Laboratorio de PRO2. Caso de estudio: experimentos inmunologicos.

v7.0 13-11-2017

Generado por Doxygen 1.8.11

Índice general

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

Indice jerárquico

2.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

Capítulo 3

Índice de clases

3.1. Lista de clases

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

Capítulo 4

Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

Capítulo 5

Documentación de las clases

5.1. Referencia de la plantilla de la Clase BinTree< T >

Clases

- struct Node

Métodos públicos

- BinTree ()
- BinTree (const T &x)
- BinTree (const T &x, const BinTree &left, const BinTree &right)
- bool empty () const
- BinTree left () const
- BinTree right () const
- const T & value () const

Métodos privados

- BinTree (shared_ptr< Node > p)

Atributos privados

- shared_ptr< Node > p

5.1.1. Descripción detallada

template<typename T>
class BinTree< T >

Definición en la línea 9 del archivo BinTree.hh.

5.1.2. Documentación del constructor y destructor

5.1.2.1. template<typename T> BinTree< T >::BinTree (shared_ptr< Node > *p*) [private]

Definición en la línea 26 del archivo BinTree.hh.

```
27     :   p(p)
28     {   }
```

5.1.2.2. template<typename T> BinTree< T >::BinTree ( )

Definición en la línea 38 del archivo BinTree.hh.

```
39     :   p(nullptr)
40     {   }
```

5.1.2.3. template<typename T> BinTree< T >::BinTree (const T & *x*)

Definición en la línea 43 del archivo BinTree.hh.

```
43                          {
44         p = make_shared<Node>(x, nullptr, nullptr);
45     }
```

5.1.2.4. template<typename T> BinTree< T >::BinTree (const T & *x*, const BinTree< T > & *left*, const BinTree< T > & *right*)

Definición en la línea 48 del archivo BinTree.hh.

```
48                                                                      {
49         p = make_shared<Node>(x, left.p, right.p);
50     }
```

5.1.3. Documentación de las funciones miembro

5.1.3.1. template<typename T> bool BinTree< T >::empty ( ) const

Definición en la línea 53 del archivo BinTree.hh.

```
53                          {
54         return not p;
55     }
```

5.1.3.2. template<typename T> BinTree BinTree< T >::left ( ) const

Definición en la línea 58 del archivo BinTree.hh.

```
58                              {
59          assert(not empty());
60          return BinTree(p->left);
61      }
```

5.1.3.3. template<typename T> BinTree BinTree< T >::right ( ) const

Definición en la línea 64 del archivo BinTree.hh.

```
64                               {
65          assert(not empty());
66          return BinTree(p->right);
67      }
```

5.1.3.4. template<typename T> const T& BinTree< T >::value ( ) const

Definición en la línea 70 del archivo BinTree.hh.

```
70                                {
71          assert(not empty());
72          return p->x;
73      }
```

5.1.4. Documentación de los datos miembro

5.1.4.1. template<typename T> shared_ptr<Node> BinTree< T >::p [private]

Definición en la línea 23 del archivo BinTree.hh.

La documentación para esta clase fue generada a partir del siguiente fichero:

- BinTree.hh

5.2. 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.
- int num_param ()

  Consultora del número de parámetros de una célula.
- void leer (int N)

  Operación de lectura.
- void escribir () const

  Operación de escritura.

Métodos públicos estáticos

- static int id_vacia ()

  Consultora del identificador especial de células vacías.

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 vacias.

5.2.1. 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.

Dado que vamos a necesitar leer árboles de células, definimos el concepto de célula vacía para disponer de un formato de entrada parecido al de las anteriores sesiones de laboratorio, en las que se emplea una "marca" para indicar la lectura de un árbol vacío.

Definición en la línea 30 del archivo Celula.hh.

5.2.2. Documentación del constructor y destructor

5.2.2.1. Celula::Celula ( )

Creadora por defecto.

Precondición

cierto

Postcondición

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

Coste

Constante

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 }
```

5.2.3. Documentación de las funciones miembro

5.2.3.1. 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

Coste

Lineal respecto al número de parámetros de una célula

Definición en la línea 14 del archivo Celula.cc.

