10) REGISTROS :

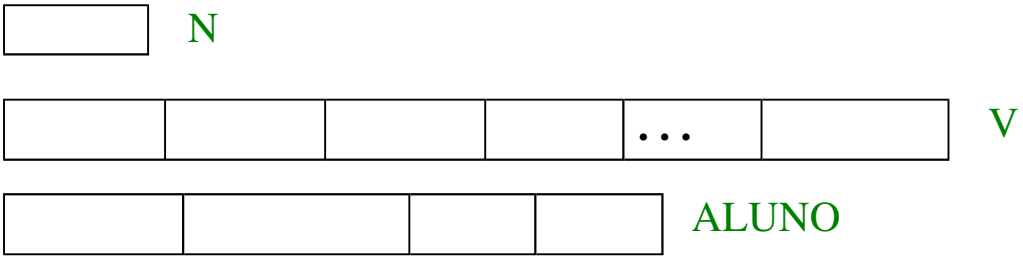
De um modo geral, as variáveis podem ser:

- simples (*integer*, *real*, *boolean*, *char*)
  e
- compostas:
  - homogêneas (*array*)
  - heterogêneas (*record*)

Exemplos:

```
var  N,I: integer;  A,B,C: real;
     CHAVE: boolean;  CARAC: char;
     V: array[1..20] of real;
     M: array[1..10,1..10] of real;

var ALUNO: record
                MATRIC: string[10];
                NOME: string[30];
                NOTA1, NOTA2: real
           end;
```



A referência aos "**campos**" da variável ALUNO é feita por: ALUNO**.**MATRIC, ALUNO**.**NOME, ALUNO**.**NOTA1 e ALUNO**.**NOTA.2.

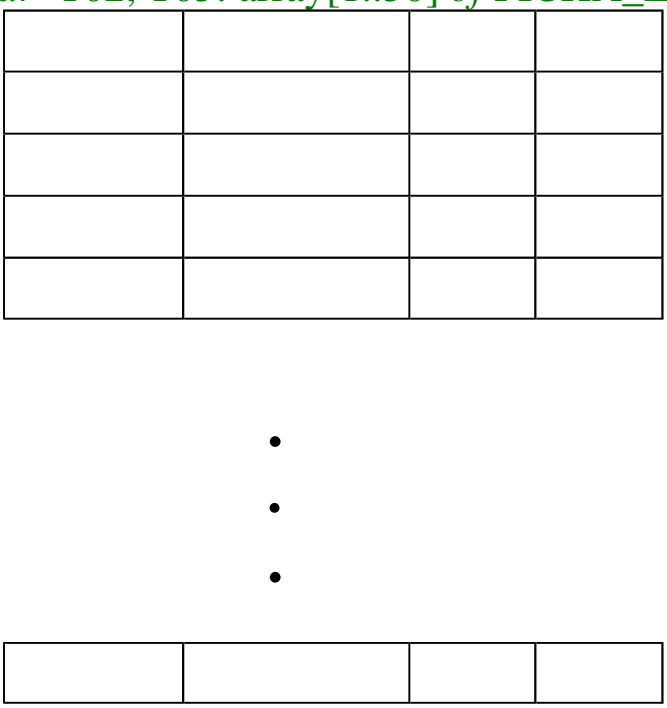
De outra maneira:

```
type FICHA_ESCOLAR = record
                        MATRIC: string[10];
                        NOME: string[30];
                        NOTA1, NOTA2: real
                     end;

var ALUNO: FICHA_ESCOLAR;
```

Podemos também declarar:

var T02, T05: array[1..50] *of* FICHA_ESCOLAR;



Para um *array* deste tipo, fazemos **referência** aos seus campos da seguinte maneira: T02[I]**.**NOME, T02[I]**.**MATRIC, T02[I]**.**NOTA1 e T02 [I]**.**NOTA2

Outros exemplos:

1) *type* COMPLEXO = *record*
RE, IM: *real*
end;

var X: COMPLEXO;

… read(X**.**RE); … ; write(X**.**RE, ‘ **+** ’,X**.**IM, ‘**i**’);

2) *type* DEPTO = *record*
COD: *integer*;
NOME: *string*[20]
end;

var TABELA: *array*[1..50] *of* DEPTO;

… *for* I:=1 *to* 50 *do*
if TABELA[I]**.**COD = COD_LIDO

```
                then write(TABELA[I].NOME);
```

