A Inteligência Artificial no computador Spectrum mostra-lhe como pode implementar rotinas IA no seu micro doméstico, transformando-o numa máquina inteligente que pode manter uma conversação consigo, dando-lhe conselhos racionais, aprendendo de si (e ensinando-o), escrevendo mesmo programas para si.

O livro explica a IA a partir de princípios básicos, partindo do princípio de que o leitor não tem qualquer conhecimento prévio do assunto. Todos os aspectos importantes de IA são cobertos, sendo completamente ilustrados com programas exemplificativos.

Durante muitos anos os livros e os filmes de ficção científica têm apresentado computadores «inteligentes», os quais parecem ser, pelo menos, iguais ao ser humano. Ainda que algumas das características aí apresentadas não passem de simples invenções, uma investigação mais profunda sobre a IA tem aproximado da realidade muitas das ideias.

Keith e Steven Brain constituem uma equipa de pai e filho, tendo já publicado diversos *best-sellers* na área da informática. São ambos colaboradores regulares do *Popular Computing Weekly*.

ARTE DE VIVER 95

ARTE DE VIVER 95 keith e steven brain spectrum inteligência artificial no spectrum

keith e steven brain

inteligência artificial no spectrum

faça o seu microcomputador pensar

PUBLICAÇÕES EUROPA-AMÉRICA

keith e steven brain

inteligência artificial no spectrum

faça o seu microcomputador pensar

ARTE DE VIVER

PUBLICAÇÕES EUROPA-AMÉRICA

Título original: Artificial Intelligence
on the Spectrum Computer

Tradução de João Manuel Coelho da Rocha

Capa: estúdios P. E. A.

First published 1984 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12-13 Little Newport Street,
LONDON WC2R 3LD

Editor: Francisco Lyon de Castro

PUBLICAÇÕES EUROPA-AMÉRICA, LDA.
Apartado 8
2726 MEM MARTINS CODEX
PORTUGAL

Edição n.º 33 095/3981

Execução técnica:
Gráfica Europam, Lda.,
Mira-Sintra — Mem Martins

ÍNDICE

NOTAS EXPLICATIVAS DO TRADUTOR

1. Na representação esquemática dos ordinogramas apresentados o sentido de fluxo da lógica da sua sistematização obedece à seguinte norma convencional: as setas são apenas indicadas em troços em que o mesmo fluxo tome a direcção para a esquerda ou para cima.

2. As palavras reservadas à linguagem de aplicação BASIC (instruções, comandos, palavras básicas na formação de determinado tipo de condições) foram estritamente mantidas em inglês, devido ao facto de a linguagem ter mesmo de observar rigorosamente a sua utilização.

As designações de funções criadas pelo próprio programador foram, quando conveniente, adaptadas à respectiva tradução equivalente do seu significado em português.

Índice pormenorizado

CAPÍTULO 6

Planificando o seu sistema para que aprenda por si próprio
Deixando que o computador elabore as suas próprias regras para dois objectos — um Spectrum mais expansível — observando-se o que se vai passando.

CAPÍTULO 7

Combinação mesclada
Recuperação de informação da mente humana — Código 'Soundex' — um programa de computorização para a conversão de nomes — restabelecimento de informação.

CAPÍTULO 8

Reconhecendo modelos
Simulação da acção dum sensor óptico — inserção em frases — uma breve interrupção dum segmento.

CAPÍTULO 9

Um instrutor inteligente
Interrogações e respostas — mantendo um marcador — transferindo o ênfase dado às perguntas para áreas de dificuldade — facilitando ou dificultando as perguntas.

CAPÍTULO 10

Combinando tudo
Travar conversação com o computador — tomando decisões, referenciadores de custos e referenciadores de lucros — o vendedor de computadores.

Introdução

A inteligência artificial é sem dúvida uma área da maior relevância no desenvolvimento da computadorização, a qual terá profundos efeitos em toda a nossa vida numas poucas de décadas próximas. O principal objectivo deste livro é introduzir o leitor em alguns dos conceitos envolvidos na inteligência artificial e mostrar-lhe como se podem desenvolver rotinas «inteligentes» em BASIC do *Sinclair*, as quais podem ser subsequentemente incorporadas nos seus programas particulares. É aqui assumida somente uma noção superficial sobre o BASIC, dedicando-se o livro a partir de princípios fundamentais, uma vez que acreditamos ser tal essencial caso pretenda realmente compreender os problemas envolvidos no desenvolvimento de sistemas inteligentes e como se preparar para os ultrapassar e resolver.

O formato básico do livro baseia-se em ideias assentes, sendo as rotinas adequadas construídas passo a passo, explorando e comparando possibilidades alternativas onde quer que sejam exequíveis. Antes de lhe proporcionar uma série completa de programas, encorajamo-lo a experimentar por diferentes aproximações de forma a permitir-lhe observar os resultados obtidos por si próprio. São incluídos ordinogramas pormenorizados da maioria das rotinas. O ênfase principal das rotinas é conferido aos aspectos da IA (inteligência artificial), tendo-se por conseguinte evitado o «empastelamento» da exposição do *écran*, uma vez que isso tende a encobrir o significado do programa em si. Em determinados sítios poder-se-á reparar que linhas singulares são redundantes, porém foram deliberadamente incluídas no interesse da claridade do fluxo do programa. Tanto quanto possível, a reinscrição de linhas foi exaustivamente evitada, mas a modificação de linhas é lugar-comum. Todas as listagens no livro se encontram na forma como aparecem no *écran*. Na maioria dos casos, espaçamentos e

parênteses foram utilizados livremente para facilitar a leitura das listagens; porém avisamo-lo de que alguns dos espaços e parênteses são essenciais, de modo que não se deixe tentar a removê-los todos. Todos os caracteres foram listados em maiúsculas, uma vez que nos parece ser assim mais fácil fazer a leitura das listagens; todavia, a inscrição em minúsculas pode igualmente ser usada, se assim o desejar. Lembre-se, contudo, que os caracteres em maiúsculas e em minúsculas não são combináveis, devendo-se, pois, ser consistente quanto à forma da inscrição. Todas as rotinas foram rigorosamente testadas, tendo todas as listagens sido igualmente verificadas bastante cuidadosamente, de maneira que esperamos que não depare com qualquer tipo de «gralhas». É um facto menos feliz que a maioria dessas «gralhas» surjam em resultado de «erros de inscrição» cometidos pelo utilizador. Pontos e vírgulas e vírgulas podem parecer muito insignificantes, porém a sua ausência pode ter efeitos bastante profundos!

A inteligência artificial vem aumentando de importância cada dia que passa. Assim, esperamos que este livro lhe proporcione uma útil compreensão nesta área. Quem sabe — se realmente se dedicar ao assunto pode bem ser capaz de persuadir a sua máquina a ler-lhe o nosso próximo livro por si mesma!

Agradecimentos sinceros são endereçados a Valerie James, que inscreveu e testou a maioria dos programas, os quais foram modificados a partir dos programas do nosso *Commodore 64 Dragon* original para o BASIC do *Sinclair* desta versão particular. Devemos uma vez mais agradecer a Liz, que aprendeu até ao momento o suficiente sobre computadores para se especializar no quadro de comandos dum forno de microondas, de maneira que o nosso café seja servido à temperatura ideal — ainda que ela esteja pacientemente à espera de alguém que desenvolva um autómato que despeje o caixote do lixo.

KEITH e STEVEN BRAIN
Groeswen, Março de 1984

CAPÍTULO 1

Inteligência artificial

Ficção

Durante gerações, os autores de obras de ficção científica têm imaginado e previsto o desenvolvimento de máquinas inteligentes que possam executar muitas das funções e tarefas realizadas pelo próprio ser humano (ou mesmo ultrapassando-o em muitas áreas), vindo a opinião pública a ser progressivamente influenciada sem dúvida alguma por tais visões e perspectivas. A imagem mais comum dum *robot* é a de uma máquina inteligente, de configuração antropomórfica (humana) na sua generalidade, a qual é eapaz de executar independentemente instruções que lhe sejam transmitidas unicamente de uma maneira muito generalizada.

Como é óbvio, a maioria das pessoas têm adquirido opiniões de índole maléfica quando se trata da abordagem tecnológica, de forma que nas primeiras versões historiadas estes *robots* tinham tendência para possuírem má imagem, sendo protagonistas determinantes no desempenho de papéis tradicionais conotados com o «mau da fita», possuindo uma força quase indestrutível e sem qualquer consciência interior. O visionário Isaac Asimov concebeu uma extensa série de histórias abordando o conceito dos «*robots* positrónicos», tendo sido provavelmente o primeiro autor que realmente se apercebeu e apoderou efectivamente das realidades inerentes a esta situação. Estabeleceu as suas famosas «Três Leis da Robótica», que determinaram as regras fundamentais que devem constar da construção de qualquer máquina

capaz de operacionalidade independente — no entanto é interessante notar que ele não foi capaz de prever a altura em que a raça humana aceitaria a presença efectiva de tais *robots* na própria vida terrestre.

A produção cinematográfica *Star Wars* (*Guerra das Estrelas*) introduziu os *robots* especializados R2D2 e C3PO, mas parece-nos que as suas características de *design* eram um pouco estranhas. Talvez haja uma Confederação Interplanetária de Robots e uma disputa de áreas impedindo a comunicação directa entre seres humanos e o R2D2. No filme *The Stepford Wives*, os vários maridos duma certa localidade reuniram-se e tiveram a (boa?) ideia de converterem as suas mulheres em andróides que fizeram automaticamente aquilo que era esperado deles, porém o resultado revelou os perigos da necessidade de continuamente se ter de reforçar a transformação metamórfica com estímulos externos! Talvez uma das esperanças para a humanidade seja a de que qualquer ser estranho, perigoso, que procure dominar-nos não tenha visto o filme *Battlestar Galactica*, construindo por conseguinte *robots* do tipo dos Cylon, os quais, à semelhança dos tradicionais e antigos Invasores do Espaço, sejam sempre eventualmente derrotados por serem totalmente previsíveis.

Claro está que os computadores inteligentes também apareceram em invólucros sem braços nem pernas, ainda que acendendo luzes, o que parece ser obrigatório. O sistema de entradas e saídas *(input/output)* deve ser obviamente oral, porém a já antiquada voz metálica passou definitivamente de moda em favor de uma personalidade um pouco mais definida. Caso todos os invólucros se assemelhem, então isso deve ser uma boa ideia, mas por favor não faça com que todos os seus se pareçam com o sargento-ajudante Zero de *Terrahawks!* O modelo de Michael Knight parece ser um tipo de aparelho razoável para conversação, sendo certamente preferível ao ESCRAVO untado e ao inofensivo ORAC de *Blake's Seven*. O ORAC parecia ter uma enorme quantidade de desprezo dentro daquela pequena caixa cheia de perspicácia, porém outros escritores têm apreciado as dificuldades que podem ser provocadas caso se faça com que a personalidade da máquina seja o mais próxima possível da do próprio ser humano.

No filme de Arthur C. Clarke *2001 — Odisseia no Espaço*, o computador de inteligência de vanguarda HAL teve eventualmente um esgotamento nervoso quando se viu confrontado com demasiadas responsabilidades; mas em *Dark Star* a bomba inteligente mostrou-se bastante contente por poder discutir questões existencialistas com o capitão Doolittle, sentindo-se contudo relutante à ideia de cancelar o

seu tempo de detonação planeado, ainda que se encontrasse bloqueada no porão da bomba. Em *Um Restaurante nos Confins do Universo*, o valor da Transportadora de Pessoas Verticais Felizes da Corporação Cibernética Sirius foi significativamente reduzido quando se recusou a funcionar uma vez que podia prever o futuro, compreendendo que caso o fizesse era provável que fosse anulado; e o Sintetizador de Bebidas Nutrientes foi obviamente projectado pela British Rail Catering, visto que sempre produziu uma bebida que era «quase, mas não completamente, diferente do chá».

Mais temas igualmente preocupantes têm também surgido recentemente. A característica mais significativa do filme *Wargames* não tinha a ver com o facto de que alguém conseguisse ter acesso a JOSHUA (o computador da Defesa dos E. U.), mas uma vez que a máquina começasse a jogar a guerra termonuclear não pararia até que alguém tivesse ganho o jogo. E em *The Forbin Project* os computadores americano e russo reuniram-se, tendo decidido que os seres humanos são de qualquer forma pouco mais ou menos irrelevantes. Certamente que se for Marvin, o Andróide Paranóico, e tiver um cérebro do tamanho dum planeta e possuir uma Personalidade Humana Genuína, pode ser bem sucedido sem o recurso a armas, conseguindo provocar a confusão da máquina inimiga, destruindo-a pela base enquanto se detiver a discutir os seus problemas pessoais.

Realidade

A definição e o reconhecimento da inteligência duma máquina é assunto da mais acesa e furiosa discussão entre os especialistas na matéria. A definição mais amplamente aceite é a proposta pela primeira vez por Alan Turing, por alturas dos finais da década de 40, quando os computadores tinham dimensões do tamanho de casas e eram mesmo mais raros do que o é uma régua de cálculo hoje em dia. Antes de tentar apresentar uma série de critérios que devem ser satisfeitos, ele perspectivou uma visão muito mais vasta acerca do problema. Considerou logicamente que a maioria dos seres humanos aceitam o facto de a maioria dos outros seres humanos serem inteligentes, e que, por consequência, se um homem não pode determinar com exactidão se se encontra ou não a lidar com um outro homem (ou mulher), ou somente com um computador, então deve

aceitar o facto de que tal máquina é inteligente. Este princípio forma a base do famoso «Teste de Turing», no qual um operador tem de manter uma conversação bidireccional com outra entidade através dum teclado, usado como via de comunicação, tentando que a outra parte se revele como sendo realmente uma máquina ou simplesmente outro ser humano — muito desajeitado!

Muitos romances de ficção baseiam-se neste teste, porém o nosso favorito é aquele em que um candidato a um emprego é instalado defronte dum teclado, deixando-se que execute aquilo que quiser por si próprio. Como é evidente, ele apercebe-se da importância deste teste para as suas perspectivas de carreira, e assim luta intrepidamente para descobrir o segredo, aparentemente sem sucesso. Todavia, depois de algum tempo decorrido, o entrevistador regressa, dá-lhe um aperto de mão, e felicita-o mais ou menos com as seguintes palavras: «Boa, velho amigo, a máquina não conseguiu determinar se você era ou não humano, de modo que é exactamente aquilo de que precisamos como um dos inspectores da Direcção-Geral das Contribuições e Impostos!»

Toda a gente já teve oportunidade de ver em anúncios televisivos que as técnicas de *design* publicitário baseadas na utilização de computadores são agora muito comuns, e que os *robots* industriais são quase os únicos habitantes das linhas de produção da indústria automóvel (seguindo o lema afixado no pára-brisas dum carro que afirma o seguinte: «Desenhado por um computador, construído por um *robot*, e conduzido por um idiota.»). De facto, a maioria destes *robots* industriais possuem na realidade uma inteligência mínima, dado que simplesmente seguem um programa pré-definido, sem terem grande capacidade no que diz respeito à tomada de decisões efectivas. Mesmo o impressionante *robot* de pintura automatizada com *spray* segue fielmente o modelo que lhe é prescrito, e aprende quando um operador desloca manualmente o seu braço, não podendo aprender a lidar com qualquer novo objecto sem prévia intervenção humana.

Por outro lado, a geração vindoura de *robots* possui sensores e *software* mais sofisticados, o que lhes permite determinar a forma, a cor e a textura dos objectos, assim como tomar decisões mais racionais. Qualquer pessoa que tenha visto programas dos lendários concursos *Micromouse*, onde definitivamente «bichos» eléctricos despelados se apressavam independente e resolutamente (?) para o centro do labirinto, não se surpreenderá com a nossa fé no futuro de *robots* inteligentes, ainda que pareça ser de menor importância o facto de lhes serem conferidos na sua constituição estrutural dois braços e duas pernas.

Outra área importante onde a inteligência artificial se encontra actualmente em fase de exploração é a relacionada com os sistemas inteligentes (especializados), muitos dos quais podem proceder tão bem como (ou mesmo melhor que) os especialistas humanos, especialmente se se tomar em consideração a previsão meteorológica. Estes sistemas podem ser peritos em qualquer número de áreas ou assuntos mas, em particular, são de importância crescente em diagnósticos e tratamentos médicos — ainda que os profissionais médicos não tenham de se preocupar demasiado com tal situação, uma vez que existirá sempre lugar para esses profissionais: os computadores não podem confortar os pacientes.

Um obstáculo de maior relevância para o utente de computadores numa escala mais ampla tem a ver com a ignorância e a obstinação dos usuários, que lerão apenas as instruções como um último recurso e que esperam que a máquina seja capaz de compreender por si própria todas as suas pequenas peculiaridades. O processamento da «linguagem natural» é consequentemente uma área de crescimento da maior relevância, e a «quinta geração» dos computadores será, pois, muito mais agradável de utilizar, bem como mais familiar.

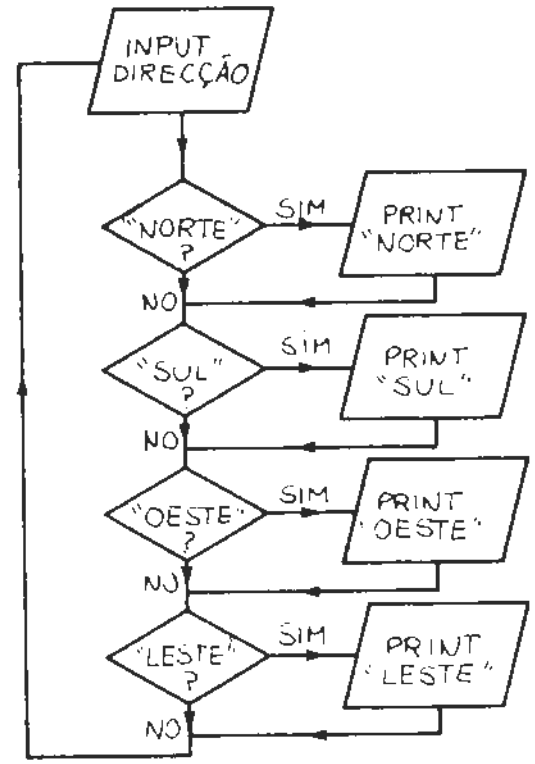
A maior parte dos trabalhos sérios da inteligência artificial utiliza linguagens mais adequadas (mas algo exóticas) que o BASIC, tais como LISP e PROLOG, as quais são pouco inteligíveis para o usuário médio, não sendo provavelmente adquiríveis para o seu micro doméstico, em qualquer caso. As rotinas em BASIC que se seguem não podem por conseguinte ser pressupostas dar-lhe a chave para o domínio do mundo, ainda que lhe devam dar uma apreciação razoável das possibilidades e dos problemas que a inteligência artificial comporta.

CAPÍTULO 2

Seguindo simplesmente instruções

Uma vez que o seu computador é na realidade totalmente ignorante, só se pode conversar com ele em termos muito simples. O primeiro passo, usado em muitos jogos simples de divertimento, é possuir uma série de ordens (instruções) preestabelecidas para as quais existem determinadas respostas rígidas. Vamos começar por uma vista de olhos ao modo de dar directrizes sobre qual o caminho a seguir. À primeira vista, a maneira mais fácil de programar isto parece ser pedir um INPUT (entrada de dados, ou de informação generalizada) a partir do utilizador (operador) e inscrever uma linha separada com a cláusula de condição alternativa IF-THEN (se-então) para cada possibilidade (ver o ordinograma 2.1).

```
 100 PRINT "DIRECCAO?";
 120 INPUT I$
 200 IF I$="NORTE" THEN PRINT "N
ORTE"
 210 IF I$="SUL" THEN PRINT "SUL
"
 220 IF I$="OESTE" THEN PRINT "O
ESTE"
 230 IF I$="LESTE" THEN PRINT "L
ESTE"
 350 GO TO 100
```

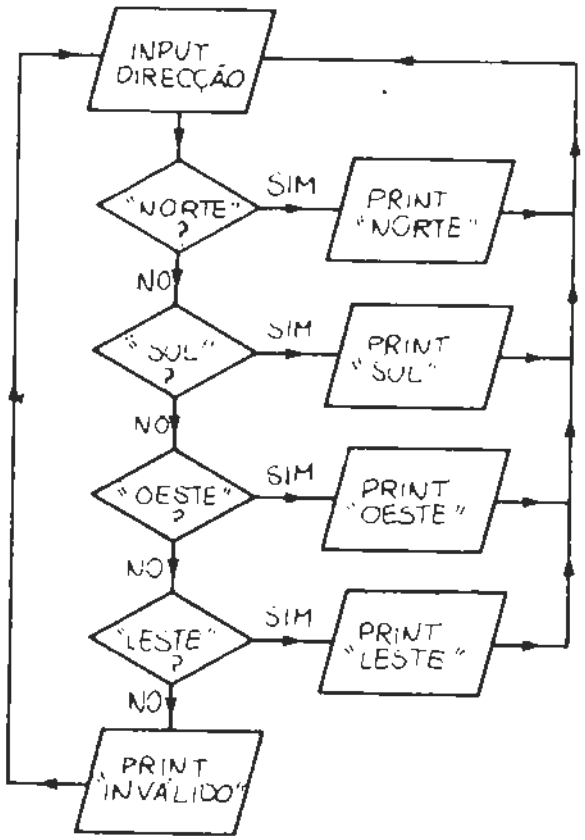
Ordinograma 2.1 — *Dando direcções*

Se inscrever qualquer outra coisa que não sejam as palavras das quatro teclas de comando especificadas, nada será impresso excepto um outro pedido de entrada (INPUT). Seria mais agradável se o computador indicasse mais claramente o facto de esse comando não ser válido. Isso poderia ser feito incluindo um teste que mostra que nenhuma das palavras de comando foi encontrada, tornando-se contudo bastante morosa tal procura, e com efeito impossível quando se tem uma extensa lista de palavras válidas.

```
 240 IF I$<>"NORTE" AND I$<>"SUL
" AND I$<>"OESTE" AND I$<>"LESTE
" THEN PRINT "PEDIDO INVALIDO"
```

Por outro lado, acrescentando GOTO 100 (ir para 100) ao fim de cada linha IF-THEN (se-então) obrigará a um salto directo de volta a INPUT (entrada) quando um comando válido é detectado. Se todos os testes IF não forem verdadeiros, então o programa dirige-se directamente até à linha (passo) 240, que imprime um aviso. Proceder a

saltos directos de retorno quando uma palavra válida é encontrada é em qualquer caso uma boa ideia, uma vez que poupa ao sistema a necessidade de fazer testes sempre que a resposta tenha já sido encontrada (ver o ordinograma 2.2).



Ordinograma 2.2 — *Eliminando testes desnecessários*

```
 200 IF I$="NORTE" THEN PRINT "N
ORTE": GO TO 100
 210 IF I$="SUL" THEN PRINT "SUL
": GO TO 100
 220 IF I$="OESTE" THEN PRINT "O
ESTE": GO TO 100
 230 IF I$="LESTE" THEN PRINT "L
ESTE": GO TO 100
 240 PRINT "PEDIDO INVALIDO"
```

Isto terá reflexos no *écran*, mas, como é óbvio, não fará na realidade o que quer que seja. Como modelo para trabalhar, começaremos numa posição definida para X = 0 e Y = 0, indicando-se movimentação para maior ou menor em relação a este ponto. Note-se que

as variáveis numéricas (inteiros) são usadas sempre que possível, uma vez que são processadas mais rapidamente que os números reais, o que também elimina a possibilidade de interferência com variáveis reservadas.

```
10 LET X=0: LET Y=0
```

Precisamos agora de juntar a resposta real ao comando, bem como a mensagem indicativa de que isso foi compreendido (ver ordinograma 2.3).

```
 200 IF I$="NORTE" THEN PRINT "N
ORTE": LET Y=Y-1: GO TO 100
 210 IF I$="SUL" THEN PRINT "SUL
": LET Y=Y+1: GO TO 100
 220 IF I$="OESTE" THEN PRINT "O
ESTE": LET X=X-1: GO TO 100
 230 IF I$="LESTE" THEN PRINT "L
ESTE": LET X=X+1: GO TO 100
```

Esta modificação mostra realmente a sua posição apropriadamente, relativa à origem. De forma a poder observar o que vai acontecendo, e onde se encontra, acrescente uma impressão da sua posição actual:

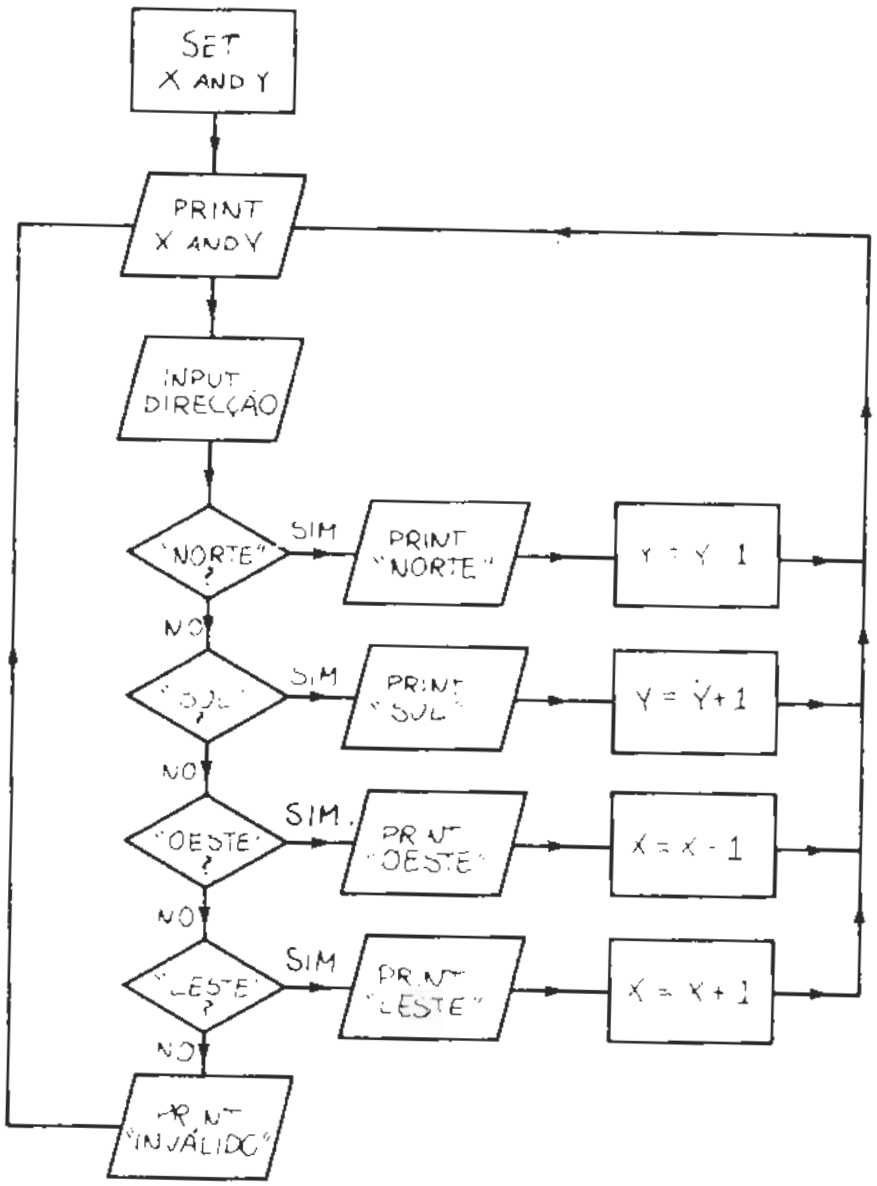
```
110 PRINT."X";X,"Y";Y
```

Utilização de sub-rotinas

Como é evidente, aquele era um exemplo muito simples e, particularmente onde os resultados das suas acções são mais complicados, é normalmente melhor colocar as respostas em sub-rotinas.

```
 200 IF I$="NORTE" THEN GO SUB 2
000: GO TO 100
 210 IF I$="SUL" THEN GO SUB 210
0: GO TO 100
 220 IF I$="OESTE" THEN GO SUB 2
200: GO TO 100
 230 IF I$="LESTE" THEN GO SUB 2
300: GO TO 100
```

```
2000 PRINT "IR PARA NORTE": LET
Y=Y-1: RETURN
2100 PRINT "IR PARA SUL": LET Y=
Y+1: RETURN
2200 PRINT "IR PARA OESTE": LET
X=X-1: RETURN
2300 PRINT "IR PARA LESTE": LET
X=X+1: RETURN
```



Ordinograma 2.3 — *Juntando uma resposta*

Mais versatilidade

Podia estender o uso destes testes IF-THEN infinitamente (ou, melhor dizendo, até à capacidade máxima da memória!), porém é uma maneira grosseira de fazer as coisas, que cria problemas quando se querem realizar programas mais sofisticados. Uma maneira mais versátil de lidar com palavras de comando e respostas é entrar com elas como DATA (conjunto de dados ou informações), lendo-as de seguida (READ) como tal, quando necessário. Se se colocarem os comandos e as respostas aos pares na declaração de DATA é muito mais difícil misturá-las e confundi-las, e é mais fácil lê-las (READ) (ver quadro 2.1).

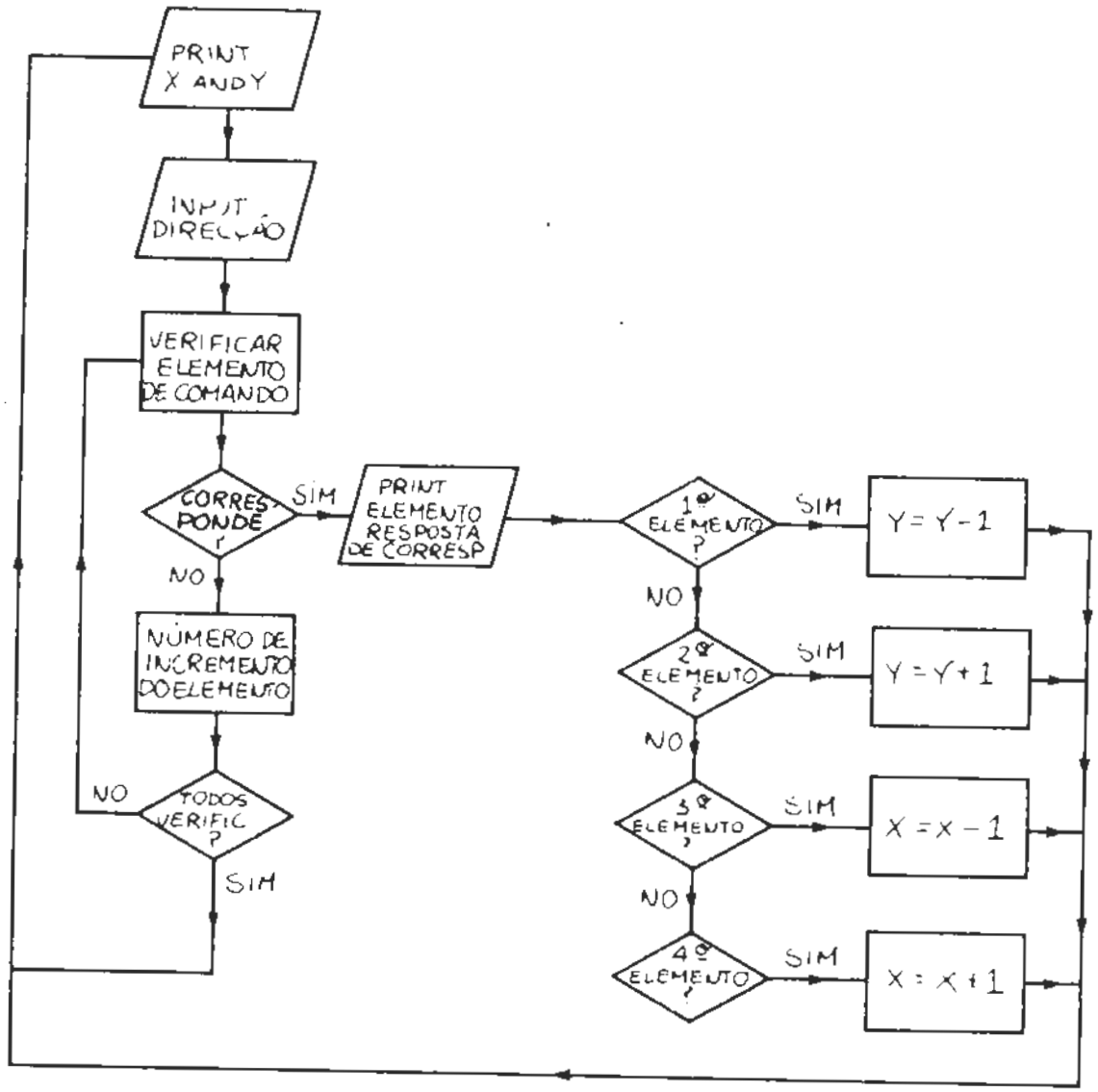
PALAVRA DE COMANDO (C$)	PALAVRAS DE RESPOSTA (R$)
NORTE	IR PARA NORTE
SUL	IR PARA SUL
OESTE	IR PARA OESTE
LESTE	IR PARA LESTE

Quadro 2.1 — *Comandos e respostas*

```
9000 DATA "NORTE","IR PARA NORTE
","SUL","IR PARA SUL","OESTE","I
R PARA OESTE","LESTE","IR PARA L
ESTE"
```

Todos estes testes IF-THEN podem agora ser substituídos por um simples ciclo que compara o seu INPUT (entrada) com cada palavra de comando, lendo (READ) cada um destes elementos a partir da DATA como C$ (ver ordinograma 2.4) e imprimindo a resposta R$ caso seja encontrada uma combinação. Note-se que o indicador de DATA deve ser restabelecido (RESTORE), de modo a começar-se sempre a ler (READ) a partir do primeiro item na lista de DATA.

```
 200 RESTORE : FOR N=1 TO 4
 210 READ C$,R$
 220 IF I$=C$ THEN PRINT R$: GO
TO 100
 230 NEXT N
```



Ordinograma 2.4 — *Mais versatilidade*

Agora SE a sua entrada (input), I$, corresponder a qualquer das palavras de comando, o programa abandona o ciclo depois de imprimir a resposta apropriada, R$.

É evidente que nos encontramos agora de novo na posição original, onde na realidade nada acontece, de forma que precisamos de ser capazes de chamar as sub-rotinas de acção. Antes de mais, vamos resolver o salto para fora do ciclo, caso uma combinação seja encontrada, por uma nova rotina na linha 300.

```
 220 IF I$=C$ THEN PRINT R$: GO
TO 300
```


Temos ainda um indicador para referir qual a palavra que correspondeu à entrada (input), uma vez que N (o número de itens de DATA verificado) comporta este valor. Podemos utilizá-lo para movimentar para rotinas apropriadas, que são semelhantes àquelas que tínhamos escrito anteriormente, excepto o facto de não haver qualquer necessidade de definir a mensagem particular: isso já foi impresso como R$.

```
 300 IF N=1 THEN GO SUB 2000
 301 IF N=2 THEN GO SUB 2100
 302 IF N=3 THEN GO SUB 2200
 303 IF N=4 THEN GO SUB 2300
 309 GO TO 100
2000 LET Y=Y-1: RETURN
2100 LET Y=Y+1: RETURN
2200 LET X=X-1: RETURN
2300 LET X=X+1: RETURN
```

Expandindo o vocabulário

A lista de DATA pode ser facilmente expandida de forma a conter mais palavras. Por exemplo, podemos acrescentar direcções geográficas que alteram ambos os eixos X e Y:

```
3010 DATA "NORDESTE","IR PARA NO
RDESTE","SUESTE","IR PARA SUESTE
"
3020 DATA "SUDOESTE","IR PARA SU
DOESTE","NOROESTE","IR PARA NORO
ESTE"
```

e acrescentar mais algumas sub-rotinas (note que o tamanho do ciclo exploratório de DATA deve também ser aumentado):

```
200 RESTORE : FOR N=1 TO 8
304 IF N=5 THEN GO SUB 2400
305 IF N=6 THEN GO SUB 2500
306 IF N=7 THEN GO SUB 2600
307 IF N=8 THEN GO SUB 2700
2400 LET Y=Y-1: LET X=X+1: RETUR
N
2500 LET Y=Y+1: LET X=X+1: RETUR
N
2600 LET Y=Y+1: LET X=X-1: RETUR
N
2700 LET Y=Y-1: LET X=X-1: RETUR
N
```

Eliminação de redundâncias

Até ao momento, todas as respostas têm incluído a palavra «GOING» (ir para), tendo esta palavra sido na realidade inscrita em cada declaração de DATA. Agora a experiência de escrever no teclado é muito reconfortante, mas seria muito mais sensato definir esta palavra comum como uma variável referenciada. Repare-se que é incluído um espaço no fim para a separar da palavra seguinte.

```
40 LET G$="GOING"
```

Podem então eliminar-se todas as ocorrências desta palavra na DATA e combinar-se G$ com cada palavra-chave na resposta, em substituição.

```
 210 IF I$=C$ THEN  PRINT G$;R$;
 GO TO 300
9000 DATA "NORTE","NORTE","SUL",
"SUL","OESTE","OESTE","LESTE","L
ESTE"
9010 DATA "NORDESTE","NORDESTE",
"SUESTE","SUESTE"
9020 DATA "SUDOESTE","SUDOESTE",
"NOROESTE","NOROESTE"
```

Agora isto começa a parecer bastante confuso, mesmo algo tolo: uma vez que ambos os referenciadores contêm exactamente as mesmas palavras, por que não nos livrarmos, portanto, da palavra de resposta, R$, e imprimir simplesmente C$? Bem, neste caso podia-o fazer sem qualquer problema; mas, como é evidente, onde as respostas não são simplesmente uma repetição da entrada (o que é muitas vezes o caso), é essencial uma segunda palavra ou frase.

Se observar demoradamente estas sub-rotinas aperceber-se-á de que todas elas executam somente uma coisa — actualizar os valores de X e Y. Podíamos, portanto, incluir aquela informação na DATA original e ao mesmo tempo livrarmo-nos de todas elas! Precisamos de acrescentar os valores apropriados das coordenadas X e Y nas linhas

de DATA depois de cada resposta, e ler internamente (READ) esta informação em blocos de quatro (INPUT, RESPONSE, X-MOVE, Y-MOVE — Entrada, resposta, Movimentar X, Movimentar Y — ver quadro 2.2).

