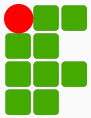
Pascal

06 - Manipulação de Arquivos

Marcos Roberto Ribeiro


Instituto Federal Minas Gerais - Campus Bambuí

2018

Introdução

- Até o momento estudamos diversas estruturas e elaboramos vários programas em Pascal, porém estes programas não são capazes de armazenar dados;
- Isto significa que quanto finalizamos um programa todos os dados inseridos no mesmo são perdidos. Quando abrimos tal programa posteriormente, somos obrigados a inserir todos os dados novamente;
- Uma maneira de manter os dados de um programa salvos para serem acessados em futuras execuções é através da utilização de arquivos;
- Exitem basicamente dois tipos de arquivos: os **arquivos estruturados** (ou de acesso aleatório) e os **arquivos de texto**.
- Os arquivos estruturados são baseados em um tipo de dado, principalmente registros;
- Já os arquivos de texto trabalham apenas com strings.

Arquivos Estruturados

- Para trabalharmos com arquivos estruturados o primeiro passo é definir o tipo do arquivo:

```
type
  TipoArquivo = file of TipoDado;
```

- Em seguida devemos declarar uma variável com tipo de arquivo definido:

```
var
  Arquivo: TipoArquivo;
```

- Depois de declarar uma variável arquivo precisamos associar esta variável a um arquivo no disco para que o programa possa realmente gravar e ler dados;
- A associação de variáveis a arquivos é feita com o procedimento **Assign(var Arquivo: TipoArquivo, NomeArquivo: String)**;
- Após a associação o arquivo poderá ser finalmente aberto.

Abrindo e Fechando Arquivos

- Um arquivo pode ser aberto pelos procedimentos **Rewrite(var Arquivo: TipoArquivo)** ou **Reset(var Arquivo: TipoArquivo)**;
- O procedimento **Rewrite()** cria um novo arquivo em branco e o procedimento **Reset()** é utilizado para abrir arquivos existentes. Portanto antes de abrirmos o arquivo devemos saber se o mesmo já existe;
- A **unit sysutils** possui a função **FileExists(NomeArquivo:String):Boolean** para realizar esta tarefa de verificação.
- Depois que abrimos um arquivo, podemos fazer realizar leituras e escritas (como veremos mais adiante) e antes de encerrar o programa devemos fechar o arquivos com o procedimento **Close(var Arquivo: TipoArquivo)**;
- Abaixo temos um programa simples que simplesmente abre e fecha um arquivo:

Unit alunos I

```
unit alunos;

interface

type
  TAluno = record          {Registro TAluno}
   Matricula: Integer;
   Nome: String[50];
   Nota: Real;
  end;
  TArq = file of TAluno; {Arquivo de registros TAluno}

procedure AbreArquivo(var Arq: TArq; NomeArq: string);

implementation
```

Unit alunos II

```
uses sysutils;

{Procedimento para abrir o arquivo}
procedure AbreArquivo(var Arq: TArq; NomeArq: string);
begin
  Assign(Arq, NomeArq);   {Associa o arquivo ao nome}
  {Verifica se o arquivo existe}
  if FileExists(NomeArq) then begin
    Reset(Arq);           {Abre arquivo existente}
  end else begin
    Rewrite(Arq);         {Cria arquivo em branco}
  end;
end;

END.
```

Programa que Abre e Fecha Arquivo

```
program AbreFechaArquivo;

uses alunos;

var
  Arq: TArq;
  NomeArq: String;

BEGIN
  NomeArq := 'alunos.bin';
  AbreArquivo(Arq, NomeArq);          {Abre o arquivo}
  Close(Arq);                         {Fecha o arquivo}
END.
```

Escrevendo Registros

- A escrita de dados em arquivos estruturados é realizada com o procedimento **Write(var Arquivo: TipoArquivo; Dado: TipoDado)**;
- Quando escrevemos dados em um arquivo devemos garantir que os registros sejam acrescentados sempre no final do arquivo. Todavia quando abrimos um arquivo, somos posicionados em seu primeiro registro, precisamos mover para o final do arquivo para evitar a escrita sobre dados já gravados;
- Para posicionarmos em um lugar específico do arquivo de forma direta podemos utilizar a função **Seek(var Arquivo: TipoArquivo, N: Integer)**, onde **N** é a posição desejada. É importante que a posição **N** não seja maior que a posição final do arquivo, pois ocorrerá erro nesta situação;
- Para chegarmos a posição final do arquivo podemos utilizar a função **FileSize(var Arquivo: TipoArquivo): Integer** que retorna o número de posições do arquivo;
- Vamos, agora, criar uma função para adicionar dados no arquivo e escrever um programa para utilizá-la.

