1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-2888-PRE/408		*Set., 1983*	☐ Interna ☒ Externa ☐ Restrita

4. Origem	Programa
DIN/DPD	*DENUME*

6. Palavras chaves - selecionadas pelo(s) autor(es)

PASCAL
COMPILADOR

7. C.D.U.: *681.322.OP*

8. Título *INPE-2888-PRE/408*

INTRODUÇÃO À LINGUAGEM PASCAL

10. Páginas: *48*

11. Última página: *46*

12. Revisada por

L. A. V. Dias

L.A. Vieira Dias

9. Autoria *Santiago Alves Tavares*

13. Autorizada por

Nelson de Jesus Parada

Nelson de Jesus Parada
Diretor

Assinatura responsável Santiago Alves Tavares

14. Resumo/Notas

Este trabalho é um resumo da linguagem Pascal.

15. Observações

Este trabalho será apresentado no 6º CNMAC, ITA, São José dos Campos, SP, no período de 26 a 30 de setembro de 1983.

INTRODUÇÃO
À
LINGUAGEM PASCAL

INTRODUÇÃO À LINGUAGEM PASCAL

Santiago Alves Tavares
Instituto de Pesquisas Espaciais - INPE
Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq
Caixa Postal 515 - 12200 - São José dos Campos - SP - Brasil

CAPÍTULO 1

A LINGUAGEM PASCAL

A linguagem Pascal é a primeira a incorporar, de um modo coerente, os conceitos de programação estruturada, definidos por Dahl et alii (1972). A linguagem Pascal foi desenvolvida por Niklaus Wirth na Eidgenossiche Technische Hochscule em Zurich. Foi derivada do Algol 60, sendo mais poderosa e de uso mais fácil. Esta linguagem foi desenvolvida com a finalidade de ensinar programação estruturada. Sendo uma linguagem pequena e de implementação fácil em microcomputadores onde a memória é limitada.

Tendo a linguagem Pascal se mostrado prática e poderosa, passou a ter seu uso difundido não só em micros, como em máquinas grandes, não só para ensino, como para o desenvolvimento de sistemas.

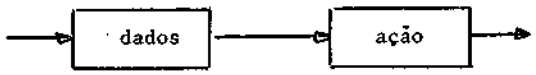
Mais detalhes sobre a linguagem Pascal podem ser vistos em Wirth (1976 e 1982) e Grogono (1978).

CAPÍTULO 2

CABEÇALHO DO PROGRAMA

2.1 - PROGRAMA

Um programa de computador é a esquematização de um algoritmo na forma adequada para que o computador possa entendê-lo e executá-lo. Um programa pode ser dividido em duas partes: descrição das ações a ser executadas e descrição dos dados a ser manipulados pelas ações. As ações são definidas pelos comandos (statements) e os dados pelas declarações e definições. Isto pode ser sintetizado como:

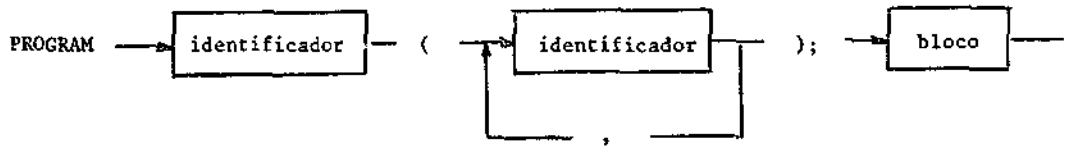


Em Pascal o programa contém um cabeçalho (heading) e um corpo chamado bloco. O cabeçalho dá um nome ao programa e apresenta a lista de seus parâmetros. O bloco contém seis partes, as quais são:

<declaração de rótulo>
<definição de constante>
<definição de tipo>
<declaração de variáveis>
<declaração de procedimentos e funções>
<comandos>

As cinco primeiras partes podem não existir no programa (vazio), a última tem existência obrigatória.

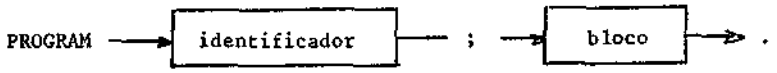
O diagrama de um programa é:



O primeiro identificador é um nome que é escolhido para o programa, por exemplo PROG1. O segundo são os identificadores de arquivo, por exemplo INPUT e OUTPUT. Então,

```
PROGRAM PROG1 (INPUT, OUTPUT);
```

No caso do Pascal, implantado no B6800, o cabeçalho apresenta-se da seguinte maneira:



Para executar o programa, o segundo identificador é definido no comando COMPILE (C) ou RUN (R), por exemplo:

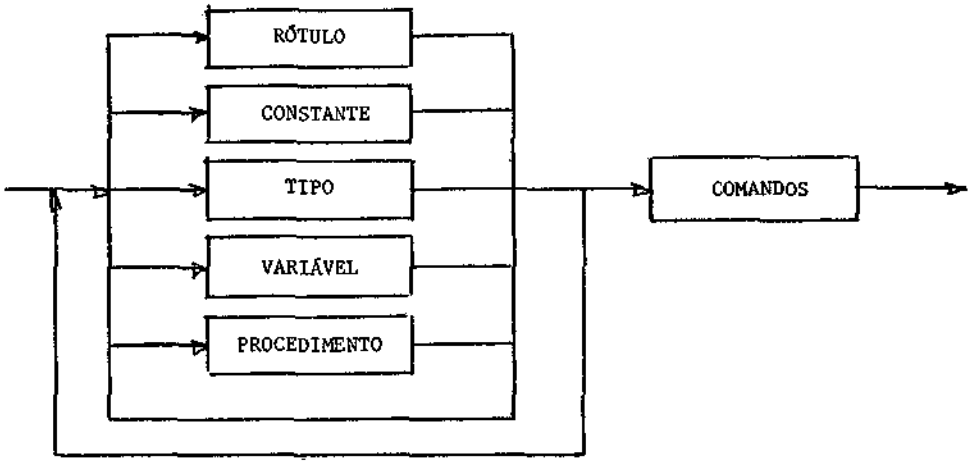
```
C; File INPUT (Kind = Remote),
   OUTPUT (Kind = remote)
```

ou

```
R; File INPUT (Kind = remote),
   output (Kind = remote)
```

Definir os arquivos de entrada e saída no comando de compilação é mais interessante, pois para executar o comando basta RUN.

O diagrama de bloco é apresentado a seguir. Cada trecho deste diagrama será explicado a medida que se for construindo os exemplos.



3.2 - NOTAÇÃO E VOCABULÁRIO

a) Vocabulário básico

O vocabulário básico consiste em símbolos compostos de:

letras,
dígitos,
símbolos especiais (operadores e delimitadores).

Os símbolos especiais são:

Símbolos, que podem ser caracteres ou par de caracteres

+	:	(
-	=	)
*	<>	]
/	<	[
:=	<=	{
.	>=	}
,	>	↑
;	.	..

ou palavras reservadas: que têm um significado fixo, não podendo ser usadas com outros fins, tais como identificadores. São elas:

and	end	nil	set
array	file	not	then
begin	for	of	to
case	function	or	type
const	get*	packed	until
div	if	procedure	var
do	in	program	while
downto	label	record	with
else	mod	repeat	

b) Comentários

São escritos entre chaves, podendo conter qualquer sequência de símbolos que não contenha chaves. Podem ser colocados entre dois identificadores, números ou símbolos especiais:

{ < qualquer sequência de símbolos não contendo " " > }

No caso de sistemas que não contenham chaves, o par de caracteres (* e *) é usado; por exemplo

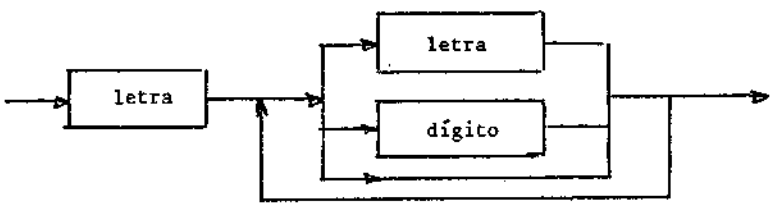
{isto é um comentário}

ou

(* isto é um comentário *)

c) Identificadores

São palavras denotando constantes, tipos, variáveis, rotinas e funções. Começam obrigatoriamente por uma letra. A sintaxe é



Embora o identificador possa ser longo, o compilador só reconhece os oito primeiros caracteres. Isto significa que

casadepedrabranca
e
casadepedraamarela

correspondem, para o compilador, a um só identificador:

casadepe

Há sistemas que reconhecem o identificador completo.

Exemplos de identificadores:

A A1 casa casa2 B45X

Não são permitidos: identificadores da forma

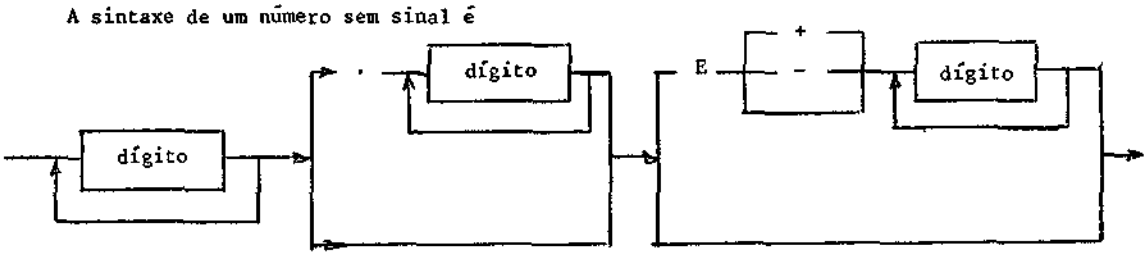
3A array casa.4 B45-X

Existem identificadores padrões predefinidos, tais como: SIN, COS, .. Como não são palavras delimitadoras (array, begin, ..) podem ser redefinidos num bloco qualquer.

Os identificadores podem ser divididos em: palavras reservadas, identificadores padrões e identificadores do programados.

d) Números

A sintaxe de um número sem sinal é



Exemplos:

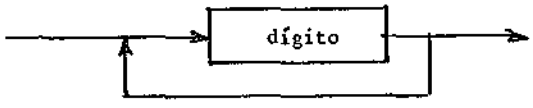
3 03 6347 0.6 5E-8

5E + 8 5 E 8 1 E 10 35.4E + 08

Não são aceitos como números:

3,45 xII .6 E10 5.E

O número inteiro é caso particular e sua sintaxe é:



Observações:

1 - Se um número contiver o ponto decimal no mínimo um dígito deve preceder e suceder o ponto. A letra E significa "10 elevado a".

2 - Brancos, fim de linha e comentários são considerados como separadores.

3 - Pode-se colocar um número qualquer de separadores entre dois símbolos consecutivos, com exceção de: identificadores, números e símbolos especiais.

4 - A primeira vez que um identificador aparece é numa declaração, na qual ele é definido. O identificador pode ser global, e vale em todo o programa, ou local, e vale em parte somente.

5 - Deve-se tomar cuidado com a escolha do identificador, pois isto reduz a quantidade de erros.

6 - Se um identificador vai ser usado muitas vezes, uma palavra que lembre o seu significado é mais interessante. Se vai ser usado em uma pequena parte do programa o uso de algumas letras é mais vantajoso.

7 - Deve-se evitar o uso de letras ou dígitos ambíguos como as letras o, q e o número zero. Se o identificador for "COMO" não há dúvida, mas se for "NO" fica-se na dúvida se é a letra o ou zero.

e) "Strings"

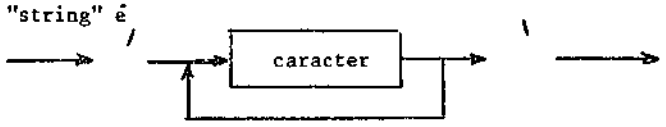
"Strings" são sequências de cadeias de caracteres dentro de aspas (uma só). Exemplos: 'a' 'casa' ',' 'begin' 'casa branca''. Pode ser uma sequência de vazios, como '------' onde o traço representa branco.

