# Manipulação de Arquivos em Pascal

## Estrutura de Dados II

Aula 03

*Para implementar programa infantil para* ***ordenação (alfabética) de palavras****, que estrutura de dados você usaria? Por quê?*

*Considerando a base de dados abaixo, referente à* ***tabela de preços*** *de uma* ***papelaria****, que* ***estrutura de dados*** *você usaria para mantê-la?*

| CÓDIGO | PRODUTO | VALOR |
|---|---|---|
| 0042 | caderno | 5,00 |
| 0102 | caneta | 0,50 |
| 0003 | borracha | 0,25 |
| 0084 | lápis | 0,20 |

# *Definição de Arquivos*

- Os **arquivos** são **elementos de armazenamento de dados** residentes em memória secundária, ou seja, **memória não volátil**.

- Portanto são diferentes dos **outros itens de armazenamento de dados** (residentes na memória principal): registros, vetores, matrizes; os quais **não preservam os dados neles depositados**, após o término da execução dos programas que os definem.

# Declaração de Arquivos em Pascal

Do ponto de vista lógico, os arquivos são coleções de registros e são declarados:

```
var
   <Nome>: file of <TipoRegistro>;
```

Onde:

- `Nome` – nome do arquivo (interno);
- `TipoRegistro` – tipo dos registros que comporão o arquivo.

# *Aplicação de Arquivos*

*Considerando, por exemplo:*

- *A necessidade de cadastrar os pacientes a serem atendidos por um médico;*
- *Sendo que as consultas são marcadas com meses de antecedência.*

Para cadastro dos dados dos pacientes, um **array** de registros poderia ser usado?

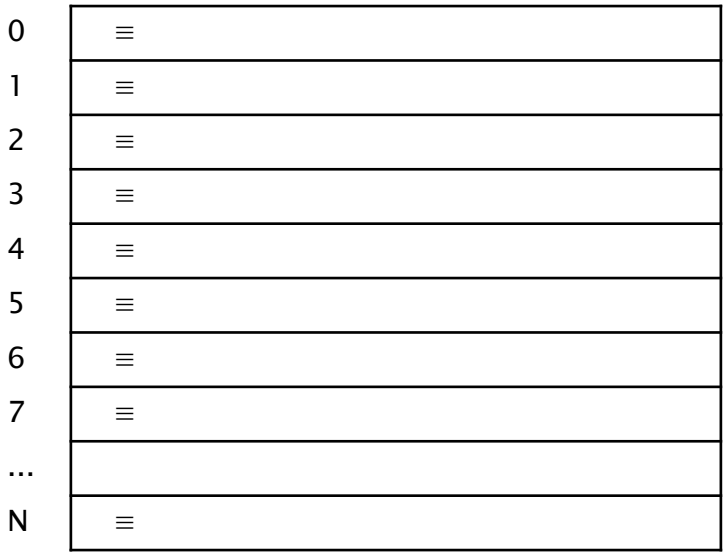
# Aplicação de Arquivos

Possível solução:

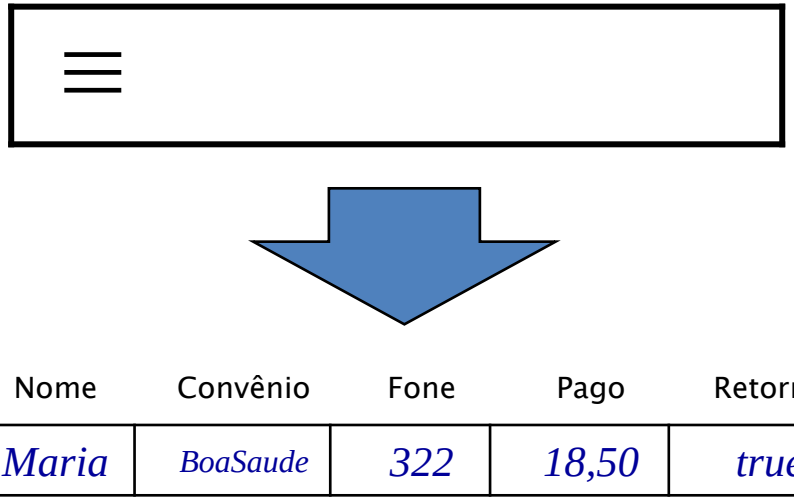
```
type
  TpRegMed = record
               Nome: string[20];
               Convenio,
               Fone: string[10];
               Pago: real;
               Retorno: boolean;
             end;
var
  ArqPacientes: file of TpRegMed;
```