Procedimento para Adicionar Registros ao Arquivo

```
unit alunos;

interface
...
procedure AdicionaArquivo(var Arq: TArq; A: TAluno);

implementation
...
{Procedimento para adicionar novo registro ao arquivo}
procedure AdicionaArquivo(var Arq: TArq; A: TAluno);
begin
  {Posiciona no final do arquivo}
  Seek(Arq, FileSize(Arq));
  {Adciona o novo registro}
  Write(Arq, A);
  {Posiciona no registro adicionado}
  Seek(Arq, FileSize(Arq) - 1);
end;
...
```

Programa para Adicionar Alunos I

```
program AdicionaAlunos;

uses alunos;

var
  Arq: TArq;
  NomeArq: String;
  Aluno: TAluno;

BEGIN
  NomeArq := 'alunos.bin';
  {Abre o arquivo}
  AbreArquivo(Arq, NomeArq);
  WriteLn('Informe os dados');
  repeat
    Write('Matrícula (0 para parar): ');
```

Programa para Adicionar Alunos II

```
    ReadLn(Aluno.Matricula);
    if (Aluno.Matricula <> 0) then begin
      Write('Nome: ');
      ReadLn(Aluno.Nome);
      Write('Nota: ');
      ReadLn(Aluno.Nota);
      AdicionaArquivo(Arq, Aluno);
    end;
  {Até que a matrícula seja 0}
  until (Aluno.Matricula = 0);
  {Fecha o arquivo}
  Close(Arq);
END.
```

Lendo Registros

- Um arquivo estruturado assemelha-se a um vetor de dados possuindo diversos registros, um em cada posição;
- A leitura de dados em arquivos é feita com o procedimento **Read(var Arquivo: TipoArquivo; var Dado: TipoDado)** que lê o conteúdo da posição atual do arquivo, grava os dados desta posição em **Dado** e passa para a posição seguinte;
- A maneira mais simples de ler o conteúdo de um arquivo é abri-lo e fazer diversas leituras com o **Read();** até chegarmos ao final do arquivo;
- O Problema é saber quando chegamos ao final. Para resolver este problema podemos utilizar a função **EOF(Arquivo: TipoArquivo): Boolean** que retorna **True** quando estamos na última posição do arquivo;
- Para entendermos melhor vamos criar um programa que lê os dados já existentes em um arquivo.

Programa com Leitura de Registros em Arquivo I

```
program LeAlunos;

uses alunos;

var
  Arq: TArq;
  NomeArq : String;
  Aluno: TAluno;

BEGIN
  NomeArq := 'alunos.bin';
  {Abre o arquivo}
  AbreArquivo(Arq, NomeArq);
  WriteLn('Lista de alunos:');
```

Programa com Leitura de Registros em Arquivo II

```
  WriteLn('Matrícula | Nome | Nota');
  repeat
    Read(Arq, Aluno);
    WriteLn(Aluno.Matricula, ' | ', Aluno.Nome, ' | ',
↪   Aluno.Nota:0:2);
  {Até chegar no fim do arquivo}
  until (EOF(Arq));
  {Fecha o arquivo}
  Close(Arq);
END.
```

Localizando Registros

- O programa anterior lê todos os registros do arquivo de forma sequencial. A leitura começa pelo primeiro registros e vai até o último;
- Muitas vezes é necessário navegar entre os registros de um arquivo, por exemplo, precisamos alterar a nota do aluno que está na segunda posição ou na terceira. Nestas situações podemos utilizar a rotina **Seek**, explicada anteriormente;
- Para sabermos a posição atual dentro de um arquivo podemos usar a função **FilePos(var Arquivo: TipoArquivo): Integer**;
- Vamos acrescentar a função **PesquisaAluno** na **Unit Alunos** e desenvolver um programa que pesquisa alunos pela matrícula.

