6) ESTRUTURA DE UM PROGRAMA

<programa> = <cabeçalho>
<unidades usadas>
<parte de declarações>
<parte de comandos>.

<cabeçalho> = *program* <identificador>;

<unidades usadas> = *uses* <lista de units>;

<parte de declarações> =
var <lista-1 de identificadores> : <tipo-1>;
<lista-2 de identificadores> : <tipo-2>;
...

const <identificador-1> = <valor-1>;
<identificador-2> = <valor-2>;
...

type <identificador-1> = <descrição-1 dos valores>;
<identificador-2> = <descrição-2 dos valores>;
...

label <lista de identificadores e/ou nºs inteiros>;

procedure <identificador>(<lista de parâmetros>);
<parte de declarações>
<parte de comandos>;
...

function <identificador> (<lista de parâmetros>):<tipo do resultado>;
<parte de declarações>
<parte de comandos>;
...

<parte de comandos> = *begin*
<comando-1>;
<comando-2>;
•
•
•
end

Observações:
(1)O **cabeçalho** é a identificação do programa;
(2)Uma **lista** de identificadores é uma sequência de nomes (de variáveis) separados por vírgula;
(3)Os identificadores de constantes servem como "**sinônimos**" para os valores indicados, os quais não são alterados pelo programa;
(4)Observe que os procedimentos(assim como as funções) têm a **mesma estrutura** de um programa, exceto pela palavra procedure (ou function) em vez de program e terminando por um ponto e vírgula em vez de ponto;
(5)Os comandos são os recursos (ié, as "**ferramentas**") da linguagem que permitem as "**ações**" do programa;
(6)Comandos podem ser **simples** e **compostos**. Neste caso, são separados por ; e delimitados pelas palavras (ié, escritos entre) *begin* e *end*.
(7)Na apresentação, a seguir, dos comandos do Pascal, vamos descrever sua **sintaxe** (ié, a forma de escrevê-lo) e sua **semântica** (ié, o efeito/a ação resultante da sua execução).

7) COMANDOS:

7.1) Comandos de Entrada:

Trata-se de como o computador recebe os dados (fornecidos por uma unidade de entrada) e os armazena em variáveis.

A **sintaxe**:

READ(<lista de identificadores>)

ou

READLN(<lista de identificadores>)

O **efeito** dos comandos READ e READLN é que os valores (separados por

pelo menos um espaço em branco, quando forem numéricos) correspondentes aos identificadores da lista, quando fornecidos..., são lidos ...a partir da posição corrente do cursor(?)... numa mesma linha... No caso do READLN, após a leitura é provocada uma mudança de linha.

7.2) Comandos de Saída:

Trata-se de como o computador fornece, numa unidade de saída, resultados do processamento e mensagens.

A **sintaxe**:

WRITE (<lista de expressões e/ou mensagens>)

ou

WRITELN(<lista de expressões e/ou mensagens>)

O **efeito** dos comandos WRITE e WRITELN é que os valores indicados (ié, os resultados das expressões, que são "neste momento" avaliadas, e os *strings* correspondentes às mensagens) são escritos na linha corrente do cursor. No caso do WRITELN, após a escrita ocorre um mudança de linha.

Observações:

1)...formatação... :**n** e :**n**:**d**

2)recursos de tela... a unidade ***crt***...

clrscr - readkey - delay - gotoxy(col,lin)
clreol – window - textcolor - textbackground
keypressed

7.3) Comando de Atribuição:

É o comando com o qual o sistema armazena dados gerados pelo programa e altera conteúdos de variável.

A **sintaxe**:

<variável>:= <expressão>

O **efeito** é o seguinte: a <expressão> é avaliada e o resultado armazenado na (“caixinha”) da variável.

Observações:

1) É preciso tomar cuidado, porque o resultado é jogado “em cima” do conteúdo anterior da variável, apagando o que tinha lá.
2) haverá erro por “incompatibilidade” se o resultado da expressão for real e a variável inteira. Caso ocorra o contrário, o valor inteiro da expressão é convertido em real na variável (ié, ocorre “submissão”).

