Caso de estudio: experimentos inmunologicos. Laboratorio de PRO2.

v4.1 20-11-2014

Generado por Doxygen 1.8.2

Lunes, 17 de Noviembre de 2014 15:17:24

Índice general

Capítulo 1

Página principal

El programa principal se encuentra en el módulo pro2.cpp. 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 de forma explícita, ya que el segundo módulo no se usa en la especificación del primero. Sin embargo, podemos prever que nos la encontraremos en la sección de implementación.

Capítulo 2

Índice de clases

2.1. Lista de clases

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

Capítulo 3

Indice de archivos

3.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

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.
- ∼Celula ()

  Destructora 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 vacias.

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 29 del archivo Celula.hpp.

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.cpp.

```
{
   // Inicializa una célula con el id de célula vacía
   id = ID_VACIA;
   i_tol = 0;
}
```

4.1.2.2. Celula::~Celula ( )

Destructora por defecto.

Definición en la línea 14 del archivo Celula.cpp.

```
{}
```

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 16 del archivo Celula.cpp.

```
{
// Se trata de obtener la diferencia entre el número de posiciones de la
      primera
// célula que superan a las de la segunda y viceversa. Después se compara dicha
// diferencia con los indicadores de tolerancia.
   int i = 0;
   int dif = 0;

// Inv: dif = diferencia entre el número de posiciones en [0..i-1] en que c1
      supera a c2
// y viceversa, 0<=i<=param.size()

   while (i < param.size()) {
     if (param[i] > c2.param[i]) ++dif;
     else if (param[i] < c2.param[i]) --dif;
     ++i;
   }
// Post1: dif = diferencia entre el número de posiciones totales en que
// c1 supera a c2 y viceversa
   int n = 3;
   if (dif > c2.i_tol) n = 1;
   else if (dif < -i_tol) n = 2;
   return n;
}
```

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 40 del archivo Celula.cpp.

```
{
   return id == ID_VACIA;
}
```

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 estándard, salvo si el identificador leído es el de una célula vacía

Definición en la línea 45 del archivo Celula.cpp.

```
{
// Simplemente lee todos los componentes de la célula teniendo en cuenta que si

// el identificador es cero se trata de una celula "marca" y no es necesario
       continuar leyendo.
   id = readint();
   if (id != ID_VACIA) {
      i_tol = readint();
      param = vector<double> (N);
      for (int i = 0; i < N; ++i) {
          param[i] = readdouble();
      }
   }
}
```

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 estándard

Definición en la línea 59 del archivo Celula.cpp.

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

4.1.4. Documentación de los datos miembro

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

Identificador especial para células vacias.

Definición en la línea 34 del archivo Celula.hpp.

4.1.4.2. int Celula::id [private]

Identificador de la célula.

Definición en la línea 36 del archivo Celula.hpp.

4.1.4.3. int Celula::i_tol [private]

Indice de tolerancia.

Definición en la línea 38 del archivo Celula.hpp.

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

Parámetros de la célula.

Definición en la línea 40 del archivo Celula.hpp.

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

- Celula.hpp
- Celula.cpp

4.2. Referencia de la Clase Organismo

Representa la información y las operaciones asociadas a un organismo.

Métodos públicos

- Organismo ()

  Creadora por defecto.
- ~Organismo ()

  Destructora 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 22 del archivo Organismo.hpp.

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 victimas

Definición en la línea 11 del archivo Organismo.cpp.

```
{
  id=0;
  maligno=false;
  victimas=0;
}
```

4.2.2.2. Organismo::∼Organismo ( )

Destructora por defecto.

