```
 1: unit ArvBin;
 2: interface
 3: type
 4:    Tipo_do_Dado = string[50];
 5:    TDirecao  = (NoEsquerdo, NoDireito, NoPai, NoRaiz);
 6:    ArvoreBinaria = ^No;
 7:    No = record
 8:            Dado  : Tipo_do_Dado;
 9:            Links : array[NoEsquerdo..NoPai] of ArvoreBinaria;
10:         end;
11:
12:    ParamVisite = procedure(Arvore : ArvoreBinaria);
13:
14:    procedure Inicializar(var Arvore : ArvoreBinaria);
15:    function  Vazia(var Arvore : ArvoreBinaria) : Boolean;
16:    procedure CriarNo(var Arvore : ArvoreBinaria; Dado : Tipo_do_Dado);
17:    function  ExisteNo(Arvore : ArvoreBinaria; Direcao : TDirecao) : boolean;
18:    procedure Deslocar(var Arvore : ArvoreBinaria; Direcao : TDirecao);
19:    procedure ObterDado(var Arvore : ArvoreBinaria; var Dado : Tipo_do_Dado);
20:    procedure AlterarDado(Arvore : ArvoreBinaria; Dado : Tipo_do_Dado);
21:    procedure AdicionarFilho(Arvore : ArvoreBinaria; Direcao : TDirecao; Dado
    : Tipo_do_Dado);
22:    procedure DisposeArvore(Arvore : ArvoreBinaria);
23:    procedure Deltree(var Arvore : ArvoreBinaria);
24:    procedure PreOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
25:    procedure InOrdem(Arvore  : ArvoreBinaria; Visite : ParamVisite);
26:    procedure PosOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
27:    procedure Caracteristicas(Arvore             : ArvoreBinaria;
28:                              var NumNos, Altura : integer;
29:                              var CompMedio      : real);
30: implementation
31:
32: (*************************************************************************)
33:
34: procedure Inicializar(var Arvore : ArvoreBinaria);
35: {
36: | Objetivo: Inicializa a arvore, tornando-a vazia, ou seja, atribuindo o
    valor nil
37: }
38: begin
39:    Arvore := nil
40: end;
41:
42: (*************************************************************************)
43:
44:
45: function Vazia(var Arvore : ArvoreBinaria) : boolean;
46: {
47: | Objetivo: Retorna true se o Arvore tem o valor nil
48: }
49: begin
50:    Vazia := Arvore = nil
51: end;
52:
53: (*************************************************************************)
54:
55: procedure CriarNo(var Arvore : ArvoreBinaria; Dado : Tipo_do_Dado);
56: {
57: | Objetivo: Cria um No, colocando neste Dado, e fazendo Arvore apontar para
58: |           ele. Arvore deve ter o valor nil, para indicar que nao aponta
59: |           para nenhum No.
```

```
 60: }
 61: var Dir : TDirecao;
 62: begin
 63:    if not Vazia(Arvore) then
 64:       begin
 65:          writeln('ERRO: O no deve estar vazio para ser criado.');
 66:          halt;
 67:       end;
 68:
 69:    new(Arvore);
 70:    Arvore^.Dado := Dado;
 71:
 72:    for Dir := NoEsquerdo to NoPai do
 73:       Inicializar(Arvore^.Links[Dir]);
 74: end;
 75:
 76: (*************************************************************************)
 77:
 78: function ExisteNo(Arvore : ArvoreBinaria; Direcao : TDirecao) : boolean;
 79: {
 80: | Objetivos: Testa se o No indicado por Direcao a partir de Arvore existe
 81: }
 82: begin
 83:    if Vazia(Arvore) then
 84:       ExisteNo := false
 85:    else
 86:       if Direcao = NoRaiz then
 87:          ExisteNo := true
 88:       else
 89:          ExisteNo := not Vazia(Arvore^.Links[Direcao]);
 90: end;
 91:
 92: (*************************************************************************)
 93:
 94: procedure Deslocar(var Arvore : ArvoreBinaria; Direcao : TDirecao);
 95: {
 96: | Objetivos: Deslocar o apontador Arvore para o No indicado por Direcao
 97: }
 98: begin
 99:    if Direcao = NoRaiz then
100:       while ExisteNo(Arvore, NoPai) do
101:          Arvore := Arvore^.Links[NoPai]
102:    else
103:       Arvore := Arvore^.Links[Direcao];
104: end;
105:
106: (*************************************************************************)
107:
108: procedure ObterDado(var Arvore : ArvoreBinaria; var Dado : Tipo_do_Dado);
109: {
110: | Objetivos: O parametro Dado recebe o dado contido no No apontado por Arvore
111: }
112: begin
113:    if Vazia(Arvore) then
114:       begin
115:          writeln('ERRO: Dado nao pode ser recuperado de um No vazio');
116:          halt;
117:       end;
118:    Dado := Arvore^.Dado;
119: end;
120:
```

```
121: (*****************************************************************************)
122:
123: procedure AlterarDado(Arvore : ArvoreBinaria; Dado : Tipo_do_Dado);
124: {
125: | Objetivos: Coloca no No apontado por Arvore, o dado contido em Dado.