Exemplos:

(1) MEDIA := (A+B+C)/3
(2) DELTA := sqr(B) – 4*A*C
(3) X1 := (-B+sqrt(DELTA))/(2*A)
(4) Y := X*X-5*X-2
(5) I := I+1
(6) N := N-1
(7) CONTA := CONTA+1
(8) FAT := 1
(9) MAIOR := A
(10) A := B
(11) {“Troca de valores”} AUX := A; A := B; B := AUX;
(12) CHAVE := false
(13) TESTE := (N>=0) and (N<=10)

7.4) Comando Condicional IF:

É o comando que permite ao sistema decidir se um comando será ou não executado, ou ainda decidir entre dois comandos qual deles será executado.

A **sintaxe:**

(a) *if* <condição> *then* <comando>

ou

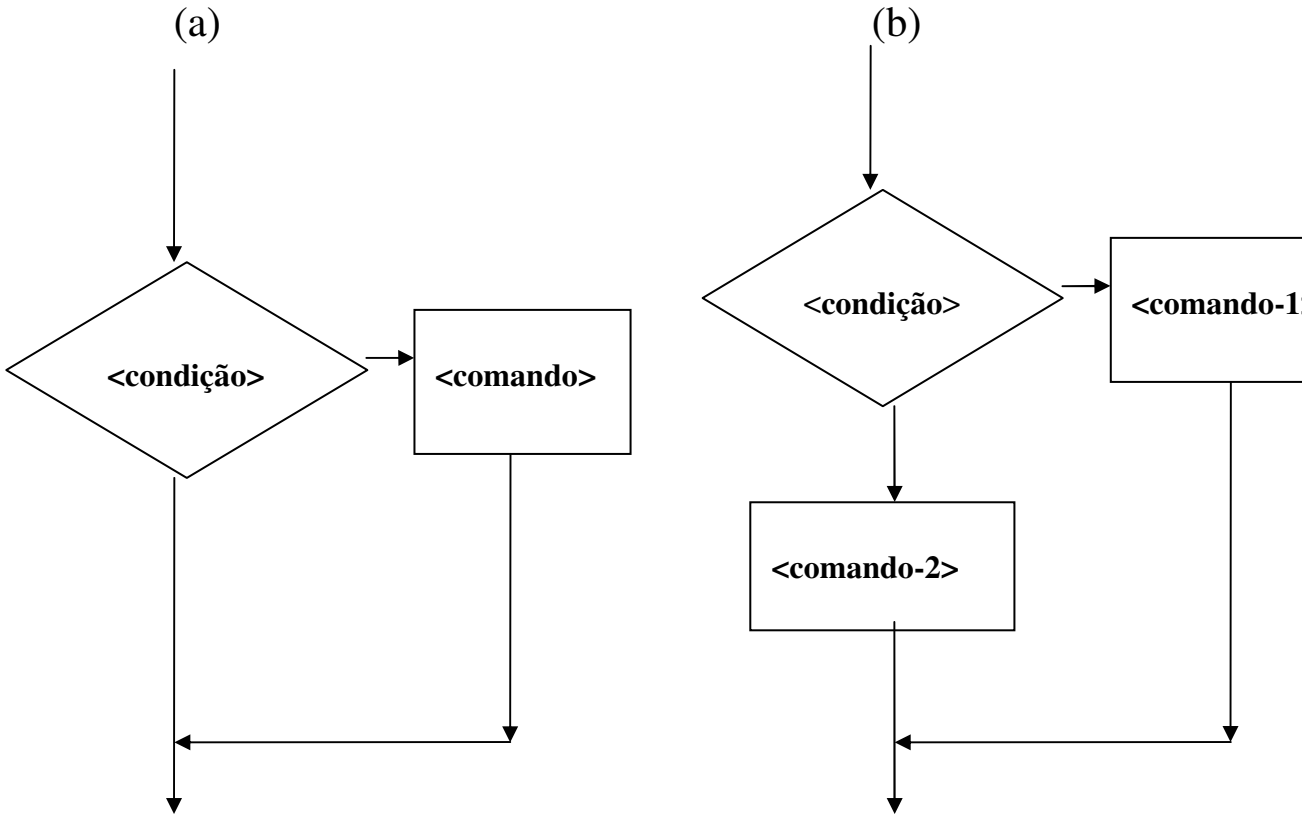
(b) *if* <condição> *then* <comando-1>
else <comando-2>