Considerando a definição dada, nesta temos que ArqPacientes é um arquivo composto por registros formados por cinco campos: `nome`, `convênio`, `fone`, `(valor) pago` e `retorno`.

```
type
  TpRegMed = record
               Nome: string[20];
               Convenio,
               Fone: string[10];
               Pago: real;
               Retorno: boolean;
             end;
var
  ArqPacientes: file of TpRegMed;
```




ArqPacientes



# Associação a Nome Externo

- Para gerenciamento dos dados mantidos em arquivos, são necessários comandos específicos; de manipulação de arquivos.
- O `assign` é o procedimento que possibilita a associação do nome externo do arquivo ao seu nome lógico (interno).
- Os arquivos são elementos que existem independente da existência do programa que o manipula. Portanto, apresentam nome externo, através do qual, por exemplo, outros programas os identificam.

# Associação a Nome Externo

Através do nome externo o arquivo mantém-se disponível para outros programas.

```
assign(var <arqinterno>; <arqexterno>: string);
```

Onde:

- `<arqinterno>` é o nome lógico do arquivo, nome interno
- `<arqexterno>` é o nome externo do arquivo

# Associação a Nome Externo

Exemplo:

```
assign(ArqPacientes,'Consultas.dat');
```

- Com a execução do programa que manipula `ArqPacientes`, considerando os efeitos da instrução dada acima, o arquivo `Consultas.dat` externo, é associado ao arquivo interno `ArqPacientes`.
- Mesmo após a conclusão do referido programa, o arquivo `Consultas.dat` existirá, preservando os dados neste mantidos.

# Associação a Nome Externo

Exemplo:

```
assign(ArqPacientes,'C18nov08.dat');
...
assign(ArqPacientes,'C20nov08.dat');
```

- Um único arquivo lógico pode ser associado a diversos arquivos externos (um por vez), dependendo da situação problema a ser solucionada.
- Antes da definição do nome externo do arquivo pode ser descrito o caminho de localização deste: `'C:\HospitalDaSaude\DrCuraTudo\Consultas.dat'`

# Abrindo Arquivos

- Para manipulação de dados mantidos em arquivos é preciso abri-los.
- Em Pascal há dois comandos para abrir arquivos:

```
reset(var <Arquivo>);
rewrite(var <Arquivo>);
```

- O `reset` é um procedimento útil para abrir arquivos já existentes. Preserva os dados mantidos neste.
- Com o `reset`, caso o arquivo não exista, ocorre um erro de execução.

# *Abrindo Arquivos*

- Como já dito, tem-se:

```
reset(var <Arquivo>);
rewrite(var <Arquivo>);
```

- O `rewrite` é um procedimento útil para criar e abrir novos arquivos.
- Caso o arquivo já exista, o `rewrite` provoca a perda dos dados mantidos neste.

# *Abrindo Arquivos*

```
reset(ArqPacientes);
rewrite(ArqPacientes);
```

- Na prática, se usamos apenas o `reset` para abrir os arquivos manipulados por um programa; na primeira vez que o usuário executar a aplicação, como o cadastro não existe, o `reset` provocaria erro de execução.

# *Abrindo Arquivos*

```
reset(ArqPacientes);
rewrite(ArqPacientes);
```

- E se usamos, na prática, apenas o `rewrite`, para abrir os arquivos manipulados por um programa; a primeira vez que o usuário executar a aplicação, como o cadastro não existe, o `rewrite` provoca a criação do arquivo; mas em todas as outras vezes, que o arquivo (cadastro) deve apenas ser aberto, o `rewrite`, provoca a perda dos dados.

# *Abrindo Arquivos*

```
reset(ArqPacientes);
rewrite(ArqPacientes);
```

- Precisamos então: sendo a primeira vez que o usuário está executando o programa (o cadastro ainda não existe), o `rewrite` deve ser usado para criar o arquivo. E, em todas as outras vezes (o cadastro já existe e deve ser preservado) usar o `reset`.

