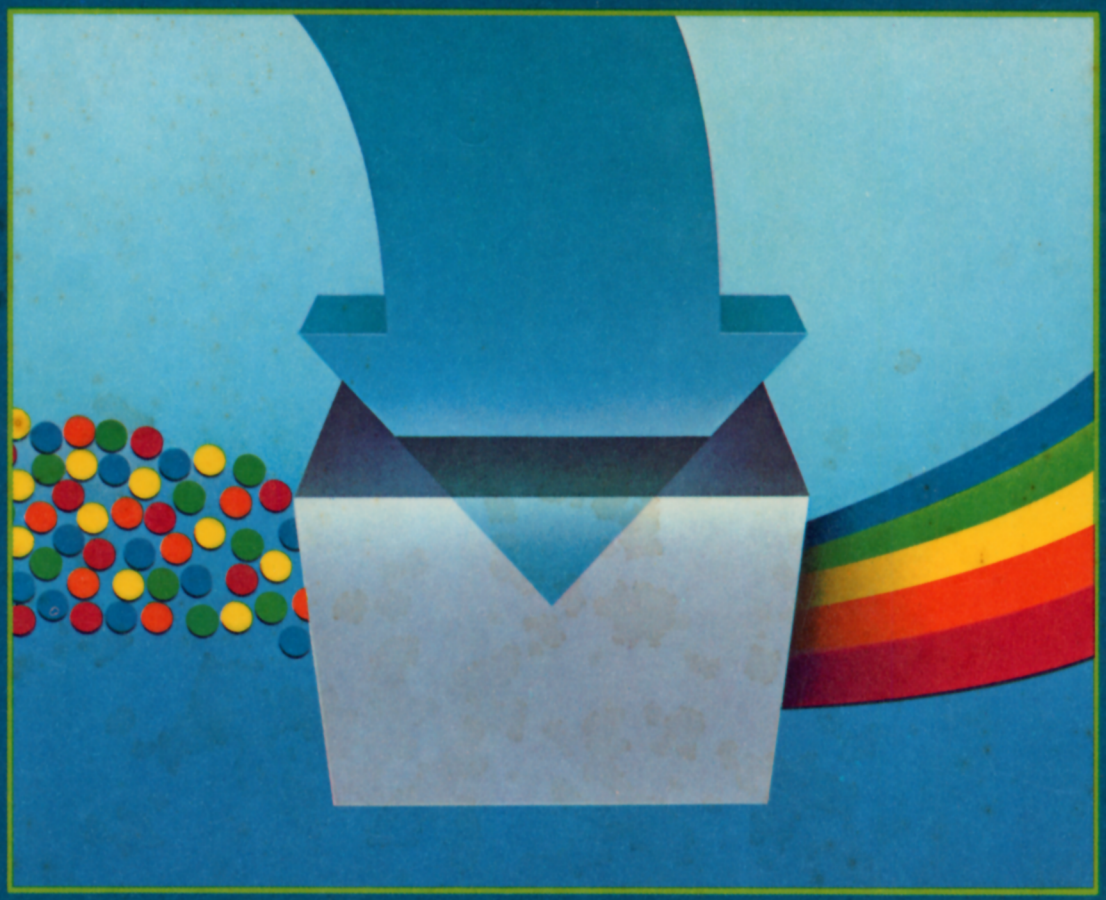
ENTENDA O COMPUTADOR

AS LINGUAGENS DO COMPUTADOR



NOVA CULTURAL

E N T E N D A O

COMPUTADOR

AS LINGUAGENS DO COMPUTADOR

NOVA CULTURAL

Authorized Portuguese Edition © 1988, Editora
Nova Cultural S.A. Original edition © 1985
Time-Life Books Inc. All rights reserved.

Nenhuma parte desta publicação pode ser reproduzida
por qualquer meio, seja eletrônico ou mecânico,
inclusive por equipamentos ou sistemas de classificação
de informações e pesquisa, sem a prévia autorização da
editora, exceto curtos trechos para resenhas, desde que
com o devido crédito.
Time-Life is a trademark of Time Inc., U.S.A.

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, 2000 – 3º andar
CEP 01452 – São Paulo – SP – Brasil
(Artigo 15 da lei 5988, de 14/12/1973)

Esta obra foi composta na AM Produções Gráficas Ltda.
e impressa na Divisão Gráfica da Editora Abril S.A.

ISBN: 85-13-00085-X

5 O instrutor invisível
O software no controle
1

23 Desenvolvimento de novas linguagens
A exigente arte do programador
2

55 Índice

56 Créditos das fotos



O instrutor invisível

Manhã do dia 10 de abril de 1981. Posicionada sobre a plataforma de lançamento do Cabo Canaveral, na Flórida, a nave espacial Columbia parecia pequenina, se comparada com seus gigantescos foguetes propulsores. Dentro da cabine, atados a seus assentos, os astronautas Robert Crippen e John Young encontravam-se prontos para serem lançados em órbita ao redor da Terra. Era um momento decisivo: pela primeira vez, depois de mais de cinco anos, norte-americanos aventuravam-se pelo espaço. E, pela primeira vez na história, os astronautas tentariam voltar à Terra num aparelho, dirigível e dotado de asas, que poderia ser reaproveitado em futuras missões. Faltando apenas 20 minutos na contagem regressiva, tudo parecia estar sob controle.

Subitamente, luzes de alarme piscaram nos equipamentos do Controle da Missão em Houston: o computador de retaguarda da nave detectara erros na ligação superior entre os computadores da base e os da nave, ligação esta que permitiria, em caso de necessidade, que o Controle da Missão substituísse a tripulação na operação da nave. Os técnicos logo descobriram que a causa das mensagens de erro era uma discrepância de tempo: os processos de computador estavam defasados em 40 milésimos de segundo.

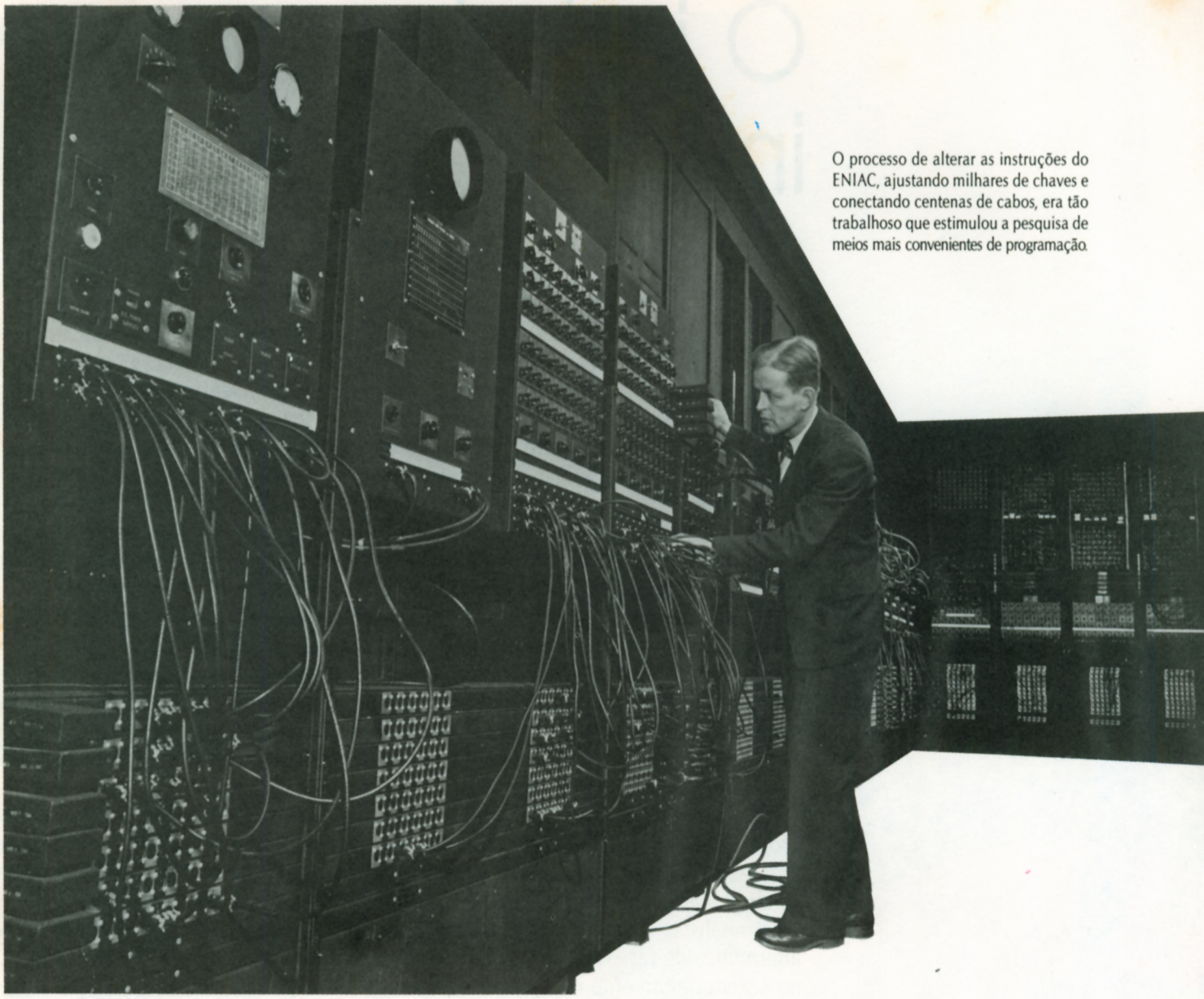
Possivelmente, o problema poderia ser sanado pela simples reinicialização dos programas. Mas, com a nave totalmente abastecida, essa opção foi considerada arriscada. De qualquer maneira, o Controle da Missão não permitiria o lançamento enquanto a natureza desse erro de sincronismo não fosse esclarecida.

Os analistas, empenhados em descobrir a falha, detectaram que o problema da alternância de fase tinha suas raízes em mudanças feitas, ao longo de vários anos, no sistema altamente complexo de software — os programas, ou listas de instruções, que ordenam ao computador o que fazer e em que momento. Descobriram também que o potencial de defasagem produzido por essas mudanças se efetivaria somente uma vez em cada 67 lançamentos e que afetaria apenas o processo inicial da entrada em operação dos programas do computador da nave. Os analistas garantiram à NASA que a missão poderia ser executada com segurança sem ser necessária qualquer modificação no software. Dois dias depois a Columbia decolou sem qualquer dificuldade.

Na verdade, o software que dirige a nave compõe-se de oito programas separados. Em conjunto, eles contêm, aproximadamente, 1/2 milhão de instruções independentes que monitoram e regulam operações, abrangendo desde a navegação e o controle do vôo até os sistemas de apoio à vida. Esse software é uma das maravilhas do mundo moderno, empreendimento comparável à construção de uma pirâmide, feita com passos lógicos em vez de blocos de pedra. E, apesar de toda a complexidade de programas como estes, eles diferem apenas em grau daqueles que permitem a um contador realizar seus cálculos em um computador de mesa e a uma criança se divertir em uma máquina de jogos eletrônicos. Desprovidos de programas, os computadores de qualquer tamanho não passam de um amontoado de circuitos inúteis.



Uma criativa representação do poder do software em computadores: o arco-íris, submetido ao programa de instruções, transforma-se em um elaborado tecido xadrez. Programas complexos dirigem todas as aplicações do computador, do processamento de textos à navegação espacial.



O processo de alterar as instruções do ENIAC, ajustando milhares de chaves e conectando centenas de cabos, era tão trabalhoso que estimulou a pesquisa de meios mais convenientes de programação.

Se o computador fosse comparado a um instrumento musical, o software seria a partitura. O termo *software* tem sido usado amplamente desde 1960, com o objetivo de distinguir as instruções que regem o computador de seus componentes físicos, ou *hardware*, a máquina em si mesma. Na computação moderna, essas instruções são gravadas como sinais magnéticos em discos ou fitas, similares àquelas usadas em gravadores: uma parte especial do computador lê esses sinais para a máquina que, além de transformá-los em pulsos elétricos codificados, os armazena em sua memória interna para que dirijam a atividade subseqüente. Mudando-se suas instruções, um computador pode passar da criação de gráficos para a impressão de um contrato padronizado, ou, no caso de uma máquina maior, da manipulação de uma planta arquitetônica para a geração de um mapa meteorológico do mundo.

Essas transformações, no entanto, nem sempre foram executadas com facilidade. Os mastodontes que iniciaram nos anos 40 e 50 a era da computação exigiam quantidades quase iguais de habilidade matemática e energia física para serem programados. A matemática Kathleen McNulty foi uma das primeiras programa-

doras que trabalharam com o ENIAC — Computador e Integrador Numérico Eletrônico (Electronic Numerical Integrator and Computer) —, uma máquina construída pela Escola de Engenharia Moore, da Universidade da Pensilvânia, para calcular tabelas de balística, muito necessárias ao Exército americano durante a Segunda Guerra Mundial. Essas tabelas forneciam o ângulo correto de elevação dos canhões para que um alvo fosse atingido a uma certa distância, com um obus de peso determinado e sob condições variáveis de temperatura do ar e velocidade do vento. Se se utilizasse uma calculadora de mesa eletromecânica, um operador muito bem capacitado demoraria cerca de três dias para calcular uma única trajetória. E o Exército precisava em média de 2 000 a 4 000 trajetórias para cada combinação de canhão e obus. A preparação do ENIAC para executar suas operações exigia um processo alucinante, pois ele consistia de um conglomerado monstruoso de válvulas e cabos de encaixe que formava quarenta painéis, dispostos em forma de ferradura ao longo das paredes de uma grande sala. "Alguém nos deu um calhamaço de desenhos", foi a definição feita por McNulty, anos depois, dos diagramas das conexões dos painéis. "Depois nos disseram", acrescenta ela, "descubram como a máquina funciona e, a seguir, como programá-la."

McNulty e os outros programadores do ENIAC, na maioria mulheres com formação em matemática, estudavam os diagramas e determinavam a disposição correta das centenas de cabos e dos quase 6 000 interruptores e botões. Para deixar a máquina pronta para calcular uma tabela de tiros eram necessários, aproximadamente, dois dias de intenso e meticuloso trabalho manual.

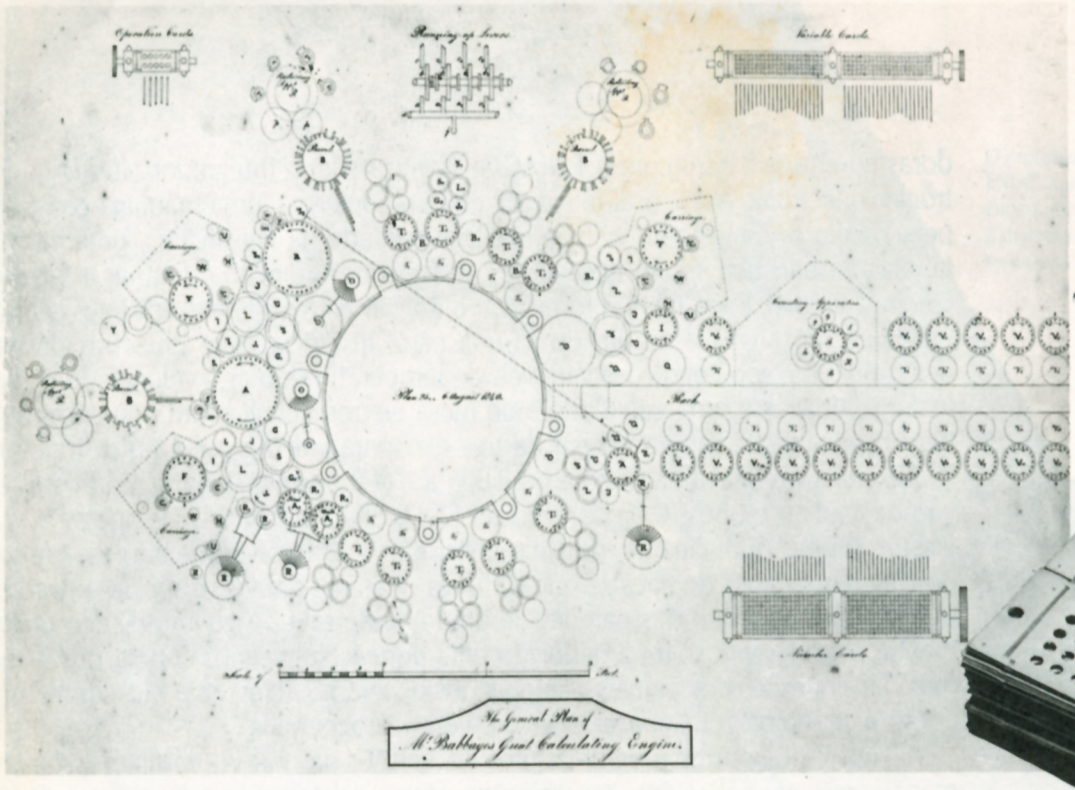
Apesar das frustrações inerentes ao funcionamento do ENIAC e dos seus sucessores do pós-guerra, através do seu uso foi possível se obter grande parte das instruções básicas da programação. Os procedimentos fundamentais criados naqueles primeiros anos se mantiveram como essenciais para a elaboração de sistemas de software que permitem o funcionamento da sociedade moderna.

PREDECESSORES DA ERA DO COMPUTADOR

Criadas inicialmente em função da guerra, essas máquinas projetaram-se em direção ao futuro, embora se baseassem, obviamente, em inspirações de épocas anteriores. O ENIAC, por exemplo, foi modificado com o objetivo de que pelo menos parte de sua programação pudesse ser feita através de cartões perfurados, um método de controle de máquinas que remete ao inventor francês do século XIX, Joseph Marie Jacquard. Em 1804, Jacquard acelerou o processo de tecelagem de elaborados brocados de seda ao introduzir um tear dirigido por uma pilha de cartões perfurados, amarrados um a um para formar um tipo de fita. O trabalho do tear era regulado por pinos de madeira em um "leitor" e esses pinos determinavam quais os fios a serem erguidos ou abaixados para criar o padrão desejado. Cerca de trinta anos depois, o matemático britânico Charles Babbage incorporou essa idéia de Jacquard na concepção de um notável instrumento que chamou de Máquina Analítica.

Muito do que se sabe acerca da máquina de Babbage provém dos textos da genial matemática amadora Augusta Ada Byron, condessa de Lovelace. Filha do poeta Lord Byron, Lovelace possuía uma inteligência extraordinária. Em 1843, ela traduziu um artigo sobre a Máquina Analítica escrito por um matemático italiano, acrescentando muitas observações a respeito do potencial desse invento.

Segundo a concepção de Babbage, a Máquina Analítica destinava-se a resolver





A Máquina Analítica de Babbage, controlada por cartões perfurados, usaria eixos e engrenagens para computar e armazenar valores.

Um instrumento de computação

Em 1822, um gênio britânico, Charles Babbage *(acima, à direita)*, iniciou uma cruzada que o absorveria por quase meio século. Ao morrer, em 1871, deixou mais de 37 metros quadrados de desenhos meticulosos, muitos deles para a Máquina Analítica *(acima)*, um predecessor dos computadores programáveis atuais. Seu primeiro trabalho foi o que ele chamou de Máquina de Diferenças, criada para produzir, automaticamente e com precisão, complexas tabelas matemáticas. Doze anos depois, cercado por dificuldades técnicas, financeiras e políticas, ele abandonou o projeto, passando a trabalhar no Engenho Analítico, uma máquina mais sofisticada, capaz de realizar um amplo espectro de tarefas; para cada tarefa, instruções e dados seriam fornecidos em cartões perfurados. Babbage devotou o resto de sua vida a essa inspiração, mas a precisão requerida para sua fabricação estava além da tecnologia da época, e ela nunca chegou a ser construída.

não só um tipo, mas uma variedade de problemas de cálculo. Na verdade, ele havia concebido e desenhado um computador programável para várias aplicações, incorporando nele muitos dos recursos que caracterizam as máquinas modernas. Embora nunca tenha chegado a construir esse ambicioso invento, seus planos detalhados mostram, por exemplo, que a Máquina Analítica deveria ter uma seção chamada "armazém" — equivalente à memória do computador moderno —, para guardar os números a serem processados. Esse engenho deveria ter também cartões perfurados que poderiam ser trocados à vontade pelo operador, para dirigir os trabalhos.

Lovelace, um dos poucos contemporâneos de Babbage que compreenderam a importância de sua invenção, escreveu que uma máquina assim tão versátil possuía um valor "quase incalculável em seus possíveis resultados finais".

Por quase um século não surgiria nada parecido com a Máquina Analítica, mas a possibilidade de se utilizar cartões perfurados no processamento de dados foi testada vinte anos depois da morte de Babbage. O inventor americano Hermann Hollerith concebeu uma máquina contadora eletromecânica com cartões perfurados para tabular os resultados do censo realizado nos Estados Unidos em 1890. O tabulador foi tão bem-sucedido que Hollerith criou sua própria firma para atender à crescente demanda por sua invenção. A companhia acabou se tornando a International Business Machines — ou IBM —, a corporação que transformou os cartões perfurados em um padrão para a programação de computadores.

UM PIONEIRO DOS COMPUTADORES NA ALEMANHA

Hollerith provou que cartões perfurados e circuitos elétricos podiam trabalhar juntos, desempenhando tarefas úteis. Seu invento, contudo, se restringia à tabulação pura e simples: os cartões perfurados não podiam dirigir cálculos mais complicados. Um engenheiro alemão chamado Konrad Zuse criou um dos primeiros computadores realmente programáveis e construiu várias máquinas de calcular e computadores no fim da década de 30 e começo dos anos 40, para solucionar equações complexas de engenharia. Suas máquinas, controladas por fitas perfuradas feitas com filmes inutilizados — uma engenhosa resposta à falta de papel nos tempos de guerra —, empregavam relés eletromecânicos do tipo então utilizado nos circuitos de comutação telefônica. Zuse foi também um dos primeiros a construir uma máquina operacional — a Z3, em 1941 — que usava o sistema numérico binário de dois dígitos em vez do costumeiro sistema decimal. No sistema binário, qualquer número decimal pode ser representado por uma série de uns e zeros, método um pouco confuso do ponto de vista humano mas perfeitamente adaptado à operação de um circuito elétrico, baseado nos dois estágios: ligado e desligado. De contribuição igualmente importante para o desenvolvimento da computação moderna, esse sistema de dois estados também adapta-se facilmente à codificação falso-verdadeiro da lógica simbólica, o que faz do computador mais que uma simples máquina aritmética.

Em virtude da guerra, o trabalho de Zuse não ultrapassou as fronteiras da Alemanha durante muitos anos. Os fatos mais tarde demonstraram que outros cientistas de ambos os lados do Atlântico perseguiam, naquele momento, o insólito objetivo de criar máquinas programáveis. Vários deles haviam chegado a conclusões semelhantes àquelas a que chegara Zuse acerca da eficiência do sistema numérico binário e da lógica simbólica para a computação elétrica. Entretanto a conversão do sistema decimal para o binário não foi imediatamente universalizada.

Enquanto Zuse dava continuidade a seus trabalhos na Alemanha, cientistas americanos trabalhavam em projetos que exerceriam um grande papel na história dos computadores programáveis. Na Universidade de Harvard, o matemático Howard Aiken e uma equipe de engenheiros da IBM davam os retoques finais no Mark I, uma calculadora de cinco toneladas e 500 mil dólares, controlada por programa e criada para resolver problemas balísticos navais. Como as máquinas de Zuse, o Mark I usava relés eletromecânicos e era dirigido por uma fita de papel perfurada; podia multiplicar dois números decimais de 23 dígitos em três segundos e era suficientemente flexível para efetuar cálculos de uma série de problemas relativos à defesa durante a guerra. Embora logo superado por outras máquinas, o Mark I funcionou no Laboratório de Computação de Harvard até 1959. Talvez sua mais

Uma das poucas pessoas que compreenderam a Máquina Analítica de Charles Babbage foi Ada Lovelace, encarada às vezes como pioneira da programação de computadores. Ao teorizar sobre a seqüência de passos necessários para uma operação de máquinas, ela desenvolveu conceitos ainda hoje utilizados na direção do fluxo dos programas. Por exemplo: ela propôs instruções que repetiriam uma série de passos até que uma condição fosse satisfeita, operação atualmente conhecida como *loop*.



importante função tenha sido a de servir como campo de treinamento para vários pioneiros da computação que mais tarde dariam notável contribuição para a emergente ciência.

