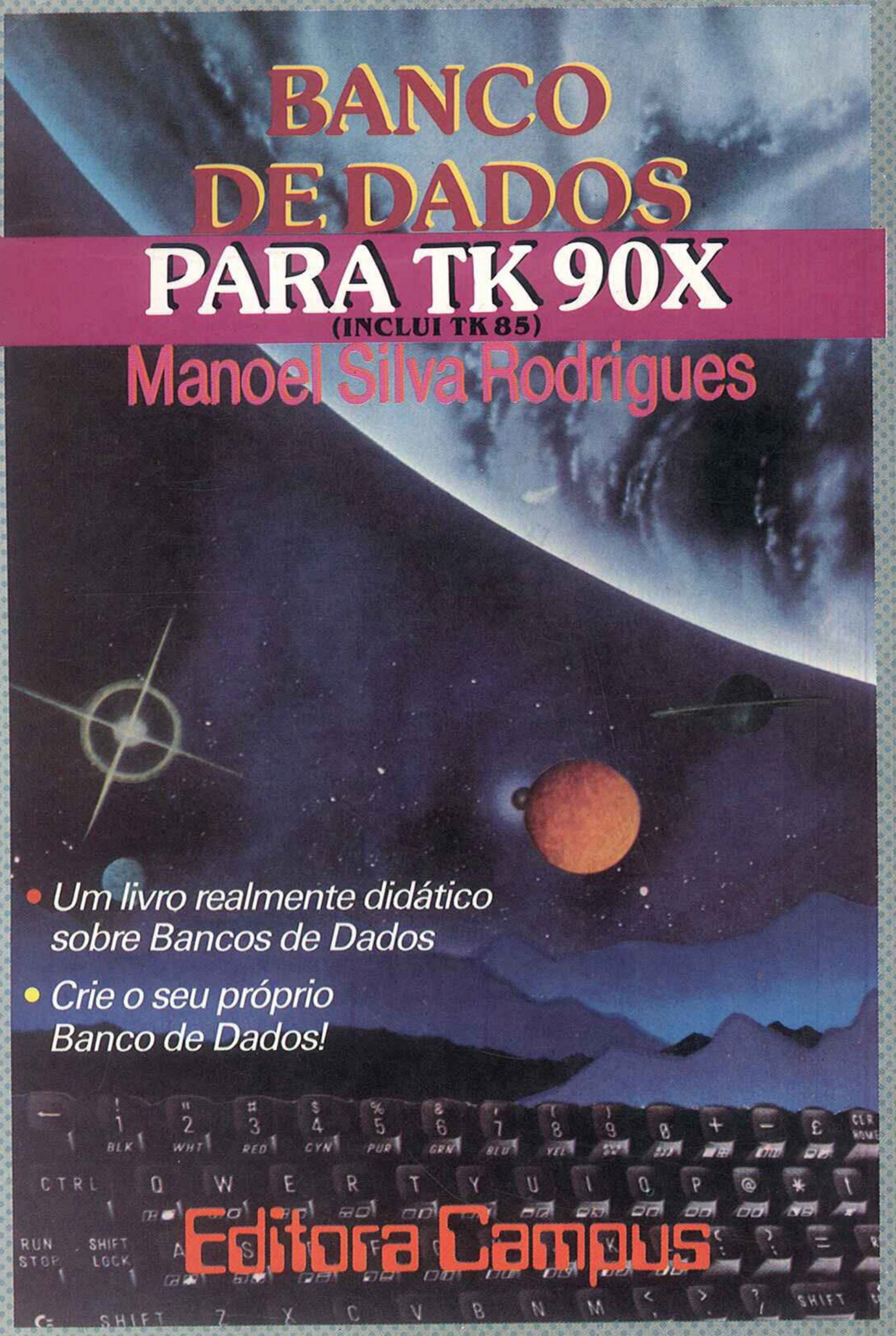
BANCO DE DADOS PARA TK 90X

(INCLUI TK 85)

Manoel Silva Rodrigues



- *Um livro realmente didático sobre Bancos de Dados*
- *Crie o seu próprio Banco de Dados!*

Editora Campus

TÍTULOS DE INTERESSE CORRELATO

FUNDAMENTOS DE ESTRUTURAS DE DADOS, 2ª ed. – *E. Horowitz e S. Sahni*

ESTRUTURAS DE DADOS, 3ª ed. – *P. Veloso et al.*

INTRODUÇÃO A SISTEMAS DE BANCOS DE DADOS, 2ª ed. – *C. J. Date*

BANCOS DE DADOS: FUNDAMENTOS – *C. J. Date*

ORGANIZAÇÃO DE BANCOS DE DADOS, 5ª ed. – *A. L. Furtado e C. S. dos Santos*

PRINCÍPIOS DE SISTEMAS DE GERÊNCIA DE BANCOS DE DADOS DISTRIBUÍDOS – *M. A. Casanova e A. V. Moura*

GERÊNCIA DE BASES DE DADOS PARA MICROCOMPUTADORES – *E. G. Brooner*

TÉCNICAS DE GERENCIAMENTO DE ARQUIVOS – *B. G. Claybrook*

FUNDAMENTOS DE PROCESSAMENTO DE DADOS – *W. T. Price*

CONCEITOS DE PROCESSAMENTO DE DADOS COM BASIC – *R. A. Stern e N. Stern*

ADMINISTRAÇÃO DE SISTEMAS DE INFORMAÇÃO, 2ª ed. – *P. Ein-Dor e E. Segev*

ANÁLISE DE SISTEMAS PARA SISTEMAS DE INFORMAÇÃO POR COMPUTADOR – *J. C. Wetherbe*

ORGANIZAÇÃO DA INFORMÁTICA NA EMPRESA – *E. Rios*

PLANEJAMENTO DE CAPACIDADE DE SISTEMAS DE COMPUTAÇÃO – *D. Menascé e V. Almeida*

REDES DE COMPUTADORES, ASPECTOS TÉCNICOS E OPERACIONAIS – *D. Menascé e D. Schwabe*

INTRODUÇÃO À SEGURANÇA DO COMPUTADOR – *M. B. Wood*

Conheça toda a linha de Informática da Editora Campus, com títulos nas áreas de: Introdução à Computação; Teoria, Organização e Processamento de Dados; Linguagens; Programação; Programas e Aplicativos; Sistemas Operacionais e Compiladores; Arquitetura e Equipamentos; Interesse Especial.

E, ainda:

DICIONÁRIO ENCICLOPÉDICO DE INFORMÁTICA – *A. H. Fragomeni*
Extenso e abrangente, reúne mais de 33.000 entradas em inglês e português pertencentes aos mais diversos campos da Informática e áreas correlatas.

O COMPUTADOR ENGUIÇOU – *Gabor*
Livro de humor, fartamente ilustrado, analisando espirituosamente a relação homem x máquina.

Procure nossas publicações nas boas livrarias ou comunique-se diretamente com:

EDITORA CAMPUS LTDA.
Livros Científicos e Técnicos
Qualidade internacional a serviço do autor e do leitor nacional.
Rua Barão de Itapagipe, 55
20261 Rio de Janeiro – RJ – Brasil
Telefone: (021) 284 8443

Atendemos também pelo reembolso postal.

BANCO DE DADOS PARA TK 90X

(INCLUI TK 85)

Consultor Técnico

Solon Benayon da Silva
Engenheiro Eletrônico UFRJ
Analista de Sistemas e Pesquisador
do Instituto Latino-Americano
de Pesquisas e Sistemas da IBM.

AVISO IMPORTANTE

Este(s) programa(s) e/ou sua documentação são vendidos na sua forma atual sujeitos à concordância implícita com o que se segue:

Devido à diversidade de condições e equipamentos nos quais o(s) programa(s) pode(m) ser usado(s), a Editora Campus Ltda. não garante sua adequação a propósitos específicos. É recomendável que o(s) programa(s) seja(m) cuidadosamente testado(s) antes do uso, cujo risco corre inteiramente por conta do usuário.

Em nenhuma circunstância caberá à Editora Campus Ltda. ou ao(s) autor(es) qualquer responsabilidade, incluindo, mas não se limitando, toda e qualquer obrigação ou ônus relativos a quaisquer prejuízos decorrentes do uso do(s) programa(s).

EDITORA CAMPUS LTDA.

BANCO DE DADOS PARA TK 90X

(INCLUI TK 85)

Manoel Silva Rodrigues

Analista de Sistemas da RFFSA

Professor da Faculdade SIMONSEN

Editora Campus Ltda.

Rio de Janeiro

Capa
Otavio Studart

Projeto Gráfico, Composição e Revisão
Editora Campus Ltda.
Qualidade internacional a serviço do autor e do leitor nacional.
Rua Barão de Itapagipe 55 Rio Comprido
Telefone: (021) 284 8443
Telex (021) 32606
20261 Rio de Janeiro RJ Brasil
Endereço Telegráfico: CAMPUSRIO

ISBN 85-7001-288-8

Ficha Catalográfica
CIP-Brasil. Catalogação-na-fonte.
Sindicato Nacional dos Editores de Livros, RJ.

Rodrigues, Manoel Silva.
R696z Banco de dados para TK90X (inclui TK85) / Manoel Silva Rodrigues. — Rio de Janeiro : Campus, 1986.

Bibliografia.
ISBN 85-7001-288-8

1. Banco de dados. 2. Microcomputadores — Programação. I. Título.

CDD — 001.642
85-0822 CDU — 001.5425

SUMÁRIO

ANEXOS

ÍNDICE DE FIGURAS

DEDICATÓRIA

A minha esposa e meus filhos pelos momentos, horas e dias em que os troquei por este trabalho.

AGRADECIMENTOS

Sinto necessidade de demonstrar meus agradecimentos. Na verdade, acredito que nada neste mundo possa ser produzido exclusivamente sozinho.

Agradeço, então, ao Luiz Scultori, colega de trabalho e professor de COBOL em cursos regulares da PUC-RJ, por ter me ajudado na otimização da Bubble Sort.

A meu filho Michel, pela colaboração em alguns desenhos.

À minha esposa Fernanda, pela paciência em datilografar e redatilografar os originais.

Em particular, ao David Anderson, editor da revista **MICROBITS**, pois foi quem vislumbrou a possibilidade de se transformar um pequeno artigo para revista neste livro. E também por ter me apresentado à Editora CAMPUS.

Manoel, 1985.

OBSERVAÇÕES SOBRE O TK90X

Após este livro estar totalmente escrito, foi lançado o "primo rico" da linha Sinclair, o TK9ØX, compatível com o ZX-Spectrum.

Assim sendo, o autor e o Editorial da CAMPUS desenvolveram um esforço para compatibilizar o livro com o TK9ØX. Sem perder a validade para o TK85 e família.

Foram desenvolvidas diversas observações identificadas no texto por (9ØX99), ou seja, 9ØX seguido do número correspondente à observação.

Os possuidores do TK9ØX devem ler normalmente o livro; encontrando o sinal identificador, reportar-se à observação correspondente e . . . bom estudo de BANCO DE DADOS NO TK9ØX!

Agora seguem-se as observações:

(9ØXØ1)

Os caracteres gráficos existem no TK9ØX, mas são obtidos de forma diferente. Observe:

– DB (graphics shift 8)
– ARQUIVO (graphics shift 2)
– REGISTRO (graphics shift 3)
– DADO (graphics 4)

(9ØXØ2)

No TK9ØX você pode usar indistintamente maiúsculas e minúsculas, inclusive para variáveis. O default é a letra minúscula. Quanto a variáveis, o TK9ØX não diferencia "a" de "A". Experimente: digite LET a = 2: PRINT A e você obterá 2 no vídeo!

(9ØXØ3)

Não existem no TK9ØX os comandos SLOW e FAST. A velocidade-padrão do TK9ØX é o FAST, sem que isto tenha o "efeito colateral" de fazer desaparecer a tela, como no TK85.

Logo, a linha 110 deve ser suprimida.

(9ØXØ4)

Para se criar uma "pausa infinita" (até que se pressione uma tecla) no TK9ØX o comando é PAUSE 0, em vez de PAUSE 4E4. A propósito, o comando PAUSE no TK9ØX não provoca nenhum "tremor" na tela, como no TK85. Assim, a linha deve ser

210 PAUSE 0.

(9ØXØ5)

O TK9ØX não tem o recurso das "aspas duplas" para introduzir o caractere *aspas* dentro de uma string. Como alternativa sugerimos a utilização do apóstrofo (symbol shift 7).

2500 PRINT "PRINT DO 'DB' "

(9ØXØ6)

Se a tela ficar completamente cheia, surge a mensagem do sistema operacional: scroll?

Se o scroll for desejado, toque qualquer tecla, excetuando N ou BREAK SPACE. Se não for, toque uma destas duas.

Assim, deve-se suprimir a linha 2510.

(9ØXØ7)

O sistema operacional do TK9ØX já providencia a impressão de mensagens relativas ao preparo do gravador e aguarda que você o ligue. Assim sendo, suprima as linhas 3500 e 3510.

(9ØXØ8)

O Capítulo 24 do manual do TK9ØX explica a organização da memória RAM. Estude-o com cuidado.

Para se obter o tamanho do programa BASIC apenas, é necessário fazer "VARADD" – "PROGBAS", ou seja:

(PEEK 23627 + 256 * PEEK 23628) – (PEEK 23635 + 256 * PEEK 23636)

Para se obter o tamanho de PROGRAMA + TELA + VARIÁVEIS, é necessário fazer "INADD" – 16384, ou seja:

(PEEK 23641 + 256 * PEEK 23642) – 16384

Finalmente, para se obter a memória disponível, devemos fazer "RAMTOP" – "ADSPFREE", ou seja:

(PEEK 23730 + 256 * PEEK 23731) – (PEEK 23653 + 256 * PEEK 23654)

Logo, esta sub-rotina deve ser digitada assim:

```
3000 PRINT AT 5, 5; "MAPA DA RAM (EM BYTES)"
3010 PRINT,, "Tamanho do programa"; TAB 25; PEEK 23627 + 256 * PEEK
23628 – PEEK 23635 – 256 * PEEK 23636, "(Prog.+ Tela + Variáveis)"; TAB 24;
"("; PEEK 23641 + 256 * PEEK 23642 – 16384; ")", "Disponível"; TAB 25;
```

PEEK 23730 + 256 * PEEK 23731 – PEEK 23653 – 256 * PEEK 23654
3020 RETURN

(9ØXØ9)

Ver 9ØXØ3; suprimir a linha 1000

(9ØX1Ø)

Ver 9ØXØ3; suprimir a linha 1140

(9ØX11)

Ver 9ØXØ3

(9ØX12)

Ver 9ØXØ3; suprimir a linha 4050

(9ØX13)

Ver 9ØXØ3; suprimir a linha 4210

(9ØX14)

Ver 9ØXØ6; assim sendo a linha 4220 deve ser suprimida

(9ØX15)

Ver 9ØXØ3

(9ØX16)

Ver 9ØXØ6

(9ØX17)

Ver 9ØXØ3; suprimir a linha 2010

(9ØX18)

Ver 9ØXØ3; suprimir a linha 2170

(9ØX19)

Ver 9ØXØ6; suprimir a linha 2200

(9ØX2Ø)

Ver 9ØXØ6; suprimir a linha 2240

(9ØX21)

Ver 9ØXØ6; suprimir a linha 2270

(9ØX22)

Ver 9ØXØ6

(9ØX23)

Ver 9ØXØ3

(9ØX24)

Ver 9ØXØ6

(9ØX25)

Ver 9ØXØ3

(9ØX26)

Ver 9ØXØ3

(9ØX27)

Embora possível, não há nenhuma vantagem em se utilizar as 24 linhas da tela do 9ØX, graças ao recurso de SCROLL automático que este micro apresenta.

CAPÍTULO

1

INTRODUÇÃO

COMO USAR ESTE MATERIAL

Primeiro, dê uma olhada no programa final. Tente entendê-lo. A listagem completa do programa está no final deste livro (página 84).

Com isto, poderá ter uma idéia da utilidade do que propomos.

Após este aquecimento, comece do início. Procure exercitar tudo que for indicado. Haverá muito o que fazer. Dê a maior atenção possível às observações, e mãos à obra!

PRÉ-REQUISITOS

Para o aproveitamento desejado deste texto, são necessários os seguintes requisitos:

HARDWARE – um microcomputador que use a lógica Sinclair, qualquer um, com um mínimo de 16K de memória RAM e um gravador.

SOFTWARE – por parte do leitor, conhecimento elementar de operação do microcomputador em questão e da linguagem BASIC.

PALAVRAS INICIAIS

Serão vistas algumas técnicas: concatenação, pesquisa em strings e em substrings, uso de sub-rotinas, economia de memória, cálculo de memória, utilização de variáveis do sistema operacional etc. Com isto, além da finalidade-título deste trabalho, haverá uma outra, que é a prática de alguns aspectos do BASIC, que nem sempre são usados; isto é, haverá chance de o leitor praticar técnicas e comandos pouco usados.

Apesar de ser todo voltado para os microcomputadores da linha Sinclair, a idéia é totalmente adaptável a outros micros.

Este livro originalmente era um artigo para ser publicado numa revista especializada. Graças ao incentivo de David Anderson, editor da **MICROBITS**, resolvemos ampliá-lo e alcançamos este resultado.

Bom proveito!

OBJETIVO DESTE TRABALHO

Objetivamos mais o aspecto didático do que o de uso efetivo para aplicações. Tentamos transmitir o conceito de BANCO DE DADOS usando como meio um pequeno computador. É um trabalho voltado totalmente para iniciantes no assunto, como o são a maioria dos usuários deste tipo de microcomputador. Na verdade, leva à criação de um programa que simula um BANCO DE DADOS on-line, totalmente conversacional e monousuário.

Um outro aspecto que deve ser ressaltado – talvez o mais importante – é a forma como é conduzido, pois permite ao final da leitura um total domínio por parte do leitor sobre o programa apresentado ao seu término, a ponto de serem possíveis implementações e alterações por parte do mesmo. Na verdade, implementações serão necessárias para que o programa se ajuste às conveniências do leitor.

O desenvolvimento é todo prático, daí ser necessário o uso concomitante de um microcomputador.

Apesar da restrição citada ao início deste capítulo, o programa final poderá ser usado dentro de suas limitações, isto é, pequenos volumes de dados e uma velocidade de consultas relativamente lenta.

Diferentemente dos DBase's, TK-File's, ZX-File's etc., que existem no mercado para linha Sinclair – excelentes softwares por sinal, mas que são na verdade geradores de fichas, o que não tem nada a ver com BANCO DE DADOS – este trabalho proporcionará ao leitor funções que relacionam dados (cálculo do salário líquido, proventos menos descontos, por exemplo).

Além de incluir e excluir registros e arquivos individualmente, permitirá listar dados de apenas um determinado arquivo escolhido pelo interessado, sendo que estes dados poderão ter os cabeçalhos que melhor convier.

Procuramos dar o mais exato possível as idéias de "dado", "registro", "arquivo" e BANCO DE DADOS. Na medida do seu desenvolvimento, serão mostradas novas possibilidades.

O leitor verá o programa crescer aos poucos, sendo adaptado, quando for o caso, podendo, desta forma, assimilar a idéia do título e não pura e simplesmente usá-lo sem ter conhecimento do que se passa. Com isto, poderá servir também como um modelo de desenvolvimento de programa. É muito comum ter-se o computador e a vontade de fazer um programa, mas não se saber como e nem por onde começar. Este será um caminho. É claro que cada idéia, cada programa pode ter várias formas de execução. Esta é apenas uma delas.

Serão sugeridas algumas implementações, com a finalidade de permitir ao leitor saber até que ponto está assimilando a idéia. Por outro lado, essas implementações não foram incluídas no programa final, para possibilitar o cunho didático deste trabalho, permitindo ser usado, inclusive, na área de ensino.

Entendendo-se este objetivo, pode-se ver que seria sem finalidade crescer o programa final aumentando o tempo de processamento, apenas para tentar torná-lo perfeito, isto é, mais próximo da idéia real de BANCO DE DADOS. Achamos desnecessário. Consideramos suficiente o programa final; as implementações ficam a gosto do leitor e a título de treinamento.

Além do objetivo didático, fica mais do que evidente o aspecto prático deste trabalho. Poderíamos ter incluído mais teoria de BANCO DE DADOS, ampliando bem este material. Mas observamos que mesmo as pessoas que se deliciam com teo-

rias e conceitos não prescindem daquela "ferramenta", daquele lado prático para dar início real a sua satisfação. Somos da opinião de que os "primeiros passos" são os mais importantes. Só se deve querer correr quando estes estão firmes. A intenção é que estes sejam seus primeiros passos.

Futuramente, pretendo fazer o equivalente do **ZX-DB** Sinclair em ASSEMBLY. Toda a lógica empregada aqui no BASIC será a mesma, as grandes diferenças ficarão por conta do tamanho do programa, que será bem menor, e de sua velocidade de execução, muitas vezes mais rápida.

Está aí, inclusive, a sugestão de um bom divertimento para aqueles que gostam de se distrair com um bom passatempo, isto é, o ASSEMBLY do Z80.

Escolhi este nome, **ZX-DB**, para o programa básico apresentado ao final deste trabalho como uma homenagem aos primeiros microcomputadores desta linha, os ingleses ZX-80 e ZX-81, e ao seu criador, Sir Clive Sinclair.

COMO SURGE A NECESSIDADE DO BANCO DE DADOS

Num primeiro momento, surge o DADO: nomes, números etc.

Depois, surge a necessidade de agrupá-los, organizá-los de acordo com um fator comum. Assim: nome do José, endereço do José, telefone do José, CEP do José etc. (José é o fator comum.)

Formamos então o "REGISTRO". Registramos, guardamos, escrevemos os dados de acordo com este tipo de fator comum (*nome* é o tipo do fator comum) e formamos uma "AGENDA de amigos e conhecidos", por exemplo.

Temos aí o ARQUIVO, que poderá ter, inclusive, uma organização lógica conveniente (em ordem alfabética crescente de nome, de A a Z).

Com o arquivo, paramos um pouco. Criamos vários arquivos. Passamos a ser redundantes. Registramos a mesma informação em vários arquivos, de acordo com a organização que dermos. Um por ordem de nome, para quando tivermos o nome e quisermos o telefone; outro por ordem de telefone, para quando tivermos o telefone e quisermos o nome; outro por endereço, para quando quisermos saber quem mora próximo do outro; e por aí afora.

Este é um exemplo fictício, mas que serve para dar a idéia das diversas causas que nos levam a criar arquivos redundantes em termos de informação.

Nas suas necessidades, surge então o DB, DATA BASE, BANCO DE DADOS ou ainda BASE DE INFORMAÇÕES, que não passa de uma "reunião de vários arquivos comuns", tendo-se o cuidado de eliminar o mais possível as redundâncias de informações.

Este "comuns" pode ser apenas o fato de lhe pertencerem, e as informações serem bem diferentes e para vários objetivos.

Você simplesmente quer ter tudo num único lugar.

A ORGANIZAÇÃO DO BANCO DE DADOS

A organização do BANCO DE DADOS e o domínio sobre a mesma é fundamental para que o mesmo possa lhe dar o que você precisa: incluir, excluir ou alterar informações, que podemos, de uma forma bem simplificada, chamar de "dados", "registros" e "arquivos".

Nesta organização, deverão ser evitados espaços em branco e redundâncias, devendo mesmo ser eliminados para não ocuparem lugar desnecessariamente, visto

que espaços são de vital importância, pois deles dependem a capacidade de armazenamento, isto é, o tamanho do seu BANCO DE DADOS em termos de quantidade de informações que comportará, e também a velocidade de acesso (performance) às informações (pois, quanto maior o caminho, mais tempo será gasto para percorrê-lo).

Considerando os dois parágrafos anteriores, passaremos a detalhar a organização escolhida e a mostrar como a mesma foi adaptada aos recursos que dispõem os microcomputadores que trabalham com a lógica Sinclair.

Existem várias estruturas (formas) para os BANCOS DE DADOS; usaremos aqui a de "árvore hierárquica", que nos faz lembrar um organograma de uma organização militar (Fig. 1).

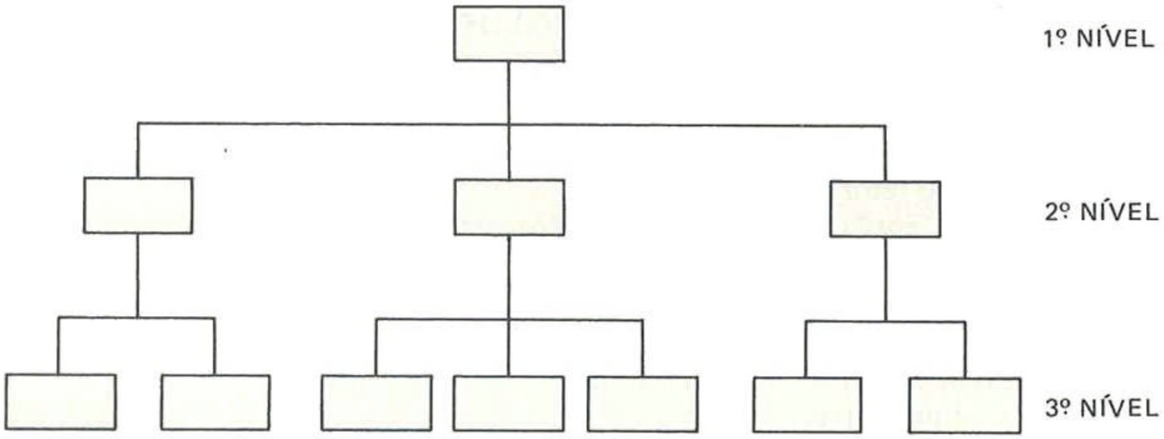


Figura 1 Árvore hierárquica.