# Abrindo Arquivos

```
procedure AbrirArquivo(Arquivo: TipoArquivo;
                       NomeArq: string);
begin
  assign(Arquivo,NomeArq);
  {$I-} reset(Arquivo); {$I+}
  if IOResult <> 0 then
    rewrite(Arquivo);
end;
```

- Neste tenta-se abrir o arquivo com `reset`, presumindo que o Arquivo já exista.
- Caso o arquivo não exista, com `{$I-}` é efetuado o desligamento da diretiva de compilação fazendo com que o erro não seja tratado através de mensagem ao usuário, mas que seja atribuído um código de erro à variável predefinida `IOResult`.

# Abrindo Arquivos

```
procedure AbrirArquivo(Arquivo: TipoArquivo;
                       NomeArq: string);
begin
  assign(Arquivo,NomeArq);
  {$I-} reset(Arquivo); {$I+}
  if IOResult <> 0 then
    rewrite(Arquivo);
end;
```

- Se a `IOResult` for diferente de zero é porque houve erro na tentativa de abrir o arquivo. Se houve erro é porque o arquivo não existia, então este deve ser criado – usando o `rewrite`.
- Se a `IOResult` for igual a zero é porque não houve erro na tentativa de abrir o arquivo. Ou seja, o arquivo já existia e foi devidamente aberto com `reset`. Neste caso o `rewrite` não precisa ser usado.

# Abrindo Arquivos

```
procedure AbrirArquivo(Arquivo: TipoArquivo;
                       NomeArq: string);
begin
  assign(Arquivo,NomeArq);
  {$I-} reset(Arquivo); {$I+}
  if IOResult <> 0 then
    rewrite(Arquivo);
end;
```

- Importante destacar que a abertura de um arquivo requer a associação prévia deste a um nome externo.
- Em aplicações que manipulam arquivo, a abertura deste deve ser efetuada no início da aplicação, e uma única vez, não deve ser feita então, por exemplo, em laços (de operações).

# *Fechando Arquivos*

```
close(var <Arquivo>);
```

- Através do comando `close` é possível fechar um arquivo.
- Vale esclarecer que a ausência do `close` não é identificada nem no processo de compilação, nem no de execução. Porém pode provocar danos nos dados mantidos no arquivo.
- O processo de fechar um arquivo deve ser efetuado ao final da manipulação deste e uma única vez. Assim, também, não deve ser mantido em laços de operações.

# *Lendo e Gravando Dados em Arquivos*

Para armazenamento, usar:

```
write(var <Arquivo>,<Registro>);
```

Para leitura, usar:

```
read(var <Arquivo>,var <Registro>);
```

- Exemplo, a instrução `write(Arq,R);` provoca o armazenamento do registro `R` no arquivo `Arq`. Mas, resta esclarecer em que posição.
- O `read` e o `write` são aplicados ao registro da posição corrente do arquivo.
- Para gerenciamento de um arquivo é mantido um ponteiro numérico que indica a posição (corrente) do registro a ser manipulado.

# *Lendo e Gravando Dados em Arquivos*

Para armazenamento, usar:

```
write(var <Arquivo>,<Registro>);
```

Para leitura, usar:

```
read(var <Arquivo>,var <Registro>);
```

- Inicialmente, logo após a abertura de um arquivo, o ponteiro lógico é ajustado para a **posição zero** deste.
- Havendo leituras (`read`) ou gravações (`write`), esta operação é efetuada sobre o registros que se encontra na posição corrente e o apontador é atualizado em uma posição.

# *Manipulando Arquivos*

- Em geral, no processo de inclusão de novos registros, estes devem ser adicionados ao final do arquivo, após o último registro. Para tanto, é útil a função `eof`.

```
eof(<Arquivo>):boolean;
```

- `eof` é uma função que retorna `true` quando é encontrado o fim do arquivo.

# *Manipulando Arquivos*

- Caso seja necessário manipular um registro que se encontra numa dada posição do arquivo, é útil o procedimento `seek`.

```
seek(var <Arquivo>; <posição>: longint);
```

- Considerando `seek(Arq,50);` o ponteiro é posicionado na posição 50 do arquivo `Arq;` então a próxima operação de leitura ou armazenamento será efetuada sobre o registro que se encontra nesta posição.

