

Laboratorio de PRO2. Caso de estudio: experimentos inmunologicos.  
v5.0 13-11-2014

Generado por Doxygen 1.8.8

Miércoles, 11 de Noviembre de 2015 16:12:12



# Índice general

<b>1</b>	<b>Página principal</b>	<b>1</b>
<b>2</b>	<b>Índice de clases</b>	<b>3</b>
2.1	Lista de clases . . . . .	3
<b>3</b>	<b>Índice de archivos</b>	<b>5</b>
3.1	Lista de archivos . . . . .	5
<b>4</b>	<b>Documentación de las clases</b>	<b>7</b>
4.1	Referencia de la Clase Celula . . . . .	7
4.1.1	Descripción detallada . . . . .	8
4.1.2	Documentación del constructor y destructor . . . . .	8
4.1.2.1	Celula . . . . .	8
4.1.3	Documentación de las funciones miembro . . . . .	8
4.1.3.1	lucha_celulas . . . . .	8
4.1.3.2	es_vacia . . . . .	9
4.1.3.3	leer . . . . .	9
4.1.3.4	escribir . . . . .	10
4.1.4	Documentación de los datos miembro . . . . .	10
4.1.4.1	ID_VACIA . . . . .	10
4.1.4.2	id . . . . .	10
4.1.4.3	i_tol . . . . .	10
4.1.4.4	param . . . . .	10
4.2	Referencia de la Clase Organismo . . . . .	10
4.2.1	Descripción detallada . . . . .	11
4.2.2	Documentación del constructor y destructor . . . . .	12
4.2.2.1	Organismo . . . . .	12
4.2.3	Documentación de las funciones miembro . . . . .	12
4.2.3.1	anadir_id . . . . .	12
4.2.3.2	incrementar_victimas . . . . .	12
4.2.3.3	lucha_organismos . . . . .	13
4.2.3.4	es_maligno . . . . .	13

4.2.3.5	num_victimas	14
4.2.3.6	leer	14
4.2.3.7	escribir	14
4.2.3.8	simetricos	15
4.2.3.9	lucha_arboles	15
4.2.3.10	leer_arbol_celulas	16
4.2.3.11	escribir_arbol_celulas_id	16
4.2.4	Documentación de los datos miembro	17
4.2.4.1	celulas	17
4.2.4.2	id	17
4.2.4.3	maligno	17
4.2.4.4	victimas	17
4.3	Referencia de la Clase Sistema	17
4.3.1	Descripción detallada	18
4.3.2	Documentación del constructor y destructor	18
4.3.2.1	Sistema	18
4.3.3	Documentación de las funciones miembro	19
4.3.3.1	anadir_organismo	19
4.3.3.2	leer	19
4.3.3.3	escribir	20
4.3.3.4	luchas_orgCola	20
4.3.3.5	clear	21
4.3.3.6	recolocar	22
4.3.3.7	escribir_sistemaCola	22
4.3.4	Documentación de los datos miembro	22
4.3.4.1	def	22
4.3.4.2	mal	23
4.3.4.3	id	23
<b>5</b>	<b>Documentación de archivos</b>	<b>25</b>
5.1	Referencia del Archivo Celula.cc	25
5.1.1	Descripción detallada	25
5.2	Referencia del Archivo Celula.hh	25
5.2.1	Descripción detallada	26
5.3	Referencia del Archivo Organismo.cc	26
5.3.1	Descripción detallada	26
5.4	Referencia del Archivo Organismo.hh	26
5.4.1	Descripción detallada	27
5.5	Referencia del Archivo pro2.cc	27
5.5.1	Descripción detallada	28

5.5.2	Documentación de las funciones . . . . .	28
5.5.2.1	main . . . . .	28
5.6	Referencia del Archivo readbool.hh . . . . .	29
5.6.1	Descripción detallada . . . . .	29
5.6.2	Documentación de las funciones . . . . .	29
5.6.2.1	readbool . . . . .	29
5.7	Referencia del Archivo Sistema.cc . . . . .	29
5.7.1	Descripción detallada . . . . .	30
5.8	Referencia del Archivo Sistema.hh . . . . .	30
5.8.1	Descripción detallada . . . . .	31
<b>Índice</b>		<b>32</b>



# Capítulo 1

## Página principal

Ejemplo de práctica resuelta, con documentación **completa** (incluyendo elementos privados y código).

El programa principal se encuentra en el módulo [pro2.cc](#). Atendiendo a los tipos de datos sugeridos en el enunciado, necesitaremos un módulo para representar el [Sistema](#) en el que se desarrollarán los experimentos, otro para el tipo [Organismo](#) y otro para el tipo [Celula](#).

Comentarios:

- En una resolución normal, comenzaríamos por considerar las operaciones necesarias para el programa principal y las clasificaríamos en los diferentes módulos. Al pasar a su implementación, quizá descubriésemos que algún módulo necesita alguna operación adicional y la incorporaríamos en ese momento (sólo si es pública, es decir, si se usa en un módulo distinto al que pertenece). Sin embargo, en un documento de estas características, se presentan los módulos completamente acabados, sin necesidad de reflejar el proceso que ha dado lugar a su especificación final.
- En cuanto a los diagramas modulares que aparecen en este proyecto, notad que la relación de uso entre [Organismo](#) y [Celula](#) no se obtiene a partir de la especificación de los elementos públicos del primero, sino de la de sus elementos privados.