Unit Alunos com Pesquisa de Registros I

```
interface
...
function BuscaArquivo(var Arq: TArq; Matricula: Integer): Integer;
...
implementation
...
{Função que pesquisa aluno pela matrícula}
function BuscaArquivo(var Arq: TArq; Matricula: Integer): Integer;
var
  {Registro par leitura do arquivo}
  A: TAluno;
  {Armazena posição do registro localizado}
  Posicao: Integer;
begin
  {Inicialmente não encontramos nada}
  Posicao := -1;
```

Unit Alunos com Pesquisa de Registros II

```
  {Posiciona na primeira posição do arquivo}
  Seek(Arq, 0);
  repeat
    {Lê a posição atual}
    Read(Arq, A);
    {Testa se o registro da posição é  o procurado}
    if (A.Matricula = Matricula) then begin
      {Armazena a posição do registro localizado}
      Posicao := FilePos(Arq) - 1;
    end;
  until (EOF(Arq) or (Posicao <> -1);
  BuscaArquivo := Posicao;
end;
...
```

Programa com Pesquisa de Registros I

```
program PesquisaAluno;

uses alunos;

var
...
  Indice: Integer;

BEGIN
  NomeArq := 'alunos.bin';
  {Abre o arquivo}
  AbreArquivo(Arq, NomeArq);
  Write('Informe a matrícula a ser procurada: ');
  ReadLn(Aluno.Matricula);
```

Programa com Pesquisa de Registros II

```
  Indice := BuscaArquivo(Arq, Aluno.Matricula);
  if (Indice <> -1) then begin
    Seek(Arq, Indice);
    ReadLn(Arq, Aluno);
    WriteLn(Aluno.Matricula, ' | ', Aluno.Nome, ' | ',
↪   Aluno.Nota:0:2);
  end else begin
    WriteLn('Registro não encontrado');
  end;
  {Fecha o arquivo}
  Close(Arq);
END.
```

Alterando Registros

- A alteração de um registro é feita em duas etapas:
  1. Movemos para a posição do registro a ser alterado com o procedimento **Seek()**;
  2. Escrevemos sobre o registro desta posição;
- Abaixo temos um procedimento que realiza esta tarefa:

```
{Procedimento para modificar um registro no arquivo}
procedure ModificaArquivo(var Arq: TArq; Posicao: Integer; A:
 ↪  TAluno);
begin
  {Posiciona no registro a ser modificado}
  Seek(Arq, Posicao);
  {Sobrescreve o registro a ser modificado}
  Write(Arq, A);
end;
```

Apagando Registos

- Para apagarmos registros, podemos usar a instrução **Truncate(var Arquivo: TArquivo)**;
- O problema é que esta instrução apaga todos os registros após a posição atual;
- Para resolver este problema, ao apagar um registro em uma posição **n**, devemos copiar todos os registros depois de **n** para sua posição anterior e depois apagar apenas o último registro (que ficará duplicado);
- Para ilustrar melhor este processo vamos considerar um arquivo representado graficamente abaixo, onde desejamos apagar o registro **C**;

- O primeiro passo é copiar os registros depois de **C** para suas posições anteriores;

- Após a cópia, o último registro encontra-se duplicado;

- Podemos então aplicar o procedimento **Truncate()** na última posição;

Apagando Registos

- Para apagarmos registros, podemos usar a instrução **Truncate(var Arquivo: TArquivo)**;
- O problema é que esta instrução apaga todos os registros após a posição atual;
- Para resolver este problema, ao apagar um registro em uma posição **n**, devemos copiar todos os registros depois de **n** para sua posição anterior e depois apagar apenas o último registro (que ficará duplicado);
- Para ilustrar melhor este processo vamos considerar um arquivo representado graficamente abaixo, onde desejamos apagar o registro **C**;

- O primeiro passo é copiar os registros depois de **C** para suas posições anteriores;

- Após a cópia, o último registro encontra-se duplicado;

- Podemos então aplicar o procedimento **Truncate()** na última posição;

