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

```
 57: }
 58: begin
 59:    Vazia := Arvore = nil
 60: end;
 61:
 62: (*************************************************************************)
 63:
 64: procedure CriarNo(var Arvore : ArvoreBinaria; Dado : Tipo_do_Dado);
 65: {
 66: | Objetivo: Cria um No, colocando neste Dado, e fazendo Arvore apontar para
 67: |           ele. Arvore deve ter o valor nil, para indicar que nao aponta
 68: |           para nenhum No.
 69: }
 70: var Dir : TDirecao;
 71: begin
 72:    if not Vazia(Arvore) then
 73:       begin
 74:          writeln('ERRO: O no deve estar vazio para ser criado.');
 75:          halt;
 76:       end;
 77:
 78:    new(Arvore);
 79:    Arvore^.Dado := Dado;
 80:
 81:    for Dir := NoEsquerdo to NoPai do
 82:       Inicializar(Arvore^.Links[Dir]);
 83: end;
 84:
 85: (*************************************************************************)
 86:
 87: function ExisteNo(Arvore : ArvoreBinaria; Direcao : TDirecao) : boolean;
 88: {
 89: | Objetivos: Testa se o No indicado por Direcao a partir de Arvore existe
 90: }
 91: begin
 92:    if Vazia(Arvore) then
 93:       ExisteNo := false
 94:    else
 95:       if Direcao = NoRaiz then
 96:          ExisteNo := true
 97:       else
 98:          ExisteNo := not Vazia(Arvore^.Links[Direcao]);
 99: end;
100:
101: (*************************************************************************)
102:
103: procedure Deslocar(var Arvore : ArvoreBinaria; Direcao : TDirecao);
104: {
105: | Objetivos: Deslocar o apontador Arvore para o No indicado por Direcao
106: }
107: begin
108:    if Direcao = NoRaiz then
109:       while ExisteNo(Arvore, NoPai) do
110:          Arvore := Arvore^.Links[NoPai]
111:    else
112:       Arvore := Arvore^.Links[Direcao];
113: end;
114:
115: (*************************************************************************)
116:
117: procedure ObterDado(var Arvore : ArvoreBinaria; var Dado : Tipo_do_Dado);
```

```
118: {
119: | Objetivos: O parametro Dado recebe o dado contido no No apontado por Arvore
120: }
121: begin
122:    if Vazia(Arvore) then
123:       begin
124:          writeln('ERRO: Dado nao pode ser recuperado de um No vazio');
125:          halt;
126:       end;
127:    Dado := Arvore^.Dado;
128: end;
129:
130: (*****************************************************************************)
131:
132: procedure AlterarDado(Arvore : ArvoreBinaria; Dado : Tipo_do_Dado);
133: {
134: | Objetivos: Coloca no No apontado por Arvore, o dado contido em Dado.
135: }
136: begin
137:    if Vazia(Arvore) then
138:       begin
139:          writeln('ERRO: Nao pode ser alterado o dado de um No vazio.');
140:          halt;
141:       end;
142:    Arvore^.Dado := Dado;
143: end;
144:
145: (*****************************************************************************)
146:
147: procedure AdicionarFilho(Arvore : ArvoreBinaria; Direcao : TDirecao; Dado :
     Tipo_do_Dado);
148: {
149: | Objetivos: Cria um filho e o inclui a partir de Arvore na direcao de
     Direcao.
150: }
151: begin
152:    if Vazia(Arvore) or (Direcao in [NoPai,NoRaiz]) or ExisteNo(Arvore,Direcao)
153:    then
154:       begin
155:          writeln('ERRO: Criacao ilegal de um filho');
156:          halt;
157:       end;
158:    CriarNo(Arvore^.Links[Direcao], Dado);
159:    Arvore^.Links[Direcao]^.Links[NoPai] := Arvore;
160: end;
161:
162: (*****************************************************************************)
163:
164: procedure DisposeArvore(Arvore : ArvoreBinaria);
165: {
166: | Objetivos: Desaloca da memoria toda a arvore
167: }
168: begin
169:    if not Vazia(Arvore) then
170:       begin
171:          DisposeArvore(Arvore^.Links[NoEsquerdo]);
172:          DisposeArvore(Arvore^.Links[NoDireito]);
173:          Dispose(Arvore);
174:       end;
175: end;
176:
```

```
177: (*************************************************************************)
178:
179: procedure Deltree(var Arvore : ArvoreBinaria);
180: {
181: | Objetivos: Desaloca da memoria Arvore e seus descendentes, atualizando, se
182: |            necessario, o apontador do pai dessa arvore ou atribuindo o valor
183: |            nil a Arvore, quando for a raiz.
