MICROCOMPUTADOR CURSO PRÁTICO

TIRE O MÁXIMO DO MICRO EM 30 SEMANAS

PUBLICAÇÃO SEMANAL ILUSTRADA

8

Cr$ 6.500

I-7000 PCxt

Pouso na Lua • Desenho de circuitos

rioGráfica

MANAUS, ALTAMIRA, MACAPÁ, JI.PARANÁ, PORTO VELHO, RIO BRANCO, SANTARÉM, BOA VISTA, SINOP, ALTA FLORESTA: Cr$ 8.450

MICROCOMPUTADOR CURSO PRÁTICO

TESTE

Depois de ler com atenção todo o fascículo, responda a estas perguntas. Veja a solução no próximo número, junto com um novo teste.

1) Qual foi a principal linguagem que originou o PASCAL?
2) Qual a expressão em inglês que qualifica um micro ou um programa que tem boa interação com o usuário leigo?
3) Qual o recurso disponível no Lotus 1-2-3 e no Symphony para armazenar seqüências de comandos freqüentemente utilizadas?
4) O que significa o símbolo Ω?
5) Qual o nome dos símbolos visuais que representam as diversas opções num menu de tela?

RESPOSTAS DO TESTE ANTERIOR

1) O programa de dump de tela imprime tudo o que estiver sendo mostrado no vídeo.
2) A grande desvantagem dos softwares integrados é que exigem grande quantidade de memória.
3) O principal recurso do BASIC para lidar com erros no programa é o comando ON ERROR GOTO.
4) O LOGO, como o BASIC, é uma linguagem interpretada, e não compilada.
5) Os procedimentos em LOGO que invocam a si mesmos ilustram o princípio da recorrência (recursion).

PLANO DA OBRA

MICROCOMPUTADOR - CURSO PRÁTICO é uma coleção de 30 fascículos de periodicidade semanal. Os 29 primeiros terão, cada um, vinte páginas internas (miolo) e quatro capas, além de uma introdução de dezesseis páginas acompanhando o fascículo 1. O fascículo 30 terá doze páginas e quatro capas, e incluirá o índice geral da obra. Os miolos, encadernados, constituem dois volumes de um curso prático de microcomputação, com seções indicadas por tarjas de diferentes cores.
As duas primeiras capas são descartáveis; as 3.as e 4.as trazem programas de jogos. Encadernadas, formam um volume de cem páginas.

COMO ENCADERNAR

As capas duras, incluindo as guardas, estarão à venda simultaneamente com os fascículos 10 (**Volume 1**) e 18 (**Volume 2** e **Jogos**).
Volume 1 - Deve ser encadernado com os elementos dispostos nesta ordem: guardas; frontispício e introdução (publicados com o fascículo 1); miolos dos fascículos de 1 a 15; e guardas.
Volume 2 - Guardas; frontispício (que virá com o fascículo 30); miolos dos fascículos de 16 a 30; e guardas.
Jogos - Guardas; frontispício e sumário (que constituem as 3.as e 4.as capas do fascículo 30); 3.as e 4.as capas dos fascículos de 1 a 29; e guardas.

rioGráfica **Diretoria Executiva:** Oscar Neves; Filipe Zander; Nilo Sérgio de Almeida; Danilo Esteves Costa; **GRUPO EDUCAÇÃO E CULTURA - Planejamento e Comercialização - Gerente de Marketing:** Jaime Rodrigues; **Gerente de Produto:** José Mauro Porto; **Gerente do P.C.P.:** Aylton Menezes; **Diretor de Vendas:** Marcos March; **Gerente de Vendas:** Rubens Barbosa; **Coordenadores de Promoção:** Paulo Cesar M. Seixas; Edgar Mello M. Neto; **Gerente de Circulação:** Norberto Martin; **Supervisor de Operações:** Abel Pereira Pinto - **Editorial - Diretor:** João J. Noro; **Editor-chefe:** Maurício Rittner; **Administração Editorial:** Heitor de Souza Paixão; **Editor:** Jesse Navarro; **Revisores:** Isis Augusta Loyolla; Cacilda Guerra; Luiz Vicente Vieira Filho; Marcos Domingos Agathão; Maria da Graça Mendonça Couto; **Diretor de Arte:** Aníbal Monteiro; **Chefe de Arte:** Bonifácio Duardes Miranda; **Chefe de Estúdio:** José Yuji Kuribayashi; **Assistente de Arte:** Miguel Luis Escámez Simón - **Colaboradores - Editores-assistentes:** Isa Mara Lando; Luiz Carlos Pizarro Marin; Claudio Marcondes; Nadya Morelo Reis; **Texto:** Paulo Brito; **Preparação de Texto:** Luiz Carlos Cardoso; **Tradução:** Fernando Dorfman Knijnik; Lúcia Maria Leal Gonçalves; Maria Clara Cescato; Maria Ester Menezes Martinho; Newton Roberval Eichemberg; Patrícia de Paiva e Castro; Regina Amarante; **Pesquisa:** Stella Maria Quentel; **Arte:** Toshio Morobosi; **Assessoria Técnica:** Geraldo Coen; Gustavo José Ferreira Grubba; Lauro de Lauro; Pierluigi Pazzi; Imarés Microcomputadores. As fotos não creditadas pertencem à obra original. Seguem-se os créditos das demais: Fernando Scavone; Jorge's Estúdio. © APSIF Copenhagen, 1984; © Orbis Publishing Ltd., 1984; © Editora Rio Gráfica Ltda., 1985, para a língua portuguesa. **Composição:** ERG Ltda. e AM Produções Gráficas Ltda. **Impressão:** JBIG.

NÚMEROS ATRASADOS - Você poderá comprar os exemplares que faltam em sua coleção, pelo preço do último fascículo posto à venda, em sua banca de jornal preferida. Caso não consiga obtê-los, faça sua solicitação por carta endereçada diretamente à Editora Rio Gráfica Ltda., rua Itapiru, 1209, Rio Comprido, Rio de Janeiro, CEP 20251. O atendimento será feito por via postal. Nas cidades do Rio de Janeiro e de São Paulo, as compras poderão ser efetuadas pessoalmente nos seguintes endereços: Rio de Janeiro - Rua Itapiru, 1209, Rio Comprido; São Paulo - Rua Frei Caneca, 1152, Consolação.

Distribuidor exclusivo para todo o Brasil: Fernando Chinaglia - Rua Teodoro da Silva, 907, Rio de Janeiro. Distribuidor para Portugal: Electroliber Lda. - Rua Prof. Reinaldo dos Santos, 1488, Lisboa.

ERRATA

MIOLO

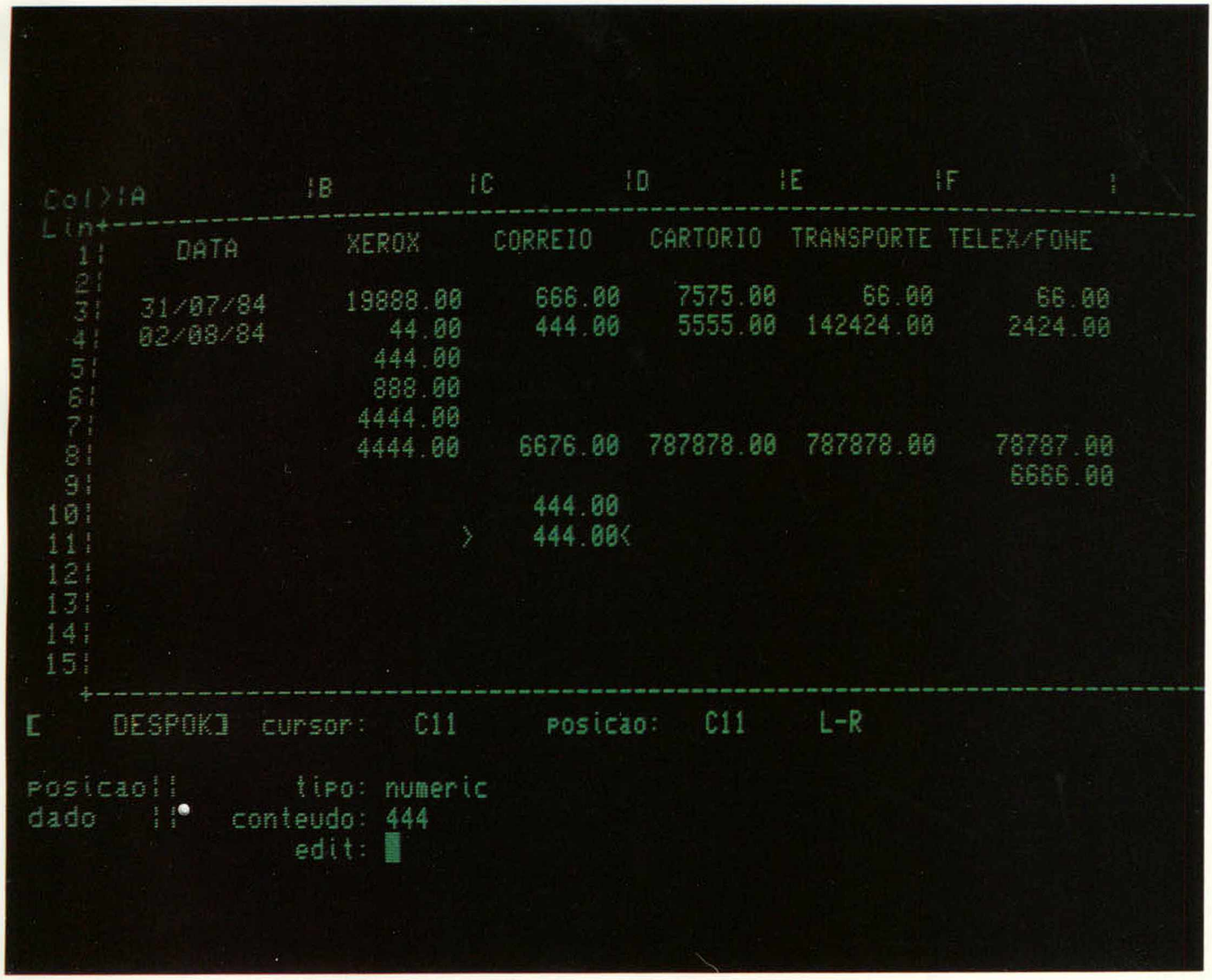
Localização	Correção
Introdução pág. 11, legenda	O TK 2000 **não** integra a linha TRS-80, **mas sim** a linha Apple.
pág. 12, tabela	A RAM básica do IBM PC **não** tem 60 K, **mas sim** 64 K.
pág. 12, tabela	A ROM da linha TRS-80 **não** tem 14 K, **mas sim** 16 K.
Número 1 pág. 25, 1.ª coluna	**Substitua** as linhas 240, 315, 445 e 680 **por:** 240 IF LEN L$<>1 THEN GOTO 220 315 PRINT AT 21,1;" 445 IF C=L THEN GOTO 100 680 GOTO 650*W+50 **Elimine** as linhas de 681 a 689.
Número 2 pág. 45, 17.ª linha	**Substitua** 3 **por** 4 e **acrescente o número** 1020 **após o comando** GOTO.
Número 3 pág. 64, 2.ª coluna, 10.ª e 11.ª linhas	**Onde se lê** O curto programa a seguir foi escrito para o CP 400, **leia-se** O curto programa a seguir e os demais desta página foram escritos para o CP 400 Color.
pág. 71, Registre, 7.ª linha	**Onde se lê** PRINT VAL A$, **leia-se** PRINT VAL (A$).
Número 4 pág. 84, 1.ª coluna, título e subtítulos	**Onde se lê** Respostas dos exercícios 5 — A), B) e C), **leia-se** Respostas dos exercícios 5, 6 e 7.
pág. 87, identificação	**Onde se lê** Unidade de disco, Sony 3 1/2, **leia-se** Unidade de disco Sony 3 1/2". **Onde se lê** conexão analógica digital, **leia-se** conexão analógica/digital. **Onde se lê** Podem ser conectados dispositivos analógicos, digitais, **leia-se** Ligação via cabo flexível entre a placa analógica e a digital.
pág. 87, ficha técnica, item DOCUMENTAÇÃO	**Onde se lê** Os manuais — MacPaint e MacWrite —, **leia-se** Os manuais para os softwares MacPaint e MacWrite.
Número 5 pág. 105, Programa-teste, linha 60	**Substitua** FOR I=3 **por** FOR I=1 TO 3.
pág. 104, Variações	**Considere sem efeito todo o quadro.**

CAPAS

Número 4	**Substitua os números de página** 17, 18, 19 e 20 **por** 16, 17, 18 e 19.
Número 5 Jogo Teste seu Q.I., págs. 21 e 22	**Substitua as linhas** 110, 120, 170, 180, 240, 250, 270, 300, 330, 380, 480, 490, 500 e 560 **por** 110 LET K=INT (RND*100) 120 LET Z=RND 170 LET P=INT (RND*5+1) 180 LET B=RND 240 LET Q=INT (RND*3+1) 250 LET E=RND 270 IF E<.6 AND Q THEN LET Q=1/Q 300 LET N=INT (RND*39+1) 330 LET N=N*P+K+N*Q 380 IF M=N THEN PRINT "***VOCE ACERTOU***" 480 IF T=V THEN GOTO 560 490 GOTO 60 500 PRINT TAB 6;"*** TESTE SEU QI ***" 560 PRINT "JA FIZEMOS O NUMERO DE TENTATIVAS DESEJADAS" **Inclua esta linha:** 15 SLOW

As listagens dos jogos Ataque Aéreo, Invasores do Espaço, Cram e Come-Come serão republicadas completas.

PROCESSADOR DE IDÉIAS



Oferecendo múltiplas possibilidades de uso, a planilha eletrônica funciona como gerador de idéias, e constitui valioso auxiliar no exame e na amostragem de informações.

Como os processadores de texto e os bancos de dados, as planilhas eletrônicas possuem recursos que raramente chegam a ser explorados pelos usuários. A maior parte das pessoas que trabalham com processadores de texto não utiliza seus comandos mais sofisticados; já os bancos de dados tendem a ser usados apenas como índices e sistemas de gerenciamento de arquivos — quase nunca para processamento de dados.

Poucos proprietários de microcomputadores vêem utilidade no uso de planilhas eletrônicas. Quase todos acreditam que programas desse tipo seriam desinteressantes e de utilidade prática restrita. Em geral, ocorre a associação da planilha aos usos financeiros e comerciais — aplicações que quase sempre intimidam os usuários domésticos de microcomputadores.

Subestima-se, assim, a importância da administração financeira do lar, e deixa-se de considerar que a planilha é um processador de idéias cujo emprego ficou comprometido com a imagem de "instrumento de contabilidade". Na verdade, a planilha eletrônica está para as idéias assim como o processador de textos está para a criatividade de quem escreve.

A planilha eletrônica é ao mesmo tempo um processador de texto e uma calculadora. O nome que recebeu deve-se menos à função que desempenha que a seu aspecto, pois divide-se em linhas e colunas, como uma planilha de contabilidade.

Funções combinadas
Uma das grandes vantagens no uso da planilha eletrônica está na possibilidade de realizar projeções. Combinando as funções de calculadora e processador de texto, esse programa permite introduzir os dados nas células com fórmulas e obter os resultados de todas as células recalculados com os novos dados.

Cálculo de notas

Formato
O comando FORMATO foi usado para estabelecer a largura da coluna D, alinhar as células de texto pela esquerda e mostrar os números com duas casas decimais.

Copiar
Com este comando, um bloco de células pode ser copiado em qualquer parte da planilha.

Peso
Multiplica a nota real para produzir uma nota ponderada.