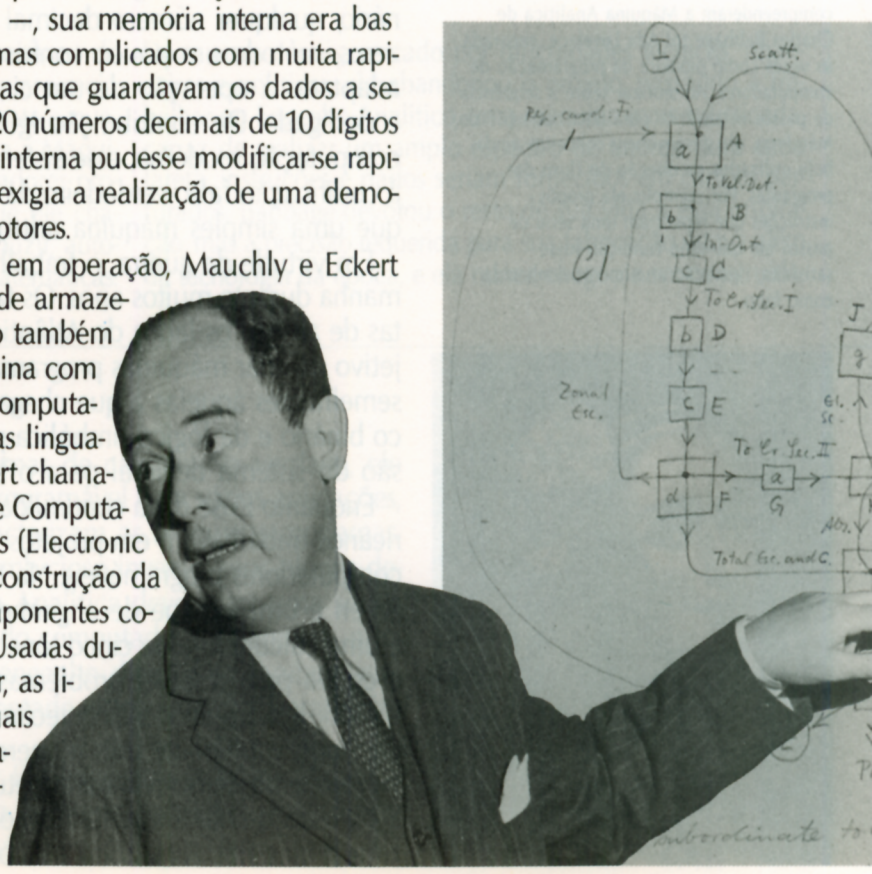
O outro projeto americano era o ENIAC, cujas difíceis exigências de programação se tornariam a dor de cabeça de técnicos como Kathleen McNulty. Criado por John W. Mauchly e seu colega da Escola Moore, J. Presper Eckert, o ENIAC, assim como o Mark I, operava com números decimais em vez de binários. Mas, por usar válvulas e não relés eletromecânicos, era mil vezes mais rápido que a máquina de Harvard.

Ao construir o ENIAC, Mauchly e Eckert tornaram pública sua fé na tecnologia, pois, até então, nenhuma máquina tinha usado mais do que 2 mil válvulas eletrônicas. O ENIAC dispunha de um número quase nove vezes maior. A facilidade com que os bulbos das válvulas queimavam não constituía um problema simples: durante um ano de operação, cerca de 19 mil válvulas do ENIAC tiveram que ser substituídas. Mesmo assim, através do seu uso ficou comprovado que a computação eletrônica era o caminho para o futuro.

A principal vantagem dos computadores eletrônicos, como Mauchly e Eckert perceberam, residia em seu potencial de armazenar grandes quantidades de dados. Os eixos rotativos e engrenagens de computadores eletromecânicos não eram apropriados para a construção de memórias capazes de conter mais que um punhado de dados. Com o advento da eletrônica, tais memórias internas subitamente se tornaram possíveis.

Mas o ENIAC constituía apenas um primeiro passo nessa direção. Concebido apressadamente — dada a premência da guerra —, sua memória interna era bastante limitada. Embora pudesse executar programas complicados com muita rapidez, seus acumuladores — o conjunto de válvulas que guardavam os dados a serem computados — podiam armazenar apenas 20 números decimais de 10 dígitos cada um. E, embora o conteúdo dessa memória interna pudesse modificar-se rapidamente, a troca das instruções para a máquina exigia a realização de uma demorada luta com os cabos e conjuntos de interruptores.

Em 1945, dois anos antes de o ENIAC entrar em operação, Mauchly e Eckert já sonhavam com uma máquina melhor, capaz de armazenar não só grandes quantidades de dados como também as instruções do programa. Sua idéia, uma máquina com programa na memória, mudou a aparência da computação e abriu caminho para o desenvolvimento das linguagens modernas de programação. Mauchly e Eckert chamaram a nova criação de EDVAC, sigla em inglês de Computador Eletrônico Automático de Variáveis Discretas (Electronic Discrete Variable Automatic Computer). Para a construção da memória interna, eles planejavam empregar componentes conhecidos como linhas de retardo de mercúrio. Usadas durante a guerra para temporizar os sinais de radar, as linhas de retardo eram tubos de mercúrio nos quais pulsos eletrônicos ricocheteavam de um lado para outro até serem captados quando se desejasse. O EDVAC também foi idealizado para usar números binários, simplificando, assim, a constru-



Nesta montagem, o matemático John von Neumann, que sintetizou brilhantemente o conceito de armazenar as instruções de um computador em sua própria memória, aponta para um fluxograma. O exemplo, extraído de um de seus textos, simula as colisões de partículas subatômicas.

ção de suas unidades aritméticas. Enquanto Mauchly e Eckert trabalhavam nesse projeto, suas idéias foram sintetizadas, em um trabalho que se tornou um marco histórico, pelo brilhante matemático John von Neumann, húngaro de nascimento, do Instituto de Estudos Avançados de Princeton. Von Neumann, que fora consultor no projeto de ENIAC, elaborou um ensaio de 101 páginas apenas como um esboço preliminar, para ser lido, avaliado e revisado pela equipe do EDVAC. Contudo Herman Goldstine, um membro da equipe, enviou o relatório, sob o título *Primeiro Esboço de um Relatório sobre o EDVAC*, para mais de trinta pessoas, em junho de 1945, citando Von Neumann como único autor. As contribuições de Von Neumann ao relatório certamente eram significativas, em particular no que dizia respeito à operação lógica da máquina. Além disso, sua brilhante reputação contribuiu para que o trabalho exercesse uma grande influência.

A distribuição prematura desse relatório, além de causar uma série de hostilidades, acabou por provocar a negação do pedido de patente solicitado por Mauchly e Eckert. Por uma série de razões — entre as quais o EDVAC —, a dupla abandonou a Escola Moore, no final de março de 1946, para formar sua própria companhia. Com a dissolução da equipe do EDVAC, o processo de desenvolvimento dos computadores sofreu uma repentina paralisação.

Apesar das controvérsias, a idéia de máquinas com programas armazenados internamente tomou corpo, de modo especial na Inglaterra. J. R. Womersley, um dirigente do Laboratório Físico Nacional Britânico, que havia sido convidado pelo governo americano para visitar os projetos do ENIAC e do Mark I, voltou para a Inglaterra com uma cópia do *Primeiro Esboço*, ansioso por organizar um projeto inglês nessas mesmas linhas.

O primeiro cientista que recrutou foi Alan Turing, membro de um grupo de pesquisadores que havia trabalhado durante a guerra em uma máquina decifradora de códigos altamente secreta, chamada Colossus. As contribuições de Turing à computação haviam se iniciado uma década antes, quando, aos 25 anos, publicou um trabalho intitulado *Sobre Números Computáveis*. Esse trabalho vislumbrava um artefato hipotético integrado apenas por uma fita sem fim e uma leitora que leria e transformaria os símbolos marcados na fita. A máquina seria capaz de solucionar qualquer problema legítimo de matemática ou lógica. A intenção de Turing não era inventar um computador, mas fazer a descrição de problemas não legítimos, ou seja, não passíveis de resolução lógica. Contudo, sua máquina hipotética aperfeiçoava certas características dos computadores modernos. A fita sem fim, por exemplo, podia ser vista como um tipo de memória universal interna.

Com a ajuda de Turing, o Colossus foi excepcionalmente bem-sucedido em sua missão de quebrar o código das mensagens codificadas alemãs. Embora houvesse sido criado para ser um computador lógico dedicado (com apenas uma função), o Colossus era na verdade suficientemente flexível para executar muitas outras tarefas. E, por ter sido a primeira máquina a utilizar grandes quantidades de válvulas — entrou em operação dois anos antes do ENIAC —, também serviu para treinar uma equipe de pesquisadores na nova área de computação eletrônica, um corpo de especialistas que conferiria uma boa vantagem à Inglaterra na corrida para a construção do primeiro computador com programa em memória.

Designado por Womersley para estudar o relatório de Von Neumann e desenvolver um computador como o EDVAC, Turing acabou planejando uma máquina que chamou de ACE, isto é, Máquina de Computação Automática (Automatic Com-

puting Engine), e escreveu alguns programas primitivos, usando um código alfanumérico. Mas o projeto ACE deparou-se com uma série de obstáculos burocráticos, o que levou Turing a se sentir frustrado e a se retirar muito antes do lançamento de uma versão de sua máquina em escala reduzida, chamada Pilot ACE, em 1950.

O próximo estágio de Turing foi na Universidade de Manchester, onde uma equipe liderada por Max Newman se encontrava trabalhando no projeto de um outro computador. Newman, um dos mais famosos matemáticos britânicos, havia colaborado no projeto Colossus e se familiarizara com os avanços na área. Mesmo antes de Turing integrar-se na equipe, o grupo de Newman já havia construído um pequeno protótipo de um computador eletrônico com programa em memória, chamado Manchester Mark I.

MEMÓRIA NA TELA

A máquina possuía sua memória interna de armazenagem eletrostática — seis tubos de raios catódicos (CRTs — *cathode-ray tubes*), como os usados nas telas de radares e aparelhos de televisão. A base do tubo apresentava um canhão de elétrons que atirava um raio desses elementos em sua superfície interna, coberta de fósforo, formando pontos luminosos. O raio de elétrons era ligado e desligado em resposta às instruções do computador, produzindo pontos que correspondiam aos zeros e uns do código binário. O conteúdo do tubo mantinha-se graças à renovação da carga de raios de elétrons a cada 1/30 de segundo. Uma série de eletrodos especiais, colocados próximos à superfície do tubo, fazia com que o computador pudesse detectar, ou ler, o local de cada ponto iluminado. Como os sinais também eram visíveis para o observador humano, um tubo tinha como função exibir aos operadores do Mark I uma amostra visual do conteúdo de qualquer tubo.

Inicialmente, o sistema apresentava algumas dificuldades. Por exemplo, quando a chave de força era ligada, "os pontos na tela do tubo começavam a dançar loucamente", segundo o depoimento do engenheiro-chefe F. C. Williams. "Nas primeiras tentativas, era uma dança da morte, que não levava a lugar algum e, o que era ainda pior, não fornecia nenhuma pista sobre o que estava errado." Mas, em 21 de junho de 1948, o Manchester Mark I rodou o primeiro programa em memória do mundo, a procura do maior divisor de um número e, para a felicidade de Williams, "lá, brilhando, reluzente, no lugar esperado, estava a resposta esperada".

Enquanto o Mark I deixava seu rastro brilhante na tela, outro grupo de ingleses, na Universidade de Cambridge, dava os retoques finais em uma máquina que demonstraria a verdadeira capacidade de máquinas com programas na memória. O EDSAC, Calculadora Automática com Armazenamento por Retardo (Electronic Delay Storage Automatic Computer), era, de fato, uma cópia direta do EDVAC de Mauchly e Eckert: no verão de 1946, Maurice Wilkes, diretor do Laboratório de Computação da universidade, visitara os Estados Unidos e assistira a uma série de conferências a respeito das máquinas de programa em memória, realizadas na Escola Moore por Eckert, Mauchly e outros.

Como na proposta desses dois norte-americanos para o EDVAC, a memória interna da máquina inglesa era feita de linhas de retardo de mercúrio. Estas, embora mais lentas e mais caras que os CRTs, podiam armazenar mais dados e eram menos sujeitas a erros. O EDSAC, que chegou dois anos na frente do azarado EDVAC, entrou em operação em junho de 1949 — um ano depois do Manchester Mark I.

Com a introdução das máquinas equipadas com memória interna e com circui-

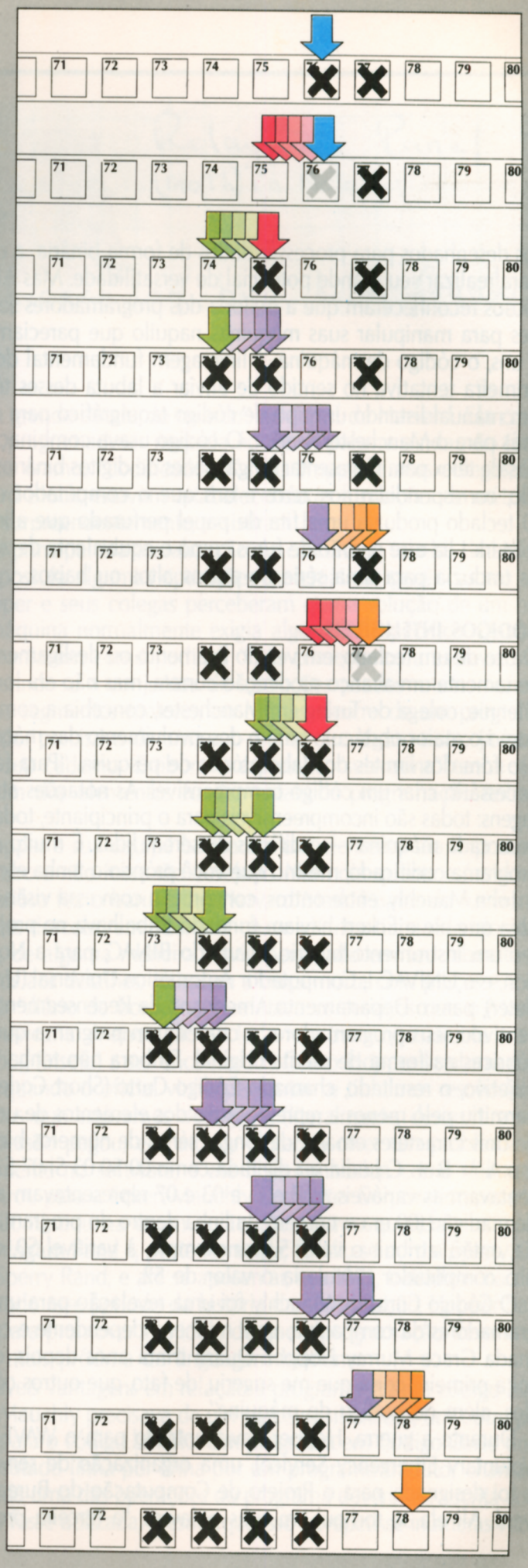


Estados da máquina	Se a leitora ler um espaço	Se a leitora ler um x
ESTADO 1	PARE	Apague; mude para o estado 2; vá para a esquerda
ESTADO 2	Escreva; mude para o estado 3; vá para a esquerda	Continue no estado 2; vá para a esquerda
ESTADO 3	Escreva; mude para o estado 4; vá para a direita	Continue no estado 3; vá para a esquerda
ESTADO 4	Mude para o estado 5; vá para a direita	Continue no estado 4; vá para a direita
ESTADO 5	PARE	Apague; mude para o estado 2; vá para a esquerda

A lógica de uma máquina Turing

Enquanto estudava na Universidade de Cambridge, em 1936, o matemático inglês Alan Turing escreveu uma dissertação intitulada "Sobre Números Computáveis", onde descrevia um instrumento hipotético, chamado posteriormente de Máquina Turing, que antevia o computador programável. Projetada para desempenhar operações lógicas, uma máquina Turing poderia ler, escrever ou apagar símbolos numa fita sem fim. A cada passo de um cálculo, a operação seguinte da máquina seria determinada pela confrontação do símbolo lido com uma lista de instruções.

Neste exemplo, uma máquina Turing é programada para dobrar números inteiros positivos. Suas instruções descrevem cinco estados — a máquina deve executar diferentes operações em cada estado. A máquina começa no estado 1, com a leitora posicionada no começo dos dados, aqui o número 2, representado por dois xis em uma linha da fita *(no alto)*. No final do programa, após dezesseis passos separados, a fita contém duas vezes o número anterior de xis *(embaixo)*.



tos desenhados para processar dados de forma binária, a computação estava apta para realizar seu grande potencial de versatilidade. Mas Alan Turing e outros estudiosos reconheceram que a maioria dos programadores sentiria grandes dificuldades para manipular suas máquinas naquilo que pareciam ser correntes de zeros e uns, o código de máquina, a linguagem fundamental dos computadores. Como primeira tentativa no sentido de aliviar a labuta desses técnicos, Turing escreveu um manual listando um tipo de código taquigráfico para a elaboração de programas para o Manchester Mark I. O código usava combinações de teclas de máquinas de telex para representar longas séries de dígitos binários. As letras TC, por exemplo, correspondiam aos zeros e uns que o computador entendia como "somar". O teclado produzia uma fita de papel perfurado que alimentava o computador: o Mark I lia esse padrão de furos como o equivalente de zeros e uns e, finalmente, os traduzia para uma série de pulsos altos ou baixos.

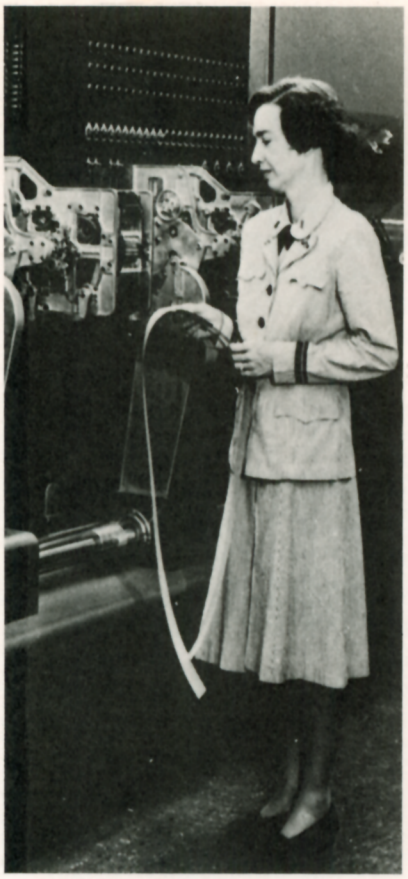
CÓDIGOS INTELIGÍVEIS

O uso de um teclado, em vez do ligamento ou desligamento de chaves, constituiu certamente um avanço na direção correta, mas não era longo o seu alcance. Alick Glennie, colega de Turing em Manchester, concebia a complicação dos programas como o maior obstáculo para o desenvolvimento das máquinas e expansão de seu uso fora dos limites dos laboratórios de pesquisa. "Para facilitar", ele escreveu, "é necessário criar um código compreensível. As notações atuais têm muitas desvantagens: todas são incompreensíveis para o principiante, todas são diferentes — uma para cada máquina — e difíceis de serem lidas. É muito complicado decifrar um programa codificado, mesmo que você próprio o tenha escrito alguns meses antes."

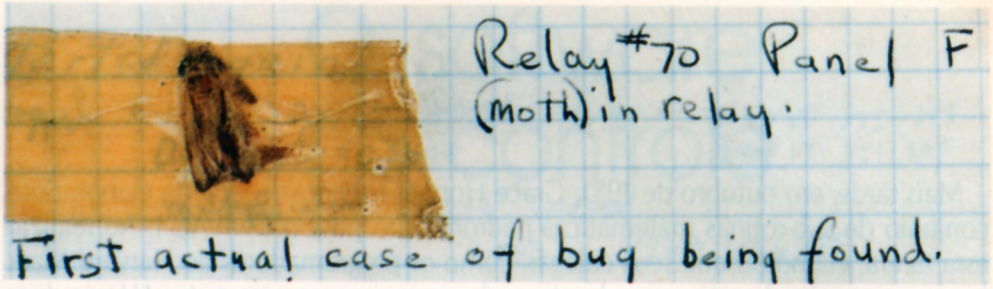
John Mauchly, entre outros, concordava com essa visão. A essa altura, a companhia que ele e Eckert haviam fundado trabalhava no projeto de dois computadores: um instrumento binário, chamado BINAC, para a Northrop Aircraft Corporation, e o UNIVAC I, Computador Automático Universal (Universal Automatic Computer), para o Departamento Americano de Recenseamento. Em 1949, Mauchly incentivou seus programadores a criarem um programa que aceitasse equações algébricas na forma de escrita original. Embora não tenha atingido esse ambicioso objetivo, o resultado, chamado Código Curto (Short Code, ou Short Order Code), permitiu pelo menos a representação dos elementos de uma equação por códigos de dois caracteres em vez de longas séries de números binários. Uma equação como A = B + C podia ser digitada como 00 S0 03 Si 07 S2, onde S0, Si e S2 representavam as variáveis A, B e C, e 03 e 07 representavam as operações de igualdade e adição (00 era o número da linha dentro do programa). O programador poderia, então, atribuir o valor 5, por exemplo, à variável S0, e o valor 3 à variável Si e o computador calcularia o valor de S2.

O Código Curto de Mauchly foi uma revelação para um dos atormentados programadores da companhia, uma mulher independente e com muita iniciativa, chamada Grace Murray Hopper. Quase trinta anos depois, ela declarou: "Acho que foi a primeira coisa que me sugeriu, de fato, que outros códigos poderiam ser usados, além do código de máquina".

Durante a guerra, Hopper havia entrado para o WAVES (Women Accepted for Voluntary Emergency Service), uma organização de serviço voluntário feminino, e foi designada para o Projeto de Computação do Bureau de Artilharia em Harvard. Ali ela se tornou, em suas palavras, "a terceira programadora do primeiro



Grace Murray Hopper, matemática e pioneira da programação, desenvolveu uma grande capacidade de solucionar situações difíceis ao trabalhar, como oficial da Marinha americana, com o computador Harvard Mark I, nos anos 40. Ela descobriu — e documentou o fato no diário do Mark I — o primeiro *bug* verdadeiro (literalmente, inseto; figurativamente, designa um erro de programação) em computadores: uma traça que ficou presa em um dos milhares de relés eletromecânicos dentro da máquina, paralisando todo o trabalho.



computador digital em grande escala do mundo", o Mark I de Harvard. Ela continua: "Nós não éramos programadores naquela época. A palavra ainda não tinha saído da Inglaterra. Nós éramos "codificadores". Mas Hopper e seus colegas, Robert Campbell e Richard Block, estavam estabelecendo as fundações de métodos de programação que, surpreendentemente, resistiram bem.

Como sua atribuição principal era a realização do cálculo de tabelas de balística, o Harvard Mark I possuía algumas operações matemáticas genéricas em sua constituição. Mas Hopper e seus colegas perceberam que a solução de um problema particular na máquina normalmente exigia algo mais específico. Quando isso ocorria, eles se arranjavam como podiam. "Nós começamos a escrever sub-rotinas", ela relata, referindo-se a conjuntos de instruções usados muitas vezes e criados para integrar os segmentos maiores, ou rotinas, dentro de um programa. Quando um programador precisava de uma sub-rotina, que algum outro já havia escrito, a copiava em seu caderno. Embora a expressão "sub-rotina" tenha sido cunhada mais tarde, um dos elementos fundamentais da programação entrava em uso já em 1944.

