Tarea Programada 2

Generado por Doxygen 1.9.1

Capítulo 1

Tarea Programada 2

Descripción: Este proyecto consiste en una serie de “problemas” y otra de “solucionadores”, con el objetivo de que cada solucionador pueda seguir una serie de pasos para llegar al estado meta de un problema, y que este se muestre al usuario. Se programaron los problemas de Ball Sort, Torres de Hanoi y Lights Out.

Para ver información específica del enunciado, así como el funcionamiento de los problemas, la forma en la que se resolvió el problema y posibles mejoras, consultar el PDF de documentación externa.

Autores:

- Fabián Orozco Chaves (B95690)
- Sofía Velásquez Shum (C08395)
- Carolina Zamora (C08633)

1.1. Compilación

1.1.1. Usando g++:

Utilizar el siguiente comando: `g++ -o solucionar *.cpp`

1.1.2. Usando Makefile:

Utilizar el comando `make` o `make solucionar` para compilar todos los archivos.

1.2. Ejecución

Utilizar el comando `./solucionar` seguido de los parámetros de entrada en orden: nombre del problema y nombre del solucionador. Ejemplo:

```
./solucionar Problema Solucionador
```

Para los problemas, los nombres registrados del equipo son:

- `ProblemaFabian`
- `ProblemaSofia`
- `ProblemaCarolina`

Para los solucionadores, los nombres registrados del equipo son:

- `SolucionadorFabian`
- `SolucionadorSofia`
- `SolucionadorCarolina`

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 Clase Estado

`#include <Estado.h>`

Diagrama de herencias de Estado

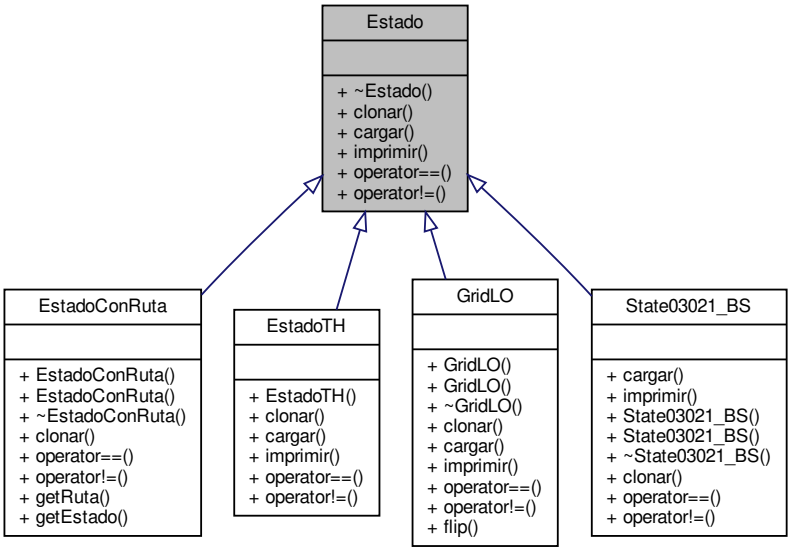
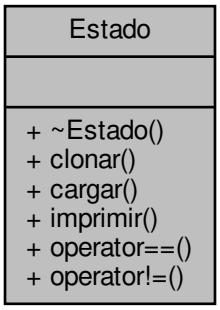


Diagrama de colaboración para Estado:



Métodos públicos

- virtual ∼Estado ()
- virtual Estado ∗ clonar ()=0
- virtual istream & cargar (istream &)=0
- virtual ostream & imprimir (ostream &)=0
- virtual int operator== (Estado ∗)=0
- virtual int operator!= (Estado ∗)=0

Amigas

- istream & operator>> (istream &entrada, Estado ∗estadoPtr)
- ostream & operator<< (ostream &salida, Estado ∗estadoPtr)

5.1.1. Documentación del constructor y destructor

5.1.1.1. ∼Estado()

```
virtual Estado::~Estado ( )  [inline], [virtual]
```

5.1.2. Documentación de las funciones miembro

5.1.2.1. cargar()

```
virtual istream& Estado::cargar (
            istream &  )  [pure virtual]
```

Implementado en EstadoTH y State03021_BS.

5.1.2.2. clonar()

```
virtual Estado* Estado::clonar ( )  [pure virtual]
```

Implementado en State03021_BS, GridLO, EstadoTH y EstadoConRuta.

5.1.2.3. imprimir()

```
virtual ostream& Estado::imprimir (
            ostream &  )  [pure virtual]
```

Implementado en EstadoTH y State03021_BS.

5.1.2.4. operatori=()

```
virtual int Estado::operator!= (
            Estado *  )  [pure virtual]
```

Implementado en GridLO, EstadoTH, State03021_BS y EstadoConRuta.

5.1.2.5. operator==()

```
virtual int Estado::operator== (
            Estado *  )  [pure virtual]
```

Implementado en GridLO, EstadoTH, State03021_BS y EstadoConRuta.