184: }
185: var
186:    PTemp : ArvoreBinaria;
187: begin
188:    PTemp := Arvore;
189:    { Testa se Arvore tem pai. Caso tenha, Arvore se desloca para ele e PTemp
190:      continua apontando para o inicio da arvore a ser deletada, depois a
191:      arvore e apagada e o apontador do pai e atualizado com nil. Caso Arvore
192:      nao tenha pai, a arvore e eliminada usando PTemp e Arvore recebe nil }
193:    if ExisteNo(Arvore, NoPai) then
194:       begin
195:          Deslocar(Arvore, NoPai);
196:          DisposeArvore(PTemp);
197:          if Arvore^.Links[NoEsquerdo] = PTemp then
198:             Arvore^.Links[NoEsquerdo] := nil
199:          else
200:             Arvore^.Links[NoDireito] := nil;
201:       end
202:    else
203:       begin
204:          DisposeArvore(PTemp);
205:          Arvore := nil;
206:       end;
207: end;
208:
209: (*************************************************************************)
210:
211: procedure PreOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
212: {
213: | Objetivos: Percorre a arvore, visitando primeiro a raiz, depois a subarvore
214: |            esquerda e por ultimo a subarvore direita.
215: }
216: begin
217:    if not Vazia(Arvore) then
218:       begin
219:          Visite(Arvore);
220:          PreOrdem(Arvore^.Links[NoEsquerdo], Visite);
221:          PreOrdem(Arvore^.Links[NoDireito], Visite);
222:       end;
223: end;
224:
225: (*************************************************************************)
226:
227: procedure InOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
228: {
229: | Objetivos: Percorre a arvore, visitando primeiro a subarvore esquerda,
230: |            depois a raiz e por ultimo a subarvore direita.
231: }
232: begin
233:    if not Vazia(Arvore) then
234:      begin
235:         InOrdem(Arvore^.Links[NoEsquerdo], Visite);
236:         Visite(Arvore);
237:         InOrdem(Arvore^.Links[NoDireito], Visite);
```

```
238:      end;
239: end;
240:
241: (*************************************************************************)
242:
243: procedure PosOrdem(Arvore : ArvoreBinaria; Visite : ParamVisite);
244: {
245: | Objetivos: Percorre a arvore, visitando primeiro a subarvore esquerda,
246: |            depois subarvore direita e por ultimo a a raiz.
247: }
248: begin
249:    if not Vazia(Arvore) then
250:      begin
251:         PosOrdem(Arvore^.Links[NoEsquerdo], Visite);
252:         PosOrdem(Arvore^.Links[NoDireito], Visite);
253:         Visite(Arvore);
254:      end;
255: end;
256:
257: (*************************************************************************)
258:
259: procedure Caracteristicas(Arvore             : ArvoreBinaria;
260:                           var NumNos, Altura : integer;
261:                           var CompMedio      : real);
262: {
263: | Objetivos: Determinar as caracteristicas de uma arvore, tal como, o numero
264: |            de Nos, Altura, e o seu comprimento medio.
265: }
266: var
267:    Soma : integer;
268:
269: procedure InOrd(Arvore : ArvoreBinaria; Nivel : integer);
270: begin  { InOrd }
271:    if not Vazia(Arvore) then
272:       begin
273:          InOrd(Arvore^.Links[NoEsquerdo], Nivel + 1);
274:          inc(NumNos);
275:          Soma := Soma + Nivel;
276:          if Nivel > Altura then
277:             Altura := Nivel;
278:          InOrd(Arvore^.Links[NoDireito], Nivel + 1);
279:       end;
280: end;  { InOrd }
281:
282: begin { Caracteristicas }
283:    NumNos := 0;
284:    Altura := 0;
285:    Soma   := 0;
286:    InOrd(Arvore, 0);
287:    CompMedio := Soma / NumNos;
288: end;  { Caracteristicas }
289:
290: (*************************************************************************)
291:
292: function EncontrarChave(Arvore : ArvoreBinaria; Chave : Tipo_da_Chave;
293:                         var P  : ArvoreBinaria) : boolean;
294: {
295: | Objetivos: Retorna true se a chave for encontrada. Neste caso, P
296: |            aponta para o No. Se a chave nao for encontrada, retorna false
297: |            e P aponta para o No que seria o seu pai (caso existisse).