Diremos, a princípio, que o BANCO DE DADOS pode ser visualizado de duas formas – a física e a lógica. O desenho da Fig. 1 nos dá uma visão da sua forma lógica, com seus níveis de importância, com o primeiro nível sendo mais importante que o segundo, o segundo mais que o terceiro e assim sucessivamente.

No BANCO DE DADOS, no primeiro nível, podemos ver o próprio BANCO DE DADOS, que por comodidade passaremos a designar simplesmente por DB, do inglês DATA BASE. No segundo nível, teremos os arquivos. No terceiro nível, os registros, e por último os dados.

Vamos inicialmente estabelecer identificadores para os níveis ou "elementos" do DB.

Para o DB, escolhemos a configuração do "espaço" invertido, pois representa o quadrado completo. Para o arquivo, o caráter gráfico gerado pelo pressionamento das teclas SHIFT + GRAPHICS e FUNCTION + Q, pois representa um quadrado sem o primeiro 1/4. Para o registro, o caráter gráfico gerado pelo pressionamento das teclas SHIFT + GRAPHICS e SHIFT + 6, pois representa 1/2 quadrado. Finalmente, para o dado, o caráter gráfico gerado pelo pressionamento das teclas SHIFT + GRAPHICS e SHIFT + 3, pois representa apenas 1/4 do quadrado.

 – DB

 – ARQUIVO

 – REGISTRO

 – DADO 9ØXØ1

Cada um com o tamanho equivalente ao seu nível de importância. Estando estes identificadores definidos, podemos começar a geração propriamente dita do DB.

FORMATAÇÃO ELEMENTAR DO BANCO DE DADOS

Continuando a conceituação e o domínio sobre a organização do DB, vamos começar sua formatação pelo elemento mais simples – "o dado".

Digite o programa que vem logo a seguir. É importante que exercite.

Após a digitação, execute-o.

As variáveis e suas finalidades:

A$ – usada para guardar os dados agrupados
N – para conter o número de dados que você deseja
A – indexador para os dados
D$ – para entrar cada dado. 9ØXØ2

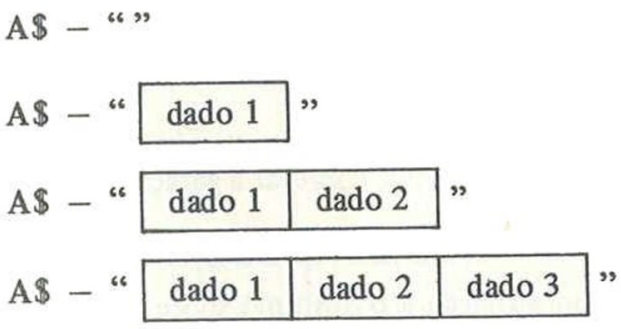
AGRUPANDO DADOS ELEMENTARES

```
 10 REM * AGRUPANDO DADOS *
 20 LET A$=""
 30 PRINT "QUANTOS DADOS DESEJA AGRUPAR ?"
 40 INPUT N
 50 FOR A=1 TO N
 60 PRINT "DADO ";A;" ?"
 70 INPUT D$
 80 LET A$=A$+"■"+D$
 90 NEXT A
100 PRINT A$
110 STOP
```

Quando solicitar – "QUANTOS DADOS DESEJA AGRUPAR?", digite um número qualquer.

Em seguida solicitará – "DADO 1?", digite o primeiro dado com o que desejar. Logo a seguir, "DADO 2?". Isto se repetirá até o total de dados solicitados por você.

Ao final, após a digitação do último dado, a variável A$ será exibida no vídeo.
Simples, não?
A forma (dado 1) + (dado 2) . . . é a mais elementar possível.
Cabe ressaltar o comando de número 80, que mostra a capacidade de expansão do arquivo A$. Esta é uma das características básicas do DB que vamos criar.
A seqüência seguinte ilustra o que dissemos em quatro momentos sucessivos:



Após seu exercício de execução do programa mostrado, digite os seguintes dados:
"3" – para "quantos dados deseja agrupar ?" e "José", "356-9911", "RUA ALICE NUMERO 39" como primeiro, segundo e terceiro dado, respectivamente.
Ao final, o programa exibe no vídeo a forma em que os dados ficam agrupados.
"■ JOSE ■ 356-9911 ■ RUA ALICE NUMERO 39"
Esta é a "forma física" do arquivo A$.
Observe o separador/identificador de dados (■).
A partir deste "agrupador", desenvolveremos todo o nosso DB.
Prossigamos com a conceituação.
Se guardarmos alguma coisa é porque pensamos ou pretendemos fazer uso dela futuramente; a isto damos o nome de "recuperação".

RECUPERANDO UM DADO

Vamos recuperar um dado apenas para exibi-lo no vídeo.
Em continuação ao programa anterior, digite os comandos que se seguem.

As variáveis e suas finalidades:

- D$ – será usado para conter o dado recuperado
- NUM – para receber o número do dado que se deseja recuperar
- A – indexador para os caracteres
- R$ – para conter o caráter recuperado (serão recuperados um-a-um)
- N – após o dado recuperado, N é usado para se verificar se este é o dado solicitado em NUM

```
120 REM * RECUPERANDO UM DADO *
130 LET D$=""
140 LET N=0
150 PRINT "QUAL DADO A RECUPERAR ?"
```

```
160 INPUT NUM
170 FOR A=1 TO LEN A$
180 LET R$=A$(A)
190 IF R$="■" THEN GOTO 220
200 LET D$=D$+R$
210 NEXT A
220 LET N=N+1
230 IF N=NUM+1 THEN GOTO 260
240 LET D$=""
250 GOTO 210
260 PRINT "DADO RECUPERADO = ";D$
270 GOTO 130
```

Crie vários arquivos (GOTO 10) e exercite a recuperação de vários dados comandando GOTO 130 para a recuperação dos mesmos.

Quando solicita: "QUAL DADO A RECUPERAR?", digite o número relativo ao dado que deseja separar dos demais.

Ao final, será exibido – "DADO RECUPERADO – . . . (conteúdo de D$)".

Este arquivo (A$) que criamos e do qual recuperamos seus dados é constituído de "dados independentes" ou "registros de um único dado" ou ainda de um único "registrão". Não tem maiores utilidades, apenas nos auxiliou na conceituação. Vamos organizá-lo melhor.

A forma física ideal deverá ser: (arquivo 1 (registro 1 (dado 1) (dado 2) . . .) (registro 2 (dado 1) (dado 2) . . .) . . .) (arquivo 2 (registro 1 (dado 1) (dado 2) . . .) (registro 2 (dado 1) (dado 2) . . .) . . .)).

Ou ainda:

■ ▟ ARQUIVO 1 ▬ REGISTRO 1 ▪ DADO 1 ▪ DADO 2 . . . ▬ REGISTRO 2 ▪ DADO 1 ▪ DADO 2 . . . . . ▟ ARQUIVO 2 ▬ REGISTRO 1 ▪ DADO 1 ▪ DADO 2 . . . ▬ REGISTRO 2 ▪ DADO 1 ▪ DADO 2 . . . . . .

Os três pontos seguidos (. . .) significam que os elementos respectivos continuam.

Esta, como dissemos, é a forma física, isto é, é a forma como os dados, registros e arquivos existem (residem) realmente na memória de seu microcomputador.

Observe os identificadores de dados (▪), registros (▬) e arquivos (▟).

Graficamente, visualmente ou ainda logicamente, podemos ver esta forma de arquivo, o DB, como mostra a Fig. 2.

Esta forma de representação é muito mais visual.

Ela evidencia os níveis de dependência (o dado é dependente do registro, que por sua vez é dependente do arquivo).

Com esta representação, pode-se ver o que se costuma conceituar como "nós". Cada círculo é um nó, e possui o dado. O que está acima é chamado de "pai" dos que estão abaixo. Os que estão abaixo são chamados de "filhos". Assim, podemos dizer que os registros são os "filhos" do arquivo e os dados "filhos" do registro. Os arquivos, como são os primeiros de cima para baixo, são também chamados de "raiz". É onde se inicia o "ramo" da árvore.

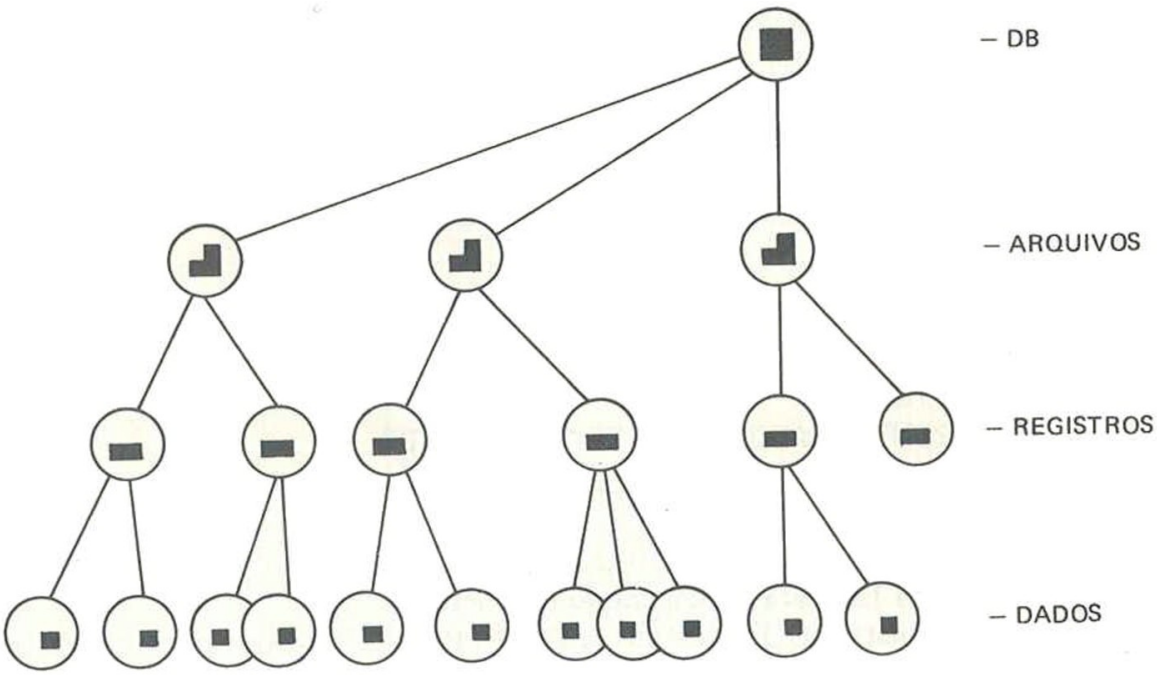


Figura 2 Forma lógica genérica.

Esta forma de representar o DB é apenas para facilitar a nossa compreensão, na verdade ela não existe fisicamente.

A forma física, nós já vimos. Nela os dados são enfileirados um depois do outro num "string" contínuo, no qual seus elementos (dados, registros e arquivos) são separados por símbolos que os identificam.

Estes separadores/identificadores (passaremos a chamar apenas de identificadores) servirão, como no comando 19Ø, para se chegar ao elemento desejado. No comando 19Ø, o identificador é usado para se chegar ao dado. Como deve ser notado, o primeiro identificador é anulado, quando é somada uma unidade à variável numérica NUM, no comando 23Ø. Desta forma, o primeiro dado só será encontrado e colocado na variável D$, quando o segundo identificador for encontrado.

A segunda parte deste programa, que chamamos RECUPERANDO UM DADO, mostra de modo simplificado a técnica que usaremos mais adiante, para chegarmos aos registros e arquivos.

O programa visto até agora nos foi útil para fixarmos certos conceitos e técnicas, que usaremos no programa principal. Aqui termina sua finalidade.

A partir do próximo capítulo, iniciaremos o estudo passo a passo do programa final, o qual nos levará a firmar a noção do que vem a ser, como se manipula e como funciona um DB.

Serão criadas e definidas sete funções, sendo:
três principais

- Criar Arquivos
- Eliminar Arquivos/Registros
- Listar Arquivo (no vídeo)

três auxiliares

- PRINT do DB
- Memória Disponível
- Salvar o **ZX-DB**

e uma específica

- Cálculo do Salário Líquido

Nada impedirá, porém, que o leitor crie outras funções. Por exemplo: listar o DB (no papel), muito útil para quem tem impressora. Será muito simples adaptar a terceira (Listar Arquivo no vídeo), apenas usando LPRINT no lugar do PRINT.

Este é apenasum exemplo, muitas outras funções poderão ser criadas.

Vamos então ao **ZX-DB**!

CAPÍTULO 2

O ZX-DB

COMEÇANDO A DESTRINCHAR O ZX-DB

Como nosso objetivo final é o DB, passaremos, a partir de agora, a designar o programa final apenas por **ZX-DB**.

CRIAR ARQUIVOS

Começaremos pela função "Criar Arquivos", uma das sete já mencionadas.

Escolhemos esta por uma questão óbvia – não poderemos manipular dados, registros ou arquivos, antes de criá-los!

As seguintes orientações preliminares são necessárias:

1 – Comande, antes da primeira execução do programa (apenas da primeira)

LET W$ = "■"

2 – Após o comando acima, não use mais o comando RUN.
Em seu lugar, use

GOTO 1ØØ.

Pois, como se sabe, o RUN apagaria todas as variáveis; no caso, apagaria inclusive W$.

Para total compreensão desta função, aqui estão as definições das variáveis usadas:

W$ – será o nosso arquivo principal, o próprio DB
N$ – para entrada dos nomes dos arquivos
R – para o número de registros de cada arquivo
D – para o número de dados de cada registro
A – para indexador de dado
B – para indexador de registro
D$ – para entrada dos dados

Lembro aqui que não há limite para o tamanho dos nomes dos arquivos ou para o tamanho dos dados. O limite fica por conta do tamanho do vídeo. Isto é muito confortável, pois permite usar nomes bem significativos e que nos facilitem lembrar suas finalidades. Uma orientação, porém, se faz necessária – não se deve abusar disto, usando-se nomes muito grandes – isto implicará uma demora maior na pesquisa para se localizar o nome desejado. O bom senso é o melhor guia.

Limpe a memória RAM de seu microcomputador, usando o comando NEW, pois iniciaremos um novo programa.

É proveitoso e didático que se digite os comandos do programa aqui apresentado. Por isto, sempre é indicada esta opção. Isto leva a uma maior assimilação. Solidifica o domínio sobre o programa.

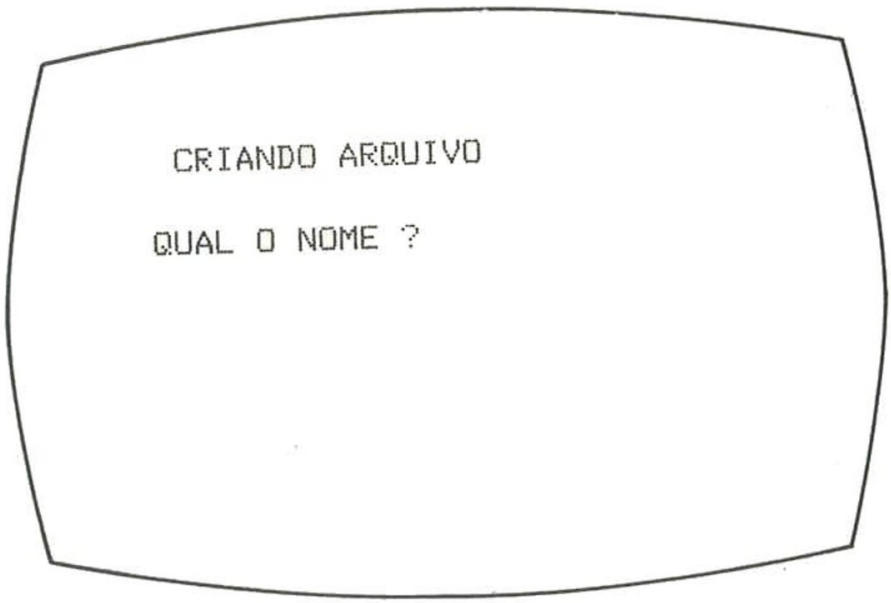
Os comandos que vêm a seguir fazem parte do programa DB, portanto, deve ser mantida a numeração usada nos comandos pois a mesma será referenciada em outras partes do programa.

Digite os comandos a seguir:

```
500 PRINT " CRIANDO ARQUIVO"
510 PRINT ,,"QUAL O NOME ?"
520 INPUT N$
530 LET W$=W$+"■"+N$+"▬"
540 PRINT "QUANTOS REGISTROS ?"
550 INPUT R
560 PRINT "QUANTOS DADOS ?"
570 INPUT D
580 CLS
590 PRINT N$
600 FOR B=1 TO R
610 FOR A=1 TO D
620 PRINT "DADO ";B;".";A;">";
630 INPUT D$
640 PRINT D$
650 LET W$=W$+D$+"■"
660 NEXT A
670 LET W$=W$( TO LEN W$-1)+"▬"
680 NEXT B
690 LET W$=W$( TO LEN W$-1)
```

Esta parte do programa pode ser executada como um programa completo, mas lembre-se: não use RUN! Use GOTO 5ØØ.

Devem surgir as seguintes informações no vídeo da TV:



Digite – "CADASTRO"

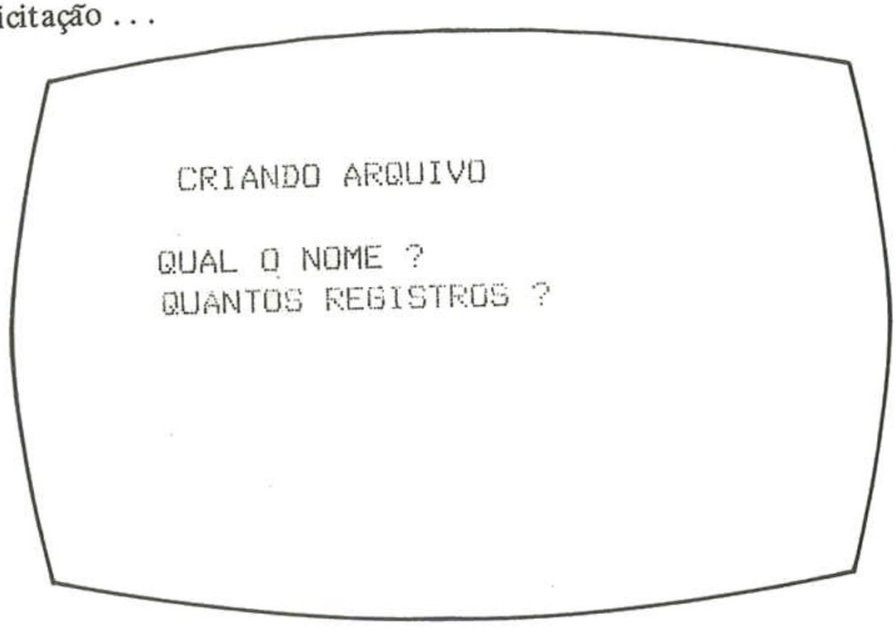
Não será verificada a existência de arquivos com o mesmo nome, visto que isto permite a criação de mais de um com o mesmo nome. Conseqüentemente, quando for acessá-lo, o primeiro encontrado é que será processado.

Tome cuidado!

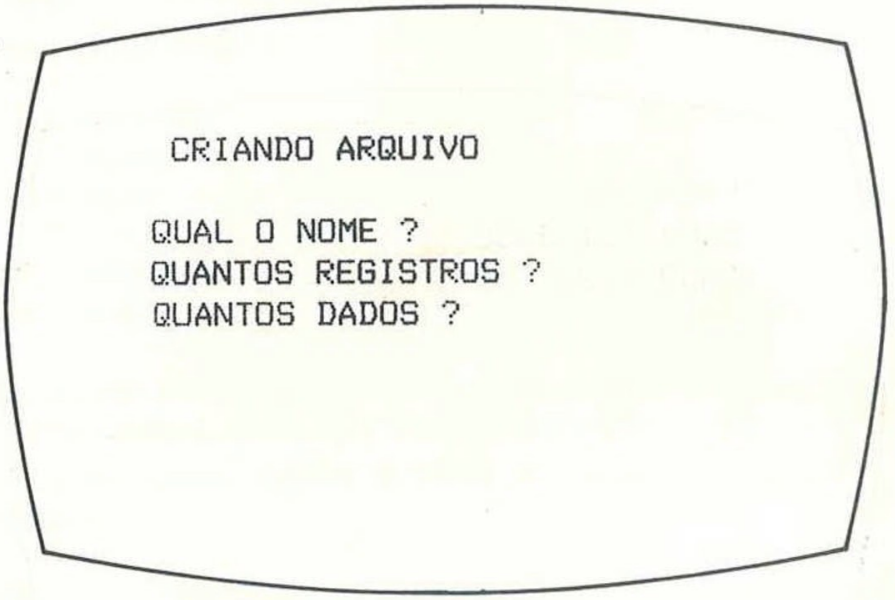
Evite usar nomes parecidos, pois podem confundir e levá-lo a criar nomes iguais. No caso de dúvida, use o comando PRINT W$ para ver os nomes já criados.

Aqui vai a primeira sugestão para implementações – crie outro arquivo do tipo W$ para conter os nomes de arquivos. Será o "diretório". Logo após entrar com um novo nome, faça uma pesquisa neste diretório para verificar se já existe algum com este nome. Caso exista, indique o fato como erro – "NOME JÁ EXISTENTE". Caso contrário, aceite o nome e inclua-o no "diretório". Mas deixe isto para mais tarde, no terceiro capítulo – PRÁTICA.

Nova solicitação . . .

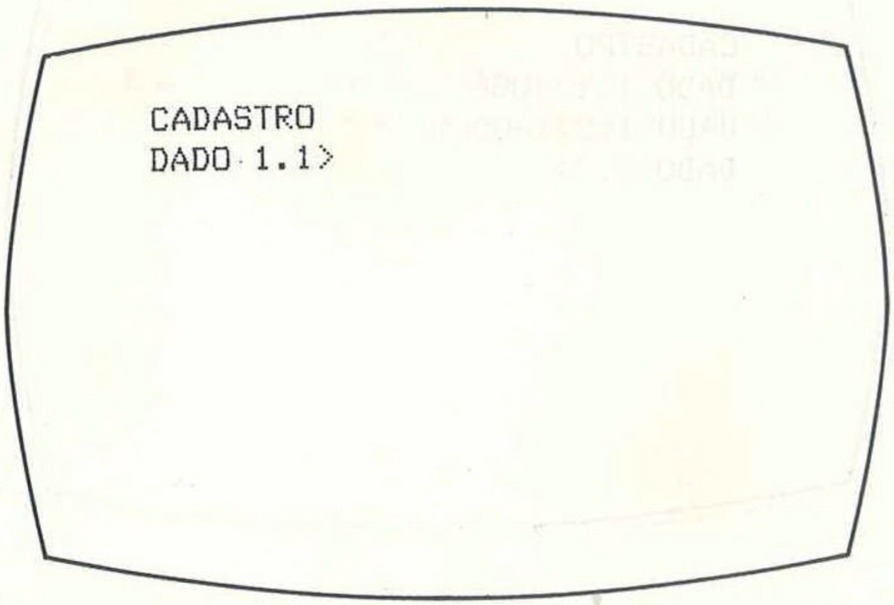


Digite – "3"
Nova solicitação . . .



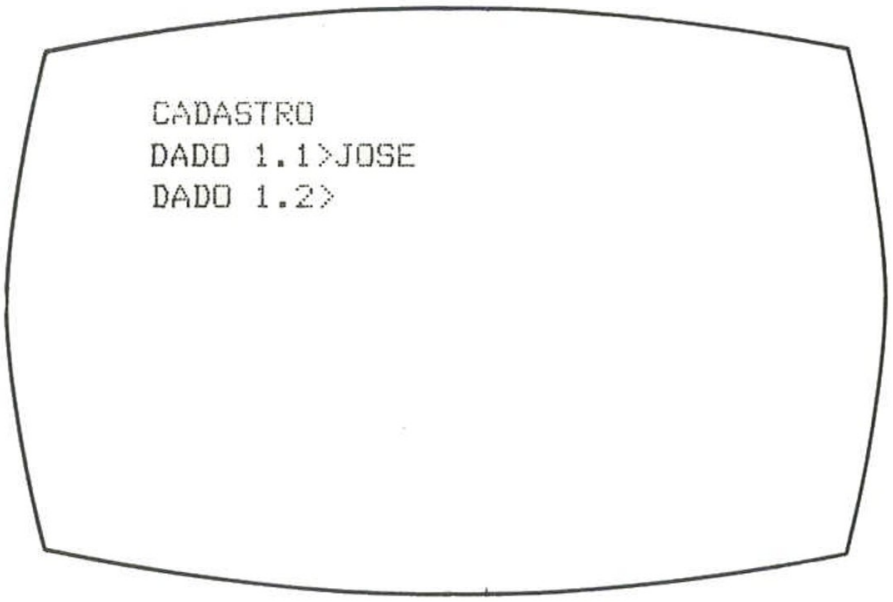
Digite – "3"
Nova solicitação . . .