# *Manipulando Arquivos*

- Para identificar em que posição se encontra o ponteiro de um arquivo há a função `filepos`.

```
filepos(<Arquivo>):longint;
```

- E para identificar quantos registros compõem o arquivo é útil a função `filesize`.

```
filesize(<Arquivo>): longint;
```

- Vale esclarecer que o número de registros que compõem um arquivo também é denominado tamanho do arquivo.

# *Utilizando Arquivos*

- Arquivos devem ser utilizados em aplicações computacionais que manipulam dados que devem ser mantidos permanentemente: agenda telefônica, estoque, cadastros.
- Vale considerar que a memória permanente tem também grande capacidade de armazenamento de dados, mas é lenta em relação à memória principal.

# •ARQUIVOS
## aplicação


*Implementar programa de cadastro dos medicamentos de uma farmácia.*

# •CADASTRO FARMÁCIA
## dados

```
type
   Str20 = string[20];
   TpReg = record
           Nome: Str20;
           Preco: real;
           Estoque: integer; {Quantidade em estoque}
   end;
var
   ArqFarma: file of TpReg;
   RegFarma: TpReg;

   Opcao: char;
   CadastroVazio,
   Existe: boolean;
   Pos: longint;
```

## •CADASTRO FARMÁCIA principal

```
begin
  CadastroVazio:=false;
  assign(ArqFarma,'Farmas.dat');
  {$I-} reset(ArqFarma); {$I+}
  if IOResult <> 0 then rewrite(ArqFarma);
  repeat
    clrscr;  writeln('*** farma ***'); writeln;
    writeln(' 1 - incluir');
    writeln(' 2 - excluir');
    writeln(' 3 - editar estoque');
    writeln(' 4 - consultar');
    writeln(' 5 - listar todos');
    writeln(' 6 - encerrar');
    writeln; writeln('Opcao:');
    repeat Opcao:=readkey;
    until Opcao in ['1','2','3','4','5','6'];
    if Opcao in ['2','3','4','5'] then
     VerifiqueVazio(CadastroVazio);
    if not CadastroVazio then
      case Opcao of
      '1': incluir;
      '2': excluir;
      '3': alterar;
      '4': consultar;
      '5': listarTodos; end;
  until Opcao = '6';
  close(ArqFarma); end.
```

# •CADASTRO FARMÁCIA principal

```
procedure VerifiqueVazio(var inicio: boolean);
begin
  inicio:=false;
  if filesize(ArqFarma)=0 then begin
    inicio:=true;
    clrscr;
    writeln('impossivel realizar operação, cadastro
    vazio');
    writeln('pressione qualquer tecla');
    readln;
  end;
end;
```

# •CADASTRO FARMÁCIA
## inclusão

```
procedure incluir;
var
  R: char;
begin
  repeat
    clrscr;
    writeln('*** inclusao ***');
    writeln;
    with RegFarma do begin
      writeln('Nome: ');    readln(Nome);
      writeln('Preco: ');   readln(Preco);
      writeln('Estoque: '); readln(Estoque);
    end;
    seek(ArqFarma,filesize(ArqFarma));
    write(ArqFarma,RegFarma);
    writeln;
    writeln('Deseja efetuar nova inclusao? S/N');
    repeat
      R:=upcase(readkey);
    until (R='S') or (R='N');
  until R='N';
end;
```

# •CADASTRO FARMÁCIA
## consulta total

```
procedure listarTodos;
begin
  clrscr; writeln('*** lista todos ***');
  writeln;
  writeln('Nome':20,'Preco':12,'Estoque':10);
  seek(ArqFarma,0);
  while not eof(ArqFarma) do begin
    read(ArqFarma,RegFarma);
    if RegFarma.Nome <> 'XXX' then
    with RegFarma do
      writeln(Nome:20,Preco:10:2,Estoque:10);
  end;
  writeln;
  writeln('pressione qualquer tecla');
  readkey;
end;
```