Onde <condição> é uma expressão lógica. E, <comando>, <comando-1> e <comando-2> podem ser qualquer comando da linguagem.

O **efeito:**

Em (a), <comando> será executado apenas se a <condição> for verdadeira. Em (b), <comando-1> será executado se a <condição> for verdadeira; ou, será executado <comando-2>, se ela for falsa.

De outra maneira:



Observações:

1) Não deve haver ponto e vírgula antes do *else*.

2) Se algum dos comandos for composto, lembrar que devem ser usados os delimitadores *begin* e *end*.

Exemplos:

```
(1) if A=0 then write('A equação não é do 2º grau');

(2) if MEDIA >= 7 then  AP := AP + 1
                        else REP := REP + 1;

(3) if N mod 2 = 0  then writeln(N,' é par')
                    else  writeln(N,' é ímpar');

(4) if ANO mod 4 = 0
       then writeln('O ano ',ANO,' é bissexto')
       else  writeln('O ano ',ANO,' não é bissexto');

(5) if  NUM mod I = 0 then PRIMO := false;

(6) if  I < 100 then I := I + 1;

(7) if (NOTA < 0) or (NOTA > 10)
       then writeln('valor inválido)
       else SOMA := SOMA + NOTA;

(8) if (NOTA >= 0) and (NOTA <= 10)
          then SOMA := SOMA + NOTA
          else writeln('valor inválido');

(9) if  A < B then MENOR := A
             else  MENOR := B;

(10) if A >= B then MENOR := B
               else  MENOR := A;

(11) MENOR := B;
     if  A < MENOR then MENOR := A;

(12) if DELTA < 0
       then writeln(' não existem raízes reais')
       else if DELTA = 0
              then writeln(' raízes iguais a ', -B/(2*A))
              else begin
                     X1 := (-B + DELTA)/(2*A);
                     writeln('1ª raiz: ', X1);
```

```
    X2 := (-B - DELTA)/(2*A);
    writeln('2ª raiz: ', X2);
end;
```

Observações:
(1) Os exemplos (7) e (8) têm exatamente o mesmo efeito.
(2) Os exemplos (9), (10) e (11) têm exatamente o mesmo efeito.

7.5) Comando Condicional CASE:

Este comando permite selecionar entre dois ou mais comandos qual será executado.

A **sintaxe**:

case <expressão> *of*
<lista-1 de constantes>: <comando-1>;
<lista-2 de constantes>: <comando-2>;
<lista-3 de constantes>: <comando-3>;
.
.
.
else <comando-n>
end

O **efeito** é o seguinte: a <expressão> é avaliada e, então, será executado o comando correspondente à lista que contenha aquele valor calculado.

Observações:

1) As constantes devem ser do mesmo tipo da expressão;
2) A "cláusula" *else* é facultativa.

Exemplos:

```
(1) case I of
        1: R := sin(X);
        2: R := cos(X);
        3: R := exp(X);
        4: R := ln(X)
    end;

(2) case OPERADOR
        '+': R := A+B;
        '-': R := A-B;
        '×': R := A*B;
        '/': R := A/B
    end;

(3) case MES of
        1, 3, 5, 7, 8, 10, 12 : DIAS := 31;
                4, 6, 9, 11: DIAS:= 30;
```

```
                    2 : if ANO mod 4 = 0 then DIAS := 29
                                         else DIAS := 28
                  else  writeln(‘ mês inválido’)
end;
```

7.6) Comando Repetitivo WHILE:

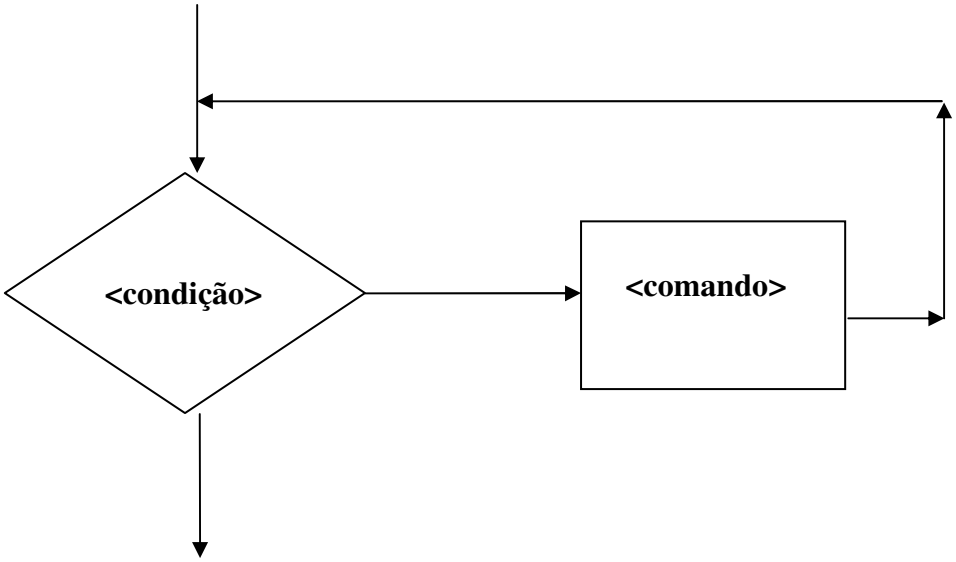
Aqui um comando é executado um número de vezes dependente do valor lógico de uma expressão.

A **sintaxe**:

while <condição> *do*
<comando>

O **efeito**: A <condição>, que é uma expressão lógica, é avaliada. Se o valor for ***true*****,** o <comando> é executado e o processamento retorna ao próprio comando ***while***; se o valor for ***false***, o <comando> não é executado e o processamento se transfere para o comando seguinte ao comando ***while***.

De outra maneira:



Exemplos:

(1)
```
A := 1;
while A <= 10 do
  begin
    write(A);
    A := A +1
  end;
```

(2)
```
write('digite 1º nome: ');
readln(NOME);
while NOME <> 'FIM' do
    begin
        write('digite 2 notas: ');
```

```
    readln(NOTA1, NOTA2);
    writeln('sua média é: ', (NOTA1 + NOTA2)/2);
    write('digite novo nome (ou "FIM" para terminar): ');
    readln(NOME)
end;
```

7.7) Comando Repetitivo REPEAT:

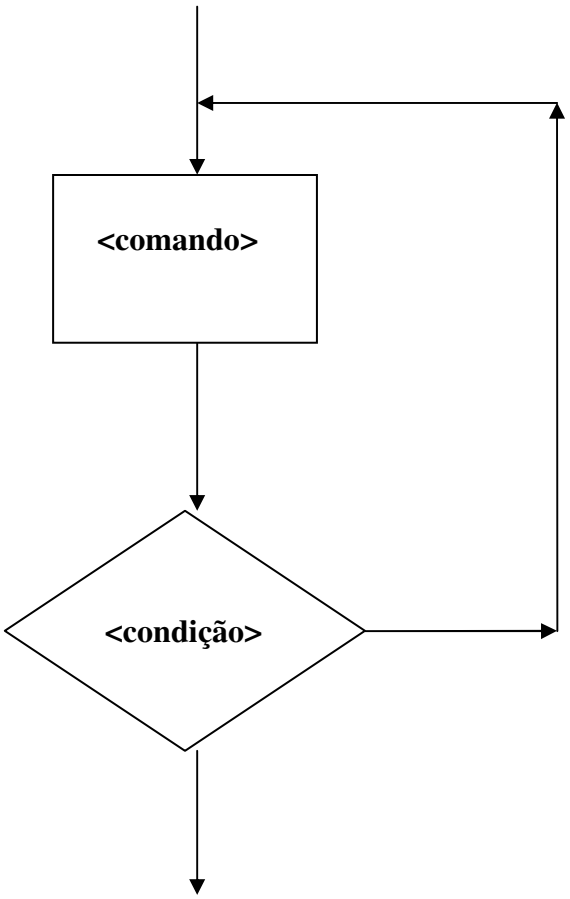
Da mesma maneira que o comando WHILE, aqui, uma sequência de comandos é executada repetidas vezes sujeita ao controle de uma expressão lógica.

