à execução dos programas por este método de trabalho é perfeitamente satisfatório para certas aplicações, mas para outras é conveniente obter resultados imediatamente, passados apenas alguns segundos e não horas. Os sistemas «time-sharing» foram pensados para estas aplicações.

«Time-Sharing» — Processamento interactivo

Este segundo tipo de processamento é normalmente designado em português por processamento interactivo ou de «partilha de tempo», em tradução literal da expressão inglesa. A ideia básica por detrás destes sistemas consiste em atribuir um certo período de tempo, em geral dez milissegundos (um centésimo de um segundo), a cada utilizador de um terminal. O terminal é um dispositivo de entrada/saída, e um computador que realize um processamento interactivo pode estar ligado a um número de terminais entre 4 e 300. O número exacto de terminais depende das dimensões do computador. Os terminais recebem as «atenções» daquele por uma ordem estrita, sendo atribuído a cada um deles um período de tempo de dez milissegundos durante o qual pode executar instruções próprias.

Pode parecer ao leitor que este tempo é muito reduzido, mas de facto a máquina pode executar muitas cen-

tenas de instruções em dez milissegundos. O utilizador normal deste sistema de processamento dá-se normalmente por satisfeito ao receber alguma resposta do computador em cada dois ou três segundos. Os períodos de dez milissegundos de actividade que se sucedem de dois em dois segundos criam no utilizador a ilusão de que o computador só o está a atender a ele...

Se tivermos em consideração o tempo perdido pelo utilizador a pensar no que quer escrever no teclado e a carregar nas teclas, torna-se óbvio que por cada ciclo de dois segundos são poucos os terminais que requerem a atenção do computador. Deste modo torna-se possível ligar muitos terminais a um mesmo computador. Um atraso de dois ou três segundos entre as respostas da máquina dificilmente é notado, sendo praticamente equivalente a uma resposta imediata quando comparado ao sistema de processamento por lotes.

Interpretadores

Como já se disse existe um segundo tipo de tradutores de linguagens de alto nível, os chamados interpretadores, muito úteis nos sistemas de processamento interactivo se bem que neste também sejam usados compiladores. O interpretador difere do compilador no modo como traduz as instruções. Dissemos anteriormente que o compilador traduz *todo* o programa antes de passar à fase de execução do código-objecto. O interpretador, pelo contrário, apenas traduz *uma* instrução de cada vez, provocando imediatamente a sua execução antes de traduzir a instrução seguinte. A vantagem deste método

de tradução consiste em que, no caso de existir um erro de sintaxe numa dada instrução, o interpretador pode enviar para a saída uma mensagem de erro e esperar que o programador o corrija. Como é óbvio, só o sistema de processamento em partilha de tempo é apropriado ao uso deste tipo de tradutor, pois só nele é possível ao programador observar o efeito do seu programa. Qualquer erro pode ser rectificado imediatamente no terminal, podendo o programa continuar a ser executado.

Uma desvantagem dos interpretadores é que quando o programa executa um loop, por exemplo ao ler um conjunto de valores (figura 3.5), as instruções são novamente traduzidas. Significa isto que um programa que leia mil valores é obrigado a traduzir a mesma instrução mil vezes. Os interpretadores tendem portanto a gastar mais tempo na execução de um programa do que os compiladores. A BASIC constitui um exemplo de linguagem de alto nível que utiliza frequentemente interpretadores.

Escolha do sistema de processamento

Se pudermos escolher o sistema que nos convém, qual deveremos preferir? Os sistemas de processamento interactivo podem ser particularmente úteis quando se verifica se um dado programa funciona correctamente. É então possível corrigir imediatamente quaisquer erros recorrendo a um programa de fábrica designado por *montador de texto* que permite ao programador eliminar ou acrescentar o que quiser em qualquer ponto do pro-

grama. Se a aplicação para a qual o programa foi escrito requer respostas imediatas, só é possível utilizar um processamento interactivo. Dois exemplos poderão ilustrar claramente isto. Um médico de um hospital pode necessitar imediatamente de dados sobre um doente, a fim de poder contrariar uma recaída inesperada. Neste caso, a natureza da aplicação exige o uso de um sistema de processamento interactivo. Por outro lado, nenhum cliente está disposto a esperar várias horas até que um sistema de processamento por lotes lhe indique se tem ou não um lugar disponível num dado avião...