Apagando Registos

- Para apagarmos registros, podemos usar a instrução **Truncate(var Arquivo: TArquivo)**;
- O problema é que esta instrução apaga todos os registros após a posição atual;
- Para resolver este problema, ao apagar um registro em uma posição **n**, devemos copiar todos os registros depois de **n** para sua posição anterior e depois apagar apenas o último registro (que ficará duplicado);
- Para ilustrar melhor este processo vamos considerar um arquivo representado graficamente abaixo, onde desejamos apagar o registro **C**;

- O primeiro passo é copiar os registros depois de **C** para suas posições anteriores;

- Após a cópia, o último registro encontra-se duplicado;

- Podemos então aplicar o procedimento **Truncate()** na última posição;

Apagando Registos

- Para apagarmos registros, podemos usar a instrução **Truncate(var Arquivo: TArquivo)**;
- O problema é que esta instrução apaga todos os registros após a posição atual;
- Para resolver este problema, ao apagar um registro em uma posição **n**, devemos copiar todos os registros depois de **n** para sua posição anterior e depois apagar apenas o último registro (que ficará duplicado);
- Para ilustrar melhor este processo vamos considerar um arquivo representado graficamente abaixo, onde desejamos apagar o registro **C**;

- O primeiro passo é copiar os registros depois de **C** para suas posições anteriores;

- Após a cópia, o último registro encontra-se duplicado;

- Podemos então aplicar o procedimento **Truncate()** na última posição;

Procedimento RemoveArquivo()

```
{Procedimento para remover o registro de um arquivo}
procedure RemoveArquivo(var Arq: TArq; Posicao: Integer);
var
  Aluno: TAluno;
begin
  Posicao:= Posicao + 1;
  {Percorre todos os registros a partir da posição atual até chegar no final do
 ↪   arquivo}
  While (Posicao < FileSize(Arq)) do begin
    {Copia cada registro para sua posição anterior}
    Seek(Arq, Posicao);
    Read(Arq, Aluno);
    Seek(Arq, Posicao - 1);
    Write(Arq, Aluno);
    Posicao:= Posicao + 1;
  end;
  {Posiciona no último registro do arquivo}
  Seek(Arq, FileSize(Arq) - 1);
  {Apaga o último registro que está duplicado}
  Truncate(Arq);
end;
```

Programa Completo com Arquivo (unit alunos) I

```
unit alunos;

interface

type
  TAluno = record
   Matricula: Integer;
   Nome: String[50];
   Nota: Real;
  end;
  TArq = file of TAluno;

procedure AbreArquivo(var Arq: TArq; NomeArq: string);
function LeArquivo(var Arq:TArq; Posicao: Integer):TAluno;
procedure AdicionaArquivo(var Arq: TArq; A: TAluno);
procedure ModificaArquivo(var Arq: TArq; Posicao: Integer; A: TAluno);
procedure RemoveArquivo(var Arq: TArq; Posicao: Integer);
function BuscaArquivo(var Arq: TArq; Matricula: Integer): Integer;
procedure NavegaArquivo(var Arq:TArq; Direcao:Integer);
```

Programa Completo com Arquivo (unit alunos) II

```
implementation

uses sysutils; {Unit sysutils para utilizar a função FileExists}

{Função que retorna o aluno de uma posição do arquivo}
function LeArquivo(var Arq:TArq; Posicao: Integer):TAluno;
var
  A: TAluno;
begin
  {Testa se a posição a ser lida é válida}
  if (Posicao >= FileSize(Arq)) or (Posicao < 0) then begin
    Posicao:= 0;
  end;
  {Posiciona na posição a ser lida}
  Seek(Arq, Posicao);
  {Verifica se não está no final do arquivo
  Isto pode acontecer se o arquivo estiver em branco}
  if not EOF(Arq) then begin
    Read(Arq, A);
  end;
  {A leitura passa para a posição seguinte
```

Programa Completo com Arquivo (unit alunos) III

```
  A próxima instrução posiciona na posição que foi lida }
  Seek(Arq, Posicao);
  {Retorna o registro lido}
  LeArquivo:= A;
end;

{Pocedimento que navega no arquivo}
procedure NavegaArquivo(var Arq:TArq; Direcao:Integer);
var
  Atual: Integer;
begin
  {Obtém a posição atual no arquivo}
  Atual:= FilePos(Arq);
  {Obtém a posição a ser movimentada}
  Atual:= Atual + Direcao;
  {Testa se a posição não está além da última posição}
  if (Atual >= FileSize(Arq)) then begin
    Atual := FileSize(Arq) - 1;
  end else begin
    {Testa se a posição não está antes da primeira posição}
    if (Atual < 0) then begin
```

