1. Especificación del TAD

```
1  espec ConjuntoGenérico
2  usa booleanos, naturales, cadenas
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 Conjunto
9  { Los valores del TAD Conjunto representan un conjunto de
10 elementos de tipo Elemento. Un conjunto es una colección
11 sin elementos repetidos y sin ningún orden específico entre
12 ellos. Se requiere por tanto que Elemento disponga de una
13 operación para conocer la igualdad entre dos elementos.
14 El TAD Conjunto dispone de operaciones típicas de
15 contenedores (para crear un conjunto, añadir o eliminar un
16 elemento, saber si un elemento pertenece al conjunto, saber
17 si un conjunto está vacío, y cuántos elementos contiene), así
18 como las relativas a conjuntos en matemáticas (unión ∪,
19 intersección ∩, diferencia \, y subconjunto ⊆). }
20
21 operaciones
22     crear: -> Conjunto
23     { Devuelve un conjunto vacío, sin elementos }
24
25     añadir: Conjunto c, Elemento e -> Conjunto
26     { Si e no pertenece a c, devuelve un nuevo conjunto igual al
27     resultante de añadir e a c. En caso contrario, devuelve c sin
28     cambios }
29
30     eliminar: Conjunto c, Elemento e -> Conjunto
31     { Si e pertenece a c, devuelve un nuevo conjunto igual al
32     resultante de eliminar e de c. En caso contrario, devuelve
33     c sin cambios }
34
35     pertenece?: Conjunto c, Elemento e -> booleano
36     { Devuelve verdad si y solo si e pertenece al conjunto c,
37     falso en caso contrario }
38
39     esVacío?: Conjunto c -> booleano
40     { Devuelve verdad si y solo si el conjunto c no tiene
41     elementos, falso en caso contrario; es decir, |c| = ∅ }
42
43     tamaño: Conjunto c -> natural
44     { Devuelve el número de elementos del conjunto c; es decir,
45     |c| }
46
```

```
47      unir: Conjunto c1, Conjunto c2 -> Conjunto
48      { Devuelve un nuevo conjunto que es la unión de los conjuntos
49      c1 y c2 (sin elementos repetidos; es decir, c1 ∪ c2) }
50
51      intersección: Conjunto c1, Conjunto c2 -> Conjunto
52      { Devuelve un nuevo conjunto con los elementos que pertenecen
53      tanto a c1 como a c2; es decir, c1 ∩ c2 }
54
55      diferencia: Conjunto c1, Conjunto c2 -> Conjunto
56      { Devuelve un nuevo conjunto con los elementos que pertenecen
57      a c1 pero no a c2; es decir, c1 \ c2 }
58
59      subconjunto?: Conjunto c1, Conjunto c2 -> booleano
60      { Devuelve verdad si todos los elementos de c1 pertenecen a c2
61      (es decir, c1 ⊆ c2), falso en caso contrario }
62
63      { operaciones para el iterador }
64
65      iniciarIterador: Conjunto c -> Conjunto
66      { Devuelve un conjunto igual al conjunto c, pero habiendo
67      preparado el iterador para que el siguiente elemento a
68      visitar sea el primer elemento añadido al conjunto, si existe
69      (situación de no haber visitado ningún elemento) }
70
71      existeSiguiente?: Conjunto c -> booleano
72      { Devuelve falso si ya se han visitado todos los elementos
73      contenidos en c. En caso contrario, devuelve verdad }
74
75      parcial siguiente: Conjunto c -> Elemento e
76      { Devuelve el siguiente elemento a visitar del conjunto c.
77      Parcial: no está definida si not existeSiguiente?(c) }
78
79      parcial avanza: Conjunto c -> Conjunto
80      { Devuelve el conjunto resultante tras avanzar el iterador
81      del conjunto c al siguiente elemento a visitar.