No entanto, os sistemas de partilha de tempo obrigam em geral a limitar as dimensões do programa, pois de outro modo o tempo global de resposta aos outros utilizadores pode ser afectado de modo inaceitável (ou *degradado*). Os problemas mais complexos devem portanto ser processados por lotes, pois neste sistema está disponível uma maior quantidade de memória central para os vários programas individuais. O processamento por lotes é também usado quando o volume de dados de entrada é muito grande, tornando-se portanto mais fácil de transcrever para cartões perfurados.

Um sistema de processamento por dados pode ainda permitir a impressão de resultados a alta velocidade em impressoras de linhas. Esta característica é particularmente conveniente no caso de aplicações que envolvam um grande número de saídas de resultados ou em que estes devam ser apresentados em impressos já preparados previamente, como acontece nos impressos usados para extractos de contas correntes, listas de salários, facturas e recibos, etc.

Qualquer que seja o sistema de processamento utilizado, afectará, mesmo que minimamente, o modo como o programador escreve os seus programas. O programa que é feito para processamento por lotes não pode ser observado ou alterado de qualquer modo durante a execução, o que obriga o programador a ter grandes cuidados enquanto o escreve. O sistema de processamento interactivo, por outro lado, permite ao programador testar o programa na máquina, «dialogando» com ele durante a execução através do programa interpretador; neste caso a verificação de um programa é ainda facilitada pela rapidez com que é possível introduzir na máquina os dados mais apropriados.

VII

O QUE ESTÁ EM CAUSA AO ESCREVER UM PROGRAMA

Fases da escrita de um programa

Uma tendência do programador principiante quando enfrenta a tarefa de escrever um programa consiste em começar a escrever imediatamente as instruções que o compõem. Na prática, esta deve ser a última coisa a fazer... A primeira fase do trabalho consiste na compreensão exacta do problema a resolver.

Um método muito eficaz de o fazer consiste em definir exactamente quais deverão ser os resultados apresentados na saída do computador. Isto pode obrigar a conceber com grande pormenor o formato exacto de saída, particularmente quando são usados impressos com listas de salários, facturas, etc., ou o melhor arranjo dos dados num visor VDU, por exemplo no caso de um sistema de reserva de lugares em aviões. Depois de conhecermos bem aquilo que o computador deve fornecer, podemos determinar com maior facilidade o tipo

de dados de entrada que a máquina deverá processar.

Um segundo aspecto a ter em conta, portanto, é o tipo e a natureza dos dados, e a ordem pela qual será mais conveniente apresentá-los à máquina. Depois de termos uma ideia clara do formato a que devem obedecer os dados de entrada e as saídas do sistema, podemos começar a definir exactamente o modo como o programa deve tratar os primeiros para obter as segundas.

A terceira fase consiste em conceber o programa propriamente dito recorrendo a algum método de trabalho, por exemplo a construção de um fluxograma. Esta representação ajuda a construir uma imagem da lógica do eventual programa. Depois de completado, o programador deve verificar completamente o fluxograma usando dados. Assume, assim, o papel do próprio computador de modo a verificar se a lógica está correcta, se os objectivos globais são atingidos e se foram tidas em conta todas as circunstâncias não usuais, como a colocação em primeiro lugar do número especial que devia estar no fim, ou a detecção de dados de entrada inválidos. Nenhum operador humano é perfeito, e os dados estarão sempre sujeitos a erros de teclado ou de perfuração no caso de cartões. O programa deve ser suficientemente robusto para não produzir uma saída errada devido às falhas humanas. Se um programa não toma isto em consideração não é um programa realista.