PALAVRA DE COMANDO CS	RESPOSTA R$	X-MOVE (movimentar) XU	Y-MOVE (movimentar) YU
NORTE	NORTE	0	−1
SUL	SUL	0	1
OESTE	OESTE	−1	0
LESTE	LESTE	1	0
NORDESTE	NORDESTE	1	−1
SUESTE	SUESTE	1	1
SUDOESTE	SUDOESTE	−1	1
NOROESTE	NOROESTE	−1	−1

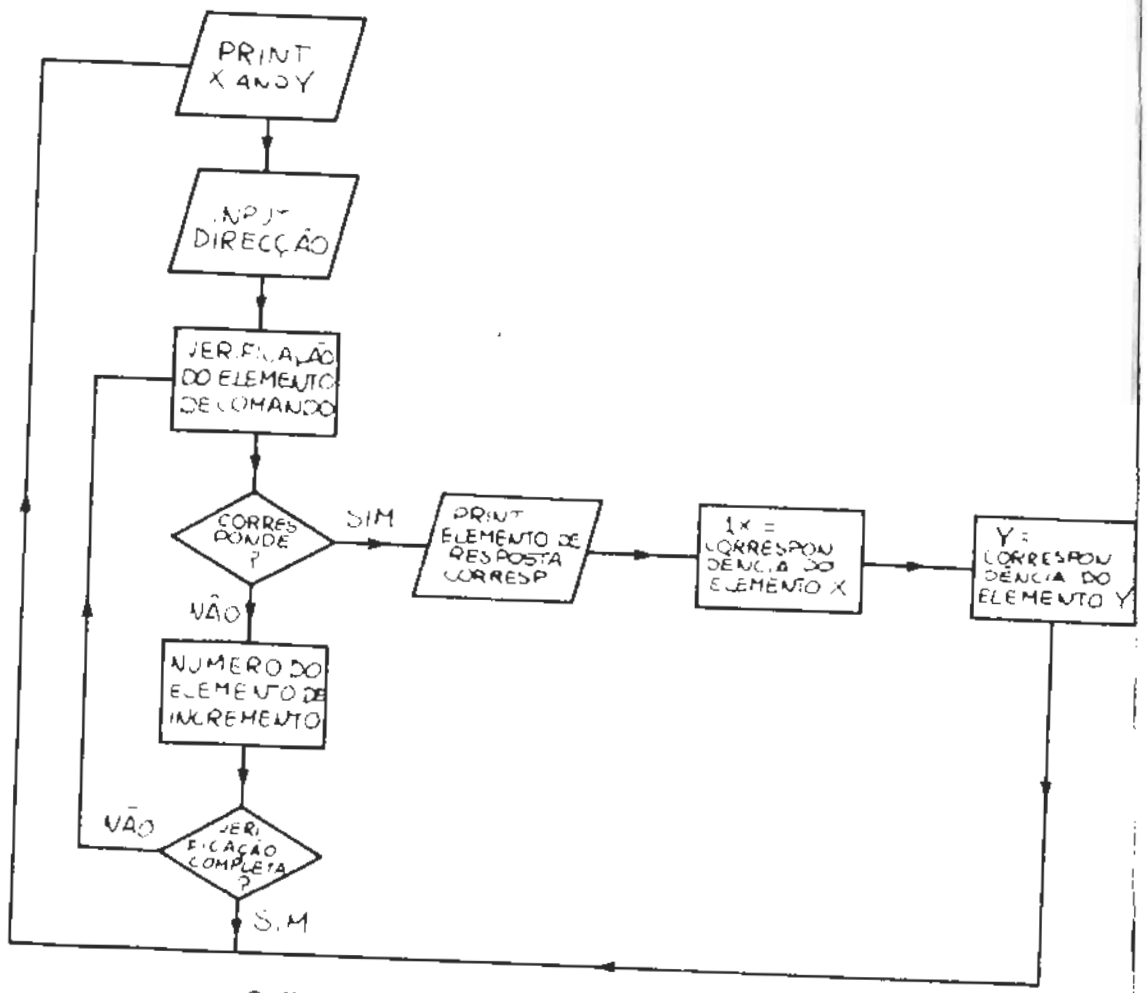
Quadro 2.2 — *Movimentações de X e Y incorporadas na data*

```
9000 DATA "NORTE","NORTE",0,-1,"
SUL","SUL",0,1,"OESTE","OESTE",-
1,0,"LESTE","LESTE",1,0
9010 DATA "NORDESTE","NORDESTE",
1,-1,"SUESTE","SUESTE",1,1
9020 DATA "SUDOESTE","SUDOESTE",
-1,1,"NOROESTE","NOROESTE",-1,-1
```

Podemos agora eliminar as linhas 300 a 2700 e modificar as linhas 210 e 220 de modo que X e Y sejam actualizados aí (ver ordinograma 2.5).

```
210 READ C$,R$,XU,YU
220 IF I$=C$ THEN  PRINT G$;R$;
LET X=X+XU: LET Y=Y+YU: GO TO 1
0
```

Este modelo de colocação em conjunto de toda a informação num modelo de ligação é uma característica muito comum, usada em vários dos programas apresentados na parte final do livro.



Ordinograma 2.5 — *Utilização de «data» de ligação*

Comandos abreviados

Até ao momento temos utilizado sempre palavras completas como comandos; todavia isso significa que se tem de escrever bastante no teclado para transmitir as instruções à máquina. Se se sentir cansado, ou mesmo com pouca predisposição para o fazer, pode pensar em modificar as palavras de comando pela primeira letra da palavra; entrará de seguida com uma única letra, e isso será o bastante. Contudo, a menos que comece a utilizar letras aleatórias, isto só funcionará desde que duas palavras não principiem pela mesma letra! Para codificar todas as oito direcções geográficas usadas acima temos de usar até um conjunto de duas letras: N, NE, E, SE, S, SO, O, NO.


```
9000 DATA "N","NORTE",0,-1,"S","
SUL",0,1,"O","OESTE",-1,0,"L","L
ESTE",1,0
9010 DATA "NE","NORDESTE",1,-1,"
SE","SUESTE",1,1
9020 DATA "SO","SUDOESTE",-1,1,"
NO","NOROESTE",-1,-1
```

Repare-se que somente mudaram as palavras de comando. O computador fornece uma descrição completa da direcção, uma vez que ainda nos encontramos a usar a segunda parte da lista de DATA que comporta a resposta.

Correspondência ou combinação parcial

Em todos os programas mencionados acima verificámos sempre se a entrada correspondia por combinação, com toda a exactidão, a uma palavra no referenciador de comando. Contudo, seria útil se pudéssemos permitir um número de palavras similares que fossem aceitáveis como significando a mesma coisa. Por exemplo, poder-se-ia verificar se a primeira letra da palavra de entrada correspondia ou não à palavra-chave abreviada, comparando-se unicamente o primeiro caracter [tomando I$(1)].

```
190 LET I$=I$(1)
```

Isso funcionará com NORTE, SUL, LESTE E OESTE, existindo porém problemas óbvios ao lidar com posições intermédias. Para acrescentar a este facto há imensas palavras que começam com as letras N, S, L e O — sendo todas elas igualmente aceitáveis para a máquina como uma direcção válida.

Por exemplo:

NÃO NORTE

produziria:

IR PARA NORTE

Um processo mais selectivo é combinar por correspondência um número de letras em vez de somente uma única. Neste exemplo as

três primeiras letras das quatro principais direcções são bastante características.

NOR
SUL
LES
OES

Se usar as seguintes como palavras de comando, por exemplo:

NOR
NORTE
NORTENHO
e NORTEADO

serão, então, todas igualmente aceitáveis, porém:

NÃO
NOMEAR
NOITE
e NOMINAL

serão todas rejeitadas.

Tudo aquilo que precisamos de fazer é considerar as três primeiras letras da entrada (input), I$(TO 3) [I$(até 3)], ainda que um factor de verificação, em que o tamanho de I$ não seja menor do que três caracteres, deva também ser incluído.

```
 190 IF LEN (I$)<3 THEN  GO TO 1
00
 195 LET I$=I$( TO 3)
3000 DATA "NOR,NORTE",0,-1,"SUL,
LES","LESTE",1,0
```

Comandos sequenciais

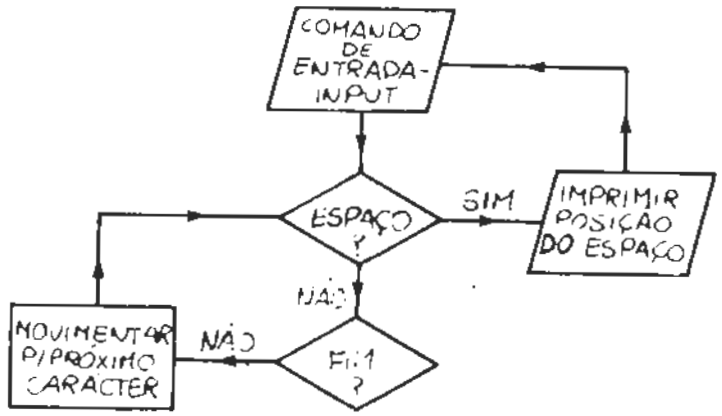
Nas rotinas acima lidámos com as orientações geográficas intermédias (pontos colaterais) como entidades separadas, mas não precisaríamos de o fazer se pudéssemos dar uma sequência de comandos

simultaneamente. Existe sempre mais de uma maneira de atingir qualquer ponto, e se mais de uma palavra de comando pudesse ser compreendida ao mesmo tempo não teríamos de nos preocupar em verificar direcções tais como «NORDESTE», uma vez que podiam ser tratadas pela combinação de «NORTE» e «LESTE» (o exemplo aqui apresentado em português é mais evidente em inglês — «NORTH EAST» é composto exactamente pelas palavras separadas «NORTH» e «EAST»).

Isto remete-nos para a questão bastante significativa de como repartir uma entrada por palavras. Primeiro deve-se interrogar a si próprio: como reconhece que uma série de caracteres perfaz uma palavra separada? A resposta, claro está, é de que há um espaço (SPACE) entre elas. Agora, se atendermos aos espaços, podemos fraccionar a entrada em palavras separadas, que poderemos tratar individualmente. A maneira mais fácil de observar os espaços em BASIC é com o comando INSTR, que pesquisa na íntegra uma cadeia de pesquisa designada para uma combinação por correspondência de valores com uma segunda cadeia de objectivos. Infelizmente, este comando não é providenciado pelo BASIC no Sinclair, de modo que teremos de usar uma série de comandos BASIC para o substituir. Estes comandos serão colocados numa sub-rotina na linha 5000, à qual nos referiremos daqui em diante como sendo simplesmente a rotina INSTR.

```
5000 FOR N=1 TO LEN (I$)
5010 IF I$(N)=" " THEN  LET SP=N
: RETURN
5020 NEXT N
5030 LET SP=0
5040 RETURN
```

Esta rotina procederá à verificação do primeiro carácter em I$ como sendo ou não um espaço. Se não for um espaço, então a rotina continuará automaticamente a verificação até que o fim de I$ seja atingido. Se nenhum espaço for encontrado em toda a extensão de I$, SP será posicionado a zero. Se for encontrado um espaço, então o valor de SP será o número de caracteres ao longo de I$ ao fim dos quais o espaço se situa (ver ordinograma 2.6).



Ordinograma 2.6 — *Situando a posição de um espaço*

Precisamos de chamar isto a partir da rotina principal. Imprimiremos o resultado quando fizermos um RETURN, de maneira que possamos observar o que está a acontecer.

```
130 GO SUB 5000
140 PRINT SP: GO TO 100
```

Experimente isto com:

NOR OES

SP 4

NORTE OESTE

SP 6

NOR NOR OESTE

SP 4

Note-se que o comprimento da palavra é justificado por SP mas que somente o primeiro espaço é encontrado. Para se encontrarem todos os espaços vamos ter de trabalhar arduamente. Em primeiro lugar elimine-se aquela linha temporária 140.

Vamos observar logicamente a entrada a partir do seu início (ST, do inglês *start*) (extremidade esquerda). Substituiremos I$(TO 3) por

I$(ST TO ST+2), de maneira que possamos atender a qualquer combinação de três letras em toda a extensão de I$. Para tornar tudo isto mais acessível chamaremos ao resultado disto W$, visto mostrar a posição duma palavra (word). Para inicializar devemos colocar a posição inicial de pesquisa ST igual a um e acrescentar um espaço à frente de I$, de forma que a primeira palavra seja também encontrada (ver ordinograma 2.7).

```
 125 LET ST=1: LET I$=" "+I$
 130 GO SUB 5000
 190 LET W$=I$(ST TO ST+2)
 210 IF W$=C$ THEN  PRINT G$;R$;
 LET X=X+XU: LET Y=Y+YU: GO TO 1
00
5000 FOR N=ST TO LEN (I$)
```

Se correr isto tal como se apresenta, então ainda encontrará somente a primeira palavra, uma vez que temos GO TO 100 no fim da linha 210. Contudo, reenviando-se simplesmente o programa ao verificador INSTR na linha 130, em alternativa, também não é de grande ajuda, uma vez que começará sempre a verificar a partir do início de I$, encontrando-se sempre o mesmo primeiro espaço. Uma vez encontrado este primeiro espaço necessitamos de movimentar a posição de início, ST, para a próxima pesquisa no carácter imediatamente a seguir a esse espaço SP+1. Quando mais nenhum espaço puder ser encontrado, então o fim da entrada foi atingido, e podemos (ir para) GO TO 100 de novo.

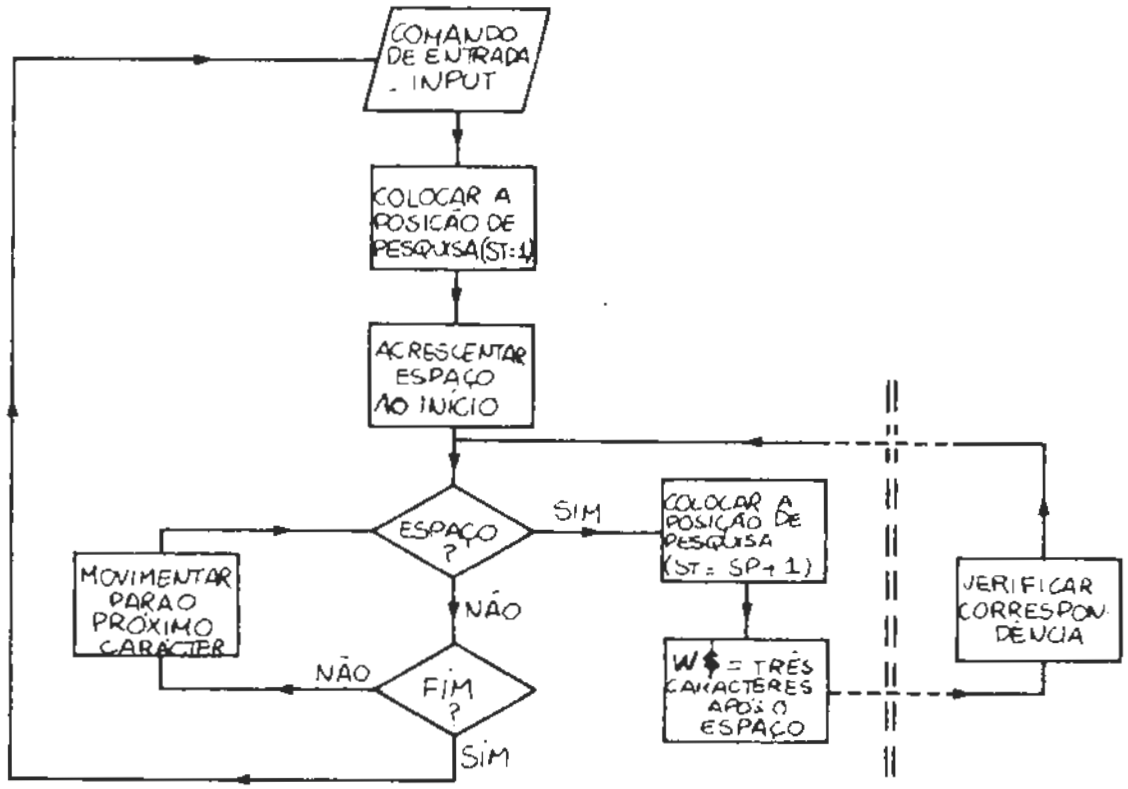
```
 140 IF SP>0 THEN  LET ST=SP+1:
GO TO 190
 150 GO TO 100
 210 IF W$=C$ THEN  PRINT G$;R$;
 LET X=X+XU: LET Y=Y+YU: GO TO 1
30
```

Escrevendo-se agora:

NORTE OESTE

produz:

IR PARA NORTE
IR PARA OESTE



Ordinograma 2.7 — *Pesquisando uma palavra-chave*

e mesmo:

NOR NOR LESTE

é decifrado como:

IR PARA NORTE
IR PARA NORTE
IR PARA LESTE

Seria muito mais apropriado se eliminássemos todos aqueles «IR PARA» redundantes e colocássemos na mesma linha todas as direcções referidas. Precisamos de (imprimir) PRINT G$ uma vez, imediatamente antes do verificador INSTR. Agora, de cada vez que percorremos o ciclo comparando a palavra actual com as armazenadas (imprimimos) PRINT R$; se houver uma correspondência. Como há um ponto-e-vírgula a seguir a isto, as palavras serão impressas na mesma linha, porém também precisamos de acrescentar espaços entre

elas. Finalmente adicionamos um simples PRINT exactamente antes de regressarmos a uma nova entrada, para movimentar a posição do cursor para a próxima linha.

```
126 PRINT G$;
145 PRINT
210 IF W$=C$ THEN  PRINT R$;" "
;: LET X=X+XU: LET Y=Y+YU: GO TO
130
```

Agora:

NORTE ORIENTALMENTE SUL OESTE

fá-lo apropriadamente andar em círculos:

IR PARA NORTE LESTE SUL OESTE

CAPÍTULO 3

Compreensão da linguagem natural

Até ao momento temos simplesmente comunicado com o computador de maneira muito restrita, uma vez que foi unicamente programado para perceber umas poucas de palavras ou letras, só as reconhecendo se forem introduzidas (input) exactamente da maneira correcta. Por exemplo, se colocar um espaço antes ou depois do seu comando, quando o entrar (INPUT) será rejeitado. Isto acontece porque estamos a comparar se as duas cadeias de dados correspondem exactamente ou não.

Por outro lado, na vida real, qualquer pessoa usa aquilo que é comummente conhecido por linguagem «natural», que é qualquer coisa de muito sofisticado e extremamente variável e a que somente o cérebro humano pode efectivamente coadunar-se. Mesmo que nos esqueçamos momentaneamente da diferença existente entre o «português» e o «brasileiro», ou mesmo entre os diversos dialectos regionais de qualquer um deles (pode «'stá lhagado peingâando'» querer realmente significar «Está a chover» — «está alagado, pingando»), o facto de se lidar com uma língua acarreta um número infinito de problemas.

Mesmo os sistemas mais sofisticados existentes no mundo não podem dar conta de tudo. Existe uma velha história que ilustra bem este aspecto. A CIA desenvolveu um soberbo programa de traduções, o qual podia instantaneamente converter qualquer coisa em inglês para russo e vice-versa. Na esperança de impressionarem o presidente fizeram uma demonstração das suas capacidades, durante a qual converteu tudo o que disse para russo, falou-o, retraduzindo então o russo de novo para inglês, isto é, fazendo posteriormente uma

retroversão. O presidente ficou bastante impressionado e totalmente absorto até que um dos seus secretários lhe lembrou que se tinha esquecido de que a sua mulher estava à sua espera lá fora. Quando comentou arrependidamente: «Fora de vista, longe do pensamento», ficou fascinado por ter ouvido a máquina retorquir-lhe com «Maníaco invisível»!

Lidando com frases

Toda a gente sabe que a linguagem real é composta por frases; todavia, que é que queremos exactamente significar com *frase?* Bem, a maneira mais óbvia pela qual reconhecemos uma frase é que vemos um ponto final no seu termo! Contudo, se estivermos prontos a ser capazes de lidar com frases, vamos ter de pensar muito mais do que nesse mero aspecto.

A definição do *Oxford Dictionary* inclui «uma série de palavras em discurso conectado ou escrita, formando gramaticalmente a expressão completa dum pensamento único, e normalmente contendo um sujeito e predicado, e exprimindo uma declaração, pergunta, uma instrução de comando ou pedido», mas também se admite que seja utilizada livremente para significar «parte de escrita ou discurso entre dois pontos finais». Caramba! Alguém é capaz de traduzir tudo isso para português corrente, por favor? As questões intrincadas e ilógicas que se prendem com a língua portuguesa são de algum modo inexoráveis; portanto, como é que podemos esperar que um computador consiga corresponder a todas as suas exigências?

Bem, vamos começar por observar alguns exemplos mais simples de frases.

EU QUERO

consiste de um sujeito e de um verbo (o verbo QUERER).

EU QUERO BISCOITOS

tem igualmente um complemento directo (BISCOITOS).

EU QUERO BISCOITOS DE CHOCOLATE

qualifica o complemento directo com um adjectivo (DE CHOCOLATE),

ALGUMAS VEZES QUERO BISCOITOS DE CHOCOLATE

qualifica o verbo com um advérbio de tempo (ALGUMAS VEZES).

A palavra mais importante em todos os exemplos descritos acima é «QUERO», já que exprime a ideia principal. O segundo exemplo é mais informativo, uma vez que indica que somente um tipo particular de complemento directo, BISCOITOS, é desejado. A adição de um adjectivo, CHOCOLATE, dá mais informação sobre o tipo de complemento directo pretendido; todavia, tudo se tornou ainda mais incerto quando o advérbio de tempo «ALGUMAS VEZES» foi incluído.

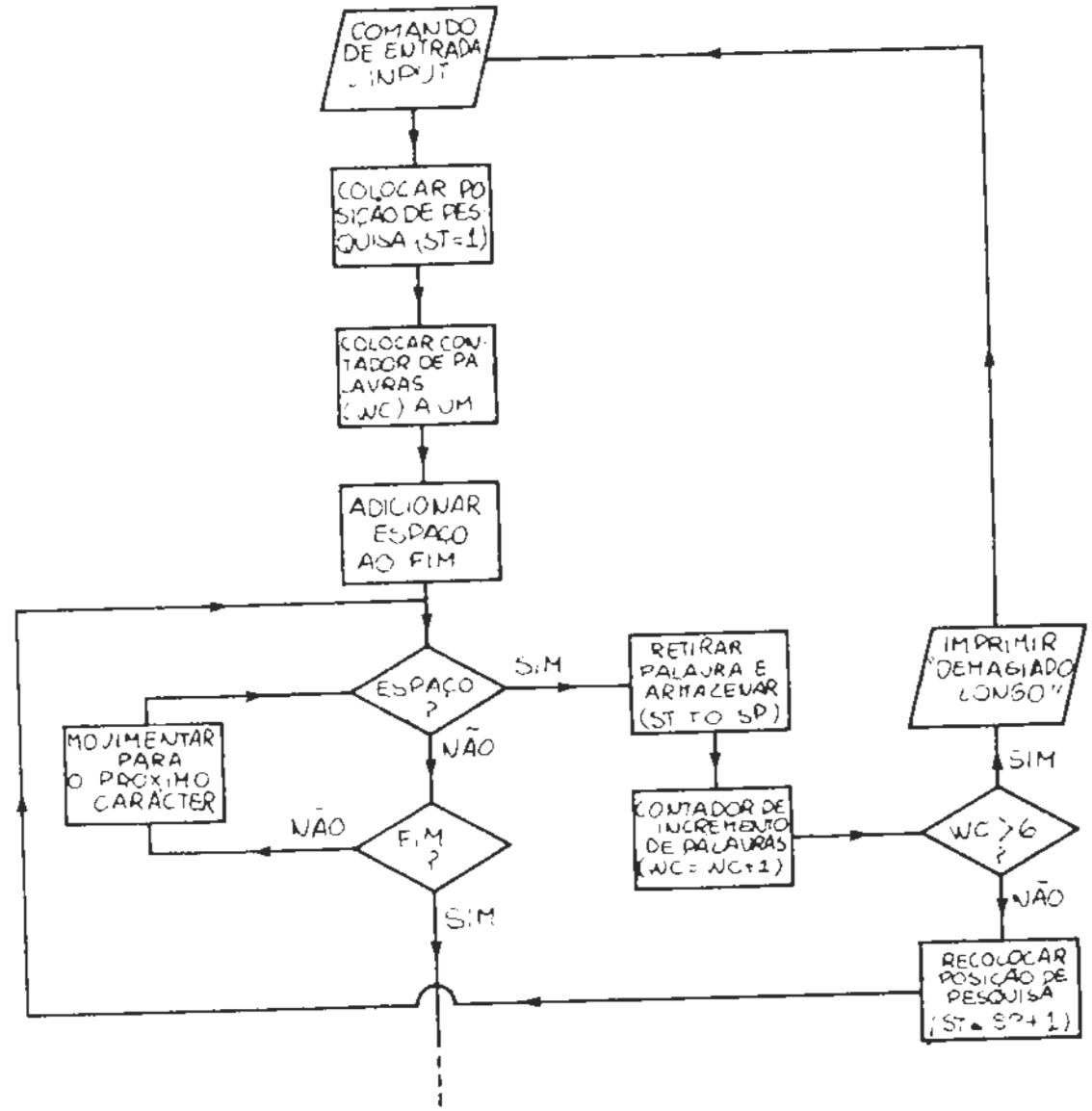
Agora como poderia um programa de computador decifrar tais frases? A resposta deve residir na descoberta de alguma estrutura lógica dentro da frase. Portanto, que «regras» poderíamos aplicar a este exemplo?

1) Todas começaram por um sujeito (EU), excepto a última, na qual o sujeito foi omitido mas se supõe subentendido, e acabaram com um ponto final.
2) A última palavra foi sempre o complemento directo «BISCOITOS», salvo nos dois últimos casos, que foi acrescido pela adjectivação adicional que lhe é inerentemente particular (a menos que houvesse falta de complemento directo, e somente duas palavras).
3) Se a palavra antes do complemento directo não fosse o verbo QUERER poderia ter sido um advérbio de quantidade, como por exemplo «MUITOS».
4) Se a palavra antes do verbo (predicado) não fosse o sujeito «EU» era o advérbio de tempo «ALGUMAS VEZES».

Vamos escrever um programa para o qual indicamos ao computador frases, pedindo-lhe que as divida sintacticamente nas suas diversas partes componentes.

Para se poder começar temos necessidade de atribuir um vocabulário de complementos directos, adjectivos e advérbios para se poder trabalhar com eles, inserindo-os numa DATA (conjunto de dados ou informações), agrupados de acordo com o seu tipo. Assim os primeiros seis itens são complementos directos, os próximos seis itens são adjectivos, e os três últimos itens são advérbios.


```
9300 REM COMPLEMENTOS DIRECTOS
9010 DATA "BISCOITOS","BOLOS","S
ORTIDOS"
9020 DATA "CAFE","CHA","AGUA"
9030 REM ADJECTIVOS
9040 DATA "CHOCOLATE","GENGIBRE"
 "MARMELADA"
9050 DATA "FRIO","QUENTE","MORNO
"
9060 REM ADVERBIOS
9070 DATA "SEMPRE","MUITAS VEZES
","ALGUMAS VEZES"
```



Ordinograma 3.1 — *Tiragem de palavras*

Precisamos agora de dividir a frase em palavras (ver ordinograma 3.1). Uma vez mais executá-lo-emos com uma rotina de pesquisa INSTR para os espaços, e para facilitar a vida acrescentaremos um espaço no fim de I$, de forma que o formato da última palavra pareça tal e qual o das outras palavras.

```
 100 INPUT I$: PRINT I$
 110 LET ST=1:
 120 LET I$=I$+" "
 130 GO SUB 5000
 190 GO TO 130
5000 FOR N=ST TO LEN (I$)
5010 IF I$(N)=" " THEN  LET SP=N
: RETURN
5020 NEXT N
5030 LET SP=0
5040 RETURN
```

O fim da frase foi atingido quando não tiverem sido encontrados mais espaços.

```
140 IF SP=0 THEN  GO TO 200
```

Se for encontrado um espaço, então a secção de I$, de ST (iniciador de pesquisa actual) a SP (espaço actual = comprimento da palavra), é retirada e armazenada numa área (tabela) de armazenamento de palavras, W$(WC)

```
  10 DIM W$(5,9)
150 LET W$(WC)=I$(ST TO SP)
```

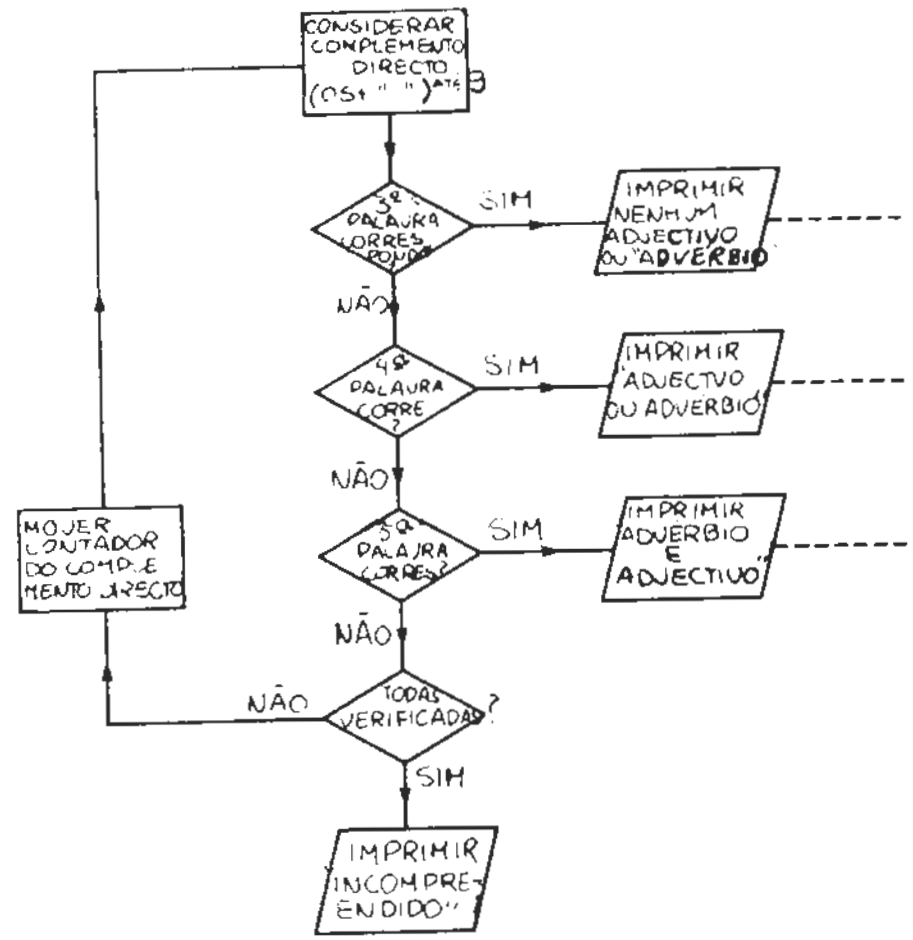
para principiar com ST = 1, de maneira que a pesquisa comece no primeiro carácter da cadeia de caracteres de entrada. A variável contadora de palavras WC é posicionada a um, de modo que a primeira palavra encontrada seja armazenada no primeiro elemento da área de armazenamento de palavras (tabela de dados — palavras).

```
110 LET ST=1: LET WC=1
```

O contador de palavras é incrementado (de maneira que o próximo elemento da tabela W$ seja usado da próxima vez) e é realizada uma verificação no sentido de se averiguar que a frase não contenha mais do que cinco palavras. A posição de inicialização para a próxima pesquisa é então colocada a mais uma unidade do que a posição do último espaço, prosseguindo-se assim a função de pesquisa.


```
 160 LET WC=WC+1
 170 IF WC>6 THEN  PRINT "FRASE
DEMASIADO LONGA": GO TO 100
 180 LET ST=SP+1
```

É agora executado um teste para se indagar se existe uma correspondência entre as palavras-chave na frase e os complementos directos contidos no vocabulário respectivo de DATA O$ (ver ordinograma 3.2).



Ordinograma 3.2 — *Procurando uma combinação por correspondência*

Note-se que, uma vez que o BASIC do *Sinclair* utiliza cadeias de dados de comprimento inflexivelmente fixo para áreas de tabelas, as quais são «empilhadas» com espaços ou brancos, devemos adicionar um número adaptável de espaços para perfazer o preenchimento da restante parte das cadeias sem utilização aproveitada, com inserção

no fim da nossa palavra de DATA para a categoria dos complementos directos, antes de as podermos comparar. Para o fazer, acrescentamos nove espaços à extremidade final direita de O$, retirando então os primeiros nove caracteres da cadeia resultante.

```
205 READ O$: LET O$=(O$+"
   ")( T O 9)
```

Somente as palavras 3, 4 e 5 são verificadas, visto estas serem as únicas posições possíveis para o complemento directo do nosso formato de frase restringido. Três rotinas diferentes são saltadas de acordo com a posição da palavra combinada em correspondência de valores, inserida na frase. Se nenhuma correspondência for encontrada, é impressa uma mensagem e solicitada uma nova entrada. A função de comando RESTORE (Redepositar) no princípio da linha 200 reenvia sempre o indicador de volta ao início da DATA (isto é, o posicionamento dos complementos directos).

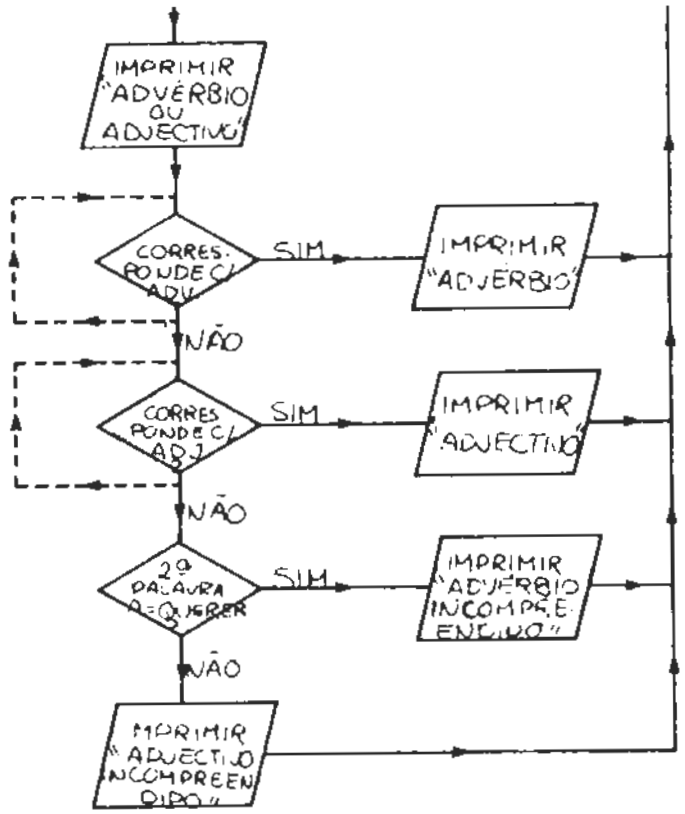
```
200 RESTORE : FOR N=1 TO 6
210 IF W$(3)=O$ THEN  GO TO 500
220 IF W$(4)=O$ THEN  GO TO 600
230 IF W$(5)=O$ THEN  GO TO 700
240 NEXT N
250 PRINT "COMPLEMENTO DIRECTO
NÃO ENCONTRADO"
260 GO TO 100
```

Se o complemento directo tiver sido encontrado para a terceira palavra, então não haveria, para primeira hipótese complementar em relação às outras categorias secundárias de sintaxe, nem adjectivo nem advérbio.

```
500 PRINT "NEM ADJECTIVO NEM AD
VERBIO"
510 GO TO 100
```

Se o complemento directo tivesse sido encontrado para a quarta palavra, poderia então haver ainda na frase ou um adjectivo ou um advérbio (ver ordinograma 3.3).

```
600 PRINT "OU ADJECTIVO OU ADVE
RBIO"
```

Ordinograma 3.3 — *Advérbio ou adjectivo*

Em primeiro lugar verificamos se há correspondência entre a segunda palavra e a DATA refcrentc aos advérbios, a qual se lê sequencialmente como VS. Contudo, devemos ter presente o facto de que o contador de DATA deve agora ser rcposicionado para o primeiro advérbio por intermédio da função RESTORE 9060, e que VS terá de ser «empilhado» com espaços, isto é, preenchido.

```
610 RESTORE 9060: FOR N=1 TO 3
615 READ V$: LET V$=(V$+" ")( T
O 9)
620 IF W$(2)=V$ THEN  GO TO 900
630 NEXT N
```

Se nenhuma correspondência tiver sido encontrada, verificamos de novo a terceira palavra, confrontando-a com a lista de adjectivos, a qual lemos (Read) como JS.

```
 640 RESTORE 9830: FOR N=1 TO 5
 645 READ J$: LET J$=(J$+" ")( T
O 9)
 650 IF W$(3)=J$ THEN  GO TO 100
0
 660 NEXT N
```

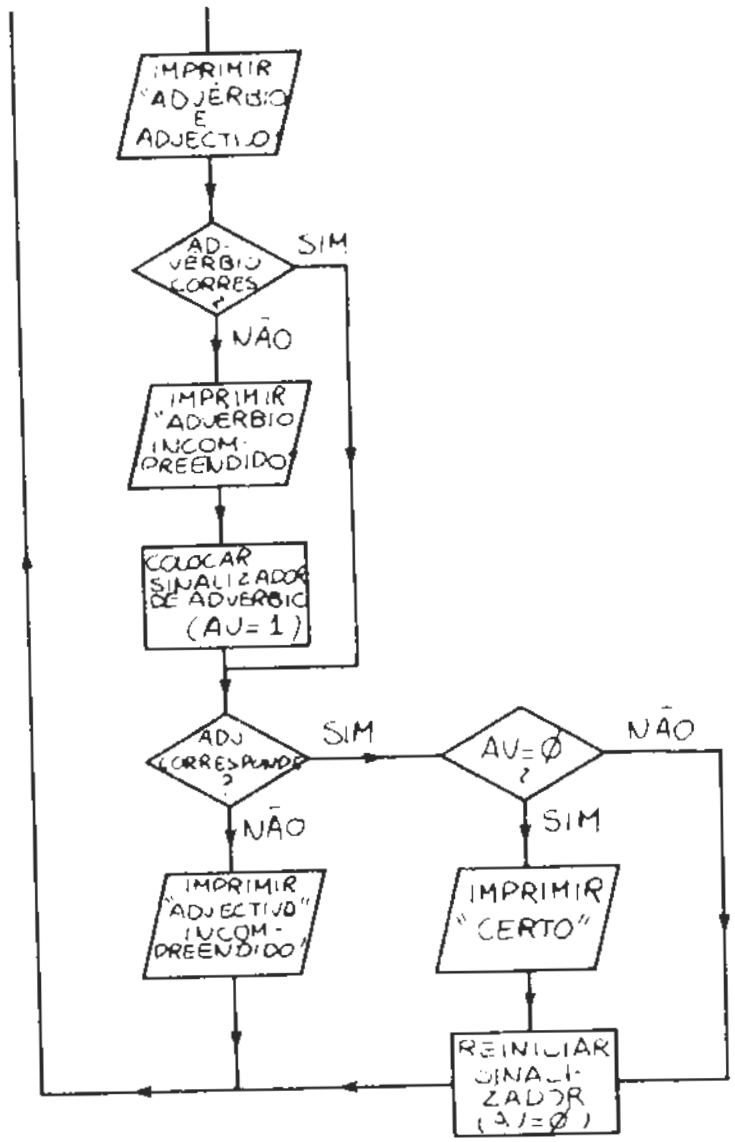
Se nenhuma correspondência tiver sido encontrada em qualquer destas listas, seria útil indicar qual das palavras não foi compreendida. A resposta mais fácil reside em verificar se a segunda palavra devia ter sido um advérbio. Por outro lado, se a segunda palavra tivesse sido o verbo, então a terceira palavra deveria ter sido um adjectivo. Repare-se que a palavra actual, para a qual não se deu a correspondência, é agora incluída na mensagem.

```
 670 IF W$(2)<>"QUERER    " THEN
 PRINT "ADVERBIO ";W$(2);"INCOMP
REENDIDO": GO TO 100
 680 PRINT "ADJECTIVO ";W$(3);"I
NCOMPREENDIDO": GO TO 100
```

Se tiver sido encontrada uma correspondência em qualquer dos testes, então é impressa uma mensagem de satisfação da condição. Note-se que qualquer destas possibilidades é exclusiva e que em quatro palavras somente podemos ter uma ou outra.

```
 900 PRINT "ADVERBIO"
 910 GO TO 100
1000 PRINT "ADJECTIVO"
1010 GO TO 100
```

Onde quer que um advérbio e um adjectivo estejam ambos presentes devemos verificar ambos igualmente, e por conseguinte uma correspondência no primeiro teste também salta para o segundo teste (ver ordinograma 3.4).