No ano seguinte, Hopper e sua equipe batizaram um elemento tão importante quanto a sub-rotina, embora não fosse tão desejado. Num dia quente e úmido de verão, uma falha misteriosa fez o barulhento Harvard Mark I parar de funcionar. Investigando a falha, eles descobriram que uma chave elétrica tinha sido bloqueada por uma traça que, de alguma maneira, havia penetrado em seu labirinto de circuitos. Sempre atentos aos detalhes, retiraram a traça morta com pinças e a colocaram no meticuloso "diário de bordo" que relatava os serviços que o Mark I prestava à Marinha americana. A anotação ao lado registrava que aquele era "o primeiro caso real de *bug* (inseto) a ser encontrado". Como Hopper lembrou mais tarde, "dali em diante, quando o oficial vinha perguntar se tínhamos resolvido algum problema, dizíamos que estávamos *debugging* (literalmente, desinsetizando) o computador". A expressão pegou, e o ato de tentar resolver um problema no computador, principalmente no software, é até hoje conhecida como *debug*.

Em 1949, Hopper entrou para a Eckert-Mauchly Computer Corporation para trabalhar no UNIVAC I. Continuou na organização, até mesmo quando a empresa foi comprada pela Remington Rand e, posteriormente, fundiu-se com a Sperry Corporation, formando a Sperry Rand, e até se aposentar, em 1971, trabalhou na divisão UNIVAC. Durante todo esse período da história da computação, Hopper batalhou incansavelmente para promover sua idéia de que a programação deveria ser feita numa linguagem mais próxima da linguagem humana.

Apesar de suas inegáveis vantagens em relação à programação em código binário, o Código Curto de Mauchly precisava de um parceiro silencioso, um outro programa, para traduzi-lo para o código binário. Um programa em Código Curto tinha de ser verificado e executado linha por linha por um programa tradutor chamado interpretador. Assim, cada uma das operações do programa deveria ser escrita completamente sempre que fosse utilizada, mesmo que isso se repetisse inúmeras vezes.

Mais tarde, em outubro de 1951, Grace Hopper foi encarregada de elaborar um conjunto de sub-rotinas matemáticas padronizadas para o UNIVAC I. Ao realizar essa tarefa, ela notou que seus colegas, como os programadores do Harvard Mark I, estavam copiando rotinas e sub-rotinas de um programa para outro. "Havia duas coisas erradas com aquela técnica", ela disse. A cópia de uma sub-rotina para outro programa implicava a mudança de todos os números, ou endereços, que designavam o local onde a sub-rotina iria residir na memória; e como esses endereços normalmente eram blocos de números seqüenciais, sua inserção em outro programa requeria que se somassem esses números a uma outra seqüência preestabelecida, "e os programadores são péssimos somadores", justificou ela.

Também eram péssimos copiadores. Hopper disse: "Era espantoso ver quantas vezes um 4 se transformava em delta, que era nosso símbolo para espaço em branco, ou em A; até a letra B passava a ser o número 13. Qualquer coisa podia acontecer quando os programadores copiavam as sub-rotinas. E, naturalmente, havia ali um aparelho que servia exatamente para somar e fazer cópias precisas".

UMA AJUDA À PROGRAMAÇÃO

Dessa observação acerca da falibilidade humana nasceu o compilador A-0, um programa que ajudava a escrever programas. A partir de um identificador, ou palavra de chamada, ele instruía o computador para buscar uma sub-rotina específica em uma espécie de biblioteca de sub-rotinas, armazenadas em fita magnética, e fazer sua leitura para um trecho da memória. O computador extrairia uma série de sub-rotinas e as "compilaria" com toda a precisão na memória. Acompanhando os computadores UNIVAC a partir do começo dos anos 50 os compiladores tornaram possível a programação automática, assim chamada porque a máquina tinha assumido grande parte da tediosa tarefa de juntar instruções para formar um programa.

Graças à inesgotável energia de Grace Hopper, os compiladores e instrumentos similares de programação ajudaram a assegurar o sucesso comercial do UNIVAC, que se tornou o primeiro computador fabricado e vendido em quantidades significativas. Em 1951, o primeiro UNIVAC foi entregue ao Departamento Americano de Recenseamento, popularizando-se, no ano seguinte, ao prever com precisão a avalanche de votos que asseguriam a vitória de Dwight D. Eisenhower sobre Adlai E. Stevenson, nas eleições presidenciais. Mas isso ocorreu depois que uma equipe de programadores criou o programa que dirigiu a análise feita pelo UNIVAC do padrão de votos por distrito.

Estimulados pelo sucesso do UNIVAC, outros produtores entraram no mercado de computadores. Era evidente que uma revolução tecnológica havia começado. A revolução baseava-se em dois elementos principais: a rapidez com que se desenvolvia a eletrônica e a elaboração de uma linguagem que avançava com segurança em direção ao objetivo de uma comunicação similar à humana. O potencial incalculável da computação, previsto havia mais de um século por Charles Babbage e pela condessa de Lovelace, estava a ponto de se tornar realidade.

O software no controle



Um poder de computação ainda maior que o das gigantescas máquinas dos anos 50 cabe agora numa pequena pastilha de silício do tamanho de uma unha, uma fantástica realização de miniaturização que coloca os computadores ao alcance de quase todos. Além disso, as máquinas modernas efetuam processos nem sequer sonhados quando os computadores dependiam de válvulas de vácuo e cartões perfurados.

O segredo dessa versatilidade está no software, ou programas de instruções, que fornecem à máquina de aço e silício suas múltiplas personalidades. Muito diferente dos programas dos primeiros modelos, que tinham que ser instalados fisicamente no computador, o software que controla o computador moderno é gravado magneticamente em fitas ou discos de metal ou plástico, tornando-se disponível assim que o computador é ligado.

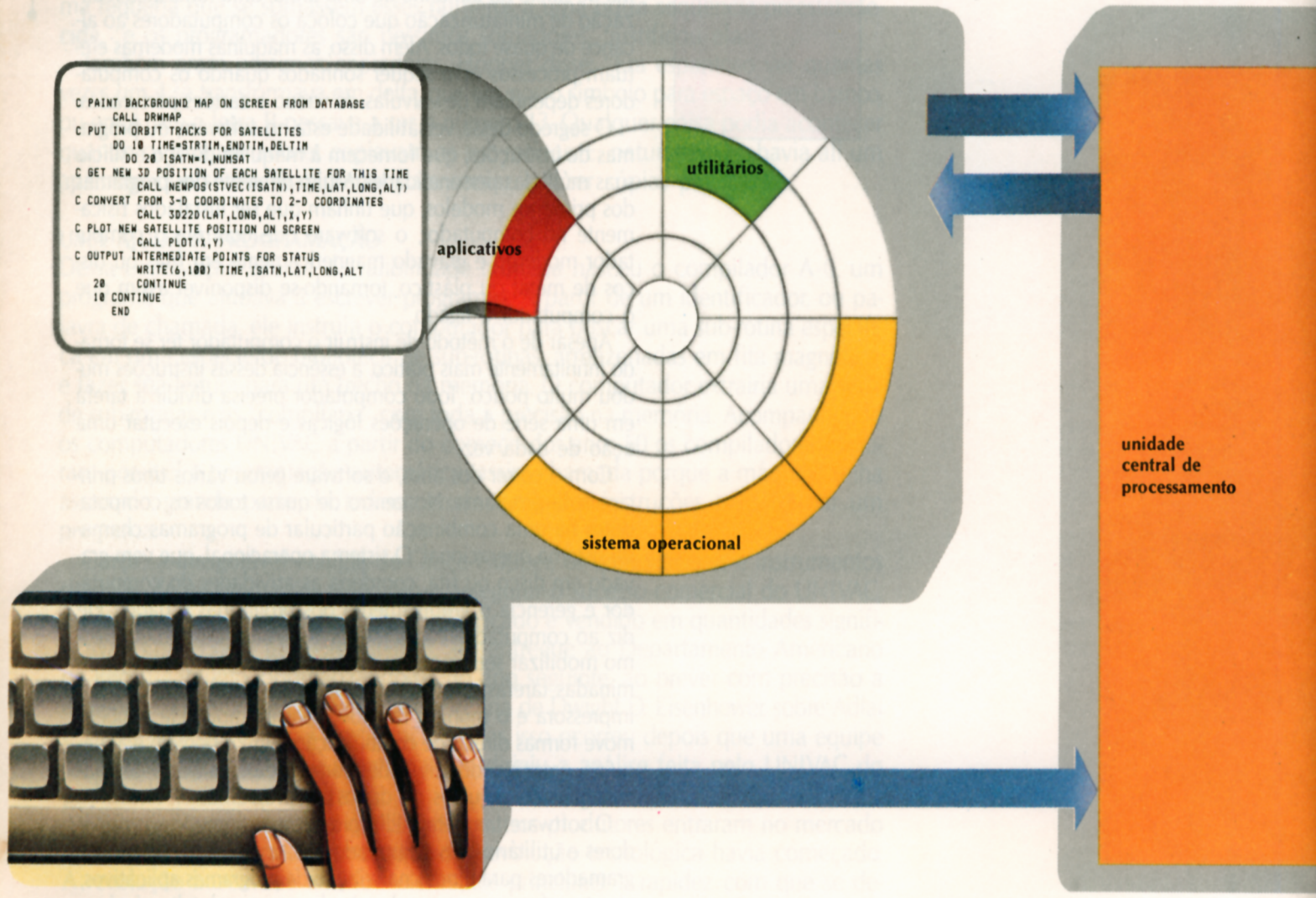
Apesar de o método de instruir o computador ter se tornado infinitamente mais prático, a essência dessas instruções mudou muito pouco. Todo computador precisa dividir a tarefa em uma série de operações lógicas e depois executar uma ação de cada vez.

Com o correr dos anos, o software gerou vários tipos principais de programa. No centro de quase todos os computadores há uma combinação particular de programas chamada sistema operacional. O sistema operacional, que vem gravado em disco ou fita, coordena as atividades do computador e gerencia o arquivamento dos programas e dados. Ele diz ao computador como interpretar instruções e dados; como mobilizar seus recursos físicos para a execução de determinadas tarefas; e como controlar seus periféricos, como a impressora e o monitor de vídeo, por exemplo. Ele também move formas diretas de comunicação entre o homem e a máquina, organizando atividades como o deslocamento e armazenagem de arquivos de dados ou programas.

O software também inclui programas chamados compiladores e utilitários. Os compiladores são utilizados pelos programadores para ajudar na criação de programas aplicativos; eles transformam comandos das linguagens de alto nível — assim chamadas por se aproximarem da linguagem humana — em instruções em linguagem de máquina, os zeros e uns que o computador vai processar como pulsos de eletricidade. Os utilitários executam tarefas triviais, mas muito necessárias, como a ordenação e a fusão de arquivos de dados.

Da entrada à saída: o processo da computação

Toda tarefa computacional envolve a entrada de informação no computador, o processamento dessa informação de maneira determinada e o fornecimento de resultados. O computador realiza a tarefa valendo-se de um complexo conjunto de componentes. Dispositivos de entrada, como o teclado ou o *mouse* (rato), controlado manualmente, permitem que dados e programas sejam fornecidos para o computador; a memória RAM, temporária, mantém os dados e o programa durante o processamento; os dispositivos de saída, como o monitor de vídeo ou impressora, exibem as conseqüências do processamento. Meios externos de armazenamento,

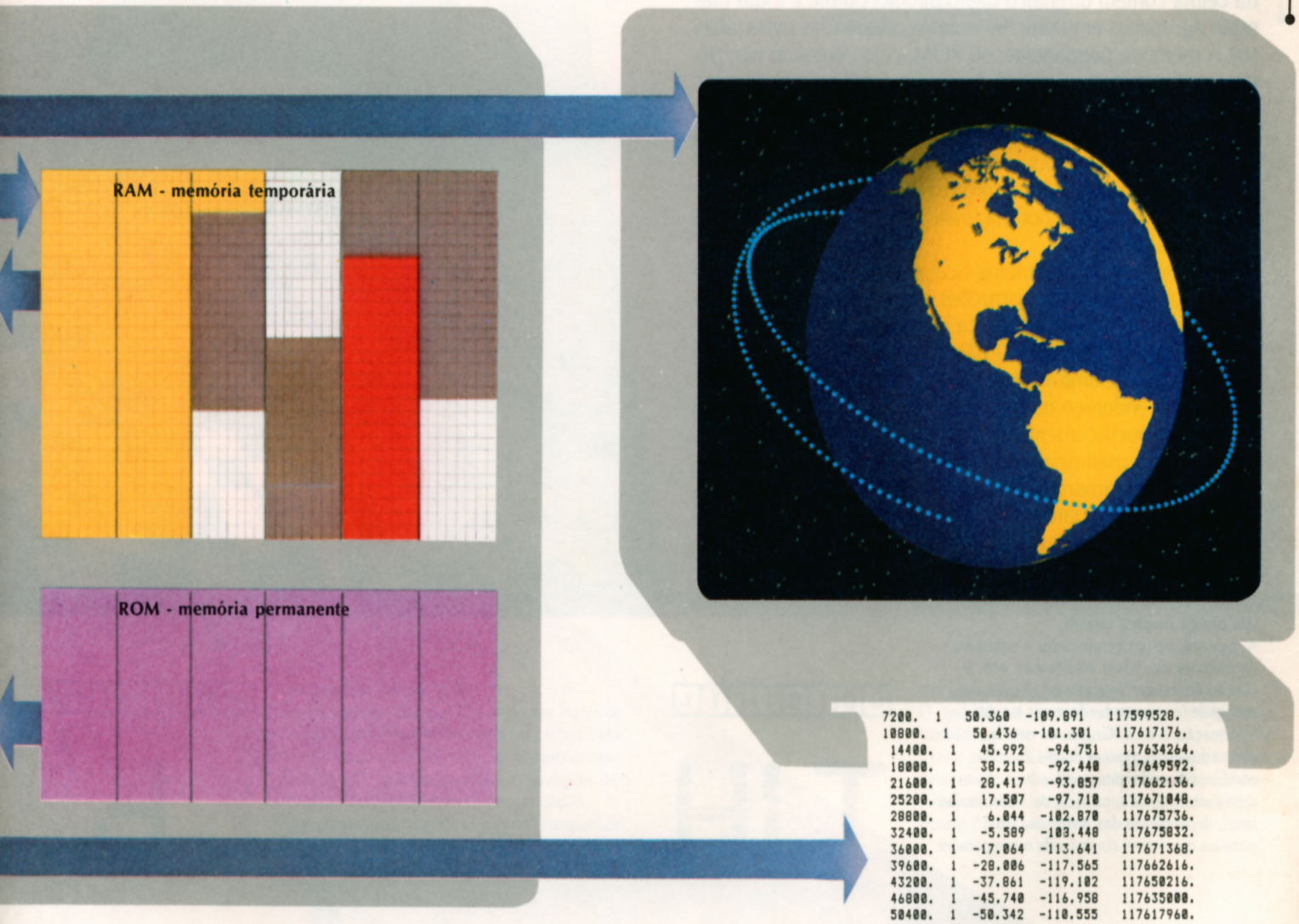


Entrada. O teclado *(acima)* é o instrumento habitual de comunicação com o computador. O software no controle da máquina identifica certas combinações de teclas como instruções a seguir ou dados a processar. Programas simples podem ser digitados, mas programas maiores ou mais complexos são normalmente carregados para a memória do computador por meio de periférico (dispositivo de entrada ou de saída), como o acionador de discos *(acima)*, contendo informações que são transmitidas para a máquina. Os discos são organizados em círculos concêntricos incluindo diferentes tipos de dados e programas gravados como sinais magnéticos que o computador é capaz de entender. Parte do programa de rastreamento de satélites é mostrada acima.

como discos ou fitas, permitem gravações eletrônicas duradouras de programas e dados. Estes combinam funções de entrada e saída: o computador pode copiar a informação gravada em disco para o processamento na memória e pode também copiar os resultados de volta para o disco ou fita.

No centro do sistema de hardware do computador encontra-se a unidade central do processamento (CPU), que coordena o movimento da informação e realiza a computação efetiva. A CPU é feita de modo a reconhecer um conjunto de instruções — os códigos eletrônicos que dizem o que deve ser feito. Todos os programas são formados por seqüências desses códigos; ao ser efetuado um programa, a CPU executa as instruções uma de cada vez, a uma grande velocidade.

Alguns programas fundamentais estão permanentemente na memória ROM (Read-Only Memory, apenas para leitura). Quando o computador é ligado, os programas internos dão à CPU as instruções iniciais, dizendo a ela como encontrar o sistema operacional no disco ou fita e como copiá-lo para a memória temporária. Em seguida, o sistema operacional assume o comando da máquina e estabelece um conjunto de comandos do usuário e respostas do computador que possibilitam o controle da máquina.



Processamento. Dentro da CPU, um processador controla a seqüência das operações, enquanto a unidade lógica aritmética executa cálculos e funções lógicas. Quando um programa está no computador, ele se encontra na memória temporária (RAM), permitindo à CPU buscar as instruções seqüencialmente, uma de cada vez. Programas fixos na memória permanente (ROM) fornecem instruções para dar partida ao processo e para a comunicação com os periféricos de entrada e saída. A ROM também pode conter uma linguagem, como o BASIC, para a programação do usuário, ou programas aplicativos, como um processador de texto.

Saída. Um monitor de vídeo mostra o resultado gráfico da computação. Normalmente os computadores mostram o que foi teclado, bem como suas próprias respostas. A tela muda rapidamente conforme a execução do programa, possibilitando uma interação rápida entre o usuário e o computador. Uma impressora pode produzir uma cópia em papel para registrar os resultados de um programa *(acima)*. A saída pode também tomar outras formas, como voz artificial ou impulsos que controlem o braço de um robô ou uma nave espacial.

Mapeamento da memória

O acesso rápido e preciso aos programas que controlam a CPU e aos dados em processamento faz com que um computador típico execute centenas de milhares de instruções por segundo. Para tornar possível esse acesso, programas e dados são armazenados como seqüências de pulsos eletrônicos, representando os dígitos binários 0 e 1, em células microeletrônicas na memória temporária, ou RAM *(à direita)*. Cada célula contém um único dígito binário, ou bit, e todas elas estão agrupadas em unidades maiores, chamadas bytes *(abaixo)*. A memória permanente, ou ROM, cujo conteúdo não pode ser apagado ou alterado, tem uma configuração similar.

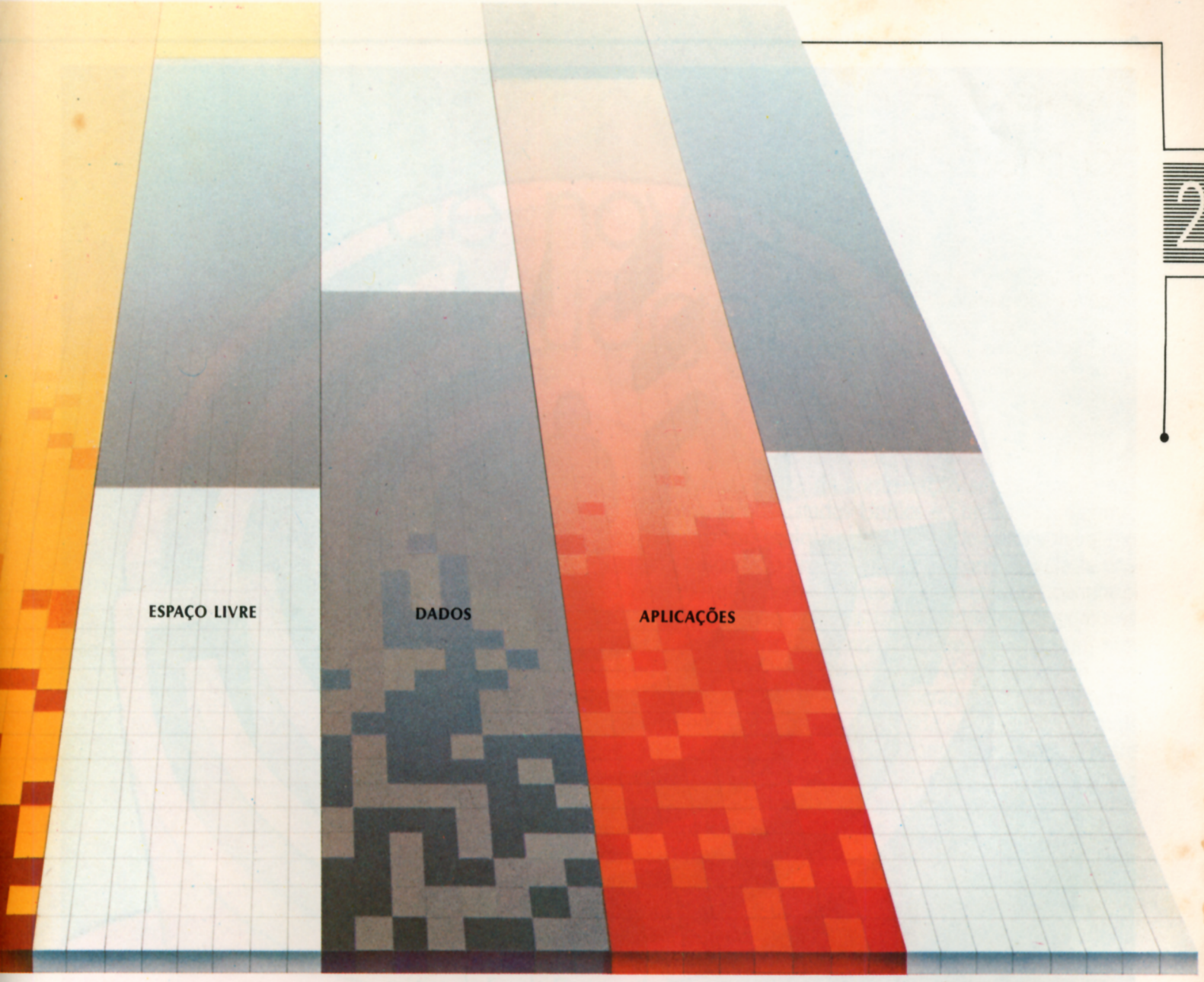
Cada byte de memória apresenta um único endereço. A CPU pode ler o conteúdo de um byte específico da memória enviando um sinal eletrônico para o endereço apropriado. Pode também transmitir uma informação para qualquer endereço na memória temporária; essa informação substituirá a que estava ali contida.

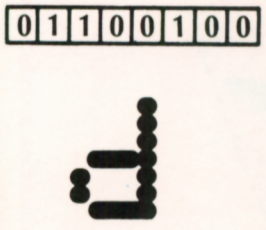
Um programa normalmente fica na memória de um bloco de bytes, com listas de instruções armazenadas em endereços consecutivos. Isso simplifica o controle do fluxo do programa, uma vez que a CPU não necessita ser informada do endereço da próxima instrução; ela passa automaticamente ao próximo endereço na seqüência, a não ser que tenha sido instruída especificamente para passar a outro endereço.

Utiliza-se a memória temporária também para reter os dados em processamento. O programa lê os dados, executa as transformações e guarda-os novamente na memória, registrando todos os endereços relevantes.

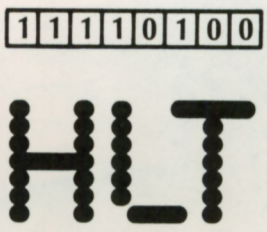


Um código simples, versátil
A memória de um computador é composta de milhares de chaves eletrônicas, com a função de indicar dois estados: ligada ou desligada, correspondendo a um bit de informação, 1 ou 0. Grupos de oito bits, chamados bytes, podem conter 256 (2^8) combinações diferentes de 1 e 0, representativas de uma série de símbolos *(ao lado)*, dependendo das instruções do software que estiver controlando o computador.