## Capítulo 2

# Índice de clases

### 2.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

<a href="#">Celula</a>	Representa el conjunto de características y operaciones de las células . . . . .	7
<a href="#">Organismo</a>	Representa la información y las operaciones asociadas a un organismo . . . . .	10
<a href="#">Sistema</a>	Representa el sistema donde se desarrollan los experimentos . . . . .	17



## Capítulo 3

# Indice de archivos

### 3.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

<a href="#">Celula.cc</a>	Código de la clase <a href="#">Celula</a> . . . . .	25
<a href="#">Celula.hh</a>	Especificación de la clase <a href="#">Celula</a> . . . . .	25
<a href="#">Organismo.cc</a>	Código de la clase <a href="#">Organismo</a> . . . . .	26
<a href="#">Organismo.hh</a>	Especificación de la clase <a href="#">Organismo</a> . . . . .	26
<a href="#">pro2.cc</a>	Programa principal . . . . .	27
<a href="#">readbool.hh</a>	Operacion para leer booleanos del canal estandar . . . . .	29
<a href="#">Sistema.cc</a>	Código de la clase <a href="#">Sistema</a> . . . . .	29
<a href="#">Sistema.hh</a>	Especificación de la clase <a href="#">Sistema</a> . . . . .	30



## Capítulo 4

# Documentación de las clases

### 4.1. Referencia de la Clase Celula

Representa el conjunto de características y operaciones de las células.

#### Métodos públicos

- `Celula ()`  
*Creadora por defecto.*
- `int lucha_celulas (const Celula &c2) const`  
*Consultora que determina el resultado de la lucha entre dos células.*
- `bool es_vacia () const`  
*Consultora que indica si la célula es vacía.*
- `void leer (int N)`  
*Operación de lectura.*
- `void escribir () const`  
*Operación de escritura.*

#### Atributos privados

- `int id`  
*Identificador de la célula.*
- `int i_tol`  
*Indice de tolerancia.*
- `vector< double > param`  
*Parámetros de la célula.*

#### Atributos privados estáticos

- `static const int ID_VACIA = 0`  
*Identificador especial para células vacías.*

#### 4.1.1. Descripción detallada

Representa el conjunto de características y operaciones de las células.

Ofrece la operación de lucha entre células y las operaciones de lectura y escritura.

El hecho de que las células se lean en el contexto de la lectura de árboles de células que componen organismos nos induce a definir el concepto de célula vacía para poder leer árboles siguiendo un formato de entrada parecido al de la sesión 7 de laboratorio, "Uso de listas y árboles", en la que se define una "marca" para indicar subárbol vacío.

Definición en la línea 32 del archivo Celula.hh.

#### 4.1.2. Documentación del constructor y destructor

##### 4.1.2.1. Celula::Celula ( )

Creadora por defecto.

##### Precondición

cierto

##### Postcondición

El resultado es una célula vacía, con índice de tolerancia cero y un número de parámetros indeterminado

Definición en la línea 7 del archivo Celula.cc.

```
8 {
9  // Inicializa una célula con el id de célula vacía
10  id = ID_VACIA;
11  i_tol = 0;
12 }
```

#### 4.1.3. Documentación de las funciones miembro

##### 4.1.3.1. int Celula::lucha\_celulas ( const Celula & c2 ) const

Consultora que determina el resultado de la lucha entre dos células.

##### Precondición

El parámetro implícito (c1) y c2 tienen el mismo número de parámetros

##### Postcondición

Retorna el resultado de la lucha entre c1 y c2, que vale 1 si y solo si c1 vence a c2; 2 si y solo si c2 vence a c1; 3 si y solo si no vence ninguna de las dos

Definición en la línea 14 del archivo Celula.cc.

```
15 {
16  // Se trata de obtener la diferencia entre el número de posiciones de la primera
17  // célula que superan a las de la segunda y viceversa. Después se compara dicha
18  // diferencia con los indicadores de tolerancia.
19  int i = 0;
20  int dif = 0;
21
22  // Inv: dif = diferencia entre el número de posiciones en [0..i-1] en que c1 supera a c2
23  // y viceversa, 0<=i<=param.size()
24 }
```

```
25 while (i < param.size()) {
26     if (param[i] > c2.param[i]) ++dif;
27     else if (param[i] < c2.param[i]) --dif;
28     ++i;
29 }
30 // Post1: dif = diferencia entre el número de posiciones totales en que
31 // c1 supera a c2 y viceversa
32 int n = 3;
33 if (dif > c2.i_tol) n = 1;
34 else if (dif < -i_tol) n = 2;
35 return n;
36 }
```

#### 4.1.3.2. bool Celula::es\_vacia ( ) const

Consultora que indica si la célula es vacía.

##### Precondición

cierto

##### Postcondición

El resultado indica si la célula es vacía o no

Definición en la línea 38 del archivo Celula.cc.

```
39 {
40     return id == ID_VACIA;
41 }
```

#### 4.1.3.3. void Celula::leer ( int N )

Operación de lectura.

##### Precondición

$N > 0$

##### Postcondición

El parámetro implícito pasa a tener un identificador, un índice de tolerancia y N parámetros nuevos, leídos del canal de entrada estándar, salvo si el identificador leído es el de una célula vacía

Definición en la línea 43 del archivo Celula.cc.

```
44 {
45     // Simplemente lee todos los componentes de la célula teniendo en cuenta que si
46     // el identificador es cero se trata de una celula "marca" y no es necesario continuar leyendo.
47
48     if (N<=0) throw PRO2Excepcio("La celula ha de tener N parametros (N>0)");
49
50     cin >> id;
51     if (id != ID_VACIA) {
52         cin >> i_tol;
53         param = vector<double> (N);
54         for (int i = 0; i < N; ++i) {
55             cin >> param[i];
56         }
57     }
58 }
```

#### 4.1.3.4. void Celula::escribir ( ) const

Operación de escritura.