Ordinograma 3.4 — *Advérbio e adjectivo*

```
 700 PRINT "ADVERBIO E ADJECTIVO
"
 710 RESTORE 9860: FOR N=1 TO 3
 715 READ V$: LET V$=(V$+" ")( T
O 9)
 720 IF W$(2)=V$ THEN  GO TO 750
 730 NEXT N
```

Se não for encontrada uma correspondência bem sucedida para o adjectivo, então o programa salta atrás depois do relatório.

```
  20 LET AV=0
 740 PRINT "ADVERBIO ";W$(2);"IN
COMPREENDIDO": LET AV=1
 750 RESTORE 9030: FOR N=1 TO 8
 755 READ J$: LET J$=(J$+" ")(
0 9)
 760 IF W$(4)=J$ THEN  GO TO 800
 770 NEXT N
```

Se o adjectivo tiver sido encontrado, então é feito um teste para se averiguar se o sinalizador AV de advérbios não foi posicionado antes de ter sido relatada uma correspondência. Em qualquer dos casos, o sinalizador é reinicializado antes de se efectuar a próxima entrada.

```
 780 PRINT "ADJECTIVO ";W$(4);"I
NCOMPREENDIDO"
 790 GO TO 100
```

Se nenhuma correspondência for encontrada para o advérbio, então este facto é relatado: um sinalizador AV é posto a 1 para indicar insucesso nesta altura, antes de o adjectivo ser verificado.

```
 800 IF AV=0 THEN  PRINT "ADJECT
IVO E ADVERBIO CERTOS"
 810 LET AV=0
 820 GO TO 100
```

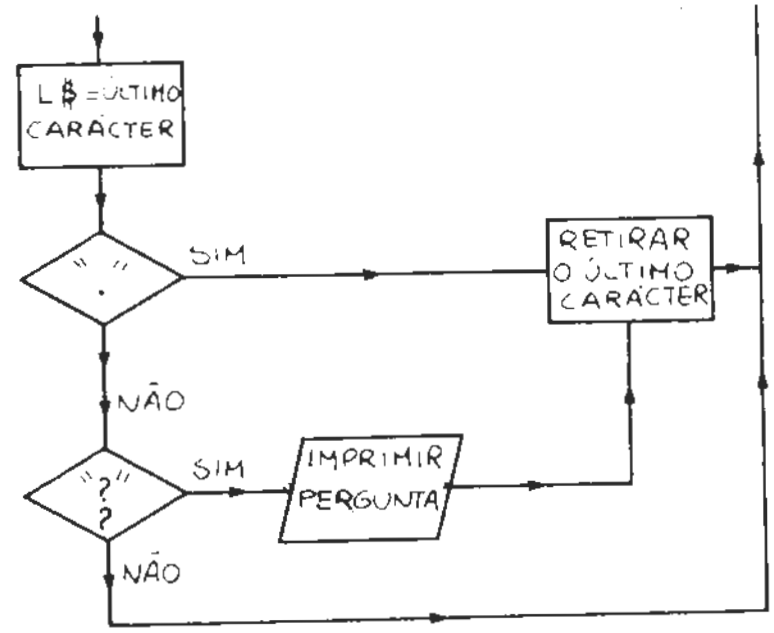
Que se passa com a pontuação?

Como já foi dito, reconhece-se normalmente o fim duma frase por causa de ter um ponto final, ainda que quando se inscreve qualquer informação no computador através dum teclado geralmente se esqueça tudo acerca de tais trivialidades. Mas que acontecerá ao programa se algum usuário «espertalhaço» lhe colocar a pontuação correcta? Se pensar por uns momentos, aperceber-se-á de que o computador principiará a «queixar-se», visto já não reconhecer a última palavra, uma vez que esta será na realidade lida como a palavra em si mais o ponto final.

Daí precisarmos de verificar se o último carácter na entrada da cadeia I$ é um ponto final: o melhor sítio para se proceder a esta

verificação deve ser logo imediatamente depois de INPUT. Se o último carácter for «.», dispensar-se-á então simplesmente este carácter, continuando-se como anteriormente.

Juntar-se-á isto como uma sub-rotina, a qual será transposta por salto mal tenha lugar uma entrada. Outros símbolos de pontuação podem igualmente aparecer no fim da frase, de modo que leremos o último carácter como uma variável L$, a qual será também usada mais tarde (ver ordinograma 3.5).



Ordinograma 3.5 — *Lidando com a pontuação*

```
 105 GO SUB 2000
2000 LET L$=I$(LEN (I$))
2010 IF L$="." THEN  GO TO 2100
2090 RETURN
2100 LET I$=I$( TO LEN (I$)-1):
RETURN
```

Um finalizador de frases muito utilizado é o ponto de interrogação, o qual indica o contexto das palavras. Podemos distingui-lo da mesma maneira e, de momento, limitar-nos-emos a relatar a sua presença.

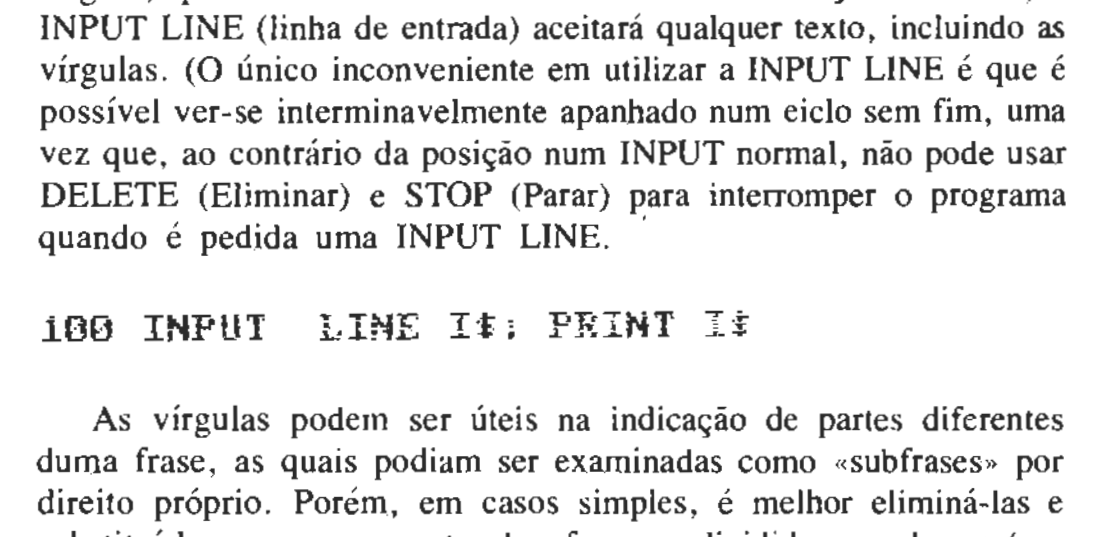
```
2030 IF L$="?" THEN  PRINT "PERG
UNTA": GO TO 2100
```

O comando INPUT normal não aceitará mais nada depois duma vírgula, que é lida como um finalizador de informação. Todavia, a INPUT LINE (linha de entrada) aceitará qualquer texto, incluindo as vírgulas. (O único inconveniente em utilizar a INPUT LINE é que é possível ver-se interminavelmente apanhado num eiclo sem fim, uma vez que, ao contrário da posição num INPUT normal, não pode usar DELETE (Eliminar) e STOP (Parar) para interromper o programa quando é pedida uma INPUT LINE.

```
100 INPUT  LINE I$: PRINT I$
```

As vírgulas podem ser úteis na indicação de partes diferentes duma frase, as quais podiam ser examinadas como «subfrases» por direito próprio. Porém, em casos simples, é melhor eliminá-las e substituí-las por espaços antes de a frase ser dividida em palavras (ver ordinograma 3.6). Repare-se que isso só funcionará bem se não houver qualquer espaço depois da vírgula, uma vez que qualquer espaço que se siga a uma vírgula substituída será detectado e tratado como uma nova palavra.

Modificaremos a nossa sub-rotina existente, INSTR, de maneira a podermos verificar I$ para qualquer cadeia de dados pré-definida, T$. Para clarificar as coisas durante o extenso processamento, faremos que a variável aponte para a posição da correspondência na cadeia IS,



Ordinograma 3.6 — *Substituição de vírgulas e apóstrofes*

a qual pode então ser trocada por qualquer número de variáveis diferentes, tais como SP. Primeiro modificamos o nosso verificador de espaços para o novo formato.

```
 130 LET T$=" ": GO SUB 5000: LE
T SP=IS
5010 IF I$(N)=T$ THEN  LET IS=N:
 RETURN
5030 LET IS=0
```

Agora o mesmo método pode ser usado para se procurar uma vírgula, antes de ser substituída por um espaço.

```
 115 LET T$=",": GO SUB 3000
3000 GO SUB 5000: LET CM=IS
3010 IF CM=0 THEN  LET ST=1: RET
URN
3020 LET I$=I$( TO CM-1)+" "+I$(
CM+1 TO )
3030 LET ST=CM+1
3040 GO TO 3000
```

Se juntarmos esta linha, pode observar-se a eliminação da pontuação da cadeia, item por item.

```
025 PRINT I$
```

Os apóstrofes podem ser tratados da mesma maneira, excepto que não os substituímos por um espaço mas simplesmente terminamos as palavras.

```
 115 LET T$=",": GO SUB 3000: LE
T T$="'": GO SUB 3100
3100 GO SUB 5000: LET ST=1: RETU
RN
3120 LET I$=I$( TO AP-1)+I$(AP+1
 TO )
3125 PRINT I$
3130 LET ST=AP+1
3140 GO TO 3100
```

Pesquisa por exclusão de hipóteses

Ainda que o método de exame duma frase descrito acima funcione, tem a desvantagem de exigir que a frase seja introduzida (Input) com um formato particular, restrito. Por exemplo, se se entrar:

BOLO DE CHOCOLATE É O QUE EU QUERO

então o computador responderá com:

COMPLEMENTO DIRECTO NÃO ENCONTRADO.

uma vez que somente procura por complementos directos na terceira, quarta ou quinta palavras.

Utilizar uma pesquisa por exclusão de hipóteses para toda a frase em cada palavra-chave, sem primeiro dividir a frase por palavras, tem a vantagem de permitir um formato de entrada totalmente livre. Neste tipo de aproximação considera-se a primeira palavra-chave e experimenta-se combiná-la por correspondência de valores, confrontando-a com o mesmo número de letras em I$, começando-se pelo primeiro carácter. Se este teste falhar, então é automaticamente repetido, começando-se pelo segundo carácter, etc., até que seja encontrada uma correspondência ou atingido o fim de I$. Por exemplo, se IS foi «EU QUERO BOLO» e a primeira palavra-chave fosse «BOLO», as comparações possíveis seriam:

passo 1	E	U		Q	
passo 2	U		Q	U	
passo 3		Q	U	E	
passo 4	Q	U	E	R	
passo 5	U	E	R	O	
passo 6	E	R	O		
passo 7	R	O		B	
passo 8	O		B	O	
passo 9		B	O	L	
passo 10	B	O	L	O	(correspondência encontrada)

Até ao momento a nossa rotina INSTR somente tem experimentado corresponder-se com um único carácter; teremos de modificar a linha 5010 outra vez, de forma a ter em conta o comprimento da cadeia em consideração, WL.

```
5000 FOR N=1 TO LEN (I$)-WL+1
5010 IF I$(N TO (N+WL-1))=T$ THE
N  LET IS=N: RETURN
5020 NEXT N
5030 LET IS=0
5040 RETURN
```

Eliminar tudo, excepto as linhas de DATA (da linha 9000 em diante), e acrescentar estas linhas para verificar a sua correspondência com um complemento directo de O$:

```
 100 INPUT I$
 210 LET T$=0: GO SUB 5000: LET
SP=IS: IF SP>0 THEN  PRINT O$;"
";
```

Cada complemento directo pode ser comparado pelo mesmo processo, formando-se para tal um ciclo; porém é essencial reposicionar o indicador de comprimento das palavras WL para o (LEN) COMprimento da cadeia actual de cada vez. (Note-se que imprimir um ponto e vírgula depois de O$ garante que todas as palavras são impressas na mesma linha.)

```
200 RESTORE : FOR M=1 TO 8
205 READ O$: LET WL=LEN (O$)
220 NEXT M
```

Verificações similares podem ser feitas para eorrespondências com palavras nas listas de advérbios e adjectivos. Note-se que a cláusula RESTORE (número de linha) é utilizada de modo a não ser necessário ler (READ) a DATA pela ordem em que na realidade foi introduzida.

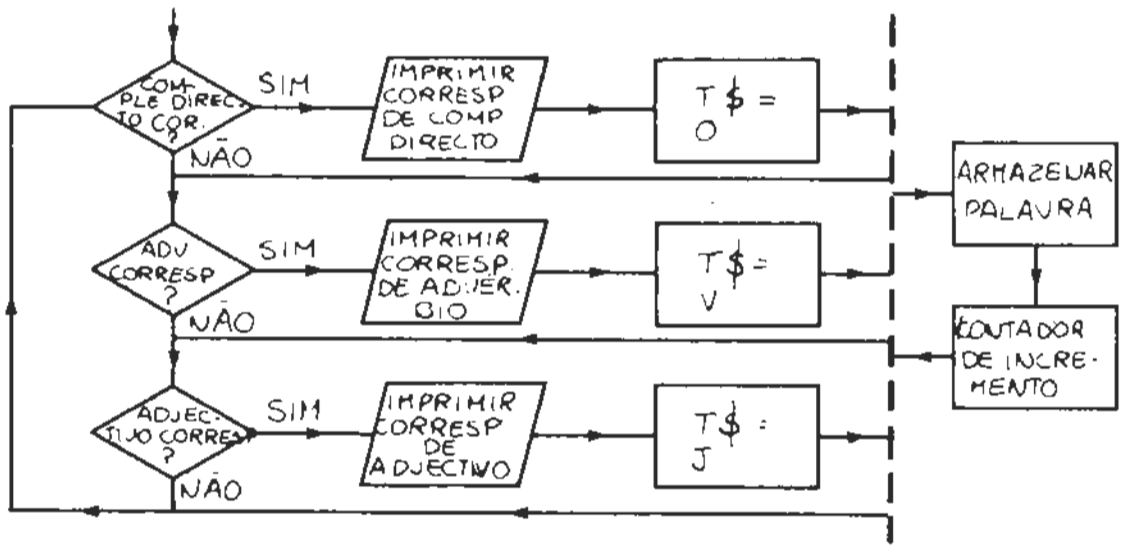
```
 300 RESTORE 9060: FOR M=1 TO 3
 305 READ V$: LET WL=LEN (V$)
 310 LET I$=V$: GO SUB 5000: LET
 SP=IS: IF SP>0 THEN PRINT V$;"
";
 320 NEXT M
```

```
400 RESTORE 9030: FOR M=0 TO 5
405 READ J$: LET WL=LEN (J$)
410 LET I$=J$: GO SUB 5000: LET
SP=IS: IF SP>0 THEN PRINT J$;"
"
420 NEXT M
500 LET I$=""
510 PRINT "OUTRA?"
520 INPUT Q$
530 CLS
540 GO TO 100
```

Para relatar o que aconteceu foi descoberta a maneira de o fazer; assim, de modo a poder-se usar mais tarde as palavras encontradas, armazenaremos cada palavra num bloco de dados à medida que for detectada. Ainda temos uma tabela de armazenamento de palavras, W$, mas a qual será ampliada para conter até 20 palavras (o que deve ser suficiente mesmo para uma frase muito prolixa!).

```
10 DIM W$(20,9)
20 LET WC=1
```

Se for detectada uma correspondência, uma cadeia temporária E$ é colocada em igualdade com a palavra combinada por correspondência, e chamada uma sub-rotina na linha 1500, que coloca a palavra detectada no primeiro elemento da tabela (ver ordinograma 3.7).



Ordinograma 3.7 — *Pesquisa por exclusão de hipóteses*


```
210 LET I$=O$: GO SUB 5000: LET
SP=IS: IF SP>0 THEN LET E$=O$:
PRINT E$;";: GO SUB 1500
1500 LET W$(WC)=E$
```

O contador de palavras WC é então incrementado, de modo que a próxima palavra seja colocada no próximo elemento, antes de se proceder a um retorno.

```
1520 LET WC=WC+1
1530 RETURN
```

Se foi utilizada uma cadeia temporária E$ na rotina de «armazenamento de palavras», isso significa que também podemos usá-la nos testes para os advérbios e para os adjectivos exactamente da mesma maneira.

```
310 LET I$=V$: GO SUB 5000: LET
SP=IS: IF SP>0 THEN LET E$=V$:
PRINT E$;" ";: GO SUB 1500
410 LET I$=J$: GO SUB 5000: LET
SP=IS: IF SP>0 THEN LET E$=J$:
PRINT E$;" ";: GO SUB 1500
```

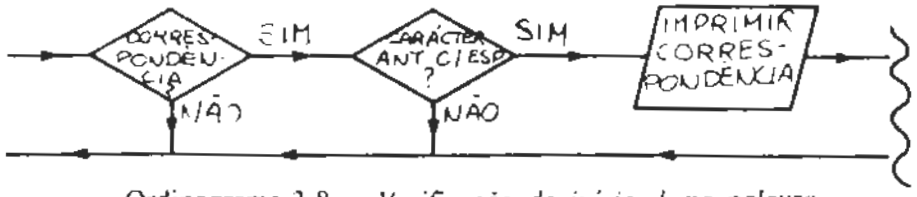
Correspondência parcial

Uma vantagem da pesquisa por exclusão de hipóteses é a de se poder facilmente proporcionar o reconhecimento duma série de palavras relacionadas entre si observando-se para tal simplesmente alguns caracteres-chave. Isto é obviamente útil, uma vez que lhe poupa o trabalho de ter de colocar para comparação ambos os substantivos no singular e no plural como, por exemplo, BISCOITO e BISCOITOS. Se emendar a DATA na linha 9010, tal como se apresenta abaixo, ambos os números morfológicos serão reconhecidos.

```
9010 DATA "BISCOITO","SORTIDO","
BOLO"
```

Todavia, a vida não é assim tão simples, uma vez que usar SORTIDO em lugar de SORTIDOS pode originar alguns resultados inesperados. No seu aspecto positivo detectará SORTIDO, SORTIDOS. Contudo, e infelizmente (no exemplo em inglês, o qual retrata melhor o inconveniente desta situação opcional), para além de «BUN» satisfazer BUN, BUNS e BUNFIGHT, infelizmente também o faz em relação a BUNCH, BUNDLE, BUNGALOW, BUNGLE, BUNK, BUNION e BUNNY!

Este problema não se restringe somente a prefixos, uma vez que o computador também não distinguirá entre PATO e SAPATO. Poder-se-ia ter incluído um verificador de que o carácter antes do início de cada correspondência era um espaço (isto é, que se tratava do princípio duma palavra, ver ordinograma 3.8). SP dá a posição actual do início de palavra, de forma que I$(SP — 1) é o carácter antes da respectiva palavra.



Ordinograma 3.8 — *Verificação do início duma palavra*

```
 210 LET I$=O$: GO SUB 5000: LET
 SP=IS: IF SP=0 THEN NEXT M: GO
TO 300
 211 IF I$(SP-1)<>" " THEN NEXT
M: GO TO 300
 212 LET E$=O$: PRINT E$;" ";: G
O SUB 1500
 310 LET I$=V$: GO SUB 5000: LET
 SP=IS: IF SP=0 THEN NEXT M: GO
TO 400
 311 IF I$(SP-1)<>" " THEN NEXT
M: GO TO 400
 312 LET E$=V$: PRINT E$;" ";: G
O SUB 1500
 410 LET I$=J$: GO SUB 5000: LET
 SP=IS: IF SP=0 THEN NEXT M: GO
TO 500
 411 IF I$(SP-1)<>" " THEN NEXT
M: GO TO 500
 412 LET E$=J$: PRINT E$;" ";: G
O SUB 1500
```


Para isto funcionar correctamente para a primeira palavra, acrescente-se simplesmente um espaço ao início de I$.

```
110 LET I$=" "+I$
```

De maneira semelhante poderíamos usar verificadores para a próxima letra depois da correspondência, ou do comprimento da palavra, para palavras reconhecidas de tamanho restrito.

Ordenando as coisas

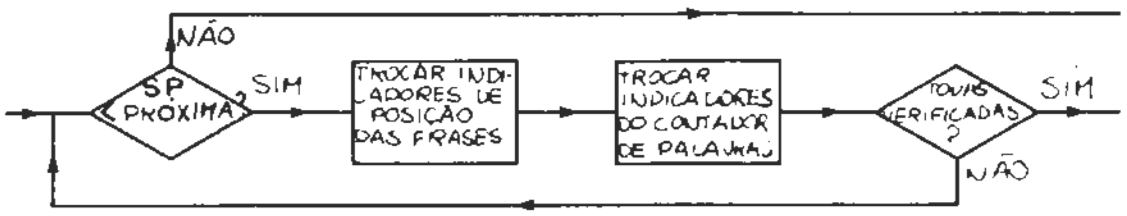
Ainda que já tenhamos detectado todas as palavras contidas na frase, a despeito das suas posições relativas ou o que quer que seja apresentado, elas são encontradas e armazenadas pela ordem pela qual aparecem na DATA. Isto acontece porque a comparação principia pelo primeiro item contido na lista dos complementos directos, em vez de o fazer com a primeira palavra na frase. Seria útil se pudéssemos rearranjar a tabela de armazenamento de palavras de forma que as palavras aí pertencentes ficassem pela ordem pela qual apareciam na frase.

Para fazer isto devemos manter um registo da posição da frase da palavra SP e do contador de palavras WC, uma vez que cada palavra é combinada por correspondência numa nova área de posicionamento de palavras P(n,n). Trata-se duma tabela a duas dimensões, com a posição da frase mantida no primeiro elemento, P(WC,0), e o contador de palavras, P(WC,1), no segundo elemento.

```
  15 DIM P(20,2)
1510 LET P(WC,1)=SP: LET P(WC,2)
=WC
```

A rotina de ordenação que executa o rearranjo encontra-se na linha 4000. Só deve ser atingida caso seja encontrada uma correspondência.

```
 440 IF WC=0 THEN GO TO 470
 450 GO SUB 4000
 460 GO TO 500
 470 PRINT "CORRESPONDENCIA NAO
ENCONTRADA"
```



Ordinograma 3.9 — *Colocando as palavras por ordem*

A rotina de ordenação (ver ordinograma 3.9) considera a posição da frase da primeira palavra encontrada [primeiro elemento na primeira dimensão P(1,1)] e compara-a com a posição da frase da segunda palavra encontrada [segundo elemento na primeira dimensão P(1 + 1,1)]. Se a variável de posieionamento para a primeira palavra for de valor superior ao da segunda palavra, então a primeira palavra encontrada está em primeiro lugar em relação à segunda palavra, daí que estas tenham, por conseguinte, de ser trocadas entre si. Isto colocará os indicadores de posição das palavras certos, mas os mareadores para a contagem das palavras também necessitam de ser rearranjados para as suas posições correctas. Este processo é repetido até que os indicadores das palavras estejam todos pela sua ordem correcta. Repare-se que os conteúdos das áreas da tabela para as cadeias de dados que contêm as palavras não são alterados, mas somente os indicadores (índices) que lhes são atribuídos.

```
4000 FOR N=1 TO WC-2
4010 IF P(N,1)<P(N+1,1) THEN NEX
T N: GO TO 4040
4020 LET D=P(N,1): LET P(N,1)=P(
N+1,1): LET P(N+1,1)=D
4030 LET D=P(N,2): LET P(N,2)=P(
N+1,2): LET P(N+1,2)=D: GO TO 40
00
```

Se as cadeias forem agora impressas pela ordem revista do contador de palavras (WC) ficarão como se encontravam na frase original, o que deverá facilitar a sua compreensão.

```
4040 PRINT : FOR N=1 TO WC-1
4050 PRINT MID$(P(N,2));" ";
4060 NEXT N: PRINT
```


Todos os elementos na tabela de posicionamento da frase P(N,1) e o contador de palavras WC devem ser reposicionados a 0 antes da próxima entrada.

```
4070 FOR N=1 TO 20
4080 LET P(N,1)=0
4090 NEXT N
4100 LET WC=1
4110 RETURN
```

CAPÍTULO 4

Dando respostas

Respostas mais precisas

Considerámos na sua amplitude como decifrar frases que são inscritas no computador, porém as respostas que têm sido produzidas até à altura têm-se mostrado limitadas e rígidas. Ainda que muitas das palavras originais numa frase sejam muitas vezes utilizadas numa resposta, numa conversação real damos atenção ao sujeito da frase e modificamos esta palavra de acordo com o contexto da resposta.

Por exemplo, a entrada seguinte:

EU PRECISO DE REPOUSO

poderá esperar pela resposta confirmatória:

VOCÊ PRECISA DE REPOUSO

e semelhantemente:

VOCÊ PRECISA DE REPOUSO

deverá originar:

EU PRECISO DE REPOUSO

Se se observar esta situação logicamente, aperceber-se-á de que para cada entrada do sujeito há uma saída de sujeito equivalente, e que simplesmente retirámos o sujeito original, acrescentando o resto da frase ao novo sujeito apropriado.

«EU» trata-se somente de dois caracteres, de forma que podíamos verificar I$(2). Se fosse «EU» então imprimiríamos (PRINT) «VOCÊ» (isto para o caso correspondente), sendo acrescentado à frente do resto da entrada I$(2 TO).

```
  10 INPUT I$
  30 IF I$( TO 2)="EU" THEN PRIN
T "VOCE"+I$(3 TO )
  60 GO TO 10
```

Do mesmo modo, os primeiros quatro caracteres I$(TO 4) podiam ser verificados em confronto com «VOCÊ» e quando necessário substituídos por «EU».

```
  50 IF I$( TO 4)="VOCE" THEN  P
RINT "EU"+I$(5 TO )
```

Se se tentar isso com uma série de frases, observar-se-á que funciona correctamente até que se inscreva qualquer coisa como o seguinte:

VOCÊ ESTÁ CANSADO

o que por sua vez é devolvido de forma bastante ininteligível:

EU ESTÁ CANSADO

Poder-se-ia contornar este problema fazendo a verificação das frases «EU ESTOU» e «VOCÊ ESTÁ», assim como «EU» e «VOCÊ» isoladamente; porém repare-se que se deve testar estes casos em primeiro lugar e juntar GO TO 10 ao fim das linhas 20 a 40 para se evitar que uma correspondência seja também encontrada com «EU» e «VOCÊ» isoladamente.

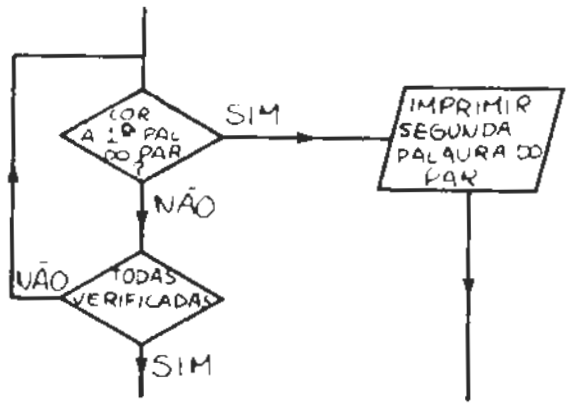
```
  20 IF I$( TO 8)="EU ESTOU" THE
N  PRINT "VOCE ESTA"+I$(9 TO ):
GO TO 10
  40 IF I$( TO 9)="VOCE ESTA" TH
EN  PRINT "EU ESTOU"+I$(10 TO ):
 GO TO 10
```

Ainda que este método funcione, rapidamente o programa se tornará ciclicamente muito extenso, visto ser necessária uma linha separada para cada possibilidade, uma vez que devemos ter em consideração o

comprimento da palavra ou frase a combinar por correspondência. Onde existir a necessidade de se verificarem por comparação muitas palavras, é por conseguinte melhor ler (READ) uma série completa de declarações de DATA e compará-las com a entrada por intermédio dum ciclo. É mais fácil evitar erros se as palavras de entrada e as correspondentes palavras de saída ou frases forem entradas na DATA em pares combinados correspondentes entre si e lidas (READ) por sua vez na tabela. Comece-se um novo programa com estas linhas, as quais instalam a DATA.

```
9000 DATA "EU","VOCE","VOCE","EU
","EU ESTOU","VOCE ESTA","VOCE E
STA","EU ESTÓU"
```

Usaremos outra vez uma pesquisa de cadeias cíclica por exclusão de hipóteses, a qual imprimirá de momento a palavra ou frase correspondente, O$, à combinada na entrada, T$ (ver ordinograma 4.1). Uma vantagem da pesquisa de cadeias por exclusão de hipóteses neste caso é que fará satisfatoriamente a correspondência com espaços inseridos em frases, uma vez que não dividimos I$ por «palavras» antes de se efectuar a correspondência por combinação.



Ordinograma 4.1 — *Usando uma resposta correspondente*

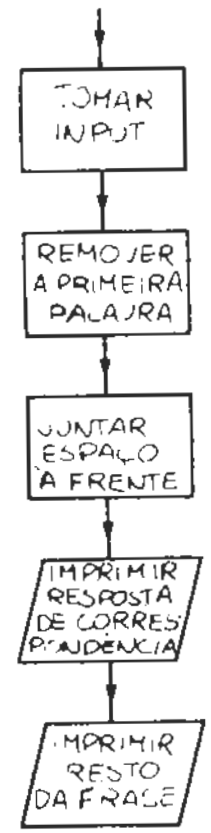
```
 100 INPUT I$: LET I$=I$+" "
 110 LET SI=2
 200 RESTORE : FOR M=1 TO 4
 205 READ N$,O$: LET WL=LEN (N$)
 210 LET T$=N$: GO SUB 5000: LET
 SP=IS: IF SP>0 THEN PRINT O$: G
O TO 100
 220 NEXT M
 250 GO TO 100
```


Se agora se entrar com qualquer frase que contenha a palavra «VOCÊ», por exemplo, o computador responderá com o oposto (ou seja, «EU»).

É melhor redefinir a palavra de resposta requerida para o efeito como uma nova cadeia de dados, a qual é a primeira parte da resposta R$, imprimindo-se então isto quando o ciclo for abandonado.

```
210 LET T$=N$: GO SUB 5000: LET
SP=IS: IF SP>0 THEN  LET R$=O$:
GO TO 230
230 PRINT R$
```

Para se conseguir uma resposta completa, precisamos de juntar isto ao resto da frase original S$ (depois de se inserir um espaço, ver ordinograma 4.2). Não é difícil definir o «resto da frase». Precisamos de subtrair a posição final da palavra do comprimento da frase. Lembre-se que SP indica o princípio da palavra combinada por correspondência, e que temos um registo do (LEN) COMprimento desta palavra em WL. Por consequência, o «resto da frase» é IS(SP + WL TO).



Ordinograma 4.2 — *Uma resposta completa*

```
210 LET I$=N$: GO SUB 5000: LET
SP=IS: IF SP=0 THEN GO TO 220
215 LET R$=O$: LET S$=" "+I$(SP
+WL TO )
230 PRINT R$;S$
```

Agora, quando tentar:

EU SOU INTELIGENTE

o computador concorda:

VOCÊ É INTELIGENTE

Porém, se premir então ENTER (entrar) outra vez ainda lhe diz que você é inteligente — o que não é verdade, uma vez que não se limpou IS, R$ e S$ antes de se voltar atrás no ciclo para a próxima entrada!

```
100 LET I$="": INPUT I$: LET I$
=I$+" "
240 LET R$="": LET S$=""
```

Antes de se julgar demasiado esperto, experimente o seguinte:

NÓS SOMOS ESTÚPIDOS

o que poderá bem surpreendê-lo quando lhe for transmitida a seguinte resposta:

VOCÊ

Se pensar por uns momentos aperceber-se-á de que uma das nossas palavras-chave se encontra escondida dentro de outra palavra nesta frase particular. Se não o conseguir vislumbrar, tente então o seguinte:

NÓS SOMOS INCOMPETENTES

ao que o computador discorda de si, respondendo-lhe:

VOCÊ COMPETENTE

Ainda que cada palavra-chave seja testada independentemente, cada uma delas é colocada em R$ sempre que uma correspondência seja detectada, de forma que somente a última correspondência é mencionada. Uma vez que a palavra-chave é unicamente verificada uma só vez por cada frase, inserir «EU» só ocasiona problemas quando tal palavra não é a palavra-chave na frase.

Para contornar este problema devemos considerar quais as palavras que criam problemas. Ainda que a letra «I» (1.ª pessoa do sujeito na língua inglesa) seja bastante comum, é muito raro que seja a última letra numa palavra, e assim poderíamos verificar se existia um espaço depois da palavra-chave. Devemos tratar todas as palavras-chave da mesma maneira, de modo a acrescentar um espaço ao fim de cada uma delas. Note-se que não existe necessidade alguma de juntar espaços ao fim das respostas.

```
9000 DATA "EU","VOCE","VOCE","EU
","EU SOU","VOCE E","VOCE E","EU
 SOU"
```

Também precisamos agora de subtrair um carácter a IS para dar S$, visto que o espaço se tornou agora parte integrante da palavra-chave.

```
215 LET R$=O$: LET S$=" "+I$(SP
+WL TO )
```

O computador concordará agora prontamente acerca da sua incompetência.

Se a primeira palavra-chave não se encontra no início da frase, tudo o que estiver antes será ignorado na resposta. Por exemplo, a resposta a:

QUE ACONTECE SE EU CAIR?

será:

VOCÊ CAI?

Alguns resultados estranhos podem ainda ocorrer quando duas palavras-chave verdadeiras se encontram presentes na frase. Por exemplo:

SE EU E VOCÊ CAIRMOS

origina:

EU E EU CAÍMOS

e o seguinte:

QUE ACONTECE SE EU E VOCÊ CAIRMOS

responderá:

EU CAIO

Contudo, acrescentando mais palavras-chave adequadas é facilmente executado ao incluir-se mais DATA, ainda que algumas combinações não sejam aceitáveis.

```
 200 RESTORE : FOR M=3 TO 7
9010 DATA "NOS","NOS","ELES","EL
ES"
```

Dirigindo-se às respostas

Até ao momento o nosso computador pouco mais inteligência apresentou que um papagaio, uma vez que tem vindo meramente a titubear uma versão ligeiramente modificada da entrada. O próximo passo, por conseguinte, é fazer com que tome algumas decisões lógicas com base nas entradas antes de dar uma resposta.

NOTA. — Alguns dos exemplos apresentados só funcionam correctamente com as indicações especificadas para os casos correspondentes na língua inglesa, já que o «resto da frase» que contém o tempo de conjugação do verbo só é invariável nesta língua, ao contrário do que acontece em português.