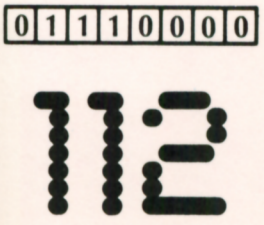




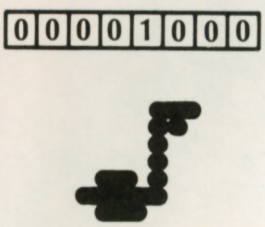
Um caractere em um byte
Um código especial, o ASCII (sigla em inglês de Código Padrão Norte-Americano para Intercâmbio de Informações) nomeia com um número decimal, entre 0 e 127, as letras do alfabeto, os dígitos de 0 a 9 e um conjunto de sinais de pontuação. Traduzidos para os dígitos binários do computador, os números codificados formam um padrão de comunicação entre computadores.



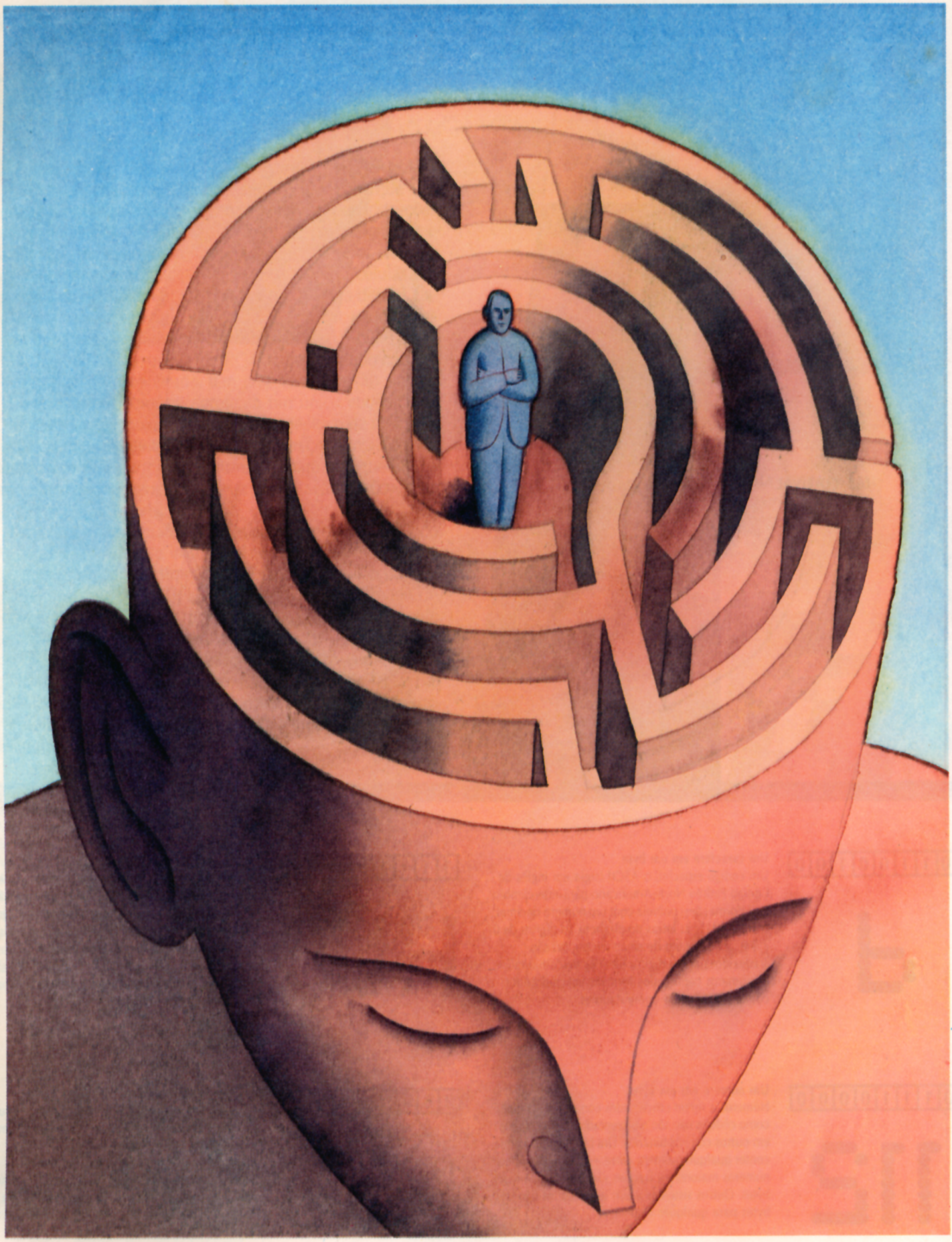
Um comando em um byte
O conteúdo de um byte, ou de uma série de bytes, também pode ser interpretado como uma instrução para a unidade central de processamento. Códigos de instrução variam de acordo com a CPU; o código mostrado aqui faria com que um tipo de CPU ficasse paralisado (HALT em inglês).



Um número em um byte
Números a serem processados como valores e não como códigos não são representados em ASCII, mas em outros códigos binários. Um byte sozinho pode conter qualquer número entre −127 e +127; para números maiores ou frações são necessários mais bytes. Outros sistemas usam um número fixo de bytes para qualquer valor.



Um símbolo em um byte
Um dos mais poderosos atributos do software é sua capacidade de conferir qualquer significado a uma determinada configuração de bytes. Além dos códigos correspondentes a letras, números e instruções, os bytes podem ser usados pelo software para conter elementos como notas musicais ou tonalidades de cores.



Desenvolvimento de novas linguagens

2

Na metade da década de 50, com a ciência da computação se enraizando nas universidades e institutos de pesquisa nos Estados Unidos e Europa, a programação entrou em um período de progresso explosivo. Essa onda de desenvolvimento não desprezou o que havia sido descoberto anteriormente; ao contrário, ela se projetou a partir das bases já estabelecidas: compiladores e interpretadores continuaram sendo essenciais para a programação de todo computador, assim como as chamadas linguagens *assembly* (linguagens de montagem), que requerem um profundo conhecimento do hardware da máquina. Mas, apesar de continuarem exercendo suas funções, esses instrumentos fundamentais de programação gradualmente se tornaram invisíveis. Um novo tipo de linguagem, baseada em sentenças e mais fluida do que a linguagem assembly, chegou para servir como intermediário entre o homem e suas máquinas.

Com o advento das linguagens de alto nível, os programadores podiam dedicar mais tempo à resolução dos problemas, deixando de lado a questão crucial de como a máquina executaria seu trabalho. Com essas linguagens foram dados também os primeiros passos no sentido de se elaborar programas que extrapolassem os limites dos laboratórios de pesquisa e dos escritórios de folhas de pagamento.

Duas figuras importantes para esse desenvolvimento foram Thomas Kurtz e John Kemeny, ambos doutorados em Princeton e membros do departamento de matemática da Escola Dartmouth. Em 1945, quando ainda era estudante, Kemeny foi a Los Alamos, no Novo México, trabalhar no supersecreto Projeto Manhattan que construiria a bomba atômica. Seu superior era John von Neumann, cujas idéias sobre computação teriam influência duradoura nesse campo.

Como Von Neumann, Kemeny era um genial imigrante húngaro, com grande talento para a matemática e um jeito especial de usar a lógica para vencer o sistema. Em 1942, com apenas 16 anos e conhecendo muito pouco a língua inglesa, Kemeny fez um exame de aptidão verbal preparado pelos professores da escola de segundo grau Washington Heights, em Manhattan, Nova York. Anos mais tarde, relatou: "Eu não tinha nenhum vocabulário: entendia apenas umas poucas palavras em cada questão — o suficiente para perceber que havia um padrão lógico de respostas às questões de múltipla escolha. Descobri o código e tive uma das notas mais altas de Nova York".

Depois da guerra, Kemeny trabalhou como assistente de Albert Einstein no Instituto de Estudos Avançados em Princeton enquanto terminava seu trabalho de doutorado. Sem conseguir emprego como professor de matemática, sua principal paixão, ele passou um período ensinando filosofia analítica. Depois, em 1953, com apenas 27 anos, foi contratado como titular do departamento de matemática de Dartmouth.

Quando Kemeny e Kurtz começaram sua parceria, Dartmouth, uma pequena escola superior em Hanover, New Hampshire, sequer possuía um computador. Ainda assim, em menos de dez anos, graças aos esforços dos dois colegas, a faculdade podia se orgulhar não apenas de possuir um notável centro de computação como também de ser o local de nascimento do BASIC, uma linguagem de programação de grande popularidade e importância.

AS DIFICULDADES DO PROCESSAMENTO EM LOTES

Nos primeiros tempos, contudo, Kemeny e Kurtz eram obrigados a rodar seus programas de matemática em qualquer computador que encontrassem. O mais próximo, um IBM 704 com a capacidade — fenomenal para a época — de armazenar 8 192 palavras de 36 bits em sua memória interna, ficava a 135 milhas dali, no Centro Regional de Computação da Nova Inglaterra, do Instituto de Tecnologia de Massachusetts (MIT). O acesso direto ao computador, no entanto, lhes era vedado em virtude do confuso sistema conhecido como processamento em lotes. Muito raros e caros, os computadores não eram postos à disposição de qualquer pessoa. Os programadores eram obrigados a entregar cartões IBM, perfurados com suas instruções, para um operador autorizado do centro de computação. Esse intermediário alimentava o computador com os cartões em grandes lotes, freqüentemente contendo as centenas de programas separados de muitos programadores. Começava, então, a espera. O computador não fornecia as respostas impressas até que todos os programas do lote tivessem sido processados, o que podia demorar até 24 horas. E, muitas vezes, a resposta era frustrantemente incompleta: algo tão banal como a falta de um parêntese era suficiente para interromper a execução do programa e o impresso continha apenas uma misteriosa mensagem de erro. O programador deveria corrigir aquela linha no programa e esperar, às vezes até o dia seguinte, pela oportunidade de incluí-lo em um outro lote que seria introduzido na máquina. Se houvesse um outro erro, em outra linha do programa, o processo teria que ser novamente recomeçado. Mesmo bons programadores costumavam demorar até duas semanas para encontrar e corrigir todos os *bugs* de um só programa.

Como se o processamento em lotes já não se constituísse em um monstruoso trabalho, a programação do 704 ainda tinha que ser feita em linguagem assembly, um pouco mais elaborada do que aquela em que tudo deveria ser escrito com os zeros e uns do código binário. Sempre que um programa falhava, a única maneira de descobrir seus erros era ler os chamados *dumps* de memória, longas séries de números impressos, de interpretação particularmente difícil.

Kemeny e Kurtz sofreram muito com o 704, aprendendo a linguagem assembly e levando as intermináveis listagens de dumps de memória. Apesar de Kemeny alimentar esperanças de introduzir a computação em Dartmouth, ele achava que jamais conseguiria transmitir esse conhecimento para seus colegas e alunos, a menos que o processo se tornasse consideravelmente mais fácil.

No começo dos anos 60, os dois professores já haviam conseguido um certo progresso. A escola adquirira um pequeno computador em junho de 1959, um Royal McBee LGP-30 com cerca da metade da memória do 704. Kurtz foi designado chefe do novíssimo centro de computação e, nos anos seguintes, os dois professores, juntamente com alunos, realizaram experimentos na tentativa de criarem linguagens simples que pudessem ser implementadas no LGP-30. Um estudante, ainda não for-

mado e sem qualquer experiência em computação, criou uma linguagem e um compilador chamado DART. O feito desse estudante confirmou a crença de Kemeny e Kurtz de que a computação estava ao alcance dos universitários. Parecia, enfim, ter chegado a hora de realizar o plano que ambos acalentavam havia muito tempo. Tratava-se de uma proposta radical, mas os dois conseguiram o apoio da escola: Dartmouth passaria a ensinar os rudimentos da computação a todos os estudantes, tanto os das ciências exatas como os da área de humanas.

Esse objetivo era extraordinário, e o método imaginado por Kemeny o era ainda mais. Em oposição ao que se praticava em outras universidades, onde os professores se contentavam com o ensino da computação através de aulas expositivas, Kemeny pretendia que os estudantes aprendessem computação pela prática: fazendo programas. Entretanto, dois grandes problemas precisavam ser resolvidos antes. Primeiro, mesmo que os estudantes não tivessem que sair do próprio centro de computação da escola para rodar seus programas, só contariam ali com um único equipamento, e não poderiam furtar-se à frustração e ao tédio aparentemente inevitáveis do processamento em lotes. Provavelmente os estudantes de engenharia e matemática suportariam a espera de horas, ou até de dias, pelos resultados, mas os estudantes de línguas ou história com certeza se revoltariam.

Segundo, se a perspectiva do processamento em lotes não era desoladora a ponto de abandonarem o projeto, a de ensinar uma linguagem de programação aos estudantes de ciências humanas parecia sê-lo. Algumas linguagens de nível mais alto já haviam surgido. A primeira entre elas, e a mais amplamente utilizada na época, era a FORTRAN, desenvolvida no fim dos anos 50 por uma equipe de programadores da IBM. Projetado para expressar fórmulas matemáticas e científicas, o FORTRAN, sigla de Formula Translator (Tradutor de Fórmulas), não era fácil de ser dominado, nem mesmo por engenheiros. Por exemplo, DO 150 I = 1,10 é o começo de um loop no programa que dá ordem ao computador para que ele execute repetidamente as instruções seguintes a essa diretiva, até a instrução de número 150 inclusive, inicialmente com I valendo 1 e com I sendo incrementado a cada vez até atingir o valor 10. Se o programador inadvertidamente teclasse um ponto no lugar da vírgula, a diretiva seria interpretada como a atribuição do valor 1,10 a uma variável chamada DO1501. Essa diretiva seria considerada válida, "legal" no jargão da computação, mas o programa não rodaria.

Os estudiosos de Dartmouth não aceitavam a intimidação imposta por esses obstáculos. Começaram, então, a procurar uma alternativa para o processamento em lotes, o que, de fato, conseguiram, através de uma idéia proposta inicialmente no MIT, em 1959, chamada time sharing (tempo compartilhado).

Time Sharing envolvia a ligação de vários terminais de teclado ligados a um *mainframe*. (Na época, *mainframe* designava simplesmente o processador central, ou principal, de um computador, que dirige as atividades da máquina; atualmente, esse termo é usado para se referir a um computador grande e poderoso, distinguindo-o dos de mesa ou menos poderosos.) Programas especiais fazem com que o mainframe mude sua atenção de um terminal para outro, executando uma pequena parte do trabalho exigido por cada um dos usuários num determinado tempo (uma "fatia de tempo", normalmente medida em milésimos de segundo). Com a incrível rapidez de processamento do mainframe, parecia que cada programador recebia toda a atenção do computador; na verdade, é claro, ele estava apenas compartilhando do seu tempo. Kemeny e Kurtz decidiram, en-

tão, que era exatamente isso que o sistema de Dartmouth estava necessitando.

Em seguida, dedicaram-se à criação de uma linguagem de programação que pudesse ser aprendida rapidamente pelos estudantes. A linguagem por eles imaginada consistia de palavras tiradas do próprio inglês, incluindo comandos como SAVE, RUN e LIST (guardar, rodar e listar). Sua sintaxe seria similar à do FORTRAN, mas muito mais fácil de ser compreendida. Por exemplo, para fazer com que o computador escrevesse uma frase, o programador teria simplesmente que teclar a instrução PRINT (imprimir) e, a seguir, a frase entre aspas. Essa nova linguagem deveria ser tão pouco exigente a ponto de, segundo Kemeny, "os estudantes poderem utilizá-la com apenas três horas de treinamento".

UM PROJETO PARA TODAS AS APLICAÇÕES

Algumas das idéias desenvolvidas por Kemeny e Kurtz eram novas, como, por exemplo, a instrução INPUT. Baseados na premissa de que os estudantes estariam teclando seus programas em terminais, ao invés de entregar cartões perfurados a um intermediário, eles projetaram uma linguagem especificamente interativa, ou seja, ela permitiria que o programador modificasse o programa enquanto este estava sendo executado, interrompendo-o e entrando com novos dados ou instruções (*input* significa "colocar dentro"). Diferentemente do FORTRAN, onde o programador deve distinguir entre números inteiros e os chamados números de ponto flutuante (números como frações, que o computador representa internamente movendo o ponto decimal), a linguagem de Kemeny e Kurtz possibilitava que todos os números fossem tratados da mesma maneira. Assim, um estudante não precisava lembrar que a divisão do número 2 pelo número 3 produz um número de tipo diferente, isto é, não inteiro. Como Kurtz definiu mais tarde: "Quase tudo que você teclasse como sendo um número era considerado legal".

A nova linguagem recebeu um nome apropriado: BASIC ou, por extenso, Beginner's All-Purpose Symbolic Instruction Code (Código de Instrução Simbólica para todos os Propósitos do Principiante). E ela era, efetivamente, projetada para todas as aplicações. Utilizando-se dela, um estudante de sociologia poderia escrever um programa para ordenar os resultados de uma pesquisa estatística, um estudante de francês poderia escrever um programa que conjugasse os verbos regulares e um de engenharia poderia escrever um programa para calcular as características de resistência dos diversos materiais para a construção de uma ponte.

No verão de 1963, Kemeny começou a trabalhar com um compilador para uma versão preliminar do BASIC. Naquele outono, dois estudantes começaram a projetar e codificar o sistema operacional para as máquinas que o projeto iria utilizar, um General Electric 225 e um Datanet-30.

Finalmente, no dia 1º de maio de 1964 todos esses esforços se juntaram. Às quatro horas da manhã, num porão dentro do campus, Kemeny e seus colegas sentaram-se à frente de três terminais de telex e começaram a digitar programas. Kemeny descreveu mais tarde: "Nós já tínhamos feito aquilo muitas vezes antes, mas aquela foi a primeira vez que os programas funcionaram. Foi o nascimento simultâneo do BASIC e do Time Sharing". Em junho, os três terminais tinham se multiplicado em 11 e, poucos meses depois, vinte terminais estavam em uso.

O Time Sharing logo se tornou o método padrão de computação em grandes organizações, tais como universidades, agências do governo e empresas, embora normalmente utilizassem simultaneamente outras linguagens. O BASIC continuou

sua carreira, tornando-se a linguagem básica de computação, resultado da aceitação quase unânime como linguagem interna dos microcomputadores que dominaram o cenário no fim dos anos 70. Na metade da década de 80, vários milhões de crianças americanas em idade escolar haviam aprendido a linguagem BASIC. E em todo mundo há mais gente com fluência nessa linguagem do que em todas as línguas escandinavas — norueguês, sueco e dinamarquês — somadas.

Linguagens de nível mais alto, raras no começo dos anos 60, acabaram por aparecer às centenas, se incluirmos os diversos "dialetos". Isso levou a várias tentativas no sentido de se criar uma língua universal para programação. Nenhuma foi inteiramente bem-sucedida; em programação, ao que parece, a diversidade propicia mais satisfação. São quase doze as linguagens mais amplamente utilizadas, e cada uma se destina a uma aplicação específica. BASIC, por exemplo, ainda é a linguagem comum para se escrever programas simples, especialmente em microcomputadores. FORTRAN, com suas detalhadas exigências numéricas, sobrevive como um clássico da programação, como se fosse latim ou grego, em termos de software bastante apropriado para aplicações matemáticas, científicas ou de engenharia. Uma outra língua, COBOL, sigla de Common Business Oriented Language (Linguagem Orientada para Negócios Comuns), foi criada em 1960, por um conjunto de fabricantes de computadores e usuários, para servir como linguagem principal de processamento de grandes massas de dados do governo e das empresas. Em COBOL, ao contrário do que ocorre na maioria das linguagens, a descrição dos dados deve estar separada da parte de instruções do programa — uma convenção que permite usar a mesma descrição de dados para diversos programas.

Outras linguagens são ainda mais especializadas. Por exemplo, o LOGO (derivado do conceito grego de "palavra") é principalmente usado para ensinar crianças em idade escolar a programar. O principiante pode controlar o movimento de um caractere, chamado de tartaruga, pela tela. Teclando comandos como FRENTE 60, ESQUERDA 45, FRENTE 75, o jovem programador faz com que o computador desenhe uma linha de 60 unidades, faça uma curva de 45 graus para a esquerda e desenhe outra linha de 75 unidades.

Outra linguagem importante, fruto, como o COBOL, do trabalho de um comitê, desta vez de caráter internacional, é o ALGOL, sigla de Algorithmic Language (Linguagem Algorítmica); algoritmos são os procedimentos, passo a passo, para a resolução de um dado problema. A primeira versão, ALGOL 58, foi projetada durante um encontro intensivo de oito dias, na primavera de 1958. Cientistas de computação dos Estados Unidos, entre eles John Backus, um dos criadores do FORTRAN, reuniram-se em Zurique com seus colegas europeus da área de computação para dar forma ao que deveria ser uma linguagem universal, não limitada a um tipo de equipamento, do mesmo nível que o FORTRAN. Na verdade, o FORTRAN poderia ter sido designado como linguagem universal, se não estivesse tão intimamente associado à IBM e às máquinas dessa companhia. O ALGOL 58 e seu sucessor, o ALGOL 60, tiveram uma acolhida polêmica. Nunca chegaram a ter a ampla aceitação que o FORTRAN acabou experimentando, mas exerceram influência significativa no desenvolvimento de muitas linguagens subseqüentes.

A linguagem Pascal, desenvolvida no final da década de 60 pelo cientista suíço Niklaus Wirth, está entre as linguagens criadas como tentativa de se melhorar o ALGOL. Em Pascal, o programador deve definir todas as variáveis numa seção separada no começo do programa. Como essas definições têm que ser declaradas ex-

plicitamente, programas em Pascal contêm relativamente poucos erros e são mais facilmente compreendidos ou alterados por outras pessoas que não o autor. Essa abordagem disciplinada faz com que o Pascal seja especialmente apropriado para a criação de programas extensos.

Apesar de proliferarem as linguagens para se escrever programas, o software não acompanhou os avanços técnicos dos computadores, propriamente ditos. Enquanto o hardware torna-se constantemente mais poderoso e mais barato, o software encarece a ponto de, freqüentemente, responder por 80% do custo total dos grandes sistemas de computador. Normalmente os engenheiros afirmam que o software é um fator típico de limitação, o funil que impede o surgimento de novos equipamentos e aplicações, principalmente por causa de sua complexidade sempre crescente. Muitos programas são tão grandes que desafiam a capacidade de entendimento até de cientistas da computação. Por exemplo, o pacote de software que controla o tráfego aéreo de vinte locais nos Estados Unidos e um na Inglaterra é composto de mais de 600 000 instruções diferentes para o computador. Dessas instruções, nada menos que 39 203 são desvios condicionais, pontos onde o controle do programa pode seguir por uma de duas direções possíveis. Como resultado, o número total de encaminhamentos possíveis do programa é astronômico, próximo do número 10 seguido por 11 800 zeros. (Em BASIC, um desvio condicional poderia estar escrito desta forma: IF N $\geqslant$ 4 THEN 220. Essa expressão ordena ao computador que pule para a instrução 220, se a variável N for maior ou igual a 4; contudo, se N for menor que 4, o computador continua automaticamente na linha seguinte do programa.)

E a complexidade do software deriva, apenas em parte, da dificuldade dos problemas a serem resolvidos. Ela é inerente à própria natureza do processo de programação, que requer grande paciência e precisão ao traduzir os problemas reais em termos que o computador possa compreender.

AMBIGÜIDADE ABOLIDA

Em programação, nada pode ser deixado ao acaso. Fascinados pela aparente mágica dos computadores, os não-programadores tendem a se esquecer de que as ações que eles encaram como básicas devem ser explicadas à máquina a um nível de detalhe exasperante. Como um cientista da computação apontou, "se um homem pode resolver como fazer alguma coisa, o fato não parece ser um grande problema. Mas o homem usa habilidades maravilhosas, como raciocínio associativo e reconhecimento de padrões".