##### Precondición

cierto

##### Postcondición

Se ha escrito el identificador del parámetro implícito por el canal de salida estándar

Definición en la línea 60 del archivo Celula.cc.

```
61 {  
62     // Análogamente, se trata sólo de escribir los componentes que nos interesan,  
63     // en este caso, según el enunciado, el identificador de la célula.  
64     cout << id;  
65 }
```

#### 4.1.4. Documentación de los datos miembro

##### 4.1.4.1. const int Celula::ID\_VACIA = 0 [static], [private]

Identificador especial para células vacías.

Definición en la línea 37 del archivo Celula.hh.

##### 4.1.4.2. int Celula::id [private]

Identificador de la célula.

Definición en la línea 39 del archivo Celula.hh.

##### 4.1.4.3. int Celula::i\_tol [private]

Índice de tolerancia.

Definición en la línea 41 del archivo Celula.hh.

##### 4.1.4.4. vector<double> Celula::param [private]

Parámetros de la célula.

Definición en la línea 43 del archivo Celula.hh.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [Celula.hh](#)
- [Celula.cc](#)

## 4.2. Referencia de la Clase Organismo

Representa la información y las operaciones asociadas a un organismo.



### Métodos públicos

- **Organismo** ()  
*Creadora por defecto.*
- void **anadir\_id** (int id)  
*Modificadora del identificador.*
- void **incrementar\_victimas** ()  
*Modificadora del número de víctimas.*
- int **lucha\_organismos** (const **Organismo** &o2) const  
*Consultora del resultado de la lucha entre dos organismos.*
- bool **es\_maligno** () const  
*Consultora de la malignidad del organismo.*
- int **num\_victimas** () const  
*Consultora del número de víctimas.*
- void **leer** (int N)  
*Operación de lectura.*
- void **escribir** (bool estr) const  
*Operación de escritura.*

### Métodos privados estáticos

- static bool **simetricos** (Arbre< **Celula** > &a1, Arbre< **Celula** > &a2)  
*Comprobación de simetría de dos árboles.*
- static pair< int, int > **lucha\_arboles** (Arbre< **Celula** > &a1, Arbre< **Celula** > &a2)  
*Lucha de dos árboles de células.*
- static void **leer\_arbol\_celulas** (int N, Arbre< **Celula** > &a)  
*Operación de lectura de un árbol de células.*
- static void **escribir\_arbol\_celulas\_id** (Arbre< **Celula** > &a)  
*Operación de escritura de un árbol de células.*

### Atributos privados

- Arbre< **Celula** > **celulas**  
*Estructura celular del organismo.*
- int **id**  
*Identificador del organismo.*
- bool **maligno**  
*Indica si es maligno (true) o defensivo (false)*
- int **victimas**  
*Número de víctimas del organismo.*

#### 4.2.1. Descripción detallada

Representa la información y las operaciones asociadas a un organismo.

Sus operaciones son las modificadoras de identificador y de número de organismos destruidos, las consultoras de si un organismo es maligno y la de su número de víctimas, la que devuelve el resultado de una lucha de dos organismos, la de lectura (única que produce un organismo nuevo) y la de escritura.

Notad que hemos declarado las operaciones auxiliares como *private* y *static*. Recordad que las operaciones *static* no admiten calificadores como *const*

Definición en la línea 26 del archivo Organismo.hh.

### 4.2.2. Documentación del constructor y destructor

#### 4.2.2.1. Organismo::Organismo ( )

Creadora por defecto.

##### Precondición

cierto

##### Postcondición

El resultado es un organismo defensivo, con id=0, sin células y sin víctimas

Definición en la línea 11 del archivo Organismo.cc.

```
12 {  
13     id=0;  
14     maligno=false;  
15     victimas=0;  
16 }
```

### 4.2.3. Documentación de las funciones miembro

#### 4.2.3.1. void Organismo::anadir\_id ( int id )

Modificadora del identificador.

##### Precondición

cierto

##### Postcondición

El parámetro implícito pasa a tener a *id* como identificador

Definición en la línea 23 del archivo Organismo.cc.

```
24 {  
25     this->id = id;  
26 }
```

#### 4.2.3.2. void Organismo::incrementar\_victimas ( )

Modificadora del número de víctimas.

##### Precondición

cierto

##### Postcondición

El parámetro implícito pasa a tener una víctima más en su cuenta

Definición en la línea 18 del archivo Organismo.cc.

```
19 {  
20     ++victimas;  
21 }
```

4.2.3.3. `int Organismo::lucha_organismos ( const Organismo & o2 ) const`

Consultora del resultado de la lucha entre dos organismos.