Programa Completo com Arquivo (unit alunos) IV

```
      Atual := 0;
    end
  end;
  {Movimenta para a posição adequada}
  Seek(Arq, Atual);
end;

{Procedimento para adicionar novo registro ao arquivo}
procedure AdicionaArquivo(var arq: TArq; A: TAluno);
begin
  {Posiciona no final do arquivo}
  Seek(Arq, FileSize(Arq));
  {Adciona o novo registro}
  Write(Arq, A);
  {Posiciona no registro adicionado}
  Seek(Arq, FileSize(Arq) - 1);
end;
```

Programa Completo com Arquivo (Programa) I

```
program AlunoCompleto;

uses
  {Utiliza a unit alunos para as rotinas de arquivo}
  alunos,
  {Utiliza a unit crt para rotinas de tela}
  crt;

var
  Arq: TArq;            {Controle do arquivo}
  NomeArq: String;      {Nome do arquivo}
  Atual: Integer;       {Posição atual no arquivo}
  Quantidade: Integer; {Quantidade de registros do arquivo}
  Opcao: Char;          {Opção escolhida pelo usuário}

{Procedimento para exibir o aluno atual}
procedure Exibe();
var
  Aluno: TAluno; {Aluno a ser obtido no arquivo}
begin
```

Programa Completo com Arquivo (Programa) II

```
  {Limpa a tela}
  ClrScr();
  {Exibe opções na tela}
  WriteLn('Cadastro de Alunos');
  WriteLn('(I)nserir, (M)odificar, (R)emover, (A)nterior');
  WriteLn('(P)róximo, (L)istar, (B)uscar, (S)air');
  {Verifica se existem alunos}
  if (Quantidade > 0) then begin
    {Exibe o aluno da posição atual}
    WriteLn('Posicao: ', Atual + 1, ' de ', Quantidade);
    Aluno := LeArquivo(Arq, Atual);
    WriteLn('Matrícula: ', Aluno.Matricula);
    WriteLn('Nome: ', Aluno.Nome);
    WriteLn('Nota: ', Aluno.Nota:0:2);
  end else begin
    WriteLn('Nenhum aluno cadastrado!');
  end;
end;
```

Programa Completo com Arquivo (Programa) III

```
{Função que lê os dados de um aluno e retorna um registro com os mesmos}
function LeDadosAluno(): TAluno;
var
  Aluno: TAluno;
begin
  Write('Matricula: ');
  ReadLn(Aluno.Matricula);
  Write('Nome: ');
  ReadLn(Aluno.Nome);
  Write('Nota: ');
  ReadLn(Aluno.Nota);
  LeDadosAluno:= Aluno;
end;
```

Programa Completo com Arquivo (Programa) IV

```
{Procedimento que adiciona um novo aluno}
procedure AdicionaAluno();
var
  Aluno: TAluno;
begin
  ClrScr();
  WriteLn('Informe os dados do novo aluno');
  {Obtém os dados do novo aluno}
  Aluno:= LeDadosAluno();
  {Rotina para adicionar o registro no arquivo}
  AdicionaArquivo(Arq, Aluno);
  WriteLn('Aluno Cadastrado!');
  WriteLn('Pressione qualquer tecla para continuar');
  ReadKey();
end;
```

Programa Completo com Arquivo (Programa) V

```
{Modifica os dados do aluno atual}
procedure ModificaAluno();
var
  Aluno: TAluno;
begin
  WriteLn('Informe os novos dados do aluno');
  {Obtém os novos dados do aluno}
  Aluno:= LeDadosAluno();
  {Rotina para modificar o registro no arquivo}
  ModificaArquivo(Arq, Atual, Aluno);
  WriteLn('Aluno Modificado!');
  WriteLn('Pressione qualquer tecla para continuar');
  ReadKey();
end;

{Remove o aluno atual}
procedure RemoveAluno();
var
  Resposta: Char;  {Armazena resposta do usuário}
begin
```