Significa que teremos três registros com três dados cada.
Nova solicitação . . .

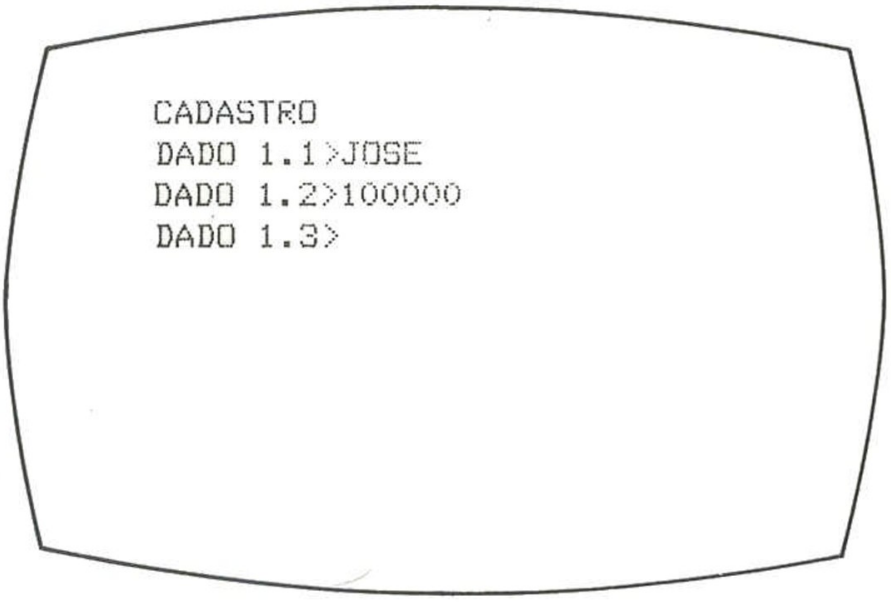


O "1.1" que aparece na tela significa "dado 1 do registro 1".

Digite – "JOSE"
Nova solicitação – pede agora o segundo dado do registro . . .



Digite – "1ØØØØØ"
E temos a solicitação do terceiro dado do primeiro registro . . .



Digite – "15ØØØ"

Surge agora "2.1" – significando "dado 1 do registro 2". Vai assim até "3.3", que será o último dado do último registro.

Digite então os seguintes dados:

2.1 – "ANTONIO"
2.2 – "2ØØØØØ"
2.3 – "3ØØØØ"
3.1 – "JOAO"
3.2 – "15ØØØØ"
3.3 – "1ØØØØ"

Teremos então um exemplo completo de um arquivo chamado "CADASTRO" com três registros apenas, tendo cada um deles três dados.

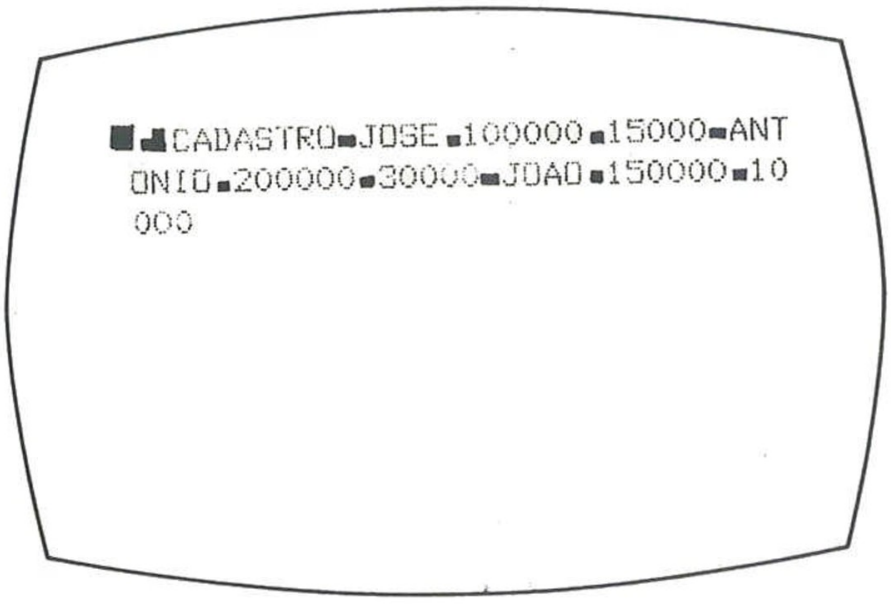
Vamos convencionar chamar os dados de "nome", "proventos" e "descontos", como segue:

NOME	PROVENTOS	DESCONTOS
JOSE	1ØØØØØ	15ØØØ
ANTONIO	2ØØØØØ	3ØØØØ
JOAO	15ØØØØ	1ØØØØ

Após execução, comande:

PRINT W$

Surgirá a seguinte visão . . . conforme a tela a seguir:



Em vez desta numeração (1.1, 1.2 etc.), poderia ter sido incluída uma sub-rotina em que fossem pedidos, além do nome do arquivo, os nomes dos dados. Quando fosse feita a "entrada" de dados, uma outra sub-rotina "recuperaria" estes nomes e os colocaria no vídeo, à medida que os mesmos fossem sendo solicitados. Tal não foi feito apenas por medida de economia.

Esta pode ser mais uma de suas implementações. Experimente!

Crie vários arquivos, com quantidade e tamanhos de registros e dados diferentes.

O comando anterior mostrou a forma física do DB. O ideal é que se execute várias vezes, quando então se poderá observar o conjunto de arquivos. A forma lógica fica sendo como mostra a Fig. 3, conforme tela seguinte.

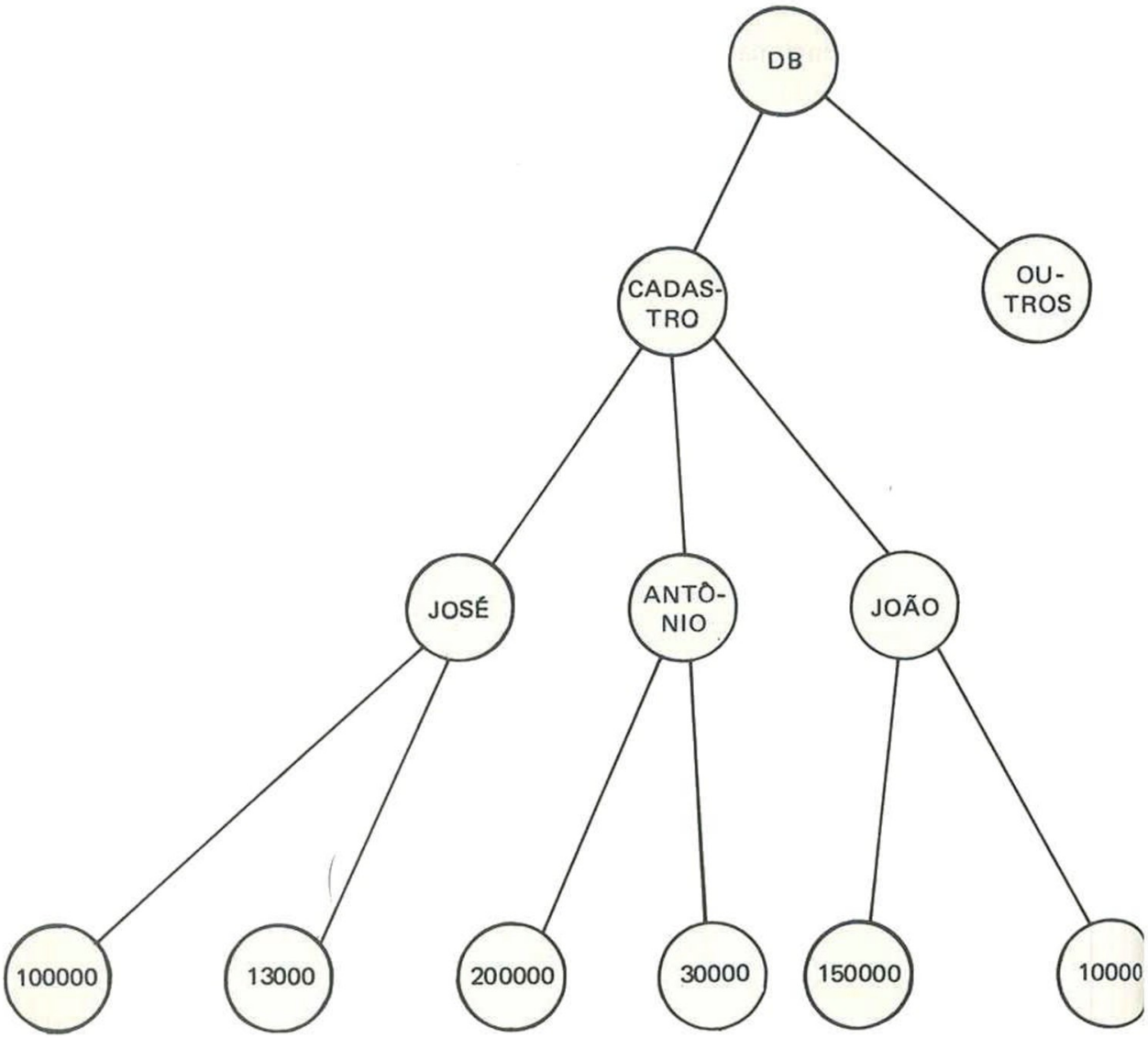


Figura 3 Forma lógica específica.

Especial atenção deve ser dada aos comandos 53Ø, 65Ø e 67Ø. Estes comandos mostram o que já foi dito no capítulo inicial – a característica básica do DB é a possibilidade de expansão, através da capacidade de "concatenação" do BASIC.

A cada novo arquivo, o comando 53Ø agrupa o identificador ▟, o nome do arquivo e o identificador para o primeiro registro do arquivo (▬).

Os dados são agrupados através do comando 65Ø, como no primeiro programa usado.

Ao final de cada conjunto de dados, o comando 67Ø agrega o identificador do próximo registro.

Conforme foi visto, não foi usado o comando DIM do BASIC, como seria de esperar nestes casos.

Como sabemos, o comando DIM cria a matriz. Nos microcomputadores que estamos usando, este comando normalmente é usado para definir pseudo-arquivos, onde o número de ocorrências seria o número de registros e o tamanho (segundo elemento), o tamanho do registro, como no exemplo:

DIM A$ (1ØØ, 8Ø) . . .

define 1ØØ registros de tamanho 8Ø.

As desvantagens das matrizes, arrays ou arranjos, como são chamados, é que fixam um mesmo tamanho tanto para registros como para o arquivo como um todo.

A "concatenação", expandindo uma variável, é bem mais interessante porque permite que dados e arquivos sejam de tamanho variável e sem perda de bytes, gastos com caracteres em branco, como seria no array.

Aproveitando esta primeira rotina que destrinchamos, base das demais, aproveito para indicar o seguinte procedimento: sempre que ficar algum ponto obscuro na apresentação de qualquer uma das rotinas, releia e revise os comandos, até dirimir qualquer que seja a dúvida, por menor que seja, antes de passar para o próximo item.

AS OPÇÕES

Agora que já temos os arquivos criados, vamos formar o *menu* de opções ou de funções, como seria mais correto dizer.

As funções já foram anunciadas no capítulo anterior, vamos apenas escrevê-las no vídeo e prever os desvios para a execução de cada uma delas.

As variáveis usadas são:

K$ – para receber a opção desejada
I – para indicar início de pesquisa do DB
INI – para guardar a posição inicial de um arquivo dentro do DB

Como não há maiores esclarecimentos a serem feitos, vamos aos comandos:

Digite os comandos a seguir, junto com os anteriores.

```
100 CLS
110 SLOW                                          9ØX03
120 PRINT AT 5,9;"OPCOES",,,,
 " FUNCOES PRINCIPAIS"
130 PRINT ,,"1 - CRIAR ARQUIVO",
```

```
   "2 - DELETAR ARQUIVO/REGISTRO",
   "3 - ■SEM FUNCAO DEFINIDA","4 -
LISTAR ARQUIVO"
 140 PRINT ,," FUNCOES AUXILIARES",,,
   "5 - PRINT DO DB",,"6 - MEMORIA DISPONIVEL",
   "7 - SALVAR ZX-
DB","8 - SALARIO LIQUIDO (ESPECIFICA"
 150 LET K$=INKEY$
 160 IF K$<"1" OR K$>"8" THEN GOTO 150
 170 LET I=1
 180 LET INI=I
 190 CLS
 200 GOSUB VAL K$*500
 210 PAUSE 4E4                                    9ØXØ4
 220 GOTO 100
```

Após a digitação, comande GOTO 1ØØ. Surgem então as opções, como mostrado na Fig. 4.

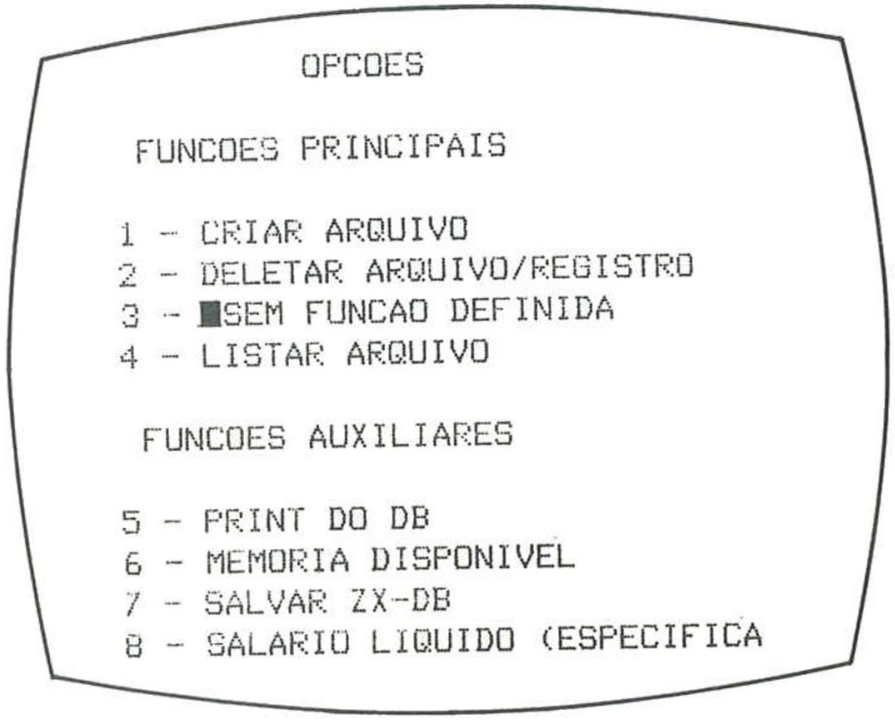


Figura 4 Tela de opções.

Por enquanto só podemos usar a opção "1"; não esqueça, porém, de incluir:

7ØØ RETURN

Observe que, após a execução de cada opção, é necessário tocar em alguma tecla, para que o programa volte ao *menu* de opções. Isto é propiciado pelo comando 21Ø, que cria uma "pausa" teoricamente interminável, isto é, paralisa o programa, após terminar a rotina, até que se pressione alguma tecla.

A FUNÇÃO MAIS SIMPLES

"PRINT DO DB" nada mais é que fixar no programa o que foi indicado que fizesse – PRINT W$

Acrescente os seguintes comandos:

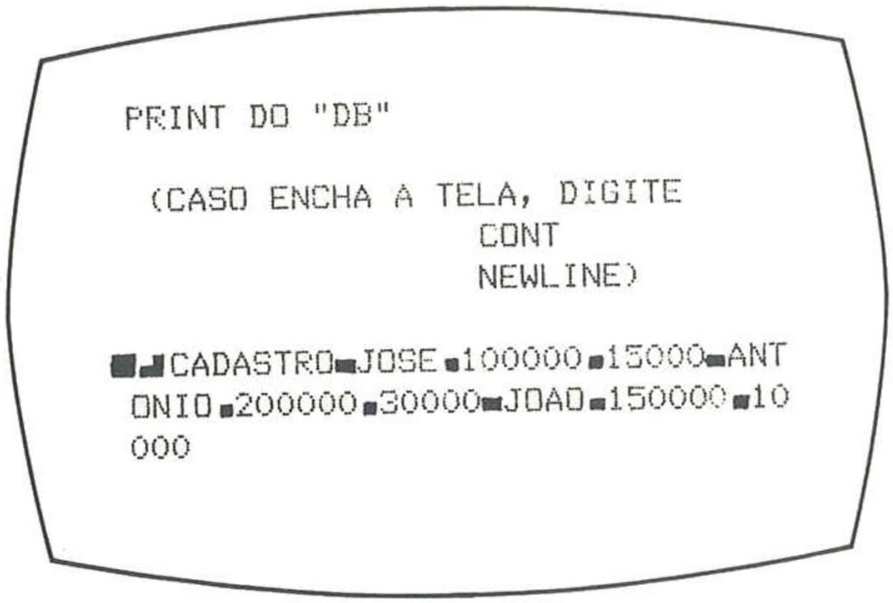
```
2500 PRINT " PRINT DO ""DB"""                    9ØXØ5
2510 PRINT ,," (CASO ENCHA A TELA,
DIGITE",," CONT ",,"NEWLINE)"                    9ØXØ6
2520 PRINT ,,W$
2530 RETURN
```

Com esta, já são duas opções que podem ser solicitadas pela tela de oções.

Esta é uma função auxiliar pois, simplesmente, evita que se fique a todo momento digitando o comando PRINT. Pressione simplesmente "5" e terá então a visão física de todo o DB criado até aqui.

Convém lembrar que se a variável W$, ou seja, o DB, for maior que a capacidade da tela, aparecerá o erro 5 no canto inferior da tela, indicando tela cheia. Neste caso, comande CONT e em seguida NEWLINE.

Se digitar a opção "5", surgirá a seguinte tela:



SALVAR O PROGRAMA ZX-DB

Continuando com as funções auxiliares, vamos implementar o SAVE através da tela de opções.

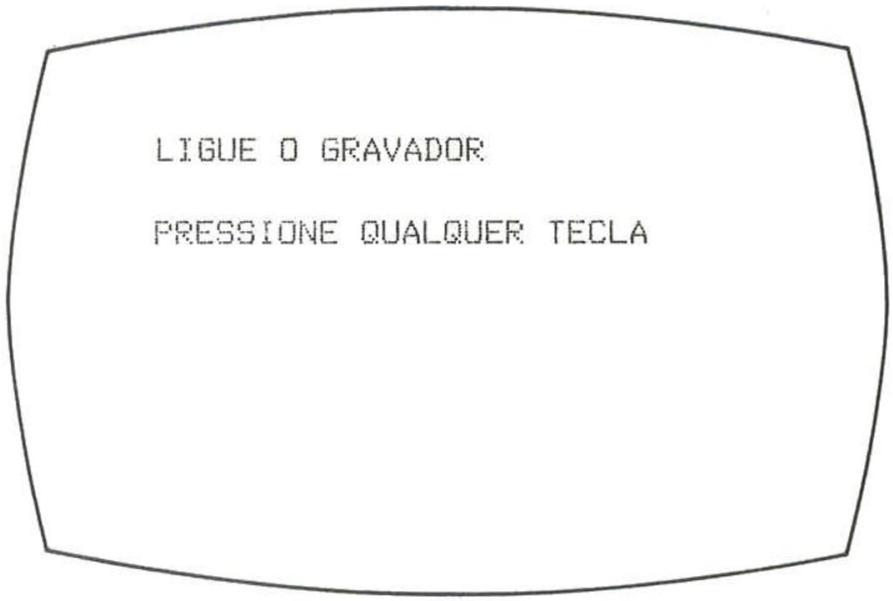
Quando se comanda SAVE no microcomputador, obtém-se uma cópia do que se está fazendo. A isto se dá o nome de "backup", ou seja, uma cópia de segurança, caso falte a luz, por exemplo, ou se, por alguma razão, o micro sair do ar. Por isto, à medida que se atualiza o DB, gravando coisas novas ou alterando, é bom que se faça um "backup" pressionando "7".

Deve-se proceder assim, sempre que se desejar guardar o que foi digitado, para uso futuro.

Acrescente os comandos:

```
3500 PRINT ,,"LIGUE O GRAVADOR",
,,"PRESSIONE QUALQUER TECLA"                    (9ØX07)
3510 PAUSE 4E4
3520 SAVE "ZX-DB"
3530 GOTO 100
```

Se digitar a opção "7", terá a seguinte tela:



A PAUSE do comando 351Ø permite a preparação do gravador antes do comando SAVE propriamente dito.

MEMÓRIA DISPONÍVEL

Concluindo as funções auxiliares, abordamos esta que é importante pelo fato de nos permitir visualizar quanto se tem ainda de memória RAM disponível. Assim, podemos prever algum futuro aumento.

Esta opção nos mostra três informações:

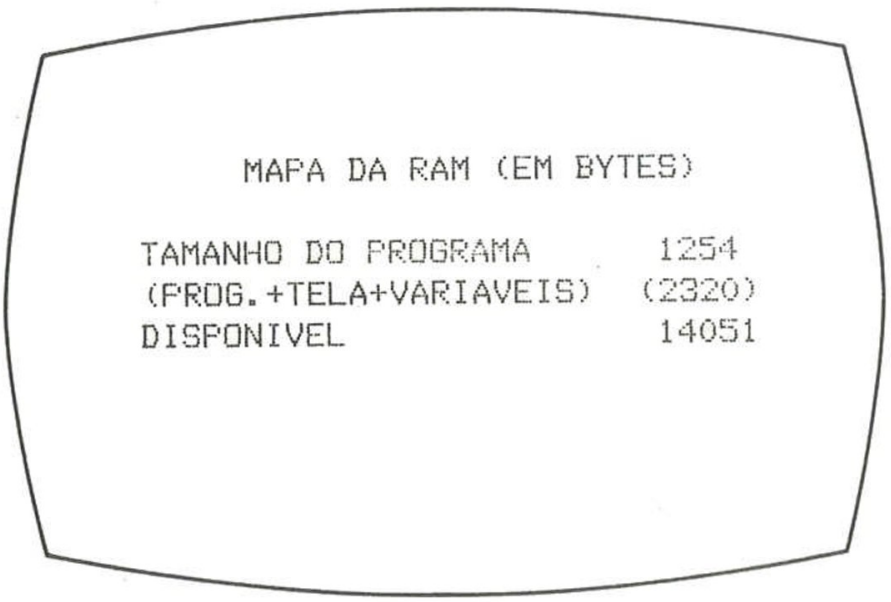
- Tamanho do Programa
- (Programa + Tela + Variáveis)
- Memória Disponível

Como será possível ao leitor acrescentar ou eliminar funções ao programa no final desta leitura, incluímos a primeira informação, "Tamanho do Programa", de modo que o mesmo possa perceber de quanto aumentou ou diminuiu o programa, independente das variáveis e dos bytes ocupados pela tela. A segunda, como está claro, dá o total ocupado até o momento, e a última é que realmente informa os bytes ainda disponíveis a futuras expansões. (9ØX08)

Acrescente:

```
3000 PRINT AT 5,5;"MAPA DA RAM (EM BYTES)"
3010 PRINT ,,"TAMANHO DO PROGRAMA";TAB 25;
PEEK 16396+256*PEEK 16397-16509,
"(PROG.+TELA+VARIAVEIS)";TAB 24;
"(";PEEK 16404+256*PEEK 16405-16384;")",
"DISPONIVEL";TAB 25;(PEEK 16386+256*PEEK 16387
)-(PEEK 16412+256*PEEK 16413)
3020 RETURN
```

Pressionando "6" na tela de opões, deverá surgir a seguinte tela:



OBS. – Os números não correspondem à realidade, são puros exemplos.

Vimos aqui o uso de variáveis do sistema operacional, residente no microcomputador.

NÚMERO DOS BYTES	VARIÁVEL
16384	– início da memória RAM
165Ø9	– início do programa
16396/16397	– início da área de vídeo
164Ø4/164Ø5	– início da linha de edição, isto é, fim de toda memória contínua, ocupada pelo seu programa, vídeo e variáveis.

NÚMERO DOS BYTES	VARIÁVEL
16386/16387	– início da pilha de endereços dos GOSUB's , isto é, final da área de memória RAM utilizada normalmente pelo programa. Após, somente RAMTOP, que no caso não é alterada.
16412/16413	– início da área livre da RAM (Veja Fig. 5)

Então, com estas variáveis, podemos calcular:

Tamanho do Programa – é só subtrair o início do programa (16509) do início da área de vídeo (número contido nos bytes 16396 e 16397), como mostrado no comando 3010;

Programa/Tela/Variáveis – é só subtrair o início da memória RAM (16384) do último byte que seu programa fez uso (número contido nos bytes 16404 e 16405);

Memória Realmente Disponível – subtrai-se o início da área livre da RAM (bytes 16412 e 16413) da última variável da RAM, que é a pilha de endereços dos GOSUB's (bytes 16386 e 16387).

