MICROINFORMÁTICA

JORGE TREVISAN

CURSO DE PROGRAMAÇÃO BASIC

BASIC

BASIC

LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.

Aos meus pais,
que não mediram
esforços para que
eu conseguisse estudar,

À minha esposa, ELAINE,
pelas horas de convívio
perdidas durante a escrita
deste trabalho.

PREFÁCIO

Esta obra apresenta a experiência do autor, que desde 1980 dedica-se ao ensino de Computação Eletrônica em cursos de Engenharia.

É um livro que visa o ensino da linguagem de programação BASIC, tanto para leitores sem nenhum conhecimento anterior sobre o assunto, bem como para leitores já introduzidos no ramo da computação, que queiram aperfeiçoar seus conhecimentos, pois todas as instruções da linguagem são minuciosamente detalhadas.

Cada instrução é acompanhada de exemplos e problemas resolvidos, com grau de dificuldade crescente, onde os exemplos e os exercícios são comentados para um melhor entendimento. São apresentadas, inicialmente, as 7 instruções fundamentais da linguagem BASIC que são: INPUT, PRINT, LET, GOTO, IF THEN, STOP, REM, com as quais podemos resolver a grande maioria dos problemas encontrados em computação.

À medida que surge a necessidade de recursos maiores para resolução de problemas mais complexos, são apresentadas novas instruções. Os problemas são resolvidos de forma que possam ser utilizados em uma grande variedade de microcomputadores, pessoais e de grande porte, não se prendendo a nenhum especificamente e tentando ser o mais geral possível, respeitadas as diferentes versões da linguagem BASIC.

Portanto, este livro atende tanto ao ensino de computação em Universidades, escolas e cursos especializados como ao autodidata e ao hobbista.

INTRODUÇÃO

A história do processamento de dados automático iniciou-se no século passado, com a tentativa de se construírem aparelhos mecânicos capazes de efetuar cálculos matemáticos.

Em 1833, o matemático inglês, Charles Babbage, apresentou um plano para construir uma máquina de calcular, chamada por ele de Engenho Analítico, provida de uma unidade aritmética (para efetuar as operações matemáticas) e de memória para 1 000 números de 50 dígitos cada, e de instruções (que foram baseadas numa máquina para controlar a tecelagem de fios coloridos).

Este Engenho Analítico somava 2 números de 50 dígitos em 1 segundo e multiplicava 2 números de 20 dígitos em 3 minutos. Porém, as idéias de Babbage ficaram esquecidas, pois não havia na época tecnologia para implementar uma máquina provida de instrução, tipo desvio condicional (*se* acontecer *então* faça isso; *caso contrário* faça aquilo).

Mais de 100 anos depois, quando foram desenvolvidos os relés e as válvulas a vácuo, é que as idéias de Babbage foram retomadas por H. Aiken, que projetou em 1939 o primeiro computador — o Harvard Mark I — que foi construído com relés pela International Business Machines (IBM) em 1944.

O pioneiro dos computadores digitais eletrônicos foi o ENIAC, encomendado pelo Laboratório de Pesquisas Balísticas do Exército americano, com a finalidade de calcular tabelas de balística. Foi construído durante a 2ª Guerra Mundial, entrando em operação em novembro de 1945. Tratava-se de uma máquina enorme, colocada numa sala do tamanho de uma quadra de tênis, usava cerca de 18 000 válvulas, 70 000 resistores, 10 000 capacitores, 7 500 relés e chaves, pesava 30 toneladas e consumia 140 000 watts de potência, o que fazia pulsarem as luzes das imediações da Universidade da Pensilvania. A programação ENIAC era feita por meio de chaves e pela colocação de vários fios em soquetes, o que tornava difícil mudar-se o programa. Evoluiu-se então para os computadores com programa armazenado, com base no revolucionário relatório feito no Departamento de Segurança do Exército americano em 1946 por Arthur W. Burks, Herman H. Goldstne e por John Von Neumann, intitulado "Preliminary discussion of the logical design of an eletronic computing instrument", quando ficaram conhecidos os conceitos das "máquinas de Von Neumann", utilizados até recentemente como básico no projeto de computadores digitais.

Sem dúvida, desde o aparecimento do ENIAC (desativado em outubro de 1955), tivemos fantásticos desenvolvimentos tecnológicos como a substituição das válvulas eletrônicas por

transistores discretos e destes por circuitos integrados, o que indicaram fases de desenvolvimento, conhecidas como primeira, segunda e terceira gerações etc.

Os computadores de *primeira geração* (1946-1956) utilizavam válvulas a vácuo e a memória era de tubos de raios catódicos (CRT) eletrostáticos ou de linhas de atraso, principalmente de mercúrio, caracterizada pela sua pequena capacidade de armazenamento (1K a 4K palavras). No final desta geração surgiram as memórias de núcleo de ferrite.

Já nos computadores de *segunda geração* (1956-1963) houve a introdução do transistor discreto, montado juntamente com outros componentes (resistores, capacitadores) em placas de circuito impresso. Houve também nesta época a separação entre computadores tipo "comercial", caracterizado por ser mais lento, sem facilidade de utilização de números reais ou ponto flutuante e "científico" que era mais veloz, com facilidade de uso de números reais.

Em 1964, quando se iniciava a *terceira geração*, começou-se a utilizar circuitos integrados compostos por transistores montados em uma pastilha, embora ainda em pequena escala, ou seja, poucos componentes por pastilha de circuito integrado ("chip").

Também a separação entre computador comercial e científico foi substituída por uma "família" de computadores, que ofereciam opções de computadores de baixa performance, conseqüentemente baixo custo, até modelos de alta performance e custo.

Como os equipamentos possuiam uma arquitetura interna (instruções, endereçamentos, tipos de interrupção, entrada/saída de dados formulados) houve a introdução da multiprogramação, que são vários programas de vários usuários simultaneamente processados pelo mesmo computador, sendo que cada usuário aparentemente "vê" o computador como se este estivesse processando apenas o seu programa.

As aplicações do computador são praticamente ilimitadas, principalmente nas áreas científicas, de controle e de informação.

Na área informativa os computadores são utilizados em empresas no controle financeiro, administrativo e de pessoal; em órgãos oficiais nas áreas de tributação federal e planejamento econômico, nas reservas em hotéis, aviões, navios etc.; na Medicina, no controle de internações nos hospitais; em diagnósticos; nas escolas, em controle acadêmico, freqüência e notas dos alunos; em bancos, como controle das contas dos clientes, cadernetas de poupança, investimentos etc.; no comércio, em controle de estoques, consumos e compras efetuadas, análise e previsão de vendas etc.

Na área científica os computadores também são largamente utilizados como na Engenharia para cálculos de reatores químicos, projetos de tubulações para gases e líquidos, análise e simulações de circuitos eletrônicos e cálculos de geração de energia elétrica em hidroelétricas e termoelétricas, distribuição de energia elétrica, cálculos de estruturas e fundações de edifícios, projetos de equipamentos mecânicos, ferramentas e máquinas etc.

Na matemática podem ser aplicados em análises estatísticas, simulações, cálculos com números complexos, raízes de funções, matrizes etc.

Na área de controles de processos, o computador é utilizado para fazer o controle, a supervisão e a monitoração de um processo industrial qualquer, recebendo de uma ou mais variáveis, processando-a e gerando novos dados, que atuarão sobre o processo.

Os controles de processos encontram aplicações nas áreas químicas onde podem ser controladas e monitoradas diversas variáveis como temperatura, vazão, rpm de motores, nível pressão etc.; em controle de tráfego, abrindo ou fechando semáforos, por maior ou menor intervalo de tempo, dependendo da demanda de veículos; controle de tráfego de trens e metrôs etc.

Existem ainda muitas outras aplicações, não citadas aqui, para utilização de computadores como jogos e diversões eletrônicas, e os polêmicos "robôs" aplicados em indústrias automobilísticas, que estariam substituindo pessoas nos trabalhos de rotina. Vê-se, portanto, a importância do domínio e aprendizado de uma linguagem de programação, uma vez que, toda e qualquer pessoa, por mais desinformada, ou cética que seja, de usar um computador, obrigatoriamente está, mesmo sem saber, fazendo uso de um, quer quando vá ao banco, quando faz uma viagem, quando vai ao seu médico, escola, emprego, ou mesmo a uma loja fazer uma compra.

SUMÁRIO

CAPÍTULO 1

CONCEITOS EM PROCESSAMENTO DE DADOS

1. DEFINIÇÕES

Quando temos que resolver um problema, necessitamos de informações sobre esse problema.

Essas informações quando quantificáveis, isto é, quando podem ser convertidas num quantidade bem definida, ou ainda em um nome (de pessoa, empresa etc.), nos fornecerá um Dado para a resolução de um problema.

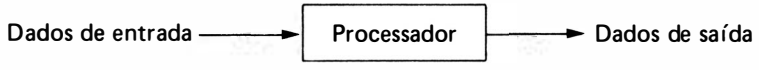
1.1. Dado

Conjunto de números ou letras que representam uma informação.

Estes dados disponíveis para a resolução de um problema, formam um conjunto inicial que operando-se ou manipulando-se conseguem-se outros dados que são a solução do problema

1.2. Processamento de Dados

Operações que se praticam com dados de entrada para se conseguir outros dados de saída.



O processamento de dados pode ser de 3 tipos:

1) Manual — quando é feito exclusivamente pelo homem.

2) Semi-automático — quando o homem já tem o auxílio de uma máquina para ajudá-lo como por exemplo, uma calculadora.

3) Automático — quando a transformação de dados de entrada em dados de saída foi feito só por uma máquina.

1.3. Computador

É a máquina capaz de processar dados automaticamente.

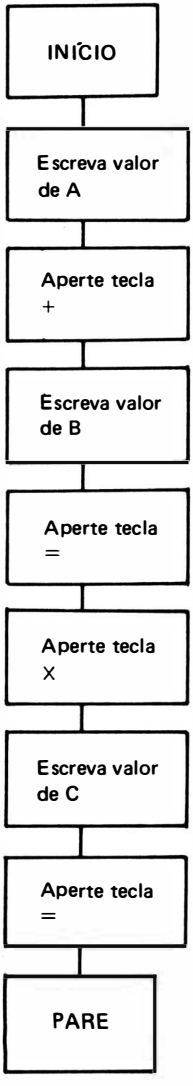
1.4. Programa

Seqüência de operações armazenadas no computador que agem sobre os dados de entrada produzindo dados de saída.

Para se fazer um programa é necessário subdividir o problema em várias etapas fáceis de fazer. Estas etapas fáceis de fazer são chamadas Instruções.

Exemplo de um Programa: Supondo-se um operador que nunca trabalhou com calculadora, vamos propor um problema simples que deva ser executado várias vezes:

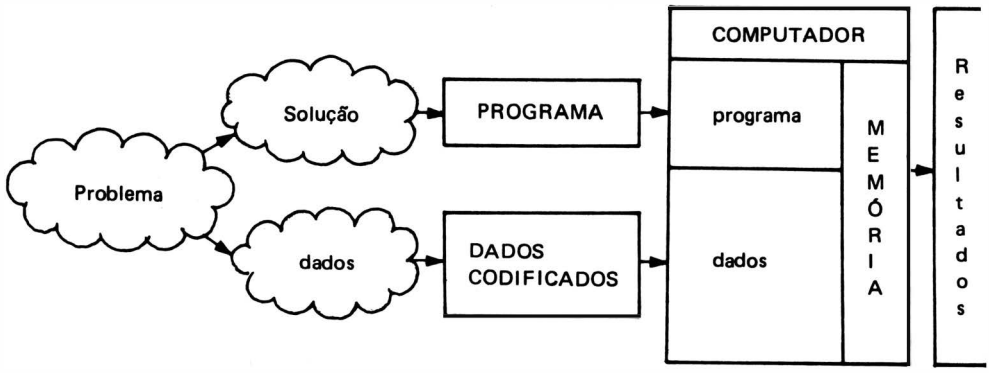
Problema: (A + B) X C



O problema neste caso foi subdividido em instruções que um operador qualquer consegue executá-lo.

Existem, na prática, vários outros exemplos de subdivisão de problemas em etapas, como por exemplo, para assar um bolo.

Esquema para resolução de um problema via computador:



A partir da existência do problema, calculamos a solução que pode ser uma ordenação qualquer devendo ser transformada num programa e enviada, então, à memória do computador O problema também nos fornecerá dados que serão codificados na forma que o computador entende e armazenados na sua memória. Com o programa e os dados será fornecido o resultado na saída do computador.

A união de um programa bem feito, isto é, uma seqüência de instruções claras e lógicas, com a automaticidade do computador, é que torna eficiente o processamento de dados eletrônicos.

Há uma mística de que o computador é uma máquina inteligente (chamado algumas vezes até de Cérebro Eletrônico), o que é um engano, o computador é um eficiente seguidor de instruções, com a vantagem de repetir qualquer seqüência de instruções, quantas vezes for necessário, sem o perigo de errar por "fadiga".

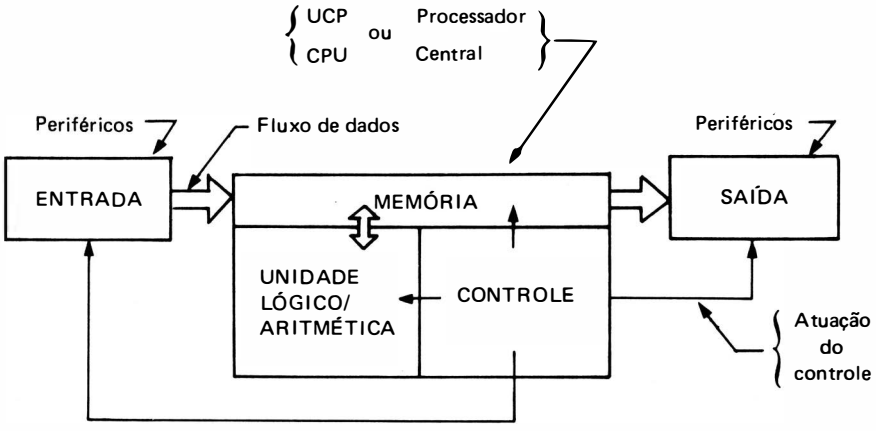
O computador também não pode ser chamado de inteligente, pois não toma decisões sozinho. Ele apenas segue uma lista de instruções fornecidas pelo programador. Se estas instruções forem lógicas e corretas, o computador executará o programa com sucesso, caso contrário, certamente recairá em algum erro, não chegando a resultado algum, ou o que é o pior, chegando a um resultado incorreto, de onde é comum a frase — O computador errou!

Resumindo, o computador executará exatamente o que lhe foi mandado fazer, em um tempo muito curto, sem considerar se o que está fazendo é errado ou não.

2. ESTRUTURA DE UM COMPUTADOR

O computador pode ser representado, genericamente, pelas suas unidades funcionais básicas, conforme é mostrado a seguir. Tal estrutura foi introduzida, inicialmente, em 1946 por John Von Neumann quando ficaram conhecidas as "máquinas de Von Neumann", e utilizada até recentemente no projeto de computadores digitais, embora a arquitetura interna varie bastante entre si.

O computador pode ser representado como sendo composto por duas unidades básicas: Unidade Central de Processamento (que contém a memória principal, Unidade Lógico-Aritmética e o Controle) e Periféricos (que são as diversas unidades de entrada e saída), assim chamados pois ficam em torno da UCP.



2.1. Unidades de Entrada

Servem para introduzir no computador os dados e o programa necessário para a resolução do problema.

Ex.: leitora de cartão perfurado
leitora de fita magnética
leitora de disco rígido/flexível (disquete)
leitora de fita de papel perfurado
máquina de escrever (console)

2.2. Unidades de Saída

Servem para exibir valores, mensagens e instruções armazenadas na memória.

Ex.: além dos apresentados para Entrada (gravadora de cartões, disco, fita etc.)
impressora
terminal de vídeo

2.2.1. Veículos de Entrada/Saída

É o meio pelo qual se consegue a entrada ou saída de dados ou programas no computador. É o suporte físico destas operações que para o caso da leitora/gravadora de cartões perfurados, é o cartão perfurado, para a impressora é o papel de relatório impresso etc.

2.3. Unidades de Memória

Servem para armazenar as instruções do programa, os dados, os resultados intermediário e os finais. Cada unidade de memória tem um único endereço.

2.4. Unidade Central de Processamento (UCP)

Também chamada de processador Central ou Central Processing Unit (CPU) é a parte dc computador que contém a unidade de memória, a unidade de controle e a unidade lógico aritmética.

2.5. Unidade Lógico-Aritmética

Serve para efetuar as operações aritméticas e lógicas, como por exemplo comparaçãc entre valores.

2.6. Unidade de Controle

Serve para controlar a execução do programa. Faz a coordenação das diversas operaçõe durante a execução do programa e controla a transferência de dados entre as diversas unidades.

3. OPERAÇÕES ELEMENTARES EFETUADAS PELO COMPUTADOR

Já dissemos anteriormente que para fazermos um programa é necessário subdividir-se c problema em operações fáceis de fazer. Estas são as operações básicas do computador:

1) *Ler* o conteúdo da Entrada
   - Este valor lido na entrada é automaticamente atribuído a uma localidade de memória.

2) *Gravar* numa localidade de memória
   - um dado de entrada
   - o conteúdo
   - o resultado de operações
   - um valor constante

3) *Efetuar operações matemáticas* entre:
   - valores dados na Entrada
   - valores constantes
   - valores de unidades de memória
   - resultados de outras operações aritméticas

4) *Escrever*
   - uma frase na saída
   - um valor de uma localidade de memória na saída

5) *Comparar* dois valores entre si
   - Verifica se um valor é maior, menor ou igual a um outro valor

6) *Seguir* e executar uma determinada seqüência de operações.

CAPÍTULO 2

LINGUAGENS DE PROGRAMAÇÃO

1. TIPOS

Através de uma linguagem é que podemos nos comunicar com o computador e nos fazer entendidos por ele.

Trata-se de um conjunto de códigos, regras, vocabulários, que passam por análises sintáticas, semânticas e léxicas, sendo que qualquer violação destas regras fará com que o computador deixe de entender a nossa instrução.

Podemos basicamente distinguir 3 tipos de linguagens de programação:

- Linguagem de máquina
- Linguagem simbólica
- Linguagem de alto nível

1.1. Linguagem de Máquina

É o método de programação diretamente na forma binária. Depende do projeto lógico do computador. Trata-se de uma linguagem praticamente impossível de ser utilizada, mesmo por programadores experientes, pois foi a primeira linguagem utilizada para programar computadores, devido a sua inerente dificuldade de estar mais próxima à máquina do que ao programador (para uma pessoa é muito difícil trabalhar com colunas de zeros e uns), além do que, um mesmo programa não serviria em outras máquinas.

1.2. Linguagem Simbólica

Para facilitar a programação, surgiu então a linguagem simbólica, a qual era convertida para a linguagem de máquina por outro programa, chamado Montador Assembler.

No Assembler, os códigos da linguagem de máquina são substituídos por códigos mnemônicos (abreviações que lembram a função da instrução.

Ex.: STO para Storage, LDA para Load Acumulator, ADD para Add etc.), tornando-se mais fácil que a programação numérica.

Exemplo de programa escrito em Assembler para fazer soma, com subtotal para ur microcomputador Olivetti – tipo AUDIT 5:

```
TSP 0                    Soma em S0
LRZ 2                    Subtotal em S1
LAB 255
NL '1', 1
HT 150
KS 15, 0
FI 8
F2 16
AR 1, 0
PEU 15, 2
B255
NL '1', 1
HT 155
LR 0, 1
PEU 15, 2
B255
NL '1', 1
NT 158
PEU 15, 2
LZ 1, 0
B255
```

1.3. Linguagem de Alto Nível

A Linguagem Assembler simplificou bastante a programação, mas ainda era dependent da estrutura dos computadores. Com a finalidade de se deixar a notação da linguagem mai próxima do elemento humano do que da máquina, e fazer-se com que um programa passad em uma determinada máquina fosse aceito por outras máquinas, de outros fabricantes ou não é que surgiram as Linguagens Automáticas ou de Alto Nível.

Elas propunham também que não se precisasse conhecer as características da máquina n qual seria passado o programa, tais como número de registradores, tipo de instruções etc onde uma instrução em linguagem de alto nível se convertesse em várias instruções de máquina.

A primeira linguagem de alto nível foi o FORTRAN elaborada para o IBM-704, em 1956 para ser utilizada por cientistas.

Para outras finalidades foram surgindo outras linguagens como COBOL (Common Busines Oriented Language) orientada para resolver problemas de uso comercial; BASIC (Beginner' All-purpose Symbolic Instruction Code) sendo a mais popular das linguagens, introduzida a partir de 1963 com a finalidade didática; APL (A Programming Language) criada em 1960 pel IBM para funcionar em tempo compartilhado (time-sharing), utilizando operadores complexos que permite ao programador expressar idéias complicadas numa única linha; ADA – linguagem desenvolvida a partir de 1975 para o Departamento de Defesa dos Estados Unidos coordena

aplicações necessárias ao Exército, Marinha e Força Aérea. Chamada a Linguagem "verde" foi rebatizada de ADA em homenagem à primeira programadora Lady Ada Augusta Byron, Condessa de Louelance, esposa do inglês Lord Byron. Será a linguagem padrão dos Estados Unidos a partir de 1985.

Existe ainda uma série quase infindável de linguagens e dialetos, citando como exemplos: PL 1, ALGOL, ALGOL 68, DOS, PASCAL, LISP, PILOT, SNOBOL etc.

2. LINGUAGEM BASIC

2.1. Introdução

A linguagem BASIC foi de início desenvolvida no Dartmouth College, sob a orientação dos professores J. G. Kemeny e T. E. Kurtz. A finalidade foi elaborar uma linguagem que fosse potente e fácil de aprender. O BASIC inicial foi sendo aperfeiçoado de tal modo, que hoje é considerada suficientemente poderosa para aplicações comerciais. É uma linguagem orientada para a resolução tanto de problemas científicos como comerciais, pois pode trabalhar tanto com fórmulas matemáticas diretamente, quanto com cadeias de caracteres alfanuméricos.

O nome BASIC vem das iniciais:

B eginner's
A ll-purpose
S ymbolic
I nstruction
C ode

Existem várias versões do BASIC atualmente disponíveis embora em 1978 o ANSI (American National Standard Institute) tenha padronizado um subconjunto essencial desta linguagem.

2.2. Caracteres Usados na Linguagem BASIC

Alfabéticos – letras maiúsculas de A-Z
Numéricos – 1, 2, . . . , Ø
Especiais – ( ) . , ; " + – = * / = $ ≠ #
Espaço em branco – ß (representação)

2.3. Variável

É uma letra ou uma letra seguida por uma cadeia de letras ou números, que podem assumir diferentes valores numéricos no decorrer do programa. Formalmente seguindo a notação BNF temos:

< variável > = < letra > |
< letra > < letra ou número >

As variáveis são endereços de posições de memórias identificadas simbolicamente pelo nome da variável.

Ex.: A, X, I, M
A1, X9
VOLT, TEMPO

se for feito: VOLT = 110

entende-se que na posição de memória associada à variável VOLT estará armazenado o valor 110.

2.3.1. Variáveis BASIC

Para o endereçamento de uma determinada posição de memória utilizamos as variáveis. Estas variáveis podem ser de dois tipos: numéricas e alfanuméricas, e seguem as seguintes regras para uma formação:

1) O primeiro caracter tem que ser obrigatoriamente uma letra.

2) Dependendo do computador utilizado, pode ter de 2 a 64 caracteres.

3) Não é permitido o uso de símbolos especiais no nome da variável.

Ex.: A, X, X1, ALFA = 10. 5
COEFICIENTEDOPOLINOMIO

Contra-exemplo: Não são nomes de variáveis:
1X — pois não inicia por letra
A — pois não é permitido o uso de símbolos especiais
(A) — idem ao anterior
A, B — idem ao anterior

OBS.: Em BASIC usa-se o ponto decimal no lugar da vírgula.

4) As variáveis alfanuméricas são seguidas de $ (string)

Ex.: A$, X$, ALFA$, A$ = "PLATINA"

Contra-exemplo: Não são variáveis alfanuméricas:
ALFA—1$ pois usa símbolo especial
$TAL o caracter $ deve vir no final
3X$ pois não inicia por letra

CAPÍTULO 3

INSTRUÇÕES BASIC

1. ENTRADA

A instrução de entrada tem por finalidade introduzir no computador valores numéricos ou alfanuméricos. Estes valores serão atribuídos a variáveis relacionadas à frente das instruções, que têm a seguinte forma geral:

nl INPUT $v_1, v_2, v_3 \ldots$

onde

nl	indica o número da linha.
v_1, v_2, v_3	são variáveis às quais serão atribuídos valores, digitados pelo programador.

Ao encontrar esta instrução o processamento é interrompido, aparecendo na tela um ponto de interrogação, ou outro símbolo, e o computador aguardará indeterminadamente a entrada dos valores correspondentes às variáveis. Há computadores que aceitam apenas uma única variável para entrada de valores. Neste caso, deve-se usar uma seqüência de instruções INPUT para obter o mesmo efeito da instrução INPUT de multivariáveis.

Ex.: 10 INPUT A
O valor digitado pelo operador será atribuído à localidade de memória associada à variável A.

50 INPUT A, B$, X
O primeiro e o último valor digitados serão atribuídos às variáveis A e X, respectivamente, e necessariamente têm que ser números, pois estas variáveis são numéricas. O segundo valor será atribuído à variável B$ que é alfanumérica e, portanto, será aceita qualquer cadeia de caracteres. Esta instrução é equivalente a:

```
10  INPUT  A
20  INPUT  B$
30  INPUT  X
```

para computadores que não aceitam a instrução INPUT multivariáveis.

Existem ainda computadores que aceitam uma frase na instrução INPUT.

Ex.: 15 INPUT "ENTRE COM O VALOR DE A"; A
Esta frase aparecerá na tela do computador no lugar do ponto de interrogação, indicando de maneira mais clara qual o tipo de dado que deve ser introduzido, o que é muito útil por exemplo, quando temos vários valores para introduzir no computador.

2. SAÍDA

Esta instrução tem por finalidade a apresentação em vídeo ou em um display de uma linha, o conteúdo das variáveis especificadas, resultado do cálculo de expressões e textos fornecidos entre aspas. A forma geral desta instrução é:

nl PRINT "mensagem"; v_1, v_2, . . . , e

onde

nl	indica o número da linha
"mensagem"	é um texto opcional que deve ser fornecido entre aspas
v_1, v_2 . . .	lista de variáveis numéricas ou alfanuméricas
e	indica uma expressão BASIC que será avaliada e exibida

Ex.: 50 PRINT X
Exibe o conteúdo da variável X

40 PRINT "NOME = "; N$
NOME = FULANO DE TAL ("FULANO DE TAL" indica o conteúdo da variável N$)

Quando um ponto e vírgula é utilizado no final de ums instrução PRINT, a próxima instrução PRINT continuará de onde a última parou, anulando o retorno do cursor e não avançando para a próxima linha:

Ex.: 10 PRINT "VOU IMPRIMIR NA MESMA";
20 PRINT" LINHA";
Provoca a impressão de:
VOU IMPRIMIR NA MESMA LINHA

A instrução PRINT pode também avaliar o resultado de uma expressão:

Ex.: 10 PRINT 15 + 5 − 124
Resulta na impressão do valor:
− 94

Porém, este valor não fica armazenado em nenhuma localidade de memória.

OBS.: O computador pode operar como uma simples calculadora, utilizando-se a instrução PRINT juntamente com a expressão que queremos avaliar sem o uso do número de linha. Este modo de operação é chamado MODO DIRETO. Quando usamos o número de linha temos o MODO PROGRAMADO.

Para diferenciar o uso do terminal de vídeo da impressora, quando desejamos a saída por um ou por outro dispositivo, utilizamos uma instrução levemente diferente, que é a instrução:

nl LPRINT "mensagem"; v1, v2, . . .

Para o acionamento da impressora, esta instrução tem o mesmo efeito da instrução PRINT.

```
40   PRINT   "RESULTADOS FINAIS"
```

Exibe a frase RESULTADOS FINAIS

Podemos fazer a separação dos itens a serem apresentados por vírgula (,), ou por ponto e vírgula (;). Se for utilizada a vírgula, a separação dos itens será feita em "zonas de impressão" de forma automática.

Para os pequenos micros que seguem a lógica SINCLAIR tipo TK, CP 200 onde a tela é organizada em 24 linhas por 32 colunas, temos 2 zonas de impressão, que divide a tela em duas partes. Para micros que seguem a lógica do TRS80 tipo CP300, CP500 etc., a tela é organizada em 16 linhas por 64 colunas e temos 4 zonas de impressão, e em ambos os casos, cada zona de impressão tem no máximo 16 caracteres.

Ex.: 20 PRINT "PONTO 1", "PONTO 2", "PONTO 3", "PONTO 4" se for executada em um micro com 2 zonas de impressão, resultará em:

```
0               16              31
PONTO 1         PONTO 2
PONTO 3         PONTO 4
```

se for executada em um micro com 4 zonas de impressão, resultará em:

```
0          16          32          48          63
PONTO 1    PONTO 2     PONTO 3     PONTO 4
```

Na separação dos itens por ponto e vírgula a apresentação dos resultados serão juntos, isto é, sem nenhum espaço em branco entre itens alfanuméricos; para itens numéricos serão inseridos dois espaços em branco: um para separação natural de itens e outro para o sinal (+ ou −).

Ex.: 10 PRINT "VALOR FINAL DE X = " ; X
Resulta na impressão de:

```
VALOR FINAL DE X = 999      (999 indica o conteúdo da variável X)
```

3. DESVIO INCONDICIONAL

Esta instrução tem por finalidade fazer um desvio do fluxo normal de processamento, que é a execução na mesma ordem que estão escritas para uma outra parte do programa independente de alguma outra condição. A forma geral desta instrução é:

nl GOTO nl_1

onde

nl é o número da linha da instrução GOTO
nl_1 é o número da linha para a qual o controle será transferido.

Ex.: 10 INPUT A
20 PRINT A
30 GOTO 10

A linha 30 faz com que o fluxo do processamento seja desviado para a linha 10, para entrada de novo valor de A que será impresso na linha 20.

Convém observarmos que um programa escrito em BASIC tem todas as instruções numeradas em ordem crescente. Por questão de facilidade, recomenda-se que esta numeração seja de 10 em 10, pois no caso de necessidade de inserção de uma linha, em parte do programa já escrito, isto seria impossível se a numeração crescente fosse de 1 em 1.

4. DESVIO CONDICIONAL

Esta instrução tem por finalidade provocar o desvio do fluxo do processamento dependendo de um determinado valor ou de uma determinada comparação. Sua forma geral é:

nl IF < condição > THEN < instrução >

onde

nl	indica o número da linha
< condição >	é uma relação de igualdade entre 2 valores ou variáveis
< instrução >	é uma instrução BASIC qualquer

se a < condição > entre o IF e o THEN for verdadeira a instrução após THEN é executada.

se a < condição > for falsa o fluxo do processamento segue para a instrução seguinte do programa.

4.1. Operadores Relacionais

A condição pode ser definida pelas seguintes relações de igualdade:

Símbolo	Significado
=	igual
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual
<> ou ≠	diferente

Ex.: 20 IF A < X THEN GOTO 150
indica que se o valor da variável A for menor que o valor da variável X o fluxo do processamento deve desviar para a instrução de nº 150.

40 IF X ≠ 0 THEN PRINT X
indica que se X for diferente de zero o valor de X deve ser impresso.

4.2. Operadores Lógicos

Além dos operadores relacionais, podemos ter operadores lógicos que relacionam duas (ou mais) condições.

AND – Este operador faz a operação lógica "E" entre duas (ou mais) condições que segue a seguinte tabela verdade:

condição 1	condição 2	resultado
falsa	falsa	falso
verdadeira	falsa	falso
falsa	verdadeira	falso
verdadeira	verdadeira	verdadeiro

seu uso geral é o seguinte:

nl IF <condição 1> AND <condição 2> THEN <instrução>

ou seja, a instrução após o THEN será executada se ambas as condições 1 e 2 forem verdadeiras.

Ex.: 100 IF A > X AND A < C THEN GOTO 500
ocorrerá o desvio para a instrução de nº 500 se o valor de A for maior que X e E se o valor de A for menor que C.

OR – Este operador faz a operação lógica "OU" entre duas condições e segue a seguinte tabela verdade:

condição 1	condição 2	resultado
falsa	falsa	falso
falsa	verdadeira	verdadeiro
verdadeira	falsa	verdadeiro
verdadeira	verdadeira	verdadeiro

seu uso geral é o seguinte:

nl IF < condição 1 > OR < condição 2 > THEN < instrução >

ou seja a instrução após o THEN será executada se pelo menos uma das condições 1 ou 2 for verdadeira.

Ex.: 300 IF A > B OR A < C THEN GOTO 100
ocorrerá o desvio para a instrução 100 quando o valor de A for maior que o valor de B OU valor de A for menor que o valor de C.

NOT – Este operador faz a operação lógica "NÃO" para uma condição seguindo a seguinte tabela verdade:

condição	resultado
verdadeira	falso
falsa	verdadeiro

seu uso geral é o seguinte:

nl IF NOT < condição > THEN < instrução >

ou seja a instrução após o THEN é executada sempre que a condição for NÃO verdadeira, portanto, falsa.

EX.: 50 IF NOT B = 0 THEN PRINT B
imprimirá o valor de B se este for NÃO igual a zero; portanto, se for diferente de zero equivalendo a:

50 IF B ≠ 0 THEN PRINT B

Dependendo do Microcomputador utilizado, a instrução de desvio condicional pode variar um pouco em sua forma sintática. Por exemplo: Existem versões em que a palavra THEN pode ser ignorada e a instrução seguinte é sempre um GOTO da forma:

nl IF < condição > GOTO nl_1

Outras versões admitem a palavra THEN e se a instrução após o THEN for um GOTO, o nome da instrução pode ser ignorado, aparecendo apenas o número da linha para onde o desvio será feito:

nl IF < condição > THEN nl_1

Entretanto, a forma mais poderosa da instrução de desvio condicional é a que tem a seguinte forma geral:

nl IF < condição > THEN < instrução 1 > ELSE < instrução 2 >

Neste caso se a condição for verdadeira será executada a instrução 1, caso contrário (ELSE) será executada a instrução 2, ou irá para a próxima instrução do programa uma vez que o ELSE é opcional.

5. PARADA

Este comando tem por finalidade indicar o fim lógico do programa, encerrando o processamento na linha em que for encontrado. Sua forma geral é:

nl STOP	ou	nl END

Ex.: 100 STOP
Ao encontrar a linha 100 o processamento é encerrado.

Algumas versões de BASIC utilizam a instrução END no lugar de STOP com o mesmo efeito. A instrução STOP informa qual a linha o programa parou, enquanto que a instrução END só encerra o programa.

6. CÁLCULO E ARMAZENAMENTO

Esta instrução tem por finalidade atribuir um valor a uma variável que pode ser numérico ou alfanumérico.

O valor a ser atribuído quando numérico, pode ser um simples constante ou o resultado do cálculo de uma expressão matemática. Se o valor for alfanumérico, isto é, uma cadeia de caracteres, a atribuição é direta, ou seja, a lista de caracteres será atribuída diretamente à variável. A forma geral da instrução aritmética é a seguinte:

nl LET v = e	para variáveis numéricas

e

nl LET v$ = "lista de caracteres"	para variáveis alfanuméricas

onde

- *nl* o número da linha da instrução
- v variável BASIC numérica
- v$ variável BASIC alfanumérica
- e expressão BASIC (semelhante a aritmética comum)
- "lista de caracteres" que deve ser fornecido entre aspas.

Ex.:
```
10 LET A = 100
30 LET X = A + 3
30 LET B$ = "INSTITUTO DE PESQUISA"
```

A instrução 10 faz com que a constante 100 fique armazenado na localidade de memória associada à variável A.

A instrução seguinte pega o conteúdo da variável A, soma a constante 3 e atribui este resultado a variável X.

A instrução 30 armazena na variável B$ a cadeia de caracteres: "INSTITUTO DE PESQUISA".

Algumas versões utilizam o LET subentendido simplificando a escrita do programa.

Ex.:
```
10 A = 100
30 X = A + 3
```

As instruções deste exemplo são idênticas à do exemplo anterior, porém sem a necessidade de se escrever a palavra LET.

A instrução aritmética da linguagem BASIC é, portanto, composta das seguintes partes:

INSTRUÇÃO ARITMÉTICA
- LET (pode ser subentendido)
- Variável
- Sinal de Igual
- Expressão BASIC

Esta instrução relaciona variáveis a expressões, calculando o valor da expressão *e* e em seguida atribuindo este valor à variável *v*. Estas expressões são escritas em BASIC de forma semelhante à aritmética comum facilitando o uso desta linguagem em problemas científicos.

6.1. Funcionamento da Instrução Aritmética

A Instrução Aritmética é executada em duas etapas distintas e independentes:

a) Cálculo do valor numérico final da expressão "e" na unidade lógico-aritmética

b) O valor calculado é transmitido à localidade de memória associada à variável "v".

Isto nos permite ter uma variável à direita e à esquerda de uma mesma instrução.

Ex.: LET I = I + 1
quer dizer
I é substituído por I + 1

portanto, a expressão I = I + 1 não corresponde a uma equação algébrica, pois o sinal = representa uma atribuição de valor e não uma identidade matemática, caso contrário teríamos:

I = I + 1
I – I = 1
0 = 1 ??
o que é evidentemente falso.

6.2. Operações Aritméticas

Os operadores aritméticos aceitos em BASIC são os seguintes:

SINAL	OPERAÇÃO	EXEMPLO
+	soma	A + 3
–	subtração	X – B
*	multiplicação	3 * Y
/	divisão	A / 2
**	potenciação	A ↑ 1.5
↑		
Λ		

6.3 Composição das Expressões BASIC

Uma expressão pode ser constituída por apenas um número, ou por uma única variável numérica, ou por uma combinação de variáveis, números, sinais de operação, funções e símbolos de agrupamento.

6.3.1. Parênteses

Na linguagem BASIC usamos como único símbolo de agrupamento o parêntese, que pode ser colocado em vários níveis. Não existem, portanto, chaves "{ }" e colchetes "[ ]" na notação BASIC.

Ex.: a expressão

$$\{3 \cdot [(2 \cdot x - 4 \cdot y) - (10 \div z)] + k\}$$

seria escrita em BASIC:

$$(3 * ((2 * X - 4 * Y) - (10/Z)) + K)$$

É muito importante que cada parêntese aberto sempre tenha o seu correspondente fechado, para se evitar confusões do tipo:

$$(A/(B + C/D)$$

podendo ser entendido como:

$$(A/(B + C)/D)$$

ou

$$(A/(B + C/D))$$

onde a falta deste parêntese faz com que o Microcomputador acuse um *erro de sintaxe,* ou seja, indica que a expressão está escrita de forma incorreta. Entretanto, pode-se deixar em uma expressão, parênteses que são visivelmente desnecessários, que esta não será invalidada.

Ex.: a expressão ((A + B) + C) possui dois pares de parênteses desnecessários, mas continua sendo avaliada corretamente. Finalmente, devemos considerar que para a linguagem BASIC, dois parênteses consecutivos não subentendem uma multiplicação.

Ex.: a expressão $(a + b)\ (c + d)$ tem que ser escrita em BASIC colocando-se o sinal de multiplicação:

$$(A + B) * (C + D)$$

6.3.2. Hierarquia de Operações

Existindo vários operadores em uma mesma expressão, estes obedecerão uma ordem de execução, pois é executada uma operação por vez. Esta hierarquia é a seguinte:

a) Expressões entre parênteses
b) Exponenciação
c) Multiplicação e divisão
d) Soma e subtração

Se não houver parênteses, a expressão é calculada a partir da esquerda para a direita, respeitando a hierarquia. Se houver vários níveis de parênteses, a expressão é calculada a partir do parênteses mais interno para o mais externo.

OBS.: Dois sinais de operação não podem aparecer juntos.

Ex.: A + B deve ser escrito A + (− B)
A ↑ − 5 deve ser escrito B ↑ (− 5)

6.3.3. Funções Biblioteca

Para cálculo de diversas funções matemáticas de maneira simplificada, existem as Funções Biblioteca, e para utilizá-las basta declarar o nome da função, seguido do argumento que se deseja calcular colocado entre parênteses. São rotinas de programação pré-elaboradas e armazenadas na memória do computador, não havendo, portanto, necessidade do programador desenvolver uma programação especial para calcular estas funções.

As Funções Biblioteca mais freqüentemente empregadas são as seguintes:

FUNÇÃO	SIGNIFICADO	ARGUMENTO	APLICAÇÃO
ABS(*x*)	$\|x\|$	real	LET Y = ABS (X)
ACS(*x*)	arc cos(*x*)	real	LET X = ACS (Y)
ASN(*x*)	arc sen(*x*)	real	LET X = ASN (Y)
ATN(*x*)	arc tg(*x*)	real	LET X = ATN (Y)
COS(*x*)	cos(*x*)	radianos	LET Y = COS (X)
EXP(*x*)	e^x	real	LET A = EXP (B)
HCS(*x*)	cos hiperbólico(*x*)	radianos	LET Y = HCS (X)
HSN(*x*)	sen hiperbólico(*x*)	radianos	LET Y = HSN (X)
HTN(*x*)	tg hiperbólico(*x*)	radianos	LET Y = HTN (X)
INT(*x*)	maior inteiro	real	LET Z = INT (Y)
LN(*x*)	ln *x* ou $\log_e x$	real	LEY Y = LN (X)
LOG(*x*)	$\log_{10} x$	real	LET W = LOG (B)
PI	fornece 3.1415927	–	LET A = PI * R ↑ 2
SGN(*x*)	se *x* for negativo → − 1 se *x* for 0 → 0 se *x* for positivo → 1	real	LET B = SGN (X)
SIN(*x*)	sen(*x*)	radianos	LET Y = SIN (X)
SQR(*x*)	$\sqrt{x}$	real	LET Y = SQR (A)
TAN(*x*)	tg(*x*)	radianos	LET Y = TAN (X)

6.3.4. Funções Derivadas

Existe ainda uma série de funções trigonométricas que podem ser facilmente derivadas das funções-padrão como por exemplo:

```
ARC SIN (X) = ATN (X/SQR (X * X + 1))
ARC COS (X) = ATN (X/SQR (X * X + 1)) + 1.5708
ARC SEQ (X) = ATN (SQR (X * X - 1)) + (SGN (X) - 1) * 1.5708
ARC CSC (X) = ATN (1/SQR (X * X - 1)) + (SGN (X) - 1) * 1.5708
ARC COT (X) = - ATN (X) + 1.5708
ARC SINH (X) = LOG (X + SQR (X * X + 1))
ARC COSH (X) = LOG (X + SQR (X * X - 1))
ARC TANH (X) = LOG ((1 + X)/(1 - X))/2
ARC SECH (X) = LOG ((SQR (X * X + 1) + 1)/X)
ARC CSCH (X) = LOG ((SGN (X) * SQR (X * X + 1) + 1)/X)
ARC COTH (X) = LOG ((X + 1)/(X - 1))/2
COT (X) = 1/TAN (X)
CSC (X) = 1/SIN (X)
SEC (X) = 1/COS (X)
COSH (X) = (EXP (X) + EXP (- X))/2
COTH (X) = EXP (- X)/(EXP (X) - EXP (- X) * 2 + 1
CSCH (X) = 2/(EXP (X) - EXP (- X))
SECH (X) = 2/(EXP (X) + EXP (- X))
SINH (X) = (EXP (X) - EXP (- X))/2
TANH (X) = - EXP (- X)/(EXP (X) + EXP (- X)) * 2 + 1
```

7. COMENTÁRIOS

Quando fazemos um programa extenso é necessário introduzir observações a respeito do que cada parte do programa faz, do que cada variável do problema representa, quem foi o programador, data de início e término de programação etc. Resumindo: é necessário, sempre, documentar-se um programa.

A instrução que tem esta finalidade tem a seguinte forma geral:

```
nl REM frase
```

onde

- *nl* é o número da linha
- frase é uma lista de caracteres, que será o nosso comentário, sendo desnecessário o uso de aspas.

Ex.:

```
 10 REM PROGRAMA FEITO POR JORGE TREVISAN
 50 REM A VARIAVEL V REPRESENTA A VELOCIDADE
100 REM SAIDA DOS RESULTADOS
```

Convém observarmos que este comentário é exclusivo para o programador, aparecendo apenas na listagem do programa, e nunca na saída de resultados do programa.

A instrução REM vem do inglês REMark, podendo ser colocada em qualquer parte do programa, pois quando o programa é executado, esta instrução é simplesmente ignorada.

8. EXERCÍCIOS RESOLVIDOS

1º) Escrever as expressões usando a notação BASIC

a) $ax + by + cz \longrightarrow$ A * X + B * Y + C * Z

b) $\frac{a}{b} + \frac{c}{d} \longrightarrow$ A/D + C/D

c) $a + 3b \longrightarrow$ A + 3 * B

d) $\frac{a \div b}{\sqrt{c}} \longrightarrow$ (A/B)/SQR (C)
(A/B)/C ↑ 0.5

e) $\frac{a}{b} x^2 + 1 \longrightarrow$ (A/B) * X ↑ 2 + 1

f) $\frac{1}{6} \pi h (3r^2 + h^2) \longrightarrow$ (1/6) * PI * H * (3 * R ↑ 2 + H ↑ 2)

g) $\cfrac{a+b}{c+\cfrac{d+e}{f+\cfrac{g+h}{x+\cfrac{y+z}{w}}}}$

(A + B)/(C + (D + E)/(F + (G + H)/(X + (Y + Z)/X)))

2º) Escreva uma declaração LET que corresponda às seguintes expressões:

a) área $= \pi r^2$ — LET AREA = PI * R ↑ 2

b) $s = s_0 + v_0 t + 1/2\,\gamma t^2$ — LET S = S0 + VO * T + 0 · 5 * GAMA * T ↑ 2

c) $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ — LET D = SQR ((X1 − X2) ↑ 2 + (Y1 − Y2) ↑ 2)

d) $f(x) = \ln|\sec x + \operatorname{tg} x|$ — LET FX = LN (ABS (1/COS (X) + TAN (X)))

obs. $\sec x = \frac{1}{\cos x}$

e) $f(x) = \log(1 + \operatorname{sen} x)$ — LET FX = LOG (1 + SIN (X))

f) $f(x) = 2e^x + 1$ — LET FX = 2 * EXP (X) + 1

g) $f(x) = e$ — LET FX = 2.718282
LET FX = EXP (1)

3º) Elaborar programa BASIC que permita calcular as raízes de $ax^2 + bx + c = 0$ com a, b, c digitados na entrada. Se não há raízes, entram novos dados sem cálculo algum, imprimindo a frase: "NÃO HÁ RAÍZES REAIS", havendo raízes imprimi-las juntamente com os valores de *a, b, c*. O encerramento ocorre quando entrar $a = 0$.

```
 10 INPUT A, B, C
 20 IF A = 0 THEN GOTO 110
 30 LET D = B * B – 4 * A * C
 40 IF D >= 0 THEN GOTO 70
 50 PRINT "PARA A = "; A;" B = "; B;" C = "; C;
"NAO HA RAIZES REAIS"
 60 GOTO 10
 70 LET X1 = (– B + SQR (D))/(2 * A)
 80 LET X2 = (– B – SQR (D))/(2 * A)
 90 PRINT "PARA A = "; A;" B = "; B;" C = "; C;
"AS RAIZES SAO: "; X1;" E "; X2
100 GOTO 10
110 STOP
```

Explicação do programa:

Na linha 10, a instrução INPUT faz com que o micro aguarde a digitação de 3 valores (A, B, C).

Para cada valor digitado ser reconhecido pelo micro, deve ser precionada a tecla ENTER.

O primeiro valor digitado será atribuído à variável A, o segundo à variável B e o terceiro à variável C.

Após o cálculo e verificação do discriminante D, haverá a impressão da mensagem na linha 50 ou o resultado das raízes na linha 90.

4º) Elaborar programa BASIC que permita exibir todos os números de 1 a 100.

```
10 LET I = 1
20 PRINT I
30 LET I = I + 1
```

```
40 IF I < = 100 THEN GOTO 20
50 STOP
```

5º) Elaborar programa BASIC que permita exibir todos os números pares de 0 a 50.

```
10 LET I = 0
20 PRINT I
30 LET I = I + 2
40 IF I < = 50 THEN GOTO 20
50 STOP
```

6º) Elaborar programa BASIC que permita exibir o fatorial (N!) de um número inteiro positivo digitado na entrada (N), com $0 < N \leqslant 33$.

```
10 INPUT "ENTRE COM O VALOR DE N"; N
20 LET FAT = 1
30 LET I = 1
40 LET FAT = FAT * I
50 LET I = I + 1
60 IF I < = N THEN GOTO 40
70 PRINT "FATORIAL DE"; N ;" = "; FAT
80 STOP
```

7º) Elaborar programa BASIC que permita fazer uma somatória de N números digitados na entrada onde o valor de N também é digitado.

```
10 INPUT "NUMERO DE PARCELAS = "; N
20 LET S = 0
30 LET I = 0
40 INPUT "ENTRE COM UMA PARCELA"; P
50 LET S = S + P
60 LET I = I + 1
70 IF I < N THEN GOTO 40
80 PRINT "SOMATORIA ="; S
90 STOP
```

8º) Elaborar programa BASIC que permita fazer uma produtória de N números digitados na entrada, onde o valor de N também é digitado.

```
10 INPUT "NUMERO DE PARCELAS = "; N
20 LET P = 1
30 LET I = 0
40 INPUT "ENTRE COM UMA PARCELA"; X
50 LET P = P * X
60 LET I = I + 1
70 IF I < N THEN GOTO 40
80 PRINT "PRODUTORIA = "; P
90 STOP
```

9º) Elaborar programa BASIC que permita calcular a média aritmética de N valores digitados na entrada, onde o valor de N também é digitado.

```
 10 INPUT "NUMERO DE VALORES = "; N
 20 LET S = 0
 30 LET I = 0
 40 INPUT "ENTRE COM UM VALOR : "; X
 50 LET S = S + X
 60 LET I = I + 1
 70 IF I < N THEN GOTO 40
 80 LET MA = S/N
 90 PRINT "MEDIA ARITMETICA = "; MA
100 STOP
```

10º) Elaborar programa BASIC que permita calcular a média geométrica de N valores digitados na entrada, onde o valor de N também é digitado.

$$XMG = \sqrt[n]{X1 \cdot X2 \ldots . Xn} = \text{média geométrica}$$

```
 10 INPUT "NUMERO DE VALORES = "; N
 20 LET P = 1
 30 LET I = 0
 40 INPUT "ENTRE COM UM VALOR : "; X
 50 LET P = P * X
 60 LET I = I + 1
 70 IF I < N THEN GOTO 40
 80 IF P < 0 THEN STOP
 90 LET MG = P ↑ (1/N)
100 PRINT "MEDIA GEOMETRICA = "; MG
110 STOP
```

11º) Elaborar programa BASIC que permita achar e exibir o maior dos valores digitados na entrada, de uma lista de N valores, onde N também é digitado na entrada.

```
 10 INPUT "QUANTOS VALORES VAI DIGITAR?"; N
 20 INPUT "ENTRE COM VALOR"; XMAX
 30 LET I = 1
 40 INPUT "ENTRE COM VALOR "; X
 50 LET I = I + 1
 60 IF X < = XMAX THEN GOTO 80
 70 LET XMAX = X
 80 IF I < N THEN GOTO 40
 90 PRINT "O MAIOR VALOR DIGITADO FOI : "; XMAX
100 STOP
```

12º) Elaborar programa BASIC que permita achar e exibir o maior e o menor dos valores digitados na entrada, de uma lista de N valores, onde N também é digitado na entrada.

```
10 INPUT "QUANTOS VALORES VAI DIGITAR "; N
20 INPUT "ENTRE COM VALOR "; X
30 LET I = 1
40 LET XMAX = X
50 LET XMIN = X
60 INPUT "ENTRE COM VALOR "; X
70 LET I = I + 1
80 IF X < = XMAX THEN GOTO 110
90 LET XMAX = X
100 GOTO 130
110 IF X >= XMIN THEN GOTO 130
120 LET XMIN = X
130 IF I < N THEN GOTO 60
140 PRINT "XMAX = "; XMAX
150 PRINT " XMIN = "; XMIN
160 STOP
```

9. EXERCÍCIOS PROPOSTOS

1º) Escreva as Expressões BASIC correspondentes a:

a) $12x - 5$

b) $9xz + 4\,(x + y)$

c) $\dfrac{4a}{b} - \dfrac{5c}{7d}$

d) $\dfrac{m}{n} - \sqrt{a'} + 3$

e) $\dfrac{2x^2 - 3x + 5}{4x + 1}$

f) $\dfrac{2a + b}{K + \dfrac{8J - 5}{L + 2M}}$

2º) Escreva a instrução de atribuição completa para as seguintes equações:

a) $Y = 3x + \dfrac{1}{5}$

b) $K = \dfrac{a^2 + b^2}{ab}$

c) $I = \dfrac{U}{\sqrt{R^2 + X^2}}$

d) $FI = (4x^3 - 2x^2 + 5 - 1)\,(3x^2 - 1)$

e) $M \dfrac{ab + cd + ef}{\sqrt{a^2 + c^2 + e^2} * \sqrt{b^2 + d^2 + f^2}}$

3º) Elaborar programa BASIC que permita exibir todos os números ímpares de 1 a 101.

4º) Elaborar programa BASIC que permita calcular o FATORIAL de 37.

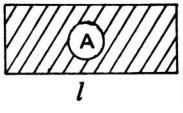
5º) Elaborar programa BASIC que permita calcular a média harmônica de 2 valores digitados na entrada.

OBS.: Média Harmônica

$$MH = \frac{2}{\frac{1}{A} + \frac{1}{B}}$$

6º) Elaborar programa BASIC que permita calcular a área de um retângulo, sendo digitados na entrada seus 2 lados.

$A = c * l$



CAPÍTULO 4

FLUXOGRAMA OU DIAGRAMA DE BLOCOS

1. INTRODUÇÃO

Em problemas complexos, é necessário fazermos um esquema gráfico das etapas em que a solução é obtida, antes de iniciarmos a programação em BASIC, ou outra linguagem qualquer.

Este algoritimo, que é uma lista de instruções para resolver um problema passo a passo, representado graficamente, é chamado de Fluxograma ou Diagrama de Blocos.

As operações são representadas num fluxograma por Símbolos Padronizados, no interior dos quais é detalhada a operação a ser efetuada. Cada Símbolo ou Bloco Padronizado indica um tipo de operação a ser realizada externamente (entradas e saídas), ou internamente (cálculo e armazenamentos, desvios condicionais ou incondicionais e paradas), evitando assim explicações escritas a respeito da operação que será executada.

O Fluxograma tem por finalidade a visualização do método escolhido para a resolução do problema de modo total, facilitando a correção de erros antes da programação, bem como é de importante ajuda na documentação do programa, facilitando a sua utilização e entendimento por outras pessoas que não participaram da programação.

Os Símbolos Padronizados que usamos para fazer o desenho do fluxograma são apresentados a seguir.

2. SÍMBOLOS PARA OPERAÇÕES EXTERNAS

As operações de Entrada e Saída de valores podem ser representados de forma genérica por um dos dois símbolos da figura abaixo:

Para fazermos a diferenciação entre Entrada e Saída, escrevemos no interior do símbolo a operação que será efetuada, além dos nomes das variáveis que serão manipuladas.

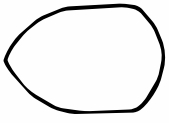
Ex.:

o conteúdo da variável A será exibido na saída

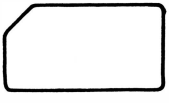
o valor apresentado na entrada será transmitido à variável M

Os computadores, invariavelmente, são providos de periféricos para entrada e saída dedicados a tarefas específicas, e é muito conveniente especificá-los quando desenhamos o fluxograma, usando os seguintes símbolos especiais:

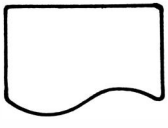
para entrada por teclado



para saída por terminal de video



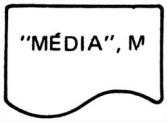
para entrada por cartões perfurados



para saída por relatório impresso

Como para os símbolos de E/S (Entrada e Saída) genéricos, escrevemos no interior do símbolo o nome das variáveis manipuladas.

Ex.:

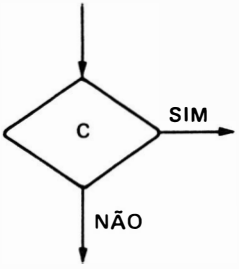


representa a saída por relatório impresso da frase "MÉDIA" e do conteúdo da variável M.

3. SÍMBOLOS PARA OPERAÇÕES INTERNAS

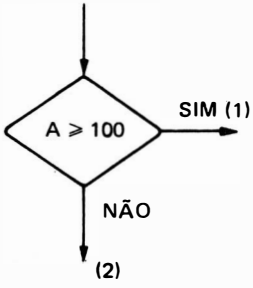
3.1. Desvio Condicional

Esta operação é representada por um losango, no interior do qual indica-se a condição a ser testada. Dos seus vértices saem os caminhos orientados de acordo com o resultado da condição C ser verdadeiro ou falso.



Estes vértices não têm posição definida para as saídas de SIM ou NÃO (ou verdadeiro ou falso)

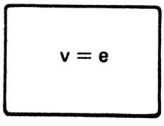
Ex.:



Indica a operação de decisão que quando A for maior ou igual a 100 deve ser seguido o caminho 1, caso contrário, deve ser seguido o caminho 2.

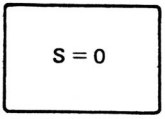
3.2. Cálculo e Armazenamento

Esta operação de atribuição é representada por um retângulo, no interior do qual indica-se o cálculo ou atribuição a ser efetuado:

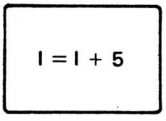


A variável **v** indica a localidade de memória onde será armazenado o resultado de expressão **e.**

Ex.:



Atribui o valor zero a variável S.



Faz a soma do conteúdo da variável I com a constante 5 a atribui este valor a variável I.

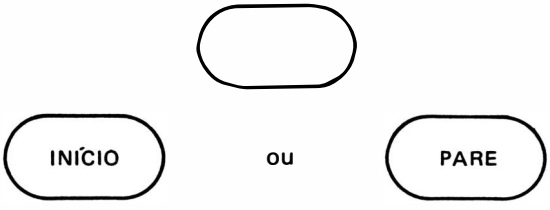
3.3. Desvio Incondicional

Esta operação é representada simplesmente por uma linha orientada que liga a última instrução com o ponto do programa onde deve ser reiniciado ou onde deve haver sua continuação.

3.4. Terminais

São símbolos que representam o início e o final do programa, representados pela figura abaixo, no interior da qual é colocada a respectiva operação.

Ex.:



3.5. Interligações

Esta operação é representada pelo círculo, servindo para unir dois trechos de programa que estejam separados.

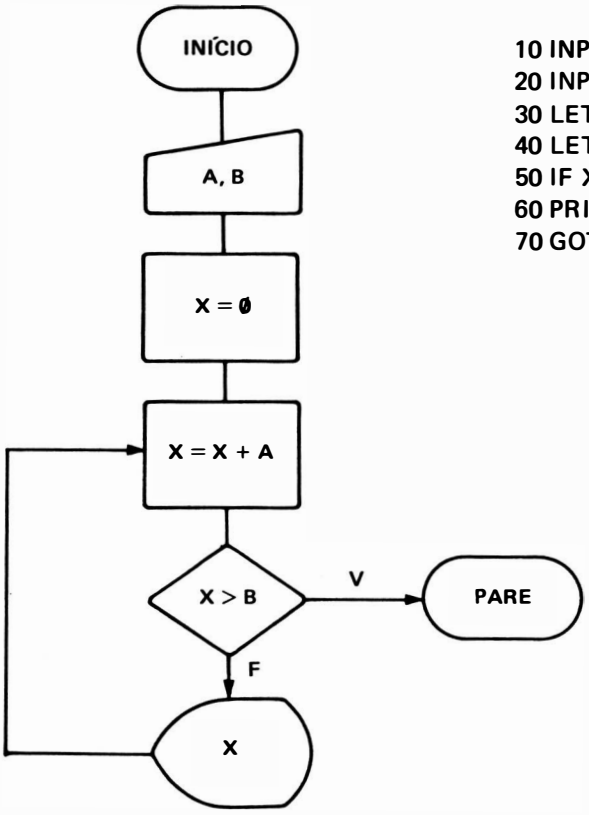
No interior do círculo colocamos um número qualquer

4. EXERCÍCIOS RESOLVIDOS

1º) Fazer um fluxograma e programa BASIC que ache e exiba os múltiplos de um número a menores ou iguais a um número limite *b*, com *a* e *b* digitados na entrada.

$$(0 < a < b)$$

FLUXOGRAMA



PROGRAMA

```
10 INPUT "A = ", A
20 INPUT "B = ", B
30 LET X = 0
40 LET X = X + A
50 IF X > B THEN STOP
60 PRINT X
70 GOTO 40
```

Supondo-se que seja digitado $a = 2.1$ e $b = 12.2$ teremos as seguintes situações na memória e na saída:

SAÍDA

```
2.1
4.2
6.3
8.4
10.5
```

MEMÓRIA

A	2.1
B	12.2
X	0/2.1/4.2/6.3/8.4/10.5/12.6

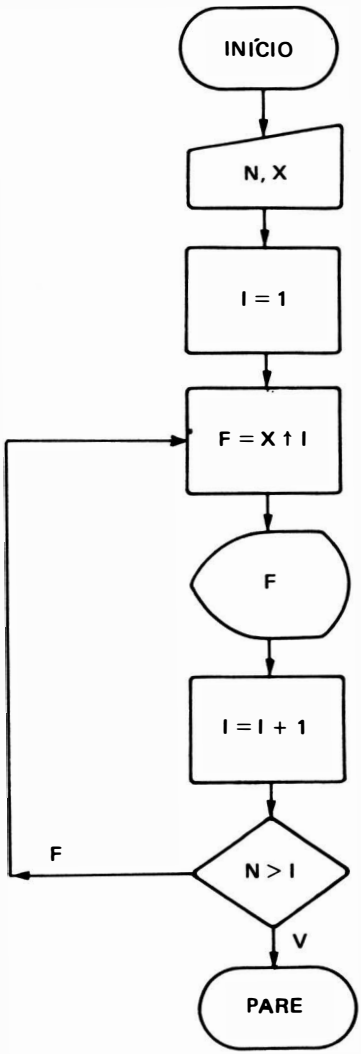
OBS.: Na localidade de memória X, a Barra (/) indica que o número foi apagado e substituído pelo próximo; o valor atual desta localidade de memória é o último (12.6).