Programa Completo com Arquivo (Programa) VI

```
  WriteLn('Pressione S para confirmar a remoção');
  {Obtém a resposta do usuário}
  Resposta:= UpCase(ReadKey());
  {Se a resposta for Sim}
  if (Resposta = 'S') then begin
    {Rotina para apagar o registro no arquivo}
    RemoveArquivo(Arq, Atual);
    {Se se o último aluno foi removido}
    if (Atual = Quantidade - 1) then begin
      {O penúltimo aluno passa a ser o último}
      Atual := Atual - 1;
    end;
    {Lê o aluno que "ocupará" a posição do excluído}
    LeArquivo(Arq, Atual);
    WriteLn('Aluno Removido!');
    WriteLn('Pressione qualquer tecla para continuar');
    ReadKey();
  end;
end;
```

Programa Completo com Arquivo (Programa) VII

```
{Procedimento que lista todos os arquivos cadastrados}
procedure ListaAlunos();
var
  Aluno: TAluno;
  Cont: Integer;
begin
  ClrScr();
  WriteLn('Listagem de alunos');
  WriteLn('Matrícula | Nome | Nota');
  {Laço de repetição para percorrer todos os alunos cadastrados}
  for Cont:= 0 to Quantidade -1 do begin
    Aluno := LeArquivo(Arq, Cont);
    WriteLn(Aluno.Matricula, ' | ', Aluno.Nome, ' | ', Aluno.Nota:0:2);
  end;
  WriteLn('Pressione qualquer tecla para continuar');
  ReadKey();
end;
```

Programa Completo com Arquivo (Programa) VIII

```
{Procedimento que busca um aluno pela matrícula}
procedure BuscaAluno();
var
  Matricula, Posicao: Integer;
begin
  ClrScr();
  Write('Informe a matrícula do aluno a ser buscado: ');
  ReadLn(Matricula);
  {Rotina que busca o registro dentro do arquivo}
  Posicao:= BuscaArquivo(Arq, Matricula);
  {Se a rotina retornar -1 é porque o aluno não foi encontrado}
  if (Posicao = -1) then begin
    LeArquivo(Arq, Atual);  {Volta para o aluno atual}
    WriteLn('Aluno não encontrado!');
    WriteLn('Pressione qualquer tecla para continuar');
    ReadKey();
  end;
end;
```

Programa Completo com Arquivo (Programa) IX

```
BEGIN {Bloco principal do programa}
  NomeArq:= 'alunos.bin';
  {Abre o arquivo}
  AbreArquivo(Arq, NomeArq);
  repeat
    {Obtém a posição atual}
    Atual:= FilePos(Arq);
    {Obtém a quantidade de alunos}
    Quantidade:= FileSize(Arq);
    {Rotina para exibir aluno atual}
    Exibe();
    {Obtém opção do usuário e a converte para maiúscula}
    Opcao:= UpCase(ReadKey());
    case Opcao of                    {Verifica a opção escolhida}
      'I': AdicionaAluno();          {Adicionar aluno}
      'M': ModificaAluno();          {Modificar aluno}
      'A': NavegaArquivo(Arq, -1); {Vai para a posição anterior}
      'P': NavegaArquivo(Arq, 1);  {Vai para a próxima posição}
      'L': ListaAlunos();  {Lista todos os alunos}
      'B': BuscaAluno();   {Rotina para buscar um aluno}
```

Programa Completo com Arquivo (Programa) X

```
      'R': RemoveAluno();  {Rotina para remover o aluno}
      else begin  {Testa se foi escolhida uma opção inválida}
        if (Opcao <> 'S') then begin
          WriteLn('Opção Inválida');
          Delay(300);
        end;
      end;
    end;
  until (Opcao = 'S');  {Somente a opção S finaliza o programa}
  Close(Arq);           {Fecha o arquivo}
END.
```

Arquivos de Texto I

- Os arquivos de texto, diferente dos arquivos estruturados, armazenam cadeias de caracteres.
- No caso dos arquivos estruturados, os dados são armazenados de forma binária e não é possível lê-los em um editor de textos;
- Já nos arquivos de texto, todos os dados são armazenados na forma de **String**, desta maneira, podemos visualizar seu conteúdo em um editor de textos qualquer;
- Para trabalharmos com um arquivo de texto devemos declarar uma variável do tipo **Text**;
- Um arquivo de texto pode ser aberto normalmente pelos procedimentos **Rewrite()** e **Reset()** depois de fazermos a associação com o **Assign()**;
- Também podemos utilizar as rotinas **Close()** e **EOF()** com arquivos de texto;