Para englobar a aspa basta escrevê-la duas vezes, por exemplo,

"lapis"

será escrito como 'lapis'

A sintaxe de "string" é



CAPÍTULO 3

BLOCO

3.1 - LITERAIS E CONSTANTES

Conceito de literal: em um cálculo, principalmente em projetos, há valores que são constantes, tais como valor de ¶, módulo de elasticidade E, dimensão de uma matriz M, número de páginas P, etc. Esses valores podem ser escritos no programa como:

```
circunferencia = 3.1415926535 * diametro
deslocamento   = q * a/(48 * inercia * 210 000)
```

O valor escrito no programa, do modo acima, é chamado literal. Pode-se associar um nome ao literal e ele passa a ser uma constante. Exemplos:

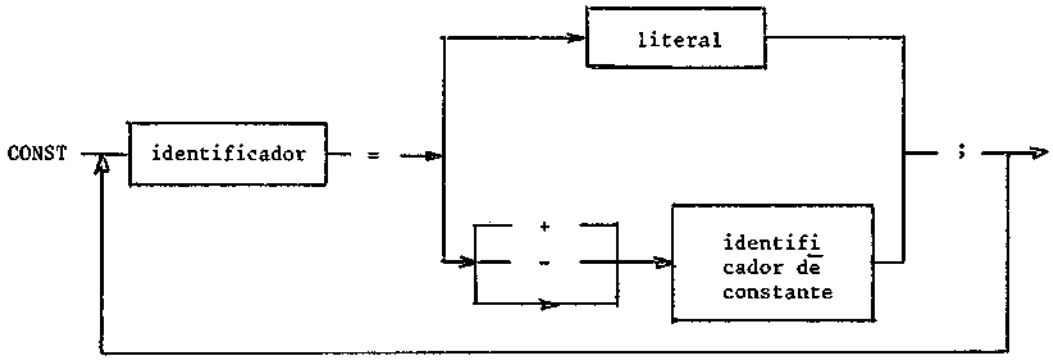
```
PI = 3.1415926535;
elasticidade = 210 000;
dimensao = 30;
pagina = 60;
```

Isto permite escrever o programa como

```
circunferencia = PI* diâmetro;
deslocamento = - q*a/(48 * inercia * elasticidade);
```

A principal vantagem de escrever o programa deste modo é poder trocar a constante com facilidade. Por exemplo, para trocar de material, isto é, de concreto E = 210 000 para madeira E = = 98 000, basta trocar o valor na declaração da constante. O mesmo pode ser feito para trocar a dimensão de uma matriz , o número de páginas, etc.

A sintaxe de uma constante é:

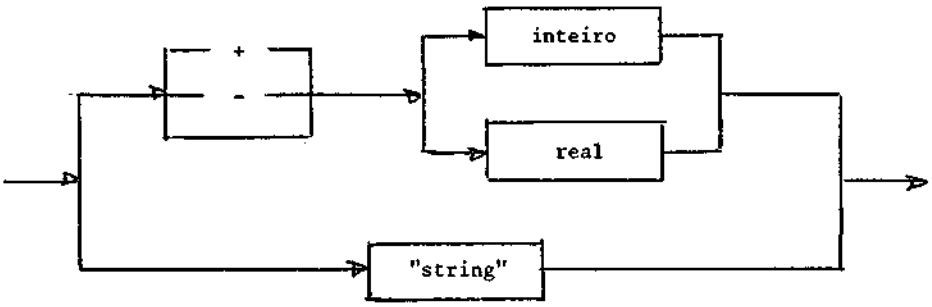


Para os exemplos anteriores a declaração é

```
CONST
   PI = 3.1415926535;
   elasticidade = 210 000;
   tensaomaxima = 1 200;
   tensaominima = -tensaomaxima;
```

As três primeiras declarações são identificadores de constante associados a literais (números); já a última, associa uma declaração de constante a um identificador de constante declarado previamente.

A sintaxe de literal é:



Exemplos

```
CONST
   Branco = '    ';
   data   = '15 de setembro, domingo';
   inercia = 4.75 E-04;
```

3.2 - DADOS E TIPOS

Sendo um programa uma sequência de regras para manusear os dados, estes são os elementos que definem a razão de ser do cálculo. Os dados podem ser constantes ou variáveis. Um dado constante tem valor fixo ao longo de todo o programa e um dado variável tem o seu valor modificado ao longo do programa.

O tipo de um dado define o conjunto de valores que uma variável pode assumir. Isto é, o tipo é o nome dado a um conjunto de variáveis. Cada variável deve pertencer somente a um conjunto de dados, ou seja, deve ser de um certo tipo, não podendo ser de dois simultaneamente. O tipo divide-se em padrão (que pode ser inteiro, real, booleano e caracter) ou definido pelo programador.

Os literais e constantes têm o seu tipo determinado pelo compilador, não sendo necessário declará-los. Já as variáveis devem ter o seu tipo declarado. A sintaxe é:

Exemplos:

```
VAR
  ordem, pagina, indice: INTEGER; comprimento, altura: REAL;
  fim, inicio: BOOLEAN; caracter: CHAR;
```

A seguir descreve-se cada um dos tipos.

A - TIPO INTEIRO

Exemplos de inteiros são 1, -45, 0, +378. Um computador pode somente representar um número finito de inteiros; assim o inteiro N pode ser representado se

$$-\text{max. int.} \leq N \leq \text{max. int.}$$

onde max. int. é o máximo inteiro que um determinado computador representa. Se durante o cálculo o número inteiro cair fora da faixa é acusado um erro em tempo de execução.

Os operadores para inteiros são:

* produto

DIV divide e trunca (isto é, o valor não é arredondado). Exemplo

5 DIV 2 = 2
3 DIV 4 = 0

MOD fornece o resto da divisão inteira
a MOD b = a - ((a div b) * b)

Exemplos

14 MOD 3 = 2

+ soma
- subtração

Observações

1) Todos os operadores são infixos, isto é, estão entre seus operandos. O operador - pode ser também usado como unário. Exemplo: -50

2) O operador / é usado para divisão real. Pode ser usado com operandos inteiros, mas o resultado será sempre real. Exemplos

```
2 DIV 3 = 0
2/3 = 0.6666667
5 DIV 1 = 5
5/1 = 5.0
```

3) Expressão inteira

É a combinação de variáveis inteiras e constantes com os operadores descritos. Exemplo:

Dada a declaração

```
CONST
   tamanholinha = 80;
VAR
   Num, cont, linha: integer;
```

as seguintes expressões inteiras são verdadeiras.

```
num + cont
num + linha DIV tamanholinha
cont - 30
linha MOD cont + 2
```

4) As operações * DIV MOD têm prioridade em relação a + - . Exemplos:

```
2 + 3 * 4 = 14
```

A alteração das prioridades de operadores é feita com parêntesis

```
(2 + 3) * 4 = 20
```

A divisão fornece sempre resultado inteiro, sendo o resto ignorado.

```
5 DIV 2 = 2
3 DIV 4 = 0
```

O resultado de uma divisão onde o dividendo é menor que o divisor é sempre zero. O resto de uma divisão inteira é obtido com o operador MOD

```
14 MOD 3 = 2
```

5) O resultado de uma expressão inteira pode ser colocado numa variável inteira ou real. Exemplo:

```
VAR
  V1, V2: integer;
  I1: real;
```

Pode-se escrever

```
I1 = V1 * V2
```

O programa calcula a expressão V1 * V2 como inteira, converte-a para real e armazena-a em I1. Esta operação é chamada "conversão implícita de tipo".

6) O comando de leitura

READ (V1)

lê o número, transforma-o em inteiro e armazena-o em V1. Os caracteres brancos são ignorados na leitura. O comando para escrever

WRITE (numero : campo)

onde número e campo são expressões aritméticas inteiras.
O número ao ser escrito é colocado mais à direita no campo. Exemplo: os comandos

```
numero = 173;
WRITE (numero: 6);
```

escrevem

bbb173
onde b = branco.

Se o número de algarismos a ser escrito é maior que o campo, o número é escrito com um campo adequado. Exemplo:

```
PROGRAM FORMATO (OUTPUT);
CONST
   multiplicador = 10;
   valorfinal = 1000000
VAR
   potencia: integer;
BEGIN
   potencia:= multiplicador;
   REPEAT
       WRITELN ('*', potencia, '*');
       potencia:= potencia * multiplicador
   UNTIL
       potencia >= valor final
   END.
```

A saída é

```
*10*
*100*
*1000*
*10000*
*100000*
*1000000*
```

No caso do campo não ser fornecido o programa adota um valor.

7) As funções de argumento inteiro e as funções que fornecem valores inteiros estão apresentadas na tabela a seguir.

ARGUMENTO / VALOR	INTEIRO	REAL	BOOLEANO	CARACTER	ARQUIVO
inteiro	PRED SUCC ABS SQR	TRUNC ROUND	ORD	ORD	
real	SIN COS ARCTAN IN EXP SQRT	ABS SQR SIN COS ARCTAN IN EXP SQRT			
booleano	ODD		PRED SUCC		EOF EOLN
caracter	CHAR			PRED SUCC	

B - TIPO REAL

As variáveis do tipo real são usadas num programa do mesmo modo que em matemática. Não se deve esquecer que o computador representa somente um subconjunto finito dos números reais. Isto, em geral, não afeta o resultado dos cálculos. Em alguns casos pode-se ter grandes problemas de imprecisão.

Exemplos de literais reais:

12.7 1.0 0.0007 9.10956E-28

Observações

1) A ordenação das variáveis reais no computador é a natural. Se "a" e "b" são dois números reais e x e y duas variáveis reais, pode-se afirmar que se a = b, para o computador x=a e y=b, então tem-se x=y. O mesmo não acontece no caso a < b, pois se "a" e "b" forem números próximos, o computador não diferencia, e não se pode afirmar que se a < b ⇒ x < y para o computador, o que se pode dizer é: se a < b ⇒ x ≤ y para o computador.

2) Os operadores com números reais são

+ - * /

Observar que as expressões são avaliadas da esquerda para a direita, logo

a / b * c

é calculado como

(a / b) * c

3) Uma expressão pode conter inteiros e reais. Se o operando de um dos operadores +, -, * for real, o operando inteiro é convertido automaticamente para real antes de executar a operação. Exemplo:

(6 + 4) * (1 + 0,1)

onde são executados os parênteses em primeiro lugar:

6 + 4 = 10 - operação com inteiros
1 + 0.1 = 1.1 - operação com inteiro e real, dando resultado real onde 1 foi transformado em 1.0;
10 * 1.1 = 10.0 * 1.1 = 11.0

O operador / transforma os operandos em real, se não o forem, fornecendo um resultado real. Não é possível armazenar o resultado de uma expressão real numa variável inteira.

4) Se o dado for uma variável real, então

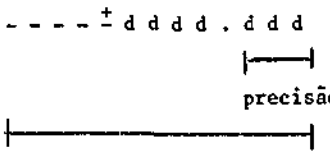
READ (d1)

lê o número e armazena-o como real na variável d1.
O comando

WRITE (numero: campo: precisão)

onde:

'campo' é o número de caracteres a ser escrito e 'precisão' é o número de dígitos após o ponto decimal, ou seja:



Se for escrito

WRITE (número : campo)

omitindo-se 'precisão', o número será escrito em notação científica:

- - - ± 0 . d d d d E ± d d

Se for escrito

WRITE (número)

o número será escrito em notação científica com o campo especificado pelo compilador.