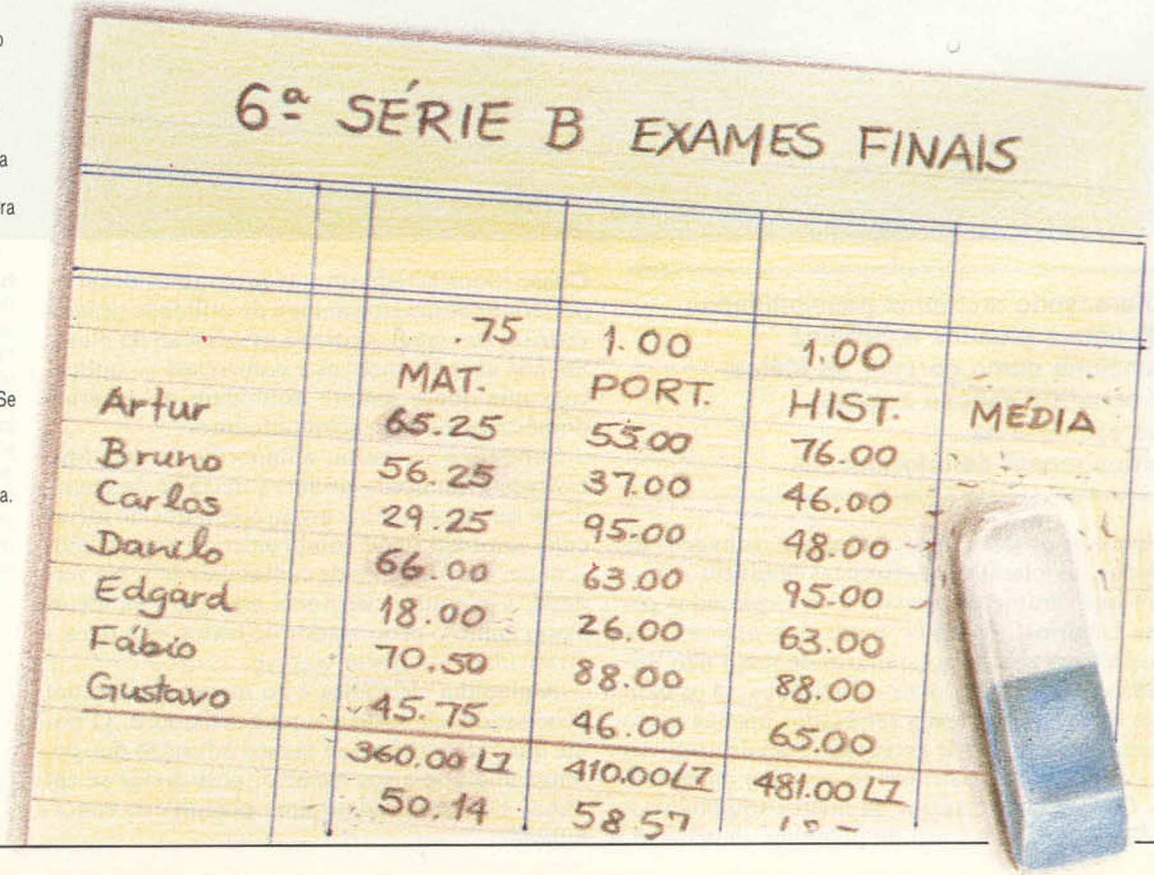
```
    !  C   !D!   E   !!   F   !!   G   !!  H   ! !  J   !!   K   !!   L   !!   M
 1                          AJUSTAMENTO DE NOTAS
 2 ***************************************************************************
 3                  NOTAS REAIS                *        NOTAS PONDERADAS
 4 ***************************************************************************
 5                                             *      .75     .86     .73
 6                  Mat.     Port.    Hist.   MEDIA  *   Mat.     Port.   Hist.   MEDIA
 7 Artur   :        87.00    55.00    76.00   72.67  *   65.25    47.30   55.48   56.01
 8 Bruno   :        75.00    37.00    46.00   52.67  *   56.25    31.82   33.58   40.55
 9 Carlos  :        39.00    95.00    48.00   60.67  *   29.25    81.70   35.04   48.66
10 Danilo  :        88.00    63.00    95.00   82.00  *   66.00    54.18   69.35   63.18
11 Edgard  :        24.00    26.00    63.00   37.67  *   18.00    22.36   45.99   28.78
12 Fabio   :        94.00    88.00    88.00   90.00  *   70.50    75.68   64.24   70.14
13 Gustavo :        61.00    46.00    65.00   57.33  *   45.75    39.56   47.45   44.25
14                  ========================================================================
15  MEDIA  :        66.86    58.57    68.71   64.71  *   50.14    50.37   50.16   50.23
16                                             *
17 ***************************************************************************
```

Texto repetido
Digitando um asterisco nesta célula, o recurso de repetição de texto da planilha preencherá toda a linha com asteriscos.

Cálculo automático
Entra-se com uma fórmula apenas uma vez e copia-se com um só comando a fórmula para outras células.

Média
Calculada por um só comando: MEDIA (célula 1... célula n).

Para comparar o desempenho de seus alunos nas diversas matérias, o professor quer ponderar todos os resultados de exames de forma que a média em cada disciplina seja a mesma. Ele tem de experimentar vários pesos para cada uma, calculando e recalculando as notas — um trabalho tedioso e fadado a erros, que uma planilha eletrônica poderia fazer em minutos. Na planilha, tudo, menos as notas reais, é calculado automaticamente. Se o peso for mudado, resultará, em poucos segundos, nova coluna de resultados ponderados para cada matéria.



Os dados são dispostos em células que, assim como as de uma planilha feita no papel, têm vários empregos. Nessas células, entra-se com textos que serão mostrados na tela ou no papel; com dados numéricos para exibição e cálculo; ou com fórmulas matemáticas que operam com o conteúdo de outras células.

Uma vez que as fórmulas estejam em seu lugar, a planilha transforma-se num programa gerado pelo usuário, pronto para a introdução de dados. Sempre que se introduzem novos dados numéricos, algébricos ou textuais, todas as células com fórmulas são recalculadas, em ordem. Dessa forma, a planilha mantém-se atualizada.

Tais características permitem utilizar a planilha eletrônica para tarefas simples de diagramação de tela e de impressão. No caso, ela facilita a formatação e a impressão não só de cálculos simples — cuja realização dispensaria a ajuda da máquina, se não fossem tão maçantes —, como também de operações complexas, nem imaginadas pelo usuário.

Em muitos casos, o uso da planilha revela algumas aplicações em que o proprietário do micro nunca havia pensado. Incluem-se aí atualização de estoques, análise de resultados esportivos e lotéricos, desenho de formas, produção de mapas de sincronização para som e luz em teatro, cálculo de impostos e até questões que exigem decisão, como optar entre a compra ou o aluguel de um televisor, considerando preço, utilização, desgaste e outros fatores.

Todas essas tarefas poderiam ser programadas por pessoas com conhecimentos de BASIC, mas cada uma delas levaria horas para ser desenvolvida. E a maior parte desse tempo seria despendida na solução e depuração dos infindáveis comandos PRINT TAB, PRINT AT e INPUT, essenciais na formatação das telas. A grande vantagem das planilhas é a possibilidade de formatar a tela à medida que se trabalha na relação entre as variáveis; ou seja, a tela é formatada de maneira simples, como se você estivesse escrevendo as informações numa folha de papel, dispondo textos, dados e resultados de cálculos nos lugares desejados.

Versões e opções

As planilhas eletrônicas possuem vários comandos que facilitam a formatação de tela. Você pode mover, copiar ou eliminar blocos de células, inserir ou eliminar colunas e linhas e definir o formato de uma célula ou bloco quanto a tamanho, justificação (alinhamento em relação aos outros itens da coluna) e posição da vírgula decimal. Na maioria das versões do BASIC, é difícil lidar com esses detalhes; no entanto, eles constituem operações essenciais na produção de relatórios, com as vantagens da facilidade de uso e da boa apresentação visual que oferecem.

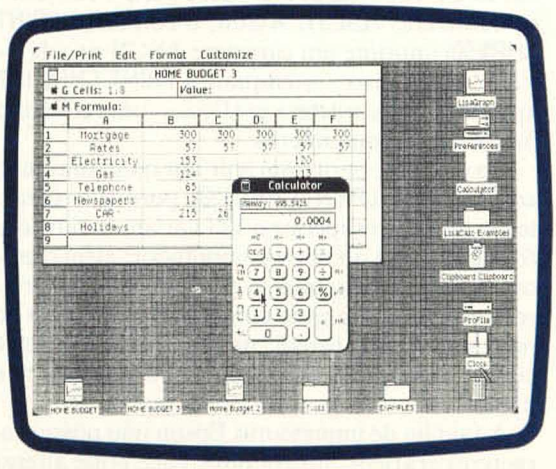
As funções de cálculo são análogas aos recursos para formatação. Com um único comando, calcula-se o valor médio de uma linha ou coluna de dados, contam-se as entradas não zero numa tabela. Somam-se os valores de uma matriz, encontram-se os valores máximo e mínimo numa lista e utilizam-se esses recursos em expressões matemáticas com funções e operadores mais conhecidos, como +, /, SQR (raiz quadrada) e ABS (valor absoluto). Mas nem todas as planilhas eletrônicas oferecem tantas e tão variadas possibilidades. As opções dependem muito da memória disponível no computador e da eficiência do programa.

Um dos comandos de planilha eletrônica mais úteis é o de cópia. Ele permite que fórmulas ou valores introduzidos numa célula sejam duplicados em qualquer número de outras células, de maneira relativa ou absoluta.

Isso possibilita montar, com poucos toques no teclado, projeções anuais e quadros de dados cumulativos — como os juros mensais de hipoteca ou os gastos semanais de uma residência. A programação de uma planilha torna viável a representação de expressões matemáticas complicadas de forma muito mais direta do que o BASIC permitiria.

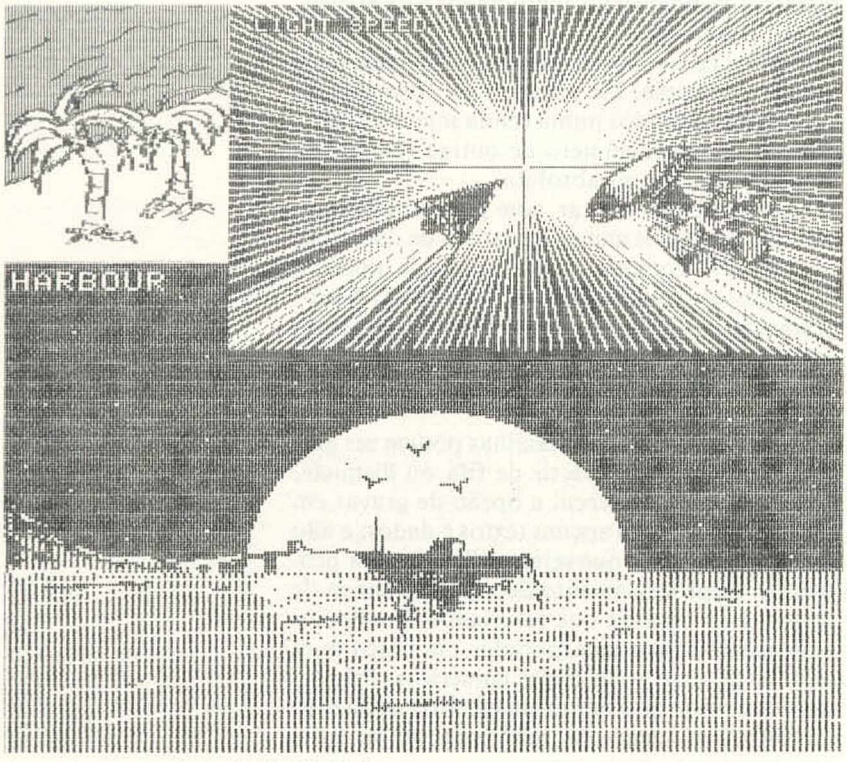
Uma vez prontas, as planilhas podem ser gravadas e acessadas a partir de fita ou disquete. Muitas versões oferecem a opção de gravar em formato de arquivo apenas textos e dados, e não fórmulas, a fim de que sejam utilizados por programas de processamento de texto ou banco de dados. Isso permite que resultados de cálculos e projeções sejam incorporados em bloco num arquivo de texto ou de dados numéricos. Tal possibilidade constitui um passo efetivo em direção aos softwares integrados, mas, em geral, só existe nos programas mais caros.

As planilhas que dispõem de um conjunto razoável de comandos mostram-se tão grandes quanto a imaginação do usuário e a memória disponível no computador. Os próprios programas são, quase sempre, extensos; além disso, as aplicações com tabelas muito longas e os recursos sofisticados de processamento dos dados podem preencher rapidamente o restante da memória disponível. Deve-se levar em conta, ainda, que cálculos mais complicados reduzem a velocidade do programa.



Transferindo dados
No LisaCalc, planilha da Apple para o Lisa, é possível chamar à tela, por meio do mouse, várias "janelas", facilitando assim a transferência de dados de uma célula para outra.

PRIMEIRAS IMPRESSÕES



Impressão de tela
Estes desenhos foram feitos na tela usando-se um tablete gráfico. Depois, o conteúdo da tela foi reproduzido por uma impressora Epson FX-80, mostrando as possibilidades gráficas da impressora do tipo matricial.

A maioria dos usuários desconhece as possibilidades das impressoras matriciais. No entanto, elas permitem a produção de gráficos atraentes, por meio de um programa de dump de tela.

A maioria dos microcomputadores tem um modo gráfico de baixa resolução; isso porque os caracteres gráficos são do mesmo tamanho que os de texto. Os códigos desses caracteres de "bloco" são superiores a 127, uma vez que os números de 0 a 127 ficam reservados para o conjunto de caracteres ASCII. Assim, o comando PRINT CHR$(90) imprime um caractere ASCII na tela — "Z", nesse caso —, enquanto PRINT CHR$(128) apresenta um caractere gráfico — um retângulo preto, em alguns micros.

No entanto, para imprimir um retângulo, não adianta teclar LPRINT CHR$(128), porque os caracteres de código superior a 127 variam muito conforme a marca do microcomputador; além disso, os fabricantes não podem produzir uma impressora especial para cada computador existente no mercado. O que fazem, na maioria das vezes, é copiar o conjunto ASCII nos códigos de 128 a 255.

A família de impressoras Epson não possui caracteres gráficos; no entanto, você pode alterar qualquer dos caracteres ASCII a fim de produzir seus próprios caracteres gráficos. Para tanto, basta enviar códigos adequados à impressora.

Os gráficos de computadores de alta resolução são formados na tela por pequenos pontos (ou pixels) e não por caracteres inteiros. Da mesma forma, a impressão de alta resolução em papel usa pequenos pontos de tinta. A cabeça de impressão numa impressora matricial tem diversas agulhas arranjadas numa linha vertical que se move de um lado para outro do papel enquanto imprime. Em geral, os caracteres são formados por uma matriz de agulhas (muitas vezes de oito por oito pontos). No entanto, é possível produzir gráficos controlando as agulhas uma a uma.

O primeiro passo consiste em ligar a impressora no modo gráfico. Como acontece com qualquer outro tipo de impressão, isso é feito enviando um código específico para o modelo de impressora. Na Epson FX-80, por exemplo, as instruções necessárias são:

```
LPRINT CHR$(27);"K";CHR$(N1);(N2);
```

A letra "K" indica o modo gráfico, e os números (N1) e (N2) determinam a largura de cada linha de gráfico — em outras palavras, o número de pontos, feitos pelas agulhas, que cabem de um lado a outro da página.

No modo gráfico comum, a FX-80 pode imprimir no máximo 480 pontos por linha. Outros modos permitem resoluções que vão de 576 a 1.920 pontos por linha. Portanto, para usar a largura toda, o comprimento da linha deve ser de 480. No nosso código, são necessários dois números para determinar a largura, pois o tamanho máximo de cada número é 255. O segundo número (N2) é, portanto, multiplicado por 256 e acrescentado ao primeiro (N1). Assim, para 480, os números são 1 e 224 (480 = 256 x 1 + 224). Portanto, na impressora Epson FX-80 precisa-se da instrução:

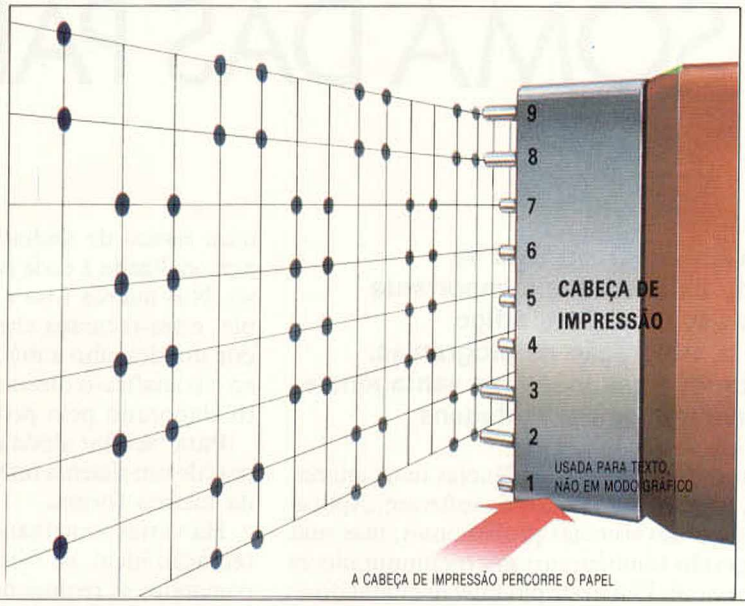
```
LPRINT CHR$(27);"K";CHR$(224);CHR$(1);
```

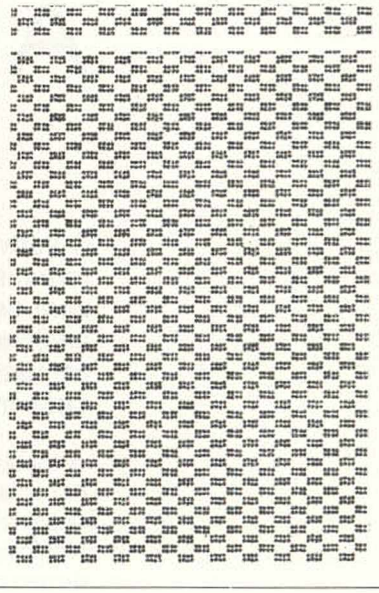
Depois de programar a impressora com o comprimento de linha gráfica, é necessário enviar os dados do gráfico. Mesmo havendo nove agulhas na cabeça de impressão de uma Epson FX-80, só as oito de cima podem ser usadas na maioria dos modos gráficos. Começando pela agulha de baixo, vamos numerá-las: 1, 2, 4, 8, 16, 32, 64 e 128. Os dados para as oito agulhas podem então ser representados por um único número, entre 0 e 255, e isso é enviado à impressora usando-se LPRINT CHR$(X), onde X é o número. Assim, para ativar apenas a agulha de baixo, bas-

Pontilhado
O desenho foi produzido numa impressora do tipo matricial, enviando-se pares alternados dos números decimais 195 e 60 para a cabeça de impressão. O esquema *abaixo* mostra como estes números binários são interpretados pelas agulhas da cabeça de impressão *(ilustração à direita)*. A alimentação controlada do papel faz com que a linha seguinte cubra o espaço deixado pela agulha 1.