Assim, uma fase necessária do trabalho de programação é a preparação de dados de ensaio, isto é, de exemplos de dados correctos (para verificar a capacidade funcional do programa) e de dados incorrectos para verificar se o programa consegue aperceber-se da sua invalidade. Certos dados são difíceis de verificar,

como acontece no caso de um nome e de um endereço. Mas existem muitos exemplos de como se pode verificar dados incorrectos, normalmente recorrendo ao uso da operação de comparação lógica. A idade ou o sexo de um dado indivíduo podem ser verificados dentro de certos limites. Uma data de nascimento posterior ao ano 2000 seria bastante suspeita num programa que realiza esses cálculos para um censo; um sexo que não fosse masculino (M) ou feminino (F) também será necessariamente incorrecto. Uma conta de electricidade de um utilizador doméstico que apresenta um valor abaixo do aluguer do contador ou superior a um dado limite é igualmente suspeita. Por outro lado, o gasto de electricidade de um consumidor pode ser comparado com o gasto verificado em igual período do ano anterior, e uma variação excessiva pode também provocar suspeitas e ser apresentada na saída do computador para verificação. Uma pessoa que trabalhe um número excessivo de horas por semana ou apenas duas horas num mês constituirá também um caso a verificar.

Em todos estes casos um programa realista não aceita os dados e imprime-os juntamente com uma mensagem pedindo confirmação do dado. Em seguida apresentamos um exemplo relativo a contas de electricidade.

```
00100 INPUT X
00110 IF X < 3.50 THEN 900
00120 IF X > 500 THEN 920
    •
    •
    •
```

```
00900 PRINT «VALOR INFERIOR AO ALU-
    GUER», X
00910 GOTO 100
00920 PRINT «VALOR SUPERIOR A 5000$00», X
00930 GOTO 100
etc.
```

Deste modo, nenhuma conta que se encontre para além destes dois limites pode ser enviada aos consumidores. É precisamente tarefa dos *analistas de sistemas,* responsáveis pela concepção de um sistema computadorizado em função da sua aplicação, sugerir este tipo de validação de dados ao departamento de programação.

Depois de o programador ter a certeza de que o fluxograma funciona correctamente para os dados experimentados, pode começar a traduzi-lo em instruções de programa. Chama-se a isto *codificar,* e no caso dos programadores experientes é um processo praticamente mecânico.

O programa é em seguida submetido ao tradutor. Todos os erros de sintaxe produzidos na fase de codificação são indicados nas mensagens de diagnóstico enviadas pelo compilador ou interpretador. Estes erros são corrigidos e o programa é executado novamente. Depois de estar completamente livre de erros são inspeccionados os resultados obtidos usando dados de ensaio. Se os resultados, juntamente com uma identificação correcta dos dados não válidos, forem os esperados, o programa pode ser usado com dados verdadeiros. Se os resultados não são correctos, o programador deve rever a lógica do fluxograma. A figura 7.1 resume estas várias fases.