Definición en la línea 18 del archivo Organismo.cpp.

```
{}
```

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 25 del archivo Organismo.cpp.

```
{
  this->id = id;
}
```

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 20 del archivo Organismo.cpp.

```
{
  ++victimas;
}
```

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 30 del archivo Organismo.cpp.

```
{
  // Ésta es la operación más importante del módulo. Dados dos organismos, hay
  // que decidir primero si van a luchar de verdad o no, es decir, si sus
  // estructuras celulares son simétricas o no. Por eso, introducimos la
  // operación auxiliar "simetricos" en la parte privada.  Notad que en la
  // cabecera de la operación "simetricos" decimos que los parámetros han de
  // ser árboles de Celula, pero la operación es independiente del tipo de los
  // elementos del árbol, ya que éstos nunca se llegan a consultar.  Asímismo,
  // debemos disponer de otra operación que aplique las luchas de las células
  // de dos árboles, sabiendo que son simétricos. Aquí sí es relevante el
  // hecho de que los árboles sean de células. Esta nueva operación privada se
  // llama "lucha_arboles".

  int n;
  Arbre<Celula> aux1(celulas);       // podemos prescindir de aux1 y
      aux2
  Arbre<Celula> aux2(o2.celulas);    // si simetricos y lucha_arboles
  if (simetricos(aux1, aux2)) {      // replantan sus parámetros para
      no
    aux1 = celulas;                  // destruirlos, pero entonces
    aux2 = o2.celulas;               // lucha_organismos no puede ser
      const
    pair<int, int> m = lucha_arboles(aux1, aux2);
    if (m.first == m.second) n = 0;
    else if (m.first < m.second) n = 1;
    else n = 2; // m.first > m.second
  }
  else n = 3;
  return n;
}
```

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 109 del archivo Organismo.cpp.

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

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 115 del archivo Organismo.cpp.

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

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 estándard formado por células de N parámetros

Definición en la línea 121 del archivo Organismo.cpp.

```
{
  // Esta operación simplemente lee la estructura celular del organismo e
  // inicializa el recuento de víctimas. Un árbol de células se lee igual que
  // un árbol de enteros. Suponemos que contamos con una marca de tipo Celula
  // para indicar que llegamos a un árbol vacío.

  celulas.a_buit();
  leer_arbol_celulas(N, celulas);
  maligno = readbool();
  victimas = 0;
}
```

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

Definición en la línea 146 del archivo Organismo.cpp.

```
{
  // Escribimos los campos del organismo.  Un árbol de células se escribe
  // igual que un árbol de enteros, suponiendo que (como es el caso) exista
  // una operación para escribir células.
  cout « id « " " « victimas;
  if (estr) {
    Arbre<Celula> copia(celulas);
    escribir_arbol_celulas_id(copia);
  }
  cout « endl;
}
```

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 59 del archivo Organismo.cpp.

```
{
  bool b;
  if (a1.es_buit() or a2.es_buit()) b = a1.es_buit() and a2.es_buit();
  else {
    Arbre<Celula> fe1, fe2, fd1, fd2;
    a1.fills(fe1,fd1); // fe1 = hi(A1); fd1 = hd(A2); a1 = buit
    a2.fills(fe2,fd2); // fe2 = hi(A2); fd1 = hd(A2); a2 = buit
    b = simetricos(fe1, fd2);
    // HI1: b indica si hi(A1) y hd(A2) son simétricos
    if (b) { b = simetricos(fd1, fe2);
             // HI2: b indica si hd(A1) y hi(A2) son simétricos
    }
  }
  return b;
}
```