NÚMERO DA AGULHA	VALOR BINÁRIO		
9	128	•	○
8	64	•	○
7	32	○	•
6	16	○	•
5	8	○	•
4	4	○	•
3	2	•	○
2	1	•	○
		195	60

• AGULHA AVANÇA
○ AGULHA NÃO AVANÇA





ta mandar a instrução CHR$(1) para a impressora; para acionar somente a de cima, envia-se CHR$(128). Para uma combinação de agulhas, somam-se os números de cada uma. Esse processo é então repetido para cada um dos 480 pontos da linha.

Na ilustração, há dois padrões de agulhas: CHR$(195) e CHR$(60). Para imprimir as quatro primeiras colunas do padrão da linha, digita-se:

```
LPRINT CHR$(195);CHR$(195);CHR$(60);
  CHR$(60);
```

Após quatro colunas, o padrão se repete, e um loop FOR-NEXT elabora o resto da linha.

No exemplo, CHR$(60) não instrui a impressora a imprimir o caractere ASCII com o código 60 — é uma forma de representar os dados para as agulhas na cabeça de impressão. A impressora o reconhece como tal porque anteriormente foi transmitida a seqüência CHR$(27);"K" para acionar o modo gráfico.

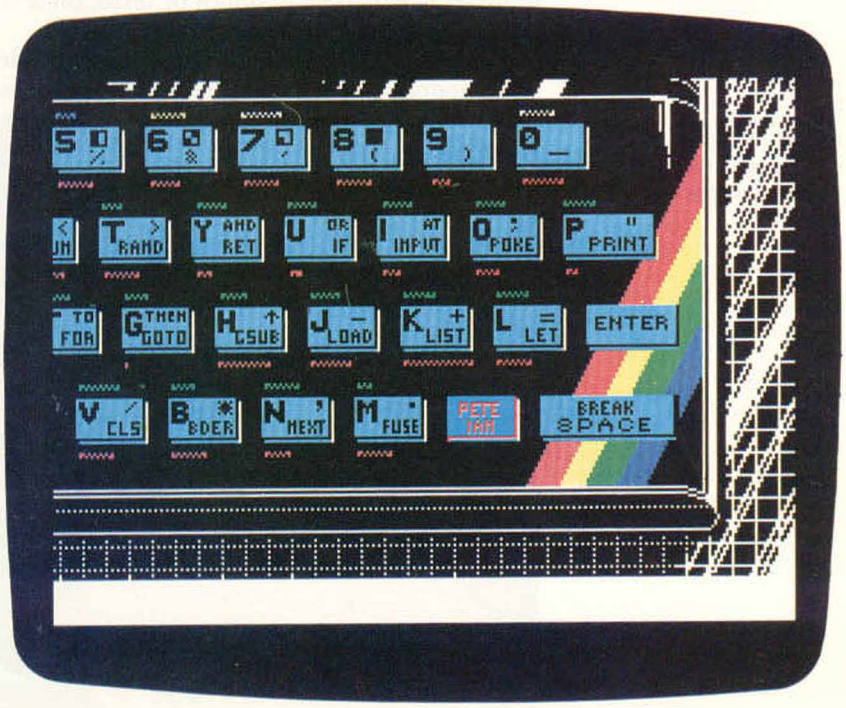
Esse método de impressão por segmentos de imagem é descrito para uma impressora Epson FX-80; outras impressoras usam métodos semelhantes, com pequenas variações. Bastante trabalhosa, a produção de gráficos dessa maneira só é adequada quando há repetição de determinados padrões. Uma maneira mais eficiente de imprimir gráficos utiliza o dump de tela, um programa que copia no papel o que é apresentado no monitor.

Verificando toda a tela no sentido vertical e horizontal, o programa testa se o pixel está ligado em cada posição. Se estiver, deseja-se, então, que uma agulha da cabeça de impressão seja acionada na posição correspondente no papel. Para tanto, usa-se a função POINT(x,y), ou comandos semelhantes existentes na maioria dos micros; se um pixel estiver aceso, então a função POINT(x,y) será 1; se estiver apagado, a função será 0. As diferentes resoluções de tela de micros distintos significam que poderiam ser necessárias algumas adaptações.

Para um programa de dump de tela trabalhar com imagens coloridas, costuma-se atribuir diferentes padrões de pontos para cada cor. Um pixel de tela preto, por exemplo, imprime-se com quatro pontos em forma de quadrado; um vermelho pode ser representado por dois pontos; e um branco, por nenhum. A função POINT(x,y) produz um número diferente, conforme a cor do pixel, e isso também pode ser utilizado.

Os programas de dump de tela são, em geral, acrescentados no final do programa que produz a imagem, em forma de sub-rotinas. Para reproduzir a imagem na impressora, aperta-se a tecla [P], e o programa passa para a sub-rotina. Um programa de dump de tela escrito em BASIC tende a ser lento — leva até 5 min para imprimir uma figura pequena. As versões em código de máquina são ligeiramente mais rápidas.

Colorido
Esta imagem do teclado do Spectrum foi produzida numa impressora que funciona com jatos de tinta colorida — na prática, ela é uma impressora matricial, com jatos de tinta substituindo as agulhas.



A SOMA DAS PARTES

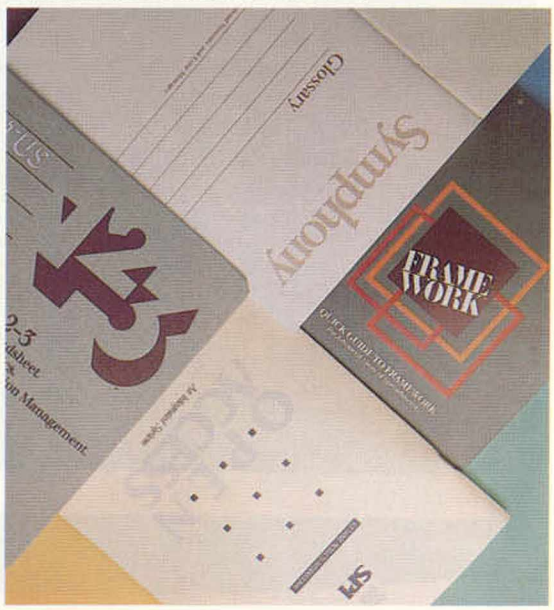
O software integrado está se tornando cada vez mais importante para os usuários. Este artigo analisa a integração de programas, comentando suas inúmeras vantagens e também algumas desvantagens.

A integração é uma das tendências mais animadoras já surgidas em termos de software. Aplica-se sobretudo aos sistemas profissionais, mas suas técnicas estão também nos microcomputadores de uso pessoal. Um exemplo internacional disso é o Sinclair QL, que possui quatro pacotes de software desenvolvidos de acordo com os princípios básicos da integração.

A principal conquista da integração é permitir que o usuário passe de um aplicativo para outro simples e rapidamente. Num sistema ideal, não haveria necessidade de encerrar um programa, retornar ao sistema operacional, trocar os disquetes e só então iniciar outro programa. Para ser eficaz, a troca de aplicativos tem de ocorrer a um toque no teclado, coisa que alguns programas, como o Lotus 1-2-3, o Symphony e o Framework, são capazes de fazer.

A possibilidade de transferir os dados facilmente entre os aplicativos também se mostra muito útil, por exemplo, para criar na planilha eletrônica uma coluna com os números referentes às vendas anuais de sua empresa e então transferi-la para o processador de texto, onde se escreverá um relatório anual.

Você pode usar, por meio do processador de texto, todos os nomes e endereços armazenados num banco de dados para escrever uma carta personalizada a cada pessoa registrada no arquivo. Nos micros Lisa e Macintosh, da linha Apple, esses recursos chegam a permitir que você crie um desenho a mão livre no programa gráfico e transfira-o diretamente para um documento elaborado pelo processador de texto.



Para avaliação
O Lotus 1-2-3, o Framework, o Open Access e o Symphony são bons exemplos de pacotes integrados.

Para facilitar ainda mais as tarefas, os programas de um sistema integrado em geral trabalham da mesma forma.

Há várias semelhanças em seu uso. Numa integração ideal, os formatos de tela, as teclas de comando, as requisições de dados, as mensagens de erro — enfim, todos os diferentes aspectos da interação com o usuário — devem ser idênticos ou, pelo menos, compatíveis. Caso contrário, o usuário não passa de um aplicativo para outro com facilidade.

A integração não seria eficaz se você tivesse de aprender a usar, digamos, cinco aplicativos com comandos de formatos diferentes. Mas se eles trabalharem da mesma forma, você só precisará aprender a usar um. Essa característica, conhecida como "padronização", vincula-se ao conceito de software integrado.

Lucros e perdas

Em síntese, o software integrado envolve três princípios básicos: a facilidade de passar de uma aplicação para outra, a possibilidade de permutar dados entre os programas e a padronização dos formatos. A integração contribui para tornar o computador mais acessível ao usuário médio, cujas necessidades são satisfeitas por dois ou três aplicativos. A integração tem aumentado a popularidade dos microcomputadores, que, graças a ela, se tornam mais eficientes e de uso menos complicado.

No entanto, os softwares integrados também têm suas desvantagens. Uma das principais é a exigência de grandes quantidades de RAM para operar. Às vezes é preciso colocar uma planilha, um processador de texto e um gerenciador de banco de dados — as três aplicações integradas com mais freqüência — em 16 ou 32 Kbytes.

Essa integração é possível, mas talvez não sobre espaço para o armazenamento dos dados. Tal problema restringe o uso dos softwares integrados às máquinas de maior capacidade: computadores com memória de 128 Kbytes ou mais. Os programas podem compartilhar algumas rotinas, que assim só precisam ser escritas uma vez. Contudo, cada aplicação tem suas próprias necessidades, que ocupam espaço na RAM.

Outra desvantagem: para economizar o espaço de memória requerido pelo programa, os pro-

gramadores de software precisam reduzir os recursos dos aplicativos. Um processador de texto integrado num pacote quase nunca é tão completo quanto um processador que opera sozinho, principalmente se este ocupa tanto espaço de memória quanto um pacote integrado.

Um bom exemplo disso são dois programas que rodam no IBM PC e em micros compatíveis. Um deles, o Multimate foi projetado a partir de software utilizado para processamento de textos: tem várias opções para criar e formatar textos, que não podem ser encontradas em programas menores e que tornam relativamente simples a elaboração de documentos longos e complexos. Sozinho, o Multimate exige 192 bytes para operar.

O outro é o Lotus 1-2-3, software que integra processador de texto, planilha e gerenciador de banco de dados e também ocupa 192 bytes. Ou seja: o processador de texto do Lotus tem apenas funções básicas, quando comparado ao Multimate.

Há uma terceira desvantagem na integração. É importante que os programas integrados se pareçam, para que seu uso seja fácil de aprender. Mas isso só é possível quando os programadores fazem algumas concessões. A melhor forma de operar uma planilha pode não ser a melhor para se usar os outros aplicativos. Os programadores precisam, então, combinar de modo eficiente os recursos mais apropriados a todos os programas.

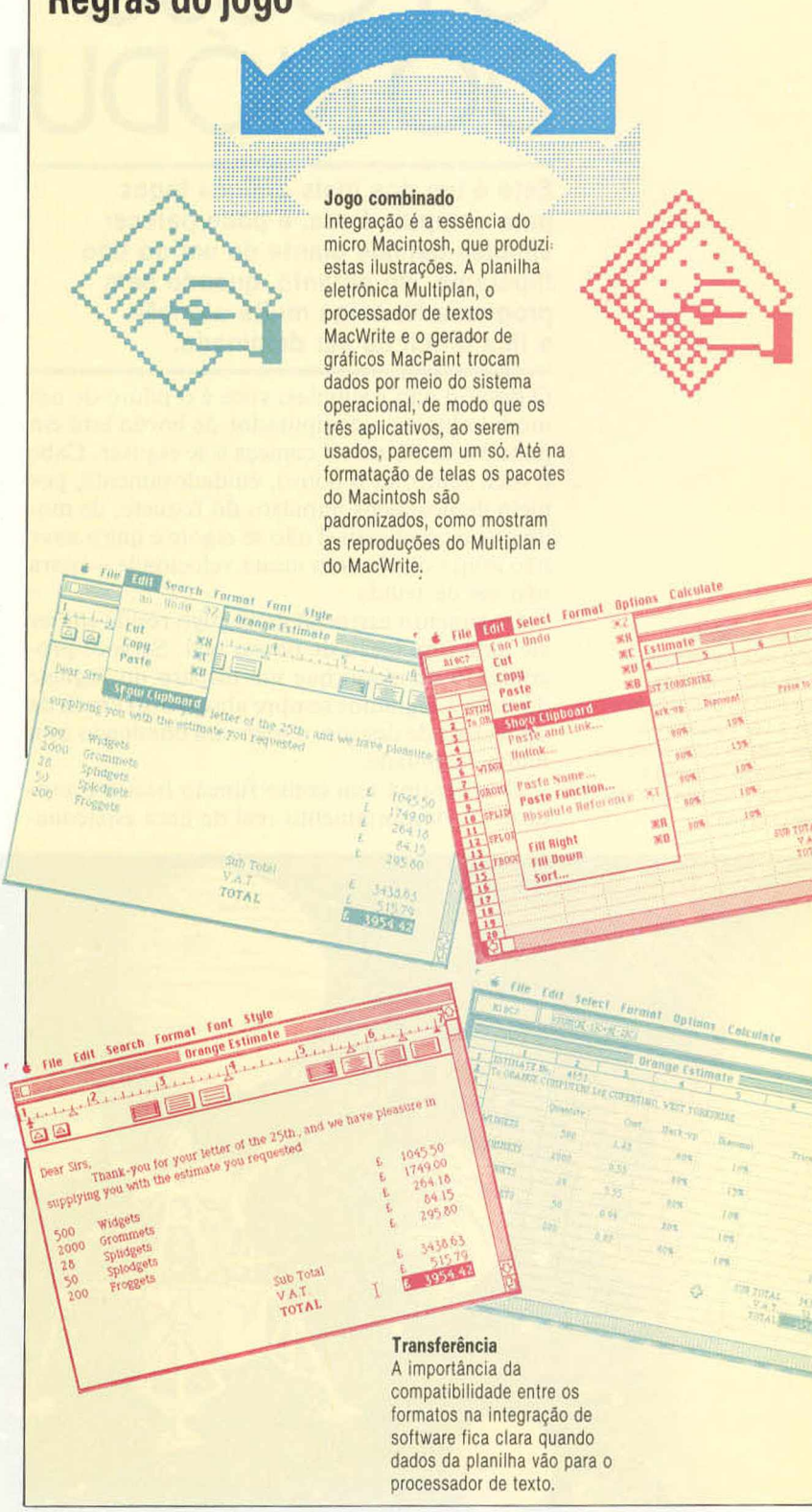
A Microsoft deparou-se com esse problema no projeto do processador de textos Word. A empresa queria que suas telas e sua operação fossem compatíveis com a planilha Multiplan (seu maior sucesso comercial), para que ambos pudessem ser facilmente integrados. Assim, a Microsoft incluiu no Word a mesma tela de menu que os usuários da planilha Multiplan haviam achado prática; só que não demorou muito para se chegar a esta conclusão: quem precisa de um programa como o Word não gosta de ter um menu na tela o tempo todo.

No lar e no trabalho

O ponto mais importante a ser lembrado em relação a softwares em geral é que eles devem fazer aquilo que o usuário quer. Se uma pessoa precisa executar várias tarefas (como cartas, mala direta e cálculos simples), o software integrado torna o trabalho muito mais fácil. Mas, caso o usuário pretenda escrever um romance ou elaborar relatórios empresariais muito longos, deve continuar utilizando programas separados, ou seja, processadores de texto, planilhas e gerenciadores de bancos de dados independentes.

Apesar disso, à medida que os programadores de software forem conseguindo comprimir instruções de programação em espaços cada vez menores, e a capacidade de memória dos equipamentos for se ampliando, o software integrado vai se tornar cada vez mais importante para os usuários, em casa e nos negócios.

Regras do jogo



Jogo combinado
Integração é a essência do micro Macintosh, que produziu estas ilustrações. A planilha eletrônica Multiplan, o processador de textos MacWrite e o gerador de gráficos MacPaint trocam dados por meio do sistema operacional, de modo que os três aplicativos, ao serem usados, parecem um só. Até na formatação de telas os pacotes do Macintosh são padronizados, como mostram as reproduções do Multiplan e do MacWrite.

Transferência
A importância da compatibilidade entre os formatos na integração de software fica clara quando dados da planilha vão para o processador de texto.

O SUPERVISOR

Sistema operacional ou **operating system:** software que controla e supervisiona todas as operações internas de um computador.

O POUSO DO MÓDULO

Este é um dos mais antigos jogos para computadores, e pode parecer simples demais diante de um do tipo fliperama. No entanto, quando bem programado, exige muita atenção e fica difícil de ser dominado.

O desafio não é simples: você é o piloto de um módulo lunar, o computador de bordo está em pane e o combustível começa a se esgotar. Cabe a você controlar o pouso, cuidadosamente, por meio de pequenos impulsos do foguete, de modo que o combustível não se esgote e que a nave não atinja o solo com muita velocidade — para não ser destruída.