Os números de sujeitos, SU, verbos, VB, e respostas, RP, são definidos como variáveis, de forma que o programa pode ser faeilmente expandido. SU define o número de sujeitos reconhecidos nas frases de entrada e saída, VB os verbos legítimos, e RP uma série de respostas correspondentes.

```
 10 LET SU=27: LET VB=7: LET RP
=7
```


As primeiras duas linhas de DATA contêm sujeitos de entrada e saída (input/output) combinados em pares (ver quadro 4.I). Uma vez que os pronomes («EU», «TU/VOCÊ», etc.) se encontram frequentemente ligados a outras palavras para formar frases (tal como «EU TENHO» — sendo mais evidente a expressão de contracção correspondente no inglês «I'VE»), estas formas combinadas são igualmente

QUADRO 4.1

Pares de sujeitos em SU$(n,n)

ENTRADA	SAÍDA	INPUT	OUTPUT
EU TENHO	TU TENS	I HAVE	YOU HAVE
EU SOU	TU ÉS	I'VE	YOU'VE
TU TENS	EU TENHO	I AM	YOU ARE
TU ÉS	EU SOU	I'M	YOU'RE
TU	EU	YOU HAVE	I HAVE
ELA TEM	ELA TEM	YOU'VE	I'VE
ELA É	ELA É	YOU ARE	I AM
ELA	ELA	YOU'RE	I'M
ELES SÃO	ELES SÃO	YOU	I
ELES	ELES	SHE HAS	SHE HAS
ELE TEM	ELE TEM	SHE IS	SHE IS
ELE É	ELE É	SHE'S	SHE'S
ELE	ELE	SHE	SHE
NÓS TEMOS	NÓS TEMOS	THEY'VE	THEY'VE
NÓS SOMOS	NÓS SOMOS	THEY ARE	THEY ARE
NÓS	NÓS	THEY'RE	THEY'RE
EU	TU	THEY	THEY
		HE HAS	HE HAS
		HE IS	HE IS
		HE'S	HE'S
		HE	HE
		WE HAVE	WE HAVE
		WE'VE	WE'VE
		WE ARE	WE ARE
		WE'RE	WE'RE
		WE	WE
		I	YOU

incluídas na DATA. Note-se que são dispostas por tal ordem que a frase mais completa, contendo uma palavra-chave, é encontrada sempre primeiro. Um espaço é acrescentado ao fim de cada elemento, de modo que é evitada qualquer sobreposição de correspondências parciais, sendo um espaço automaticamente formado na resposta.

```
9000 DATA "EU TENHO ","TU TENS "
,"EU SOU ","TU ES ","TU TENS ","
EU TENHO "
9010 DATA "TU ES ","EU SOU ","TU
 ","EU "
9020 DATA "ELA TEM ","ELA TEM ",
"ELA E ","ELA E ","ELA ","ELA "
9030 DATA "ELES SÃO ","ELES SAO
" ,"ELES ","ELES "
9040 DATA "ELE TEM ","ELE TEM ",
"ELE E ","ELE E ","ELE ","ELE ",
"NOS TEMOS ","NOS TEMOS "
9050 DATA "NÓS SOMOS ","NOS SOMO
S","NOS ","NOS ","EU ","TU "
```

```
9000 DATA "I HAVE ","YOU HAVE "
"I'VE","YOU'VE ","I AM ","YOU A
RE ","I'M ","YOU'RE ","YOU HAVE
" "I HAVE "
9010 DATA "YOU'VE ","I'VE ","YOU
 ARE ","I AM ","YOU'RE ","I'M ",
"YOU ","I "
9020 DATA "SHE HAS ","SHE HAS ",
"SHE IS ","SHE IS ","SHE'S ","SH
E'S ","SHE ","SHE "
9030 DATA "THEY'VE ","THEY'VE ",
"THEY ARE ","THEY ARE ","THEY'RE
 ","THEY'RE ","THEY ","THEY "
9040 DATA "HE HAS ","HE HAS ","H
E IS ","HE IS ","HE'S ","HE'S ",
"HE ","HE ","WE HAVE ","WE HAVE
"
9050 DATA "WE'VE ","WE'VE ","WE
ARE ","WE ARE ","WE'RE ","WE'RE
","WE ","WE ","I ","YOU "
```

A próxima linha de DATA contém os verbos principais. O verbo «to be» (ser ou estar) é omitido, visto as suas variações de conjugação serem tão complicadas, além de muitas das suas versões serem já consideradas na verificação do sujeito.

```
9060 DATA "HATE ","LOVE ","KILL
","DISLIKE ","LIKE ","FEEL ","KN
OW "
```


O último conjunto de DATA contém as respostas. Para simplificar a compreensão das coisas e proceder à verificação nesta fase, todas as respostas contêm o verbo original, ainda que, como é óbvio, pudessem referir o que quer que fosse.

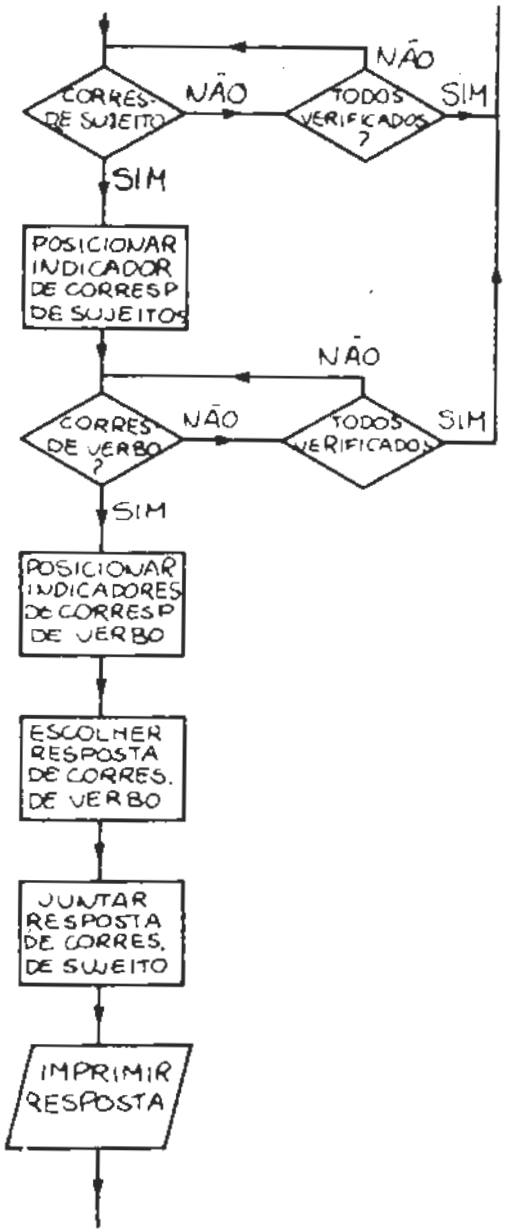
```
9070 DATA "PROBABLY HATE YOU AS
NELL ","LOVE YOU TOO ","KILL YOU
"
9080 DATA "DISLIKE LOTS OF THING
S ","LIKE CHIPS ","FEEL POWERFUL
","KNOW EVERYTHING "
```

NOTA. — Estes últimos exemplos de descrição do programa não foram aqui adaptados em tradução aos casos correspondentes em português pelas mesmas razões atrás expostas.

Correspondência

Os primeiros pares SU dos elementos de DATA são agora lidos, sendo a cadeia de dados de entrada comparada com a lista de sujeitos de entrada, N$ (ver ordinograma 4.3). Se não existir qualquer correspondência, então é pedida uma nova entrada, ou então A$ é declarado como sendo igual à palavra correspondente, B$ é posicionado no seu valor inverso, e uma variável de correspondência do sujeito, SM, é posicionada no número de sujeito para o qual foi encontrada uma correspondência, M.

```
 100 LET I$="": INPUT I$: LET I$
=I$+" "
 200 RESTORE : FOR M=1 TO SU
 210 READ N$,O$: LET WL=LEN (N$)
 220 LET T$=N$: GO SUB 5000: LET
 SP=IS: IF SP>0 THEN LET A$=N$:
LET B$=O$: LET SM=M: GO TO 300
 230 NEXT M
 240 GO TO 100
```



Ordinograma 4.3 — *Posicionamento de indicadores de referência de correspondências*

Toda a DATA de verbos (VB) é agora comparada com **I$**. Se nenhum verbo for encontrado, a entrada é rejeitada, ou então a variável de correspondência do verbo VM é colocada igual a M. Note-se que o indicador de DATA deve ser redepositado (RESTORE) para a linha 9060, visto que se sai da lista de DATA de sujeitos mal uma correspondência seja detectada.


```
 300 RESTORE 9060: FOR M=1 TO VB
 310 READ V$: LET WL=LEN (V$)
 320 LET T$=V$: GO SUB 5000: LET
 SP=IS: IF SP>0 THEN LET VM=M: G
0 TO 500
 330 NEXT M
```

Dando respostas

Agora que o sujeito e o verbo foram identificados, podemos escolher a resposta apropriada utilizando VM como um referenciador para a DATA de respostas, que principia na linha 9070.

```
 500 RESTORE 9070: FOR N=1 TO VM
: READ R$: NEXT N
```

No caso mais simples podemos apenas juntar o sujeito apropriado (A$) à frente de R$ antes de o imprimir.

```
520 LET R$=A$+R$
550 PRINT R$
560 GO TO 100
```

Agora, por exemplo, se inscrever:

I HATE COMPUTERS (EU ODEIO COMPUTADORES)

o programa responder-lhe-á com o seguinte:

I PROBABLY HATE YOU AS WELL (EU PROVAVELMENTE ODEIO-O IGUALMENTE)

e o seguinte:

I KNOW A LOT (EU SEI BASTANTE)

origina:

I KNOW EVERYTHING (EU SEI TUDO)

Sujeitos alternativos

Se preferir que a máquina concorde consigo, em lugar de tentar derrotá-lo no seu próprio jogo, mude simplesmente o sujeito acrescentado R$ à segunda metade do par (B$ = o «oposto»).

```
520 LET R$=B$+R$
```

agora, o seguinte:

I KNOW A LOT

origina:

YOU KNOW EVERYTHING (TU SABES TUDO)

Para mais diversidade, pode-se escolher o sujeito aleatoriamente a partir do primeiro ou segundo elementos, de forma que a resposta não seja predizível.

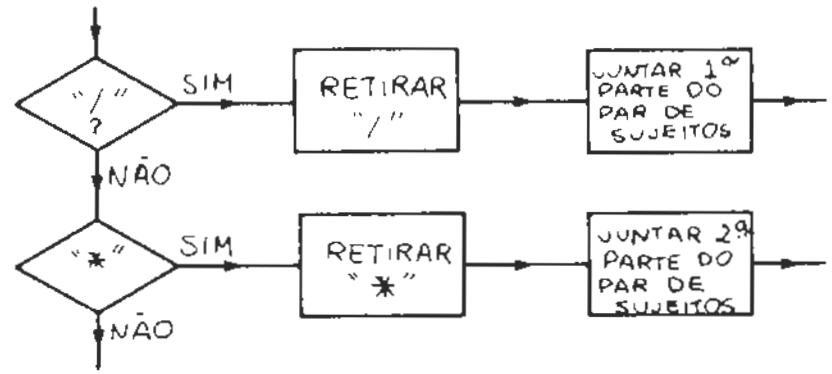
```
 510 LET RS=INT (2*RND)
 515 IF RS=0 THEN LET R$=A$+R$:
GO TO 550
 530 LET R$=A$+R$
```

Introduzindo o sujeito no contexto

Seria no conjunto mais correcto se escolhêssemos o sujeito adequado de acordo com o contexto da resposta, porém para o fazer devemos dispor de marcadores nas respostas. Usaremos uma barra, «/», para indicar que a primeira palavra no par de DATA dos sujeitos é para ser utilizada, e um asterisco, «*», para indicar que a segunda palavra no par de DATA é para ser utilizada.

```
9070 DATA "/PROBABLY HATE YOU AS
 WELL ","/LOVE YOU TOO ","/KILL
YOU "
9080 DATA "*DISLIKE LOTS OF THIN
GS ","*LIKE CHIPS ","*FEEL POWER
FUL ","*KNOW EVERYTHING "
```


Podemos pesquisar a cadeia de respostas R$ indicada pelo marcador de verbos VM por uma barra «/» referenciadora, contanto que redesignemos isto como I$ antes de entrarmos com o verificador INSTR. Se for encontrada uma barra, então o conteúdo da primeira parte do par de sujeitos (A$) é acrescentado à resposta R$, menos o primeiro carácter (a barra, ver ordinograma 4.4).



Ordinograma 4.4 — *Introduzindo o sujeito no contexto*

```
 510 LET I$=R$: LET WL=1: LET T$
="/": GO SUB 5000
 520 IF IS>0 THEN GO TO 800
 800 LET R$=A$+R$(2 TO )
 810 GO TO 530
```

Se nenhuma barra for encontrada na resposta, uma segunda pesquisa é executada por um asterisco, «*». Caso este seja encontrado, então a segunda metade do par (B$) é usada da mesma maneira.

```
530 LET T$="*": GO SUB 5000
540 IF IS>0 THEN GO TO 820
820 LET R$=B$+R$(2 TO )
830 GO TO 550
```

Agora, se tivermos o seguinte:

I LOVE ME (EU GOSTO DE MIM)

originará:

I LOVE YOU TOO (EU GOSTO TAMBÉM DE TI)

mas:

I FEEL POWERFUL (EU SINTO-ME FORTE)

produz o seguinte:

YOU FEEL PORWERFUL (TU SENTES-TE FORTE)

Inserção em frases

Para simplificar as coisas, começámos sempre as nossas frases de resposta com o sujeito, mas na vida real nem sempre é este o caso. Agora que temos marcadores de referência nas respostas para indicar qual o tipo de sujeito a ser acrescentado, podemos igualmente utilizá-los para indicar onde se deve inserir esta palavra ou frase na resposta. Em primeiro lugar corrigiremos a DATA de modo a que a palavra a ser inserida nunca se encontre no início, para fazer com que o processo de inserção seja óbvio.

```
9070 DATA "DO YOU REALISE THAT /
PROBABLY HATE YOU AS WELL "  "WEL
L /LOVE YOU TOO "  ,"IF /DON'T KIL
L YOU FIRST "
9080 DATA "SO WHAT *DISLIKE LOTS
 OF THINGS "  ,"DO /LIKE CHIPS "  "
WHY DO *FEEL POWERFUL "  ,"HOW DO
*KNOW EVERYTHING "
```

Já temos um registo da posição onde se deve inserir a palavra, uma vez que IS nos indica onde foi encontrada uma barra ou um asterisco na resposta. Tudo o que precisamos de fazer é considerar a parte da resposta antes do marcador de referência, R$(TO IS − 1), juntar a versão correcta do sujeito (A$ ou B$), e só então o resto da resposta, R$(IS + 1 TO).

```
 800 LET R$=R$( TO IS-1)+A$+R$(I
S+1 TO )
 820 LET R$=R$( TO IS-1)+D$+R$(I
S+1 TO )
```


Agora, o seguinte:

I WILL KILL HIM (EU MATÁ-LO-EI)

produz:

IF I DON'T KILL YOU FIRST (SE EU NÃO TE MATAR PRIMEIRO)

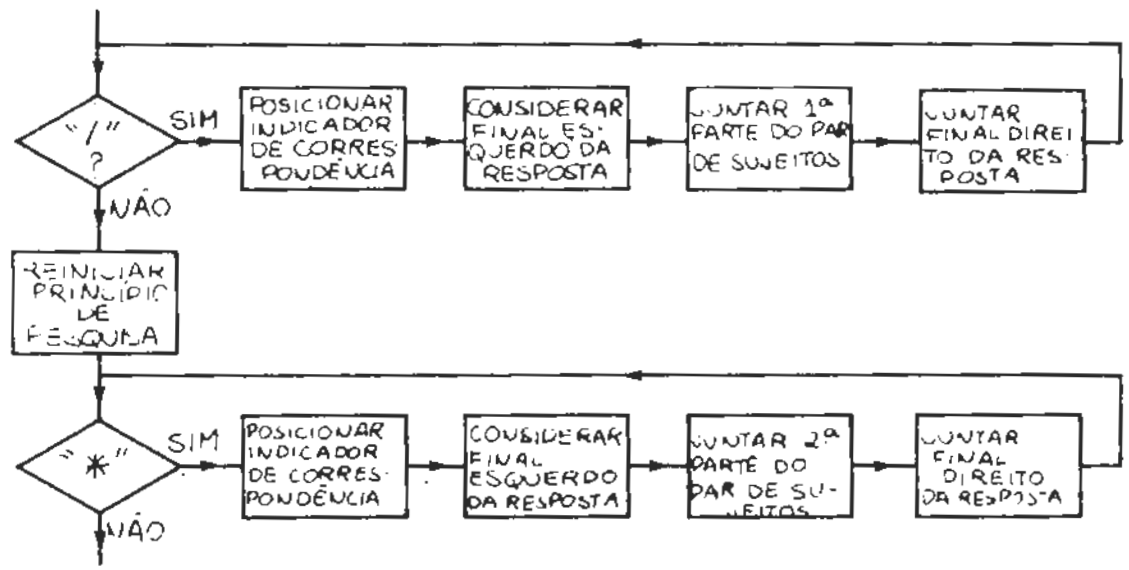
e o seguinte:

I DISLIKE COMPUTERS (EU NÃO APRECIO COMPUTADORES)

proporciona o seguinte:

SO WAHT YOU DISLIKE LOTS OF THINGS (PORTANTO TU NÃO APRECIAS UMA SÉRIE DE COISAS)

Ainda que estejamos agora a inserir o sujeito na frase de resposta de uma forma mais natural, estamos unicamente a lidar com um sujeito por frase. Uma outra modificação menor permitir-nos-á inserir qualquer número de sujeitos numa frase. Tudo o que precisamos de fazer é continuar a repetir a pesquisa para marcadores até que mais nenhum seja encontrado. Uma variável de início, ST, é definida como I na linha 500, sendo então processada uma pesquisa para o



Ordinograma 4.5 — *Inserção numa frase*

primeiro tipo de marcador. Quando uma correspondência é detectada, ST é reposicionado a mais um do que a posição de correspondência. Quando RS tiver sido modificado pela linha 800, precisaremos então de saltar atrás para a linha 510 para procurar por mais marcadores. Se nenhuma correspondência tiver sido encontrada para o primeiro marcador, então ST é reposicionado a 1. O segundo tipo de marcador é então verificado da mesma maneira.

```
 505 LET ST=1
 520 IF IS>0 THEN LET ST=IS+1: G
O TO 800
 525 LET ST=1
 530 LET I$=R$: LET T$="*": GO S
UB 5000
 540 IF IS>0 THEN LET ST=IS+1: G
O TO 820
 810 GO TO 510
 830 GO TO 530
9080 DATA "SO WHAT /DISLIKE LOTS
 OF THINGS ESPECIALLY * ","DO /L
IKE CHIPS ","WHY DO *FEEL POWERF
UL ","*THINK *KNOW EVERYTHING "
```

Agora, o seguinte:

I KNOW EVERYTHING (EU SEI TUDO)

produz:

YOU THINK YOU KNOW EVERYTHING (TU PENSAS QUE SABES TUDO)

e:

I DISLIKE COMPUTERS (EU NÃO APRECIO COMPUTADORES)

dá:

SO WHAT I DISLIKE LOTS OF THINGS ESPECIALLY YOU (PORTANTO EU NÃO APRECIO UMA SÉRIE DE COISAS ESPECIALMENTE TU)

Objecções no sujeito

Tudo parece estar a começar a ficar favoravelmente fácil até se experimentar qualquer coisa como o seguinte:

EU ODEIO-TE

que dá como resposta:

DO YOU REALISE THAT YOU PROBABLY HATE YOU AS WELL (APERCEBES-TE QUE PROVAVELMENTE TE DETESTAS IGUALMENTE)

O problema que se põe aqui é que estamos a sair da rotina de pesquisa mal a primeira correspondência é detectada, e ainda que estejamos a fazer a verificação para o sujeito «I», encontramos o complemento directo «YOU» (TE) primeiro. Uma vez que «YOU» vem antes de «I» na tabela de sujeitos, é encontrado primeiro, apesar do facto de aparecer mais adiante na frase.

Visto não podermos praticamente simular todas as complexidades do cérebro humano, teremos de assumir que o sujeito vem sempre antes do verbo, e o complemento directo depois deste. Até ao momento, no programa, temos estado a verificar o sujeito antes de o fazermos para o verbo, de forma que teremos de inverter essa ordem, substituindo as linhas 200-240 pelas linhas equivalentes 400-440. A posição do verbo na entrada é o valor de SP quando é encontrado um verbo, de modo que guardaremos tal como um indicador da posição do verbo, VP.

```
220 LET T$=V$: GO SUB 5000: LET
 SP=IS: IF SP>0 THEN LET VM=M: L
ET VP=SP: GO TO 400
```

Agora, quando uma correspondência com a tabela de sujeitos for encontrada, podemos comparar essa posição SP com o indicador de verbo armazenado VP, e rejeitar a correspondência caso esta esteja posicionada depois do verbo (ver ordinograma 4.6).

```
 400 RESTORE : FOR M=1 TO SU
 405 READ N$,O$: LET WL=LEN (N$)
 410 LET I$=N$: GO SUB 5000: LET
 SP=IS: IF SP>0 AND SP<VP THEN L
ET A$=N$: LET B$=O$: LET SM=L: G
O TO 500
```

Alteração do tempo do verbo

Ainda que ambos os verbos «LIKE» e «DISLIKE» contenham a sequência «L-I-K-E», encontramos «DISLIKE» correctamente uma vez que se encontra antes de «LIKE» na tabela. Se mudarmos para o tempo passado do verbo, pode ser ou não encontrado. Com os cinco primeiros verbos a situação é francamente simples: para se mudar para o tempo passado acrescentamos-lhe simplesmente um «D» ao fim da forma no tempo presente. Ambas as formas são por conseguinte aceites.

Ordinograma 4.6 — *Rejeição de correspondências para os complementos directos*

HATE	HATED
LOVE	LOVED
KILL	KILLED
DISLIKE	DISLIKED
LIKE	LIKED

Contudo, com os dois últimos verbos a palavra muda completamente, de forma que não pode haver uma correspondência directamente simples. Ainda que possamos conseguir uma maneira de fazer a verificação para «KN», uma vez que se trata duma combinação rara, não existe qualquer maneira prática de podermos usar um grupo tão comum como «FE» para uma palavra-chave.

FEEL	FELT
KNOW	KNEW

É mais fácil se tratarmos todos os verbos da mesma maneira e, se não existirem quaisquer restrições na memória, podemos simplesmente colocar todas as possíveis versões na tabela de verbos por pares.

```
  40 LET SU=27: LET VB=14: LET R
P=7
9060 DATA "HATE ","HATED ","LOVE
 "," LOVED ","KILL ","KILLED ","D
ISLIKE ","DISLIKED
9065 DATA "LIKE ","LIKED ","FEEL
 ","FELT ","KNOW ","KNEW "
```

A menos que queiramos ter diferentes respostas para os diferentes tempos dos verbos, teremos agora de dividir em duas partes a variável de correspondêneia do verbo VM, para indicar a resposta correcta para ambas as formas.

```
230 LET VM=M/2: LET VP=IS
```

CAPÍTULO 5

Sistemas inteligentes especializados

Um especialista humano é qualquer pessoa que tem conhecimentos com bastante profundidade e extensão sobre um assunto particular e que lhe pode dar conselhos precisos («opinião de especialista») sobre questões relacionadas com esse assunto em particular. Tal especialização somente é adquirida depois de um longo período de treino e de bastante experiência, de modo que infelizmente os verdadeiros especialistas são poucos, encontrando-se bastante espalhados entre a população. Para além do facto de na maioria das vezes não se encontrarem à mão quando um problema precisa de ser resolvido.

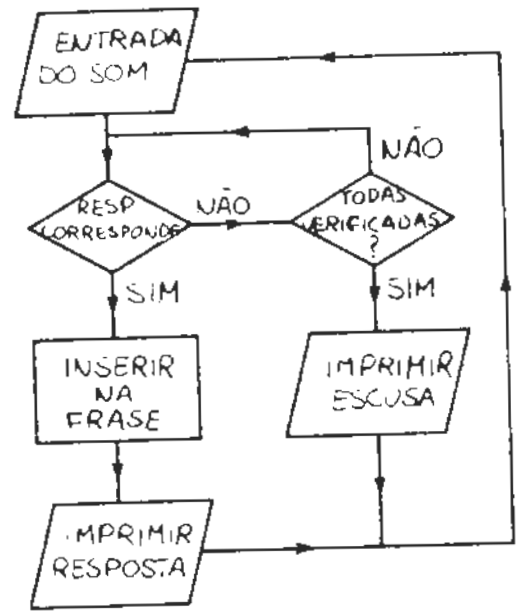
Os cientistas, por conseguinte, têm-se aplicado ao problema da produção de programas para computadores, os quais simulam as funções desses especialistas humanos. Tais programas têm a vantagem de poderem ser facilmente copiados para produzir uma infinidade de especialistas, e, como é óbvio, não precisam de intervalos para o café, repouso, aumentos de ordenado, etc., ou tudo isto em conjunto! Claro que o computador deve ser totalmente lógico, podendo somente seguir instruções pré-programadas dadas pelo programador. É interessante notar que os autores de romances de ficção científica tenham previsto problemas quando os especialistas da «última vaga» (tais como HAL em *2001: Odisseia no Espaço* ou os *robots* positrónicos de Isaac Asimov) se defrontam com direcções alternativas que entram em conflito com mais de uma das suas primeiras directivas, e as quais produzem, não avarias ou destruição dos sistemas, mas sim «pseudo-colapsos nervosos».

Antes de podermos começar a escrever programas para «sistemas inteligentes», devemos perguntar a nós próprios como é que trabalha um especialista humano.

Vamos primeiro considerar a situação mais simples, onde a tarefa do especialista é encontrar a resposta para um problema conhecido.

- Primeiro que tudo ele faz um levantamento sobre a informação respeitante à tarefa actual.
- Em segundo lugar compara esta informação com a que se encontra armazenada no seu cérebro e procura estabelecer uma correspondência de valores.
- Por fim faz um relatório sobre a situação, que indica se uma correspondência foi ou não encontrada.

Aquilo de que necessitamos aqui é simplesmente de um programa de base de dados, o qual tenta o estabelecimento da correspondência existente entre uma entrada em confronto com informação que se encontre armazenada (ver ordinograma 5.1). Um sistema utilitário-familiar aceitaria linguagem natural (ver atrás), mas para manter as coisas simples limitar-nos-emos a um formato de entrada fixo. Para começar vamos dar atenção a animais sonoramente reconhecíveis, observando em registo os sinais sonoros que produzem. Fixamos pares de DATA nos quais as primeiras palavras, Q$, contêm os sons que são reconhecíveis, e as segundas palavras, A$, contêm os nomes respectivos dos animais relevantes.



Ordinograma 5.1 — *Um simples «especialista»*

```
9000 DATA "MIAOW","GATO","QUAW"
"CAO","MEUH","VACA","HUT","MOCHÓ
","YIEEH","CAVALO"
```

Agora precisamos apenas de perguntar por um som e compará-lo com os conteúdos de cada possível Q$ de cada vez.

```
  20 PRINT "QUE SOM PRODUZ"
  30 INPUT I$: PRINT I$
  40 RESTORE : FOR N=1 TO 5: REA
D Q$,A$: IF I$=Q$ THEN GO TO 100
  50 NEXT N
  60 PRINT "DESCULPE, NAO ENCONT
RO ESSE"
  70 GO TO 20
 100 PRINT "UM ANIMAL QUE ";Q$;"
 E' UM ";A$
 110 GO TO 20
```

Talvez devêssemos referir nesta altura que o nosso computador especializado pode bem ser melhor nesta tarefa que o género humano, visto não poder elaborar julgamentos subjectivos, tornar-se talvez fastidioso, ou acidentalmente esquecer-se de verificar toda a informação contida na sua memória. Por outro lado, não é literalmente muito explítico.

Ramificando-se

O exemplo descrito acima é bastante simples, uma vez que apenas é feita uma pergunta, havendo somente uma possível resposta. Na realidade necessitamos de ser capazes de lidar com problemas mais difíceis, onde a resposta não pode ser encontrada sem se ter de passar por uma série de questões. Por exemplo, que deveria executar um especialista se metesse a chave no interruptor da ignição do seu carro e a ligasse, mas nada acontecesse?

Poderia haver um determinado número de razões para tal:

BATERIA EM BAIXO (DESCARREGADA)
MÁS LIGAÇÕES ELÉCTRICAS
INTERRUPTOR DANIFICADO
ARRANCADOR EMPERRADO
ARRANCADOR PARTIDO
BOBINAS PARTIDAS

Para descobrir a causa deve-se seguir um percurso lógico e proceder a um determinado número de verificações. A primeira coisa a fazer é verificar se é somente ou não o arrancador do motor que não está a funcionar:

ESTÁ A LUZ DA IGNIÇÃO LIGADA? (S/N)

Se a resposta a isto for «N» (para «não», obviamente), então existe ausência de corrente no interruptor, de modo que a causa se deve encontrar entre as três primeiras possibilidades listadas acima, sendo, consequentemente, válida apenas uma. Podemos reduzir o raio de possibilidades ainda mais, averiguando se as luzes funcionam ou não:

AS LUZES FUNCIONAM CORRECTAMENTE? (S/N)

Se a resposta for afirmativa, então a bateria não se pode encontrar descarregada, devendo pois encontrar-se ligada correctamente à luz do interruptor. Presumivelmente o interruptor está partido, podendo-se fazer uma sugestão com vista à sua substituição.

SUBSTITUIR INTERRUPTOR DA IGNIÇÃO

Se as luzes não trabalharem, então devem ser verificadas as ligações eléctricas.

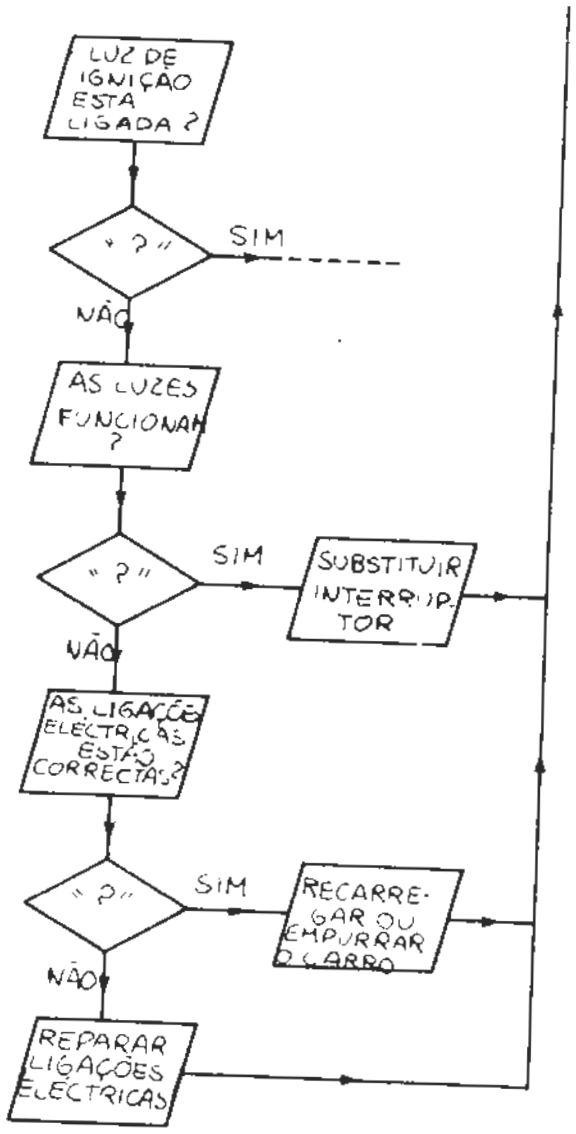
ESTÃO AS LIGAÇÕES DA BATERIA EM BOM ESTADO? (S/N)

Se a resposta for sim, então a bateria está descarregada, de modo que se deve recarregá-la (ou empurrar o carro!).

RECARREGAR BATERIA OU EMPURRAR O CARRO

Pelo mesmo processo, podia ter sido executada uma sequência de verificações para se poder lidar com uma situação onde haja carga, mas na qual o mecanismo de arranque não funciona (as três últimas possibilidades).

A maneira mais simples de programar a estrutura ramificada do ordinograma 5.2 faz-se por intermédio duma série encadeada de testes com a cláusula IF-THEN.



Ordinograma 5.2 — *Uma ramificação especializada*

```
  10 PRINT "DIAGNOSTICO DE FALHA
S"
  20 PRINT
  30 PRINT "ESTA' LIGADA A LUZ D
A IGNICAO (S/N)"
  40 INPUT I$
  50 IF I$="S" THEN GO TO 180
  60 PRINT "AS LUZES FUNCIONAM C
ORRECTAMENTE (S/N)"
  70 INPUT I$
  80 IF I$="S" THEN GO TO 110
  90 PRINT "SUBSTITUIR O INTERRU
PTOR DA IGNICAO"
  95 INPUT D$
```



```
 100 RUN
 110 PRINT "ENCONTRAM-SE AS LIGA
COES DA BATERIA EM BOM ESTADO (S
/N)"
 120 INPUT I$
 130 IF I$="S" THEN GO TO 160
 140 PRINT "REPARAR AS LIGACOES"
 145 INPUT D$
 150 RUN
 160 PRINT "CARREGAR BATERIA OU
EMPURRAR O CARRO"
 165 INPUT D$
 170 RUN
 180 ----- etc -----
```

Este género de programa é relativamente fácil de escrever; mas, como já é hábito, torna-se ineficiente à medida que aumenta em extensão e se complica.

Determinando a direcção a tomar

Um modo mais eficiente de lidar com a situação está em colocar o texto em tabelas e dispor de indicadores de referência (índices), que o dirigem à próxima questão ou resposta, consoante for afirmativa ou negativa a resposta que se dá à pergunta actual (ver ordinograma 5.3).

O formato para a entrada de DATA para cada ponto de bifurcação dum ramo é, então:

(TEXTO), (Indicador para «SIM»), (Indicador para «NÃO»)

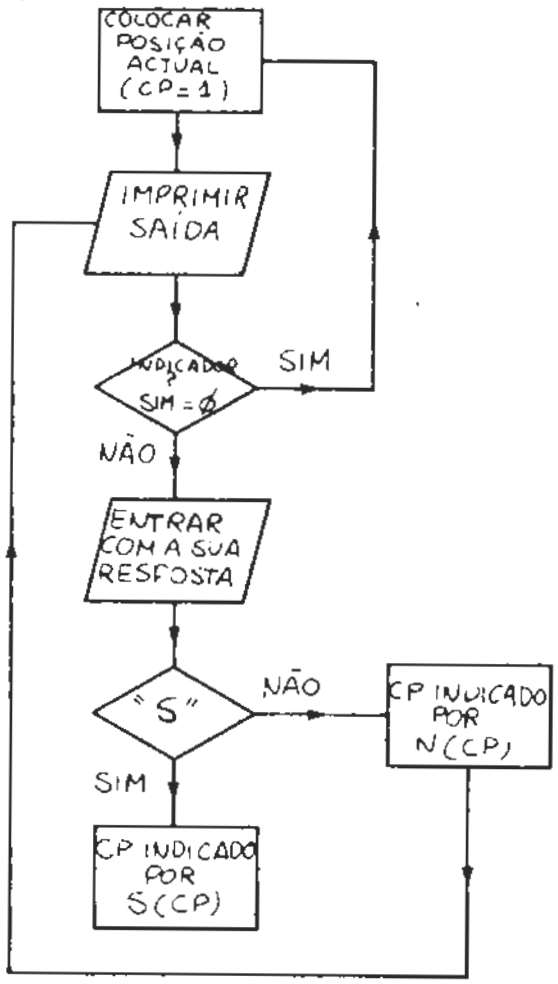
A primeira pergunta foi a seguinte:

ESTÁ A LUZ DA IGNIÇÃO LIGADA? (S/N) . . . 1

Se a resposta tivesse sido «N», então precisaria de perguntar pela segunda questão:

AS LUZES FUNCIONAM CORRECTAMENTE? (S/N) . . . 2

Caso contrário terá necessidade de continuar com a outra parte do diagnóstico (a qual não foi aqui incluída, mas que seria o ponto 7 a considerar).



Ordinograma 5.3 — *Dirigindo-se à próxima saída*

Precisamos de estabelecer três tabelas: O$(n) contém a saída (texto), S(n) o indicador para «sim», e N(n) o indicador para «não». As tabelas para «sim» e «não» contêm somente um carácter, mas o comprimento da tabela de saída (L) deve ser tomado em consideração. Para se conseguir que o programa seja fácil de modificar, é utilizada uma variável NP para o número de pontos a considerar. A DATA é lida por grupos de três para cada elemento destas tabelas. Onde o ponto da DATA corresponde a um possível fim do programa, isso é indicado pelos indicadores S(n) e N(n), que são inicializados a zero.

```
  10 GO SUB 8000
8000 LET NP=7: LET L=26
8010 DIM O$(NP,L): DIM S(NP): DI
M N(NP)
8020 FOR N=1 TO NP
8030 READ O$(N),S(N),N(N)
8040 NEXT N
8050 RETURN
9000 DATA "ESTA' LIGADA A LUZ DA
 IGNICAO",0,0
9010 DATA "AS LUZES FUNCIONAM CO
RRECTAMENTE",3,4
9020 DATA "SUBSTITUIR O INTERRUP
TOR",0,0
9030 DATA "ENCONTRAM-SE AS LIGAC
OES DA BATERIA EM BOM ESTADO",5,
6
9040 DATA "RECARREGAR A BATERIA
OU EMPURRAR O CARRO",0,0
9050 DATA "REPARAR AS LIGACOES",
0,0
9060 DATA "-resto do programa-",
0,0
```

A passagem da rotina no programa é muito simples. Um indicador CP é usado para indicar a posição actual na tabela: para se principiar coloca-se a 1, sendo impresso o primeiro texto. Caso seja um indicador de fim S(CP)=0 (dificilmente seria por agora este o estado de posição condicional!), então CP é reinicializado a 1, de modo que a sequência recomece de novo. Por outro lado, se um indicador efectivamente real se encontrar presente, então é solicitada uma entrada (INPUT). Se a entrada for «S», então CP é posicionado no valor contido no elemento apropriado da tabela S(n); de outra maneira é colocado no valor contido na tabela N(n).

```
  20 LET CP=1
  30 PRINT O$(CP)
  40 IF S(CP)=0 THEN GO TO 20
  50 INPUT I$
  60 IF I$="S" THEN LET CP=S(CP)
: GO TO 30
  70 LET CP=N(CP)
  80 GO TO 30
```

Uma aproximação melhor

Uma alternativa para o método de ramificação sequencial (ou encadeamento sequencial) descrito acima é a aproximação paralela, a qual pergunta sempre por todas as possíveis questões antes de atingir a sua conclusão. Este método normalmente leva mais tempo a desenvolver do que seguir uma «estrutura em árvore» eficiente, sendo

porém mais provável que produza a resposta correcta, uma vez que não são omitidos nenhuns pontos concordantes.

Vamos considerar como podemos distinguir entre diversas formas de transporte.

Consideraremos oito características e marcaremos 1 ou 0 para a presença ou ausência respectivamente dessas características em cada um dos cinco modos de transporte (ver quadro 5.1). Se observar de perto aperceber-se-á de que o modelo dos resultados varia para cada uma das diferentes possibilidades, de forma que deve ser possível fazer a distinção entre elas por intermédio destas características.