**Precondición**

El parámetro implícito (o1) y o2 están compuestos por células con el mismo número de parámetros

**Postcondición**

Retorna el resultado de la lucha entre o1 y o2, que vale 0 si y solo si o1 y o2 resultan destruidos; 1 si y solo si o1 resulta destruido y o2 no; 2 si y solo si o1 no resulta destruido y o2 sí; 3 si y solo si ni o1 ni o2 resultan destruidos

Definición en la línea 28 del archivo Organismo.cc.

```

29 {
30     // Ésta es la operación más importante del módulo. Dados dos organismos, hay
31     // que decidir primero si van a luchar de verdad o no, es decir, si sus
32     // estructuras celulares son simétricas o no. Por eso, introducimos la
33     // operación auxiliar "simetricos" en la parte privada. Notad que en la
34     // cabecera de la operación "simetricos" decimos que los parámetros han de
35     // ser árboles de Celula, pero la operación es independiente del tipo de los
36     // elementos del árbol, ya que éstos nunca se llegan a consultar. Asimismo,
37     // debemos disponer de otra operación que aplique las luchas de las células
38     // de dos árboles, sabiendo que son simétricos. Aquí sí es relevante el
39     // hecho de que los árboles sean de células. Esta nueva operación privada se
40     // llama "lucha_arboles".
41
42     int n;
43     Arbre<Celula> aux1(celulas);           // podemos prescindir de aux1 y aux2
44     Arbre<Celula> aux2(o2.celulas);        // si simetricos y lucha_arboles
45     if (simetricos(aux1, aux2)) {          // replantan sus parámetros para no
46         aux1 = celulas;                   // destruirlos, pero entonces
47         aux2 = o2.celulas;                // lucha_organismos no puede ser const
48         pair<int, int> m = lucha_arboles(aux1, aux2);
49         if (m.first == m.second) n = 0;
50         else if (m.first < m.second) n = 1;
51         else n = 2; // m.first > m.second
52     }
53     else n = 3;
54     return n;
55 }
```

4.2.3.4. `bool Organismo::es_maligno ( ) const`

Consultora de la malignidad del organismo.

**Precondición**

cierto

**Postcondición**

El resultado es cierto si el parametro implícito es un organismo maligno y falso en caso contrario

Definición en la línea 107 del archivo Organismo.cc.

```

108 {
109     // Devuelve el valor del campo "maligno" del organismo correspondiente.
110     return maligno;
111 }
```

#### 4.2.3.5. `int Organismo::num_victimas ( ) const`

Consultora del número de víctimas.

##### Precondición

cierto

##### Postcondición

El resultado es el número de organismos destruidos por el parámetro implícito

Definición en la línea 113 del archivo Organismo.cc.

```
114 {  
115     // Devuelve el valor del campo "victimas" del organismo correspondiente.  
116     return victimas;  
117 }
```

#### 4.2.3.6. `void Organismo::leer ( int N )`

Operación de lectura.

##### Precondición

$N > 0$

##### Postcondición

El parámetro implícito es un organismo tomado del canal de entrada estándar formado por células de N parámetros

Definición en la línea 119 del archivo Organismo.cc.

```
120 {  
121     // Esta operación simplemente lee la estructura celular del organismo e  
122     // inicializa el recuento de víctimas. Un árbol de células se lee igual que  
123     // un árbol de enteros. Suponemos que contamos con una marca de tipo Celula  
124     // para indicar que llegamos a un árbol vacío.  
125  
126     if (N<=0) throw PRO2Excepcio("Las celulas del organismo han de tener N parametros (N>0)");  
127  
128     celulas.a_buit();  
129     leer_arbol_celulas(N, celulas);  
130     maligno = readbool();  
131     victimas = 0;  
132 }
```

#### 4.2.3.7. `void Organismo::escribir ( bool estr ) const`

Operación de escritura.

##### Precondición

cierto

**Postcondición**

Se ha escrito el identificador del parámetro implícito y el número de rivales que ha destruido por el canal de salida estándar; si estr es cierto también se ha escrito su estructura celular

Definición en la línea 146 del archivo Organismo.cc.

```

147 {
148     // Escribimos los campos del organismo. Un árbol de células se escribe
149     // igual que un árbol de enteros, suponiendo que (como es el caso) exista
150     // una operación para escribir células.
151     cout << id << " " << victimas;
152     if (estr) {
153         Arbre<Celula> copia(celulas);
154         escribir_arbol_celulas_id(copia);
155     }
156     cout << endl;
157 }

```

**4.2.3.8. bool Organismo::simetricos ( Arbre< Celula > &a1, Arbre< Celula > &a2 ) [static], [private]**

Comprobación de simetría de dos árboles.

**Precondición**

a1 = A1; a2 = A2

**Postcondición**

El resultado indica si A1 y A2 son simétricos

Definición en la línea 57 del archivo Organismo.cc.

```

58 {
59     bool b;
60     if (a1.es_buit() or a2.es_buit()) b = a1.es_buit() and a2.es_buit();
61     else {
62         Arbre<Celula> fe1, fe2, fd1, fd2;
63         a1.fills(fe1,fd1); // fe1 = hi(A1); fd1 = hd(A2); a1 = buit
64         a2.fills(fe2,fd2); // fe2 = hi(A2); fd1 = hd(A2); a2 = buit
65         b = simetricos(fe1, fd2);
66         // HI1: b indica si hi(A1) y hd(A2) son simétricos
67         if (b) { b = simetricos(fd1, fe2);
68             // HI2: b indica si hd(A1) y hi(A2) son simétricos
69         }
70     }
71     return b;
72 }

```

**4.2.3.9. pair< int, int > Organismo::lucha\_arboles ( Arbre< Celula > &a1, Arbre< Celula > &a2 ) [static], [private]**

Lucha de dos árboles de células.