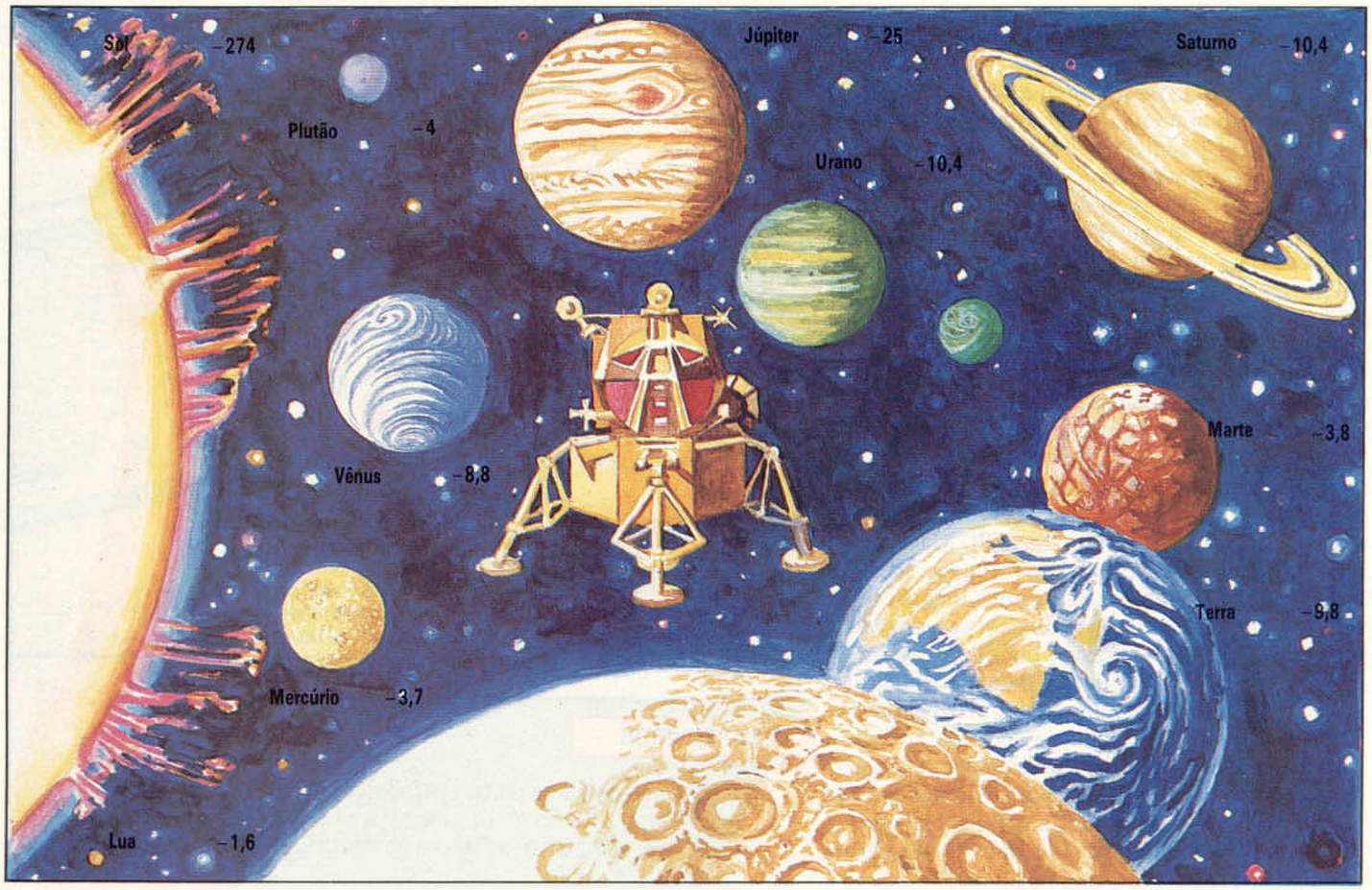
O elemento essencial desse jogo reside em ser ele uma simulação de pouso real. Se você programá-lo de modo que um disparo do foguete durante 2 segundos sempre absorva 10 km/h da velocidade de descida, o jogo será dominado com muita facilidade.

O programa tem como função básica reproduzir o comportamento real de uma espaçonave, em condições tão precisas quanto possível. A matemática necessária para isso envolve cálculos muito complicados; portanto, o programa apresentado aqui é uma versão simplificada. Ainda assim, ele tem muitas das características de uma simulação autêntica.

Vamos ver em detalhes as condições para o pouso:

- O planeta no qual você está descendo tem determinada força de gravidade. Isso fará com que a espaçonave ganhe velocidade à medida que se aproxime dele.
- A espaçonave tem foguetes para contrabalançar os efeitos da gravidade empurrando o veículo no sentido contrário.
- A espaçonave é dotada de massa (e, sob gravidade, tem peso). Quanto maior essa massa, menor o efeito dos foguetes. Ao peso da espaçonave deve-se acrescentar o do combustível que ela transporta. À medida que o combustível é consumido, o módulo vai ficando mais leve.

Força da gravidade
Abaixo você encontra os valores aproximados da aceleração da gravidade para o Sol, a Lua e planetas de nosso sistema solar. Esses são os valores da variável g e devem ser colocados na linha 30 do programa.



Assim, para traduzir matematicamente a maneira pela qual nosso veículo se comporta, precisamos de um conjunto de equações que envolvem aceleração, massa, velocidade etc.

Se elas vão ser muito simples ou muito complicadas, depende do grau de precisão e do número de pormenores que desejamos levar em conta. Mas, para os propósitos de nosso jogo, vamos manter esses fatores num nível razoavelmente simples.

A informação mais importante de que precisamos é a altitude em que a nave se encontra. Ela se move continuamente, seja caindo sob efeito da gravidade, seja indo para longe do planeta devido ao uso exagerado dos foguetes. Para que possamos saber sua posição em determinado instante, dividimos o tempo numa série de passos — ou períodos.

Em cada período, podemos calcular o espaço percorrido pelo módulo, as mudanças no valor de sua velocidade, seu peso etc. Esses períodos podem ter a duração que você quiser — quanto menores, mais precisa a simulação.

A velocidade é medida em unidade por hora, o que se reduz à fórmula:

distância = tempo x velocidade

Podemos, assim, calcular a distância que o módulo percorre, para cima ou para baixo, multiplicando sua velocidade pela duração do período (que definimos como unidade). Dessa forma, ajustamos a velocidade do módulo acelerando-o com a ajuda da gravidade do planeta e desacelerando-o por meio dos foguetes.

A aceleração da gravidade, sempre constante, corresponde à variável g, no programa. Ela difere em cada planeta onde você irá pousar. A ilustração mostra os valores de g para os planetas de nosso sistema solar, mas você poderá experimentar outros valores, ou fazer o programa gerar para g um valor aleatório, produzindo assim um jogo mais difícil.

Simular o motor do foguete é um pouco mais complicado. Na versão que apresentamos, o jogador consome de uma a nove unidades de combustível por período; o programa calculará a aceleração resultante, levando em conta a massa do veículo. Na vida real, a fórmula exata depende da potência do motor do foguete e do tipo de combustível utilizado. Em nosso programa, os valores foram escolhidos de maneira a tornar o jogo difícil; no entanto, você pode tentar alterá-los para ver como eles afetam o resultado.

Um detalhe acrescentável é a possibilidade de inserir novos dados em tempo real, ou seja, sem que o programa pare de rodar. Essa é, muitas vezes, a única diferença que transparece entre o uso de uma instrução INPUT (que interrompe a execução do programa a fim de reunir informações que o usuário forneça) e uma instrução INKEY$ ou GET.

O jogo ficará melhor se não tiver de ser interrompido para calcular quanto combustível deve ser queimado. Caso isso aconteça, o jogador terá tempo para examinar a situação e fazer alguns cálculos — ao passo que, na vida real, isso exigiria raciocínio muito rápido.

Nesse programa de pouso, um loop é executado todas as vezes que se alcança a linha 250. Ajustando-se o período de modo a igualar o tempo gasto na execução do loop, a simulação vai funcionar em tempo real: a alunissagem, na simulação, vai durar tanto quanto uma real.

Embora numa simulação isso seja desejável, pode ser muito difícil de conseguir num jogo como este. Em geral, a simulação é muito demorada para apresentar interesse como jogo.

Elementos extras

Existem muitas coisas que você pode fazer em seu programa básico de alunissagem. A mais óbvia delas consiste em acrescentar alguns detalhes gráficos para descida.

As idéias para colocar isso em prática variam desde um simples altímetro com o mostrador redondo até a paisagem do local de pouso, vista de cima e graduada por uma escala. Você também pode acrescentar movimentos laterais, embora o módulo, normalmente, não se desloque nessa direção.

Porém, se estiver pensando num planeta com atmosfera, você poderá acrescentar, como uma dificuldade a mais, os efeitos de um vento na superfície ou coisa semelhante.

Algumas versões sofisticadas do programa possuem várias áreas de pouso, situadas no fundo de túneis e de crateras, o que exige uma pilotagem cuidadosa.

Alunissagem pelo vídeo

```
  10 REM PARA A LINHA SINCLAIR
  20 REM      POUSO NA LUA
  30 LET G=-1.6
  40 LET T=1
  50 LET C=1000
  60 LET V=0
  70 LET H=2000
  80 LET M=2000+C
  90 LET G=G+T
 100 REM ATUALIZA TELA
 110 PRINT AT 0,9;"POUSO NA LUA"
 120 PRINT AT 2,0;"ALTURA....:";
INT H;"  "
 130 PRINT ,,"VELOCIDADE....:";I
NT V;"  "
 140 PRINT ,,"COMBUSTIVEL....:";
INT C;"  ",,,
 150 IF H<0 THEN GOTO 310
 160 IF C>0 THEN GOTO 190
 170 PRINT ,,"SEM COMBUSTIVEL
     "
 180 GOTO 200
 190 PRINT ,,"POTENCIA DO MOTOR
0-9"
 200 LET B=0
 210 IF C<0 THEN GOTO 240
 220 LET A$=INKEY$
 230 IF A$>="0" AND A$<="9" THEN
 LET B=VAL A$
 240 IF B>C THEN LET B=0
 250 LET H=H+V+T
 260 LET V=V+G
 270 LET V=V+(B+3000)/M
 280 LET C=C-B
 290 LET M=M-B
 300 GOTO 100
 310 REM NA SUPERFICIE DA LUA
 320 IF V>-10 THEN PRINT AT 10,0
;"POUSO PERFEITO...PARABENS"
 330 IF V<=-10 AND V>-20 THEN PR
INT AT 10,0;"CRASH...VOCE ARREBE
NTOU A NAVE  MAS OS TRIPULANTES
SOBREVIVERAM"
 340 IF V>-20 THEN GOTO 370
 350 PRINT AT 10,0;"CRUNCH...ESP
ACONAVE DESTRUIDA...SEM SOBREVIV
ENTES"
 360 PRINT "VOCE ACABA DE CRIAR
UMA NOVA CRATERA DE ";INT (-V*.1
);" KM DE DIAMETRO"
 370 PRINT ,,"JOGA NOVAMENTE ?(S
/N)"
 380 LET A$=INKEY$
 390 IF A$="" THEN GOTO 380
 400 IF A$="S" THEN RUN
 410 IF A$<>"N" THEN GOTO 380
 420 STOP
```

Este programa roda em TK 85, TK 90X e CP 200.

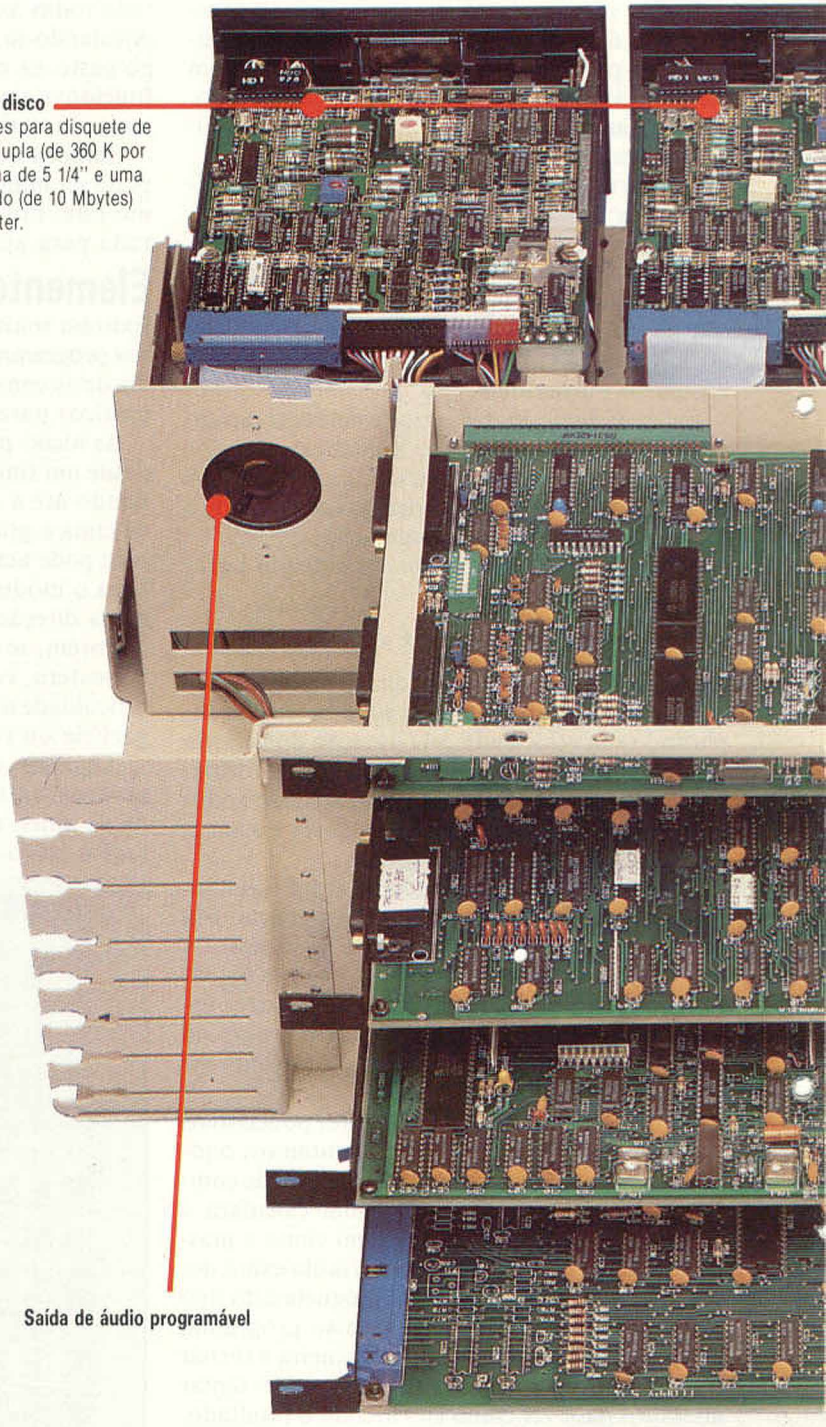
I-7000 PCxt

Incorporando diversos recursos em sua moderna arquitetura de 16 bits, o I-7000 PCxt, da Itautec, oferece alto desempenho e compatibilidade com a maior parte dos aplicativos e linguagens de programação.

Equipado com memória de 256 Kbytes, expansível até 640 Kbytes, duas CPUs de alta velocidade e dois processadores auxiliares, o I-7000 PCxt deve à sua arquitetura de 16 bits o elevado desempenho que o caracteriza. Capaz de processar grande volume de dados em curto espaço de tempo, esse equipamento aceita qualquer programa, periférico ou acessório, compatível com o IBM PC.

Sua alta resolução gráfica, de 640 x 400 pixels endereçáveis na tela, permite criar figuras com grande riqueza de detalhes. Utilizando monitor em cores, trabalha com até dezesseis cores ao mesmo tempo; e, na resolução gráfica máxima, a máquina pode gerar telas compostas de até quatro cores.