É conveniente declarar o campo e a precisão como constantes. Exemplo

```
CONST
   precisão = 6;
   campo    = 16;
VAR
   número   = real;
  :
  :
WRITELN (número : campo : precisão);
```

pois isto facilita a mudança de formato.

C - TIPO BOOLEANO

As variáveis booleanas podem ter um de dois valores, representados por verdadeiro (TRUE) ou falso (FALSE).

Há três operadores booleanos: AND, OR e NOT.

Exemplos:

```
VAR
  V1, V2, V3 : boolean;
```

As seguintes expressões possuem valores booleanos

V_1 AND V_2 OR V_3

NOT V_1 OR V_2

V_1 AND (V_2 OR V_3)

O operador NOT é sempre aplicado antes. A segunda expressão equivale a

(NOT V_1) OR V_2

O operador AND é sempre aplicado antes do OR. A primeira expressão é equivalente a

(V_1 AND V_2) OR V_3

Esta ordem pode ser modificada com o uso de parêntes; como foi feito na terceira expressão.

O resultado das operações booleanas estão no quadro abaixo

V_1	V_2	NOT V_1	V_1 AND V_2	V_1 OR V_2
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

Os operadores relacionais ou de comparação são

< menor que
≤ menor que ou igual
= igual a
≠ diferente
≥ maior que ou igual a
> maior que

e fornecem valores boleanos quando usados com expressões de qualquer tipo, para as quais tenha sido definida uma ordenação. Exemplo:

2 < 3 = True
3 < - 5 = False
(V_1 = V_2) AND (L_1 ≥ H) AND V_3

É necessário o uso dos parênteses quando expressões de comparação são separadas por operadores booleanos. A relação

$V_1 \leq V_2 \leq V_3$

deve ser escrita como

$(V_1 \leq V_2)$ AND $(V_2 \leq V_3)$

Não se deve comparar igualdade de números reais, pois devido ao modo com que os números são representados há propagação de erro. No lugar de

a = b

deve-se fazer

ABS (a - b) ≤ epsilon

As funções EOLN (f_1), fim de linha e EOF (f_2) fim de arquivo têm valores booleanos. O argumento é o nome de um arquivo (ver capítulo sobre arquivos). Se o arquivo ao qual se refere a função é INPUT pode-se omitir o argumento. Exemplo

```
IF EOF
    THEN WRITE ('fim de arquivo')
    ELSE READ (ch)
```

A função ODD (X) tem argumento X inteiro. O resultado é TRUE se X for ímpar, e FALSE se X for par.

Se V_1 é uma variável booleana, o comando

WRITE (V_1)

escreve 'TRUE' ou 'FALSE'. O comando WRITE (V_1 : campo) irá escrever ou TRUE precedido de (campo-4) brancos ou FALSE precedido de (campo-5) brancos, desde que o campo seja suficiente para isto.

O comando READ não aceita argumento booleano.

D - TIPO CARACTER

Uma variável do tipo CHAR tem um valor que é um caracter de impressão. O branco é um caracter, mas CR (carriage return), LF (line feed) e outros caracteres de controle não o são. O literal CHAR é um caracter entre aspas simples. Exemplos

```
'a' representa a letra a
' ' representa um branco
' '' ' representa o caracter ' (aspa)
```

Os caracteres são numerados internamente no computador, o que permite fazer

'a' < 'b' < 'c' < ... < 'z'

e

'0' < '1' < ... '9'

Os operadores de comparação podem ser usados com caracteres, dando um resultado booleano. Nenhum outro operador pode ser usado com caracteres.

Sendo os caracteres ordenados, a expressão

casa < caso

é válida pois, 'a' vem antes de 'o'.

Do mesmo modo vale

ORD (casa)<ORD (caso)

Para a maioria dos computadores

ORD ('o') ≠ 0

o que permite transformar um caracter numérico em número. Mas pode ser feito

ORD ('7') - ORD ('0') = 7

o que permite a conversão de caracter em número.

A função CHR faz o inverso de ORD. Exemplo:

```
CHR (dígito + ORD ('o')) = 'digito'
CHR (3 + ORD ('0')) = '3'
```

Caracteres podem ser lidos ou escritos

```
C1, C2 : char;
  ...
READ  (C );
WRITE (C );
```

O comando

WRITE (C_1 : campo)

onde campo é uma expressão inteira escreve (campo-1) brancos e depois o caracter C_1. Um caso particular é

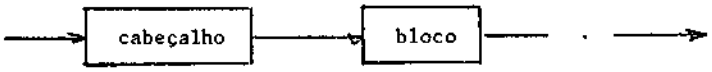
WRITE (' ' : campo)

que escreve um número de brancos igual a campo.

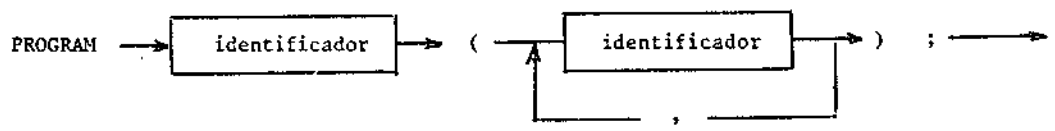
3.3 - CONSTRUÇÃO DE PROGRAMAS

A - DECLARAÇÕES

Um programa consiste em um cabeçalho e um bloco, finalizando por um ponto



O cabeçalho é

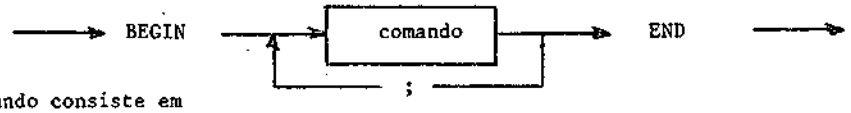


e o bloco

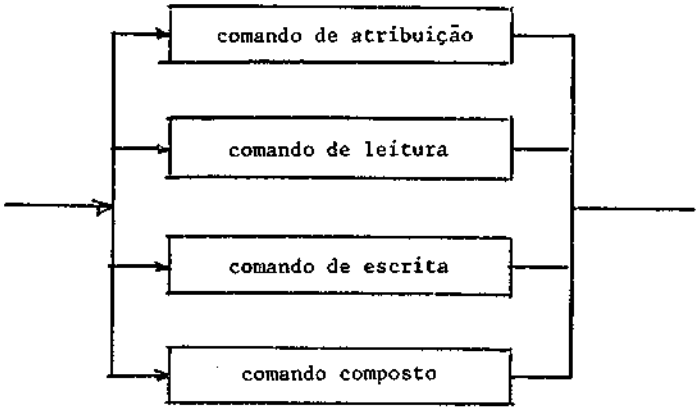
As declarações no bloco são opcionais. Se houver constantes no programa, sua declaração deve vir antes das de variáveis.

B - COMANDOS

O comando composto é uma sequência de comandos, começa por BEGIN e termina por END, separados por ponto e vírgula. Não há necessidade de ponto e vírgula entre o último comando e o END.



Um comando consiste em



Observe-se que um comando pode ser um comando composto.

Um comando de atribuição é do tipo

variável := expressão

O comando de leitura para ler três variáveis pode ser:

```
READ (V1)
READ (V2)
READ (V3)
```

ou, no mesmo comando,

```
READ (V1, V2, V3)
```

O comando para escrita aceita "strings" constantes e literais. Exemplo:

```
CONST
   V1 = 'o lápis é azul';
...
WRITE (V1);
```

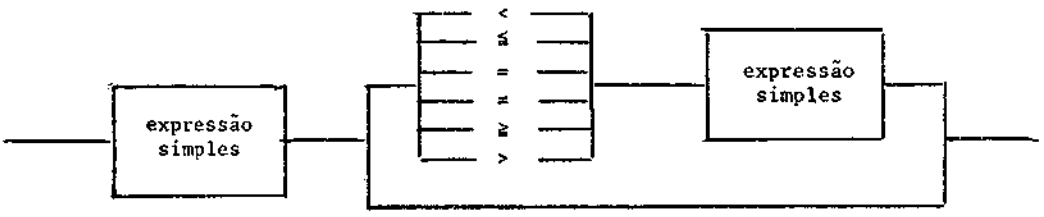
ou, na forma de literal,

```
WRITE ('o lápis é azul')
```

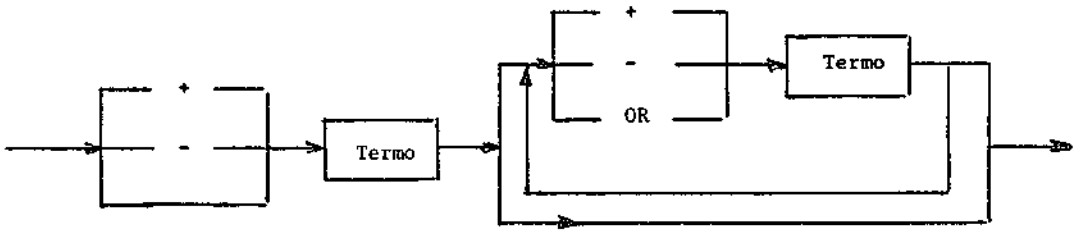
O comando WRITELN envia o retorno do carro para o arquivo ("file") de saída.

C - EXPRESSÕES

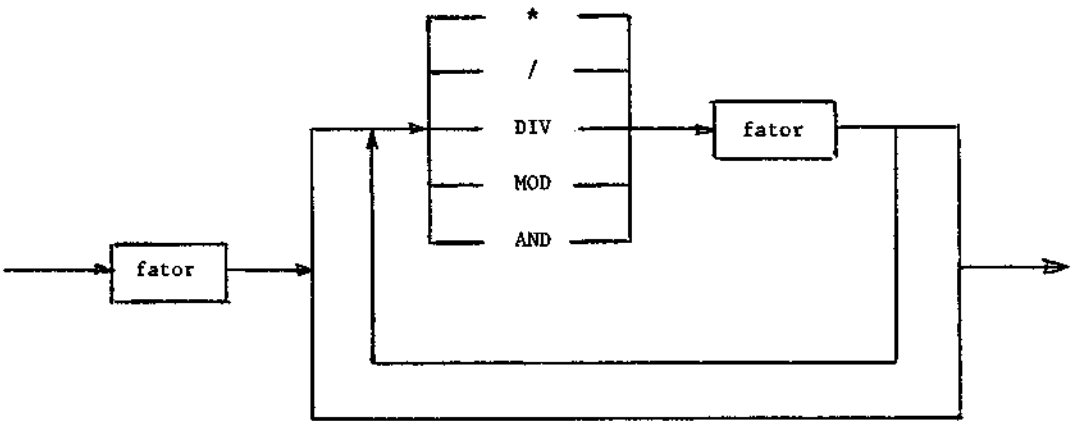
A sintaxe de uma expressão é



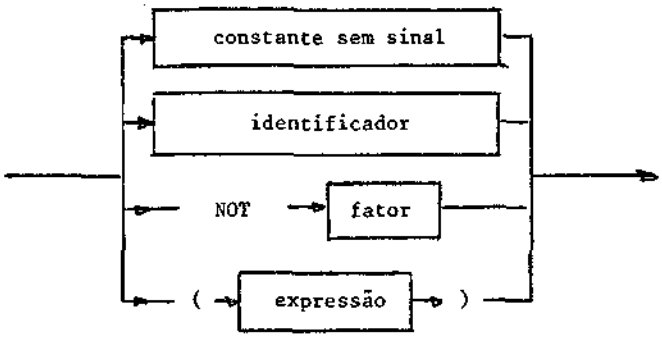
Uma expressão simples é



Um termo é



Um fator é



Exemplos:

a) Fator

A , 5 , NOT A , NOT 5 , (A ≤ 5)

b) Termo

A * B , A/5 , B DIV C

Deve-se observar que os fatores usados nos termos devem ser compatíveis com os operadores. Assim "NOT A" é do tipo booleano, e não seria possível como fator

B * NOT A

mas

B AND NOT A é possível

c) Expressão simples

A
A + B * C onde os termos são A e B e C
A OR B AND NOT C
A + B * C - D

d) Expressão

A + B * C ≤ D
(A1 ≤ A2) AND (A2 ≤ A3)

Neste caso A1 ≤ A2 e A2 ≤ A3 são fatores, o que é possível pois a definição de fator inclui uma expressão dentro de parênteses.