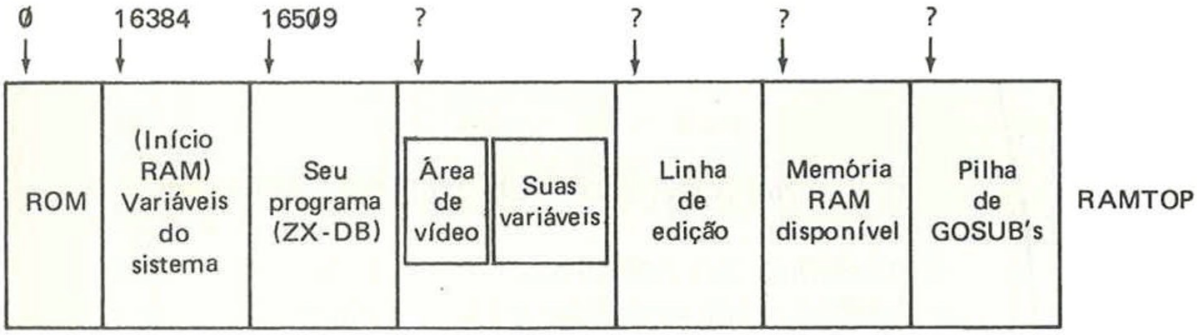


Figura 5 Mapa da memória.

As interrogações (?) da Fig. 5 significam que estes endereços iniciais serão calculados, sempre que se pedir esta opção, segundo o comando 3010.

SUB-ROTINAS

Agora vamos falar sobre as sub-rotinas.

Como você deve saber, o uso de sub-rotinas, através dos comandos GOSUB e RETURN, evita que se repitam desnecessariamente conjuntos de comandos, diminuindo com isto o tamanho final do programa, poupando, conseqüentemente, preciosos bytes da memória RAM.

Elas serão apresentadas em três módulos:

- Diferenciando Arquivo de Registro
- Chegando ao Início dos Arquivos e Registros
- Acesso aos Arquivos e Registros
- Recuperação de Dados

Vá digitando à medida que forem apresentadas.

DIFERENCIANDO ARQUIVOS DE REGISTROS

Para eliminar arquivo ou registro, será necessário chegar a este arquivo ou registro específico, com procedimentos iguais, para ambos. Por isto, usaremos a mesma sub-rotina. A diferença será exercida através das variáveis X$ e Y$.

Quando a pesquisa for de arquivo, as variáveis terão os seguintes conteúdos:

X$ = "▄"
Y$ = "▬"

Quando a pesquisa for de registro:

X$ = "▬"
Y$ = "▄"

Para isto, foram criados os comandos:

```
9500 LET X$="▄"
9510 LET Y$="▬"
9520 RETURN
9530 LET X$="▬"
9540 LET Y$="▄"
9550 RETURN
```

Apesar de serem os últimos, já podem ser digitados.

CHEGANDO AO INÍCIO DO ARQUIVO OU REGISTRO, ACESSÁ-LO

```
9000 REM * SUB-ROTINAS *
9010 REM CHEGAR INICIO ARQ/REG
9020 FOR A=I TO LEN W$
9030 LET R$=W$(A)
9035 IF R$="▄" THEN LET INI=A
9040 IF R$=X$ THEN GOTO 9100
9050 NEXT A
9060 PRINT N$;" NAO EXISTE"
9080 RETURN
9090 REM ACESSAR ARQ/REG
9100 LET D$=""
9110 FOR B=A+1 TO LEN W$
9120 LET R$=W$(B)
9130 IF R$=Y$ OR R$=X$ OR R$="▄" THEN GOTO 9160
9140 LET D$=D$+R$
9150 NEXT B
9160 IF D$=N$ THEN GOTO 9210
9170 LET A=B
9180 LET D$=""
```

```
9190 IF B<LEN W$ THEN GOTO 9150
9195 LET I=B-1
9200 GOTO 9050
9210 LET C=B
9220 RETURN
```

Vamos identificar as duas etapas:

- Chegar ao início dos arquivos/registros
- Acessar os arquivos/registros

A primeira vai dos comandos 9Ø1Ø ao 9Ø8Ø. Consiste em encontrar um identificador de arquivo ou de registro, dependendo do conteúdo da variável X$, conforme mostra o comando 9Ø4Ø. Encontrado o identificador, o programa desvia então para 91ØØ, segunda etapa, onde "recupera" então, na variável D$, o nome do arquivo ou o primeiro dado do registro, que passaremos a chamar de "chave" do registro. Esta segunda parte vai de 9Ø9Ø a 922Ø.

Se o nome, ou chave, não corresponder ao que se procura, o indexador A é atualizado com o valor de B, a variável D$ é esvaziada e, se o final do DB não tiver sido atingido, o programa volta para procurar outro arquivo ou chave, conforme mostram os comandos de 916Ø a 92ØØ.

A variável N$ contém o nome do arquivo ou chave.

Caso o nome tenha sido informado errado ou não exista, o comando 9Ø6Ø indicará. Caso contrário, isto é, o arquivo ou chave seja encontrado e, se a pesquisa for de arquivo, a variável C receberá o valor de B, conforme comando de número 921Ø.

RECUPERANDO DADOS

Completando as sub-rotinas, veremos agora como recuperar os dados individualmente:

```
9300 REM RECUPERAR DADOS
9310 GOSUB 9530
9320 LET B=C
9330 LET P$=""
9340 FOR C=B+1 TO LEN W$
9350 LET R$=W$(C)
9360 IF R$=Y$ OR R$=X$ THEN GOTO 9420
9380 IF R$="■" THEN GOTO 9410
9390 LET P$=P$+R$
9400 NEXT C
9410 LET FIMARQ=1
9420 RETURN
```

Usamos a variável P$ para conter o dado recuperado.

Esta sub-rotina sempre é solicitada após localização do arquivo ou registro. A variável C funciona como o indexador, daí o comando 934Ø pegar uma posição além do conteúdo do indexador B, o qual tem o início do arquivo.

Se um novo dado é iniciado, o programa retoma para onde veio; se for fim do registro, a pesquisa continua.

Se é o fim do arquivo, a variável FIMARQ é preenchida com "1", conforme mostram os comandos de 936Ø a 942Ø.

Agora, estamos aptos a implementar as outras funções.

Já podemos *criar* arquivos, vamos agora *eliminar*: arquivos, pelo seu "nome", e registro pela sua "chave".

Este item não terá teste, pois ele é dependente dos seguintes, isto é, estas subrotinas são solicitadas pelo programa nos próximos itens.

ELIMINANDO ARQUIVOS E REGISTROS

"Deletar" é uma forma aportuguesada do verbo inglês "to delete", que pode ser traduzido como eliminar ou cancelar, daí a função "2 – Deletar Arquivo/Registro".

Para isto, será necessário informar pela variável K$ se vai deletar A para arquivo, ou R para registro. Conforme o caso, pedir nome do arquivo ou chave do registro. Ainda, automaticamente conforme o caso, serão preenchidas as variáveis X$ diferenciando arquivos de registros.

O elemento sendo encontrado, a variável D$ será igual a N$. Na variável A está o início do elemento. A rotina principal busca então o início do próximo elemento. Ele estará em C. Então, o comando 121Ø elimina o pedaço indesejável, como na Fig. 6.

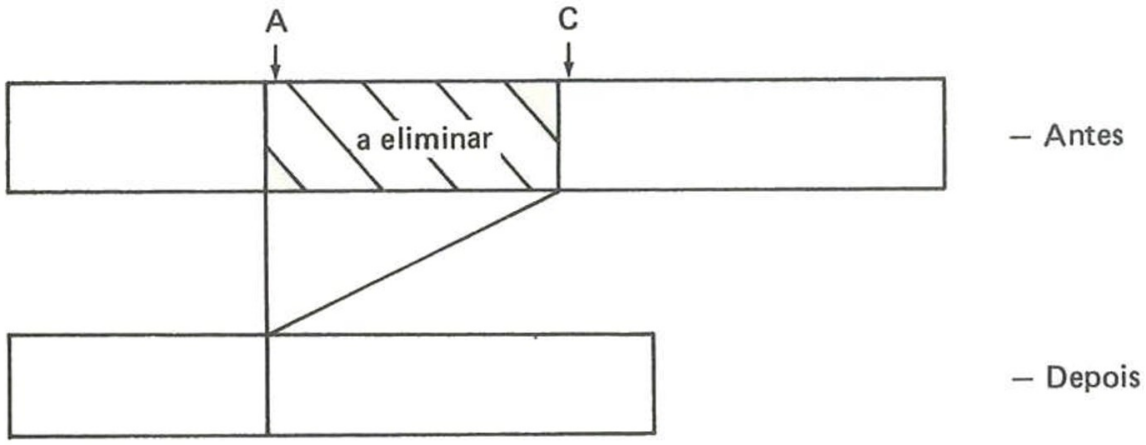


Figura 6 Eliminando arquivo ou registro.

Digite:

```
1000 SLOW                                          (9ØX Ø9)
1010 PRINT " DELETAR..."
1020 PRINT ,,"ARQUIVO ?",,"REGISTRO ?"
1030 IF INKEY$="" THEN GOTO 1030
1040 IF INKEY$="A" THEN GOTO 1070
```

```
1050 IF INKEY$="R" THEN GOTO 1100
1060 GOTO 1030
1070 GOSUB 9500
1080 PRINT "NOME ARQUIVO ?"
1090 GOTO 1120
1100 GOSUB 9530
1110 PRINT "CHAVE REGISTRO ?"
1120 INPUT N$
1130 PRINT N$
1140 FAST                                          (9ØX1Ø)
1150 GOSUB 9000
1160 IF N$<>D$ THEN RETURN
1170 FOR C=B-1 TO LEN W$
1180 LET R$=W$(C)
1190 IF R$=X$ OR R$="▟" THEN GOTO 1210
1200 NEXT C
1210 LET W$=W$( TO A-1)+W$(C TO )
1220 PRINT "DELETADO"
1230 RETURN
```

Com a execução desta opção, surgirá a seguinte tela . . .

Digite: A – para arquivo
R – para registro

Escolhendo "R", teremos a tela seguinte . . .

Digite "JOAO" e terá . . .

A partir deste ponto, a tela ficará toda branca pois estará na modalidade FAST, que permitirá uma velocidade maior de pesquisa (cerca de quatro vezes mais rápida que SLOW). (9ØX11)

O que estará acontecendo será uma pesquisa byte-a-byte do DB, conforme mostram os comandos 9Ø2Ø a 9Ø5Ø, 911Ø a 915Ø e de 932Ø a 938Ø.

Do modo como esta função foi criada, na primeira coincidência, isto é, quando o nome do arquivo ou da chave for encontrado, a pesquisa é encerrada e o elemento é eliminado (arquivo ou registro).

Caso tenha no mesmo arquivo ou em arquivos diferentes a mesma "chave", esta função eliminará o primeiro registro encontrado com esta chave. Este fato pode ser evitado, tendo-se o cuidado, na criação dos arquivos e registros, de se consultar os arquivos existentes, listando-os. Nos grandes computadores isto não ocorre, porque nestes as chaves em caso algum podem ser duplicadas.

O tempo de pesquisa variará de acordo com o tamanho do DB e de onde se encontra o elemento pesquisado. Não se preocupe então, caso a tela fique branca por mais tempo que o normal. Se achar que a pesquisa está demorando demais e que o programa esteja num LOOP interminável, pressione BREAK e depois CONT. No BREAK, poderá observar por que comando estava passando. Um PRINT das variáveis A, B e C, dependendo do comando indicado no BREAK, indicará em que ponto do DB está a pesquisa. Ao final surgirá a tela, conforme folha seguinte.

Caso deseje confirmar a "deleção", comande o PRINT do DB, pressionando a opção "5". O registro . . .

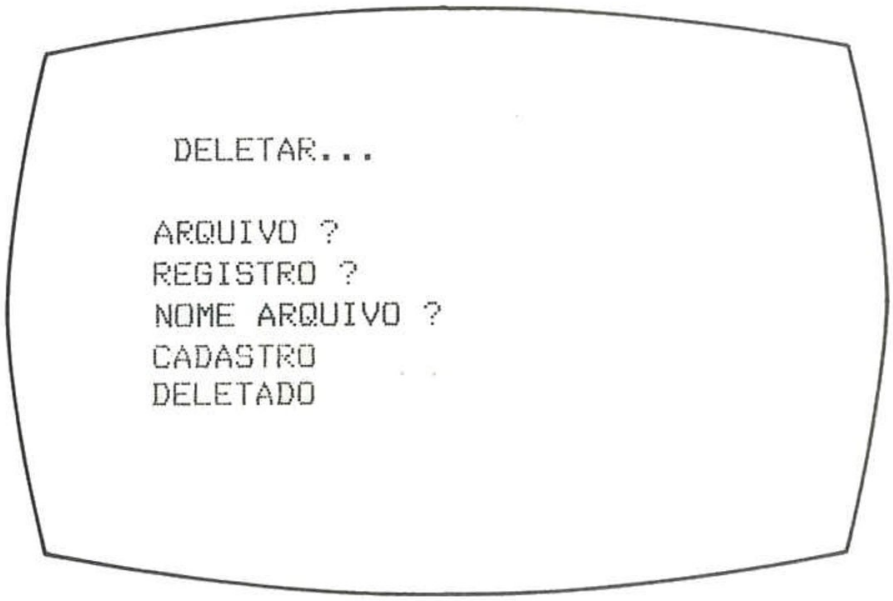
■ JOAO ■15ØØØØ ■1ØØØØ, deverá ter sumido.

Para deletar o arquivo, deverá digitar o nome do arquivo, e a partir daí tudo ocorrerá de modo semelhante.

Pratique! Experimente! Crie arquivos e registros.

Para deletar um arquivo – digite CADASTRO

Deverá surgir . . .



UMA FUNÇÃO BEM ESPECÍFICA

Para usar a sub-rotina que recupera um dado, criamos a função "SALÁRIO LÍQUIDO".

Claro que, para execução desta função, será necessário um arquivo semelhante ao definido no capítulo "CRIAR ARQUIVO", onde temos:

- nome
- proventos
- descontos

Na verdade, ela é apenas um exemplo de como manipular dados separadamente. Tendo-a como exemplo, criam-se várias outras. Ela inclusive poderá ser eliminada, caso não seja de seu interesse, mas observe-a para ver como as fases se processam.

```
4000 LET I=1
4010 PRINT "CALCULANDO SALARIO"
4020 PRINT "  ARQUIVO ?"
4030 INPUT N$
4040 PRINT N$
4050 FAST                                    (9ØX12)
4060 GOSUB 9500
4070 GOSUB 9000
4080 IF D$<>N$ THEN RETURN
4090 GOSUB 9530
4100 PRINT "DIGITE NOME (CHAVE)"
```

```
4110 LET I=INI
4120 INPUT N$
4130 PRINT ,,"NOME","SALARIO"
4140 PRINT N$,
4150 GOSUB 9000
4160 IF D$<>N$ THEN RETURN
4170 GOSUB 9300
4180 LET D$=P$
4190 GOSUB 9300
4200 PRINT VAL D$-VAL P$
4210 SLOW                                          (9ØX13)
4220 IF PEEK 16442<6 THEN CLS                      (9ØX14)
4230 PRINT ,,"OUTRO ? (S/N)"
4240 IF INKEY$="S" THEN GOTO 4100
4250 IF INKEY$="" THEN GOTO 4240
4260 RETURN
```

Variáveis e suas finalidades:

I – inicialmente, conterá o início do DB. Após localizar o arquivo desejado, conterá o início deste.

N$ – inicialmente, o nome do arquivo, depois a chave do registro, isto é, o nome.

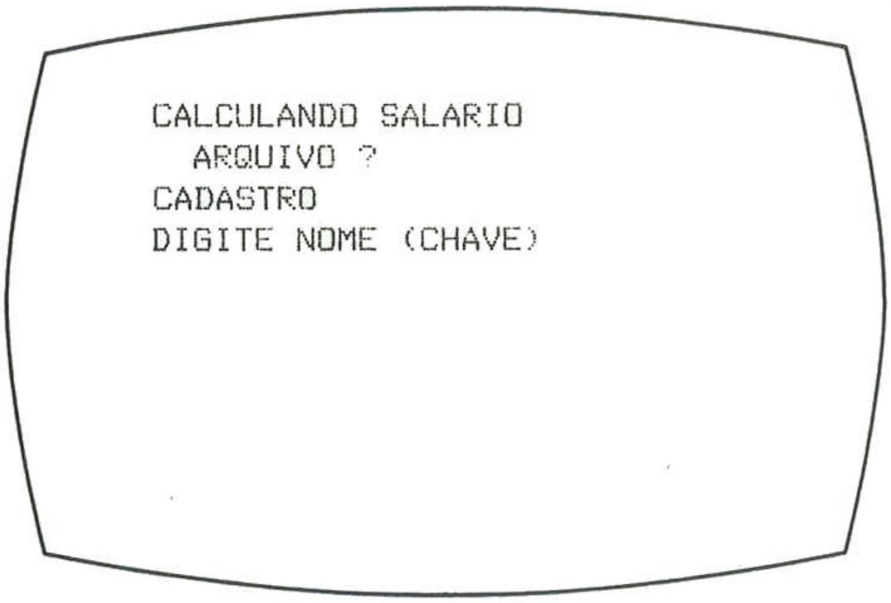
D$ – proventos.

P$ – é usada para conter o dado recuperado até passá-lo para a variável final. Como não há mais que dois dados a recuperar, o segundo fica na própria variável P$ (é o desconto).

Executando a opção "8", a primeira tela a surgir será:

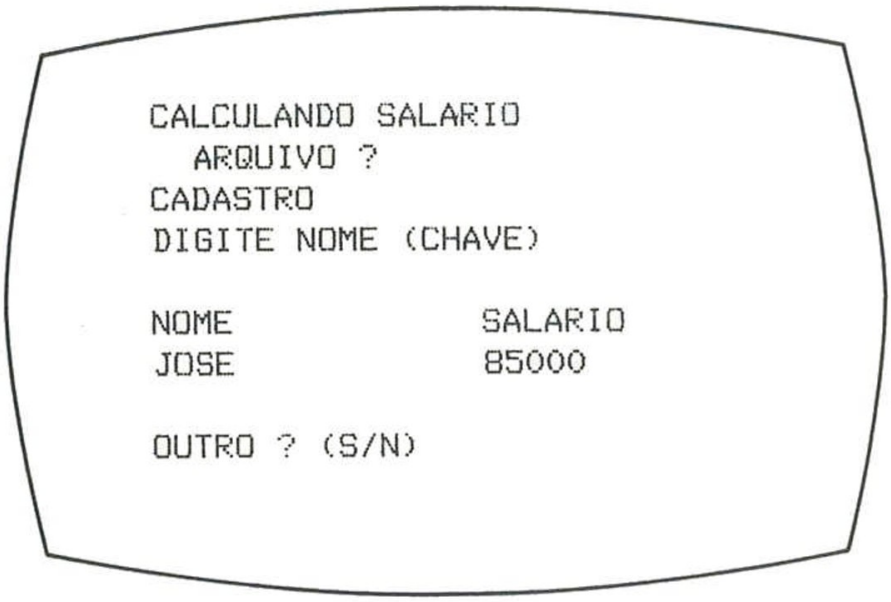
Digite – CADASTRO

A pesquisa começará, a tela ficará branca por um tempo, por causa do FAST e . . . (9ØX15)



Isto, se você criou o arquivo em CRIAR ARQUIVO, se não, exibirá – "NÃO EXISTE".

Para a chave, digite – "ANTONIO" ou "JOSE" (lembre-se de que JOAO foi deletado).



Se digitar "S", será pedido outro nome (chave); pode, inclusive, ser o mesmo. Podem ser solicitados fora de ordem. A variável I cuida para que sempre que se começar nova pesquisa ela contenha o início do arquivo.

Observe que no comando 4Ø6Ø, é executada a sub-rotina que prepara as variáveis X$ e Y$ para recuperar o nome do arquivo. Em seguida, a sub-rotina que recupera arquivo é solicitada no comando 4Ø7Ø.

Na pesquisa da chave, comandos de 4Ø9Ø a 412Ø, é solicitada a sub-rotina que prepara as variáveis X$ e Y$ para recuperar a chave do registro, isto é, o nome sugerido. Em seguida, a sub-rotina que recupera registro é solicitada no comando 415Ø (observe que é a mesma para ARQUIVO e REGISTRO).

Como no comando 4Ø8Ø, o comando 416Ø verifica se o elemento pesquisado foi encontrado. Se não, o comando RETURN faz com que o processamento retome à tela de "opções". Caso contrário, começa a "recuperação dos dados" propriamente dita com GOSUB 93ØØ do comando 417Ø.

Recupera proventos e o coloca em D$. Retorna à sub-rotina. Recupera o segundo dado (descontos) em P$.

O salário líquido é calculado. Será a diferença entre D$ (proventos) menos P$ (descontos). É exibido no vídeo, à frente do nome respectivo.

Neste ponto, é usada mais uma variável do sistema. É a posição 16442 da memória RAM. Ela contém o número de linhas que ainda faltam para completar a tela. Tem início com 23 e chega a zero. Com o comando 422Ø, sempre que a tela estiver para ser completada, antes do erro "5" (tela cheia), ocorre um CLS (limpeza de tela) automático, evitando, desta forma, uma parada indesejada. (9ØX16)

Além do uso desta variável, outro detalhe cabe ser observado nesta parte do programa. É a idéia de "estruturação", que se pode notar quando se solicita seis vezes a participação de sub-rotinas. Isto foi possível graças à fórma em que o programa foi escrito, isto é,

- Um menu de opões
- Quatro rotinas e
- Sete sub-rotinas

A sub-rotina pressupõe sua utilização repetidas vezes, como foi o caso nesta rotina para cálculo do "salário líquido" e outras rotinas, como, por exemplo, "ACESSO AO DADO" é utilizada pelas rotinas de DELETAR, LISTAR e esta que acabamos de ver. Aqui no exemplo, algumas são acessadas direto pelo *menu* de opções, como o SAVE, PRINT do DB e Memória DISPONÍVEL. Há o caso também da rotina "CRIAR ARQUIVOS", que não usa sub-rotina nenhuma (Fig. 7).

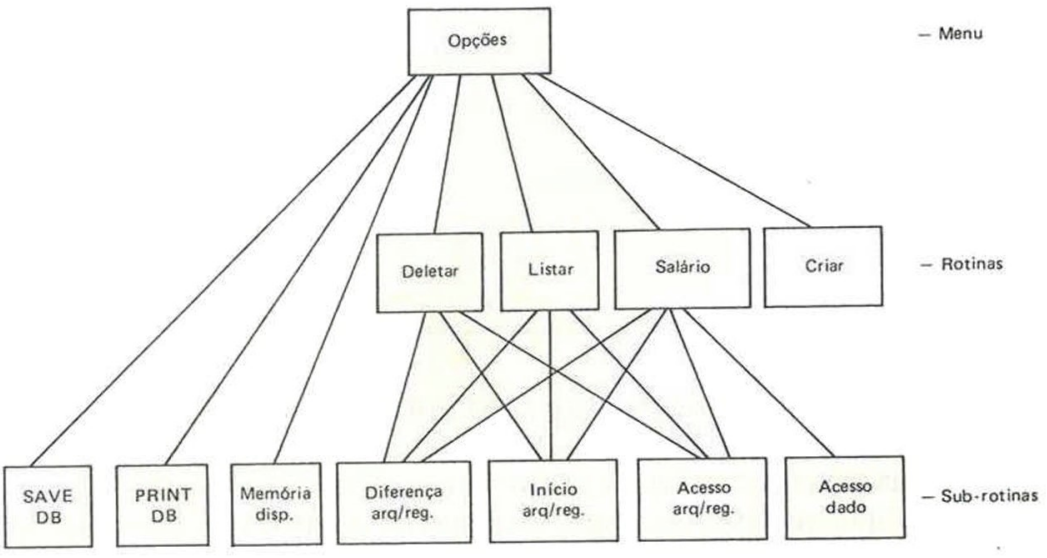


Figura 7 Estrutura geral do programa.

LISTAR ARQUIVO

Esta será a última rotina que veremos juntos e que já faz parte do **ZX-DB**. Vamos então à seqüência de comandos:

```
2000 PRINT "LISTAR ARQUIVO ?"
2010 FAST                                          (9ØX17)
2020 GOSUB 9500
2030 LET FIMARQ=0
2040 INPUT N$
2050 PRINT N$
2060 GOSUB 9000
2070 IF D$<>N$ THEN RETURN
2080 PRINT "DADOS POR REGISTRO ? ";
2090 INPUT N
2100 PRINT N
2110 DIM T$(N,10)
2120 PRINT "TITULOS DOS DADOS ? (TAMANHO 10)"
2130 FOR A=1 TO N
2140 INPUT T$(A)
2150 NEXT A
2160 PRINT
2170 SLOW                                          (9ØX18)
2180 FOR A=1 TO N
2190 GOSUB 9300
2200 IF PEEK 16442<4 THEN SCROLL                   (9ØX19)
2210 PRINT T$(A);"-";P$
2220 IF FIMARQ=1 THEN GOTO 2270
2230 NEXT A
2240 IF PEEK 16442<4 THEN SCROLL                   (9ØX2Ø)
2250 PRINT
2260 GOTO 2180
2270 SCROLL                                        (9ØX21)
2280 PRINT "   FIM"
2290 RETURN
```

As variáveis e suas finalidades:

FIMARQ – conterá 1 se for atingido o final do arquivo. Caso contrário, conterá zero.

