```
 1: unit LstOrd;
 2:
 3: interface
 4:
 5: type
 6:    { Tipo de chave do item da lista }
 7:    Tipo_Chave = integer;
 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) : integer;
37: function Cheia (var Lista : Lista_Ordenada) : boolean;
38: function Vazia (var Lista : Lista_Ordenada) : boolean;
39:
40: implementation
41:
42: procedure Inicializar (var Lista : Lista_Ordenada);
43: {
44:   Objetivo: Inicializa a lista passada, fazendo com que a cabeca da lista
45:             aponte para nil.
46: }
47: begin
48:    Lista.Cabeca := nil
49: end;
50:
51:
52: function Inserir (Item : Tipo_Item; var Lista : Lista_Ordenada) : boolean;
53: {
54:   Objetivo: Insere o item passado como parametro na lista passada.
55:             Se a lista ja estiver cheia ou existir algum item com a mesma
56:             chave, a funcao Inserir retorna false.
57: }
58: var
59:    PAnterior, PAtual, PNovo : Pont_Item_Lista;
60: begin
61:    Inserir := false;
```

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

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

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

```
245: begin
246:    P := Lista.Cabeca;
247:    Cont := 0;
248:
249:    while P <> nil do
250:    begin
251:       inc(Cont);
252:       P := P^.Proximo
253:    end;
254:
255:    Tamanho := Cont;
256: end;
257:
258:
259: function Cheia (var Lista : Lista_Ordenada) : boolean;
260: {
261:   Objetivo: Retorna true se nao ha mais memoria disponivel
262:             para inserir um item na lista
263: }
264: begin
265:    Cheia := MaxAvail < SizeOf(Item_Lista)
266: end;
267:
268:
269: function Vazia (var Lista : Lista_Ordenada) : boolean;
270: {
271:   Objetivo: Retorna true se a lista esta vazia (Cabeca = nil)
272: }
273: begin
274:    Vazia := Lista.Cabeca = nil
275: end;
276:
277: end.
```