2º) Fazer um fluxograma e programa BASIC que permita calcular e exibir as potências:

$X, X^2, X^3 \ldots X^N$, com N e X digitados na entrada com

X e $N > 0$ e $N \in N^*$

FLUXOGRAMA



PROGRAMA

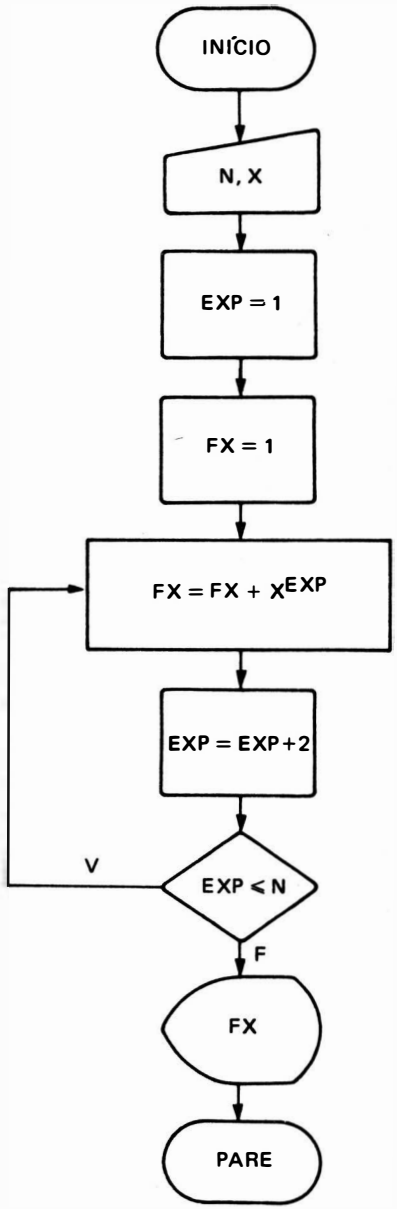
```
10 INPUT "N = "; N
20 INPUT "X = "; X
30 LET I = 1
40 LET F = X ↑ I
50 PRINT F
60 LET I = I + 1
70 IF > N THEN STOP
80 GOTO 40

RUN

2
4
8
16
32
64
128
256
512
1024
```

3º) Fazer um fluxograma e programa BASIC para calcular e exibir o valor de $f(x) = 1 + X + X^3 + X^5 + \ldots X^N$, sendo digitados N e X e supondo-se N ímpar.

FLUXOGRAMA

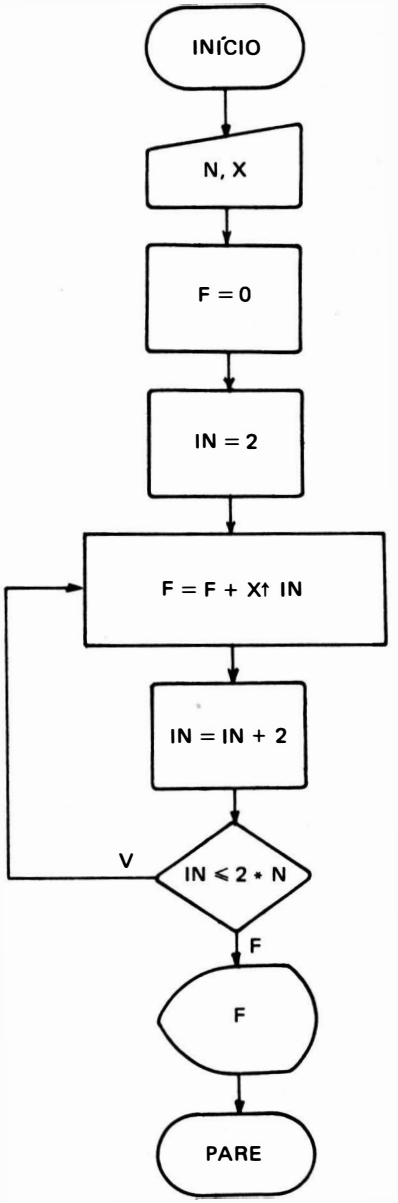


PROGRAMA

```
10 INPUT "N = "; N
20 INPUT "X = "; X
30 LET I = 1
40 LET FX = 1
50 LET FX = FX + X ↑ I
60 LET I = I + 2
70 IF I < = N THEN GOTO 50
80 PRINT "F (X) = "; FX
90 STOP
```

4º) Fazer um fluxograma e programa BASIC para calcular $F = X^2 + X^4 + X^6 + \ldots + X^{2N}$, sendo digitados N e X

FLUXOGRAMA

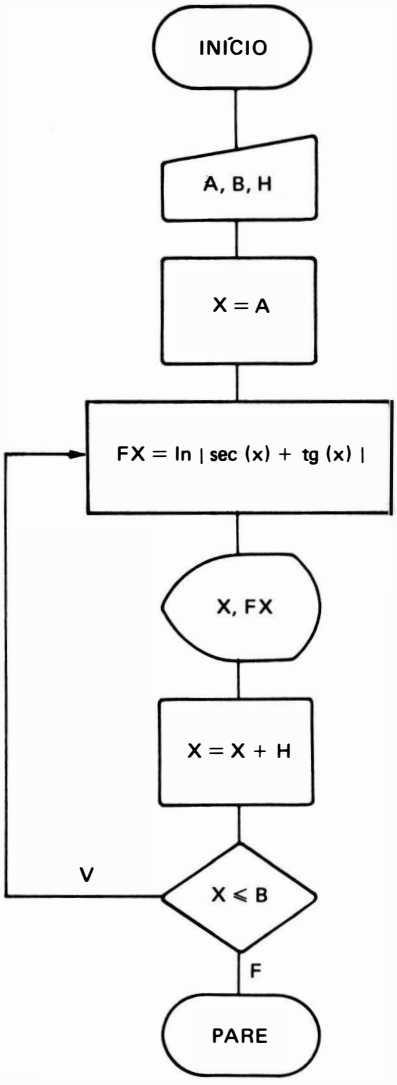


PROGRAMA

```
10 INPUT "N = "; N
20 INPUT "X = "; X
30 LET F = 0
40 LET IN = 2
50 LET F = F + X ↑ IN
60 LET IN = IN + 2
70 IF IN < = 2 * N THEN GOTO 50
80 PRINT "F = "; F
90 STOP
```

5º) Fazer um fluxograma e programa BASIC para calcular e exibir os valores da função $f(x) = \ln|\sec(x) + \operatorname{tg}(x)|$ para x variando no intervalo $[a, b]$ com passo h, sendo digitados a, b e h nesta ordem.

FLUXOGRAMA



PROGRAMA

```
10 INPUT "A = "; A
20 INPUT "B = "; B
30 INPUT "H = "; H
40 LET X = A
50 LET FX = LN (ABS (1/COS (X) + TAN (X)))
60 PRINT "PARA X = "; X;" F (X) = "; FX
70 LET X = X + H
80 IF X <= B THEN GOTO 50
90 STOP
```

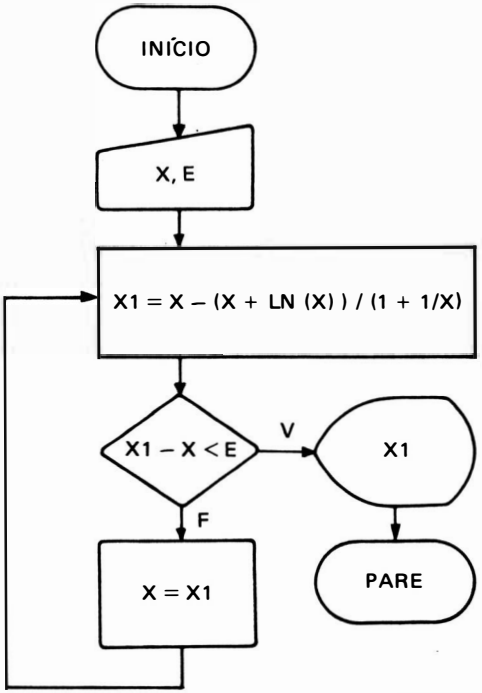
6º) Fazer fluxograma BASIC em que são digitados X e E na entrada, e achar a raiz aproximada da função $f(x) = x + \ln x$ usando o método das tangentes (de Newton) com aproximação inicial X e erro inferior a E.

OBS.: método das tangentes:

$$x_1 = x - \frac{f(x)}{f'(x)} \qquad \text{onde } f'(x) = 1 + \frac{1}{x}$$

substitui-se x por x_1 até que $x_1 - x < E$

FLUXOGRAMA



PROGRAMA

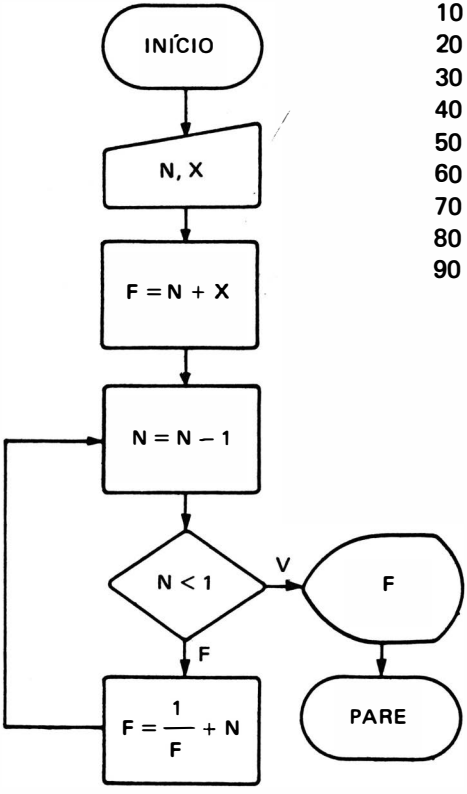
```
10 INPUT "X = "; X
20 INPUT "E = "; E
30 LET X1 = X – (X + LN (X))/(1 + 1/X)
40 IF ABS (X1 – X) < E THEN GOTO 70
50 LET X = X1
60 GOTO 30
70 PRINT "RAIZ APROXIMADA = "; X1
80 STOP
```

7º) Fazer fluxograma e programa BASIC para calcular o valor de $f(x)$

$$f(x) = 1 + \cfrac{1}{2 + \cfrac{1}{3 + \cfrac{1}{\ddots \cfrac{1}{n - 1 + \cfrac{1}{n + x}}}}}$$

sendo N e X digitados na entrada.

FLUXOGRAMA



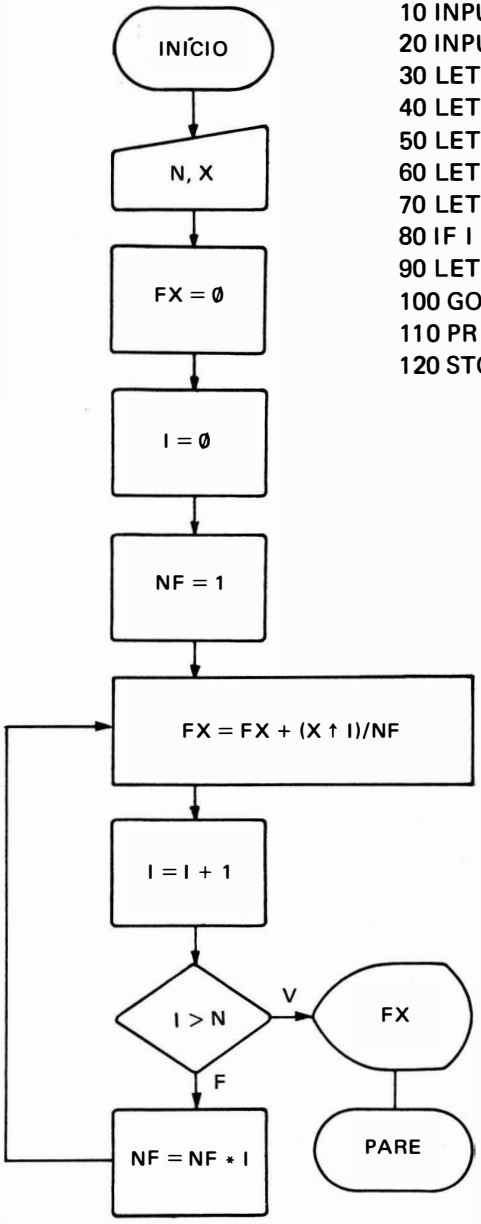
PROGRAMA

```
10 INPUT "N = "; N
20 INPUT "X = "; X
30 LET F = N + X
40 LET N = N – 1
50 IF N < 1 THEN GOTO 80
60 LET F = 1/F + N
70 GOTO 40
80 PRINT "VALOR FINAL DA FUNCAO = "; F
90 STOP
```

8º) Digitados N e X na entrada, fazer fluxograma e programa BASIC para calcular o valor da função:

$$f(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

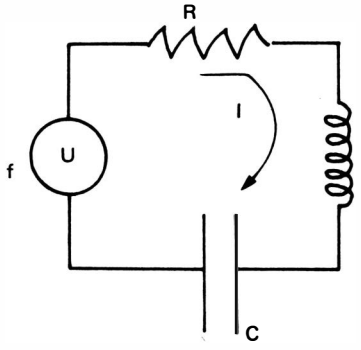
FLUXOGRAMA



PROGRAMA

```
10 INPUT "N = "; N
20 INPUT "X = "; X
30 LET FX = 0
40 LET I = 0
50 LET NF = 1
60 LET FX = FX + (X ↑ I)/NF
70 LET I = I + 1
80 IF I > N THEN GOTO 110
90 LET NF = NF * I
100 GOTO 60
110 PRINT "VALOR FINAL DA FUNCAO = "; FX
120 STOP
```

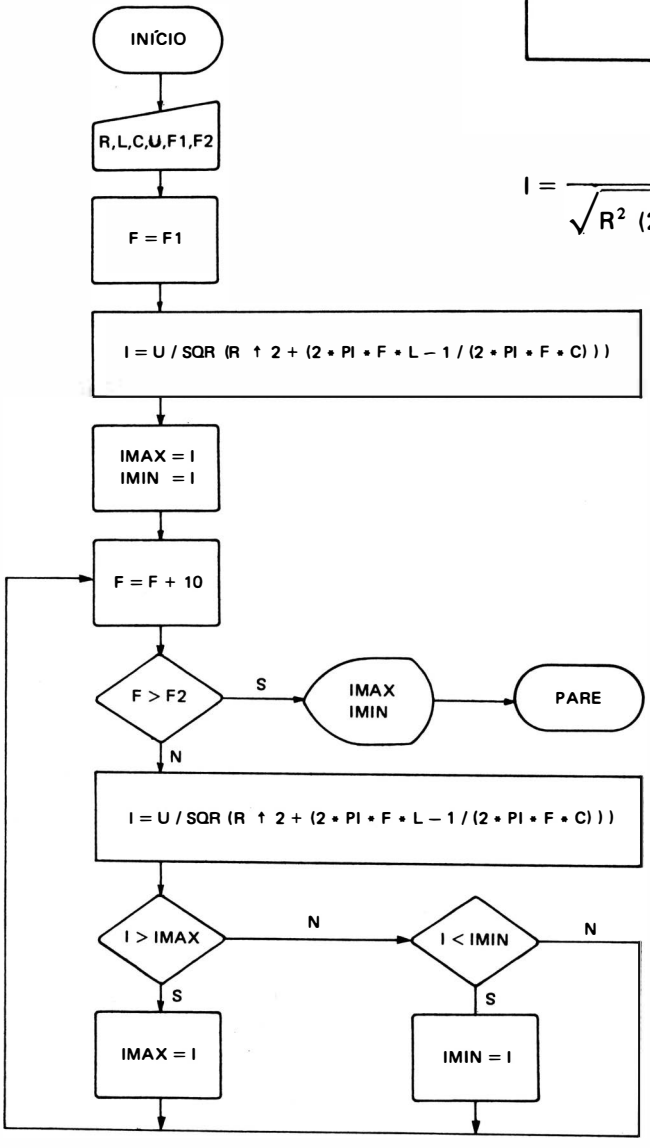
9º) Determinar o máximo e o mínimo valor que a corrente I assume num circuito RLC série, quando a freqüência varia de F1 a F2 com incrementos $\Delta F = 10$ Hz, sendo digitados na entrada os valores de R, L, C e U além das freqüências F1 e F2.



Sabe-se que:

$$I = \frac{U}{\sqrt{R^2 \ (2\pi F \cdot L - \frac{1}{2\pi F \cdot C}}}$$

FLUXOGRAMA



EXERCÍCIO 9

```
10 REM PROGRAMA PARA CALCULAR IMAXIMO E IMINIMO
20 INPUT "ENTRE COM O VALOR DA RESISTENCIA R   "; R
30 INPUT "ENTRE COM O VALOR DA INDUTANCIA L   "; L
40 INPUT "ENTRE COM O VALOR DA CAPACITANCIA C"; C
50 INPUT "ENTRE COM O VALOR DA TENSAO U       "; U
60 INPUT "ENTRE COM A FREQUENCIA INICIAL F1    "; F1
70 INPUT "ENTRE COM A FREQUENCIA FINAL F2      "; F2
80 REM FIM DA ENTRADA DE DADOS
90 LET F = F1
95 P = 3.1415927
100 LET I = U/SQR (R ↑ 2 + (2 * P * F * L – 1 / (2 * P * F * C)))
110 LET IMAX = I
120 LET IMIN = I
130 LET F = F + 10
140 IF F > F2 THEN GOTO 230
150 LET I = U/SQR (R ^ 2 + (2 * P * F * L – 1 / (2 * P * F * C)))
160 IF I > IMAX THEN GOTO 190
170 IF I < IMIN THEN GOTO 210
180 GOTO 130
190 LET IMAX = I
200 GOTO 130
210 LET IMIN = I
220 GOTO 130
230 PRINT "IMAX = "; IMAX; " IMIN = "; IMIN
240 STOP
```

5. EXERCÍCIOS PROPOSTOS

1.º) Fazer fluxograma e programa BASIC para imprimir os múltiplos de 11 entre 0 e 1 000.

2.º) Fazer fluxograma para exibir todos os números de 1 a 100.

3.º) Fazer fluxograma para exibir todos os números pares de 0 a 50.

4.º) Fazer fluxograma para calcular N, sendo N digitado na entrada.

5.º) Fazer fluxograma que permita calcular uma somatória de N números digitados na entrada, onde o valor de N também é fornecido.

6.º) Fazer fluxograma que permita calcular uma produtória de N números digitados na entrada, onde o valor de N também é fornecido.

7.º) Fazer fluxograma que permita resolver os problemas 5.º e 6.º juntos.

8.º) Fazer fluxograma que permita calcular a média Aritmética e Geométrica de N valores digitados na entrada.

9.º) Fazer fluxograma que permita exibir o maior número de uma lista de N valores digitados na entrada. O valor de N também é fornecido.

CAPÍTULO 5

COMANDOS BASIC

INTRODUÇÃO

Quando terminamos de escrever um determinado programa em linguagem BASIC, o mesmo estará em condições de ser processado, após ser introduzido no microcomputador. Para escrevermos um programa, utilizamos as instruções BASIC, que se seguidas passo a passo, nos fornecerá a resposta do programa. Entretanto, para operarmos o microcomputador, precisamos dar ordens que irão listar, executar, arquivar ou armazenar, para utilização futura, apagar outros programas da memória etc. Estas ordens específicas para a operação da máquina são chamadas de COMANDOS e poderão ser facilmente executadas digitando-se uma série de palavras-chave ou simplesmente apertando uma tecla do terminal do computador.

Podemos ter dois tipos de comandos: diretos e indiretos. Os comandos diretos são aqueles que não podem estar contidos no interior de um programa, enquanto que os comandos indiretos podem estar em linhas de um programa, sendo tratados, neste caso, como uma instrução comum.

Os comandos aceitos normalmente pela maioria dos microcomputadores são os seguintes:

1. ENTER — é uma tecla que deve ser apertada após a digitação de cada instrução dado ou comando BASIC, para que estes sejam reconhecidos como válidos pelo microcomputador. Pode ser encontrada com nomes diferentes, dependendo do fabricante do microcomputador, como: NEW LINE, CR ou RETURN.

2. BREAK — é uma tecla que tem por finalidade interromper o processamento de qualquer operação que está sendo efetuada pelo micro, devolvendo o controle ao teclado, para que o programador possa tomar alguma decisão, ou para entrada de nova operação, ou ainda para desistir-se da execução de um programa que está muito demorada.

3. RESET — esta tecla tem a finalidade de levar o microcomputador ao estado inicial de funcionamento, caso ocorra uma anomalia muito grave durante a operação do microcomputador, como por exemplo uma falha momentânea de energia, ou o "travamento", de tal forma que a tecla BREAK não seja obedecida. É usada como último recurso, sendo equivalente a desligar e religar o microcomputador, o que apaga qualquer programa BASIC contido na memória da máquina. Por medida de segurança, esta tecla é normalmente utilizada em conjunto com alguma outra, evitando-se assim um acionamento involuntário. Nos micros da linha Apple entretanto, o acionamento do RESET não apaga a memória.

4. SHIFT – esta tecla tem a finalidade de permitir ao usuário, a entrada de símbolos pelo teclado que estão no modo superior, análogo a máquina de escrever, onde algumas teclas possuem dois símbolos: um superior e outro inferior, sendo este último digitado de modo direto, enquanto que o superior é digitado em conjunto com a tecla "MAIÚSCULA".

Ex.: Para introduzirmos o caracter "!" (ponto de exclamação) que está, na maioria dos micros, no modo superior da tecla que contém o nº 1, digitamos a tecla SHIFT e a tecla 1.

5. BACK SPACE – esta tecla é utilizada para eliminar o último ou últimos caracteres digitados, produzindo um retrocesso, com o apagamento do caracter, que é geralmente indicado por um cursor. Dependendo do micro, esta tecla pode também chamar-se RUBOUT, sendo mais comum a tecla [←].

6. NEW – este comando apaga todos os dados e programas existentes na memória, deixando-a limpa para receber um novo programa. Deve ser utilizado sempre que for ser iniciada a digitação de um novo programa

7. RUN – este é o comando que manda executar ou "rodar" o programa que está na memória do micro.
Existem versões que admitem um número após a palavra RUN. Este número é um número de comando e indica que o programa deverá ser rodado a partir da linha com o número indicado.

Ex.: RUN 1000

Indica que o programa deverá ser executado a partir da linha nº 1000.

Isto é muito útil quando temos mais de um programa na memória, e com este recurso podemos executá-los independentemente.

8. LIST – este comando fornece a listagem ordenada de um programa inteiro na tela do micro.

Existem versões que admitem um número após a palavra LIST. Este número é um número de comando e indica que o programa deverá ser listado a partir da linha indicada.

Ex.: LIST 100

Caso o microcomputador tenha display de apenas uma linha, o comando LIST mostra a linha do programa, apertada a tecla ENTER passa à seguinte.

9. LLIST – este comando tem o mesmo efeito que o comando LIST, porém o programa é listado na IMPRESSORA.

Este comando deve ser utilizado somente quando a impressora está fisicamente conectada ao microcomputador, caso contrário poderá haver, em alguns casos, problemas que exijam que seja digitado a RESET, perdendo-se o programa.

10. SAVE – este comando copia o programa que está na memória do microcomputador em fita cassette. Existem versões que permitem "batizar" o programa a ser copiado da memória colocando-se o nome desejado do programa entre aspas após a palavra-chave SAVE:

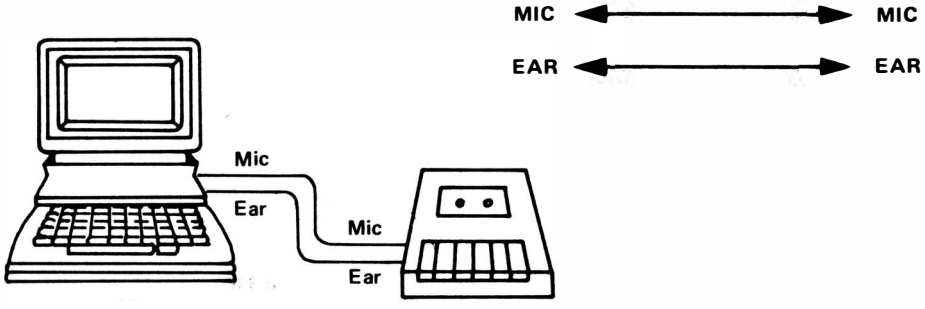
Ex.: SAVE "nome do programa"

11. LOAD – este comando copia o programa da fita cassette na memória do micro. Nas versões em que cada programa da fita possui um nome, apenas um será reconhecido, ignorando os restantes.

Ex.: LOAD "Nome do programa"

11.1. Operação com Gravador Cassette

Quando utilizamos um gravador cassette comum, a entrada e saída de audio do gravador devem estar conectadas ao micro, conforme a figura:



O micro não comanda o gravador e as operações "PLAY" ou "RECORD" devem ser efetuadas manualmente, seguindo as seguintes operações:

1) Posicionar a fita no ponto de gravar/ler;

2) Digitar SAVE (para gravar)/LOAD (para ler) a tecla ENTER;

3) Acionar os controles de gravar/ler do gravador;

4) Digitar a tecla de controle ENTER, o que faz SAVE ou LOAD ser executado.

Quando o micro possui sistema de disco flexível, não há trabalho manual, e os comandos SAVE/LOAD acionam diretamente o disco. Nestes micros, para o acionamento de um gravador cassette para arquivos em fita, usa-se CSAVE ou CLOAD.

12. AUTO – este comando liga uma função automática de numeração de linhas, para a entrada de programas, facilitando a digitação de um programa.

Cada vez que a tecla ENTER for apertada, aparece automaticamente na tela o número da próxima linha.

Possui a seguinte forma geral:

```
AUTO n
```

onde n indica o passo de numeração.

Se n for omitido, a numeração gerada será de 10 em 10.

Ex.: AUTO
Gera as linhas
10 . . .
20 . . .
30 . . .
.
.
.

esta função é desligada pela tecla BREAK.

13. CONT — este comando continua a execução do programa após sua interrupção pela instrução STOP ou pelo acionamento da tecla BREAK.

14. COPY — este comando tem por finalidade copiar na impressora uma imagem estacionária na tela do micro, possibilitando a impressão de gráficos ou desenhos que não poderiam ser efetuados diretamente na impressora, enquanto estes estivessem sendo construídos.

15. EDIT — este comando permite que sejam feitas correções em alguma linha do programa. Sua principal finalidade é evitar que a linha a ser corrigida tenha que ser redigitada completamente.

A forma geral deste comando é

```
EDIT nl
```

onde *nl* é o número da linha que se pretende corrigir.

A forma de correção, entretanto, varia muito de um computador para outro, havendo em alguns casos um cursor de edição que desloca-se para a direita e para a esquerda, outros pela digitação de subcomandos de edição tais como I para inserir um caracter, D para apagar ou deletar um caracter etc.

16. DELETE — este comando tem a finalidade de apagar uma ou mais linhas de um programa que está armazenado na memória do microcomputador.

A forma geral deste comando é a seguinte:

DELETE ni – nf

onde ni indica a linha inicial que será apagada.
nf indica a linha final que será apagada.

Podendo ser empregado da seguinte forma:

DELETE n – apaga apenas a linha n

Ex.: DELETE 20 – apaga a linha nº 20
DELETE ni-nf – apaga todas as linhas compreendidas entre ni e nf.

Ex.: DELETE 30-50
Apaga todas as linhas que estiverem entre 30 e 50 inclusive.

DELETE – nf – apaga todas as linhas até a linha nf.

Ex.: DELETE – 100
Apaga todas as linhas que estiverem entre 1 e 100 inclusive.

OBS.: Para apagarmos apenas uma linha é mais prático e rápido digitarmos o número da linha e a tecla ENTER.

Ex.: Para apagarmos a linha 50, digitamos apenas 50 e apertamos a tecla ENTER.

17. RENUMBER – este comando tem a finalidade de renumerar, automaticamente, todas as linhas de um programa, deixando a versão final de um programa mais elegante graças a numeração com passo fixo.

Sua forma geral é a seguinte:

RENUMBER n

onde n é o passo que se deseja para cada número de linha.

Se n for omitido a renumeração será de 10 em 10.

CAPÍTULO 6

NÚMEROS

1. INTRODUÇÃO

No tempo das cavernas, o homem se esforçava para simbolizar as quantidades, de forma a poder controlar com segurança o seu rebanho. Iniciou fazendo uma equivalência biunívoca com pedras, de forma que para cada ovelha que adentrava o curral, era colocada uma pedra em um saco. Mas, à medida que o rebanho aumentou, começou a ficar também grande a quantidade de pedras correspondentes, o que o obrigou a criar um novo símbolo, uma pedra pouco maior, para conjuntos ou grupos de ovelhas. Observando suas mãos adotou dois conjuntos para 5 e 10 ovelhas e mais tarde evoluiu, criando símbolos para 50 e 100 ovelhas.

Atualmente, continuamos adotando símbolos para representar informações. Observamos que uma máquina elétrica apresenta uma característica binária, isto é, apresenta, invariavelmente, apenas 2 estados de funcionamento possíveis; por exemplo: uma chave pode estar ligada ou desligada; a voltagem em um circuito pode ser alta ou baixa; pode haver corrente elétrica num sentido ou no sentido oposto; uma lâmpada pode estar acesa ou apagada; um núcleo de ferrite pode estar magnetizado num sentido ou no sentido oposto e inúmeras outras situações. Sempre que tivermos apenas 2 estados possíveis podemos representar estes estados simbolicamente como 0 a 1, ou ainda falso e verdadeiro. Esta representação nos fornece a unidade básica de informação, que é chamada "*bit*", da aglutinação das palavras *b*inary dig*it* ou dígitos binários.

2. SISTEMAS DE NUMERAÇÃO

Existem vários sistemas de numeração, como: os binários, octais, hexadecimais e o de numeração decimal, que é o sistema de numeração ao qual estamos acostumados.

Os números inteiros e positivos, de um determinado sistema de numeração, podem ser calculados pela seguinte expressão:

$$N = A_0 B^n + A_1 B^{n-1} + \ldots + A_{n-2} B^2 + A_{n-1} B + A_n B^0$$

onde B indica a base do sistema de numeração que pode ser:

2 para o sistema de numeração binária
8 para o sistema de numeração octal

10 para o sistema de numeração decimal
16 para o sistema de numeração hexadecimal.

Exs.:

O número 1253 decimal representa o valor:

$$1 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 = 1\,000 + 200 + 50 + 3 = 1\,253$$

O número 0110 binário representa o valor:

$$0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2 = 6 \text{ decimal}$$

OBS.: Para a representação de frações, a mesma fórmula anterior pode ser entendida para potências negativas da base.

Ex.: O número 0,948 decimal pode ser representado por:

$$9 \times 10^{-1} + 4 \times 10^{-2} + 8 \times 10^{-3}.$$

3. TIPOS DE NÚMEROS ACEITOS PELA LINGUAGEM BASIC

As quantidades numéricas normalmente aceitas pela linguagem BASIC são decimais e podem ser representadas de duas maneiras diferentes:

a) *Número Inteiro* — também chamado de constante inteira ou ponto fixo, são números escritos sem o uso do ponto decimal.

b) *Número Real* — também chamado de constante real ou ponto flutuante, são números escritos com o uso de ponto decimal.

Exs.:	10	1024	0	121	são inteiros
	10.5	0.2	.2	10.	são reais

Para a representação dos números devemos levar em conta:

1) Não se usa vírgula na representação do valor numérico real mas sim o ponto decimal.

2) Os números podem ser precedidos dos sinais + ou — (o sinal + é opcional).

3) Os números podem ser escritos com 8 a 15 algarismos significativos, dependendo das versões BASIC e do computador utilizado.

4. ORDEM DE GRANDEZA

A ordem de grandeza normalmente está na faixa de 10^{-38} a 10^{+38} variando de um computador para outro.

Ex.:	CP2000	10^{-38} a 10^{+38}
	CP300	10^{-38} a 10^{+38}
	CP500	10^{-38} a 10^{+38}
	MICROTEC MT400	10^{-127} a 10^{+127}
	TRS80	10^{-38} a 10^{+38}
	APPLE] [	10^{-38} a 10^{+38}

Caso um número assuma um valor maior que o máximo permitido, o computador entra em estado de "excesso" ou OVERFLOW. Se um número assume um valor menor que o mínimo permitido, o computador entra em estado de "falta" ou UNDERFLOW.

5. NOTAÇÃO EXPONENCIAL

Existe uma forma diferente para representarmos números reais muito grandes ou diminutos que é a notação exponencial, ou notação científica padrão, cuja forma geral é:

m E c

onde: m = mantissa — número real com até 9 casas decimais
c = característica — número inteiro com 2 casas e sinal.

Esta forma representa o valor:

$m\ 10^c$

ou seja, um número m multiplicado por uma potência de 10.

Ex.:	$6{,}02 \times 10^{23}$	é escrito em BASIC como: 6.02 E 23
	0,000005	é escrito em BASIC como: 5E − Ø6
	$-3{,}45 \times 10^{-2}$	é escrito em BASIC como: − 3.45E − 02
	10^8	é escrito em BASIC como: 1E + 8
	10^{-5}	é escrito em BASIC como: 1E − 5

6. EXERCÍCIOS

1. Que quantidade decimal representa cada uma das representações numéricas binárias abaixo:

a) $1000 = 1 \times 2^3 = 8$

b) $0101 = 1 \times 2^2 + 1 \times 2^0 = 5$

c) $1111 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 15$

d) $1010 = 1 \times 2^3 + 1 \times 2^1 = 10$

e) $11101011 = 2^7 + 2^6 + 2^5 + 2^3 + 2^1 + 2^0 = 235$

f) $00100001 = 2^5 + 2^0 = 33$

g) $11111111 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 255$

h) $01010101 = 2^6 + 2^4 + 2^2 + 2^0 = 85$

2. Quais das seguintes representações de número inteiro são incorretas para a linguagem BASIC. Indique, se for o caso, a forma correta:

a) 0. — é real pois tem ponto decimal
Forma inteira correta: 0

b) + 0 — correto

c) − 0 — correto

d) 1.0 — é real pois tem ponto decimal
Forma inteira correta = 1

e) 0.1E + 4 — é real pois tem ponto decimal
Forma inteira correta: 1000 ou 1E + 3

f) − 613E − 7 — impossível pois o número é real = 0.0000613

g) − 6.13E − 7 — idem ao anterior

h) 3E + 6 — correto ou 3000000

i) 001 — correto

3. Quais destes pares representam o mesmo número em BASIC:

a) 3.14 = + 03.14 correto

b) 3.14 = + 3.4E + 01 correto

c) 0.00314 = 314.E − 05 correto

d) 3140 = 3.14E + 03 falso, pois mistura inteiro com real

CAPÍTULO 7

CONTROLE DA MALHA ITERATIVA

1. INTRODUÇÃO

Como vimos, as operações repetitivas de um determinado programa, eram feitas por uma variável controladora, à qual era definido um valor inicial, incrementada de uma constante e comparada com um valor limite, numa instrução IF-THEN, até que fosse possível sair desta Malha Iterativa ("LOOP").

Ex.: Imprimir os números inteiros de 1 a 100:

```
10 LET I = 1
20 PRINT I
30 LET I = I + 1
40 IF I > 100 THEN STOP
50 GOTO 20
```

onde: I é a variável controladora da malha, anel, "loop", ou ainda, I é uma variável controladora do programa.

Existe, porém, outra maneira de se conseguir o mesmo efeito da variável controladora, de forma mais simplificada, através do uso das instruções FOR-NEXT.

Ex.:
```
10 FOR I = 1 TO 100
20 PRINT I
30 NEXT I
40 STOP
```

onde a instrução 20 é repetida enquanto a variável controladora I varia automaticamente de 1 em 1 desde I = 1 até I = 100.

2. INSTRUÇÃO FOR

A instrução FOR é um dos mais úteis e poderosos recursos da linguagem BASIC. É uma instrução de iterações (contagem) que substitui o emprego de contadores com uma variável

inicial, que seria incrementada sucessivamente até um limite, facilitando e simplificando programação.

Forma Geral:

nl FOR i = c1 TO c2 STEP c3

onde nl indica o número da linha
i = variável controladora
c1, c2, c3 variáveis ou constantes

Significado:

c1 = valor inicial da variável controladora i
c2 = valor final de i
c3 = acréscimo que ganha a variável controladora i em cada ciclo — (passo ou incremento)

OBS.: Se c_3 for igual a 1, este poderá ser omitido.

Ex.: FOR I = 1 TO 10

O número máximo de repetições em um anel pode ser calculado por:

$$\left[\frac{c2 - c1}{c2}\right] + 1$$

onde [ ] indica "maior inteiro contido".

2.1. Funcionamento da Instrução FOR

A instrução FOR causa a repetida execução de todas as instruções seguintes até a instrução NEXT. Na primeira vez que esta instrução é executada, a variável controladora é igual c1, e em cada passagem sucessiva, é aumentada pelo valor c3 até que na passagem final igua a c2. Neste ponto, o laço FOR-NEXT "fica satisfeito" e o controle é transferido para a instrução que se segue imediatamente após a instrução NEXT.

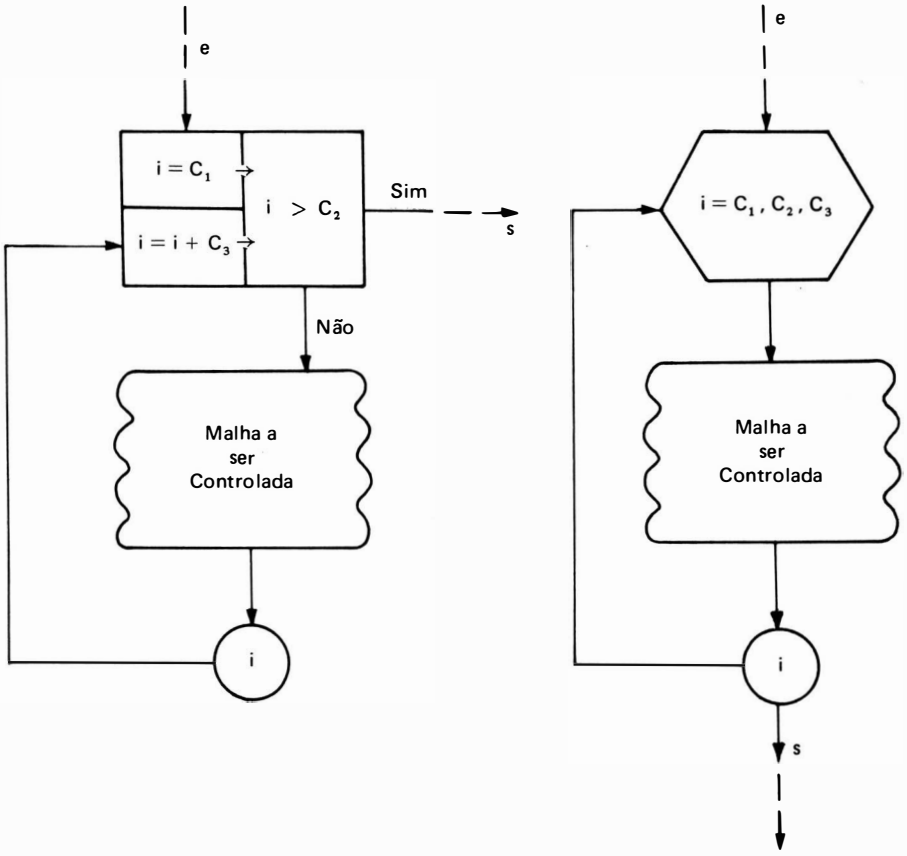
OBS.: O laço FOR-NEXT fica satisfeito no momento em que a variável controladora i se torn maior que c2.

Ex.: Calcular a soma: 1 + 3 + 5 + 7 + 9 . . . N

```
10 INPUT N
20 LET S = 0
30 FOR I = 1 TO N STEP 2
40 LET S = S + I
50 NEXT I
60 PRINT "S = "; S
70 STOP
```

2.2. Representação no Diagrama de Blocos

Podemos ter as seguintes representações para as instruções FOR-NEXT no diagrama de blocos:



onde *e* representa a entrada de fluxo do processamento nos respectivos blocos, e *s* representa o ponto de saída.

A malha a ser controlada pode conter um conjunto de instruções, que eventualmente pode não conter nenhuma instrução.

3. INSTRUÇÃO NEXT

O comando NEXT sempre é representado simplesmente por

nl NEXT i

onde *nl* é o número da linha, i é a variável controladora da instrução FOR correspondente.

É uma instrução fictícia, pois não participa de nenhum papel ativo na computação, se vindo meramente de ponto de referência.

A utilização da instrução NEXT é sempre a última instrução dentro de um campo d atividades de um laço FOR.

Cada instrução FOR deve ter obrigatoriamente a sua instrução NEXT correspondente.

OBS.: Quando o controle sai do laço FOR-NEXT, depois que este for satisfeito, o valor do ín dice *i* fica definido como sendo c2 + c3, podendo ser usado para cálculos subsequentes.

4. REGRAS PARA USO DAS INSTRUÇÕES FOR-NEXT

Os parâmetros da instrução FOR, isto é c1, c2, c3 não devem ser modificados durante execução do anel.

Ex.: não são corretos os seguintes trechos de programa (apesar de executáveis):

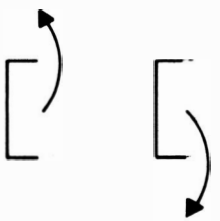
```
10 LET A = 10
20 LET N = 1
30 FOR I = 1 TO 100 STEP N
40 LET N = N + A
50 NEXT I
```

Neste exemplo, o valor do passo *N* está sendo modificado a cada execução do anel FOR NEXT.

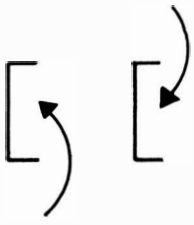
```
10 LET X = 20
20 FOR B = 10 TO 500
30 LET B = B + X
40 PRINT B
50 NEXT B
```

Neste exemplo, o valor da variável controladora B está sendo modificado.

O controle do processamento pode sair de dentro de um laço FOR-NEXT sem que est esteja satisfeito, através de instruções de desvio (IF, THEN, GOTO etc.), entretanto, *NÃO* permitida a transferência de fora para dentro.



Permitido



Não Permitido

5. LAÇOS EMBUTIDOS

Um laço FOR-NEXT pode aparecer no interior de outro anel FOR-NEXT, desde que todos os comandos que apareçam no anel mais interno estejam totalmente dentro do anel mais externo.

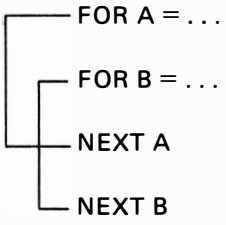
```
Ex.: 10 FOR I = 1 TO 5
     20      FOR J = 1 TO 3
     30      PRINT I, J
     40      NEXT J
     50 NEXT I
```

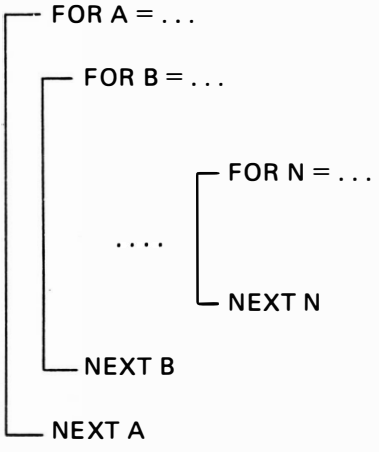
Este programa imprime a seguinte lista de valores para I e J:

I	J
1	1
1	2
1	3
2	1
2	2
2	3
.	
.	
.	
5	1
5	2
5	3

Esquematicamente:

Embutimento Não Permitido





Embutimentos Permitidos

É boa prática de programação, quando escrevemos um programa BASIC, que utilize laços embutidos de instruções FOR-NEXT, que a escrita do laço mais interno seja destacada colocando-a algumas colunas mais à direita, bem como todas as instruções pertencentes a esta malha iterativa, em relação ao laço mais externo, da forma:

```
100 FOR J = 1 TO N
110 LET A = 3 * X
120     FOR M = K TO 100 STEP X
```

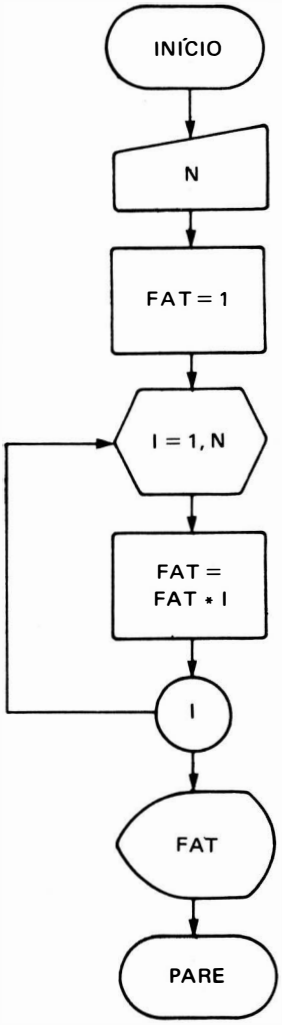
```
130     PRINT M
140     LET P = P + 1
150     NEXT M
160 NEXT J
```

Tal procedimento facilita muito a visualização dos laços FOR-NEXT no programa para uma possível correção de programa.

OBS.: O número máximo de laços embutidos varia de um computador para outro, estando em torno de 9 níveis de aninhamento.

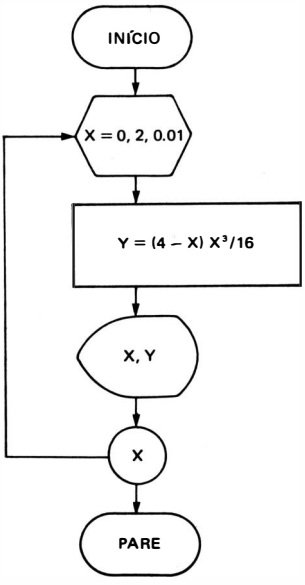
6. EXERCÍCIOS

1. Fazer fluxograma e programa BASIC para calcular o fatorial de um número N (N!) que é digitado na entrada. (N ∈ N * < 33).



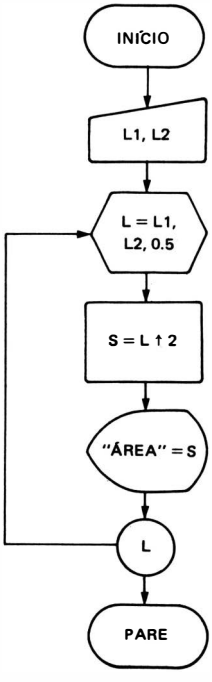

```
10 REM PROGRAMA PARA CALCULAR O FATORIAL
20 INPUT "ENTRE COM O VALOR DE N "; N
30 LET FAT = 1
40 FOR I = 1 TO N
50     FAT = FAT * I
60 NEXT I
70 PRINT "FATORIAL DE "; N;" = "; FAT
80 STOP
```

2. Fazer fluxograma e programa BASIC que calcule os valores de $y = \frac{(4 - x)\, x^3}{16}$ para x variando de 0 a 2 com passo ou incremento h = 0,01.



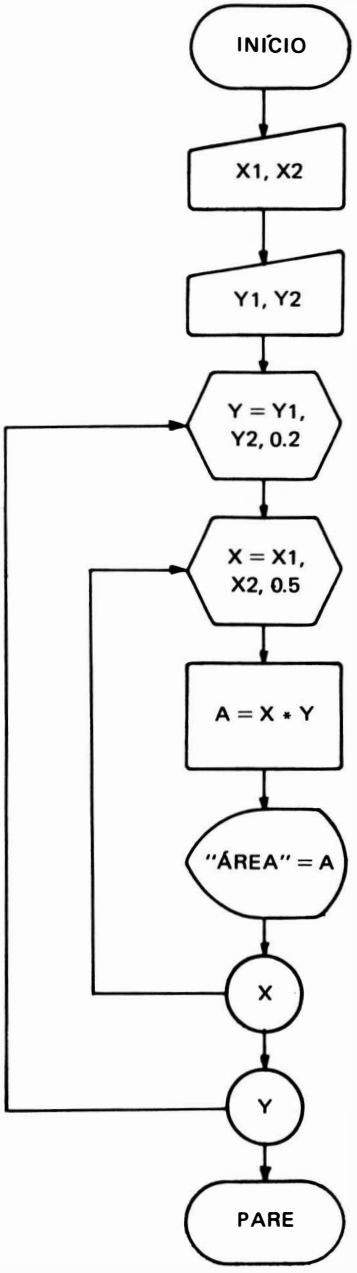

```
10 FOR X = 0 TO 2 STEP 0.01
20    LET Y = ((4 – X) * X ↑ 3)/16
30    PRINT X, Y
40 NEXT X
50 STOP
```

3. Fazer fluxograma BASIC para calcular a área de vários quadrados cujo lado L varia de L1 a L2. Admitir incrementos de 0,5 cm para o cálculo do próximo quadrado. Os valores de L1 e L2 são digitados na entrada.



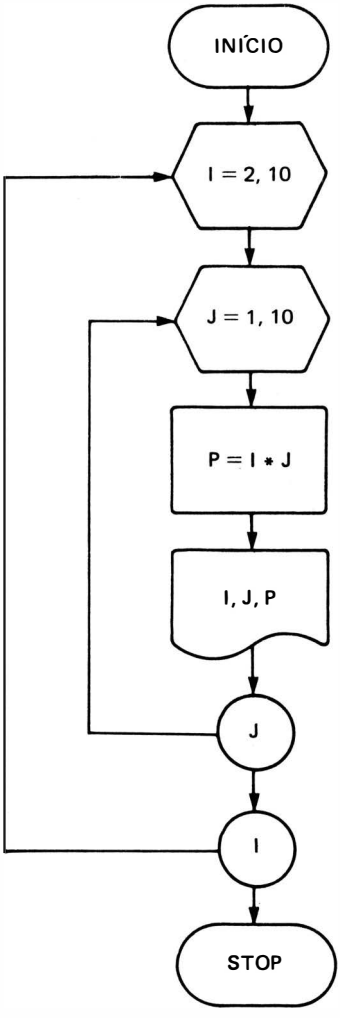

```
10 INPUT "L1 = "; L1
20 INPUT "L2 = "; L2
30 FOR L = L1 TO L2 STEP 0.5
40    LET S = L ↑ 2
50    PRINT "AREA DO QUADRADO
      COM LADO = "; L ;" = "; S
60 NEXT L
70 STOP
```

4. Fazer fluxograma e programa BASIC para calcular a área de vários retângulos com o la menor X variando de X1 a X2 de 0,5 em 0,5 cm e com o lado maior Y variando de Y1 a de 0,2 em 0,2 cm. Os valores de X1, X2, Y1 e Y2 são digitados na entrada.




```
10 REM PROGRAMA PARA CALCULAR A ARE
DE RETANGULOS
20 REM ENTRE OS EXTREMOS X1, X2 E Y1, Y
30 INPUT "X1 = ", X1
40 INPUT "X2 = ", X2
50 INPUT "Y1 = ", Y1
60 INPUT "Y2 = ", Y2
70 FOR Y = Y1 TO Y2 STEP .2
80     FOR X = X1 TO X2 STEP .5
90         LET A = X * Y
100         PRINT "AREA DO RETANGULO CO
LADOS "; X;" E "; Y;" = "; A
110   NEXT X
120 NEXT Y
130 STOP
```

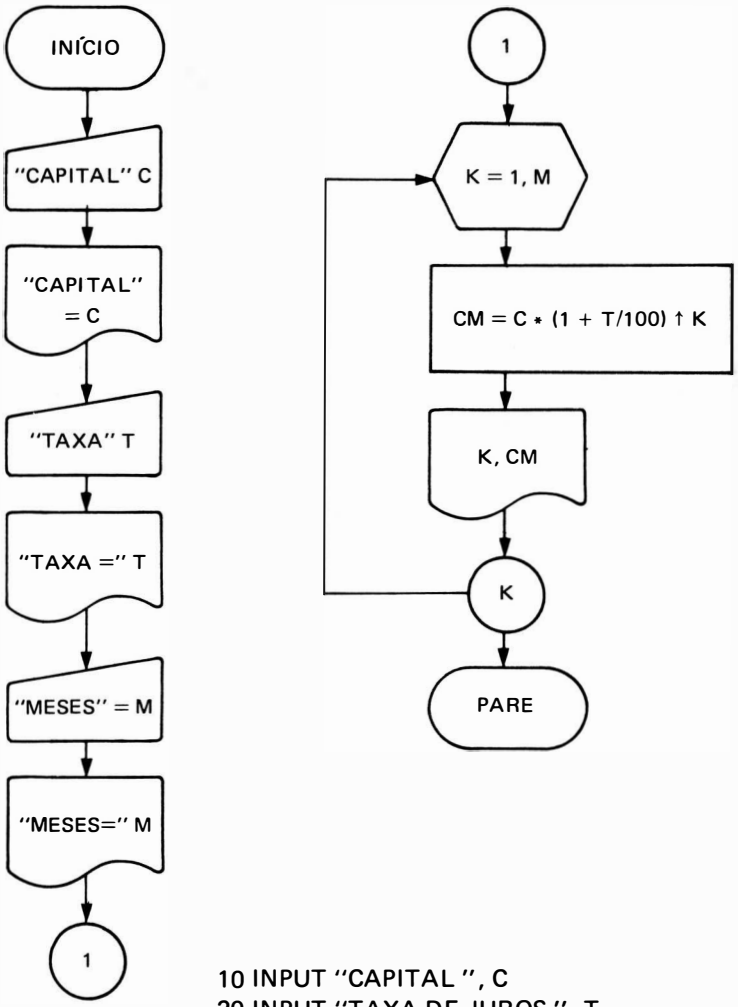
5. Fazer fluxograma e programa BASIC que imprima as tabuadas do 2 ao 10 usando as instruções FOR-NEXT.




```
10 REM TABUADA
20 FOR I = 2 TO 10
30     FOR J = 1 TO 10
40     LET P = I * J
50     LPRINT I;" * ";J;" = ";P
60     NEXT J
70     LPRINT
80     LPRINT
90 NEXT I
100 STOP
```

6. Fazer fluxograma e programa BASIC para calcular a rentabilidade de um capital aplicado e RDB (recibo de depósito bancário) a uma taxa de juros mensal T, tabelando por um prazo d M meses, com os valores de C, T e N digitados na entrada.

Sabe-se que $CM = C * (1 + T/100)^M$ onde CM é o capital montante.




```
10 INPUT "CAPITAL ", C
20 INPUT "TAXA DE JUROS ", T
30 INPUT "MESES     ", M
40 LPRINT "CAPITAL          "; C
50 LPRINT "TAXA DE JUROS "; T
60 LPRINT "MESES            "; M
70 FOR K = 1 TO M
80    LET C1 = C * (1 + T/100) ↑ K
90    LPRINT "EM   "; K;" MESES : MONTANTE = "; (
100 NEXT K
110 STOP
```

7. Fazer fluxograma e programa BASIC para exibir todos os números primos de 1 a N, com N digitado na entrada (N inteiro positivo). Para processo quando for digitado N = 0.

Definição: número primo é o número que é divisível somente por ele mesmo ou por 1.

Seja P o número inteiro para o qual faremos o teste de verificação de número primo; seja J seu divisor.

Logo:

$$Q = \frac{P}{J} \qquad Q = \text{Quociente}$$

Para verificarmos se P é um número primo deveremos variar o valor de J de 2 a P. Se o quociente Q, for um número inteiro para algum valor da seqüência de J então P não é primo, para isso faremos a comparação de

$$Q1 = INT\ (Q) \text{ como quociente } Q$$

Se Q1 = Q a divisão foi exata e, portanto, P não é primo.

Melhorando o teste:

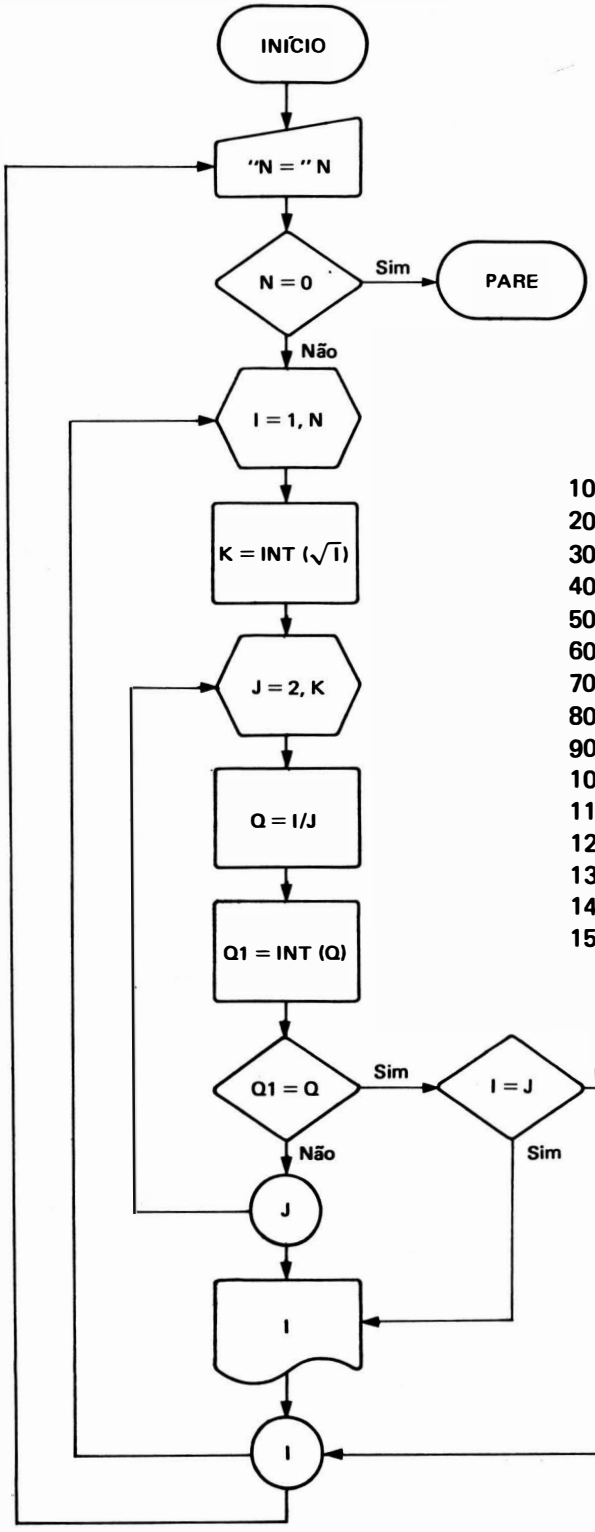
Sabemos que se 2 é divisor do número P $\Rightarrow \frac{P}{2}$ também é

se 3 é divisor do número P $\Rightarrow \frac{P}{3}$ também é

.
.
.

se D é divisor do número P $\Rightarrow \frac{P}{D}$ também é

Portanto, $D \leqslant \sqrt{P}$ pois como podemos observar: $D \leqslant \frac{P}{D}$ e para $D = \frac{P}{D}$ temos $D = \sqrt{P}$ isto implica que não precisamos pesquisar a seqüência toda, mas apenas até $\sqrt{P}$.

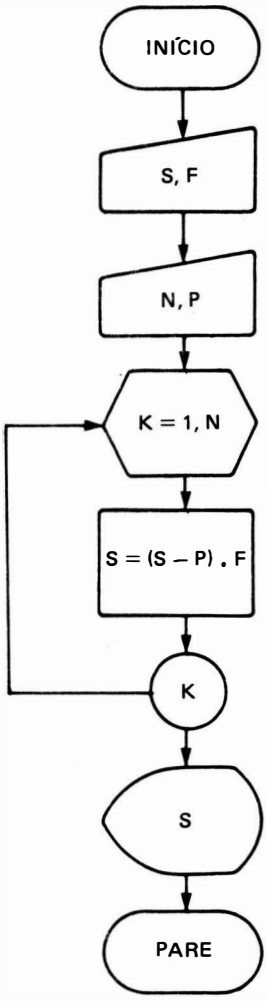



```
10 INPUT "N = "; N
20 IF N = 0 THEN STOP
30 FOR I = 1 TO N
40    LET K = INT (SQR (I))
50    FOR J = 2 TO K
60       LET Q = I/J
70       LET Q1 = INT (Q)
80       IF Q1 = Q THEN GOTO 130
90    NEXT J
100 LPRINT I;
110 NEXT I
120 LPRINT
130 GOTO 10
140 IF I = J THEN GOTO 100
150 GOTO 110
```

8. Fazer fluxograma e programa BASIC para calcular o saldo final de uma dívida a ser paga er *N* prestações iguais cujo valor é P com saldo inicial valendo 5. De acordo com a tabel PRICE, o saldo remanescente é corrigido mensalmente por um fator F. Para a correção d saldo abate-se a prestação e aplica-se o fator:

$$S = (S - P) \cdot F$$

Os valores de S, F, N e P são digitados na entrada.