O I-7000 PCxt opera automaticamente com os sistemas SIM/M e SIM/DOS, compatíveis com o CP/M e o MS/DOS, recurso responsável pela versatilidade do equipamento: pode ser utilizado com diversos aplicativos e linguagens de programação, além de rodar também programas desenvolvidos para o I-7000 e o I-7000 Jr., da

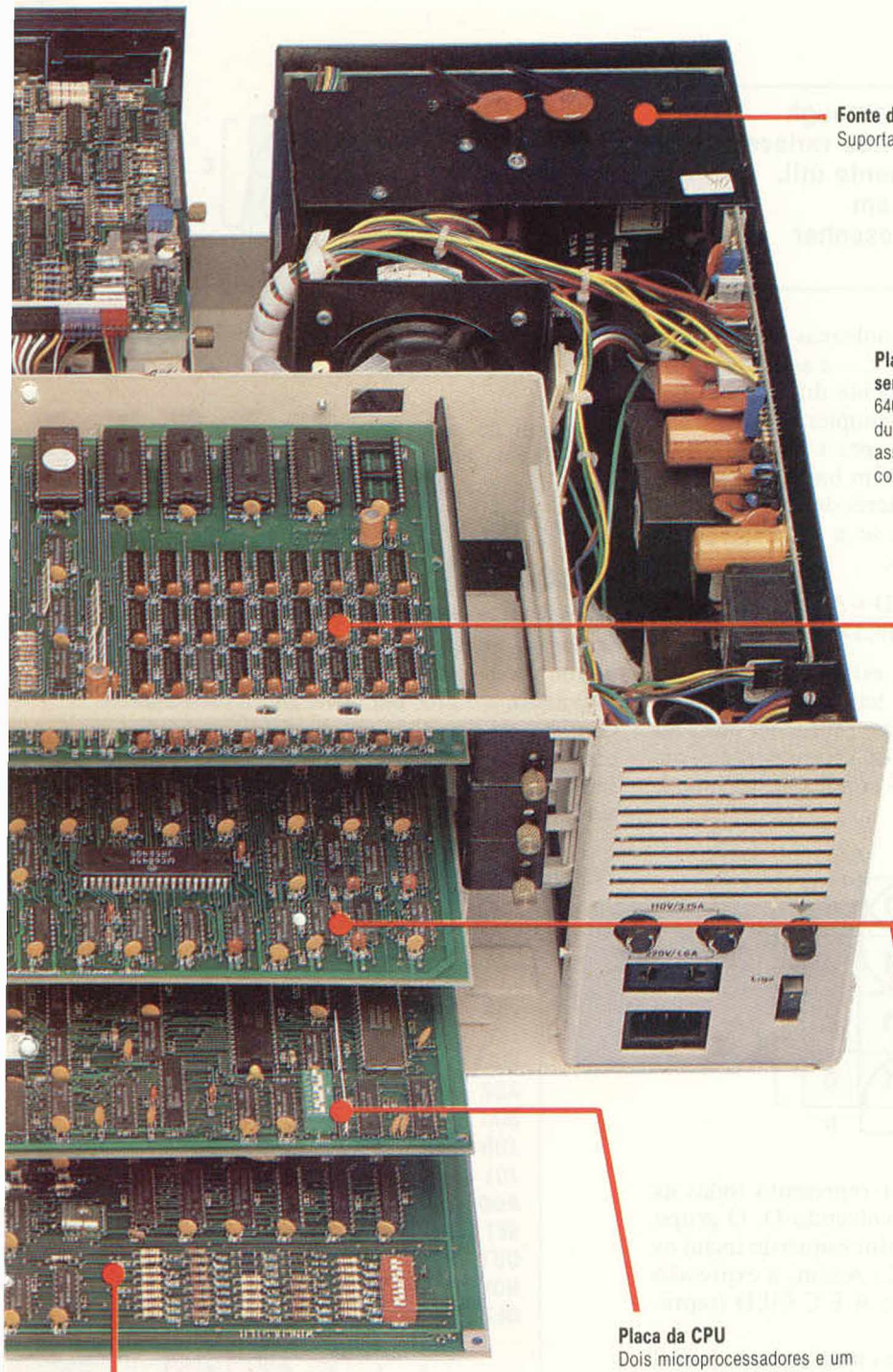


Unidades de disco
Duas unidades para disquete de 5 1/4", face dupla (de 360 K por disco), ou uma de 5 1/4" e uma de disco rígido (de 10 Mbytes) tipo Winchester.

Saída de áudio programável

tautec. Graças às interfaces de comunicação, o I-7000 PCxt tem acesso a computadores de grande porte — IBM, DEC, Burroughs —, o que viabiliza a criação de redes de micros e conexão a sistemas de comunicação de dados como, por exemplo, Cirandão e Videotexto.

O teclado projetado para a língua portuguesa tem inclinação ajustável, é independente e dis-



Fonte de alimentação
Suporta disco rígido.

Placa de memória e interfaces seriais
640 K de RAM e 48 K de ROM; duas interfaces seriais assíncronas RS232C ou loop de corrente.

Placa de vídeo e impressora
Interface paralela padrão Centronics. Controladora de vídeo colorido (dezesseis cores) com capacidade gráfica de 640 x 400 pixels.

Placa da CPU
Dois microprocessadores e um co-processador numérico.

Placa controladora de disquete 5 1/4" e Winchester
Controla quatro disquetes e o disco rígido.

I-7000 PCxt

MICROPROCESSADORES

CPUs: 8088-2 e Z80B.
Auxiliares: 8035 (controlado por teclado) 8087-2 (aritmético de ponto flutuante — opcional).

CLOCK

8088-2: 4,77 ou 8 MHz, selecionáveis por software
Z80B: 6 MHz

MEMÓRIA

256 K de RAM, expansíveis até 640 K;
48 K de ROM.

VÍDEO

Monitor de 12", fósforo verde (oito tons).

SISTEMA OPERACIONAL

SIM/M, compatível com CP/M. SIM/DOS, compatível com MS/DOS.

TECLADO

99 teclas (com todos os caracteres da língua portuguesa), sendo doze de funções programáveis e quatro para controle do cursor. Bloco numérico separado.

LINGUAGENS

BASIC, COBOL, LOGO e outras que rodem sob CP/M ou MS/DOS.

INTERFACES

Saída SASI para disco rígido; saída paralela padrão Centronics; duas saídas seriais padrão RS232C.

DOCUMENTAÇÃO

Manual do usuário, completo e bem ilustrado.

põe de 99 teclas tipo máquina de escrever, doze das quais programáveis e quatro destinadas ao controle do cursor. Há também um bloco numérico em separado.

O equipamento compõe-se de três módulos independentes: módulo básico, teclado e monitor de vídeo. A montagem "em torre" do módulo básico permite colocá-lo no solo, em posição vertical, o que resulta em melhor aproveitamento do espaço, pois libera a mesa do operador.

O I-7000 PCxt conta com duas unidades para disquete de 5 1/4 polegadas, face dupla, ou uma de 5 1/4 polegadas e uma de disco rígido. Todos os elementos da máquina estão contidos em quatro placas de circuito. Há ainda três slots livres para placas de expansão.

MAPAS E CIRCUITOS

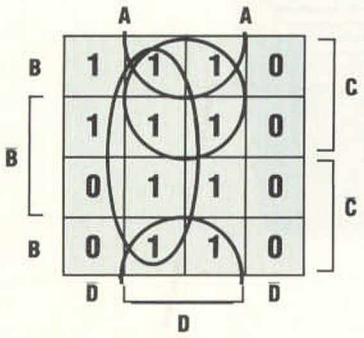
O uso dos mapas de Karnaugh nos casos de mais de três variáveis costuma ser extremamente útil. E os mapas-k simplificam muito o processo de desenhar circuitos eletrônicos.

No caso de expressões booleanas envolvendo quatro variáveis, os mapas-k — e as próprias expressões — são aparentemente difíceis. No entanto, aplicando-se noções simples, estabelecidas quando examinamos os mapas-k de duas e três variáveis, eles logo se tornam bastante familiares e, mais ainda, bem fáceis de manipular.

Por exemplo, suponha-se a necessidade de simplificar esta expressão:

$$ABC\overline{D} + ABCD + \overline{A}BCD + A\overline{B}C\overline{D} + \overline{B}CD + \\ + A\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}D + B\overline{C}D$$

Pode-se perceber que ela exige um mapa-k de quatro variáveis. Embora haja oito componentes na expressão, será preciso preencher dez 1 no mapa-k (os termos $\overline{B}CD$ e $B\overline{C}D$ representam, cada um, dois casos). Nessas condições, o mapa-k assim se apresenta:



O grupo central de oito 1 representa todas as combinações possíveis envolvendo D. O grupo de quatro 1 no canto superior esquerdo inclui os casos que envolvem A E C. Assim, a expressão pode ser simplificada para A E C OU D (representada como $A.C + D$).

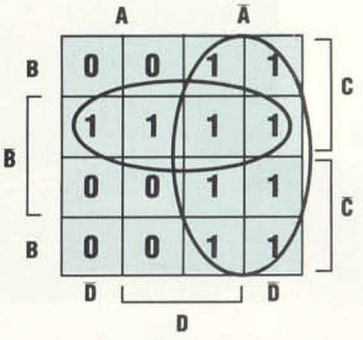
Às vezes é necessária uma manipulação inicial da expressão para dispô-la numa forma adequada para a representação num mapa-k, como no seguinte exemplo:

$$\overline{A + B + C} + \overline{A}.B + \overline{B + C}$$

Aqui devemos aplicar a lei de De Morgan à expressão, antes que seja possível obter o mapa-k. Disso resulta:

$$\overline{A}.\overline{B}.\overline{C} + \overline{A}.B + \overline{B}.C$$

e o mapa-k que essa expressão produz é:



Por meio do mapa-k, essa expressão pode ser simplificada para: $\overline{A} + \overline{B}.C$. Novamente pela lei de De Morgan, simplifica-se para:

$$\overline{A.(B + \overline{C})}$$

Desenho do circuito

Exemplo 1: meses de trinta dias

Suponhamos que cada mês do ano seja indicado num código binário de 4 bits, desde 0001 para janeiro, até 1100 para dezembro. Nossa tarefa consiste em desenhar um circuito que admita o código de 4 bits como entrada e produza uma saída de 1 se o mês que for introduzido tiver trinta dias.

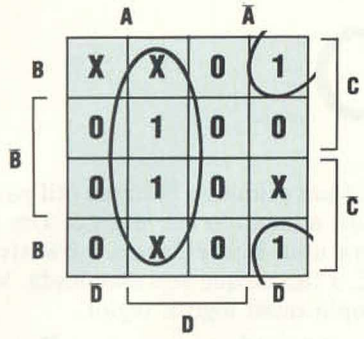
A tabela de validação para um circuito desse tipo será:

MÊS	ENTRADA				SAÍDA
	A	B	C	D	S
	0	0	0	0	X
JAN	0	0	0	1	0
FEV	0	0	1	0	0
MAR	0	0	1	1	0
ABR	0	1	0	0	1
MAI	0	1	0	1	0
JUN	0	1	1	0	1
JUL	0	1	1	1	0
AGO	1	0	0	0	0
SET	1	0	0	1	1
OUT	1	0	1	0	0
NOV	1	0	1	1	1
DEZ	1	1	0	0	0
	1	1	0	1	X
	1	1	1	0	X
	1	1	1	1	X

A saída X da tabela de validação significa uma entrada não válida. Vamos supor que o circuito *não* vá receber tais sinais. Pela tabela de validação e sempre que $S = 1$, podemos formar a seguinte expressão booleana com os bits binários da entrada:

$$S = \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.C.\overline{D} + A.\overline{B}.\overline{C}.D + \\ + A.\overline{B}.C.D$$

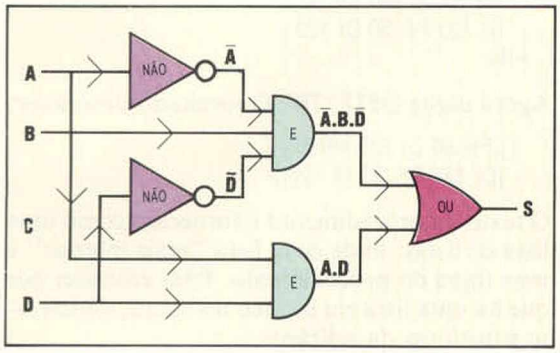
Levando essa expressão a um mapa-k, juntamente com as condições (X) de "entrada não válida", obtemos:



A partir desse mapa-k, podemos ver que a expressão se reduz a:

$$A.D+\bar{A}.B.\bar{D}$$

E assim, nosso circuito de sinal "mês de trinta dias" pode ser construído:



Exemplo 2: números ímpares

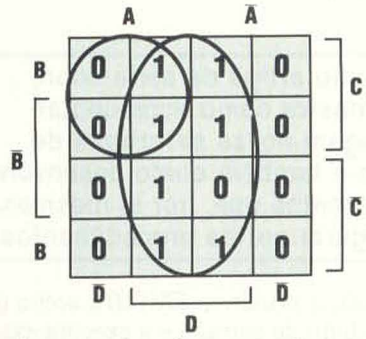
Considerando que os números de 0 a 15 podem ser codificados por quatro dígitos binários (0000 a 1111), desenhar um circuito que admita o código de 4 bits como uma entrada e produza uma saída de 1, sendo a saída um número ímpar maior que 2. A primeira providência é construir uma tabela de validação para todas as condições:

NÚMERO DECIMAL	ENTRADAS A	B	C	D	SAÍDA S
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Com essa tabela de validação, formamos a seguinte expressão da álgebra booleana, para todas as condições em que S é verdadeira (=1):

$$S=\bar{A}.\bar{B}.C.D+\bar{A}.B.\bar{C}.D+\bar{A}.B.C.D+$$
$$+A.\bar{B}.\bar{C}.D+A.\bar{B}.C.D+A.B.\bar{C}.D+$$
$$+A.B.C.D$$

O mapa de Karnaugh para essa expressão é:



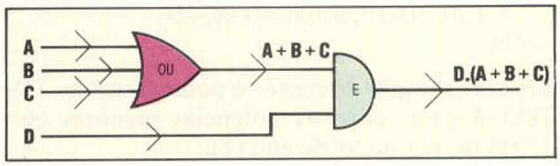
A partir do mapa-k, três grupos de 4 podem ser isolados segundo a expressão

$$S=A.D+C.D+B.D$$

Isso pode ser mais simplificado ainda, por meio da lei distributiva, para obter:

$$S=D.(A+B+C)$$

Conseqüentemente, o circuito resultante é:



O próximo artigo da série contém uma revisão dos aspectos mais importantes da lógica booleana e apresenta exercícios de revisão.

Exercício 5

1) Simplificar as seguintes expressões booleanas, utilizando os mapas de Karnaugh:

a) $A.B.C+A.\bar{B}.\bar{C}+\bar{A}.\bar{C}+\bar{A}.B.C+A.B.\bar{C}$

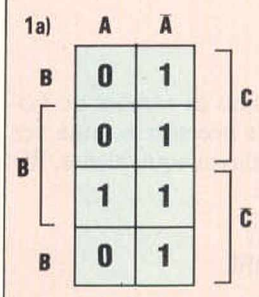
b) $\overline{B+C}+B.\bar{C}+A.C$

c) $A.\bar{B}.D+\bar{A}.D+A.B.C.D+A.B.\bar{C}+\bar{A}.B.\bar{C}.\bar{D}$

2) Desenhar um circuito que admita as representações binárias dos números inteiros entre 0 e 7, inclusive. O circuito deverá resultar numa saída se o número introduzido for ímpar ou se for um múltiplo de 3 (isto é, 3 ou 6). A partir da tabela de validação e da expressão simplificada, desenhe um circuito lógico para essa função.

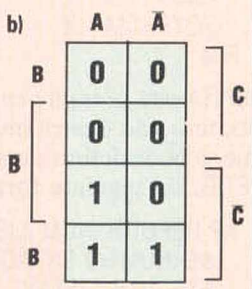
Respostas do exercício 4

1a)



$\bar{A}+\bar{B}.\bar{C}$

b)



$(A+B).\bar{C}$

REPITA O DESEMPENHO

Este último artigo da série sobre o LOGO mostra como acrescentar à linguagem novas estruturas de controle e também como desenvolver procedimentos que, por si mesmos, podem gerar outros procedimentos.

No MLOGO, o primitivo EXECUTE aceita uma lista como dado de entrada e a executa exatamente como se fosse uma linha de procedimento. Emprega-se esse recurso para o acréscimo — quando necessário — de novas estruturas de controle à linguagem. Assim, define-se o procedimento ENQUANTO da seguinte forma:

```
AP ENQUANTO :CONDICAO :ACAO
  SE NAO (EXECUTE :CONDICAO) ENTAO PARE
  EXECUTE :ACAO
  ENQUANTO :CONDICAO :ACAO
FIM
```

Eis um exemplo de como se poderia usá-lo. POTENCIA gera todas as potências menores que 1.000 de seu dado de entrada:

```
AP POTENCIA :X
  FACA "P :X
  ENQUANTO [:P<1000][MOSTRE :P
    FACA "P :P * :X]
FIM
```

Embora comuns em outras linguagens, as estruturas de controle — como ENQUANTO e REPITA — não são realmente necessárias em MLOGO. Um modo mais natural de escrever POTENCIA em MLOGO seria:

```
AP POTENCIA1 :P
  SE NAO :P < 1000 ENTAO PARE
  MOSTRE :P
  POTENCIA1 :P * :I
FIM

AP POTENCIA :X
  FACA "I :X
  POTENCIA1 :X
FIM
```

REPITA está presente em todas as versões do LOGO, mas não é estritamente necessário, uma vez que se pode definir uma palavra equivalente, REPETIR, da seguinte forma:

```
AP REPETIR :NUM :LISTAV
  SE :NUM=0 ENTAO PARE
  EXECUTE :LISTAV
  REPETIR :NUM-1 :LISTAV
FIM
```

EXECUTE é um primitivo bastante útil para o trabalho mais adiantado em MLOGO. Um programa monta uma lista e depois a transfere para EXECUTE, a fim de que seja executada. Veremos um exemplo disso logo a seguir.