Toda ação projetada para o computador deve ser dividida em seus componentes mais elementares para que se produza um procedimento conhecido como algoritmo. Em sua forma mais simples, um algoritmo não difere muito de uma receita de bolo. Mas um programador de computadores deve especificar passos que a mente humana tende a ignorar. Uma receita de bolo, por exemplo, poderia dizer que é necessário usar três ovos, sem especificar que os ovos devem ser frescos, não podem ter sido cozidos e que devem ser quebrados para que se use somente a parte interna deles.

Quando centenas ou milhares de instruções, cobrindo todas as possibilidades, têm que ser escritas, o custo sobe naturalmente e os bugs, ou erros de programação, começam a aparecer. Esses bugs que aparecem todos os dias são o que o cientista holandês Edsger W. Dijkstra chamou de "aborrecimentos menores em gran-

des quantidades". Dijkstra, um pesquisador da Burroughs Corporation, é um dos teóricos de programação mais influentes em todo o mundo e crítico ferino do que ele considera como prática leviana de programação. Há anos ele vem sustentando que a maior parte dos erros pode ser evitada, pois provém da maneira desleixada com que a programação tem sido tratada, tornando-se uma arte intuitiva e vulgar, em vez de se transformar em uma ciência rigorosa.

Numa época em que o processamento de textos já é corriqueiro, o método escolhido por Dijkstra para divulgar suas controvertidas opiniões é ostensivamente anticonvencional. Em sua casa, perto da cidade industrial de Eindhoven, ele publica periodicamente um boletim intitulado EWD (das iniciais de seu nome). Ele o escreve a mão, tira fotocópias e as distribui para uns vinte ou trinta colegas.

A contribuição essencial de Dijkstra para a teoria da programação aconteceu em 1968, quando ele contava 38 anos de idade. Num tratado curto, intitulado *Notas sobre Programação Estruturada*, ele argumenta que a maior parte dos programas é desnecessariamente complexa por falta de uma estrutura matemática rigorosa. De fato, muitos outros membros da comunidade de computação vinham se preocupando havia anos com o estado da arquitetura de software. Com uma freqüência cada vez maior, novos sistemas de software destinados ao governo e a empresas particulares, com custo de milhões de dólares, ficavam prontos com meses de atraso e ainda continham milhares de erros ao serem, finalmente, liberados. Enquanto Dijkstra compunha seu tratado, a OTAN — Organização do Tratado do Atlântico Norte — patrocinava uma conferência mundial para tratar do que ela chamava a "crise do software".

UM APELO AO RIGOR MATEMÁTICO

O principal foco das críticas de Dijkstra era um comando de programação conhecido como desvio incondicional, a instrução GOTO — VÁ PARA. Esse comando, que transfere o controle de um ponto do programa para outro, é extremamente útil, mas atrapalha o entendimento do programa (por humanos, é claro; os computadores não têm dificuldade alguma em seguir instruções GOTO). É como se o leitor de um romance estivesse constantemente sendo instruído para saltar para a frente e para trás nas páginas do livro. Ao quebrar o fluxo lógico, os comandos GOTO fazem com que seja praticamente impossível alguém, além do autor, compreender os mecanismos internos do programa, a ponto de poder modificá-lo sem introduzir erros em locais inesperados. Como um cientista da computação definiu, "se você colocar muitos GOTOs em um programa, este parecerá um espaguete".

Para substituir esses comandos, Dijkstra propunha o uso de três tipos de estrutura de controle: seqüenciamento simples, ou fazer uma coisa e depois fazer a próxima; alternância, onde um programa realiza uma de duas ou mais ações possíveis; e repetição, na qual um programa realiza continuamente uma mesma operação enquanto uma determinada condição permanecer verdadeira. Ele insistia que, com o uso dessas três estruturas, o programador poderia dispensar todos os GOTOs.

A princípio, as idéias de Dijkstra enfrentaram o sarcasmo de programadores nada dispostos a abandonar os GOTOs. Mais tarde, no começo da década de 70, uma equipe da IBM, coordenada pelo cientista Harlan Mills, usou a programação estruturada para criar um banco de informações para o jornal *The New York Times*. O trabalho correu extraordinariamente bem, e o programa, quando pronto, quase não continha erros.

Desde então, a programação estruturada tem exercido grande influência no desenvolvimento de software, desde programas para computadores domésticos até projetos de segurança nacional de muitos milhões de dólares. Várias das principais linguagens de programação, tais como a Pascal, agora incorporam os radicais princípios de Dijkstra.

Um marco da aceitação popular ocorreu em 1975, quando John Kemeny e Thomas Kurtz, autores da mais democrática linguagem, o Dartmouth BASIC, a revisaram e incorporaram a ela recursos estruturados, abandonando, de vez, as instruções GOTO. Dez anos mais tarde, os dois autores submeteram a linguagem a mais uma reforma geral. Kemeny e Kurtz não estavam acompanhando os avanços de sua criação no mercado até o começo dos anos 80, quando Kurtz, por acaso, usou um computador pessoal pela primeira vez. Tendo o BASIC se tornado uma linguagem padrão entre os microcomputadores, cuja limitação de memória exigiu várias modificações no BASIC original, um grande número de dialetos de má qualidade havia surgido. "Fiquei assustado com o que vi", declarou Kurtz. Em conjunto com o ANSI, Instituto Nacional Americano de Padrões (American National Standards Institute), Kurtz e Kemeny começaram a desenvolver uma versão definitiva para microcomputadores, que batizaram de True BASIC (BASIC verdadeiro).

TESTES DE TENTATIVA E ERRO

Edsger Dijkstra, tendo ajudado a mudar o modo de escrever programas e até linguagens, não abriu mão do seu papel de crítico persistente da computação. Os testes e *debug*, a fase final do desenvolvimento do software, incomodaram-no tanto quanto as instruções GOTO. Desde os tempos do processamento em lotes, essa fase tradicionalmente tem sido baseada em tentativa e erro. O programa completo é carregado no computador e testado. Se for encontrado um erro, este será corrigido e o programa será executado novamente. Se um outro erro aparecer, o procedimento se repete. Executada dessa maneira, a fase de testes e correções freqüentemente demora mais que todas as outras fases somadas. Além disso, os testes só podem mostrar a presença de erros, nunca sua ausência. Mesmo depois que um programa parece totalmente corrigido, novos inputs podem causar uma pane.

Ao invés de testar programas, Dijkstra quer verificá-los pela aplicação de princípios matemáticos. Ele foi bem-sucedido com programas muito pequenos, mas não com os grandes. À medida que os programas crescem, as provas matemáticas exigidas tornam-se incontroláveis, tão longas e propensas a conterem erros quanto os próprios programas. Dijkstra acredita que, para provar grandes programas, os matemáticos e cientistas da computação terão que "aumentar o poder de raciocínio exponencialmente". Ele admite que esse objetivo é ambicioso, "mas não necessariamente irreal".

A longo prazo, muitos cientistas da computação desejam mudar também as outras fases do desenvolvimento de software. Um objetivo importante na cruzada contra a complexidade é a padronização de partes individuais de programas, de modo similar à criação de bibliotecas de sub-rotinas, de maneira que elas possam ser trocadas ou substituídas tão facilmente como as peças do equipamento. O criador de software não teria que começar do zero a cada projeto. Como um engenheiro juntando CIs de memória e outros componentes já existentes, o programador poderia construir um novo sistema com módulos de software criados anteriormente.

A exigente arte do programador

2



A tarefa de escrever um programa chega a ser humilhante em seu rigor. Os computadores exigem precisão e abrangência totais em suas instruções: eles farão apenas o que lhes for ordenado e as instruções não podem sofrer a menor ambigüidade. Isso se aplica a todo tipo de software, tanto um programa simples que faça um pequeno computador tocar alguns acordes, como um prodigiosamente complexo, que controle o tráfego aéreo de um aeroporto.

A programação seria duplamente assustadora se tivesse que ser feita com os uns e zeros do código binário. No nível mais básico, essa é a única comunicação que um computador pode entender, mas os programadores criaram linguagens especializadas que permitem a construção de conjuntos de comandos sem ter de recorrer aos dígitos binários. Existem muitas dessas linguagens chamadas de alto nível, com sintaxe e gramática próprias e destinadas a desempenhar tarefas específicas. Nenhuma das linguagens pode se proclamar de utilidade universal. Por exemplo, uma linguagem projetada para aplicações científicas não é adequada à programação de um banco de dados para folha de pagamento. Isso poderia ser feito, mas acarretaria a criação de comandos especializados que já fazem parte das linguagens orientadas para a administração.

A escolha da linguagem apropriada é, portanto, o primeiro passo na criação do software. Em seguida o programador deve definir o trabalho que o computador irá desempenhar, juntamente com o método a ser utilizado, na forma de algoritmo — um plano de ação passo a passo. A lógica do algoritmo deve ser perfeita, caso contrário o programa produzirá resultados incoerentes, ou simplesmente não se efetuará.

O programador deve então considerar o tipo de dados que o computador manipulará, decidindo quais as melhores maneiras de arquivá-los e recuperá-los para o processamento. Se o programa tiver de reagir a atividades onde o fator tempo é crítico — como, por exemplo, o controle de uma reação em indústria química —, os melhores métodos serão os mais rápidos.

Os programadores elaboraram uma série de estratégias e técnicas úteis na criação de software para uma grande variedade de problemas e aspirações humanas. A escolha adequada da linguagem, da lógica e das técnicas de programação possibilita o perfeito domínio dos computadores, com o máximo de eficiência.

Os perigos da imprecisão

Embora fosse conveniente escrever as instruções em inglês, francês, português ou qualquer outra língua natural, os computadores são rígidos demais para compreender as sutilezas da comunicação humana. Ao conversar, as pessoas complementam suas palavras com gestos e expressões faciais. Fazem uso de exagero, omissões, ironias, subentendidos e outras figuras de retórica. Empregam palavras de múltiplos significados, seguras de que o contexto, a inflexão, a expressão corporal e outros fatores esclarecerão o que pretendem dizer. Mesmo na linguagem escrita, usam uma série de recursos para decifrar as mensagens das afirmações alheias.

Suponha que alguém diga: "Este carro é quente". Se foi um jovem aficionado por automóveis, pode-se supor que fale do carro da moda. Se a mesma frase foi dita por um oficial de polícia, pode significar que o carro não é roubado, que está com os documentos em ordem.

A inteligência humana pode resolver o quebra-cabeça da linguagem natural, mas um computador requer um sistema de comunicações rígido, matematicamente preciso: cada símbolo, ou grupo de símbolos, sempre deve significar exatamente a mesma coisa, e toda afirmação deve ser usada literalmente. Nunca pode haver ironia, coloquialismo ou variação de sentido.



As linguagens naturais humanas estão repletas de palavras cujo significado varia conforme o contexto. Ao contrário dos computadores, uma pessoa logo perceberia que a instrução "corte esse parágrafo" refere-se a excluí-lo do texto em sua versão final, não a cortá-lo fisicamente com uma faca.



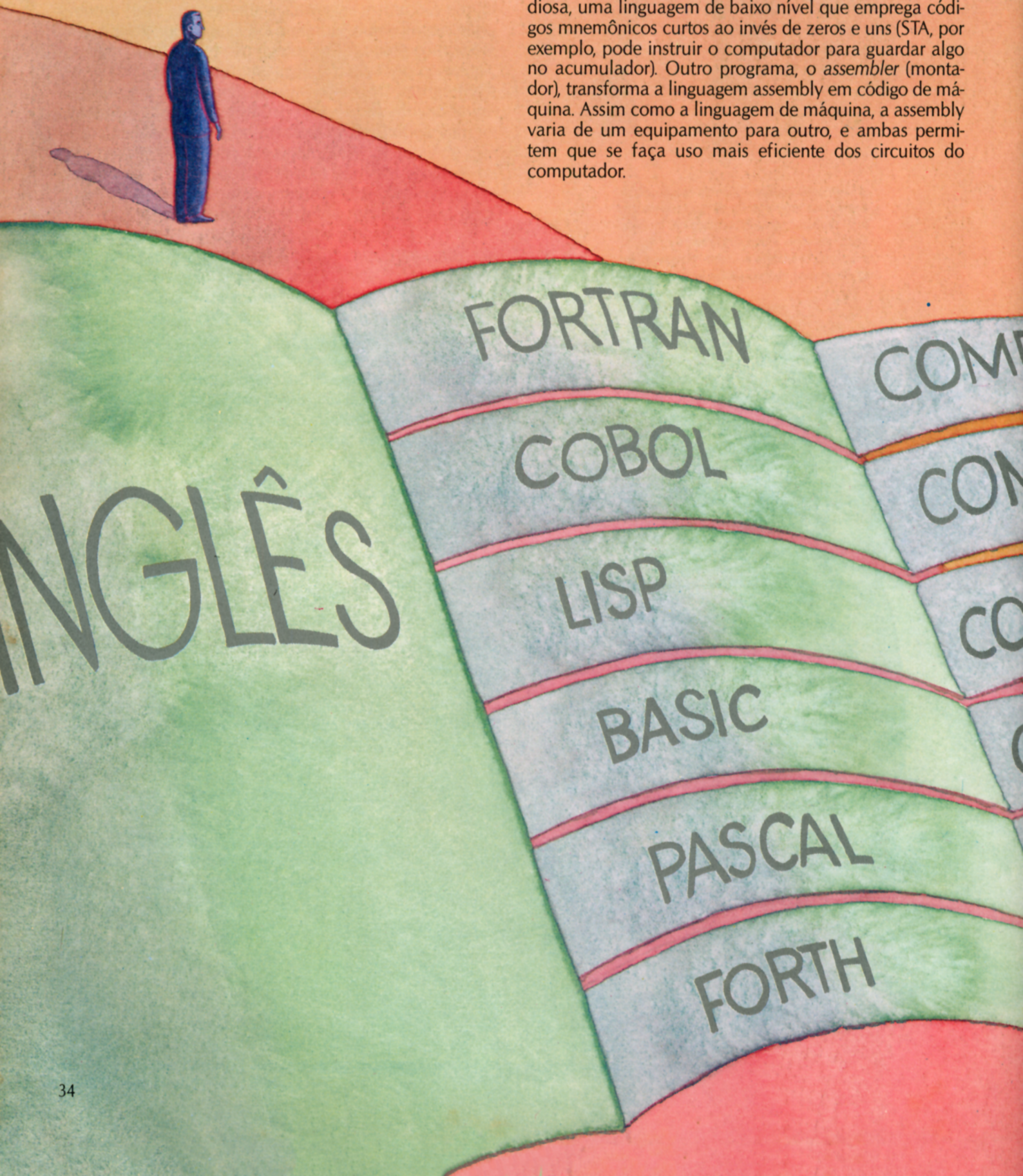
A comunicação em linguagem natural freqüentemente depende de uma espécie de taquigrafia aprendida culturalmente. Um computador não saberia, por exemplo, que "Vou tomar café com bolo" significa "Vou tomar café e comer bolo", e não "Vou tomar café misturado com bolo".



A ambigüidade, muito comum na linguagem natural, é fatal para o computador. Por exemplo, a frase "Eu vejo uma mulher na praia com meus binóculos" pode significar: "Eu vejo uma mulher na praia, e vejo meus binóculos na praia". Outro significado seria: "Eu vejo uma mulher na praia usando meus binóculos". E poderia significar: "Olhando através de meus binóculos, vejo uma mulher na praia". Ao ouvir a frase, uma pessoa escolheria entre as possibilidades e encontraria um sentido para ela. Um computador iria simplesmente ficar travado pela falta de compreensão.

O trabalho pesado dos programas tradutores

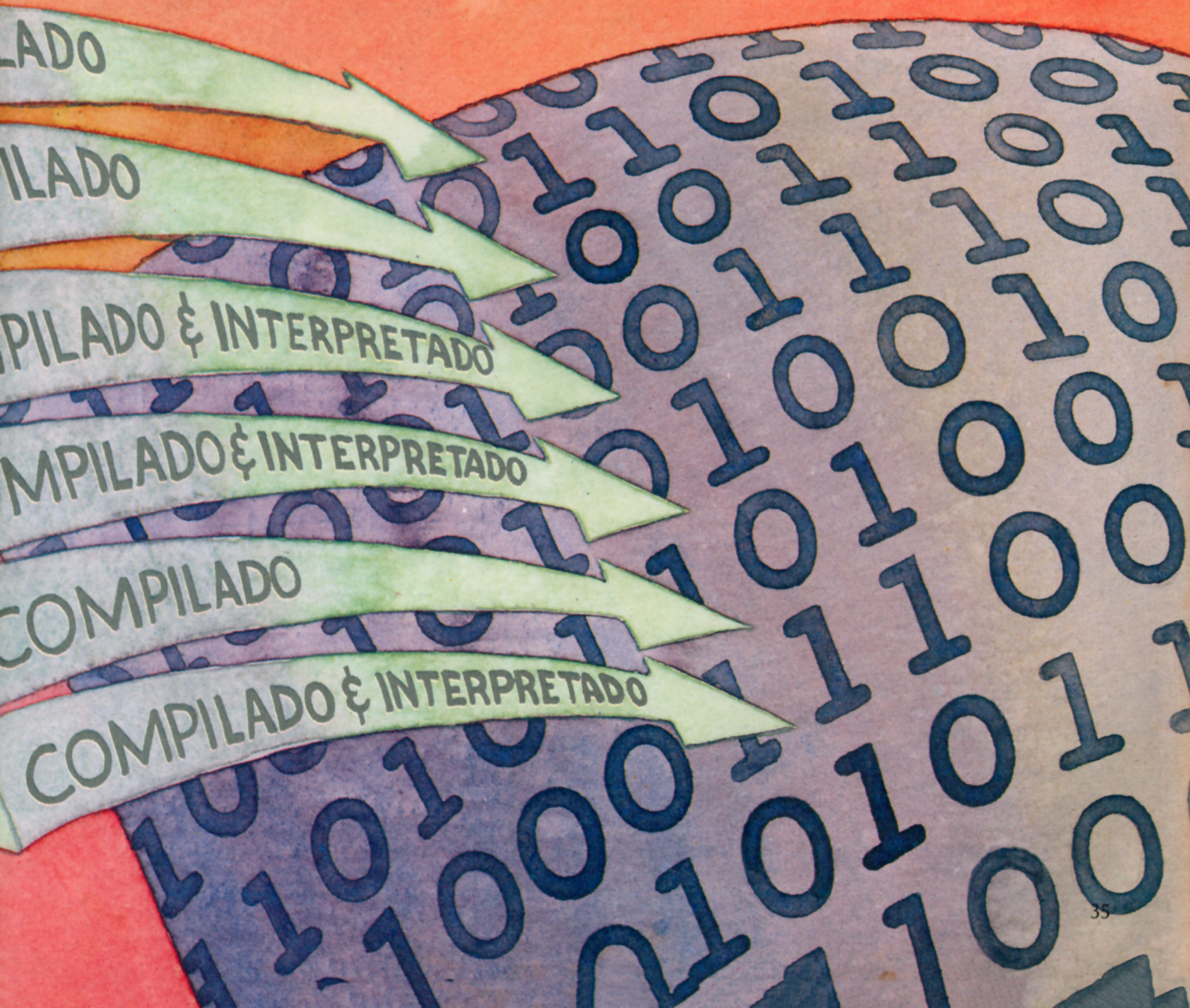
Em última instância, um computador só entende instruções e dados transmitidos em forma de código de máquina, uma linguagem de uns e zeros que age diretamente sobre os circuitos daquele computador específico. Contudo, escrever um programa em linguagem de máquina é uma tarefa extremamente enfadonha. A linguagem assembly (linguagem de montagem) é um pouco menos tediosa, uma linguagem de baixo nível que emprega códigos mnemônicos curtos ao invés de zeros e uns (STA, por exemplo, pode instruir o computador para guardar algo no acumulador). Outro programa, o *assembler* (montador), transforma a linguagem assembly em código de máquina. Assim como a linguagem de máquina, a assembly varia de um equipamento para outro, e ambas permitem que se faça uso mais eficiente dos circuitos do computador.



A maioria dos programadores prefere usar linguagens de nível mais alto, como COBOL ou BASIC, mais próximas da linguagem natural humana. Contudo, um computador não pode obedecer a nenhuma instrução até que esta seja traduzida para zeros e uns. A tradução pode ser feita por compiladores ou por interpretadores, programas especiais desenvolvidos sob medida para a linguagem e equipamento em questão.

Os dois tradutores apresentam modos distintos de operação. Um compilador lê o programa do começo ao fim, faz a tradução e produz uma versão completa em linguagem de máquina a ser posteriormente introduzida no computador e executada. Um interpretador traduz e executa o programa, linha a linha. Assim que é compilado um programa, o original e o compilador tornam-se desnecessários. Já o programa escrito em linguagem interpretada deve ser interpretado todas as vezes que for executado. Dessa maneira, os programas compilados são muito mais rápidos, mas os interpretados se apresentam mais fáceis de corrigir ou modificar.

Uma linguagem pode ser compilada ou interpretada, dependendo do uso para o qual foi criada. A linguagem FORTRAN, por exemplo, destina-se a programas extensos para aplicações matemáticas e científicas, onde a velocidade de execução é importantíssima; portanto, normalmente é implementada com compiladores. O BASIC, por outro lado, foi projetado para ser usado por programadores novatos. Em alguns casos, a mesma linguagem dispõe de compiladores e interpretadores — um programa pode ser desenvolvido usando-se um interpretador para a fase de testes, e depois ser compilado, para se obter maior velocidade.

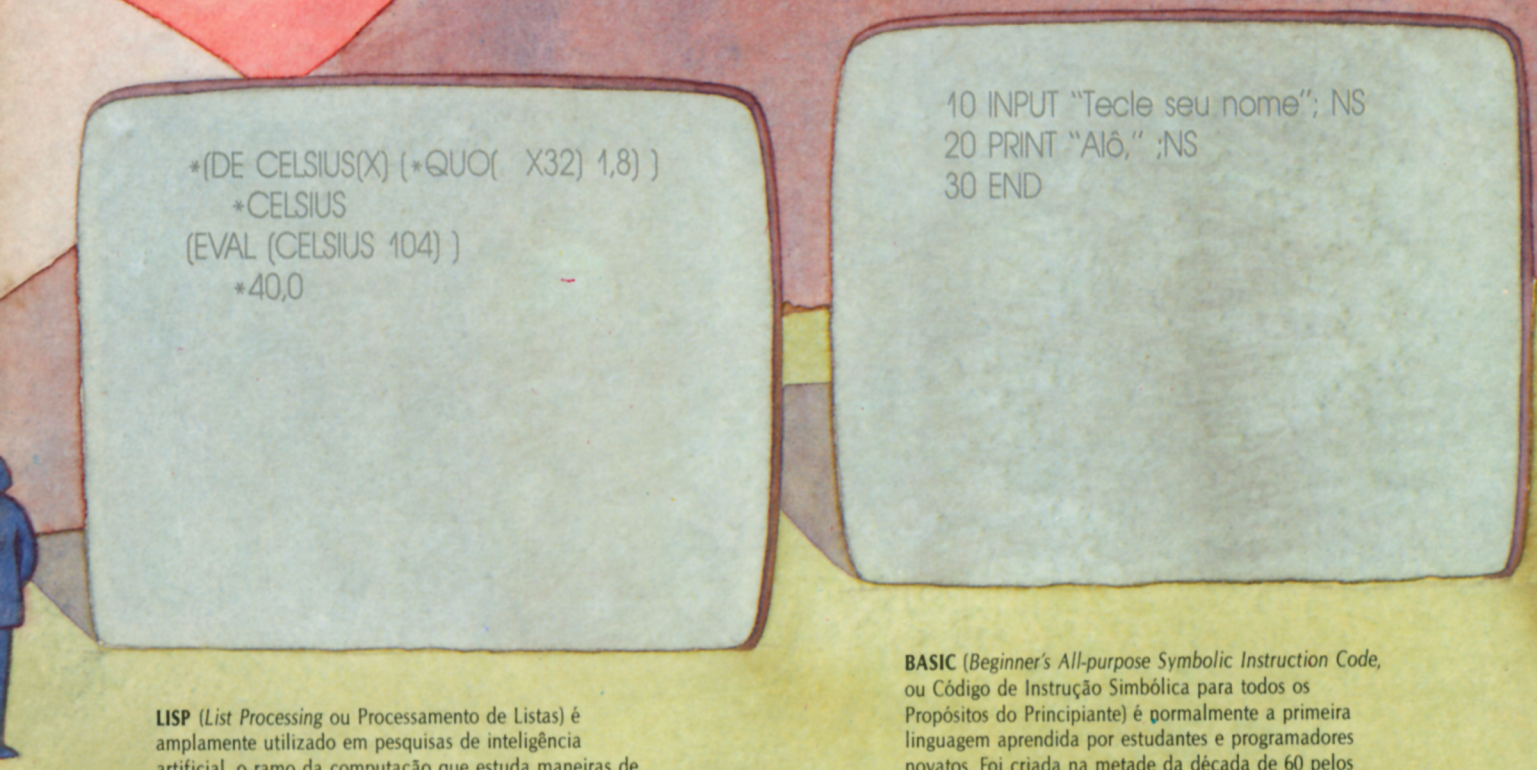


Linguagem e trabalho

Se for feita uma pesquisa junto a uma dúzia de programadores sobre qual a melhor linguagem de programação, é provável que se obtenha uma dúzia de respostas diferentes, pois não existe uma linguagem "melhor" de programação, assim como nenhuma língua humana é superior a qualquer uma das outras. Teoricamente, a maior parte das tarefas de programação poderia ser efetuada em qualquer linguagem, mas é realmente bem mais fácil criar um programa para uma determinada finalidade usando-se uma determinada linguagem, pois as necessidades específicas de programação definem maneiras próprias de se expressar.