**Precondición**

a1 y a2 son simétricos y están compuestos por células con el mismo número de parámetros; a1 = A1; a2 = A2

**Postcondición**

El primer componente del resultado es el número de células de A1 que vencen a su correspondiente en A2; el segundo es el número de células de A2 que vencen a su correspondiente en A1

Definición en la línea 74 del archivo Organismo.cc.

```

75 {
76     pair<int,int> n;
77
78     if (a1.es_buit()) {
79         n.first = 0;
80         n.second = 0;
81     }
82     else {
83         int nraiz = a1.arrel().lucha_celulas(a2.arrel());
84         n.first = 0;
85         if (nraiz == 1) ++n.first;
86         n.second = 0;
87         if (nraiz == 2) ++n.second;
88         Arbre<Celula> fel, fe2, fd1, fd2;
89         a1.fill(fel,fd1); // fel = hi(A1); fd1 = hd(A2); a1 = buit
90         a2.fill(fe2,fd2); // fe2 = hi(A2); fd1 = hd(A2); a2 = buit
91         pair<int,int> na = lucha_arboles(fel, fd2);
92         //      HI1: na.first = número de células de hi(A1) que vencen a su
93         //              correspondiente en hd(A2);
94         //      na.second = número de células de hd(A2) que vencen a su
95         //              correspondiente en hi(A1)
96         pair<int,int> nb = lucha_arboles(fd1, fe2);
97         //      HI2: nb.first = número de células de hd(A1) que vencen a su
98         //              correspondiente en hi(A2);
99         //      nb.second = número de células de hi(A2) que vencen a su
100        //              correspondiente en hd(A1)
101         n.first += na.first + nb.first;
102         n.second += nb.second + na.second;
103     }
104     return n;
105 }

```

**4.2.3.10.** void Organismo::leer\_arbol\_celulas ( int N, Arbre< Celula > & a ) [static],[private]

Operación de lectura de un árbol de células.

**Precondición**

$N > 0$ ; a es vacío

**Postcondición**

a contiene el árbol de células leído de la entrada

Definición en la línea 134 del archivo Organismo.cc.

```

135 {
136     Arbre<Celula> a1, a2;
137     Celula c;
138     c.leer(N);
139     if (not c.es_vacia()) {
140         leer_arbol_celulas(N, a1);
141         leer_arbol_celulas(N, a2);
142         a.plantar(c, a1, a2);
143     }
144 }

```

**4.2.3.11.** void Organismo::escribir\_arbol\_celulas\_id ( Arbre< Celula > & a ) [static],[private]

Operación de escritura de un árbol de células.

**Precondición**

cierto

**Postcondición**

Se ha escrito a por el canal de salida estándar

Definición en la línea 159 del archivo Organismo.cc.

```
160 {  
161     if (not a.es_buit()) {  
162         Arbre<Celula> a1, a2;  
163         Celula aux = a.arrel();  
164         a.fillls(a1,a2);  
165         escribir_arbol_celulas_id(a1);  
166         cout << " ";  
167         aux.escribir();  
168         escribir_arbol_celulas_id(a2);  
169     }  
170 }
```

**4.2.4. Documentación de los datos miembro****4.2.4.1. Arbre<Celula> Organismo::celulas [private]**

Estructura celular del organismo.

Definición en la línea 37 del archivo Organismo.hh.

**4.2.4.2. int Organismo::id [private]**

Identificador del organismo.

Definición en la línea 39 del archivo Organismo.hh.

**4.2.4.3. bool Organismo::maligno [private]**

Indica si es maligno (true) o defensivo (false)

Definición en la línea 41 del archivo Organismo.hh.

**4.2.4.4. int Organismo::victimas [private]**

Número de víctimas del organismo.

Definición en la línea 43 del archivo Organismo.hh.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [Organismo.hh](#)
- [Organismo.cc](#)

**4.3. Referencia de la Clase Sistema**

Representa el sistema donde se desarrollan los experimentos.

## Métodos públicos

- **Sistema** ()  
*Creadora por defecto.*
- void **anadir\_organismo** (**Organismo** &o, bool &sobrevive)  
*Modificadora que gestiona el intento de entrada de un organismo en el sistema.*
- void **leer** (int N)  
*Operación de lectura.*
- void **escribir** (bool tipo, bool estr) const  
*Operación de escritura.*

## Métodos privados estáticos

- static void **luchas\_orgCola** (queue< **Organismo** > &c, **Organismo** &o, bool &sobrevive)  
*Lucha de un organismo contra los organismos de una cola.*
- static void **clear** (queue< **Organismo** > &q)  
*Elimina todos los elementos de una cola de organismos.*
- static void **recolocar** (int n, queue< **Organismo** > &c)  
*Pasa los n primeros organismos de una cola al final de ésta, conservando el orden relativo.*
- static void **escribir\_sistemaCola** (const queue< **Organismo** > &c, bool estr)  
*Operación de escritura de una cola de organismos.*

## Atributos privados

- queue< **Organismo** > **def**  
*Cola de organismos defensivos.*
- queue< **Organismo** > **mal**  
*Cola de organismos malignos.*
- int **id**  
*Primer número disponible de identificador de organismo.*

### 4.3.1. Descripción detallada

Representa el sistema donde se desarrollan los experimentos.

Ofrece operaciones para introducir un organismo en el sistema, de lectura y escritura del sistema.

Definición en la línea 24 del archivo Sistema.hh.

### 4.3.2. Documentación del constructor y destructor

#### 4.3.2.1. Sistema::Sistema ( )

Creadora por defecto.