```
15 {
16
17   if (param.size()!=c2.param.size()) throw PRO2Excepcio("Las dos células han de tener
       el mismo número de parámetros");
18
19   // Se trata de obtener la diferencia entre el número de posiciones de la primera
20   // célula que superan a las de la segunda y viceversa. Después se compara dicha
21   // diferencia con los indicadores de tolerancia.
22   int i = 0;
23   int dif = 0;
24
25   // Inv: dif = diferencia entre el número de posiciones en [0..i-1] en que c1 supera a c2
26   // y viceversa, 0<=i<=param.size()
27
28   while (i < param.size()) {
29     if (param[i] > c2.param[i]) ++dif;
30     else if (param[i] < c2.param[i]) --dif;
31     ++i;
32   }
33   // Post1: dif = diferencia entre el número de posiciones totales en que
34   // c1 supera a c2 y viceversa
35   int n = 3;
36   if (dif > c2.i_tol) n = 1;
37   else if (dif < -i_tol) n = 2;
38   return n;
39 }
```

5.2.3.2. 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

Coste

Constante

Definición en la línea 41 del archivo Celula.cc.

```
42 {
43   return id == ID_VACIA;
44 }
```

5.2.3.3. int Celula::num_param ( )

Consultora del número de parámetros de una célula.

Precondición

cierto

Postcondición

El resultado es el número de parámetros de una célula

Coste

Constante

Definición en la línea 46 del archivo Celula.cc.

```
46                      {
47   return param.size();
48 }
```

5.2.3.4. int Celula::id_vacia ( ) [static]

Consultora del identificador especial de células vacías.

Precondición

cierto

Postcondición

El resultado es el identificador de célula vacía

Coste

Constante

Definición en la línea 50 del archivo Celula.cc.

```
51 {
52   return ID_VACIA;
53 }
```

5.2.3.5. void Celula::leer (int *N*)

Operación de lectura.

Precondición

N>0, el canal de entrada estándar contiene un entero; si dicho entero no corresponde a un identificador de célula vacía, a continuación contiene otro entero y N doubles

Postcondición

El parámetro implícito pasa a tener un identificador (el primer entero del canal de entrada estándar); si éste no es el de una célula vacía, el p.i. también tendrá un índice de tolerancia y N parámetros nuevos, leídos del canal de entrada estándar

Coste

Lineal respecto a N (número de parámetros de una célula)

Definición en la línea 55 del archivo Celula.cc.

```
56 {
57   // Simplemente lee todos los componentes de la célula teniendo en cuenta que si
58   // el identificador es cero se trata de una celula "marca" y no es necesario continuar leyendo.
59
60   if (N<=0) throw PRO2Excepcio("La celula ha de tener N parametros (N>0)");
61
62   cin » id;
63   if (id != ID_VACIA) {
64     cin » i_tol;
65     param = vector<double> (N);
66     for (int i = 0; i < N; ++i) {
67       cin » param[i];
68     }
69   }
70 }
```

5.2.3.6. void Celula::escribir ( ) const

Operación de escritura.

Precondición

cierto

Postcondición

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

Coste

Lineal respecto al número de parámetros de una célula

Definición en la línea 72 del archivo Celula.cc.

```
73 {
74   // Análogamente, se trata sólo de escribir los componentes que nos interesan,
75   // en este caso, según el enunciado, el identificador de la célula.
76   cout « id;
77 }
```

5.2.4. Documentación de los datos miembro

5.2.4.1. const int Celula::ID_VACIA = 0 [static], [private]

Identificador especial para células vacias.

Definición en la línea 35 del archivo Celula.hh.

5.2.4.2. int Celula::id [private]

Identificador de la célula.

Definición en la línea 37 del archivo Celula.hh.

5.2.4.3. int Celula::i_tol [private]

Indice de tolerancia.

Definición en la línea 39 del archivo Celula.hh.

5.2.4.4. vector<double> Celula::param [private]

Parámetros de la célula.

Definición en la línea 41 del archivo Celula.hh.

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

- Celula.hh
- Celula.cc

5.3. Referencia de la Estructura BinTree< T >::Node

Métodos públicos

- Node (const T &x, shared_ptr< Node > left, shared_ptr< Node > right)

Atributos públicos

- T x
- shared_ptr< Node > left
- shared_ptr< Node > right

5.3.1. Descripción detallada

template<typename T>
struct BinTree< T >::Node

Definición en la línea 11 del archivo BinTree.hh.

5.3.2. Documentación del constructor y destructor

5.3.2.1. template<typename T> BinTree< T >::Node::Node (const T & *x*, shared_ptr< Node > *left*, shared_ptr< Node > *right*)

Definición en la línea 16 del archivo BinTree.hh.