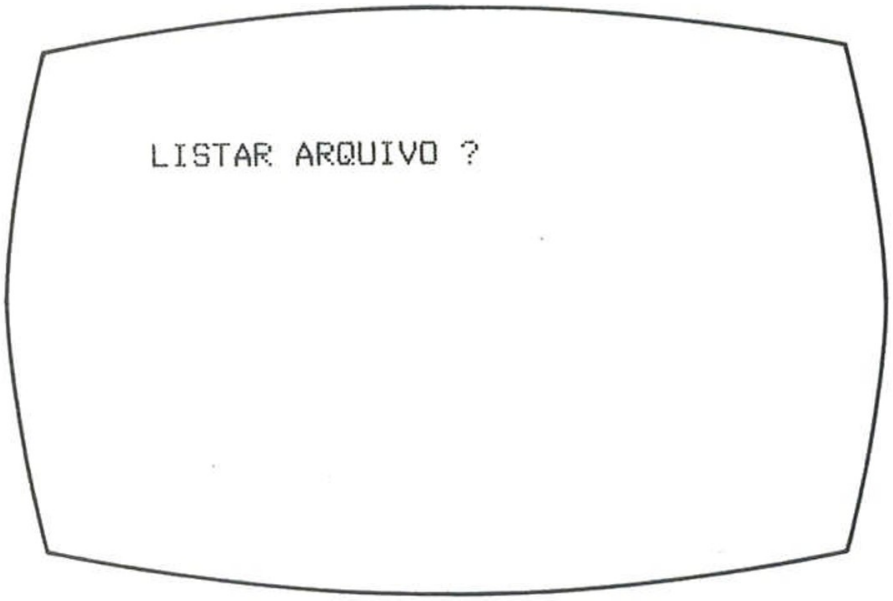
N$ – conterá o nome do arquivo a ser PRINT.

N – conterá o número de dados por registro. Caso você crie o "diretório" sugerido, este dado pode constar do registro que contém o nome do arquivo, não sendo necessário solicitá-lo.

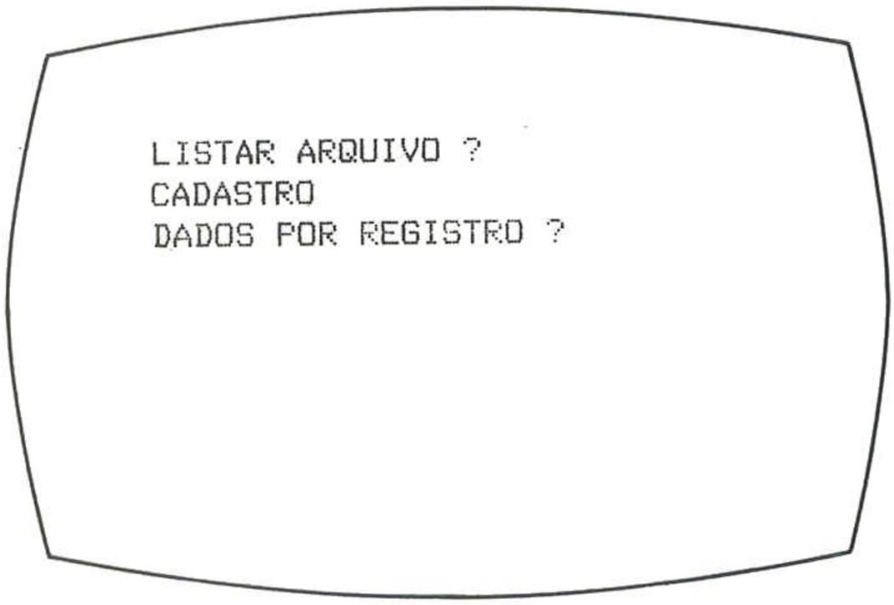
T$ – conterá os nomes dos dados, os quais serão "encontrados" no comando 211Ø e poderão ter tamanho máximo de 1Ø caracteres.
A – indexador para os títulos dos dados.
P$ – conterá os dados recuperados (um por vez).

Mais uma vez, é usada aqui a variável "número da linha do vídeo", posição 16442. (9ØX22)

A execução desta opção mostra:



Digite – CADASTRO e será solicitado:



Digite para o caso – "3"

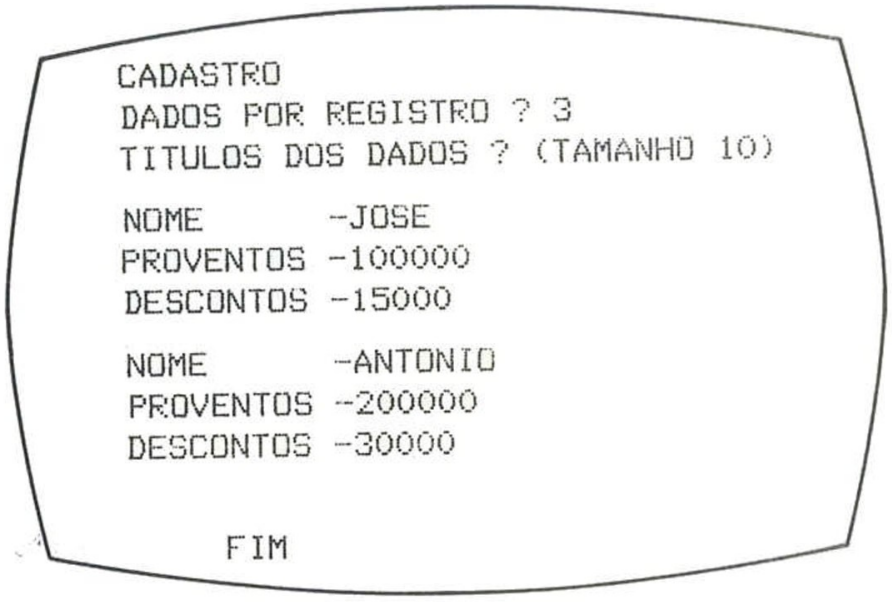
Digite a seqüência – "NOME", "PROVENTOS" e "DESCONTOS". Não serão exibidos, serão guardados nas variáveis T$(1), T$(2) e T$(3), respectivamente.

A seguir, a tela fica em branco e começa a pesquisa. (9ØX23)

Sendo encontrado o arquivo, é iniciada a pesquisa dos dados.

Os dados serão recuperados N-a-N até o final do arquivo, quando então será exibida no vídeo a palavra "FIM". Caso contrário, os títulos dos dados serão colocados no vídeo em situação vertical (um embaixo do outro), de modo a possibilitar qualquer quantidade de dados. Os títulos devem ter no máximo 1Ø caracteres, para permitirem um certo alinhamento. Caso a tela fique cheia, haverá um SCROLL automático, conforme controlado pelos comandos 221Ø e 224Ø. (9ØX24)

Neste caso particular, deve surgir a seguinte listagem no vídeo:



Como observamos, é bem simples.

Crie outros arquivos e liste-os para treinar. Observe que pode listar qualquer dos seus arquivos, bastando que dê o nome do arquivo e o número de dados por registros.

Caso siga outra sugestão dada, imprimir o arquivo em papel; antes do comando 222Ø PRINT T$(A); "–"; P$, inclua um teste para K$, para verificar qual foi a opção. Dependendo desta opção, desvie para um comando que tenha LPRINT no lugar do PRINT. O restante é totalmente igual.

CAPÍTULO 3

PRÁTICA

Vamos verificar se foi conseguido um domínio do **ZX-DB** suficiente para permitir as implementações particulares, mencionadas várias vezes nos capítulos anteriores.

Serão feitas algumas sugestões, como:

- Rotina de alteração de dados;
- Rotina de ordenação de arquivo;
- Inclusão de novos dados em arquivo já existente;
- Exemplo de aplicação usando mais de um arquivo;
- Implementando o Diretório;
- Salvando e Recuperando somente o DB (dados), para quem tem o TK85 ou o TK9ØX.

CRIANDO JUNTOS UMA NOVA FUNÇÃO

Como já sugerimos, deve-se pelo menos tentar criar outras funções. A função "3", por exemplo – "SEM FUNÇÃO DEFINIDA", está aí para você definir.

Sugestão para o nome – "ALTERAR DADOS".

Serão necessários os seguintes passos.

- Peça o nome do arquivo
- Acesse o arquivo
- Peça o nome da chave do registro
  (você poderá alterar, inclusive, a própria chave)
- Acesse o dado
- Altere o dado

Comentários sobre as variáveis:

- "B" conterá a posição de início do dado, isto é, a posição do identificador (do▬, para a chave e do ■, para os dados propriamente ditos),
- "C" conterá a posição do identificador (do ▟ para arquivo, do ▬ para o registro e do ■ do próximo dado)
- Para entrar com o novo dado, use uma variável qualquer já existente, N$ por exemplo.

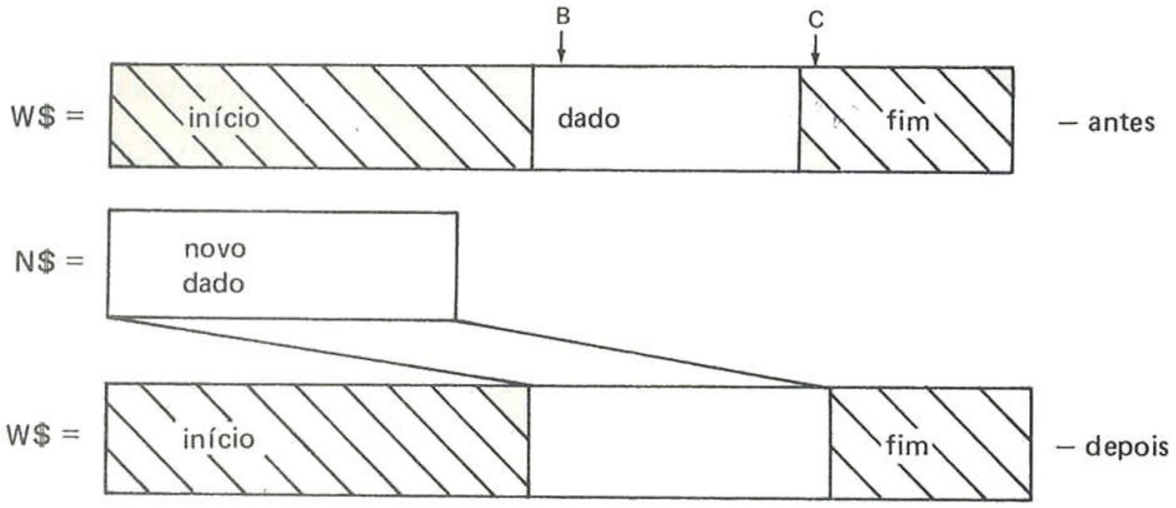


Figura 8 Alteração de registro.

Como sempre vimos fazendo, vamos, através da Fig. 8, observar como as coisas acontecerão no DB, para se alterar um dado.

A opção "3", no momento, estaria simplesmente assim:

```
15ØØ PRINT AT 5,5; "SEM FUNÇÃO DEFINIDA"
151Ø RETURN
```

Estes comandos seriam apenas para evitar a paralisação do programa por erro, quando se pedir a opção.

Vamos fazer juntos, passo a passo, a nova rotina.

Peça o nome do arquivo

Seguindo os modelos das outras rotinas, vamos colocar no vídeo o título da função e pedir o nome do arquivo que contém o dado a ser alterado:

```
1500 PRINT "ALTERANDO DADOS..."
1510 PRINT " QUAL ARQUIVO ?"
1520 INPUT N$
1530 PRINT N$
```

Para que se possa ir testando à medida que inserimos novos comandos, vamos incluir o RETURN com um número maior possível dentro desta rotina.

```
1999 RETURN
```

Já podem ser testados os PRINT's, basta comandar GOTO 1ØØ e solicitar a opção "3".

Não esqueça de alterar o nome da função no *menu* de opções.

Acesse o Arquivo

Vamos acessar o arquivo através das sub-rotinas já existentes:

```
1540 GOSUB 9500
1550 GOSUB 9000
1560 IF N$<>D$ THEN RETURN
```

Neste ponto, já podemos testar a existência do arquivo solicitado (para isto, digite nome de arquivo que não exista).

Tudo OK até aqui? Vamos prosseguir, mas, antes, dê uma verificada no que já fez.

Peça a chave do registro

Vamos pedir a chave e acessar o registro exatamente como é feito em outras rotinas:

```
1570 PRINT,,"QUAL A CHAVE ?"
1580 INPUT N$
1590 PRINT N$
1600 GOSUB 9530
1610 GOSUB 9000
1620 IF N$<>D$ THEN RETURN
```

Como foi feito no arquivo, teste a existência da chave solicitada.

Acesse o dado

Vamos acessar o dado, pelo seu número de localização dentro do registro.

Para ter o controle sobre esta informação (número do dado dentro do registro), você tem duas opções.

1 – documentar externamente, através de anotações em fichas ou qualquer tipo de papel, o nome dos arquivos, contendo, entre outras coisas, o layout do registro, que deverá mostrar a posição de cada dado ou,

2 – usar a opção "4" que lista o arquivo.

A segunda opção é mais cômoda, mas talvez mais demorada, caso você tenha muitos arquivos e careça de absoluta certeza do que deseja. A primeira sempre é muito útil, inclusive tendo outras finalidades.

Procure documentar sempre o que fizer. Só trará benefícios a você mesmo.

Vamos aos comandos:

```
1630 PRINT ,"QUAL DADO DEVE SER ALTERADO ?"
1640 INPUT N
1650 IF N=1 THEN GOTO 1700
1660 FOR A=2 TO N
1670 GOSUB 9300
1680 NEXT A
1690 GOTO 1720
1700 LET P$=D$
1710 LET B=B-(LEN P$+1)
1720 PRINT " ANTES - ";P$
```

Como você observa no comando 165Ø, se o dado a ser alterado é a própria chave ou o dado 1, ele já foi acessado pelo GOSUB 9ØØØ do comando 161Ø e está em D$. Não será necessário pesquisá-lo, bastando desviar para o comando 17ØØ. Aí, então, D$ será colocado em P$, e do indexador B serão subtraídos o tamanho da chave, mais uma posição para o identificador.

Para os demais dados (do segundo em diante), será suficiente repetir o GOSUB 93ØØ (acessar o dado) tantos quantos forem os dados, a partir do segundo, conforme mostra o comando 166Ø.

Verifique se está acessando exatamente o dado que você deseja, executando a rotina e em seguida dando um comando PRINT das variáveis B e C.

Altere o dado

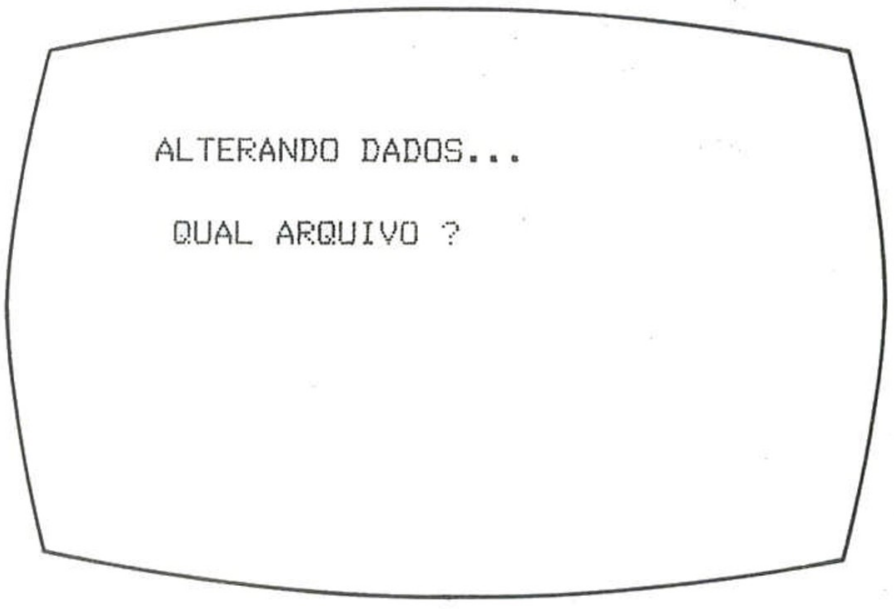
Vamos alterar o dado, conforme mostrado na Fig. 8 e no comando 175Ø.

```
1730 PRINT "DIGITE O NOVO DADO"
1740 INPUT N$
1750 LET W$=W$( TO B)+N$+W$(C TO )
1760 PRINT " DEPOIS - ";N$
```

Que tal? Não foi simples? Isto porque temos o domínio sobre a estrutura do DB e de suas funções básicas.

Execute agora a função "3", completamente, para confirmar sua eficiência (verifique se não esqueceu de alterar o título de função "3" no *menu* de opções).

Primeira Tela



Digite – CADASTRO

```
ALTERANDO DADOS...

QUAL ARQUIVO ?
CADASTRO
QUAL A CHAVE ?
```

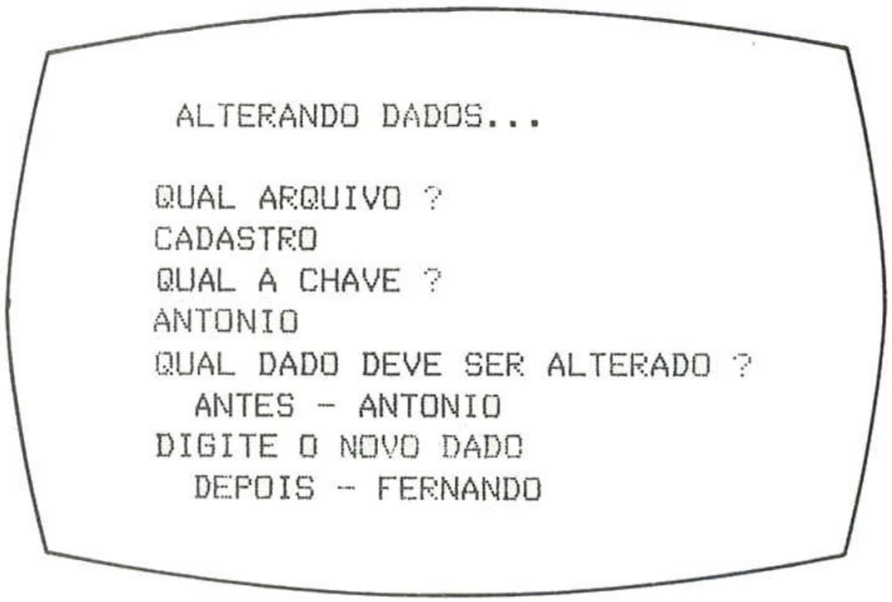
Digite – ANTONIO

```
ALTERANDO DADOS...

QUAL ARQUIVO ?
CADASTRO
QUAL A CHAVE ?
ANTONIO
QUAL DADO DEVE SER ALTERADO ?
```

Digite "1" (para alterar o nome, a própria chave)

```
ALTERANDO DADOS...

QUAL ARQUIVO ?
CADASTRO
QUAL A CHAVE ?
ANTONIO
QUAL DADO DEVE SER ALTERADO ?
  ANTES - ANTONIO
DIGITE O NOVO DADO
```

Digite – FERNANDO



Pronto! Está alterado.

Execute agora a função "5" (PRINT DO DB), para confirmar a alteração. Veja se "ANTONIO . . ." sumiu e se em seu lugar aparece "FERNANDO . . ."

Tudo certo? Caso haja algo errado, verifique os comandos, corrija e tente novamente.

Como esta função não está presente no **ZX-DB**, salve esta versão do programa com a função "7" (SALVAR O **ZX-DB**).

Caso venha a implementar o "diretório", após a execução desta rotina, será necessário atualizá-lo, pois o dado pode ter alterado o tamanho do arquivo e, conseqüentemente, a partir dele, todos os indicadores de início e fim dos arquivos estarão errados. Isto é apenas um lembrete, deixe a preocupação para depois.

Agora faça um sozinho.

CRIANDO SOZINHO UMA NOVA FUNÇÃO

Daremos apenas uma mãozinha.

Sugerimos – "ORDENAR CRESCENTEMENTE O ARQUIVO PELA CHAVE", como função "9".

Deverão ser observados os seguintes passos:

– Solicitação,
– Preparação,
– Ordenação propriamente dita e
– Retorno do arquivo.

Detalhando os passos:

Solicitação

Inclua no *menu* de opções o título e a possibilidade do número "9" como entrada (como está só aceita números de 1 a 8).

Preparação

Comece a rotina no número 45ØØ, pois VAL de "9" x 5ØØ é igual a 45ØØ.

Limpe a tela.

Peça o nome do arquivo.

Pesquise o arquivo. Encontrando o nome desejado, vá até o final do mesmo (até o próximo ▟). Use a sub-rotina de recuperação de dados, (93ØØ). Use dois FOR, um para os registros (QR) e outro para os dados (N).

Para esta pesquisa, crie algumas variáveis como indicado a seguir:

- Guarde em TR o tamanho do maior registro, inclusive os identificadores.
- Em TC, guarde o tamanho da *maior* chave, substituindo o valor sempre que encontrar um tamanho de chave maior (lembre-se de que o tamanho das chaves e dos registros são variáveis).
- A cada ▬, conte o número de registros e guarde na variável QR.
- Lembre-se de que *A* contém o início do arquivo e *B* tem o início do primeiro dado (chave).

Defina uma matriz para colocar temporariamente o arquivo:

DIM M$ (QR, TR)

M$ será um arquivo de trabalho.

Volte a pesquisar, desde o início, o arquivo dos dados, isto é, a partir de *B*, e vá preenchendo a matriz, tendo o cuidado de colocar a chave alinhada no início e mantendo os identificadores dos elementos (dados ■ e registros ▬), simplesmente para não ter que reincluir-se depois; como *B* será alterado em 93ØØ, salve seu conteúdo em *INI* (variável já existente).

Use novamente a sub-rotina de recuperação de dados (93ØØ).

Lembre-se de que, lá, *C* é igualado a *B* (LET B = C). Para incluir os identificadores, será necessário subtrair uma unidade de *B*, e conseqüentemente de *C* (vide 93ØØ). Como 93ØØ pára no primeiro "▬" encontrado e o primeiro carácter do registro é um "▬", acrescente uma unidade ao número de dados informado (N = N + 1) e elimine uma unidade de *C* (C = C – 1), a cada volta de 93ØØ.

Novamente use dois FOR, um para os registros e outro para os dados.

Use uma variável qualquer; P$, por exemplo, para salvar temporariamente a chave e D$ para os demais dados.

Os dados devem ir sendo agrupados até o próximo identificador de registro.

LET M$ (N, TO TC) = P$ → para a chave

LET D$ = D$ + "■" + P$ } para os demais dados

LET M$ (N, TC + 1) = D$ }

"N" pode ser qualquer indexador já usado ou não, diferente de A, B e C. Ele será o indexador de registro.

Graficamente, a preparação será como mostra a Fig. 9.

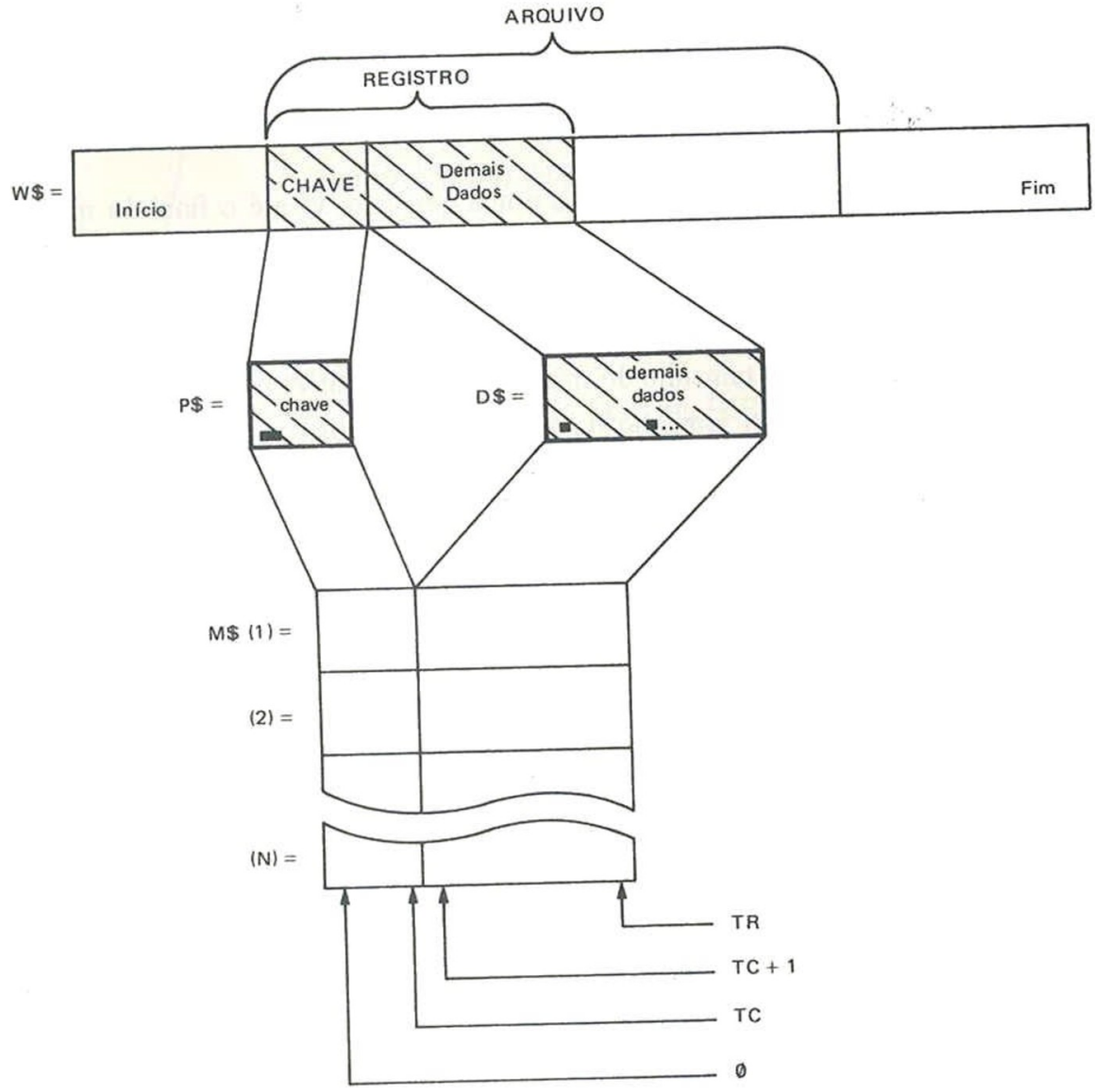


Figura 9 Preparando arquivo para ordenação.