D - BRANCOS E COMENTÁRIOS

Os brancos podem ocorrer em qualquer lugar do programa, não podendo estar em identificadores, palavras reservadas ou símbolos compostos. Estes são

:= ..

ou ≤ ≠ ≥ { } que são representados por <= <> >= (* *)

Vários brancos são equivalentes a um só. Entre duas linhas de um programa há um branco implícito, o que proíbe a separação em duas linhas de identificadores, símbolos compostos ou palavras reservadas.

O comentário é da forma

{ cadeia de caracteres }

onde, na cadeia, não deve aparecer o símbolo }.

Um comentário é equivalente a um branco, portanto pode ser colocado em qualquer lugar onde possa existir um branco.

CAPÍTULO 4

TIPOS

O Pascal possui dois conjuntos de tipos.

- tipos padrões (já definidos no compilador),
- tipos definidos pelo programador.

Os tipos padrões são inteiro, real, booleano e caracter, que já foram vistos.

4.1 - ESCALARES

A declaração de um tipo escalar é uma lista de valores que as variáveis deste tipo podem assumir, isto é

TYPE T = (c_1, c_2,..., c_n)

Exemplificando:

```
TYPE unidades = (kilo, metro, segundo);
```

estabelece que as variáveis do tipo unidade só podem ter um destes três valores. As variáveis são declaradas como no caso do tipo padrão. Exemplo:

```
VAR
  lado, peso: unidades;
```

Com isto, as seguintes atribuições são válidas

```
lado := metro
peso := kilo
```

mas não é aceito

```
lado := centímetro
```

As declarações de tipo e variável podem ser englobadas em uma única, como

```
VAR
lado, peso: (kilo, metro, segundo);
```

apesar de nem sempre ser conveniente proceder deste modo, pois pode tornar o programa menos claro.

Um valor não pode aparecer em mais de uma declaração, como

```
TYPE
  frutas = (banana, laranja, limao, mamao);
  citricos = (laranja, tangerina)
```

onde laranja aparece em duas declarações. Não se pode misturar atribuições. Exemplo, tendo sido declarado

```
VAR
  V1, V2: frutas;
  X1, X2: cítricos;
```

não se pode atribuir

```
X1 := banana;
V1 := tangerina;
```

Os únicos operadores permitidos com variáveis escalares são os relacionais, e o resultado da operação é um valor booleano. Os escalares são automaticamente ordenados na sequência que aparecem na declaração tipo, donde serem válidas as expressões

```
kilo < metro
laranja > banana
```

As funções PRED e SUCC, sendo definidas para argumentos escalares, podem ser usadas e, neste caso, fornecem o antecessor ou sucessor na declaração de tipo. Exemplo:

```
SUCC (kilo) = metro
PRED (mamao) = limão
```

O primeiro membro da lista não tem antecessor nem o último tem sucessor.

A função ORD tem argumento escalar e fornece um valor inteiro correspondente à ordem do valor escalar na lista. O primeiro da lista tem número zero. Exemplo:

```
ORD (kilo) = 0
ORD (mamao) = 3
```

Não se pode ler ou escrever um escalar. Não é permitido. Se for usado o compilador acusará érro. Exemplo:

```
V1 := limão;
WRITE (V1)
```

Mas pode ser feito

```
WRITE (ORD (V1))
```

e neste caso será escrito o número 2, correspondente à ordem de limão na lista.

4.2 - SUBINTERVALO

É frequente o caso no qual a variável assume valores de um certo tipo somente dentro de um intervalo. Isto é feito definindo que a variável pertence a um subconjunto, cujo tipo é definido como

```
TYPET = min .. max
```

onde min e max são os limites do intervalo. Exemplos:

```
TYPE   ano = 1900 .. 1999
TYPE   letra = 'A' .. 'Z'
TYPE   dígito = '0' .. '9'
TYPE   verão = dezembro .. março
```

onde verão é um subintervalo dos meses do ano.

A declaração de uma variável é

```
VAR  y : ano; L:  letra
     m : verão;
```

assim as seguintes atribuições são válidas

```
y := 1983
L := 'S'
m := janeiro
```

e não são permitidas

```
y := 1500
L := '9'
m := abril
```

A declaração de subintervalo pode fazer parte da declaração de variável. Exemplo:

```
VAR
  V1, V2: 1 .. 20;
  I1, I2: 'i' .. 'n';
```

mas é aconselhável declará-las separadamente.

Podem-se misturar diferentes subintervalos do mesmo tipo

```
VAR
  V1, V2: 1 .. 10;
  X1, X2: 0 .. 100;
  I1, I2: Integer
```

4.3 - CONJUNTOS

Um conjunto é uma coleção de objetos do mesmo tipo. Seja o tipo escalar

```
TYPE
ingredientes = (maça, morango, banana, abacaxi, manga, mamao)
```

Pode-se declarar o conjunto tipo

```
TYPE
sobremesa = SET OF ingredientes;
```

As variáveis desse tipo são declaradas do modo usual, isto é

```
VAR
suco, salada, sorvete: sobremesa;
```

O tipo sobremesa é o conjunto tipo associado ao tipo ingredientes; ou, de modo contrário, ingrediente é o tipo base do tipo sobremesa. A declaração é de forma

TYPE T = SET OF T_o

onde os valores possíveis da variável x do tipo T são os conjuntos de elementos de T_o

Se T_o = (1, 2, 3) o conjunto definido pelo SET OF é

T = { { }, {1}, {2}, {1,2}, {1,3}, {2,3}, {1,2,3}}

que é o conjunto potência de T_o.

O valor de uma constante ou variável do tipo T é representado pela lista de seus elementos, colocado entre colchetes. Se for declarado

VAR T : T;

então pode ser feita a seguinte atribuição

T := [1,3]; ou TI := [ ];

que é o conjunto vazio.

Se os elementos de um conjunto são valores consecutivos do tipo base T_o, pode-se especificar somente o primeiro e o último

[maçã .. abacaxi]
[1 .. 3]

As operações com conjuntos são

a) União

A união de dois conjuntos é o conjunto que contém os elementos dos dois conjuntos componentes. O operador é +. Existem versões onde a união é feita com OR. Exemplo:

[1,2] + [1,4,5] = [1,2,4,5]

b) Interseção

A interseção de dois conjuntos é o conjunto que contém somente os elementos que aparecem nos dois conjuntos componentes. O operador de interseção é *. Existem versões onde a interseção é feita por AND.

[1,2] * [1,3,4] = [1]

[banana, maça] * [maça, uva, pera] = [maça]

c) Diferença

A diferença de dois conjuntos é o conjunto que contém todos os elementos do primeiro conjunto que não pertencem ao segundo. O operador de diferença é - (sinal menos). Exemplo:

{1,2,3,4} - {1,4,5} = {2,3}

d) Pertence

Para testar se um elemento pertence a um conjunto usa-se o operador IN (palavra reservada em Pascal). Exemplo:

E1 IN [conjunto]

Se E1 pertence ao conjunto o resultado da operação é TRUE, se não pertence é FALSE. Exemplo:

banana IN [laranja, banana, mamão]

tem como resultado TRUE.

A prioridade dos operadores é

* + - IN

A seguir estão apresentados alguns exemplos de operações e o equivalente, com parênteses, para indicar as prioridades de execução:

```
r * s + t = (r * s) + t
r - s * t = r - (s * t)
r - s + t = (r - s) + t
x IN s + t = x IN (s + t)
```

Os operadores relacionais podem ser usados para comparar conjuntos

= igualdade entre conjuntos
≠ desigualdade entre conjuntos
≤ está contido em
≥ contém

Exemplos

[1,2,3] = [2,3,1]

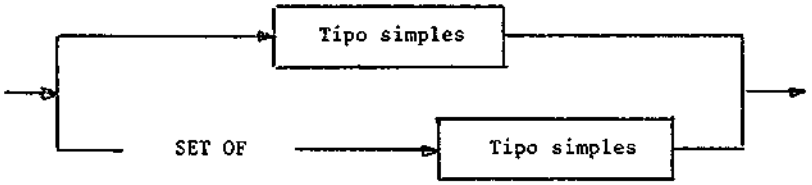
[1,2] ≠ [4,5,6]

[1,2] ≤ [1,2,4,5]

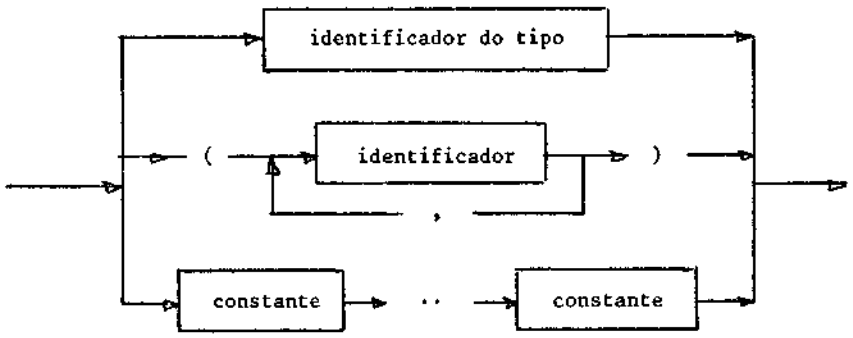
[1 .. 20] ≥ [5 .. 10]

Observe-se que um conjunto x contém um conjunto y se cada elemento de y é também um elemento de x.

A sintaxe de tipo é



e a de tipo simples é



4.4 - MATRIZES

Uma matriz coluna ou linha ("array") é uma estrutura homogênea, onde todos os componentes são do mesmo tipo, chamado tipo base. O "array" é uma estrutura de acesso aleatório. Para definir os elementos de um "array" usa-se individualmente um índice de um certo tipo. A definição é

TYPE T = ARRAY [I] OF T_o

onde T_o é o tipo base, T é o tipo do "array" e I é o tipo do índice. Exemplos

```
TYPE matrizlinha  = ARRAY [1 .. 5] OF real;
TYPE alfa = ARRAY [1 .. 10] OF char;
```

O índice pode ser declarado como um tipo escalar, por exemplo:

```
TYPE
   coordenadas = (x, y, z);
   vetor = ARRAY [coordenadas] OF real;
```

onde o tipo base é 'REAL' e o do ARRAY é 'vetor'. As variáveis são declaradas como

```
VAR
  s, t : coordenada;
  u, v : vetor;
```

A variável do tipo vetor (u, v) tem três componentes (são três índices: x, y e z) que são

v [x] v [y] v [z]

Na notação da cinemática as componentes do vetor velocidade $\vec{v}$ costumam ser expressas por

v_x v_y v_z

Por analogia os índices do tipo vetor v são chamados subescritos. A norma desse vetor é

```
norma := SQR (v[x]) + SQR (v[y])+ SQR (v[z])
```

pois os valores de v [x], v [y] e v [z] são reais, conforme declaração de tipo. Esta mesma norma pode ser calculada por:

```
BEGIN
   Norma := 0;
   For s := x TO z DO
   norma := norma + SQR (v[s])
END
```

A variável s sendo do tipo coordenada só pode assumir os valores x, y e z. Sendo esta a ordem de declaração, o comando FOR executa nesta sequência.

