```
 1: unit LstOrd;
 2:
 3: interface
 4:
 5: type
 6:    { Tipo de chave do item da lista }
 7:    Tipo_Chave = longint;
 8:
 9:    { Tipo do item }
10:    Tipo_Item = record
11:                   Chave : Tipo_Chave;
12:                   Dado  : String[30];
13:                end;
14:
15:    { Tipo do apontador para um item da lista }
16:    Pont_Item_Lista = ^Item_Lista;
17:
18:    { Tipo do item da lista }
19:    Item_Lista = record
20:                   Item : Tipo_Item;
21:                   Proximo : Pont_Item_Lista
22:                 end;
23:
24:    { Tipo da lista ordenada }
25:    Lista_Ordenada = record
26:                        Cabeca : Pont_Item_Lista;
27:                      end;
28:
29: procedure Inicializar (var Lista : Lista_Ordenada);
30: function Inserir (Item : Tipo_Item; var Lista : Lista_Ordenada) : boolean;
31: function Remover (Chave : Tipo_Chave; var Lista : Lista_Ordenada) : boolean;
32: function Alterar (Item : Tipo_Item; var Lista : Lista_Ordenada) : boolean;
33: procedure Obter (Chave : Tipo_Chave; var Lista : Lista_Ordenada;
34:                  var Item : Tipo_Item; var Sucesso : boolean);
35: procedure Apagar (var Lista : Lista_Ordenada);
36: function Tamanho (var Lista : Lista_Ordenada) : longint;
37: function Vazia (var Lista : Lista_Ordenada) : boolean;
38:
39: implementation
40:
41: procedure Inicializar (var Lista : Lista_Ordenada);
42: {
43:   Objetivo: Inicializa a lista passada, fazendo com que a cabeca da lista
44:             aponte para nil.
45: }
46: begin
47:    ReturnNilIfGrowHeapFails := true;
48:    Lista.Cabeca := nil
49: end;
50:
51:
52: function AlocarItem (var PItem : Pont_Item_Lista) : boolean;
53: {
54:   Objetivo: Tentar alocar um Item usando o apontador passado,
55:             retornando true se conseguir e false caso contrario.
56: }
57: begin
58:    New(PItem);
59:    AlocarItem := PItem <> nil;
60: end;
61:
```

```
62:
63: function Inserir (Item : Tipo_Item; var Lista : Lista_Ordenada) : boolean;
64: {
65:   Objetivo: Insere o item passado como parametro na lista passada.
66:             Se a lista ja estiver cheia ou existir algum item com a mesma
67:             chave, a funcao Inserir retorna false.
68: }
69: var
70:    PAnterior, PAtual, PNovo : Pont_Item_Lista;
71: begin
72:    Inserir := false;
73:    PAnterior := nil;
74:    PAtual := Lista.Cabeca;
75:
76:    { Percorre a lista ate PAtual apontar para o Item que deve vir apos
77:      ele e PAnterior para o item que deve ficar antes }
78:    while PAtual <> nil do
79:    begin
80:       { Verifica se ja passou da posicao possivel }
81:       if PAtual^.Item.Chave > Item.Chave then
82:          break
83:       else
84:          { Chave igual nao e permitida }
85:          if PAtual^.Item.Chave = Item.Chave then
86:             exit;
87:
88:       { Chave ainda menor do que a atual }
89:       PAnterior := PAtual;
90:       PAtual := PAtual^.Proximo
91:    end;
92:
93:    if AlocarItem(PNovo) then
94:       begin
95:          Inserir := true;
96:          PNovo^.Item := Item;
97:
98:          if PAnterior = nil then
99:             { Inserir na cabeca da lista }
100:            begin
101:               PNovo^.Proximo := Lista.Cabeca;
102:               Lista.Cabeca := PNovo
103:            end
104:         else
105:            { Inserindo no meio ou no final }
106:            begin
107:               PNovo^.Proximo := PAtual;
108:               PAnterior^.Proximo := PNovo
109:            end
110:      end
111: end;
112:
113:
114: function Remover (Chave : Tipo_Chave; var Lista : Lista_Ordenada) : boolean;
115: {
116:   Objetivo: Remove o item cuja chave coincide com o parametro Chave
117:             passado. Caso nao haja um item com essa chave, retorna
118:             false. Se o item foi removido, retorna true.