Ordenação

Agora pode ser feita a ordenação propriamente dita, isto é, comparar dados 2-a-2 e trocar de posição caso "dados de nº E + 1" sejam menores que "dados de nº E".

Esta técnica é chamada de "Bubble-Sort". O nome sugere que os dados menores, como bolhas mais leves, subam mais que os maiores. O fluxo da Fig. 1Ø e o exemplo de passos a seguir mostram como isto ocorre.

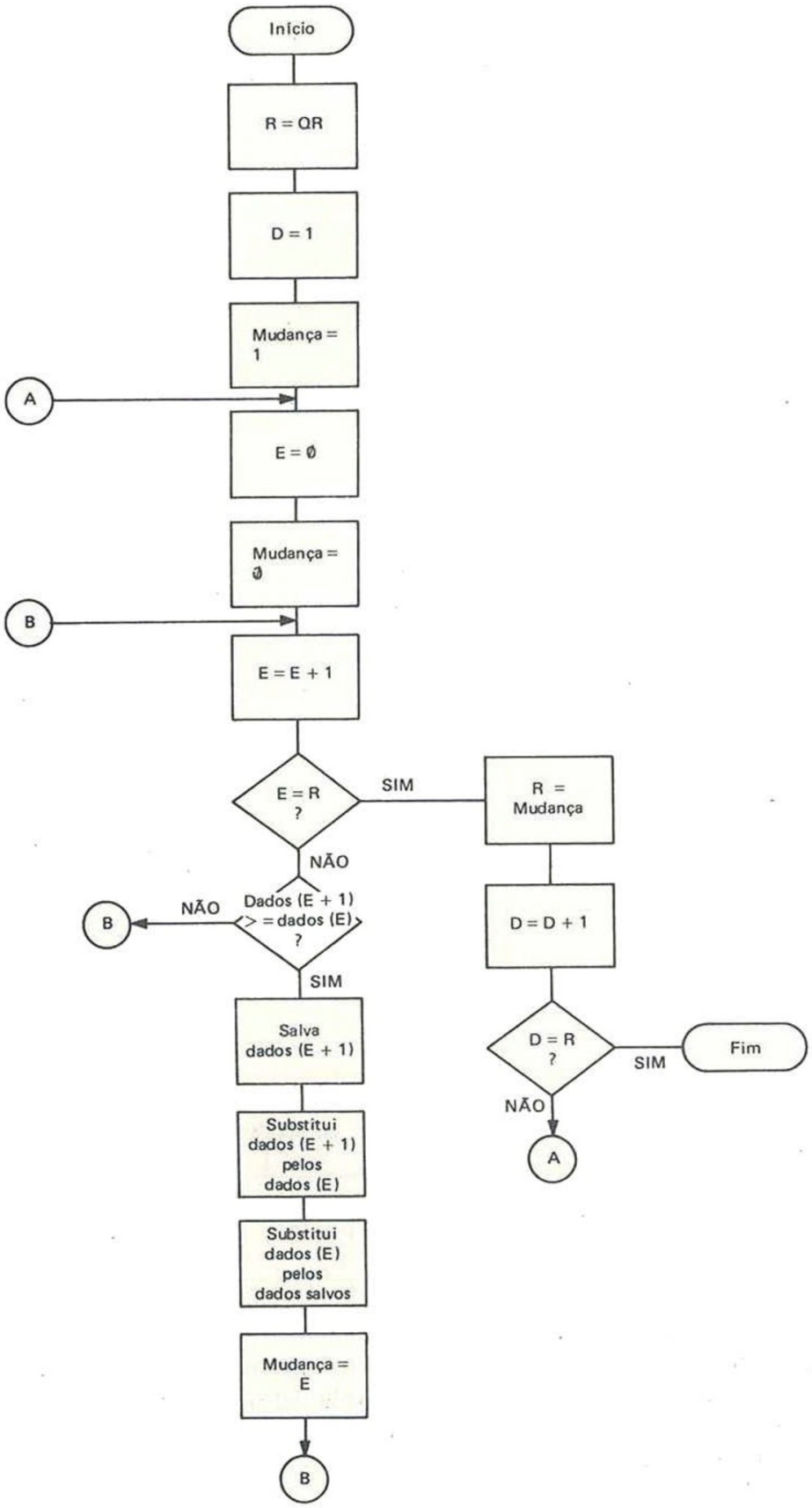
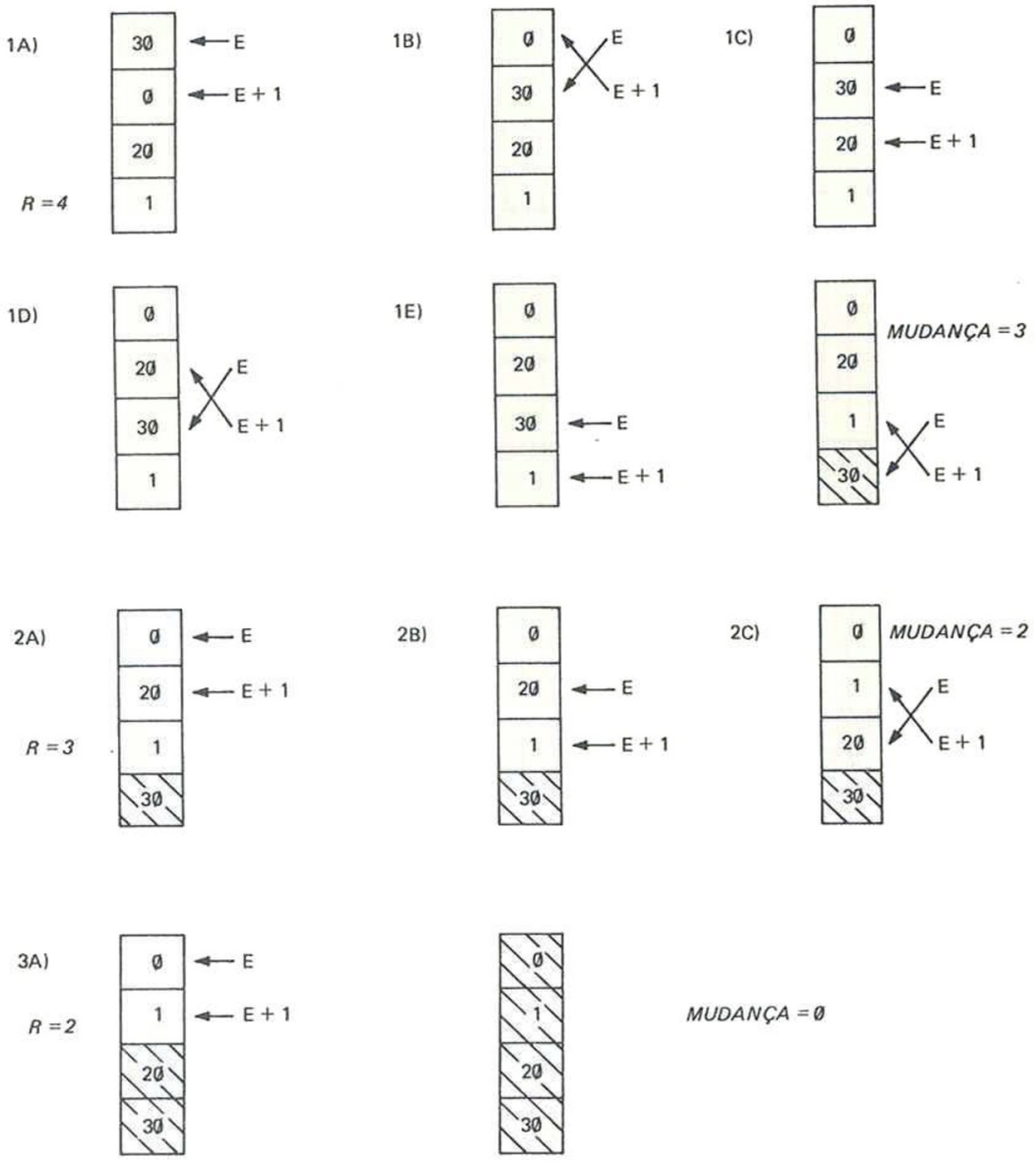


Figura 10 Bubble-Sort.

PASSOS DO SORT

Se considerarmos os dados 3Ø, Ø, 2Ø e 1, teremos as seguintes passagens:



O aspecto interessante deste *SORT* é que na passagem em que MUDANÇA for igual a zero significa que todos os dados estão ordenados. Isto pode acontecer numa das passagens intermediárias, sem exigir que necessariamente seja feito um número de passagens para completar o *FOR*. Esta é a Bubble-Sort otimizada.

A codificação ficaria como se segue:

```
LET R = QR
LET MUDANÇA = 1
```

```
        FOR D = 1 TO R
        LET MUDANÇA = ∅
        FOR E = 1 TO R – 1
        IF M$ (E + 1, TO TC) > = M$ (E, TO TC
        ) THEN GOTO NÃO
        LET D$ = M$ (E + 1)
        LET M$ (E + 1) = M$ (E)
        LET M$ (E) = D$
        LET MUDANÇA = E
NÃO     NEXT E
        LET R = MUDANÇA
        NEXT D
```

RETORNO DO ARQUIVO

Agora que a matriz M$ está ordenada de modo crescente de chave, o arquivo pode ser recolocado em seu lugar no DB.

Podem ser usados dois caminhos:

1 – Simplesmente incluir registro-a-registro com espaços extras das chaves menores e os da parte final do registro.

2 – Incluir registro-a-registro, eliminando os espaços. Graficamente, a primeira opção fica como mostra a Fig. 11.

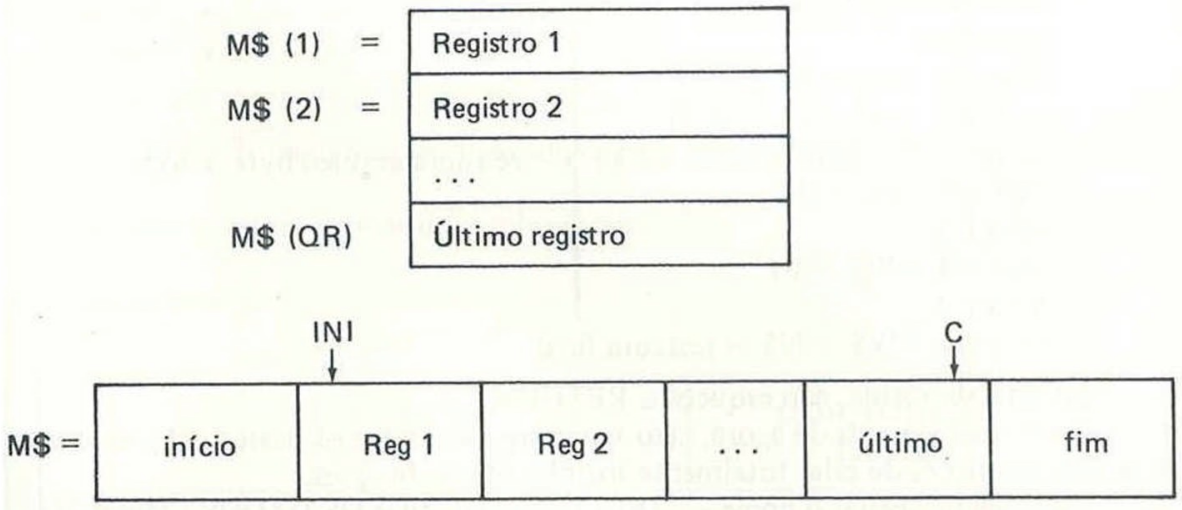


Figura 11 Retornando arquivo sem eliminar espaços.

Observar que, na "mãozinha" anterior, foram usados indexadores diferentes de *INI* e *C*, para não se perder o início (INI) e o fim (C) do arquivo.

Mais uma mãozinha . . .

LET N$ = W$ (C + 1 TO) – primeiro, salva final em N$

LET W$ = W$ (TO INI – 1) – depois, salva em W$ só o início.

Esta ordem deve ser observada para não se perder o final.

```
FOR D = 1 TO QR
LET W$ = W$ + M$ (D)   } restaura arquivo
NEXT D                 } registro-a-registro
LET W$ = W$ + N$ → restaura final
```

Na segunda opção, será necessário usar um passo intermediário onde se recrie os registros sem espaços, como mostra a Fig. 12.

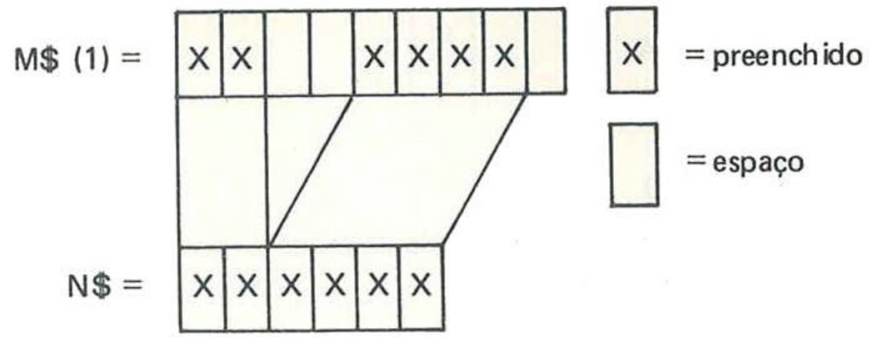


Figura 12 Retornando arquivo eliminando espaços.

Para isto, é necessário pesquisar byte-a-byte; caso seja espaço, não agrupá-lo. A codificação poderá ser a que se segue:

```
        FOR A = 1 TO QR
        LET P$ = " "
        FOR Z = 1 TO TR
        LET R$ = M$ (A, Z TO Z)
        IF R$ = " " THEN GOTO NEXT  } – restaura arquivo byte-a-byte
        LET P$ = P$ + R$
NEXT    NEXT Z
        LET W$ = W$ + P$
        NEXT A
        LET W$ = W$ + N$ → restaura final
```

Ao final da rotina, não esqueça o RETURN.

Creio que, a partir de agora, caso tenha seguido todos os passos indicados, você já tem condições de criar totalmente sozinho outras funções.

Vou sugerir apenas o nome – "INCLUSÃO DE NOVOS DADOS EM ARQUIVO JÁ EXISTENTE".

Não se esqueça de ver os passos necessários:

- Enumere os passos
- Detalhe um pouco mais
- Codifique
- Sempre que possível, represente antes graficamente o que você deseja. Procure com o desenho figurar o que quer que aconteça com o DB, a cada passo.
- E mãos à obra!

UTILIDADE PRÁTICA DO ZX-DB

Aqui vão algumas sugestões para você criar suas próprias funções específicas do tipo da "Calculando Salário Líquido".

Usando mais de um arquivo

Quando se trabalha com DB, é muito comum o relacionamento entre arquivos e entre dados. Dados, inclusive, de arquivos diferentes. Esta é uma das grandes utilidades do DB. Esta utilização de mais de um arquivo deve ser bem entendida para que a tarefa se torne fácil. Os arquivos estão todos ali. Muito próximos. Juntos mesmo.

Um Sistema Gerenciador de Banco de Dados (SGBD), de grande porte, prevê o uso de diversas linguagens de programação e adaptações.

Aqui, em termos de linguagem, podemos contar com o BASIC e o ASSEMBLY.

Normalmente, em grandes computadores, são confeccionados diversos programas para as aplicações que acessam o DB. No nosso caso, os programas são as funções ou opções, ou ainda, as rotinas: CRIAR, DELETAR, LISTAR, ORDENAR, ALTERAR etc. e outras mais que porventura venha a criar.

A nossa limitação aqui será a de sempre – o tamanho da memória. Teremos que arranjar arquivos e programas no mesmo espaço, sejam quantos forem, exceção apenas para o TK85 e para o TK9ØX. Mas para o nosso objetivo didático, tudo está sob controle. Vamos então apresentar uma idéia que poderá ser usada na prática, na qual veremos como utilizar mais de um arquivo em uma mesma função. Seria o equivalente a se dizer "um programa que utiliza mais de um arquivo".

Vamos ao exemplo!

1º) *Controle de Compras Mensais de Alimentos*

Os arquivos podem ser visualizados como tabelas. Veja Figs. 13, 14, 15, 16, 17 e 18.

Vamos ver duas formas de combinações:

A primeira forma:

PRODUTO		QUANTIDADE EM KG
CÓDIGO	NOME	
Ø1Ø	ARROZ	1Ø
Ø15	FEIJÃO	5Ø
Ø52	SAL	1
1Ø9	CEBOLA	2
371	GORDURA	5

Figura 13 Relação de compras com quantidades fixas.

CÓDIGO	PREÇO POR KG
Ø1Ø	9ØØ
Ø15	12ØØ
Ø2Ø	5ØØ
Ø25	3ØØ
Ø52	4ØØ
1Ø9	1ØØØ
2ØØ	888
371	781

Figura 14 Preços de janeiro.

CÓDIGO	PREÇO POR KG
Ø1Ø	95Ø
Ø15	15ØØ
Ø25	7ØØ
Ø52	41Ø
1Ø9	85Ø
371	8ØØ

Figura 15 Preços de fevereiro.

Estes arquivos podem ser criados normalmente pela função "1 – CRIAR ARQUIVOS".

– A Primeira combinação

Em princípio, os arquivos são independentes e individuais. Podem ser criados, alterados, ordenados, listados etc.

Para podermos ter um relacionamento entre eles, é necessário que nos arquivos envolvidos exista pelo menos um dado que seja equivalente. No caso exemplificado, este dado vem a ser o CÓDIGO DO PRODUTO.

Com o relacionamento destes arquivos, pode-se, por exemplo: calcular o "Valor Total do Estoque"; saber quais os dados que estão com "Estoque Zero" (a cada utilização do produto, deverá ser feita uma alteração na quantidade); "Diferença das Compras" do mês *X* para as do mês *Y* etc. . . .

O código será a "chave". Os arquivos deverão ser pesquisados pela mesma chave, e os dados que virem a interessar deverão ser guardados em variáveis independentes, as quais, juntas, formarão o registro lógico. Aqui, por exemplo, teremos:

Arroz para janeiro

A$ = Ø10 ——————— código
B$ = arroz ——————— nome
C$ = Ø1Ø ——————— quantidade ou quilos

Dados extraídos do primeiro arquivo (Fig. 12).

D$ = 9ØØ ——————————— valor de um quilo

Dados extraídos do segundo arquivo (Fig. 13).
Para o valor do estoque de arroz, bastaria multiplicar C$ por D$.

E$ = C$ * D$ ——————————— total da compra de janeiro

Arroz para fevereiro

F$ = 95Ø ——————————— valor de um quilo
G$ = C$ * F$ ——————————— total da compra de fevereiro

Para a diferença das duas compras, basta subtrair E$ de G$, isto é, compra total de fevereiro menos compra total de janeiro.

H$ = G$ – E$ ——————————— diferença

O registro lógico completo ficaria:

A$ = ØlØ ——————————— código do produto
B$ = ARROZ ——————————— nome do produto
C$ = ØlØ ——————————— quantidade do produto em quilos
D$ = 9ØØ ——————————— valor de um quilo em janeiro
E$ = 9ØØØ ——————————— valor total da compra de janeiro
F$ = 95Ø ——————————— valor de um quilo em fevereiro
G$ = 95ØØ ——————————— valor total da compra em fevereiro
H$ = 5ØØ ——————————— diferença valor total das compras

Isto pode ser expandido para todos os produtos, e no final ter-se a diferença total das compras.

Podem também ser apurados os produtos que não foram comprados em fevereiro, o consumo médio etc. . . .

Normalmente, estes tipos de informações são apuradas ou geradas, para que juntas formem um determinado Relatório.

– A Segunda combinação:

Outra disposição dos mesmos dados:

CÓDIGO	NOME
ØlØ	ARROZ
Øll	FEIJÃO
Øl3	SAL
Øl5	CEBOLA
Ø2Ø	GORDURA
Ø25	MACARRÃO
Ø3Ø	PEIXE
Ø52	CARNE
lØ9	LINGÜIÇA
2ØØ	OVOS
371	FARINHA

Figura 16 Produtos.

CÓDIGO	QUANTIDADE	PREÇO
Ø1Ø	1Ø	9ØØ
Ø13	5	12ØØ
Ø2Ø	3	8ØØ
Ø3Ø	4	2ØØØ
1Ø9	1Ø	32ØØ
2ØØ	5	12ØØ

Figura 17 Compras de janeiro.

CÓDIGO	QUANTIDADE	PREÇO
Ø1Ø	5	91Ø
Ø11	3	8ØØ
Ø2Ø	1Ø	11ØØ
Ø25	4	5ØØ
Ø3Ø	7	25ØØ
Ø52	1Ø	5ØØØ
1Ø9	9	12ØØ
2ØØ	1	14ØØ
371	2	6ØØ

Figura 18 Compras de fevereiro.

Um arquivo (Fig. 16), com apenas os códigos e nomes dos produtos, pode ser ampliado à medida que surjam novas necessidades. Por exemplo, um arquivo de compras (Figs. 17 e 18) para cada mês, contendo os códigos, quantidades e preços dos produtos comprados. Com estes, pode-se apurar o "Consumo e o custo médio mensal"; "Consumo total por produto até o último mês", fornecendo-se para a função apenas o nome ou o código do produto.

Dois caminhos que podem ser seguidos:

1 – Reunindo os arquivos por programa
2 – Reunindo os arquivos externamente (redigitando)

Reunindo os arquivos por programa

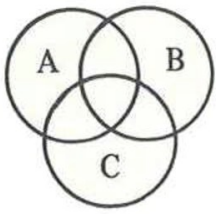
Antes de mais nada, precisamos determinar claramente quais informações finais desejamos obter.

Vamos escolher – "*Consumo médio* e *preço médio* por produto até o mês..."

Como sempre existem vários modos de se obter o mesmo resultado, o que vamos indicar é apenas uma opção:

– Criar por programa um único arquivo intermediário com os dados dos vários arquivos envolvidos, eliminando-se, o máximo possível, os dados redundantes.

Em matemática, ao arquivo resultante dá-se o nome de conjunto união.



A = {1, 2, 3, 4, 5}
B = {1, 5, 6}
C = {1, 6, 7}
resultante = {1, 2, 3, 4, 5, 6, 7}

– O arquivo intermediário terá então o seguinte layout para o registro (vide Fig. 19).

NOME/MÊS	QUANTIDADE	PREÇO

– Será criado um novo dado NOME/MÊS, unindo-se o nome do produto com os dois algarismos respectivos ao mês das compras. Com isto, será criada mais de uma chave diferente e conseqüentemente mais de um registro, isto porque são *mais* de um registro realmente diferentes com quantidades e preços diferentes.
O código do produto no arquivo resultante perde sua função.
– O arquivo intermediário resultante, após sua utilização (impressão, consulta etc.), deverá ser "deletado", pois também perde sua finalidade, visto que pode ser recriado a qualquer momento.
– A seguir será criado o arquivo final, com o seguinte layout do registro.

NOME	QUANTIDADE MÉDIA	PREÇO MÉDIO

PASSOS

– Isolar arquivo de produtos num arquivo matriz (DIM) auxiliar, como feito na ordenação, com a vantagem de a chave ter tamanho fixo.
Como a função aqui é bem específica, não há necessidade da solicitação dos nomes dos arquivos, visto que os mesmos também são bem específicos.
– Criar arquivo final.
– Os arquivos intermediários resultantes poderão ser agregados ao final do DB, caso se deseje listá-los, por exemplo, antes de deletá-los.

Reunindo os arquivos externamente

A necessidade de utilizar dois arquivos, como já foi dito, quase sempre pode ser substituída pela recriação do arquivo, incluindo nesta recriação todos os dados que venham a interessar. Por exemplo: no caso de ter-se um arquivo com CÓDIGO e NOME DO PRODUTO (Fig. 16), outros com CÓDIGO, QUANTIDADE COMPRA-

DA e PREÇO DO PRODUTO para cada mês (Figs. 17 e 18), pode-se recriar um único eliminando-se o código do produto e juntando-se ao NOME dois algarismos para indicar o mês de compra. Sendo este dado a chave, com a ordenação teremos todos os nomes juntos e em ordem de mês, isto é, o registro de janeiro na frente do registro de fevereiro e assim sucessivamente até dezembro. A cada mês pode ser incluído o registro do mês e em seguida ordenar (Fig. 19).