QUADRO 5.1

Presença ou ausência das características

	bicicleta	carro	comboio	avião	cavalo
rodas	1	1	1	1	0
asas	0	0	0	1	0
motor	0	1	1	1	0
pneus	1	1	0	1	0
carris	0	0	1	0	0
vidros	0	1	1	1	0
corrente	1	0	0	0	0
condução	1	1	0	1	1

Entraremos com estes valores como DATA, lendo-os então (READ) e transferindo-os para uma tabela bidimensional F(n,n), a qual manterá uma cópia deste modelo, juntamente com uma tabela de cadeias contendo os nomes dos objectos OS(n).

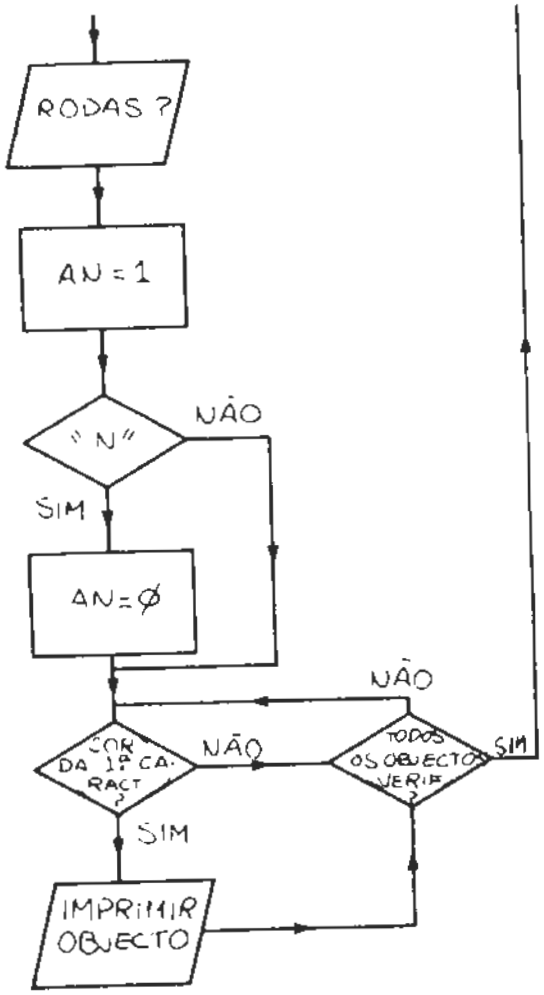
```
  10 GO SUB 8000
8000 DIM O$(5,9): DIM F(5,8)
8020 LET AP=0
9000 DATA "BICICLETA",1,0,0,1,0,
0,1,1
9010 DATA "CARRO",1,0,1,1,0,1,0,
1
9020 DATA "COMBOIO",1,0,1,0,1,1,
0,0
9030 DATA "AVIAO",1,1,1,1,0,1,0,
1
9040 DATA "CAVALO",0,0,0,0,0,0,0
,1
9050 FOR N=1 TO 5
```



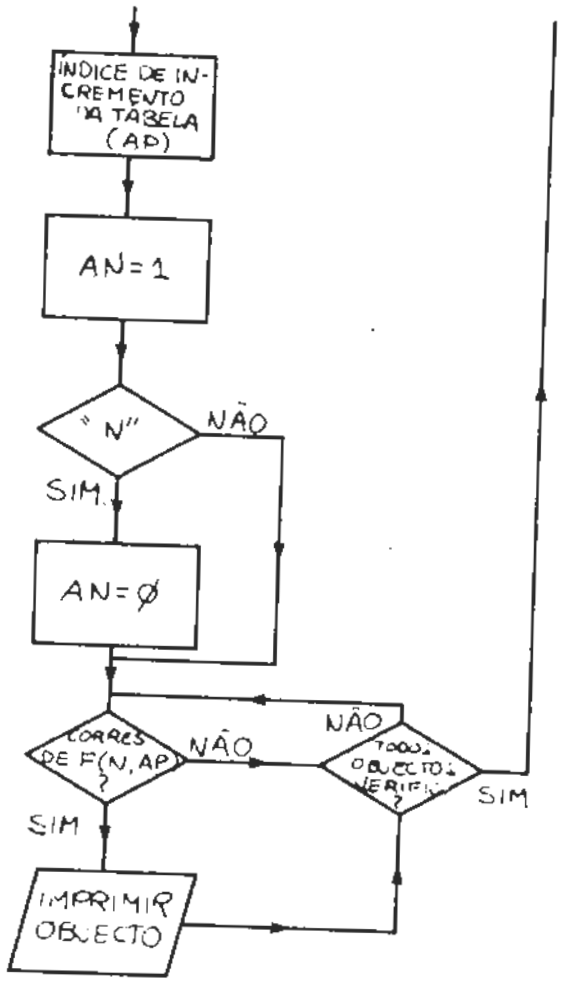
```
9060 READ O$(N)
9070 FOR M=1 TO 8
9080 READ F(N,M)
9090 NEXT M: NEXT N
```

Podemos começar agora a perguntar se a primeira característica se encontra ou não presente, utilizando a resposta para se imprimirem quais os modos de transporte que correspondem a este ponto particular (ver ordinograma 5.4).

```
100 PRINT "TEM RODAS"
500 INPUT I$
 510 LET AN=1: IF I$="N" THEN LE
T AN=0
 520 FOR N=1 TO 5
 530 IF F(N,1)=AN THEN PRINT O$(
N)
 540 NEXT N
```



Ordinograma 5.4 — *Uma aproximação paralela*



Ordinograma 5.5 — *Verificando as características de cada vez*

Neste caso, a resposta «S» produzirá uma impressão do seguinte:

BICICLETA
CARRO
COMBOIO
AVIÃO

e a resposta «N» produzirá uma impressão de unicamente o seguinte:

CAVALO

Isto demonstra claramente uma possível desvantagem do método paralelo, uma vez que, ainda que tenhamos justamente demonstrado que somente um cavalo não tem rodas, o programa insiste em que ainda perguntemos por todas as outras questões antes de se compro-

meter. Isto não é realmente assim tão estúpido como parece à primeira vista, já que se responder «S» à próxima questão («tem asas») reparará que o computador logicamente se recusa completamente a acreditar em cavalos alados.

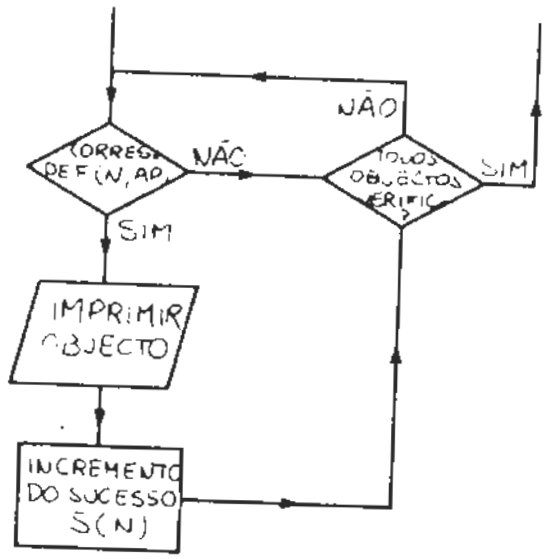
Se instalássemos a parte de comparação como uma sub-rotina podíamos usá-la para verificar todas as oito características de cada vez. Precisaríamos de proceder a ligeiras modificações, acrescentando um índice de tabela AP, o qual é incrementado para averiguar do próximo elemento da tabela de características F(N,AP) em cada ciclo executado (ver ordinograma 5.5).

```
100 PRINT "TEM RODAS"
110 GO SUB 500
120 PRINT "TEM ASAS"
130 GO SUB 500
140 PRINT "TEM UM MOTOR"
150 GO SUB 500
160 PRINT "TEM PNEUS"
170 GO SUB 500
180 PRINT "PRECISA DE CARRIS"
190 GO SUB 500
200 PRINT "TEM VIDROS"
210 GO SUB 500
220 PRINT "TEM UMA CORRENTE"
230 GO SUB 500
240 PRINT "É CONDUZIVEL"
250 GO SUB 500
400 PRINT
410 RUN
510 LET AP=AP+1: LET AN=1: IF I
$="N" THEN LET AN=0
530 IF F(N,AP)=AN THEN PRINT O$
(N)
560 RETURN
```

O máximo dos máximos

A rotina anterior imprimirá uma lista de correspondências para cada questão individual à medida que for processada, mas não nos dirá exactamente que conjunto de DATA é uma correspondência total para todas as hipóteses de resposta a todas as questões. Podemos criar um marcador de contagem (SCORE), que mostra com que exactidão as respostas correspondem com a DATA, possuindo para tal um

elemento da tabela de factor bem sucedido S(n) para cada objecto, o qual só é incrementado quando for encontrada uma correspondência F(N,AP) = AN (ver ordinograma 5.6).



Ordinograma 5.6 — *Avaliando a amplitude do sucesso*

```
260 PRINT
270 PRINT "SCORE"
280 PRINT
300 FOR N=1 TO 5
310 PRINT O$(N),S(N)
320 NEXT N
530 IF F(N,AP)=AN THEN PRINT O$
(N): LET S(N)=S(N)+1
8010 DIM S(5)
```

Se for encontrada uma correspondência completa, então S(n) ficará igual a 8. Onde um ou mais pontos foram detectados como incorrectos, o marcador de contagem (SCORE) será rebaixado no seu valor. Fazer a contagem por marcação desta maneira é particularmente útil quando as respostas às várias questões em consideração são mais uma questão de opinião do que factual (por exemplo: um cavalo é verdadeiramente conduzível?), uma vez que o valor mais alto na realidade obtido para a contagem provavelmente aponta sempre para a resposta correcta. (Repare-se que neste caso cada resposta certa tem um grau de avaliação idêntico.)

Melhor em bits

Pode ter notado que simplesmente aconteceu termos usado oito características para comparação, podendo ter-lhe ocorrido que esta escolha não tenha sido inteiramente acidental, visto existirem oito bits num byte. Se tormarmos em consideração cada característica como representando um dígito binário (ver quadro 5.2), em lugar dum valor absoluto, então cada objecto pode ser descrito por um único número decimal, o qual é a soma dos dígitos binários, em vez de o ser por oito valores distintos e separados entre si. Procederemos à conversão para decimal com o bit menos significativo pelo topo da tabela das características, de modo que, começando no cimo da lista por «rodas», cada característica, incluindo mesmo esta primeira, é equivalente aos valores 1, 2, 4, 8, 16, 32, 64, 128 em notação decimal.

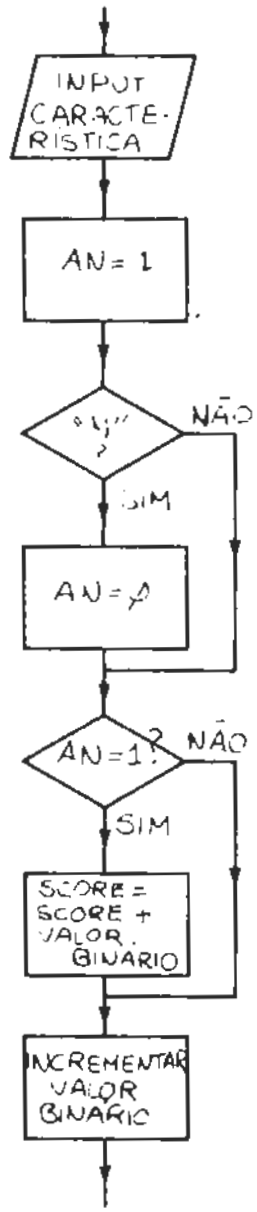
QUADRO 5.2

Características avaliadas em binário

	bicicleta	carro	comboio	avião	cavalo
rodas	1	1	1	1	0
asas	0	0	0	2	0
motor	0	4	4	4	0
pneus	8	8	0	8	0
carris	0	0	16	0	0
vidros	0	32	32	32	0
corrente	64	0	0	0	0
condução	128	128	0	128	128
soma total	201	173	53	175	128

Não se torna muito difícil converter a nossa marcação «score» de 1 a 8 para valores apropriados em binário, desde que nos lembremos de que o valor decimal para o dígito binário BV deve dobrar de cada vez que descemos na lista, e de que devemos somente juntar a «score» o valor binário actual, caso a resposta seja «sim» (AN = 1, ver ordinograma 5.7).

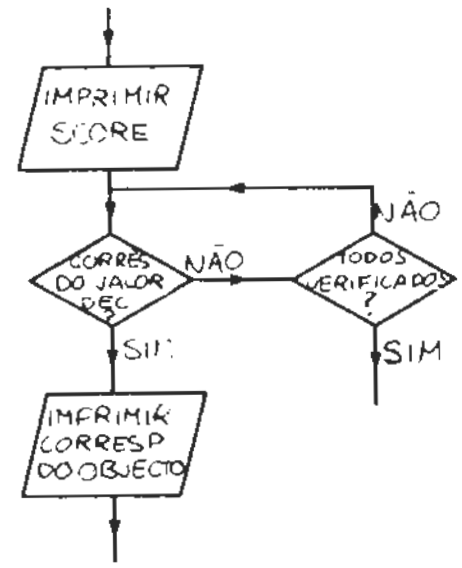
Se pensar por uns momentos, aperceber-se-á de que só necessitamos de nos manter ao corrente do número total produzido, S,



Ordinograma 5.7 — *Produzindo um «score» binário*

acrescentando-se por soma os valores binários das repostas «sim». Não há necessidade de saltar atrás e verificar cada parte dos conteúdos da tabela de cada vez, ou sequer possuir uma tabela bidimensional! Toda a DATA que precisamos de fazer entrar são os valores decimais globais para cada objecto, D(n), e quando todas as questões tiverem sido postas podemos verificá-las em confronto com o valor decimal obtido pela conversão binária das respostas «sim/não», SU (ver ordinograma 5.8). A coisa mais simples a fazer agora é eliminar tudo que se encontre para lá da linha 260 e começar a entrar outra vez!


```
 270 PRINT ,"SCORE";SU
 280 PRINT
 300 FOR N=1 TO 5
 310 IF D(N)=SU THEN PRINT ,O$(N
): GO TO 400
 320 NEXT N
 330 PRINT ,"OBJECTO NAO ENCONTR
ADO"
 400 PRINT
 410 INPUT I$: RUN
 500 INPUT I$
 510 LET AN=1: IF I$="N" THEN LE
T AN=0
 520 IF AN=1 THEN LET SU=SU+BV
 530 LET BV=BV+BV
 540 RETURN
8000 DIM O$(5,9): DIM D(5): LET
BV=1: LET SU=0
9000 DATA "BICICLETA",201
9010 DATA "CARRO",173
9020 DATA "COMBOIO",53
9030 DATA "AVIAO",175
9040 DATA "CAVALO",128
9050 FOR N=1 TO 5
9060 READ O$(N),D(N)
9070 NEXT N
9080 RETURN
```



Ordinograma 5.8 — *Fazendo corresponder o valor decimal*

Esta aproximação poupa obviamente uma grande porção de memória e de tempo, uma vez que cada elemento da tabela ocupa vários bytes e deve ser localizado antes de sér comparado, de forma que é particularmente útil quando se está a lidar com grandes quantidades de informação. Por outro lado, significa que se tem de calcular os decimais equivalentes de todos os modelos de bits antes de os poder utilizar, não lhe dando também quaisquer indícios quando não é encontrada uma correspondência completa. (Repare-se que não se pode simplesmente tomar em consideração aqui o valor decimal mais aproximado, visto que o valor equivalente decimal de cada resposta correcta depende da sua posição.)

Como é evidente, podia ter efectuado os cálculos pelo processo mais difícil e trabalhoso. Todavia, o *Spectrum* tem a função BIN, de forma que pode simplesmente solicitar-lhe uma impressão (PRINT) duma série de dígitos binários como um comando directo; por exemplo:

```
PRINT BIN 10101010
```

proporcionar-lhe-á o valor decimal 170. (Repare-se que se lê a lista da tabela de cima para baixo, de modo que o byte menos significativo seja lido primeiro.)

CAPÍTULO 6

Planificando o seu sistema inteligente para que aprenda por si próprio

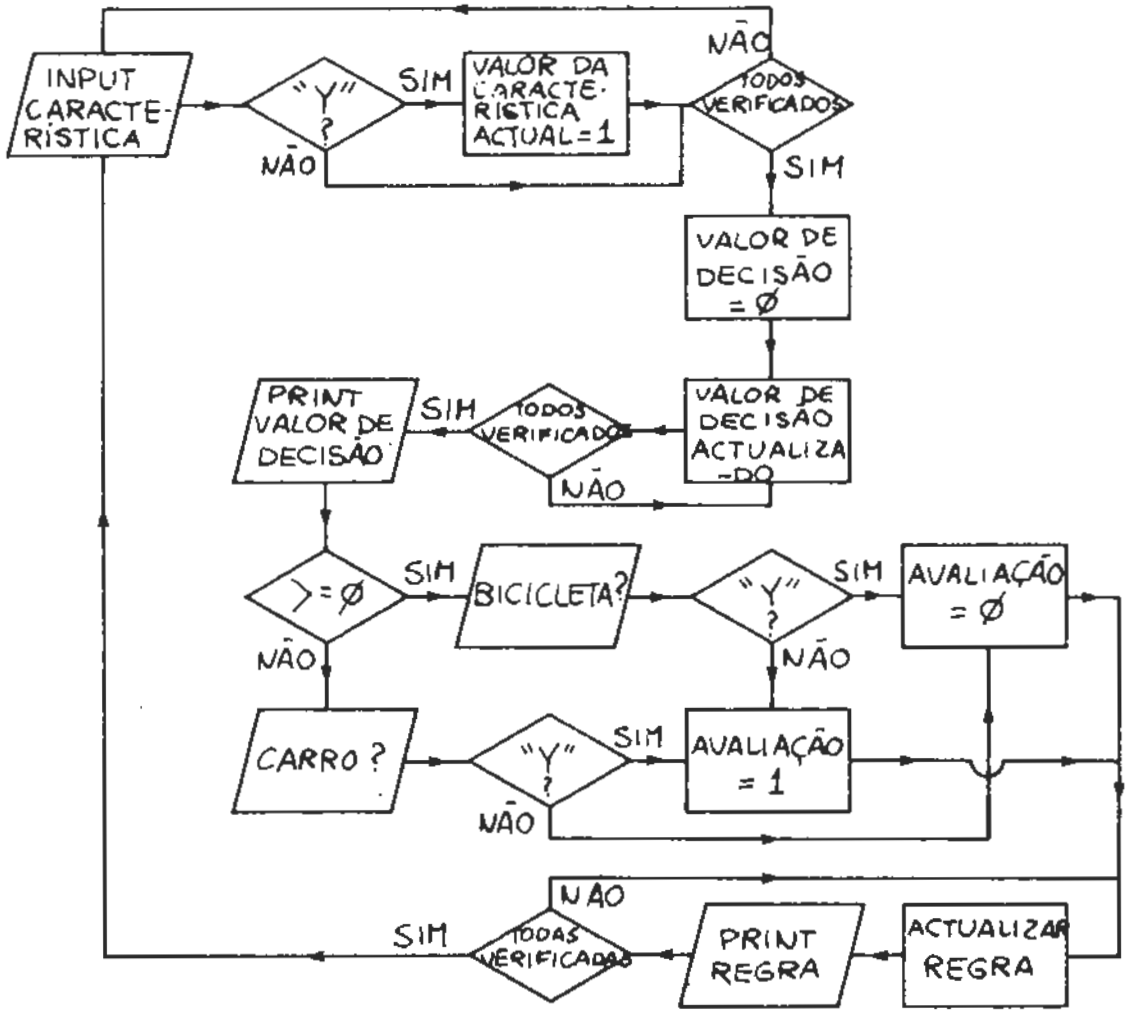
Embora os sistemas «inteligentes» descritos até ao momento funcionem bem, todos eles requerem que se lhes dê as regras correctas sobre que basearão antecipadamente as suas deeisões, o que pode ser bastante entediante.

Todavia, é possível construir um programa especializado que pode aprender a partir dos seus próprios erros e desenvolver as regras de decisão para si mesmo, contanto que se lhe possa dizer quando (ainda que não em que ponto, onde) está a correr mal. Isto é obviamente uma vantagem, caso você próprio não esteja completamente certo das regras correctas! Neste caso começamos com uma série de características que lhe devem permitir distinguir entre os diferentes objectos, mas sem qualquer modelo pré-definido sim/não destas características («regra de decisão») para nos orientar. Em vez disso utilizamos o próprio programa para calcular qual deva ser o modelo.

Trabalharemos com o nosso exemplo familiar de transporte, principiando por estabelecer algumas variáveis. FE é o número de características a serem consideradas (8), F$(n) é uma tabela contendo os nomes destas características, F(n) é uma tabela contendo os nomes destas características, F(n) é uma tabela que conterá os valores que conferirá a cada característica como entrada em qualquer ponto particular (0 ou 1), e R(n) é uma tabela que conterá os valores globais actuais da regra de decisão para cada característica.

```
  10 GO SUB 8000
8000 LET FE=8
8010 DIM F$(FE,8): DIM F(FE): DI
M R(FE)
8020 FOR N=1 TO FE
8030 READ F$(N)
8040 NEXT N
9000 DATA "RODAS","ASAS","MOTOR"
,"PNEUS","CARRIS","VIDROS","CORR
ENTE","CÓNDUCAO"
9010 RETURN
```

Cada característica é considerada de cada vez (ver ordinograma 6.1). Primeiro o valor da característica actual, F(n), sendo para tal o ciclo posicionado a zero, e de seguida é pedida ao usuário uma entrada (INPUT) I$ sim/não para cada ponto a considerar. Se I$ for «S», o elemento com o valor da característica F(N) é colocado em 1; de outra forma permanece posicionado a zero. Isto produzirá um modelo, o qual descreve o objecto como sendo «0» e «1» na tabela F(n).



Ordinograma 6.1 — *Aprendendo a distinguir entre dois objectos*


```
 60 FOR N=1 TO FE
 70 LET F(N)=0
 80 PRINT F$(N);" ";
 90 PAUSE 0
100 PRINT I$
110 IF I$="S" THEN LET F(N)=1
120 NEXT N
```

Agora uma variável de decisão DE é posicionada a zero. Isto é então recalculado como a soma do valor *actual* de DE, mais cada uma das características avaliadas nos seus valores F(n) entrados, multiplicando-se o resultado obtido pelos valores das regras de decisão actuais R(n).

```
125 LET DE=0
130 FOR N=1 TO FE
150 LET DE=DE+F(N)*R(N)
160 NEXT N
170 PRINT "DE= ";DE
```

Qual é qual?

Para começar consideremos a situação mais simples, onde só existem duas possibilidades — uma bicicleta ou um carro. Inicialmente fazemos a distinção entre estas bastante arbitrariamente, referindo-se que se o valor final de DE for igual ou maior do que 0 então é uma bicicleta, enquanto se DE for menor do que 0 então é um carro. Não importa realmente que isto não seja totalmente verdade, uma vez que o sistema será corrigido em breve. Quando o programa tiver procedido a uma decisão com base no valor de DE, solicita confirmação (ou o que quer que seja) do resultado.

```
 180 IF DE>=0 THEN PRINT "E UMA
BICICLETA  ";: INPUT I$: PRINT I
$: GO TO 200
 190 IF DE<0 THEN PRINT "E UM CA
RRO  ";: INPUT I$: PRINT I$: GO
TO 220
```

Três possíveis direcções de acção podem ser tomadas de acordo com o facto de a decisão do computador ser ou não correcta. Caso tenha sido correcta, então nenhuma acção é efectivamente tomada (uma variável de avaliação WT é posicionada a zero), e o programa

salta atrás por ciclo para mais uma tentativa. Se DE foi >=0 mas o computador estiver errado, então a variável de avaliação WT é posicionada a menos um; enquanto se DE for <0 mas o computador estiver errado, então WT é posicionado a mais um.

```
200 IF I$="S" THEN LET WT=0: GO
TO 240
210 LET WT=-1: GO TO 240
220 IF I$="S" THEN LET WT=0: GO
TO 240
230 LET WT=1
```

O efeito da variável de avaliação é modificar os valores na tabela de regras R(N), baixando-lhes o valor quando forem demasiado elevados, e subindo-os quando forem muitos reduzidos.

```
240 FOR N=1 TO FE
250 LET R(N)=R(N)+F(N)*WT
260 PRINT R(N),
270 NEXT N
280 PRINT : PRINT
290 GO TO 60
```

O processo de trabalho do sistema é melhor observado por intermédio duma demonstração. Escreva RUN e depois esta sequência de entradas. (Repare que a pontuação foi desenhada para conferir um formato de *écran* que indica claramente a relação entre a sua entrada de valores e os valores de regras de decisão.)

Primeiro que tudo entrar com estes valores:

```
RODAS S        ASAS N
MOTOR N        PNEUS S
CARRIS N       VIDROS N
CORRENTE S     CONDUÇÃO S
```

O programa retornará com um valor de decisão DE de zero, visto ser este o valor inicial, não tendo ocorrido qualquer modificação:

```
DE=0
```

Uma vez que DE é 0, o sistema assume que este é uma bicicleta, pedindo uma confirmação, para a qual a resposta é, como é óbvio, «sim».


```
É UMA BICICLETA    S
```

A impressão dos conteúdos da tabela de regras R(n) mostra que estas não mudaram de zero, visto a resposta correcta ter sido, por pura sorte, obtida.

```
0              0
0              0
0              0
0              0
```

Experimente agora entrar com esta sequência, a qual descreve um carro:

```
RODAS S        ASAS N
MOTOR S        PNEUS S
CARRIS N       VIDROS S
CORRENTE N     CONDUÇÃO S
```

DE ainda se encontra a zero, de forma que a conclusão errada é alcançada, sendo posta a pergunta errada, para a qual a resposta deve ser «N».

```
DE=0
É UMA BICICLETA    N
```

Agora, que um erro foi cometido, a regra de decisão é modificada, subtraindo-se um de cada valor contido na tabela de regras onde foi dada uma resposta «sim». Os conteúdos da tabela de regras tornam-se, pois, assim, no seguinte:

```
-1             0
-1            -1
 0            -1
 0            -1
```

Se entrar uma vez mais com os valores que descrevem um carro, o programa aparecerá com a resposta correcta:

```
RODAS S        ASAS N
MOTOR S        PNEUS S
```

```
CARRIS N        VIDROS S
CORRENTE N      CONDUÇÃO S
DE=-5
É UM CARRO       S
-1                0
-1               -1
 0               -1
 0               -1
```

Antes de se sentir demasiado satisfeito consigo próprio, experimente dar-lhe outra vez os valores para uma bicicleta, que ele tomará incorrectamente!

```
RODAS S         ASAS N
MOTOR N         PNEUS S
CARRIS N        VIDROS N
CORRENTE S      CONDUÇÃO S
DE=-3
É UM CARRO       N
 0                0
-1                0
 0               -1
 1                0
```

Contudo, as características positivas que são comuns à bicicleta e ao carro são agora automaticamente aumentadas numa unidade, de maneira que se repetir esta última sequência produzirá então a conclusão correcta.

```
RODAS S         ASAS N
MOTOR N         PNEUS S
CARRIS N        VIDROS N
CORRENTE S      CONDUÇÃO S
DE=1
É UMA BICICLETA    S
 0                 0
-1                 0
 0                -1
 1                 0
```

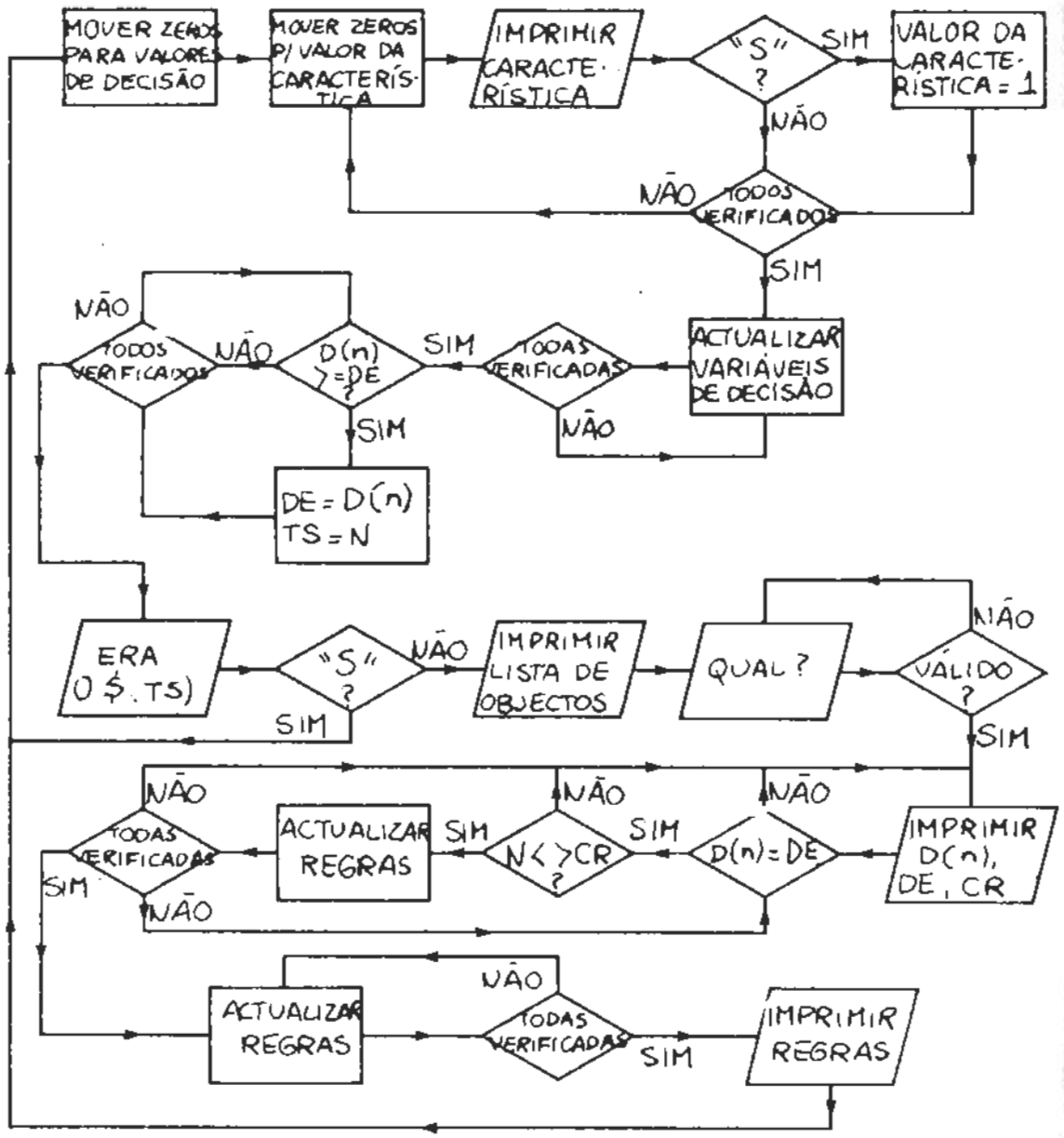

A situação encontra-se agora estabilizada e o programa reconhecerá sempre correctamente tanto o carro como a bicicleta, de cada vez que se entrar com as características que os definem:

```
RODAS S         ASAS N
MOTOR S         PNEUS S
CARRIS N        VIDROS S
CORRENTE N      CONDUÇÃO S
DE=-2
É UM CARRO       S
 0                0
-1                0
 0               -1
 1                0
```

Repare que o valor final de DE para uma bicicleta é 1, e que para um carro é −2. Se consultar os valores da tabela de regras, observará que estes valores correspondem, tanto no número como na posição, às características únicas que distinguem estes objectos (CORRENTE para bicicleta, e MOTOR e VIDROS para carro).

Um «Spectrum» mais expansível

Ainda que tenhamos até agora ministrado alguma coisa ao seu computador, não deixa de ser fastidioso ser-se capaz de distinguir somente entre dois objectos. Vamos expandir o sistema para se poder lidar com um espectro *(spectrum)* mais amplo de possibilidades (ver ordinograma 6.2). Para começar precisamos de definir o número de objectos que desejamos ser capazes de reconhecer, OB, denominá-los como DATA que lemos (READ) para uma nova tabela, O$(OB), mudar a nossa tabela de regras de decisão para um formato bidimensional, (R(FE,OB), o qual pode comportar regras para cada um dos objectos separadamente, e instalar uma tabela de decisões, D(n), para suportar valores para cada objecto.



Ordinograma 6.2 — *Aprendendo as regras para um «Spectrum» mais expansível nas suas possibilidades*

```
  10 GO SUB 8000
8000 LET FE=8: LET OB=5
8010 DIM F$(FE,8): DIM F(FE): DI
M R(FE,OB): DIM O$(OB,7): DIM D(
OB)
8020 FOR N=1 TO FE
8030 READ F$(N)
8040 NEXT N
8050 FOR N=1 TO OB
8060 READ O$(N)
```



```
8070 NEXT N
9000 DATA "RODAS","ASAS","MOTOR"
,"PNEUS","CARRIS","VIDROS","CORR
ENTE","CONDUCAO"
9010 DATA "BICICLETA","CARRO","C
OMBOIO","AVIAO","CAVALO"
9999 RETURN
```

Em vez de termos somente uma única variável de decisão DE, precisamos de determinar neste momento um valor de decisão para cada objecto de cada vez. Por cada ciclo devemos primeiro posicionar DE a zero, e de seguida movimentar zero para cada elemento na tabela de decisões D(n), de forma a começarmos com um quadro limpo para cada objecto.

```
20 LET DE=0
30 FOR N=1 TO OB
40 LET D(N)=0
50 NEXT N
```

Os valores para cada característica são então entrados exactamente da mesma maneira que anteriormente.

```
 60 FOR N=1 TO FE
 70 LET F(N)=0
 80 PRINT F$(N);" ";
 90 INPUT I$
100 PRINT I$
110 IF I$="S" THEN LET F(N)=1
120 NEXT N
```

Cada elemento da tabela de decisões D(n) é agora actualizado de acordo com o estabelecido para os valores entados F(n) e os conteúdos do elemento apropriado da tabela de regras R(n,m).

```
130 FOR N=1 TO FE
140 FOR M=1 TO OB
150 LET D(M)=D(M)+F(N)*R(N
160 NEXT M: NEXT N
```

Precisamos agora de observar se qualquer dos valores de decisão para qualquer dos objectos D(n) é maior do que ou igual ao valor global da decisão DE. Se tal for o caso, posicionamos uma variável TS dum «marcador de cabeça» igual ao número do objecto que produz a melhor correspondência, N.

```
170 FOR N=1 TO OB
180 IF D(N)>=DE THEN LET DE=D(N
); LET TS=N
190 NEXT N
```

O melhor palpite do sistema é que esta é a resposta correcta, portanto uma vez mais solicita a confirmação respectiva, retornando simplesmente para uma nova entrada sem proceder a quaisquer alterações caso a resposta fosse correcta.

```
200 PRINT "ERA ";O$(TS);
210 INPUT I$: PRINT I$
220 IF I$="S" THEN GO TO 20
```

Caso esta não tenha sido a resposta correcta, os nomes e os números de todos os objectos são impressos, sendo-se solicitado para o número da resposta correcta CR. (As limitações em CR evitam que se destrua o programa ao entrar com um valor ilegal.)

```
230 FOR N=1 TO OB
240 PRINT N;O$(N)
250 NEXT N: PRINT
260 PRINT "QUAL ERA ";
270 INPUT CR: IF CR<1 OR CR>5 T
HEN GO TO 270
275 PRINT CR
```

É agora executada uma verificação tanto para quando o valor de decisão para cada objecto D(n) é maior do que como para quando é igual ao valor de decisão global DE e para averiguar se o objecto a ser considerado é ou não a resposta correcta. Se ambas estas condições são verdadeiras, as regras são actualizadas outra vez, subtraindo-se os valores correctos das características F(n), a favor da resposta correcta.

```
280 PRINT TAB (8);"D(N)    DE
CR": FOR N=1 TO OB
290 PRINT TAB (9);D(N);"
";DE;"      ";CR
300 IF D(N)>=DE AND N<>CR THEN
FOR M=1 TO FE: LET R(M,N)=R(M,N)
-F(M): NEXT M
310 NEXT N
```

Agora os valores correctos das características F(n) são acrescentados à tabela de regras para o objecto correcto, na direcção oposta.


```
320 FOR M=1 TO FE
330 LET R(M,CR)=R(M,CR)+F(M)
340 NEXT M: PRINT
```

Finalmente, as categorias das tabelas de regras são impressas, de modo a poder-se observar aquilo que se vai passando.

```
350 FOR M=1 TO OB
360 FOR N=1 TO FE
370 PRINT TAB (N*3)-3;R(N,M);
380 NEXT N
390 PRINT " ";O$(M)
400 NEXT M
410 INPUT Q$: CLS : GO TO 20
```

Uma vez mais uma demonstração é a melhor maneira de que se dispõe para se perceber o que vai acontecendo, de forma que se entra com a seguinte sequência:

```
RODAS S        ASAS N
MOTOR N        PNEUS S
CARRIS N       VIDROS N
CORRENTE S     CONDUÇÃO S
```

O programa dará em retorno a conclusão errónea de que se tratava dum cavalo, de forma que se deve comunicar-lhe que estava errado no seu parecer, quando lhe perguntar pela resposta correcta (bicicleta = 1):

```
ERA UM CAVALO  N
1 BICICLETA        2 CARRO
3 COMBOIO          4 AVIÃO
5 CAVALO
QUAL ERA  1
```

As categorias das várias decisões e regras das tabelas correspondentes são aqui impressas para sua informação.

D(N)	DE	CR
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1

1	0	0	1	0	0	1	1	bicicleta
−1	0	0	−1	0	0	−1	−1	carro
−1	0	0	−1	0	0	−1	−1	comboio
−1	0	0	−1	0	0	−1	−1	avião
−1	0	0	−1	0	0	−1	−1	cavalo
A	B	C	D	E	F	G	H	

(A = rodas B = asas C = motor D = pneus
E = carris F = vidros G = corrente H = condução)

Se observar de perto, verificará que as características que causaram alterações nas tabelas de regras são rodas, corrente e condução — as quais são todas características que definimos como sendo parte duma bicicleta, mas que não são encontradas num cavalo. Adicionalmente verá que os valores para estas características, na tabela de regras para o caso da bicicleta, se encontram agora todos a mais uma unidade, ao passo que os valores para estas características para os outros objectos estão agora todos a menos uma unidade.