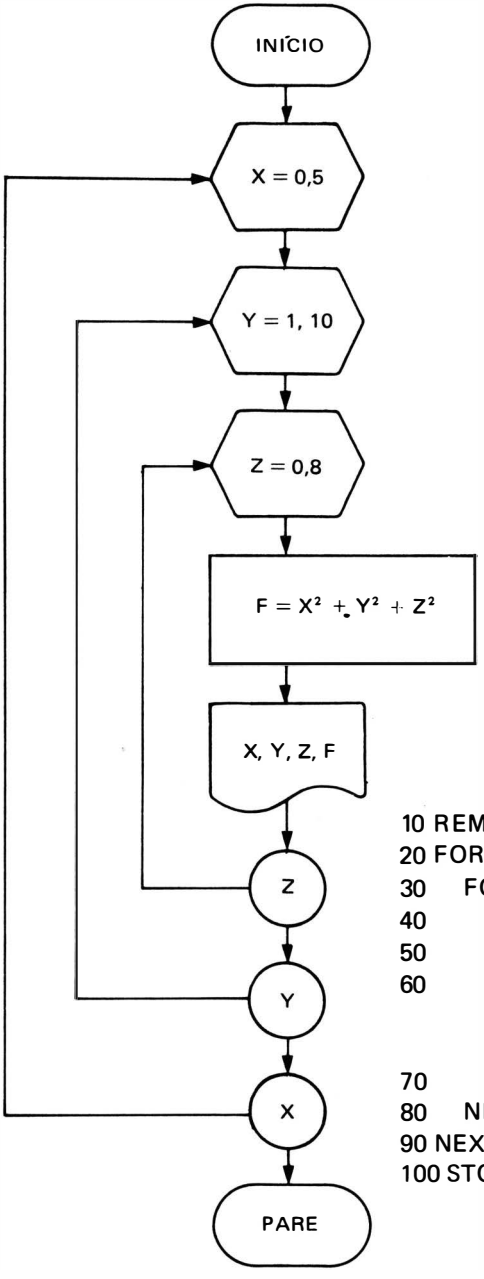



```
10 INPUT "SALDO INICIAL "; S
20 INPUT "FATOR MENSAL "; F
30 INPUT "NO. DE PRESTACOES "; N
40 INPUT "VALOR DE CADA PRESTACAO "; P
50 LPRINT "SALDO INICIAL                "; S
60 LPRINT "FATOR MENSAL                 "; F
70 LPRINT "NO. DE PRESTACOES            "; N
80 LPRINT "VALOR DE CADA PRESTACAO      "; P
90 REM CALCULO DO SALDO FINAL
100 FOR K = 1 TO N
110     S = (S – P) * F
120 NEXT K
130 LPRINT "SALDO FINAL = "; S
140 STOP
```

9. Fazer fluxograma e programa BASIC para calcular o valor de $f(x, y, z) = x^2 + y^2 + z^2$:

para x variando no intervalo de 0 a 5.
para y variando no intervalo de 1 a 10.
para z variando no intervalo de 0 a 8.

Considerar incrementos unitários para x, y, z.




```
10 REM TABELA DA FUNCAO X ↑ 2 + Y ↑ 2 + Z ↑ 2
20 FOR X = 0 TO 5
30    FOR Y = 0 TO 3
40       FOR Z = 0 TO 4
50          LET F = X ↑ 2 + Y ↑ 2 + Z ↑ 2
60          LPRINT "PARA
            X = "; X;" Y = "; Y;" Z = "; Z;" A
            FUNCAO VALE "; F
70       NEXT Z
80    NEXT Y
90 NEXT X
100 STOP
```

CAPÍTULO 8

LEITURA E ARQUIVO DE DADOS NO PROGRAMA

Existem programas BASIC onde há necessidade de introduzirmos uma grande quantidad de dados, ou mesmo repetirmos a introdução de um conjunto de dados várias vezes. Como já vimos anteriormente, isto pode ser realizado com o auxílio da instrução INPUT. Porém é muit desconfortável para o programador digitar um grande número de dados sendo, portanto, ma conveniente, introduzir os dados uma única vez, através da instrução DATA (dados), e quand necessário, lê-los através de uma instrução READ.

1. INSTRUÇÃO DATA

A instrução DATA permite que seja colocada no programa, uma lista de dados para seren usados em cálculos, sem que o programador tenha que introduzi-los através de uma ou mai instruções INPUT, fazendo com que o programa seja executado mais rapidamente, pois n tempo gasto para digitar-se um certo valor, o computador executa centenas de operações.

Forma Geral:

nl DATA d1, d2, d3, . . .

onde *nl* indica o número da linha
d1, d2, d3 . . . dados numéricos ou alfanuméricos

OBS.: Os dados alfanuméricos colocados em uma lista DATA devem obrigatoriamente se colocados entre aspas.

Ex.: 500 DATA "SAO PAULO", 25, 1, 1554

2. INSTRUÇÃO READ

Os dados de uma instrução DATA podem ser copiados seqüencialmente por uma instru- ção READ, que tem por finalidade especificar as variáveis cujos valores serão introduzidos n computador.

Forma Geral:

```
nl READ v1, v2, v3 . . .
```

onde *nl* indica o número da linha
v1, v2, v3, . . . são variáveis BASIC correspondentes em tipo e ordem aos dados d1, d2, d3 . . .

Cada vez que a instrução READ é executada, é lido o dado seguinte ao último valor copiado da instrução DATA. Isto é feito automaticamente pelo micro, pois este guarda a posição do último dado lido numa variável interna chamada "apontador", que não é disponível ao usuário.

```
Ex.: 10 READ D
     20 FOR I = 1 TO D
     30    PRINT I;
     40 NEXT I
     50 GOTO 10
     100 DATA 31, 28, 31, 30, 31, 30, 31, 31, 30 31, 30, 31
```

Este programa lê um valor da lista DATA, que será o parâmetro final de uma instrução FOR, gerando a impressão de 1 até este valor final.

Os valores da lista DATA representam o último dia de cada um dos 12 meses do ano.

OBS.:

a) As instruções DATA não precisam ter correspondência com uma instrução READ em particular, desde que respeitada a ordem e o tipo de variáveis da instrução READ.

b) As instruções DATA podem aparecer em qualquer parte do programa, embora seja conveniente colocá-las todas agrupadas no final do programa.

c) Para uma instrução READ podemos ter várias linhas de instruções DATA. Terminados os elementos da primeira instrução DATA a leitura prossegue no conteúdo da instrução DATA seguinte, e assim sucessivamente.

Ex.:

```
  10 READ A$, T
       .
       .
       .
       .
100 DATA "ALFA", 10, "BETA", 50
110 DATA "GAMA", 30
```

3. INSTRUÇÃO RESTORE

Existem programas em que é necessário ler-se o mesmo conjunto de dados várias vezes Isto é possível com o uso da instrução RESTORE. Esta instrução permite "restaurar" à con dição original, todas as linhas de dados (DATA) do programa, permitindo a leitura dos dado a partir do começo.

Forma Geral:

nl RESTORE

Ex.: 10 READ A, B, C
20 READ D, E
30 RESTORE
40 READ D, E
50 DATA 10, 20, 30, 40, 50, 60

Neste exemplo, quando a linha 10 for executada o ponteiro aponta para o primeiro valo da lista DATA. Então, na execução das linhas 10 do programa, a variável A fica com o valor 10 a variável B fica com o valor 20, a variável C fica com o valor 30, a variável D fica com o valor 40 e a variável E fica com o valor 50. Ao encontrar a linha 30, o ponteiro é "restaurado" para o início da linha DATA e quando a leitura da linha 40 for executada, a variável D passa a fica com o valor 10 e a variável E passa a ficar com o valor 20. O mapa de memória fica, então, c seguinte:

A	B	C	D	E
10	20	30	40 / 10	50 / 20

4. CONTROLE DE LEITURA DE UMA LISTA DATA

Sempre que temos um conjunto de valores em uma lista DATA, devemos fazer o con trole de leitura para que todos os dados sejam realmente lidos, de forma a não haver dado "esquecidos" ou mesmo tentar ler valores que não existem. O número de valores da lista DATA pode ser conhecido a priori, ou desconhecido.

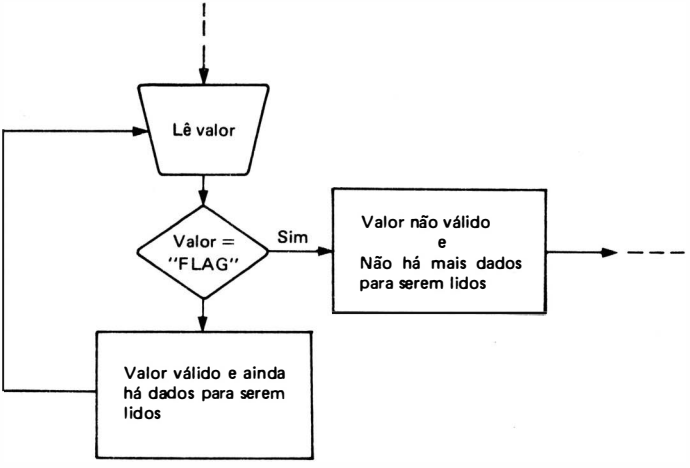
4.1. Número de Valores Desconhecidos

Quando não sabemos a priori quantos serão os valores contidos na(s) lista(s) DATA, a solução para fazermos o controle de leitura dos valores, é colocarmos um valor especial no final da lista de dados, com um valor fictício. Este dado é chamado "FLAG" (bandeira). Para o controle feito a cada leitura de um valor da lista DATA, compara-se o valor lido com o "FLAG". Se o valor lido for igual ao "FLAG", isto indica que já foram lidos todos os valores da lista DATA; se for diferente, indica que ainda há valores na lista DATA para serem lidos.

Devemos levar em consideração que o dado fictício colocado como "Flag" deve estar de acordo com o tipo de dados que estamos lendo, além do que, este dado tem que ser diferente de todos os valores da linha de dados. Assim, se estivermos trabalhando com uma lista al-

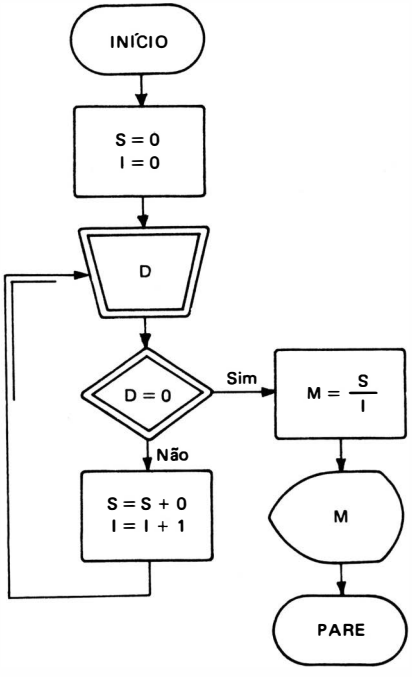
fanumérica, o "FLAG" tem que ser um alfanumérico inédito; se for uma lista numérica, o "FLAG" tem que ser numérico, inalcansável ou impossível para a situação.

O fluxograma genérico deste tipo de controle é o seguinte:



Ex.: Fazer fluxograma e programa BASIC para calcular a média dos diâmetros de um conjunto de esferas metálicas cujos valores estão colocados em uma lista DATA. Não se sabe quantas esferas existem e portanto, foi colocado como último diâmetro o valor fictício d = 0 (zero).

FLUXOGRAMA



PROGRAMA

```
10 S = 0
20 I = 0
30 READ D
40 IF D = 0 THEN GOTO 80
50 S = S + D
60 I = I + 1
70 GOTO 30
80 M = S/I
90 PRINT "MEDIA = "; M
100 STOP
110 DATA 10, 10.5, 9.8, 10.2
120 DATA 9.9, 11.5, 9.7, 10.1
130 DATA 10.9, 11, 10.7, 10
140 DATA 0
```

Podemos notar que a parte do fluxograma genérico deste tipo de controle está presente neste fluxograma, e que o valor do "FLAG" é realmente o último da lista e diferente de todos os anteriores, colocado na linha 140.

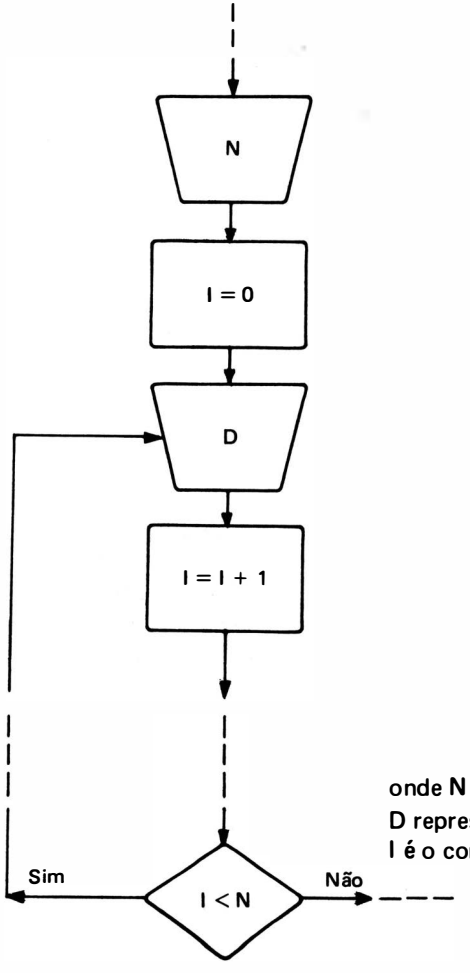
4.2. Número de Valores Conhecidos

Quando conhecemos o número total de valores da lista data, este valor deve ser o primeiro valor a ser lido, separadamente dos dados propriamente ditos.

Para fazer-se o controle, temos basicamente que, a cada valor lido, atualizar um contador de valores e comparar com o número total de valores da lista já lidos separadamente. Isto pode ser feito de 2 modos:

4.2.1. Modo Crescente

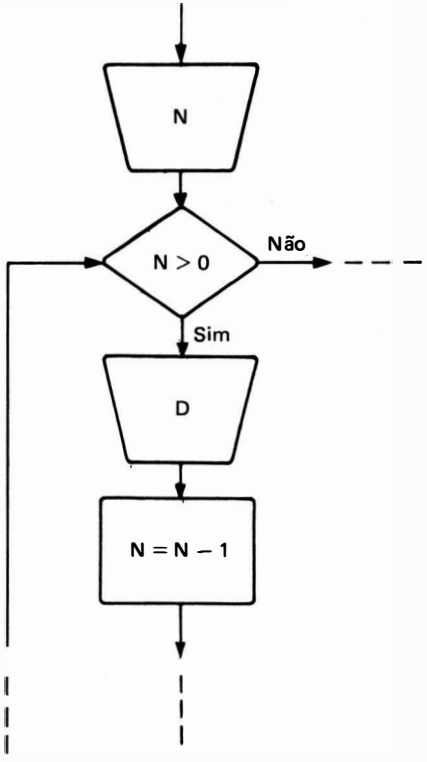
Este método consiste em ler o número total de valores a ser lido (N), e a cada leitura de um dado, um contador (I) é incrementado até chegar ao limite (N), que é o número total de valores. O esquema geral deste método é o seguinte:



onde N representa o número de valores da lista DATA.
D representa cada dado a ser lido.
I é o contador de dados lidos.

4.2.2. Modo Decrescente

Este método consiste em ler o número total de valores a ser lido, e a cada leitura de um dado, um contador é decrementado, até chegar a zero. O esquema geral deste método é o seguinte:

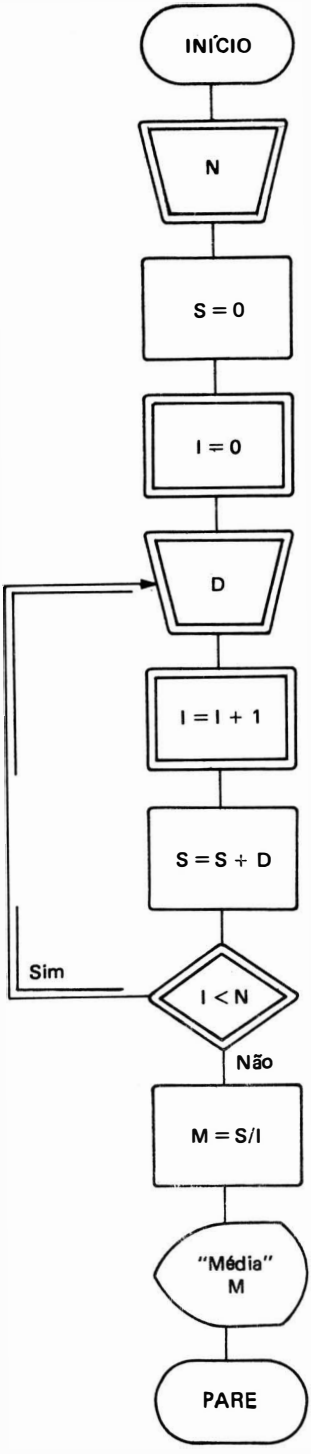


A única inconveniência deste método é que o número total de valores (N), após a leitura de todos os dados valerá zero, o que é facilmente resolvível, atribuindo o valor de N, inicialmente a uma outra variável qualquer.

Ex.: Aplicar os modos crescentes e decrescentes para o problema de cálculo de média das esferas:

a) Modo Crescente

FLUXOGRAMA

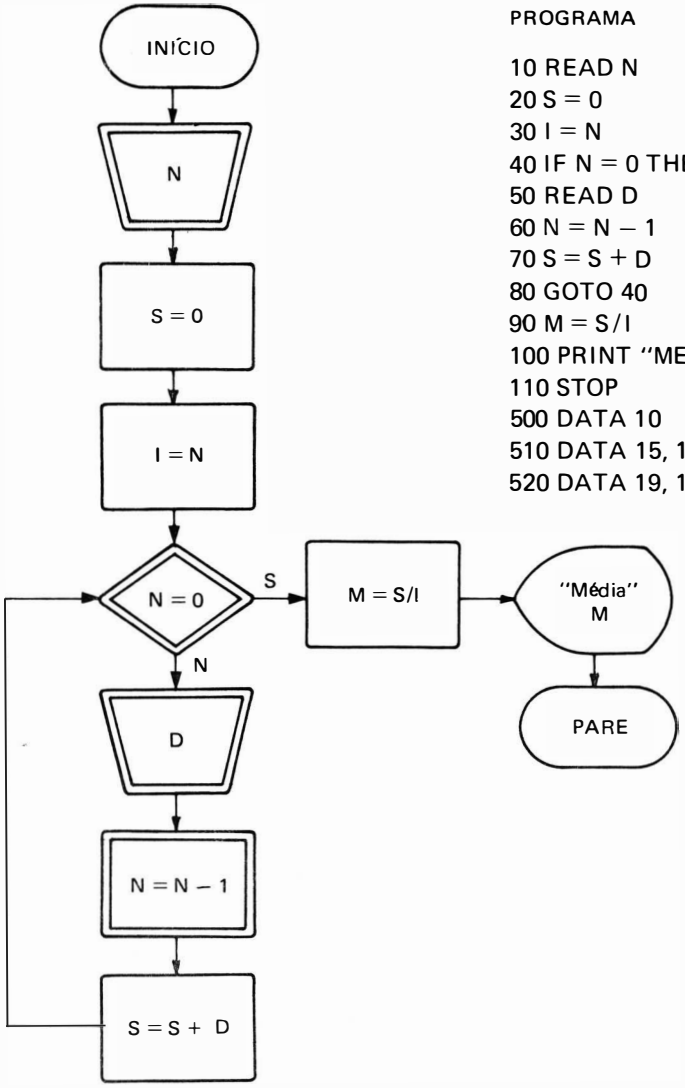


PROGRAMA

```
10 READ N
20 S = 0
30 I = 0
40 READ D
50 I = I + 1
60 S = S + D
70 IF I < N THEN GOTO 40
80 M = S/I
90 PRINT "MEDIA = "; M
100 STOP
500 DATA 10
510 DATA 15, 12, 17, 21, 18
520 DATA 19, 14, 20, 16, 17
```

b) Modo Decrescente

FLUXOGRAMA

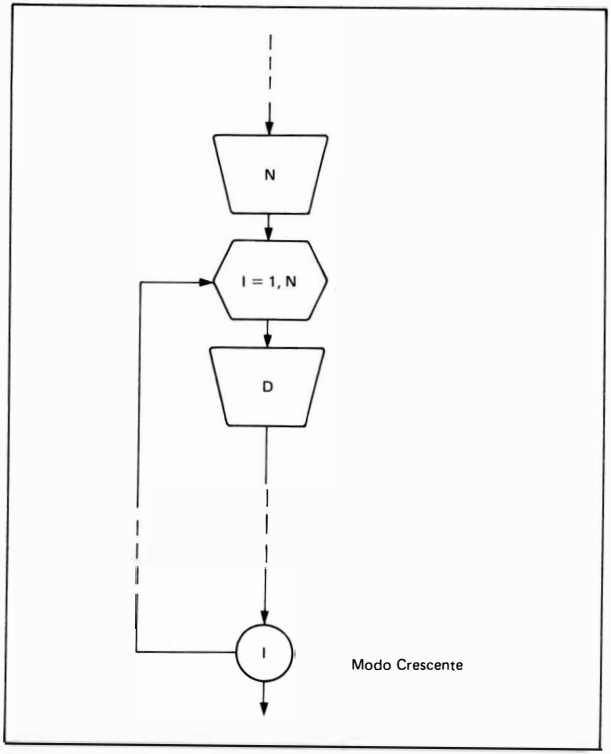


PROGRAMA

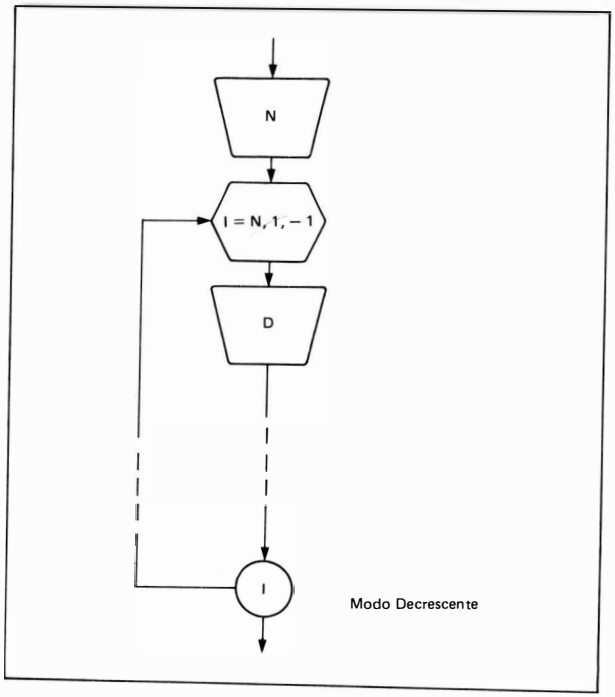
```
10 READ N
20 S = 0
30 I = N
40 IF N = 0 THEN GOTO 90
50 READ D
60 N = N – 1
70 S = S + D
80 GOTO 40
90 M = S/I
100 PRINT "MEDIA = "; M
110 STOP
500 DATA 10
510 DATA 15, 12, 17, 21, 18
520 DATA 19, 14, 20, 16, 17
```

Nos fluxogramas estão marcadas as estruturas gerais de controle de leitura de valores da lista DATA genérica para os dois modos descritos. O valor da lista DATA da linha 500 indica o número de dados que devem ser lidos em ambos os casos.

Devemos ressaltar que, por motivos puramente didáticos, a apresentação dos modos de controle foram feitos com o uso de contadores e comparações com limites, o que podem, obviamente, ser substituídos pela malha iterativa FOR-NEXT, da seguinte forma:



Modo Crescente



Modo Decrescente

5. EXERCÍCIOS

1. Fazer fluxograma e programa BASIC que permita determinar o valor da prestação P a pagar a partir do preço à vista PAV, taxa de juros J e número de parcelas N. Usar as instruções READ/DATA para os valores de PAV. J.

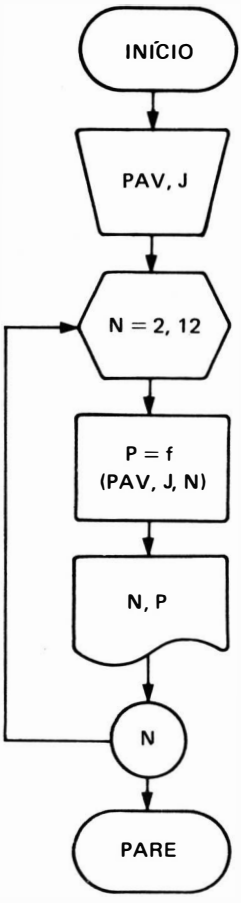
   Fazer uma tabela para o número de prestações e valor de prestação para N de 2 a 12.

   a) Se considerarmos pagamento sem entrada temos:

$$P = \frac{PAV * J}{1 - \frac{1}{(1+J)^N}}$$

   b) Se considerarmos que a primeira parcela seja paga como entrada temos:

$$P = \frac{PAV * J}{1 + J - \frac{1}{(1+J)^{N-1}}}$$



A) CONSIDERANDO O PAGAMENTO SEM ENTRADA:

```
10 READ PAV, J
20 FOR N = 2 TO 12
30 P = (PAV * J)/(1 - 1/(1 + J) ↑ N))
40 PRINT N, P
50 NEXT N
60 STOP
100 DATA 100000, 0.1
```

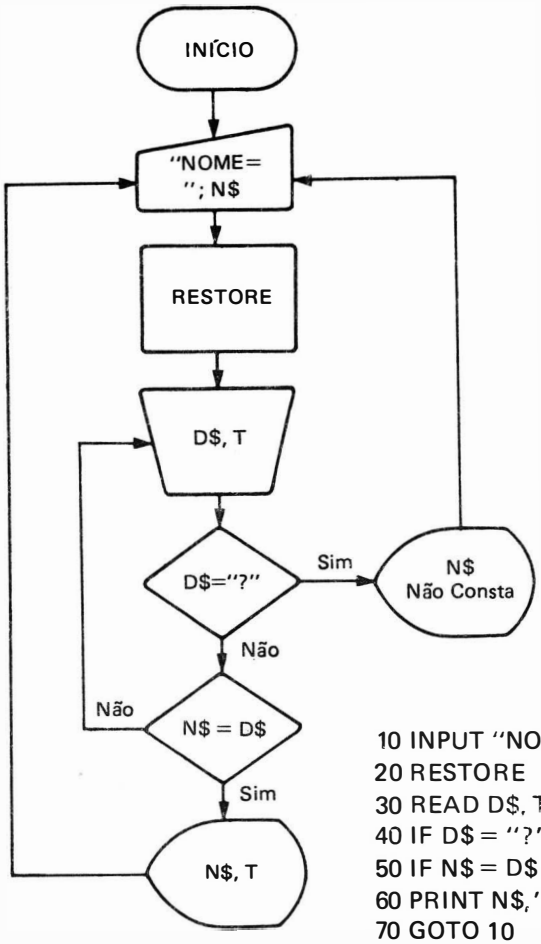
B) CONSIDERANDO A PRIMEIRA PARCELA COMO ENTRADA:

```
10 READ PAV, J
20 FOR N = 2 TO 12
30 P = (PAV * J)/(1 + J - 1/(1 + J) ↑ (N - 1))
40 PRINT N, P
50 NEXT N
60 STOP
100 DATA 100000, 0.1
```

2. Fazer fluxograma e programa BASIC para construir uma agenda telefônica organizada em uma lista de DATA, na forma:

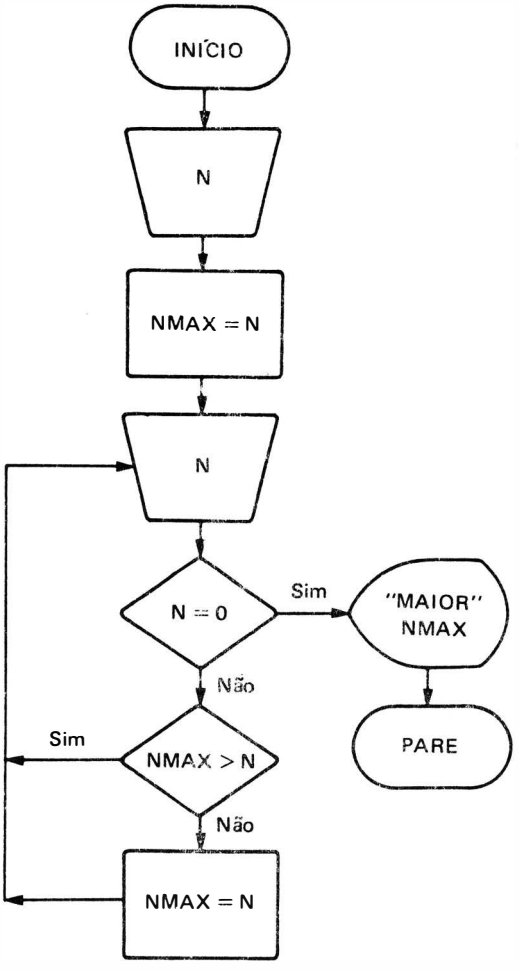
```
DATA "ABEL", 2953045, "MARTA", 2223588
DATA "PAI", 2521133, "TIA", 1113063
  .
   .
    .
```

Deve ser digitado o nome da pessoa que se quer saber o telefone. Colocar como final da lista de dados um que será informado pelo micro marcador qualquer do tipo: DATA "?", 0 Se a lista inteira for pesquisada até chegar ao caracter "?" imprime a mensagem "NAO CONSTA NA LISTA".




```
10 INPUT "NOME = "; N$
20 RESTORE
30 READ D$, T
40 IF D$ = "?" THEN GOTO 80
50 IF N$ = D$ THEN GOTO 30
60 PRINT N$," FONE "; T
70 GOTO 10
80 PRINT N$," NAO CONSTA DA LISTA "
90 GOTO 10
100 DATA "ABEL", 9483063, "MARTHA", 2223588
110 DATA "BENE", 1110055, "CHICO", 3334422
120 DATA "FIRMA", 5551100, "TIA ANA", 2741948
200 DATA "?", 0
```

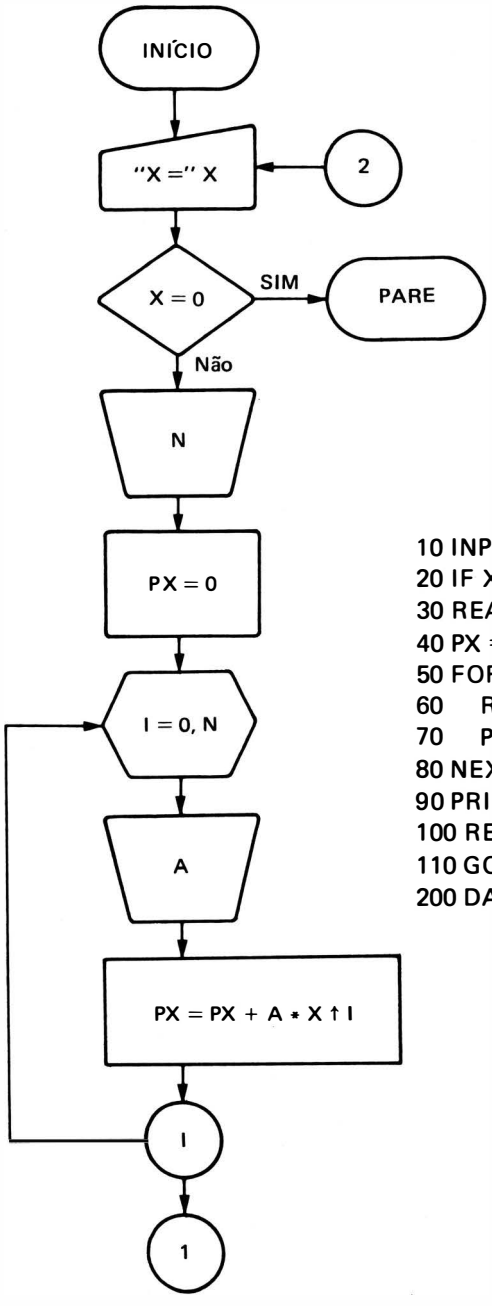
3. Dada uma lista de números em uma declaração DATA, fazer fluxograma e Programa BASIC para achar o maior deles sabendo-se que o último número da lista é zero.

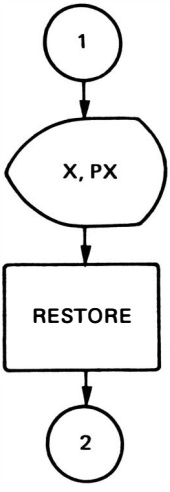



```
10 READ N
20 REM LE O PRIMEIRO NUMERO E ATRIBUI A MAXIMO
30 NMAX = N
40 READ N
50 IF N = 0 THEN GOTO 90
60 IF NMAX > N THEN GOTO 40
70 NMAX = N
80 GOTO 40
90 PRINT "O MAIOR NUMERO E" "; NMAX
100 STOP
200 DATA 5, 7.2, – 3, 100, 95, – 1, 100.9, 4, 0
```

4. Fazer fluxograma e programa BASIC para calcular e exibir o valor de $P(x) = a_0 a_1 x + a_2 x^2 + +. . . a_{n-1} x^{n-1} + a_n x^n$, sendo dados em uma lista DATA o grau N do polinômio e todos os seus coeficientes $(a_0 . . . a_n)$.

   O(s) valor(es) de x para o(s) qual(is) deve(m) ser calculado(s) o(s) P (x) deve(m) ser digitado(s) na entrada. (Encerrar processo se entrar $x = 0$).

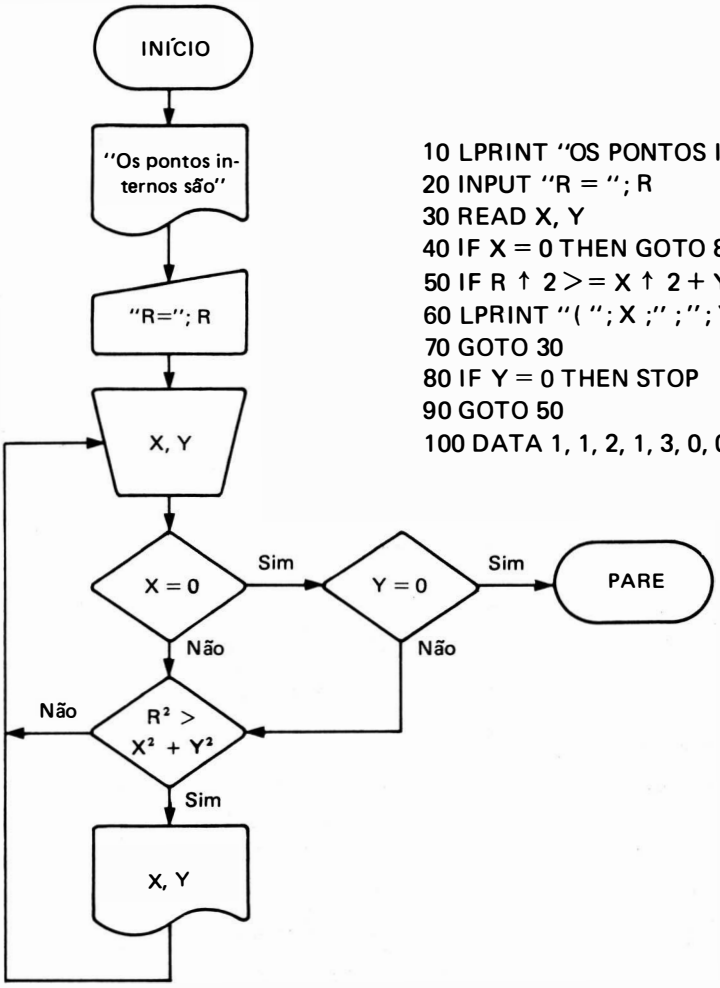





```
10 INPUT "X = "; X
20 IF X = 0 THEN STOP
30 READ N
40 PX = 0
50 FOR I = 0 TO N
60     READ A
70     PX = PX + A * X ↑ I
80 NEXT I
90 PRINT "PARA X = "; X ;" P (X) = "; PX
100 RESTORE
110 GOTO 10
200 DATA 5, 2, 3, – 1, 4, 0, 6
```

N A0 A1 A2 A3 A4 A5

5. Sendo digitado na entrada o valor do raio R de uma circunferência $x^2 + y^2 = R^2$, fazer fluxograma e programa BASIC, determinar os pares x e y de uma lista DATA que correspondem a pontos internos à circunferência ($x^2 + y^2 < R^2$). Terminar o processamento quando for encontrado o par 0, 0 para x e y. Colocar um cabeçalho do tipo: "OS PONTOS INTERNOS SÃO:"




```
10 LPRINT "OS PONTOS INTERNOS SAO :"
20 INPUT "R = "; R
30 READ X, Y
40 IF X = 0 THEN GOTO 80
50 IF R ↑ 2 >= X ↑ 2 + Y ↑ 2 THEN GOTO 30
60 LPRINT "( "; X ;" ";" "; Y ;" )"
70 GOTO 30
80 IF Y = 0 THEN STOP
90 GOTO 50
100 DATA 1, 1, 2, 1, 3, 0, 0, 4, 0, 0
```

6. Dados em uma lista DATA, um conjunto de NP pares de valores (x, y) que representam pontos de um gráfico cartesiano, fazer fluxograma e programa BASIC para determinar os coeficientes A e B da reta $y = A + Bx$ melhor ajustada a estes pontos.

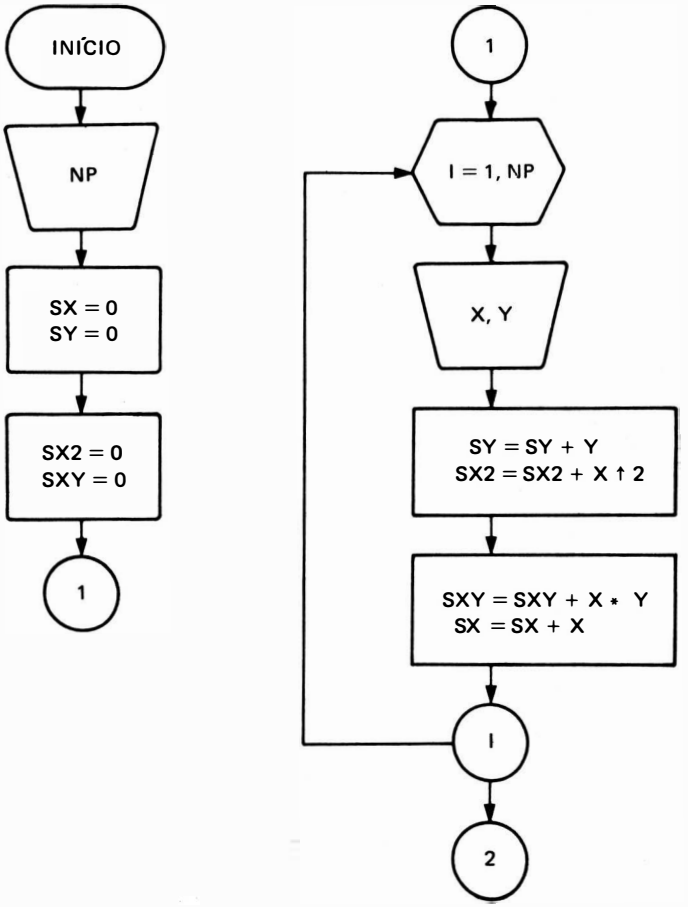
OBS.:

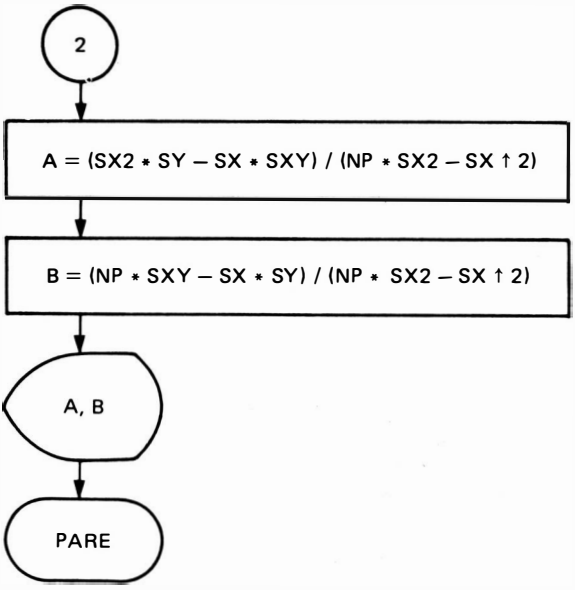
1) a linha DATA contém o valor de *NP* que corresponde ao número de pares *x* e *y* que seguem este valor.

2) A reta $a + bx$ que melhor se ajusta ao conjunto de pontos pode ser calculada por:

$$a = \frac{\Sigma x^2 \cdot \Sigma y - \Sigma xy \cdot \Sigma x}{NP\,\Sigma x^2 - (\Sigma x)^2}$$

$$b = \frac{NP\,\Sigma xy - \Sigma x \Sigma y}{NP\,\Sigma x^2 - (\Sigma x)^2}$$

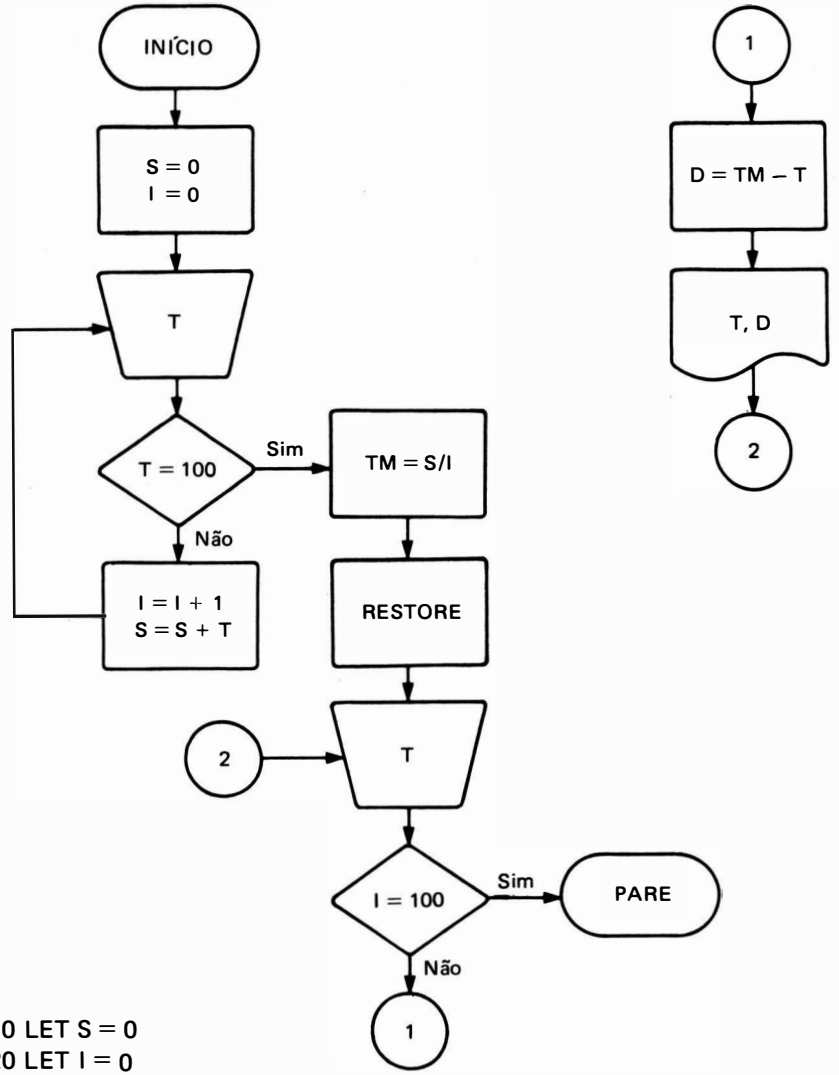





```
10 READ NP
20 SX = 0
30 SY = 0
40 SX2 = 0
50 SXY = 0
60 FOR I = 1 TO NP
70    READ X, Y
80    SX = SX + X
90    SY = SY + Y
100   SX2 = SX2 + X ↑ 2
110   SXY = SXY + X * Y
120 NEXT I
130 A = (SX2 * SY – SX * SXY)/(NP * SX2 – SX ↑ 2)
140 B = (NP * SXY – SX * SY)/(NP * SX2 – SX ↑ 2)
150 PRINT "A RETA MELHOR AJUSTADA E' : ("; A ;") + ("; B ;") X"
160 STOP
200 DATA 5, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5
```

7. Fazer fluxograma e programa BASIC para calcular a temperatura ambiente média de um conjunto de valores de uma lista DATA sabendo-se que o último valor da lista vale 100 e é um valor fictício que deve ser ignorado. Fazer uma tabela de temperatura e desvio em relação a média: $D = TM - T$, onde:

D = Desvio
TM = Temperatura Média
T = Cada Temperatura




```
10 LET S = 0
20 LET I = 0
30 READ T
40 IF T = 100 THEN GOTO 80
50 LET I = I + 1
60 LET S = S + T
70 GOTO 30
80 LET TM = S/I
90 RESTORE
100 READ T
110 IF T = 100 THEN STOP
120 LET D = TM – T
130 PRINT T, D
140 GOTO 100
200 DATA 20, 31, 27, 39, 40, 38, 29, 25,
210 DATA 100
```

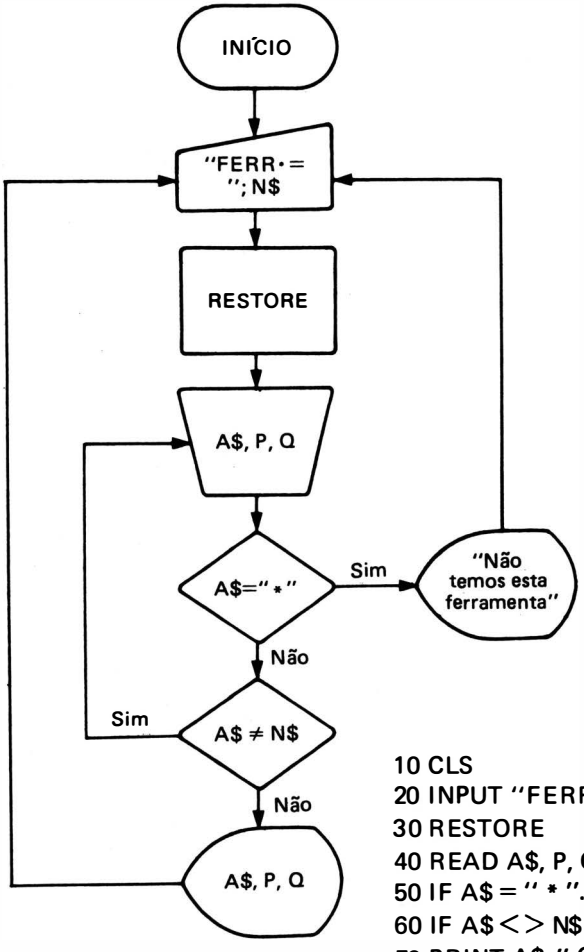
8. Uma loja de ferramentas tem junto ao caixa um microcomputador, que permite ao freguês, digitando o nome da ferramenta, saber se a mesma existe, e em caso afirmativo, o seu preço e a quantidade mínima a ser comprada, e se tem 10% de desconto.

   Elaborar programa, sabendo-se que a loja vende as seguintes ferramentas:

   - formão, Cr$ 1.253,10 unidades
   - serrote, Cr$ 1.532,15 unidades
   - martelo, Cr$ 121,20 unidades
   - alicate, Cr$ 1.800,3 unidades

   .
   .
   .

   Sabe-se que como marcador de final de lista ferramentas do tipo " * "




```
10 CLS
20 INPUT "FERRAMENTA = "; N$
30 RESTORE
40 READ A$, P, Q
50 IF A$ = " * ". THEN GOTO 110
60 IF A$ <> N$ THEN GOTO 40
70 PRINT A$;" CR$"; P ;" MINIMO PARA DESCONTO :
   "; Q ;" UNIDADES"
80 FOR I = 1 TO 1000
```

```
90 NEXT I
100 GOTO 10
110 PRINT "NÃO TEMOS "; N$
120 GOTO 80
200 DATA "FORMAO", 1253, 10
210 DATA "SERROTE", 1532, 15
220 DATA "MARTELO", 121, 20
230 DATA "ALICATE", 1800, 3
300 DATA "*", 0, 0
```

Neste programa, a linha 10 limpa a tela de video (Ver Cap. 15) deixando-a preparada para impressão de mensagens. A linha 30 coloca o ponteiro da lista DATA no primeiro valor da primeira instrução DATA, ou seja, na palavra "FORMAO" as linhas 80 e 90 fazem uma "pausa" no processamento para que possa ser lida a mensagem enviada pelo micro, enquanto é executado internamente um "loop" de 1 a 1000.

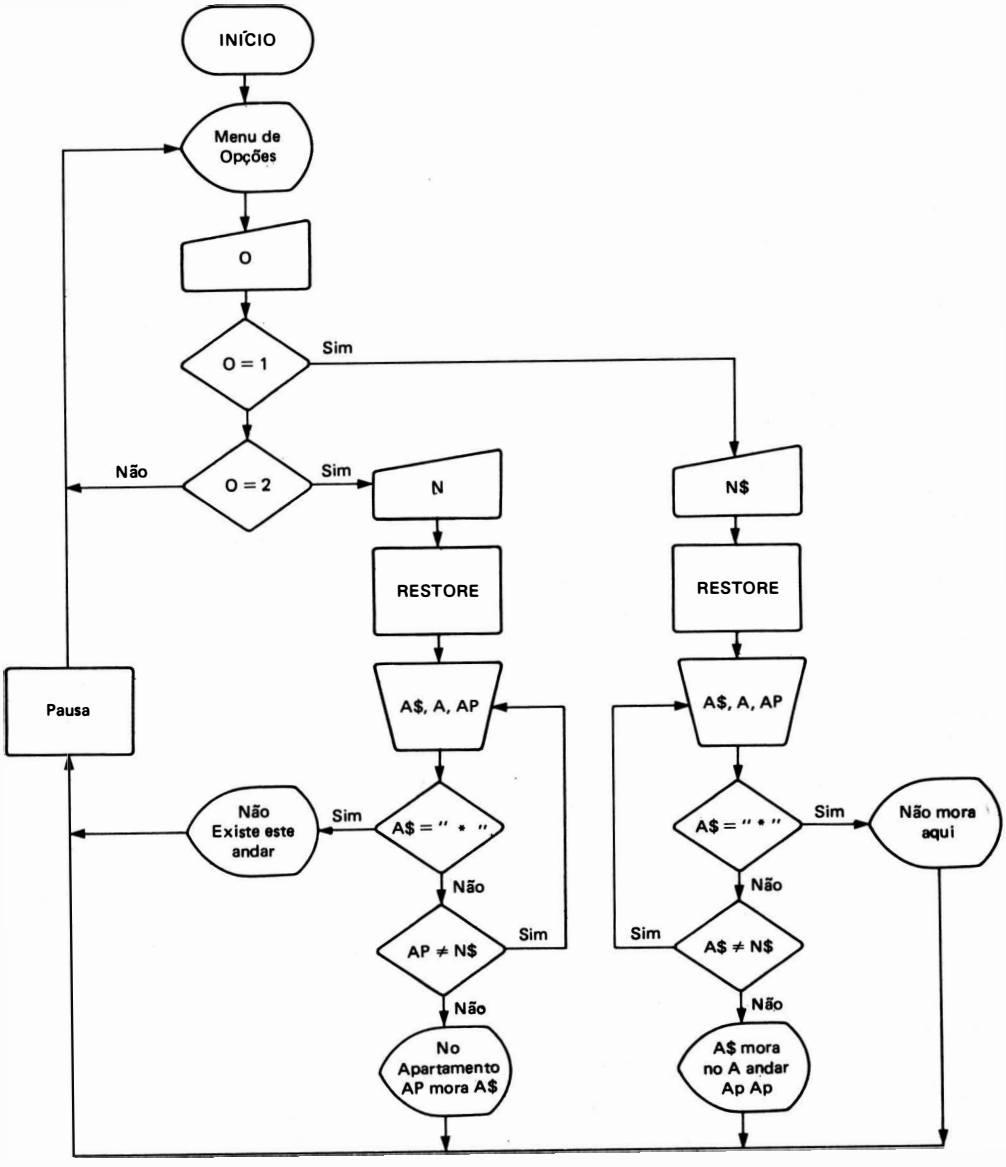
9. No hall de entrada de um grande edifício foi colocado um microcomputador com a finalidade de fornecer aos visitantes o andar e o número do apartamento da pessoa que o habita, quando este nome for digitado, ou ainda fornecer o nome da pessoa que habita o apartamento do nº que foi digitado.

   O micro deve indicar também se o nome digitado não consta da lista de nomes de pessoas, ou se o número do apartamento não existe.

   Elaborar fluxograma e programa BASIC para este micro, conhecendo-se a lista de moradores do edifício:

Nome	Andar	Apartamento
João	1	101
Pedro	1	102
Maria	2	201
.	.	.
.	.	.
.	.	.
Alcides	7	702
Patrícia	8	801
.	.	.
.	.	.
.	.	.
Antonio	10	1 001
.	.	.
.	.	.
.	.	.

Colocar um marcador como final de lista de nomes do tipo "*"


INÍCIO
Menu de Opções
O
O = 1
Sim
Não
O = 2
Sim
N
N$
RESTORE
RESTORE
Pausa
A$, A, AP
A$, A, AP
Não Existe este andar
Sim
A$ = " * "
Não
AP ≠ N$
Sim
Não
No Apartamento AP mora A$
A$ = " * "
Sim
Não mora aqui
Não
Sim
A$ ≠ N$
Não
A$ mora no A andar Ap Ap

```
10 CLS
20 PRINT "RELACAO DE MORADORES"
30 PRINT
40 PRINT
50 PRINT "(1) ENTRAR COM NOME DO MORADOR"
60 PRINT
70 PRINT "(2) ENTRAR COM O NUMERO DO APARTAMENTO
80 PRINT
90 INPUT "OPCAO = "; 0
100 IF 0 = 1 THEN GOTO 140
110 IF 0 = 2 THEN GOTO 300
120 GOTO 10
130 REM FINAL DO MENU.
140 INPUT "NOME = "; N$
150 RESTORE
160 READ A$, A, AP
170 REM A$ = VARIAVEL PARA NOME
180 REM A  = VARIAVEL PARA ANDAR
190 REM AP = VARIAVEL PARA APARTAMENTO
200 IF A$ = "* " THEN GOTO 250
210 IF A$ <> N$ THEN GOTO 160
220 PRINT
230 PRINT A$;" MORA NO"; A ;" 0. ANDAR, APARTAMENTO NO. "; AP
240 GOTO 270
250 PRINT
260 PRINT A$;" NAO MORA NESTE EDIFICIO"
270 FOR I = 1 TO 1000
280 NEXT I
290 GOTO 10
300 INPUT "NO. DO APARTAMENTO = "; N
310 RESTORE
320 READ A$, A, AP
330 IF A$ = "* " THEN GOTO 380
340 IF AP <> N THEN GOTO 320
350 PRINT
360 PRINT "NO APARTAMENTO"; AP ;" MORA  "; A$
370 GOTO 270
380 PRINT
390 PRINT "NAO EXISTE ESTE APARTAMENTO"
400 GOTO 270
500 DATA "JOAO", 1, 101
510 DATA "PEDRO", 1, 102
520 DATA "MARIA", 2, 201
530 DATA "ALCIDES", 7, 702
540 DATA "PATRICIA", 8, 801
550 DATA "ANTONIO", 10, 1001
800 DATA "* ", 0, 0
```

CAPÍTULO 9

OPERAÇÕES COM TABELAS E MATRIZES

1. VARIÁVEL INDEXADA

As variáveis BASIC utilizadas até agora (chamadas variáveis escalares), correspondem a uma única localidade de memória, na qual apenas um valor é armazenado. Entretanto, é possível operarmos com um conjunto de localizações distintas e contíguas da memória. Estes conjuntos são denominados matrizes, tabelas ou vetores, sendo compostos por itens. Trata-se de um único endereço de memória no qual podemos armazenar vários valores, que são localizados por meio de índices.

Como sabemos, os endereços de memória são associados aos nomes das variáveis, que neste caso passa a ter um índice e chamada variável indexada ou subscritada. Portanto, a variável indexada é composta pelo nome da variável seguida de um índice entre parênteses.

Ex.: Álgebra	BASIC
x_3	X (3)
a_i	A (I)
$r_{i,j,k}$	R (I, J, K)
$k_{i,j}$	K (I, J)

onde o valor do índice corresponde a ordem, ou a posição do elemento na tabela. Então, X (3) indica o terceiro elemento da tabela X; A (I) representa o conjunto de elementos que podem estar armazenados na localidade de memória A, e cada elemento terá um valor de índice I em particular.

Ex. 1:

A (I, J) =

21	48	50
30	15	− 12
− 17	20	50

então A (3, 1) = − 17

Ex. 2

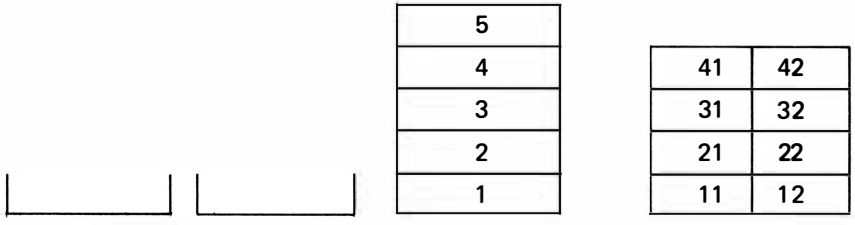
B (K) =

B (0)	B (1)	B (2)	B (3)	B (4)	B (5)
9	– 5	– 3	– 2	0	5

Obs.: É válido, em linguagem BASIC, uso do índice Ø (zero), como no exemplo, B (Ø) é uma localidade da variável indexada B (K). No primeiro exemplo, apesar de existir a localidade A (0, 0), esta foi desconsiderada para melhor analogia com as matrizes matemáticas.

Uma analogia com o cotidiano, seria termos a memória do computador como sendo uma rua, que possui várias casas e prédios de apartamentos. Em cada casa mora apenas uma pessoa, o que corresponderia a uma variável simples (escalar), sendo que neste endereço (correspondente ao nome da variável) podemos armazenar apenas um valor. Para as variáveis indexadas, a analogia seria a do prédio onde em um único endereço existem vários apartamentos, em cada qual reside apenas uma pessoa, que é localizado por um índice do número do andar e do apartamento.

Esquematicamente:



As variáveis A e B são escalares, e representam uma casa térrea; a variável C (I) um prédio de apartamentos com um apartamento por andar; a variável D (I, J) um prédio de apartamentos com 2 apartamentos por andar.

Forma geral:

$v\,(id_1)$ ou $v\,(id_1, id_2)$

onde v é o nome da variável
id_1, id_2 são os índices da variável indexada

O número de índices colocados entre parênteses, determina a dimensão da variável indexada. Assim uma variável com apenas um índice, do tipo X (I), é dita unidimensional e variáveis de dois índices, do tipo X (I, J), é dita bidimensional. Existem, entretanto, sistemas BASIC que admitem 5,6 ou qualquer dimensão para a variável indexada, sendo limitados apenas pelo tamanho da memória do computador.

É comum chamarmos uma variável indexada unidimensional de vetor, e a bidimensional de matriz, fazendo analogia com matemática.

O *nome* de uma variável indexada, é formado de acordo com as mesmas regras usadas para dar nomes às variáveis não indexadas (ou escalares).

Exs. de variável indexada:

A (I) X (M, N) S (50)
C (K) B$ (L)

Não são variáveis indexadas:

(I) — falta o nome da variável
1X (J) — nome da variável não começa por letra
B (3X) — nome de índice não começa por letra

1.1. Utilização

Existem problemas que precisamos criar centenas, ou mesmo milhares de variáveis escalares para armazenarmos elementos diferentes, o que não é prático nem conveniente. Neste caso utiliza-se um conjunto de variáveis com o mesmo nome.

Ex.: A matriz A de elementos a_{ij} com 10 linhas e 10 colunas, se não for tratada por uma variável indexada, serão necessários criar 100 nomes de variáveis do tipo A11, A12, . . . , A1010; e com a utilização da variável indexada A (I, J), variando I de 1 a 10 e J de 1 a 10 temos as 100 localidades de memória necessária em apenas um endereço.

2. INSTRUÇÃO DIM

É uma forma abreviada da palavra DIMENSION e permite ao computador reservar uma área na memória capaz de armazenar o número máximo de elementos da variável indexada. Este comando supre o computador das informações necessárias para a identificação da variável indexada, tais como nome, dimensão e valor máximo que cada índice pode assumir.

Forma Geral:

nl DIM v id1	ou	nl DIM v (id1, id2)

onde v é o nome de uma variável indexada
nl indica o número da linha
id1, id2 são os valores máximos que cada índice irá assumir no programa (sempre constantes inteiras positivas).