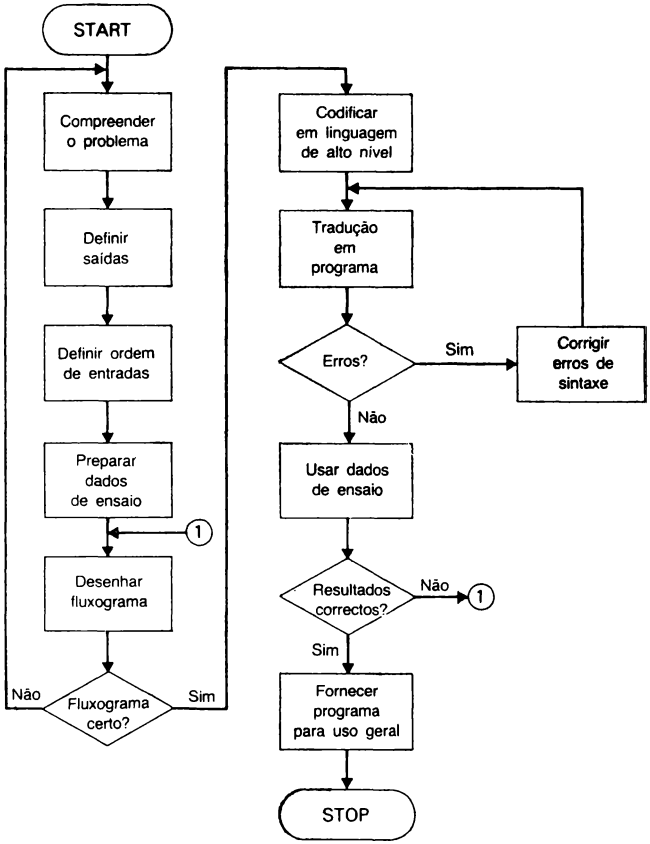


Figura 7.1

Documentação

A escrita de um programa de acordo com o esquema da figura 7.1 é obviamente uma parte importante de qualquer trabalho de programação, mas existe um outro

aspecto importante, em particular no caso de programas «a sério». A documentação adequada de um programa é essencial. Os utilizadores do programa necessitam de saber o que pode ou não fazer; o modo como preparar dados para o programa e como interpretar a saída. Em segundo lugar, os programas com algum comprimento tendem a ser usados repetidamente, talvez durante muitos anos. Podem-se verificar alterações durante os meses ou anos intervenientes. Um programa de salários deve ser suficientemente flexível para ter em consideração as alterações dos impostos e das contribuições; uma companhia pode utilizar recipientes em vidro para um dado produto mas depois mudar para recipientes em plástico. Os programas devem portanto admitir alterações. Por outro lado, um programa ao ser escrito pode ter uma série limitada de objectivos, mas admitir a possibilidade de ser ampliado mais tarde de modo a aumentar o número de objectivos a realizar.

Só se existir uma documentação bastante clara é possível a outras pessoas alterar ou modificar um programa existente. Se a documentação for bastante fraca (ou não existir) é muitas vezes mais barato e mais fácil reescrever todo o programa, perdendo portanto todo o tempo e esforço dispendidos no programa original.

Apresenta-se em seguida um tipo de documentação. Talvez não se adapte a todas as situações possíveis; é de facto muitas vezes preferido um outro tipo de documentação. Uma aplicação de natureza matemática ou de engenharia pode necessitar de pormenores sobre a teoria matemática usada; uma aplicação comercial requer necessariamente pormenores sobre todos os fichei-

ros de dados ligados ao programa. No entanto, o conjunto de dados de documentação apresentado em seguida é o usado em muitos departamentos e de facto é apropriado a bastantes situações.

A documentação pode tornar-se bastante volumosa, pelo que a estes conjuntos de informações se chama «ficheiro de documentação». É possível dividi-lo em quatro secções principais:

1) Identificação:
   Página de título — nome do programa
   Conteúdo do ficheiro de documentação
   Autor(es)
   Versão em uso e data(s)
   Descrição geral do programa
   Linguagem de programação usada

2) Especificação do programa:
   Teoria matemática/Ficheiros de dados comerciais usados
   Descrição completa do programa
   Restrições ou limitações do programa.

3) Informações para o utilizador:
   Detalhes sobre os dados de entrada
   Descrição das saídas
   Sistema requerido para executar o programa
   Instruções operativas para os utilizadores

4) Informação de programação:
   Fluxograma detalhado ou outro método esquemático
   Lista de nomes dos dados

Listagem do programa
Detalhes sobre os dados de teste usados
Exemplo de saída usando dados de teste
Informações sobre o desenvolvimento do programa
Instruções de utilização para os operadores

Listagem do programa: expressão usada para designar a lista completa de instruções do código-fonte.