298: }
```

```
299: var
300:    PAnt : ArvoreBinaria;
301:    Achou : boolean;
302: begin  { EncontrarChave }
303:    Achou := false;
304:    Pant := nil;
305:    P := Arvore;
306:
307:    { Laco que fara o deslocamento de P ate que tenha chegado ao local onde
308:      deveria estar o No ou tenha o encontrado }
309:    while not Vazia(P) and not Achou do
310:       begin
311:          PAnt := P;
312:          if Chave = P^.Dado.Chave then
313:             Achou := true
314:          else
315:             if Chave < P^.Dado.Chave then
316:                Deslocar(P, NoEsquerdo)
317:             else
318:                Deslocar(P, NoDireito);
319:       end;
320:
321:    { Testa se nao achou a chave na arvore, pois nesse caso P devera estar
322:      apontando para o No que seria seu pai, ou seja, PAnt }
323:    if not Achou then
324:       P := PAnt;
325:
326:    EncontrarChave := Achou;
327: end;  { EncontrarChave }
328:
329: (*************************************************************************)
330:
331: function Inserir(var Arvore : ArvoreBinaria; Dado : Tipo_do_Dado) : boolean;
332: {
333: | Objetivos: Tenta inserir um No em Arvore (Arvore binaria de busca). Se
334: |            conseguir, retorna true
335: }
336: var
337:    PPai : ArvoreBinaria;
338: begin { Inserir }
339:    Inserir := true;
340:
341:    { Se Arvore estiver vazia entao so e necessario criar o No, mas se nao
342:      estiver, entao sera feita a procura (pela chave) na arvore. Se for achada
343:      alguma ocorrencia da chave na arvore(chave duplicada), entao retornara
344:      false. Caso contrario Dado sera adicionado em uma das subarvores de
    Arvore }
345:    if Vazia(Arvore) then
346:       CriarNo(Arvore, Dado)
347:    else
348:       begin
349:          if EncontrarChave(Arvore, Dado.Chave, PPai) then
350:             Inserir := false
351:          else
352:             if Dado.Chave < PPai^.Dado.Chave then
353:                AdicionarFilho(PPai, NoEsquerdo, Dado)
354:             else
355:                AdicionarFilho(PPai, NoDireito, Dado);
356:       end;
357: end;  { Inserir }
358:
```

```
359: (*************************************************************************)
360:
361: procedure Substituir(Arvore : ArvoreBinaria; var Sucessor : ArvoreBinaria);
362: {
363:   Objetivo: Determina o sucessor imediato do No apontado por Arvore. Quando
364:             encontrar, copia para Arvore os os dados contidos no Sucessor e
365:             muda o apontador Sucessor para o seu No direito. No final,
366:             apaga o No onde se encontrava o Sucessor
367: }
368: var
369:    PApagar : ArvoreBinaria;
370: begin  { Substituir }
371:    { Se nao ha mais Nos a esquerda, entao Sucessor ja eh o sucessor imediado }
372:    if Sucessor^.Links[NoEsquerdo] = nil then
373:       begin
374:          Arvore^.Dado := Sucessor^.Dado;
375:          PApagar := Sucessor;
376:
377:          { Se existir, corrige o apontador (Links[NoPai]) do seu filho direito
378:            do Sucessor, fazendo-o apontar para o pai do sucessor }
379:          if not Vazia(Sucessor^.Links[NoDireito]) then
380:             Sucessor^.Links[NoDireito]^.Links[NoPai] :=
    Sucessor^.Links[NoPai];
381:
382:          Sucessor := Sucessor^.Links[NoDireito];
383:          Dispose(PApagar)
384:       end
385:    else
386:       Substituir(Arvore, Sucessor^.Links[NoEsquerdo])
387: end;  { Substituir }
388:
389: (*************************************************************************)
390:
391: function Remover(var Arvore : ArvoreBinaria; Chave : Tipo_da_Chave) : boolean;
392: {
393: | Objetivos: Retira o No contendo Chave da arvore apontada por Arvore. Se
394: |            esse No for retirado Ok retorna true, caso contrario, retorna
395: |            false
396: }
397: var
398:    PApagar : ArvoreBinaria;
399: begin { Remover }
400:    { Se a arvore estiver vazia, o No nao sera retirado, pois nao existe. Mas
401:      se nao estiver vazia, sera feita a procura atraves de chamadas
402:      recursivas de DeleteNo, pegando a subarvore esquerda quando Chave for
403:      menor que o valor da chave no No ou a subarvore direita quando for maior.