Ex. 1:

```
10 DIM A (15)
```

Significado: O programa utilizará uma variável numérica indexada de nome A, unidimensional com no máximo 16 elementos, ou itens A (I) com $0 \leqslant I \leqslant 15$.

Ex. 2:

```
15 DIM X$ (80)
```

Significado: O programa utilizará uma variável alfanumérica indexada, de nome X$, unidimensional com no máximo 81 caracteres.

X$ (I) com $0 \leqslant I \leqslant 80$.

Dependendo do microcomputador utilizado, esta mesma instrução pode ter um significado totalmente diferente, ou seja, a instrução DIM X$ (80) pode significar que o programa utilizará uma variável indexada alfanumérica, unidimensional com no máximo 81 itens X$ (I), sendo que cada um destes itens pode conter uma lista de até 255 caracteres alfanuméricos.

Ex. 3:

```
30 DIM B (100, 100)
```

Significado: O programa utilizará uma variável numérica indexada, de nome B, bidimensional, com no máximo 10.201 elementos (101 × 101), ou itens B (I, J) com $0 \leqslant I \leqslant 100$ e $0 \leqslant J \leqslant 100$.

3. REGRAS PARA USO DE VARIÁVEL INDEXADA

3.1. O nome usado para uma certa variável, não pode ser usado como nome de outra variável indexada ou mesmo variável simples.

Ex.: Num mesmo programa não pode haver variáveis do tipo: X, X (I) ou X (I, J), pois todas elas têm o mesmo nome (X), o que representa um mesmo endereço de memória.

3.2. A instrução DIM deve vir sempre *antes* de qualquer referência a uma variável indexada.

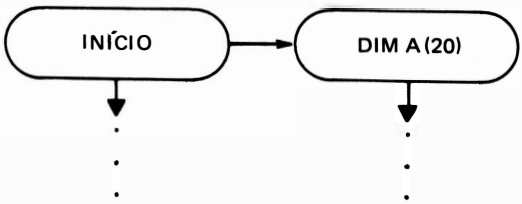
3.3. Num programa não se deve redimensionar uma mesma variável.

Com as duas últimas regras verificamos que, por medida de segurança, é mais conveniente DIMensionarmos as variáveis indexadas no início do programa, de forma que num eventual retorno ao início do programa, o fluxo do processamento não passe novamente pela instrução DIM, o que seria entendido como redimensionamento da(s) variável(eis).

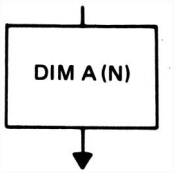
A maioria dos sistemas BASIC entretanto, não necessita de dimensionamento de variáveis indexadas, se o número de itens for menor que 10, com uma única dimensão.

A instrução DIM é uma instrução não executável, isto é, uma instrução que apenas informa o computador das características de variáveis, servindo apenas para consulta.

No fluxograma é comum colocarmos a instrução do dimensionamento (DIM) no bloco que corresponde ao início do programa. Por exemplo:



Caso a instrução DIM precise ficar no interior do programa, esta deverá ser indicada no fluxograma dentro de um bloco de atribuições ou operações internas, tomando sempre o cuidado para não dimensionar mais de uma vez uma mesma variável. Por exemplo:



4. USO DE ÍNDICES NA VARIÁVEL INDEXADA

Conforme já foi dito anteriormente, o valor do índice corresponde a posição do elemento na tabela. Este índice pode ser escrito de várias maneiras, sendo:

v = variável numérica (para o índice)
c = constante inteira positiva

O(s) índice(s) da variável indexada pode ser escrito conforme a tabela:

Forma do índice	Exemplo
c	X (5)
v	A (I)
v + *c*	B (J + 5)
v − *c*	X (K − 2)
c * *v* + *c'*	M (3 * K + 1)
c * *v* − *c'*	L (5 * J − 2)

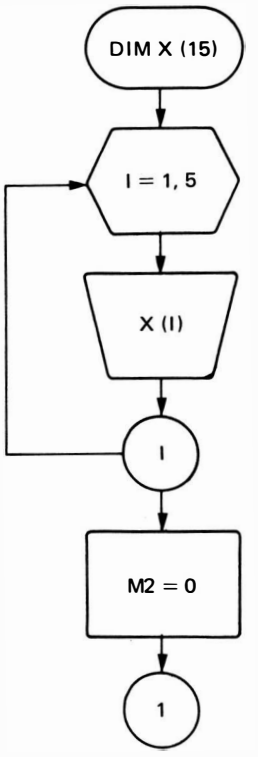
Estas são as formas mais usuais para os índices de uma variável indexada, embora a maioria dos atuais computadores com linguagem BASIC, aceitem qualquer tipo de expressão para o cálculo do índice.

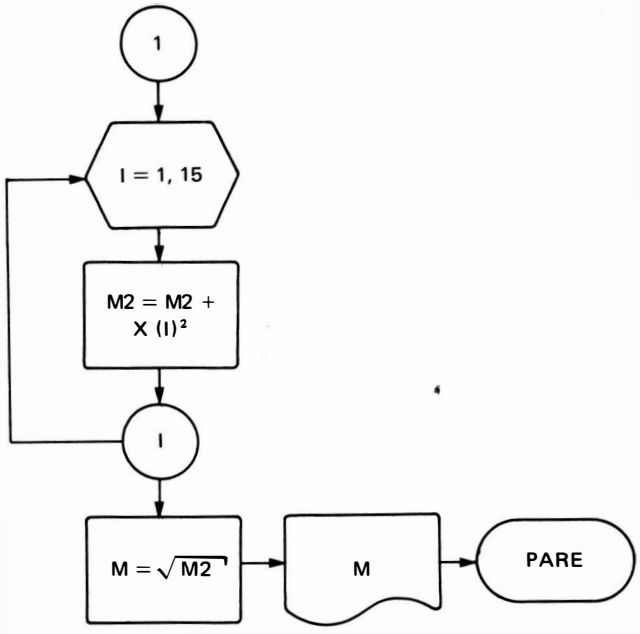
Como observação final, devemos verificar que o índice de uma variável indexada, pela sua natureza, tem que ser sempre um número inteiro positivo, pois este aponta para uma localidade de memória, que são posições discretas, não existindo, por exemplo, a localidade 17,5 de uma tabela, somente a posição 17 ou a posição 18.

Exemplo de utilização de uma indexada:

- Calcular o módulo de um vetor de 15 coordenadas, armazenando-as em uma tabela $X(I) \cdot (1 \leqslant I \leqslant 15)$.

$$M = \sqrt{X(1)^2 + X(2)^2 + \ldots + X(15)^2}$$

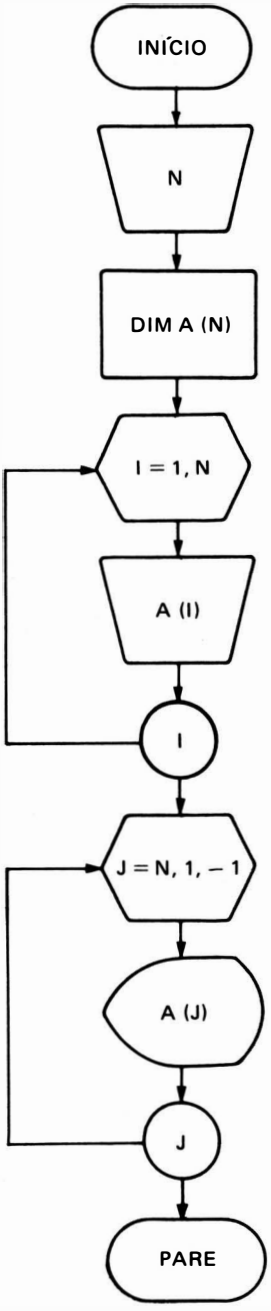





```
10 DIM X (15)
20 FOR I = 1 TO 15
30     READ X (I)
40 NEXT I
50 M2 = 0
60 FOR I = 1 TO 15
70     M2 = M2 + X (I) ↑ 2
80 NEXT I
90 M = SQR (M2)
100 PRINT "MODULO DO VETOR = "; M
110 STOP
200 DATA 1, – 1, 5, 6, 4
210 DATA 3, – 3, 15, 18, 12
220 DATA 0, 1, – 5, 3, 2
```

5. EXERCÍCIOS

1. Em uma lista DATA são dados o número de elementos N e os N valores de volumes de peças cúbicas. Ler os volumes desta lista em um vetor A (I) e imprimi-los em ordem oposta a que forem lidos.

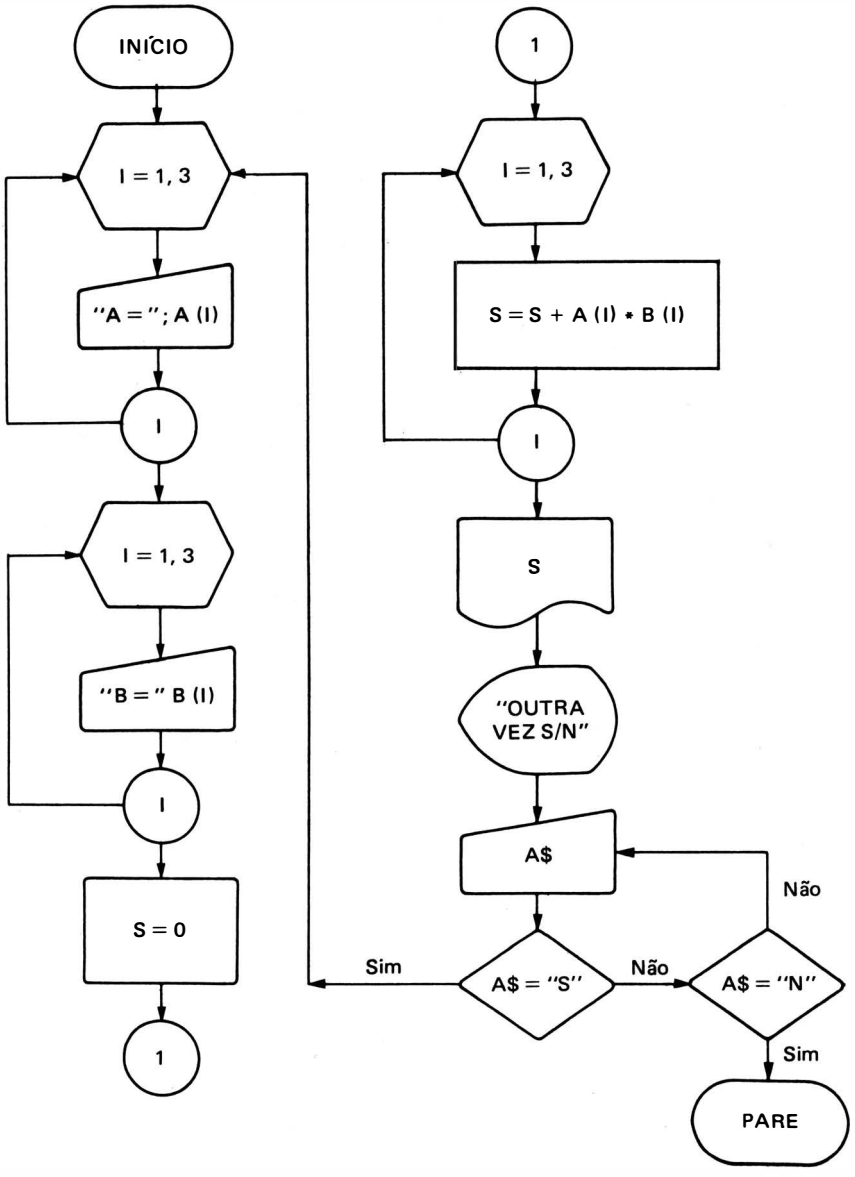



```
10 READ N
20 DIM A (N)
30 FOR I = 1 TO N
40     READ A (I)
50 NEXT I
60 FOR J = N TO 1 STEP – 1
70     PRINT A (J);
80 NEXT J
90 STOP
100 DATA 10, 50, 65, 32, 44, 51
110 DATA 60, 100, 80, 35, 94
```

2. Calcular o produto escalar de 2 vetores A e B de 3 coordenadas.
As 3 coordenadas de A, ou seja, (x_1, y_1, z_1) e as 3 coordenadas de B, ou seja, (x_2, y_2, z_2) são digitadas na entrada. Encerrar o processo respondendo a pergunta "outra vez?". Se a resposta for "S", repete o programa; se for "N" pára, e se não for nenhuma das duas letras repete a pergunta.

Por definição

$$A * B = \sum_{i=1}^{3} ai * bi$$



PROGRAMA BASIC

```
10 FOR I = 1 TO 3
20    INPUT " A = "; A (I)
30 NEXT I
40 FOR I = 1 TO 3
50    INPUT " B = "; B (I)
60 NEXT I
70 S = 0
80 FOR I = 1 TO 3
90    S = S + A (I) * B (I)
100 NEXT I
110 PRINT "PRODUTO ESCALAR = "; S
120 PRINT
130 PRINT "OUTRA VEZ ? (S OU N)";
140 INPUT A$
150 IF A$ = "S" THEN GOTO 10
160 IF A$ = "N" THEN STOP
170 GOTO 140
```

Comentários:

— O primeiro loop FOR-NEXT (limites 10 a 30) faz a entrada das três coordenadas do vetor A enquanto o segundo (linhas 40 a 60) faz a entrada das três coordenadas do vetor B.

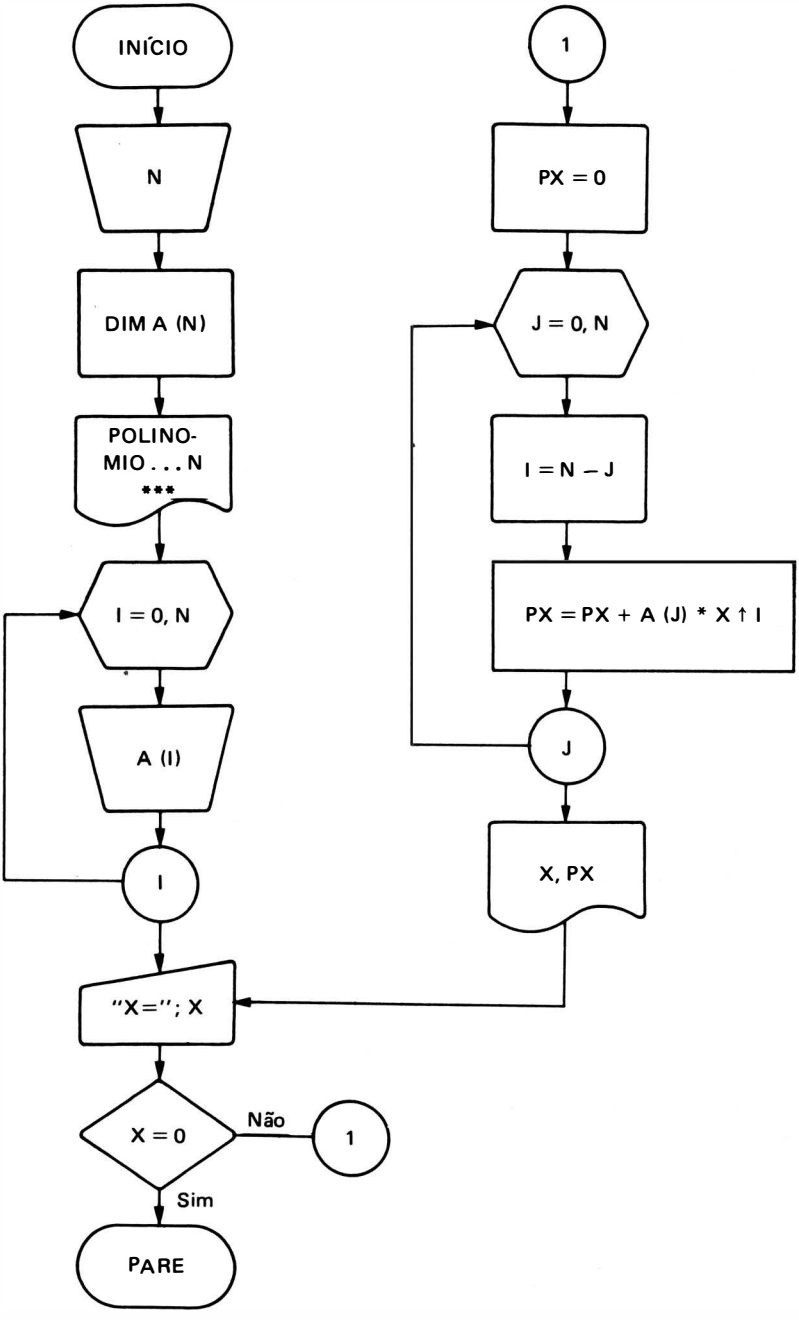
— A linha 70 S = 0 inicia a somatória que ocorre na linha 90, onde é calculado o produto escalar.

— As linhas 130 a 170 fazem o controle para executar novamente o programa. Note-se que apenas as letras S ou N são aceitas, pois se for digitada uma letra diferente desta será feito o retorno para entrar nova opção.

3. Em uma lista DATA são dados N (grau do polinômio) e todos os coeficientes ($a_0, a_1, \ldots a_n$) do polinômio.

$$P(x) = a_0 X^n + a_1 X^{n-1} + \ldots a_{n-1} X + a_n$$

Fazer programa para calcular o valor de $P(X)$ para todos os valores de x que forem digitados na entrada. Encerrar processo quando $x = 0$. Imprimir cabeçalho conforme a figura:


INÍCIO
N
DIM A (N)
POLINO-
MIO . . . N
***
I = 0, N
A (I)
I
"X="; X
X = 0
Não
1
Sim
PARE
1
PX = 0
J = 0, N
I = N – J
PX = PX + A (J) * X ↑ I
J
X, PX

PROGRAMA BASIC

```
10 READ N
20 DIM A (N)
30 LPRINT "POLINOMIO DE GRAU "; N
40 FOR I = 1 TO 20
50    LPRINT "*";
60 NEXT I
70 FOR I = 0 TO N
80    READ A (I)
90 NEXT I
100 INPUT "X = "; X
110 IF X = 0 THEN STOP
120 PX = 0
130 FOR J = 0 TO N
140   I = N – J
150   PX = PX + A (J) * X ↑ I
160 NEXT J
170 LPRINT "PARA X = "; X ;" P (X) = "; PX
180 GOTO 100
200 DATA 5, 1, – 1, 0, 3, – 2
```

Comentários:

— Em problema com cabeçalho é recomendável que a sua impressão seja uma das primeiras tarefas do programa, pois se for deixado para ser impresso junto com a saída normal, haverá uma impressão de cabeçalho para cada resultado que for exibido, o que é indesejável.

4. Um teste composto de 10 questões foi proposto numa classe. Cada questão admitindo 5 alternativas identificadas pelos números de 1 a 5. Querendo usar seu computador pessoal para corrigir o teste, o professor organizou os alunos em várias listas DATA nas quais constavam o número de alunos (N) e suas 10 respostas do tipo:

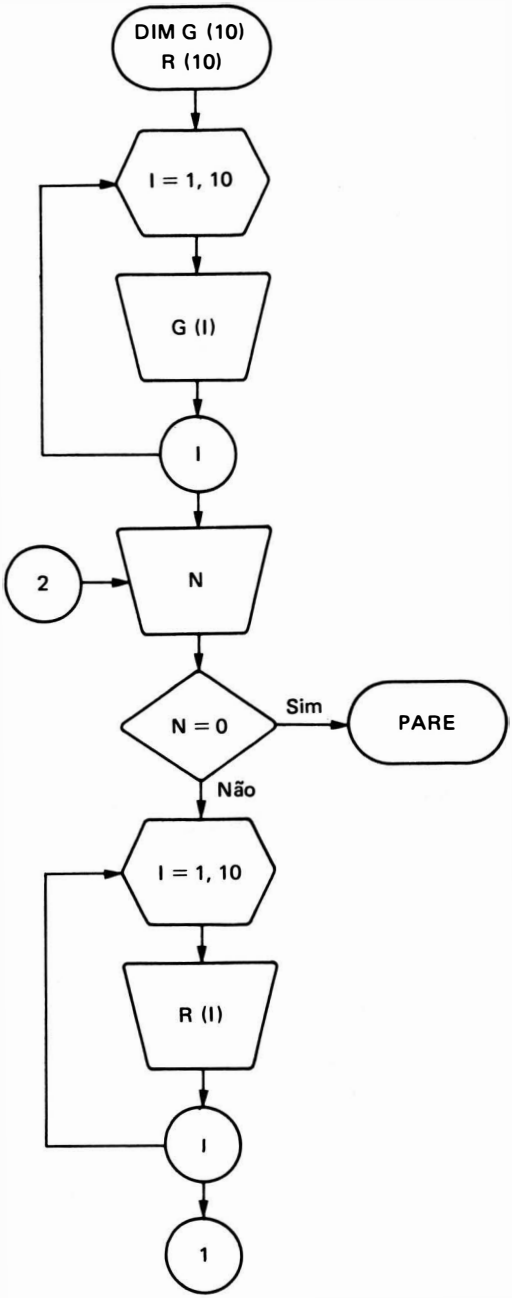
DATA 17, 1, 2, 1, 3, 5, 4, 2, 1, 2, 3

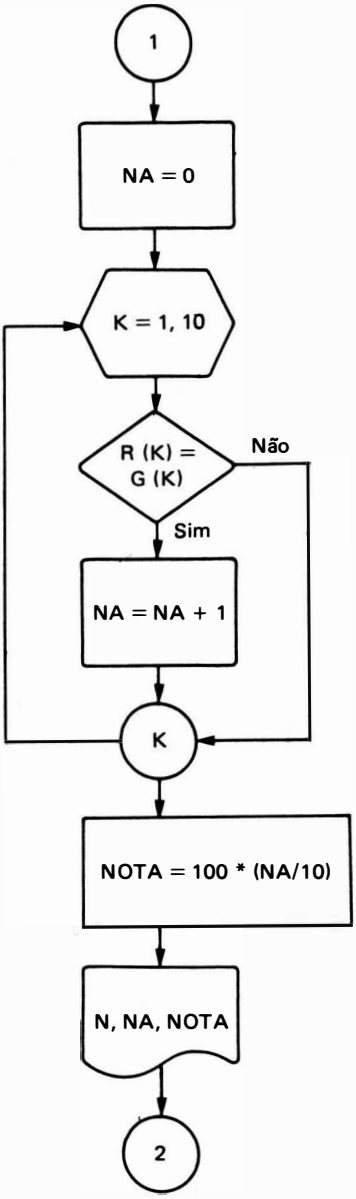
colocou também em outra lista DATA o gabarito do tipo:

DATA 1, 3, 1, 4, 5, 4, 2, 3, 1, 3

Para identificar o término da leitura dos alunos colocou para o último aluno um valor fictício do número do aluno valendo 0.

Fazer programa que imprima o número do aluno, número de acertos e nota do teste (de 0 a 10).






```
10 DIM G (10), R (10)
20 REM LEITURA DO GABARITO
30 FOR I = 1 TO 10
40     READ G (I)
50 NEXT I
```

```
60 REM LEITURA DO NUMERO DO ALUNO (N)
70 READ N
80 IF N = 0 THEN STOP
90 REM LEITURA DAS RESPOSTAS DO ALUNO NUMERO N
100 FOR I = 1 TO 10
110   READ R (I)
120 NEXT I
130 REM CORRECAO DO TESTE DO ALUNO NUMERO N
140 NA = 0
150 FOR K = 1 TO 10
160   IF R (K) <> G (K) THEN GOTO 180
170   NA = NA + 1
180 NEXT K
190 REM CALCULO DA NOTA E SAIDA
200 NOTA = 100 * (NA/10)
210 LPRINT N, NA, NOTA
220 GOTO 70
250 REM GABARITO
260 DATA 1, 3, 1, 4, 5, 4, 2, 3, 1, 3
270 REM NUMERO DO ALUNO E RESPOSTAS
280 DATA 1, 1, 3, 1, 5, 4, 5, 2, 3, 1, 3
290 DATA 2, 1, 3, 2, 4, 5, 1, 3, 1, 3, 2
300 DATA 3, 2, 2, 2, 1, 2, 4, 1, 2, 3, 3
310 DATA 4, 1, 3, 1, 4, 5, 4, 2, 3, 1, 3
320 DATA 5, 1, 3, 2, 4, 5, 1, 2, 1, 2, 1
330 DATA 6, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5
340 DATA 0
```

5. Dada a matriz A (n, n) escrever programa BASIC que calcule:

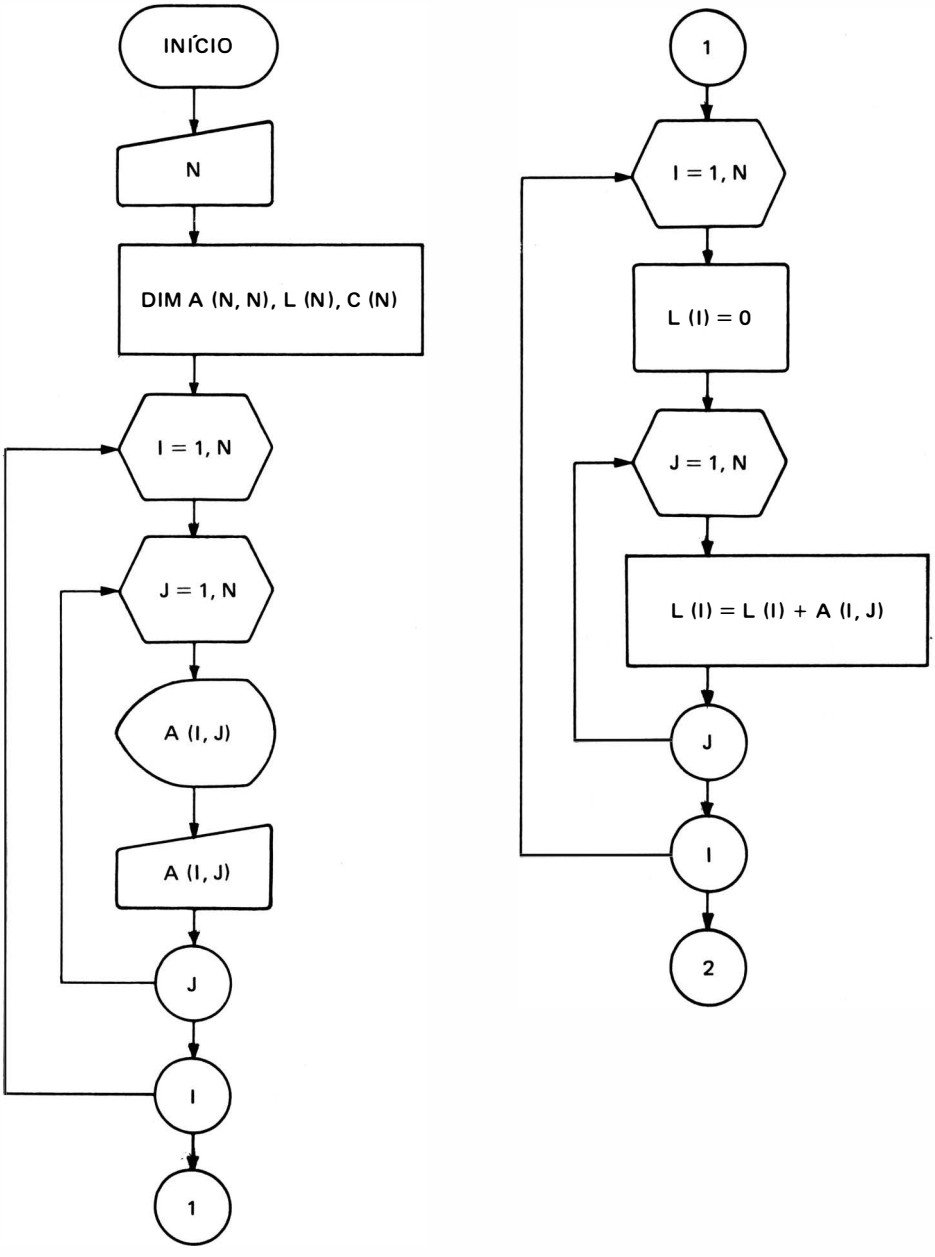
   a) O vetor dos elementos das linhas da matriz (L (I)).

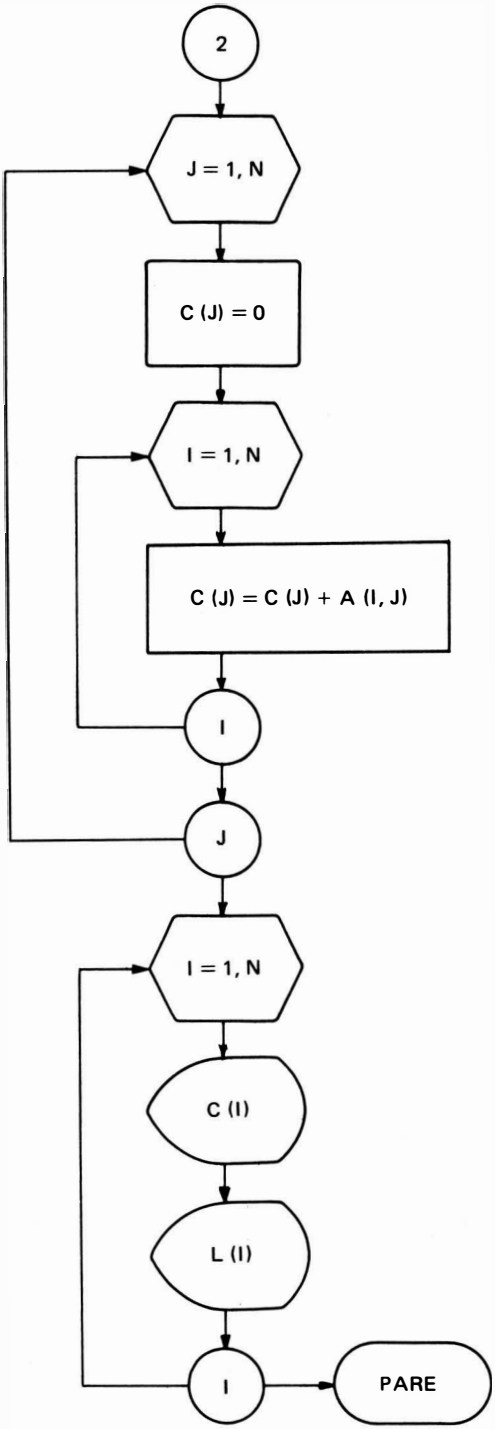
   b) O vetor soma dos elementos das colunas da matriz (C (I)).

$$\begin{bmatrix} a_{11}, a_{12} \ldots a_{1n} \\ a_{21}, a_{22} \ldots a_{2n} \\ \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \\ a_{n1}, a_{n2} \ldots a_{nn} \end{bmatrix} \qquad \begin{bmatrix} L_1 \\ L_2 \\ \cdot \\ \cdot \\ \cdot \\ L_n \end{bmatrix}$$

$$\begin{bmatrix} c_1, c_2 \ldots c_n \end{bmatrix}$$

Digitar os valores de *n* e os elementos da matriz A (I, J).




2
J = 1, N
C (J) = 0
I = 1, N
C (J) = C (J) + A (I, J)
I
J
I = 1, N
C (I)
L (I)
I
PARE

```
10 INPUT "N = "; N
20 PRINT
30 DIM A (N, N), C (N), L (N)
40 FOR I = 1 TO N
50    FOR J = 1 TO N
60       PRINT "A ("; I ;", "J;") = ";
70       INPUT A (I, J)
80       PRINT
90    NEXT J
100    PRINT
110 NEXT I
120 FOR I = 1 TO N
130    LET L (I) = 0
140    FOR J = 1 TO N
150       LET L (I) = L (I) + A (I, J)
160    NEXT J
170 NEXT I
180 FOR J = 1 TO N
190    LET C (J) = 0
200    FOR I = 1 TO N
210       LET C (J) = C (J) + A (I, J)
220    NEXT I
230 NEXT J
240 FOR I = 1 TO N
250    PRINT "C ("; I ;") = "; C (I);"  L ("; I ;") = "; L (I)
260 NEXT I
270 STOP
```

6. Escrever fluxograma e programa BASIC para calcular o produto da matriz A pelo vetor X:

$$\tilde{y} = [A] * \tilde{x} \text{ ou}$$

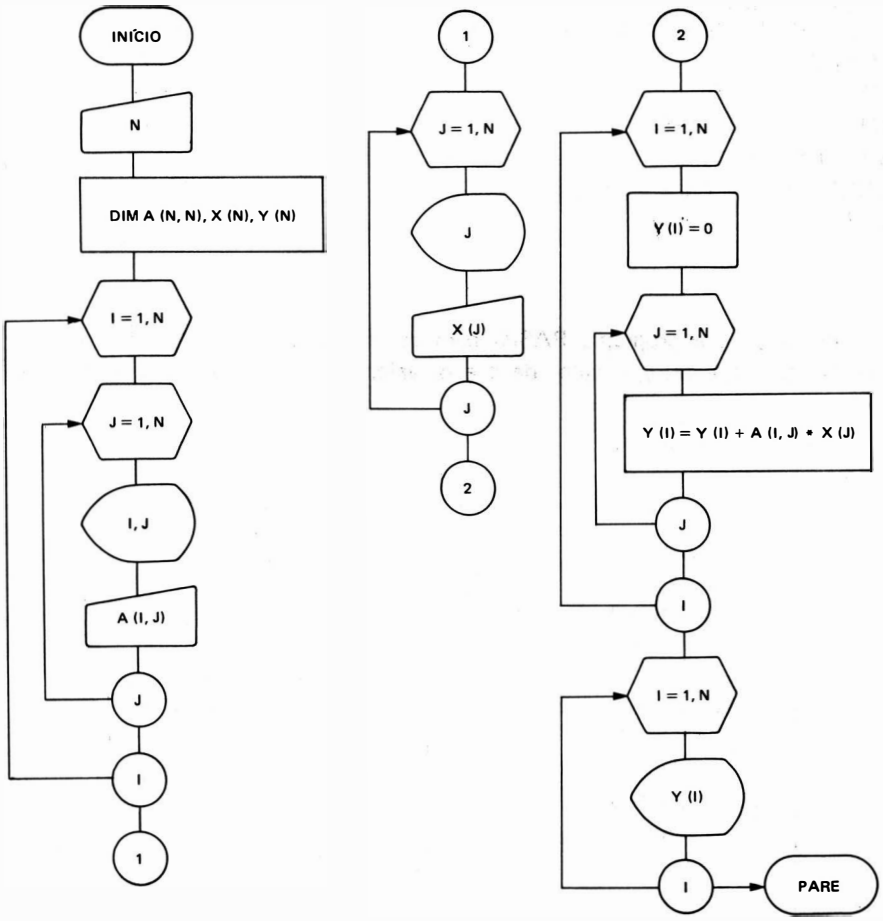
$$\begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11}, a_{12} \ldots a_{1n} \\ a_{21}, a_{22} \ldots a_{2n} \\ . \\ . \\ . \end{bmatrix} * \begin{bmatrix} X_1 \\ X_2 \\ \\ \\ \\ X_n \end{bmatrix}$$

$$a_{ij} \cdot X_j =$$
$$y_1 = a_{11} \cdot X_1 + a_{12} \cdot X_2$$
$$y_2 = a_{22}, \cdot Xi + a_{22} \cdot X_2$$

Por definição:

$$y_i = \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, 2, \ldots, n$$

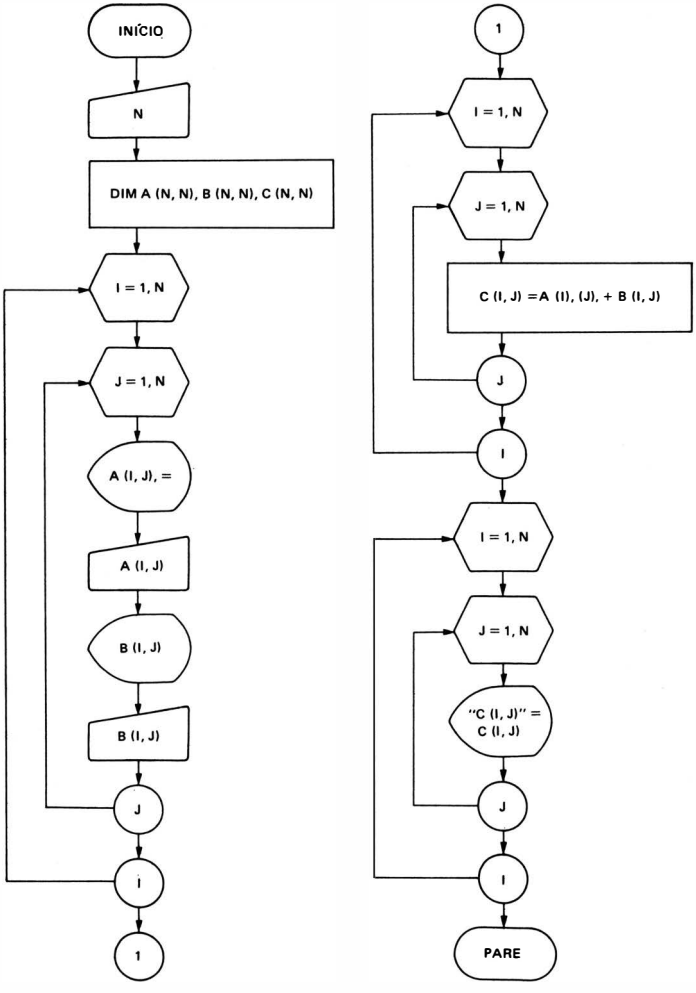
Digitar na entrada os valores dos elementos da matriz A (I, J) e os elementos do vetor X.




```
10 INPUT "N = ";N
20 PRINT
30 DIM A (N, N), X (N), Y (N)
40 FOR I = 1 TO N
50    FOR J = 1 TO N
60       PRINT "A (";I ;"," J ;") = ";
70       INPUT A (I, J)
80       PRINT
90    NEXT J
100    PRINT
110 NEXT I
120 FOR J = 1 TO N
130    PRINT "X (";J ;") = ";
140    INPUT X (J)
150    PRINT
160 NEXT J
```

```
170 FOR I = 1 TO N
180     LET Y (I) = 0
190     FOR J = 1 TO N
200         LET Y (I) = Y (I) + A (I, J) * X (J)
210     NEXT J
220 NEXT I
230 PRINT
240 FOR I = 1 TO N
250     PRINT "Y (";I ;") = "; Y (I)
260 NEXT I
270 STOP
```

7. Escrever fluxograma e programa BASIC para calcular a soma de 2 matrizes A (*n*, *n*) e B (*n*, *n*). Digitar na entrada o valor de *n* e os valores dos elementos das matrizes A (I, J) e B (I, J).



```
10 INPUT "N = "; N
20 PRINT
30 DIM A (N, N), B (N, N), C (N, N)
40 FOR I = 1 TO N
50    FOR J = 1 TO N
60       PRINT "A ("; I ;","; J ;") = ";
70       INPUT A (I, J)
80       PRINT
90       PRINT "B ("; I ;","; J ;") = ";
100        INPUT B (I, J)
110        PRINT
120   NEXT J
130   PRINT
140 NEXT I
150 FOR I = 1 TO N
160    FOR J = 1 TO N
170          LET C (I, J) = A (I, J) + D (I, J)
180    NEXT J
190 NEXT I
200 FOR I = 1 TO N
210    FOR J = 1 TO N
220          PRINT "C ("; I ;","; J ;") = "; C (I, J)
230    NEXT J
240 NEXT I
250 STOP
```

8. Uma determinada firma comercial possui um quadro de 10 vendedores que ganham a importância fixa de 30 ORTN's mais uma comissão de 5% sobre as vendas. Elaborar um fluxograma e programa BASIC para calcular e exibir a venda mensal de cada vendedor, bem como o seu salário. Calcular também a venda semanal geral, a venda mensal geral e a receita bruta desta firma. Sabe-se que ao encerrar suas atividades no dia 30 do mês, o quadro de vendas era o seguinte:

sem. / V	1ª semana	2ª semana	3ª semana	4ª semana
1	12	80	46	40
2	31	72	49	35
3	83	58	72	29
4	69	28	73	32
5	37	51	51	43
6	81	35	63	32
7	29	15	85	39
8	72	36	29	40
9	71	48	19	32
10	52	74	15	20

Os valores de venda são dados em ORTN's e a receita bruta, para facilidade de cálculo, é considerada como sendo apenas o total de vendas do qual são subtraídos o total de salários.

Os dados são digitados na entrada, linha por linha, e os resultados deverão ser exibidos da seguinte forma:

Vendedor . . . Total de Vendas = . . . Salário . . .

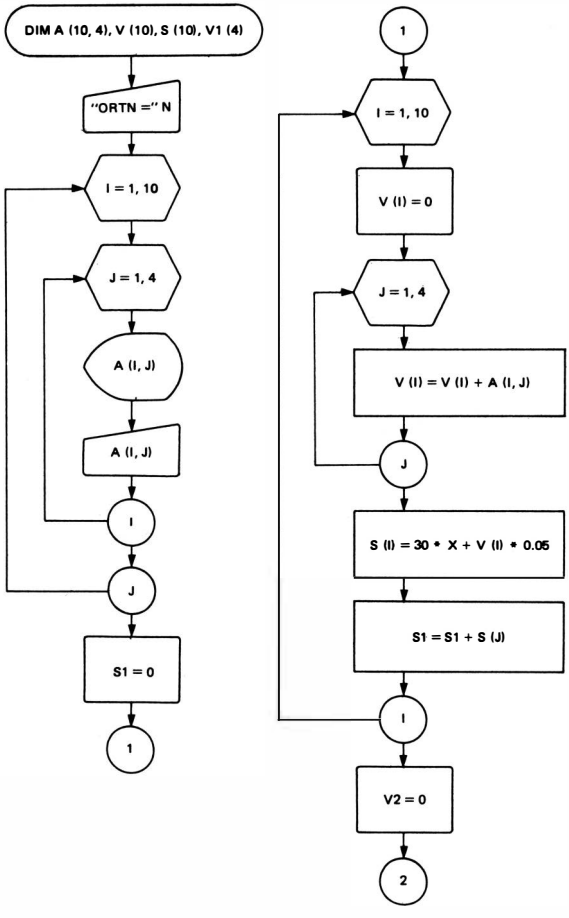
.
.
.

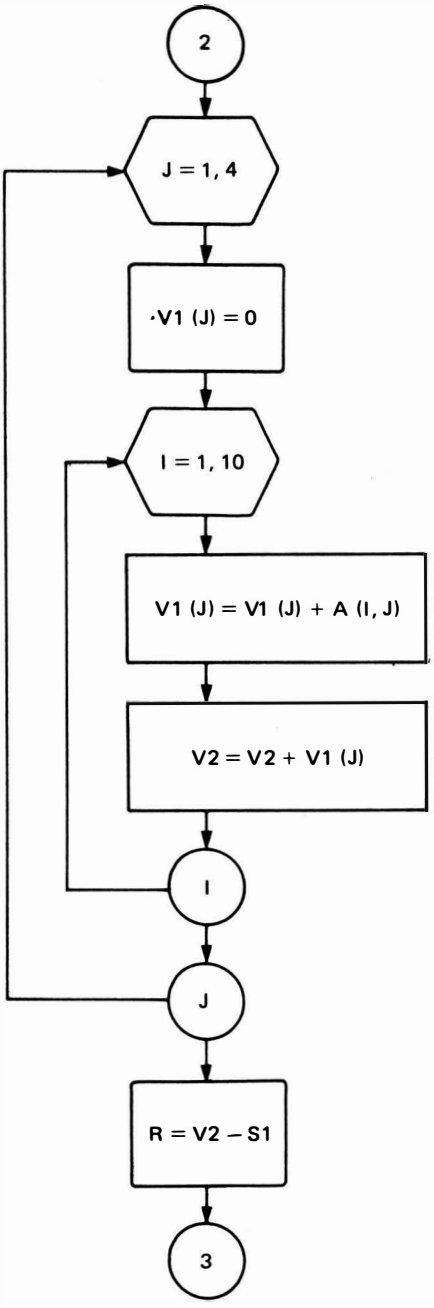
Vendas da Semana . . . = . . .

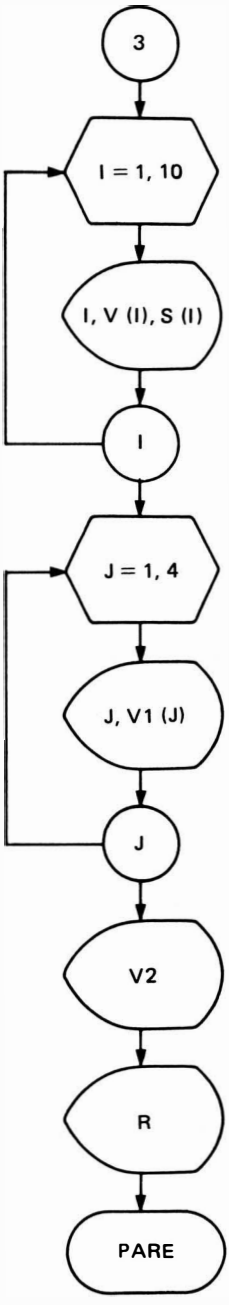
.
.
.

Venda Mensal = . . .

Receita Bruta = . . .




2
J = 1, 4
V1 (J) = 0
I = 1, 10
V1 (J) = V1 (J) + A (I, J)
V2 = V2 + V1 (J)
I
J
R = V2 – S1
3


3
I = 1, 10
I, V (I), S (I)
I
J = 1, 4
J, V1 (J)
J
V2
R
PARE

```
10 DIM A (10, 4), V (10), S (10), V1 (4)
20 REM X       = VALOR DA ORTN
30 REM A (I, J) = VENDAS DO VENDEDOR "I", NA SEMANA "J"
40 REM ENTRADA DE VALORES
50 PRINT
60 INPUT "ENTRE COM O VALOR DA ORTN "; X
70 PRINT
80 PRINT
90 PRINT "ENTRE COM OS VALORES DA MATRIZ"
100 PRINT
110 FOR I = 1 TO 10
120   FOR J = 1 TO 4
130     PRINT "A ("; I ;","; J ;") = ";
140     INPUT A (I, J)
150     PRINT
160   NEXT J
170 NEXT I
180 REM CALCULOS DE TOTAL DE VENDAS E SALARIOS
190 REM S1    = VARIAVEL PARA SOMATORIA DE SALARIOS
200 REM V (I) = VENDAS DO VENDEDOR "I"
210 REM S (I) = SALARIO DO VENDEDOR "I"
220 LET S1 = 0
230 FOR I = 1 TO 10
240   LET V (I) = 0
250   FOR J = 1 TO 4
260     LET V (I) = V (I) + A (I, J)
270   NEXT J
280   LET V (I) = V (I) * X
290   LET S (I) = 30 * X + V (I) * 5E - 02
300   LET S1 = S1 + S (I)
310 NEXT I
320 REM CALCULOS DO TOTAL DE VENDAS DA SEMANA E RENDA BRUTA
330 REM V2 = SOMATORIA DAS VENDAS SEMANAIS (VENDA MENSAL)
340 REM R = RECEITA BRUTA
350 LET V2 = 0
360 FOR J = 1 TO 4
370   LET V1 (J) = 0
380   FOR I = 1 TO 10
390     LET V1 (J) = V1 (J) + A (I, J)
400   NEXT I
410   LET V1 (J) = V1 (J) * X
420   LET V2 = V2 + V1 (J)
430 NEXT J
440 LET R = V2 - S1
450 REM SAIDA DOS RESULTADOS
460 PRINT
470 FOR I = 1 TO 10
480   PRINT "VENDEDOR "; I ;" TOTAL DE VENDAS "; V (I) ;" SALARIO "; S (I)
```

```
490 NEXT I
500 PRINT
510 FOR J = 1 TO 4
520   PRINT "VENDAS DA SEMANA "; J ;" = "; V1 (J)
530 NEXT J
540 PRINT
550 PRINT "VENDA MENSAL = "; V2
560 PRINT
570 PRINT "RECEITA BRUTA = "; R
580 STOP
```

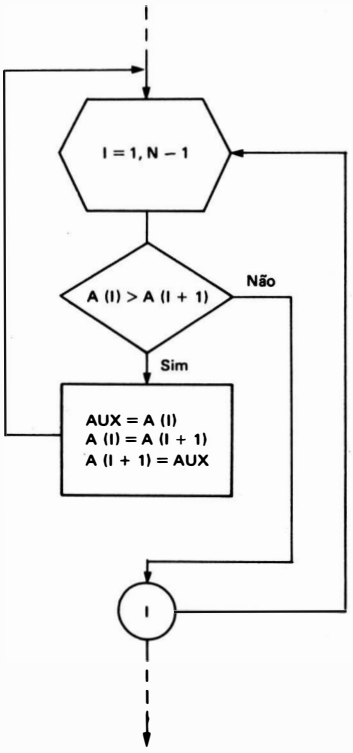
CAPÍTULO 10

APLICAÇÕES ESPECIAIS

1. ORDENAÇÃO NUMÉRICA

Consiste basicamente em colocarmos os números de uma tabela desordenada em ordem crescente ou decrescente. Por exemplo: a lista de valores 10, 4, − 1, 5, 0, se ordenada crescentemente teremos: − 1, 0, 4, 5, 10. Para isso, podemos utilizar o seguinte método:

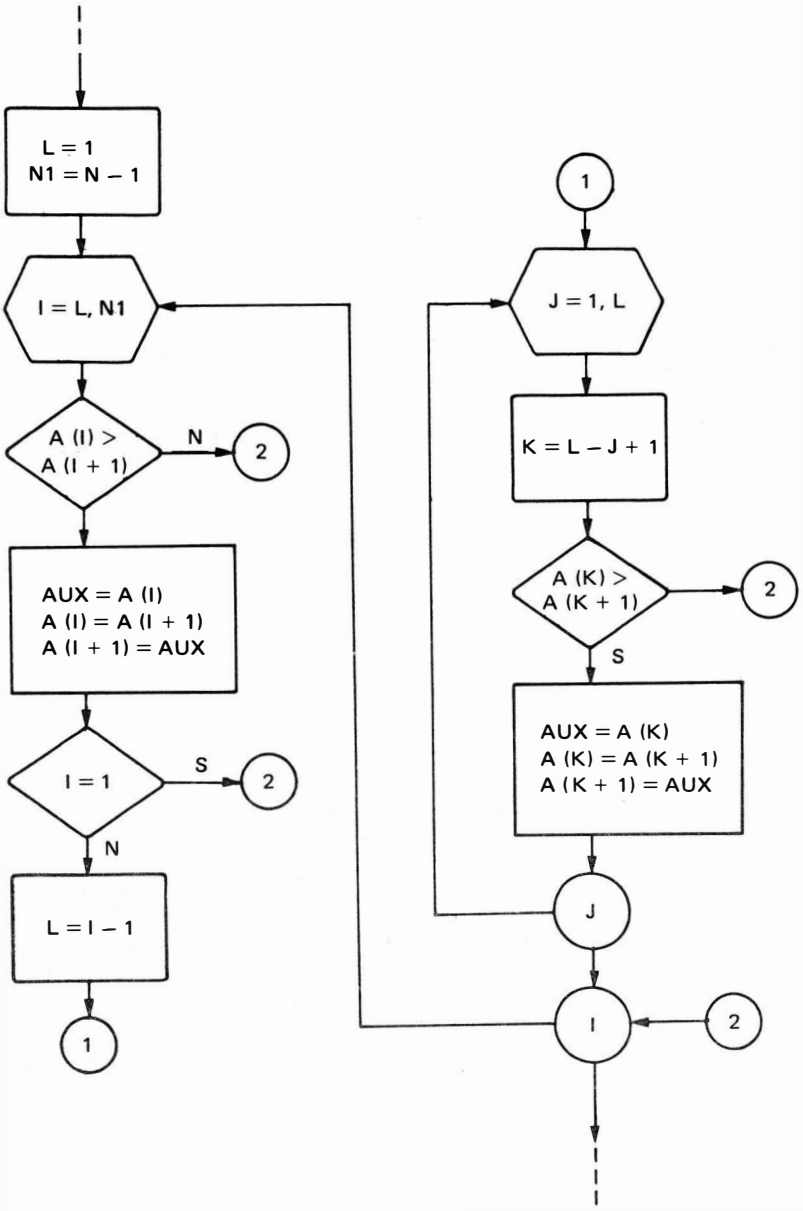
- Coloca-se todos os valores em um vetor (ou variável indexada unidimensional) A (I). Compara-se então 2 elementos consecutivos desta tabela: A (I) com A (I + 1), trocando-os de posição quando A (I) > A (I + 1), repetindo então todo o processo desde o início, conforme o fluxograma:



- A instrução FOR termina em (N − 1), pois a comparação é feita entre A (I) e A (I + 1), que deve ser no máximo igual a N (que é o número de elementos da tabela).

- A variável AUX, é auxiliar para fazer a troca de A (I) por A (I + 1).

Existem, entretanto, outros algoritmos mais eficientes para a classificação de itens em uma tabela (como o que é mostrado a seguir) que não repete todo o processo quando a tabela já está ordenada até o ponto de troca, nele, quando há uma troca de A (I) com A (I + 1), o valor de A (I) é colocado no lugar certo da parte já ordenada:

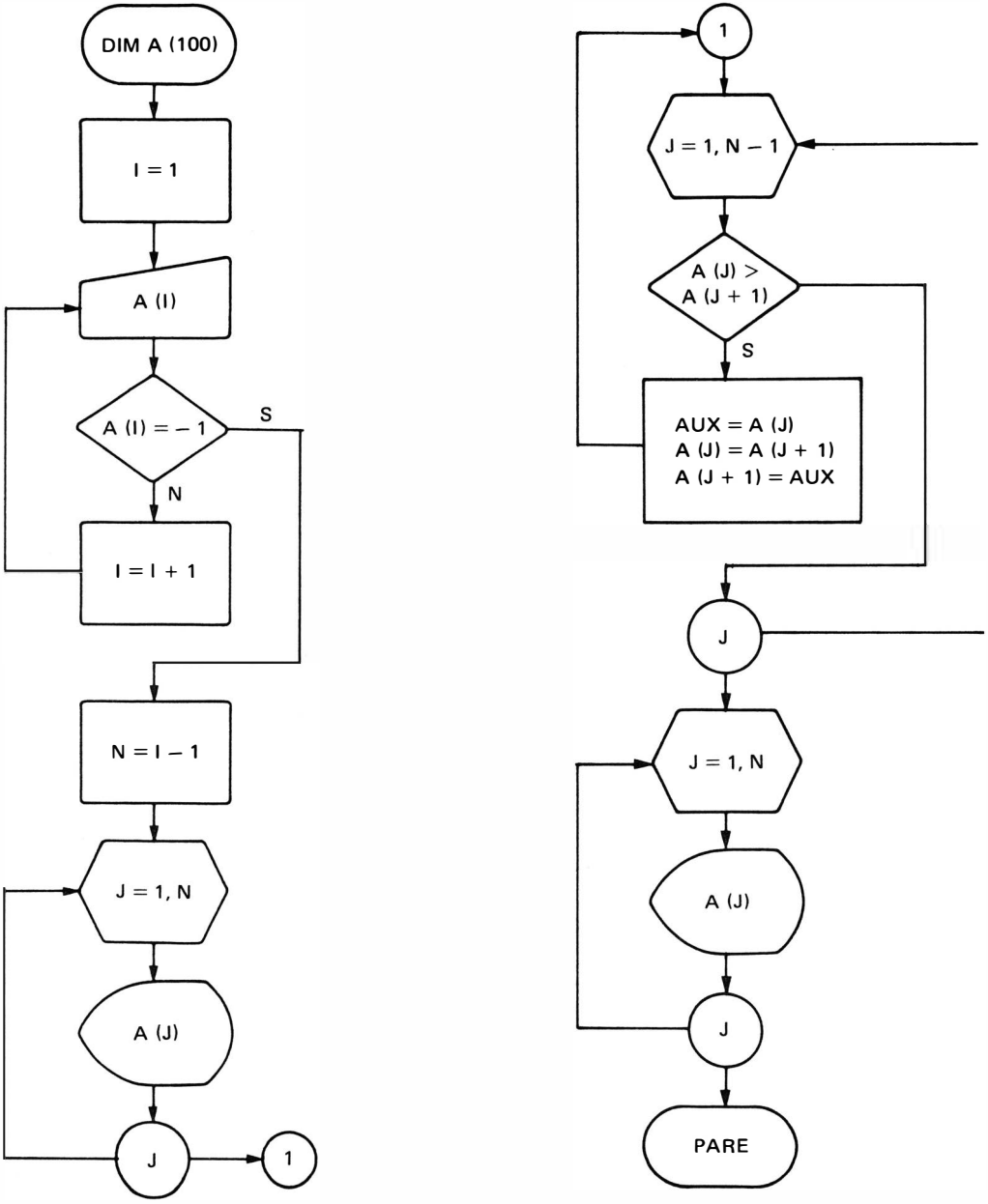


PROBLEMA RESOLVIDO

- Ordenar crescentemente uma lista de diâmetros de peças cilíndricas digitadas pelo operador. A lista deverá ter no máximo 100 elementos e a entrada de valores terminará quando

for digitado o valor — 1, quando será impressa a listagem original dos números, após a qual será iniciada a ordenação crescente e a impressão dos resultados.

FLUXOGRAMA



PROGRAMA

```
10 REM PROGRAMA PARA ORDENACAO CRESCENTE
20 REM - - - - - - - - - - - - - - - - - - - - - - -
30 REM              DIMENSIONAMENTO
40 REM - - - - - - - - - - - - - - - - - - - - - - -
50 DIM A [100]
60 REM ENTRADA DE VALORES
70 LET I = 1
80 PRINT "A (";I ;") = ";
90 INPUT A (I)
100 IF A (I) = - 1 GOTO 130
110 LET I = I + 1
120 GOTO 80
130 REM LISTAGEM ORIGINAL DOS NUMEROS
140 LET N = I - 1
150 REM N = NUMERO REAL DE ELEMENTOS DIGITADOS
160 PRINT
170 PRINT "LISTAGEM ORIGINAL DOS NUMEROS"
180 PRINT
190 FOR J = 1 TO N
200   PRINT A (J);
210 NEXT J
220 PRINT
230 REM ORDENAÇÃO CRESCENTE
240 FOR J = 1 TO N - 1
250   IF A (J) < = A (J + 1) GOTO 300
260   LET W = A (J)
270   LET A (J) = A (J + 1)
280   LET A (J + 1) = W
290   GOTO 230
300 NEXT J
310 REM LISTAGEM ORDENADA
320 PRINT "LISTAGEM ORDENADA"
330 PRINT
340 FOR J = 1 TO N
350   PRINT A (J);
360 NEXT J
370 PRINT
380 STOP
```

Comentários do Problema:

1. A instrução DIM A (100) reserva uma área na memória capaz de armazenar até 101 elementos da tabela A (I) · (0 ⩽ I ⩽ 100).

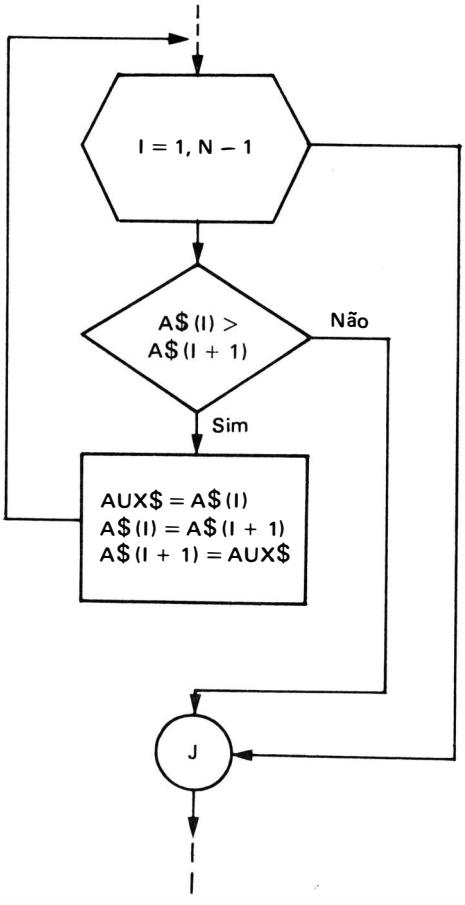
2. A digitação de valores é feita até que se digite o valor − 1, que é um "FLAG", ou seja, um valor fictício que deve ser ignorado, e por isso a tabela real tem N elementos, que vale (I − 1), onde I é o número de valores total digitados (incluindo o valor − 1).

3. Para imprimir os resultados de forma decrescente, basta fazer a variação do laço FOR-NEXT, variar de N até 1 com passo − 1.

2. ORDENAÇÃO ALFABÉTICA

Consiste, basicamente, em compararmos os caracteres de duas variáveis alfanuméricas, trocando-as de posição se estiverem desordenadas. Para isso, colocam-se todos os nomes que se deseja ordenar em uma tabela A$ (I), compara-se então 2 elementos consecutivos desta tabela: A$ (I) com A$ (I + 1), trocando-os de posição quando A$ (I) > A$ (I + 1), repetindo o processo desde o início. Convém observarmos que na comparação A$ (I) > A$ (I + 1) não são os comprimentos dos nomes armazenados em A$ (I) e A$ (I + 1) que são comparados entre si, mas sim os valores que cada caracter assume no padrão ASCII (American Standard Code for Information Interchange). Assim, se A$ (1) = "A" e A$ (2) = "B" temos que A$ (1) < A$ (2), pois o código da letra A é 65 (decimal) e da letra B é 66 (decimal) e portanto A < B.