82      Parcial: no está definida si not existeSiguiente?(c) }
83
84  fespec
```


2. Pseudocódigo de la implementación (estática) del TAD

```
1  módulo genérico ConjuntoGenérico
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 conjunto
9      { Implementación limitada a almacenar un máximo de TAM_MAX
10     elementos }
11
12     procedimiento crear(sal c: conjunto)
13     { Crea un conjunto vacío }
14
15     procedimiento añadir(e/s c: conjunto; ent e: elemento;
16                                  sal error: booleano)
17     { Añade el elemento e al conjunto c si no está presente y
18     si hay espacio, y error = falso. Si no, deja c tal cual y
19     error = verdad }
20
21     procedimiento eliminar(e/s c: conjunto; ent e: elemento)
22     { Elimina el elemento e del conjunto c, si está presente }
23
24     función pertenece?(c: conjunto; e: elemento) devuelve booleano
25     { Comprueba si el elemento e pertenece al conjunto c }
26
27     función esVacío?(c: conjunto) devuelve booleano
28     { Devuelve verdad si el conjunto está vacío, falso en caso
29     contrario }
30
31     función tamaño(c: conjunto) devuelve natural
32     { Devuelve el número de elementos en el conjunto c }
33
```



```
34      { Operaciones del TAD conjunto relacionadas con conjuntos }
35
36       procedimiento unir(ent c1: conjunto; ent c2: conjunto;
37                          sal c3: conjunto; sal error: booleano)
38       { Si tamaño(c1) and tamaño(c2) ≤ TAM_MAX, crea el conjunto
39       c3 como un nuevo conjunto con todos los elementos de c1 y c2
40       (es decir, c3 = c1 ∪ c2) y error = falso. En caso contrario,
41       error = verdad y c3 queda indefinido }
42
43      procedimiento intersección(ent c1: conjunto; ent c2: conjunto;
44                                 sal c3: conjunto)
45       { Crea el conjunto c3 como un nuevo conjunto con
46       todos los elementos comunes de c1 y c2; es decir, c3 = c1 ∩ c2 }
47
48      procedimiento diferencia(ent c1: conjunto; ent c2: conjunto;
49                               sal c3: conjunto)
50       { Crea el conjunto c3 como un nuevo conjunto con
51       los elementos de c1 que no están en c2, es decir, c3 = c1 \ c2 }
52
53      función subconjunto?(c1: conjunto, c2: conjunto)
54                                 devuelve booleano
55       { Devuelve verdadero si c1 es subconjunto de c2, es decir,
56       c1 ⊆ c2; falso en caso contrario }
57
58      { operaciones para el iterador }
59
60      procedimiento iniciarIterador(e/s c: conjunto)
61      { Prepara el iterador para que el siguiente elemento a visitar
62      sea el primer elemento insertado en el conjunto c, si existe
63      (situación de no haber visitado ningún elemento) }
64
65      función existeSiguiente?(c: conjunto) devuelve booleano
66      { Devuelve falso si ya se han visitado todos los elementos
67      contenidos en c, verdad en caso contrario }
68
69      procedimiento siguiente(e/s c: conjunto; sal e: elemento;
70                  sal error: booleano)
71      { Si existeSiguiente?(c), error = falso y e toma el valor del
72      siguiente elemento del conjunto, y se avanza el iterador al
73      siguiente elemento en el conjunto (en orden de inserción).
74      En caso contrario, error = verdad, e queda indefinido y c
75      queda como estaba }
76
```

```
77  implementación
78      conjunto = registro
79                     elementos: vector [1..TAM_MAX] de elemento;
80                     tamaño: natural; { número actual de elementos }
81                     iter: natural; { iterador }
82                 freg
83
84      { Operaciones del TAD conjunto }
85
86      procedimiento crear(sal c: conjunto)
87      { Crea un conjunto vacío }
88      principio
89          c.tamaño := 0;
90      fin
91
92      procedimiento añadir(e/s c: conjunto; ent e: elemento;
93                                      sal error: booleano)
94      { Añade el elemento e al conjunto c si no está presente y
95      si hay espacio, y error = falso. Si no, deja c tal cual y
96      error = verdad }
97      principio
98          si c.tamaño < TAM_MAX
99                   andThen not pertenece?(c, e) entonces
100             c.elementos[c.tamaño + 1] := e;
101             c.tamaño := c.tamaño + 1;
102             error := falso;
103         sino
104             error := verdad;
105         fsi
106     fin
107
108     procedimiento eliminar(e/s c: conjunto; ent e: elemento)
109     { Elimina el elemento e del conjunto c, si está presente }
110     variables
111         i: natural
112     principio
113         i := 1;
114         mientrasQue (i ≤ c.tamaño) hacer
115             si c.elementos[i] = e entonces
116                 { Desplazar los elementos para eliminar e }
117                 para j := i hasta c.tamaño - 1 hacer
118                     c.elementos[j] := c.elementos[j + 1];
119                 fpara
120                 c.tamaño := c.tamaño - 1;
121             fsi
122             i := i + 1;
123         fmq
124     fin
125
```

```
126     función pertenece?(c: conjunto; e: elemento) devuelve booleano
127     { Comprueba si el elemento e pertenece al conjunto c }
128     variables
129         i: natural;
130      principio