Cada linguagem de programação tem seu vocabulário próprio de palavras-chaves, que podem ser palavras, letras, números ou outros símbolos. As palavras-chaves correspondem a operações específicas ou séries de operações a serem executadas pelo computador. Algumas funcionam como verbos, outras como substantivos, advérbios ou sinais de pontuação. São agrupadas de acordo com as regras de sintaxe da linguagem para formar o equivalente às sentenças. Na escolha da linguagem mais apropriada para um trabalho de programação, vários fatores devem ser levados em consideração, entre eles a experiência do programador e o tamanho do programa a ser feito. BASIC é relativamente fácil de se aprender e bem adaptado à criação de programas simples e curtos — programas longos em BASIC podem ser difíceis de controlar e organizar. Já a linguagem Pascal é coerente com programas longos, impondo uma estrutura lógica rígida, muito valorizada mesmo por principiantes, devido à disciplina que se adquire com seu uso. Contudo, para programas curtos, essa linguagem pode ser excessivamente trabalhosa.

Outras considerações importantes incluem a rapidez de execução e a facilidade de manutenção. Isto é, a facilidade com que o programa pode ser alterado após entrar em operação. Programas escritos em FORTH, uma linguagem extremamente compacta, tomam muito pouco espaço na memória; várias palavras-chaves de FORTH são apenas sinais de pontuação. Isso torna os programas em FORTH bem mais rápidos que os programas em BASIC, mas também bem mais difíceis de serem lidos ou modificados. No outro extremo, a manutenção de programas em COBOL é especialmente fácil. São divididos em partes distintas e seguem regras de sintaxe similares à da língua inglesa.



LISP (*List Processing* ou Processamento de Listas) é amplamente utilizado em pesquisas de inteligência artificial, o ramo da computação que estuda maneiras de fazer um computador imitar os processos humanos de raciocínio. A linguagem LISP, criada no fim dos anos 50 pelo matemático do MIT, John McCarthy, é mais apropriada para a manipulação de símbolos do que para cálculos com números comuns.

Este programa converte graus Fahrenheit em Celsius.

BASIC (*Beginner's All-purpose Symbolic Instruction Code*, ou Código de Instrução Simbólica para todos os Propósitos do Principiante) é normalmente a primeira linguagem aprendida por estudantes e programadores novatos. Foi criada na metade da década de 60 pelos professores John Kemeny e Thomas Kurtz, da Escola Dartmouth, e ganhou popularidade no começo da década seguinte, por ser compacta o suficiente para caber na memória dos primeiros computadores pessoais.

Este é um jogo de nomes muito curtos.

```
      SOMA = 0.0
C Calcular a média de um conjunto de
      valores
      DO 100 I = 1,ÚLTIMO
100   SOMA = SOMA A(I)
      MÉDIA = SOMA / ÚLTIMO
```

FORTRAN (*Formula Translator* ou Tradutor de Fórmulas) foi criado na metade da década de 50 por programadores da IBM. Usado principalmente para programas que envolvem cálculos científicos e matemáticos, o FORTRAN teve influência duradoura. Como declarou um cientista: "De certa maneira, toda pesquisa de linguagens de programação desde 1957 foi motivada por tentativas de corrigir falhas do FORTRAN".

Este programa calcula uma média aritmética.

```
IMPRIME-QUANTIA-NO-CHEQUE.
  MOVE QUANTIA-A-PAGAR-FUNCIONÁRIO
    TO QUANTIA-CRUZADOS-E-CENTAVOS-CHEQUE
  MOVE QUANTIA-A-PAGAR-FUNCIONÁRIO
    TO QUANTIA-TRABALHO-TOTAL
  MOVE QUANTIA-TRABALHO-TOTAL
    TO QUANTIA-CRUZADOS-CHEQUE
```

COBOL (*Common Business Oriented Language* ou Linguagem Orientada para Negócios Comuns) nasceu no Pentágono, em 1960, fruto de um esforço conjunto do governo federal americano e da indústria de computadores. Projetada para ser facilmente compreendida por não-cientistas do mundo dos negócios, a linguagem usa estrutura e vocabulário similar aos da língua inglesa. É a linguagem de processamento de dados mais popular em bancos e companhias de seguro.

Acima, um trecho de um programa emissor de cheques.

```
VAR
    mensagem : STRING;
BEGIN
    mensagem: = 'Penso,
    logo existo.';
    write 1 n(mensagem);
END
```

PASCAL, cujo nome homenageia o matemático francês do século XVII, Blaise Pascal, é uma linguagem concebida pelo cientista da computação Niklaus Wirth, de origem suíça. Considerado como instrumento fundamental para o aprendizado da disciplina da programação estruturada, Pascal é desde 1983 a linguagem exigida dos estudantes secundaristas americanos que desejem entrar para os cursos avançados de ciência da computação.

Este programa faz o computador imprimir na tela a mensagem "Penso, logo existo".

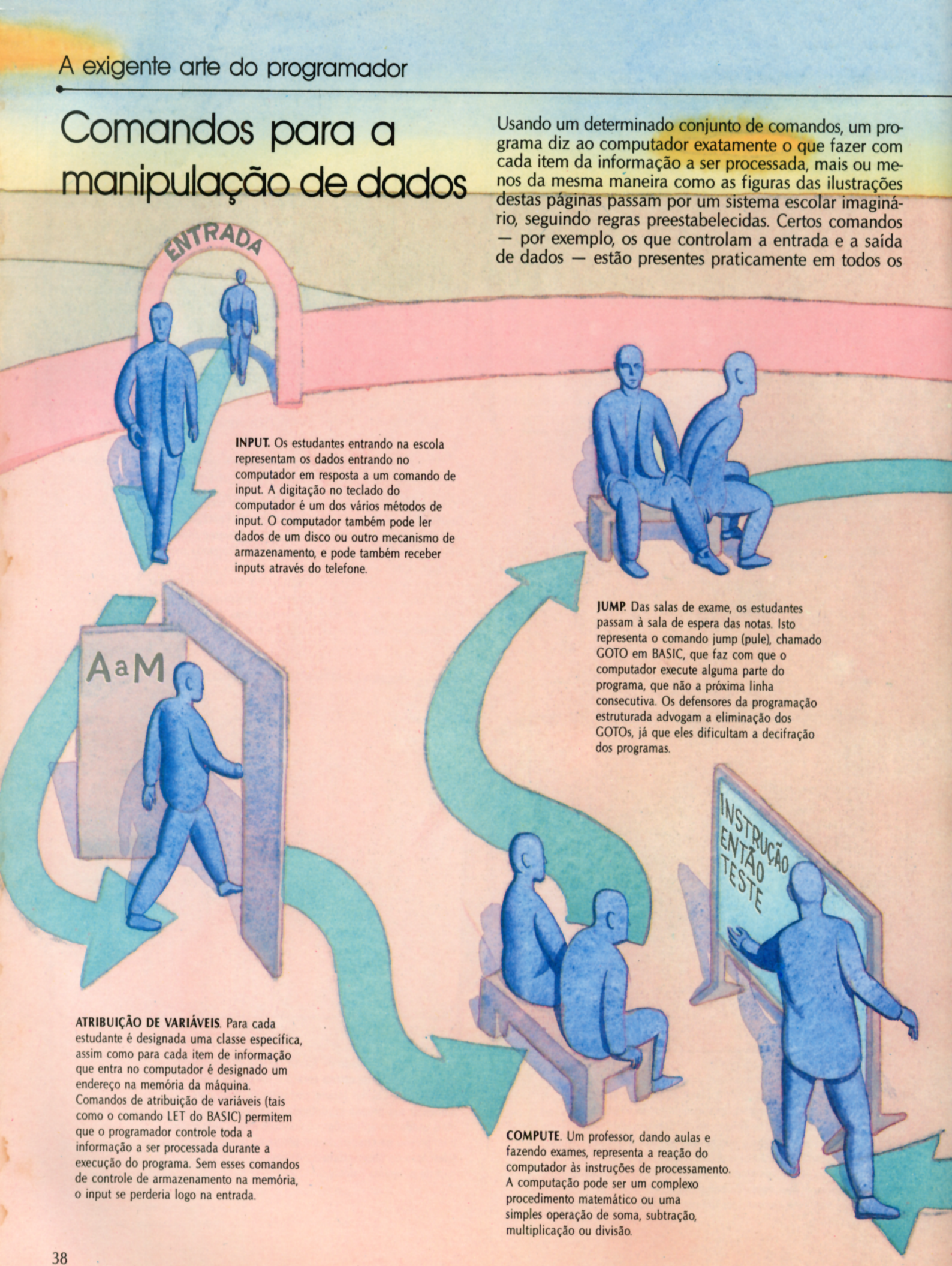
```
7 2 * 5 + 19ok
```

FORTH (nome derivado de *fourth*, quarta, de linguagem de programação de quarta geração) foi criado pelo programador Charles Moore, por volta de 1970, para aumentar sua própria produtividade. FORTH passou a ser mais usada em aplicações de controle depois que Moore a empregou na criação de um programa que controlava o telescópio do observatório do Arizona.

Este programa calcula 5+(2 x 7).

Comandos para a manipulação de dados

Usando um determinado conjunto de comandos, um programa diz ao computador exatamente o que fazer com cada item da informação a ser processada, mais ou menos da mesma maneira como as figuras das ilustrações destas páginas passam por um sistema escolar imaginário, seguindo regras preestabelecidas. Certos comandos — por exemplo, os que controlam a entrada e a saída de dados — estão presentes praticamente em todos os



INPUT. Os estudantes entrando na escola representam os dados entrando no computador em resposta a um comando de input. A digitação no teclado do computador é um dos vários métodos de input. O computador também pode ler dados de um disco ou outro mecanismo de armazenamento, e pode também receber inputs através do telefone.

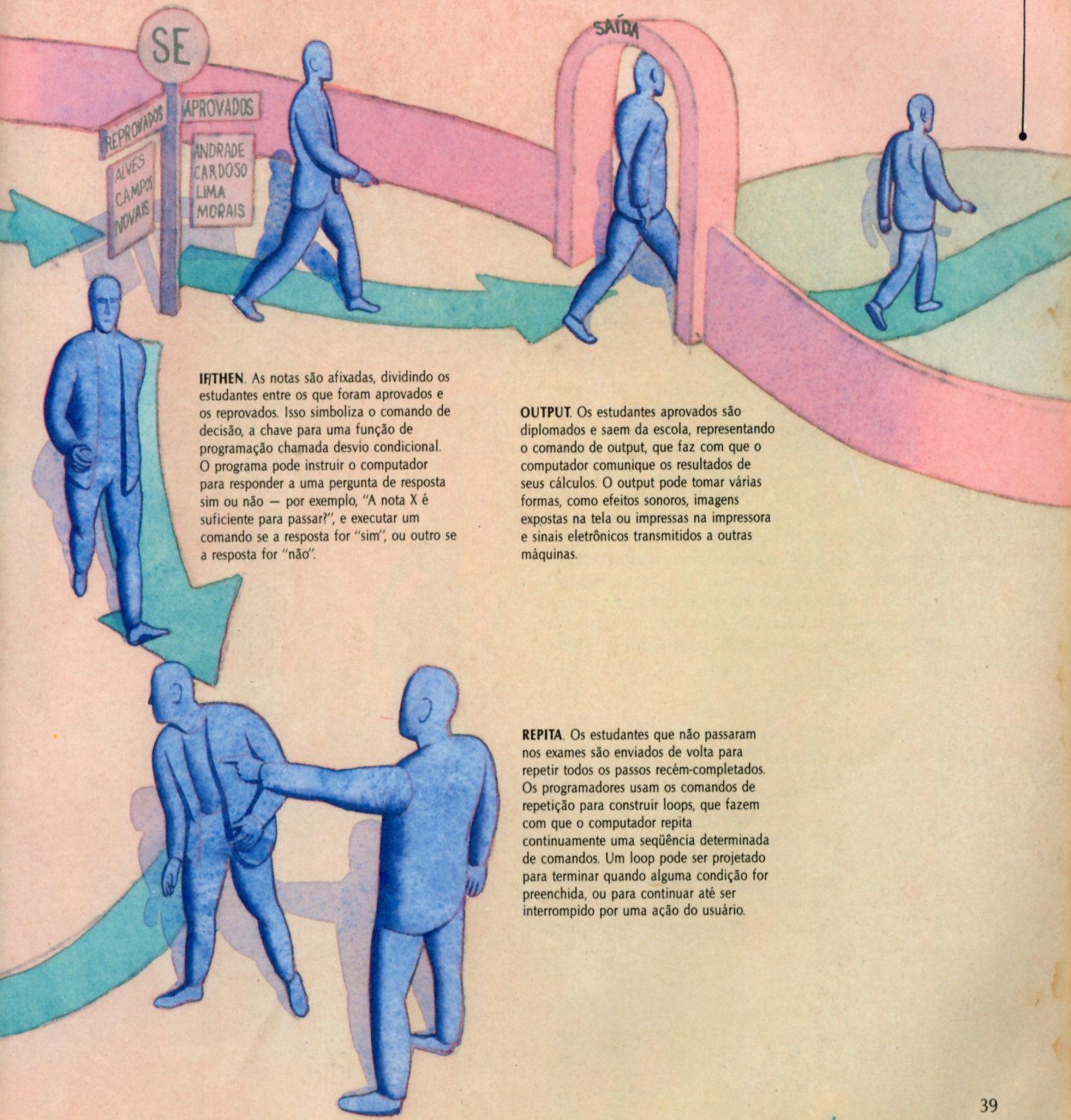
JUMP. Das salas de exame, os estudantes passam à sala de espera das notas. Isto representa o comando jump (pule), chamado GOTO em BASIC, que faz com que o computador execute alguma parte do programa, que não a próxima linha consecutiva. Os defensores da programação estruturada advogam a eliminação dos GOTOs, já que eles dificultam a decifração dos programas.

ATRIBUIÇÃO DE VARIÁVEIS. Para cada estudante é designada uma classe específica, assim como para cada item de informação que entra no computador é designado um endereço na memória da máquina. Comandos de atribuição de variáveis (tais como o comando LET do BASIC) permitem que o programador controle toda a informação a ser processada durante a execução do programa. Sem esses comandos de controle de armazenamento na memória, o input se perderia logo na entrada.

COMPUTE. Um professor, dando aulas e fazendo exames, representa a reação do computador às instruções de processamento. A computação pode ser um complexo procedimento matemático ou uma simples operação de soma, subtração, multiplicação ou divisão.

programas; outros podem não aparecer em programas bem simples. Mas as sete ações ilustradas abaixo são tão fundamentais que dificilmente deixam de aparecer em programas de poder ou complexidade significativos.

Um programador não tem necessariamente de usar esses comandos na seqüência aqui exposta. Input (entrada) e output (saída), por exemplo, podem ocorrer não só no começo e no fim de um programa, como também em qualquer outro ponto dele. Os demais comandos podem estar dispostos em qualquer seqüência, repetindo-se quantas vezes forem necessárias, dentro de um determinado programa. Como são indispensáveis para os programadores, esses comandos estão presentes em todas as linguagens de computação, controlando sempre o mesmo tipo de ação, embora escritos de maneira diferente em cada linguagem.



IF/THEN. As notas são afixadas, dividindo os estudantes entre os que foram aprovados e os reprovados. Isso simboliza o comando de decisão, a chave para uma função de programação chamada desvio condicional. O programa pode instruir o computador para responder a uma pergunta de resposta sim ou não — por exemplo, "A nota X é suficiente para passar?", e executar um comando se a resposta for "sim", ou outro se a resposta for "não".

OUTPUT. Os estudantes aprovados são diplomados e saem da escola, representando o comando de output, que faz com que o computador comunique os resultados de seus cálculos. O output pode tomar várias formas, como efeitos sonoros, imagens expostas na tela ou impressas na impressora e sinais eletrônicos transmitidos a outras máquinas.

REPITA. Os estudantes que não passaram nos exames são enviados de volta para repetir todos os passos recém-completados. Os programadores usam os comandos de repetição para construir loops, que fazem com que o computador repita continuamente uma seqüência determinada de comandos. Um loop pode ser projetado para terminar quando alguma condição for preenchida, ou para continuar até ser interrompido por uma ação do usuário.

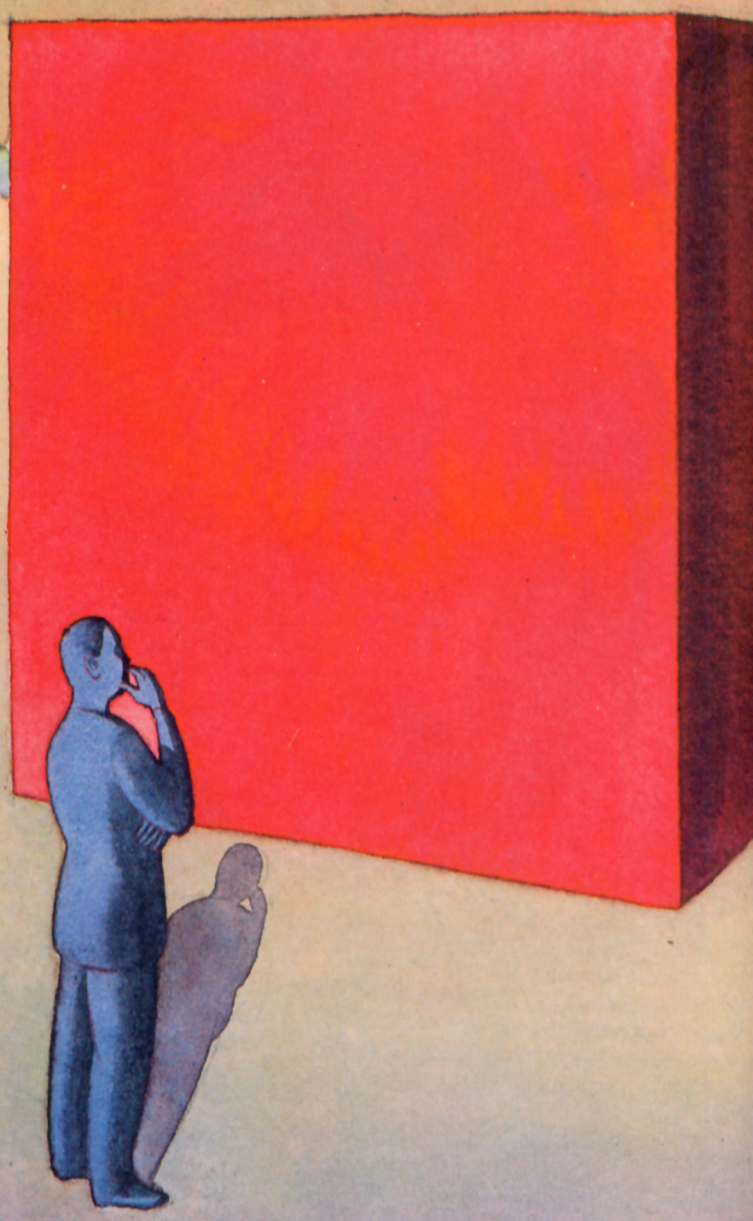
O plano de ação

O objetivo do programador não é resolver um problema, mas criar instruções que façam o computador descobrir a solução. Devido a sua velocidade na manipulação de números, um computador é adequado à resolução de problemas que envolvam cálculos difíceis ou repetitivos. Conforme ilustrado no programa simulado nestas páginas e nas seguintes, a máquina é também capaz de analisar problemas que possam ser expressos por meio de declarações do tipo "if/then" (se/então) da lógica formal ou de seqüências de opções do tipo falso/verdadeiro.

Se a efetuação de cálculos rápidos é um trabalho sob medida para os computadores, a tarefa mais racional de elaborar o algoritmo, isto é, a receita passo a passo que diz ao computador exatamente como desempenhar um determinado trabalho, é atributo da inteligência humana. Para fazer o algoritmo tão claro e conciso quanto possível, um programador sempre começa organizando os fatos conhecidos e definindo precisamente o problema. Só então elabora a seqüência lógica de passos que o computador irá seguir no caminho da solução.

Depois de formular o algoritmo necessário, o desafio seguinte para o programador é transformar cada passo em instruções precisas, de acordo com as regras da linguagem escolhida. Freqüentemente, o programador irá usar casos extremos do problema só para ver se a lógica do algoritmo se sustenta. Esse teste muitas vezes descobre passos omitidos ou instruções ambíguas que provocariam mensagens de erro no computador.

Nenhum detalhe, por mais transparente que seja para a mente humana, pode ser omitido ou tomado como óbvio em programas de computador. Um bom programador precisa ter a visão ampla, para conceber algoritmos eficientes, e atenção minuciosa, para converter esses algoritmos em código de computador, sem ambigüidades.



Esta demonstração do processo de criação de um programa de computador começa com a formulação de um problema imaginário. A caixa à direita representa a caixa externa de um conjunto de caixas vermelhas, que se encaixam perfeitamente uma dentro da outra, sem nenhum espaço entre suas paredes. No centro dessas caixas há uma esmeralda de valor incalculável, que foi escondida por um imperador. A fabulosa jóia, com exatamente 1 polegada de diâmetro, está encerrada na menor caixa capaz de contê-la. Cada uma das caixas, da maior, que tem 10 pés de cada lado — lembre-se de que 1 pé equivale a 12 polegadas —, até a menor, que contém a esmeralda, possui paredes de 1/4 de polegada de espessura. O imperador prometeu dar a esmeralda como prêmio a quem calcular quantas caixas devem ser abertas até que se encontre a esmeralda.

NOTAS:

- Redefinir o problema
  Quantas caixas devem ser abertas para se alcançar a jóia?

- Fatores conhecidos;
  O tamanho da caixa externa -- 10 pés de lado.
  A espessura da parede -- 1/4 de polegada
  O tamanho da jóia -- 1 polegada de diâmetro (isto determina o tamanho da caixa menor).
  O tamanho da caixa menor -- deve medir no mínimo 1 1/2 polegada de lado (medida externa).

LÓGICA DOS PASSOS (método)

Dadas as dimensões da caixa e da jóia, o procedimento para resolver o problema é ir subtraindo o dobro da espessura das paredes (1/2 polegada) da medida da caixa maior enquanto o resultado for maior ou igual à medida da menor caixa possível. O número de subtrações requeridas é a resposta do problema.

