```
 1: unit ArvAVL;
 2: { Unit que implementa uma arvore AVL }
 3: interface
 4:
 5: type
 6:    Tipo_da_Chave = Integer;
 7:    Tipo_do_Dado = record
 8:                      Chave : Tipo_da_Chave;
 9:                      Nome  : string;
10:                      { Outras informacoes }
11:                   end;
12:    TDirecao = (NoEsquerdo, NoDireito);
13:    PNo = ^No;
14:    No = record
15:            Dado : Tipo_do_Dado;
16:            Links : array[NoEsquerdo..NoDireito] of PNo;
17:            Balanco : -1..1;
18:         end;
19:    ArvoreAVL = PNo;
20:    ParamVisite = procedure(Arvore : ArvoreAVL);
21:
22:    procedure Inicializar(var Arvore : ArvoreAVL);
23:    function  Vazia(var Arvore : ArvoreAVL) : boolean;
24:    function EncontrarChave(Arvore : ArvoreAVL; Chave : Tipo_da_Chave;
25:                            var P  : ArvoreAVL) : boolean;
26:    function Obter(Arvore : ArvoreAVL; Chave : Tipo_da_Chave;
27:                   var Dado : Tipo_do_Dado) : boolean;
28:    function Alterar(Arvore : ArvoreAVL; Dado : Tipo_do_Dado) : boolean;
29:    function Inserir(var Arvore : ArvoreAVL; Dado : Tipo_do_Dado) : boolean;
30:    function Remover(var Arvore : ArvoreAVL; Chave : Tipo_da_Chave) : boolean;
31:    procedure PreOrdem(Arvore : ArvoreAVL; Visite : ParamVisite);
32:    procedure InOrdem(Arvore  : ArvoreAVL; Visite : ParamVisite);
33:    procedure PosOrdem(Arvore : ArvoreAVL; Visite : ParamVisite);
34:    procedure DisposeArvore(Arvore : ArvoreAVL);
35:
36: implementation
37:
38: (*************************************************************************)
39:
40: procedure Inicializar(var Arvore : ArvoreAVL);
41: {
42: | Objetivo: Inicializa a arvore, tornando-a vazia, ou seja, atribuindo o
    valor nil
43: }
44: begin
45:    Arvore := nil
46: end;
47:
48: (*************************************************************************)
49:
50: function Vazia(var Arvore : ArvoreAVL) : boolean;
51: {
52: | Objetivo: Retorna true se o Arvore tem o valor nil
53: }
54: begin
55:    Vazia := Arvore = nil
56: end;
57:
58: (*************************************************************************)
59:
60: function EncontrarNoEPivo(Arvore : ArvoreAVL; Chave : Tipo_da_Chave ;
```

```
61:                            var Pivo, PaiPivo, P : PNo) : boolean;
62: {
63:   Objetivo: Procura uma chave na arvore binaria AVL. Se achar, retorna true
64:             e P, Pivo e PaiPivo retornam o apontador para o No, para o Pivo
65:             e para o pai do Pivo, respectivamente. Se nao achar, retorna
66:             false e P retorna o apontador do No apropriado se a chave
67:             estivesse presente na arvore
68: }
69: var
70:    Q, PPai : PNo;
71:    Ok : boolean;
72: begin
73:    Ok      := false;
74:    PaiPivo := nil;
75:    PPai    := nil;
76:    Pivo    := Arvore;
77:    if Arvore <> nil then
78:    begin
79:       P := Arvore;
80:       repeat
81:          PPai := P;
82:          if Chave = P^.Dado.Chave then
83:             Ok := true
84:          else
85:             begin
86:                if Chave < P^.Dado.Chave then
87:                   Q := P^.Links[NoEsquerdo]
88:                else
89:                   Q := P^.Links[NoDireito];
90:                if Q <> nil then
91:                   if Q^.Balanco <> 0 then
92:                   begin
93:                      PaiPivo := P;
94:                      Pivo    := Q;
95:                   end;
96:                P := Q;
97:             end;
98:       until Ok or (P = nil);
99:       if not Ok then
100:         P := PPai;
101:   end;
102:   EncontrarNoEPivo := Ok;
103: end;
104:
105: (*************************************************************************
     )
106:
107: function EncontrarNo(Arvore : ArvoreAVL; Chave : Tipo_da_Chave;
108:                       var P  : ArvoreAVL) : boolean;
109: {
110: | Objetivos: Retorna true se a chave for encontrada. Neste caso, P
111: |            aponta para o No. Se a chave nao for encontrada, retorna false
112: |            e P aponta para nil.