Desmontando os procedimentos

Em primeiro lugar, definimos um procedimento para desenhar um triângulo da forma habitual:

```
AP TRI
  FR 50 DI 120 FR 50
  DI 120 FR 50 DI 120
FIM
```

Agora digite LISTE "TRI. O resultado deverá ser:

```
[[][FR 50 DI 120 FR 50]
  [DI 120 FR 50 DI 120]]
```

O texto do procedimento é fornecido como uma lista de listas, onde cada lista "mais interior" é uma linha do procedimento. Para entender por que há uma lista em branco no início, defina este substituto da adição:

```
AP SOMA :A :B
  MOSTRE :A+:B
FIM
```

Agora, obteremos, com LISTE "SOMA:

```
[:A :B][MOSTRE :A+:B]
```

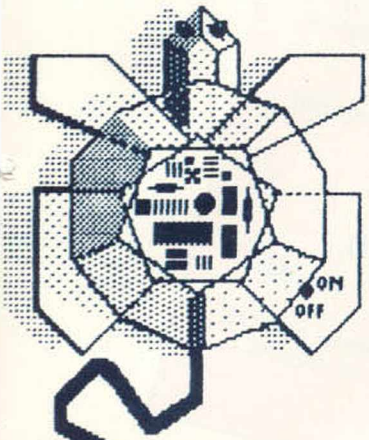
A primeira lista contém os dados para o procedimento. Assim, LISTE permite verificar o que está contido num procedimento. DEFINA, pelo contrário, permite-nos definir um procedimento como uma lista de listas sem que seja preciso utilizar o editor. Tente agora DEFINA "L [[:A] [FR :A][DI 90][FR :A/2]] e então execute L usando, por exemplo, L 30. O uso de DEFINA em modo imediato não apresenta vantagens em relação ao uso do editor, mas apenas na capacidade de um procedimento gerar outro procedimento.

Crescimento

Vamos agora desenvolver um pequeno sistema para o crescimento de figuras. Seus comandos básicos são FAZER, que escolhe a forma com que iremos lidar, e CRESCER, que modifica o tamanho da forma escolhida. Por exemplo, FAZER "QUADRADO desenha um quadrado e então CRESCER [*10] o apaga, desenhando-o novamente com os lados aumentados por um fator 10.

Para simplificar os programas, aceitaremos algumas restrições quanto ao que podemos realizar com esses comandos. Os programas para

Desenhe uma tartaruga
Não se pode avançar muito no LOGO sem a recorrência: algo definido em termos de si mesmo, como os procedimentos que se invocam, as listas definidas por listas e os procedimentos para criar procedimentos. Com um pouco de imaginação, é possível fazer um desenho no estilo de Escher, usando uma tartaruga para gerar outra tartaruga que desenha uma terceira tartaruga...



desenhar formas — fornecidos a FAZER como entradas — não podem conter o comando REPITA ou chamada de outros programas. Em segundo lugar, o sistema não funciona com resultados negativos. Nenhum desses dois problemas será muito difícil de resolver se você quiser melhorar o que aqui apresentamos.

FAZER atribui o nome da forma à variável global "ATUAL e então executa o programa. Para isso, ele cria uma lista de um item — o nome do programa — e então usa EXECUTE para rodá-lo.

```
AP FAZER :OBJETO
  MOSTRET
  FACA "ATUAL :OBJETO
  EXECUTE (LISTA :OBJETO)
FIM
```

CRESCER primeiro elimina o desenho inicial e depois usa DEFINA para definir o procedimento atual como um procedimento reescrito. A cor retorna então ao normal e a nova forma é desenhada. Observe que a entrada para CRESCER é armazenada na variável OPLISTA — que teremos de usar depois.

```
AP CRESCER :OPLISTA
  COR 0
  EXECUTE (LISTA :ATUAL)
  DEFINA "ATUAL ESCREVA.PROC LISTE
    :ATUAL
  COR 1
  EXECUTE (LISTA :ATUAL)
FIM
```

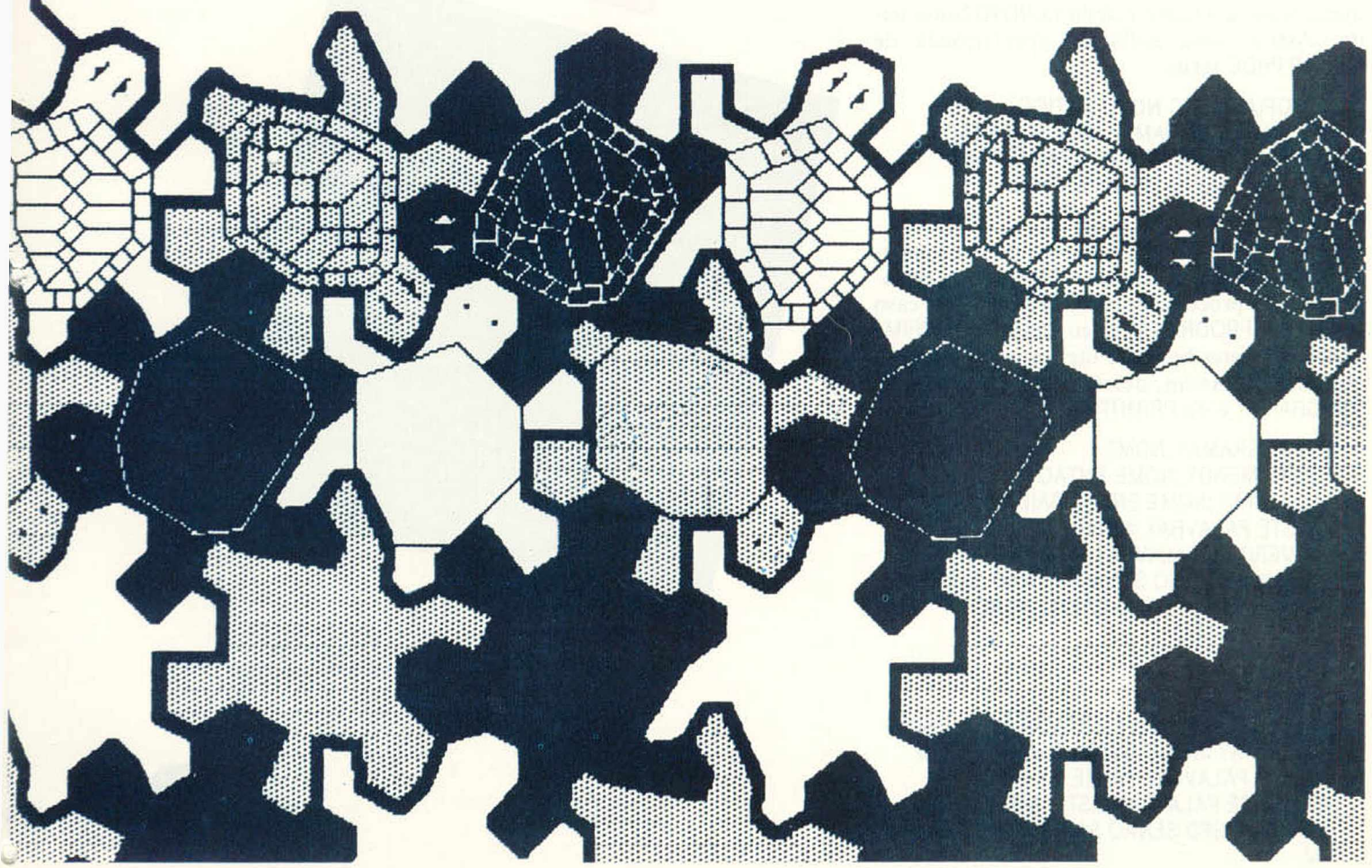
ESCREVA.PROC divide o texto em linhas e as transfere, uma por vez, para ESCREVA.LINHA:

```
AP ESCREVA.PROC :TEXTO
  SE :TEXTO = [] ENTAO SAIDA []
  SAIDA PENT ESCREVA.LINHA PRIMEIRO
    :TEXTO ESCREVA.PROC SEMPRIMEIRO
    :TEXTO
FIM
```

ESCREVA.LINHA examina uma linha, buscando um FR ou FRENTE. Se encontrar, ele passa o resto da linha para MUDE.

```
AP ESCREVA.LINHA. :LINHA
  SE :LINHA = [] ENTAO SAIDA []
  SE SEUM PRIMEIRO :LINHA = "FR
    PRIMEIRO :LINHA = "FRENTE ENTAO
    SAIDA MUDE SEMPRIMEIRO :LINHA
  SAIDA PENT PRIMEIRO :LINHA
    ESCREVA.LINHA SEMPRIMEIRO :LINHA
FIM
```

MUDE constrói a linha "reescrita". O primeiro



item de LISTAV — a entrada para MUDE — teria sido a entrada para FRENTE no procedimento inicial. Se ela for, por exemplo, 50 e se OPLISTA contiver [*2], então SENTENCA PRIMEIRO :LISTAV :OPLISTA será [50 * 2]. MUDE agora usa EXECUTE para calcular o valor dessa lista (obtendo o resultado 100). Finalmente, obtém-se uma lista que consiste em FR, na quantidade cujo valor acaba de ser calculado e, a seguir, na reescrita do resto da linha.

```
AP MUDE :LISTAV
  SAIDA (SENTENCA "FR (EXECUTE
    SENTENCA PRIMEIRO :LISTAV
    :OPLISTA) ESCREVA.LINHA
    SEMPRIMEIRO :LISTAV)
FIM
```

Cópias

Às vezes é útil fazer uma cópia de um programa. Assim, vamos definir um programa COPIAR.PROC — de modo que COPIAR.PROC "NOVO. .NOME "NOME.ANTIGO defina NOVO.NOME como uma cópia de NOME.ANTIGO (NOME.ANTIGO permanece não afetado). Uma definição evidente é a seguinte:

```
AP COPIAR.PROC NOVO ANTIGO
  DEFINA "NOVO LISTE "ANTIGO
FIM
```

O problema com essa definição está em que, se ANTIGO não existir, então o programa simplesmente seguirá adiante e definirá NOVO como nada. Assim, uma definição aperfeiçoada de COPIAR.PROC seria:

```
AP COPIAR.PROC NOVO ANTIGO
  SE NAO PROGRAMA? :ANTIGO ENTAO
    MOSTRE [NAO E PROGRAMA] PARE
  DEFINA "NOVO LISTE "ANTIGO
FIM
```

Essa definição utiliza um procedimento chamado PROGRAMA?, que resulta VERD se sua entrada é um procedimento, e FALSO em caso contrário. PROGRAMA? e seu equivalente PRIMITIVO? são testes bastante úteis que não existem em MLOGO. Assim, desenvolvemos versões de PROGRAMA? e de PRIMITIVO?:

```
AP PROGRAMA? :NOME
  SE NUMERO? :NOME ENTAO SAIDA "FALSO
  SE LISTA? :NOME ENTAO SAIDA "FALSO
  TESTE PALAVRA? :NOME
  SEVERD SE PALAVRA? LISTE :NOME ENTAO
    SAIDA "FALSO SENAO SE NAO (LISTE
    :NOME = []) ENTAO SAIDA "VERD
  SAIDA "FALSO
FIM

AP PRIMITIVO? :NOME
  SE NUMERO? :NOME ENTAO SAIDA "FALSO
  SE LISTA? :NOME ENTAO SAIDA "FALSO
  TESTE PALAVRA? :NOME
SEVERD SE PALAVRA? LISTE :NOME ENTAO
  SAIDA "VERD SENAO SAIDA "FALSO
FIM
```

Vantagens do LOGO

- É interpretado como o BASIC, o que facilita a adaptação dos programas.
- É estruturado com procedimentos, e não com sub-rotinas, como no BASIC.
- É ampliável como o FORTH — palavras novas podem ser acrescentadas ao vocabulário do computador.
- Opera com listas como o LISP, o que permite pesquisas na área da inteligência artificial.

O LOGO é uma linguagem ideal para "explorações". Elaboramos a maioria dos programas desta série a partir de um procedimento simples que executava parte da tarefa. A seguir, ele foi sendo aperfeiçoado, à medida que crescia nossa compreensão do problema. No final conseguimos algoritmos perfeitos e bem projetados.

Desvantagens do LOGO

- O espaço operativo é pequeno e a velocidade de execução, baixa.
- A maioria das versões não dispõe de matrizes nem de recursos para manipulação de arquivos e depuração de programas.

Registre

Linha Apple

ARTISTAS

Este programa gera linhas horizontais e verticais de tamanho e cores aleatórios, dando a impressão de um tecido trabalhado com muitas linhas. Para interromper o programa tecle [CTRL]-C.

```
10  REM *** ARTISTAS ***
20  GR:USA PAGINA GRAFICA 1
30  POKE -16302,0:REM TELA
    INTEIRA
40  CALL -1998: LIMPA 48 LINHAS
50  REM *** INICIO ***
60  COLOR = INT (RND(1) * 16)
70  HLIN 0,INT (RND(1)*48)
80  COLOR = INT (RND(1)*16)
90  VLIN 0,INT (RND(1) * 48) AT
    INT (RND(1) * 40 )
100 GOTO 60
```

FONTES DE ERROS

Enganos conceituais ou erros de lógica podem produzir resultados catastróficos. As partes potencialmente problemáticas são as ligações entre as rotinas e entre o programa e seu usuário.

OVERFLOW

Estouro, excesso. Quantidade que ultrapassa a capacidade de armazenamento de um registro.

Existem muitas fontes potenciais de erros a cada estágio da elaboração do programa, desde sua especificação até os testes, passando pelo projeto e codificação. Nos estágios de especificação e projeto, os erros costumam ocorrer quando não se verifica com precisão o tipo de problema a ser resolvido e não se toma cuidado suficiente para garantir que o programa faça exatamente o que foi determinado.

A probabilidade de erros diminui quando se segue o método de projeto estruturado. Outros erros ainda podem acontecer ao se transformar o projeto em instruções, na fase de digitação ou na de testes e eliminação de erros. A própria correção de um erro leva às vezes a outros.

Checagem de erros

Uma abordagem logicamente estruturada constitui a essência para evitar e eliminar erros; a lista de checagem de erros deste quadro é um exemplo abreviado de tal abordagem:

Variáveis

1 Todas as variáveis têm nomes exclusivos, levando-se em consideração que muitos interpretadores usam somente os dois primeiros caracteres de cada nome?
2 Alguma variável (sobretudo contadores de loop ou parâmetros de sub-rotinas) foi reutilizada enquanto seu conteúdo era ainda significante?
3 Os índices das matrizes estão dentro dos limites? São números inteiros?
4 Os índices das matrizes começam com o elemento 0 ou 1?

Cálculos

1 Os cálculos produzem resultados numéricos ou alfabéticos? E as variáveis para os resultados são alfabéticas ou numéricas?
2 Algum cálculo resulta num número pequeno ou grande demais para o computador trabalhar? Isso pode ocasionar um erro de divisão por zero?
3 Os erros de arredondamento de números podem ser significativos?
4 As operações numa expressão executam-se na ordem lógica correta ou na ordem imposta pela prioridade dos operadores aritméticos?

Comparações

1 Os strings são sempre comparados com strings, e os números com números?
2 Faz diferença se um string de teste é escrito em maiúsculas ou minúsculas?
3 Há comparações de strings de tamanhos diferentes?
4 Os operadores booleanos e os de comparação estão sendo utilizados de modo adequado? Por exemplo, A>B OU C não é o mesmo que A>B OU A>C. A prioridade dos operadores booleanos e dos de comparação afeta a execução das expressões de comparação?

Controle

1 Os loops e os algoritmos terminam qualquer que seja a situação das variáveis?
2 Os loops e as rotinas têm, cada um, somente um ponto de entrada e um ponto de saída?
3 Quando um comando IF-THEN falha, o controle passa para a próxima instrução ou para a próxima linha do programa?
4 O que acontece se nenhuma das condições de teste numa ramificação múltipla é satisfeita?

Mas é nas interfaces — entre rotinas e entre o programa e o usuário — que ocorre a maioria dos erros. Deve-se ter muito cuidado para os valores que passam pelas interfaces serem do tipo correto e estarem dentro dos limites exigidos pelo programa. Os valores podem ser conferidos dentro da rotina que os envia ou na rotina que os recebe, evitando-se, assim, muitos erros.

A checagem, nesse caso, é feita tanto para os valores fornecidos por um usuário como para os lidos de um arquivo. É sempre necessário conferir os valores de entrada numa rotina. As sub-rotinas são projetadas para gerar um conjunto bem definido de saídas, mas não se espera o mesmo dos seres humanos, que podem responder a uma pergunta de diferentes maneiras. Assim, as rotinas que aceitam dados do usuário precisam ser checadas com rigor.

Aconselha-se igual procedimento para todas as rotinas de manipulação de arquivos, pois ocorrem problemas até no momento da leitura dos dados.