1

Redefinição do problema

O trabalho do programador começa com estas notas preliminares. O primeiro passo da criação de um programa de computador é a redefinição do problema em seus termos essenciais, incluindo apenas os fatos que forem relevantes para sua solução. Neste caso, os detalhes como a cor das caixas e o valor da esmeralda não têm qualquer influência no resultado. Os únicos fatos significativos para o programador são o tamanho da jóia, a espessura das paredes das caixas e as dimensões da caixa maior. Aplicando a lógica e a engenhosidade a esses poucos fatos, o programador começa a delinear um método para encontrar a solução.

INPUTS

MAIORCAIXA (a dimensão externa da maior caixa)
ESPESSURA (a espessura das paredes)
JÓIA (o tamanho da jóia)

OUTPUTS

CONTA (quantas caixas foram abertas)
Mensagens de erro - Mostrar mensagem de erro se CONTA for igual a 0, se ESPESSURA for menor ou igual a (<=)0 e se JÓIA for menor ou igual a (<=)0.

O QUE FAZER

INPUT - MAIORCAIXA, ESPESSURA, JÓIA
Dados os INPUTS, verificar se o problema pode ser resolvido.
Se passar na verificação, inicializar.
Começar o procedimento colocando as seguintes variáveis na memória:

```
CONTA ← 0
ESTACAIXA ← MAIORCAIXA
MENORCAIXA ← JÓIA + (2 x ESPESSURA)
```

Repetir enquanto ESTACAIXA >= MENORCAIXA
ESTACAIXA ← ESTACAIXA - (2 x ESPESSURA)
CONTA ← CONTA + 1

OUTPUT CONTA e/ou mensagem de erro.



Concepção do algoritmo

Para determinar os passos para a solução, o programador primeiro decide que informações o computador necessita, atribuindo nomes a essas variáveis de INPUT, para facilitar a referência a elas. Em seguida, o programador determina que informação o computador vai fornecer. Esses OUTPUTs incluem as mensagens de erro com as quais o computador vai avisar que os INPUTs foram incorretos ou ilegais. Finalmente, o programador cria o algoritmo, usando o símbolo ← para atribuir valores a variáveis tais como ESTACAIXA, que é a dimensão da nova caixa descoberta a cada abertura.

MAIOR CAIXA = 6 POLEGADAS
ESPESSURA DA PAREDE = 0,25 POLEGADA
TAMANHO DA JÓIA = 1 POLEGADA

6 - 0,50 = 5,50	CONTA 1
5,50 - 0,50 = 5	CONTA 2
5 - 0,50 = 4,50	CONTA 3
4,50 - 0,50 = 4	CONTA 4
4 - 0,50 = 3,50	CONTA 5
3,50 - 0,50 = 3	CONTA 6
3 - 0,50 = 2,50	CONTA 7
2,50 - 0,50 = 2	CONTA 8
2 - 0,50 = 1,50	CONTA 9
1,50 é a menor caixa possível	CONTA 10

Isto funciona.

MAIOR CAIXA = 0,1 PÉ (1,2 POLEGADA)
ESPESSURA DA PAREDE = 0,25 POLEGADA
TAMANHO DA JÓIA = 1 POLEGADA

Uma vez que a menor caixa possível mede 1,5 polegada, o valor da maior caixa torna impossível a resolução do problema. Este só teria solução se a caixa maior medisse mais de 1,5 polegada.

3 Execução de um teste

Para descobrir se a lógica do algoritmo é eficiente, o programador faz dois testes simples, atribuindo valores às variáveis de INPUT e executando o programa manualmente. O algoritmo funcionará sempre que a caixa maior for maior que a esmeralda.

```
10 INPUT "Qual o tamanho da caixa maior (em polegadas)"; MAIOR
20 INPUT "Qual a espessura das paredes das caixas (frações decimais de
   polegadas)"; ESPESSURA
30 INPUT "Qual o tamanho da jóia (em polegadas)"; JÓIA
40 IF ESPESSURA <= 0 OR JÓIA <= 0 THEN PRINT "Você deve estar
   brincando...":END
50 IF MAIOR < JÓIA + (2 * ESPESSURA) THEN PRINT "A caixa maior é MUITO
   pequena para conter a jóia":END
60 LET CONTA = 0
70 LET MENOR = JÓIA + (2 * ESPESSURA)
80 LET ESTACAIXA = MAIOR
90 REM esta é a parte que vai se repetir...
100 WHILE ESTACAIXA > = MENOR
110 LET ESTACAIXA = ESTACAIXA – (2 * ESPESSURA)
120 LET CONTA = CONTA + 1
130 PRINT "Estou abrindo mais uma caixa.";CONTA;" até agora..."
140 Wend : REM O loop termina aqui
150 PRINT "Nós abrimos um total de ";CONTA;" caixas."
160 END
    RUN
```



4 Como se escreve um programa

O programa na tela foi escrito em BASIC, com os números de linhas sendo incrementados de 10 em 10, para facilitar a inserção de linhas, caso seja necessário. As linhas de INPUT dizem ao computador que pergunte os valores fixos que definirão o problema; LET faz com que se atribuam valores a variáveis que vão mudar durante a execução do programa. O computador ignora todas as linhas que começam com REM, expressão usada pelos programadores para escrever comentários de uso pessoal. As linhas 100-140 formam o cerne do programa, um loop que o computador repetirá até alcançar a solução.



5

Rodando o programa

Uma vez digitado o programa no computador, basta teclar o comando RUN e a máquina começa a trabalhar, abrindo matematicamente as caixas através da repetição do loop de subtração incluso no programa. O programador pode agora entrar com o problema e encontrar a solução: uma jóia de 1 polegada de diâmetro está dentro da menor caixa possível aninhada dentro de sucessivas caixas sem espaço nenhum entre elas. As caixas têm paredes de 1 polegada de espessura. A maior caixa (a caixa de fora) tem 10 pés (120 polegadas) de lado. Para se encontrar a jóia, seria preciso abrir 238 caixas.



Uma representação visual do problema

Um instrumento muito útil da programação é o fluxograma, uma representação gráfica do sistema invisível concebido pelo programador. Empregando símbolos padronizados para as várias partes do programa (*abaixo*), ele fornece um mapa — de fácil leitura — do processo passo a passo que a máquina deverá executar. Muitas vezes, os programadores esboçam seus algoritmos na forma de fluxogramas, como meio de organizar os pensamentos antes de começar a escrever o código exigido pelas linguagens de programação. Escrito o programa, o fluxograma pode servir também para verificar se não falta nenhum passo essencial.

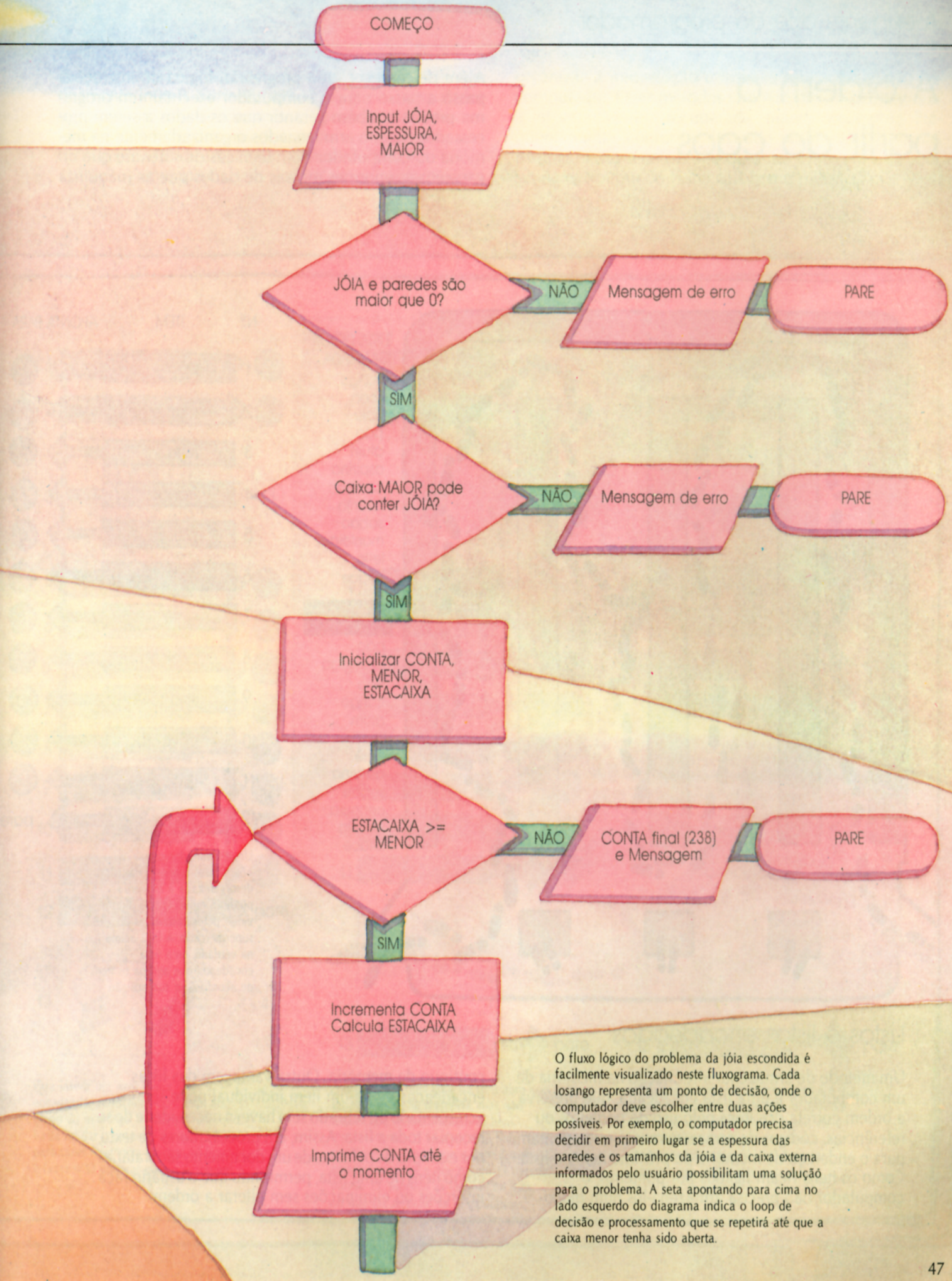
Embora fluxogramas detalhados, com diversos caminhos de decisão, sejam bastante trabalhosos para se desenhar, o formato gráfico desse recurso é muito útil, particularmente quando vários programadores estão colaborando em um projeto complexo. O conjunto de símbolos fornece uma indicação visual de como cada contribuição individual se encaixa na estrutura geral. O governo americano, a IBM e muitas outras organizações exigem que seus programadores documentem todo seu trabalho com fluxogramas.

O fluxograma na página ao lado documenta o programa usado para se resolver o problema da jóia escondida. Ele acompanha o fluxo lógico do programa, desde o momento em que o usuário fornece os números essenciais até a exibição na tela da solução do problema. A cada vez que o computador tem que tomar uma decisão, o fluxograma se divide em duas direções, representando a escolha feita pelo computador diante das opções sim/não ou falso/verdadeiro. Freqüentemente uma escolha vai provocar a repetição dos passos de um loop.

De acordo com as instruções do símbolo de output no fim do fluxograma, o computador vai mostrar os números mudando a cada passagem pelo loop do programa, indicando quantas caixas foram abertas até o momento. Quando finalmente a jóia for encontrada e o programa escapar do loop, o número resultante da última passagem será apresentado como solução.



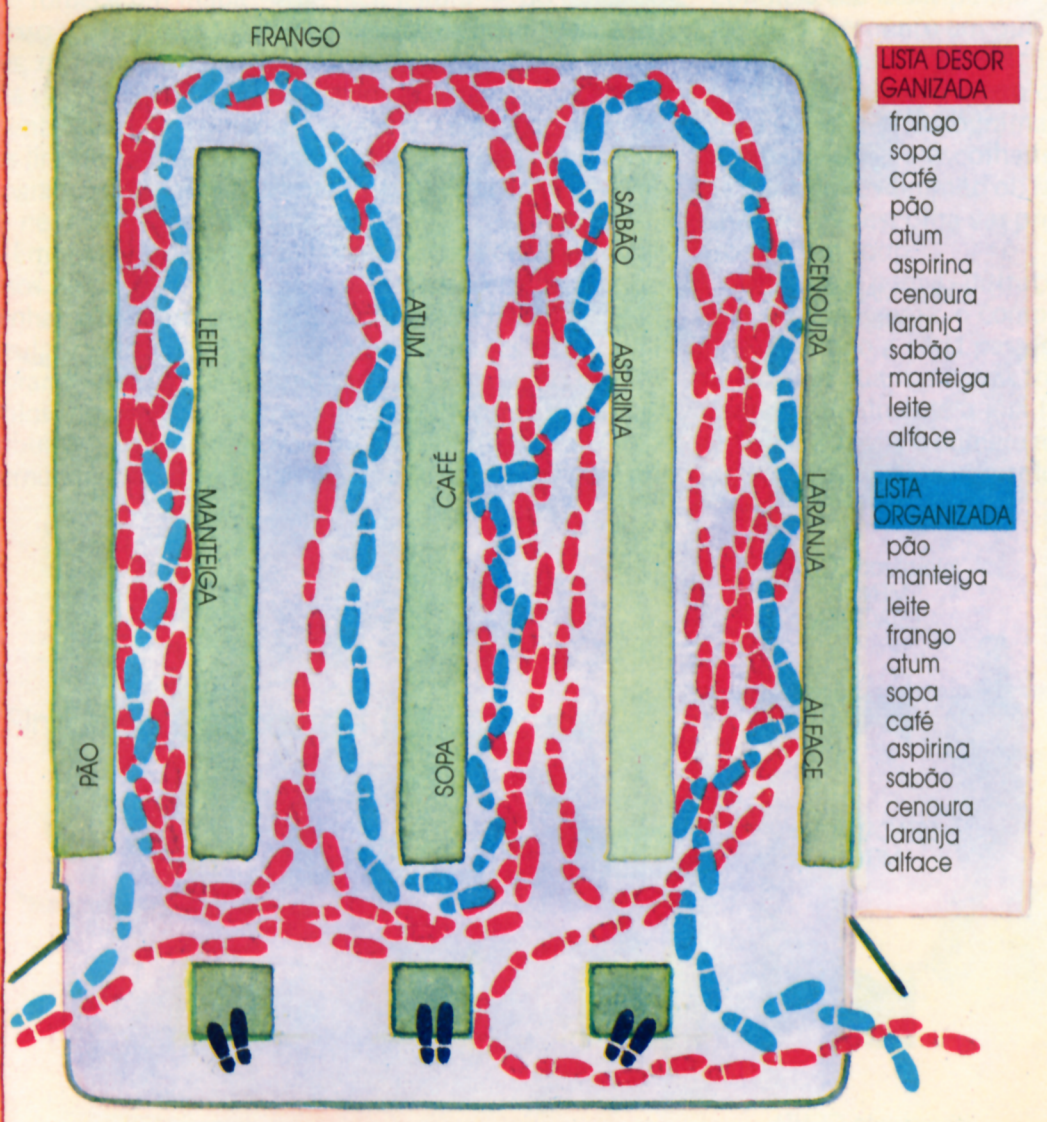
Os símbolos fundamentais de fluxogramas incluem o oval, que designa os pontos iniciais e finais de um programa, o paralelograma, que representa informação fornecida pelo usuário ou gerada pelo computador, e o retângulo, que representa o processamento efetuado pelo computador. O losango simboliza uma decisão, que determina a próxima ação do computador.



O fluxo lógico do problema da jóia escondida é facilmente visualizado neste fluxograma. Cada losango representa um ponto de decisão, onde o computador deve escolher entre duas ações possíveis. Por exemplo, o computador precisa decidir em primeiro lugar se a espessura das paredes e os tamanhos da jóia e da caixa externa informados pelo usuário possibilitam uma solução para o problema. A seta apontando para cima no lado esquerdo do diagrama indica o loop de decisão e processamento que se repetirá até que a caixa menor tenha sido aberta.

A ordem a partir do caos

Além de fornecer uma progressão lógica de instruções passo a passo para o computador executar, um programa precisa também garantir que os dados a serem processados fiquem armazenados organizadamente na memória do computador. Sem técnicas de software que organizem as enormes massas de dados que as máquinas



LISTA DESORGANIZADA
frango
sopa
café
pão
atum
aspirina
cenoura
laranja
sabão
manteiga
leite
alface

LISTA ORGANIZADA
pão
manteiga
leite
frango
atum
sopa
café
aspirina
sabão
cenoura
laranja
alface

Nº	ITEM	APONTA PARA
1	pão	2
2	manteiga	3
3	leite	4
4	frango	5
5	atum	6
6	sopa	7
7	café	8
8	aspirina	9
9	sabão	10
10	cenoura	11
11	laranja	12
12	alface	par

Um consumidor que obedecesse à ordem da lista desorganizada à esquerda perderia muito tempo indo e vindo pelos corredores do supermercado. A lista acima é, na verdade, uma lista encadeada que agrupa os itens de acordo com sua posição.

Listas e listas encadeadas

Um método de armazenar informação na memória de um computador é uma lista encadeada, que preserva a ordem correta dos itens pelo uso de ponteiros ou referências, códigos de armazenamento que apontam para o endereço de memória do próximo item na lista. Como os itens ficam ligados por ponteiros, o computador não precisa armazená-los num bloco contínuo de memória. Outra vantagem das listas encadeadas é que um item individual pode ser acrescentado ou retirado sem haver a necessidade de se recopiar toda a lista. Assim, em processadores de texto, por exemplo, as palavras de um documento são tratadas como itens de uma lista, permitindo que o usuário faça inserções ou eliminações sem alterar a ordem.

devem guardar e processar, os computadores seriam praticamente inúteis. Seria tão difícil usar um computador sem software organizador quanto procurar uma informação em um dicionário que listasse as entradas aleatoriamente, em vez de utilizar a ordem alfabética.

Os programadores usam uma série de técnicas para organizar a informação e armazená-la na memória do computador. Esses métodos também devem permitir que a informação seja recuperada quando necessário. Freqüentemente, conforme ilustrado nesta e nas próximas páginas, essas técnicas são descritas em termos de seus similares no mundo físico, tais como pilhas, filas e listas.



Para inserir um item numa lista encadeada, acrescentando queijo ao grupo da manteiga e leite, por exemplo, basta mudar o ponteiro do item 3 (leite) para fazer referência ao item 13 (queijo) e fazer o 13 apontar para o 4 (frango).



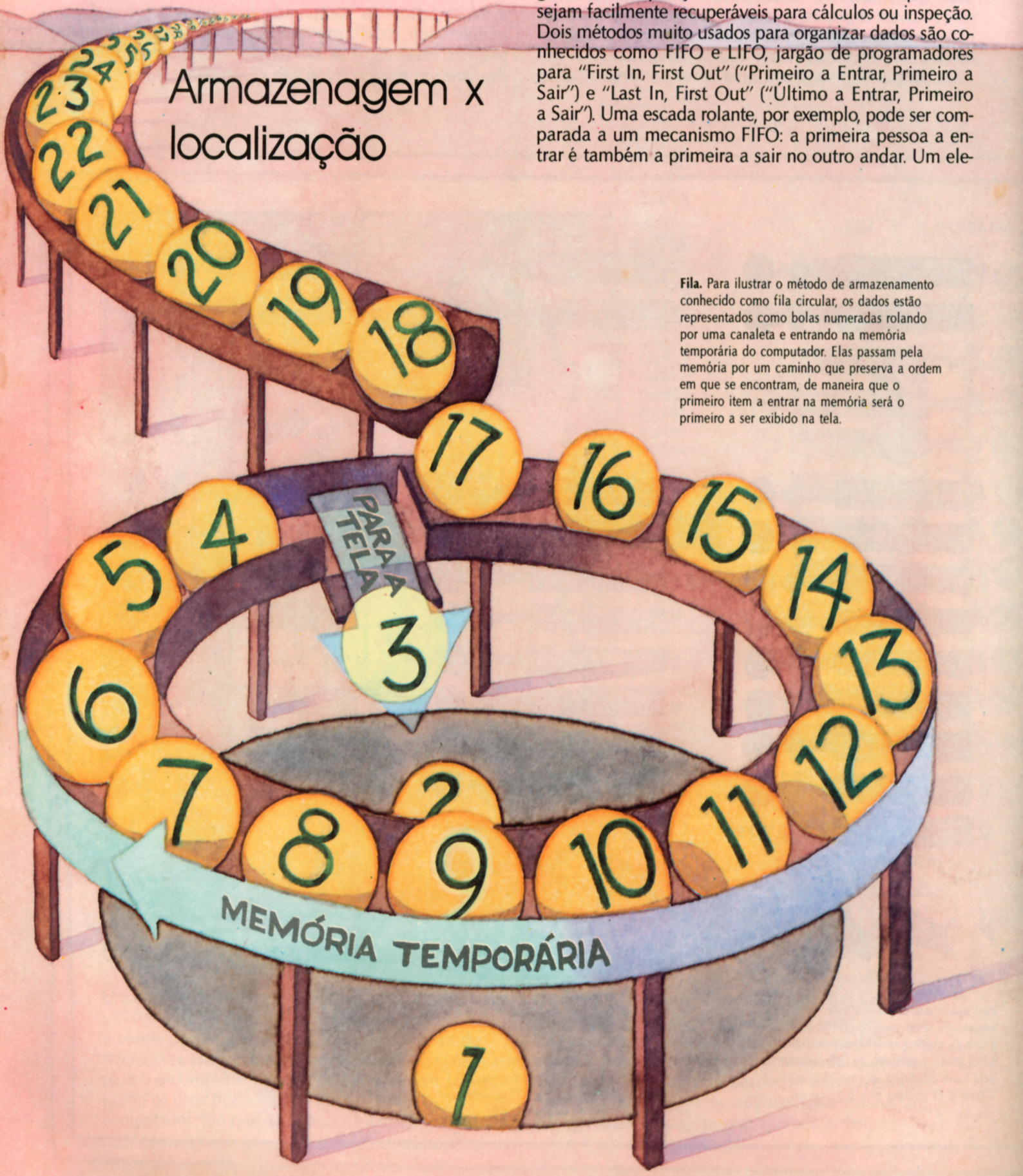
A retirada de dois itens da lista abre dois espaços livres na memória, mas não requer que se modifiquem os itens remanescentes. A mudança de dois ponteiros, de modo que o item 2 passe a apontar para o 4 e que o item 11 aponte para o 13, suprime as lacunas na lista, no que diz respeito ao computador.



Chá, que se encaixa entre os itens 8 e 9, pode ser acrescentado à lista ocupando o espaçò vazio do item 3. Depois ele é colocado em ordem pela mudança dos ponteiros relevantes: o item 8 passa a apontar para o 3 e este para o 9.