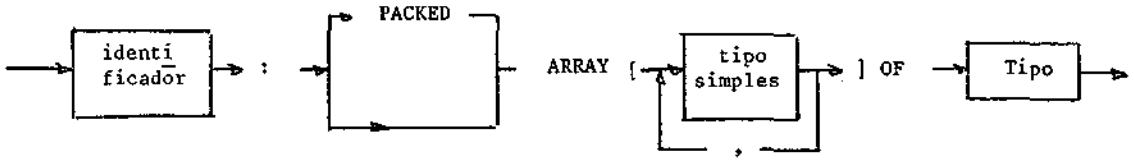
O valor de um ARRAY pode ser atribuído a outro do mesmo tipo, isto é

```
u := v
```

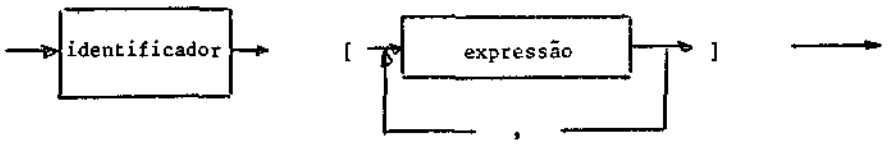
onde u e v são do tipo vetor. Esta atribuição equivale a

```
u [x] := v [x]    u [y] := v [y]    u [z] := v [z] .
```

A sintaxe do "array" é



e a dos componentes de um "array" é



O tipo de uma variável ARRAY pode ser declarado na declaração de variável

```
VAR A: ARRAY [1 .. N + 1] OF T;
```

MATRIZES MULTIDIMENSIONAIS

O tipo base de um ARRAY pode ser outro ARRAY. Nestas declarações o tipo base da matriz é coluna. Exemplo:

```
CONST
    N = 10;
TYPE
    indice = 1 .. N;
    coluna = ARRAY [indice] OF real;
    matriz = ARRAY [indice] OF coluna;
```

A declaração de coluna pode ser englobada na de matriz. Exemplo:

```
TYPE
   matriz = ARRAY [indice] OF
            ARRAY [indice] OF real;
```

ou colocado de modo simplificado como

```
TYPE
   matriz = ARRAY [indice, indice] OF real;
```

MATRIZES COMPACTADAS (packed arrays)

Os componentes de um ARRAY são armazenados em palavras consecutivas na memória do computador. Este é um modo eficiente de armazenar inteiros e reais, pois necessitam de uma palavra completa ou mais.

Já para variáveis de outros tipos este modo de armazenamento não é prático por gastar espaço inutilmente. Estes espaços podem ser eliminados pelo armazenamento de vários componentes do ARRAY numa só palavra. Exemplo:

```
VAR
  longo : array [1 .. 1000] OF char;
```

ocupa 1000 palavras da memória. Já

```
VAR
  longo : PACKED ARRAY [1 .. 1000] OF char;
```

ocupa 100 palavras de memória num CDC série 6000 (que armazena 10 caracteres por palavra) e 250 palavras num IBM 360/370 (que armazena 4 caracteres por palavra).

4.5 - RECORDS

Um RECORD, de modo análogo ao ARRAY, é formado por vários elementos, diferindo nos seguintes pontos: cada elemento pode ter tipo diferente dos outros, sendo acessados por nome e não por subscrito (índice). Exemplos:

- números complexos compostos de dois números reais;
- coordenadas de um ponto no R^N composto de N números reais;
- descrição de pessoas por seu nome, data de nascimento, sexo e estado civil.

Em matemática, um tipo composto, como os exemplificados, é chamado produto cartesiano de seus tipos constituintes. Um tipo RECORD é definido como:

```
TYPE T = RECORD S1 : T1;
                S2 : T1;
                ...
                SN : TN
         END
```

Exemplos:

```
TYPE complexo = RECORD re : real;
                       im : real
                END;

TYPE data = RECORD dia : 1 .. 31;
                   mes : 1 .. 12;
                   ano : 1 .. 2000
            END;

TYPE funcionario = RECORD
       sobrenome = alfa;
            nome = alfa;
      nascimento = data;
            sexo = (masculino, feminino);
     estadocivil = (solteiro, casado, viuvo, divorciado)
            END;
```

Seja a descrição de um planeta com as seguintes informações

nome
visível ou não a olho nu
diâmetro
raio médio da órbita

Pode-se usar um "string" para o nome; uma variável booleana para visível ou não e variáveis reais para o resto. Exemplo:

```
TYPE planeta = RECORD
     nome : PACKED ARRAY [1 .. 10] OF char;
     VISÍVEL : boolean;
     diâmetro, raiomed : real
END;
```

As variáveis deste tipo são declaradas do modo usual

```
VAR
   interior, exterior: planeta;
```

Dada uma variável x : T sua i-ésima componente é indicada por

x.s;

assim:

interior . nome;

é o nome de um planeta interior, e

exterior . diâmetro;

é o diâmetro de um planeta exterior.

Esses nomes são chamados seletor do RECORD e usados num programa como se fossem variáveis do mesmo tipo. Podem-se atribuir valores como

$x . s_i := x_i$

onde x_i é um valor (expressão) do tipo T_i. Portanto,

```
interior . nome     := 'venus'
interior . visivel  := true;
interior . diametro := 12104; {km}
interior . raiomed  := 108.2;
exterior . nome     := 'netuno';
exterior . visivel  := false;
exterior . diametro := 49500;
exterior . raiomed  := 4496.6
```

Podem-se combinar RECORDS com ARRAYS. Exemplo:

```
CONST
   longe = 10
TYPE
   planeta = RECORD
      nome : PACKED ARRAY [1 .. 10] OF char;
      visível : bolean;
      diâmetro, raiomed : real
END;
VAR
  sistema solar : ARRAY [1 .. longe] OF planeta;
  numeroplaneta : 1 .. longe;
```

Com estas declarações, podem ser executados comandos como

```
FOR numeroplaneta := 1 TO longe DO
IF sistema solar [numeroplaneta]. raiomed < 4000
THEN sistema solar [numeroplaneta]. VISIVEL = TRUE
ELSE sistema solar [numeroplaneta]. VISIVEL := FALSE;
```

Observações:

1 - Os nomes dos componentes devem ser únicos dentro do RECORD. A palavra 'nome' não pode ser reusada dentro do record 'planeta'. Mas 'nome' pode ser usado como uma variável ou componente de um outro RECORD.

2 - Não há operadores que possam utilizar os RECORDS como operandos. Não há ordenação associada aos "records".

3 - O valor de um RECORD pode ser atribuído a um outro "record". Assim, a atribuição

```
interior := exterior
```

é equivalente às seguintes atribuições:

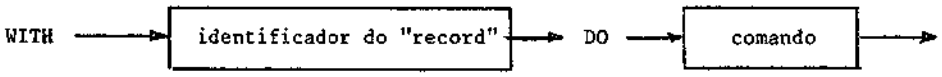
```
interior . nome     := exterior . nome
interior . vísivel  := exterior . visível
interior . diâmetro := exterior . diâmetro
interior . raiomed  := exterior . raiomed
```

Comando WITH

Frequentemente é necessário acessar o mesmo componente de um RECORD, ou diferentes componentes do mesmo RECORD várias vezes. Exemplo:

```
FOR  numeroplaneta := 1 TO longe DO
WITH  sistemasolar [numeroplaneta] DO
   VISIVEL := raiomed < 4000;
```

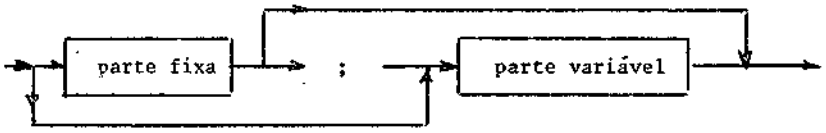
A forma do comando WITH é



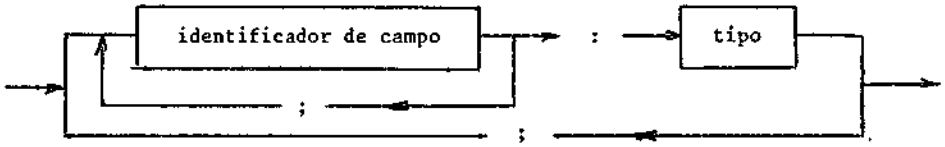
O comando WITH não deve ser usado indistintamente, pois pode haver ambiguidade.

A sintaxe da declaração "record" é

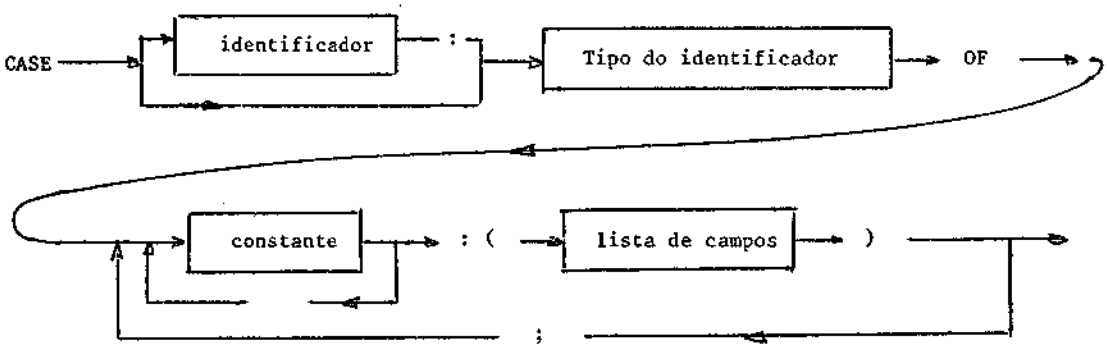
A lista dos campos é



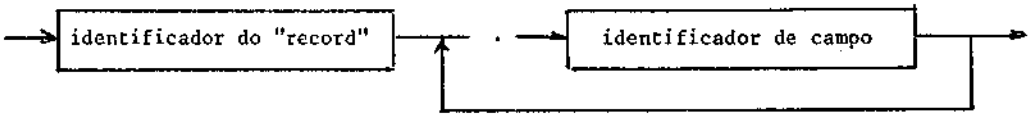
A parte fixa é



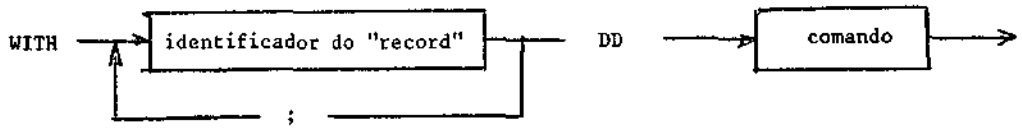
A parte variável é



A sintaxe de um elemento de "record" é



A sintaxe do comando WITH é



Exemplo de programa:

```
PROGRAM ESFERA (input, output);
   CONST
      PI = 3.1415926535;
      largura = 10;
      precisão = 4;
   VAR
      raio, area, volume: real;
   BEGIN
      READ (raio);
      area := 4*PI*SQR (raio);
      volume := raio*area/3;
      WRITELN ('medidas da esfera');
      WRITELN ('raio = ', raio : largura : precisão);
      WRITELN ('volume = ' volume : largura : precisão);
   END. {esfera}
```

Entrada 10

Saída

```
   Medidas da esfera:
   Raio     =      10.0000
   Area           1256.6372
   Volume   =    4188.7906
```

CAPÍTULO 5

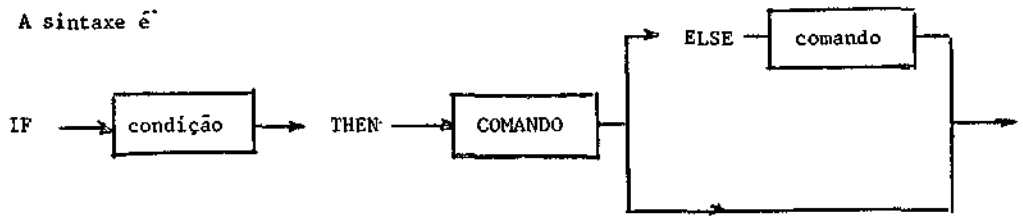
COMANDOS DE DECISÃO

5.1 - COMANDO IF

a) Comandos Simples com IF

O comando IF toma a decisão entre dois caminhos a escolher.