Precondición

cierto



**Postcondición**

El resultado es un sistema en que no ha entrado ningún organismo

Definición en la línea 7 del archivo Sistema.cc.

```
8 {  
9   id = 1;  
10 }
```

**4.3.3. Documentación de las funciones miembro****4.3.3.1. void Sistema::anadir\_organismo ( Organismo & o, bool & sobrevive )**

Modificadora que gestiona el intento de entrada de un organismo en el sistema.

**Precondición**

o es un organismo sin identificador

**Postcondición**

El parámetro implícito contiene el estado del sistema después del intento de entrada del organismo o; o pasa a tener identificador y a contener el número de organismos que ha destruido; sobrevive indica si o queda vivo en el parámetro implícito o no

Definición en la línea 12 del archivo Sistema.cc.

```
13 {  
14 // Proporciona su identificador al nuevo organismo y organiza las luchas de  
15 // éste. Para ello, emplea la operación privada "luchas_orgCola" que lo enfrenta  
16 // a los de la cola correspondiente y obtiene la información necesaria.  
17   o.anadir_id(id);  
18   ++id;  
19   if (o.es_maligno()) {  
20     luchas_orgCola(def, o, sobrevive);  
21     if (sobrevive) mal.push(o);  
22   }  
23   else {  
24     luchas_orgCola(mal, o, sobrevive);  
25     if (sobrevive) def.push(o);  
26   }  
27 }
```

**4.3.3.2. void Sistema::leer ( int N )**

Operación de lectura.

**Precondición**

$N > 0$

**Postcondición**

El parámetro implícito está formado por un grupo de organismos leídos por el canal de entrada estándar y formados por células de N parámetros

Definición en la línea 107 del archivo Sistema.cc.

```
108 {  
109 // Esta operación actualiza todos los componentes de un sistema, leyéndolos  
110 // del canal estándar; ofrece la posibilidad de leer y almacenar algunos  
111 // organismos defensivos. El número de éstos se proporciona al comienzo de
```

```

112 // la operación.
113
114 if (N<=0) throw PRO2Excepcio("Las celulas de los organismos del sistema han de tener N parametros (N>0)")
    ;
115
116 clear(def);
117 clear(mal);
118 int num; // numero de organismos iniciales del sistema
119 cin >> num;
120 Organismo o;
121 for (int i = 1; i <= num; ++i) {
122     o.leer(N);
123     o.anadir_id(i);
124     if (o.es_maligno()) mal.push(o);
125     else def.push(o);
126 }
127 id = num+1;
128 }

```

#### 4.3.3.3. void Sistema::escribir ( bool *tipo*, bool *estr* ) const

Operación de escritura.

##### Precondición

cierto

##### Postcondición

Si *tipo* es cierto, se han escrito en el canal de salida estándar, por orden de identificador, los organismos defensivos del parámetro implícito, en caso contrario se han escrito los malignos; si *estr* es cierto, cada organismo se escribe con su estructura celular, en caso contrario sólo se escribe su identificador

Definición en la línea 130 del archivo Sistema.cc.

```

131 {
132 // Esta operación simplemente recorre la cola de los organismos pedidos,
133 // proporcionando la información requerida, mediante una operación privada
134 // auxiliar: "escribir_sistema_cola".
135
136 if (tipo) escribir_sistema_cola(def, estr);
137 else escribir_sistema_cola(mal, estr);
138 }

```

#### 4.3.3.4. void Sistema::luchas\_org\_cola ( queue< Organismo > & *c*, Organismo & *o*, bool & *sobrevive* ) [static], [private]

Lucha de un organismo contra los organismos de una cola.

##### Precondición

*c* = C; C está ordenada crecientemente según los identificadores de sus organismos

##### Postcondición

*c* contiene los organismos de C, ordenados crecientemente por identificador y con su contador de víctimas actualizado, excepto los que hayan muerto al enfrentarse con *o*; *sobrevive* indica si *o* queda vivo despues de sus enfrentamientos; *o* pasa a contener el número de organismos que ha destruido

Definición en la línea 29 del archivo Sistema.cc.

```

30 {
31 // Busca el primer organismo de la cola c que consigue matar al organismo o y
32 // borra (y cuenta) todos los que resultan destruidos por éste. Para que el
33 // borrado sea efectivo, los supervivientes se van añadiendo al final de la
34 // cola. Al final, los que no hayan luchado se pasan al final la cola mediante
35 // una nueva operación privada auxiliar: "recolocar".
36
37 Organismo actual;
38 int resultado;
39
40 sobrevive = true;
41 int longCola = c.size();
42
43 // Inv: sobrevive = ninguno de los elementos visitados de C destruye a o;
44 //      c = elementos no visitados de C seguidos por los elementos visitados
45 //      con su contador de víctimas actualizado, excepto los que hayan muerto
46 //      al enfrentarse con o, en el mismo orden en que estaban en C;
47 //      longCola = número de elementos no visitados de C;
48 //      o tiene actualizado su contador de víctimas
49 //
50 // Cota: longCola
51
52 while (longCola>0 and sobrevive) {
53     actual = c.front();
54     resultado = o.lucha_organismos(actual);
55     switch (resultado) {
56         case 0: { // los dos mueren
57             sobrevive = false; // no actualizamos las víctimas de actual
58             o.incrementar_victimas(); // porque éste ya no pertenece al sistema
59             break;
60         }
61         case 1: { // o muere, actual sobrevive
62             sobrevive= false;
63             actual.incrementar_victimas();
64             c.push(actual);
65             break;
66         }
67         case 2: { // o sobrevive, actual muere
68             o.incrementar_victimas();
69             break;
70         }
71         case 3: { // los dos sobreviven
72             c.push(actual);
73             break;
74         }
75     }
76     c.pop();
77     --longCola;
78 }
79
80 // Post1: Inv1 y (longCola == 0 or not sobrevive)
81
82 // Falta pasar al final de c los elementos no visitados de C (por Inv1, son
83 // los longCola primeros de c), para mantener el orden por identificador
84
85 recolocar(longCola,c);
86 }

```

#### 4.3.3.5. void Sistema::clear ( queue< Organismo > & q ) [static], [private]

Elimina todos los elementos de una cola de organismos.