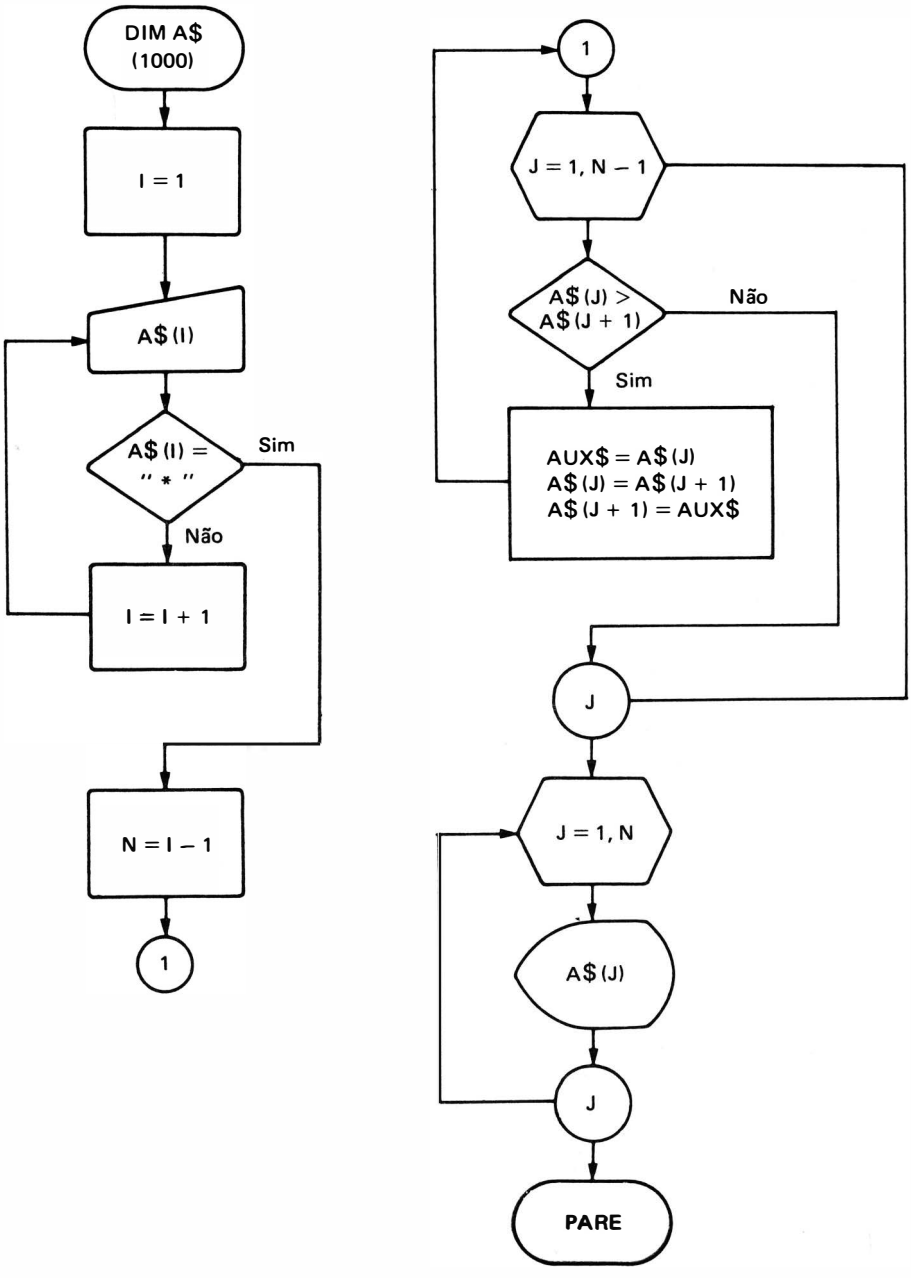
Esquematicamente:



Existem métodos mais eficientes, semelhantes ao apresentado, para a ordenação numérica.

EXEMPLO RESOLVIDO

• Colocar em ordem alfabética a lista de nomes de clientes de uma loja que serão digitados na entrada, supondo-se que haja no máximo 1 000 clientes, encerrando a digitação quando entrar o caracter " * ".



PROGRAMA:

```
10 REM PROGRAMA PARA ORDENACAO ALFABETICA
20 DIM A$ (1000)
30 REM ENTRADA DOS NOMES
40 I = 1
50 INPUT "NOME = "; A$ (I)
60 IF A$ (I) = "*" GOTO 90
70 I = I + 1
80 GOTO 50
90 N = I - 1
100 FOR J = 1 TO N - 1
110   IF A$ (J) < = A$ (J + 1) GOTO 160
120   AUX$ = A$ (J)
130   A$ (J) = A$ (J + 1)
140   A$ (J + 1) = AUX$
150   GOTO 100
160 NEXT J
170 REM SAIDA DOS NOMES ORDENADOS
180 FOR J = 1 TO N
190   PRINT A$ (J)
200 NEXT J
210 STOP
```

Comentários:

1. A instrução DIM A$ (1000) reserva área na memória capaz de armazenar até 1001 nomes da tabela A$ (J) ($0 < J \leqslant 1000$).

2. Cada nome de pessoa pode ter até um máximo de 255 caracteres, sendo que este valor depende do computador utilizado.

3. Este método não é válido no caso do computador interpretar a instrução DIM A$ (N) como sendo A$ uma cadeia com N caracteres e não N cadeia de 255 caracteres.

4. A digitação dos nomes é feita até que seja digitado o caracter "*", que deve ser ignorado e por isso a tabela real tem N elementos que vale (I − 1), onde I é o número total de nomes digitados (incluindo o *).

3. PESQUISA DE TABELAS

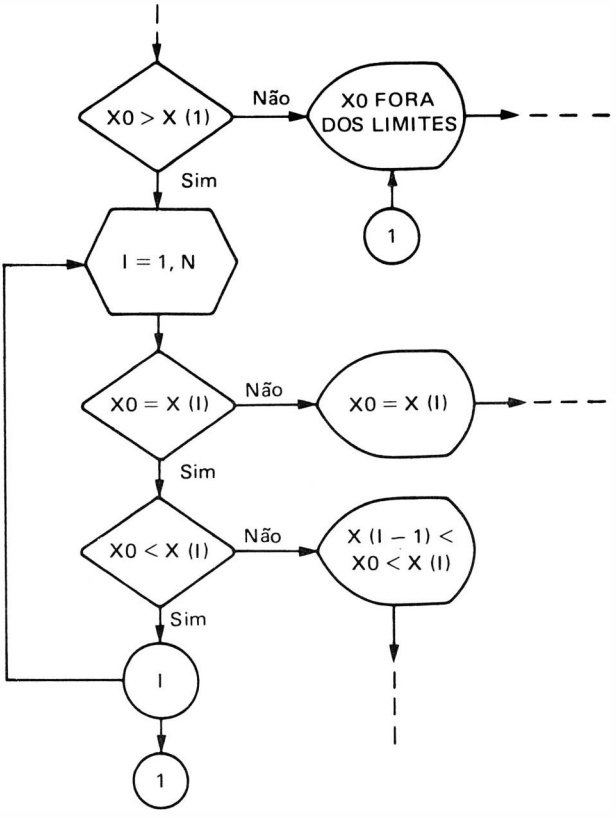
Um dos problemas mais freqüentes na área de processamento de dados consiste em procurar um determinado item de uma tabela ordenada crescentemente. O valor do item procurado deve estar entre dois elementos consecutivos da tabela X (I) < X0 < X (I + 1) ou mesmo ser um elemento da tabela, que é o caso mais geral. Qualquer que seja o método utilizado, deve-se sempre verificar se o item procurado está na faixa entre os limites da tabela, caso contrário, ou seja, X0 < X (1) ou X0 > X (N) é o índice máximo da tabela X), deve-se enviar uma mensagem do tipo → "valor procurado não está nos limites da tabela".

Existem vários métodos para se fazer uma pesquisa de tabelas e podemos destacar os seguintes:

3.1. Busca Seqüencial ou Linear

Utilizado para tabelas ordenadas onde o valor X0 (que é o valor a ser procurado na tabela) é comparado seqüencialmente com os itens X (1), X (2), X (3) etc. Para se encontrar um elemento em uma tabela de N itens, o tempo médio necessário é proporcional a N/2 já que em média a metade da tabela é comparada. Este método é bom quando o número de itens da tabela é pequeno.

Esquematicamente temos:

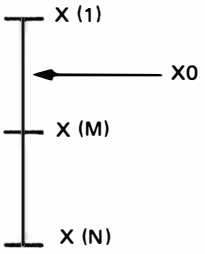


OBS.: No caso da busca seqüencial de um elemento que com certeza pertence à tabela, a mesma pode ser desordenada.

3.2. Busca Binária ou Bissecção

Utilizada para tabelas ordenadas, consiste em dividir a tabela ao meio, verificando-se em qual das metades X_0 (que é o valor a ser procurado na tabela) pode ser achado, o que é feito comparando-se X0 com o elemento intermediário da tabela: X (M). Se X0 > X (M) está na

metade inferior da tabela. Se X0 < X (M) está na metade superior da tabela. Ignora-se então, a metade que não possui o valor de X0 e repete-se o processo até que X0 fique colocado entre 2 elementos consecutivos.

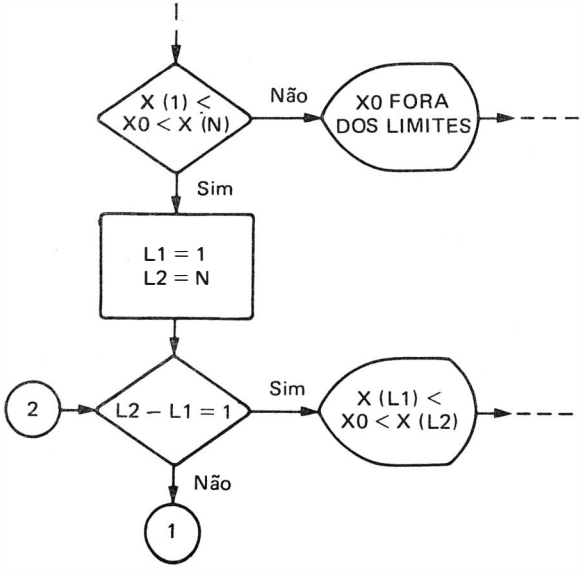


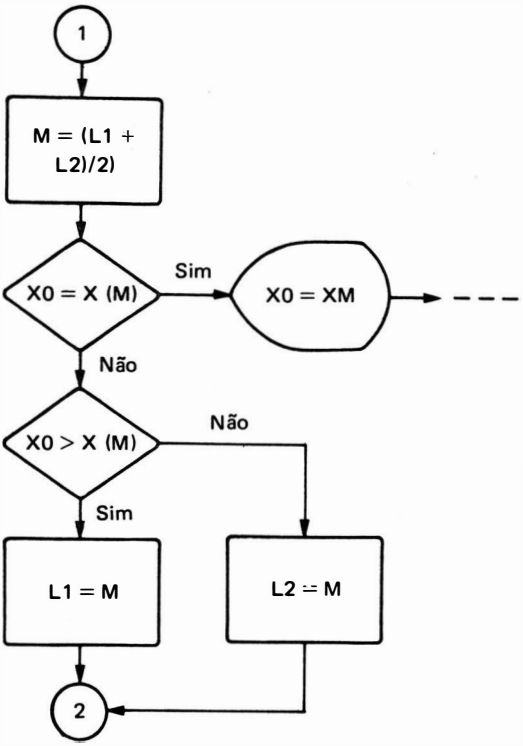
Este método é bastante eficiente, para tabelas com grande números de itens, pois verifica-se que logo inicialmente metade da tabela já é descartada. O tempo necessário para encontrar-se um elemento é, portanto, muito menor que na busca seqüencial e é proporcional a $\log_2 N$.

Comparando-se os dois métodos, podemos construir a seguinte tabela:

Nº de Elementos	Tempo Médio na Busca Seqüencial	Tempo na Busca Binária
10	5	3,32
100	50	6,64
1000	500	9,96

esquematicamente temos:





Para calcularmos o valor do índice do elemento médio da tabela, consideramos na primeira vez a tabela toda fazendo L1 = 1 e L2 = N (nº de elementos da tabela) M será a parte inteira de (L1 + L2)/2.

Para X0 > X (M) → faz-se L1 = M e
Para X0 < X (M) → faz-se L2 = M

Repetindo-se o processo até que L2 − L1 = 1, ou seja, até que X0 fique entre 2 elementos consecutivos da tabela.

PROBLEMA RESOLVIDO – INTERPOLAÇÃO LINEAR

A função $y = f(x)$ nos fornece uma seqüência de pares de pontos Xi, Yi ($1 \leqslant i \leqslant N$) ordenados crescentemente e tabelados em uma lista DATA da seguinte forma:

DATA N, X1, Y1, X2, Y2, X3, Y3, . . . , Xn, Yn

onde n é o número de pares X, Y e não superior a 100.

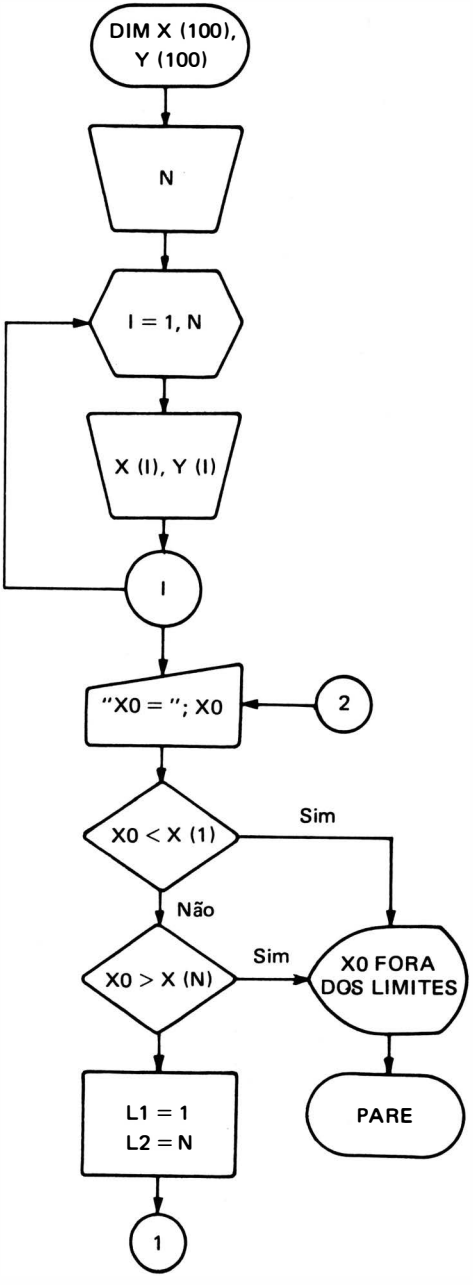
Achar um valor i, de modo que um certo valor X0 digitado na entrada fique entre 2 valores de X da tabela, $X(I) < X0 \leqslant X(I + 1)$.

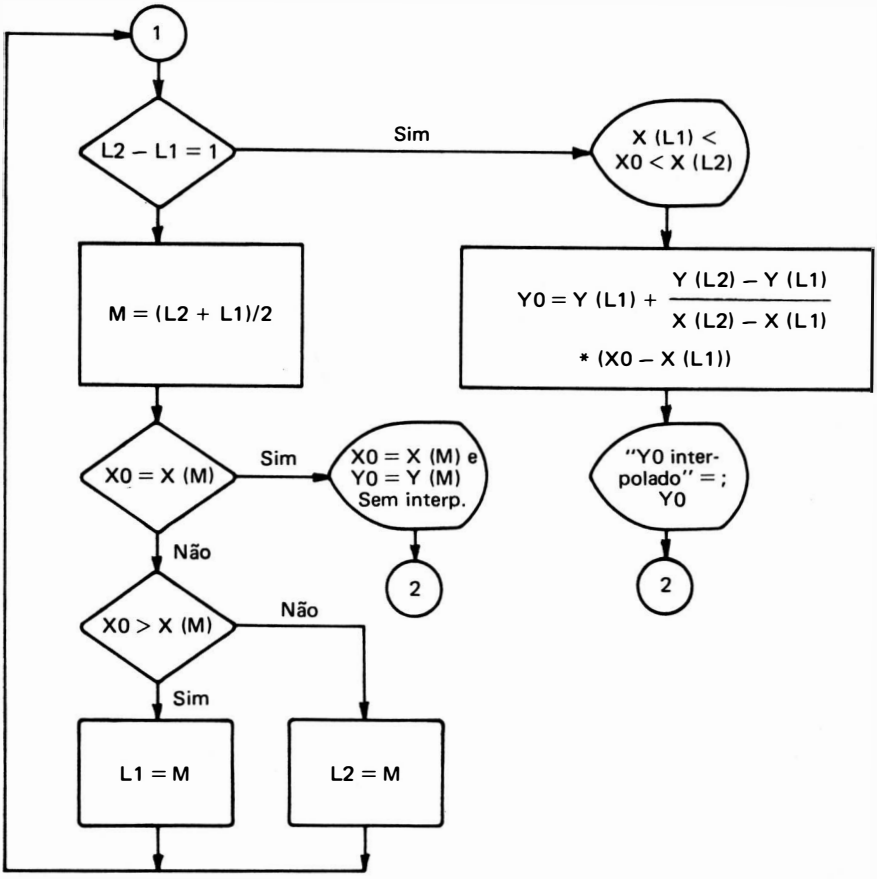
Calcular também o valor de Y0, fazendo uma interpolação linear:

$$Y0 = Yi + \frac{Yi + 1 - Yi}{Xi + 1 - Xi} (X0 - Xi)$$

O processo encerra-se quando entrar um valor de X0 fora da tabela, após a devida impressão de mensagem do tipo:

"X0 fora dos limites"





PROGRAM BASIC

```
10 REM PROGRAMA PARA INTERPOLACAO LINEAR
20 DIM X (100), Y (100)
30 READ N
40 FOR I = 1 TO N
50     READ X (I), Y (I)
60 NEXT I
70 INPUT "X0 = "; X0
80 REM X0 E' O VALOR A SER PESQUISADO NA TABELA
90 PRINT
100 IF X0 < X (1) GOTO 290
110 IF X0 > X (N) GOTO 290
120 LET L1 = 1
130 LET L2 = N
140 IF L2 – L1 = 1 GOTO 250
150 LET M = INT ((L2 + L1)/2)
```

```
160 IF X0 = X (M) GOTO 220
170 IF X0 > X (M) GOTO 200
180 LET L2 = M
190 GOTO 140
200 LET L1 = M
210 GOTO 140
220 PRINT X0;" = X (";M;") E Y0 = ";Y(M)
230 PRINT "SEM INTERPOLAR"
240 GOTO 70
250 PRINT "X (";L1;") < = ";X0;" < = X (";L2;")"
260 LET Y0 = Y (L1) + ((Y (L2)− Y [L1]/(X (L2) − X (L1)) * (X0 − X (L1))
270 PRINT "Y0 INTERPOLADO = ";Y0
280 GOTO 70
290 PRINT "X0 FORA DOS LIMITES DA TABELA"
300 STOP
310 DATA 10
320 DATA 1, 1, 2, 2, 3, 3, 4, 4, 5, 5
330 DATA 6, 6, 7, 7, 8, 8, 9, 9, 10, 10
```

4. INTERPOLAÇÃO PELO POLINÔMIO DE LAGRANGE

Deseja-se calcular o valor de uma certa função $f(x)$, definida no intervalo $[a, b]$ em um certo ponto X0 pertencente a esse intervalo. Entretanto, a função $f(x)$ não é dada como uma fórmula matemática, mas somente como uma tabela com um certo número finito de pontos. Se o valor de X0 coincide com qualquer valor dado na tabela, então o valor procurado de $Y0 = f(X0)$ é imediato. O problema ocorre quando o valor de X0 encontra-se entre dois itens quaisquer da tabela. Para achar este valor Y0 existem várias fórmulas, a mais comumente empregada é a interpolação polinomial de Lagrange que tem o seguinte aspecto:

$$Y0 \cong \frac{(X0 - X2)\,(X0 - X3) \ldots (X0 - Xn)}{(X1 - X2)\,(X1 - X3) \ldots (X1 - Xn)}\ Y1 + \frac{(X0 - X1)\,(X0 - X3) \ldots (X0 - Xn)}{(X2 - X1)\,(X2 - X3) \ldots (X2 - Xn)}\ Y2 +$$

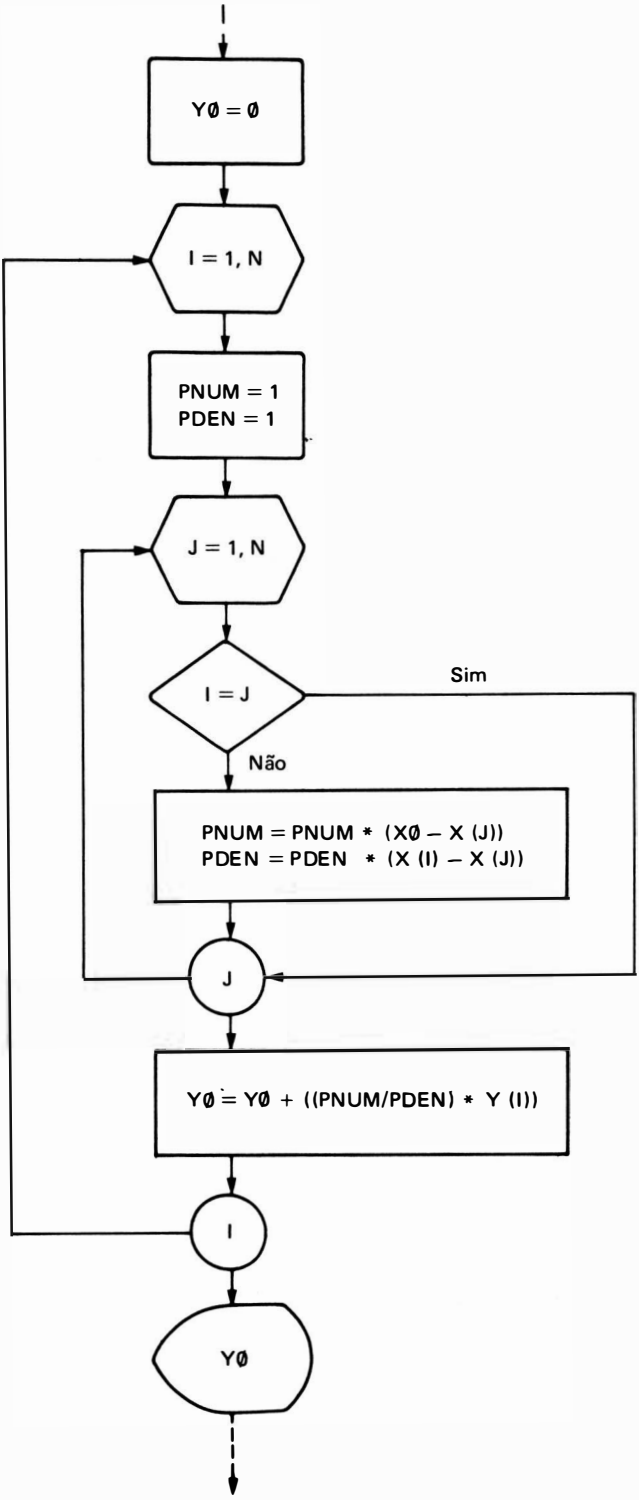
$$+ \frac{(X0 - X1)\,(X0 - X2) \ldots (X0 - Xn - 1)}{(Xn - X1)\,(Xn - X2) \ldots (Xn - Xn - 1)}\ Yn$$

Como podemos observar trata-se de uma somatória de um quociente de duas produtórias conforme a fórmula compactada:

$$Y0 = \sum_{i=1}^{n} \left\{ \frac{(X0 - Xi)}{(Xi - Xj)} \right\} Yi \qquad \text{com } i \neq j$$

$$J = 1 \ldots N$$

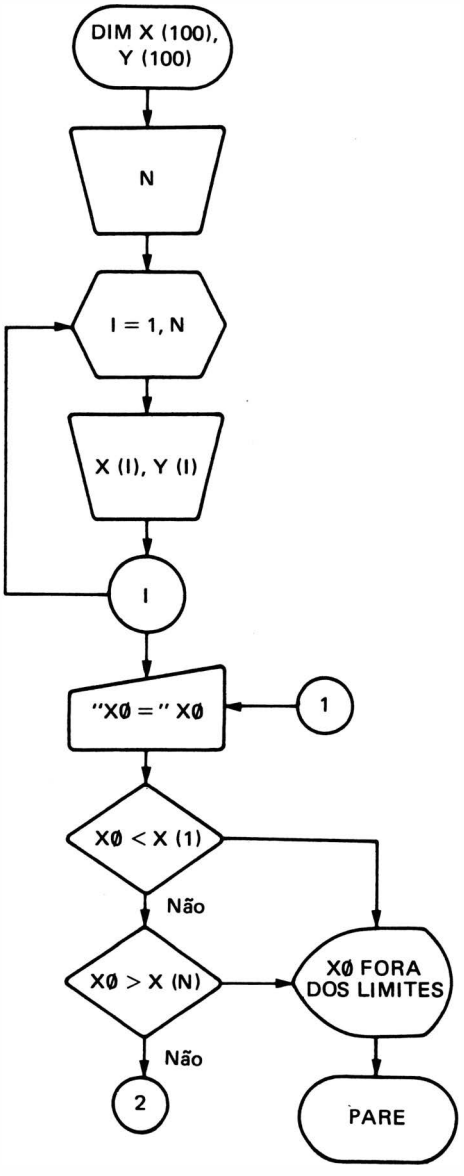
Esta fórmula é facilmente implementada para cálculo por computador, pois temos apenas somatórias e produtórias, conforme o algoritmo:

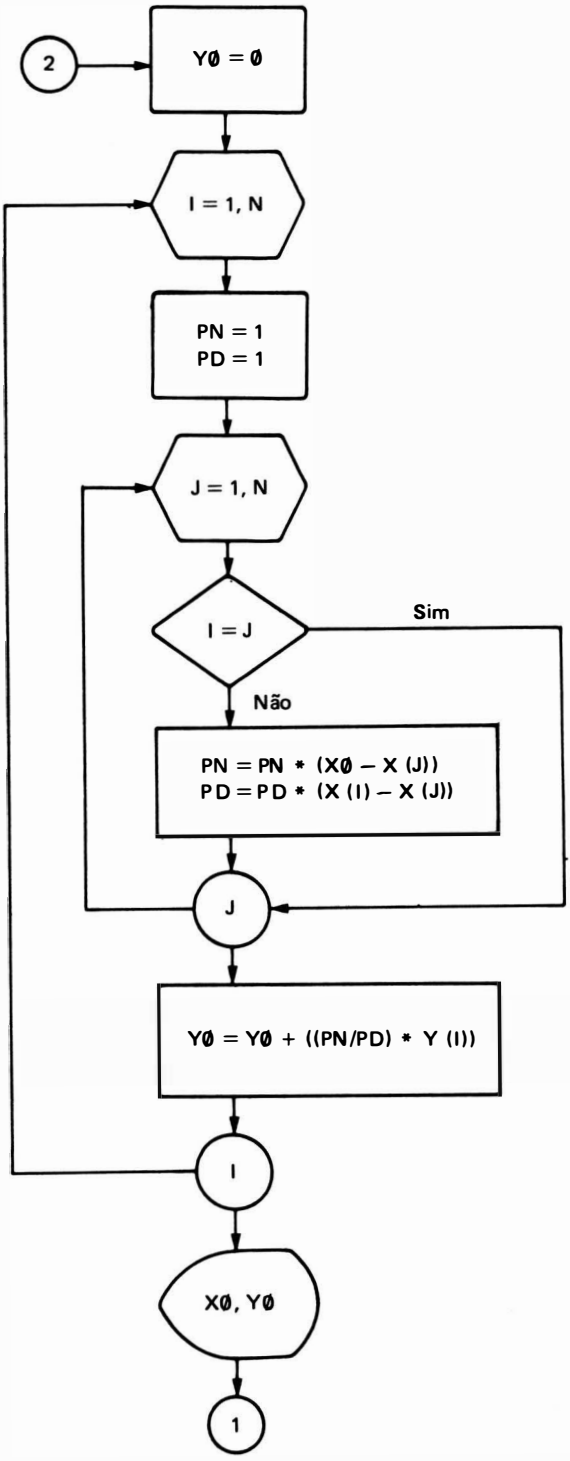

I
YØ = Ø
I = 1, N
PNUM = 1
PDEN = 1
J = 1, N
I = J
Sim
Não
PNUM = PNUM * (XØ – X (J))
PDEN = PDEN * (X (I) – X (J))
J
YØ = YØ + ((PNUM/PDEN) * Y (I))
I
YØ

Podemos notar que para este caso de interpolação por polinômio de Lagrange, são utilizados todos os valores da função tabelada, não exigindo, como no caso da interpolação linear, uma pesquisa de tabela.

PROBLEMA RESOLVIDO – INTERPOLAÇÃO LAGRANGEANA

- Vamos calcular para o mesmo problema proposto anteriormente, o valor de Y0 por interpolação Lagrangeana para um valor X0 digitado na entrada. O diagrama de blocos para este problema será:




2
YØ = Ø
I = 1, N
PN = 1
PD = 1
J = 1, N
I = J
Sim
Não
PN = PN * (XØ – X (J))
PD = PD * (X (I) – X (J))
J
YØ = YØ + ((PN/PD) * Y (I))
I
XØ, YØ
1

```
PROGRAM BASIC

10 REM PROGRAMA PARA INTERPOLACAO LAGRANGEANA
20 DIM X (100), Y (100)
30 READ N
40 FOR I = 1 TO N
50    READ X (I), Y (I)
60 NEXT I
70 INPUT "X0 = "; X0
80 PRINT
90 IF X0 < X (1) GOTO 250
100 IF X0 > X (N) GOTO 250
110 LET Y0 = 0
120 FOR I = 1 TO N
130   LET P1 = 1
140   LET P2 = 1
150   FOR J = 1 TO N
160      IF I = J GOTO 190
170      LET P1 = P1 * (X0 - X (J))
180      LET P2 = P2 * (X (I) - X (J))
190   NEXT J
200   LET Y0 = Y0 + (P1/P2) * Y (I)
210 NEXT I
220 PRINT "PARA X0 = "; X0 ;" O VALOR DA FUNCAO VALE "; Y0
230 PRINT "POR INTERPOLACAO LAGRANGEANA"
240 GOTO 70
250 PRINT "X0 ESTA FORA DOS LIMITES DA TABELA"
260 STOP
270 DATA 10
280 DATA 1, 2, 2, 4, 3, 6, 4, 8, 5, 10
290 DATA 6, 12, 7, 14, 8, 16, 9, 18, 10, 20
300 GOTO 90
310 PRINT "X0 FORA DOS LIMITES DA TABELA"
320 STOP
```

5. INTEGRAÇÃO NUMÉRICA

Um dos problemas mais comuns para cientistas e engenheiros é a avaliação de uma expressão do tipo:

$$y = \int_a^b f(x)\, dx$$

que é na realidade a resolução de uma equação diferencial que nos fornece:

$$y' = F(X)$$

Para avaliarmos o valor da integral definida para um extremo inferior (a) até um extremo superior (b) calculamos:

$$A = \int_a^b f(x)\,dx = F(b) - F(a)$$

Existem, entretanto, funções que não podem ser integradas por métodos de cálculo, como por exemplo, funções que estão tabeladas para um número finito de pontos X e Y, o que requer a aplicação de algum processo numérico para integração.

5.1. Regra dos Trapézios

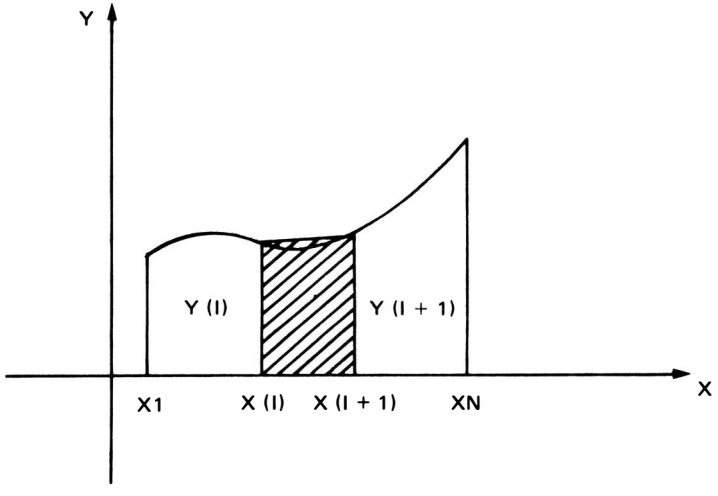
É utilizada quando todos os valores de X_i são igualmente espaçados de um passo h ou:

$$(X_{i+1} - X_i) = h = \frac{(b-a)}{N-1}$$

A integral pode ser avaliada então pela seguinte fórmula:

$$A = \int_{X1}^{Xn} f(x)\,dx \cong \sum_{i=1}^{n} \frac{Y(I+1) + Y(I)}{2} \cdot (X(I+1) - X(I))$$

Graficamente isto significa calcular a área do trapézio abaixo:

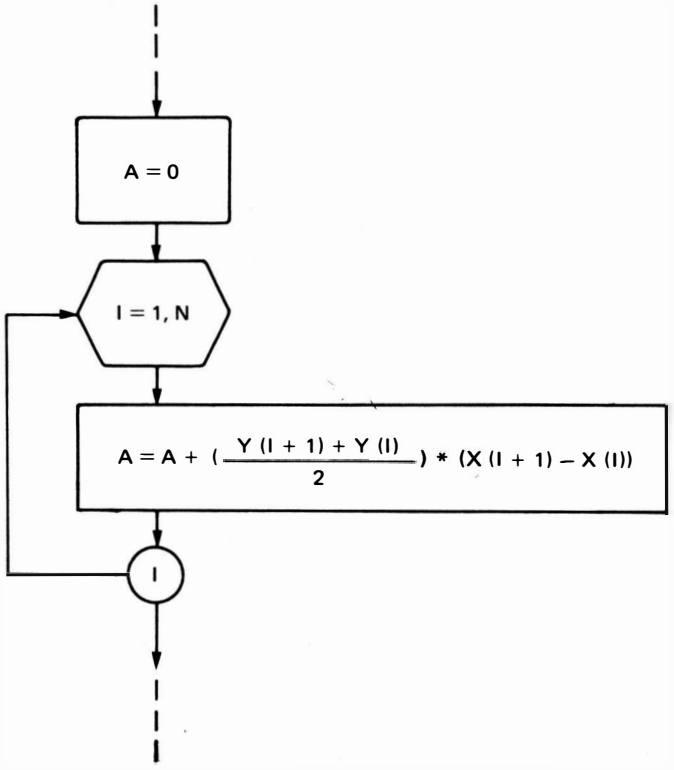


Podemos notar que existe um certo erro, que pode ser diminuído, à medida que diminuímos o passo h entre os valores X_i consecutivos. Este erro pode ser estimado através da fórmula:

$$E = \frac{(b-a) \cdot h^2}{12} \cdot f''(\epsilon)$$

onde $f''(\epsilon)$ é a derivada segunda da função, calculada para um ponto entre o intervalo de integração.

O fluxograma genérico para a regra dos trapézios será, então, o seguinte:



PROBLEMA RESOLVIDO

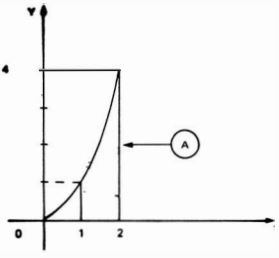
• Calcular a área sob a curva parabólica $Y = X^2$, no intervalo 0 a 2, usando a regra dos trapézios.

Solução:

Deseja-se, então, resolver o seguinte problema:

$$A = \int_0^2 X^2 \, dx$$

ou graficamente:



Este problema, em particular, é possível de ser resolvido por cálculo integral:

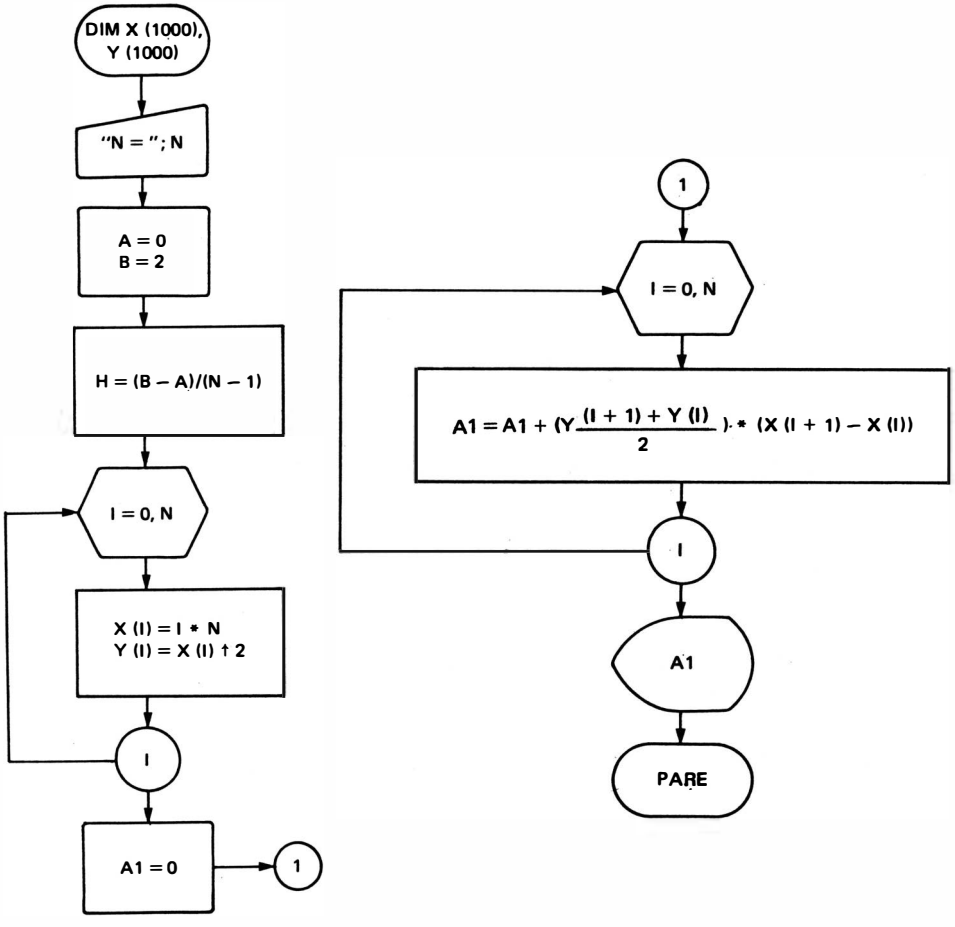
$$A = \int_0^2 x^2\,dx \rightarrow A(X) = \frac{X^3}{3}\Bigg|_0^2 \rightarrow \left[\frac{2^3}{3} - \frac{0^3}{3}\right] =$$

$$A = \frac{8}{3} = 2.6666\ldots$$

Podemos calcular numericamente este valor com a precisão desejada (dentro das limitações do computador utilizado), escolhendo um número de pontos conveniente para o cálculo do passo entre os valores Xi consecutivos.

Pela equação que nos fornece o erro, verificamos que este é da ordem de h^2. Se a função tiver segunda derivada ($f''(x)$) com valores pequenos (ou seja, uma função "suave"), pode-se esperar uma precisão até na quarta casa decimal para um passo $h = 0{,}01$.

O fluxograma para a resolução deste problema seria o seguinte:



PROGRAMA BASIC

```
10 REM INTEGRACAO PELA REGRA DOS TRAPEZIOS
20 DIM X (1000), Y (1000)
30 INPUT "ENTRE COM O NUMERO DE PONTOS (MAXIMO = 1000) "; N
40 PRINT
50 REM OS VALORES DE A E B DEFINEM O INTERVALO DE INTEGRACAO
60 LET A = 0
70 LET B = 2
80 REM CALCULO DO PASSO DE INTEGRACAO (H)
90 LET H = (B - A)/(N - 1)
100 REM CALCULO DOS VALORES DE X (I) E Y (I)
110 FOR I = 0 TO N
120   LET X (I) = I * H
130   LET Y (I) = X (I) ↑ 2
140 NEXT I
150 REM CALCULO DO VALOR DA INTEGRAL
160 LET A1 = 0
170 FOR I = 0 TO N - 1
180   LET A1 = A1 + ((Y (I + 1) + Y (I))/2) * (X (I + 1) - X (I))
190 NEXT I
200 REM SAIDA DO VALOR DA INTEGRAL
210 PRINT "AREA SOB A CURVA Y = X ↑ 2 = "; A1
220 STOP
```

Comentários Sobre o Programa:

1. A instrução DIM reserva área na memória para o cálculo de até 1000 pontos.

2. Os cálculos mostram resultados com precisão simples, que o valor da área vai tendendo ao calculado matematicamente à medida que discretizamos mais a função.

5.2. Regra de Simpson

Os processos de integração numérica desenvolvidos por Simpson, também conhecidos por "regra 1/3" e "regra 3/8" de Simpson (devido aos coeficientes iniciais que aparecem em cada uma destas fórmulas) apresentam resultados melhores que a regra dos trapézios.

A "regra 1/3" exige um número par de subintervalos de tamanho h (sendo, portanto, n ímpar) e se reduz a:

$$\int_a^b f(X)\,dx \cong A \cong \frac{h}{3}(Y_1 + 4Y_2 + 2Y_3 + \ldots + 4Y_{n-1} + Y_n)$$

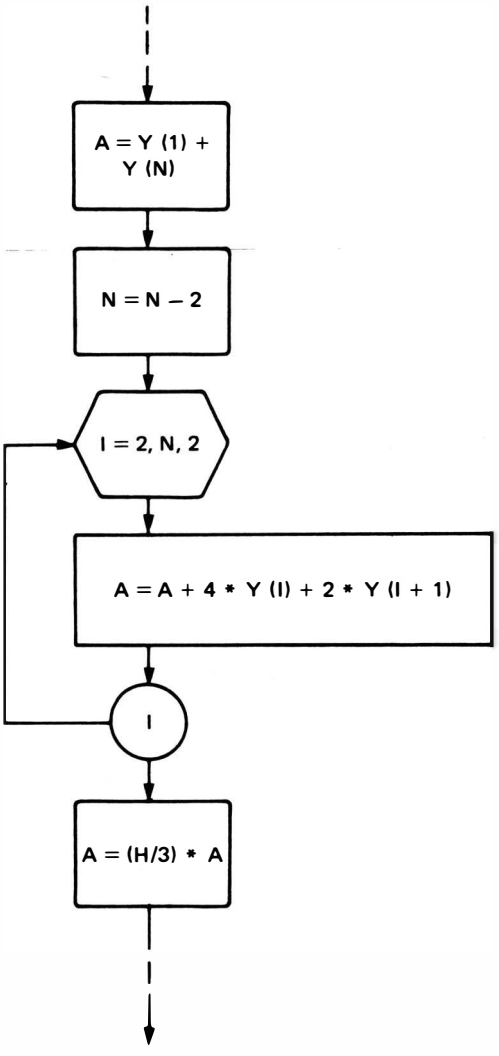
ou

$$A = \frac{h}{3} [Y_1 + Y_n + 4(Y_2 + Y_4 + Y_6 \ldots Y_{n-1}) + 2(Y_3 + Y_5 + \ldots + Y_{n-2})]$$

A "regra 3/8", entretanto, exige que se tenha um número de elementos N de tal forma que N − 1 se torne divisível por 3, e tenha a seguinte forma:

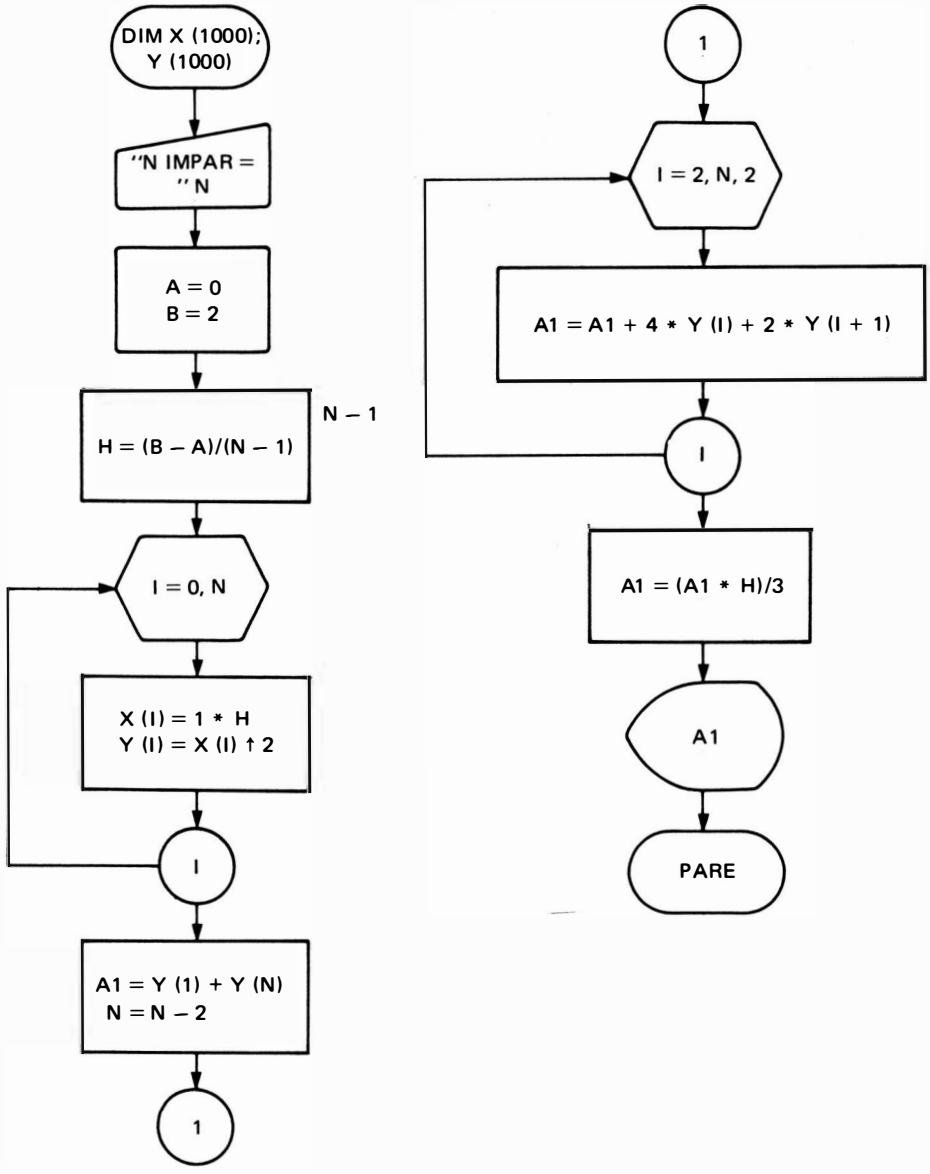
$$A \cong \frac{3}{8} h [Y_1 + 3Y_2 + 3Y_3 + 2Y_4 + \ldots + 3Y_{n-1} + Yn]$$

Como ambas as regras de Simpson apresentam precisão da ordem de h^4, será estudado apenas o algoritmo para a "regra 1/3", conforme segue:



PROBLEMA RESOLVIDO

- Calcular a área sob a curva parabólica $Y = X^2$, no intervalo 0 a 2 usando a regra de Simpson.




```
10 REM INTEGRACAO PELA REGRA DE SIMPSON
20 DIM X (1000), Y (1000)
30 INPUT "ENTRE COM NUMERO DE PONTOS (N IMPAR < 1000) "; N
```

```
40 PRINT
50 LET A = 0
60 LET B = 2
70 LET H = (B – A)/(N – 1)
80 FOR I = 1 TO N
90    LET X (I) = I * H
100    LET Y (I) = X (I) ↑ 2
110 NEXT I
120 LET A1 = Y (1) + Y (N)
130 LET N = N – 2
140 FOR I = 2 TO N STEP 2
150   LET A1 = A1 + 4 * Y (I) + 2 * Y (I + 1)
160 NEXT I
170 LET A1 = (A1 * H)/3
180 PRINT ''AREA = '';A1
190 STOP
```

CAPÍTULO 11

IMPRIMINDO ESTETICAMENTE

Conforme foi visto nos capítulos iniciais a impressão dos resultados é conseguida por meio da instrução PRINT, que pode exibir:

- o valor de uma variável numérica ou alfanumérica;
- o resultado da execução de uma expressão;
- uma lista de caracteres alfanuméricos colocados entre aspas.

As variáveis ou itens a serem impressos podem ser separados por vírgula (colocando os itens em zonas de impressão, que podem ser duas ou quatro, dependendo da máquina), ou separados por ponto e vírgula colocando os itens juntos (deixando apenas, no caso de valores numéricos, dois espaços em branco, um para o sinal e outro para separação natural).

Com apenas esses recursos, o programador ressente-se da necessidade de funções, que lhe permitam manipular o tipo de saída impressa (ou em video) que melhor lhe convenha. Existem para tal as funções a seguir apresentadas.

1. FUNÇÃO TAB

Essa função tem a finalidade de tabular uma determinada linha de impressão, de modo análogo a uma máquina de datilografia, fazendo com que o curso da linha de impressão se desloque para uma determinada posição especificada como argumento da função TAB.

Forma Geral:

nl PRINT TAB (ou expressão); "frase"; o

onde:

nl = número da linha

n = número inteiro ou variável numérica

expressão = expressão BASIC que fornece um valor numérico

"frase" = lista de caracteres fornecidos entre aspas (opcional)

o = variável numérica ou alfanumérica (opcional)

OBS.: O sinal de separação (;) entre a função TAB e a "frase" ou variável é opcional em algumas máquinas.

Esta função é útil quando temos saídas tabeladas, ou a necessidade de cabeçalhos.

```
Ex.: 10 PRINT TAB (1) ;" PRIMEIRA POSICAO"; TAB (20) ;"POSICAO N. 20"
     PRIMEIRA POSICAO    POSICAO N. 20
```

O argumento da função TAB (N), ou seja, o valor de n, varia de um computador para outro, podendo ser de 0 a 31 para micros da lógica SINCLAIR e de 0 a 63 para micros com formatações de videos de 64 colunas. Caso a função TAB seja usada com a instrução LPRINT, o argumento da função poderá variar até o número máximo de colunas que a impressora possui. Assim, para uma impressora de 80 colunas, N poderá variar de 0 a 79; para a impressora de 132 colunas, N poderá variar de 0 a 131 etc.

```
Ex.: 10 LPRINT TAB (0) ;" P0"; TAB (77) ;" P77";
     20 LPRINT
     30 STOP
     0                    .....                    77 78 79
     P0                   .....                      P77
```

A maioria dos micros existentes no mercado, exige que as várias funções TAB que estejam na mesma linha tenham argumentos crescentes, isto é, que TAB (1) venha antes de TAB (10), que deve vir antes de TAB (50) etc, existindo, entretanto, alguns micros onde estes argumentos podem estar em qualquer ordem.

O argumento da função TAB pode ser, além de um número inteiro, também uma variável, fazendo com que a coluna onde se iniciará a impressão seja função desta variável.

```
Ex.: 10 FOR X = – 8 TO + 8
     20    PRINT TAB (X ↑ 2) ;" * "
     30 NEXT X
```

Imprimindo na tela do micro uma parábola com o caracter "*", nota-se que cada ponto da parábola aparece em uma linha, ficando, portanto, a parábola "deitada" na forma:

```
                                                  *
                                              *       *
                                          *
                                      *
                                  *
                              *
                          *
                      *
                  *
                      *
                          *
                              *
                                  *
                                      *
                                          *
                                              *       *
                                                  *
```

2. INSTRUÇÃO PRINT @

Esta instrução tem por finalidade a impressão de algum item em um ponto determinado na tela de video.

Forma Geral:

```
PRINT @ n ou expressão, "frase", o
```

onde:

n = número inteiro ou variável numérica

expressão = expressão BASIC que fornece um valor numérico

"frase" = lista de caracteres fornecidos entre aspas (opcional)

o = variável numérica ou alfanumérica (opcional)

Esta função é comum para micros com video formatado em 16 linhas por 64 colunas, o que nos fornece um total de 1024 pontos acessíveis e, portanto, o valor de n pode variar de 0 a 1023 da seguinte forma:

1ª linha 0 a 63
2ª linha 64 a 127
.
.
.
16ª linha 960 a 1023

Uma visão geral da tela de video pode ser observada na figura abaixo, que representa o "mapa de video" de uma tecla formatada para 16 linhas e 24 colunas como é o caso dos micros de linha do TRS 80.

Ex.:
```
10 CLS
20 PRINT@ 479, "*"
```

Depois de deixar a tela limpa (CLS), imprime o caracter "*" no meio da tela.

Ex.: 2

```
10 CLS
20 PRINT @ 138, "CRONOMETRO"
30 PRINT @ 407, "MINUTOS :SEGUNDOS"
40 FOR M = 0 TO 59
50    FOR S = 0 TO 59
60       PRINT @ 473, M;"  :  ";S
70          FOR T = 0 TO 400
```

```
80          NEXT T
90        NEXT S
100   NEXT M
110 GOTO 40
```

Este programa exibe um cronômetro digital de minutos e segundos na tela do micro, as linhas 70 e 80 fornecem o ajuste necessário para que o tempo decorrido após sua execução e a impressão de um novo S seja aproximadamente 1 segundo, devendo ser ajustado de acordo com o micro utilizado.

2.1. Instrução PRINT AT

Esta instrução é semelhante ao PRINT @ tendo a mesma finalidade, isto é, a impressão de algum item em um ponto determinado na tela de vídeo, com a diferença de que neste caso existem dois parâmetros para determinar o ponto: um para a linha e outro para a coluna.

Forma Geral:

PRINT AT *l*, c, "frase", v

onde:

l = representa o número da linha

c = representa o número da coluna

"frase" = lista de caracteres fornecidos entre aspas (opcional)

v = variável

Esta função é comum em micro com video formatado em 22 linhas por 32 colunas, como por exemplo, o da lógica SINCLAIR. Onde esta função é uma das palavras-chave do micro.

Ex. 1:

```
10 PRINT AT 12, 16; "0"
```

Imprime o número zero no meio da tela de 22 linhas por 32 colunas.

Ex. 2:

```
10 CLS
20 LET A$ = "ESTOU AQUI"
30 LET B$ = "FIM"
40 PRINT AT 10, 16; A$
50 PRINT AT 20, 27; B$
60 STOP
```

Imprime, a partir da linha 10 e coluna 16, a frase "ESTOU AQUI"; e a partir da linha 20 e coluna 27 a frase "FIM".

Convém observar que existem sistemas BASIC que atuam com PRINT @ *l*, c (com 2 parâmetros) bem como PRINT AT n (com apenas um parâmetro), devendo, portanto, respeitar-se as características do micro em uso.

OBS.: Em algumas versões, a instrução PRINT @ , os parâmetros linha e coluna são invertidos da forma PRINT @ c, *l* devendo-se para tanto consultar o manual do microcomputador.

3. INSTRUÇÃO PRINT USING

Esta instrução é a forma mais poderosa de se imprimir um resultado. Trata-se de um recurso que permite ao usuário escolher o formato de saída que mais lhe convém através do uso de uma "máscara" de impressão.

Forma Geral:

```
PRINT USING "máscara"; v
```

onde

"máscara" = é um conjunto de caracteres que especificam o formato de saída

v = variável numérica ou alfanumérica

A máscara, por ser um conjunto de caracteres alfanuméricos, pode ser definida em uma variável alfanumérica, facilitando eventuais mudanças no decorrer do programa.

Os caracteres que podem ser usados para especificar formatos de impressão, podem ser os seguintes:

3.1. # (cancela ou libra) serve para especificar uma determinada posição numérica. Se houver um valor numérico com número de dígitos menor que o especificado, as posições mais à esquerda ficarão como espaços em branco.

```
Ex.: 10 A = 1234
     20 PRINT USING "####"; A
     RESULTARA' NA IMPRESSAO :
     1234

     10 A = 123
     20 PRINT USING "####"; A

     RESULTARA' NA IMPRESSAO :
     b123
```

3.2. . (ponto decimal) serve para indicar a separação da parte inteira da parte fracionária, podendo ser utilizado em qualquer parte de um formato numérico.

Ex.:
```
10 A = 123.456
20 B$ = "###.##"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

123.46

Observa-se que como o número especificado é maior que a máscara há, portanto, um truncamento da parte decimal, bem como um arredondamento automático, que ocorre somente quando há um truncamento da parte decimal.

Se for trocada a máscara para:

```
20 B$ = "####.#####"
```

RESULTARA' NA IMPRESSAO :

123.45700

Neste caso percebe-se que como a máscara é maior que o número especificado, tanto para a parte decimal como para a parte inteira, as posições à esquerda ficam com espaço em branco, enquanto que as à direita são preenchidas com zero, sem ocorrer arredondamento.

3.3. , (Vírgula) serve para indicar a separação em grupos de 3 algarismos para facilitar a leitura de números grandes. Deve ser colocada apenas na parte inteira do valor numérico, podendo, eventualmente, aparecer em qualquer posição sem ser nas ternárias.

Ex. 1:

```
10 A = 12345
20 B$ = "###,###"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

12,345

Este número deve ser lido como doze mil, trezentos e quarenta e cinco, pois a vírgula é apenas um separador (como é para nós o ponto decimal).

A vírgula pode estar em qualquer ponto da máscara para a separação a cada 3 dígitos, sendo necessária apenas uma vírgula para todas as separações necessárias.

Ex. 2:

```
10 A = 1234567.89
20 B$ = "#, ###, ###.##"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

1, 234, 567.89

OBS.: Caso o número de dígitos à esquerda do ponto decimal for maior que o formato fornecido na máscara de impressão, o número é normalmente impresso, aparecendo o sinal % na frente do número, indicando essa falha.

Ex.:
```
10 A = 123456.78
20 B$ = "##, ###.##"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

% 123, 456.78

3.4. ** (dois asteriscos) servem para preencher todas as posições que seriam espaços em branco com o caracter *. Devem ser colocados no início do formato, aumentando a máscara total em duas posições.

Ex.:
```
10 A = 123.45
20 B$ = "** ###, ###.##"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

***** 123.45

Se alterarmos o valor de A para

10 A = 123456.78

mantendo a mesma máscara teremos:

** 123, 456.78

Este artifício é muito útil quando temos impressões de valores numéricos que não devem ser "alterados" posteriormente como relatórios de contabilidade, impressões de cheques etc.

3.5. $ (um cifrão ou dólar) este símbolo se colocado no início da máscara sairá impresso na primeira coluna, à frente do número, ficando sempre alinhado em relação ao ponto decimal.

Ex.:
```
10 A = 1234.56
20 B$ = "$###,###.##"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

```
$          1.23
```

3.6. $$ (dois cifrões ou dólares) têm o mesmo efeito que o cifrão único, com a diferença que não mais ficará fixo na primeira coluna mas, ficará flutuante, ou seja, ocupará a primeira posição anterior ao número.

Ex.:
```
10 A = 12.34
20 B$ = "$$###,###.##"
20 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

```
        $     12.34
```

Se A for alterado para 1234.56 teremos:

```
        $ 1, 234.56
```

3.7. + (sinal de mais) este símbolo se colocado no início ou no final da máscara de impressão faz com que saia impresso o caracter "+" se o número for positivo, e o caracter "−" se o número for negativo.

Ex.:
```
10 A = 12.34
20 B$ = "+###,###.##"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

```
+          12.34
```

3.8. − (sinal de menos) este símbolo também pode ser colocado no início ou no final da máscara de impressão, faz com que saia impresso o caracter "−" se o número for positivo ou negativo, e neste caso 2 sinais negativos aparecerão.

Ex.:
```
10 A = 12.34
20 B$ = "-###,###.##"
30 PRINT USING B$; A
```

RESULTARA' NA IMPRESSAO :

– 12.34

Se A for alterado para – 1234.56 teremos:

– – 1,234.56

3.9. ↑↑↑↑ (quatro setas) esta especificação é colocada no final da máscara de impressão, fazendo com que o número saia impresso na notação científica ou exponencial (E ou D). Há micros que não possuem este caracter, sendo impresso o caracter [ou ^.

Ex.: 10 A = 123456.78
20 B$ = "##.####↑↑↑↑"
30 PRINT USING B$; A

RESULTARA' NA IMPRESSAO :

1.2345 D + 05

Se B$ for alterado para:

20 B$ = "##.##########↑↑↑↑"

teremos a impressão de:

1.2345678000 D + 05

Se A for alterado para – 12.345E + 17 mantendo a mesma máscara

– 1.2345000000 D + 18

Obs.: D é utilizado para notação exponencial e precisão dupla.

3.10. %...% (símbolos de porcentagem) estes símbolos limitam o tamanho de uma lista de caracteres alfanuméricos (que podem ser de uma variável alfanumérica). O número de espaços reservados será o número de espaços em branco entre os símbolos de mais de 2 caracteres, ou seja, incluindo os demarcadores.