A sintaxe é



Exemplos:

```
   Program quadrado (input, output);
VAR
   numero, raiz : inteiro;
   lado, area : real;
```

```
BEGIN
     READ (numero, raiz, lado, area);

     IF numero < raiz - 1

        THEN numero := numero + 1
        ELSE numero := 0;
WRITELN (numero)
IF area >= 0
     THEN lado := SQRT (area)
     ELSE
       BEGIN
       lado := 0;
       WRITELN ('area negativa')
       END;
WRITELN ('lado = ', lado);
END.
```

No caso de haver comandos compostos, o IF é da forma:

```
IF condição
   THEN
      BEGIN
         comandos
      END
  ELSE
      BEGIN
         comandos
      END
```

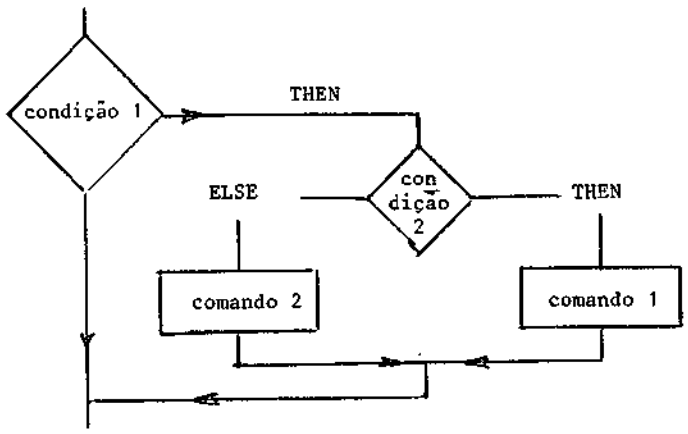
b) Comandos Compostos com IF

Os comandos após o THEN e o ELSE podem ser outro IF. Neste caso diz-se que o comando é um IF composto.

O ELSE pertence sempre ao IF mais próximo para o qual não há a condição ELSE. Exemplo:

```
IF condição 1
   THEN
      IF condição 2
         THEN
           . comando-1
         ELSE
            comando 2;
```

Neste caso o IF mais próximo do ELSE é o segundo. O ponto e vírgula termina o primeiro IF no qual está englobado o segundo. O diagrama é:



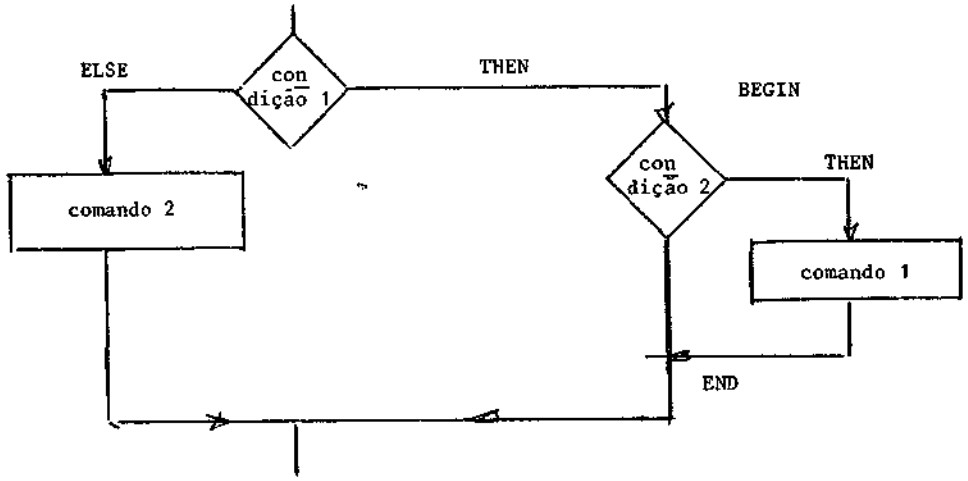
Já para o programa:

```
IF condição 1
   THEN
      Begin
      IF condição 2
         THEN
            comando 1
      END
    ELSE
      comando 2;
```

A decisão é feita de modo diferente. Observe-se que no comando 1 não há necessidade de ponto e vírgula por vir antes do END. O BEGIN-END engloba o segundo IF no THEN do primeiro IF como comando composto, o que torna o programa da forma apresentada no exemplo a seguir:

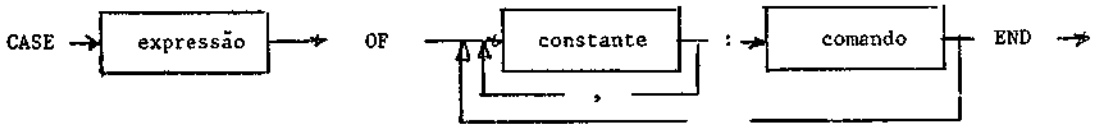
```
IF condição 1
   THEN
      comando composto
   ELSE
      comando 2;
```

O diagrama de bloco para este caso é:



5.2 - COMANDO CASE

O comando CASE toma a decisão entre vários caminhos a escolher. A sintaxe é:



Para exemplificar:

```
TYPE
   mes = (jan, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez);
   ano = 1900 .. 2000;.
   Tamanhomes = 28 .. 31;
VAR
   A1 : ano;
   M1 : mes;
   T1 : tamanhomes
....
   case mm OF
      jan, mar, jul, ago, out, dez : T1 := 31;
      abr, jun, set, nov: T1 := 30;

fev:
   IF (A1 MOD 4 = 0) AND
      (A1 MOD 100 = 0) THEN T1 := 29
      ELSE T1 := 28
  AND (* do case *);
```

O CASE pode ser usado com rótulo numérico:

```
CASE EI OF
   2,5 : BEGIN ... END;
     4 : comando;
   10,11,12 : comando
END
```

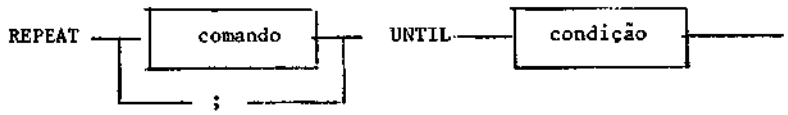
onde EI é expressão aritmética inteira.

CAPÍTULO 6

COMANDOS DE ITERAÇÃO

6.1 - Comando REPEAT

A sintaxe do comando REPEAT é:



Exemplo:

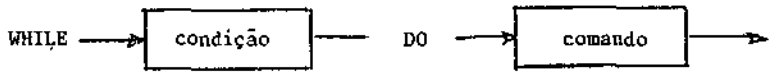
Seja obter o número de termos da série, necessários a satisfazer a condição:

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} > \text{limite}$$

```
VAR
   contador : integer;
   soma, limite: real;
BEGIN
   contador := 0;
   soma     := 0;
   READ (limite)
   REPEAT
      contador := contador + 1;
      soma     := soma + 1/contador
   UNTIL
      soma > limite
   WRITE (contador)
END.
```

6.2 - COMANDO WHILE

É semelhante ao comando REPEAT, mas a condição é avaliada no início do "loop" e não no fim. A sintaxe é:



Exemplo:

```
VAR
  numero: integer;
BEGIN
  numero: 0;
WHILE numero <= 10 DO
  BEGIN
    numero := numero + 1;
    WRITE (numero)
  END
END.
```

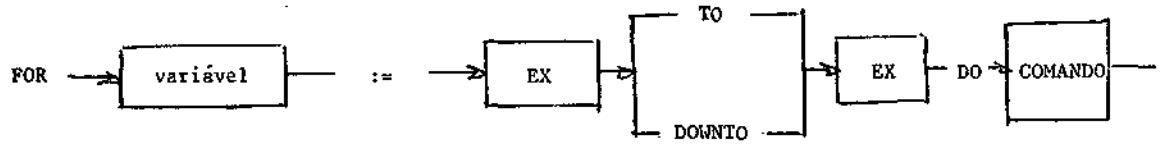
O programa escreve os números de 1 a 11, pois para numero = 0 a condição é verdadeira, portanto executa mais um ciclo e para no número = 11.

6.3 - COMANDO FOR

Este comando é usado no caso em que se quer fazer várias repetições e o número de repetições não depende do efeito sobre o comando:

```
READ (numero);
SOMA := 0;
FOR Termo := 1 TO numero DO
Soma := soma + 1/termo;
WRITELN (soma)
```

A sintaxe:



A palavra TO desloca a variável para o seu sucessor e a DOWNTO para o seu antecessor. Exemplo:

```
VAR
  MES: (jan, fev, mar, abr, mai);

  BEGIN
    FOR MES := jan TO mai DO
      comando;
```

CAPÍTULO 7

PROCEDIMENTOS E FUNÇÕES

7.1 - PROCEDIMENTOS OU ROTINAS

O comando

```
REPEAT
  READ (ch)
UNTIL ch ≠ branco;
```

lê caracteres do arquivo de entrada até aparecer um caracter não-branco. Este comando pode ser chamado pula brancos. Na forma de procedimento é escrito como:

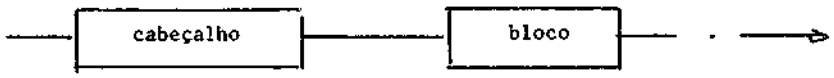
```
PROCEDURE:  pulabranco;
BEGIN
  REPEAT
    READ (ch)
  UNTIL CH ≠ branco
END;
```

Para chamar este procedimento no programa basta escrever;

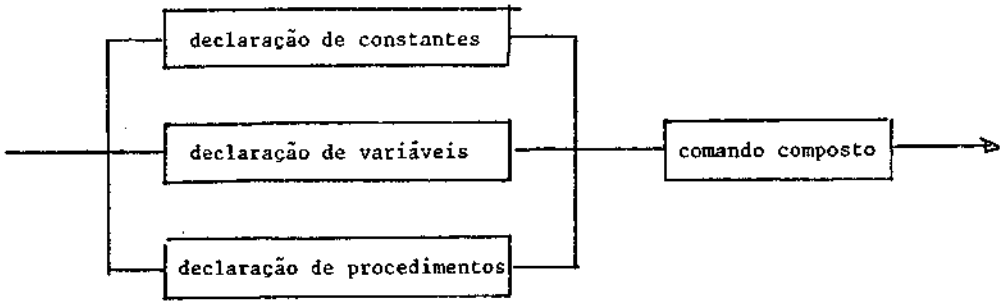
```
pulabranco;
```

O nome definido deste modo traduz "o que o comando realiza". Já na descrição do procedimento, os comandos entre "BEGIN" e "END" dizem "de que modo o comando realiza a ação".

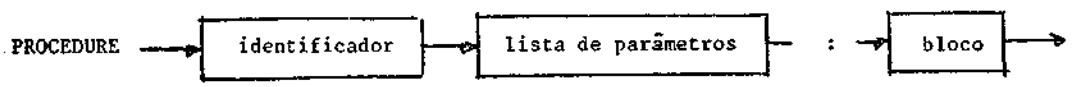
A sintaxe de um programa é:



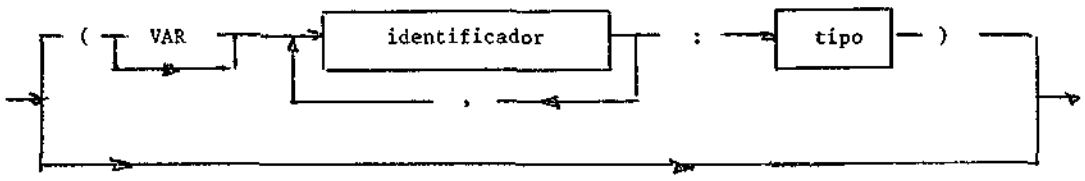
onde o bloco é:



A sintaxe de procedimentos é:



e a lista de parâmetros é:



7.1.1 - ELEMENTOS LOCAIS E GLOBAIS

Existem elementos que só são usados em uma parte do programa, isto é, são locais. Nestes casos pode-se usar um subprograma para executar este trecho. Os elementos locais do programa (constante, variável, procedimentos, função ou tipo) são declarados no cabeçalho da rotina:

As variáveis declaradas no corpo da rotina são locais e válidas somente dentro deste subprograma, desaparecendo com o término deste. Numa posterior chamada elas estão indefinidas como na primeira chamada. As variáveis fora da rotina são chamadas globais. Um mesmo identificador, x por exemplo, pode ser declarado como global e local; deste modo, são dois identificadores diferentes para o programa. O x global não deve ter significado para a rotina a fim de não causar conflitos.