```
17         :   x(x), left(left), right(right)
18         {   }
```

5.3.3. Documentación de los datos miembro

5.3.3.1. template<typename T> T BinTree< T >::Node::x

Definición en la línea 12 del archivo BinTree.hh.

5.3.3.2. template<typename T> shared_ptr<Node> BinTree< T >::Node::left

Definición en la línea 13 del archivo BinTree.hh.

5.3.3.3. template<typename T> shared_ptr<Node> BinTree< T >::Node::right

Definición en la línea 14 del archivo BinTree.hh.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- BinTree.hh

5.4. 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 (const BinTree< Celula > &a1, const BinTree< Celula > &a2)

  Comprobación de simetría de dos árboles.
- static pair< int, int > lucha_arboles (const BinTree< Celula > &a1, const BinTree< Celula > &a2)

  Lucha de dos árboles de células.
- static void leer_arbol_celulas (int N, BinTree< Celula > &a)

  Operación de lectura de un árbol de células.
- static void escribir_arbol_celulas_id (const BinTree< Celula > &a)

  Operación de escritura de un árbol de células.

Atributos privados

- BinTree< 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.

5.4.1. 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.

5.4.2. Documentación del constructor y destructor

5.4.2.1. Organismo::Organismo ( )

Creadora por defecto.

Precondición

cierto

Postcondición

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

Coste

Constante

Definición en la línea 11 del archivo Organismo.cc.

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

5.4.3. Documentación de las funciones miembro

5.4.3.1. void Organismo::anadir_id (int *id*)

Modificadora del identificador.

Precondición

cierto

Postcondición

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

Coste

Constante

Definición en la línea 23 del archivo Organismo.cc.

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

5.4.3.2. void Organismo::incrementar_victimas ( )

Modificadora del número de víctimas.

Precondición

cierto

Postcondición

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

Coste

Constante

Definición en la línea 18 del archivo Organismo.cc.

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

5.4.3.3. 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

Coste

Lineal respecto al mínimo del número de células del p.i. y o2 (el coste de tratar una célula es lineal respecto a su número de parámetros)

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.  Asímismo,
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   if (simetricos(celulas, o2.celulas)) {
44     pair<int, int> m = lucha_arboles(celulas, o2.celulas);
45     if (m.first == m.second) n = 0;
46     else if (m.first < m.second) n = 1;
47     else n = 2; // m.first > m.second
48   }
49   else n = 3;
50   return n;
51 }
```

5.4.3.4. 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

Coste

Constante

Definición en la línea 97 del archivo Organismo.cc.

```
98 {
99   // Devuelve el valor del campo "maligno" del organismo correspondiente.
100   return maligno;
101 }
```

5.4.3.5. 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

Coste

Constante

Definición en la línea 103 del archivo Organismo.cc.

```
104 {
105   // Devuelve el valor del campo "victimas" del organismo correspondiente.
106   return victimas;
107 }
```

5.4.3.6. void Organismo::leer (int *N*)

Operación de lectura.

Precondición

N>0; el canal estándar de entrada contiene un organismo formado por células de N parámetros

Postcondición

El parámetro implícito es el organismo tomado del canal de entrada estándar

Coste

Lineal respecto al número de células del organismo leído (ver comentario en el coste de la operación "lucha↩ _organismos")

Definición en la línea 109 del archivo Organismo.cc.

```
110 {
111   // Esta operación simplemente lee la estructura celular del organismo e
112   // inicializa el recuento de víctimas. Un árbol de células se lee igual que
113   // un árbol de enteros. Suponemos que contamos con una marca de tipo Celula
114   // para indicar que llegamos a un árbol vacío.
115
116   if (N<=0) throw PRO2Excepcio("Las celulas del organismo han de tener N parametros (N>0)");
117
118   leer_arbol_celulas(N, celulas);
119   maligno = readbool();
120   victimas = 0;
121 }
```

5.4.3.7. 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 estándard; si estr es cierto también se ha escrito su estructura celular