Em vez de usar o nome, pode-se usar o código que é bem menor. No caso de necessidade do nome, pesquisa-se o arquivo de CÓDIGO e NOME. Isto se torna bem interessante quando se têm nomes muito grandes. No caso em questão, os nomes são bem pequenos e podemos tolerar suas repetições nos vários registros.

Por isto, é importante que antes de se criar o arquivo se pense bem na utilização que se fará dele, deixando para usar mais de um arquivo quando não houver outro jeito.

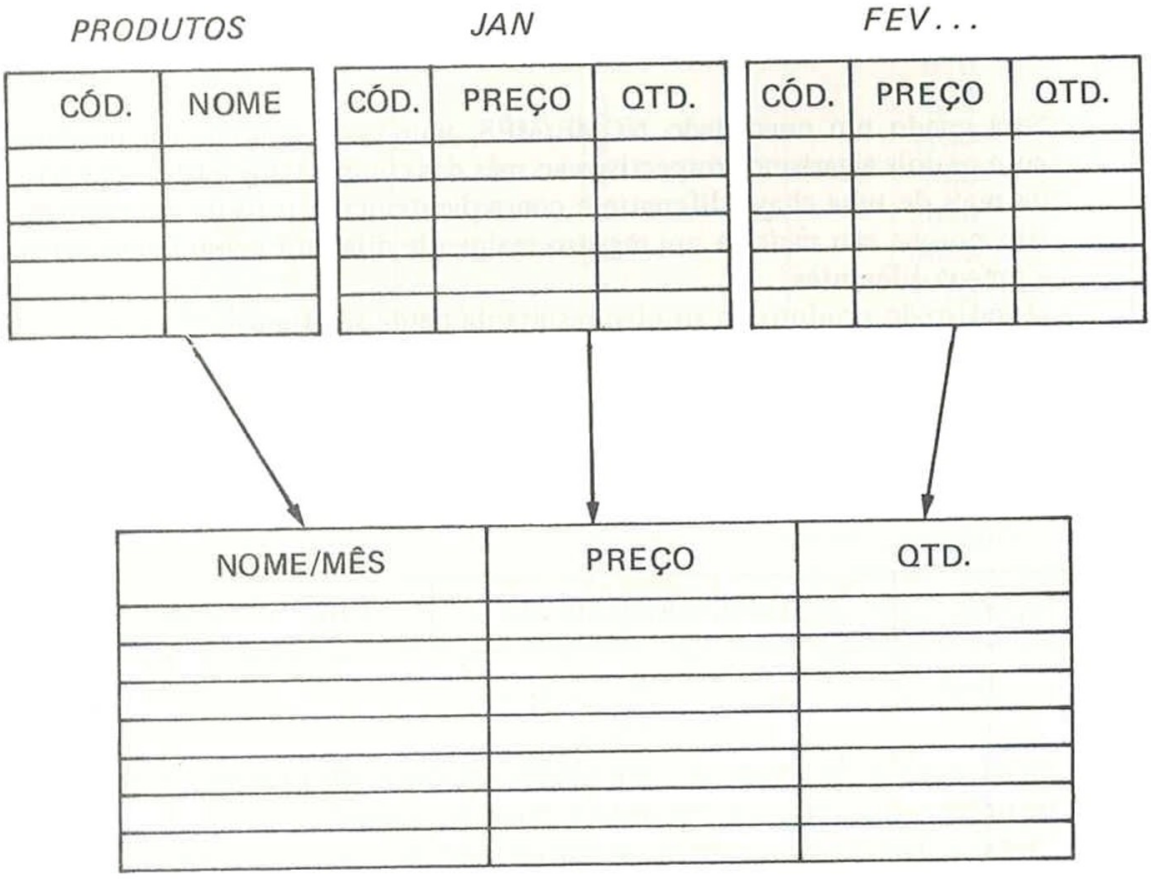


Figura 19 Recriando um único arquivo.

Podemos criar um arquivo (tabela) de meses conforme Fig. 20. Este arquivo será muito útil, inclusive, em outras aplicações que necessitem do nome dos meses.

Como sabemos que a chave é o nome do produto acrescido de duas posições para o mês de aquisição, para se obter o nome do mês basta pegar as duas últimas posições da chave. Tendo o número, acessa-se o arquivo de nome dos meses.

Logo no início da função, você deve ler o arquivo de meses e colocá-lo numa matriz DIM V$ (12, 11), ou criá-lo já em um Array.

Para obter o consumo e o preço médio, basta acumular valores e quantidades independentemente, é claro, e dividir este somatório pelo número do último mês da compra, que será obtido após a ordenação e quando na pesquisa o nome do próximo produto for diferente. Este mesmo número pode ser usado para acessar a tabela de meses para a impressão da seguinte condição: "Consumo e preço médio até o mês . . .". O uso da sub-rotina que acessa o dado será semelhante ao do cálculo do salário líquido.

2º) *Controle do consumo médio de combustível:*

Para isto, crie um arquivo com os seguintes dados:

DT – Data do Abastecimento
KI – Quilometragem Inicial
KF – Quilometragem Final
L – Quantidade de Litros
V – Valor pago

A chave pode ser a quilometragem Inicial e/ou a data. Quando se ordenar os arquivos, os lançamentos automaticamente vão ficar na ordem do abastecimento.

Os cálculos:

- Quilômetros – KR = KF – KI
- Dividir estes quilômetros rodados pelos números de litros para achar o consumo – C = KR/L
- Com o valor, pode-se calcular o total gasto no mês acumulando-se todos os valores dentro do mês.

Como na sugestão anterior, a recuperação dos dados deve ser feita como no "Cálculo do Salário Líquido".

Pelo que você viu, há boa margem de utilização. O segredo é a forma (quais os dados e como arranjá-los) que se deve definir no arquivo.

Nos dois exemplos, os arquivos tiveram formas diferentes e com isto puderam fornecer informações diferentes. Então, atenção: quando for projetar os arquivos, procure imaginar qual ou quais informações pretenderá extrair deles.

Com o parágrafo anterior, deu para se abordar a diferença entre "dado" e "informação", o que geralmente é confundido. Os dados estão no arquivo. Você os digitou. A informação foi o resultado, a conseqüência que você extraiu deles. Por exemplo, qual a "diferença das compras de um mês para o outro".

Aqui vai a seguinte sugestão de ordem prática:

- Crie vários DB. Cada um com arquivos respectivos à aplicação imaginada.
- Salve cada um numa fita K7 em separado.
- Cole uma etiqueta no K7 com dizeres que identifiquem a aplicação.
- Não esqueça de documentar em papel o mínimo necessário para lhe facilitar a utilização.
- Guarde uma versão básica (original) do programa com o DB = " ■ ", sem as funções específicas que não interessarem a todas as aplicações.

NÚMERO	NOME
Ø1	JANEIRO
Ø2	FEVEREIRO
Ø3	MARÇO
Ø4	ABRIL
Ø5	MAIO
Ø6	JUNHO
Ø7	JULHO
Ø8	AGOSTO
Ø9	SETEMBRO
1Ø	OUTUBRO
11	NOVEMBRO
12	DEZEMBRO

Figura 2Ø Meses.

IMPLEMENTANDO O DIRETÓRIO

Caso tenha aceito minha primeira sugestão de criar um "diretório", seguem mais algumas palavras e um modelo (Fig. 21).

	NOME ARQUIVO	INÍCIO	TAMANHO	TOTAL REGISTROS	DADOS POR REGISTROS
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Figura 21 Diretório.

O que se pode obter de útil com o diretório?

Considere o diretório como o índice de um livro. Quais os benefícios que se tira do índice de um livro?

- Com uma breve consulta, sabe-se que o assunto procurado está presente ou não no livro em questão.
- Se está presente, vê-se logo onde começa.

– Pelo início do próximo assunto, obtém-se o número de páginas que compõem o assunto que nos interessa.

– Havendo um subíndice, sabem-se quais os tópicos "chaves" dentro do assunto.

– Etc. . .

De modo muito semelhante, é usado o diretório.

Ainda como um fichário em que se possa incluir ou retirar folhas, sempre que isto ocorrer será necessário corrigir-se a numeração indicada das páginas. De modo semelhante, deverá ser atualizado o diretório, sempre que se criar, deletar ou alterar dados, registros ou arquivos.

Com os indicadores de início dos arquivos, pode-se ir, como num livro, direto ao ponto desejado sem ter que passar por todos os assuntos anteriores.

Somente por estas breves palavras, cremos já ter mostrado como um diretório pode ser útil, abreviando enormemente nossas funções e o tempo de acesso aos arquivos.

Quando surgem as informações para o diretório? As iniciais? Assim que se cria os arquivos. As de atualização, exatamente no momento da execução da função que altera o arquivo.

O dado "início" é chamado "ponteiro" ou "pointer". Ele aponta, exatamente como indica seu nome, o início do arquivo dentro do DB. Temos usado os indexadores A, B e C como ponteiros.

Com o diretório, a deleção de um arquivo pode ser bem mais rápida, visto que, após consultá-lo e obter os respectivos início e fim, bastará eliminá-lo do DB e seu conseqüente registro no próprio diretório. Em seguida, deve ser processada a atualização das novas posições dos arquivos no diretório.

Aos registros do diretório, dá-se o nome de "entrada". Cada arquivo tem uma entrada no diretório. Alterando-se um arquivo, todas as entradas a partir daquele deverão ser alteradas.

O que é mais imediato é a possibilidade de verificação da existência ou não de um nome que se supõe novo, entre os já existentes no diretório.

O diretório pode ser enriquecido com outras informações sobre o arquivo. Deve-se evitar, no entanto, o exagero.

O diretório poderá ser um arquivo independente ou fazer parte do próprio DB.

Com o diretório, todas as funções deverão acessar primeiramente o diretório. Isto significa que todas as funções que acessam os arquivos deverão ser alteradas, ou melhor, abreviadas.

CRIAÇÃO DO DIRETÓRIO

Vamos criá-lo independente, por achar mais prático.

Faremos também algumas restrições tais como:

– Tamanho do nome = 8 símbolos, que deverão ser totalmente preenchidos, mesmo com espaços se for o caso.

– Número de entradas (arquivos) = 1Ø

– Total de registros, no máximo 999, isto é, um número com no máximo três algarismos.

– Dados por registro, no máximo nove, isto é, um número com no máximo um algarismo.
(estes números poderão ser alterados a seu critério).

Para criarmos a matriz do diretório, devemos dar o comando BASIC em formato livre:

DIM V$ (1Ø, 2Ø)

Devemos considerar cada entrada com o seguinte layout:

	NOME	INÍCIO (I)	TAMANHO (T)	REG'S (QR)	DADOS (QD)
V$ (N,	TO 8)	9 TO 12)	13 TO 16)	17 TO 19)	20)

Inicialize a matriz V$, com zeros com os seguintes comandos, que depois devem ser apagados:

```
FOR F = 1 TO 1Ø
LET V$ (F) = "ØØØØØØØØØØØØØØØØØØØØ"
NEXT F
```

É muito bom também que inclua a função auxiliar – "LISTAR DIRETÓRIO".

```
PRINT "DIRETÓRIO"
PRINT , , "NOME INIC TAM REG DADOS"
FOR A = 1 TO 1Ø
PRINT V$ (A, TO 8); " " ; V$ (A),
9 TO 12); " " ; V$ (A, 13 TO 16); " ";
V$ (A, 17 TO 19); " " ; V$ (A, 2Ø)
NEXT A
RETURN
```

Sempre quando incluir uma nova função, atualize o *menu* de opções e os testes de modo a permitirem a aceitação do novo código.

Deve ser criada uma nova sub-rotina – ACESSANDO DIRETÓRIO:

```
9600 REM ** ACESSANDO DIR **
9610 FOR F=1 TO 10
9620 LET D$=V$(F, TO 8)
9630 IF D$=N$ OR D$="00000000" THEN GOTO 9660
9640 NEXT F
9650 PRINT "FIM DO DIRETORIO"
9660 IF D$=N$ THEN PRINT N$," JA  CRIADO"
9670 IF D$=N$ THEN GOTO 9700
9680 PRINT N$;" NAO CRIADO"
```

```
9690 LET F=F-1
9700 LET I=VAL V$(F,9 TO 12)
9710 LET T=VAL V$(F,13 TO 16)
9720 LET QR=VAL V$(F,17 TO 19)
9730 LET QD=VAL V$(F,20)
9740 IF D$<>N$ THEN LET F=F+1
9750 LET T2=T+I
9760 RETURN
```

Como já foi dito, todas as funções que alteram o DB deverão ser ajustadas à nova situação, isto é, considerando-se agora o diretório.

Conforme a rotina, após solicitação do nome do arquivo e acesso ao diretório, devem ser seguidos respectivamente os seguintes passos:

1 – *Para criar Arquivo* (se N$ <> D$ e F < 11)

– GOSUB 96ØØ

Caso dê "inexistente" e não "fim do diretório"

– Criar o arquivo, guardando:

Tamanho do DB antes (T1 = LEN W$)
NOME → N$
INÍCIO → T1 + 1
Tamanho do DB depois (T2 = LEN W$)
Tamanho → T2 – T1
Quantidade de Registros (QR) → R
Quantidade de dados (QD) → D

– *F*, apontará para a primeira entrada disponível.

2 – *Para Deletar:*

2.1 – *Arquivo* (se N$ = D$)

– Deleção propriamente dita do arquivo baseando-se no *I* e no *T*

LET W$ = W$ (TO I) + W$ (I + T TO)

– Atualização do diretório

- Subtraindo a partir do arquivo *F* + 1, *T* dos *I*'s seguintes
- Substituindo dados de *F* pelos dados dos *F*'s seguintes, ficando o último zerado.

2.2 – *Registro*: (se N$ = D$)

– Deleção propriamente dita do registro

- Buscando registro, a partir de *I* pela chave N$, solicitada
- Eliminar a parte relativa ao registro
- Criar *T1* (contendo o tamanho do registro a eliminar)

- Atualização do diretório
  - Subtraindo *T1* de *T*
  - Subtraindo *T1* dos I's seguintes ao *F*

3 – *Para Alterar*: (se N$ = D$)

- Acessar arquivo a partir de *I*
- Guardar o LEN W$ em T1, antes
- Guardar o LEN W$ em T2, depois
- Subtrair T2 de T1
  - Se T2 > T1, será negativo
  - Se T2 < T1, será positivo
- Somar algebricamente o resultado ao *T* do arquivo e aos *I*'s seguintes ao *F*.

4 – *Para Inclusão de Registros*: (se N$ = D$)

- Acessar final do arquivo N$ com T2
- Salvar o tamanho do DB antes da inclusão
- GOTO 54Ø para incluir (como se fosse criar)
  Guardar e somar respectivamente:

  T2 = LEN W$ – T1 (tamanho após inclusão)
  R ⟶ QR + R
  N ⟶ QN + N
- Somar T2 aos I's a partir de F + 1

5 – *Ordenar, Listar e Funções específicas* (se N$ = D$)

Basicamente sofrerão alteração em suas lógicas; apenas não necessitarão solicitar – "NÚMERO DE DADOS POR REGISTRO". Poderão usar QD (quantidade de dados) diretamente do diretório.

Deverão acessar inicialmente o diretório e testar sua existência.

DETALHANDO UM POUCO MAIS OS PASSOS

1 – *Criando Arquivo*

- Verificar existência do arquivo e guardar o tamanho do DB antes da criação do novo arquivo.
  Acrescentar após comando 52Ø INPUT N$:

  ```
  GOSUB 96ØØ
  IF N$ = D$ OR F > 11 THEN RETURN
  LET T 1 = LEN W$
  ```
- Entrada normal através da rotina 1
- Ao término da digitação, após comando

  69Ø LET W$ = W$ (TO LEN W$ – 1)

  Atualize o diretório com os comandos:

  (nome) – LET V$ (F, TO 8) = N$

```
(I) – IF F > 1 THEN LET V$ (F, 9 TO 12)
          = STR$ (I + T)
      IF F = 1 THEN LET V$ (F, 9 TO 12) = "2"
(T) – LET V$ (F, 13 TO 16) = STR$ (LEN W$ – T1)
(QR) – LET V$ (F, 17 TO 19) = STR$ R
(QD) – LET V$ (F, 2Ø) = STR$ D
```

Atenção:

Sempre que for criar um arquivo, mesmo que não use os oito símbolos, complete com brancos. Esta é uma forma prática de evitar duplicata de nomes.

2 – *Deletando Arquivo/Registro:*

Deverá ser alterada um pouco a rotina já existente.

– Após solicitação do nome do arquivo e verificada sua existência,

(IF N$ <> D$ THEN RETURN), solicitar

- Todo o arquivo? (S/N)
- *Sim* – vai até o final da rotina
- *Não* – pede a chave

Atenção:

F neste caso é o número da entrada a ser eliminada. Vamos ver o primeiro caso:

2.1 – *Todo o Arquivo*

– Deletar o arquivo do DB com o comando:

```
LET W$ = W$ (TO I) + W$ (I + T TO)
```

– Atualizar o diretório

```
INICIO  IF F + 1 > 1Ø THEN GOTO FIM
        IF V$ (F + 1, TO 3) = "ØØØ" THEN GOTO FIM
        LET V$ (F, TO 8) = V$ (F + 1, TO 8)
        LET V$ (F, 9 TO 12) = STR$ (VAL V$ (F + 1, 9 TO 12) – T)
        LET V$ (F, 13 TO) = V$ (F + 1, 13 TO)
        LET F = F + 1
        IF V$ (F, TO 3) <> "ØØØ" THEN GOTO INICIO
FIM     LET V$ (F) = "ØØØØØØØØØØØØØØØØØØØØ"
        RETURN
```

2.2 – *Somente um Registro*

– Solicitar a chave do registro

```
PRINT "CHAVE DO REGISTRO?"
INPUT N$
```

Neste caso, será necessária a pesquisa do DB, mas não a partir do início do DB e sim do início do arquivo. Para isto, *I* já conterá esta localização.

Como o registro pode conter apenas um dado, é necessário subtrair 1 de *I*.

Aqui serão usadas as sub-rotinas 953Ø e 9ØØØ, normalmente, e o teste de chave encontrada.

Terminado isto, teremos em *A* o início do registro. Para obter o final, usaremos os mesmos comandos da rotina de deleção usada até então. Em *B*, temos o início dos dados. Começamos a pesquisa a partir daí, somando 1 a *B*.

```
FAST                                                    (9ØX25)
GOSUB 953Ø
LET I = I – 1
GOSUB 9ØØØ
IF D$ <> N$ THEN RETURN
IF QD = 1 THEN GOTO DEL
FOR C = B + 1 TO LEN W$
LET R$ = X$ OR R$ = "▟" THEN GOTO DEL
NEXT C
```

À pesquisa, encontrando o próximo arquivo ou ainda o final do DB, restará apenas deletar o registro.

– *Vamos deletar:*

```
DEL   LET W$ = W$ (TO A – 1) + W$ (C TO)
      PRINT N$; "DELETADO"
```

– Vamos atualizar o diretório
O pedaço eliminado tem o tamanho de *C* menos *A*.

```
LET T1 = C – A
LET T = T – T1
LET V$ (F, 13 TO 16) = STR$ T
```

– Vamos eliminar 1 registro do QR

```
LET QR = QR – 1
LET V$ (F, 17 TO 19) = STR$ QR
```

Estando esta entrada atualizada, vamos ajustar apenas o início dos arquivos seguintes a este, subtraindo de seus inícios o tamanho do pedaço (T1) eliminado:

```
INICIO1 IF F + 1 > 1Ø THEN RETURN
        LET F = F + 1
        IF V$ (F, TO 3) <> "ØØØ" THEN
        LET V$ (F, 9 TO 12) = STR$ (VAL V$ (F, 9 TO 12) – T1)
        GOTO INICIO1
```

Tendo agora T2 (posição do início + o tamanho do arquivo), podemos substituir nas sub-rotinas a expressão LEN W$ por *T2 – 1*, o que abreviará bastante a pesquisa.

Algumas alterações e implementações devem ser feitas na sub-rotina 9ØØØ.

– Após 915Ø → GOTO 9Ø55
918Ø → LET D$ = " "
LET I = B – 1
IF QD = 1 THEN GOTO 91ØØ
IF B > T2 THEN GOTO 9Ø55

– Substituir 92ØØ de GOTO 9Ø5Ø
para GOTO 9Ø2Ø
93ØØ de RETURN
para LET C = B

– Após 923Ø → RETURN

3 – *Alterando Arquivo*

– Após pedir nome do arquivo, acessar diretório, verificando sua existência. Caso positivo, guardar o tamanho do DB:

```
GOSUB 96ØØ
IF N$ <> D$ THEN RETURN
LET T1 = LEN W$
```

Podem ser eliminados os GOSUB 95ØØ
GOSUB 9ØØØ

usados para se chegar ao início do arquivo.

– Solicitar chave do registro a ser alterado

```
PRINT "QUAL A CHAVE?"
INPUT N$
```

– Como no caso anterior, pesquisar a partir do *I – 1*, pois o registro pode ter apenas um dado.

```
FAST                                    (9ØX26)
GOSUB 953Ø
LET I = I – 1
GOSUB 9ØØØ
IF D$ <> N$ THEN RETURN
```

– A partir daqui, segue a codificação já existente.
Apenas acrescente o comando:

```
IF C <= B THEN LET C = T1 + 1
```

para o caso específico de:

- alteração do último dado e
- alterar diminuindo

– Ao final, atualizar diretório: guarde em *T2* a diferença do tamanho do DB, "depois" menos "antes"

```
LET T2 = LEN W$ – T1
```

– Se a diferença for positiva, o DB aumentou de tamanho. Caso contrário, o conteúdo de *T2* conterá quantos bytes diminuíram. Some algebricamente esta diferença ao T Do arquivo.

```
LET V$ (F, 13 TO 16) = STR$ T + T2
```

– Para os demais:

```
INICIO2   IF F + 1 > 1Ø THEN RETURN
          LET F = F + 1
          IF V$ (F, TO 3) <> "ØØØ" THEN
```

```
LET V$ (F, 9 TO 12) = STR$
(VAL V$ (F, 9 TO 12) + T2
GOTO INICIO2
```

4 – *Incluindo Novos Dados*

Como nos outros casos: acessar diretório e verificar existência ou não do arquivo, pelo teste:

```
IF N$ = D$
```

– Salvar o tamanho do DB antes da inclusão em *T1*

```
T1 = LEN W$
```

Lembre-se de que *T2* tem o início do próximo arquivo, então, para salvar do fim deste arquivo ao fim do DB

```
LET P$ = W$ (T2 TO)
```

– Preparar para inclusão e incluir:

```
LET W$ = W$ (TO T2 – 1) + "■"
GOTO 54Ø
```

No *menu*, foi estabelecido um código a ser digitado para a opção de "Inclusão de Registros".

Este código deverá estar na variável K$. Antes da pergunta "QUANTOS DADOS POR REGISTRO?" pode ser incluído um teste,

```
IF K$ <> "A" THEN GOTO X
LET D = QD
GOTO Y
```

Considerando que "A" é o código para inclusão e que estamos aproveitando a rotina e "Criar Arquivos".

```
X = (comando que pede nº de dados)
Y = (comando após pedido do nº de dados)
```

Na rotina de inclusão, após o comando:

```
69Ø LET W$ = W$ (TO LEN W$ – 1), deve
ser incluído um teste (inclusão = "A")
IF K$ = "A" THEN GOTO 5Ø8Ø
```

– Juntar as duas partes W$ + P$

```
LET W$ = W$ + P$
```

– Atualizar o diretório

Calcular tamanho acrescido, corrigir *T* e *QR* deste arquivo:

```
LET T2 = LEN W$ – T1
LET T  = T + T2
LET V$ (F, 13 TO 16) = STR$ T
LET V$ (F, 17 TO 19) = STR$ (QR + R)
```

Corrigir *I* dos arquivos seguintes:

```
INICIO 4  IF F + 1 > 1Ø THEN RETURN
          LET F = F + 1
          IF V$ (F, TO 3) <> "ØØØ" THEN
             LET V$ (F, 9 TO 12) = STR$
                (VAL V$ (F, 9 TO 12) + T2
          GOTO INICIO 4
```

5 – *Ordenando*

Após solicitação do nome do arquivo (N$):

```
GOSUB 96ØØ
IF N$ <> D$ THEN RETURN
```

Antes do primeiro GOSUB 93ØØ, para o maior tamanho de chave (TC) e maior registro (TR) fazer:

```
C = I + 9
```

Nos comandos que pedem o número de dados por registro, substituir *N* por *QD*.

Nos comandos que usam a expressão LEN W$, é mais interessante usar a expressão T2 – 1, pois diminui a pesquisa.

O comando 462Ø deve ser eliminado.

Antes do segundo GOSUB, fazer:

```
C = I + 8
```

para preencher a matriz M$.

Antes de retornar o arquivo ao DB, nos comandos que salvam o início e o final, usar T2 e a expressão I + 8:

```
LET N$ = W$ (T2 TO) – final
LET W$ = W$ (TO I + 8) – início
```

6 – *Listando*

Logo de início, eliminar o comando 2Ø2Ø.

No lugar do GOSUB 9ØØØ, usar GOSUB 96ØØ.