Versão em uso: Se um dado programa foi modificado e se passou a usar uma versão mais desenvolvida, é necessário anotar o facto.

Lista de nomes: Significado dos nomes dos dados; por exemplo:

IMP = imposto
HRS = horas de trabalho
LIQ = salário líquido
BRT = Volume bruto dos salários

Informações sobre desenvolvimento do programa: Nesta secção descrevem-se todas as dificuldades encontradas durante o desenvolvimento do programa e o modo como foram ultrapassadas.

Sistema requerido para execução: Indicam-se aqui os pormenores quanto ao tipo de hardware necessário para executar o programa, juntamente com todos os pormenores relevantes sobre as unidades periféricas de que o sistema deve dispor: por exemplo, mínimo de memória central requerida; tipo de dispositivos auxiliares; uso de VDU's (visual display units, visores semelhantes aos de televisão) ou de leitores de cartões perfurados; uso de impressoras, etc.

VIII

ESTUDO INTRODUTÓRIO DAS LINGUAGENS DE ALTO NÍVEL

Existem muitas linguagens de alto nível; este facto coloca o problema da escolha da mais apropriada para uma dada utilização. Se recordarmos que as linguagens de alto nível estão essencialmente orientadas para um dado tipo de problemas, torna-se óbvio que uma das principais razões para a escolha de uma linguagem em vez de qualquer outra depende do tipo de problemas que se pretende resolver. Algumas das categorias de problemas mais vulgares são os científicos (matemáticos, de engenharia), os comerciais, os de simulação e os de tratamento de texto.

Todos os computadores são vendidos pelo fabricante juntamente com um certo número de compiladores. Assim, a PL/1 pode ser usada nos computadores IBM. Um computador CDC não pode traduzir programas PL/1 porque não é vendido com o compilador adequado.

Em seguida vamos apresentar alguns dados históricos sobre as linguagens de alto nível mais vulgares

e discutir muito resumidamente as suas características mais interessantes.

FORTRAN (FORmula TRANslator)

A primeira linguagem de alto nível que surgiu no mercado foi a FORTRAN, desenvolvida em 1956 para utilização numa máquina da IBM. Foi concebida para resolver problemas matemáticos e de tipo científico capazes de serem apresentados numa simples forma algébrica; assim,

$$x = a + b - c$$

pode ser escrito no programa sob a forma

$$X = A + B - C$$

Em meados da década de 60 passou a ser muito usada em bastantes máquinas diferentes, resultando numa variedade de dialectos entre os quais os mais importantes são o IBM FORTRAN II e o IBM FORTRAN IV. Em 1962, o American Standards Institute (ASA) organizou um grupo de trabalho encarregue de definir as especificações da linguagem que deveria ser usada em todos os computadores. O êxito desta iniciativa assegurou que a maior parte dos fabricantes de computadores viessem a produzir compiladores para a linguagem FORTRAN de acordo com essas especificações. Foi definido um novo conjunto de especificações sob o nome de FORTRAN 77, contendo algumas características que não fazem parte da FORTRAN IV. A FORTRAN é muito usada em bastantes universidades

dedicadas a estudos científicos ou de engenharia. Foi também usada com êxito em problemas de tipo comercial, se bem que isto sirva apenas para demonstrar a sua facilidade de uso e a possibilidade de a utilizar em qualquer computador, mais do que uma capacidade inerente para satisfazer uma vasta gama de aplicações. É a linguagem mais *portátil* hoje existente, isto é, qualquer programa escrito em FORTRAN normalizada pode ser usado com êxito em praticamente todos os computadores.