131         para i := 1 hasta c.tamaño hacer
132             si c.elementos[i] = e entonces
133                 devuelve verdadero;
134             fsi
135         fpara
136         devuelve falso;
137     fin
138
139     función esVacío?(c: conjunto) devuelve booleano
140     { Devuelve verdad si el conjunto está vacío, falso en caso
141     contrario }
142     principio
143         devuelve c.tamaño = 0;
144     fin
145
146     función tamaño(c: conjunto) devuelve natural
147     { Devuelve el número de elementos en el conjunto c }
148     principio
149         devuelve c.tamaño;
150     fin
151
```



```
152     { Operaciones del TAD conjunto relacionadas con conjuntos }
153
154     procedimiento unir(ent c1: conjunto; ent c2: conjunto;
155                        sal c3: conjunto)
156     { Si tamaño(c1) and tamaño(c2) ≤ TAM_MAX, crea el conjunto
157     c3 como un nuevo conjunto con todos los elementos de c1 y c2
158     (es decir, c3 = c1 ∪ c2) y error = falso. En caso contrario,
159     error = verdad y c3 queda indefinido }
160     principio
161         crear(c3);
162         si tamaño(c1) + tamaño(c2) ≤ TAM_MAX entonces
163             para i := 1 hasta c1.tamaño hacer
164                 { copiar elementos de c1 a c3 }
165                 añadir(c3, c1.elementos[i]);
166             fpara
167             para i := 1 hasta c2.tamaño hacer
168                 { añade elementos de c2 que no están en c3 }
169                 añadir(c3, c2.elementos[i]);
170             fpara
171             error := falso;
172         sino
173             error := verdad;
174         fsi
175     fin
176
177     procedimiento intersección(ent c1: conjunto; ent c2: conjunto;
178                                sal c3: conjunto)
179      { Crea el conjunto c3 como un nuevo conjunto con
180      todos los elementos comunes de c1 y c2; es decir, c3 = c1 ∩ c2 }
181      variables
182         aux: elemento;
183      principio
184         para i := 1 hasta c1.tamaño hacer
185             aux := c1.elementos[i];
186             si pertenece?(c2, aux) entonces
187                 añadir(c3, aux);
188             fsi
189         fpara
190      fin
191
```

```
192     procedimiento diferencia(ent c1: conjunto; ent c2: conjunto;
193                              sal c3: conjunto)
194      { Crea el conjunto c3 como un nuevo conjunto con
195      los elementos de c1 que no están en c2, es decir, c3 = c1 \ c2 }
196      variables
197         aux: elemento;
198      principio
199         para i := 1 hasta c1.tamaño hacer
200             aux := c1.elementos[i];
201             si not pertenece?(c2, aux) entonces
202                 añadir(c3, aux);
203             fsi
204         fpara
205      fin
206
207     función subconjunto?(c1: conjunto, c2: conjunto)
208                              devuelve booleano
209      { Devuelve verdadero si c1 es subconjunto de c2, es decir,
210      c1 ⊆ c2; falso en caso contrario }
211     variables
212         i : natural;
213      principio
214         para i := 1 hasta c1.tamaño hacer
215             si not pertenece?(c2, c1.elementos[i]) entonces
216                 devuelve falso;
217             fsi
218         fpara
219         devuelve verdad;
220      fin
221
```

```
222     { Operaciones para el iterador }
223
224     procedimiento iniciarIterador(e/s c: conjunto)
225     { Prepara el iterador para que el siguiente elemento a visitar
226     sea el primer elemento insertado en el conjunto c, si existe
227     (situación de no haber visitado ningún elemento) }
228     principio
229         c.iter := 1;
230     fin
231
232     función existeSiguiente?(c: conjunto) devuelve booleano
233     { Devuelve falso si ya se han visitado todos los elementos
234     contenidos en c, verdad en caso contrario }
235     principio
236          devuelve c.iter ≤ c.tamaño
237     fin
238
239     procedimiento siguiente(e/s c: conjunto; sal e: elemento;
240                  sal error: booleano)
241     { Si existeSiguiente?(c), error = falso y e toma el valor del
242     siguiente elemento del conjunto, y se avanza el iterador al
243     siguiente elemento en el conjunto (en orden de inserción).
244     En caso contrario, error = verdad, e queda indefinido y c
245     queda como estaba }
246     principio
247         error := verdad;
248         si existeSiguiente?(c) entonces
249             e := c.datos[c.iter];
250             c.iter := c.iter + 1;
251             error := falso;
252         fsi
253     fin
254
255 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 `conjunto`, 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 `conjunto`, 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 un conjunto c. Sean N_1 y N_2 el número de elementos contenidos en los conjuntos c_1 y c_2, respectivamente. Entonces, los costes temporales de las operaciones implementadas en caso peor son:

Operación	Coste temporal
`crear`	$\mathcal{O}(1)$
`añadir`	$\mathcal{O}(N)$
`eliminar`	$\mathcal{O}(N)$
`pertenece?`	$\mathcal{O}(N)$
`esVacío?`	$\mathcal{O}(1)$
`tamaño`	$\mathcal{O}(1)$
`unir`	$\mathcal{O}(N_1 + N_2)$
`intersección`	$\mathcal{O}(N_1 \cdot N_2)$
`diferencia`	$\mathcal{O}(N_1 \cdot N_2)$
`subconjunto?`	$\mathcal{O}(N_1 \cdot N_2)$