Arquivos de Texto II

- No caso dos arquivos de texto existe mais um procedimento para abertura que é o **Append(var Arquivo: Text)**. A diferença entre o **Reset()** e o **Append()** é que o segundo, além de abrir o arquivo, posiciona no final do arquivo;
- A leitura e escrita de dados em arquivos de texto é realizada pelos procedimentos **Read()** e **Write()** de forma semelhante a leitura e escrita na tela;
- É importante lembrar que em arquivos de texto não podemos utilizar a rotina **Seek()** e portanto não podemos transitar livremente pelo arquivo. Deste modo, os arquivos de texto são mais adequados para situações onde a navegação pelos dados não é necessária, por exemplo, gravação de logs;
- Como exemplo vamos criar um programa que escreva dados e outro que lê dados em um arquivo de texto.

Programa para Gravar em Arquivos de Texto I

```
program EscreveTexto;

uses sysutils;

var
  Arq: Text;              {Controle do arquivo}
  NomeArq: String;        {Nome do arquivo}
  Matricula: Integer;     {Matrícula do aluno}
  Nome: String[50];       {Nome do aluno}
  Nota: Real;             {Nota do aluno}
```

Programa para Gravar em Arquivos de Texto II

```
{Procedimento para abrir o arquivo}
procedure AbreArquivo(var Arq: Text; NomeArq: string);
begin
  {Associa o arquivo ao nome}
  Assign(Arq, NomeArq);
  {Verifica se o arquivo existe}
  if FileExists(NomeArq) then begin
    {Abre arquivo existente e posiciona no final}
    Append(Arq);
  end else begin
    {Cria arquivo em branco}
    Rewrite(Arq);
  end;
end;
```

Programa para Gravar em Arquivos de Texto III

```
BEGIN
  NomeArq:= 'alunos.txt';
  AbreArquivo(Arq, NomeArq);
  WriteLn('Informe os dados (Matrícula = 0 finaliza).');
  repeat
    Write('Matricula: ');
    ReadLn(Matricula);
    {Verifica se a matrícula é válida}
    if (Matricula <> 0) then begin
      Write('Nome: ');
      ReadLn(Nome);
      Write('Nota: ');
      ReadLn(Nota);
      {Grava dados no arquivo}
      WriteLn(Arq, Matricula, ', ', Nome, ', ', Nota:0:2);
    end;
  until (Matricula = 0);
  Close(Arq);
END.
```

Programa para Ler Arquivos de Texto I

```
program LeTexto;

uses sysutils;

var
  Arq: Text;              {Controle do arquivo}
  NomeArq: String;        {Nome do arquivo}
  Linha: String;
```

Programa para Ler Arquivos de Texto II

```
{Procedimento para abrir o arquivo}
procedure AbreArquivo(var Arq: Text; NomeArq: string);
begin
  Assign(Arq, NomeArq);
  if FileExists(NomeArq) then begin
    Reset(Arq);
  end else begin
    Rewrite(Arq);
  end;
end;
```

Programa para Ler Arquivos de Texto III

```
{Bloco principal do programa}
BEGIN
  NomeArq:= 'alunos.txt';
  AbreArquivo(Arq, NomeArq);
  WriteLn('Os dados do arquivo são:');
  WriteLn('Matrícula, Nome, Nota');
  {Lê e imprime todas as linhas do arquivo}
  while (not EOF(Arq)) do begin
    Read(Arq, Linha);
    WriteLn(Linha);
  end;
  Close(Arq);
END.
```

Referências I

Canneyt, M. V.
Free pascal reference guide.
`http://www.freepascal.org/docs-html/ref/ref.html`.

Evaristo, J. (1999).
***Programando com Pascal*.**
Book Express, Rio de Janeiro, 2 edition.

Manzano, J. A. N. G. and Yamatumi, W. Y. (2001).
***Programando em turbo pascal 7.0 e free pascal compiler*.**
Érica, São Paulo, 9 edition.