404:      Existem dois pontos de parada, o primeiro quando a subarvore onde esta
405:      sendo feita a busca esta vazia e o segundo quando a Chave for encontrada.
406:      Nesse caso acontecera a delecao }
407:    if Vazia(Arvore) then
408:       Remover := false
409:    else
410:       if Chave < Arvore^.Dado.Chave then
411:          Remover := Remover(Arvore^.Links[NoEsquerdo], Chave)
412:       else
413:          if Chave > Arvore^.Dado.Chave then
414:             Remover := Remover(Arvore^.Links[NoDireito], Chave)
415:          else
416:             { Como nesse ponto Chave = Arvore^.Dado.Chave, devemos verificar
417:               se o No possui duas, uma ou nenhuma subarvore. No primeiro caso
418:               deve ser procurado na subarvore direita o sucessor imediato do
    No
```

```
419:                e coloca-lo no lugar do No removido. Nos outros dois casos, so
420:                e necessario remover o No e ajustar os apontadores }
421:             begin
422:                Remover := true;
423:
424:                if Vazia(Arvore^.Links[NoEsquerdo]) then
425:                   { Arvore nao tem subarvores ou tem somente a direita }
426:                   begin
427:                      PApagar := Arvore;
428:
429:                      { Se existir, corrige o apontador (Links[NoPai]) do filho
430:                        direito do No a ser removido, fazendo-o apontar para o
431:                        pai do No a ser removido }
432:                      if not Vazia(Arvore^.Links[NoDireito]) then
433:                         Arvore^.Links[NoDireito]^
434:                                .Links[NoPai] := Arvore^.Links[NoPai];
435:
436:                      Arvore := Arvore^.Links[NoDireito];
437:                      Dispose(PApagar)
438:                   end
439:                else
440:                   if Vazia(Arvore^.Links[NoDireito]) then
441:                      { Arvote tem somente o filho esquerdo }
442:                      begin
443:                         PApagar := Arvore;
444:
445:                         { Corrige o apontador (Links[NoPai]) do filho
446:                           esquerdo do No a ser removido, fazendo-o
447:                           apontar para o pai do No a ser removido }
448:                         Arvore^.Links[NoEsquerdo]^
449:                                .Links[NoPai] := Arvore^.Links[NoPai];
450:
451:                         Arvore := Arvore^.Links[NoEsquerdo];
452:                         Dispose(PApagar);
453:                      end
454:                   else
455:                      Substituir(Arvore, Arvore^.Links[NoDireito]);
456:             end;
457: end;  { Remover }
458:
459: (*************************************************************************)
460:
461: function Obter(Arvore : ArvoreBinaria; Chave : Tipo_da_Chave;
462:                var Dado : Tipo_do_Dado) : boolean;
463: {
464:   Objetivos: Procura um No que contenha uma chave igual a passada. Caso
465:              encontre, copia o dado do No em Dado e retorna true. Se
466:              nao encontrar, retorna false e Dado nao e modificado
467: }
468: var
469:    P : ArvoreBinaria;
470: begin { Obter }
471:    if EncontrarChave(Arvore, Chave, P) then
472:       begin
473:          Obter := true;
474:          Dado := P^.Dado
475:       end
476:    else
477:       Obter := false;
478: end; { Obter }
479:
```

```
480: (*************************************************************************)
481:
482: function Alterar(Arvore : ArvoreBinaria; Dado : Tipo_do_Dado) : boolean;
483: {
484:   Objetivos: Procura um No que contenha uma chave igual a passada. Caso
485:              encontre, copia Dado sobre o dado do No e retorna true. Se
486:              nao encontrar, retorna false
487: }
488: var
489:    P : ArvoreBinaria;
490: begin { Alterar }
491:    if EncontrarChave(Arvore, Dado.Chave, P) then
492:       begin
493:          Alterar := true;
494:          P^.Dado := Dado;
495:       end
496:    else
497:       Alterar := false;
498: end; { Alterar }
499:
500: end.
```