119: }
120: var
121:    PAtual, PAnterior : Pont_Item_Lista;
122: begin
```

```
123:    Remover := false;
124:
125:    if Vazia(Lista) then
126:       exit;
127:
128:    PAnterior := Lista.Cabeca;
129:
130:    if Lista.Cabeca^.Item.Chave = Chave then
131:       begin
132:          { Remove o item da cabeca da lista e a cabeca passa a ser
133:            o item seguinte ao removido}
134:          Lista.Cabeca := Lista.Cabeca^.Proximo;
135:          Dispose(PAnterior);
136:          Remover := true
137:       end
138:    else
139:       { Percorre a lista ate encontrar um item com a chave procurada.
140:         Remove o item e corrige o apontador do item anterior para
141:         apontar para o proximo item }
142:       while PAnterior^.Proximo <> nil do
143:       begin
144:          PAtual := PAnterior^.Proximo;
145:
146:          if PAtual^.Item.Chave = Chave then
147:             begin
148:                PAnterior^.Proximo := PAtual^.Proximo;
149:                Dispose(PAtual);
150:                Remover := true;
151:                break
152:             end
153:          else
154:             if PAtual^.Item.Chave > Chave then
155:                break;
156:
157:          PAnterior := PAtual
158:       end
159: end;
160:
161:
162: function Alterar (Item : Tipo_Item; var Lista : Lista_Ordenada) : boolean;
163: {
164:   Objetivo: Altera os dados de um item existente na lista passada
165:             de forma que fique igual ao do item passado como parametro.
166:             Se o item for encontrado e alterado, retorna true. Caso
167:             contrario, retorna false.
168: }
169: var
170:    PAtual : Pont_Item_Lista;
171: begin
172:    Alterar := false;
173:    PAtual := Lista.Cabeca;
174:
175:    while PAtual <> nil do
176:    begin
177:       { Verifica se ja passou do ponto aonde o item poderia se encontrar }
178:       if PAtual^.Item.Chave > Item.Chave then
179:          exit;
180:
181:       if PAtual^.Item.Chave = Item.Chave then
182:          { Quando e igual, encontrou o item }
183:          begin
```

```
184:             PAtual^.Item := Item;
185:             Alterar := true;
186:             break
187:          end
188:       else
189:          PAtual := PAtual^.Proximo
190:    end
191: end;
192:
193:
194: procedure Obter (Chave : Tipo Chave; var Lista : Lista Ordenada;
195:                  var Item : Tipo_Item; var Sucesso : boolean);
196: {
197:   Objetivo: Procura na lista usando a chave passada. Caso encontre
198:             Sucesso contem o valor true e Item contem o Item obtido.
199:             Caso contrario, Sucesso retorna true e Item nao e alterado
200: }
201: var
202:    PAtual : Pont_Item_Lista;
203: begin
204:    Sucesso := false;
205:    PAtual := Lista.Cabeca;
206:
207:    while PAtual <> nil do
208:    begin
209:       { Verifica se ja passou do ponto aonde o item poderia se encontrar }
210:       if PAtual^.Item.Chave > Chave then
211:          exit;
212:
213:       if PAtual^.Item.Chave = Chave then
214:          { Quando e igual, encontrou o item }
215:          begin
216:             Item := PAtual^.Item;
217:             Sucesso := true;
218:             break
219:          end
220:       else
221:          { Move o Apontador para o proximo item }
222:          PAtual := PAtual^.Proximo
223:    end
224: end;
225:
226:
227: procedure Apagar (var Lista : Lista_Ordenada);
228: {
229:   Objetivo: Retorna o tamanho da lista passada
230: }
231: var
232:    PAtual, PApagar : Pont_Item_Lista;
233: begin
234:    PAtual := Lista.Cabeca;
235:
236:    while PAtual <> nil do
237:    begin
238:       PApagar := PAtual;
239:       PAtual := PAtual^.Proximo;
240:       Dispose(PApagar)
241:    end;
242:
243:    Lista.Cabeca := nil
244: end;
```

```
245:
246:
247: function Tamanho (var Lista : Lista_Ordenada) : longint;
248: {
249:   Objetivo: Retorna o tamanho da lista passada
250: }
251: var
252:    P : Pont_Item_Lista;
253:    Cont : longint;
254: begin
255:    P := Lista.Cabeca;
256:    Cont := 0;
257:
258:    while P <> nil do
259:    begin
260:       inc(Cont);
261:       P := P^.Proximo
262:    end;
263:
264:    Tamanho := Cont;
265: end;
266:
267:
268: function Vazia (var Lista : Lista_Ordenada) : boolean;
269: {
270:   Objetivo: Retorna true se a lista esta vazia (Cabeca = nil)
271: }
272: begin
273:    Vazia := Lista.Cabeca = nil
274: end;
275:
276: end.
```