ALGOL (ALGOrithmic Language)

Tal como a FORTRAN, esta linguagem foi concebida para a solução de problemas numéricos e científicos, pouco diferindo da FORTRAN em termos de potencialidades. A ALGOL foi originalmente desenvolvida em 1958, de onde decorre o seu nome — ALGOL 58. Foi revista em 1960, e o interesse da nova versão (ALGOL 60) decorre da elegância da estrutura, algo que sobreviverá bastante ao seu desaparecimento.

A ALGOL 60 trouxe para o campo dos computadores conceitos estruturais, um rigor de definição da linguagem, e uma certa disciplina dos métodos de programação. Estas características tinham-se mantido na FORTRAN, pelo contrário, basicamente simples.

A versão mais recente, e mais poderosa, é a ALGOL 68. Desenvolve todos os diversos componentes da linguagem dando-lhes um elevado grau de sofisticação, permitindo no entanto o uso e o aprendizado fáceis de uma sua variante mais elementar.

APL (A Programming Language)

A APL, concebida para processamentos em time-sharing, foi desenvolvida com base na notação matemática inventada por Kenneth E. Iverson em 1962. Os defensores da APL afirmam tratar-se de uma linguagem coerente, concisa e poderosa. Grande parte das suas capacidades decorre do rico conjunto de operadores de que dispõe, permitindo a fácil manipulação de matrizes e de *arrays* (quadros) de ordem elevada.

Os programas APL podem tratar texto ou valores numéricos. Têm sido usados em montagens de texto, simulação de sistemas e métodos educativos.

COBOL (COmmon Business Oriented Language)

A COBOL transformou-se numa linguagem de negócios internacional. Surgiu em 1958, devendo-se o seu aparecimento às necessidades do Departamento de Defesa americano no campo do tratamento dos seus muitos problemas comerciais quotidianos. Pretende ser uma espécie de inglês simplificado, e nestas condições os programas escritos em COBOL tendem a ser um tanto palavrosos. As suas estruturas de definição de dados foram concebidas de modo a permitir o tratamento fácil de ficheiros.

RPG (Report Program Generator)

A RPG constitui um exemplo de uma linguagem comercial popular que é igualmente especializada. As

linguagens como a FORTRAN e a COBOL são linguagens de uso geral, isto é, são usadas por pessoas que se vêem obrigadas a escrever muitos tipos de programas diferentes e desejam utilizar a mesma linguagem para todos eles. A RPG, pelo contrário, só pode ser usada para produção de relatórios, mas relatórios num sentido lato incluindo a preparação de facturas e de pagamentos em cheque, além de «relatórios» normais.

Esta linguagem foi desenvolvida pela IBM devido aos pedidos dos seus clientes no sentido da criação de um mecanismo fácil e económico para produção de «relatórios», tendo sido lançada no mercado em 1961.

BASIC (Beginners All-Purpose Symbolic Instruction Code)

Trata-se de uma linguagem concebida principalmente para uso em sistemas computadorizados usando processamento em time-sharing. Para entender uma linguagem de processamento interactivo como a BASIC não é necessário aprender técnicas de programação complexas. Foi de facto concebida para aqueles que não possuem qualquer experiência de uso de computadores ou de escrita de programas para computadores. A BASIC possui poucas regras gramaticais, podendo-se dizer que é mais orientada para o utilizador do que para o sistema. Assemelha-se à FORTRAN em muitos aspectos, utilizando uma notação matemática normalizada, mas adapta-se igualmente bem a aplicações comerciais. Pode ser aprendida em algumas horas de estudo concentrado e, apesar de simples, é flexível e razoavelmente

poderosa. Esta linguagem foi desenvolvida pelos professores John Kemeny e Thomas Kurtz em meados dos anos 60, no Dartmouth College dos Estados Unidos, para uso num sistema de time-sharing, mas a maior parte da sua estrutura está igualmente adaptada a um processamento por lotes. Devido à sua simplicidade e à sua orientação para o utilizador, é uma linguagem óptima para uso em educação.