Ex.: 10 A$ = "JORGE TREVISAN"
20 PRINT USING "% %"; A$

RESULTARA' NA IMPRESSAO :

JORGE

Alterando a máscara teremos:

```
10 A$ = "JORGE TREVISAN"
20 PRINT USING "%                    %"; A$
```

TEREMOS :

JORGE TREVISAN

Podemos ter, entretanto, várias variáveis alfanuméricas que devem ser impressas na mesma linha e para isso usamos os símbolos % concatenados, conforme o exemplo:

```
10 A$ = "JANEIRO"
20 B$ = "E' UM MES"
30 C$ = " QUENTE"
40 X$ = "%        %%        %%        %"
50 PRINT USING X$; A$; B$; C$
```

OBTENDO A IMPRESSAO :

JANEIRO E' UM MES QUENTE

Podemos também ter associações destas especificações apresentadas como por exemplo:

```
10 A = 12.3
20 B$ = "** $###,###.##"
30 PRINT USING B$ + A
```

RESULTANDO NA IMPRESSAO :

******* $ 12.30

Ou ainda, querendo deixar o valor em cruzeiros e eliminar as casas não utilizadas com o caracter "*" basta mudar B$ para:

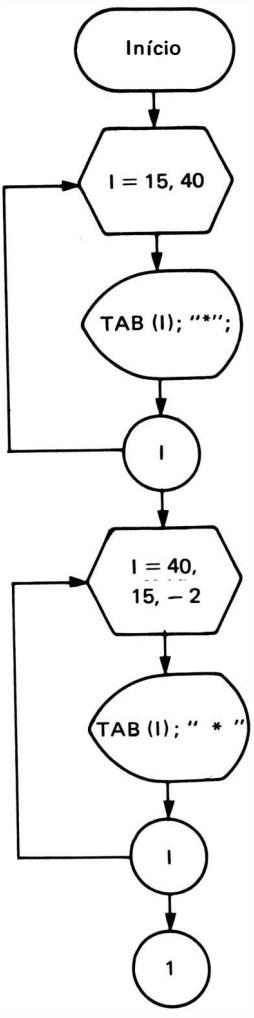
```
20 B$ = "CR$ **###,###.##"
```

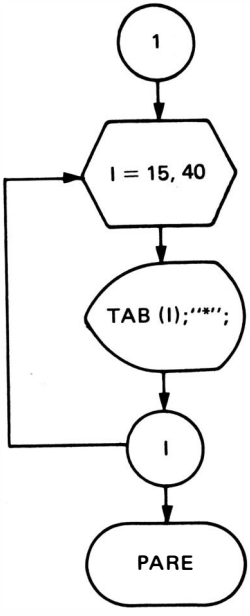
OBTENDO A SAIDA :

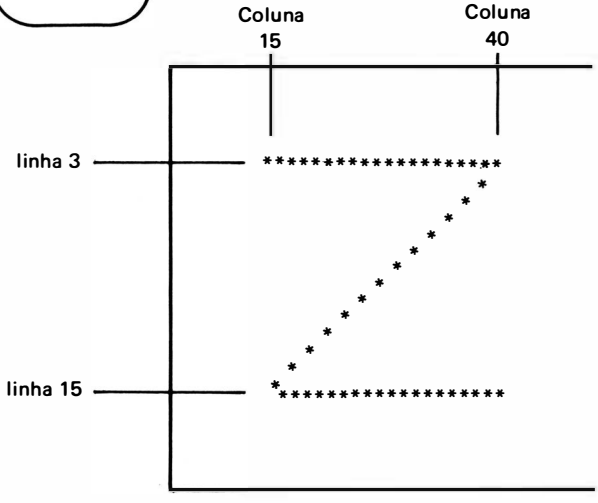
CR$ ******* 12.30

4. EXERCÍCIOS

1. Elaborar programa que imprima a letra Z na tela do micro. Considerar que o micro tenha um video de 16 linhas por 64 colunas e que a letra deve estar na posição indicada na figura.

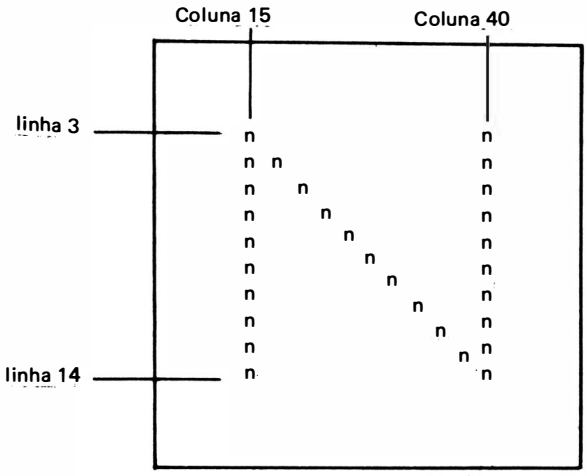


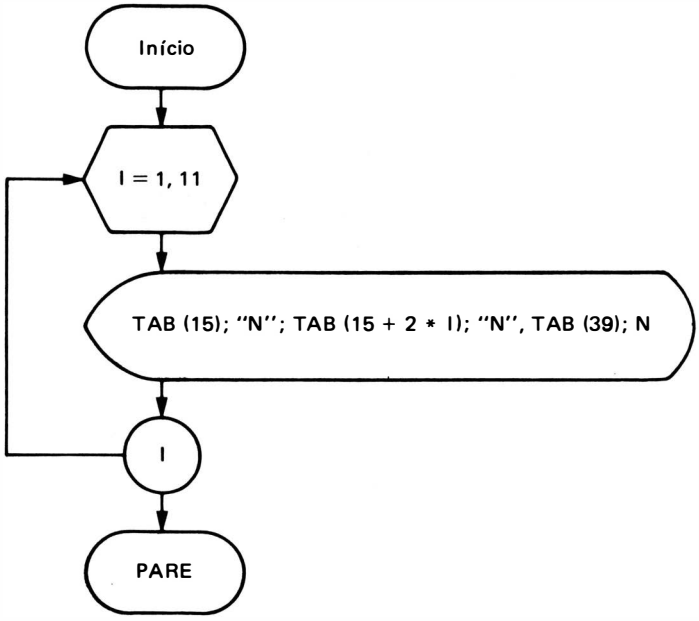





```
10 CLS
20 PRINT
30 PRINT
40 PRINT
50 FOR I = 15 TO 40
60     PRINT TAB (I) ;" * " ;
70 NEXT I
80 FOR I = 40 TO 15 STEP – 2
90     PRINT TAB (I) ;" * "
100 NEXT I
110 FOR I = 15 TO 40
120     PRINT TAB (I) ;" * " ;
130 NEXT I
140 STOP
```

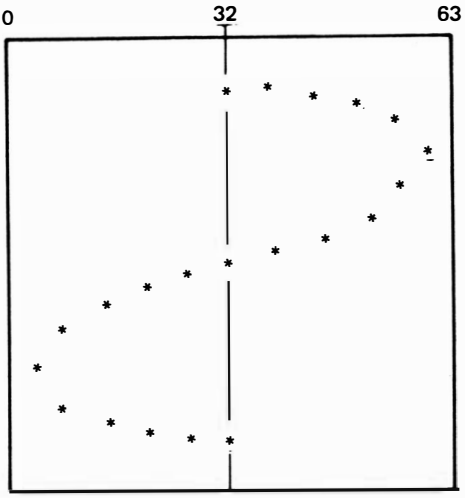
2. Idem ao anterior para a letra N com a seguinte composição:






```
10 CLS
20 PRINT
30 PRINT
40 PRINT
50 FOR I = 1 TO 11
60     PRINT TAB (15);"N";TAB (15 + 2 * I);"N";TAB (39);"N"
70 NEXT I
80 STOP
```

3. Elaborar programa que imprima na tela do micro o gráfico do seno. Considerar o micro com 64 colunas e que a linha "0" fique na coluna 32.



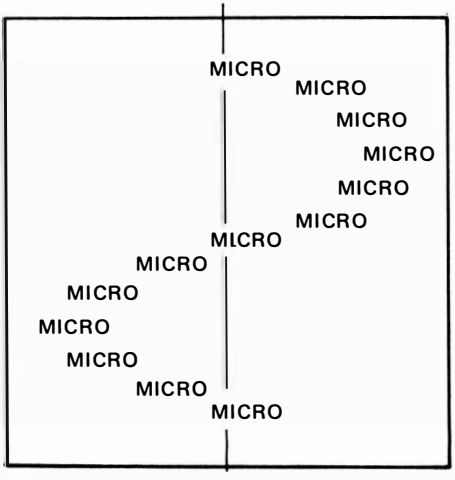

```
10 CLS
20 S = 10
30 PI = 3.1415927
40 A = PI/S
50 M = 100 * PI
60 FOR I = 0 TO M STEP A
70     X = SIN (I)
80     B = 32 + 31 * X
90     PRINT TAB (B); "*"
100 NEXT I
110 STOP
```

Devemos calcular o fator de escala:
sen x varia entre -1 e 1.

$\therefore$ se **0** é na coluna 32 e podemos variar de 0 a 63
temos: $32 + 31 * (\text{SIN}(X))$

$$\begin{cases} \text{Max} = 63 \\ \text{Min} = 1 \end{cases}$$

4. Modificar o programa anterior de forma a escrever a palavra "MICRO" segundo uma senóide.



Neste caso, o fator de escala deve ser alterado, pois a palavra "MICRO" acusa 5 posições na tecla:
teremos então

$$32 + 25 * \text{SIN}(X)$$

```
10 CLS
20 S = 10
30 PI = 3.1415927
40 A = PI/S
50 H = 100 * PI
60 FOR I = 0 TO M STEP A
70    X = SIN (I)
80    B = 32 + 25 * X
90    PRINT TAB (B) ;" MICRO"
100 NEXT I
110 STOP
```

OBS.: Modifique a linha 70 para:

a) X = COS (I)
b) X = SIN (1.5 * I) + COS (I)
c) X = SIN (0.9 * I) + COS (2.1 * I)

5. Elaborar programa que imprima em relatório o seguinte cabeçalho para a posterior exibição de resultados:

```
**********************************************************************
*                                                                    *
*                                                                    *
*                       R E L A T O R I O  M E N S A L               *
*                                                                    *
*                                                                    *
**********************************************************************
*                                    *                               *
*                                    *                               *
*              A T I V O             *        P A S S I V O          *
*                                    *                               *
*                                    *                               *
**********************************************************************
```

```
 10 FOR I = 0 TO 78
 20    LPRINT TAB (I);"*";
 30 NEXT I
 40 LPRINT
 50 LPRINT TAB (0) ;"*";TAB (78) ;"*"
 60 LPRINT TAB (0) ;"*";TAB (24) ;"RELATORIO MENSAL";
 65 LPRINT TAB (78) ;"*"
 70 LPRINT TAB (0) ;"*";TAB (78) ;"*"
 80 FOR I = 0 TO 78
 90    LPRINT TAB (I) ;"*";
100 NEXT I
110 LPRINT
120 LPRINT TAB (0) ;"*";TAB (40) ;"*";TAB (78) ;"*"
130 LPRINT TAB (0) ;"*";TAB (16) ;"A T I V O";TAB (40) ;"*";
135 LPRINT TAB (52) ;"P A S S I V O";TAB (78) ;"*"
140 LPRINT TAB (0) ;"*";TAB (40) ;"*";TAB (78) ;"*"
```

```
150 FOR I = 0 TO 78
160     LPRINT TAB (I) ;"* ";
170 NEXT I
180 LPRINT
```

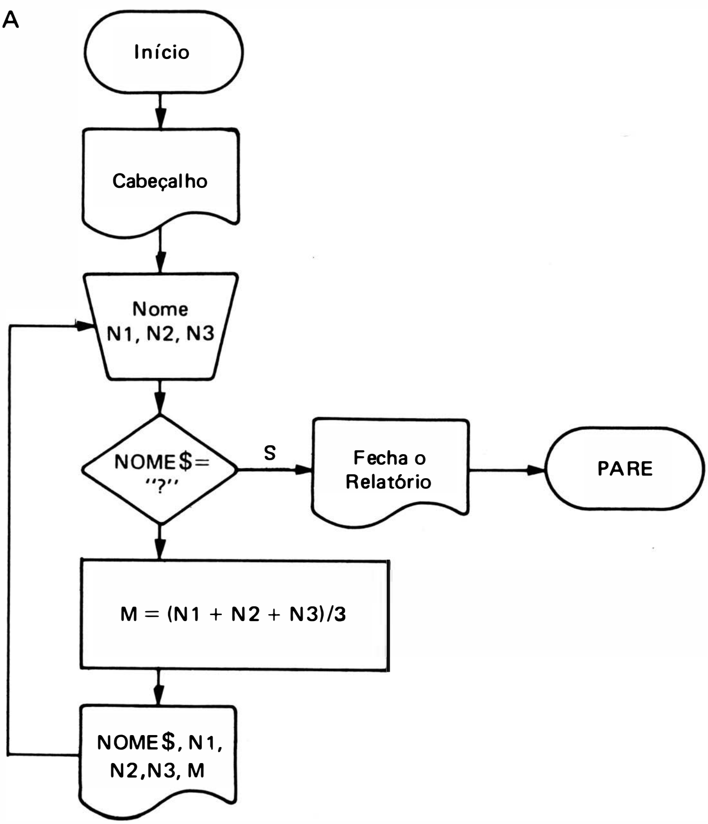
6. Escrever um programa BASIC para imprimir a saída do seguinte relatório:

```
*****************************************************************
*                         *        *        *        *
*          N O M E        * NOTA 1 * NOTA 2 * NOTA 3 * MEDIA
*                         *        *        *        *
*****************************************************************
*                         *        *        *        *
*   ANTONIO DOS SANTOS    * 8.0    * 7.0    * 5.5    * 6.83
*                         *        *        *        *
*                         *        *        *        *
```

onde o nome, nota 1 e nota 2, estão armazenados em instruções DATA do tipo 1000 DATA "ANTONIO SANTOS", 80, 7,0, 5.3 e a última linha DATA contenha DATA "?"; 0, 0, 0

A média é calculada somando-se as 3 notas e dividindo-se por 3.

FLUXOGRAMA



```
10 FOR I = 0 TO 78
20    PRINT  TAB (I) ;" * ";
30 NEXT I
40 PRINT
50 PRINT  TAB (0) ;" * "; TAB (13) ;"N O M E"; TAB (40) ;" * "; TAB (43) ;"NOTA 1";
60 PRINT  TAB (50) ;" * "; TAB (53) ;"NOTA 2"; TAB (60) ;" * "; TAB (63) ;"NOTA 3";
70 PRINT  TAB (70) ;" * "; TAB (72) ;"MEDIA"; TAB (78) ;" * "
80 FOR I = 0 TO 78
90    PRINT TAB (I) ;" * ";
100 NEXT I
110 PRINT
120 READ N$, N1, N2, N3
130 IF N$ = "?" GOTO 210
140 LET M = (N1 + N2 + N3)/3
150 LET M = 10 * M
160 LET M = INT (M)/10
170 PRINT TAB (0) ;" * ";TAB (2) ; N$ ;TAB (40) ;" * ";TAB (43) ; N1 ;
180 PRINT TAB (50) ; " * ";TAB (53) ; N2 ; TAB (60) ;" * ";TAB (63) ; N3 ;
190 PRINT TAB (70) ;" * " ;TAB (73) ; M ; TAB (78) ; " * "
200 GOTO 120
210 FOR I = 1 TO 78
220   PRINT TAB (I) ;" * ";
230 NEXT I
240 PRINT
250 STOP
500 DATA "ANTONIO DOS SANTOS", 8.0, 7.0, 5.5
510 DATA "ELAINE ADAMOV", 9.0, 8.5, 9.5
520 DATA "JOSE CARLOS DA SILVA", 6.5, 4.5, 7.5
530 DATA "NILDA A. S. SOUZA", 7.0, 7.5, 8.0
540 DATA "MARGARIDA BATISTA", 9.5, 9.0, 8.5
550 DATA "JOSE DA SILVA", 3.5, 4.0, 3.0
560 DATA "ALICE RODRIGUES", 6.5, 7.5, 9.0
570 DATA "?", 0, 0, 0
```

7. Elaborar um programa que imprima automaticamente uma nota fiscal, conforme a figura abaixo a partir da digitação do código do produto vendido e da quantidade. O código, o nome, o preço unitário do produto estão armazenados em uma linha DATA do tipo:

```
DATA 002, "CHAVE DE FENDA", 995.00
DATA 003, "CHAVE INGLESA", 5349.90
DATA 001, "ALICATE DE BICO", 3599.90
DATA 004, "MARTELO", 3500.00
DATA 005, "SERROTE", 9.839.90
DATA 006, "TALHADEIRA", 932.50
```

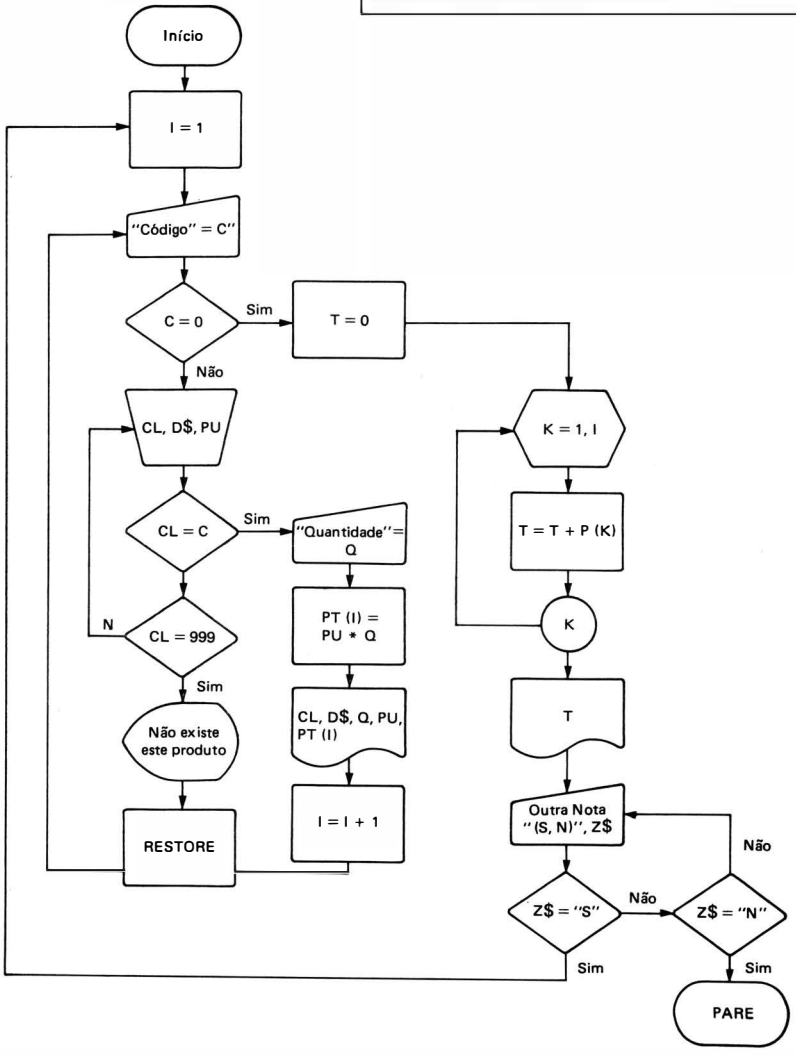
A lista é encerrada com o código fictício "999", enviando mensagem de inexistência do produto.

Para totalizar a nota, digitar o código "0" (zero). A nota fiscal tem o seguinte aspecto:

JT FERRAMENTAS LTDA – NF = nº 99999

CLIENTE ———————————

Código	Descrição	Quant.	P. Unitário	P. Total
999	% %	999	–.– – –.– – –,– –	.–.– – –.– – –,– –
		TOTAL	–.– – –.– – –.– – –,– –	



PROGRAMA

Levando-se em conta que a nota fiscal já vem impressa, e que entra na impressora em forma de formulário contínuo onde apenas é necessário preencher-se os espaços em branco, teremos:

```
10 DIM PT (15)
20 I = 1
30 CLS
40 INPUT "CODIGO = "; C
50 IF C = 0 GOTO 200
60 READ CL, D$, PU
70 IF CL = C GOTO 140
80 IF CL <> 999 GOTO 60
90 PRINT @ 460; "NAO EXISTE ESTE PRODUTO"
100 FOR K = 1 TO 1000
110 NEXT K
120 RESTORE
130 GOTO 30
140 INPUT "QUANTIDADE = "; Q
150 PT (I) = PU + Q
160 B$ = "##, ###, ###.##"
170 LPRINT TAB (2) ; CL ; TAB (15) ; D$ ; TAB (25) ; Q ;
180 LPRINT USING B$; TAB (30) ; PU
190 LPRINT USING B$ ; TAB (45) ; PT (I)
200 I = I + 1
210 GOTO 120
220 T = 0
230 FOR K = 1 TO I
240   T = T + P (K)
250 NEXT K
260 B1$ = "#, ###, ###, ###. ##"
270 LPRINT USING B1$ ; TAB (35) ; T
280 INPUT "OUTRA NOTA (S OU N) ? "; Z$
290 IF Z$ = "S" GOTO 30
300 IF Z$ = "N" THEN STOP
310 GOTO 280
500 DATA 002, "CHAVE DE FENDA", 995
510 DATA 003, "CHAVE INGLESA", 5349
520 DATA 001, "ALICATE DE BICO", 3599
530 DATA 004, "MARTELO", 3500
540 DATA 999, " ", 0
```

CAPÍTULO 12

DEFINIÇÃO DE FUNÇÕES PARTICULARES E SUBROTINAS

1. INTRODUÇÃO

Conforme foi visto anteriormente, existe uma série de funções, que a linguagem BASIC pode acessar diretamente e chamadas de Funções Biblioteca. Assim, se precisamos calcular a função $y = \sqrt{3\,e^t}$ usaremos as funções SQR para raiz quadrada e EXP para a função exponencial resultando na seguinte instrução:

```
LET Y = SQR (3 * (EXP (T)))
```

o que torna a programação muito mais simplificada, pois o programador não precisa desenvolver uma rotina para calcular a raiz quadrada, para a função exponencial etc. Entretanto, o número de Funções Biblioteca disponível é limitado, trazendo a necessidade de recursos adicionais que permitam definir novas funções, que não foram definidas na Biblioteca de funções, ou até mesmo programas inteiros que serão utilizados como auxiliares na execução de um outro programa. Estes recursos são: a instrução de definição de função e subprogramas tipo subrotinas.

2. INSTRUÇÃO DE DEFINIÇÃO DE FUNÇÃO (DEF FN)

Esta instrução permite ao programador definir a sua própria função, que não existe no conjunto de Funções Biblioteca, evitando assim a repetição de um determinado conjunto de cálculos.

Esta instrução é utilizada quando a função pode ser definida por meio de uma expressão aritmética que define um único valor numérico (que é o valor final da função). A expressão aritmética pode depender de um ou mais parâmetros, também chamados variáveis independentes ou fictícias pois servem apenas para indicar o tipo de argumento, além das operações que serão executadas no cálculo da função.

As variáveis fictícias são, portanto, totalmente inoperantes.

Forma Geral:

```
nl DEF FN l (p1, p2, . . . , pn) = e
```

onde:

nl representa o número da linha

l é uma letra (A – Z)

p1, p2, . . . , pn são parâmetros da função

e = expressão BASIC

```
Ex.: 10 DEF FNA (X) = X ↑ 2 + 4 * X + 1
     20 DEF FND (A, B, C) = B ↑ 2 – 4 * A * C
```

A função A (linha 10) permite definir um polinômio, que depende apenas da variável fictícia X (um só parâmetro), enquanto que a função D (linha 20) define o cálculo de um discriminante ($\Delta = b^2 - 4ac$) para o cálculo de raízes de um polinômio do segundo grau.

A função D depende de três parâmetros: A, B e C. Como o nome da função é composto por três letras, e as duas principais são obrigatoriamente FN, temos a possibilidade de criar até 26 funções próprias num mesmo programa (letras de A a Z) do tipo: FNA, FNB, FNC etc.

Esta instrução pode estar em qualquer parte do programa BASIC (desde que antes de uma referência a ela, na maioria dos micros). A prática mais comum, é contudo, o agrupamento de todas as definições de funções no início do programa.

Para usarmos esta função particular em nosso programa basta chamá-la pelo nome por nós batizado, como se fosse uma Função Biblioteca comum. O nome da função deve também ser seguido da lista de parâmetros para os quais desejamos calcular a função.

Ex. 1

```
10 DEF FNA (A, B, C) = SQR (B ↑ 2 – 4 * A * C)
20 INPUT R, S, T
30 X1 = (– S + FNA (R, S, T)  /  (2 * R)
40 X2 = (– S – FNA (R, S, T)  /  (2 * R)
50 PRINT X1, X2
60 GOTO 20
```

As linhas 30 e 40 deste pequeno programa avaliam a função definida na linha 10 que tem como parâmetros fictícios A, B e C. As variáveis R, S e T são os argumentos da função que queremos calcular e são substituídos, respectivamente, por A, B e C.

Ex. 2:

```
10 DEF FNW (T) = T ↑ 2 – 5 * T + 2
      .
      .
      .
      .
50 Y = FNW (X)
```

A linha 50 utiliza a função W, definida na linha 10 com a variável X, como argumento ou parâmetro real.

3. SUBROTINAS

Com a finalidade de reduzir o trabalho de programação existe uma forma de se estruturar um programa, que é a subrotina, de modo a evitar a repetição de uma determinada seqüência de instruções várias vezes em diferentes partes do programa.

As subrotinas são pequenos programas embutidos em um outro programa, chamado principal, podendo ser referenciadas em qualquer parte do programa, como ocorre com as definições de novas funções. Entretanto, as subrotinas não possuem um nome associado, podendo ser empregadas na determinação de uma ou mais quantidades numéricas, sem a necessidade de uso de argumentos, o que torna a subrotina uma ferramenta muito mais poderosa que a instrução de definição de função.

3.1. Instrução GOSUB

Esta instrução tem por finalidade transferir o fluxo do processamento para uma subrotina indicada pelo número que segue a declaração:

Forma Geral:

nl GOSUB n

onde:

nl indica o número da linha da instrução

n representa o número da linha onde inicia a subrotina

O valor de n pode ser uma constante ou uma variável que conterá o valor da linha da subrotina.

Ex 1:

```
100 GOSUB 1000
```

A transferência de processamento para uma subrotina que inicia na linha nº 1000 do programa é indicada no exemplo a seguir, tendo o mesmo efeito do exemplo anterior.

Ex. 2:

```
10 A = 1000
   .
   .
   .
   .
100 GOSUB A
```

Após a execução da subrotina o controle do processamento retornará para a instrução seguinte à instrução GOSUB que a chamou. Convém observarmos que um programa principal pode chamar em diversas posições diferentes a execução de uma determinada subrotina, e o endereço de retorno armazenado será o da instrução GOSUB específica que chamou a execução da subrotina.

```
10 REM PROGRAMA EXEMPLO
      .
      .
      .
100 GOSUB 1000
110 LET J = A + 1
      .
      .
      .
      .
200 GOSUB 1000
210 PRINT K
      .
      .
      .
      .
1000 REM SUBROTINA 1
      .
      .
      .
      .
1500 REM FIM DA SUBROTINA 1
```

Neste exemplo, se a linha 100 for executada, o controle do processamento será transferido para a linha 1000 onde inicia a subrotina 1 e após sua execução o fluxo do processamento retornará para a linha 110. Se a linha 200 for executada, o retorno será para a linha 210.

3.1.1. Subrotinas Embutidas

Uma subrotina pode conter em seu interior uma referência a uma outra subrotina, que por sua vez pode referenciar uma terceira e assim sucessivamente, até um nível máximo que depende do microcomputador, variando de 8 a 12 níveis de embutimento.

Ex.:

```
10 REM INICIO DO PROGRAMA PRINCIPAL
        .
        .
        .
        .
100 GOSUB 1000
110 X = X + 1
        .
        .
        .
        .
200 GOSUB 1000
210 A = J + X
        .
        .
        .
        .
500 REM FINAL DO PROGRAMA PRINCIPAL

1000 REM SUBROTINA 1
        .
        .
        .
1100 GOSUB 2000
1110 M = M + 4
        .
        .
        .
1200 GOSUB 2000
1210 K = J + 5
        .
        .
        .
1500 REM FINAL DA SUBROTINA 1

2000 REM SUBROTINA 2
        .
        .
        .
2300 REM FINAL DA SUBROTINA 2
```

Neste exemplo temos um programa principal e duas subrotinas. O programa principal que inicia na linha 10 e termina na linha 500 tem em seu interior referências a uma subrotina iniciada na linha 1000. Esta por sua vez referencia uma outra subrotina iniciada na linha 2000.

Supondo-se que o programa principal está sendo executado, ao encontrar a linha 100 o fluxo do processamento é transferido para a subrotina da linha 1000 que por sua vez quando encontrar a linha 1100 transferirá o fluxo do processamento para a subrotina da linha 2000. Quando terminada esta segunda subrotina, o fluxo do processamento retornará à linha 1110, continuando a execução da subrotina 1, que tem nova chamada à segunda subrotina na linha 1200 repetindo o processo anterior só que voltando para a linha 1210. Terminada a execução da subrotina 1 o fluxo do processamento retornará para a linha 110. Encontrando a linha 200, todo o processo descrito é repetido retornando agora para a linha 210.

OBS.: Quando usamos subrotinas embutidas, se uma subrotina 1 chama para executar uma subrotina 2, esta não pode chamar a subrotina 1.

3.2. Instrução RETURN

Esta instrução tem por finalidade encerrar a execução de uma subrotina, fazendo com que o fluxo de processamento seja transferido para a instrução imediatamente seguinte à instrução GOSUB que chamou esta subrotina.

Forma Geral:

nl RETURN

onde:

nl é o número da linha que termina a subrotina

Ex.:

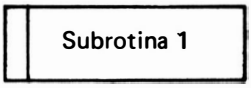
```
10 REM PROGRAMA PRINCIPAL
      .
      .
100 GOSUB 1000
      .
      .
500 REM FINAL DO PROGRAMA PRINCIPAL
1000 REM SUBROTINA 1
      .
      .
      .
1500 RETURN
1510 REM FINAL NA SUBROTINA 1
```

A instrução RETURN é equivalente ao STOP ou END do programa principal, ou seja, na subrotina estes comandos são substituídos pelo RETURN. Em uma mesma subrotina pode haver várias instruções de RETURN e neste caso, a primeira desta instrução encontrada provocará o retorno ao programa principal.

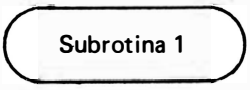
Convém observarmos que o fluxo de processamento *NUNCA* poderá ser transferido da subrotina para o programa principal por meio de uma instrução de desvio como GOTO ou IF. A única instrução que tem esta finalidade é o RETURN.

3.3. Representação de Subrotinas no Diagrama de Blocos

É comum encontrarmos autores que utilizam o retângulo da instrução de atribuição como bloco de chamada para subrotina. Entretanto, o bloco mais geral é o retângulo modificado conforme a figura:



Para a instrução de início da subrotina utiliza-se o bloco de início com o nome da subrotina em seu interior:



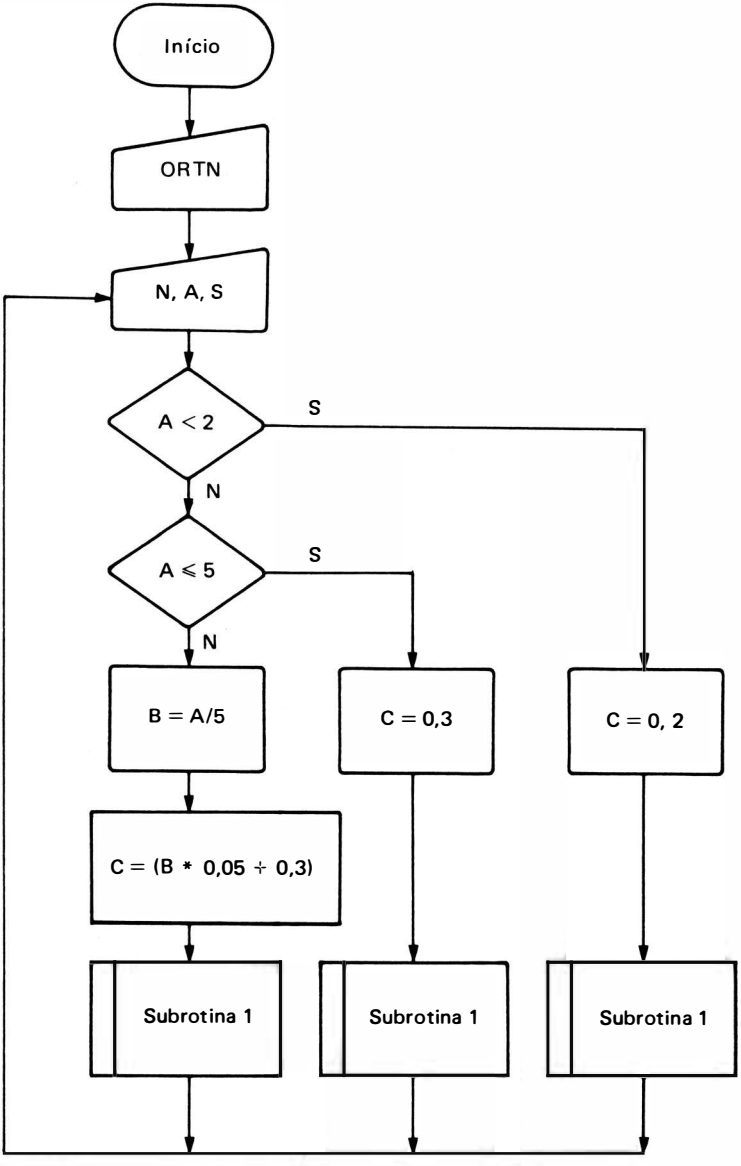
Para o comando de RETURN utiliza-se o mesmo bloco com a palavra RETURN (ou "VOLTA") em seu interior:

Ex.: Desenvolver fluxograma e programa BASIC que calcule um abono para os funcionários de uma firma, de acordo com a seguinte tabela:

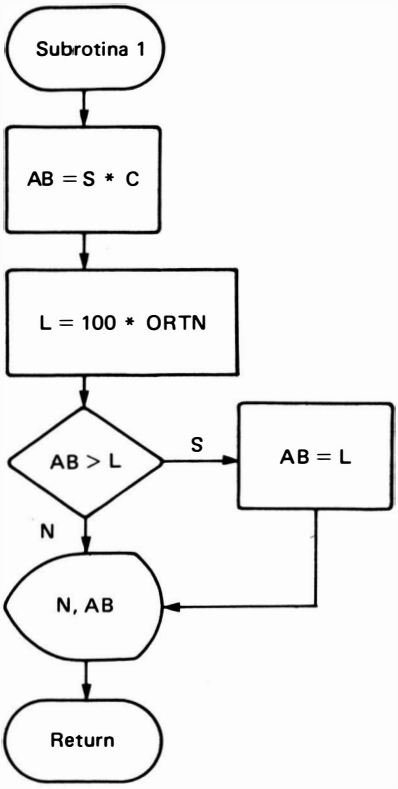
Nº de Anos de Trabalho	Abono
até 2	20%
de 2 a 5	30%
acima de 5 anos	30% + 5% para cada 5 anos

O abono máximo é limitado a 100 ORTNs.
A entrada de dados deve conter:

- nº de funcionários,
- nº de anos,
- salário atual



N = nº de funcionários
A = nº de anos de casa
S = salário atual
C = abono merecido (%)
AB = abono que será dado



PROGRAMA BASIC

```
10 INPUT "ORTN = "; ORTN
20 INPUT "NUMERO DO FUNCIONARIO = "; N
30 INPUT "NUMERO DE ANOS DE CASA = "; A
40 INPUT "SALARIO ATUAL = "; S
50 IF A < 2 THEN GOTO 140
60 IF A < =
60 IF A < = 5 THEN GOTO 110
70 B = A  5
80 C = (B * 0.05 + 0.3)
90 GOSUB 200
100 GOTO 20
110 C = 0.3
120 GOSUB 200
130 GOTO 20
140 C = 0.2
150 GOSUB 200
160 GOTO 20
200 REM SUBROTINA ABONO
210 AB = C * S
220 L = 100 * ORTN
230 IF AB > L THEN GOTO 260
240 PRINT N, AB
250 RETURN
260 PRINT N, L
270 RETURN
280 REM FIM DA SUBROTINA ABONO
```

Comentários:

A subrotina 200 é chamada em 3 partes diferentes do programa, ou seja, nas linhas 90, 120 e 150. Se, por exemplo, a subrotina 200 for chamada pela linha 120 do programa, o retorno da subrotina será para a linha 130 e assim por diante. Este programa deve ser interrompido pela tecla BREAK.

4. EXERCÍCIOS

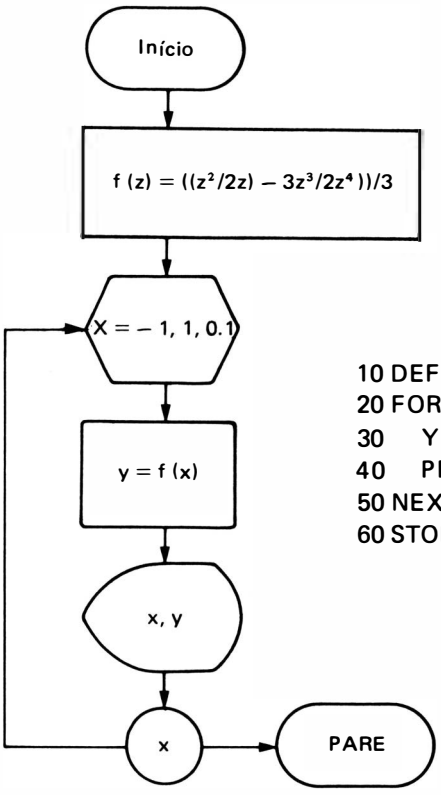
1. Desenvolver fluxograma e programa BASIC para o cálculo da função:

$$f(x) = \frac{(x^2/2x) - (3x^3/2x^4)}{3}$$

para x variando de -1 a 1 com passo 0,1.

Exibir os valores de x e $f(x)$.

Usar a instrução de definição de função:




```
10 DEF FNA (Z) = ((Z ↑ 2/2 * Z) – (3 * Z ↑ 3/2 * Z ↑ 4))/3
20 FOR X = – 1 TO 1 STEP 0.1
30     Y = FNA (X)
40     PRINT X, Y
50 NEXT X
60 STOP
```

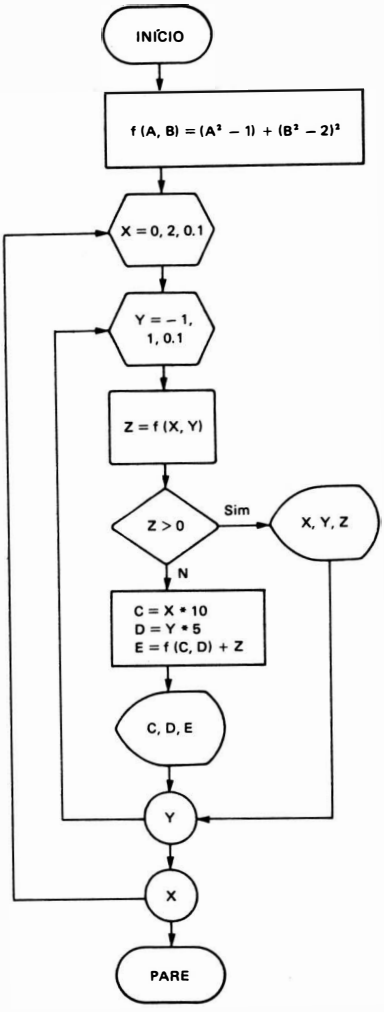
Comentários:

Na linha 10 a variável Z é fictícia servindo apenas para indicar as operações a serem efetuadas com a variável.

2. Desenvolver fluxograma e programa BASIC para o cálculo da função:

$$F(x, y) = (x^2 - 1)^2 + (y^2 - 2)^2$$

para x variando no intervalo $(0 - 2)$ e y de $[-1$ a $1]$ de 0,1 em 0,1. Se o valor da função for negativo, aumentar o valor de x em 10 vezes e o valor de y em 5 vezes e calcular o valor da função, somando o valor calculado anteriormente. Imprimir uma tabela de (x, y) e $f(x, y)$.



EXERCICIO 2 CAP 11

```
10 DEF FNE (A, B) = (A ↑ 2 – 1) ↑ 2 + (B ↑ 2 – 1) ↑ 2
20 FOR X = 0 TO 2 STEP 0.1
30    FOR Y = – 1 TO 1 STEP 0.1
40       Z = FNE (X, Y)
50       IF Z > 0 THEN GOTO 110
60       C = X * 10
70       D = Y * 5
80       E = FNE (C, D) + Z
90       PRINT C, D, E
100      GOTO 120
110      PRINT X, Y, Z
120   NEXT Y
130 NEXT X
140 STOP
```

3. Desenvolver fluxograma e programa BASIC para o cálculo da função:

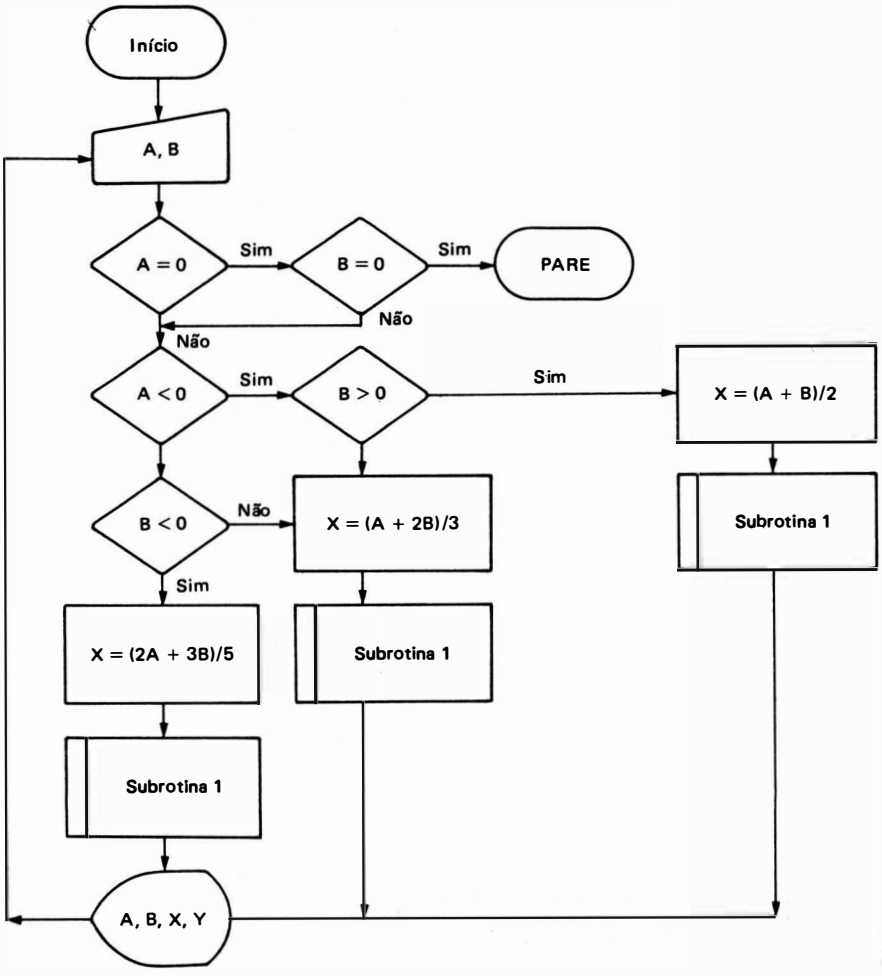
$$f(x) = \begin{cases} x^2 \text{ para } x \geqslant 2 \\ 1 \text{ para } -2 < x < 2 \\ \log |x| \text{ para } x \leqslant -2 \end{cases}$$

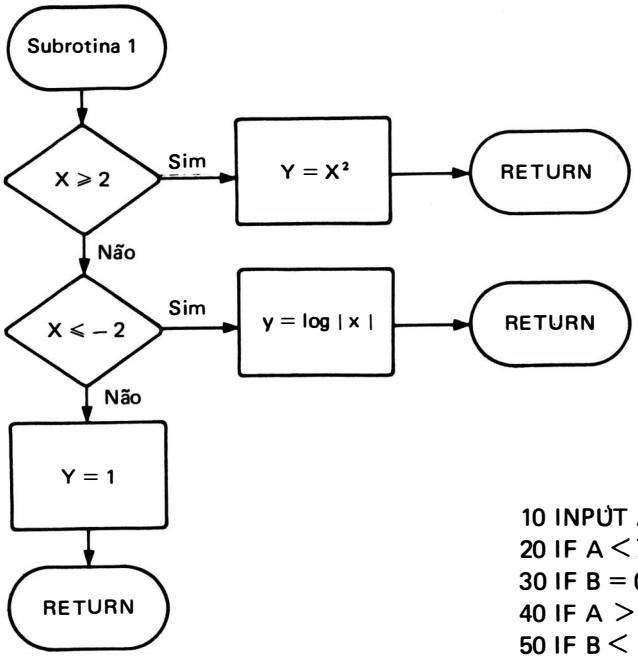
Os valores de x são calculados a partir de um par de valores (A, B) digitados na entrada conforme a tabela:

Para A e B $> 0 \rightarrow$ X = (A + B)/2	Para A > 0 e B < 0
Para A e B $< 0 \rightarrow$ X = (3A + 2B)/5	ou A < 0 e B > 0
	X = (A + 2B)/3

Imprimir tabela contendo A, B, X e C (X).

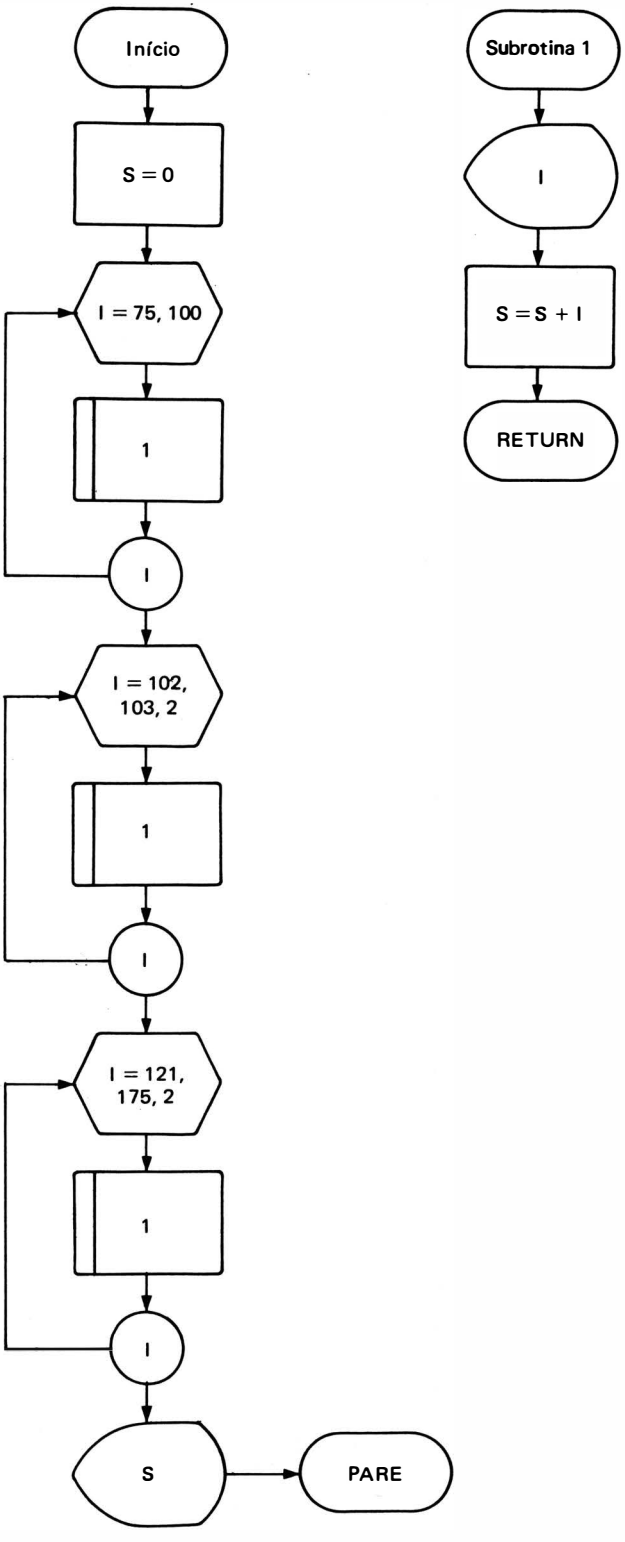
Encerrar o processo quando for digitado A = 0 e B = 0.






```
10 INPUT A, B
20 IF A <> 0 GOTO 40
30 IF B = 0 GOTO 180
40 IF A > 0 GOTO 130
50 IF B < 0 GOTO 100
60 LET X = (A + 2 * B)/3
70 GOSUB 500
80 PRINT A, B, X, Y
90 GOTO 10
100 LET X = (2 * A + 3 * B)/5
110 GOSUB 500
120 GOTO 80
130 IF B >= 0 GOTO 150
140 GOTO 60
150 LET X = (A + B)/2
160 GOSUB 500
170 GOTO 80
180 STOP
500 REM SUBROTINA FUNCAO
510 IF X >= 2 GOTO 570
520 IF X <= – 2 GOTO 550
530 LET Y = 1
540 RETURN
550 LET Y = LOG < ABS (X))
560 RETURN
570 LET Y = X ↑ 2
580 RETURN
```

4. Elaborar fluxograma e correspondente programa BASIC para exibir os números inteiros de 75 a 100, os pares de 102 a 130, e os ímpares de 131 a 175 na sua ordem natural, exibindo no final sua soma total.


Início
S = 0
I = 75, 100
1
I
I = 102, 103, 2
1
I
I = 121, 175, 2
1
I
S
PARE
Subrotina 1
I
S = S + I
RETURN

```
10 REM INICIO DO PROGRAMA PRINCIPAL
20 LET S = 0
30 FOR I = 75 TO 100
40    GOSUB 500
50 NEXT I
60 FOR I = 102 TO 130 STEP 2
70    GOSUB 500
80 NEXT I
90 FOR I = 131 TO 175 STEP 2
100    GOSUB 500
110 NEXT I
120 PRINT "SOMA TOTAL = ";S
130 STOP
500 REM SUBROTINA 1
510 PRINT I;
520 LET S = S + I
530 RETURN
540 REM FIM DA SUBROTINA 1
```

5. Um processo químico exige monitoração das variáveis: temperatura, vazão e pressão.

   Fazer fluxograma e programa BASIC, que forneça um menu principal, conforme a figura 1, para escolha de uma destas variáveis, para o cálculo de estatísticas de média e somatória de um conjunto de valores digitados, até que seja digitado o valor Ø (zero), escolhidas também por um segundo menu (Fig. 2).

   Imprimir o resultado da estatística pedida, retornando ao menu secundário para escolha de outra estatística daquela variável.

   Prever opção para volta ao menu principal (no secundário) para troca de variável monitorada, bem como opção para término do programa:

Figura 1

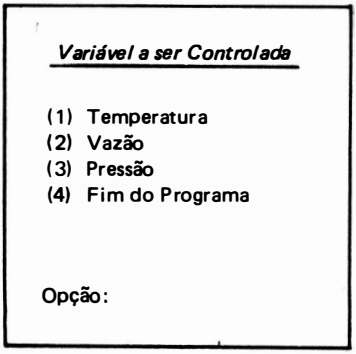
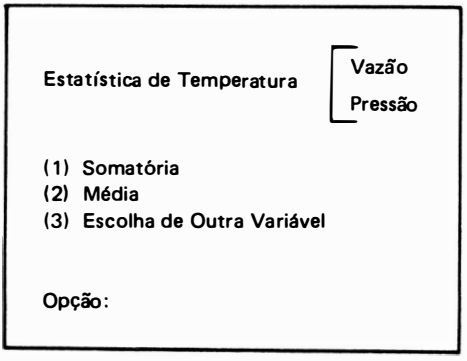
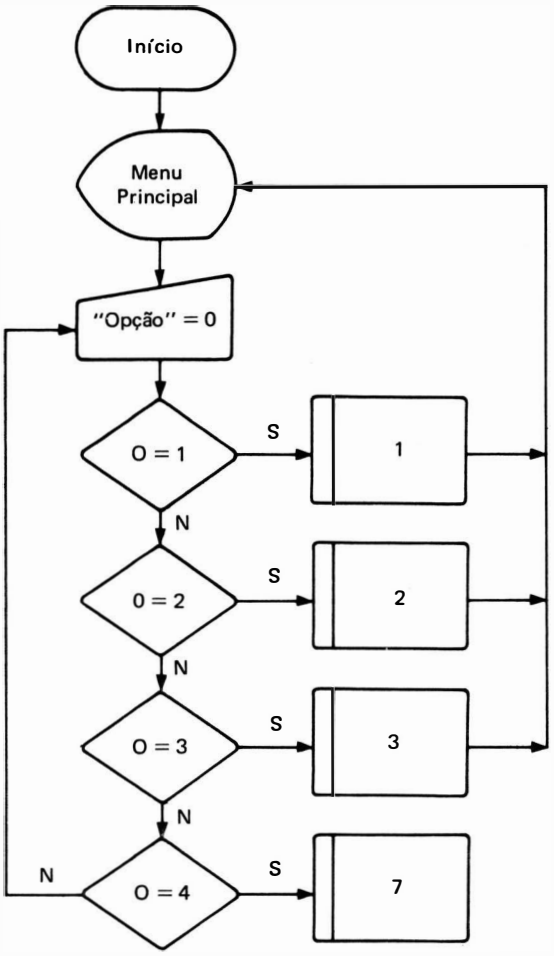
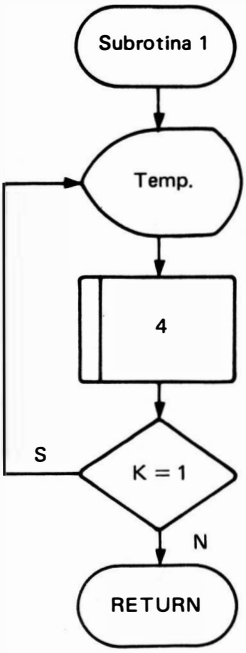


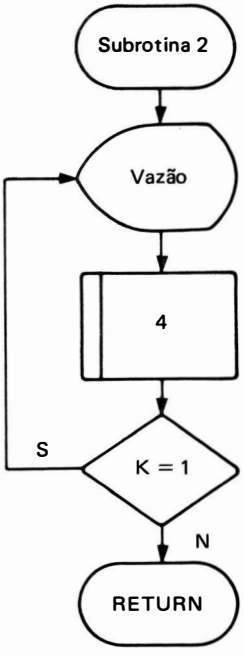
Figura 2

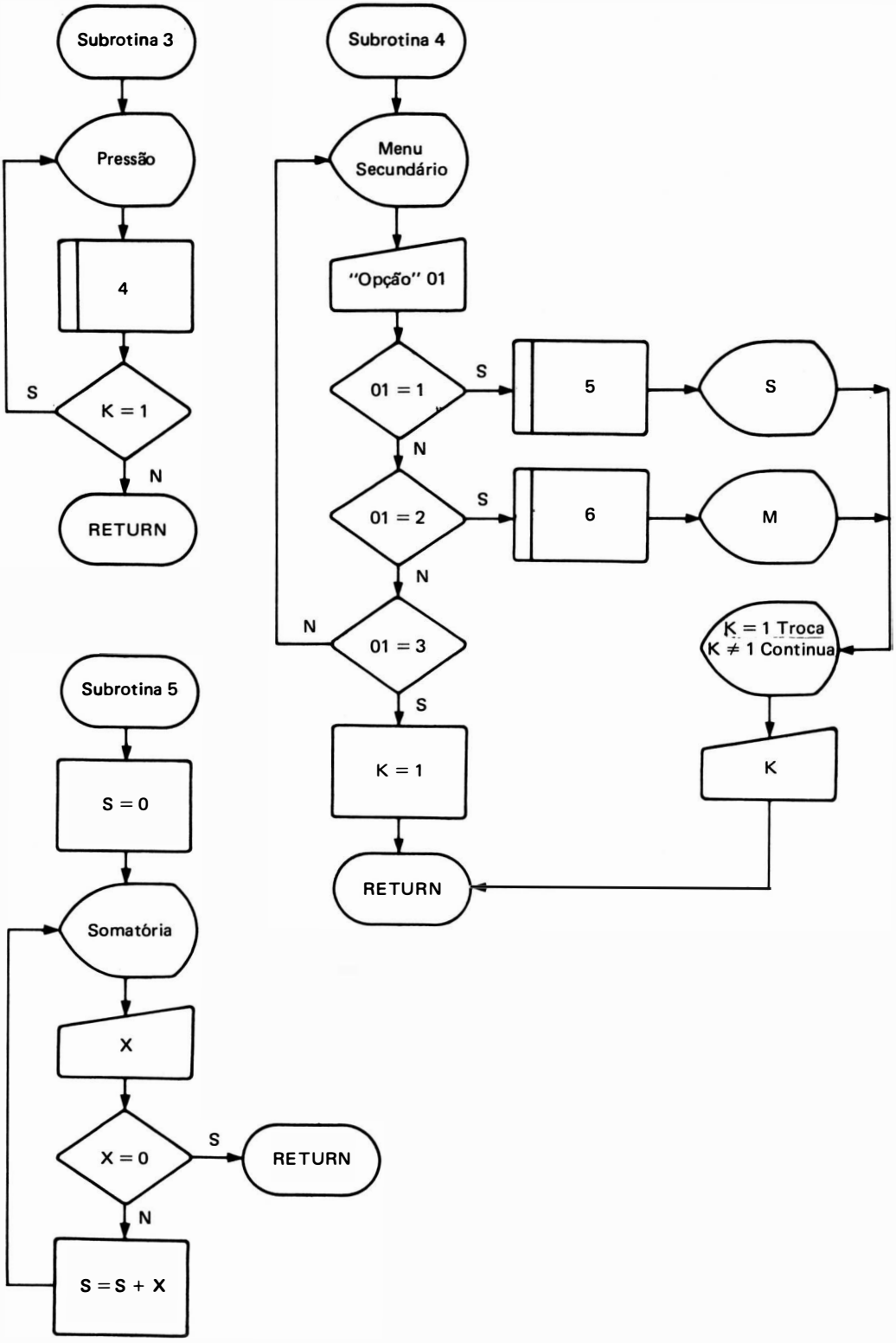


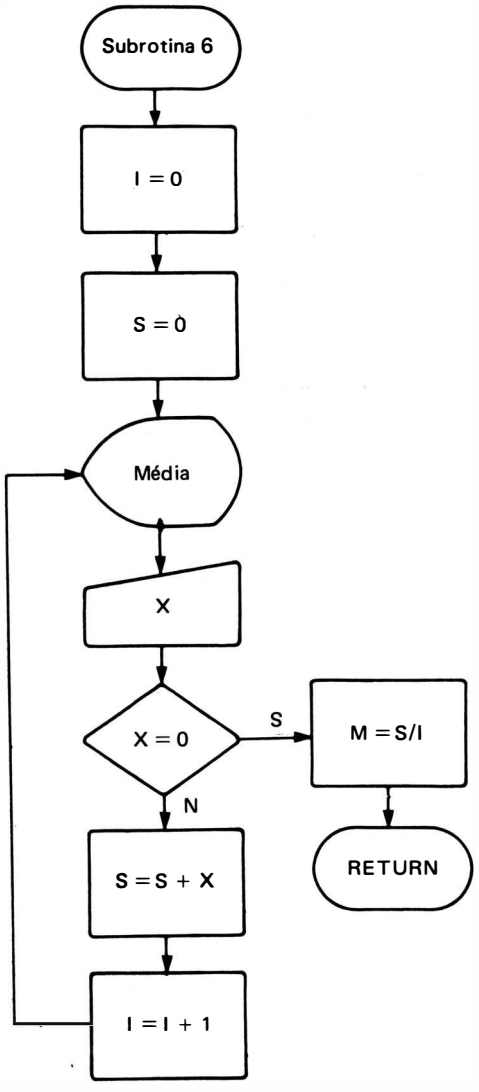
PROGRAMA PRINCIPAL

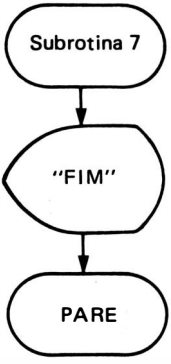




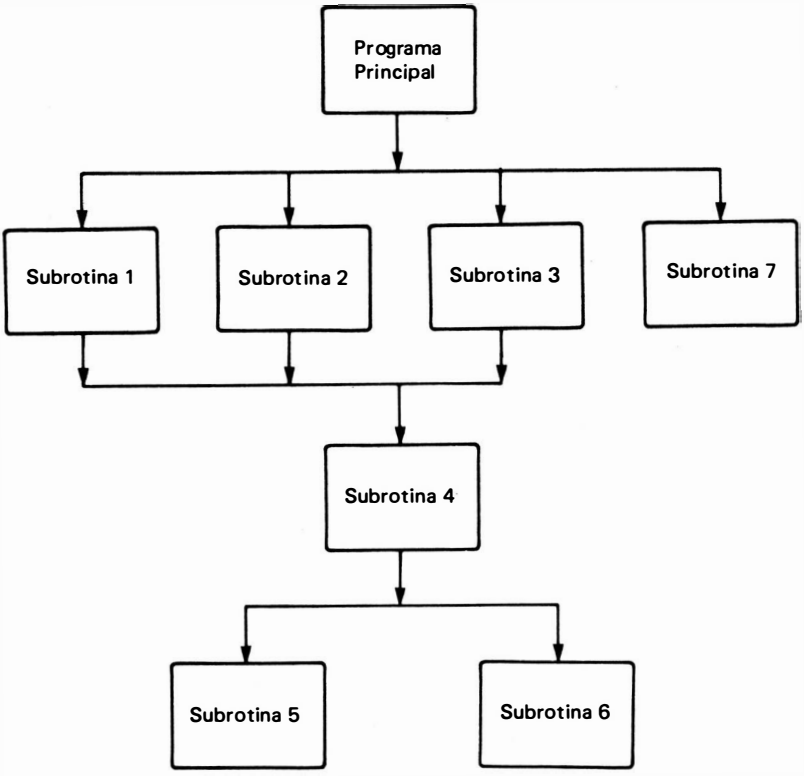



Subrotina 3
Pressão
4
K = 1
S
N
RETURN
Subrotina 4
Menu Secundário
"Opção" 01
01 = 1
S
5
S
N
01 = 2
S
6
M
N
01 = 3
N
S
K = 1
RETURN
K = 1 Troca
K ≠ 1 Continua
K
Subrotina 5
S = 0
Somatória
X
X = 0
S
RETURN
N
S = S + X


Subrotina 6
I = 0
S = 0
Média
X
X = 0
S
M = S/I
RETURN
N
S = S + X
I = I + 1


Subrotina 7
"FIM"
PARE

Esquema das chamadas subrotinas




```
10 REM PROGRAMA PRINCIPAL
20 CLS
30 PRINT "VARIAVEL A SER CONTROLADA :"
40 PRINT
50 PRINT
60 PRINT "(1)  TEMPERATURA"
70 PRINT "(2)  VAZAO"
80 PRINT "(3)  PRESSAO"
90 PRINT "(4)  FIM DE PROGRAMA"
100 PRINT
110 PRINT
120 INPUT "OPCAO = ";O
130 IF O = 1 GOSUB  1000
140 IF O = 2 GOSUB  2000
150 IF O = 3 GOSUB  3000
160 IF O = 4 GOSUB  7000
170 GOTO  10
1000 REM SUBROTINA 1
1010 CLS
1020 PRINT "ESTATISTICAS DE TEMPERATURA"
1030 GOSUB  4000
1040 IF K = 1 RETURN
```

```
1050 GOTO 1010
2000 REM SUBROTINA 2
2010 CLS
2020 PRINT "ESTATISTICAS DE VAZAO"
2030 GOSUB  4000
2040 IF K = 1 RETURN
2050 GOTO 2010
3000 REM SUBROTINA 3
3010 CLS
3020 PRINT "ESTATISTICAS DE PRESSAO"
3030 GOSUB  4000
3040 IF K = 1 RETURN
3050 GOTO 3010
4000 REM SUBROTINA 4
4010 PRINT
4020 PRINT
4030 PRINT "(1)  SOMATORIA"
4040 PRINT "(2)  MEDIA"
4050 PRINT "(3)  ESCOLHA DE OUTRA VARIAVEL"
4060 PRINT
4070 PRINT
4080 INPUT "OPCAO = "; O1
9090 IF O1 = 1 GOTO 4130
4100 IF O1 = 2 GOTO 4160
4110 IF O1 = 3 LET K = 1
4120 RETURN
4130 GOSUB  5000
4140 PRINT "  SOMATORIA = "; S
4150 GOTO 4180
4160 GOSUB  6000
4170  "  MEDIA = "; M
4180 PRINT
4190 PRINT
4200 PRINT "DIGITE 1 PARA TROCAR DE VARIAVEL, OU OUTRO
NUMERO PARA CONTINUAR"
4220 INPUT K
4230 RETURN
5000 REM SUBROTINA 5
5010 LET S = 0
5020 CLS
5030 PRINT "PARA TERMINAR A SOMATORIA DIGITE O NUMERO 0 (ZERO)"
5040 PRINT
5050 PRINT
5060 INPUT "VALOR = "; X
5070 IF X = 0 RETURN
5080 LET S = S + X
5090 GOTO 5020
6000 REM SUBROTINA 6
```

```
6010 LET I = 0
6020 LET S = 0
6030 CLS
6040 PRINT "PARA CALCULAR A MEDIA DIGITE O NUMERO 0 (ZERO) "
6050 PRINT
6060 PRINT
6070 INPUT "VALOR = "; X
6080 IF X = 0 GOTO 6120
6090 LET S = S + X
6100 LET I = I + 1
6110 GOTO 6030
6120 LET M = S/I
6130 RETURN
7000 REM SUBROTINA 7
7010 CLS
7020 PRINT @ 605, "F I M"
7030 FOR I = 1 TO 5
7040   PRINT
7050 NEXT I
7060 END
```

CAPÍTULO 13

ACESSO A POSIÇÕES DE MEMÓRIA

1. MEMÓRIAS

Toda computação matemática, seja ela mental, mecânica ou eletrônica, requer um sistema de armazenamento de valores de algum tipo, como por exemplo, escrever um resultado parcial de uma conta em um pedaço de papel, "guardá-lo" no cérebro ou ainda em algum dispositivo mecânico ou eletrônico.