Testar a existência:

```
IF N$ <> D$ THEN RETURN
```

Como no caso anterior, usar *QD* no lugar da variável *N*, solicitada.

Antes do GOSUB 93ØØ, para recuperar dados a serem listados, faça:

```
C = I + 9
```

para que a pesquisa comece exatamente no primeiro dado.

7 – *Nas Aplicações específicas*

Neste caso, temos a função "8" – "Salário líquido":

– Após solicitação e fornecimento do nome do arquivo:

```
GOSUB 96ØØ
```

– Teste de existência

```
IF D$ <> N$ THEN RETURN
```

GOSUB 953Ø
LET C = I + 9

– A partir daí, a rotina:

Se for acessar chaves:

GOSUB 9ØØØ

Se for acessar dados:

GOSUB 93ØØ

O restante não há mudanças nem adaptações.

NOTA: Nos casos em que os comandos BASIC apresentados não foram acompanhados por números ou apresentaram um label no lugar dos números (tipo INÍCIO, INÍCIO 1 etc.) objetivou deixar o interessado à vontade para escolher os números que melhor convier.

SALVANDO E RECUPERANDO APENAS O DB NO TK85

Salvando

O recomendável é que se reescreva a rotina de BACKUP do DB, isto é, a opção 7.

Pode ser aberta em várias sub-opções:

1 – "DB" + PROGRAMA
2 – SOMENTE O "DB"

e suas variações de alta e baixa velocidade.

Vamos ficar, para o exemplo, apenas com estas duas opções. Na primeira, o SAVE normal, podendo ser HISAVE. Na segunda, na verdade deverão ser salvos o DB, isto é, a variável W$ e o diretório daquele DB (caso tenha implementado), isto é, o array V$.

Após a gravação, tanto W$ como V$ devem ser reduzidas à situação inicial, isto é, W$ = "■" e V$ ser totalmente zerado.

Vamos ver como fica a nova codificação da opção 7:

```
3500 PRINT "SALVANDO DB (BACKUP)"
3510 PRINT ,,"1 - ""D""+PROGRAMA"
3520 PRINT "2 - SOMENTE O ""DB"""
3530 SLOW
3540 LET K$=INKEY$
3550 IF K$="1" THEN GOTO 3590
3560 IF K$="2" THEN GOTO 3640
3580 GOTO 3540
3590 PRINT "PREPARE O GRAVADOR, PRESSIONE PLAY E NEW LINE"
```

```
3600 PAUSE 4E4
3610 LET A=USR 8405
3620 IF A<>0 THEN GOTO 3830
3630 GOTO 100
3640 DIM C$(LEN W$-T1)
3650 LET Z=LEN W$
3660 LET C$=W$
3670 LET Z$="C"
3680 PRINT "PREPARE O GRAVADOR, PRESSIONE PLAY E NEW LINE"
3690 PAUSE 4E4
3700 LET A=USR 9008
3710 IF A<>0 THEN GOTO 3830
3720 LET W$="■"
3730 DIM C$(200)
3740 LET Z=200
3750 FOR A=1 TO 10
3760 LET C$=C$+V$(A)
3770 NEXT A
3780 LET C$="C"
3790 LET A=USR 9008
3800 IF A<>0 THEN GOTO 3830
3810 GOTO 5550
3820 RETURN
3830 PRINT "ERRO=";A
3840 RETURN
```

A variável C$ está funcionando como o BUFFER, isto é, área ou variável de saída.

O primeiro tamanho de C$ é o tamanho exato do DB obtido pela expressão LEN W$. O segundo é o tamanho exato do diretório, que pode ser determinado pelo número de entrada vezes (X) 20.

O comando 381Ø desvia para uma sub-rotina que zera o diretório, após sua gravação.

Caso ocorra algum erro, o código ficará na variável *A* e o mesmo será impresso junto com a palavra ERRO. Caso isto ocorra, consulte o manual que acompanha o TK85 para análise da causa do erro.

Conforme já foi dito inicialmente, nos comandos de gravação, qualquer que seja, pode-se optar pela velocidade normal ou HIGH-SPEED, conforme indicado no próprio comando.

Procure pressionar PLAY antes de NEW LINE.

Recuperando

Neste caso, terá que ser criada uma rotina para tal. Considerando que se tenha usado 9 para a rotina de ordenação e B para listar o diretório, podemos usar C para esta.

É boa prática manter um controle sobre o tamanho dos DB's gravados. Por exemplo, nunca maior que 3K bytes. Isto é necessário, neste caso, pois na leitura da fita é obrigatória a definição de um tamanho determinado para o BUFFER, ou variável, de leitura. Vamos considerar o tamanho igual a 3K.

Vamos ver como fica a codificação:

```
6000 DIM C$(3000)
6010 LET Z=0
6020 LET Z$="C"
6030 PRINT "POSICIONE A FITA, PRESSIONE NEW LINE E PLAY"
6040 PAUSE 4E4
6050 LET A=USR 9189
6060 IF A<>0 THEN GOTO 3830
```

Como sabemos pelo manual do equipamento, após esta leitura, a variável *Z* conterá o tamanho exato da variável recuperada; com isto podemos recriar o DB, ou seja, a variável W$ com o seu tamanho exato.

```
6070 LET W$=C$( TO Z)
6080 DIM C$(200)
6090 LET Z=200
6100 LET A=USR 9189
6110 IF A<>0 THEN GOTO 3830
6120 LET B=1
6130 FOR A=1 TO 10
6140 LET V$(A)=C$(B TO (20*A))
6150 LET B=B+20
6160 NEXT A
6170 LET C$=""
6180 RETURN
```

As observações quanto a algum possível erro na leitura são as mesmas da gravação; inclusive, está sendo desviado para os mesmos comandos de erro.

É óbvio que a velocidade de leitura deve ser a mesma usada na gravação.

Para a remontagem do diretório, foi usada a variável *B* para auxiliar no algoritmo que transforma a variável linear C$ no array V$, fazendo exatamente o inverso da rotina de gravação que transforma o array V$ de 1Ø ocorrências na variável linear C$.

Procure pressionar NEW LINE antes do PLAY.

SALVANDO E RECUPERANDO APENAS O DB NO TK9ØX

No TK9ØX é muito fácil gravar programas ou apenas dados. Esta rotina faz exatamente isto.

```
3500 PRINT "SALVANDO ZX-DB"
3510 PRINT,, "1 – Programa + 'DB' "
3520 PRINT "2 – Somente o 'DB' "
3540 LET k$ = INKEY$
3550 IF k$ = "1" THEN GOTO 3610
3560 IF k$ = "2" THEN GOTO 3640
3580 GOTO 3540
3610 SAVE "ZX-DB": PAUSE 0
3630 GOTO 40
3640 SAVE "ZX-DB" DATA w$ ( ): PAUSE 0
3650 GOTO 40
```

Para recuperar a matriz w$ gravada, basta comandar (modo direto) LOAD "ZX-DB" DATA w$ ( ).

Para entrar no modo direto basta, após o menu entrar na opção 1, deletar ambas as aspas e digitar STOP seguido de ENTER. Você obterá a mensagem do sistema operacional

H STOP em INPUT 52Ø:1

e terá todo o acesso ao modo direto e à listagem.

PALAVRAS FINAIS

Após este capítulo, segue uma listagem completa do programa DB.

Poderá ser verificado que foram feitas algumas pequenas alterações visando poupar alguns bytes.

Observe que as constantes numéricas aparecem agora entre aspas (") e com a função VAL prefixando-as. Foram incluídos os comandos de 3Ø a 9Ø, onde algumas constantes foram colocadas em variáveis (J = 9ØØØ, K = 93ØØ, L = 95ØØ, M = 953Ø, V = 5 e UM = 1). As constantes "Ø" (zero) foram substituídas pela expressão PI-PI. Para ganhar mais bytes, elimine os comandos REMS.

Desejo sinceramente que tenha obtido ou que venha a obter algum proveito deste trabalho.

Sabemos que tudo que se faz, por melhor que fique, sempre pode ser melhorado.

Não temos a menor pretensão de sermos completos, muito menos perfeitos, por isto, aceitamos críticas e sugestões de toda natureza. Aceitamos também consultas para alguns esclarecimentos extras.

Para tal, basta escrever para a editora indicando nosso nome e o do livro, ou diretamente para:

Caixa Postal 57.041 – Rio de Janeiro

Qualquer que seja a crítica, será considerada como colaboração.

Manoel Silva Rodrigues (1986)

ANEXOS

LISTAGEM COMPLETA DO PROGRAMA ZX-DB

Lembretes:

– Comande antes da primeira execução do programa (apenas na primeira) LET W$ = "■"

– Nunca comande RUN

VARIÁVEIS/CONSTANTES

```
30 REM VARIAVEIS/CONSTANTES
40 LET V=VAL "5"
50 LET UM=VAL "1"
60 LET J=VAL "9000"
70 LET K=VAL "9300"
80 LET L=VAL "9500"
90 LET M=VAL "9530"
```

Atenção para estas variáveis. Não use nenhuma com o mesmo nome. Na verdade, são constantes repetidas várias vezes.

Fixando-as em variáveis, economizam-se bytes.

MENU DE OPÇÕES

```
100 CLS
110 SLOW
120 PRINT AT V,VAL "9";"OPCOES"
,,,,"   FUNCOES PRINCIPAIS"
130 PRINT ,,"1 - CRIAR ARQUIVO",
"2 - DELETAR ARQUIVO/REGISTRO",
"3 - ■SEM FUNCAO DEFINIDA","4 -
LISTAR ARQUIVO"
```

```
 140 PRINT ,,"   FUNCOES AUXILIARES",,,
"5 - PRINT DO DB",,
"6 - MEMORIA DISPONIVEL","7 - SALVAR "
"DB""",,"8 - SALARIO LIQUIDO (ESPECIFICA)"
 150 LET K$=INKEY$
 160 IF K$<STR$ UM OR K$>"8" THEN GOTO VAL "150"
 170 CLS
 180 LET I=UM
 190 LET INI=I
 200 GOSUB VAL K$*VAL "500"
 210 PAUSE VAL "4E4"
 220 GOTO VAL "100"
```

CRIAR ARQUIVOS

```
 500 PRINT " CRIANDO ARQUIVO"
 510 PRINT ,,"QUAL O NOME ?"
 520 INPUT N$
 530 LET W$=W$+"▟"+N$+"▬"
 540 PRINT "QUANTOS REGISTROS ?"
 550 INPUT R
 560 PRINT "QUANTOS DADOS ?"
 570 INPUT D
 580 CLS
 590 PRINT N$
 600 FOR B=UM TO R
 610 FOR A=UM TO D
 620 PRINT "DADO ";B;".";A;">";
 630 INPUT D$
 640 PRINT D$
 650 LET W$=W$+D$+" ▪"
 660 NEXT A
 670 LET W$=W$( TO LEN W$-UM)+"▬"
 680 NEXT B
 690 LET W$=W$( TO LEN W$-UM)
 700 RETURN
```

DELETAR ARQUIVO

```
1000 SLOW
1010 PRINT "DELETAR..."
1020 PRINT ,,"ARQUIVO ?",,"REGISTRO ?"
1030 IF INKEY$="" THEN GOTO VAL "1030"
```

```
1040 IF INKEY$="A" THEN GOTO VAL "1070"
1050 IF INKEY$="R" THEN GOTO VAL "1100"
1060 GOTO VAL "1030"
1070 GOSUB L
1080 PRINT "NOME ARQUIVO ?"
1090 GOTO VAL "1120"
1100 GOSUB M
1110 PRINT "CHAVE REGISTRO ?"
1120 INPUT N$
1130 PRINT N$
1140 FAST
1150 GOSUB J
1160 IF N$<>D$ THEN RETURN
1170 FOR C=B-UM TO LEN W$
1180 LET R$=W$(C)
1190 IF R$=X$ OR R$="■" THEN GOTO VAL "1210"
1200 NEXT C
1210 LET W$=W$( TO A-UM)+W$(C TO )
1220 PRINT "DELETADO"
1230 RETURN
```

SEM FUNÇÃO

```
1500 PRINT AT V,V;"SEM FUNCAO DEFINIDA"
1510 RETURN
```

LISTAR ARQUIVO

```
2000 PRINT " LISTAR ARQUIVO ?"
2010 GOSUB L
2020 LET FIMARQ=PI-PI
2030 INPUT N$
2040 PRINT N$
2050 PRINT "DADOS POR REGISTRO ?";
2060 INPUT N
2070 PRINT N
2080 DIM T$(N,VAL "10")
2090 PRINT "TITULOS DOS DADOS ?(TAMANHO 10)"
2100 FOR A=UM TO N
2110 INPUT T$(A)
2120 NEXT A
2130 PRINT
2140 FAST
```

```
2150 GOSUB J
2160 IF D$<>N$ THEN RETURN
2170 SLOW
2180 FOR A=UM TO N
2190 GOSUB K
2210 IF PEEK VAL "16442"<VAL "4" THEN SCROLL
2220 PRINT T$(A);"-";P$
2230 NEXT A
2240 IF PEEK VAL "16442"<VAL "4" THEN SCROLL
2250 PRINT
2260 IF FIMARQ<>UM THEN GOTO VAL "2180"
2270 SCROLL
2280 PRINT "    FIM"
2290 RETURN
```

PRINT DO DB

```
2500 PRINT " PRINT DO ""DB"""
2510 PRINT ,," (CASO ENCHA A TELA, DIGITE",,
" CONT ",,"NEWLINE)"
2520 PRINT ,,W$
2530 RETURN
```

MEMÓRIA DISPONÍVEL

```
3000 PRINT AT V,V;"MAPA DA RAM"
3010 PRINT ,,"TAMANHO PROGRAMA";
TAB 25;PEEK 16396+256*PEEK 16397-16509,
"PROG.+TELA+VARIAVEIS)";T
AB 24;"(";PEEK 16404+256*PEEK 16405-16384;")",
"DISPONIVEL";TAB 25;(PEEK
16386+256*PEEK 16387)-(P
EEK 16412+256*PEEK 16413)
3020 RETURN
```

SALVAR DB

```
3500 PRINT ,,"LIGUE O GRAVADOR",,,
"PRESSIONE QUALQUER TECLA"
3510 PAUSE VAL "4E4"
3520 SAVE "ZX-DB"
3530 GOTO VAL "40"
```

CÁLCULO DO SALÁRIO LÍQUIDO

```
4000 LET I=UM
4010 PRINT "CALCULANDO SALARIO"
4020 PRINT "ARQUIVO ?"
4030 INPUT N$
4040 PRINT N$
4050 FAST
4060 GOSUB L
4070 GOSUB J
4080 IF D$<>N$ THEN RETURN
4090 GOSUB M
4100 PRINT "DIGITE NOME(CHAVE)"
4110 LET I=INI
4120 INPUT N$
4130 PRINT ,,"NOME","SALARIO"
4140 PRINT N$,
4150 GOSUB J
4160 IF D$<>N$ THEN RETURN
4170 GOSUB K
4180 LET D$=P$
4190 GOSUB K
4200 PRINT VAL D$-VAL P$
4210 SLOW
4220 IF PEEK VAL "16442"<VAL "6" THEN CLS
4230 PRINT ,,"OUTRO ?(S/N)"
4240 IF INKEY$="S" THEN GOTO VAL "4100"
4250 IF INKEY$="N" THEN GOTO VAL "4270"
4260 GOTO VAL "4240"
4270 RETURN
```

SUB-ROTINAS

```
9000 REM * SUBROTINA *
9010 REM CHEGAR INICIO ARQ/REG
9020 FOR A=I TO LEN W$
9030 LET R$=W$(A)
9035 IF R$="■" THEN LET INI=A
9040 IF R$=X$ THEN GOTO VAL "9100"
9050 NEXT A
9060 PRINT N$;" NAO EXISTE"
9080 RETURN
9090 REM CHEGAR AO ARQ/REG
```

```
9100 LET D$=""
9110 FOR B=A+UM TO LEN W$
9120 LET R$=W$(B)
9130 IF R$=Y$ OR R$=X$ OR R$="▟" THEN GOTO VAL "9160"
9140 LET D$=D$+R$
9150 NEXT B
9160 IF D$=N$ THEN GOTO VAL "9210"
9170 LET A=B
9180 LET D$=""
9190 IF B<LEN W$ THEN GOTO VAL "9150"
9195 LET I=B-UM
9200 GOTO VAL "9050"
9220 LET C=B
9230 RETURN
9300 REM RECUPERAR DADO
9310 GOSUB M
9320 LET B=C
9330 LET P$=""
9340 FOR C=B+UM TO LEN W$
9350 LET R$=W$(C)
9360 IF R$=Y$ OR R$=X$ OR R$="▟" THEN GOTO VAL "9410"
9370 LET P$=P$+R$
9380 NEXT C
9390 LET FIMARQ=UM
9410 IF R$="▟" THEN LET FIMARQ=UM
9411 RETURN
9500 LET X$="▟"
9510 LET Y$="▬"
9520 RETURN
9530 LET X$="▬"
9540 LET Y$=" ▪"
9550 RETURN
```

VARIÁVEIS DO SISTEMA DO TK85

ENDEREÇOS		CONTEÚDO
DECIMAL	HEXA	
16384	4ØØØ	– Código de erro
16385	4ØØ1	– Diversos FLAGS
16386	4ØØ2	– Endereço da pilha de GOSUB
16388	4ØØ4	– Área a ser protegida (RAMTOP)
1639Ø	4ØØ6	– Modalidade do cursor corrente
16391	4ØØ7	– Número da linha que está sendo executada
16393	4ØØ9	– Primeira variável do sistema a ser salva
16394	4ØØA	– Número da linha que tem o cursor
16396	4ØØØ	– Endereço de início do arquivo de vídeo
16398	4ØØE	– Endereço da posição a ser PRINT
164ØØ	4Ø1Ø	– Início da área de variáveis
164Ø2	4Ø12	– Endereço da variável que está sendo usada
164Ø4	4Ø14	– Início da linha de edição
164Ø6	4Ø16	– Próximo caractere a ser interpretado
164Ø8	4Ø18	– Caractere precedente ao erro de sintaxe assinalado
1641Ø	4Ø1A	– Pilha de cálculos
16412	4Ø1Ø	– Final da pilha de cálculos
16414	4Ø1E	– Usado p/cálculo de ponto flutuante
16415	4Ø1F	– Início da área de cálculos da memória
16417	4Ø21	– Um byte de trabalho
16418	4Ø22	– Número de linhas para entrada
16419	4Ø23	– Número de linhas do topo do vídeo
16421	4Ø25	– Última tecla pressionada
16423	4Ø27	– Estado de 'debounce' do teclado
16424	4Ø28	– Número de linhas em branco embaixo ou acima do vídeo
16425	4Ø29	– Próxima linha do programa a ser executada
16427	4Ø2B	– Número da linha a ser executada caso comande CONT
16429	4Ø2D	– Diversos FLAGS
1643Ø	4Ø2E	– Tamanho do STRING
16432	4Ø3Ø	– Próximo item da tabela de sintaxe
16434	4Ø32	– Semente do número randômico gerado
16436	4Ø34	– Incrementado de um a cada quadro exibido. Usado pelo PAUSE
16438	4Ø36	– Coordenadas do último PLOT
1644Ø	4Ø39	– Coordenadas do PRINT AT
16443	4Ø3B	– FLAGS relativos às modalidades FAST/SLOW
16444	4Ø3C	– Áreas para guardar saídas LPRINT
16477	4Ø5D	– Área usada pelo calculador de memória
165Ø7	4Ø7B	– Dois bytes de trabalho
165Ø9	4Ø7D	– Início do programa em BASIC

FOLHA DE CODIFICAÇÃO BASIC

Como venho sugerindo implementar novas rotinas no DB, nada mais justo que facilitar esta tarefa.

Já venho dando "mãozinhas" na codificação, agora vou apresentar uma folha que facilita a escrita desta codificação. É a folha de "Codificação BASIC". Não chega a ser imprescindível, é apenas um subsídio a mais. Com ela pode-se codificar o programa sem ter que usar o microcomputador. Vamos vê-la!

Ela possui apenas três campos de informação:

- OBSERVAÇÕES';
- NÚMERO e
- CÓDIGOS E OPERANDOS

Possui ainda uma numeração equivalente a linhas e colunas do vídeo. Esta numeração visa facilitar o comando PRINT. A princípio, ela é para codificação BASIC, mas pode ser usada também para simular os "Relatórios" no vídeo. Neste caso, quando escrever o PRINT, já estarão definidos os operandos de linha e coluna.

O número de linhas aqui é o da tela normal (22 linhas), pois ela também pode ser usada como cópia da listagem (LIST) do programa. Caso queira trabalhar com 24 linhas, dê o comando

```
POKE 16418,Ø                    (9ØX27)
```

esta variável (byte 16418) contém o número de linhas para a entrada de dados. Normalmente, contém o número 2. Colocando zero, ficam as 24 linhas para os PRINT's e nenhuma para entrada. Antes do próximo INPUT, volte a ter duas linhas para entrada de dados com:

```
POKE 16418,2
```

O campo OBSERVAÇÕES não é para ser digitado. Seu objetivo é o de ser usado para se escrever "LABEL's". O Label é equivalente à numeração dos comandos, só que pode ser muito mais significativo. Você pode usar palavras que lembrem facilmente a utilidade da sub-rotina ou comando.

Ele se mostra útil também quando você ainda não numerou os comandos, isto é, codifica tudo sem número (somente no papel é lógico) depois então escolhe a numeração ideal.

Como o nome diz, serve também para se escrever qualquer observação que possa interessar.

O campo NÚMERO é bem conhecido nosso; resta apenas lembrar que mesmo os números que possuam menos que quatro algarismos ocupam quatro colunas. Procure numerar sempre lembrando-se disto. O número 8, por exemplo, deve ser: três espaços e o 8 na quarta casa.

Na continuação da linha que porventura venha a usar, comece a escrever na primeira quadriculada da linha seguinte, exatamente como acontece na tela.

O campo CÓDIGO e OPERANDOS nada mais é do que o lugar para se escrever a codificação propriamente dita.

Procure deixar sempre um espaço entre os elementos da codificação (número, comando, parâmetros etc.). Procure lembrar como é no vídeo e siga o mesmo espacejamento.

Uma das principais finalidades desta folha é a de servir como listagem do programa, para quem não tem impressora. Esta listagem passa a fazer parte da documentação do programa.

Outra finalidade é de ser o rascunho, no qual você pode rabiscar, apagar etc., em qualquer lugar, mesmo onde não se possa ligar o microcomputador.

Neste livro incluímos esta folha reduzida fixa, para você tirar cópias e usar.

PARA A LINHA SINCLAIR
FOLHA DE CODIFICAÇÃO BASIC
NOME DO PROGRAMA:

FOLHA
___/___

OBSERVAÇÕES		NÚMERO				CÓDIGO E OPERANDOS																													
		Ø	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
	Ø																																	Ø
	1																																	1
	2																																	2
	3																																	3
	4																																	4
	5																																	5
	6																																	6
	7																																	7
	8																																	8
	9																																	9
	10																																	10
	11																																	11
	12																																	12
	13																																	13
	14																																	14
	15																																	15
	16																																	16
	17																																	17
	18																																	18
	19																																	19
	20																																	20
	21																																	21
		Ø	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	

FORM. MSR-1

Figura 22

BIBLIOGRAFIA

- *Introdução a Sistemas de Bancos de Dados*
  C. J. Date – Editora Campus.
- *Estrutura de Dados*
  Paulo Veloso/Clésio dos Santos/Paulo Azeredo/Antônio Furtado – Editora Campus.
- *Banco de Dados*
  Chu Shao Young – Editora Atlas.
- *Gerência de Bases de Dados para Microcomputadores*
  E. G. Brooner – Editora Campus.
- *Sistema de Gerência de Banco de Dados Distribuídos*
  Couceiro/Barrenecha – Livros Técnicos e Científicos Editora.
- *44 Dynamic ZX-81 Games and Recreations*
  Ian Davies – Editora Prentice-Hall.
- *Princípios de Sistemas de Gerência de Bancos de Dados Distribuídos*
  Marco Antonio Casanova e Arnaldo Vieira Moura – Editora Campus.

Impressão e acabamento
(com filmes fornecidos):
EDITORA SANTUÁRIO
Fone (0125) 36-2140
APARECIDA - SP

BANCO DE DADOS PARA TK 90X

(INCLUI TK 85)

Este livro leva à criação de um programa que simula um Banco de Dados **on-line**, monousuário e totalmente conversacional, no TK 90X (ou em qualquer micro compatível com o TK 85).

Como o programa é construído por partes, em módulos, o leitor tem total domínio sobre ele. Ao final do livro, três coisas terão acontecido a quem o estudou:

- **seu BASIC estará grandemente enriquecido;**
- **poderá ampliar como quiser o programa, para adaptá-lo às suas necessidades;**
- **terá uma excelente introdução teórico-prática ao fascinante mundo dos Bancos de Dados.**

Assim, quando o leitor desejar estudar os papas de Banco de Dados, como Paulo Veloso, Antonio Luz Furtado, Chris Date e outros, tudo será mais fácil, pois uma sólida base terá sido construída.

MANOEL SILVA RODRIGUES

Analista de Sistemas da RFFSA e professor da Faculdade SIMONSEN, sendo profundo conhecedor das linhas Sinclair e Apple.

ISBN 85-7001-288-8