Armazenagem x localização

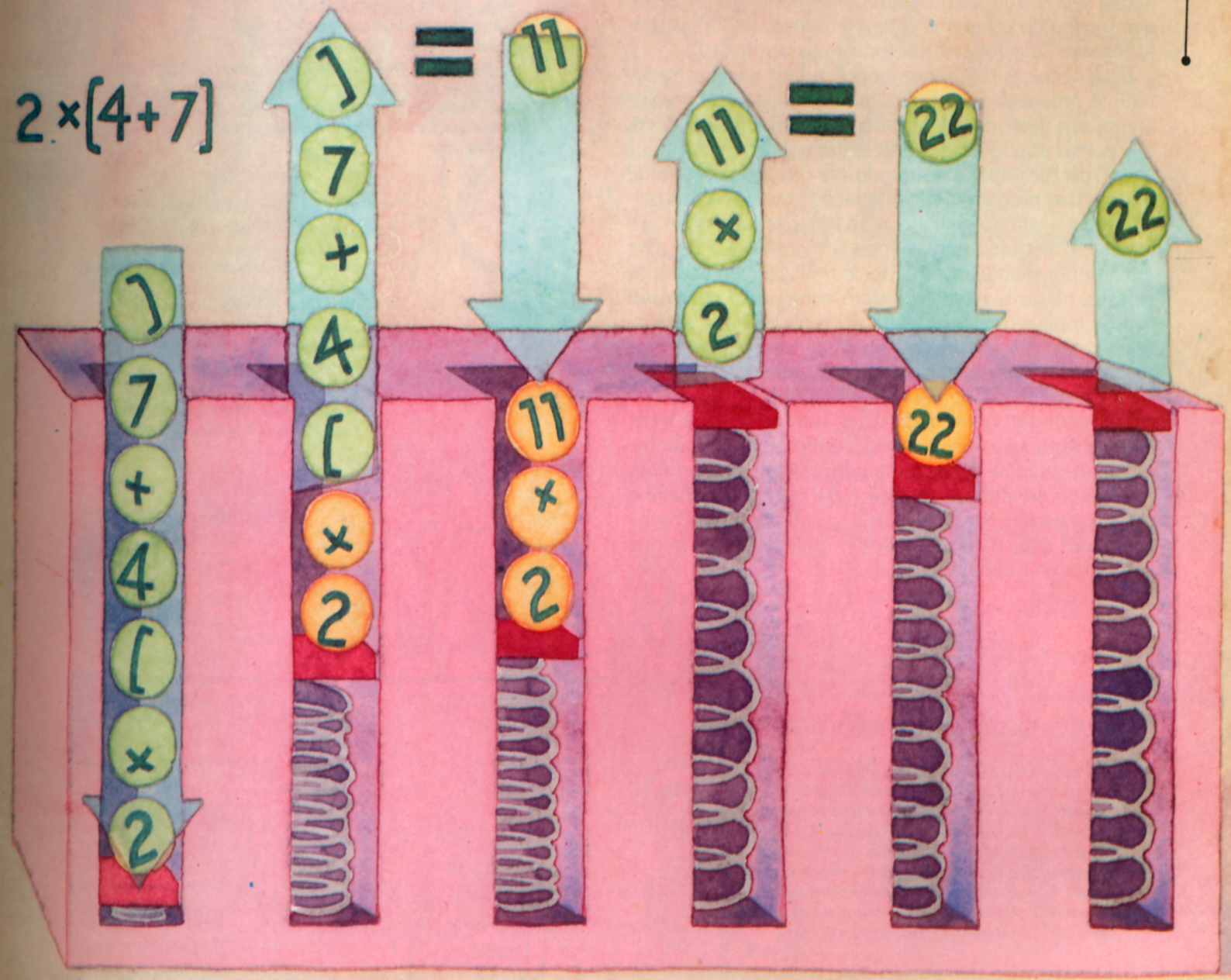
Um fator crítico para a execução eficiente de um programa é a disposição dos dados de maneira que estes sejam facilmente recuperáveis para cálculos ou inspeção. Dois métodos muito usados para organizar dados são conhecidos como FIFO e LIFO, jargão de programadores para "First In, First Out" ("Primeiro a Entrar, Primeiro a Sair") e "Last In, First Out" ("Último a Entrar, Primeiro a Sair"). Uma escada rolante, por exemplo, pode ser comparada a um mecanismo FIFO: a primeira pessoa a entrar é também a primeira a sair no outro andar. Um ele-



Fila. Para ilustrar o método de armazenamento conhecido como fila circular, os dados estão representados como bolas numeradas rolando por uma canaleta e entrando na memória temporária do computador. Elas passam pela memória por um caminho que preserva a ordem em que se encontram, de maneira que o primeiro item a entrar na memória será o primeiro a ser exibido na tela.

vador, pelo contrário, costuma funcionar pelo mecanismo LIFO: os primeiros a entrar são empurrados para o fundo, de modo que a última pessoa a entrar está em posição de sair primeiro quando a porta se abrir. O método de armazenagem FIFO também é conhecido como fila, ao passo que o método LIFO é conhecido como pilha.

Os programadores usam a armazenagem FIFO quando os dados devem ser processados na ordem em que entraram. Um processador de textos, por exemplo, deve mostrar as letras na tela na mesma ordem em que foram tecladas. Em algumas linguagens de programação, como FORTH, as pilhas LIFO de memória são usadas para manipular certas funções aritméticas. No exemplo abaixo, o computador coloca um problema em pilha, pois este não pode ser resolvido pelo processamento dos números na ordem em que entraram: a parte de multiplicação não pode ser resolvida antes da adição (entre parênteses). O problema precisa ser resolvido de trás para a frente.



Pilha. Os números e outros símbolos para esse cálculo são empilhados na ordem de entrada.

Assim, a adição de 4 e 7 é a primeira operação que sobe para ser processada.

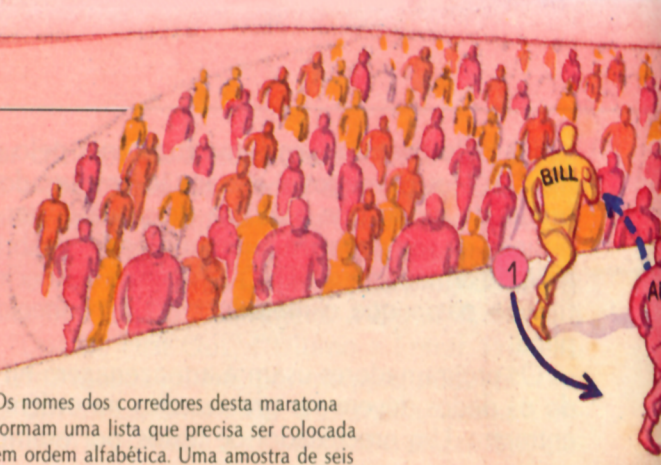
Quando o cálculo é efetuado, a soma, 11, é novamente colocada na pilha.

A próxima linha a ser processada é a multiplicação de 2 por 11.

O produto desse cálculo, 22, volta para a pilha.

Agora o único item da pilha, 22, é apresentado como resposta.

Técnicas de ordenação e pesquisa



Os nomes dos corredores desta maratona formam uma lista que precisa ser colocada em ordem alfabética. Uma amostra de seis corredores tirados do total será usada para demonstrar três métodos de ordenação. Para colocá-la em ordem alfabética usando exclusivamente o método da bolha, o computador faz doze comparações e trocas. Usando o método de partição, são necessárias oito mudanças, para três partições e cinco subpartições. Uma combinação dos métodos da bolha e de partição resolve o problema com apenas seis movimentos.

A ordenação de listas de dados de maneira organizada é uma função de computação tão importante e se tornou uma arte tão destacada no mundo da programação que foram criados programas específicos para essa tarefa. Ao escrever qualquer tipo de programa onde a ordenação seja usada, os programadores podem esperar até que o resto do programa esteja terminado e depois simplesmente selecionar o método mais apropriado.

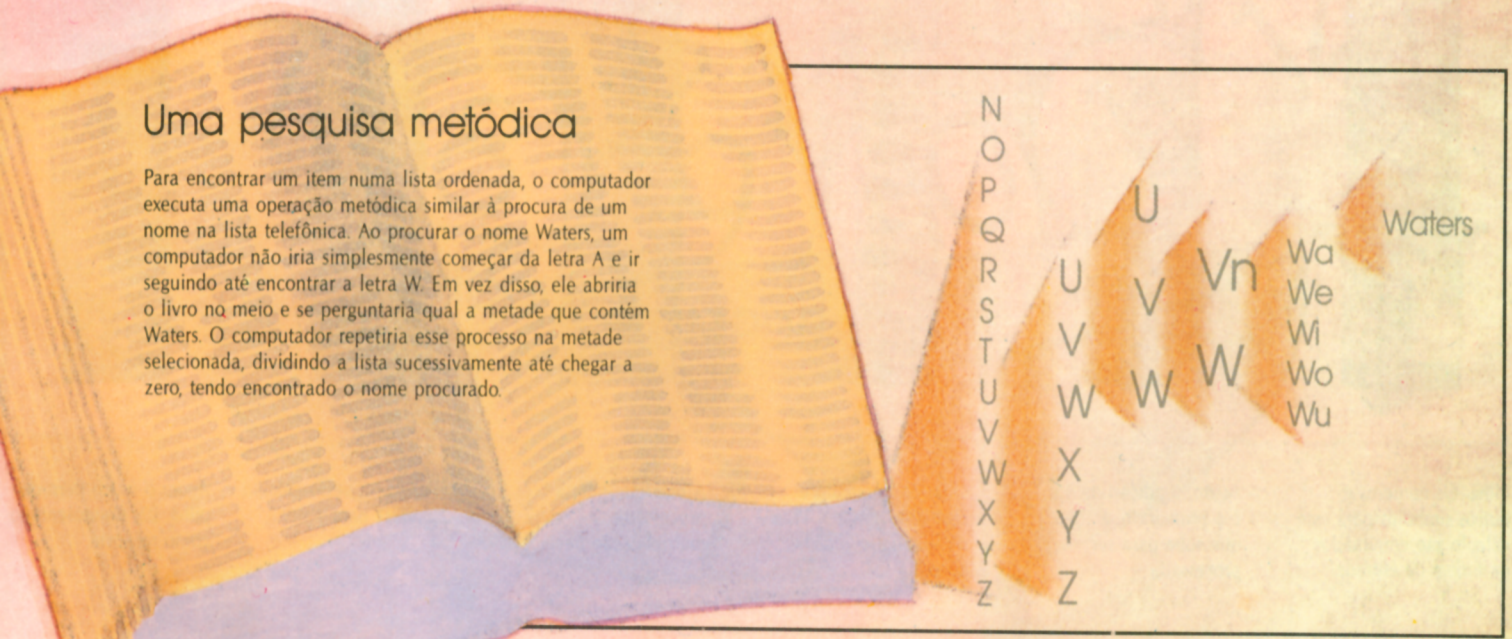
Duas técnicas de grande popularidade são ilustradas aqui. A ordenação por bolha foi assim chamada porque alguns dos itens, ao serem ordenados, sobem na lista como bolhas em líquidos. No desenho à direita, o corredor chamado Alex aparece subindo até o topo da lista alfabética. Um programa de ordenação por bolha vai percorrendo a lista do começo até o fim, comparando o primeiro e o segundo itens, depois o segundo e o terceiro, e assim por diante, invertendo os membros de cada par se estes não estiverem na ordem correta. Ao chegar ao fim da lista, o programa volta ao começo e faz seu caminho até o fim novamente, repetindo o processo até que todos os itens estejam na ordem correta.

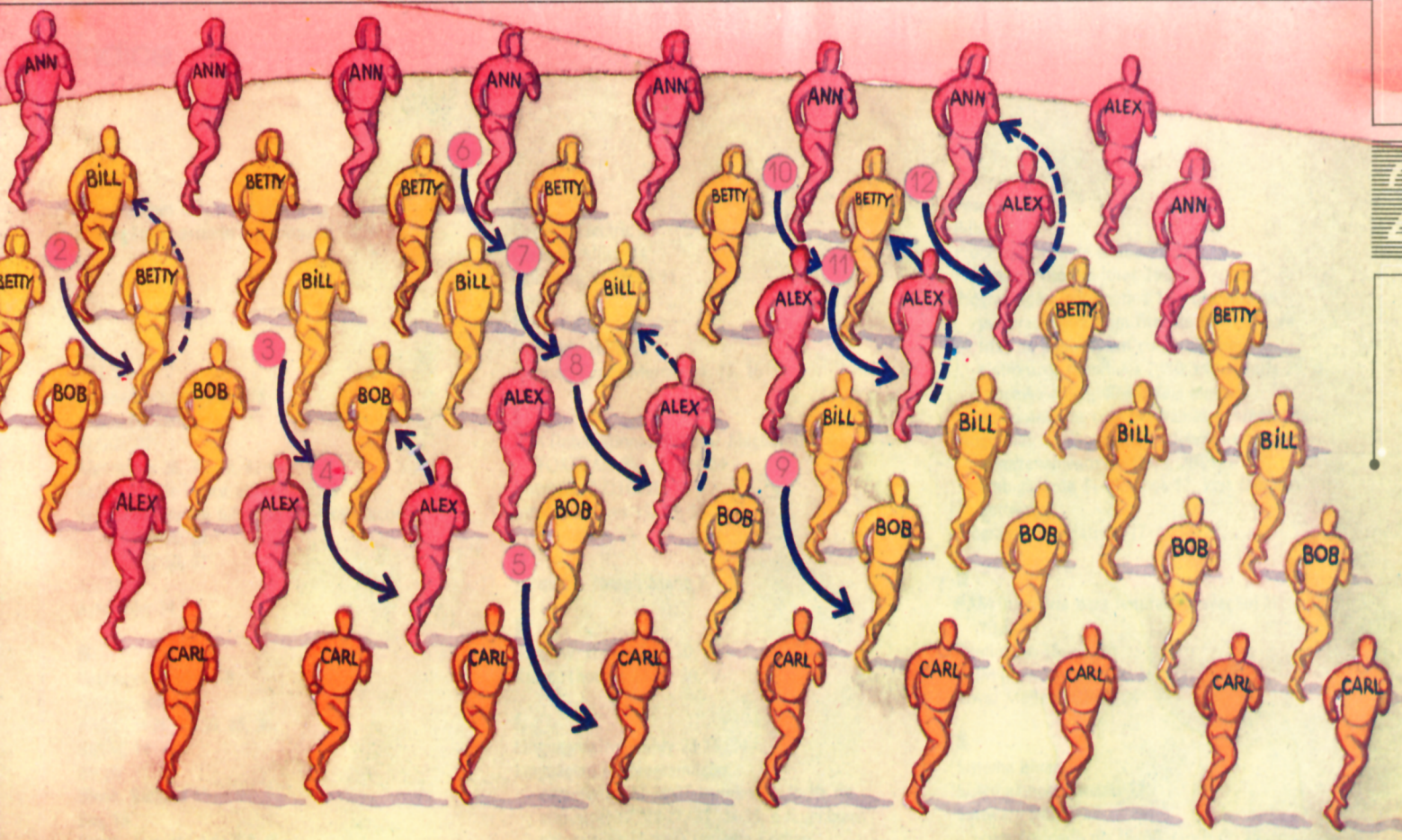
Já um programa de classificação por partição coloca os itens em partições imaginárias, uma para cada letra, por exemplo, se a lista estivesse sendo ordenada alfabeticamente: depois divide cada partição em subpartições e continua ordenando até que cada item esteja sozinho em sua partição. Para algumas listas longas, uma combinação das duas técnicas pode ser mais eficiente do que qualquer uma isoladamente.

O programa de ordenação por bolha começa comparando os dois primeiros nomes da lista, Bill e Ann, e trocando sua ordem (1). Bill, que agora é o item 2 da lista, é então comparado com Betty e trocado (2), passando a ser o item 3. O programa então compara o item 3 com o item 4: não há mudança (3). Bob, o item 4, é comparado e trocado com Alex (4), e depois com Carl (5). A cada passada subseqüente pela lista, o programa move Alex uma posição para cima, até que a lista esteja em ordem.

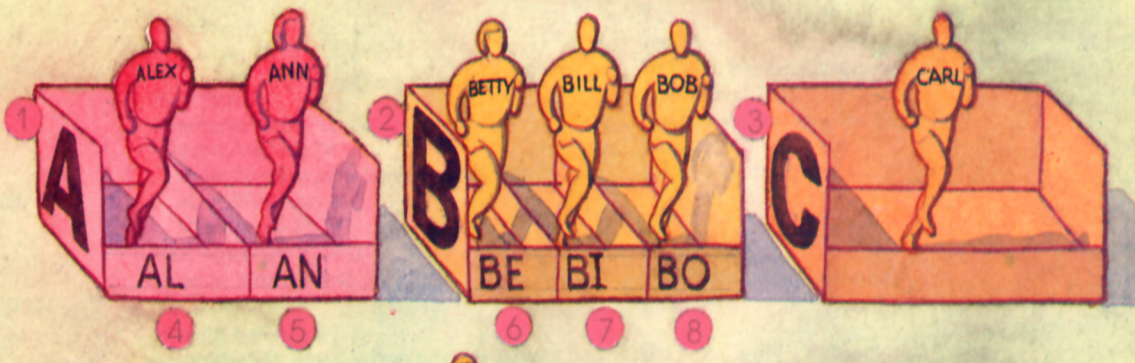
Uma pesquisa metódica

Para encontrar um item numa lista ordenada, o computador executa uma operação metódica similar à procura de um nome na lista telefônica. Ao procurar o nome Waters, um computador não iria simplesmente começar da letra A e ir seguindo até encontrar a letra W. Em vez disso, ele abriria o livro no meio e se perguntaria qual a metade que contém Waters. O computador repetiria esse processo na metade selecionada, dividindo a lista sucessivamente até chegar a zero, tendo encontrado o nome procurado.

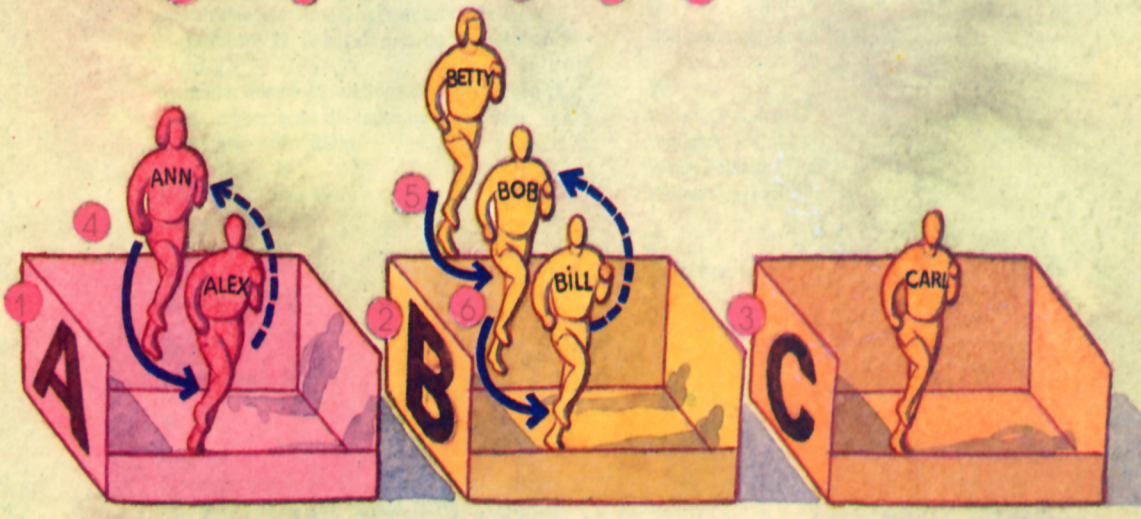




Um programa de ordenação por partição imediatamente coloca Ann e Alex na partição A; Bill, Betty e Bob na partição B; e Carl na partição C. A partição A é então dividida nas partições Al e An, e a partição B é dividida em Be, Bi e Bo. Como não há necessidade de subpartições em C, a ordenação se completa depois de oito passos. Se houvesse uma Alice na partição Al junto com Alex, ou um Ben com Betty, mais movimentos seriam necessários.



A maneira mais eficiente de ordenar a lista consiste em combinar os métodos, colocando-se os seis nomes em três partições e depois começando-se a usar o método da bolha. Nas partições A e B, os nomes são ordenados com três movimentos do tipo comparação e troca em vez de cinco movimentos para criar cinco subpartições. Novamente, a partição C não precisa ser ordenada; portanto, o trabalho completa-se em seis passos.





Índice

Numerais em *itálico* indicam uma ilustração do assunto mencionado.

A
ACE 11
Acionadores de discos *18*
Aiken, Howard 9
ALGOL 27
Algoritmos 28, 31, 40, 42-43
ALU (unidade lógica aritmética) 19
Armazenagem em tubos de raios catódicos 12
  A. FIFO *50*, 51
  A. LIFO 50, *51*
ASCII 21
Assembler 34

B
Babbage, Charles *7-8*
Backus, John 27
BASIC 24-26, 28-30, 35, *36*
BINAC 14
Bit 20
Block, Richard 15
Bug 15
Byron, Augusta Ada 7-8, 9
Byte 20

C
Campbell, Robert 15
Cartões perfurados 7-9
COBOL 27, 36, *37*
Código 12; curto 14-15; de máquina 12; simplificação do 14-16
Códigos (formatos), tipos *20-21*
Colossus 11
Compilador 17, 23, 35
Computadores com programas em memória 10-12; o primeiro *11*
Computadores eletrônicos primitivos 10
Conjuntos de instruções 19, 21
CPU (unidade central de processamento) 19-21

D
Dartmouth, Escola 23, 24-26
Datanet-30 26
Debug 15, 24, 30
Desvio condicional 28
Discos *18*
Dijkstra, Edsger J. 28-29, 30

E
Eckert, J. Presper 10-12, 14
Eckert-Mauchly Computer Corporation 11, 15
EDSAC *12*
EDVAC 10-11, *12*
ENIAC *6-7*, 10

F
Fila *50*, 51
Fluxograma *46-47*
FORTH 36, *37*
FORTRAN 25, 27, 35, *37*

G
General Electric 26
Glennie, Allick 14
Goldstine, Herman 11

H
Hollerith, Herman 9
Hopper, Grace Murray 14, *15*, 16

I
IBM (International Business Machines) 9; e FORTRAN 25, 37; 704 24
Instrução GOTO 29, *38*
Interpretadores 15, 23, 35

J
Jacquard, Joseph Marie 7

K
Kemeny, John 23-26, 30, 36
Kurtz, Thomas 23-26, 30, 36

L
Linguagens Assembly 23-24, 34
Linguagens de programação: características *36-37*; comandos *38-39*; de alto nível 17, 23-27, 31, 34-35; e linguagens naturais *32-33*. *Veja também* Código
Linhas de retardo de mercúrio 10, 12
LISP *36*
Listas encadeadas *48-49*
LOGO 27
Loop 9

M
Mainframe 25
Manchester Mark I 12, 14
Máquina Analítica *7-8*
  M. de Computação Automática (ACE) 11-12
  M. de Diferenças 8
  M. de Turing 13
Mark I (Harvard) 9, 14-15
Mauchly, John W. 10-12, 14
McCarthy, John 36
McNulty, Kathleen 6-7
Mecanismos de armazenagem: tubos de raios catódicos 12; linhas de retardo de mercúrio 10, 12
Memória: dumps 24; eletrônica versus mecânica 10; organização de dados na *48-49*. *Veja também* RAM; ROM
Mills, Harlan 29
Monitor de vídeo 19
Moore, Charles 37

N
Newman, Max 12

O
Ordenamento *52-53*

P
Pane 30
Pascal 27, 29-30, 36, *37*
Periféricos de input e output *18-19*
Pesquisa *52*
Pilha *51*
Processamento de textos 48, 51
Processamento em lotes 24-25
Programação: automática 16; com cartões perfurados 7-9; com fitas de filme 9; com fitas de papel 9; em computadores eletrônicos primitivos 7, 10; estruturada 29; o processo da 31, *40-45*. *Veja também* computadores com programas em memória.
Programas: acesso a *20-21*; algoritmos em 28; armazenamento de 18-19, *20-21*; e ambigüidade 31-33; tipos 17. *Veja também* Software.
Programas Aplicativos 17

R
RAM (memória para leitura e gravação) 18, *19-21*
Remington 15
ROM (memória só para leitura) *19-20*
Royal McBee LGP-30 24

S
Sistema binário 9
Sistemas operacionais 17
Software 5-6, 17-22
  complexidade do 28, 30; custo do 28
Software do ônibus espacial 5
Sperry Rand 15
Sub-rotina 15

T
Teclado *18*
Técnicas de ordenação e pesquisa 52-53
Time sharing 25-26
Tipos de armazenagens de dados *48-51*
Tradutores 17
Turing, Alan 11-12, *13*, 14

U
UNIVAC 14-16
Universidade da Pennsylvania 7
Utilitários 17

V
Von Neumann, John *10-11*, 23

W
Wilkes, Maurice 12
Williams, F. C. 12
Wirth, Niklaus 27, 37
Womersley, J. R. 11

Z
Zuse, Konrad 9

ISBN: 85-13-00085-X