1. Especificación del TAD

```
1  espec ListaEstáticaGenérica
2  usa naturales
3  parámetro formal
4      genéro Elemento
5      operación _"="_: Elemento e1, Elemento e2 -> booleano
6      { Devuelve verdad si y solo si e1 y e2 son iguales }
7  fpf
8  género ListaEstática
9  { Los valores del TAD ListaEstática representan una secuencia
10 finita de elementos de tipo Elemento. Una lista es una colección
11 donde puede haber elementos repetidos entre ellos.
12 El TAD ListaEstática dispone de operaciones típicas
13 de contenedores (para crear, añadir o eliminar un elemento,
14 saber si un elemento pertenece a la colección, saber
15 si la colección está vacía, y cuántos elementos contiene). }
16
17 operaciones
18     crear: -> ListaEstática
19     { Devuelve una lista vacía, sin elementos }
20
21     añadir: ListaEstática l, Elemento e ->  ListaEstática
22     { Devuelve una nueva lista igual a la resultante de añadir
23     e a l }
24
25     pertenece?: ListaEstática l, Elemento e -> booleano
26     { Devuelve verdad si y solo si e pertenece a la lista l,
27     falso en caso contrario }
28
29     eliminar: ListaEstática l, Elemento e ->  ListaEstática
30     { Si pertenece?(l, e), devuelve una nueva lista igual a
31     la resultante de eliminar e a l. En caso contrario,
32     devuelve la lista l sin cambios }
33
34     tamaño: ListaEstática -> natural
35     { Devuelve el número de elementos almacenados en la lista l }
36
37     parcial consultarPos: ListaEstática l, Elemento e -> natural
38     { Devuelve la posición del elemento e en la secuencia de
39     elementos contenidos en la lista l. }
40     { Parcial: no está definida si not pertenece?(l, e) }
41
42     parcial consultar: ListaEstática l, natural p -> Elemento
43     { Devuelve el elemento que ocupa la posición p en la secuencia
44     de elementos contenidos en la lista l. }
45     { Parcial: no está definida si p > tamaño(l) }
46
```



```
47      parcial actualizar: ListaEstática l, natural p,
48                                  Elemento e -> ListaEstática
49      { Devuelve la lista resultante de sustituir el elemento que
50      ocupa la posición p en la secuencia de la lista l por e. }
51      { Parcial: no está definida si p > tamaño(l) }
52
53      esVacía?: ListaEstática l -> booleano
54      { Devuelve verdad si y solo si la lista l está vacía, falso en
55      caso contrario }
56
57      { operaciones para el iterador }
58
59      iniciarIterador: ListaEstática l -> ListaEstática
60      { Devuelve una lista igual a la lista l, pero habiendo
61      preparado el iterador para que el siguiente elemento a
62      visitar sea el primer elemento añadido a la lista, si existe
63      (situación de no haber visitado ningún elemento) }
64
65      existeSiguiente?: ListaEstática l -> booleano
66      { Devuelve falso si ya se han visitado todos los elementos
67      contenidos en l. En caso contrario, devuelve verdad }
68
69      parcial siguiente: ListaEstática l -> Elemento e
70      { Devuelve el siguiente elemento a visitar de la lista l.
71      Parcial: no está definida si not existeSiguiente?(l) }
72
73      parcial avanza: ListaEstática l -> ListaEstática
74      { Devuelve la lista resultante tras avanzar el iterador
75      de la lista l al siguiente elemento a visitar.
76      Parcial: no está definida si not existeSiguiente?(l) }}
77  fespec
```