A **sintaxe**:

repeat
<lista de comandos>
until <condição>

O **efeito**: A <lista de comandos> é executada e então a <condição>, que é uma expressão *booleana*, é avaliada. Se o valor for ***true,*** a execução do *repeat* termina e o processamento passa para o comando seguinte; se o valor for ***fals***e, a lista de comandos é novamente executada e tudo se repete. Em outras palavras, a <lista de comandos> é executada repetidamente até que a <condição> tenha valor ***true***.

De outra maneira:



Exemplos:

```
(1) A:= 1;
    repeat
       write(A);
        A := A +1;
    until A > 10;

(2) repeat
        write('N = ');
        readln(N)
    until (N>= 0) and (N<= 10);

(3) repeat
        •
        •
        •
        write('quer continuar(S/N)? ');
        readln(RESP)
    until (RESP = 'N') or (RESP = 'n');
```

Observações:

1) No comando *while* a condição é de **"repetição"**, enquanto que no comando *repeat* é de **"interrupção"**. Portanto, elas serão "contrárias" se construirmos dois trechos de programa equivalentes, um com while e o outro com repeat.

2) No comando *while,* se a <condição>, logo de início, é falsa então o comando não será executado nenhuma vez. Enquanto que, se a <condição> for sempre verdadeira então o "programa entra em *loop*", ié, não pára nunca! (será necessário usar conjuntamente as teclas **Ctrl** e **Break**).

3) No comando *repeat*, como a <condição> é testada no fim, a <lista de comandos> é executada pelo menos uma vez.

4) Obviamente, alguma "coisa" deve ocorrer no interior do <comando>, no caso do *while* (ou da <lista de comandos>, no caso do *repeat*) para que as condições de controle se modifiquem de maneira que, em algum momento, a execução do programa saia do "laço de repetição".

7.8) Comando Repetitivo FOR:

Este comando permite que a execução de um comando seja repetida um número fixo de vezes.