Uma computação eletrônica não pode ser conseguida sem um sistema de armazenamento de valores, de tal forma que a capacidade de um computador é medida em função da capacidade de memória disponível, bem como sua velocidade de operação.

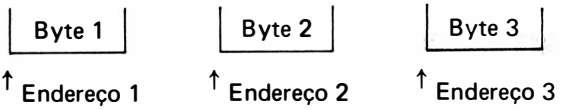
Como vimos anteriormente, o computador trabalha internamente com uma linguagem de máquina que manipula "bits" de informação.

Os "bits" representados pelos algarismos 0 e 1 formam os números binários que são organizados em grupos de 8 chamados "BYTES". Por exemplo, a seqüência de 8 bits 10101111 representa 1 Byte. Podemos verificar que o valor de 1 Byte pode variar de 00000000 até 11111111 que corresponde à faixa de 0 a 225 decimal.

A capacidade de memória de um computador é medida em Bytes. É comum ouvirmos falar em computador com 16KBytes de memória o que significa que o usuário tem disponível aproximadamente 16.000 posições de memória. A rigor, temos disponível um pouco mais, pois em computação 1K significa 1024 ou (2^{10}) e não 1000 como estamos acostumados, o que nos fornece 16.384 posições diferentes de memória. O mesmo acontece com outras constantes do tipo M (Mega) que vale 1024 $\times$ 1000 Bytes, G (Giga) que vale 1024 $\times$ 1000 $\times$ 1000 Bytes etc.

Devemos ressaltar ainda que cada Byte de memória do computador possui um endereço, que é também um conjunto de bits, que corresponde à localização do Byte na memória, não devendo nunca ser confundido com o conteúdo do Byte que é a própria informação armazenada.

Esquematicamente:



Podemos classificar a memória do computador como sendo de 2 tipos: "voláteis" e "não voláteis".

- Memórias do tipo Volátil: são aquelas que perdem a informação armazenada quando há falta de energia, como por exemplo, as memórias eletrônicas a semicondutores.

- Memórias do tipo Não Volátil: são aquelas que não perdem a informação armazenada, mesmo quando há falta de energia, como por exemplo, as memórias de núcleo de ferrite em discos magnéticos, em fitas etc.

1.1. Memórias Eletrônicas

Com o avanço da tecnologia de semicondutores tornou-se possível a construção de dispositivos eletrônicos capazes de armazenar informações utilizando-se de técnicas de construção de circuitos monolíticos, como a Bipolar e a MOS ("Metal Oxide Semiconductor"). Em ambas as técnicas, a unidade mínima de informação é armazenada em circuitos eletrônicos chamados "flip-flops", que é um circuito digital que segue a lógica binária, isto é, tem apenas 2 estados exclusivos: "0" e "1", que é o nosso bit. Estes circuitos, devido a sua relativa simplicidade, podem ser encapsulados em uma pastilha ("Chip") numa larga escala de integração obtendo em dimensões reduzidas uma grande quantidade de memória disponível.

1.1.1. Tipos de Memórias Eletrônicas

Podemos separar as memórias eletrônicas em 2 tipos básicos: Memórias RAM e memórias ROM.

As memórias RAM, forma abreviada de "Random Acess Memory", ou seja, memórias de acesso aleatório, são memórias que permitem tanto leitura como escrita em uma determinada posição ou endereço. São memórias do tipo volátil, ou seja, ao desligarmos a energia, toda informação nela contida desaparece. Isto é inconveniente quando temos informações que devem ficar armazenadas permanentemente como o interpretador BASIC, o Monitor Assembler, o nosso programa etc. A solução é usarmos um outro tipo de memória de acesso aleatório chamada ROM ou "Real Only Memory", que são memórias que permitem fazer apenas leituras de seu conteúdo. Esta memória é do tipo não volátil e armazena todas as informações que não podem ser perdidas com o desligamento da energia.

Existem vários tipos de memória ROM que podemos destacar:

- ROM Programada por Máscara: é uma memória cujas células de armazenamento são projetadas pelo fabricante e construídas permanentemente com Bytes imutáveis de "zeros" e "uns".

- PROM ou "Programmable Read Only Memory": é um tipo de memória, cujas células de armazenamento interno são construídas com um pequeno fusível interno, tendo originalmente todas as posições o valor "1". Para programar esta memória deve-se selecionar, para cada endereço, os bits que devem tornar-se "0" pela aplicação de uma alta voltagem que queima definitivamente o fusível interno. Portanto é possível trocar-se todos os "1" por "0", não sendo mais possível retornar ao estado original.

- EPROM ou "Erasable-Programmable Read Only Memory": é o tipo mais popular de memórias para uso em microcomputadores, pois o usuário pode apagar e reprogramar a ROM quando necessário.

Estas memórias baseiam-se em circuitos cujas situações de "conduz" ou "não conduz" resultam nos valores binários "1" e "0". Para apagar estas memórias basta expor a pastilha a uma fonte de luz ultravioleta, uma vez que as mesmas apresentam uma abertura transparente que permite o acesso de luz diretamente sobre os elementos de silício.

Existem ainda vários outros tipos de memórias no mercado, como as EEPROMs que podem ser apagadas eletricamente; as memórias de Bolha magnética, que retêm a informação quando sem energia e podem ser escritas como uma RAM quando há energia.

2. FUNÇÃO PEEK

Muitas versões da linguagem BASIC possuem recursos para efetuar operações ligadas diretamente à linguagem de máquina. A função PEEK (espiar em inglês) permite verificarmos o conteúdo de um Byte de memória, no endereço especificado. Sua forma geral é:

PEEK < end >

onde

< end > representa o endereço do Byte a ser lido na memória principal.

Devemos observar que por ser uma função, o valor conseguido pela sua aplicação, deverá ser atribuído a uma variável.

Ex 1:

```
10 A = PEEK (16500)
```

Atribui à variável A o conteúdo da memória 16500 que é um código entre 0 e 255. Para sabermos que caracter ou símbolo este código representa basta fazermos:

```
20 PRINT A, CHR$ (A)
```

Ex. 2:

Podemos "espiar" o conteúdo da memória ROM do microcomputador usando a função PEEK da seguinte forma:

```
10 FOR K = 0 TO 16000
20    PRINT K, PEEK (K), CHR$ (PEEK (K))
90 NEXT K
```

Este programa imprime uma tabela que consta de endereço do Byte (K), contendo decimal (PEEK (K)) e caracter, ou símbolo, que o código representa: CHR$ (PEEK (K)).

3. INSTRUÇÃO POKE

Esta instrução tem por finalidade armazenar um nº entre 0 e 255, que corresponde a um número de um conjunto de 8 bits, em um Byte de memória com o endereço especificado. Sua forma geral é a seguinte:

nl POKE < end >, x

onde:

nl indica o número da linha

< end > indica o endereço de memória

x indica o valor a ser armazenado

A instrução POKE (colocar em inglês) pode armazenar um valor em qualquer lugar da memória RAM, não tendo sentido tentar colocar um valor na área de memória ROM, pois estas são gravadas previamente pelo fabricante e só podem ser lidas e não gravadas.

Ex.: 50 POKE 30000, 0

Esta instrução guarda no endereço 20000 de memória o código 0.

Deve-se tomar muito cuidado ao usar esta instrução, pois muitos microcomputadores usam localidades da memória RAM para armazenar informações do BASIC residente, que também é usada para armazenar programas e dados e, se uma das posições ocupadas for alterada pela gravação de algum valor, causará a perda do programa, obrigando a operação de RESET.

Ex.: 15 POKE 1000,255

Não pode ser executada, pois a posição 1000 é da memória ROM.

Existem várias aplicações para a instrução POKE, como por exemplo, elaborar gráficos preenchendo diretamente todas as posições de memórias reservadas para o mapa de tela. Outra

utilização possível é o armazenamento com economia de posições de memória, de valores entre 0 e 255, usando, neste caso, apenas 1 Byte de memória ao invés de variáveis, que ocupam no mínimo 4 Bytes para armazenar o mesmo valor. Portanto, há uma economia de no mínimo 3 Bytes de memória para cada valor armazenado. O exemplo mais comum para este tipo de utilização é a aquisição de dados de outros aparelhos usando-se uma INTERFACE de ligação entre este aparelho e o micro. Os dados podem ser qualquer grandeza física que possa ser transformada em sinal elétrico, e convertida em um número na faixa entre 0 e 255 como por exemplo, temperatura, pressão, velocidade, vazão, voltagem, amperagem, intensidade luminosa etc.

Estes valores podem ser colocados na memória usando-se a instrução POKE, para utilização futura, como seu processamento para fornecer gráficos, atuação no aparelho fazendo assim um controle de processo utilizando microcomputador. etc.

Ex.: Mil valores de velocidade de rotação de um motor (rpm) são "lidos" por meio de uma INTERFACE, e armazenado temporariamente em uma variável auxiliar V. Cada valor obtido em V é normalizado (colocado na faixa entre 0 e 255) e armazenado em um único Byte de memória a partir do endereço 30.000.

```
10 M = 30000
20 REM ACIONAMENTO DA INTERFACE DE LEITURA
      .
      .
      .
40 REM FINAL DE LEITURA
50 V = ###
60 REM ### INDICA UM NUMERO NA FAIXA ENTRE 0 E 255
70 POKE M, V
80 M = M + 1
90 IF M < = 31000 THEN GOTO 20
100 STOP
```

Os valores armazenados nos Bytes de endereços de 30000 a 31000 podem ser colocados em um gráfico para verificação do comportamento do motor da seguinte forma:

```
200 M = 30000
210 V = PEEK (M)
220 REM AJUSTE PARA IMPRESSORA DE 80 COLUNAS
230 V1 = INT (V/ 3.18)
240 LPRINT TAB (V1) ;"*"
250 M = M + 1
260 IF M < = 31000 THEN GOTO 210
270 STOP
```

CAPÍTULO 14

OPERAÇÕES COM CADEIAS DE CARACTERES

1. FUNÇÃO CHR$

Esta função tem por finalidade transformar o código decimal do argumento da função no caracter ou símbolo correspondente da tabela de caracteres ASCII.

O argumento pode ser um número entre 0 e 255 como também uma variável.

Forma Geral:

CHR$ (v)

Ex.:

a) PRINT CHR$ (42)
Imprime o caracter *

b) 10 FOR I = 0 TO 255
20 PRINT CHR$ (I);
30 NEXT I

Imprime todos os caracteres ASCII disponíveis.

2. FUNÇÃO LEN

Esta função tem por finalidade fornecer o número de caracteres de uma variável alfanumérica, incluindo sinais de pontuação e espaços em branco.

Forma Geral:

LEN (v)

v = variável alfanumérica

Ex.: 10 A$ = "COMPUTACAO"
20 PRINT LEN (A$)
Imprime o número de caracteres da palavra COMPUTACAO que é 10.

3. FUNÇÃO VAL

Esta função serve para ser aplicada em variáveis alfanuméricas cujos caracteres são apenas números, convertendo-os em valores numéricos, ignorando espaços em branco internos, a variáveis se houver.

Forma Geral:

```
VAL (v)
```

v = variável alfanumérica que contém apenas caracteres numéricos.

Ex.: 10 B$ = "1253"
20 X = VAL (B$)
30 PRINT X

Imprime o número 1253

4. FUNÇÃO STR$

Esta função converte uma variável, ou um valor numérico, em uma lista de caracteres (numéricos) que serão atribuídos a uma variável alfanumérica.

Forma Geral:

```
STR$ (v)
```

v = variável numérica

Ex.: 10 X = 31.12
20 A$ = STR$ (X)

```
30 Y = .55
40 B$ = STR$ (Y)
50 PRINT A$ ; B$
```

Resulta como impressão a cadeia de caracteres numéricos 31.12.55.

5. FUNÇÃO MID$

Esta função tem por finalidade separar, ou extrair, o conteúdo de uma variável alfanumérica a partir de um determinado ponto:

Forma Geral:

```
MID$ (v, i, n)
```

v = nome da variável

i = posição inicial de extração

n = número de caracteres a ser extraído

```
Ex.: 10 A$ = "AMORTECEDOR"
     20 E$ = MID$ (A$, 2, 5)
     30 PRINT E$
```

resulta na impressão da subpalavra: MORTE.

6. FUNÇÃO INKEY$ (GET)

Esta instrução tem por finalidade a introdução de um caracter pelo teclado, mas sem o uso da tecla ENTER. Como a execução é muito rápida, em comparação com o tempo que o operador digita um valor, e sem a digitação de uma tecla em tempo hábil, faz com que seja assumido um caracter nulo continuando a execução, por isso, esta instrução fica geralmente dentro de um LOOP até que o usuário digite um caracter chave.

```
Ex.: 10 PRINT "PRECIONE C PARA CONTINUAR"
     20 A$ = INKEY$
     30 IF A$ <> "C" THEN GOTO 20
     40 PRINT "CONTINUANDO. . ."
```

Enquanto não for digitada a tecla C (sem necessidade de ENTER) o programa ficará logo no loop das linhas 20 e 30.

CAPÍTULO 15

INSTRUÇÕES ADICIONAIS DO BASIC

1. GRÁFICOS

Quase todos os microcomputadores pessoais têm a capacidade de gerar gráficos por meio de instruções especiais. Entretanto, estas instruções diferem muito de um microcomputador para outro, fugindo de qualquer padronização. De um modo geral, as instruções gráficas se resumem em fazer acender um ponto em determinadas coordenadas da tela de vídeo. Quanto menor for este ponto, cujo nome técnico é "pixel", maior será a resolução gráfica obtida, ou seja, melhor será a qualidade do gráfico obtido na tela de vídeo.

1.1. Instrução CLS (HOME)

Esta instrução tem a finalidade de "limpar" a tela de vídeo, desativando todos os pontos gráficos e textos existentes, posicionando o cursor no canto superior esquerdo da tela. Sua forma geral é:

nl CLS	ou	*nl* HOME	*

onde *nl* indica o número da linha.

Esta instrução deve ser sempre usada antes de ser iniciada uma atividade gráfica, pois ela prepara a tela para receber um texto ou um gráfico.

Ex.:

O programa:

```
10 FOR I = 1 TO 10
20 PRINT I ;
30 NEXT I
```

---

* Para linha Apple

resulta na impressão dos números:

1 2 3 4 5 6 7 8 9 10

logo após o programa, enquanto que o programa:

```
10 CLS
20 FOR I = 1 TO 10
30 PRINT I ;
40 NEXT I
```

provoca a mesma saída, porém, na primeira linha de uma tela totalmente limpa:

1 2 3 4 5 6 7 8 9 10

1.2. Instrução SET

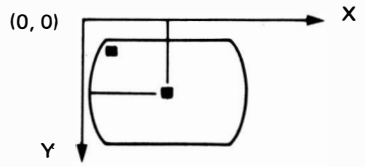
Esta instrução tem a finalidade de ativar na tela de vídeo um ponto gráfico em uma determinada posição.

Sua forma geral é a seguinte:

nl SET (X, Y)

onde *nl* indica o número da linha
X indica a ordenada da coluna
Y indica a abscissa da linha

Normalmente, a origem das coordenadas X e Y, isto é, o ponto (0, 0) é situado no canto superior esquerdo da tela de vídeo, conforme a figura:



O número de pontos possíveis varia muito de um microcomputador para outro. Por exemplo, os micros que seguem a lógica do TRS-80 dividem a tela no modo gráfico em 6144 pontos compostos por 128 pontos horizontais ($0 < X < 127$) e 48 pontos verticais ($0 < Y < 47$); os micros que seguem a lógica SINCLAIR dividem a tela no modo gráfico em 2916 pontos compostos por 64 pontos horizontais ($0 < X < 63$) e 44 pontos verticais ($0 < Y < 43$).

Portanto, no modo gráfico, a tela do micro é dividida em reticulado, no qual cada ponto pode ser acessado independentemente.

Para efeito de exemplos, supomos a tela dividida em 128 colunas por 48 linhas com a origem colocada no canto superior esquerdo.

Ex.:

1. O programa:

```
10 CLS
20 SET (64, 24)
```

ativa o ponto central da tela de vídeo.

2. O programa:

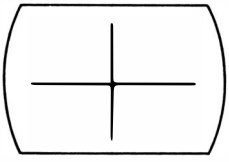
```
10 CLS
20 FOR X = 1 TO 127
30 SET (X, 1)
40 NEXT X
```

faz com que sejam ativados todos os pontos da linha 1, ou seja, todas as colunas de 1 a 127, gerando, portanto, uma "linha horizontal" na primeira linha da tela do micro.

3. O programa:

```
10 CLS
20 FOR I = 0 TO 127
30    SET (I, 21)
40 NEXT I
50 FOR J = 1 TO 42
60    SET (64, J)
70 NEXT J
```

gera os eixos X e Y com cruzamento no centro da tela, conforme a figura:



Convém ressaltar que em alguns micros não existe a instrução SET mas, sim a instruçãc PLOT, para "plotar" um ponto em uma posição determinada. Estas instruções são equivalentes devendo utilizar uma ou outra conforme o microcomputador utilizado. Entretanto, a instruçãc PLOT não exige parênteses para o par de coordenadas, conforme mostra a seguinte forma geral:

```
nl PLOT X, Y
```

onde *nl*, X e Y têm o mesmo significado da instrução SET.

1.3. Instrução RESET

Esta instrução tem a finalidade de desativar da tela de vídeo um ponto gráfico que estava ativado em uma determinada posição.

Sua forma geral é a seguinte:

```
nl RESET (X, Y)
```

onde *nl* indica o número da linha
X indica a abscissa da coluna
Y indica a ordenada da linha

Esta instrução tem além dos mesmos parâmetros X e Y que a instrução SET, os mesmos limites, porém, funcionando de maneira oposta.

Exs.:

1. A instrução

```
30 RESET (64, 24)
```

desativa o ponto localizado no centro da tela de vídeo.

2. O programa

```
50 FOR X = 127 TO 1 STEP – 1
60     RESET (X, 1)
70 NEXT X
```

se colocado após o segundo programa exemplo da instrução SET, faz com que sejam desativados todos os pontos da linha 1.

Analogamente à relação que fizemos com respeito às instruções SET e PLOT, existe uma instrução equivalente ao RESET em alguns micros, que é a instrução UNPLOT, cuja forma geral é:

```
nl UNPLOT X, Y
```

onde *nl*, X e Y têm o mesmo significado da instrução RESET.

1.4. Instrução POINT

Esta instrução tem por finalidade verificar se um determinado ponto da tela de vídeo está ativado ou não.

Possui a seguinte forma geral:

nl POINT (X, Y)

onde nl indica o número da linha
X indica ordenada da coluna
Y indica a abscissa da linha

Se o ponto estiver ativado, isto é, foi utilizada a instrução SET, a instrução POINT fornecerá o código − 1, e caso o ponto não esteja ativado, fornecerá o código 0.

```
Ex.: 10 CLS
     20 SET (64, 24)
     30 IF  POINT (64, 24) = − 1 THEN GOTO 60
40   40 PRINT "PONTO 64, 24 ESTA' APAGADO"
     50 STOP
     60 PRINT "PONTO 64, 24 ESTA' ACESO"
     70 STOP
```

Este programa imprime a mensagem:

"PONTO 64, 24 ESTA ACESO"

pois foi ativado na instrução 20.

Se trocarmos esta instrução por:

20 RESET (64, 24)
a mensagem será:

"PONTO 64, 24 ESTA APAGADO"

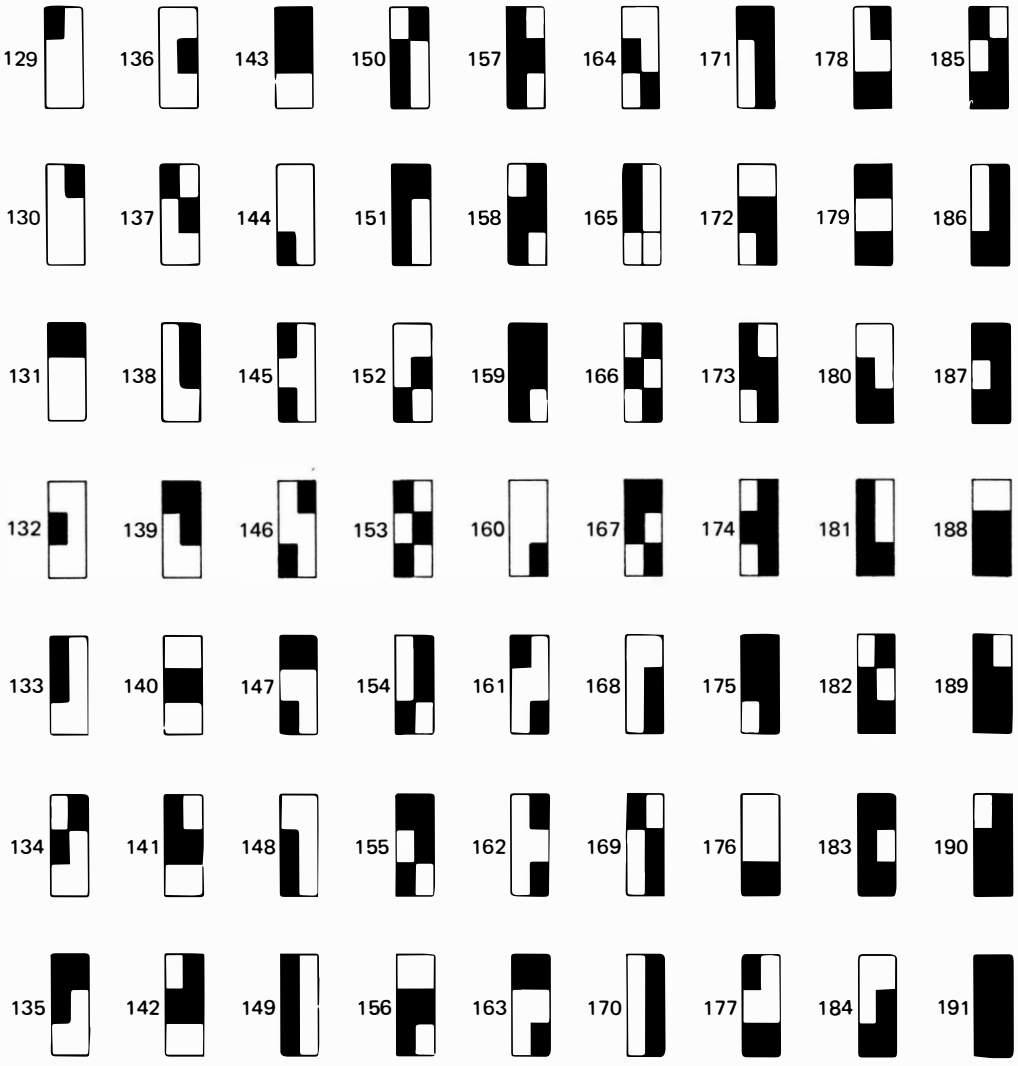
1.5. Caracteres Gráficos

Estes caracteres são utilizados na formação de desenhos, figuras, gráficos etc. São tratados como caracteres normais e podem ser armazenados em variáveis alfanuméricas, como qualquer outro caracter.

O conjunto de caracteres gráficos varia muito de um computador para outro, portanto abordaremos neste texto apenas os caracteres gráficos dos micros que seguem a lógica do TRS80 (tipo CP 300, CP 500), e SINCLAIR, (como TK85, CP 200), e de forma especial os gráficos p/ micros da linha Apple.

Os caracteres gráficos do CP 500 podem ser obtidos por meio da função CHR$ (I), onde I é um código da lista ASCII (conforme apêndice I) e pertencem à faixa 128 a 191, onde o caracter 128 fornece um espaço em branco.

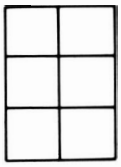
Estes caracteres gráficos podem ser vistos na figura abaixo:



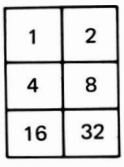
Para obtermos estes caracteres na tela do microcomputador basta rodar o seguinte programa:

```
10 CLS
20 J = 0
30 FOR I = 128 TO 191
40 J = J + 1
50 IF J < = 16 GOTO 80
60 J = 0
70 PRINT
80 A$ = CHR$ (I)
90 PRINT A$ ;"   " ;
100 NEXT I
110 PRINT
120 PRINT
130 STOP
```

Podemos observar que cada caracter gráfico é composto por 6 "pixeis" obtidos com a instrução SET, dispostos em um retângulo que possui 3 linhas e duas colunas conforme a figura:

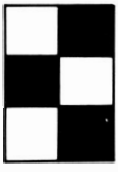


Cada um dos pequenos retângulos é ativado dependendo do código do caracter gráfico. A maneira mais inteligente de usar os caracteres gráficos para compor uma figura não é simplesmente olhar na tabela, encontrar o caracter desejado e o seu respectivo código. Existe uma "lei de formação" para encontrarmos o código do caracter desejado, onde a cada um dos 6 pequenos retângulos, componentes do caracter gráfico, é dado um valor, conforme a figura:



Para encontrarmos o código do caracter gráfico desejado, fazemos a somatória dos pesos de cada pixel que desejarmos ativar, somando o resultado final ao valor 128 (que corresponde aos seis pixeis desativados).

Por exemplo, se desejarmos obter o código ASCII do caracter



fazemos a soma dos pesos dos pixeis ativados (achurados):

$$2 + 4 + 32 + 128 = 166$$

Como observação final, devemos ressaltar o fato de que um programa para desenhar uma figura feita com caracteres gráficos é seis vezes mais rápido que feito com a instrução SET, pois o caracter gráfico desenha 6 pixeis com uma única instrução, ao passo que precisaríamos de 6 instruções SET para desenharmos a mesma figura.

Os microcomputadores que seguem a lógica SINCLAIR, apresentam recursos gráficos que podem ser obtidos de maneira direta, ou seja, basta acionar apenas uma tecla, para obter-se o caracter gráfico desejado.

Estes microcomputadores apresentam um cursor que informa em que posição da tela nos encontramos, bem como o modo de operação que está para ser executado.

Existem 4 modos de operação a saber:

[K] para introdução de número de linha e palavras-chave de instruções.

[L] para introdução de letras ou números.

[F] para introdução de palavras-chave de funções.

[G] para operação no modo gráfico.

Para deixarmos o cursor no modo gráfico, é necessário que precionemos simultaneamente as teclas [SHIFT] e [GRAPHICS] (tecla nº 9).

Qualquer letra precionada neste modo aparecerá no modo inverso ao da tela normal, ou seja, se a tela normalmente apresenta caracteres pretos sobre um fundo branco, no modo gráfico aparecerá o caracter branco sobre um contorno da letra em preto e vice-versa.

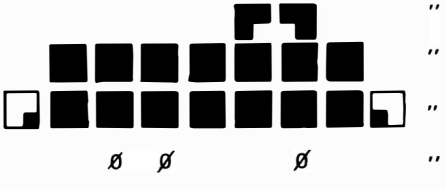
Os caracteres gráficos destes microcomputadores estão no modo superior das letras normais, e para colocá-los na tela preciona-se [SHIFT] juntamente com a letra correspondente ao caracter gráfico.

O conjunto de caracteres gráficos disponíveis são os seguintes:

Símbolo	Código	Como obter	Símbolo	Código	Como obter
	0	K ou L SPACE		128	G SPACE
	1	G SHIFT1		129	G SHIFTQ
	2	G SHIFT2		130	G SHIFTW
	3	G SHIFT7		131	G SHIFT6
	4	G SHIFT4		132	G SHIFTR
	5	G SHIFT5		133	G SHIFT8
	6	G SHIFTT		134	G SHIFTY
	7	G SHIFTE		135	G SHIFT3
	8	G SHIFTA		136	G SHIFTH
	9	G SHIFTD		137	G SHIFTG
	10	G SHIFTS		138	G SHIFTF

EXEMPLO:

```
10 FOR L = 0 TO 24
30 PRINT AT 10, L ;"
40 PRINT AT 11, L ;"
50 PRINT AT 12, L ;"
60 PRINT AT 13, L ;"
70 NEXT M
```



Desenvolvendo suficientemente a habilidade de trabalhar com os recursos gráficos do microcomputador, podemos criar uma quantidade imaginável de jogos que além de divertir farão com que seja conseguido um maior domínio sobre a linguagem BASIC, bem como sobre o microcomputador.

2. GRÁFICOS PARA MICROS DA LINHA APPLE II

Trataremos nesta seção apenas dos recursos gráficos dos microcomputadores compatíveis com a linha Apple II, que a nosso ver apresenta a melhor saída gráfica dos pequenos computadores.

Estes micros possuem três modos de exibição de dados no terminal de vídeo que são:

- modo gráfico de baixa resolução
- modo gráfico de alta resolução
- modo texto

Cada um destes modos será agora detalhado separadamente, nos subitens que seguem.

2.1. Modo Gráfico de Baixa Resolução

Neste modo a tela do terminal de vídeo fica dividida em 40 colunas por 48 linhas que perfazem um total de 1920 coordenadas.

2.1.1. Instrução GR

A instrução GR é a responsável pela conversão da tela para o modo gráfico de baixa resolução. A forma geral desta instrução é:

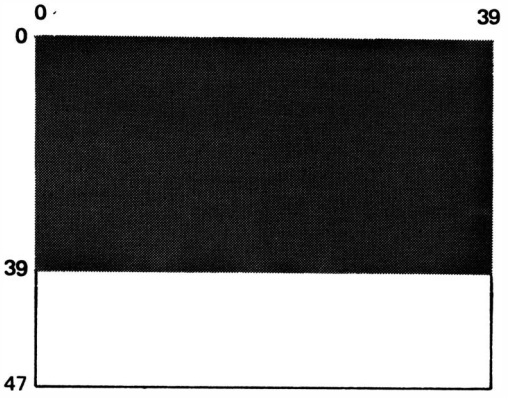
nl GR

onde *nl* indica o número da linha do programa.

Após a execução desta instrução a tela de modo gráfico de baixa resolução é limpa e um espaço de 40 linhas por 40 colunas estará pronto para mostrar os gráficos deste modo.

Deve-se notar que o espaço de linhas não foi de 48 e sim 40 linhas. Isto se deve ao fato de serem reservadas as quatro últimas linhas da tela para textos. Esta área da tela é chamada de *janela de texto*, e pode ser vista na figura a seguir:

Existem situações onde se deseja que a tela toda esteja em modo gráfico de baixa resolução, ou seja, a janela de texto não é desejada.



Pode-se conseguir deixar a tela toda no modo gráfico através do comando

POKE – 16302, 0 ou POKE – 49234, 0

Este comando, quando executado, substitui a área de texto por área de gráfico.

Este comando deve ser executado em conjunto com a instrução que limpa todas as 48 linhas do modo gráfico que é uma chamada a uma subrotina interna do Basic que se inicia na posição – 1998 que é a seguinte:

CALL – 1998 ou CALL 63538

Pode-se fazer reaparecer a janela de texto a partir de uma tela que está no modo gráfico total através de um comando parecido, que é

POKE – 16301,0 ou POKE 49235,0

2.1.2. Instrução COLOR

Esta instrução tem a finalidade de especificar qual será usada no modo gráfico de baixa resolução.

Sua forma geral é a seguinte:

nl COLOR = n

onde nl indica o número da linha e n é um número que indica o código da cor a ser utilizada.

Existem 16 cores disponíveis, conforme indica a tabela a seguir, e são identificadas pelos números. Códigos de 0 a 15.

N	COR
0	Preta
1	Magenta/Vermelha
2	Azul Escuro
3	Violeta/Roxo
4	Verde Escuro
5	Cinza
6	Azul
7	Azul Claro

N	COR
8	Marrom
9	Laranja
10	Cinza
11	Rosa
12	Verde
13	Amarela
14	Azul Piscina
15	Branco

Convém observarmos que se não for especificada nenhuma cor quando da ativação do modo gráfico de baixa resolução, é automaticamente adotada a cor zero (preta), e, portanto, estando a tela com fundo preto, não se perceberá o resultado de impressão dos gráficos.

Ex.: 10 GR
20 COLOR = 15

Muda para modo gráfico de baixa resolução e seleciona a cor branca (15) para a plotagem dos gráficos.

2.1.3. Instrução PLOT

Esta instrução tem a finalidade de colocar na tela de vídeo um ponto gráfico em uma determinada posição da tela.

Sua forma geral é a seguinte:

nl PLOT c, l

onde nl indica o número da linha do programa onde a instrução aparece
l indica a ordenada da linha e
c indica a abscissa da coluna.

Conforme a figura anterior podemos notar que a origem das coordenadas c e l, isto é, o ponto (0, 0) está situado no canto superior esquerdo da tela de vídeo.

Ex. 1: 40 GR
50 COLOR = 15
60 PLOT 20, 20

Este trecho de programa coloca um ponto branco na linha 20 e coluna 20 deixando uma janela de texto de 4 linhas na parte inferior da tela.

Ex. 2:

```
10  GR
20  POKE 49234,0
30  CALL 63538
40  COLOR = 15
50  PLOT 0,0
60  PLOT 39,0
70  PLOT 39,47
80  PLOT 0,47
90  PLOT 20,24
100   GET A$
110   END
```

Neste programa a linha 10 muda a tela para o modo gráfico de baixa resolução; a linha 20 deixa toda a tela em modo gráfico, eliminando a janela de texto; a linha 30 limpa toda a tela gráfica; a linha 40 seleciona a cor branca e as linhas de 50 a 90 colocam os pontos nas coordenadas indicadas.

2.1.4. Instrução HLIN

Para traçarmos uma linha horizontal na tela de vídeo podemos executar a seqüência de comandos no modo gráfico de baixa resolução que segue:

```
30  GR
40  COLOR = 15
50  FOR I = 0 TO 39
60  PLOT I,20
70  NEXT I
```

Este programa traça uma linha horizontal na tela de micro da coluna 0 até a coluna 39, na linha 20.

Existe uma forma de simplificar o traçado de linhas horizontais que é a instrução HLIN.

A forma geral desta instrução é:

n*l* HLIN c1, c2 AT *l*

onde: n*l* é o número da linha da instrução
c1 é o número da coluna onde se inicia a linha que será traçada
c2 é o número da coluna onde termina o traçado da linha
l é o número da linha da tela onde será desenhada a linha horizontal desejada.

OBS.: Os valores de c1, c2 e *l* devem respeitar os limites da tela do gráfico de baixa resolução, ou seja:

$$0 \leqslant c1 \leqslant 39, \quad 0 \leqslant c2 \leqslant 39$$
$$0 \leqslant l \leqslant 47$$

Ex.: Para traçarmos a mesma linha do exemplo anterior, podemos fazer:

```
30 GR
40 COLOR = 15
50 HLIN 0,39 AT 20
```

o que simplifica a programação.

2.1.5. Instrução VLIN

Para traçarmos uma linha vertical, de modo semelhante às linhas horizontais, podemos executar a seqüência de comandos no modo gráfico de baixa resolução, que segue:

```
30 GR
40 COLOR = 15
50 FOR I = 0 TO 39
60 PLOT 20, I
70 NEXT I
```

Este programa traça uma linha vertical na tela do micro da linha 0 até a linha 39, na coluna 20.

Existe, analogamente, uma forma de simplificar o traçado de linhas verticais, que é a instrução VLIN.

A forma geral desta instrução é:

nl VLIN *l1*, *l2* AT C

onde: *nl* é o número da linha da instrução
l1 é o número da linha onde se inicia o traçado da linha a ser desenhada
l2 é o número da linha onde termina o traçado da linha a ser desenhada
C é o número da coluna onde será desenhada a linha vertical desejada.

OBS.: Os valores de *l1*, *l2* e C devem respeitar os limites da tela do gráfico de baixa resolução, ou seja:

$$0 \leqslant l1 \leqslant 47, \quad 0 \leqslant l2 \leqslant 47 \quad \text{e} \quad 0 \leqslant C \leqslant 39$$

Ex.: Para traçarmos a mesma linha vertical do exemplo anterior, podemos fazer:

```
30 GR
40 COLOR = 15
50 VLIN 0,39 AT 20
```

o que também simplifica a programação.

2.2. Modo Gráfico de Alta Resolução

Este sem dúvida é o modo gráfico mais utilizado pelos programadores, devido a grande qualidade dos gráficos gerados.

Neste modo, a tela do terminal de vídeo fica dividida em 280 colunas por 192 linhas que perfazem um total de 53760 pontos que podem ser acessados independentemente.

2.2.1. Instrução HGR

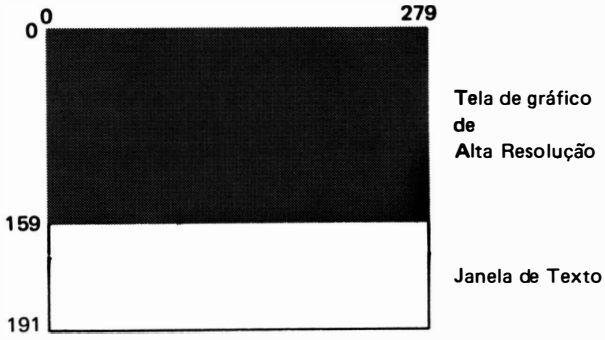
Esta instrução converte a tela do microcomputador para o modo gráfico de alta resolução.

A forma geral desta instrução é:

nl HGR

onde *nl* é o número da linha do programa.

Esta instrução, quando executada, limpa a tela e entra no modo gráfico de alta resolução deixando acessível um espaço de 280 colunas por 160 linhas, deixando as últimas 32 linhas da parte de baixo da tela para a janela de texto. A representação da tela gráfica de alta resolução é mostrada na figura a seguir:



De forma análoga ao modo gráfico de baixa resolução pode-se suprimir a janela de texto, deixando toda a tela no modo gráfico de alta resolução através do comando

POKE – 16302,0 ou POKE 49234,0

Podendo-se também reativar a área de texto com o comando

POKE – 16301,0 ou POKE 49233,0

2.2.2. Instrução HCOLOR

Esta instrução tem a finalidade de especificar qual será a cor adotada no modo gráfico de alta resolução para o desenho das próximas figuras, após o comando ser executado.

Sua forma geral é a seguinte:

nl HCOLOR = n

onde: nl indica o número da linha
n é um número que indica o código da cor a ser utilizada

Existem neste modo gráfico uma menor quantidade de cores disponíveis, e podem ser identificadas pelos número-códigos de 0 a 7, conforme a tabela abaixo:

N	Cor
0	Preta
1	Verde
2	Roxo/Violeta
3	Branca
4	Preta
5	Laranja
6	Azul
7	Branca

Quando o modo gráfico de alta resolução é ativado, a cor adotada é a preta (0) e se não for selecionada uma nova cor, não se perceberá a execução do resultado da impressão do gráfico, uma vez que o fundo da tela também é preto.

Ex.: 20 HGR
30 HCOLOR = 7

Muda para o modo gráfico de alta resolução e seleciona a cor branca (7) para a plotagem dos gráficos.

2.2.3. Instrução HPLOT

Esta instrução tem a finalidade de colocar na tela de vídeo um ponto gráfico em uma determinada posição da tela.

A forma geral desta instrução é a seguinte:

n*l* HPLOT c, *l*

onde: n*l* indica o número da linha do programa onde a instrução aparece
l indica a ordenada da linha e
c indica a abscissa da coluna

A origem das coordenadas c e *l*, isto é, o ponto (0,0) está situado no canto superior esquerdo da tela de vídeo.

Ex. 1:

```
100 HGR
110 HCOLOR = 7
120 HPLOT 140,96
```

Este programa coloca um ponto branco na coluna 140 e na linha 96 da tela de vídeo (centro da tela), deixando uma janela de texto de 4 linhas na parte inferior da tela.

Ex. 2:

```
10  HGR
20  POKE 49234,0
30  HCOLOR = 7
40  HPLOT 0,0
50  HPLOT 279,0
60  HPLOT 279,191
70  HPLOT 0,191
80  HPLOT 140,96
90  GET A$
100  END
```

Este programa coloca 4 pontos de alta resolução nos 4 extremos da tela gráfica total, sem a janela de texto que é eliminada pela linha 20.

2.2.4. Traçando Retas no Gráfico de Alta Resolução

É bem mais fácil e versátil traçar retas em gráficos de alta resolução que no modo de baixa resolução. Basta indicarmos na instrução HPLOT o ponto de origem até (TO) o ponto destino,

e estes dois pontos serão unidos por uma reta, que não precisa ser só vertical ou só horizontal, como no caso de baixa resolução.

A forma geral desse comando é a seguinte:

nl HPLOT c1, l1 TO c2, l2

onde: nl é o número da linha do programa onde aparece esta instrução
c1 e l1 coordenadas do ponto de origem
c2 e l2 coordenadas do ponto destino

Ex.: 10 HGR : HCOLOR = 7
20 HPLOT 0,90 TO 279,90

Desenha uma linha horizontal da coluna 0 até a coluna 279 na linha 30.

Pode-se encadear vários pontos em uma mesma instrução, o que faz aparecer na tela o desenho de várias linhas de uma só vez.

Ex.: 10 HGR : HCOLOR = 7
20 HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159
TO 0,0

Desenha uma linha ao redor da tela gráfica deixando a janela de texto.

Outra forma muito utilizada do uso da instrução HPLOT é a seguinte:

nl HPLOT TO c, l

onde: nl indica o número da linha onde esta instrução aparece
c e l são as coordenadas de um ponto da tela gráfica de alta resolução.

Nesta forma, o ponto adotado como ponto inicial para o traçamento da reta é o último ponto marcado na tela, e com a mesma cor deste último ponto.

OBS.: A cor deste novo segmento de reta não é alterada com a colocação de um comando HCOLOR, prevalecendo realmente a cor de plotagem do último ponto marcado.

2.3. Modo Texto

Este é o modo normal de edição de programas. Neste modo, a tela dos micros compatíveis com o Apple II ficam divididas em 40 colunas por 24 linhas que podem exibir um total de 960 caracteres na tela.

2.3.1. Instrução TEXT

Esta instrução retorna para o modo texto a partir de qualquer um dos modos gráficos.

A forma geral desta instrução é:

nl TEXT

onde: *nl* é o número da linha.

Quando esta instrução é executada, o prompt, que é o caracter indicativo de modo de comando (]) juntamente com o cursor são colocados na última linha da tela. Esta instrução não apaga a tela e como o modo texto utiliza a mesma área de memória do modo gráfico de baixa resolução, invariavelmente a tela fica cheia de caracteres estranhos, o que é um fato normal.

EXERCÍCIO

- Elaborar programa BASIC que receba um conjunto de 10 pontos (x, y) através do teclado e coloque-os em um gráfico cartesiano, unindo os pontos entre si.

  Sabe-se que os valores de x e y são positivos e menores que 10.

  Colocar o eixo x na linha 150 e o eixo y na coluna 20.

  Colocar o ponto $x = 10$, $y = 0$ na linha 150, e coluna 160 e o ponto $x = 0$, $y = 10$ na linha 10, coluna 20.

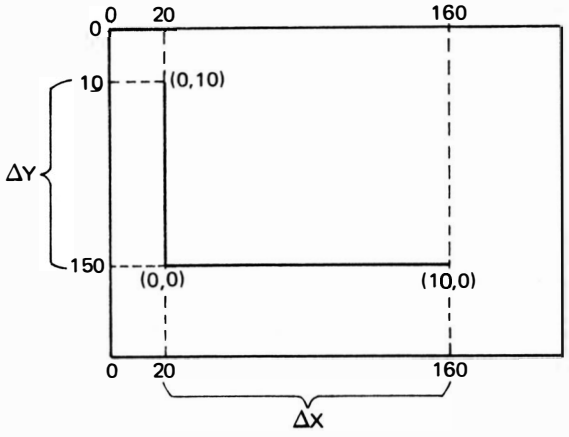
Algoritmo

1º) entrada dos pontos nos vetores X (I) e Y (I) (verifica faixa opcional)
2º) ordenar os pontos com relação ao eixo x
3º) modularizar os pontos
4º) plotar eixos
5º) plotar pontos
6º) voltar para o modo texto depois de uma pausa

Modularização dos Pontos

Devemos transformar as coordenadas do ponto P (x, y) para um sistema de eixos onde a origem do sistema, ou seja, as coordenadas do ponto P (0, 0) está situada na interseção das linhas 150 com a coluna 20.

Esquematicamente temos:



Verificamos então que:

- todos os pontos *cy* $y = 0$ estão na linha 150
- todos os pontos *cy* $x = 0$ estão na coluna 20

∴ as coordenadas modularizadas serão:

$$l = y\,(0) - \text{INT}\left(\frac{\Delta y * y}{\text{NDv}}\right)$$

e

$$c * \text{X}\,(0) + \text{INT}\left(\frac{\Delta x * x}{\text{NDh}}\right)$$

onde: $\Delta x = X_{máx} - X_{mín} = 160 - 20 = 140$

$\Delta y = Y_{máx} - Y_{mín} = 150 - 10 = 140$

NDv = número de divisões verticais
NDh = número de divisões horizontais

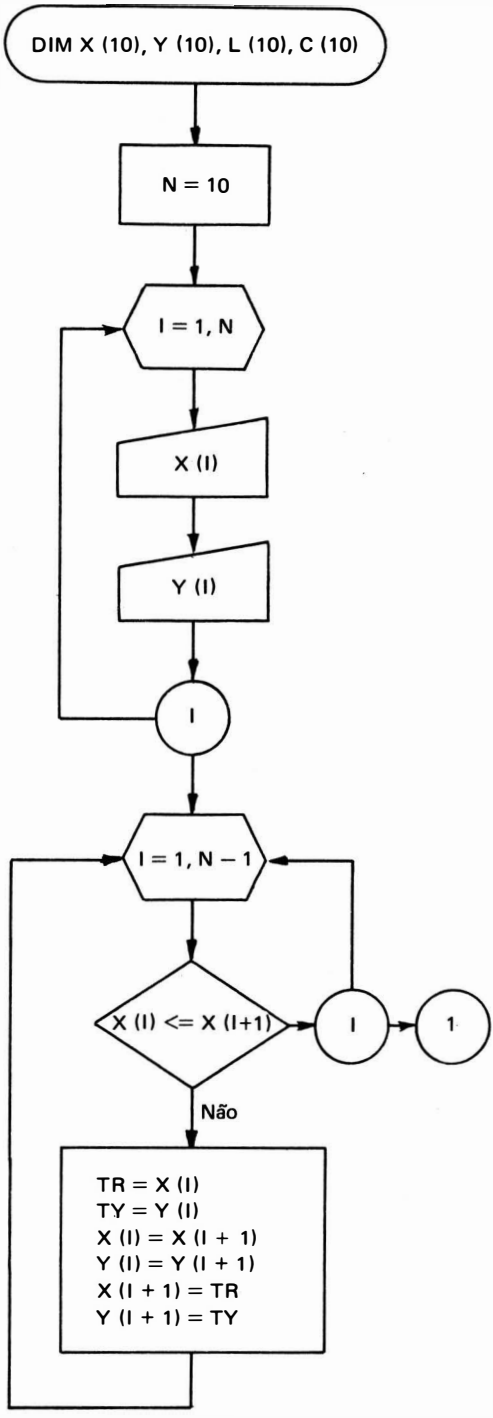
Como o maior valor de x e de y vale 10, adotaremos NDv = NDh = 10, y (0) e x (0) são as coordenadas o ponto de origem, que no nosso caso valem

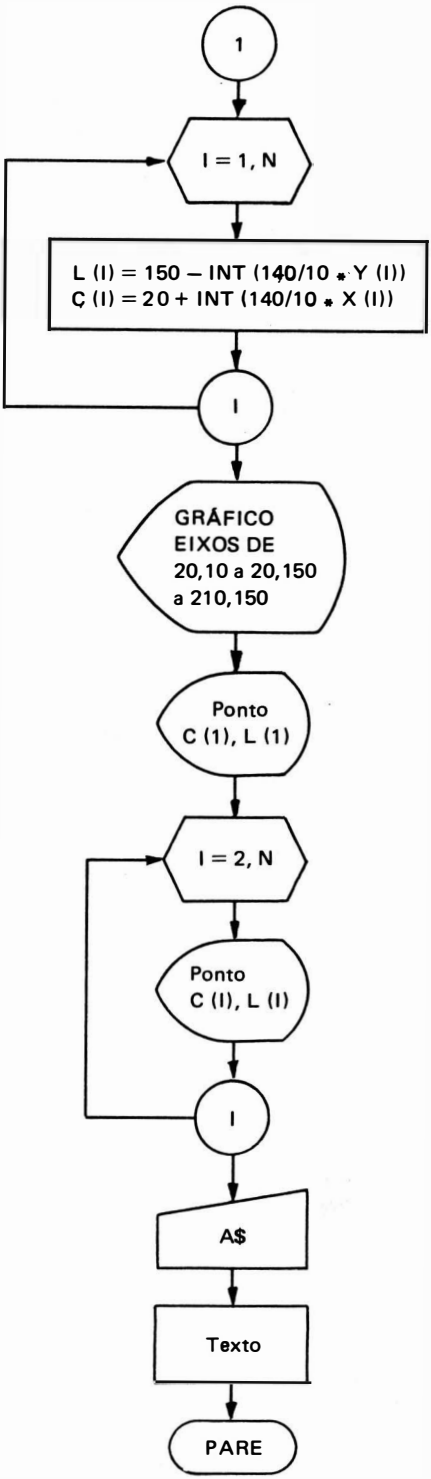
$$y\,(0) = 150$$
$$x\,(0) = 20$$

∴ o resultado da modularização para o problema proposto será:

```
L (I) = 150 – INT (140/10 * Y (I))
C (I) = 20 + INT (140/10 * X (I))
```

FLUXOGRAMA:


DIM X (10), Y (10), L (10), C (10)
N = 10
I = 1, N
X (I)
Y (I)
I
I = 1, N – 1
X (I) <= X (I+1)
I
1
Não
TR = X (I)
TY = Y (I)
X (I) = X (I + 1)
Y (I) = Y (I + 1)
X (I + 1) = TR
Y (I + 1) = TY


1
I = 1, N
L (I) = 150 – INT (140/10 * Y (I))
C (I) = 20 + INT (140/10 * X (I))
I
GRÁFICO
EIXOS DE
20,10 a 20,150
a 210,150
Ponto
C (1), L (1)
I = 2, N
Ponto
C (I), L (I)
I
A$
Texto
PARE

PROGRAMA

```
10  DIM X (10), Y (10), L (10), C (10)
20  REM ***** ENTRADA DE DADOS *****
30 N = 10
40  HOME
50  FOR I = 1 TO N
60  PRINT "ENTRE COM X ("; I;") = ";
70  INPUT "  "; X (I)
80  PRINT "ENTRE COM Y ("; I;") = ";
90  INPUT "  "; Y (I)
100  PRINT
110  NEXT I
120  REM  ORDENACAO DOS PONTOS
130  HOME : PRINT "ORDENANDO"
140  FOR I = 1 TO N – 1
150  IF X (I) < = X (I + 1) THEN GOTO 230
160 TR = X (I)
170 TY = Y (I)
180 X (I) = X (I + 1)
190 Y (I) = Y (I + 1)
200 X (I + 1) = TR
210 Y (I + 1) = TY
220  GOTO 140
230  NEXT I
240  HOME  :  PRINT "MODULARIZANDO"
250  FOR I = 1 TO N
260 L (I) = 150 – INT (140 / 10 * Y (I))
270 C (I) = 20 + INT (140 / 10 * X (I))
280  NEXT I
290  REM PLOTANDO EIXOS
300  HGR
310  HCOLOR = 7
320  HPLOT 20,10 TO 20,150 TO 210,150
330  REM PLOTA PONTOS
340  HCOLOR = 5: REM LARANJA
350  HPLOT C (1), L (1)
360  FOR I = 2 TO N
370  HPLOT TO C (I), L (I)
380  NEXT I
390  GET A$
400  TEXT
410  HOME
```

3. NÚMEROS ALEATÓRIOS

Números aleatórios são aqueles obtidos através do acaso e comandados pelas leis da probabilidade, ou seja, um número cujo valor é imprevisível de ser calculado antes deste ser gerado.

Estes números são utilizados para cálculos estatísticos, simulações de problemas técnicos e científicos e principalmente para jogos utilizando o microcomputador.

3.1. Função RND

Esta função tem a finalidade de gerar um número aleatório, cujo nome RND é a forma abreviada da palavra RANDOM, que em inglês significa casual ou aleatório.

Sua forma geral é a seguinte:

RND (exp)

onde exp indica uma expressão BASIC.

a) se exp for igual a zero, a função RND (Ø) fornece um número aleatório entre 0 e 1.

Dependendo do microcomputador estes limites podem variar, como por exemplo nos micros:

SINCLAIR: $0 < X < 1$
TRS80: $0.000001 < X < 0.999999$ inclusive.

b) Se exp for uma expressão BASIC, o número aleatório será fornecido na faixa entre 1 e INT (exp) dentro dos limites de $1 <$ exp < 32768.

A função RND não fornece um número verdadeiramente randômico ou aleatório, pois é gerado a partir de uma seqüência de 65536 valores misturados de forma a parecer aleatório. Por este motivo, a função RND normalmente é descrita como um gerador de números pseudo-aleatórios.

Ex.:

1. 10 A = RND (0)
Armazena na variável A um número aleatório.

2. 10 X = RND (INT 75 * A + 1) − 1/65536)
Armazena na variável X um número aleatório entre 1 e a parte inteira da expressão entre parênteses.

3.
```
50 A = INT (RND (Ø) * 6) + 1
60 PRINT A
```

Imprime um número aleatório entre 1 e 6.

3.2. Instrução RANDOM, RANDOMIZE ou RAND

Esta instrução tem a finalidade de deixar a função RND verdadeiramente aleatória. Para a geração dos números pseudo-aleatórios da função RND existe um procedimento computacional fixo, que gera sempre a mesma seqüência de números que parecem aleatórios, a partir de um número incial também chamado "semente" ou "raiz" da função aleatória.

A função RANDOM varia a "semente" ou "raiz" a partir da qual é gerada a seqüência pseudo-aleatória da função RND.

Sua forma geral pode ser uma das seguintes, dependendo do microcomputador:

nl RAND	ou	*nl* RANDOM	ou	*nl* RANDOMIZE

onde *nl* indica o número da linha.

Ex.:
```
10 RANDOM
20 A = RND (0)
30 PRINT A ;
40 GOTO 10
```

Imprime uma seqüência de números verdadeiramente aleatórios.

Convém observarmos que enquanto RANDOM é uma instrução BASIC, RND é uma função, não podendo, portanto, ser confundidas ou trocadas.

A instrução RANDOM utiliza-se de variáveis do sistema para gerar o próximo valor de RND, como por exemplo, contar quantos caracteres foram impressos na tela do terminal de vídeo, e utilizar este valor, que é aleatório, como "semente" da função RND.

4. DESVIOS MÚLTIPLOS PROGRAMADOS

Existem problemas em computação onde, na elaboração de um programa, há a necessidade lógica que o fluxo do processamento seja dirigido para uma dentre várias direções possíveis. Para resolver este problema existem as instruções de desvios múltiplos programados.

4.1. Instrução ON – GOTO

Esta instrução tem a finalidade de desviar o fluxo do processamento para uma das linhas especificadas logo após GOTO. A forma geral desta instrução é a seguinte:

```
nl ON v GOTO nl1, nl2, nl3 . . .
```

onde nl indica o número da linha da instrução.

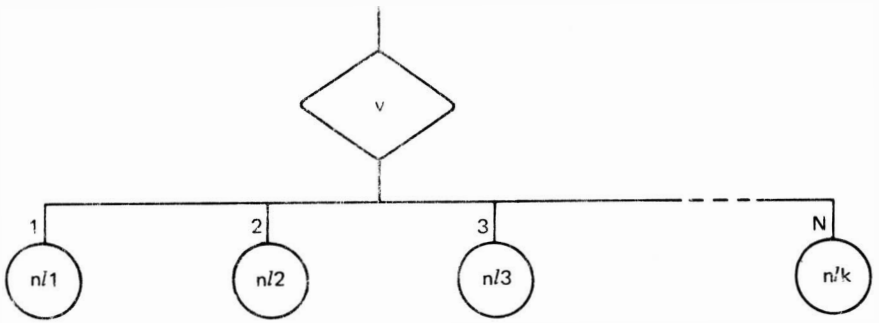
v = variável BASIC
nl_1, nl_2 . . . indicam números de linhas para as quais poderão ser desviados os fluxos de processamento.

Quando o computador encontra esta instrução, o valor da variável v é testado. Se v = 1, o fluxo de processamento é desviado para a instrução cujo número de linha é nl_1; se for igual a 2, desvia para nl_2 e assim por diante.

Ex.: 50 ON X GOTO 100, 10, 90, 500

se X for igual a 1, desvia para a linha 100;
se X for igual a 2, desvia para a linha 10;
se X for igual a 3, desvia para a linha 90;
se X for igual a 4, desvia para a linha 500.