126: }
127: begin
128:    if Vazia(Arvore) then
129:       begin
130:          writeln('ERRO: Nao pode ser alterado o dado de um No vazio.');
131:          halt;
132:       end;
133:    Arvore^.Dado := Dado;
134: end;
135:
136: (*****************************************************************************)
137:
138: procedure AdicionarFilho(Arvore : ArvoreBinaria; Direcao : TDirecao; Dado :
     Tipo_do_Dado);
139: {
140: | Objetivos: Cria um filho e o inclui a partir de Arvore na direcao de
     Direcao.
141: }
142: begin
143:    if Vazia(Arvore) or (Direcao in [NoPai,NoRaiz]) or ExisteNo(Arvore,Direcao)
144:    then
145:       begin
146:          writeln('ERRO: Criacao ilegal de um filho');
147:          halt;
148:       end;
149:    CriarNo(Arvore^.Links[Direcao], Dado);
150:    Arvore^.Links[Direcao]^.Links[NoPai] := Arvore;
151: end;
152:
153: (*****************************************************************************)
154:
155: procedure DisposeArvore(Arvore : ArvoreBinaria);
156: {
157: | Objetivos: Desaloca da memoria toda a arvore
158: }
159: begin
160:    if not Vazia(Arvore) then
161:       begin
162:          DisposeArvore(Arvore^.Links[NoEsquerdo]);
163:          DisposeArvore(Arvore^.Links[NoDireito]);
164:          Dispose(Arvore);
165:       end;
166: end;
167:
168: (*****************************************************************************)
169:
170: procedure Deltree(var Arvore : ArvoreBinaria);
171: {
172: | Objetivos: Desaloca da memoria Arvore e seus descendentes, atualizando, se
173: |            necessario, o apontador do pai dessa arvore ou atribuindo o valor
174: |            nil a Arvore, quando for a raiz.
175: }
176: var
177:    PTemp : ArvoreBinaria;
178: begin
179:    PTemp := Arvore;
```

```
180:    { Testa se Arvore tem pai. Caso tenha, Arvore se desloca para ele e PTemp
181:      continua apontando para o inicio da arvore a ser deletada, depois a
182:      arvore e apagada e o apontador do pai e atualizado com nil. Caso Arvore
183:      nao tenha pai, a arvore e eliminada usando PTemp e Arvore recebe nil }
184:    if ExisteNo(Arvore, NoPai) then
185:       begin
186:          Deslocar(Arvore, NoPai);
187:          DisposeArvore(PTemp);
188:          if Arvore^.Links[NoEsquerdo] = PTemp then
189:             Arvore^.Links[NoEsquerdo] := nil
190:          else
191:             Arvore^.Links[NoDireito] := nil;
192:       end
193:    else
194:       begin
195:          DisposeArvore(PTemp);
196:          Arvore := nil;
197:       end;
198: end;
199:
200: (*************************************************************************)
201:
202: procedure PreOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
203: {
204: | Objetivos: Percorre a arvore, visitando primeiro a raiz, depois a subarvore
205: |            esquerda e por ultimo a subarvore direita.
206: }
207: begin
208:    if not Vazia(Arvore) then
209:       begin
210:          Visite(Arvore);
211:          PreOrdem(Arvore^.Links[NoEsquerdo], Visite);
212:          PreOrdem(Arvore^.Links[NoDireito], Visite);
213:       end;
214: end;
215:
216: (*************************************************************************)
217:
218: procedure InOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
219: {
220: | Objetivos: Percorre a arvore, visitando primeiro a subarvore esquerda,
221: |            depois a raiz e por ultimo a subarvore direita.
222: }
223: begin
224:    if not Vazia(Arvore) then
225:      begin
226:         InOrdem(Arvore^.Links[NoEsquerdo], Visite);
227:         Visite(Arvore);
228:         InOrdem(Arvore^.Links[NoDireito], Visite);
229:      end;
230: end;
231:
232: (*************************************************************************)
233:
234: procedure PosOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
235: {
236: | Objetivos: Percorre a arvore, visitando primeiro a subarvore esquerda,
237: |            depois subarvore direita e por ultimo a a raiz.
238: }
239: begin
240:    if not Vazia(Arvore) then
```

```
241:      begin
242:         PosOrdem(Arvore^.Links[NoEsquerdo], Visite);
243:         PosOrdem(Arvore^.Links[NoDireito], Visite);
244:         Visite(Arvore);
245:      end;
246: end;
247:
248: (*************************************************************************)
249:
250: procedure Caracteristicas(Arvore             : ArvoreBinaria;
251:                           var NumNos, Altura : integer;
252:                           var CompMedio      : real);
253: {
254: | Objetivos: Determinar as caracteristicas de uma arvore, tal como, o numero
255: |            de Nos, Altura, e o seu comprimento medio.
256: }
257: var
258:    Soma : integer;
259:
260: procedure  InOrd(Arvore : ArvoreBinaria; Nivel : integer);
261: begin  { InOrd }
262:    if not Vazia(Arvore) then
263:       begin
264:          InOrd(Arvore^.Links[NoEsquerdo], Nivel + 1);
265:          inc(NumNos);
266:          Soma := Soma + Nivel;
267:          if Nivel > Altura then
268:             Altura := Nivel;
269:          InOrd(Arvore^.Links[NoDireito], Nivel + 1);
270:       end;
271: end;  { InOrd }
272:
273: begin { Caracteristicas }
274:    NumNos := 0;
275:    Altura := 0;
276:    Soma   := 0;
277:    InOrd(Arvore, 0);
278:    CompMedio := Soma / NumNos;
279: end;  { Caracteristicas }
280:
281: end.
```