O programa principal tem a forma de uma rotina, com declarações no cabeçalho, que são globais a todas as rotinas definidas em seu interior.

Exemplo de rotina com aparente conflito de identificadores:

```
PROGRAM TESTE;
VAR a, b, d, e: integer (variáveis globais)
  PROCEDURE produto; (rotina global)
    VAR C, d: integer (variáveis locais)
    Begin
      C := a ; d := b ; e e = 0 ;
  WHILE d ≠ 0  DO
  Begin IF odd(d) THEN e := e + c;
    c := 2 *c ; d := d DIV 2
  END
  END;

  Begin (programa principal)
  a := 5; b := 7; d := 10;
  multiplique
  END.
```

Este programa fornece como resultado a = 5 b = 7 d = 10 e = 35.

A variável d local tem o valor d = 7, passando a d = 7 DIV 2 = 3. A variável d global vale d = 10.

7.1.2 - PARÂMETROS DE SUBPROGRAMAS

Os indicadores do cabeçalho do subprograma são chamados parâmetros formais. São usados somente no corpo do subprograma, onde são locais.

Os parâmetros especificados na chamada de um subprograma são chamados reais e substituem os formais na execução.

No cabeçalho indica-se o tipo do parâmetro formal, que deve ser o mesmo do correspondente parâmetro real. Além disso, também se indica a espécie de substituição a fazer (pelo valor presente, pela identidade da variável ou pela expressão real).

a) Substituição de valor

É a mais comumente usada. O parâmetro real é calculado e seu resultado é dado ao parâmetro formal e usado no subprograma. Exemplo:

```
PROGRAM TESTE;
VAR i : integer;
    a : array [1 .. 2] of integer;
PROCEDURE P (x: integer)
  BEGIN
    i := i + 1; x := x + 2
  END

BEGIN (programa principal)
a[1] := 10; a[2] := 20; i := 1; P(a[i])
END.
```

Antes da chamada tem-se a[1] = 10. Ao ser feita a chamada, o valor de a[1] é passado para x, isto é, é feito x := a[1] := 10 e dentro do subprograma ter-se-á x = 10. Deste modo, o valor de a[1] não é modificado com as modificações do x.

(x := x + 2 = 10 + 2 = 12)

b) Substituição de variáveis

O parâmetro real é uma variável e não um valor. Se ela for indexada, seus índices são calculados e substitui o parâmetro formal correspondente. Deve ser empregada quando o parâmetro representa o resultado de um subprograma, isto é, quando o subprograma vai mudar o valor da variável no programa principal.

Para indicar este tipo de variável para o subprograma, deve-se usar o símbolo VAR na frente dos parâmetros formais aos quais se aplica. Exemplo:

```
PROGRAM TESTE;
VAR i : integer;
    a : array [1 .. 2] of integer;
PROCEDURE P(VAR x : integer)
  BEGIN
    i := i +1; x := x + 2
  END;

  BEGIN
    a[1] := 10; a[2] := 20; i := 1
    P(a[i])
  END.
```

Antes da chamada tem-se a [1] := 10. Na chamada é feita a equivalência x = a[1] e dentro do subprograma a variável x será substituída por a[1]. Assim, o comando x := x + 2 passa a ser a[1] := a[1] + 2 = 10 + 2 = 12. Deste modo, o valor de a[1] que antes da chamada era 10, após a chamada será 12. Sendo i uma variável global, o seu valor após a chamada é 2. Não havendo uma segunda chamada a[2] permanece como 20.

c) Substituição por nome

O parâmetro real substitui literalmente o formal correspondente, sem qualquer cálculo. É o caso do uso de um subprograma (rotina ou função) como parâmetro de outro subprograma. Exemplo:

```
PROCEDURE BASE (VAR a, b : real; PROCEDURE F);

     VAR i,j : integer;
         c,d : real;
     BEGIN
       REPEAT
         i := K * 2;
         j := M + R;
         a := F(i + j);
         b := F(i * j);
         i := i + 1; j := j + 4
     UNTIL i > N
END;
```

Na chamada quer-se fazer o cálculo para três PROCEDURES diferentes G1, G2 e G3, assim:

```
PROGRAM TESTE;
  VAR K, M, N: integer;
      b1, b2 : real
  BEGIN
  BASE (b1, b2, G1);
  BASE (b1, b2, G2);
  BASE (b1, b2, G3);
END.
```

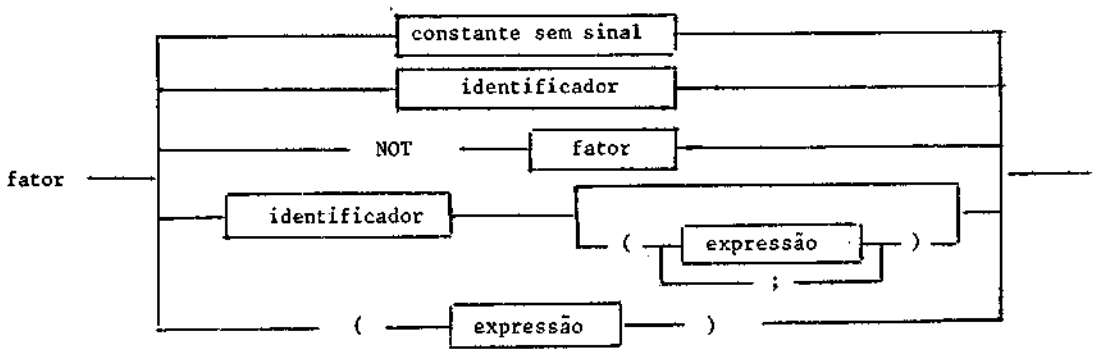
7.2 - FUNÇÕES

O Pascal possui funções padrões como SQR, SQRT, SIN, etc., que possuem um só argumento do tipo inteiro ou real. Enquanto uma rotina exerce um determinado efeito em parte do programa, a função fornece um valor.

A definição de função é semelhante à de procedimento. A sintaxe é:

Observe-se que deve ser fornecido o tipo associado à função.

A chamada de uma função é uma variedade de fator, cuja sintaxe engloba a função, e é:



CAPÍTULO 8

ESTRUTURA DINÂMICA DE DADOS

Uma estrutura estática de dados é aquela que tem o tamanho constante ao longo de sua vida. Podem-se definir estruturas estáticas na linguagem PASCAL tanto com ARRAY quanto com RECORD.

Uma estrutura dinâmica de dados tem o seu tamanho modificado ao longo do programa.

1) Pointer

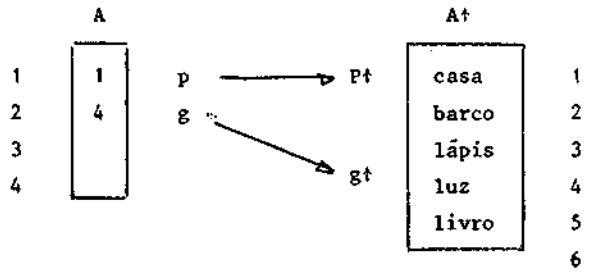
POINTER (apontador) é um tipo simples do mesmo modo que INTEGER, REAL ou BOOLEAN. Não é um identificador padrão, mas é declarado como tal. Exemplo:

```
TYPE
  indicador = ↑ objeto
```

Esta declaração significa que o tipo *indicador* contém endereços relativos à localização na memória do tipo objeto. A seta ↑ é que diz isso.

Podem-se associar variáveis do tipo objeto com apontadores do tipo *indicador*.

Para representar o conceito de POINTER esquematicamente, considerem-se duas matrizes colunas, onde A contém os apontadores e A↑ os elementos apontados:



É como se fosse feito: A↑ [A[1]] := casa A↑ [A[2]] := A↑ [4] := luz, mas num sentido mais amplo.

O primeiro elemento da matriz A contém o endereço de onde se encontra 'casa' na matriz A↑. De modo análogo, as variáveis p e g do tipo indicador contêm o endereço da localização na memória dos elementos p↑ e g↑.

A notação com seta antes do indicador é usada na declaração de tipo. A seta após o identificador forma um indicador que contém o elemento (conteúdo da memória). O mesmo identificador, sem seta, contém o endereço da posição da memória onde se encontra o respectivo elemento. Isto é,

```
p = 1    e    p↑ = casa
g = 4    e    g↑ = luz
```

Há três indicações básicas que podem ser feitas com ponteiro:

a) Não indica posição alguma

O ponteiro não contém endereço algum. Neste caso é indicado pela atribuição:

```
p := NIL
```

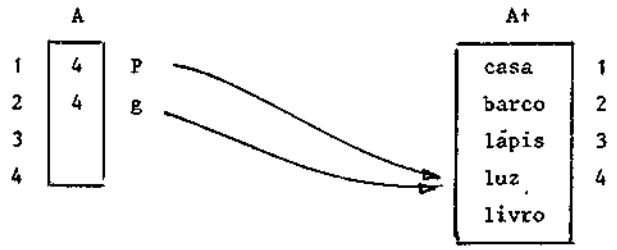
onde NIL significa : nenhum endereço.

O símbolo NIL é uma palavra reservada na linguagem PASCAL.

b) Indicação do mesmo objeto

Se se desejar que um POINTER contenha o mesmo endereço que o g, basta fazer a atribuição:

p := g



Isto é análogo a:

A [A[1]] := A [4] := luz

A [A[2]] := A [4] := luz

c) Indicação de objetos iguais em endereços diferentes

Se se quizer que dois POINTERS diferentes (isto é, com endereços diferentes) contenham o mesmo objeto, deve-se fazer a atribuição:

p↑ := g↑

voltando à analogia com matriz, inicialmente tem-se

A↑[1] := casa A↑[4] := luz

a atribuição:

A↑[1]:= A↑[4]

faz com que se tenha

A↑[1] := luz A↑[4] := luz

isto é, o mesmo objeto em duas posições diferentes. O valor existente em A↑[4] foi copiado para A↑[1].

O uso do POINTER em Pascal

Para usar o POINTER poderiam ser empregados os seguintes tipos: POINTER, ARRAY ou RECORD. O pointer só pode conter ele mesmo, o ARRAY só pode conter elementos do mesmo tipo. Logo, o RECORD é o tipo adequado para trabalhar com POINTER. Exemplo:

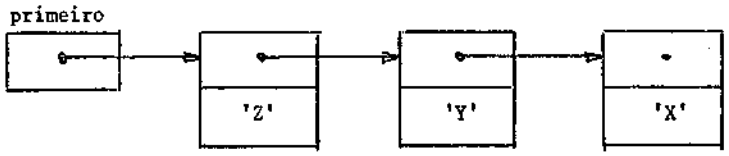
```
TYPE
  indicador := ↑pessoa;
  pessoa = RECORD
           ligação : indicador;

           data : tipo data
  END;
```

Observe-se que neste caso usou-se tipo "pessoa" (na declaração de POINTER) antes de ele ter sido declarado (no "RECORD"). Neste caso a inversão é aceita.