Coste

Lineal respecto al número de células del organismo escrito (si "estr" es cierto, ver comentario en el coste de la operación "lucha_organismos")

Definición en la línea 135 del archivo Organismo.cc.

```
136 {
137   // Escribimos los campos del organismo.  Un árbol de células se escribe
138   // igual que un árbol de enteros, suponiendo que (como es el caso) exista
139   // una operación para escribir células.
140   cout « id « " " « victimas;
141   if (estr) {
142     escribir_arbol_celulas_id(celulas);
143   }
144   cout « endl;
145 }
```

5.4.3.8. bool Organismo::simetricos (const BinTree< Celula > & *a1*, const BinTree< 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

Coste

Lineal respecto al mínimo del número de células de A1 y A2 (ver comentario en el coste de la operación "lucha_organismos")

Definición en la línea 53 del archivo Organismo.cc.

```
54 {
55   bool b;
56   if (a1.empty() or a2.empty()) b = a1.empty() and a2.empty();
57   else {
58     b=simetricos(a1.left(),a2.right());
59     // HI1: b indica si hi(A1) y hd(A2) son simétricos
60     if (b) { b=simetricos(a2.left(),a1.right());
61              // HI2: b indica si hd(A1) y hi(A2) son simétricos
62     }
63   }
64   return b;
65 }
```

5.4.3.9. pair< int, int > Organismo::lucha_arboles (const BinTree< Celula > & *a1*, const BinTree< 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

Coste

Lineal respecto al número de células de A1 (ver comentario en el coste de la operación "lucha_organismos")

Definición en la línea 67 del archivo Organismo.cc.

```
68 {
69   pair<int,int> n;
70
71   if (a1.empty()) {
72     n.first = 0;
73     n.second = 0;
74   }
75   else {
76     int nraiz = a1.value().lucha_celulas(a2.value());
77     n.first = 0;
78     if (nraiz == 1) ++n.first;
79     n.second = 0;
80     if (nraiz == 2) ++n.second;
81     pair<int,int> na = lucha_arboles(a1.left(),a2.right());//fe1, fd2);
82     //       HI1: na.first = número de células de hi(A1) que vencen a su
83     //                        correspondiente en hd(A2);
84     //            na.second = número de células de hd(A2) que vencen a su
85     //                        correspondiente en hi(A1)
86     pair<int,int> nb = lucha_arboles(a2.left(),a1.right());//fd1, fe2);
87     //       HI2: nb.first = número de células de hd(A1) que vencen a su
88     //                       correspondiente en hi(A2);
89     //            nb.second = número de células de hi(A2) que vencen a su
90     //                        correspondiente en hd(A1)
91     n.first += na.first + nb.first;
92     n.second += nb.second + na.second;
93   }
94   return n;
95 }
```

5.4.3.10. void Organismo::leer_arbol_celulas (int *N*, BinTree< 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

Coste

Lineal respecto al número de células del árbol leído (ver comentario en el coste de la operación "lucha_↩ organismos")

Definición en la línea 123 del archivo Organismo.cc.

```
124 {
125   BinTree<Celula> a1, a2;
126   Celula c;
127   c.leer(N);
128   if (not c.es_vacia()) {
129     leer_arbol_celulas(N, a1);
130     leer_arbol_celulas(N, a2);
131     a=BinTree<Celula>(c, a1, a2);
132   }
133 }
```

5.4.3.11. void Organismo::escribir_arbol_celulas_id (const BinTree< Celula > & *a*) [static], [private]

Operación de escritura de un árbol de células.

Precondición

cierto

Postcondición

Se ha escrito a por el canal de salida estándard

Coste

Lineal respecto al número de células del árbol escrito (ver comentario en el coste de la operación "lucha_↩ organismos")

Definición en la línea 147 del archivo Organismo.cc.

```
148 {
149   if (not a.empty()) {
150     //Celula aux = a.value();
151     escribir_arbol_celulas_id(a.left());
152     cout « " ";
153     a.value().escribir();
154     escribir_arbol_celulas_id(a.right());
155   }
156 }
```

5.4.4. Documentación de los datos miembro

5.4.4.1. BinTree<Celula> Organismo::celulas [private]

Estructura celular del organismo.

Definición en la línea 37 del archivo Organismo.hh.

5.4.4.2. int Organismo::id [private]

Identificador del organismo.