PASCAL

Este nome, que corresponde de facto ao conhecido matemático francês e não a uma simples colecção de iniciais, foi dado a uma daquelas linguagens de alto nível que os respectivos entusiastas afirmam ser bastante agradável de usar. Foi originalmente concebida por Niklaus Wirth em 1968, estando relacionada com a ARGOL, e o seu primeiro compilador operacional surgiu no mercado em 1970. Uma das suas características mais importantes consiste em permitir ao programador estruturar dados da maneira que ele próprio deseja. A PASCAL standard foi originalmente desenvolvida e usada na série de computadores CDC 6000, mas tem sido desenvolvida para uso noutras máquinas. É uma linguagem bem concebida para aplicações educativas, estando a tornar-se bastante popular.

Nota: As descrições das linguagens ALGOL, RPG, BASIC e PASCAL são reproduzidas de Hunt e Shelley, *Computers and Commonsense,* 2.ª ed., 1979, págs. 141-5, sob autorização da Prentice-Hall International Inc., Londres.

ÍNDICE

I — O SISTEMA COMPUTADOR ... 7
Dispositivos de dois estados ... 8
O sistema binário 8
Concepção de um computador ... 10
Dispositivos de entrada/saída ... 12
Memória limitada 13
Um sistema computador 14
Codificação 15
Porque são tão úteis os computadores? .. 16

II — UMA INTRODUÇÃO À PROGRAMAÇÃO ... 19
Operações de entrada ... 19
Dados 20
Operações de saída 21
Movimento de dados e criação de listagens ... 21
Operações aritméticas 22
Comparações e operações lógicas 23

III — OS FLUXOGRAMAS — UM MODO DE RESOLVER PROBLEMAS 33

IV — MAIS PRINCIPIOS DE PROGRAMAÇÃO ... 49
Nomes de dados 52
Armazenamento de valores pelos respectivos nomes 54
Conjuntos de caracteres 55
Instruções «DIM» e «FOR-NEXT» ... 63

V — TRADUTORES DE LINGUAGENS E NÍVEIS DE LINGUAGEM 75
Níveis de linguagem ... 75
Linguagens de baixo nível ... 75
Linguagens de alto nível ... 76
Tradutores de linguagens ... 77
Tipos de erros ... 78
Erros de sintaxe ... 79
Erros de lógica 79
Erros de dados . 80

VI — SISTEMAS «BATCH» E «TIME-SHARING» ... 81
Processamento por lotes 81
«Time-Sharing» — Processamento interactivo ... 82
Interpretadores 83
Escolha do sistema de processamento ... 84

VII — O QUE ESTÁ EM CAUSA AO ESCREVER UM PROGRAMA 87
Fases da escrita de um programa ... 87
Documentação 91

VIII — ESTUDO INTRODUTÓRIO DAS LINGUAGENS DE ALTO NÍVEL 95
FORTRAN (Formula Translator) 96
ALGOL (Algorithmic Language) 97
APL (A Programming Language) 98
COBOL (Common Business Oriented Language) 98
RPG (Report Program Generator) 98
BASIC (Beginners All-Purpose Symbolic Instruction Code) 99
PASCAL ... 100

Este livro acabou de se imprimir
em 1983
. para a
EDITORIAL PRESENÇA, LDA.
na
SOC. COM. BEIRA DOURO, LDA.
1500 Lisboa

O ABC DA PROGRAMAÇÃO foi concebido pensando naqueles que se interessam por informática, mas que pouca ou nenhuma experiência possuem na área da programação. Além de fazer uma breve apresentação do sistema computador e suas vantagens, esta obra enumera e explica quais os princípios que presidem à programação, os níveis e tradutores das linguagens, os tipos de erros que podem ocorrer e qual o sistema a escolher quando se pretende escrever um programa. É, em síntese, um excelente guia introdutório às linguagens de alto nível como a FORTRAN, COBOL, BASIC e PASCAL entre outras.

EDITORIAL PRESENÇA / MARTINS FONTES