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 76 del archivo Organismo.cpp.

```
{
  pair<int,int> n;

  if (a1.es_buit()) {
    n.first = 0;
    n.second = 0;
  }
  else {
    int nraiz = a1.arrel().lucha_celulas(a2.arrel());
    n.first = 0;
    if (nraiz == 1) ++n.first;
    n.second = 0;
    if (nraiz == 2) ++n.second;
    Arbre<Celula> fe1, fe2, fd1, fd2;
    a1.fills(fe1,fd1); // fe1 = hi(A1); fd1 = hd(A2); a1 = buit
    a2.fills(fe2,fd2); // fe2 = hi(A2); fd1 = hd(A2); a2 = buit
    pair<int,int> na = lucha_arboles(fe1, fd2);
    //       HI1: na.first = número de células de hi(A1) que vencen a su
    //                       correspondiente en hd(A2);
    //            na.second = número de células de hd(A2) que vencen a su
    //                        correspondiente en hi(A1)
    pair<int,int> nb = lucha_arboles(fd1, fe2);
    //       HI2: nb.first = número de células de hd(A1) que vencen a su
    //                       correspondiente en hi(A2);
    //            nb.second = número de células de hi(A2) que vencen a su
    //                        correspondiente en hd(A1)
    n.first += na.first + nb.first;
    n.second += nb.second + na.second;
  }
  return n;
}
```

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.cpp.

```
{
  Arbre<Celula> a1, a2;
  Celula c;
  c.leer(N);
  if (not c.es_vacia()) {
    leer_arbol_celulas(N, a1);
    leer_arbol_celulas(N, a2);
    a.plantar(c, a1, a2);
  }
}
```

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 estándard

Definición en la línea 159 del archivo Organismo.cpp.

```
{
  if (not a.es_buit()) {
    Arbre<Celula> a1, a2;
    Celula aux = a.arrel();
    a.fills(a1,a2);
    escribir_arbol_celulas_id(a1);
    cout « " ";
    aux.escribir();
    escribir_arbol_celulas_id(a2);
  }
}
```

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 33 del archivo Organismo.hpp.

4.2.4.2. int Organismo::id [private]

Identificador del organismo.

Definición en la línea 35 del archivo Organismo.hpp.

4.2.4.3. bool Organismo::maligno [private]

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

Definición en la línea 37 del archivo Organismo.hpp.

4.2.4.4. int Organismo::victimas [private]

Número de víctimas del organismo.

Definición en la línea 39 del archivo Organismo.hpp.

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

- Organismo.hpp
- Organismo.cpp

4.3. Referencia de la Clase Sistema

Representa el sistema donde se desarrollan los experimentos.

Métodos públicos

- Sistema ()

  Creadora por defecto.
- ∼Sistema ()

  Destructora 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.

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 21 del archivo Sistema.hpp.

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.cpp.

```
{
  id = 1;
}
```

4.3.2.2. Sistema::∼Sistema ( )

Destructora por defecto.

Definición en la línea 12 del archivo Sistema.cpp.

```
{}
```

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 14 del archivo Sistema.cpp.

```
{
// Proporciona su identificador al nuevo organismo y organiza las luchas de
// éste. Para ello, emplea la operación privada "luchas_org_cola" que lo
      enfrenta
// a los de la cola correspondiente y obtiene la información necesaria.
   o.anadir_id(id);
   ++id;
   if (o.es_maligno()) {
      luchas_org_cola(def, o, sobrevive);
      if (sobrevive) mal.push(o);
   }
   else {
      luchas_org_cola(mal, o, sobrevive);
      if (sobrevive) def.push(o);
   }
}
```

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 estándard y formados por celulas de N parámetros

Definición en la línea 109 del archivo Sistema.cpp.

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

  clear(def);
  clear(mal);
  int num = readint();
  Organismo o;
  for (int i = 1; i <= num; ++i) {
    o.leer(N);
    o.anadir_id(i);
    if (o.es_maligno()) mal.push(o);
    else def.push(o);
  }
  id = num+1;
}
```

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 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

Definición en la línea 129 del archivo Sistema.cpp.

```
{
// Esta operación simplemente recorre la cola de los organismos pedidos,
// proporcionando la información requerida, mediante una operación privada
// auxiliar: "escribir_sistema_cola".

  if (tipo) escribir_sistema_cola(def, estr);
  else escribir_sistema_cola(mal, estr);
}
```

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 31 del archivo Sistema.cpp.