Definición en la línea 39 del archivo Organismo.hh.

5.4.4.3. bool Organismo::maligno [private]

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

Definición en la línea 41 del archivo Organismo.hh.

5.4.4.4. 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

5.5. Referencia de la Clase PRO2Excepcio

Métodos públicos

- PRO2Excepcio (const char *mot)
- const char * what () const throw ()

Atributos privados

- const char * mensaje

5.5.1. Descripción detallada

Definición en la línea 9 del archivo PRO2Excepcio.hh.

5.5.2. Documentación del constructor y destructor

5.5.2.1. PRO2Excepcio::PRO2Excepcio (const char * *mot*)

Definición en la línea 11 del archivo PRO2Excepcio.hh.

```
11 : exception(), mensaje(mot) {}
```

5.5.3. Documentación de las funciones miembro

5.5.3.1. const char* PRO2Excepcio::what ( ) const throw)

Definición en la línea 12 del archivo PRO2Excepcio.hh.

```
12 {return mensaje;};
```

5.5.4. Documentación de los datos miembro

5.5.4.1. const char* PRO2Excepcio::mensaje [private]

Definición en la línea 12 del archivo PRO2Excepcio.hh.

La documentación para esta clase fue generada a partir del siguiente fichero:

- PRO2Excepcio.hh

5.6. 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_org_cola (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_sistema_cola (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.

5.6.1. 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.

5.6.2. Documentación del constructor y destructor

5.6.2.1. Sistema::Sistema ( )

Creadora por defecto.

Precondición

cierto

Postcondición

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

Coste

Constante

Definición en la línea 7 del archivo Sistema.cc.

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

5.6.3. Documentación de las funciones miembro

5.6.3.1. 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

Coste

Lineal respecto al número de organismos del sistema (el coste de tratar cada organismo es lineal respecto a su número de células y el coste de tratar cada célula es lineal respecto a su número de parámetros)

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_org_cola" 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_org_cola(def, o, sobrevive);
21       if (sobrevive) mal.push(o);
22    }
23    else {
24       luchas_org_cola(mal, o, sobrevive);
25       if (sobrevive) def.push(o);
26    }
27 }
```

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

Operación de lectura.

Precondición

N>0, el canal estándard de entrada contiene un entero M > 0 seguido de los datos de M organismos

Postcondición

El parámetro implícito está formado por los M organismos procedentes del canal estándard de entrada y formados por celulas de N parámetros

Coste

Lineal respecto al número de organismos leídos (ver comentario en el coste de la operación "anadir_↩ organismo")

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 posiblidad 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 }
```

5.6.3.3. 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 estándard, 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

Coste

Lineal respecto al número de organismos escritos (si "estr" es cierto, ver comentario en el coste de la operación "anadir_organismo")

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 }
```

5.6.3.4. 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

Coste

Lineal respecto al número de organismos en C (ver comentario en el coste de la operación "anadir_organismo")

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 long_cola = 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 //      long_cola = número de elementos no visitados de C;
48 //      o tiene actualizado su contador de víctimas
49 //
50 // Cota: long_cola
51
52    while (long_cola>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       --long_cola;
78    }
79
80 // Post1: Inv1 y (long_cola == 0 or not sobrevive)
81
82 // Falta pasar al final de c los elementos no visitados de C (por Inv1, son
83 // los long_cola primeros de c), para mantener el orden por identificador
84
85    recolocar(long_cola,c);
86 }
```

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

Elimina todos los elementos de una cola de organismos.

Precondición

c = C;

Postcondición

c es vacía

Coste

Lineal respecto al número de organismos en C (ver comentario en el coste de la operación "anadir_organismo")

Definición en la línea 102 del archivo Sistema.cc.

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

5.6.3.6. 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 <= 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

Coste

Lineal respecto a n (ver comentario en el coste de la operación "anadir_organismo")

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 }
```

5.6.3.7. void Sistema::escribir_sistema_cola (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

Coste

Lineal respecto al número de organismos escritos (si "estr" es cierto, ver comentario en el coste de la operación "anadir_organismo")

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 }
```

5.6.4. Documentación de los datos miembro

5.6.4.1. 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.

5.6.4.2. 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.