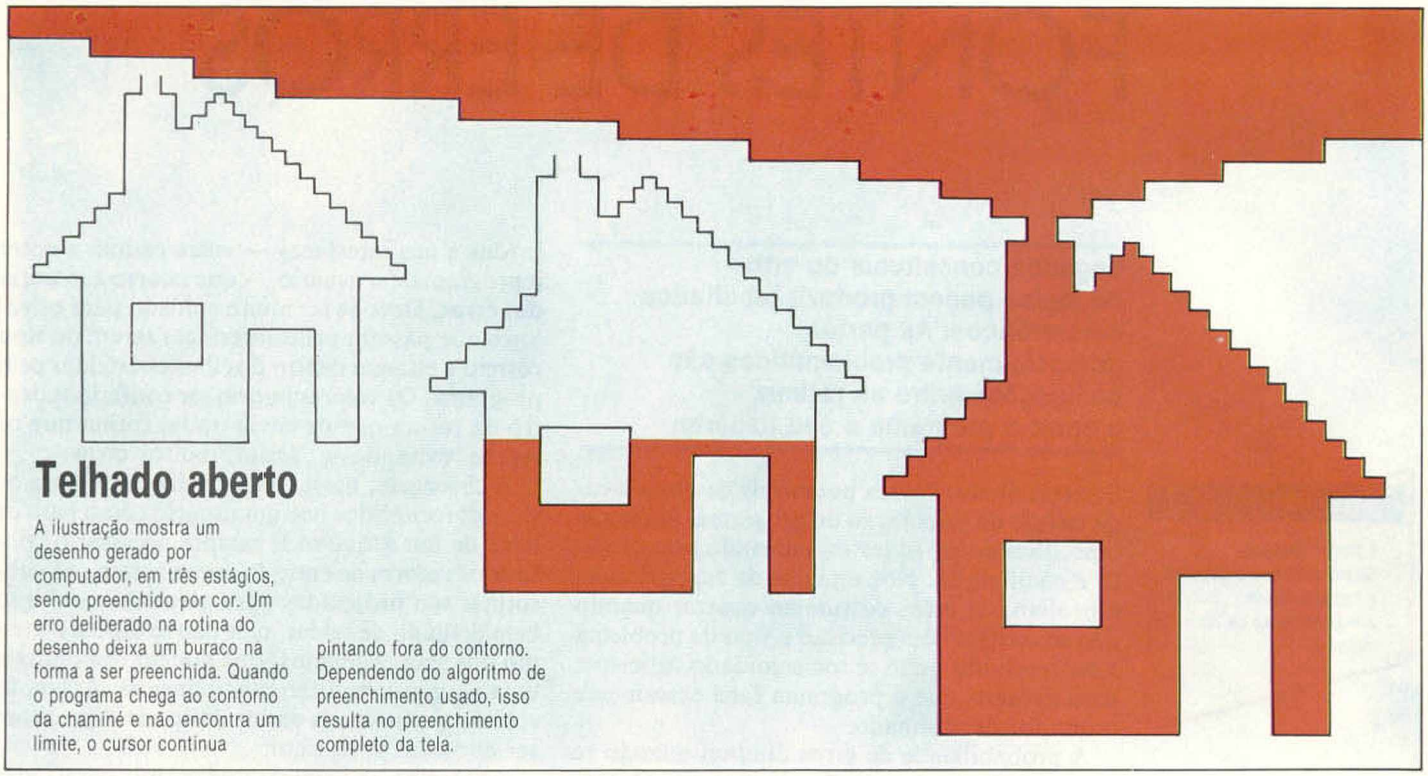
Não é comum os programas pararem de rodar em razão dos erros. Isso só acontece quando se desrespeita alguma regra da linguagem (usando ilegalmente um operador, como em RESULTADO = PRIMEIRO$ + SEGUNDO$) ou do sistema operacional (abrindo arquivos demais ao mesmo tempo). Por exemplo, as instruções a seguir parecem constituir um programa correto:

```
10 FOR CONT = 1 TO 10
20 SOMA = SOMA + 1
30 PRINT CONT, SOMA
40 GOTO 10
50 NEXT CONT
```

No entanto, é um algoritmo sem fim que vai provocar problemas pelo próprio funcionamento da linguagem. Nesse caso, a linguagem (BASIC) possui um controle para acompanhar os loops FOR-NEXT, somando um a cada vez que um novo loop começa.

Neste programa, a linha 50 (com o comando NEXT que diminuiria o contador) nunca é alcançada; assim, o contador aumenta gradualmente até a emissão da mensagem de "estouro" ("overflow"), quando o programa é interrompido pelo interpretador. Erros como este são detectados com facilidade, mas, se aparecerem em rotinas pouco usadas, precisa-se realizar um teste completo para descobri-los.

Um tipo de erro mais sutil é o que permite rodar o programa, mas invalida os resultados. Ocorre, por exemplo, no desenho de formas preenchidas com cor. As rotinas de preenchimento verificam os limites da forma. Quando um dos



Telhado aberto

A ilustração mostra um desenho gerado por computador, em três estágios, sendo preenchido por cor. Um erro deliberado na rotina do desenho deixa um buraco na forma a ser preenchida. Quando o programa chega ao contorno da chaminé e não encontra um limite, o cursor continua pintando fora do contorno. Dependendo do algoritmo de preenchimento usado, isso resulta no preenchimento completo da tela.

limites é alcançado, o computador desvia o cursor e continua desenhando até atingir outro limite. Para uma rotina de preenchimento funcionar, os limites precisam ser bem definidos e completos. Isso significa que não pode haver espaço aberto no contorno da forma, senão a rotina espalha a cor além dos limites.

As versões da linguagem BASIC usadas pela maioria dos micros possibilitam manipular os erros com facilidade, produzindo mensagens claras e concisas. Permitem também dar prosseguimento aos programas com problemas após a alteração dos valores das variáveis no teclado — um recurso útil quando os erros de um programa estão sendo eliminados. Na maioria das versões do BASIC usa-se o comando ON ERROR GOTO com o objetivo de transferir o fluxo de controle para uma rotina especial, que dá um tratamento específico a erros. Para tanto, inclui-se no programa uma linha como:

```
30 ON ERROR GOTO 20000: REM rotinas de tratamento de erros
```

Assim, a ocorrência de um erro faz o programa se comportar como se tivesse encontrado a instrução GOTO 20000. Às vezes, a instrução ON ERROR cria duas variáveis: a primeira armazena um número que indica o tipo de erro ocorrido e a outra identifica o número da linha em que o erro foi encontrado. Os nomes dados a essas variáveis e os números dos erros resultantes diferem conforme o equipamento, o que exige consulta ao manual. Uma vez detectado o erro, o fluxo do programa desvia-se para a linha 20000; o número armazenado na variável permite a identificação do erro e, assim, a ação correta é executada.

Um programa bem escrito não tem mais do que uma rotina ON ERROR. Tais rotinas não são capazes de lidar com erros de sintaxe, escassez de memória ou overflow. O máximo que esse recurso oferece é um encerramento organizado do programa, garantindo que todos os arquivos sejam fechados e que o usuário saiba exatamente o que aconteceu.

Alguns erros, como a divisão por zero, podem ser eliminados por essa rotina, mas convém utilizar outro método. Há várias razões para isso:

- A instrução ON ERROR GOTO e a subseqüente volta ao programa principal constituem entrada e saída extras de uma rotina. Isso viola o princípio de programação estruturada, de acordo com o qual uma rotina deve ter apenas um ponto de entrada e um de saída.

- O lugar apropriado para se proteger de uma divisão por zero é a própria rotina que faz a divisão. Não constitui boa prática projetar algoritmos que interrompam o sistema. Se a checagem extra de erro diminuir a velocidade do programa a um grau inaceitável, a rotina deverá ser reprojetada de forma a otimizá-la.

- As rotinas de tratamento de erros tornam-se complicadas se existem cadeias de IF-THEN-ELSE com saídas múltiplas. Elas são restritas pela numeração de linhas do resto do programa e, assim, torna-se necessário reescrevê-las quando se refaz qualquer rotina que se sirva delas. Essas rotinas são difíceis de projetar, testar e corrigir; se contiverem erros, os problemas introduzidos não serão detectados com facilidade, devido ao desvio do fluxo de controle por caminhos imprevisíveis.

OUSADOS CONSERVADORES

O fornecimento regular de componentes permitiu à Commodore invejáveis cifras de vendagem de computadores no mercado internacional. Esse êxito se deve, em grande parte, ao talento de Jack Tramiel e de Chuck Peddle.

A Commodore é uma enorme companhia e, devido a seu volume de produção, pode fazer bons negócios com fornecedores externos: em alguns casos, paga apenas a metade do que seus competidores pagam por chips.

A forte posição da empresa deve muito ao sucesso do CBM PET. Chuck Peddle, responsável por seu design, baseou a máquina no processador 6502, equipou-a com o BASIC da Microsoft e lhe forneceu um editor de tela completo, muito mais fácil de usar do que os painéis simples que os entusiastas vinham utilizando desde o advento do microprocessador, na metade da década de 70. Até hoje, o IBM PC, supostamente da próxima geração, não dispõe, no modelo padrão, de um editor de tela completo.

A Commodore estava em posição privilegiada para construir tal máquina — ela já possuía a MOS Technology, que tinha os direitos de produção do microprocessador 6502. Isso foi fundamental: ambos os competidores do PET — o Apple II e o Tandy TRS-80 — usavam a CPU 6502. Assim, a Commodore podia acompanhar a produção dessas duas companhias. Contudo, o nascimento do PET foi problemático. O presidente da empresa, Jack Tramiel, insistia em que os componentes da memória do PET também fossem da MOS Technology, ao contrário do que queria Peddle. Isso levou a uma disputa interna que resultou na demissão do designer.

Originalmente acondicionado num gabinete de aço, o Commodore PET recebeu depois novo revestimento plástico, que lhe deu uma aparência modernizada. Seu modelo com capacidade de 22 Mbytes armazenados em disco constitui a série 8000.

Apesar da idade, o PET vendia muito bem ainda em 1985. Clientes mais conservadores sentem-se à vontade com ele e não vêem por que mudar seu software para a nova geração de computadores baseados na CPU 8088. Seus fornecedores alegam até mesmo que — para seu próprio espanto — o PET ainda pode competir com o IBM PC e o Apricot.

Por conservadorismo e por um desejo de conter os custos, a empresa nunca se preocupou em ampliar as especificações de sua máquina para principiantes. A maior parte do software original do PET ainda pode rodar nas máquinas de hoje. O BASIC da Microsoft versão 2.0 ainda é, em grande parte, o mesmo da época em que Peddle o adotou.

Isso implica desvantagem para os amadores mais adiantados. O Microsoft 2.0 foi desenvolvido numa época em que os recursos gráficos e de som eram considerados supérfluos em computadores de baixo custo.

Os computadores para amadores que a Commodore lançou recentemente são bem equipados com essas características, mas o BASIC não tem os comandos necessários e exige lento e laborioso uso de comandos POKE para endereçar posições específicas da memória. Isso, no entanto, não é problema com o software em cartucho já pronto, do qual a Commodore oferece grande variedade.

O sobrevivente de Auschwitz

Devido sobretudo ao fornecimento regular de semicondutores da matriz para as fábricas, a Commodore conseguiu superar as dificuldades que forçaram seus competidores a derivar para o mercado de jogos. A Texas e a Mattel retiraram-se do mercado dos micros não profissionais e a Atari não vem se destacando no setor. A Commodore, com o fornecimento garantido de peças a baixo custo, podia forçar a redução dos preços do computador e do software em cartucho. Assim, eliminava a competição e ainda obtinha lucro. Seus cartuchos chegaram a custar um terço do que custavam os de seus competidores. Uma fábrica moderna e automatizada e a disponibilidade de peças a baixo preço foram o resultado de dois decênios de experiência em fabricação.

A Commodore não alcançou essa posição privilegiada por acaso. Ela tem enfrentado situações difíceis e esteve pelo menos duas vezes à beira da falência.

O polonês Jack Tramiel era adolescente quando chegou aos Estados Unidos, após a Segunda Guerra Mundial, como sobrevivente do campo de concentração de Auschwitz. Em 1955, conseguiu formar a CBM (Commodore Business Machines), cujo nome foi escolhido devido à semelhança com o da IBM. A matriz ficava em Toronto, Canadá, onde a companhia iniciou suas atividades como montadora de máquinas de escrever, sob licença da Tchecoslováquia.

Em 1975, após duas décadas de comercialização na área de produtos para escritório, a Commodore fez grandes progressos.

Os construtores
A força motriz da Commodore tem sido seu presidente, Jack Tramiel.

Chuck Peddle
É o homem por trás do projeto do PET e do chip 6502 nele incorporado. Peddle saiu da Commodore para formar sua própria companhia e produzir a máquina para uso profissional Sirius.

Na época, houve uma guerra pelo mercado das calculadoras, vencida pelos japoneses. Mas Tramiel — respeitado e temido por seus métodos de competição — conseguiu atravessar o período. Ele percebeu o potencial de mercado do microcomputador e, em 1976, levou Chuck Peddle para a companhia. Em menos de dez anos, o capital da Commodore aumentou cinqüenta vezes.

A quinta geração

Tramiel fez dela uma formidável empresa de fabricação e comercialização, mas não obteve grande progresso no desenvolvimento de novos produtos.

A filosofia da companhia é: "Nós vendemos para o povo, não para a elite". Tramiel acredita que o cliente comprará o produto que tiver mais a oferecer em troca de seu investimento. Mas as exigências de produção barata e em massa podem agir contra a incorporação de tecnologia mais avançada.

No final de 1982, como boa parte da pequena equipe de pesquisa e desenvolvimento havia deixado a Commodore, ela passou a apoiar-se nas pesquisas realizadas por outras companhias. Iniciou entendimentos com empresas no Extremo Oriente, visando à fabricação de unidades de disco. E tem feito contato com empresas como a Sony, para a aquisição de tecnologias caras da quinta geração: reconhecimento de voz, robôs para uso doméstico e sofisticados dispositivos para armazenamento.

A Commodore chegou mesmo a abordar o criativo projetista Paul Johnson, da Oric, tentando conseguir que ele projetasse um chip para sua nova série de micros.

Na metade da década de 80, a Commodore parecia mais confiante do que nunca e continuava a apostar no preço baixo e na simplicidade. Ela apresentou dois micros (o 264 e o V364) baseados nos novos processadores 7501. O V364 possui um sintetizador de fala com vocabulário de 250 palavras incorporado. Seguindo as tendências atuais, os softwares para processamento de texto, cálculo de folhas eletrônicas e gráficos estarão disponíveis como opcionais.

Os campeões da Commodore

1982
O **CBM 700** é o substituto das máquinas 8032 e deve elevar os computadores profissionais da Commodore ao nível dos micros mais modernos.

1984
O **SX64** é a versão modernizada do 64 em gabinete portátil com tela em cores e unidade de disco.

1977
Commodore PET original foi o primeiro microcomputador produzido para o grande público. Após numerosas modificações, ainda vende bem.

1979
O **Vic-20** foi o primeiro micro de baixo custo da Commodore. Apesar das limitações e da forte competição, ainda é bastante popular.

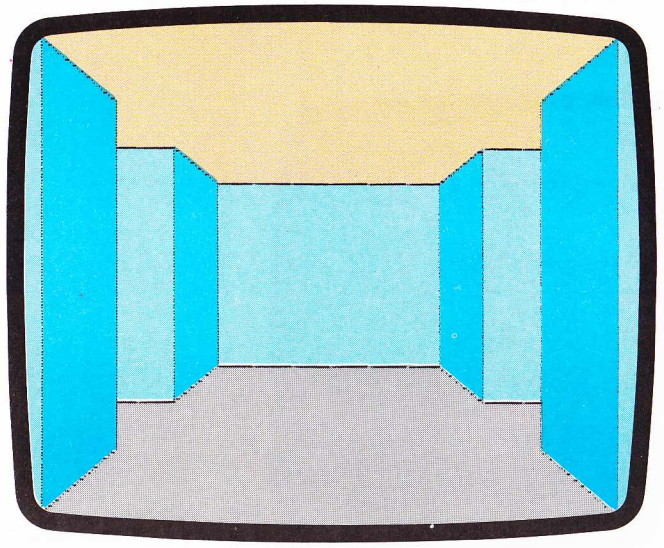
1981
O **CBM 8032** equipou a série dos PET com capacidade de 80 colunas, dando às máquinas a possibilidade de rodar software profissional.