3)
```
type FICHA_CADASTRAL =
                record
                    MATRIC: string[10];
                    NOME: string[30];
                    SALARIO, CPF: real;
                    SEXO: char;
                    NASC, ADMISSAO: DATA;
                    ENDERECO: LOCALIZACAO;
                    NUM_DEPEND: integer;
               end;
  var FUNC: FICHA_CADASTRAL;
```

obs:
Neste caso, teríamos ainda de definir (antecipadamente!):

```
type DATA = record
                    DIA: 1..31;
                    MES: 1..12;
                    ANO: integer
                end;

type LOCALIZACAO =
                record
                    RUA: string[30];
                    NUMERO: integer;
                    CEP: string[9]
                    CIDADE: string[20];
                    UF: string[2]
                end;
```

E, para referência aos campos, devemos utilizar: FUNC.MATRIC, FUNC.NOME, FUNC.CPF, FUNC.NASC.MES, FUNC.ENDERECO.CEP, etc

4)
```
type PRODUTO = record
                        CODIGO: integer;
                        DENOM: string[30];
                        QUANTIDADE: integer;
                        PRECO: real;
```

end;

Registro é um Tipo de Dados Estruturado. Trata-se de um conjunto de dados "logicamente relacionados", mas possivelmente de tipos diferentes. Uma variável do tipo *record* corresponde a um conjunto de posições de memória (ié, de **campos**) conhecidas por um mesmo nome, e individualizadas por identificadores associados a cada campo (cada um com o seu tipo).

Como declarar:

var <lista de id-variáveis> :
record
<id-campo-1>:<tipo-1>;
<id-campo-2>:<tipo-2>;
...
*en*d;

ou

<id-tipo> = *record*
<id-campo-1>:<tipo-1>;
<id-campo-2>:<tipo-2>;
...
*en*d;

var <lista de id-variáveis>: <id-tipo>;

Como fazer referência ao conteúdo do **campo-*i*** de uma variável**:** <id-variável>**.**<id-campo-*i*>

Observação: O comando with pode ser utilizado para representar os campos de um registro referindo-se apenas a esses campos.

A **sintaxe**:

with <id-variável.> *do*
begin
•
•
•
end;

11) ARQUIVOS:

Algumas limitações ao usar *arrays*:

1) São armazenados na memória principal por inteiro.
2) Se seu tamanho é muito grande, o espaço de memória utilizado para armazenamento tb é.
3) A sua existência tem o tempo de execução do programa.
4) Se houver queda de energia!...

Arquivo é um Tipo de Dados Estruturados, que são manipulados fora do ambiente do programa, ié, fora da memória principal.

Arquivos são armazenados em dispositivos externos de memória (ié, discos, fitas magnéticas, etc), que têm como vantagem adicional, serem mais seguros, mais baratos e não se extinguirem ao desligar o computador. Além de permitir sua portabilidade para outros computadores.

Um **arquivo** é uma sequência, de "tamanho variável", de elementos do mesmo tipo (usualmente do tipo *record*). Somente uma parte é conservada na memória principal. Apenas um elemento é acessado de cada vez.

Em Pascal, temos dois tipos de arquivo: **binário** e tipo-**texto**. Os arquivos têm organização **seqüencial** e o seu acesso pode ser **seqüencial** ou **direto**.

Arquivos podem ser criados, consultados e modificados por programas.

Como declarar:

var <lista de id-var-arquivo>: *file of* <tipo-dado>;

ou

type <id-tipo-arquivo> = *file of* <tipo-dado>;
var <lista de id-var-arquivo>: <id-tipo-arquivo>;

onde: <tipo-dado> é o tipo dos "componentes" do arquivo, podendo ser qq tipo, mas que usualmente é o tipo *record*.

Exemplo:

```
type REG = record
                NOME, EMAIL: string[30];
                NTEL: string[10];
                NASC: DATA;
                OBS: string[20];
```

```
                    end;
    var ARQ: file of REG;
        X: REG;
```

ou

```
    type REG = ...
    type ARQUIVO = file of REG;
    var  ARQ: ARQUIVO;
        X: REG;
```

FUNCÕES e **PROCEDIMENTOS**, já existentes na linguagem, para manipular arquivos:

(1) **ASSIGN**: associa o nome do arquivo no programa com o nome externo (no disco, etc).

A **sintaxe:**
ASSIGN(<id-var-arquivo>,<'nome externo'>**)**
ou
ASSIGN(<id-var-arquivo>,<id-string>**)**

onde: - o arquivo não pode estar aberto.
- <id-var-arquivo> é o nome da variável de tipo file declarada no programa.Também se diz que é o "nome interno" do arquivo.
- o "nome externo" (escrito entre apóstrofos) é o nome com que está gravado no computador.
- <id-string> é uma variável do tipo *string* que deve ser lida pelo programa antes deste comando, e conter o "nome externo" do arquivo.