##### Precondición

c = C;

##### Postcondición

c es vacía

Definición en la línea 102 del archivo Sistema.cc.

```

103 {
104     while (not q.empty()) q.pop();
105 }

```

#### 4.3.3.6. void Sistema::recolocar ( int *n*, queue< Organismo > & *c* ) [static],[private]

Pasa los *n* primeros organismos de una cola al final de ésta, conservando el orden relativo.

##### Precondición

$n = N \leq c.size()$ ,  $c = C$ ;

##### Postcondición

*c* contiene los mismos elementos que *C*, pero los *N* primeros elementos de *C* están al final de *c*, en el orden relativo original; los restantes también conservan su orden relativo original

Definición en la línea 88 del archivo Sistema.cc.

```
89 {
90     // Inv:  n<=N, c contiene los mismos elementos que C, pero los N-n primeros
91     //        elementos de C están al final de c, en el orden relativo original;
92     //        los restantes también conservan su orden relativo original
93     // Cota: n
94
95     while (n>0) {
96         c.push(c.front());
97         c.pop();
98         --n;
99     }
100 }
```

#### 4.3.3.7. void Sistema::escribir\_sistemaCola ( const queue< Organismo > & *c*, bool *estr* ) [static],[private]

Operación de escritura de una cola de organismos.

##### Precondición

cierto

##### Postcondición

Se han escrito en el canal estándar de salida los organismos de *c*, por orden de identificador; si *estr* es cierto, cada organismo se escribe con su estructura celular, en caso contrario sólo se escribe su identificador

Definición en la línea 140 del archivo Sistema.cc.

```
141 {
142     queue<Organismo> aux(c);
143     while (not aux.empty()) {
144         aux.front().escribir(estr);
145         aux.pop();
146     }
147 }
```

### 4.3.4. Documentación de los datos miembro

#### 4.3.4.1. queue<Organismo> Sistema::def [private]

Cola de organismos defensivos.

Ordenada crecientemente por el id de sus componentes

Definición en la línea 40 del archivo Sistema.hh.

4.3.4.2. `queue<Organismo> Sistema::mal [private]`

Cola de organismos malignos.

Ordenada crecientemente por el id de sus componentes

Definición en la línea 45 del archivo Sistema.hh.

4.3.4.3. `int Sistema::id [private]`

Primer número disponible de identificador de organismo.

Definición en la línea 48 del archivo Sistema.hh.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [Sistema.hh](#)
- [Sistema.cc](#)



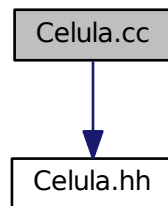
## Capítulo 5

# Documentación de archivos

### 5.1. Referencia del Archivo Celula.cc

Código de la clase [Celula](#).

Dependencia gráfica adjunta para Celula.cc:



#### 5.1.1. Descripción detallada

Código de la clase [Celula](#).

Definición en el archivo [Celula.cc](#).

### 5.2. Referencia del Archivo Celula.hh

Especificación de la clase [Celula](#).

#### Clases

- class [Celula](#)

*Representa el conjunto de características y operaciones de las células.*

### 5.2.1. Descripción detallada

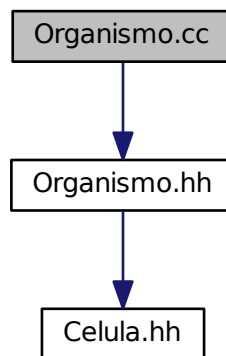
Especificación de la clase [Celula](#).

Definición en el archivo [Celula.hh](#).

## 5.3. Referencia del Archivo Organismo.cc

Código de la clase [Organismo](#).

Dependencia gráfica adjunta para Organismo.cc:



### 5.3.1. Descripción detallada

Código de la clase [Organismo](#).

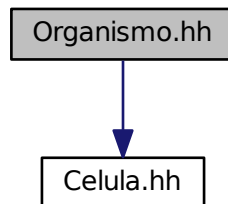
Definición en el archivo [Organismo.cc](#).

## 5.4. Referencia del Archivo Organismo.hh

Especificación de la clase [Organismo](#).



Dependencia gráfica adjunta para Organismo.hh:



## Clases

- class [Organismo](#)

*Representa la información y las operaciones asociadas a un organismo.*

### 5.4.1. Descripción detallada

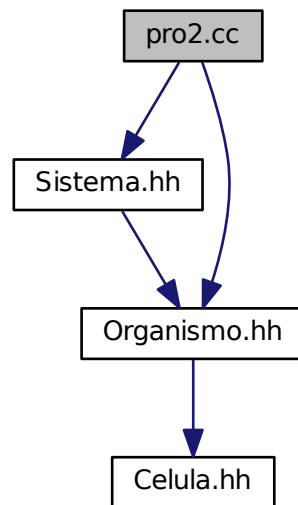
Especificación de la clase [Organismo](#).