```
{
// Busca el primer organismo de la cola c que consigue matar al organismo o y
// borra (y cuenta) todos los que resultan destruidos por éste. Para que el
// borrado sea efectivo, los supervivientes se van añadiendo al final de la
// cola. Al final, los que no hayan luchado se pasan al final la cola mediante
// una nueva operación privada auxiliar: "recolocar".

   Organismo actual;
   int resultado;

   sobrevive = true;
   int long_cola = c.size();

// Inv: sobrevive = ninguno de los elementos visitados de C destruye a o;
//      c = elementos no visitados de C seguidos por los elementos visitados
//      con su contador de víctimas actualizado, excepto los que hayan muerto
//      al enfrentarse con o, en el mismo orden en que estaban en C;
//      long_cola = número de elementos no visitados de C;
//      o tiene actualizado su contador de víctimas
//
// Cota: long_cola

   while (long_cola>0 and sobrevive) {
      actual = c.front();
      resultado = o.lucha_organismos(actual);
      switch (resultado) {
         case 0: {                    // los dos mueren
            sobrevive = false;        // no actualizamos las víctimas de actual

            o.incrementar_victimas(); // porque éste ya no
     pertenece al sistema
            break;
         }
         case 1: {                    // o muere, actual sobrevive
            sobrevive= false;
            actual.incrementar_victimas();
            c.push(actual);
            break;
         }
         case 2: {                    // o sobrevive, actual muere
            o.incrementar_victimas();
            break;
         }
         case 3: {                    // los dos sobreviven
            c.push(actual);
            break;
         }
      }
      c.pop();
      --long_cola;
   }

// Post1: Inv1 y (long_cola == 0 or not sobrevive)

// Falta pasar al final de c los elementos no visitados de C (por Inv1, son
// los long_cola primeros de c), para mantener el orden por identificador

   recolocar(long_cola,c);
}
```

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 104 del archivo Sistema.cpp.

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

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 <= 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 90 del archivo Sistema.cpp.

```
{
  // Inv:  n<=N, c contiene los mismos elementos que C, pero los N-n primeros
  //       elementos de C están al final de c, en el orden relativo original;
  //       los restantes también conservan su orden relativo original
  // Cota: n

  while (n>0) {
    c.push(c.front());
    c.pop();
    --n;
  }
}
```

4.3.3.7. 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

Definición en la línea 139 del archivo Sistema.cpp.