Proporcione-lhe agora as características dum carro, que ele pensa ser uma bicicleta, corrigindo-o depois. Repare que as tabelas de regras para uma bicicleta e um carro são agora corrigidas para se tomar em consideração a nova informação.

```
RODAS S                ASAS N
MOTOR S                PNEUS S
CARRIS N               VIDROS S
CORRENTE N             CONDUÇÃO S
ERA UMA BICICLETA      N
1 BICICLETA            2 CARRO
3 COMBOIO              4 AVIÃO
5 CAVALO
QUAL ERA    2
```


D(N)	DE	CR
3	3	2
−3	3	2
−3	3	2
−3	3	2
−3	3	2

0	0	−1	0	0	−1	1	0	bicicleta
0	0	1	0	0	1	−1	0	carro
−1	0	0	−1	0	0	−1	−1	comboio
−1	0	0	−1	0	0	−1	−1	avião
−1	0	0	−1	0	0	−1	−1	cavalo
A	B	C	D	E	F	G	H	

De seguida dê-lhe um avião, o qual ele decide ser um carro, e corrija-o outra vez.

```
RODAS S                ASAS S
MOTOR S                PNEUS S
CARRIS N               VIDROS S
CORRENTE N             CONDUÇÃO S
ERA UM CARRO           N
1 BICICLETA            2 CARRO
3 COMBOIO              4 AVIÃO
5 CAVALO
QUAL ERA 4
```

E agora um comboio, para o qual ele ainda faz a escolha incorrecta!

```
RODAS S                ASAS N
MOTOR S                PNEUS N
CARRIS S               VIDROS S
CORRENTE N             CONDUÇÃO N
ERA UM AVIÃO           N
1 BICICLETA            2 CARRO
3 COMBOIO              4 AVIÃO
5 CAVALO
QUAL ERA    3
```

E finalmente um cavalo, o qual é dado como um avião!

```
RODAS N              ASAS N
MOTOR N              PNEUS N
CARRIS N             VIDROS N
CORRENTE N           CONDUÇÃO S
ERA UM AVIÃO         N
1 BICICLETA          2 CARRO
3 COMBOIO            4 AVIÃO
5 CAVALO
QUAL ERA   5
```

Se continuar a alimentar o seu «perito em informação», ele dar-lhe-á eventualmente a resposta certa todas as vezes. O tempo necessário para este processo depende da extensão das diferenças existentes entre as características dos objectos, e da ordem pela qual os objectos são apresentados ao «perito». Fique de sobreaviso acerca do facto de poder levar bastante tempo antes de se tornar infalível! Eis aqui uma sequência que eventualmente veio a revelar-se correcta todas as vezes.

avião(comboio)	carro(avião)	bicicleta(SIM)
carro(SIM)	avião(carro)	avião(SIM)
cavalo(SIM)	avião(bicicleta)	carro(avião)
avião(carro)	avião(carro)	carro(avião)
carro(SIM)	avião(carro)	avião(SIM)
carro(SIM)	avião(SIM)	cavalo(SIM)
bicicleta(SIM)	comboio(carro)	comboio(SIM)
bicicleta(SIM)	carro(avião)	carro(SIM)
avião(carro)	avião(SIM)	carro(avião)
carro(SIM)	avião(SIM)	carro(SIM)
bicicleta(carro)	carro(SIM)	avião(SIM)
comboio(SIM)	cavalo(SIM)	bicicleta(SIM)

Para se observar o estado final da tabela de regras quando o último estado é atingido, pode-se parar o programa, escrevendo-se então «GO TO 350» como um comando directo. Como a escala final de valores varia de +6 a −2, não deve ficar surpreendido pelo facto de ter levado bastante tempo até chegar a esse ponto.

1	0	−1	1	0	−2	3	0	(bicicleta)
−1	4	1	0	−1	1	−2	0	(carro)
0	−1	1	−2	2	1	−1	−2	(comboio)
−2	6	0	−1	−1	0	−2	−2	(avião)
−1	0	0	−1	0	0	−1	0	(cavalo)
A	B	C	D	E	F	G	H	

(A = rodas B = asas C = motor D = pneus
E = carris F = vidros G = corrente H = condução)

Como é evidente, numa aplicação real de um tal sistema especializado poder-se-ia alimentá-lo com uma massa de informação colectada numa determinada área de assuntos, e as conclusões, automaticamente, e então deixá-lo sozinho para «digerir» isto e estabelecer as regras na sua devida e boa altura. Uma vez que estas regras são armazenadas em tabelas, poder-se-ia facilmente escrever uma rotina para as guardar a salvo para reutilização posterior.

CAPÍTULO 7

Combinação mesclada

Os computadores são totalmente lógicos, porém a nossa própria memória na sua amplitude é muito menos fidedigna, o que pode levar a problemas quando se está a experimentar recolher informação sobre um determinado assunto. Por exemplo, o inglês é uma língua muito variável, existindo frequentemente pronúncias alternativas para os mesmos (ou bastante parecidos) apelidos, o que pode ocasionar dificuldades.

Uma maneira de rodear este problema é tentar fazer corresponder o som da palavra, ao contrário das letras realmente contidas nele, por meio da codificação *Soundex*, a qual foi originalmente desenvolvida para auxiliar no processamento do censo de 1890 nos EUA. Este método de codificação assegura que palavras que soem semelhantemente tenham quase a mesma sequência de código.

As regras para a codificação duma palavra são as seguintes:

1) Conservar sempre a primeira letra da palavra como o primeiro carácter do código.

A partir da segunda letra:

2) Ignorar vogais (a, e, i, o, u).
3) Ignorar as letras w, y, q e h.

4) Ignorar os sinais de pontuação.
5) Codificar as outras letras com os valores 1-6 como se segue:

Letras	Código
bfpv	1
cgjksxz	2
dt	3
l	4
mn	5
r	6

6) Onde quer que letras adjacentes tenham o mesmo código, somente a primeira é retida.
7) Se o comprimento do código for maior do que quatro caracteres, considere então somente os primeiros quatro.
8) Se o comprimento do código for menor do que quatro caracteres, preencha então com zeros o espaço restante até quatro caracteres.

Para esclarecer as coisas neste ponto, eis alguns exemplos de nomes codificados por *Soundex:*

BRAIN — B650

(B é retido, R é 6, A e são rebaixados, N é 5 e o zero é acrescentado para completar o código.)

CUNNINGHAM — C552

(C é retido, U é rebaixado, ambos os N são representados pelo único código 5, I é rebaixado, o terceiro N é representado por 5, G é 2, H e A são rebaixados, e M é codificado como sendo 5 — mas o código resultante (C5525 é truncado para ter só quatro caracteres.)

SANGUE — S520

(S é retido, A é rebaixado, N é 5, G é 2, U e E são ambas rebaixadas, acrescentando-se um zero para preencher o espaço restante do código.)

IRLANDA — I645

(I é retido, R é 6, L é 4, A é rebaixado, N é 5, D é 3 — mas o código resultante, I6453, é truncado para ter só quatro caracteres.)

ESCOCÊS — E222

(E é retido, S é 2, C é rebaixado por se encontrar no mesmo grupo que S, O é rebaixado, C é 2, E é rebaixado, e S é 2. Nestas duas últimas consoantes, apesar de tomarem o mesmo valor, a segunda não é rebaixada por se encontrar uma vogal entre as duas consoantes, que não são, assim, adjacentes.)

Se o seu nome tiver bastantes vogais e outras letras rejeitadas, descobrirá então que o seu código é de algum modo abreviado!

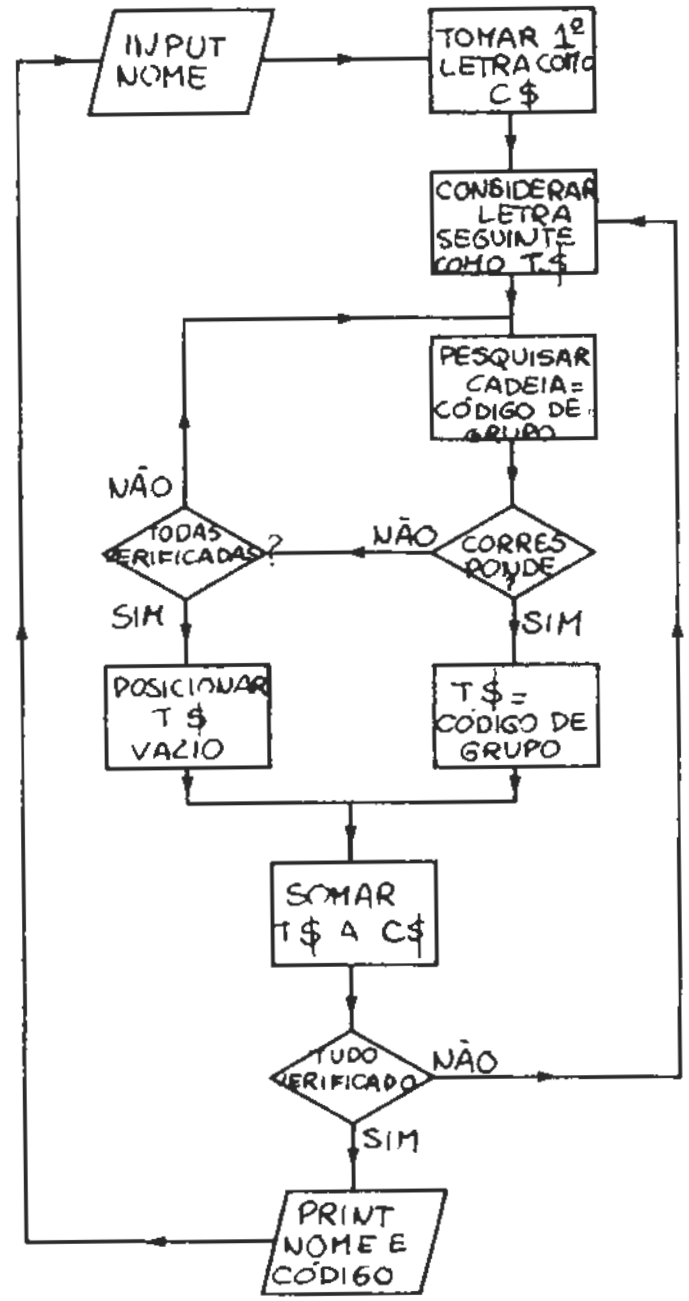
HEYHOE — H000

(H é retido, todas as outras letras são rejeitadas (!), sendo o código preenchido com zeros.)

Rotina de codificação

Para se poupar a todo esse trabalho cerebral vamos desenvolver um programa que lhe permite fazer entrar uma palavra em português, emitindo-a para a saída em código *Soundex* (ver ordinograma 7.1). A primeira coisa a fazer é instalar algumas declarações de DATA contendo as letras retidas nos seus grupos apropriados. (Repare que os grupos são arranjados por ordem crescente dos seus valores de código.)

```
9000 DATA "BFPV","CGJKSXZ","DT"
"L","MN","R"
```

Ordinograma 7.1 — *Produzindo um código -Soundex-*

Podemos agora fazer entrar a palavra a converter, **I$**, e, para começar faz-se a versão codificada disto, **C$**, com a primeira letra dessa palavra (seguindo-se a primeira regra acima).

```
100 RESTORE : INPUT I$
110 LET C$=I$(1)
```

Precisamos agora de verificar as outras letras da palavra, 2 TO LEN(I$), cada uma por sua vez, isto depois de primeiro se criar uma cadeia temporária T$ igual à letra correntemente a ser transferida.

```
120 FOR N=2 TO LEN I$: RESTORE
130 LET T$=I$(N)
```

Uma vez que a conversão para os números de código será pedida em várias ocasiões no problema final, instalaremos este processo como uma sub-rotina na linha 1000.

```
140 GO SUB 1000
```

Temos de verificar T$ em confronto com cada letra individualizada em cada grupo de letras para se encontrar uma correspondência. Para verificar cada grupo de letras precisamos de percorrer o mesmo ciclo seis vezes, fazendo duma cadeia de pesquisa S$ o grupo de código *Soundex* actual, e utilizando uma rotina INSTR que verifica cada letra no grupo em confronto com T$ de todas as vezes.

```
1000 FOR P=1 TO 6
1010 READ S$
1020 GO SUB 5000
```

A rotina INSTR é semelhante à usada em capítulos anteriores.

```
5000 FOR M=1 TO LEN S$
5010 IF S$(M)=T$ THEN LET SP=M:
RETURN
5020 NEXT M
5030 LET SP=0
5040 RETURN
```

Quando a verificação INSTR foi executada, temos de determinar se foi ou não encontrada uma correspondência para qualquer dos grupos *Soundex*, e se assim for, para qual dos grupos. Se nenhuma correspondência for encontrada, então SP será posicionado a zero. Se tiver sido encontrada uma correspondência, então SP será posicionado para M, o qual indicará o valor do grupo de código correspondido.

Se for encontrada uma correspondência (SP>O), convertemos o valor do ciclo que percorre os grupos de código P numa cadeia T$, que substitui a nossa cadeia temporária original.

```
1030 IF SP>0 THEN LET T$=STR$ (P
): RETURN
```


Se nenhuma correspondência for encontrada nesse grupo, temos de verificar o próximo grupo.

```
1040 NEXT P
```

Se de todo em todo nenhuma correspondência for encontrada, T$ deve conter um dos caracteres a ignorar. Portanto, reposicionamos T$ a limpo (T$=″″) e RETURN.

```
1050 LET T$=""
1060 RETURN
```

Podemos agora tornar a cadeia codificada C$ igual à cadeia original mais o novo carácter convertido T$.

```
170 LET C$=C$+T$
```

Voltamos agora atrás no ciclo para lidar eom o próximo carácter em I$.

```
180 NEXT N
```

Quando o fim de IS for alcançado, imprimimos a entrada, I$, e a cadeia codificada C$ por inteiro, antes de se voltar atrás, à linha 100, para outra entrada.

```
 210 PRINT : PRINT "NOME","CODIG
O": PRINT I$,C$
 320 GO TO 100
```

Se entrarmos com BRAIN obter-se-á o código B650, que está correcto. Por outro lado, se tentar o nome ESTÊVÃO ou a palavra CUNNINGHAM obter-se-ão agora os códigos S310 e C55525, respectivamente. O código para ESTÊVÃO é demasiado curto, precisando, pois, de ser preenchido com zeros, e o código para CUNNINGHAM é demasiado longo, sendo os mesmos códigos repetidos um a seguir ao outro para a letra N.

Lidando com pormenores

Para resolver o problema da repetição do mesmo código para letras adjacentes, precisamos de manter um registo da última cadeia temporária, L$. Precisamos de tornar o código de L$ o primeiro carácter em I$ para começar, de forma que a letra inicial não seja repetida. À medida que prosseguimos com o ciclo FOR-NEXT, devemos comparar L$ com T$, e se ambas as cadeias forem o mesmo, não devemos adicionar T$ a C$. Além disso, precisamos de fazer de L$ o mais recente T$.

```
110 LET T$=I$(1): LET C$=T$: GO
SUB 1000: LET L$=T$
150 IF T$=L$ THEN GO TO 180
160 LET L$=T$
```

Podemos agora solucionar o problema de o código ser demasiado curto. Antes de mais verificamos o comprimento da cadeia, LEN(C$)<4. Se for demasiado curta acrescentamos três zeros no fim, recortando então a cadeia para o tamanho correcto (quatro caracteres).

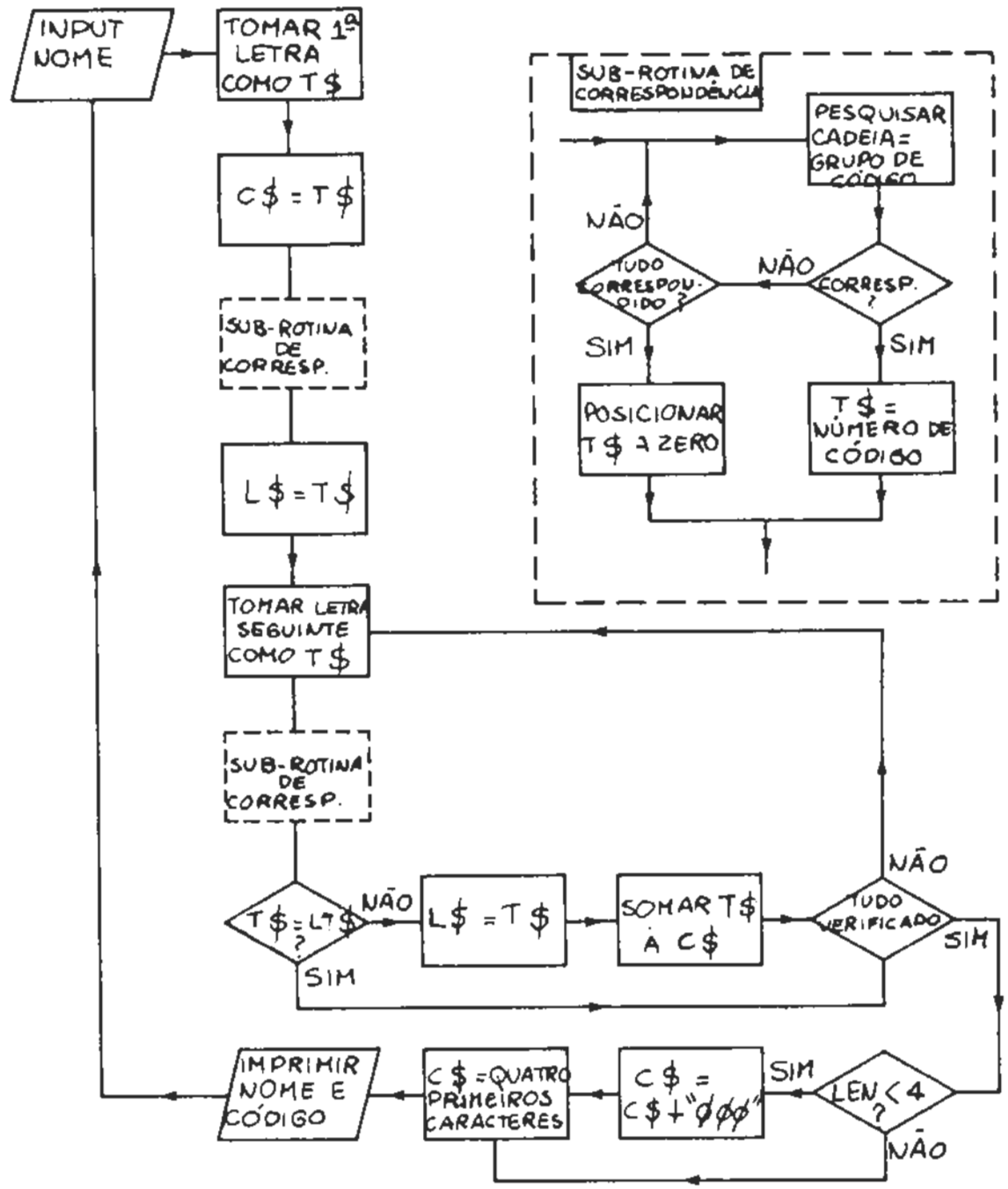
```
 190 IF LEN C$(4 THEN LET C$=C$+
"000": LET C$=C$( TO 4)
```

Finalmente, se a cadeia for demasiado longa recortamos então de novo para o tamanho certo.

```
200 IF LEN C$>4 THEN LET C$=C$(
TO 4)
```

Construção da correspondência

Agora que possuímos um método fidedigno na produção de códigos *Soundex*, vamos dar-lhe qualquer coisa para trabalhar. A primeira tarefa é elaborar uma lista de nomes para declarações de DATA, e lê-las (READ) para uma tabela de cadeias de nomes, N$(n). A nossa lista de demonstração consiste somente de dezoito nomes — se quiser mais,



Ordinograma 7.2 — *Lidando com pormenores*

uma rápida consulta à lista telefónica deverá resolver-lhe rapidamente esse problema! Note que o número de palavras é igualmente armazenado como NW.

```
  10 GO SUB 8000
8000 LET NW=18
8010 DIM N$(NW,9)
9010 DATA "ABRAHAM","ABRAHAMS","
ABRAMS","ADAM","ADAMS","ADDAMS",
"ADAMSON","ALAN","ALLAN","ALLEN"
9020 DATA "ANTHANY","ANTHONY","A
NTONY","ANTROBUS","APPERLEY","AP
PLEBEE","APPLEBY","APPLEFORD"
9030 RESTORE 9010: FOR N=1 TO NW
9040 READ N$(N)
9050 NEXT N
```

A ideia global de fazer corresponder códigos *Soundex* reside no facto de se usar o código *Soundex* para proceder à correspondência antes de imprimir as possíveis palavras. Temos por conseguinte de encontrar os códigos para cada um dos nomes da DATA e colocar estes códigos numa tabela de cadeias equivalente, O$(n). A rotina para descobrir o código *Soundex* é virtualmente idêntica à usada para encontrar o código duma entrada, tal como foi descrito acima.

```
9500 DIM O$(NW,4)
3510 PRINT : PRINT "NOME","CODIG
O": PRINT
9520 FOR Q=1 TO NW
9530 PRINT N$(Q),
9540 LET T$=N$(Q)( TO 1): LET C$
=T$: RESTORE : GO SUB 1000: LET
L$=T$
9550 FOR N=2 TO LEN (N$(Q))
9560 LET T$=N$(Q)(N)
9570 RESTORE : GO SUB 1000
9580 IF T$=L$ THEN NEXT N: GO TO
 9620
9590 LET L$=T$
9600 LET C$=C$+T$
9610 NEXT N
9620 IF LEN C$<4 THEN LET C$=C$+
"000": LET C$=C$( TO 4)
9630 IF LEN C$>4 THEN LET C$=C$(
 TO 4)
9640 PRINT C$
9650 LET O$(Q)=C$
9660 NEXT Q
9670 RETURN
```


Se correr isto agora (RUN), observará que todos os códigos para a DATA são produzidos antes do pedido de entrada.

NOME	CÓDIGO
ABRAHAM	A165
ABRAHAMS	A165
ABRAMS	A165
ADAM	A350
ADAMS	A352
ADDAMS	A352
ADAMSON	A352
ALAN	A450
ALLAN	A450
ALLEN	A450
ANTHANY	A535
ANTHONY	A535
ANTONY	A535
ANTROBUS	A536
APPERLEY	A164
APPLEBEE	A141
APPLEBY	A141
APPLEFORD	A141

A única coisa que precisamos de fazer é descobrir quais os códigos destes nomes que se combinam por correspondência com o código da sua entrada, imprimindo-se então estes nomes com um ciclo FOR-NEXT (PARA-PRÓXIMO).

```
 240 PRINT
 250 FOR N=1 TO NW
 260 IF C$=O$(N) THEN PRINT N$(N
),O$(N)
 270 NEXT N
```

Isto imprimirá apenas palavras com códigos *Soundex* exactamente correspondentes. Por exemplo, se experimentar entrar o nome APPLEBE obterá a seguinte resposta:

```
?APPLEBE

NOME          CÓDIGO
APPLEBE       A141
```

APPLEBEE A141
APPLEBY A141
APPLEFORD A141

Ainda que APPLEBE (um E no fim!) não se encontre presente na DATA, encontrámos APPLEBEE e APPLEBY, assim como APPLEFORD (cuja interessante sonorização no fim foi cortada).

Correspondência parcial

Note que, por outro lado, APPERLEY foi rejeitada, ainda que soe de modo semelhante a princípio. Seria por conseguinte útil se pudéssemos igualmente imprimir correspondências parciais.

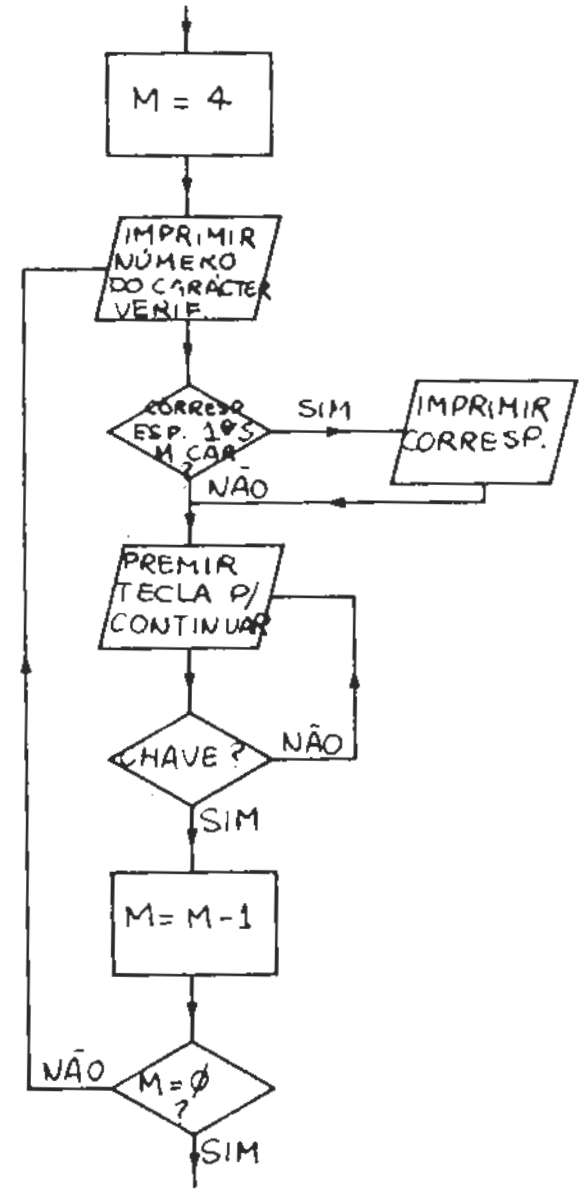
Isto pode ser facilmente executado acrescentando um ciclo extra FOR-NEXT, o qual compara uma secção decrescente da entrada com comprimentos decrescentes dos códigos armazenados (ver ordinograma 7.3).

```
 230 FOR M=4 TO 1 STEP -1
 240 PRINT : PRINT M;" CARACTERE
S CORRESPONDEM": PRINT
 260 IF C$( TO M)=O$(N)( TO M) T
HEN PRINT N$(N),O$(N)
 280 PRINT : PRINT "PREMIR TECLA
 PARA CONTINUAR"
 290 PAUSE 0
 300 PRINT : PRINT
 310 NEXT M
```

Se experimentarmos agora APPLEBE, pode-se observar toda a extensão de possibilidades.

?APPLEBE

NOME CÓDIGO
APPLEBE A141



Ordinograma 7.3 — *Correspondência parcial*

4 CARACTERES CORRESPONDEM
APPLEBEE A141
APPLEBY A141
APPLEFORD A141
PREMIR TECLA PARA CONTINUAR

3 CARACTERES CORRESPONDEM
APPLEBEE A141

```
APPLEBY      A141
APPLEFORD    A141
PREMIR TECLA PARA CONTINUAR

2 CARACTERES CORRESPONDEM
ABRAHAM      A165
ABRAHAMS     A165
ABRAMS       A165
APPERLEY     A164
APPLEBEE     A141
APPLEBY      A141
APPLEFORD    A141
PREMIR TECLA PARA CONTINUAR

1 CARÁCTER CORRESPONDE
ABRAHAM      A165
ABRAHAMS     A165
ABRAMS       A165
ADAM         A350
ADDAMS       A352
ADAMSON      A352
ALAN         A450
ALLAN        A450
ALLEN        A450
ANTHANY      A535
ANTHONY      A535
ANTONY       A535
ANTROBUS     A536
APPERLEY     A164
APPLEBEE     A141
APPLEBY      A141
APPLEFORD    A141
PREMIR TECLA PARA CONTINUAR
```


CAPÍTULO 8

Reconhecendo modelos

Normalmente reconhecemos objectos utilizando os nossos sentidos da visão, audição, sabor e tacto, ao passo que o nosso computador básico pode somente, como é óbvio, obter informação através do teclado. Embora seja possível produzir sensores, os quais podem ser interconectados *(interfaced)* com a sua máquina para lhe dar outra visão do mundo exterior, a sua construção requer uma quantidade razoável de conhecimentos de electrónica e de mecânica, assim como de perícia. Faremos a sua construção substitutiva com uma simulação da acção dum sensor luminoso, para ilustrar como é que os modelos podem ser reconhecidos.

Vamos pensar, para começar, em três modelos simples — uma linha vertical, um quadrado e um triângulo rectângulo.

Podemos reconhecer estes modelos ao observarmos a forma que eles produzem numa rede imaginária e decidindo se existe ou não um ponto colocado em cada coordenada X e Y.

No caso duma linha, somente a primeira coordenada X é utilizada, embora utilizando-se todas as coordenadas Y. Um quadrado é um pouco mais complicado, visto que todas as coordenadas X nas filas 1 e 8 de Y são colocadas, e das filas Y, de 2 a 7, somente o primeiro e último pontos de X são posicionados. Finalmente, um triângulo é ainda mais complicado, uma vez que o declive é produzido incrementando-se o eixo X de cada vez.

QUADRO 8.1

Valores decimais de modelos descritos em formato binário

Fila Y	Linha	Quadrado	Triângulo
1	1	255	1
2	1	129	3
3	1	129	5
4	1	129	9
5	1	129	17
6	1	129	33
7	1	129	65
8	1	255	255

Uma maneira óbvia de descrever estas figuras particulares seria representar cada ponto por um único bit e produzir um valor decimal para cada fila, da mesma maneira que fizemos anteriormente, quando estávamos a dar atenção aos sistemas inteligentes (ver quadro 8.1).

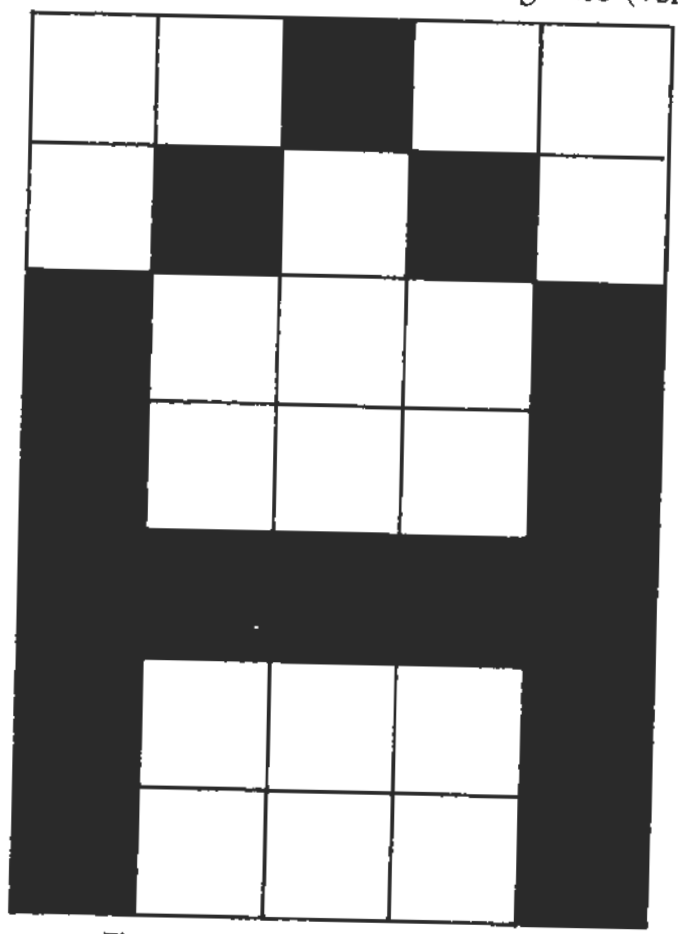


Figura 8.1 — *Formando a letra «A»*

De facto, este tipo de aproximação é usado para produzir os caracteres que visualiza no seu *écran* de exposição, os formatos para os quais são armazenados na memória justamente nesta forma. Por exemplo, a figura 8.1 mostra como a letra «A» é construída.

Existem agora máquinas no mercado (leitoras de caracteres ópticos) que podem reverter este processo. Elas «lêem» na realidade uma página impressa percorrendo o papel num formato em rede e dimensionando se a luz é ou não reflectida em coordenadas particulares.

O que elas na realidade introduzirão será um modelo de «sim» e «não» para cada coordenada, e, como é evidente, isto deve ser decifrado e comparado com os modelos para formas conhecidas. O processo mais óbvio de fazer esta comparação seria considerar cada um dos pontos por sua vez como um algarismo binário, convertendo-se de seguida cada fila de novo para um valor decimal, o qual podia ser comparado com um quadro de valores conhecidos. Porém, isto tem a desvantagem de que temos na realidade de verificar cada ponto individualmente na rede (64 pontos).

Um corte curto duma ramificação

Uma aproximação mais rápida reside no facto de cada carácter poder na realidade ser detectado observando-se somente um número muito mais pequeno de características críticas do modelo. Por exemplo, a figura 8.2 dá uma árvore de decisão, a qual descobrirá todas as letras maiúsculas do alfabeto usando somente 12 pontos (ver figura 8.3), não sendo mesmo necessário verificar todos os 12 em qualquer caso particular. Se seguir cada um dos percursos, observará que o número máximo de passos a serem seguidos é sete, e que a maioria das letras são encontradas em menos de cinco passos (quadro 8.2). Isto deve obviamente ser mais rápido do que se comparar todos os 64 pontos!