5.6.4.3. 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:

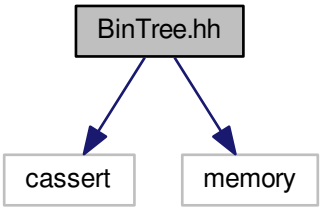
- Sistema.hh
- Sistema.cc

Capítulo 6

Documentación de archivos

6.1. Referencia del Archivo BinTree.hh

Dependencia gráfica adjunta para BinTree.hh:



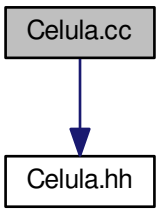
Clases

- class BinTree< T >
- struct BinTree< T >::Node

6.2. Referencia del Archivo Celula.cc

Código de la clase Celula.

Dependencia gráfica adjunta para Celula.cc:



6.2.1. Descripción detallada

Código de la clase Celula.

6.3. Referencia del Archivo Celula.hh

Especificación de la clase Celula.

Clases

- class Celula

  Representa el conjunto de características y operaciones de las células.

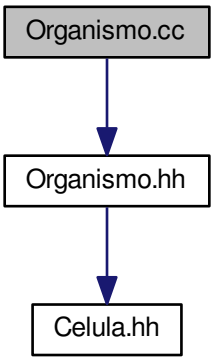
6.3.1. Descripción detallada

Especificación de la clase Celula.

6.4. Referencia del Archivo Organismo.cc

Código de la clase Organismo.

Dependencia gráfica adjunta para Organismo.cc:



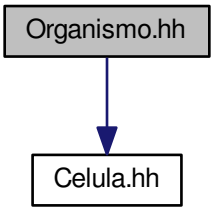
6.4.1. Descripción detallada

Código de la clase Organismo.

6.5. 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.

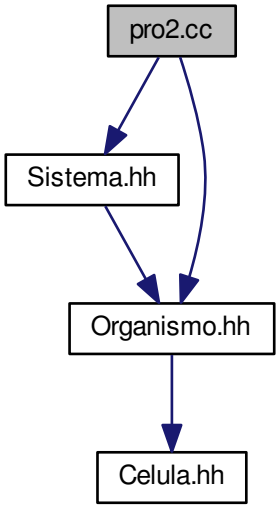
6.5.1. Descripción detallada

Especificación de la clase Organismo.

6.6. Referencia del Archivo pro2.cc

Programa principal.

Dependencia gráfica adjunta para pro2.cc:



Funciones

- int main ()

6.6.1. Descripción detallada

Programa principal.

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

6.6.2. Documentación de las funciones

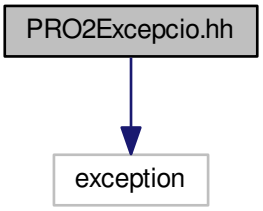
6.6.2.1. int main ( )

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 }
```

6.7. Referencia del Archivo PRO2Excepcio.hh

Dependencia gráfica adjunta para PRO2Excepcio.hh:



Clases

- class PRO2Excepcio

6.8. Referencia del Archivo readbool.hh

operacion para leer booleanos del canal estandar

Funciones

- bool readbool ()

  Lee un booleano por el canal estandar.

6.8.1. Descripción detallada

operacion para leer booleanos del canal estandar

6.8.2. Documentación de las funciones

6.8.2.1. 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 leido "true" y falso si no

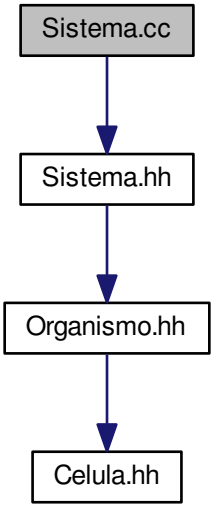
Definición en la línea 17 del archivo readbool.hh.

```
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 }
```

6.9. Referencia del Archivo Sistema.cc

Código de la clase Sistema.

Dependencia gráfica adjunta para Sistema.cc:



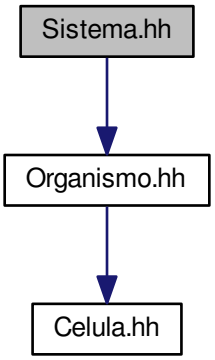
6.9.1. Descripción detallada

Código de la clase Sistema.

6.10. 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.

6.10.1. Descripción detallada

Especificación de la clase Sistema.

Índice alfabético