```
 1: unit LstCirC;
 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 Circular com no Cabeca }
25:    Lista_Circular = record
26:                        Cabeca  : Pont_Item_Lista;
27:                        Tamanho : integer;
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) : integer;
38: function Cheia (var Lista : Lista_Circular) : boolean;
39: function Vazia (var Lista : Lista_Circular) : boolean;
40:
41: implementation
42:
43: procedure Inicializar (var Lista : Lista_Circular);
44: {
45:   Objetivo: Inicializa a lista passada, criando o no cabeca e fazendo
46:             com que ele aponte circulamente para ele. Alem disso,
47:             faz com que o contador do tamanho seja igual a 0
48: }
49: begin
50:    New(Lista.Cabeca);
51:    Lista.Cabeca^.Proximo := Lista.Cabeca;
52:    Lista.Tamanho := 0
53: end;
54:
55:
56: function Inserir (Item : Tipo_Item; var Lista : Lista_Circular) : boolean;
57: {
58:   Objetivo: Insere o item passado como parametro na lista passada.
59:             Se a lista ja estiver cheia, a funcao Inserir retorna false.
60: }
61: var
```

```
62:    PNovo : Pont_Item_Lista;
63: begin
64:    if Cheia(Lista) then
65:       Inserir := false
66:    else
67:       begin
68:          New(PNovo);
69:          PNovo^.Item := Item;
70:          PNovo^.Proximo := Lista.Cabeca^.Proximo;
71:          Lista.Cabeca^.Proximo := PNovo;
72:          inc(Lista.Tamanho);
73:          Inserir := true
74:       end
75: end;
76:
77:
78: function Remover (Chave : Tipo_Chave; var Lista : Lista_Circular) : boolean;
79: {
80:   Objetivo: Remove o item cuja chave coincide com o parametro Chave
81:             passado. Caso nao haja um item com essa chave, retorna
82:             false. Se o item foi removido, retorna true.
83: }
84: var
85:    PAtual, PAnterior : Pont_Item_Lista;
86: begin
87:    Remover := false;
88:
89:    PAnterior := Lista.Cabeca;
90:
91:    { Percorre a lista ate encontrar um item com a chave procurada.
92:      Remove o item e corrige o apontador do item anterior para
93:      apontar para o proximo item }
94:    while PAnterior^.Proximo <> Lista.Cabeca do
95:    begin
96:       PAtual := PAnterior^.Proximo;
97:
98:       if PAtual^.Item.Chave = Chave then
99:          begin
100:            PAnterior^.Proximo := PAtual^.Proximo;
101:            Dispose(PAtual);
102:            dec(Lista.Tamanho);
103:            Remover := true;
104:            break
105:         end;
106:
107:      PAnterior := PAtual;
108:   end
109: end;
110:
111:
112: function ObterNo (Chave : Tipo_Chave; var Lista : Lista_Circular) :
     Pont_Item_Lista;
113: {
114:   Objetivo: Retorna um apontador para o No que contem o Item com a chave
115:             igual a passada. Caso nao seja encontrado, a funcao retorna nil
116: }
117: var
118:    PAtual : Pont_Item_Lista;
119: begin
120:    ObterNo := nil;
121:    PAtual := Lista.Cabeca^.Proximo;
```

```
122:
123:    while PAtual <> Lista.Cabeca do
124:       if PAtual^.Item.Chave = Chave then
125:          begin
126:             ObterNo := PAtual;
127:             break
128:          end
129:       else
130:          PAtual := PAtual^.Proximo
131: end;
132:
133:
134: procedure Obter (Chave : Tipo_Chave; var Lista : Lista_Circular;
135:                 var Item : Tipo_Item; var Sucesso : boolean);
136: {
137:   Objetivo: Procura na lista usando a chave passada. Caso encontre
138:             Sucesso contem o valor true e Item contem o Item obtido.
139:             Caso contrario, Sucesso retorna true e Item nao e alterado
140: }
141: var
142:    PAtual : Pont_Item_Lista;
143: begin
144:    Sucesso := false;
145:    PAtual := ObterNo(Chave, Lista);
146:
147:    if PAtual <> nil then
148:       begin
149:          Item := PAtual^.Item;
150:          Sucesso := true
151:       end
152: end;
153:
154:
155: function Alterar (Item : Tipo_Item; var Lista : Lista_Circular) : boolean;
156: {
157:   Objetivo: Altera os dados de um item existente na lista passada
158:             de forma que fique igual ao do item passado como parametro.
159:             Se o item for encontrado e alterado, retorna true. Caso
160:             contrario, retorna false.
161: }
162: var
163:    PAtual : Pont_Item_Lista;
164: begin
165:    Alterar := false;
166:    PAtual := ObterNo(Item.Chave, Lista);
167:
168:    if PAtual <> nil then
169:       begin
170:          PAtual^.Item := Item;
171:          Alterar := true
172:       end
173: end;
174:
175:
176: procedure Apagar (var Lista : Lista_Circular);
177: {
178:   Objetivo: Apaga a lista passada, incluindo o no cabeca. Nao
179:             e correto chamar nenhum outra operacao depois de
180:             Apagar, a nao ser Inicializar
181: }
182: var
```

```
183:    PAtual, PApagar : Pont_Item_Lista;
184: begin
185:    PAtual := Lista.Cabeca^.Proximo;
186:
187:    while PAtual <> Lista.Cabeca do
188:    begin
189:       PApagar := PAtual;
190:       PAtual := PAtual^.Proximo;
191:       Dispose(PApagar)
192:    end;
193:
194:    Dispose(Lista.Cabeca);
195:    Lista.Tamanho := -1
196: end;
197:
198:
199: function Tamanho (var Lista : Lista_Circular) : integer;
200: {
201:   Objetivo: Retorna o tamanho da lista passada
202: }
203: begin
204:    Tamanho := Lista.Tamanho;
205: end;
206:
207:
208: function Cheia (var Lista : Lista_Circular) : boolean;
209: {
210:   Objetivo: Retorna true se nao ha mais memoria disponivel
211:             para inserir um item na lista
212: }
213: begin
214:    Cheia := MaxAvail < SizeOf(Item_Lista)
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.
```