Observação: Cada arquivo possui um "apontador", (que é uma variável) que indica a posição do registro corrente, que está pronto para ser usado (ié, lido ou gravado).

---

(2) **REWRITE**: cria um arquivo novo (inicialmente vazio) deixando-o pronto para ser gravado. O apontador indica onde ficará o 1º registro

A **sintaxe:**
REWRITE(<id-var-arquivo>**)**

cuidado: Se já existir um arquivo com aquele nome externo passado pelo comando ASSIGN, ele é apagado e re-criado (porém vazio!).

(3) **RESET**: abre um arquivo já existente, deixando-o pronto para ser lido, com o apontador indicando o 1º registro. Neste caso, pode-se também gravar novos registros no arquivo, assim como "regravar" registros (com modificações).

A **sintaxe:**
RESET(<id-var-arquivo>**)**

obs: Se o arquivo já estiver aberto, ele é "reaberto", isto é, ele se re-posiciona no 1º registro. E, se ele não existir, ocorrerá ERRO de execução.

(4) **CLOSE**: fecha o arquivo, levando informações atualizadas ao Sistema Operacional sobre o arquivo externo associado.

A **sintaxe**:
CLOSE(<id-var-arquivo>**)**

(5) **EOF** ("END-OF-FILE"): situa o ultimo registro do arquivo. É uma função com resultado booleano: TRUE se o arquivo não tem mais elementos (ou se estiver vazio) ou FALSE em caso contrário. Ela permite o uso de um "laço" para leitura enquanto não for atingido o fim do arquivo.

A **sintaxe**:
EOF(<id-var-arquivo>**)**

(6) **READ**: Lê os dados do registro corrente e os armazena na variável indicada. A seguir, o apontador avança de uma posição, ié, posiciona-se no registro seguinte.

A **sintaxe**:
READ(<id-var-arquivo>,<id-var-tipo-dado>**)**

onde <id-var-tipo-dado> é o nome de uma variável do mesmo tipo (do registro) do arquivo.

(7) **WRITE**: grava no arquivo o conteúdo da variável indicada, posicionando o ponteiro no registro seguinte, ié,
avançando uma posição.

A **sintaxe**:
WRITE(<id-var-arquivo>,<id-var-tipo-dado>**)**

(8) **FILESIZE**: dá a quantidade de registros gravados no arquivo.

A **sintaxe**:
FILESIZE(<id-var-arquivo>**)**

(9) **SEEK**: posiciona o apontador do arquivo num determinado registro. Trata-se de um "acesso direto".

A **sintaxe**:
SEEK(<id-var-arquivo>,NUM**)**

onde: NUM é um valor inteiro no intervalo de 0(zero) a FILESIZE-1, que corresponde ao "número de ordem"
do registro.

(10) **FILEPOS**: É uma função que dá a posição do registro corrente. É normalmente utilizada como argumento do
procedimento SEEK.

A **sintaxe**: **FILEPOS**(<id-var-arquivo>)

Exemplos:
1) SEEK(ARQ,FILESIZE(ARQ)) – posiciona o apontador após o último registro gravado.
2) SEEK(ARQ,FILEPOS(ARQ) – 1) – posiciona o apontador no registro anterior.

---

(11) **ERASE(**<id-var-arquivo>**)**: apaga do disco (ou outra unidade) o arquivo associado (pelo comando ASSIGN)
à variável.

(12) **RENAME(**<id-var-arquivo>,<id-string>**)**: renomeia o arquivo do disco (...) com o (novo) nome dado
diretamente ou pela variável (como no comando ASSIGN). **Obs**: o arquivo deve estar fechado (pelo comando
CLOSE).

(13) **TRUNCATE(**<id-var-arquivo>**)**: apaga todos os registros do arquivo a partir do registro corrente.

- **Arquivos de tipo-*TEXT***:
...

– (o uso de) read/readln, write/writeln

– APPEND
– EOLN

Exemplos de "operações básicas" com arquivos:

(1) Criar (e gravar) arquivo (prog60)
(2) Exibir tudo (consulta total) (prog61)
(3) Exibir registro (consulta individual) (prog62)
(4) Exibir parte (consulta parcial) (prog63)
(5) Copiar total (prog64)
(6) Copiar parcial (ié, "selecionar") (prog65)
(7) *idem* em arquivo com outro registro (prog66)
(8) *idem* consultando dois arquivos (prog67)
(9) Ampliar (prog68)
(10) Modificarr registro (prog69)
(11) Incluir registro (prog70)
(12) Excluir registro (prog71)
(13) SORT

(14) MERGE