113: }
114: begin  { EncontrarNo }
115:    P := Arvore;
116:
117:    { Laco que fara o deslocamento de P ate que tenha chegado ao local onde
118:      deveria estar o No ou tenha o encontrado }
119:    while P <> nil do
120:       if Chave = P^.Dado.Chave then
```

```
121:          break
122:       else
123:          if Chave < P^.Dado.Chave then
124:             P := P^.Links[NoEsquerdo]
125:          else
126:             P := P^.Links[NoDireito];
127:
128:    { Se P = nil então o laço encerrou porque a chave não foi encontrada }
129:    EncontrarNo := P <> nil;
130: end;  { EncontrarNo }
131:
132: (*****************************************************************************
     )
133:
134: procedure CriarNo(var P : PNo; Dado : Tipo_do_Dado);
135: {
136:   Objetivo: Criar um No e inicializar guardar o Dado passado nele.
137:             O balanco do No sera inicializado com 0 e os apontadores
138:             para as subarvores com nil
139: }
140: var Dir : TDirecao;
141: begin
142:    New(P);
143:    P^.Dado := Dado;
144:    P^.Balanco := 0;
145:    for Dir := NoEsquerdo to NoDireito do
146:       P^.Links[Dir] := nil;
147: end;
148:
149: (*****************************************************************************
     )
150:
151: function EncontrarChave(Arvore : ArvoreAVL; Chave : Tipo_da_Chave;
152:                         var P  : ArvoreAVL) : boolean;
153: begin
154:    EncontrarChave := EncontrarNo(Arvore, Chave, P);
155: end;
156:
157: (*****************************************************************************
     )
158:
159: function Obter(Arvore : ArvoreAVL; Chave : Tipo_da_Chave; var Dado :
     Tipo_do_Dado) : boolean;
160: var P : PNo;
161: begin
162:    if EncontrarNo(Arvore, Chave, P) then
163:       begin
164:          Obter := true;
165:          Dado := P^.Dado;
166:       end
167:    else
168:       Obter := false;
169: end;
170:
171: (*****************************************************************************
     )
172:
173: function Alterar(Arvore : ArvoreAVL; Dado : Tipo_do_Dado) : boolean;
174: var P : PNo;
175: begin
176:    if EncontrarNo(Arvore, Dado.Chave, P) then
```

```
177:       begin
178:          Alterar := true;
179:          P^.Dado := Dado;
180:       end
181:    else
182:       Alterar := false;
183: end;
184:
185: (*****************************************************************************
     )
186:
187: function Inserir(var Arvore : ArvoreAVL; Dado : Tipo_do_Dado) : boolean;
188: {
189:   Objetivo: Insere um No na arvore AVL
190: }
191:
192:    procedure RotacaoParaDireita(var P : PNo);
193:    {
194:      Fazendo a rotacao em torno de A teriamos a situacao ao lado
195:
196:          P -->  A                              B
197:                /   \                         /   \
198:     Q -->  B       C       =>               D     A
199:           / \     / \                           / \
200:          D  E    F  G                         E   C
201:                                                  / \
202:                                                 F   G
203:    }
204:    var Temp, Q : PNo;
205:    begin
206:       Q                    := P^.Links[NoEsquerdo];
207:       Temp                 := Q^.Links[NoDireito];
208:       Q^.Links[NoDireito]  := P;
209:       P^.Links[NoEsquerdo] := Temp;
210:    end;
211:
212:    procedure RotacaoParaEsquerda(var P : PNo);
213:    {
214:      Fazendo a rotacao em torno de A teriamos a situacao ao lado
215:
216:                A  <-- P                          C
217:               /   \                             /   \
218:           B       C  <-- Q      =>            A     G
219:          / \     / \                          / \
220:         D  E    F  G                        B   F
221:                                            / \
222:                                           D   E
223:    }
224:    var Temp, Q : PNo;
225:    begin
226:       Q                    := P^.Links[NoDireito];
227:       Temp                 := Q^.Links[NoEsquerdo];
228:       Q^.Links[NoEsquerdo] := P;
229:       P^.Links[NoDireito]  := Temp;
230:    end;
231:
232: var
233:    Pivo, PaiPivo, FilhoPivo, P, PPai, Q : PNo;
234:    Desequilibrio : -1..1;
235:
236: begin { Inserir }
```

```
237:    if EncontrarNoEPivo(Arvore, Dado.Chave, Pivo, PaiPivo, PPai) then
238:    begin
239:       Inserir := false;
240:       exit;
241:    end;
242:
243:    { Como a chave nao existe, pode ser incluida }
244:    Inserir := true;
245:
246:    { Cria o No com o Dado e faz com P aponte para ele }
247:    CriarNo(P, Dado);
248:
249:    { Se a arvore for vazia, define a raiz da arvore e retorna }
250:    if Arvore = nil then
251:    begin
252:       Arvore := P;
253:       exit;
254:    end;
255:
256:    { Ajusta o ponteiro do pai para apontar para o novo No.