```
{
  queue<Organismo> aux(c);
  while (not aux.empty()) {
    aux.front().escribir(estr);
    aux.pop();
  }
}
```

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 37 del archivo Sistema.hpp.

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 42 del archivo Sistema.hpp.

4.3.4.3. int Sistema::id [private]

Primer número disponible de identificador de organismo.

Definición en la línea 45 del archivo Sistema.hpp.

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

- Sistema.hpp
- Sistema.cpp

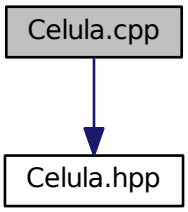
Capítulo 5

Documentación de archivos

5.1. Referencia del Archivo Celula.cpp

Código de la clase Celula.

Dependencia gráfica adjunta para Celula.cpp:



5.1.1. Descripción detallada

Código de la clase Celula.

Definición en el archivo Celula.cpp.

5.2. Referencia del Archivo Celula.hpp

Especificación de la clase Celula.

Clases

- class Celula

  Representa el conjunto de características y operaciones de las células.

5.2.1. Descripción detallada

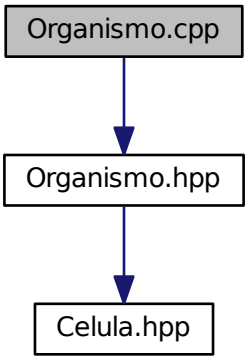
Especificación de la clase Celula.

Definición en el archivo Celula.hpp.

5.3. Referencia del Archivo Organismo.cpp

Código de la clase Organismo.

Dependencia gráfica adjunta para Organismo.cpp:



5.3.1. Descripción detallada

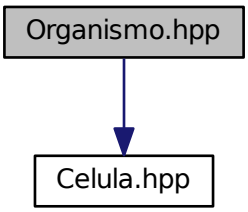
Código de la clase Organismo.

Definición en el archivo Organismo.cpp.

5.4. Referencia del Archivo Organismo.hpp

Especificación de la clase Organismo.

Dependencia gráfica adjunta para Organismo.hpp:



Clases

- class Organismo

  Representa la información y las operaciones asociadas a un organismo.

5.4.1. Descripción detallada

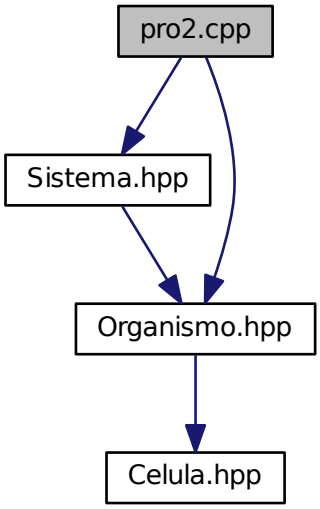
Especificación de la clase Organismo.

Definición en el archivo Organismo.hpp.

5.5. Referencia del Archivo pro2.cpp

Programa principal.

Dependencia gráfica adjunta para pro2.cpp:



Funciones

- int main ()

5.5.1. 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.

Definición en el archivo pro2.cpp.

5.5.2. Documentación de las funciones

5.5.2.1. int main ( )

Definición en la línea 40 del archivo pro2.cpp.

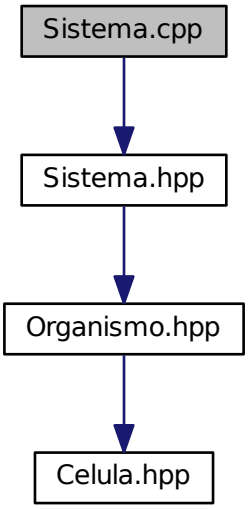
```
{
  int N = readint(); //  Número de parámetros de las células
  Sistema S;
  S.leer(N);
  int op; // Código de operación
  op = readint();
  while (op != -3) {
    if (op == -1 ) {
      Organismo O;
      O.leer(N);
      bool sobrevive;
      S.anadir_organismo(O, sobrevive);
      cout « "Entrada del nuevo organismo" « endl;
      cout « O.num_victimas() « " " « sobrevive « endl;
    }
    else if (op == -2) {
      bool tipo = readbool();
```

```
      bool estr = readbool();
      if (tipo){
  if (estr) cout « "Defensivos del sistema con estructura" « endl;
  else cout « "Defensivos del sistema sin estructura" « endl;
      }
      else {
  if (estr) cout « "Malignos del sistema con estructura" « endl;
  else cout « "Malignos del sistema sin estructura" « endl;
      }
      S.escribir(tipo, estr);
    }
    op = readint();
  }
}
```

5.6. Referencia del Archivo Sistema.cpp

Código de la clase Sistema.

Dependencia gráfica adjunta para Sistema.cpp:



5.6.1. Descripción detallada

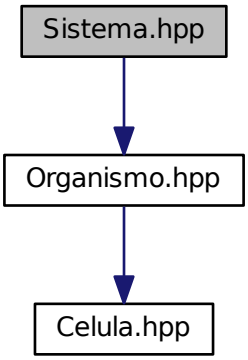
Código de la clase Sistema.

Definición en el archivo Sistema.cpp.

5.7. Referencia del Archivo Sistema.hpp

Especificación de la clase Sistema.

Dependencia gráfica adjunta para Sistema.hpp:



Clases

- class Sistema

  Representa el sistema donde se desarrollan los experimentos.

5.7.1. Descripción detallada

Especificación de la clase Sistema.

Definición en el archivo Sistema.hpp.

Índice alfabético