1983
O **Commodore 64,** com tela de 40 colunas e memória de 64 K, superou as limitações do Vic.

1980
Com o **SuperPET** (ou **CBM 9000**), tentou-se produzir uma versão profissional do PET.

```
 GO SUB 1000: NEXT i
 156 IF dx=-1 AND y=9 AND x<5 TH
EN PRINT AT 10,14;"Saida"
 158 LET a$=INKEY$: IF a$="" THE
N GO TO 155
 160 IF a$<>"m" THEN GO TO 190
 170 IF (dx=1 AND v(x+1,y)=0) OR
 (dx=-1 AND v(x,y)=0) OR (dy=1 A
ND h(x,y+1)=0) OR (dy=-1 AND h(x
,y)=0) THEN LET x=x+dx: LET y=y+
dy
 190 IF x=1 AND y=9 THEN GO TO 9
990
 200 IF a$="r" THEN LET dx=-dx:
LET dy=-dy: GO TO 130
 210 IF a$="o" THEN GO TO 300
 215 IF a$<>"p" THEN GO TO 130
 220 IF ABS dx=1 THEN LET dy=-dx
: LET dx=0: GO TO 130
 230 LET dx=dy: LET dy=0: GO TO
130
```

```
1310 GO SUB 7000+i
1350 IF v(Ix+1,Iy)=0 THEN GO TO
8500+i
1360 GO TO 6000+i
6000 PLOT 110,76: DRAW 0,24: DRA
W 36,0: DRAW 0,-24: DRAW -36,0:
RETURN
6001 PLOT 108,75: DRAW 0,26: DRA
W 40,0: DRAW 0,-26: DRAW -40,0:
RETURN
6002 PLOT 106,74: DRAW 0,28: DRA
W 44,0: DRAW 0,-28: DRAW -44,0:
RETURN
6003 PLOT 102,71: DRAW 0,34: DRA
W 52,0: DRAW 0,-34: DRAW -52,0:
RETURN
6004 PLOT 94,65: DRAW 0,46: DRAW
 68,0: DRAW 0,-46: DRAW -68,0: R
ETURN
6005 PLOT 82,57: DRAW 0,62: DRAW
 92,0: DRAW 0,-62: DRAW -92,0: R
```



Decisões apressadas
Além de um bom senso de direção, você não pode vacilar na escolha de alguma saída, pois a rapidez é fundamental nesse jogo.

```
 300 IF ABS dx=1 THEN LET dy=dx:
 LET dx=0: GO TO 130
 310 LET dx=-dy: LET dy=0: GO TO
 130
1000 IF dx<>1 THEN GO TO 1100
1010 IF h(Ix,Iy+1)=0 THEN GO SUB
 7500+i: GO TO 1050
1020 GO SUB 7000+i
1050 IF h(Ix,Iy)=0 THEN GO TO 85
00+i
1060 GO TO 6000+i
1100 IF dx<>-1 THEN GO TO 1200
1110 IF h(Ix,Iy)=0 THEN GO SUB 7
500+i: GO TO 1150
1120 GO SUB 7000+i
1150 IF h(Ix,Iy+1)=0 THEN GO TO
8500+i
1160 GO TO 6000+i
1200 IF dy<>-1 THEN GO TO 1300
1210 IF v(Ix+1,Iy)=0 THEN GO SUB
 7500+i: GO TO 1250
1220 GO SUB 7000+i
1250 IF v(Ix,Iy)=0 THEN GO TO 85
00+i
1260 GO TO 6000+i
1300 IF v(Ix,Iy)=0 THEN GO SUB 7
500+i: GO TO 1350
```

```
ETURN
6006 PLOT 64,45: DRAW 0,86: DRAW
 128,0: DRAW 0,-86: DRAW -128,0:
 RETURN
6007 PLOT 40,29: DRAW 0,118: DRA
W 176,0: DRAW 0,-118: DRAW -176,
0: RETURN
6008 PLOT 8,7: DRAW 0,162: DRAW
240,0: DRAW 0,-162: DRAW -240,0:
 RETURN
7000 PLOT 108,75: DRAW 2,1: DRAW
 0,24: DRAW -2,1: RETURN
7001 PLOT 106,74: DRAW 2,1: DRAW
 0,26: DRAW -2,1: RETURN
7002 PLOT 102,71: DRAW 4,3: DRAW
 0,28: DRAW -4,3: RETURN
7003 PLOT 94,65: DRAW 8,6: DRAW
0,34: DRAW -8,6: RETURN
7004 PLOT 82,57: DRAW 12,8: DRAW
 0,46: DRAW -12,8: RETURN
7005 PLOT 64,45: DRAW 18,12: DRA
W 0,62: DRAW -18,12: RETURN
7006 PLOT 40,29: DRAW 24,16: DRA
W 0,86: DRAW -24,16: RETURN
7007 PLOT 8,7: DRAW 32,22: DRAW
0,118: DRAW -32,22: RETURN
7008 PLOT 0,1: DRAW 8,6: DRAW 0,
```



```
162: DRAW -8,6: RETURN
7500 PLOT 108,76: DRAW 2,0: DRAW
 0,24: DRAW -2,0: RETURN
7501 PLOT 106,75: DRAW 2,0: DRAW
 0,26: DRAW -2,0: RETURN
7502 PLOT 102,74: DRAW 4,0: DRAW
 0,28: DRAW -4,0: RETURN
7503 PLOT 94,71: DRAW 8,0: DRAW
0,34: DRAW -8,0: RETURN
7504 PLOT 82,65: DRAW 12,0: DRAW
 0,46: DRAW -12,0: RETURN
7505 PLOT 64,57: DRAW 18,0: DRAW
 0,62: DRAW -18,0: RETURN
7506 PLOT 40,45: DRAW 24,0: DRAW
 0,86: DRAW -24,0: RETURN
7507 PLOT 8,29: DRAW 32,0: DRAW
0,118: DRAW -32,0: RETURN
7508 PLOT 0,7: DRAW 8,0: DRAW 0,
162: DRAW -8,0: RETURN
8000 PLOT 148,75: DRAW -2,1: DRA
W 0,24: DRAW 2,1: RETURN
8001 PLOT 150,74: DRAW -2,1: DRA
W 0,26: DRAW 2,1: RETURN
8002 PLOT 154,71: DRAW -4,3: DRA
W 0,28: DRAW 4,3: RETURN
8003 PLOT 162,65: DRAW -8,6: DRA
W 0,34: DRAW 8,6: RETURN
8004 PLOT 174,57: DRAW -12,8: DR
AW 0,46: DRAW 12,8: RETURN
8005 PLOT 192,45: DRAW -18,12: D
RAW 0,62: DRAW 18,12: RETURN
8006 PLOT 216,29: DRAW -24,16: D
RAW 0,86: DRAW 24,16: RETURN
8007 PLOT 248,7: DRAW -32,22: DR
AW 0,118: DRAW 32,22: RETURN
8008 PLOT 255,1: DRAW -8,6: DRAW
 0,162: DRAW 8,6: RETURN
8500 PLOT 148,76: DRAW -2,0: DRA
W 0,24: DRAW 2,0: RETURN
8501 PLOT 150,75: DRAW -2,0: DRA
W 0,26: DRAW 2,0: RETURN
8502 PLOT 154,74: DRAW -4,0: DRA
W 0,28: DRAW 4,0: RETURN
8503 PLOT 162,71: DRAW -8,0: DRA
W 0,34: DRAW 8,0: RETURN
8504 PLOT 174,65: DRAW -12,0: DR
AW 0,46: DRAW 12,0: RETURN
```

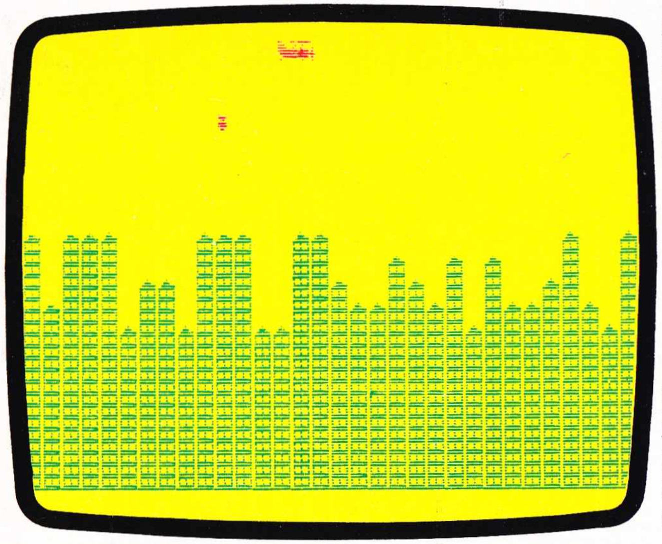
```
8505 PLOT 192,57: DRAW -18,0: DR
AW 0,62: DRAW 18,0:::: RETURN
8506 PLOT 216,45: DRAW -24,0: DR
AW 0,86: DRAW 24,0: RETURN
8507 PLOT 248,29: DRAW -32,0: DR
AW 0,118: DRAW 32,0: RETURN
8508 PLOT 255,7: DRAW -7,0: DRAW
 0,162: DRAW 7,0: RETURN
9000 FOR i=1 TO 9: LET h(i,1)=1:
 LET h(i,10)=1: FOR j=2 TO 9: RE
AD a: LET h(i,j)=a: NEXT j: NEXT
 i
9010 FOR i=1 TO 9: LET v(1,i)=1:
 LET v(10,i)=1: FOR j=2 TO 9: RE
AD a: LET v(j,i)=a: NEXT j: NEXT
 i
9020 RETURN
9100 DATA 1,1,1,0,0,0,0,1
9101 DATA 0,0,1,0,0,1,0,1
9102 DATA 1,1,0,0,0,0,1,1
9103 DATA 0,0,1,1,0,0,1,0
9104 DATA 1,1,1,0,0,1,1,0
9105 DATA 1,0,0,0,0,0,0,1
9106 DATA 1,1,1,0,1,0,0,1
9107 DATA 1,1,1,0,0,1,1,0
9108 DATA 1,0,0,0,0,0,0,0
9200 DATA 0,0,0,0,0,0,0,0
9201 DATA 1,1,1,1,1,0,1,0
9202 DATA 0,0,0,0,0,0,0,1
9203 DATA 0,1,0,0,1,0,0,0
9204 DATA 1,1,0,1,1,1,1,1
9205 DATA 1,1,1,1,1,1,0,1
9206 DATA 0,0,0,0,0,0,1,0
9207 DATA 1,1,0,0,1,1,0,0
9208 DATA 0,0,0,1,0,0,0,1
9990 LET te=PEEK 23672+256*PEEK
23673+4096*PEEK 23674
9991 LET t=te-ti
9992 PRINT INK 2;AT 0,4;"Muito b
em! Voce conseguiu em "
9993 PRINT INK 2;AT 1,4;t/50;" s
egundos"
9994 PAUSE 100: CLS : PRINT INK
2;" Quer tentar outra vez? "
9995 IF INKEY$="" THEN GO TO 999
5
9996 IF INKEY$<>"" THEN RUN
```

BOMBAS

PARA TK 90X

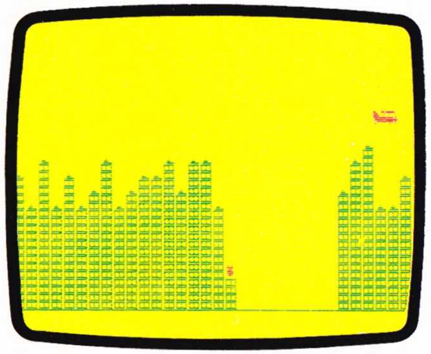
Neste programa, você é um destruidor de concreto armado, pronto para bombardear todos os edifícios que aparecem na tela. Seu objetivo é limpar o terreno para aterrissar suavemente com seu avião. Mas, cuidado para não esbarrar nos prédios, porque à medida que passa o tempo, a altitude da aeronave diminui. Por isso, siga este conselho: comece bombardeando os edifícios mais altos. Para lançar as bombas, tarefa que exige muita rapidez, basta apertar qualquer tecla. Você vai perceber a velocidade do BASIC do TK 90X e sua alta resolução gráfica. O número de pontos depende da quantidade de edifícios atingidos. Os mais altos valem mais.

```
  10 REM ************************
  20 REM * PROGRAMA PARA TK 90X*
  30 REM *  PROGRAMA  BOMBAS    *
  40 REM ************************
  50 CLS : GO SUB 9000: GO SUB 8
000
  60 FOR a=0 TO 20
  70 FOR b=0 TO 31
  80 PRINT AT a,b;" "
  90 IF SCREEN$ (a,b+3)<>" " THE
N GO TO 270
 100 IF INKEY$<>"" THEN GO SUB 1
60
 110 FOR p=1 TO 8: NEXT p
```



Antes os mais altos
Calcule bem sua trajetória, a fim de apertar a tecla na hora certa, procurando destruir os edifícios mais altos em primeiro lugar.

Rapidez e resolução
A rapidez da linguagem BASIC do TK 90X, sua alta resolução gráfica e a variedade de cores oferecida são bastante exploradas neste jogo.

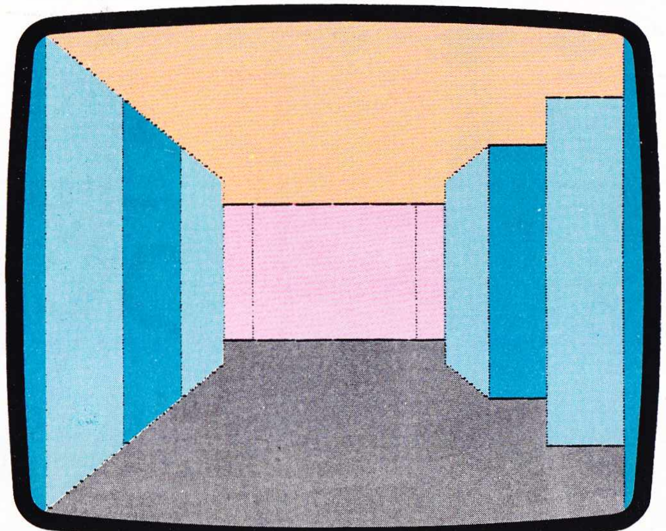


```
120 NEXT b: NEXT a
130 PRINT AT 10,10; FLASH 1; BR
IGHT 1;"Aterrissagem perfeita"
140 INPUT BRIGHT 1; FLASH 1; IN
VERSE 1;"Aperte ENTER para jogar
de novo "; LINE a$: RUN
150 STOP
160 LET b1=b
170 FOR x=a TO 20
180 PRINT AT x-1,b1;" "
190 PRINT AT x,b1; INK 2;"¥"
200 LET b=b+1
210 IF b>31 THEN LET a=a+1: LET
b=0
220 PRINT AT a,b;" ": IF SCRE
EN$ (a,b+3)<>" " THEN GO TO 270
230 BEEP .008,-(x-20)
240 NEXT x
250 PRINT AT x-1,b1;" "
260 RETURN
270 FOR c=a TO 19: PRINT AT c,b
+1;" ";AT c+1,b+1;" "
280 BEEP .05,-(c+2): NEXT c
282 PAUSE 100
285 PRINT AT c+1,b+1;" ": PRI
NT FLASH 1;AT 19,b+1*(b<31);" "
  FLASH 1;AT 20,b+1*(b<31);" "
286 PAUSE 200
290 PRINT AT 0,11; FLASH 1; BRI
GHT 1;"";AT 0,9; INVERSE 1;"Pont
uacao";(a*10)+b
300 GO TO 140
8000 FOR k=0 TO 31
8010 LET r=INT (RND*6)+2: LET j=
INT (RND*5)+10
8020 PRINT AT j,k; INK r;"▲"
8030 FOR j=j+1 TO 20
8040 PRINT AT j,k; INK r;"▦"
8050 NEXT j: NEXT k
8055 PRINT #1; INK 6; BRIGHT 1;A
T 0,0;"Aperte uma tecla p/ solta
r bomba"
8060 PLOT 0,7: DRAW 255,0: RETUR
N
9000 BORDER 0: PAPER 0: INK 7: C
LS
9010 FOR u=USR "a" TO USR "i"+7
9020 READ user: POKE u,user
9030 NEXT u: RETURN
```

```
9040 DATA 199,225,242,255,127,63
,64,0
9050 DATA 252,32,66,250,255,250,
34,0
9060 DATA 0,36,60,24,60,60,24,0
9070 DATA 0,8,28,63,127,73,73,12
7
9080 DATA 127,73,73,127,127,73,7
3,127
9090 DATA 0,70,66,146,202,21,123
,11
9100 DATA 33,58,4,74,81,40,196,2
40
9110 DATA 3,255,5,13,117,117,117
,138
9120 DATA BIN 11001111,240,191,1
72,163,163,81,0
```


LABIRINTO
PARA TK 90X

Embora o minotauro não esteja em seu encalço, você precisa encontrar a saída do labirinto o mais rapidamente possível. Criado para aproveitar a ótima resolução gráfica do TK 90X, esse jogo fará com que você, que nunca aparece na tela, percorra os salões do labirinto. Movendo-se para a frente com a tecla [M], escolha uma das saídas do salão em que está e, quando ela atingir o limite da tela, aperte as teclas para mudança de rumo — [O] para a esquerda e [P] para a direita. Se quiser tentar outra parte do labirinto, use alternadamente as teclas [E] e [R].



Mudança de rumo
Para entrar na primeira saída à direita, é preciso mover-se para a frente e esperar que a linha demarcadora saia da tela.

```
1 REM ***********************
2 REM *PROGRAMA PARA TK 90X *
3 REM *  PROGRAMA LABIRINTO *
4 REM ***********************
10 BORDER 1: PAPER 1: BRIGHT 1
: INK 9: CLS
20 PRINT '''" Para andar no l
abirinto, voce deve usar as tec
las:"
30 PRINT ''""" M "" - Para ir
em frente"
40 PRINT ''""" P "" e "" O ""P
ara virar para os lados"
50 PRINT ''""" E "" ou "" R "
"para socorro"
60 PRINT '' FLASH 1; BRIGHT 1;
" BOA SORTE E MUITO CUIDADO PARA
NAO SE PERDER!
"
65 PRINT #1;AT 0,0;"Aperte qua
lquer tecla
70 PAUSE 0: FLASH 0: CLS : BRI
GHT 1
100 DIM v(10,10): DIM h(10,10)
110 GO SUB 9000
120 LET x=1: LET y=1: LET dx=1:
LET dy=0
125 LET ti=PEEK 23672+256*PEEK
23673+4096*PEEK 23674
130 LET I=9: LET Ix=x+(dx=-1):
LET Iy=y+(dy=-1)
132 LET I=I-1: LET Ix=Ix+dx: LE
T Iy=Iy+dy
134 IF dx<>0 AND v(Ix,Iy)=0 THE
N GO TO 132
136 IF dy<>0 AND h(Ix,Iy)=0 THE
N GO TO 132
140 CLS : GO SUB 6000+I
145 LET Ix=Ix-(dx=-1): LET Iy=I
y-(dy=-1)
150 FOR i=I TO 8
155 LET Ix=Ix-dx: LET Iy=Iy-dy:
```