Definición en el archivo [Organismo.hh](#).

## 5.5. Referencia del Archivo pro2.cc

Programa principal.

Dependencia gráfica adjunta para pro2.cc:



## Funciones

- `int main ()`

### 5.5.1. Descripción detallada

Programa principal.

Estamos suponiendo que los datos leídos siempre son correctos, ya que no incluimos comprobaciones al respecto. Por último, puesto que los datos de los organismos y células son naturales (identificadores, ...) usaremos números negativos para las opciones.

Definición en el archivo `pro2.cc`.

### 5.5.2. Documentación de las funciones

#### 5.5.2.1. `int main ( )`

Definición en la línea 45 del archivo `pro2.cc`.

```

46 {
47     int N; // Número de parámetros de las células
48     cin » N;
49     Sistema S;
50     S.leer(N);
51     int op; // Código de operación
52     cin » op;
53     while (op != -3) {
54         if (op == -1 ) {
55             Organismo O;
56             O.leer(N);
57             bool sobrevive;
58             S.anadir_organismo(O, sobrevive);
59             cout « "Entrada del nuevo organismo" « endl;
60             cout « O.num_victimas() « " " « sobrevive « endl;

```

```

61     }
62     else if (op == -2) {
63         bool tipo = readbool();
64         bool estr = readbool();
65         if (tipo){
66             if (estr) cout << "Defensivos del sistema con estructura" << endl;
67             else cout << "Defensivos del sistema sin estructura" << endl;
68         }
69         else {
70             if (estr) cout << "Malignos del sistema con estructura" << endl;
71             else cout << "Malignos del sistema sin estructura" << endl;
72         }
73         S.escribir(tipo, estr);
74     }
75     cin >> op;
76 }
77 }

```

## 5.6. Referencia del Archivo readbool.hh

operacion para leer booleanos del canal estandar

### Funciones

- bool `readbool()`

*Lee un booleano por el canal estandar.*

#### 5.6.1. Descripción detallada

operacion para leer booleanos del canal estandar

Definición en el archivo [readbool.hh](#).

#### 5.6.2. Documentación de las funciones

##### 5.6.2.1. bool readbool ( )

Lee un booleano por el canal estandar.

##### Precondición

La primera string valida del canal estandar es "true" o "false"

##### Postcondición

El resultado es cierto si se ha leído "true" y falso si no

Definición en la línea 17 del archivo readbool.hh.

```

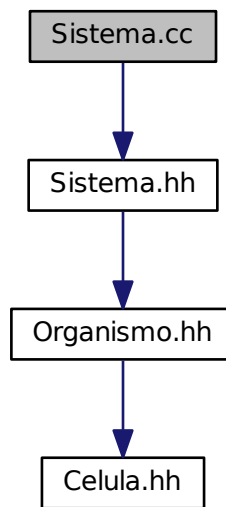
18 {
19     string n;
20     cin >> n;
21     if (n!="true" and n!="false") throw PRO2Excepcio("S'havia de llegir un boolea");
22     return (n=="true");
23 }

```

## 5.7. Referencia del Archivo Sistema.cc

Código de la clase [Sistema](#).

Dependencia gráfica adjunta para Sistema.cc:



#### 5.7.1. Descripción detallada

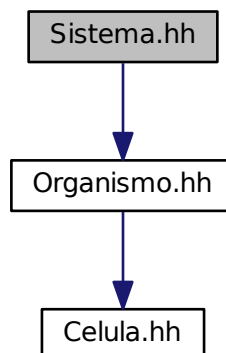
Código de la clase [Sistema](#).

Definición en el archivo [Sistema.cc](#).

### 5.8. Referencia del Archivo Sistema.hh

Especificación de la clase [Sistema](#).

Dependencia gráfica adjunta para Sistema.hh:



## Clases

- class [Sistema](#)

*Representa el sistema donde se desarrollan los experimentos.*

### 5.8.1. Descripción detallada

Especificación de la clase [Sistema](#).

Definición en el archivo [Sistema.hh](#).

# Índice alfabético

Celula, [7](#)  
    Celula, [8](#)  
    escribir, [9](#)  
    id, [10](#)  
    leer, [9](#)  
    param, [10](#)  
celulas  
    Organismo, [17](#)  
clear  
    Sistema, [21](#)  
  
def  
    Sistema, [22](#)  
  
escribir  
    Celula, [9](#)  
    Organismo, [14](#)  
    Sistema, [20](#)  
  
id  
    Celula, [10](#)  
    Organismo, [17](#)  
    Sistema, [23](#)  
  
leer  
    Celula, [9](#)  
    Organismo, [14](#)  
    Sistema, [19](#)  
  
mal  
    Sistema, [22](#)  
maligno  
    Organismo, [17](#)  
  
Organismo, [10](#)  
    celulas, [17](#)  
    escribir, [14](#)  
    id, [17](#)  
    leer, [14](#)  
    maligno, [17](#)  
    Organismo, [12](#)  
    simetricos, [15](#)  
    victimas, [17](#)  
  
param  
    Celula, [10](#)  
  
recolocar  
    Sistema, [21](#)  
  
simetricos  
  
Organismo, [15](#)  
Sistema, [17](#)  
    clear, [21](#)  
    def, [22](#)  
    escribir, [20](#)  
    id, [23](#)  
    leer, [19](#)  
    mal, [22](#)  
    recolocar, [21](#)  
    Sistema, [18](#)  
  
victimas  
    Organismo, [17](#)