A representação desta instrução no fluxograma é a seguinte:



Na maioria das versões BASIC em que esta instrução existe, a variável v pode ser substituída por uma expressão qualquer, desde que o resultado seja um número inteiro entre 1 e o número máximo de linhas de desvios escritas após o GOTO. Assim, no exemplo abaixo:

150 ON A * 3 – J GOTO 100, 200, 300, 400

o resultado da expressão entre o ON e o GOTO tem que estar necessariamente na faixa entre 1 e 4.

EXEMPLO DE UTILIZAÇÃO:

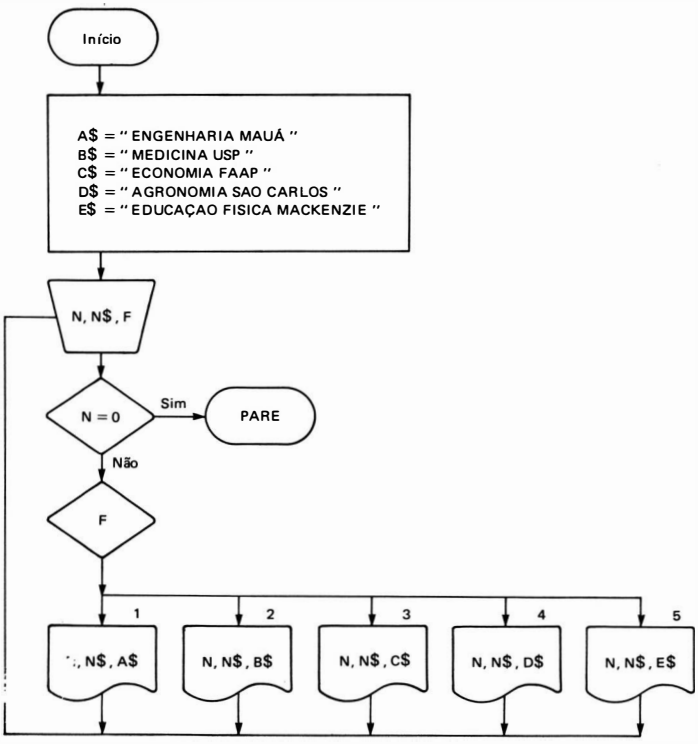
Os dados referentes aos candidatos de um vestibular unificado, no qual participam 5 faculdades foram codificados em uma lista DATA onde consta: nº de inscrição (N), nome do candidato (N1) e código da faculdade de interesse do candidato (F). Cada candidato pode escolher apenas uma das 5 faculdades, pois estas são de áreas totalmente diferentes entre si.

Imprimir um relatório que forneça o nome do candidato, nº de inscrição e o nome da escola na qual está inscrito, sabendo-se que:

Código (F)	Nome da Faculdade
1	Engenharia Mauá
2	Medicina USP
3	Economia FAAP
4	Agronomia São Carlos
5	Educação Física Mackenzie

Considerar que o último candidato tem nº de inscrição zero.

FLUXOGRAMA



```
10 A$ = "ENGENHARIA MAUA ' "
20 B$ = "MEDICINA USP"
30 C$ = "ECONOMIA FAAP"
40 D$ = "AGRONOMIA SAO CARLOS"
50 E$ = "EDUCACAO FISICA MACKENZIE"
60 READ N, N$, F
70 IF N = 0 THEN STOP
80 ON F GOTO 90, 110, 130, 150, 170
90 LPRINT TAB (2) ; N ; TAB (10) ; N$ ; TAB (50) ; A$
100 GOTO 60
110 LPRINT TAB (2) ; N ; TAB (10) ; N$ ; TAB (50) ; B$
120 GOTO 60
130 LPRINT TAB (2) ; N ; TAB (10) ; N$ ; TAB (50) ; C$
140 GOTO 60
150 LPRINT TAB (2) ; N ; TAB (10) ; N$ ; TAB (50) ; D$
160 GOTO 60
170 LPRINT TAB (2) ; N ; TAB (10) ; N$ ; TAB (50) ; E$
180 GOTO 60
200 DATA 0001, "ANTONIO CARLOS" ; 1
210 DATA 0005, "FRANCISCO COSTA", 3
220 DATA 0105, "JOAO DA SILVA", 5
230 DATA 1539, "PEDRO ALONSO", 4
      .
      .
      .
1000 DATA 0000, "X", 0
```

4.2. Instrução ON-GOSUB

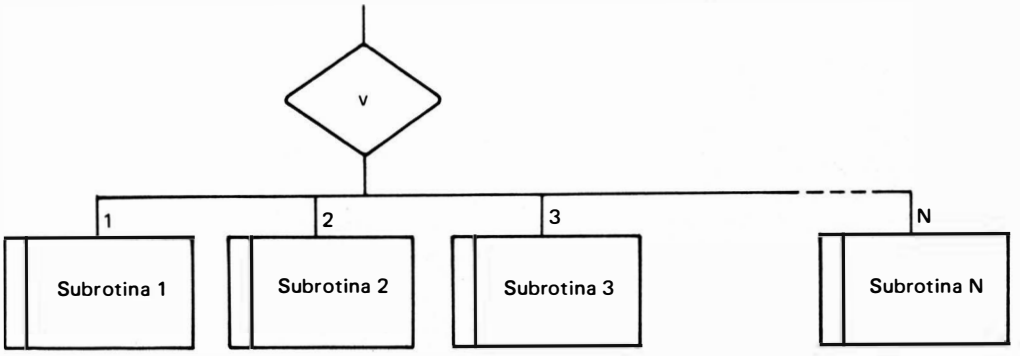
Esta instrução é semelhante à instrução ON-GOTO, com a única diferença de que o fluxo do processamento é desviado para subrotinas que iniciam nas linhas especificadas logo após o GOSUB. A forma geral desta instrução é a seguinte:

$$nl \text{ ON v GOSUB } nl_1, nl_2, nl_3 \ldots$$

onde nl indica o número da linha da instrução

v = variável BASIC
nl_1, nl_2, . . . indicam números de linhas onde se iniciam as subrotinas para as quais poderão ser desviados o fluxo de processamento.

Sendo o funcionamento semelhante à instrução ON-GOTO, possui também uma representação semelhante no fluxograma:



Ex.: 100 ON K GOSUB 1000, 2000, 3000, 4000

Se K for um número inteiro entre 1 e 4, o fluxo do processamento será desviado para as subrotinas: 1000, 2000, 3000, 4000. Se K for, respectivamente, 1, 2, 3 ou 4.

Existem micros que não têm disponível estas duas instruções. Entretanto, elas podem ser facilmente simuladas. A instrução ON-GOTO pode ser simulada pela instrução GOTO v, onde v é uma variável que assume o número da linha para a qual queremos desviar o fluxo do processamento.

```
Ex.: 40 INPUT I
     50 A = 100 * I
     60 GOTO A
```

Dependendo do valor digitado de I, o processamento será desviado para as linhas 100, 200, 300 etc. Analogamente, a instrução ON-GOSUB pode ser simulada pela instrução GOSUB v, onde v é uma variável que assume os números das linhas onde iniciam subrotinas.

```
10 INPUT K
20 M = K * 1000
30 GOSUB M
 .
 .
 .
```

Dependendo do valor de K, será executada uma das subrotinas que iniciam em 1000 ou 2000 ou 3000 etc. . .

Convém observarmos, entretanto, que existem microcomputadores que não aceitam as simulações descritas, notadamente aquelas que possuem as instruções ON-GOTO e ON-GOSUB.

EXEMPLO DE APLICAÇÃO:

Um pesquisador deseja conhecer estatísticas de média, variância e desvio-padrão da variável concentração de monóxido de carbono no ar, que ocorrerem durante um mês, em uma região poluida do Estado de São Paulo. Tem-se disponível a tabela que fornece o dia e a concentração dos 30 dias do mês em questão.

Fazer fluxograma e programa BASIC que permita digitar os 30 valores de concentração e obter através de um menu, conforme Fig. 1, a estatística desejada.

Fig. 1

ESTATÍSTICAS DE CONCENTRAÇÃO DE CO

(1) Média
(2) Variância
(3) Desvio-padrão
(4) Fim de Programa

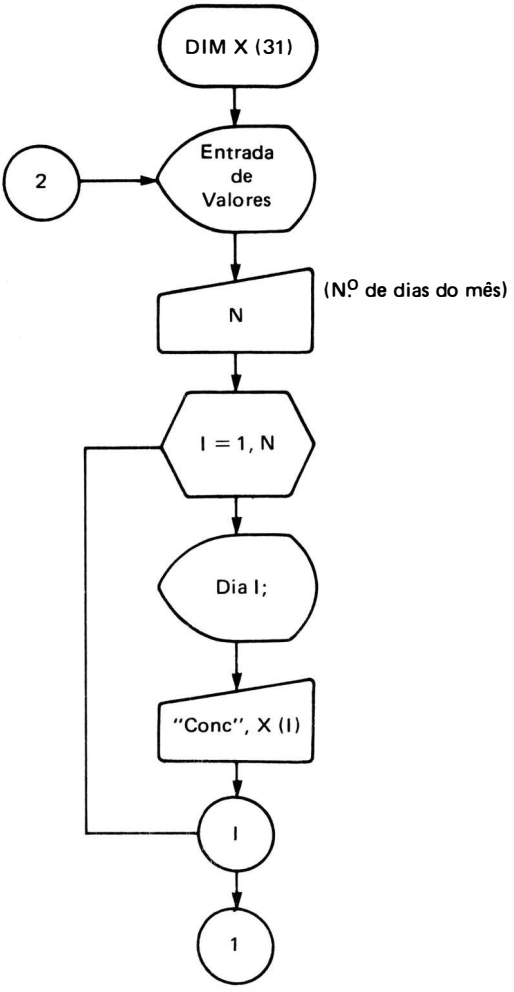
Opção:

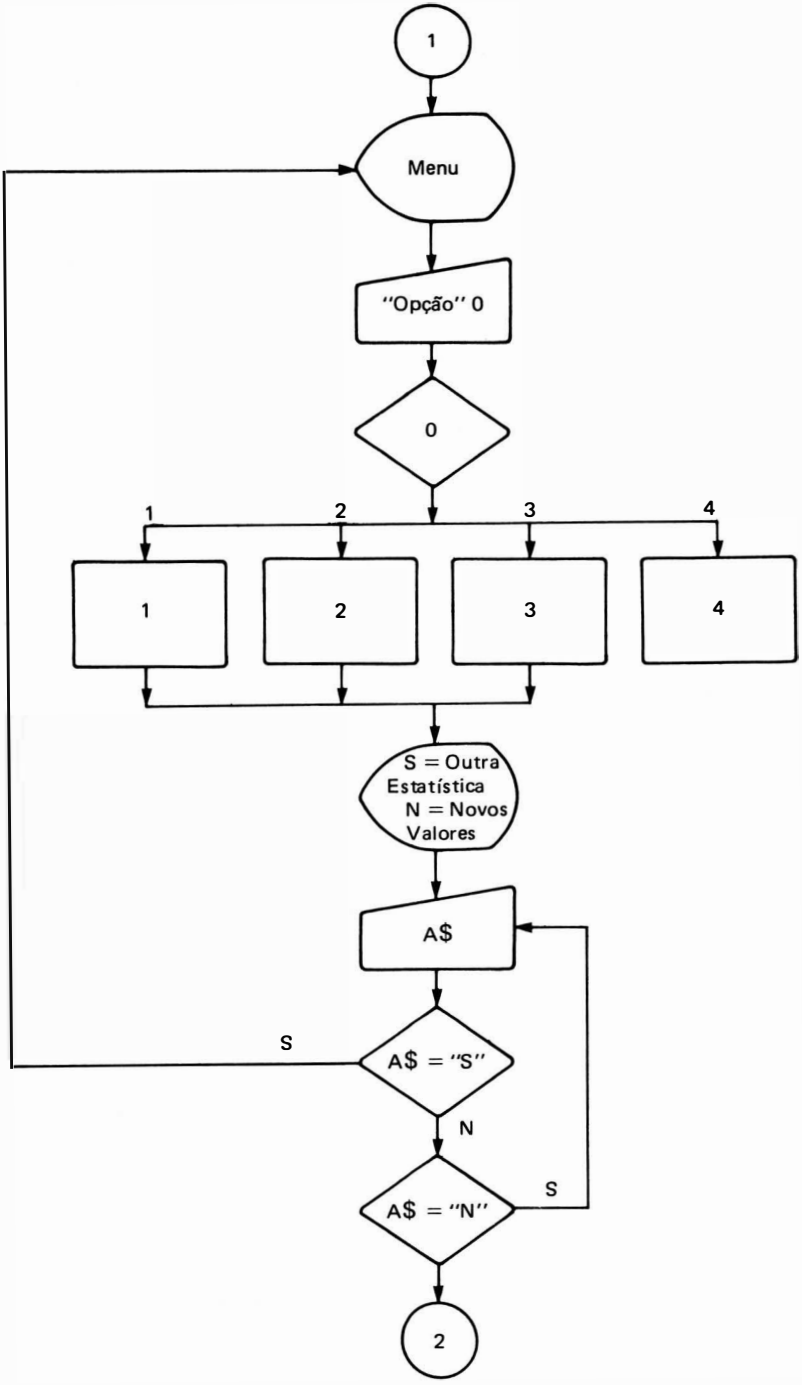
OBS.:

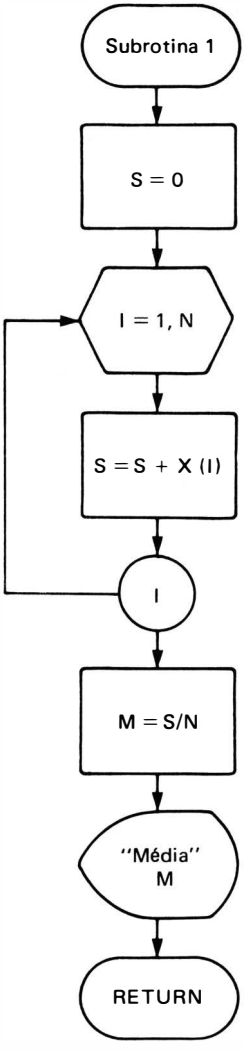
$$\text{Média} = M = \frac{\sum_{i=1}^{N} Xi}{N}$$

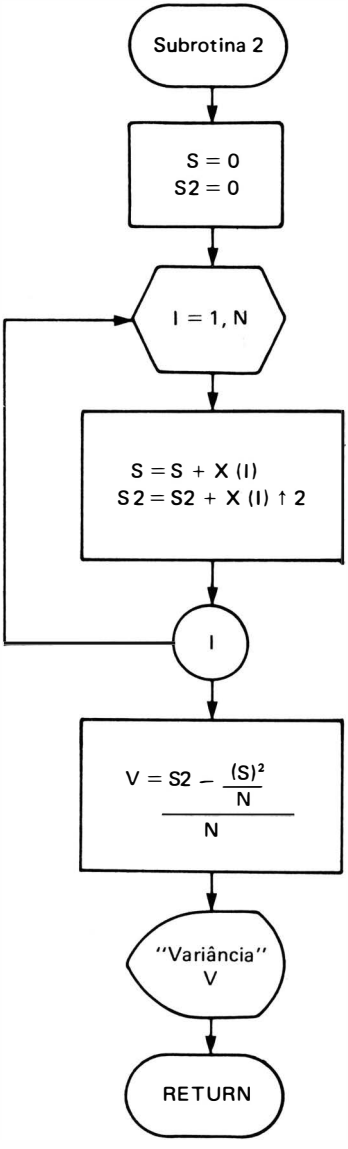
$$\text{Var} = \frac{\sum_{i=1}^{N} Xi^2 - \frac{\left(\sum_{i=1}^{N} Xi\right)^2}{N}}{N}$$

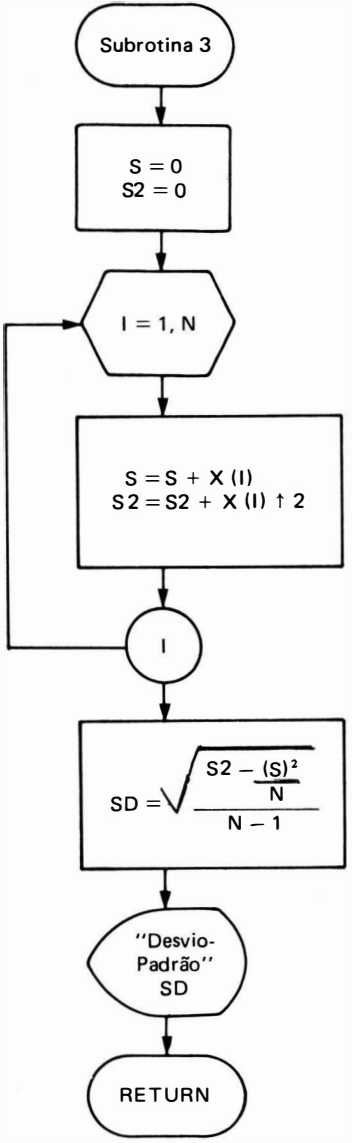
$$\text{Desvio-padrão} = \sqrt{\frac{\sum_{i=1}^{N} Xi^2 - \frac{\left(\sum_{i=1}^{N} xi\right)^2}{N}}{N-1}}$$

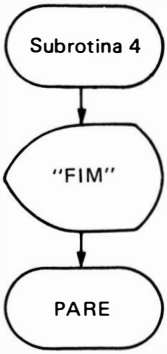

DIM X (31)
2
Entrada de Valores
N
(Nº de dias do mês)
I = 1, N
Dia I;
"Conc", X (I)
I
1


1
Menu
"Opção" 0
0
1
2
3
4
1
2
3
4
S = Outra Estatística
N = Novos Valores
A$
S
A$ = "S"
N
A$ = "N"
S
2


Subrotina 1
S = 0
I = 1, N
S = S + X (I)
I
M = S/N
"Média"
M
RETURN


Subrotina 2
S = 0
S2 = 0
I = 1, N
S = S + X (I)
S2 = S2 + X (I) ↑ 2
I
V = S2 − (S)²/N / N
"Variância"
V
RETURN


Subrotina 3
S = 0
S2 = 0
I = 1, N
S = S + X (I)
S2 = S2 + X (I) ↑ 2
I
SD = √((S2 − (S)²/N)/(N − 1))
"Desvio-Padrão" SD
RETURN


Subrotina 4
"FIM"
PARE

```
10 DIM X (31)
20 CLS
30 PRINT "ENTRADA DE VALORES DE CONCENTRACAO"
40 PRINT
50 PRINT
60 INPUT "NUMERO DE DIAS DO MES = " ; N
70 PRINT
80 PRINT
90 FOR I = 1 TO N
100   PRINT "DIA " ; I ;
110   INPUT "CONCENTRACAO DE CO = " ; X (I)
120   PRINT
130 NEXT I
140 CLS
150 PRINT "ESTATISTICAS DE CONCENTRACAO DE CO"
160 PRINT
170 PRINT
180 PRINT
190 PRINT " (1) MEDIA"
200 PRINT " (2) VARIANCIA"
210 PRINT " (3) DESVIO PADRAO"
220 PRINT " (4) FIM DE PROGRAMA"
230 PRINT
240 PRINT
250 INPUT "OPCAO = " ; O
260 ON O GOSUB 340, 440, 560, 680
270 PRINT
280 PRINT "PARA OUTRA ESTATISTICA DIGITE S "
290 PRINT "PARA ENTRADA DE NOVOS DADOS DIGITE E"
300 INPUT A$
310 IF A$ = "S" GOTO 140
320 IF A$ = "E" GOTO 20
330 GOTO 300
340 REM SUBROTINA DE MEDIA
350 LET S = 0
360 FOR I = 1 TO N
370   LET S = S + X (I)
380 NEXT I
390 LET M = S/N
400 PRINT
410 PRINT "MEDIA = " ; M
420 PRINT
430 RETURN
440 REM SUBROTINA DE VARIANCIA
450 LET S = 0
460 LET S2 = 0
470 FOR I = 1 TO N
480   LET S = S + X (I)
```

```
490   LET S2 = S2 + X [I] ↑ 2
500 NEXT I
510 LET V = (S2 – S ↑ 2/N)/N
520 PRINT
530 PRINT "VARIANCIA = " ; V
540 PRINT
550 RETURN
560 REM SUBROTINA DE DESVIO PADRAO
570 LET S = 0
580 LET S2 = 0
590 FOR I = 1 TO N
600   LET S = S + X (I)
610   LET S2 = S2 + X (I) ↑ 2
620 NEXT I
630 LET SD = SQR ((S2 – S ↑ 2/N)/(N – 1))
640 PRINT
650 PRINT "DESVIO PADRAO = " ; SD
660 PRINT
670 RETURN
680 REM SUBROTINA DE FIM DE PROGRAMA
690 CLS
700 PRINT @ 605, "F I M"
710 END
```

APÊNDICE A

1. CARACTERES ASCII

Como sabemos, o computador trabalha apenas com números binários e por isso foram criados códigos que representassem os valores alfanuméricos. Estes códigos são conjuntos de bits através dos quais um valor alfanumérico pode ser representado, existindo um código único para cada alfanumérico.

O código mais utilizado para codificação de caracteres é o da American Standard Code for Information Interchange, ou ASCII (cuja pronúncia correta é "AS-KEY").

Existem códigos ASCII de 7 e 8 bits, que são capazes de representar, respectivamente, 128 e 256 caracteres, sendo compostos por letras maiúsculas e minúsculas do alfabeto inglês, caracteres decimais numéricos, caracteres especiais e de operação e caracteres de controle, conforme a tabela mostrada a seguir:

Caracter	Código Decimal	Código Hexadecimal	Observação
NUL	00	00	Null
SOH	01	01	Start of Heading
STX	02	02	Start of Text
ETX	03	03	End of Text
EOT	04	04	End of Transmission
ENQ	05	05	Enquiry
ACK	06	06	Acknowledge
BEL	07	07	Bell
BS	08	08	Backspace
HT	09	09	Horizontal Tab
LF	10	OA	Line Feed
VT	11	OB	Vertical Tab
FF	12	OC	Form Feed
CR	13	OD	Carriage Return
SO	14	OE	Shift Out
SI	15	OF	Shift In

Caracter	Código Decimal	Código Hexadecimal	Observação
DLE	16	10	Data Link Escape
DCI	17	11	Device Control 1
DC2	18	12	Device Control 2
DC3	19	13	Device Control 3
DC4	20	14	Device Control 4
NAK	21	15	Negative Acknowledge
SYN	22	16	Synchronous Idle
ETB	23	17	End of Block
CAN	24	18	Cancel
EM	25	19	End of Medium
SUB	26	1A	Substitute
ESC	27	1B	Escape
FS	28	1C	File Separator
GS	29	ID	Group Separator
RS	30	1E	Record Separator
US	31	IF	Unit Separator
⌀	32	20	Space
!	33	21	
"	34	22	
#	35	23	
$	36	24	
%	37	25	
&	38	26	
'	39	27	
(	40	28	
)	41	29	
*	42	2A	
+	43	2B	
,	44	2C	
–	45	2D	
.	46	2E	
/	47	2F	
0	48	30	
1	49	31	
2	50	32	
3	51	33	
4	52	34	
5	53	35	
6	54	36	
7	55	37	
8	56	38	
9	57	39	
:	58	3A	
;	59	3B	
<	60	3C	

Caracter	Código Decimal	Código Hexadecimal	Observação
=	61	3D	
>	62	3E	
?	63	3F	
@	64	40	
A	65	41	
B	66	42	
C	67	43	
D	68	44	
E	69	45	
F	70	46	
G	71	47	
H	72	48	
I	73	49	
J	74	4A	
K	75	4B	
L	76	4C	
M	77	4D	
N	78	4E	
O	79	4F	
P	80	90	
Q	81	51	
R	82	52	
S	83	53	
T	84	54	
U	85	55	
V	86	56	
W	87	57	
X	88	58	
Y	89	59	
Z	90	5A	
[	91	5C	
\	92	5C	
]	93	5D	
↑	94	5E	
_	95	5F	
`	96	60	
a	97	61	
b	98	62	
c	99	63	
d	100	64	
e	101	65	
f	102	66	
g	103	67	
h	104	68	
i	105	69	
j	106	6A	

Caracter	Código Decimal	Código Hexadecimal	Observação
k	107	6B	
l	108	6C	
m	109	6D	
n	110	6E	
o	111	6F	
p	112	70	
q	113	71	
r	114	72	
s	115	73	
t	116	74	
u	117	75	
v	118	76	
w	119	77	
x	120	78	
y	121	79	
z	122	7A	
{	123	7B	
|	124	7C	
}	125	7D	
~	126	7E	
±	127	7F	
	128	80	Os códigos decimais de 128 a 191 são caracteres gráficos. Ver tabela de item 1.5 do Cap. 15.
	.		
	.		
	.		
	191	BF	
	192	CO	Na faixa de 192 a 255 estão os códigos especiais de compressão de espaços.
	.		
	.		
	.		
	255	FF	

2. SISTEMAS DE NUMERAÇÃO

Conforme foi visto no Cap. 11 existem vários sistemas de numeração. Apresentamos neste apêndice um resumo das conversões de base mais usuais em computação.

DECIMAL	HEXADECIMAL	BINÁRIO
0	00	0000 0000
1	01	0000 0001
2	02	0000 0010
3	03	0000 0011
4	04	0000 0100
5	05	0000 0101
6	06	0000 0110
7	07	0000 0111
8	08	0000 1000
9	09	0000 1001
10	OA	0000 1010
11	OB	0000 1011
12	OC	0000 1100
13	OD	0000 1101
14	OE	0000 1110
15	OF	0000 1111
16	10	0001 0000
17	11	0001 0001
18	12	0001 0010
19	13	0001 0011
20	14	0001 0100
21	15	0001 0101
22	16	0001 0110
23	17	0001 0111
24	18	0001 1000
25	19	0001 1001
26	1A	0001 1010
27	1B	0001 1011
28	1C	0001 1100
29	1D	0001 1101
30	1E	0001 1110
31	1F	0001 1111

DECIMAL	HEXADECIMAL	BINÁRIO
32	20	0010 0000
33	21	0010 0001
34	22	0010 0010
35	23	0010 0011
36	24	0010 0100
37	25	0010 0101
38	26	0010 0110
39	27	0010 0111
40	28	0010 1000
41	29	0010 1001
42	2A	0010 1010
43	2B	0010 1011
44	2C	0010 1100
45	2D	0010 1101
46	2E	0010 1110
47	2F	0010 1111
48	30	0011 0000
49	31	0011 0001
50	32	0011 0010
51	33	0011 0011
52	34	0011 0100
53	35	0011 0101
54	36	0011 0110
55	37	0011 0111
56	38	0011 1000
57	39	0011 1001
58	3A	0011 1010
59	3B	0011 1011
60	3C	0011 1100
61	3D	0011 1101
62	3E	0011 1110
63	3F	0011 1111

DECIMAL	HEXADECIMAL	BINÁRIO
64	40	0100 0000
65	41	0100 0001
66	42	0100 0010
67	43	0100 0011
68	44	0100 0100
69	45	0100 0101
70	46	0100 0110
71	47	0100 0111
72	48	0100 1000
73	49	0100 1001
74	4A	0100 1010
75	4B	0100 1011
76	4C	0100 1100
77	4D	0100 1101
78	4E	0100 1110
79	4F	0100 1111
80	50	0101 0000
81	51	0101 0001
82	52	0101 0010
83	53	0101 0011
84	54	0101 0100
85	55	0101 0101
86	56	0101 0110
87	57	0101 0111
88	58	0101 1000
89	59	0101 1001
90	5A	0101 1010
91	5B	0101 1011
92	5C	0101 1100
93	5D	0101 1101
94	5E	0101 1110
95	5F	0101 1111

DECIMAL	HEXADECIMAL	BINÁRIO
96	60	0110 0000
97	61	0110 0001
98	62	0110 0010
99	63	0110 0011
100	64	0110 0100
101	65	0110 0101
102	66	0110 0110
103	67	0110 0111
104	68	0110 1000
105	69	0110 1001
106	6A	0110 1010
107	6B	0110 1011
108	6C	0110 1100
109	6D	0110 1101
110	6E	0110 1110
111	6F	0110 1111
112	70	0111 0000
113	71	0111 0001
114	72	0111 0010
115	73	0111 0011
116	74	0111 0100
117	75	0111 0101
118	76	0111 0110
119	77	0111 0111
120	78	0111 1000
121	79	0111 1001
122	7A	0111 1010
123	7B	0111 1011
124	7C	0111 1100
125	7D	0111 1101
126	7E	0111 1110
127	7F	0111 1111

DECIMAL	HEXADECIMAL	BINÁRIO
128	80	1000 0000
129	81	1000 0001
130	82	1000 0010
131	83	1000 0011
132	84	1000 0100
133	85	1000 0101
134	86	1000 0110
135	87	1000 0111
136	88	1000 1000
137	89	1000 1001
138	8A	1000 1010
139	8B	1000 1011
140	8C	1000 1100
141	8D	1000 1101
142	8E	1000 1110
143	8F	1000 1111
144	90	1001 0000
145	91	1001 0001
146	92	1001 0010
147	93	1001 0011
148	94	1001 0100
149	95	1001 0101
150	96	1001 0110
151	97	1001 0111
152	98	1001 1000
153	99	1001 1001
154	9A	1001 1010
155	9B	1001 1011
156	9C	1001 1100
157	9D	1001 1101
158	9E	1001 1110
159	9F	1001 1111

DECIMAL	HEXADECIMAL	BINÁRIO
160	A0	1010 0000
161	A1	1010 0001
162	A2	1010 0010
163	A3	1010 0011
164	A4	1010 0100
165	A5	1010 0101
166	A6	1010 0110
167	A7	1010 0111
168	A8	1010 1000
169	A9	1010 1001
170	AA	1010 1010
171	AB	1010 1011
172	AC	1010 1100
173	AD	1010 1101
174	AE	1010 1110
175	AF	1010 1111
176	B0	1011 0000
177	B1	1011 0001
178	B2	1011 0010
179	B3	1011 0011
180	B4	1011 0100
181	B5	1011 0101
182	B6	1011 0110
183	B7	1011 0111
184	B8	1011 1000
185	B9	1011 1001
186	BA	1011 1010
187	BB	1011 1011
188	BC	1011 1100
189	BD	1011 1101
190	BE	1011 1110
191	BF	1011 1111

DECIMAL	HEXADECIMAL	BINÁRIO
192	C0	1100 0000
193	C1	1100 0001
194	C2	1100 0010
195	C3	1100 0011
196	C4	1100 0100
197	C5	1100 0101
198	C6	1100 0110
199	C7	1100 0111
200	C8	1100 1000
201	C9	1100 1001
202	CA	1100 1010
203	CB	1100 1011
204	CC	1100 1100
205	CD	1100 1101
206	CE	1100 1110
207	CF	1100 1111
208	D0	1101 0000
209	D1	1101 0001
210	D2	1101 0010
211	D3	1101 0011
212	D4	1101 0100
213	D5	1101 0101
214	D6	1101 0110
215	D7	1101 0111
216	D8	1101 1000
217	D9	1101 1001
218	DA	1101 1010
219	DB	1101 1011
220	DC	1101 1100
221	DD	1101 1101
222	DE	1101 1110
223	DF	1101 1111

DECIMAL	HEXADECIMAL	BINÁRIO
224	E0	1110 0000
225	E1	1110 0001
226	E2	1110 0010
227	E3	1110 0011
228	E4	1110 0100
229	E5	1110 0101
230	E6	1110 0110
231	E7	1110 0111
232	E8	1110 1000
233	E9	1110 1001
234	EA	1110 1010
235	EB	1110 1011
236	EC	1110 1100
237	ED	1110 1101
238	EE	1110 1110
239	EF	1110 1111
240	F0	1111 0000
241	F1	1111 0001
242	F2	1111 0010
243	F3	1111 0011
244	F4	1111 0100
245	F5	1111 0101
246	F6	1111 0110
247	F7	1111 0111
248	F8	1111 1000
249	F9	1111 1001
250	FA	1111 1010
251	FB	1111 1011
252	FC	1111 1100
253	FD	1111 1101
254	FE	1111 1110
255	FF	1111 1111

3. DEFINIÇÃO FORMAL DA LINGUAGEM BASIC (Notação BNF)

A maneira formal de se definir uma linguagem de programação utiliza a notação BNF, ou Backus-Naur-Form (ainda alguns autores a citam como Backus-Normal-Form). Esta notação utiliza-se de uma metalinguagem, ou seja, uma linguagem que explica uma outra linguagem, que é por sua vez composta por metasímbolos definidos a seguir:

a) O metasímbolo : = é lido "definido como sendo", indicando a definição de uma tese.

b) Os metasímbolos < > indicam que a tese está sendo definida, é a que estiver entre estes dois sinais.

c) O metasímbolo | é lido como "ou", indicando que a definição da tese pode ter mais de um significado.

d) Os metasímbolos $\{\ \}_m^M$ indicam a repetição da(s) tese(s) que está entre estes dois sinais.

O "m" indica o menor número de repetições possível, enquanto que o "M" indica o maior número possível de repetições daquela tese.

Como a linguagem BASIC não é padronizada, existindo uma infinidade de dialetos para a mesma, será definida a linguagem BASIC original, ou seja, a sua primeira versão, desenvolvida no Dartmouth College, que é de onde todos os dialetos desta linguagem se originam.

< caracter alfabético > : = A | B | C | D | E | F | G | H | I | J | K | L | M |
N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

< dígito > : = Ø | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

< caracter especial > : = + | − | * | / | = | (|) | > | < | . | , | ; | ↑ | b

OBS.: b indica espaço em branco

< inteiro > : = $\{$< dígito >$\}_1^9$

< fracionário > : = . < inteiro >

< número decimal > : = $\{$< dígito >$\}_1^{n \leq 9}$. $\{$< dígito >$\}_0^{9-n}$

< sinal > : = < nada > | + | −

< expoente > : = E < sinal > $\{$< dígito >$\}_1^2$

< número > : = $\begin{cases} \{ < \text{inteiro} > \mid < \text{fracionário} > \mid < \text{número decimal} > \}_1^1 \\ \{ < \text{expoente} > \}_0^1 \end{cases}$

< número com sinal > : = < sinal > < número >

< variável simples > : = < caracter alfabético > $\{$< dígito >$\}_0^1$

< variável indexada > : =
< caracter alfabético > (< expressão > $\{$, < expressão >$\}_0^1$)

< variável > : = < variável simples > | < variável indexada >

< nome de função > : = ABS | ATN | COS | EXP | INT | LOG | RND |
SIN | SQR | TAN | FN < caracter alfabético >

< função argumentada > : = < nome da função > (< expressão >)

< argumento > : = < número > | < variável > |
< função argumentada > | (< expressão >)

< fator de potenciação > : = < argumento > | < fator de potenciação >
< fator de potenciação > ↑ < argumento >

< fator multiplicador > : = < fator de potenciação > |
< fator multiplicador > $\left\{ * \mid / \right\}_1^1$ < fator de potenciação >

< expressão > : = < fator multiplicador > | < sinal > < expressão > |
< expressão > $\left\{ + / - \right\}_1^1$ < fator multiplicador >

< instrução de atribuição > : = LET < variável > = < expressão >

< lista de leitura > : = < variável > $\left\{ , < \text{variável} > \right\}_0^\infty$

< instrução READ > : = READ < lista de leitura >

< lista de números > : = < número com sinal > $\left\{ , < \text{número com sinal} > \right\}_0^\infty$

< Instrução DATA > : = DATA < lista de números >

< Mensagem > : = " $\left\{ < \text{caracter alfabético} > \mid < \text{dígito} > \mid < \text{caracter especial} > \right\}_0^\infty$ "

< Item de impressão > : = < expressão > | < mensagem > | < mensagem > < expressão >

< lista de impressão > : = < nada > | < item de impressão > $\left\{ , \text{item de impressão} > \right\}_0^\infty$
$\left\{ , \right\}_0^1$

< instrução PRINT > : = PRINT < impressão de listagem >

< número de linha > : = $\left\{ < \text{dígito} > \right\}_1^3$

< Instrução GOTO > : = GO $\left\{ \not{b} \right\}_0^1$ TO < número de linha >

< Comentário > : = REM $\left\{ < \text{caracter alfabético} > \mid < \text{dígito} > \mid < \text{caracter especial} > \right\}_0^\infty$

< Operador Relacional > : = > | < | < = | < > | > = | =

< Instrução IF > : = IF < expressão > < operador relacional > < expressão >
THEN < número de linha >

< Instrução FOR > : = FOR < variável simples > = < expressão > TO
< expressão > $\left\{ \text{STEP} < \text{expressão} > \right\}_0^1$

< Instrução NEXT > : = NEXT < variável simples >

< Instrução END > : = END

< Dimensão > : = < inteiro > $\left\{ , < \text{inteiro} > \right\}_0^1$

< Variável de dimensionamento > : = < caracter alfabético > (< dimensão >)

< Instrução DIMension > : = DIM < variável de dimensionamento >
$\left\{ , < \text{variável de dimensionamento} > \right\}_0^\infty$

< Instrução de DEFinição > : = DEF $\not{b}$ FN < caracter alfabético > (< variável simples >)
= < expressão >

< Instrução GOSUB > : = GOSUB < número de linha >

< Instrução RETURN > : = RETURN

< Corpo de Instrução > : = < Instrução de Atribuição > |
< Instrução READ > | < Instrução DATA > |
< Instrução GOTO > | < Instrução PRINT > |
< Instrução FOR > | < Instrução NEXT > |
< Instrução DIMension > |
< Instrução de DEFinição > |
< Instrução GOSUB > | < Instrução RETURN > |
< Comentário >

< Número de Comando > : = $\left\{ < \text{dígito} > \right\}_{1}^{3}$ ℬ

< Instrução BASIC > : = < número de comando > < corpo de instrução >

< Programa BASIC > : = $\left\{ < \text{instrução BASIC} > \right\}_{1}^{\infty}$
< número de comando > < instrução END >

APÊNDICE B

UNIDADE DE ACIONAMENTO DE DISCO

A unidade de disco, ou disc driver, é um componente do sistema de computação eletrônica que permite o armazenamento de arquivos e programas de forma não volátil, isto é, as informações não são perdidas quando desligamos o computador. As informações são gravadas e lidas de um disco de forma magnética, semelhante a um gravador de fitas, porém na unidade de disco não é necessário que para encontrar uma informação, arquivo ou programa, todo o disco seja lido desde o início, como no caso da fita magnética, o que caracteriza a unidade de acionamento de disco como sendo um dispositivo de *acesso aleatório*.

Existem vários tipos de discos, como os discos rígidos, com capacidade de 5 a 10 megabytes; discos tipo winchester, que funcionam em uma câmara lacrada isenta de qualquer tipo de pó ou sujeira e permite um armazenamento de informações até dez vezes maior que os discos de tecnologia convencional; temos ainda os disquetes, discos flexíveis ou em inglês floppy discs que são os discos mais utilizados em sistemas de microcomputadores e por isso vamos estudá-lo melhor.

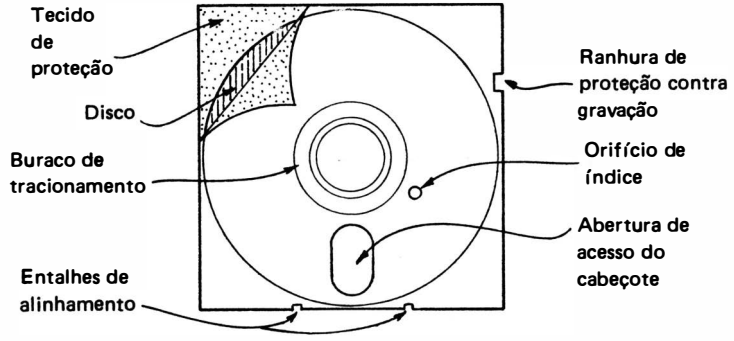
1. DISQUETES

Os disquetes são feitos de um material magnético que é depositado sobre uma base de vinil ou PVC flexível e colocado dentro de um envelope de PVC uniforme e opaco que serve de proteção para o material magnético, além de possibilitar o manejo do disco. Este envelope tem uma abertura central que permite o encaixe do mecanismo de rotação, fazendo com que o disco gire dentro do envelope; possui também uma abertura oval que permite que a cabeça de leitura e gravação entre em contato com a superfície do disco.

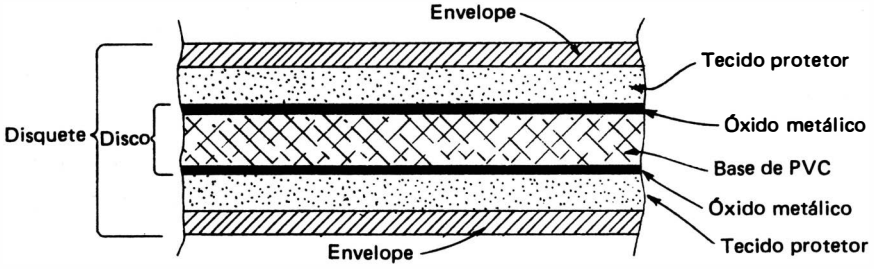
O disquete também possui dois entalhes para alinhamento dentro da unidade de disco, um orifício que indica o início dos setores para disquetes setorizados por hardware (ver mais adiante), e finalmente um recorte lateral que serve para permitir ou não a gravação de dados, arquivos ou programas no disquete, dependendo se este recorte estiver aberto — permitindo a gravação — ou coberto — não permitindo gravação.

Internamente o envelope é revestido por um tecido a base de silicone com espessura média de 0,2 mm e tem por finalidade limpar a superfície do disco continuamente, eliminando

principalmente poeira; prevenir a formação de cargas estáticas que poderiam danificar a gravação de dados ou programas além de reduzir ao mínimo o atrito com o envelope.



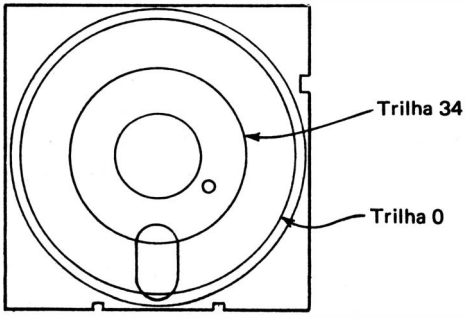
Esquematicamente, se fizermos um corte vertical no disquete teríamos as camadas, conforme a figura:



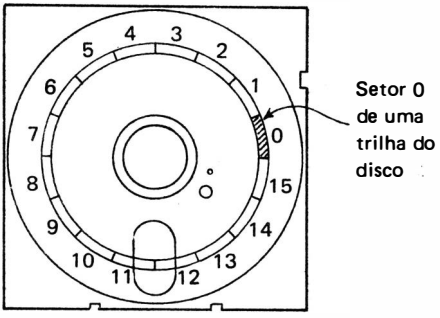
1.1. Organização do Disquete

O rápido acesso a uma informação quando utiliza-se uma unidade de disco deve-se principalmente à sua organização na forma de distribuir os dados pela superfície do disco, chamada *formatação*. Trata-se da divisão de disquete na forma de trilhas e setores.

As trilhas são círculos concêntricos, isto é, um dentro do outro, iniciando pela trilha Ø na parte mais externa do disco e terminando na última trilha na parte mais interna do disco, que no caso dos micros da linha Apple é a trilha nº 34, conforme a figura:



Os setores são subdivisões das trilhas e também no caso dos micros Apple são de número de 16 setores de 256 bytes cada, numerados de 0 a 15 conforme a figura:



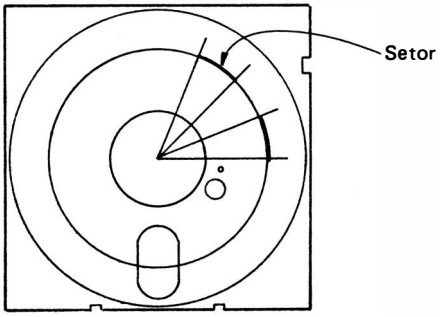
Para localizar-se um determinado byte armazenado no disco, basta-se localizar o número da trilha, o número do setor e procurá-lo apenas entre 256 bytes, o que é muito mais rápido que procurá-lo entre 143.360 bytes que cada disquete possui.

Para localizar-se a trilha de um determinado disco, existe um mecanismo que move a cabeça de leitura e gravação até a posição do disco onde se encontra a trilha 0, e a partir daí, move-se até a trilha desejada em múltiplos da distância entre duas trilhas consecutivas, valor este que é conhecido (0,53 mm para disquetes 5 1/4").

Para localizar-se um setor existem 3 métodos diferentes que definem o tipo de formatação utilizada pela unidade de disco, que são:

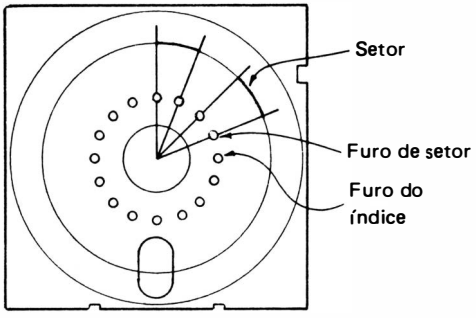
- setorização por software padrão IBM 3710
- setorização por software padrão Apple
- setorização hardware

Na setorização por software padrão IBM 3740, os disquetes têm apenas um orifício de índice que indica posição do setor 0. Neste método, existe um sistema eletrônico de emissão e recepção de luz (na faixa infravermelho) que envia um sinal aos circuitos do acionador de discos, quando o emissor de luz "vê" o receptor de luz através do furo de índice, indicando o início dos setores. Os demais setores são localizados a partir de uma temporização feita e baseada na velocidade de rotação do disco (300 rpm). Esquematicamente teremos:



A setorização por software padrão Apple não utiliza o furo de sensor de índice, sendo que a identificação de setor 0 é feita através do software. Não existe para este método o sistema de indicação de início de setores, sendo a temporização totalmente eletrônica e, portanto, podem ser utilizados tanto disquetes para setorização por software como os disquetes especiais para setorização por hardware.

Na setorização por hardware o disco é especial, possuindo um furo no início de cada setor, além de um furo especial a mais no início do primeiro setor. A localização de um determinado setor é feita através da contagem de furos a partir do primeiro setor. Esquematicamente teríamos:



1.2. Cuidados com o Disquete e com a Unidade de Disco

Para o correto funcionamento do sistema de armazenamento secundário, composto pela unidade de acionamento de disco e pelo disquete, devemos ter alguns cuidados para garantir uma longa vida útil dos mesmos.

A unidade de disco é um dispositivo que capta informações do disquete através de movimentos mecânicos de precisão. Possui interiormente ajustes de posicionamentos mecânicos extremamente sensíveis de forma a identificar uma trilha de gravação de 0,30 mm de largura e uma distância entre trilhas de 0,53 mm.

Transporte-a com cuidado, não dê batidas nem deixe-a cair, nem tampouco deixe objetos e, é claro, líquidos cair sobre a unidade de disco.

A cabeça de leitura e gravação é sem dúvida o componente mais delicado da unidade de disco. Ela deve ser limpa somente quando houver acúmulo de óxido em sua superfície, utilizando um tecido macio e seco. Pode-se utilizar uma solução de álcool isopropílico para facilitar a operação de limpeza, mas enxugando sempre com um pano seco. Jamais fume durante a limpeza ou toque a cabeça de leitura e gravação com os dedos pois resíduos de óxido metálico, poeira e fuligem podem danificar definitivamente a cabeça. Existe no mercado um tipo especial de disquete para limpeza de cabeçote, devendo ser utilizado de 3 em 3 meses ou quando estiverem ocorrendo erros de leitura de forma muito freqüente.

O disquete também requer um tratamento especial, para que não comprometamos a sua vida útil.

Os disquetes não são eternos. Possuem uma vida média da ordem de 40 horas de uso contínuo. Este tempo é relativamente longo uma vez que uma operação de leitura ou gravação no disquete leva apenas alguns segundos. Entretanto, existem algumas trilhas que são sempre consultadas em todas as operações, e são justamente nelas que irão ocorrer o maior desgaste do disquete.

As recomendações no trato com o disquete são as seguintes:

1) Jamais toque nas áreas expostas do disco, nem com os dedos, tampouco com materiais metálicos.
   A impressão digital deixada na superfície do disco, impedirá a leitura daqueles setores;

2) Não permita que seu disquete fique contaminado com o acúmulo de poeira, graxa, álcool, óleo ou cinza de cigarro;

3) Nunca tente limpar a superfície do disquete pois poderão ser retiradas partículas do óxido metálico danificando definitivamente o disquete;

4) Cuidado com campos magnéticos. O disquete armazena dados magneticamente e um campo magnético externo perto do disquete, como um imã de alto-falante ou o tubo de imagem de seu terminal de vídeo, podem apagar partes de seus dados;

5) Não deixe o disquete exposto ao sol. Altas temperaturas deformam fisicamente o disco, fazendo com que sejam alteradas as distâncias entre trilhas não permitindo sua leitura, o que pode ser até de forma definitiva. As condições climáticas para o perfeito funcionamento são: temperatura de 10° C a 50° C e umidade relativa de 8% a 80%.

6) Não escreva os rótulos do disquete com caneta esferográfica ou lápis quando o rótulo já estiver colocado ao disquete. Utilize canetas de ponta porosa ou heliográficas.

7) Recoloque o disquete no envelope protetor sempre que for removido da unidade de disco, nunca deixando-os sobre mesas ou sobre o equipamento sem esta proteção.

8) Mantenha seus disquetes mais importantes e/ou os mais utilizados com uma cópia sempre atualizada (back-up) e guardado em lugar seguro, utilizando para isso sempre disquetes de boa qualidade, o que previne desagradáveis surpresas.

GLOSSÁRIO

A

- ABS – absoluto; módulo
- Acesso – maneira pela qual o computador faz referência ou atinge um conjunto de dados ou arquivos.
- Acumulador – registrador para armazenamento de dados temporários, durante a execução de um programa.
- Aleatório – imprevisível; número que é conseguido ou gerado pelas leis da probabilidade.
- Alfanumérico – contração das palavras alfabético e numérico. Qualquer caracter que compõe uma linguagem de programação, incluindo os alfabéticos, os números, caracteres especiais e o espaço em branco.
- Algoritmo – descrição de um processo para se resolver um problema passo a passo.
- Alocação – reserva de área de memória onde são fixados os endereços simbólicos de forma absoluta.
- Analógico – forma contínua de apresentação de dados de grandezas físicas como temperatura, pressão, tensão, corrente etc.
- Argumento – variável independente que define o valor para o qual será calculada uma operação ou função.
- Arquivo – conjunto de dados armazenados na memória, relacionados entre si, formando uma unidade.
- Array – matriz, tabela ou valor de dados indexados que podem ser acessados diretamente.
- ASCII – abreviação em inglês de American Standard Code for Information Interchange, ou seja, código padrão americano para intercâmbio de informações que emprega um conjunto de 8 bits representando 128 caracteres, mais um bit de paridade, adotado com o objetivo de padronizar a troca de dados entre sistemas de processamentos e periféricos de diferentes fabricações.
- Assembler – montador – programa que atua sobre as instruções de uma linguagem simbólica, produzindo a partir delas instruções de máquina.
- Assembly – montagem.
- A to D converter – conversor de analógico para digital.

- Atribuição — Instrução da Linguagem BASIC que associa um valor numérico ou alfanumérico a um endereço de uma localidade da memória.

B

- Base de numeração — número que identifica a quantidade de elementos permitidos em um sistema de numeração.

- BCD — abreviação em inglês de Binary-Coded-Decimal ou decimal codificado em binário.

- Binário — sistema de numeração que utiliza a base 2 dispondo assim apenas dos algarismos 0 e 1.

- Bit — aglutinação das palavras BInary digiT ou dígito binário; unidade mínima de informação.

- Buffer — unidade de memória de uso temporário que compatibiliza as diferentes velocidades de programação, muito mais rápida que as dos periféricos.

- Byte — menor unidade endereçável na memória do computador consistindo em um conjunto de 8 bits (variavelmente, 1 bit a mais para paridade).

C

- Caracter — um dos símbolos utilizados na representação de dados, pertencente ao conjunto de caracteres da linguagem.

- Código — conjunto de sinais, símbolos, regras usadas para fazer a conversão de dados de um sistema de representação para outro.

- Código Objeto — resultado da conversão feita por um compilador ou interpretador, ou um Assembler especial; forma de instrução executável pelo computador.

- Comando — ordem que inicia ou encerra uma determinada operação do computador.

- Comentário — observações feitas no interior do programa com a finalidade de documentá-lo. Não toma participação durante a execução, aparecendo apenas durante a listagem.

- Compilação — processo de conversão de uma linguagem de alto nível na qual é escrito um programa fonte para um programa objeto escrito em linguagem de máquina.

- Compilador — programa que faz a compilação.

- Computador — máquina capaz de fazer processamento de dados automaticamente.

- Contador — variável que é incrementada ou decrementada, normalmente de uma unidade, a cada ocorrência de um evento durante a execução de um programa.

- CPU — abreviação de Central Processing Unit ou unidade central de processamento.

- Cursor – elemento indicativo do terminal de vídeo que indica a posição de entrada do próximo caracter.

D

- Dados – representação de uma informação quantificável que pode ser processada por um computador.

- Debug – ver Depurar.

- Decodificar – operação interna na qual um determinado código pode ser interpretado e entendido.

- Delete – operação de eliminação ou remoção, desde um caracter, uma linha, um programa até um arquivo.

- Depurar – processo de localização e correção de erros de um programa, através de testes sucessivos, até que funcione corretamente.

- Digitar – processo de introdução de dados no computador através do teclado.

- Dígito – um dos 10 símbolos que representam números inteiros de 0 a 9.

- Disco flexível – unidade de armazenamento auxiliar, para guardar de maneira não volátil programas e dados.

E

- Editar – processo de corrigir informações, dados ou programas.

- Evento – acontecimento de alguma ação que modifique o valor das variáveis.

- Execução – fase do processamento automático de dados onde os dados de entrada são utilizados pelo programa para obter-se os dados de saída.

- Expoente – número que indica o número de vezes que um valor é multiplicado por si mesmo.

- Expressão – combinação de variáveis, operadores e funções da linguagem na qual está sendo escrito o programa.

F

- Firmware – conjunto de instruções armazenadas em memória ROM. É um software na forma de hardware.

- Flag – valor numérico ou alfanumérico que sinaliza o fim de um arquivo de dados; Bandeira.

- Fluxo de processamento — caminho percorrido pelo programa durante sua execução.

- Fluxograma — esquema gráfico, com símbolos padronizados, das operações necessárias na resolução de um problema, para posterior programação.

- Função — conjunto de operações que são efetuadas sobre uma variável independente, chamada argumento para obter-se um determinado resultado.

G

- Gap — separação.

- Gate — dispositivo eletrônico que realiza operações lógicas como E, OU, NOT.

H

- Hardware — conjunto de circuitos eletrônicos, elementos mecânicos, elétricos e magnéticos que compõem um computador. É a parte física do computador.

- Hexadecimal — sistema de numeração que utiliza a base 16, onde os dígitos são os números de 0 a 9 e as letras de A a F, que correspondem aos números de 10 a 15 decimal, mas que devem ser representados por um único símbolo no sistema de base 16.

I

- Impressora — periférico do computador que fornece relatórios impressos na forma de listagens.

- Índice — variável independente que identifica um elemento de uma matriz, tabela ou vetor.

- Interface — dispositivo que une o hardware do computador a um outro ponto ou dispositivo externo de forma a deixá-los compatíveis entre si.

- Instrução — código binário que indica ao computador qual a operação que deve ser executada.

- Iteração — repetição de um trecho de programa.

J

- Jack — tomada de gravador.

- Job — trabalho ou tarefa; conjunto de programas afins.

- Jump — salto para um determinado ponto do programa.

K

- KBytes — kilobytes ou conjunto de 1024 bytes.

L

- Laço — lista de instruções que se repete várias vezes.

M

- Matriz — variável indexada com duas dimensões.
- Memória — dispositivo que tem a capacidade de reter uma informação para aproveitamento posterior.
- Microcomputador — computador completo com CPU, unidade de entrada (teclado), unidade de saída (vídeo ou impressora e memória), porém, de pequena dimensão.

N

- Nulo — caracter ASCII 00; ausência de informação (diferente do espaço em branco).

O

- On Line — equipamento que está ligado ao computador, trocando informações e sendo controlado pela CPU.
- Overflow — estado causado por valores numéricos que excedem o valor máximo permitido pelo computador.

P

- Paridade — método de checagem de um número binário, podendo ser do tipo par ou ímpar. É um bit colocado a mais que acumula a soma de todos os bits 1 do byte.
- Passo — valor de incremento de uma variável controladora.
- Periférico — unidade externa a CPU que promove sua comunicação com o meio externo.
- Programa — lista de instruções armazenadas na memória do computador que seguem um algoritmo para resolver um problema.

R

- RAM — Random Acess Memory ou memória de acesso aleatório.
- Reset — inicializar o computador, colocar um dispositivo em sua condição inicial de operação.
- ROM — Read Only Memory ou memória apenas de leitura. Memória que não pode ser alterada e não volátil que contém rotinas necessárias para o funcionamento do computador.

Rotina — trecho de programa, colocado em posição definida na memória que faz o computador executar uma tarefa ou operação determinada.

S

Sintaxe — regra que determina a estrutura de uma palavra escrita em uma determinada linguagem.

Sistema — conjunto de componentes ou equipamentos arranjados de forma a realizar uma ou várias ações determinadas.

Sistema Operacional — conjunto de programas que controlam, executam e supervisionam as operações de um computador.

Software — conjunto de programas, documentação e procedimentos relativos a operação de um computador.

String — cadeia de caracteres alfanuméricos e especiais.

T

Tempo compartilhado — forma de um sistema de computação onde podem ser executados vários programas diferentes ao mesmo tempo em diferentes terminais.

Tempo real — regime de trabalho de um computador onde as respostas são dadas imediatamente após a entrada dos dados.

Terminal — dispositivo de entrada ou saída que permite a comunicação entre o usuário e a CPU do computador.

U

UPC — unidade central de processamento. Ver CPU.

V

Variável — letra ou letra seguida de um número que pode assumir diferentes valores no decorrer de um programa.

Variável indexada — variável seguida por um índice que aponta a posição real do valor armazenado na tabela.

BIBLIOGRAFI

BARRAS, R. — *Os Cientistas precisam escrever*. São Paulo, EDUSP, 1979.

BATISTA, L. & KATAKURA, G. M. — *Elementos de programação em BASIC*. São Pau Edgard Blücher, 1983.

BORATTO, F. — *BASIC para engenheiros e cientistas*. Rio de Janeiro, LTC, 1984.

BORGES, J. A. S. — *BASIC — aplicações comerciais*, Rio de Janeiro, LTC, 1983.

CAMPOS, R. J. A. — *Computação básica e programação*. São Paulo, Atlas, 1979.

FREGNI, E. & LANGDON Jr., G. G. — *Projeto de computadores digitais*. São Paulo, Edga Blücher, 1974.

GONICK, L. — *Introdução ilustrada à computação*. São Paulo, Harper & Row do Brasil, 1984.

GOTTFRIED, B. S. — *Programação com BASIC*. São Paulo, MacGraw-Hill, 1984.

MIRSHAWKA, V. — *BASIC sem segredos*. São Paulo, Nobel, 1983.

PACITTI, T. & ATKINSON, C. P. — *Programação e métodos computacionais*, Rio de Janei LTC, 1981.

PACITTI, T. — *FORTRAN — monitor — princípios*, Rio de Janeiro, LTC, 1981.

POOLE, I. et alii — *Programas usuais em BASIC para sistemas compatíveis com o TRS 80*. S Paulo, McGraw-Hill, 1984.

RAHNSTORFI, G. — *Processamento de Dados*. São Paulo, Polígono, 1969.

SALVETTI, D. D. & AZEVEDO, A. S. — *Elementos de programação FORTRAN IV*. São Paul Editora Nacional, 1976.

SHEID, F. — *Computadores e programação*. São Paulo, McGraw-Hill, 1984.

SHEID, F. — *Introdução à ciência dos computadores*. São Paulo, MacGraw-Hill, 1971.

SHIMIZU, T. — *BASIC — exercícios e problemas resolvidos — aplicações comerciais e científi cas — simulação de jogos e gráficos*. São Paulo, Atlas, 1984.

TREMBLAY, J. P. — *Ciência dos computadores — uma abordagem algorítmica*. São Paul MacGraw-Hill, 1983.

WARDLE, M. E. — *Computação — do problema ao programa*. RJ, Guanabara Dois, 1979.

WOLF, G. W. — *Computers peripherals that you can build*. USA, Tab. Books, 1982.

ZUFFO, J. A. — *Microprocessadores — dutos de sistema técnicas de interface e sistemas comunicação de dados*. São Paulo, Edgard Blücher, 1981.

CURSO DE PROGRAMAÇÃO BASIC

JORGE TREVISAN

O Computador é uma ferramenta de trabalho imprescindível nos dias de hoje. Portanto, o aprendizado de uma linguagem de programação e o domínio de um computador é fundamental.

Este livro procura abordar de forma clara, direta e metódica, todos os tópicos necessários para o ensino da linguagem BASIC a alunos que não tenham conhecimento anterior sobre o assunto.

Os textos vêm acompanhados de exercícios, com grau crescente de dificuldade, e quando necessário, é incluído texto explicativo ao exercício.

Os problemas são resolvidos de forma que possam ser utilizados em grande variedade de microcomputadores, pessoais e profissionais, não se prendendo a nenhum especificamente, e tentando ser o mais geral possível, respeitadas as diferentes versões de linguagem BASIC.

MAIS UM LANÇAMENTO
DA

LIVROS TECNICOS E CIENTIFICOS EDITORA S.A.

ISBN: 85-216-0422-X