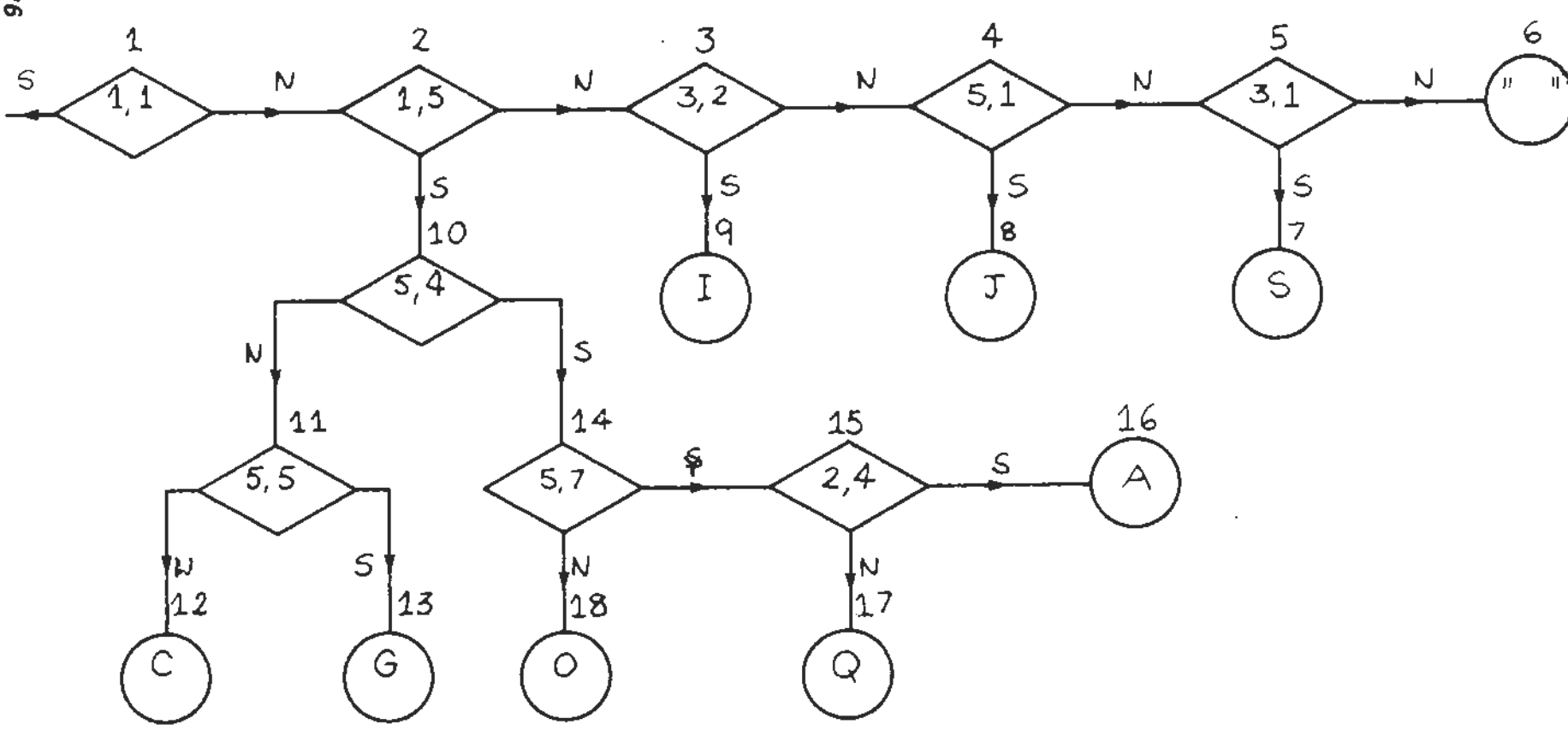
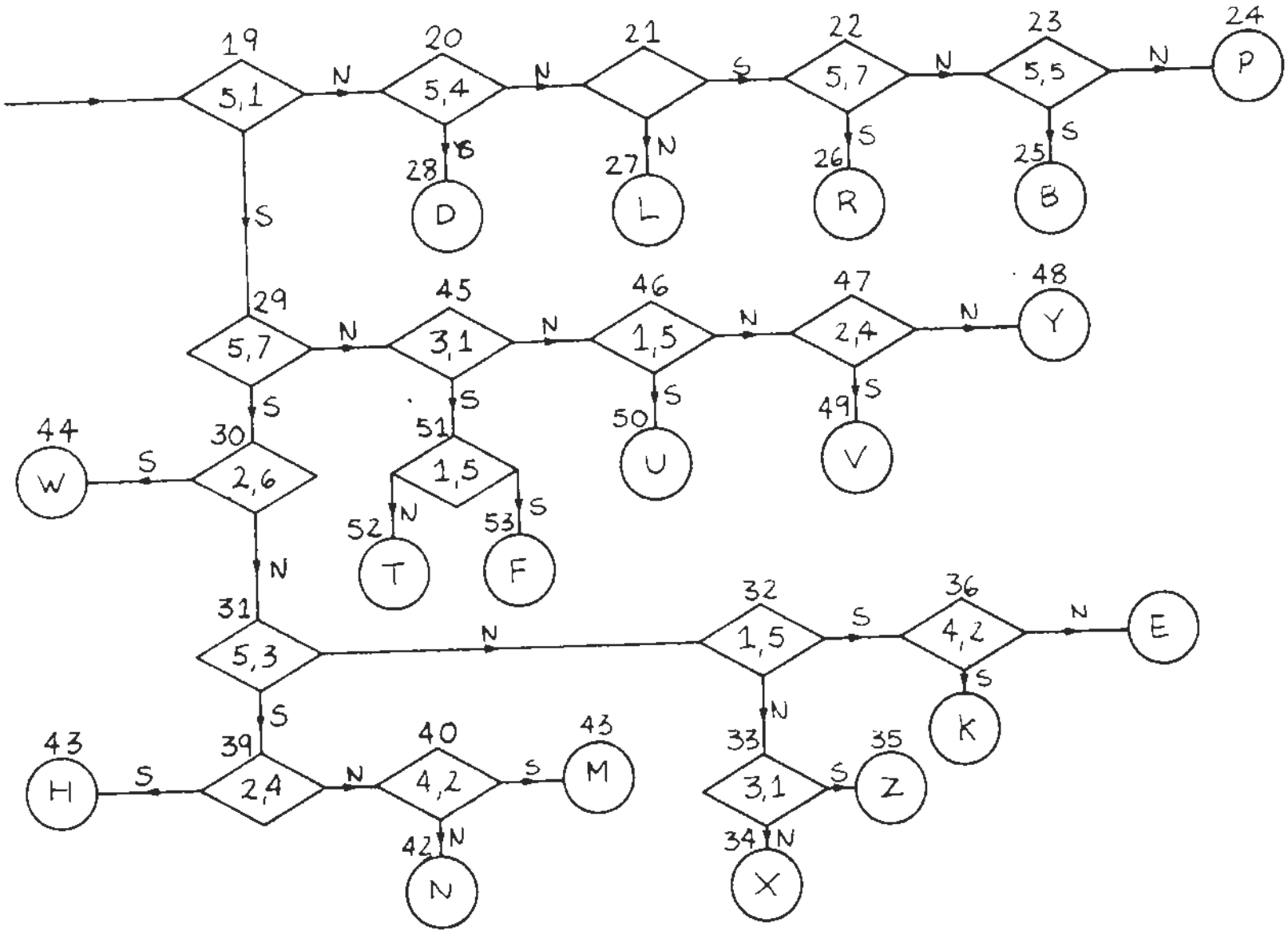


Figura 8.2-*a* — *Árvore de decisão para o alfabeto*



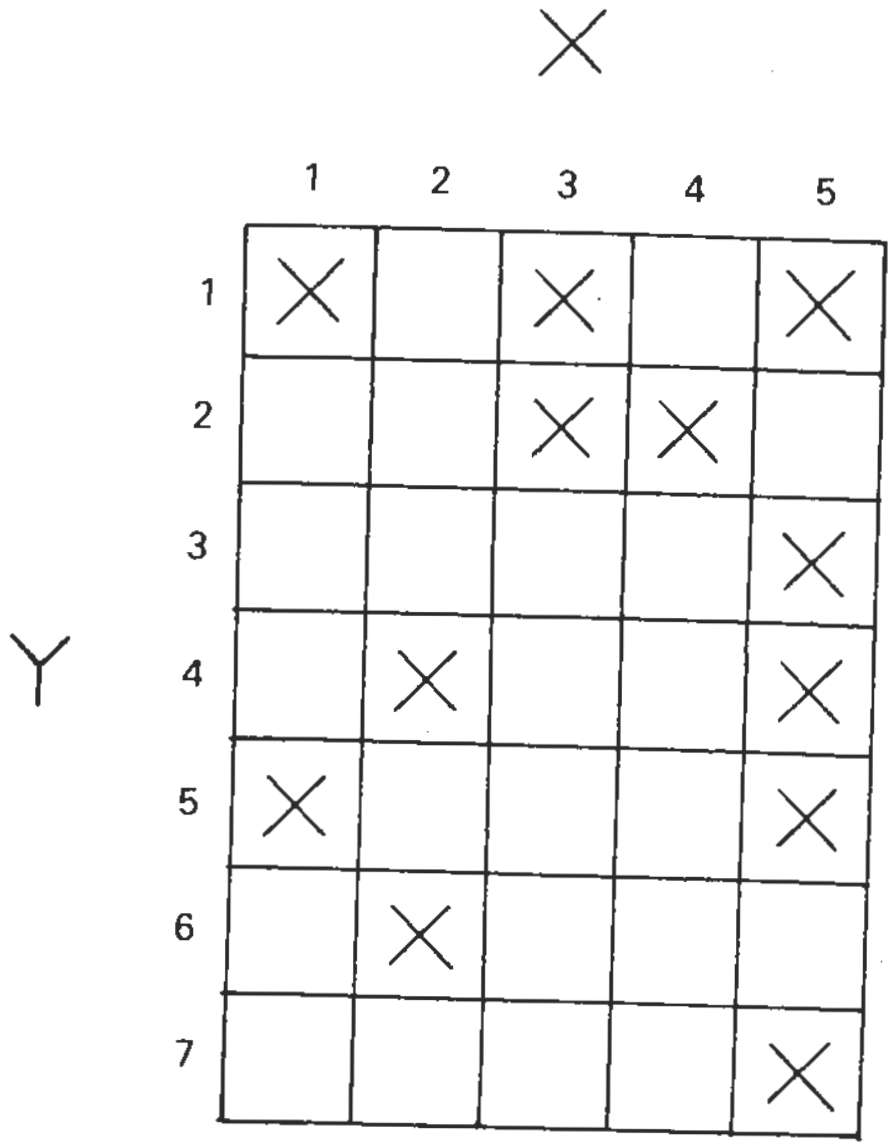


Figura 8.3 — *Pontos usados na árvore de decisão*

QUADRO 8.2

Números de passos requeridos para reconhecimento de cada carácter

3 passos — I, D
4 passos — L, J, C, G, O, W
5 passos — S, A, Q, R, T, F, U, espaço
6 passos — P, V, Y, H
7 passos — B, M, N, E, K, X, Z

Para demonstrar como é que esta aproximação funciona, simularemos a acção da cabeça de perscrutamento, produzindo uma rede no *écran*, na qual se podem construir caracteres.

O *écran* é limpo e uma área preta (PAPER 0), com blocos de 6×8, é instalada no canto superior esquerdo do mesmo. Uma rede amarela (PAPER 6), de 5×7, é então sobreposta nesta para se demarcar a área real de trabalho (claro está que deve existir uma margem em redor do bordo, de forma que os caracteres não se confundam).

```
   10 GO SUB 9000
9000 CLS
9010 FOR X=1 TO 7
9020 FOR Y=1 TO 9
9030 PRINT AT Y,X; PAPER 0;" ";
9040 NEXT Y: NEXT X
9050 FOR X=2 TO 10
9060 FOR Y=2 TO 8
9070 PRINT AT Y,X; PAPER 6;" ";
9080 NEXT Y: NEXT X
9090 LET X=2: LET Y=2
9200 RETURN
```

Um cursor tremeluzente é agora produzido para mostrar a sua posição. O carácter na posição actual no *écran*, SCREEN$(Y,X), é encontrado e armazenado como C$. Um sinal de cardinal (#) é então impresso (PRINT) na (AT) posição, sendo o carácter original C$ impresso de novo, de forma que não exista qualquer efeito duradoiro.

```
  20 LET A$=INKEY$
  30 LET C$=SCREEN$ (Y,X): PRINT
AT Y,X; PAPER 6;"#": PAUSE 10:
PRINT AT Y,X; PAPER 6;
  40 IF A$="" THEN GO TO 20
```

As coordenadas X e Y são actualizadas de acordo com o movimento das teclas do cursor, e se a barra de espaços for premida é impresso um asterisco na posição actual. Se se cometer um erro, premindo-se «x» apaga a posição actual, imprimindo um espaço, ou a maiúscula «Z» («z»+CAPS/SHIFT) salta para a rotina inicial e apaga toda a rede actual. Premindo-se ENTER(CHR$(13)) conduz à rotina de decifração, ou em qualquer outro caso o programa volta atrás no ciclo à verificação de tecla.

```
   50 IF A$="2" THEN   LET X=X+1
   60 IF A$="5" THEN   LET X=X-1
   70 IF A$="6" THEN   LET Y=Y+1
   80 IF A$="7" THEN   LET Y=Y-1
   90 IF A$=" " THEN   PRINT AT Y,
X; PAPER 6;"*";: LET X=X+1
  100 IF A$="X" THEN   PRINT AT Y,
X;" ";
  110 IF A$="Z" THEN   GO SUB 3010
  120 IF A$=CHR$ (13) THEN   GO TO
  2000
  170 GO TO 20
```

Devem ser estabelecidos limites para evitar que o cursor vá para além da área da rede de 5×7.

```
130 IF X<2 THEN   LET X=2
140 IF X>13 THEN   LET X=13
150 IF Y<2 THEN   LET Y=2
160 IF Y>8 THEN   LET Y=8
```

A árvore de decisão é mantida numa série de tabelas interligadas, onde NB é o número de ramos, L$(n) assegura os nomes das letras, J(n) a coordenada X a ser verificada a seguir, K(n) a coordenada Y a ser verificada a seguir, L(n) o próximo elemento a utilizar se a resposta for «não», e M(n) o próximo elemento a usar se a resposta for «sim».

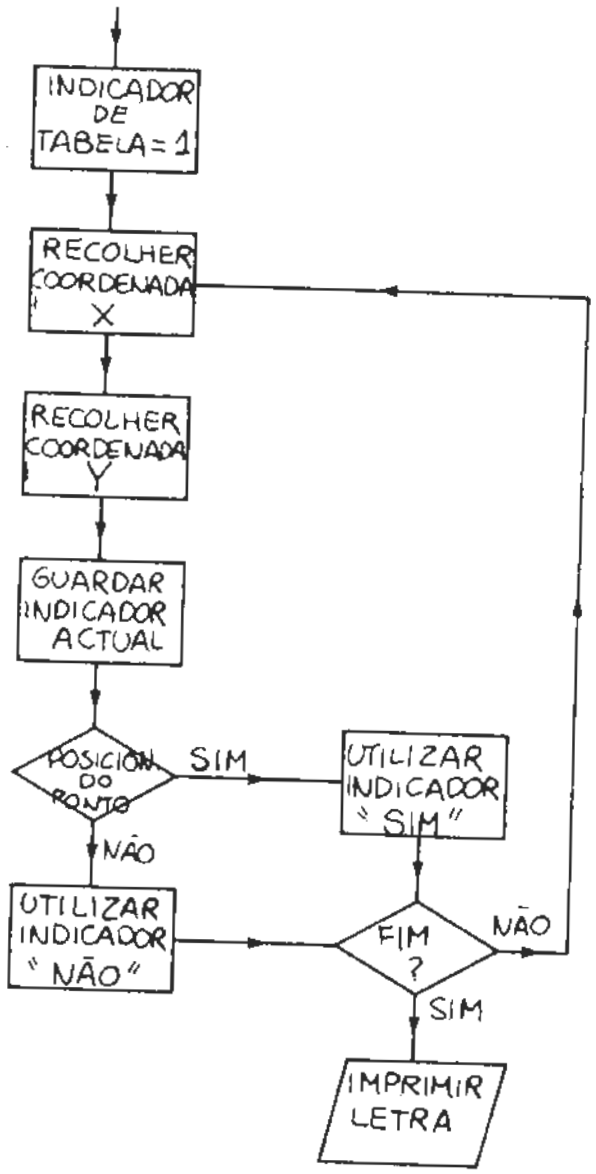
```
9100 LET NB=53
9110 RESTORE : DIM L$(NB,1): DIM
  J(NB): DIM K(NB): DIM L(NB): DI
M M(NB)
9120 FOR N=1 TO NB
9130 READ L$(N),J(N),K(N),L(N),M
(N)
9140 NEXT N
```

A melhor maneira de se entrar com a DATA é provavelmente com 53 linhas separadas (uma para cada ponto de ramificação), uma vez que tal disposição facilita a sua entrada assim como a edição de quaisquer erros. Infelizmente, o BASIC do Sinclair requer que se entre na realidade com cadeias vazias ou preenchidas com zeros onde quer que não se deseje entrar com quaisquer valores!


```
9300 DATA "",13,15,0,0
9310 DATA "",11,9,8,7
9320 DATA "",9,3,4,5,6
9330 DATA "",7,1,2,5,5
9340 DATA "",5,3,1,0,4
9350 DATA "S",0,0,0,0,3
9360 DATA "J",0,0,0,0,2
9370 DATA "I",0,0,0,0,1
9380 DATA "",0,0,0,0,0
9390 DATA "C",0,0,0,0,0
9400 DATA "G",0,0,0,0,0
9410 DATA "",0,0,0,0,0
9420 DATA "A",0,0,0,0,0
9430 DATA "Q",0,0,0,0,0
9440 DATA "O",0,0,0,0,0
9450 DATA "",0,0,0,0,0
9460 DATA "P",0,0,0,0,0
9470 DATA "B",0,0,0,0,0
9480 DATA "R",0,0,0,0,0
9490 DATA "L",0,0,0,0,0
9500 DATA "D",0,0,0,0,0
9510 DATA "",5,7,45,30
9520 DATA "",2,6,31,44
9530 DATA "",5,3,32,39
9540 DATA "",1,5,33,36
9550 DATA "",3,1,34,35
9560 DATA "X",0,0,0,0
9570 DATA "Z",0,0,0,0
9580 DATA "",4,2,38,37
9590 DATA "K",0,0,0,0
9600 DATA "E",0,0,0,0
9610 DATA "",2,4,40,43
9620 DATA "",4,2,42,41
9630 DATA "M",0,0,0,0
9640 DATA "N",0,0,0,0
9650 DATA "H",0,0,0,0
9660 DATA "W",0,0,0,0
9670 DATA "",3,1,46,51
9680 DATA "",1,5,47,50
9690 DATA "",2,4,48,49
9700 DATA "Y",0,0,0,0
9710 DATA "V",0,0,0,0
9720 DATA "U",0,0,0,0
9730 DATA "",1,5,52,53
9740 DATA "T",0,0,0,0
9750 DATA "F",0,0,0,0
```

Se estiver mais confiante (ou se estiver a tentar poupar espaço), então toda a DATA pode ser condensada num número mais reduzido de linhas bastante mais ilegíveis, as quais são aceitáveis para aqueles que sejam bons contadores de vírgulas, mas difícil de entrar correctamente ou de editar.

Para se verificar o desenho produzido em confronto com os modelos disponíveis (ver ordinograma 8.1), o indicador de referência (índice) da tabela AP é em primeiro lugar posicionado a 1, de forma que a pesquisa seja iniciada a partir do princípio. As coordenadas X e Y são lidas a partir dos elementos endereçados J(AP) e K(AP), e o último



Ordinograma 8.1 — *Reconhecimento de caracteres*

indicador de posição LP é posto em igualdade eom o actual indicador de tabela AP.

O carácter existente nestas coordenadas é agora determinado por SCREEN$(X,Y). Caso este seja um asterisco, então este ponto foi posicionado, devendo ser seguido o indicador «sim», M(AP). Se qualquer outro valor for encontrado, então é seguido o indicador «não», L(AP). Em qualquer dos casos procede-se agora a uma verificação para averiguar se o elemento referenciado contém ou não um zero (o qual indica o fim extremo dum ramo), que refere que um carácter foi encontrado. Se assim for, a letra apropriada L$(LP) é impressa, sendo a exposição mantida até que uma tecla seja premida, quando for iniciado um novo ciclo. Desde que seja encontrado um valor mais elevado do que zero, isso deve corresponder a um outro ramo, de modo que o programa salta atrás no ciclo, tomando os novos valores de J(AP) e K(AP).

De forma a permitir-lhe observar que pontos é que foram verificados, estes são posicionados a «@» à medida que forem encontrados. Quaisquer pontos que tenham sido posicionados mas não testados permanecerão como asteriscos.

```
2000 LET AP=1
2010 LET X=J(AP)+1: LET Y=K(AP)+
1: LET LP=AP
2020 LET P$=SCREEN$ (Y,X)
2030 IF P$=" " THEN LET AP=M(AP)
: GO TO 2050
2040 LET AP=L(AP)
2050 IF AP=0 THEN GO TO 2070
2060 PRINT AT Y,X; PAPER 6;"@";:
 GO TO 2010
2070 PRINT AT 12,4; PAPER 6;L$(L
P);
2080 PAUSE 0
2090 GO SUB 9000: GO TO 20
```

Se desejar ver que parte da árvore foi realmente seguida, acrescentem-se então estas modificações que imprimirão a sequência seguida.

```
2000 LET AP=1: LET A=3: PRINT AT
 1,15; PAPER 6;"AP"
2050 PRINT AT A,15; PAPER 6;AP;
LET A=A+1: IF AP=0 THEN  GO TO 2
070
```

A desvantagem deste método mais rápido, de somente verificar pontos críticos, é que procederá a uma correspondência errónea caso depare com um modelo que não se encontre na árvore; ao passo que, se todos os pontos forem verificados, nenhuma correspondência será encontrada em tal caso.

As primeiras leitoras de caracteres ópticos aceitavam somente um único tipo de cabeçote impressor particular, porém as máquinas mais recentes não só aceitam diferentes estilos de impressão, mas também aprendem realmente as regras de reconhecimento por si próprias, por intermédio dum sistema inteligente de construção interna. Ensina-as mostrando-lhes umas poucas de páginas de texto e entrando então com estes mesmos caracteres através do teclado. Todavia apercebemo-nos de que ainda é muito cedo para que alguém consiga reproduzir numa máquina a *nossa* própria escrita à mão por intermédio da sua leitura!

CAPÍTULO 9

Um instrutor inteligente

Uma outra área em que a inteligência artificial pode ser particularmente útil é na instrução de programas. É óptimo possuir um programa que teste os conhecimentos dum estudante ao acaso, porém essa não é exactamente a maneira como um verdadeiro professor humano trabalha. Expondo as diversas questões ao aluno, ele mantém-se atento ao progresso dos estudantes, aumenta o grau de dificuldade das questões à medida que a experiência aumenta, testando-os com maior rigor naqueles géneros de problemas com os quais têm mais dificuldades. Por exemplo, se uma criança fizer um teste que envolva operações de adição, subtracção, multiplicação e divisão, mas que só tenha erradas as perguntas referentes à divisão, segue-se então que se deveria dar à criança mais questões sobre a divisão, de futuro, de forma a proporcionar-lhe maior prática.

Vamos dar atenção à maneira como se podem introduzir estas qualidades «humanas» num programa de instrução.

Perguntas e respostas

Precisamos de criar números (INT) aleatórios (RND) a serem utilizados na primeira pergunta, para a qual procederemos a uma adição. Usando INT(RND*10) dará números entre 0 e 9.

```
20 LET A=INT (RND*10)
30 LET A=INT (RND*10)
```

O computador soma estes dois números, indo de seguida para uma entrada e sub-rotina de verificação na linha 1000.

```
40 LET C=A+B: GO SUB 1000
```

Em primeiro lugar a rotina deve imprimir a pergunta e dar a entrada à sua resposta, IP.

```
1000 PRINT A;"+";B;"=",
1010 INPUT IP: PRINT IP
```

A sua resposta deve então ser verificada. Se a resposta C for a mesma que a sua resposta, então é impresso CORRECTO, voltando a rotina para a linha 40. De outra forma, é impresso ERRADO, seguido da resposta correcta.

```
1020 IF C=IP THEN PRINT "CORRECT
O": RETURN
1030 PRINT "ERRADO, A RESPOSTA C
ORRECTA ERA ";C
1040 RETURN
```

As outras três operações (subtracção, multiplicação e divisão) podem ser facilmente tratadas da mesma maneira se substituirmos o sinal «+» na linha 1000 por um sinal de cadeia S$, o qual pode ser posicionado no carácter apropriado na devida altura. Visto que INT(RND*10) é comum a todos os cálculos, podemos igualmente definir isto como uma função, RD.

```
  10 LET X=0
  15 DEF FN R(X)=INT (RND*10)
  20 LET A=FN R(X)
  30 LET B=FN R(X)
  40 LET S$="+": LET C=A+B: GO S
UB 1000
  50 LET A=FN R(X)
  60 LET B=FN R(X)
  70 LET S$="-": LET C=A-B: GO S
UB 1000
  80 LET A=FN R(X)
  90 LET B=FN R(X)
 100 LET S$="*": LET C=A*B: GO S
UB 1000
 110 LET A=FN R(X)
 120 LET B=FN R(X)
 130 LET S$="/": LET C=A/B: GO S
UB 1000
1000 PRINT A;S$;B;"=";
```


Finalmente saltamos atrás para a linha 20, para introduzir mais questões.

```
140 GO TO 20
```

Dividindo por zero

Tal como se encontra, o programa pode falhar se B for zero quando for escolhida uma divisão. Isto pode ser facilmente solucionado adicionando sempre um a B neste caso específico.

```
120 LET B=FN R(X)+1
```

Eliminando decimais

Estamos a utilizar variáveis de algarismos para nos fornecer números inteiros, mas, como é óbvio, uma divisão pode também produzir uma resposta fraccionada, a qual não se pode fazer entrar correctamente, uma vez que IP será arredondado por defeito, por exemplo:

3/2 = 1,5

Porém o programa aceitará como correcto 1, 1,5, 1,9 ou qualquer outro número entre 1 e 1,999...

Para evitar a reprodução de decimais, A precisa de ser múltiplo de B. Para se conseguir isto calcula-se B em primeiro lugar, fazendo-se A igual a B multiplicado por um número aleatório compreendido entre 0 e 10.

```
110 LET B=FN R(X)+1
120 LET A=INT (FN R(X))*B
```

Mantendo um marcador de contagem

Agora que temos o teste a trabalhar por si próprio, precisamos de considerar como manter um contador. O processo mais simples é incrementar uma variável de teste TR de cada vez que a sub-rotina na linha 1000 é utilizada, e incrementar uma variável de contagem SC de cada vez que uma resposta é obtida.

```
  10 LET X=0: LET TR=0: LET SC=0
1010 INPUT IP: LET TR=TR+1
1020 IF C=IP THEN PRINT "CORRECT
O": LET SC=SC+1: GO TO 1040
1040 PRINT "O SEU RESULTADO E ";
S;"/";TR: RETURN
```

Se preferir o resultado da contagem como uma percentagem, emende-se então a linha 1040 como se segue:

```
1040 PRINT "VOCE TINHA ";INT ((S
C/TR)*100);" CORRECTO ": RETURN
```

Quantas perguntas?

Tal como se encontra, o programa aceitará uma questão de cada tipo sequencialmente, até ao infinito. Podemos limitar o número definindo o número de questões, NQ, como uma variável.

```
  10 LET X=0: LET TR=0: LET SC=0
: LET NQ=32
```

De cada vez que for feita uma pergunta, NQ é baixado em uma unidade: quando NQ = 0, os testes acabam (depois de oito perguntas de cada tipo terem sido respondidas).

```
 140 IF NQ>0 THEN GO TO 20
 150 STOP
1010 INPUT IP:LET TR=TR+1:LET NQ
=NQ-1
```


Mudando a ênfase

Se deslocarmos as perguntas em benefício de áreas de dificuldade maior, precisamos de manter um registo de qualidade da execução *(performance)* para cada área individualizada. Precisamos por conseguinte de variáveis separadas para cada tipo de questão (AD para a adição, SU para a subtracção, MU para a multiplicação, e DI para a divisão). Estas variáveis são definidas em termos de um oitavo do número total de perguntas a serem feitas, NQ.

```
  10 LET X=0: LET TR=0: LET SC=0
  LET NQ=32:LET AD=NQ/8:LET SU=A
D:LET MU=AD:LET DI=AD
```

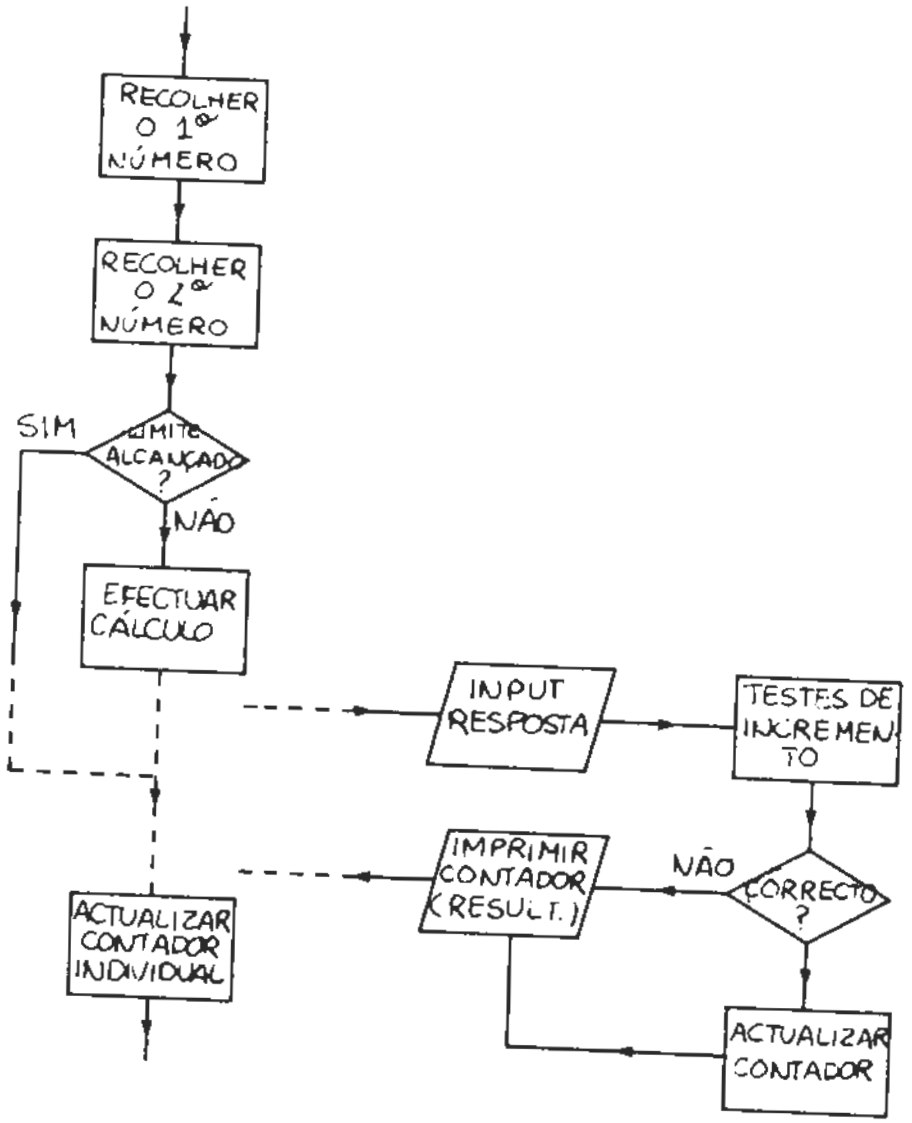
Agora, se a resposta correcta for a mesma que a sua resposta IP, é posicionada a −1 uma variável de incremento INC, é impresso CORRECTO, e a rotina retorna (Return). De outro modo, IN é posicionado a 1, e é impresso ERRADO, seguido pela resposta certa.

```
1020 IF C=IP THEN LET INC=-1: PR
INT "CORRECTO": RETURN
1030 LET INC=1: PRINT "ERRADO, A
 RESPOSTA CERTA ERA ";C
1040 RETURN
```

INC é somado ao número individualizado apropriado da variável de perguntas, AD, SU, MU ou DI, em retorno, produzindo-se um aumento neste valor se a resposta fosse errada, ou um rebaixamento se a resposta fosse correcta.

```
  40 LET S$="+": LET C=A+B: GO S
UB 1000: LET AD=AD+INC
  70 LET S$="-": LET C=A-B: GO S
UB 1000: LET SU=SU+INC
 100 LET S$="*": LET C=A*B: GO S
UB 1000: LET MU=MU+INC
 130 LET S$="/": LET C=A/B: GO S
UB 1000: LET DI=DI+INC
```

Acrescentamos agora uma verificação para ver se todas as perguntas dum tipo particular foram ou não respondidas correctamente (por exemplo, AD>0, ver ordinograma 9.1). Se todas as questões dum determinado tipo tiverem sido correctamente respondidas, então nenhuma mais deste tipo será apresentada, uma vez que a linha correspondente é ultrapassada. Se o número apropriado de cada tipo tiver sido respondido correctamente (AD = 0, SU = 0, MU = 0, DI = 0), então o programa acaba.



Ordinograma 9.1 — *Instrutor inteligente*


```
  40 IF AD>0 THEN LET S$="+": LE
T C=A+B: GO SUB 1000: LET AD=AD+
INC
  70 IF SU>0 THEN LET S$="-": LE
T C=A-B: GO SUB 1000: LET SU=SU+
INC
 100 IF MU>0 THEN LET S$="*": LE
T C=A*B: GO SUB 1000: LET MU=MU+
INC
 130 IF DI>0 THEN LET S$="/": LE
T C=A/B: GO SUB 1000: LET DI=DI+
INC
 140 IF AD=0 AND SU=0 AND MU=0 A
ND DI=0 THEN GO TO 150
```

Note-se que não voltam a apresentar-se questões acerca de áreas para as quais se respondeu correctamente a quatro perguntas sem se cometer quaisquer erros. Se houve um erro, então AD, etc., será aumentado, e, por conseguinte, ter-se-á de responder correctamente a mais de quatro antes de AD atingir o valor zero.

Graus de dificuldade

Que tal tornarmos mais fáceis ou mais difíceis as questões, de acordo com o sucesso da sua execução (isto é, os valores de AD, SU, MU e DI)? Até ao momento, os valores actuais de A e B têm sido sempre entre 0 e 9, uma vez que foram reproduzidos por RND*10, porém necessitamos agora de alterar os números reproduzidos para as questões — para valores mais elevados, caso se esteja certo, e para valores mais baixos, caso se esteja errado. Ao mesmo tempo, devemos certificar-nos de que não se reproduzem valores negativos, se a qualidade da sua execução for profunda.

O «caso mais grave» será se se obtiverem todas as perguntas erradas no último grupo. Neste caso serão somente apresentadas quatro questões nos três primeiros grupos, deixando 32−(3*4) = 20 questões para serem apresentadas no último grupo. Além disso, devemos recordar que X (por exemplo, AD) começa num valor de quatro, de forma que o valor máximo que poderia ser obtido para X é 20 + 4 = 24.

Instalamos, por conseguinte, uma variável de avaliação (ou ponderabilidade) WT, a qual é calculada subtraindo-se três vezes o

número de questões a serem apresentadas em cada grupo (3*AD) do número total de perguntas (NQ), acrescentando-as ao número de perguntas no grupo AD, no início.

WT = NQ − (3*AD) + AD

Isto é mais facilmente expresso pelo seguinte:

WT = NQ − (2*AD)

```
  10 LET X=0: LET TR=0: LET SC=0
: LET NQ=32: LET AD=NQ/8: LET SU
=AD: LET MU=AD: LET DI=AD: LET W
T=NQ-(2*AD)
  20 LET A=FN R(AD)
  30 LET B=FN R(AD)
  50 LET A=FN R(SU)
  60 LET B=FN R(SU)
  80 LET A=FN R(MU)
  90 LET B=FN R(MU)
 110 LET B=FN R(DI)+1
 120 LET A=INT (FN R(DI))*B
```

Substituímos agora o valor fixado em dez pela diferença entre WT e X.

```
 15 DEF FN R(X)=INT (RND*(WT-X)
)
```

Para se inicializar, WT = 24 e X = 24, portanto serão seleccionados números entre 0 e 19. Se for dada uma resposta correcta, então X é reduzido para 3 e serão escolhidos números entre 0 e 20. Após quatro respostas correctas, X não mudará (para este tipo de questão), uma vez que terá atingido o valor zero, sendo saltada a linha respectiva. Os últimos valores serão, por conseguinte, compreendidos entre 0 e 22.

Contudo, se a primeira resposta for incorrecta, então X aumentará uma unidade, sendo a amplitude dos números produzidos reduzida uma unidade (0-18). No «pior dos casos», X será aumentado de 20 para 24 vezes e (WT-X) cairá para zero para ambos, A e B (portanto deve-se ser capaz de resolver esse problema particular!).

CAPÍTULO 10

Combinando tudo

Nos capítulos anteriores lidámos, a partir de princípios básicos, com vários aspectos da inteligência artificial. Neste último capítulo interligámos muitas destas ideias individualizadas num único programa completo.

O programa «inteligente» original era o famoso «ELIZA», o qual era um pseudoprograma psiquiátrico escrito para viabilizar um estilo particular de terapia psiquiátrica. Resistimos à tentação de seguir esta orientação, tendo optado em seu lugar pela reprodução dum seu substituto acessível ao vendedor médio de computadores. Este programa combina algumas ideias sobre o processamento da linguagem natural, assim como acerca de sistemas inteligentes especializados, para produzir um resultado que deve não só compreender os seus pedidos como fazer sugestões que tomem em consideração as suas solicitações e um determinado número de factos comerciais complexos.

Foram já incluídos palavras suficientes e valores para fazer com que o programa se tornasse interessante, porém pode facilmente adaptá-lo juntando as suas próprias ideias à DATA. (Não assumimos qualquer responsabilidade pelos valores incluídos até ao momento, os quais servem, unicamente, propósitos de demonstração ou observações sobre determinadas máquinas expressas pelo programa!) O programa em si é bastante complexo mas segue os métodos descritos anteriormente no livro, e as funções das diversas variáveis de linha e tabelas são apresentadas no quadro 10.1.

Tal como se encontra de momento, o programa pode à *justa* (!) ser «espremido» num *Spectrum* de 16K (se tiver mais memória disponível não há quaisquer problemas, pode facilmente expandir o programa caso o desejc). Todavia, uma vez que é utilizado virtualmente o último byte num aparelho de 16K, aperceber-se-á que se as suas inscrições de entrada forem extensas o programa falhará e dar-lhe-á uma informação de erro «4 Out of memory» durante as comparações de cadeia. Não se pode deitar um almude num recipiente de litro; portanto, se quiser ser capaz de entrar com inscrições muito compridas terá de cortar em qualquer outro ponto para lhe dar lugar (por exemplo, encurte o tamanho das tabelas, usando para tal menos características — reduzindo o valor de FE na linha 9999 para 16, e removendo os dois últimos itens de DATA nas linhas 940 e 1010).

QUADRO 10.1

Variáveis principais para o «Vendedor»

N$	cadeia de entrada
WL	comprimento da palavra
J$	cadeia de pesquisa
I$	cadeia de objectivos
SP	indicador de pesquisa
FT	inicializador de pesquisa
PH	número da frase
QP	número de diferentes questões
R$	nome da característica actual
Q$	pergunta actual
Q	característica actual
CR	taxa actual de custos
PR	taxa actual de lucros
TP	lucro total
TC	custo total
R	número de regras ou normas
RU	marcador de actualização de regras
OB	número de objectos, ou complementos directos
AJ	número de adjectivos
AV	número de advérbios
LI	número de preferências
DL	número de rejeições

NJ	número de adjectivos negativos
NV	número de advérbios negativos
HM	número de economias/dispêndios
BB	saldo bancário
FE	número de características
CT	número de taxas de custo
CS	número de sugestões de custo
EX	número de pedidos de desculpa
HI	número de sugestões de preço máximo
LO	número de sugestões de preço mínimo
LD	preferência/rejeição (aprovação/desaprovação)
NP	negativo
OF	sinalizador de objecto
OM	correspondência de objecto
D$	lista de taxa de custos dos aparelhos
R(n)	regra de decisão
I(n)	regra de lucro
J(n)	regra de custo

Travando conversação

O programa está elaborado de maneira a interrogá-lo acerca dos seus pontos de vista sobre cada uma das possíveis características de cada vez (o palavreado exacto da questão será seleccionado aleatoriamente a partir duma selecção de frases). Note que a palavra-chave ou frase é inserida na declaração onde for necessária e que é aplicada a conjugação correeta.

A sua entrada é examinada pormenorizadamente por palavras-chave, sendo actualizada uma tabela de regras de acordo com os seus pedidos. Um BEEP de curta duração soa de cada vez que a rotina INSTR (linha 10) é chamada, de forma que possa certificar-se de que a máquina não se «engasgou» caso tenha entrado com uma longa declaração, sendo produzida uma impressão da tabela de regras de modo a poder observar-se a sua actualização a realizar-se. A primeira fila de números é a regra de decisão, a segunda a taxa de custo e a terceira a taxa de lucro. Muitas das palavras-chave são truncadas, de

forma a que uma mesma verificação possa ser feita para um certo número de palavras similares. Repare-se que nenhuma verificação foi incluída para se observar se um espaço precede uma cadeia combinada por correspondência — portanto palavras inseridas tais como «desAPROVAR» *serão* encontradas por este programa.

A resposta mais fácil é «SIM» ou «NÃO», a qual adiciona ou subtrai 1 da regra para essa característica. Se se mencionar o nome da característica (por exemplo «GRÁFICOS»), então um 1 posterior é adicionado à regra. Além disso, utilizando um adjectivo ou advérbio «positivos» acrescenta-se à regra, ao passo que com um adjectivo ou advérbio «negativos» subtrai-se da regra. Separar as palavras por classes diferenciadas entre si permite-lhe realizar mais de uma modificação à regra ao mesmo tempo.

Assim:

SIM	acrescenta um
GRÁFICOS SIM	acrescenta dois
SOM NECESSÁRIO SIM	acrescenta três
BOM TECLADO NECESSÁRIO SIM	acrescenta quatro

Ao passo que:

NÃO	subtrai um
MEMÓRIA NÃO	subtrai dois

Mais ainda: os verbos são agrupados como «APROVAÇÕES» e «DESAPROVAÇÕES», em que o último dos casos inverte a acção do resto das palavras.

Assim:

EU DETESTO UNIDADES DE DISQUETE MACRO	subtrai dois

Em inglês, ambos os casos «NO-» e «N'T» são reconhecidos, e a maioria das negativas duplas são interpretadas correctamente nesta língua.

Assim, temos:

I DON'T LIKE SOUND	subtrai dois
I DON'T HATE SOUND	acrescenta dois

Qualquer coisa que apareça no princípio duma declaração, seguida por uma vírgula, é normalmente cortada e efectivamente ignorada.

Assim, temos:

NÃO, SOM	acrescenta dois à regra de som

Aplica-se a excepção quando «E» ou «MAS» forem incluídos, quando ambas as partes da declaração forem efectuadas independentemente.

Assim, se a pergunta for:

VOCÊ QUER GRÁFICOS?

e a resposta for:

NÃO, MAS BASIC

então é subtraída uma unidade da regra de gráficos, acrescentando-se dois à regra BASIC.