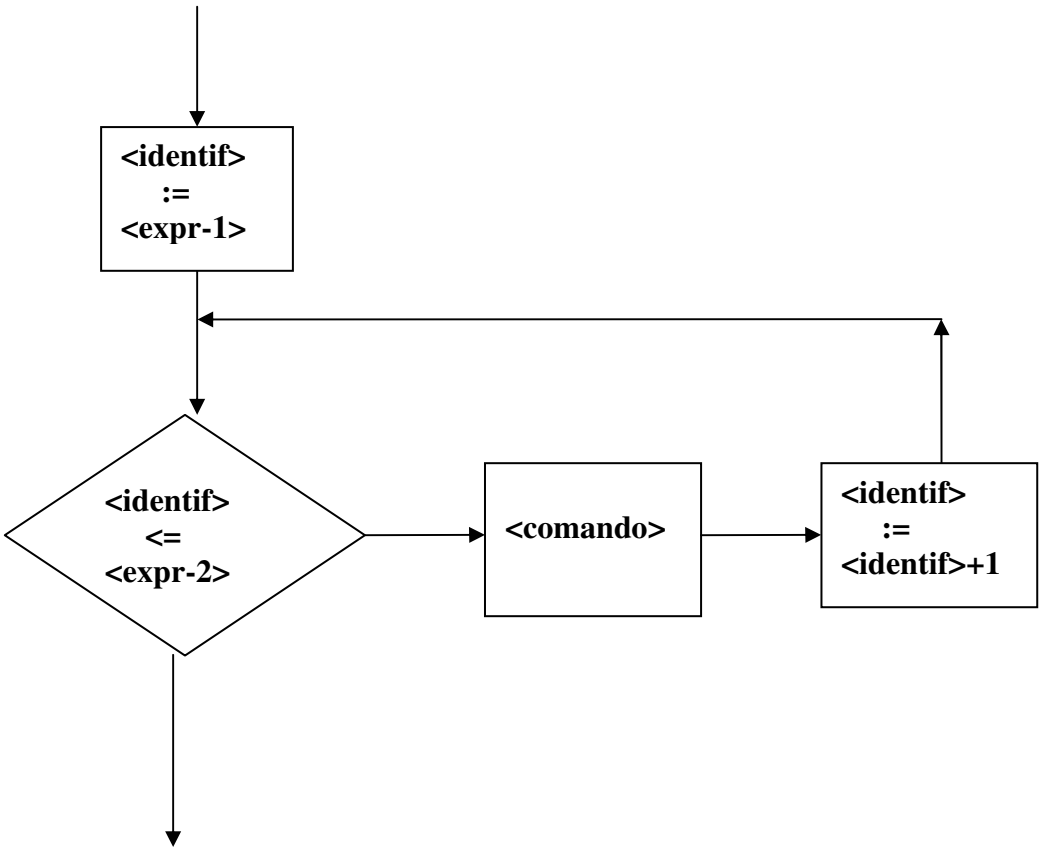
A **sintaxe**:

for <identificador> := <expressão-1> *to* <expressão-2> *do*
<comando>

onde o <identificador> é o nome de uma variável, chamada variável de controle, que deve ser de um tipo ordenado. As duas <expressões> devem assumir valores do mesmo tipo da variável.

O **efeito**: A variável de controle é incializada com com o valor da <expressão-1> e esse valor é comparado com o valor da <expressão-2>. Se for menor então o <comando> é executado; a seguir, a variável é incrementada de uma unidade e é feita nova comparação com o valor da <expressão-2>. Isso se repete até que o valor da variável de controle ultrapasse o da <expressão-2>, quando então o processamento passara para o comando seguinte.

De outra maneira:



Exemplos:

(1) for A := 1 to 10 do write(A);

(2) read(N);
SOMA := 0;
for I := 1 to N do SOMA := SOMA + I;
writeln('SOMATÓRIO de ', N, 'é: ', SOMA);

(3) read(N);
SOMA := 0;
for I := N downto 1 do SOMA := SOMA + I;
writeln('SOMATÓRIO de ', N, 'é: ', SOMA);

Observações:

1) Uma alternativa para o comando for é o uso da palavra ***downto*** em vez de ***to***. Neste caso, a variável começa com o valor da <expressão-1> e, a cada passo, ela é decrementada de uma unidade até que seu valor seja menor que o da <expressão-2>, quando <comando> não será mais executado. Observe que os exemplos (2) e (3) são portanto "equivalentes".

2) O comando *for* é mais indicado (que *while* e *repeat*) se o número de

repetições é conhecido de antemão.

3) O valor da variável de controle não pode ser alterado pela sequência de comandos. E, após a execução do comando *for*, o seu valor fica indefinido.

7.10) Comando de Desvio Incondicional GOTO:

Este comando serve para desviar o fluxo de execução do programa.

A **sintaxe**:

goto <rótulo>

onde <rótulo> é um valor inteiro ou um identificador, que deve ser colocado à esquerda de algum comando, seguido de dois pontos.

O **efeito**: a execução do programa continua com o comando rotulado com aquele <rótulo>.

Observação: Não é permitido o desvio para "dentro" de um comando composto nem para o interior de um comando estruturado. Mas, sim é permitido, para "fora" de um comando composto através de um comando condicional.