2. Pseudocódigo de la implementación (estática) del TAD

```
1  módulo genérico ListaEstáticaGenérica
2  parámetro
3      tipo elemento { necesitamos ops. de asignación e igualdad }
4      con función "="(e1, e2: elemento) devuelve booleano
5  exporta
6      constante
7          TAM_MAX = 1000
8      tipo listaEstática
9      { Implementación limitada a almacenar un máximo de TAM_MAX
10     elementos }
11
12     procedimiento crear(sal l: listaEstática)
13     { Crea una lista vacía }
14
15     procedimiento añadir(e/s l: listaEstática; ent e: elemento;
16                                  sal error: booleano)
17     { Añade el elemento e a la lista l si hay espacio, y
18     error = falso. Si no, deja l tal cual y error = verdad }
19
20     función pertenece?(l: listaEstática; e: elemento)
21                         devuelve booleano
22     { Comprueba si el elemento e pertenece a la lista l }
23
24     procedimiento eliminar(e/s l: listaEstática; ent e: elemento)
25     { Elimina el elemento e de la lista l, si está presente }
26
27     función tamaño(l: listaEstática) devuelve natural
28     { Devuelve el número de elementos en la lista l }
29
30     procedimiento consultarPos(ent l: listaEstática;
31                                ent e: elemento;
32                                sal p: natural; sal error: booleano)
33     { Si pertenece?(l, e), asigna a p la posición en la secuencia
34     de elementos contenidos en l que ocupa el elemento e y
35     error = falso.
36     En caso contrario, error = verdad y p queda indefinido }
37
38     procedimiento consultar(ent l: listaEstática; ent p: natural;
39                                sal e: elemento; sal error: booleano)
40     { Si p ≤ tamaño(l), consulta el elemento que ocupa la
41     posición p en la secuencia de la lista l, dejándolo en e,
42     y error = falso.
43     En caso contrario, error = verdad y e queda indefinido }
44
```

```
45      procedimiento actualizar(e/s l: listaEstática; ent p: natural;
46                               ent e: elemento; sal error: booleano)
47      { Si p ≤ tamaño(l), sobrescribe el elemento que ocupa la
48      posición p en la secuencia de la lista l con el elemento e,
49      y error = falso. En caso contrario, error = verdad }
50
51      función esVacía?(l: listaEstática) devuelve booleano
52      { Devuelve verdad si el conjunto está vacío, falso en caso
53      contrario }
54
55      { Operaciones para el iterador }
56
57      procedimiento iniciarIterador(e/s l: listaEstática)
58      { Prepara el iterador para que el siguiente elemento a visitar
59      sea el primer elemento insertado en la lista l, si existe
60      (situación de no haber visitado ningún elemento) }
61
62      función existeSiguiente?(l: listaEstática) devuelve booleano
63      { Devuelve falso si ya se han visitado todos los elementos
64      contenidos en l, verdad en caso contrario }
65
66      procedimiento siguiente(e/s l: listaEstática; sal e: elemento;
67                              sal error: booleano)
68      { Implementa las ops. siguiente y avanza de un iterador.
69      Si existeSiguiente?(l), error = falso y e toma el valor del
70      siguiente elemento del conjunto, y se avanza el iterador al
71      siguiente elemento en la lista (siguiendo orden de inserción).
72      En caso contrario, error = verdad, e queda indefinido y l
73      queda como estaba }
74
```



```
75  implementación
76      listaEstática = registro
77                          elementos: vector [1..TAM_MAX] de elemento;
78                          tamaño: natural; { núm. de elementos }
79                          iter: natural; { iterador }
80                      freg
81
82      { Operaciones del TAD listaEstática }
83
84      procedimiento crear(sal l: listaEstática)
85      { Crea una lista vacía }
86      principio
87          l.tamaño := 0;
88      fin
89
90      procedimiento añadir(e/s l: listaEstática; ent e: elemento;
91                                   sal error: booleano)
92      { Añade el elemento e a la lista l si hay espacio, y
93      error = falso. Si no, deja l tal cual y error = verdad }
94      principio
95          si l.tamaño < TAM_MAX entonces
96              l.elementos[l.tamaño + 1] := e;
97              l.tamaño := l.tamaño + 1;
98              error := falso;
99          sino
100             error := verdad;
101         fsi
102     fin
103
104     procedimiento eliminar(e/s l: listaEstática; ent e: elemento)
105     { Elimina el elemento e de la lista l, si está presente }
106     variables
107         i: natural
108     principio
109         i := 1;
110         mientrasQue (i ≤ l.tamaño) hacer
111             si l.elementos[i] = e entonces
112                 { Desplazar los elementos para eliminar e }
113                 para j := i hasta l.tamaño - 1 hacer
114                     l.elementos[j] := l.elementos[j + 1];
115                 fpara
116                 l.tamaño := l.tamaño - 1;
117             fsi
118             i := i + 1;
119         fmq
120     fin
121
```

```
122     función pertenece?(l: listaEstática; e: elemento)
123                         devuelve booleano
124     { Comprueba si el elemento e pertenece a la lista l }
125     variables
126         i: natural;
127      principio
128         para i := 1 hasta l.tamaño hacer
129             si l.elementos[i] = e entonces
130                 devuelve verdad;
131             fsi
132         fpara
133         devuelve falso;
134     fin
135
136     función tamaño(l: listaEstática) devuelve natural
137     { Devuelve el número de elementos en la lista l }
138     principio
139         devuelve l.tamaño;
140     fin
141
142     procedimiento consultarPos(ent l: listaEstática;
143                                ent e: elemento;
144                                sal p: natural; sal error: booleano)
145     { Si pertenece?(l, e), asigna a p la posición en la secuencia
146     de elementos contenidos en l que ocupa el elemento e y
147     error = falso.
148     En caso contrario, error = verdad y p queda indefinido }
149     variables
150         i: natural;
151         encontrado: booleano := falso;
152     principio
153         i := 1;
154         encontrado := falso;
155         mientrasQue (i ≤ l.tamaño andThen not encontrado) hacer
156             si l.elementos[i] = e entonces
157                 encontrado := verdad;
158                 p := i;
159             fsi
160             i := i + 1;
161         fmq
162         error := not encontrado;
163     fin
164
```

```
165     procedimiento consultar(ent l: listaEstática; ent p: natural;
166                             sal e: elemento; sal error: booleano)
167     { Si p ≤ tamaño(l), consulta el elemento que ocupa la
168     posición p en la secuencia de la lista l, dejándolo en e,
169     y error = falso.
170     En caso contrario, error = verdad y e queda indefinido }
171     principio
172         si p ≤ tamaño(l) entonces
173             e := l.elementos[p];
174             error := falso;
175         sino
176             error := verdad;
177         fsi
178     fin
179
180     procedimiento actualizar(e/s l: listaEstática; ent p: natural;
181                              ent e: elemento; sal error: booleano)
182     { Si p ≤ tamaño(l), sobrescribe el elemento que ocupa la
183     posición p en la secuencia de la lista l con el elemento e,
184     y error = falso. En caso contrario, error = verdad }
185     principio
186         si p ≤ tamaño(l) entonces
187             e := l.elementos[p];
188             error := falso;
189         sino
190             error := verdad;
191         fsi
192     fin
193
194     función esVacía?(l: listaEstática) devuelve booleano
195     { Devuelve verdad si el conjunto está vacío, falso en caso
196     contrario }
197     principio
198         devuelve l.tamaño = 0;
199     fin
200
```

```
201     { Operaciones para el iterador }
202
203     procedimiento iniciarIterador(e/s l: listaEstática)
204     { Prepara el iterador para que el siguiente elemento a visitar
205     sea el primer elemento insertado en la lista l, si existe
206     (situación de no haber visitado ningún elemento) }
207     principio
208         l.iter := 1;
209     fin
210
211     función existeSiguiente?(l: listaEstática) devuelve booleano
212     { Devuelve falso si ya se han visitado todos los elementos
213     contenidos en l, verdad en caso contrario }
214     principio
215          devuelve l.iter ≤ l.tamaño
216     fin
217
218     procedimiento siguiente(e/s l: listaEstática; sal e: elemento;
219                 sal error: booleano)
220     { Implementa las ops. siguiente y avanza de un iterador.
221     Si existeSiguiente?(l), error = falso y e toma el valor del
222     siguiente elemento del conjunto, y se avanza el iterador al
223     siguiente elemento en la lista (siguiendo orden de inserción).
224     En caso contrario, error = verdad, e queda indefinido y l
225     queda como estaba }
226     principio
227         error := verdad;
228         si existeSiguiente?(l) entonces
229             e := l.datos[l.iter];
230             l.iter := l.iter + 1;
231             error := falso;
232         fsi
233     fin
234 fin
```


