```
 1: unit LstDEnc;
 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,
22:                   Anterior : Pont_Item_Lista
23:                 end;
24:
25:    { Tipo da lista Duplamente Encadeada }
26:    Lista_Duplamente_Enc = record
27:                              Cabeca : Pont_Item_Lista;
28:                              Cauda  : Pont_Item_Lista;
29:                              Tamanho : longint
30:                            end;
31:
32: procedure Inicializar (var Lista : Lista_Duplamente_Enc);
33: function Inserir (Item : Tipo_Item; var Lista : Lista_Duplamente_Enc) :
    boolean;
34: function Remover (Chave : Tipo_Chave; var Lista : Lista_Duplamente_Enc) :
    boolean;
35: function Alterar (Item : Tipo_Item; var Lista : Lista_Duplamente_Enc) :
    boolean;
36: procedure Obter (Chave : Tipo_Chave; var Lista : Lista_Duplamente_Enc;
37:                  var Item : Tipo_Item; var Sucesso : boolean);
38: procedure Apagar (var Lista : Lista_Duplamente_Enc);
39: function Tamanho (var Lista : Lista_Duplamente_Enc) : longint;
40: function Vazia (var Lista : Lista_Duplamente_Enc) : boolean;
41:
42: implementation
43:
44: procedure Inicializar (var Lista : Lista_Duplamente_Enc);
45: {
46:   Objetivo: Inicializa a lista passada, fazendo com que a cabeca e a cauda
47:             da lista apontem para nil.
48: }
49: begin
50:    Lista.Cabeca := nil;
51:    Lista.Cauda := nil;
52:    Lista.Tamanho := 0;
53:    ReturnNilIfGrowHeapFails := true
54: end;
55:
56:
57: function AlocarItem (var PItem : Pont_Item_Lista) : boolean;
58: {
```

```
59:   Objetivo: Tentar alocar um Item usando o apontador passado,
60:             retornando true se conseguir e false caso contrario.
61: }
62: begin
63:    New(PItem);
64:    AlocarItem := PItem <> nil;
65: end;
66:
67:
68: function Inserir (Item : Tipo_Item; var Lista : Lista_Duplamente_Enc) :
    boolean;
69: {
70:   Objetivo: Insere o item passado como parametro na lista passada.
71:             Se a lista ja estiver cheia, a funcao Inserir retorna false.
72: }
73: var
74:    PNovo : Pont_Item_Lista;
75: begin
76:    Inserir := false;
77:
78:    if AlocarItem(PNovo) then
79:       begin
80:          PNovo^.Item := Item;
81:          PNovo^.Proximo := Lista.Cabeca;
82:          PNovo^.Anterior := nil;
83:
84:          if Lista.Cabeca <> nil then
85:             Lista.Cabeca^.Anterior := PNovo;
86:
87:          Lista.Cabeca := PNovo;
88:
89:          if Lista.Cauda = nil then
90:             Lista.Cauda := PNovo;
91:
92:          inc(Lista.Tamanho);
93:          Inserir := true
94:       end
95: end;
96:
97:
98: function Remover (Chave : Tipo_Chave; var Lista : Lista_Duplamente_Enc) :
    boolean;
99: {
100:   Objetivo: Remove o item cuja chave coincide com o parametro Chave
101:             passado. Caso nao haja um item com essa chave, retorna
102:             false. Se o item foi removido, retorna true.