257:      Se a chave do novo No for menor sera inserido na
258:      subarvore esquerda. Caso contrario sera inserido na
259:      subarvore direita }
260:    if Dado.Chave < PPai^.Dado.Chave then
261:       PPai^.Links[NoEsquerdo] := P
262:    else
263:       PPai^.Links[NoDireito] := P;
264:
265:    { Ajusta os balancos dos Nos do caminho abaixo do pivo }
266:    if Dado.Chave < Pivo^.Dado.Chave then
267:       Q := Pivo^.Links[NoEsquerdo]
268:    else
269:       Q := Pivo^.Links[NoDireito];
270:    FilhoPivo := Q;  { Filho na direcao da insercao }
271:    while Q <> P do
272:       if Dado.Chave < Q^.Dado.Chave then
273:          begin
274:             Q^.Balanco := 1;
275:             Q := Q^.Links[NoEsquerdo];
276:          end
277:       else
278:          begin
279:             Q^.Balanco := -1;
280:             Q := Q^.Links[NoDireito];
281:          end;
282:
283:    { Verifica em que lado sera inserido }
284:    if Dado.Chave < Pivo^.Dado.Chave then
285:       Desequilibrio := 1
286:    else
287:       Desequilibrio := -1;
288:
289:    if Pivo^.Balanco = 0 then
290:       { O balanco do pivo e' zero, portanto a insercao em qualquer
291:         subarvore mantem a arvore balanceada }
292:       Pivo^.Balanco := Desequilibrio
293:    else
294:       if Pivo^.Balanco <> Desequilibrio then
295:          { A insercao sera' feita na subarvore menor, mantendo, portanto
296:            a arvore equilibrada }
297:          Pivo^.Balanco := 0
```

```
298:       else
299:          begin
300:             { A insercao sera' feita na subarvore maior. O Rebalanceamento
301:               sera' feito atraves de rotacoes }
302:             if FilhoPivo^.Balanco = Desequilibrio then
303:                { O desequilibrio foi feito na mesma direcao do Pivo e do
304:                  seu filho }
305:                begin
306:                   Q := FilhoPivo;
307:                   { Efetua uma rotacao a depender do lado do desequilibrio }
308:                   if Desequilibrio = 1 then
309:                      RotacaoParaDireita(Pivo)
310:                   else
311:                      RotacaoParaEsquerda(Pivo);
312:                   Pivo^.Balanco := 0;
313:                   FilhoPivo^.Balanco := 0;
314:                end
315:             else
316:                { O desequilibrio foi feito na direcao contraria do Pivo
317:                  e seu filho }
318:                begin
319:                   { Efetua duas rotacoes }
320:                   if Desequilibrio = 1 then
321:                      begin
322:                         Q := FilhoPivo^.Links[NoDireito];
323:                         RotacaoParaEsquerda(FilhoPivo);
324:                         Pivo^.Links[NoEsquerdo] := Q;
325:                         RotacaoParaDireita(Pivo);
326:                      end
327:                   else
328:                      begin
329:                         Q := FilhoPivo^.Links[NoEsquerdo];
330:                         Pivo^.Links[NoDireito] := Q;
331:                         RotacaoParaDireita(FilhoPivo);
332:                         RotacaoParaEsquerda(Pivo);
333:                      end;
334:                   { Ajuste dos balancos dos nos envolvidos }
335:                   if Q^.Balanco = 0 then
336:                      begin
337:                         Pivo^.Balanco := 0;
338:                         FilhoPivo^.Balanco := 0;
339:                      end
340:                   else
341:                      if Q^.Balanco = Desequilibrio then
342:                         begin
343:                            Pivo^.Balanco := -Desequilibrio;
344:                            FilhoPivo^.Balanco := 0;
345:                         end
346:                      else
347:                         begin
348:                            Pivo^.Balanco := 0;
349:                            FilhoPivo^.Balanco := Desequilibrio;
350:                         end;
351:                   Q^.Balanco := 0;
352:                end;
353:             { Ajusta os ponteiros da arvore apos a rotacao }
354:             if PaiPivo = nil then
355:                Arvore := Q
356:             else
357:                if Pivo = PaiPivo^.Links[NoDireito] then
358:                   PaiPivo^.Links[NoDireito] := Q
```

```
359:                 else
360:                    PaiPivo^.Links[NoEsquerdo] := Q;
361:          end;
