```
 1: unit LstCirC;
 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 Circular com no Cabeca }
25:    Lista_Circular = record
26:                        Cabeca  : Pont_Item_Lista;
27:                        Tamanho : longint;
28:                      end;
29:
30: procedure Inicializar (var Lista : Lista_Circular);
31: function Inserir (Item : Tipo_Item; var Lista : Lista_Circular) : boolean;
32: function Remover (Chave : Tipo_Chave; var Lista : Lista_Circular) : boolean;
33: function Alterar (Item : Tipo_Item; var Lista : Lista_Circular) : boolean;
34: procedure Obter (Chave : Tipo_Chave; var Lista : Lista_Circular;
35:                  var Item : Tipo_Item; var Sucesso : boolean);
36: procedure Apagar (var Lista : Lista Circular);
37: function Tamanho (var Lista : Lista_Circular) : longint;
38: function Vazia (var Lista : Lista_Circular) : boolean;
39:
40: implementation
41:
42: procedure Inicializar (var Lista : Lista_Circular);
43: {
44:   Objetivo: Inicializa a lista passada, criando o no cabeca e fazendo
45:             com que ele aponte circulamente para ele. Alem disso,
46:             faz com que o contador do tamanho seja igual a 0
47: }
48: begin
49:    New(Lista.Cabeca);
50:    Lista.Cabeca^.Proximo := Lista.Cabeca;
51:    Lista.Tamanho := 0;
52:    ReturnNilIfGrowHeapFails := true;
53: end;
54:
55:
56: function AlocarItem (var PItem : Pont_Item_Lista) : boolean;
57: {
58:   Objetivo: Tentar alocar um Item usando o apontador passado,
59:             retornando true se conseguir e false caso contrario.
60: }
61: begin
```

```
62:    New(PItem);
63:    AlocarItem := PItem <> nil;
64: end;
65:
66:
67: function Inserir (Item : Tipo_Item; var Lista : Lista_Circular) : boolean;
68: {
69:   Objetivo: Insere o item passado como parametro na lista passada.
70:             Se a lista ja estiver cheia, a funcao Inserir retorna false.
71: }
72: var
73:    PNovo : Pont_Item_Lista;
74: begin
75:    Inserir := false;
76:
77:    if AlocarItem(PNovo) then
78:       begin
79:          PNovo^.Item := Item;
80:          PNovo^.Proximo := Lista.Cabeca^.Proximo;
81:          Lista.Cabeca^.Proximo := PNovo;
82:          inc(Lista.Tamanho);
83:          Inserir := true
84:       end
85: end;
86:
87:
88: function Remover (Chave : Tipo_Chave; var Lista : Lista_Circular) : boolean;
89: {
90:   Objetivo: Remove o item cuja chave coincide com o parametro Chave
91:             passado. Caso nao haja um item com essa chave, retorna
92:             false. Se o item foi removido, retorna true.
93: }
94: var
95:    PAtual, PAnterior : Pont_Item_Lista;
96: begin
97:    Remover := false;
98:
99:    PAnterior := Lista.Cabeca;
100:
101:   { Percorre a lista ate encontrar um item com a chave procurada.
102:     Remove o item e corrige o apontador do item anterior para
103:     apontar para o proximo item }
104:   while PAnterior^.Proximo <> Lista.Cabeca do
105:   begin
106:      PAtual := PAnterior^.Proximo;
107:
108:      if PAtual^.Item.Chave = Chave then
109:         begin
110:            PAnterior^.Proximo := PAtual^.Proximo;
111:            Dispose(PAtual);
112:            dec(Lista.Tamanho);
113:            Remover := true;
114:            break
115:         end;
116:
117:      PAnterior := PAtual;
118:   end
119: end;
120:
121:
122: function ObterNo (Chave : Tipo_Chave; var Lista : Lista_Circular) :
     Pont_Item_Lista;
```

```
123: {
124:   Objetivo: Retorna um apontador para o No que contem o Item com a chave
125:             igual a passada. Caso nao seja encontrado, a funcao retorna nil
126: }
127: var
128:    PAtual : Pont_Item_Lista;
129: begin
130:    ObterNo := nil;
131:    PAtual := Lista.Cabeca^.Proximo;
132:
133:    while PAtual <> Lista.Cabeca do
134:       if PAtual^.Item.Chave = Chave then
135:          begin
136:             ObterNo := PAtual;
137:             break
138:          end
139:       else
140:          PAtual := PAtual^.Proximo
141: end;
142:
143:
144: procedure Obter (Chave : Tipo_Chave; var Lista : Lista_Circular;
145:                  var Item : Tipo_Item; var Sucesso : boolean);
146: {
147:   Objetivo: Procura na lista usando a chave passada. Caso encontre
148:             Sucesso contem o valor true e Item contem o Item obtido.
149:             Caso contrario, Sucesso retorna true e Item nao e alterado
150: }
151: var
152:    PAtual : Pont_Item_Lista;
153: begin
154:    Sucesso := false;
155:    PAtual := ObterNo(Chave, Lista);
156:
157:    if PAtual <> nil then
158:       begin
159:          Item := PAtual^.Item;
160:          Sucesso := true
161:       end
162: end;
163:
164:
165: function Alterar (Item : Tipo_Item; var Lista : Lista_Circular) : boolean;
166: {
167:   Objetivo: Altera os dados de um item existente na lista passada
168:             de forma que fique igual ao do item passado como parametro.
169:             Se o item for encontrado e alterado, retorna true. Caso
170:             contrario, retorna false.
171: }
172: var
173:    PAtual : Pont_Item_Lista;
174: begin
175:    Alterar := false;
176:    PAtual := ObterNo(Item.Chave, Lista);
177:
178:    if PAtual <> nil then
179:       begin
180:          PAtual^.Item := Item;
181:          Alterar := true
182:       end
183: end;
```

```
184:
185:
186: procedure Apagar (var Lista : Lista_Circular);
187: {
188:   Objetivo: Apaga a lista passada, incluindo o no cabeca. Nao
189:             e correto chamar nenhum outra operacao depois de
190:             Apagar, a nao ser Inicializar
191: }
192: var
193:    PAtual, PApagar : Pont_Item_Lista;
194: begin
195:    PAtual := Lista.Cabeca^.Proximo;
196:
197:    while PAtual <> Lista.Cabeca do
198:    begin
199:       PApagar := PAtual;
200:       PAtual := PAtual^.Proximo;
201:       Dispose(PApagar)
202:    end;
203:
204:    Dispose(Lista.Cabeca);
205:    Lista.Tamanho := -1
206: end;
207:
208:
209: function Tamanho (var Lista : Lista_Circular) : longint;
210: {
211:   Objetivo: Retorna o tamanho da lista passada
212: }
213: begin
214:    Tamanho := Lista.Tamanho;
215: end;
216:
217:
218: function Vazia (var Lista : Lista_Circular) : boolean;
219: {
220:   Objetivo: Retorna true se a lista esta vazia
221: }
222: begin
223:    Vazia := Lista.Tamanho = 0;
224: end;
225:
226: end.
```