8.2 - LISTAS LIGADAS

a) Construção

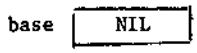
A lista ligada é o tipo mais simples de estrutura dinâmica de dados. Seja construir a seguinte estrutura:



Construção de três arquivos ligados

```
VAR base, novo: indicador;
    s: data;
```

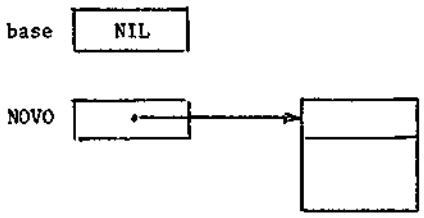
base := NIL; (a variável base não contém endereço)



O comando

```
NEW (NOVO);
```

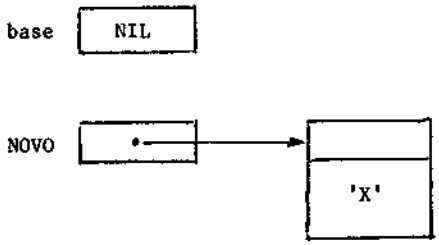
cria uma área (RECORD) com o nome de NOVO↑ cujo endereço está armazenado na variável NOVO. A situação agora é:



Nos dois comandos:

```
READ(s);
NOVO↑ . data := s;
```

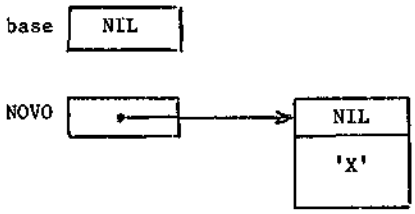
é lida a variável s := 'x' e armazenada. A nova configuração é:



O comando:

NOVO↑ . ligação := base

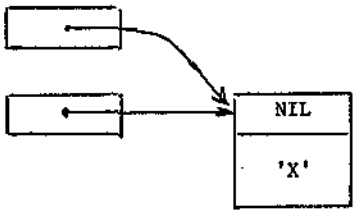
armazena no RECORD o endereço NIL. Deste modo, fica indicado que não vem área alguma antes dessa.



O comando:

base := novo

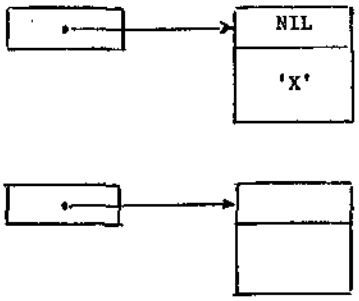
faz com que as variáveis base e NOVO contenham o endereço da primeira área,



Para se criar uma nova área faz-se

NEW (novo)

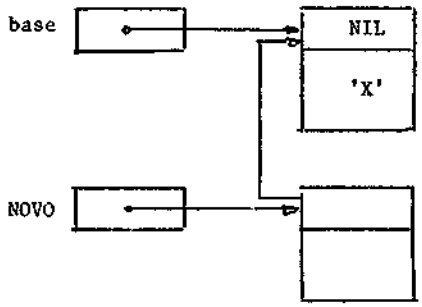
O endereço da nova área está em NOVO e o da área antiga em base,



O comando

NOVO↑ . ligação := base

armazena na segunda área o endereço da primeira,



Os comandos

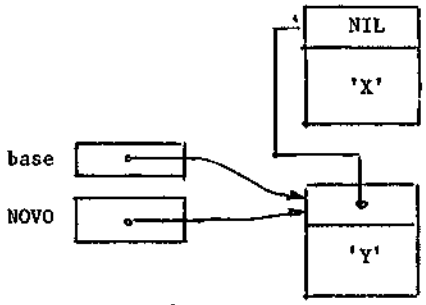
```
read (s);
NOVO↑ . data := s;
```

lêem a variável s := y e armazenam-na na segunda área.

O comando

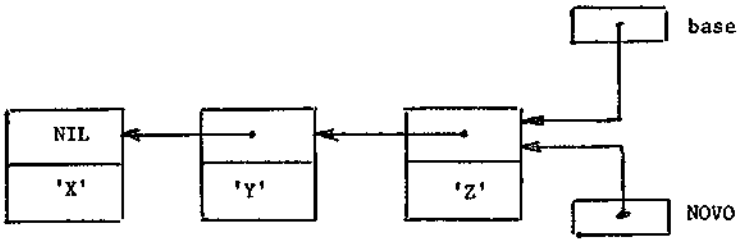
```
base := NOVO;
```

faz com que a variável base contenha também o endereço da segunda área,



De modo análogo, cria-se a terceira área;

```
new (NOVO);
NOVO↑ . ligação := base
read (s);
NOVO↑ . data := s;
base := NOVO;
```



Todos os comandos anteriores podem ser reunidos num comando de repetição:

```
base := NIL
FOR I := 1  TO  3  DO
Begin
New (NOVO);
NOVO↑ . ligação := base
read (s);
NOVO↑ . data := s;
base := novo;
end;
```

b) Acesso

Uma vez construída a lista ligada, quer-se acessar uma certa área com uma informação, por exemplo a área onde está armazenado 'Y'.

O acesso à última área criada é simples, pois a variável base contém o seu endereço. Assim:

```
s := base↑ . data
WRITE (s)
```

Então ter-se-á

```
s = 'z'
```

O endereço da penúltima área está armazenado na última área e pode ser recuperado por:

```
pt := base↑ . ligação;
```

O acesso à penúltima área pode, então, ser feito como:

```
s := pt↑ . data;
WRITE (s);
```

Então, ter-se-á s = 'Y'. Ou, do seguinte modo:

```
s := base↑ . ligação↑ . data;
write(s);
```

Um modo de percorrer todas as áreas é por meio de:

```
pt := base;
WHILE pt ≠ NIL DO
  pt := pt↑ . ligação;
```

CAPÍTULO 9

9.1 - DECLARAÇÃO DE ARQUIVOS

A entrada e a saída de dados num programa são feitas por meio de arquivos (FILES). A comunicação de um processo com o seu ambiente é feita por meio do FILE. Num arquivo podem-se armazenar muito mais dados do que num programa.

Em Pascal um arquivo é uma variável, mas com características diferentes, pois pode existir antes e depois da execução do programa; assim como pode ser maior que o próprio programa.

Um arquivo é um grande volume de informações do mesmo tipo, guardados sob um nome. A declaração de arquivo faz-se de modo semelhante à de variáveis:

```
TYPE <identificador> = FILE OF <tipo>,
```

onde se vê que todos os elementos do arquivo devem ser do mesmo tipo. Uma variável de arquivo f é declarada como qualquer variável:

VAR f : <tipo file>

ou

VAR f : FILE OF <tipo>.

O nome do FILE deve ser também incluído no cabeçalho do programa:

PROGRAM TESTE (input, output, f);

A declaração de cada variável f de arquivo cria automaticamente uma área ("buffer") para ela, denotada por f↑.

9.2 - GERAÇÃO DE ARQUIVOS

A declaração de uma variavel de arquivo determina a sua identificação, estrutura e tipo.

Um arquivo é estruturado como uma sequência de áreas, as quais podem ser representadas esquematicamente como:

O número de componentes é chamado comprimento do arquivo. Se o comprimento for zero, ele é chamado vazio e representado por <>. O comprimento é aumentado dinamicamente. Para colocar um elemento no arquivo f, usa-se o operador

PUT (f).

Exemplo:

Gerar um arquivo de inteiros tal que seu i-ésimo componente tenha o valor i^2 e que contenha os quadrados dos números naturais menores que n. O algoritmo para gerar os dados para o arquivo é:

$$b_i = b_{i-1} + 2$$
$$a_i = a_{i-1} + b_i \qquad \text{Para } i > 1$$

onde os valores iniciais são $a_1 = b_1 = 1$, obtendo-se $a = i^2$.

E o programa para gerar o arquivo é:

```
VAR A, B : integer;
  f : FILE OF integer;
BEGIN
  A := 1 ; B := 1;
REPEAT
 f↑:= A ; put (f);
 B := B + 2 ; A := A + B
UNTIL A ≥ n
END.
```

Observe que A é primeiramente colocado na variável auxiliar f↑ e depois o valor é passado de f↑ para o arquivo por meio de PUT(f).

9.3 - CONSULTA A ARQUIVO

A consulta é feita em ordem sequencial, a começar do primeiro componente do arquivo. O operador

RESET (f)

faz com que o arquivo f esteja na posição inicial para leitura, onde transfere a informação da primeira posição para f↑.

Para ler o componente seguinte usa-se a rotina

GET (f)

a qual desloca para o sucessor, armazenando-o em f↑. Se não houver sucessor algum f↑ é indefinida.

Pode-se saber se há sucessor ou não por meio do comando

EOF (f)

que retorna o valor FALSE, se ainda não chegou o fim do arquivo. Neste caso, o sucessor está armazenado em f após o uso do GET (f); ou, então, retorna o valor TRUE, se chegou ao fim do arquivo, e f↑ é indefinido.

Se não houver componente algum após o uso do RESET (f), EOF (f) será imediatamente TRUE.

Para apagar um arquivo f é usado o comando

REWRITE (f).

Apesar de não serem comandos padrões, os compiladores PASCAL combinam as operações de arquivos sequenciais por meio de

READ (f,x);

a qual é equivalente a

x := f↑ ; GET (f):

e de

WRITE (f,x)

que equivale a

f↑ := x; PUT (f);

Exemplo:

Ler em arquivo de números reais e calcular sua soma

```
VAR f : FILE OF real;
  s, x : real;
Begin
  s := Ø; reset (f);
  WHILE NOT EOF (f) DO
    Begin
    READ (f,x); S := s + x;
    END
END.
```

CAPÍTULO 10

RÓTULOS

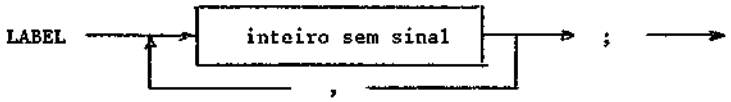
10.1 - O COMANDO GOTO

Este comando permite transferir o controle do programa de uma parte para outra, com algumas restrições.

Uma das aplicações do GOTO é para sair de LOOPS; outra, se houver um erro dentro de um procedimento e se se quizer parar o programa, basta fazer um GOTO para o fim.

10.2 - DECLARAÇÃO DE RÓTULO

A sintaxe de rótulo é:



A declaração de rótulo é a primeira das declarações.

O pulo dado pelo GOTO deve ser dentro do bloco onde ele é usado ou para fora. Não deve ser usado de um bloco externo para um interno. Em particular, pode-se sair de um procedimento, mas não se pode entrar nele. Exemplo:

```
  LABEL 2, 5;
     ...
  PROCEDURE TESTE;
  LABEL 4;
  Begin
    ....
  GOTO 4;
     ...
  4 : SS
  END;
BEGIN
     ...
  IF A THEN GOTO 2 ELSE GOTO 5;
     ...
  2 : AS;
     ...
  5 : (*FIM*)
  END.
```

REFERÊNCIAS BIBLIOGRÁFICAS

DAHL; DJKSTRA; HOARE. Structured Programming, capítulo Notes on Structured Programming, New York, NY, Academic Press, 1972, pag. 1-82.

GROGONO; P. "Programming in Pascal, Addison-Wesley, Reading, Massachussetts, 1978.

WIRTH, N. Algorithms Data Structures. Prentice Hall, Englewood. Cliffs; NJ, 1976.

——— Programação Sistemática em Pascal; Editora Campus; Rio de Janeiro; 1982.

APOIO TÉCNICO:
INSTITUTO DE PESQUISAS ESPACIAIS - INPE
• Grupo de Divulgação

IMPRESSO NA GRÁFICA DO INPE