3. Costes de la implementación decidida para el TAD

3.1. Costes espaciales de la estructura

En el momento de crear una variable de tipo `listaEstática`, se asigna todo el espacio necesario para almacenar hasta `TAM_MAX` elementos en un vector. Este es el tamaño máximo predefinido para la estructura, y aunque inicialmente esté vacío o la capacidad de almacenamiento del conjunto no se use en su totalidad, se reserva ese espacio desde el comienzo. Por tanto, el coste en espacio del TAD `listaEstática`, implementado como aparece en el pseudocódigo de la Sección 2, es $\mathcal{O}(\texttt{TAM\_MAX} \cdot e + n)$, donde e es el espacio ocupado en memoria por un `elemento` y n es el espacio ocupado por un natural (campo `tamaño` de la estructura).

3.2. Costes temporales de las operaciones

Sea N el número de elementos contenidos en una lista l. Entonces, los costes temporales de las operaciones implementadas en caso peor son:

Operación	Coste temporal
`crear`	$\mathcal{O}(1)$
`añadir`	$\mathcal{O}(1)$
`pertenece?`	$\mathcal{O}(N)$
`eliminar`	$\mathcal{O}(N)$
`tamaño`	$\mathcal{O}(1)$
`consultarPos`	$\mathcal{O}(N)$
`consultar`	$\mathcal{O}(1)$
`actualizar`	$\mathcal{O}(1)$
`esVacía?`	$\mathcal{O}(1)$