## •CADASTRO FARMÁCIA
# consulta

```
procedure consultar;
var
  Remedio: Str20;
  R: char;
begin
  repeat
    clrscr; writeln('*** consulta ***');
    writeln;
    writeln('Nome: ');  readln(Remedio);
    localizar(Remedio,Existe,Pos);
    if Existe then
      with RegFarma do begin
        writeln('Nome: ',Nome);
        writeln('Preco: ',Preco);
        writeln('Estoque: ',Estoque);
      end
    else
      writeln('*** medicamento inexistente ***');
    writeln;
    writeln('Deseja efetuar nova consulta? S/N');
    repeat
      R:=upcase(readkey);
    until (R='S') or (R='N');
  until R = 'N';
end;
```

# •CADASTRO FARMÁCIA
## consulta

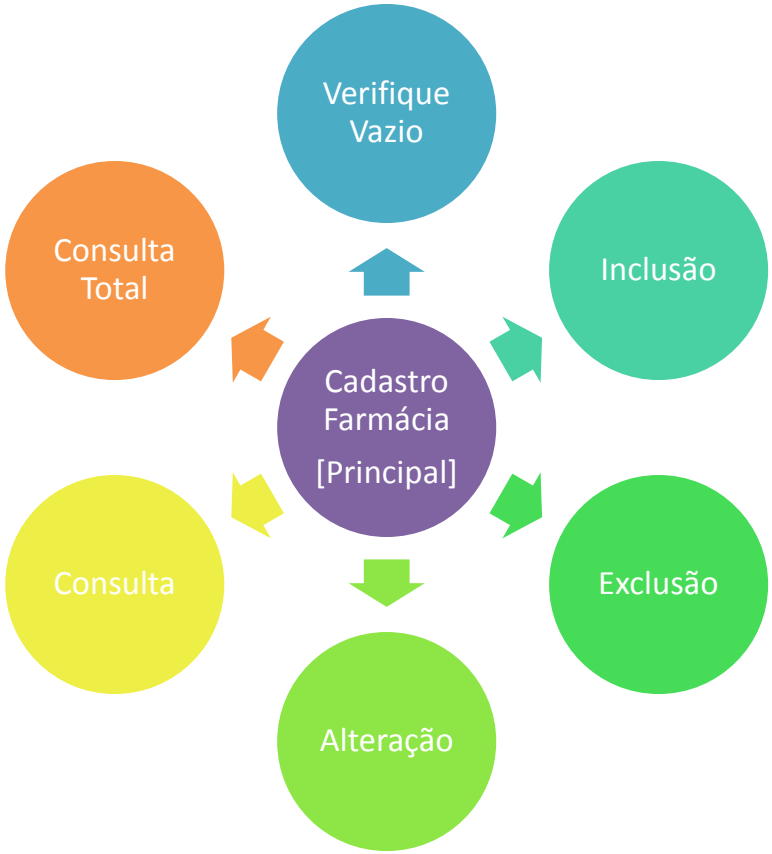
```
procedure localizar(N: Str20;
                    var Achou: boolean;
                    var Posicao: longint);
begin
  Achou:=false;
  Posicao:=-1;
  seek(ArqFarma,0);
  while not eof(ArqFarma) do begin
    read(ArqFarma,RegFarma);
    if RegFarma.Nome = N then begin
      Achou:=true;
      Posicao:=filepos(ArqFarma)-1; end;
  end;
end;
```

Por que continuar a busca mesmo já tendo localizado o elemento desejado?

# ARQUIVOS
## exercícios

1. *Implementar procedimento de exclusão (lógica).*
2. *Ajustar procedimento de inclusão de forma a evitar redundância de dados.*
3. *Implementar procedimento de alteração (ajuste do estoque).*

# ARQUIVOS
## exercícios




*Analisando a situação problema, que outras alterações (edições) são provavelmente necessárias?*

# ARQUIVOS
## exercícios


*Criar programa Pascal para manter as notas dos alunos da turma de Programação Imperativa: 3 notas (de 0 a 10).*

# *ARQUIVOS*

COMPLEMENTAR ESTUDOS:

**Fundamentos da Programação de Computadores**

*Ana Fernanda Gomes Ascencio*
*Edilene Aparecida Veneruchi de Campos*

# Capítulo Arquivo