362: end; { Inserir }
363:
364: (*************************************************************************
     )
365:
366: function Remover(var Arvore : ArvoreAVL; Chave : Tipo_da_Chave) : boolean;
367:
368:    procedure BalanceL(var P : PNo; var H : boolean);
369:    var P1, P2 : PNo;
370:        B1, B2 : -1..1;
371:    begin
372:       case P^.Balanco of
373:          1 : P^.Balanco := 0;
374:          0 : begin
375:                 P^.Balanco := -1;
376:                 H := false;
377:              end;
378:         -1 : begin   { Rebalanco }
379:                 P1 := P^.Links[NoDireito];
380:                 B1 := P1^.Balanco;
381:                 if B1 <= 0 then  { Rotacao simples }
382:                    begin
383:                       P^.Links[NoDireito] := P1^.Links[NoEsquerdo];
384:                       P1^.Links[NoEsquerdo] := P;
385:                       if B1 = 0 then
386:                          begin
387:                             P^.Balanco := -1;
388:                             P1^.Balanco := 1;
389:                             H := false;
390:                          end
391:                       else
392:                          begin
393:                             P^.Balanco := 0;
394:                             P1^.Balanco := 0;
395:                          end;
396:                       P := P1;
397:                    end
398:                 else  { Rotacao dupla RL }
399:                    begin
400:                       P2 := P1^.Links[NoEsquerdo];
401:                       B2 := P2^.Balanco;
402:                       P1^.Links[NoEsquerdo] := P2^.Links[NoDireito];
403:                       P2^.Links[NoDireito]  := P1;
404:                       P^.Links[NoDireito] := P2^.Links[NoEsquerdo];
405:                       P2^.Links[NoEsquerdo] := P;
406:                       if B2 = -1 then
407:                          P^.Balanco := 1
408:                       else
409:                          P^.Balanco := 0;
410:                       if B2 = 1 then
411:                          P1^.Balanco := -1
412:                       else
413:                          P1^.Balanco := 0;
414:                       P := P2;
415:                       P2^.Balanco := 0;
416:                    end;
417:              end;
418:       end;
```

```
419:    end;   { BalancoL }
420:
421:    procedure BalanceR(var P : PNo; var H : boolean);
422:    var P1, P2 : PNo;
423:        B1, B2 : -1..1;
424:    begin
425:       case P^.Balanco of
426:         -1 : P^.Balanco := 0;
427:          0 : begin
428:                 P^.Balanco := 1;
429:                 H := false;
430:              end;
431:          1 : begin   { Rebalanco }
432:                 P1 := P^.Links[NoEsquerdo];
433:                 B1 := P1^.Balanco;
434:                 if B1 >= 0 then  { Rotacao simples }
435:                    begin
436:                       P^.Links[NoEsquerdo] := P1^.Links[NoDireito];
437:                       P1^.Links[NoDireito] := P;
438:                       if B1 = 0 then
439:                          begin
440:                             P^.Balanco := 1;
441:                             P1^.Balanco :=- 1;
442:                             H := false;
443:                          end
444:                       else
445:                          begin
446:                             P^.Balanco := 0;
447:                             P1^.Balanco := 0;
448:                          end;
449:                       P := P1;
450:                    end
451:                 else  { Rotacao dupla LR }
452:                    begin
453:                       P2 := P1^.Links[NoDireito];
454:                       B2 := P2^.Balanco;
455:                       P1^.Links[NoDireito] := P2^.Links[NoEsquerdo];
456:                       P2^.Links[NoEsquerdo] := P1;
457:                       P^.Links[NoEsquerdo] := P2^.Links[NoDireito];
458:                       P2^.Links[NoDireito] := P;
459:                       if B2 = 1 then
460:                          P^.Balanco := -1
461:                       else
462:                          P^.Balanco := 0;
463:                       if B2 = -1 then
464:                          P1^.Balanco := 1
465:                       else
466:                          P1^.Balanco := 0;
467:                       P := P2;
468:                       P2^.Balanco := 0;