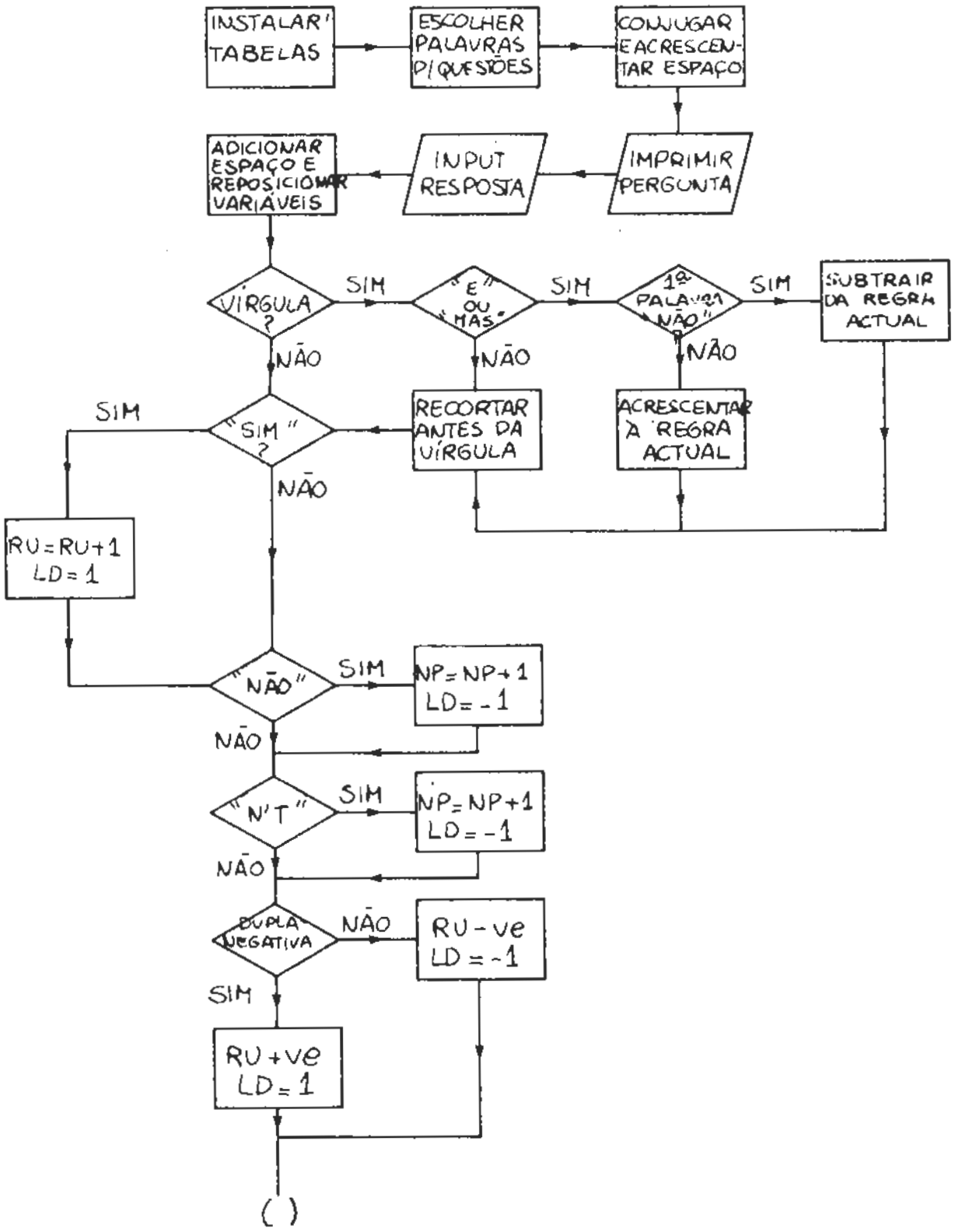
Se o programa não encontrar quaisquer palavras-chave na entrada, pede-lhe atenciosamente que experimente outra vez:

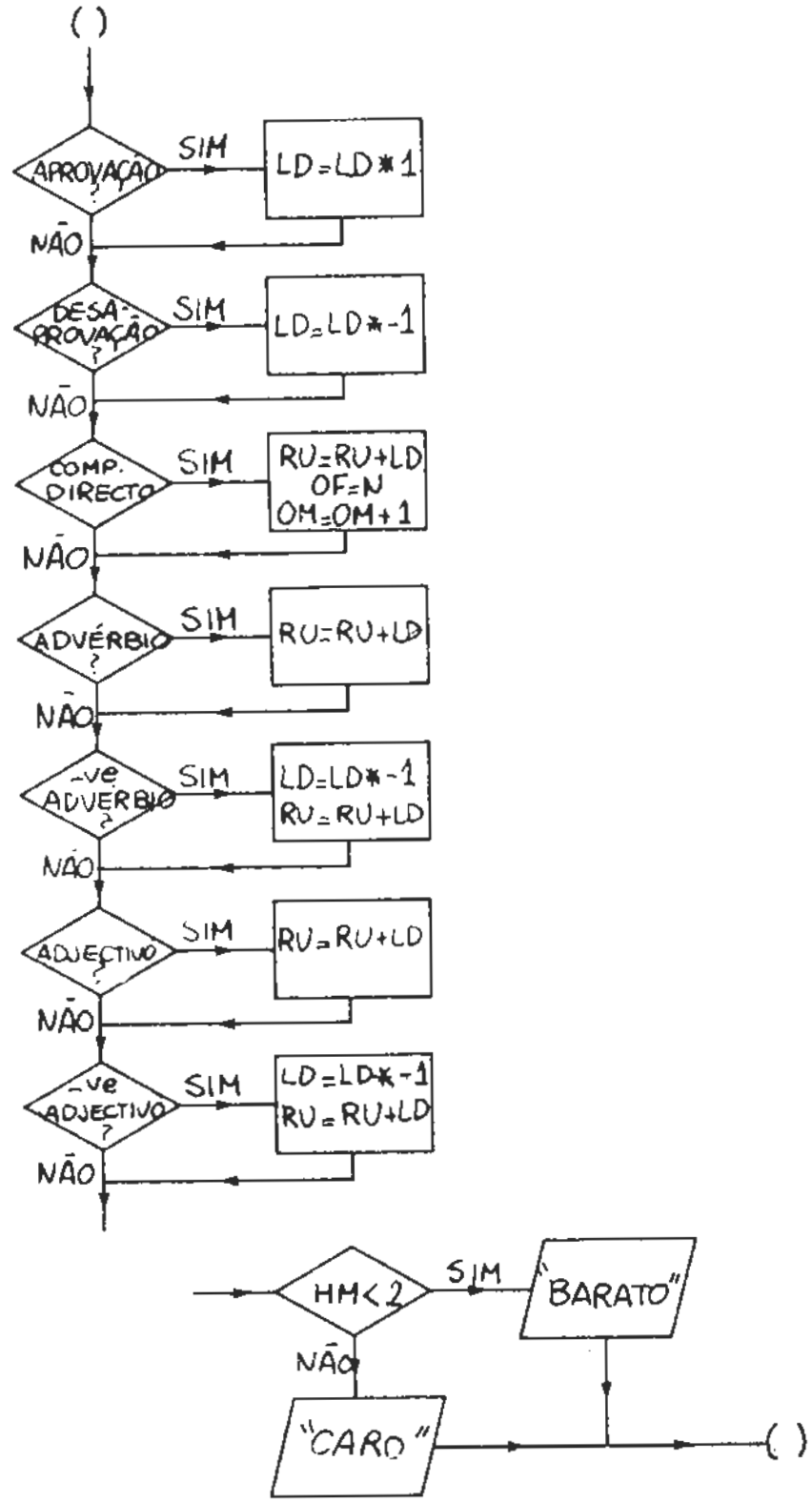
PERDÃO, DESCULPE-ME, MAS...

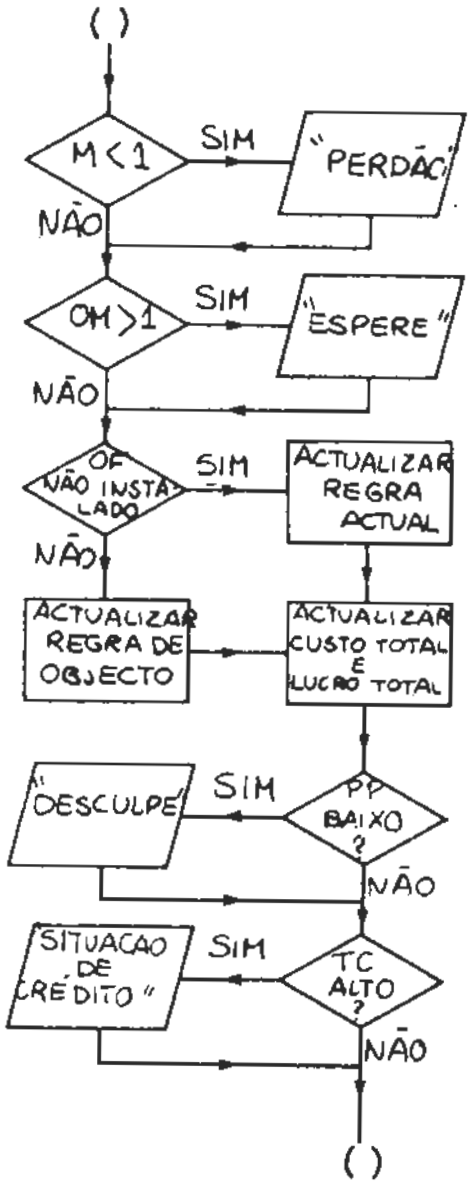
O programa só pode corresponder a uma característica de cada vez, portanto se experimentar perguntar por «SOM e GRÁFICOS» ao mesmo tempo, por exemplo, obterá um pedido de repetição da pergunta.

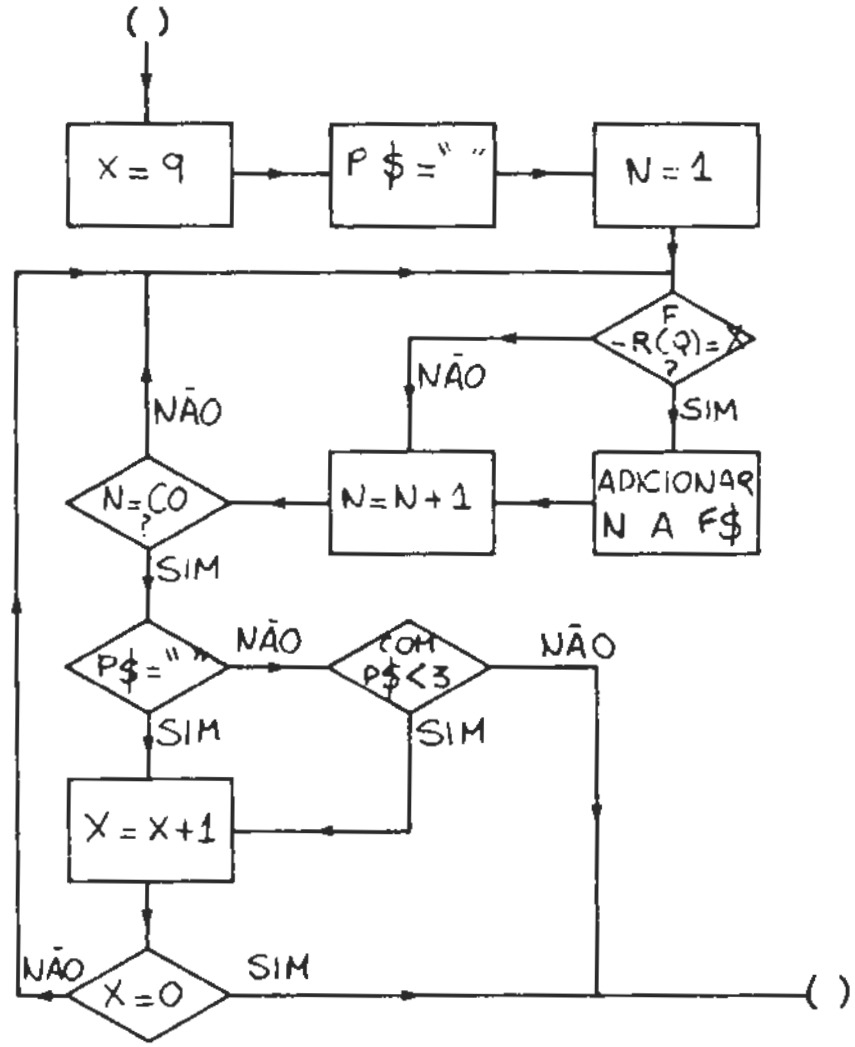
UM MOMENTO — UMA COISA DE CADA VEZ

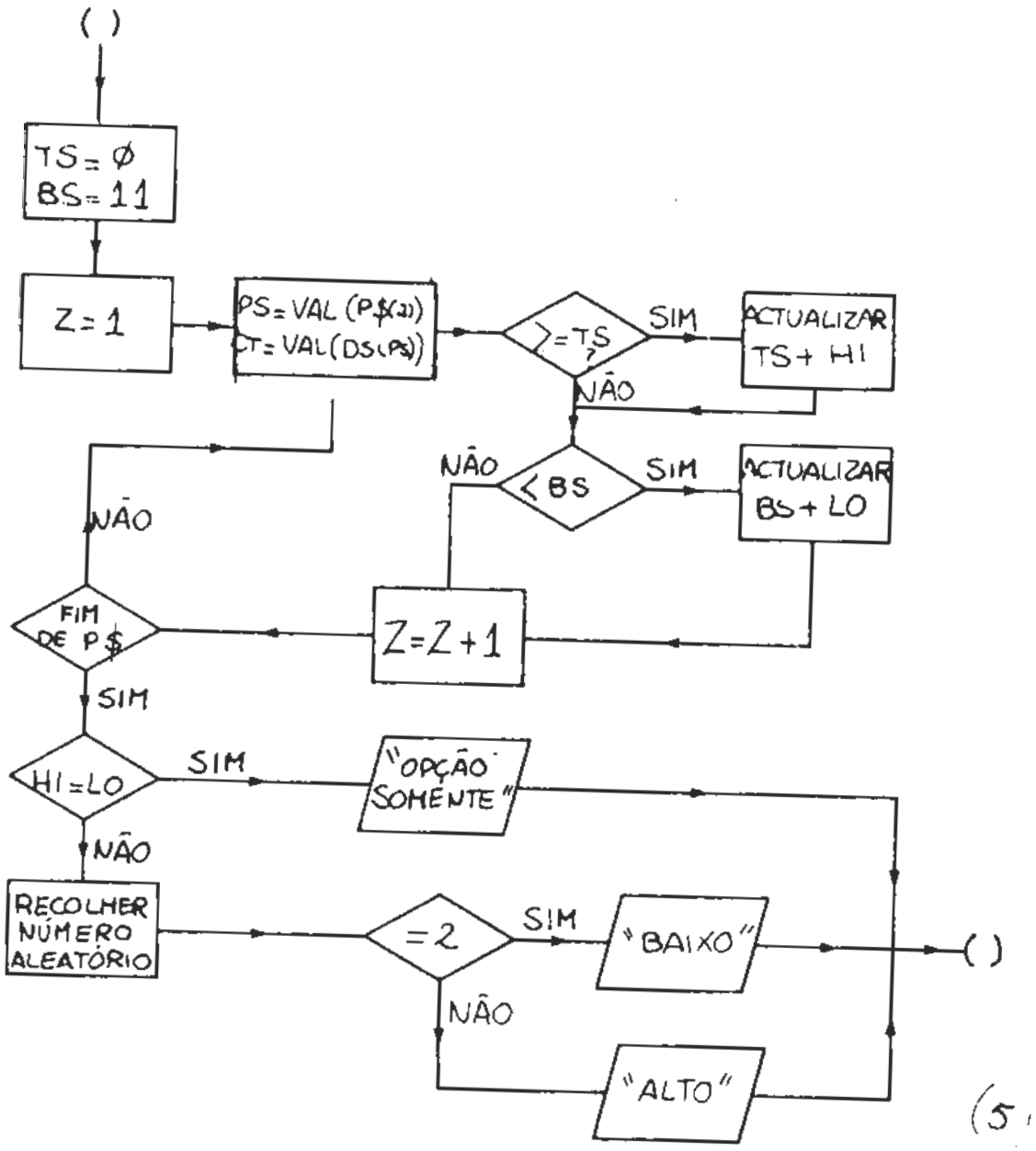
Contudo, é possível fazer comentários aerca de características simples que não estejam a ser-lhe pedidas nessa altura, e estas entradas ainda actualizarão as regras (tal como no exemplo «MAS» acima).


INSTALAR TABELAS
ESCOLHER PALAVRAS P/QUESTÕES
CONJUGAR E ACRESCENTAR ESPAÇO
IMPRIMIR PERGUNTA
INPUT RESPOSTA
ADICIONAR ESPAÇO E REPOSICIONAR VARIÁVEIS
VÍRGULA ?
SIM
"E" OU "MAS"
SIM
1ª PALAVRA "NÃO"
SIM
SUBTRAIR DA REGRA ACTUAL
NÃO
NÃO
NÃO
"SIM" ?
SIM
RECORTAR ANTES DA VÍRGULA
ACRESCENTAR À REGRA ACTUAL
NÃO
RU=RU+1
LD=1
"NÃO"
SIM
NP=NP+1
LD=-1
NÃO
"N'T"
SIM
NP=NP+1
LD=-1
NÃO
DUPLA NEGATIVA
NÃO
RU-ve
LD=-1
SIM
RU+ve
LD=1
( )


( )
APROVAÇÃO ?
SIM
LD=LD*1
NÃO
DESAPROVAÇÃO ?
SIM
LD=LD*-1
NÃO
COMP. DIRECTO
SIM
RU=RU+LD
OF=N
OM=OM+1
NÃO
ADVÉRBIO ?
SIM
RU=RU+LD
NÃO
-ve ADVÉRBIO ?
SIM
LD=LD*-1
RU=RU+LD
NÃO
ADJECTIVO ?
SIM
RU=RU+LD
NÃO
-ve ADJECTIVO ?
SIM
LD=LD*-1
RU=RU+LD
NÃO
HM<2
SIM
"BARATO"
NÃO
"CARO"
( )


( )
M<1
SIM
"PERDÃO"
NÃO
OM>1
SIM
"ESPERE"
NÃO
OF NÃO INSTALADO
SIM
ACTUALIZAR REGRA ACTUAL
NÃO
ACTUALIZAR REGRA DE OBJECTO
ACTUALIZAR CUSTO TOTAL E LUCRO TOTAL
"DESCULPE"
SIM
PP BAIXO ?
NÃO
SITUACAO DE CRÉDITO "
SIM
TC ALTO ?
NÃO
( )


( )
X = 9
P $ = " "
N = 1
F -R(Q) = ?
NÃO
SIM
NÃO
N = CO ?
N = N + 1
ADICIONAR N A F$
SIM
P$ = " "
NÃO
COM P$<3
NÃO
SIM
SIM
X = X+1
NÃO
X = 0
SIM
( )



Decisões

Em adição à tabela de regras, existem outras duas tabelas que são interligadas a esta. A primeira é a «tabela de custos», a qual dá uma indicação do custo desta opção particular, e a segunda é a «tabela de lucros», que indica ao vendedor qual o esforço que vale a pena atribuir à venda desta característiea. Os valores para estas duas últimas tabelas são reproduzidos multiplicando o conteúdo do elemento correspondente da tabela de regras por factores que são entrados originalmente como DATA nas linhas 1010, etc., cujo formato é o seguinte:

(característica de descrição da frase, custo, lucro)

Depois de cada entrada (input), o vendedor considera as consequências dos seus pedidos. Primeiro que tudo observa se a soma total do custo de todos os seus pedidos excede o seu saldo bancário. Se assim for, imprime um de entre a série de comentários cáusticos acerca do seu valor de crédito, tal como:

ESTA ESPECIFICAÇÃO PARECE ESTAR
A EXCEDER O SEU LIMITE DE CRÉDITO

Observa igualmente qual o lucro provável que realizará na venda: se este for muito baixo, ele começará a perder interesse, apresentando comentários tais como:

TENHO UMA ENTREVISTA IMPORTANTE

ou

FECHAMOS DENTRO DE CINCO MINUTOS

Ao mesmo tempo, será mais prestável no que diz respeito a qual dos computadores disponíveis servirá melhor as suas necessidades, esboçando uma curta lista ao comparar o valor atribuído originalmente a esta característica na descrição de cada computador com o valor que você lhe propõe. O formato para a descrição é o seguinte:

(nome, valor da característica 1, valor da característica 2, valor da característica 3, etc.)

O aparelho mais cotado será sempre escolhido primeiro; mas, se possível, pelo menos três aparelhos (possivelmente com cotaçõcs inferiores) serão seleccionados, sendo a escolha final concretizada a partir destas várias propostas em alternativa, com as suas respectivas características. Tanto o computador com o preço mais baixo como o de preço mais elevado serão seleccionados para referência (aleatoriamente), por exemplo:

SE DESEJA UM ROLLS-ROYCE
ENTÃO DÊ ATENCÃO AO SEGUINTE...

e

SE ESTIVER NO MERCADO DE ORÇAMENTOS
QUE SE PASSA ENTÃO ACERCA DE...

Se somente uma máquina serve a proposta, o programa virá com o seguinte:

A SUA ÚNICA OPÇÃO É...

Vendedor

```
  1 GO TO 9999
 10 BEEP .01,0: LET WL=LEN (J$)
: FOR I=FT TO LEN (I$)-WL+1: IF
I$(I TO (I+WL-1))=J$ THEN LET SP
=I: RETURN
 20 NEXT I: LET SP=0: RETURN
940 DATA "BASIC","GRAFICO","SOM
","TECLADO","FUNCÃO","MEMORIA","
FITA","UNIDADE DE DISQUETE MACRO
","DISQUETE","SOFTWARE","CARTUCH
O","COMANDO DE CONTROLO","ASSEMB
L","CENTRONICS","RS232","EXPANSA
O","REDE DE TRABALHO","16-BIT"
950 DATA "BOM","EXCELENTE","SUP
ER","MAGNIF","PRIMEIRO","RAPIDO"
,"EFICIENTE","ESSENCIAL","MUITO"
955 DATA "REAL","MUITO","FREQUE
NTE","NECESSARIO","VERDADEIRO"
960 DATA "MAU","LIXO","POBRE","
LENTO","INEFICIENTE","POUCO","PI
OR","MENOR","MENOS"
980 DATA "NUNCA","INFREQUENTE"
990 DATA "QUERER","GOSTAR","PRE
CISAR","SOLICITAR"
```



```
1000 DATA "ODIAR","DESAPROVAR","
DETESTAR"
1010 DATA "&BOM BASIC",5,2,"@GRA
FICOS",7,2,"&SOM",6,2,"&UM BOM T
ECLADO",4,2,"@TECLAS DE FUNCOES"
,1,5,"&UMA GRANDE MEMORIA",3,6,"
&UMA FITA DE INTERFACE",2,5,"@UN
IDADES DE DISQUETE MACRO",2,4,"@
DISQUETES",5,8,"&SOFTWARE EXTENS
IVEL",0,9,"&UMA CARTUCHEIRA",1,6
,"&UM PAINEL DE CONTROLO",1,7,"&
UM ASSEMBLER",2,1,"&UMA ESTACAO
CENTRONICS",2,5,"&UM ADAPTADOR R
S232",2,6,"&EXPANSIVIDADE",2,9,"
&REDE DE TRABALHO",3,4,"&UMA CPU
 DE 16-BIT",1,7
1030 DATA "GOSTARIA DE","QUE SE
PASSA ACERCA DE","COMO E ACERCA
DE","VOCE EXIGE","/* IMPORTANTE"
1032 DATA "BARATO","NAO DISPENDI
OSO"
1033 DATA "CARO","DISPENDIOSO"
1100 DATA "JNC PC","KNACT SERIOU
S","CLEARSIN MT","ACHRON ILLUSIO
N","BANANA IIE","SI ELITE","COLE
CTOVISION CARBAGE","CANDY COLOUR
ED COMPUTER","COMANDEAR 64","ATR
IA 600GI"
1110 DATA 7,6,9,8,3,8,5,7,2,1
1120 DATA 8,7,7,6,5,8,5,6,8,8
1130 DATA 8,6,7,6,2,8,5,4,9,8
1140 DATA 9,8,7,6,5,7,5,2,7,8
1150 DATA 8,8,8,7,8,0,7,0,7,6
1160 DATA 8,8,8,8,3,4,8,5,2,5
1170 DATA 8,8,8,8,7,6,8,5,7,5
1180 DATA 8,0,9,9,0,0,5,5,0,0
1190 DATA 9,8,8,8,5,3,7,5,4,6
1200 DATA 9,8,8,8,5,0,2,1,9,6
1210 DATA 7,8,8,7,0,3,7,7,8,7
1220 DATA 7,8,8,7,0,5,4,7,7,7
1230 DATA 6,7,8,7,0,6,0,0,0,0
1240 DATA 7,6,7,7,0,0,0,0,0,0
1250 DATA 8,8,8,7,0,6,6,6,6,6
1260 DATA 8,8,8,7,4,7,0,5,3,6
1270 DATA 8,8,8,7,9,1,0,0,0,0
1280 DATA 9,9,9,9,0,0,0,9,0,0
1290 DATA 9,9,9,9,0,0,0,0,0,5
1300 DATA 9,7,1,2,4,0,0,6,6,1
1350 DATA 10,9,7,3,8,4,6,5,2,1
1390 DATA "PENSO QUE ESTA A IR A
LEM DOS LIMITES DO SEU PRECO","E
STA ESPECIFICACAO PARECE ESTAR A
 EXCEDER O SEU LIMITE DE CREDITO
```

```
" ,"NAO PENSO QUE POSSA DAR-SE A
TAIS LUXOS"
1400 DATA "DESCULPE-ME, ESTOU A
OUVIR O TELEFONE A TOCAR" ,"TENHO
 UMA ENTREVISTA URGENTE" ,"FECHAM
OS DENTRO DE CINCO MINUTOS"
1440 DATA "SE ESTA NO MERCADO DE
 ORCAMENTOS, ENTAO QUE SE PASSA
ACERCA DE" ,"UMA ESCOLHA NAO DISP
ENDIOSA E" ,"VOCE OBTEM UM BOM V
ALOR PARA O SEU DINHEIRO COM"
1450 DATA "SE DESEJA UM PRODUTO
DE PRIMEIRA CLASSE DEVE ENTAO EX
PERIMENTAR" ,"POR CAUSA DA SITUAC
AO DA TECNOLOGIA DA ARTE VOCE NA
O PODE BATER" ,"SE QUER UM ROLLS-
ROYCE, DE ENTÃO ATENCAO A"
2000 LET PH=INT (RND*QP)+1: REST
ORE 1030: FOR N=1 TO PH: READ R$
: NEXT N
2100 LET I$=R$: LET J$="/": LET
FT=1: GO SUB 10: RESTORE 1010: F
OR N=1 TO 8: READ Q$,CR,PR: NEXT
 N
2200 IF SP<>0 THEN IF Q$(1)="@"
THEN LET R$=R$( TO SP-1)+"SAO"+R
$(SP TO )
3000 IF SP<>0 THEN IF Q$(1)="&"
THEN LET R$=R$( TO SP-1)+"E"+R$
(SP TO )
4000 LET I$=R$: LET J$="*": LET
FT=1: GO SUB 10: IF SP=0 THEN GO
 TO 4400
4200 LET R$=R$( TO SP-2)+" "+Q$(
2 TO )+R$(SP+1 TO ): GO TO 5000
4400 LET R$=R$+" "+Q$(2 TO )
5000 PRINT : PRINT : PRINT R$: I
NPUT N$: PRINT N$
5010 LET LD=0: LET TC=0: LET TP=
0: LET LD=1: LET OF=0: LET FS=1:
 LET NP=0: LET RU=0: LET M=0: LE
T OM=0: LET S1=0: LET S2=0
5020 LET I$=N$: LET J$=" ": LET
FT=1: GO SUB 10: LET CM=SP
5030 IF CM=0 THEN GO TO 5110
5040 LET I$=N$: LET J$="E": LET
FT=1: GO SUB 10: LET S1=SP
5050 LET I$=N$: LET J$="MAS": LE
T FT=1: GO SUB 10: LET S2=SP
5060 IF S1+S2=0 THEN GO TO 5100
5070 IF N$( TO 2)<>"NAO" THEN GO
 TO 5090
5080 LET P(Q)=P(Q)-1: LET I(Q)=I
(Q)-CR: LET J(Q)=J(Q)-PR: GO TO
5100
5090 LET P(Q)=P(Q)+1: LET I(Q)=I
(Q)+CR: LET J(Q)=J(Q)+PR
5100 LET N$=N$(CM TO )
5110 LET I$=N$: LET J$="SIM": LE
T FT=FS: GO SUB 10
5120 IF SP>0 THEN LET RU=RU+1: L
ET LD=1: LET M=1: LET FS=SP+1: G
O TO 5110
5130 LET I$=N$: LET J$="NAO": LE
T FT=FS: GO SUB 10
5140 IF SP>0 THEN LET LD=-1: LET
 M=1: LET FS=SP+1: LET NP=NP+1:
GO TO 5130
5160 LET I$=N$: LET J$="N'T": LE
T FT=FS: GO SUB 10
5170 IF SP>0 THEN LET LD=-1: LET
 M=1: LET FS=SP+1: LET NP=NP+1:
GO TO 5160
5180 RESTORE 1000: FOR N=0 TO DL
5190 LET I$=N$: READ J$: LET FT=
1: GO SUB 10
5200 IF SP>0 THEN LET LD=LD*-1:
LET M=1: LET NP=NP+1
5210 NEXT N
6000 RESTORE 990: FOR N=0 TO LI
6010 LET I$=N$: READ J$: LET FT=
1: GO SUB 10: IF SP=0 THEN GO TO
 6030
6020 LET LD=LD*1: LET M=1
6030 NEXT N
6040 IF NP=0 THEN GO TO 6100
6050 IF INT (NP/2)=NP/2 THEN LET
 RU=RU+1: LET LD=1: GO TO 6100
6060 LET RU=RU-1: LET LD=-1
6100 RESTORE 940: FOR N=1 TO 9B
6110 LET I$=N$: READ J$: LET FT=
1: GO SUB 10
6120 IF SP>0 THEN LET RU=RU+LD:
LET OF=N: LET M=1: LET OM=OM+1
6130 NEXT N
6200 RESTORE 950: FOR N=0 TO AU+
AJ
6210 LET I$=N$: READ J$: LET FT=
1: GO SUB 10: IF SP=0 THEN GO TO
 6230
6220 LET RU=RU+LD: LET M=1
6230 NEXT N
6300 RESTORE 950: FOR N=0 TO NU+
NJ
6310 LET I$=N$: READ J$: LET FT=
1: GO SUB 10: IF SP=0 THEN GO TO
```

```
 6330
6320 LET LD=LD*-1: LET RU=RU+LD:
 LET M=1
6330 NEXT N
6400 RESTORE 1032: FOR N=0 TO HM
6410 LET I$=N$: READ J$: LET FT=
1: GO SUB 10: IF SP=0 THEN GO TO
 6440
6420 LET XX=N: IF XX<2 THEN PRIN
T "BARATO E MAU": GO TO 6500
6430 PRINT "BASTANTE DISPENDO50"
: GO TO 6450
6440 NEXT N
6500 IF M<1 THEN PRINT "DESCULPE
-ME, MAS": GO TO 2000
6510 IF OM>2 THEN PRINT "ESPERE
- UMA COISA DE CADA VEZ!!!": GO
TO 2000
6520 IF OF>0 THEN LET P(OF)=P(OF
)+RU: LET I(OF)=I(OF)+(CR*RU): L
ET J(OF)=J(OF)+(PR*RU): GO TO 65
50
6540 LET P(Q)=P(Q)+RU: LET I(Q)=
I(Q)+(CR*RU): LET J(Q)=J(Q)+(PR*
RU)
6550 FOR N=1 TO R: PRINT TAB (N-
1)*3;P(N);" ";: LET TC=TC+I(N):
LET TP=TP+J(N): NEXT N: PRINT :
PRINT : FOR N=1 TO R: PRINT TAB
(N-1)*3;I(N);" ";: NEXT N: PRINT
 : PRINT : FOR N=1 TO R: PRINT T
AB (N-1)*3;J(N);" ";: NEXT N: PR
INT
6600 IF TC>BB THEN LET PT=RND*CS
+0.5: RESTORE 1390: FOR N=0 TO P
T: READ Y$: NEXT N: PRINT Y$
6610 IF TP(Q*5 THEN LET TX=RND*E
X+0.5: RESTORE 1400: FOR N=0 TO
TX: READ X$: NEXT N: PRINT X$
6620 IF TC>BB THEN LET PT=RND*CS
: RESTORE 1390: FOR N=0 TO PT: R
EAD Y$: NEXT N: PRINT Y$
7000 LET P$="": FOR X=9 TO 0 STE
P -1: RESTORE 1100+(Q*10): FOR N
=1 TO CO: READ F: IF F-P(Q)=X TH
EN LET P$=P$+STR$ (N)
7010 NEXT N: IF LEN (P$)<=3 THEN
 NEXT X
7200 LET TS=0: LET BS=11: FOR Z=
1 TO LEN (P$): LET PS=VAL (P$(Z)
): LET CT=VAL (D$(PS)): IF CT>TS
 THEN LET TS=CT: LET HI=PS
7220 IF CT<BS THEN LET BS=CT: LE
T LO=PS
7230 NEXT Z: RESTORE 1100: IF HI
=LO THEN PRINT "A SUA UNICA OPCA
O E'": FOR Z=1 TO HI: READ Z$: N
EXT Z: PRINT Z$: GO TO 9000
7250 LET SE=INT ((RND*3)+1): LET
 SL=INT (RND*3)+1: IF SE=2 THEN
GO TO 7290
7280 RESTORE 1450: FOR Z=1 TO SL
: READ Z$: NEXT Z: PRINT Z$,: RE
STORE 1100: FOR Z=1 TO HI: READ
Z$: NEXT Z: PRINT Z$: GO TO 9000
7290 RESTORE 1440: FOR Z=1 TO SL
: READ Z$: NEXT Z: PRINT Z$,: RE
STORE 1100: FOR Z=1 TO LO: READ
Z$: NEXT Z: PRINT Z$
9000 LET Q=Q+1: IF Q<FE+1 THEN G
O TO 2000
9010 STOP
9999 LET R=19: LET FE=0: LET P=F
E: LET OB=P-1: LET LI=3: LET DL=
3: LET AJ=8: LET AV=4: LET NJ=8;
 LET NV=1: LET HM=3: LET QP=5: L
ET BB=100: LET CO=9: LET CT=8: L
ET CS=2: LET EX=2: LET HI=2: LET
 LO=2: LET Q=1: DIM P(R): DIM I(
R): DIM J(R): LET D$="9862735419
": GO TO 2000
```


Comentário

Linha 9999:	Contém a rotina de inicialização.
Linhas 10-20:	Contêm uma rotina INSTR.
Linha 940:	Contém palavras-chave.
Linha 950:	Contém adjectivos.
Linha 955:	Contém advérbios.
Linha 960:	Contém adjectivos negativos.
Linha 980:	Contém verbos negativos.

Linha 990: Contém aprovações.

Linha 1000: Contém desaprovações.

Linha 1010: Contém palavras de questões e as suas taxas de custo e lucro.

Linha 1030: Contém declarações de questões.

Linha 1032: Contém palavras de baixo custo.

Linha 1033: Contém palavras de elevado custo.

Linha 1100: Contém nomes de computadores.

Linhas 1110-1350: Contêm valores de características para cada computador.

Linha 1390: Contém mensagens de taxas de crédito.

Linha 1400: Contém desculpas.

Linha 1440: Contém mensagens económicas.

Linha 1450: Contém mensagens económicas.

Linhas 2000-2440: Recolhe as palavras a serem utilizadas na próxima questão, e selecciona a conjugação correcta.

Linhas 5000-5010: Instala as suas variáveis de INPUT (entrada) e RESET (reinicialização).

Linhas 5020-5030: Verificação para uma vírgula.

Linhas 5040-5060: Verifica «E» e «MAS» e, se nenhum destes casos se encontrar presente, o programa salta para a linha 5100.

Linhas 5070-5080: Actualiza a regra actual negativamente se «E» ou «MAS» se encontrarem presentes e a primeira palavra for «NÃO».

Linha 5090: Actualiza a regra actual positivamente se «E» ou «MAS» se encontrarem presentes e a primeira palavra não for «NÃO».

Linha 5100: Elimina o que existir antes de uma vírgula.

Linhas 5110-5170: Verifica «SIM», «NÃO», actualizando em concordância a regra actual.

Linhas 5180-5210: Verifica as «desaprovações».

Linhas 6000-6030: Verifica as «aprovações».

Linha 6040: Verifica a ocorrência de nenhuma negação.

Linha 6050: Verifica uma dupla negativa.

Linhas 6100-6330: Verifica objectos (complementos directos), adjectivos e advérbios.

Linhas 6400-6440: Verifica correspondências por combinação de valores para palavras-chave de baixo e elevado custo.

Linha 6500: Verifica a ocorrência de nenhuma correspondência e faz um relatório.

Linha 6510: Faz a verificação para mais de um complemento directo.

Linhas 6520-6540: Actualiza a regra actual, ou outra regra, de acordo com o facto de o complemento directo (objecto) corresponder ou não à questão actual.

Linha 6550: Imprime as regras e actualiza o custo total e o lucro total nos seus respectivos valores.

Linha 6600: Imprime um aviso caso a despesa seja demasiado elevada.

Linha 6610: Imprime uma deseulpa caso o lucro pareça ser muito baixo.

Linhas 7000-7010: Procura computadores que satisfaçam as suas necessidades.

Linhas 7200-7210: Recolhe as máquinas de mais alto ou mais baixo preço que correspondam às suas necessidades e respectivas especificações.

Linha 7220: Averigua se somente uma máquina foi seleccionada.

Linhas 7250-7290: Imprime o nome da máquina com o preço mais alto ou da que tem o preço mais baixo.

Linha 9000: Actualiza a característica a ser verificada e retorna (RETURN) para outra entrada.

O resto é consigo

A inteligência artificial é um assunto fascinante, e pensamos ter-lhe proporcionado informação suficiente para que consiga iniciar-se com as suas próprias experiências nesta área. Certamente que apreciámos o termos feito as nossas próprias pesquisas enquanto elaborávamos este livro, porém começámos a questionar-nos sobre quanto tempo decorrerá antes que alguém projecte um programa dum sistema inteligente especializado que possa escrever livros...