103: }
104: var
105:    PAtual : Pont_Item_Lista;
106: begin
107:    Remover := false;
108:
109:    PAtual := Lista.Cabeca;
110:
111:    while PAtual <> nil do
112:    begin
113:       if PAtual^.Item.Chave = Chave then
114:          begin
115:             if PAtual = Lista.Cabeca then
116:                Lista.Cabeca := PAtual^.Proximo;
117:
```

```
118:             if PAtual = Lista.Cauda then
119:                Lista.Cauda := PAtual^.Anterior;
120:
121:             if PAtual^.Proximo <> nil then
122:                PAtual^.Proximo^.Anterior := PAtual^.Anterior;
123:
124:             if PAtual^.Anterior <> nil then
125:                PAtual^.Anterior^.Proximo := PAtual^.Proximo;
126:
127:             Dispose(PAtual);
128:             dec(Lista.Tamanho);
129:             Remover := true;
130:             break
131:          end
132:       else
133:          PAtual := PAtual^.Proximo
134:    end;
135: end;
136:
137:
138: function ObterNo (Chave : Tipo_Chave; var Lista : Lista_Duplamente_Enc) :
     Pont_Item_Lista;
139: {
140:   Objetivo: Retorna um apontador para o No que contem o Item com a chave
141:             igual a passada. Caso nao seja encontrado, a funcao retorna nil
142: }
143: var
144:    PAtual : Pont_Item_Lista;
145: begin
146:    ObterNo := nil;
147:    PAtual := Lista.Cabeca;
148:
149:    while PAtual <> nil do
150:       if PAtual^.Item.Chave = Chave then
151:          begin
152:             ObterNo := PAtual;
153:             break
154:          end
155:       else
156:          PAtual := PAtual^.Proximo
157: end;
158:
159:
160: procedure Obter (Chave : Tipo_Chave; var Lista : Lista_Duplamente_Enc;
161:                  var Item : Tipo_Item; var Sucesso : boolean);
162: {
163:   Objetivo: Procura na lista usando a chave passada. Caso encontre
164:             Sucesso contem o valor true e Item contem o Item obtido.
165:             Caso contrario, Sucesso retorna true e Item nao e alterado
166: }
167: var
168:    PAtual : Pont_Item_Lista;
169: begin
170:    Sucesso := false;
171:    PAtual := ObterNo(Chave, Lista);
172:
173:    if PAtual <> nil then
174:       begin
175:          Item := PAtual^.Item;
176:          Sucesso := true
177:       end
```

```
178: end;
179:
180:
181: function Alterar (Item : Tipo_Item; var Lista : Lista_Duplamente_Enc) :
     boolean;
182: {
183:   Objetivo: Altera os dados de um item existente na lista passada
184:             de forma que fique igual ao do item passado como parametro.
185:             Se o item for encontrado e alterado, retorna true. Caso
186:             contrario, retorna false.
187: }
188: var
189:    PAtual : Pont_Item_Lista;
190: begin
191:    Alterar := false;
192:    PAtual := ObterNo(Item.Chave, Lista);
193:
194:    if PAtual <> nil then
195:       begin
196:          PAtual^.Item := Item;
197:          Alterar := true
198:       end
199: end;
200:
201:
202: procedure Apagar (var Lista : Lista_Duplamente_Enc);
203: {
204:   Objetivo: Apaga a lista passada
205: }
206: var
207:    PAtual : Pont_Item_Lista;
208: begin
209:    if Vazia(Lista) then
210:       exit;
211:
212:    PAtual := Lista.Cabeca;
213:
214:    while PAtual <> Lista.Cauda do
215:    begin
216:       PAtual := PAtual^.Proximo;
217:       Dispose(PAtual^.Anterior)
218:    end;
219:
220:    Dispose(Lista.Cauda);
221:
222:    Lista.Cabeca := nil;
223:    Lista.Cauda := nil;
224:    Lista.Tamanho := 0
225: end;
226:
227:
228: function Tamanho (var Lista : Lista_Duplamente_Enc) : longint;
229: {
230:   Objetivo: Retorna o tamanho da lista passada
231: }
232: begin
233:    Tamanho := Lista.Tamanho
234: end;
235:
236:
237: function Vazia (var Lista : Lista_Duplamente_Enc) : boolean;
```

```
238: {
239:   Objetivo: Retorna true se a lista esta vazia
240: }
241: begin
242:    Vazia := Lista.Tamanho = 0
243: end;
244:
245: end.
```