469:                    end;
470:              end;
471:       end;
472:    end;   { BalancoR }
473:
474:    procedure Delete(var P : PNo; var H : boolean);
475:    var Q : PNo;
476:       procedure Del(var R : PNo; var H : boolean);
477:       begin
478:          if R^.Links[NoDireito] <> nil then
479:             begin
```

```
480:                Del(R^.Links[NoDireito], H);
481:                if H then
482:                   BalanceR(R, H)
483:             end
484:          else
485:             begin
486:                Q^.Dado := R^.Dado;
487:                Q := R;
488:                R := R^.Links[NoEsquerdo];
489:                H := true;
490:             end;
491:       end;  { Del }
492:
493:    begin { Delete }
494:       if P = nil then
495:          Remover := false
496:       else
497:          if P^.Dado.Chave > Chave then
498:             begin
499:                Delete(P^.Links[NoEsquerdo], H);
500:                if H then
501:                   BalanceL(P, H);
502:             end
503:          else
504:             if P^.Dado.Chave < Chave then
505:                begin
506:                   Delete(P^.Links[NoDireito], H);
507:                   if H then
508:                      BalanceR(P, H);
509:                end
510:             else
511:                begin
512:                   Q := P;
513:                   if Q^.Links[NoDireito] = nil then
514:                      begin
515:                         P := Q^.Links[NoEsquerdo];
516:                         H := true;
517:                      end
518:                   else
519:                      if Q^.Links[NoEsquerdo] = nil then
520:                         begin
521:                            P := Q^.Links[NoDireito];
522:                            H := true;
523:                         end
524:                      else
525:                         begin
526:                            Del(Q^.Links[NoEsquerdo], H);
527:                            if H then
528:                               BalanceL(P, H);
529:                         end;
530:                   dispose(Q);
531:                end;
532:    end;  { Delete }
533:
534: var H : Boolean;
535:
536: begin { Remover }
537:    Remover := True;
538:    Delete(Arvore, H);
539: end;
540:
```

```
541: (*****************************************************************************)
542:
543: procedure PreOrdem(Arvore : ArvoreAVL; Visite : ParamVisite);
544: {
545: | Objetivos: Percorre a arvore, visitando primeiro a raiz, depois a subarvore
546: |            esquerda e por ultimo a subarvore direita.
547: }
548: begin
549:    if not Vazia(Arvore) then
550:       begin
551:          Visite(Arvore);
552:          PreOrdem(Arvore^.Links[NoEsquerdo], Visite);
553:          PreOrdem(Arvore^.Links[NoDireito], Visite);
554:       end;
555: end;
556:
557: (*****************************************************************************)
558:
559: procedure InOrdem(Arvore : ArvoreAVL; Visite : ParamVisite);
560: {
561: | Objetivos: Percorre a arvore, visitando primeiro a subarvore esquerda,
562: |            depois a raiz e por ultimo a subarvore direita.
563: }
564: begin
565:    if not Vazia(Arvore) then
566:      begin
567:         InOrdem(Arvore^.Links[NoEsquerdo], Visite);
568:         Visite(Arvore);
569:         InOrdem(Arvore^.Links[NoDireito], Visite);
570:      end;
571: end;
572:
573: (*****************************************************************************)
574:
575: procedure PosOrdem(Arvore : ArvoreAVL; Visite : ParamVisite);
576: {
577: | Objetivos: Percorre a arvore, visitando primeiro a subarvore esquerda,
578: |            depois subarvore direita e por ultimo a a raiz.
579: }
580: begin
581:    if not Vazia(Arvore) then
582:      begin
583:         PosOrdem(Arvore^.Links[NoEsquerdo], Visite);
584:         PosOrdem(Arvore^.Links[NoDireito], Visite);
585:         Visite(Arvore);
586:      end;
587: end;
588:
589: (*****************************************************************************)
590:
591: procedure DisposeArvore(Arvore : ArvoreAVL);
592: {
593: | Objetivos: Desaloca da memoria toda a arvore
594: }
595: begin
596:    if not Vazia(Arvore) then
597:       begin
598:          DisposeArvore(Arvore^.Links[NoEsquerdo]);
599:          DisposeArvore(Arvore^.Links[NoDireito]);
600:          Dispose(Arvore);
601:       end;
```