5.1.3. Documentación de las funciones relacionadas y clases amigas

5.1.3.1. operator<<

```
ostream& operator<< (
            ostream & salida,
            Estado * estadoPtr )  [friend]
```

5.1.3.2. operator>>

```
istream& operator>> (
            istream & entrada,
            Estado * estadoPtr )  [friend]
```

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

- Estado.h

5.2. Referencia de la Clase EstadoConRuta

```
#include <EstadoConRuta.h>
```

Diagrama de herencias de EstadoConRuta

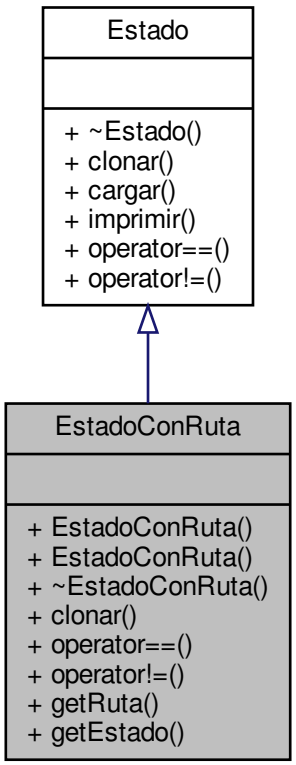
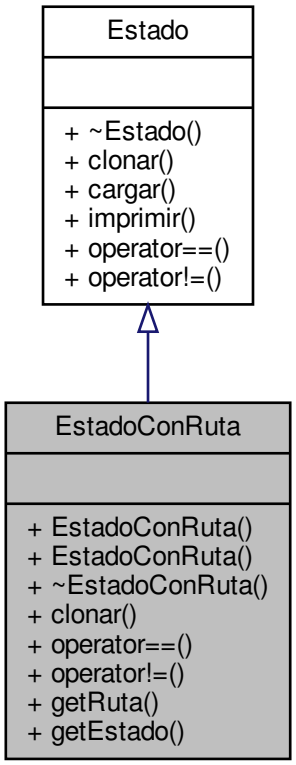


Diagrama de colaboración para EstadoConRuta:



Métodos públicos

- EstadoConRuta (Estado ∗, Lista &)

  Constructor con parámetros, se encarga de inicializar el 'estadoDerivado' realizando un clon con el metodo sobrecargado del método pasado por parámetro. Además, inicializa la ruta creando una nueva por copia y añadiendo el 'estadoDerivado' propio.
- EstadoConRuta (EstadoConRuta &)

  Constructor de copia, se encarga de inicializar el EstadoConRuta propio con los atributos de otro pasado por parámetro.
- ∼EstadoConRuta ()

  Destructor, se encarga de liberar memoria.
- EstadoConRuta ∗ clonar ()

  Método que clona un EstadoConRuta.
- int operator== (Estado ∗)

  Sobrecarga de operator ==. NO se utiliza. Se implementa por necesidad del polimorfismo (Clase Padre con virtual puro).
- int operator!= (Estado ∗)

  Sobrecarga de operator !=. NO se utiliza. Se implementa por necesidad del polimorfismo (Clase Padre con virtual puro).
- Lista ∗ getRuta ()

  Método que retorna el atributo ruta del objeto.
- Estado ∗ getEstado ()

  Método que retorna el atributo estado del objeto.

5.2.1. Documentación del constructor y destructor

5.2.1.1. EstadoConRuta() [1/2]

```
EstadoConRuta::EstadoConRuta (
            Estado * estado,
            Lista & lista )
```

Constructor con parámetros, se encarga de inicializar el 'estadoDerivado' realizando un clon con el metodo sobrecargado del método pasado por parámetro. Además, inicializa la ruta creando una nueva por copia y añadiendo el 'estadoDerivado' propio.

Parámetros

estado	Recibe un puntero a Estado con el estado que se quiere clonar y almacenar.
lista	Recibe una lista por referencia, ésta debe contener el camino con los estados que se exploraron para llegar al 'estadoDerivado'.

5.2.1.2. EstadoConRuta() [2/2]

```
EstadoConRuta::EstadoConRuta (
            EstadoConRuta & otro )
```

Constructor de copia, se encarga de inicializar el EstadoConRuta propio con los atributos de otro pasado por parámetro.

Parámetros

otro	Recibe un EstadoConRuta por referencia, del cual se quieren clonar sus atributos.

5.2.1.3. ~EstadoConRuta()

```
EstadoConRuta::~EstadoConRuta ( )
```

Destructor, se encarga de liberar memoria.

5.2.2. Documentación de las funciones miembro

5.2.2.1. clonar()

```
EstadoConRuta * EstadoConRuta::clonar ( )  [virtual]
```

Método que clona un EstadoConRuta.

Devuelve

EstadoConRuta* Devuelve un nuevo EstadoConRuta resultado del constructor de copia.

Implementa Estado.

5.2.2.2. getEstado()

```
Estado * EstadoConRuta::getEstado ( )
```

Método que retorna el atributo estado del objeto.

Devuelve

Estado* Devuelve el estado ('estadoDerivado') clonado y almacenado.

5.2.2.3. getRuta()

```
Lista * EstadoConRuta::getRuta ( )
```

Método que retorna el atributo ruta del objeto.

Devuelve

Lista* Devuelve la ruta (lista de 'estadosDerivados').

5.2.2.4. operatori=()

```
int EstadoConRuta::operator!= (
            Estado * estado )  [virtual]
```

Sobrecarga de operator !=. NO se utiliza. Se implementa por necesidad del polimorfismo (Clase Padre con virtual puro).

Devuelve

int Devuelve un 0;

Implementa Estado.

5.2.2.5. operator==()

```
int EstadoConRuta::operator== (
            Estado * estado )  [virtual]
```

Sobrecarga de operator ==. NO se utiliza. Se implementa por necesidad del polimorfismo (Clase Padre con virtual puro).

Devuelve

int Devuelve un 0;

Implementa Estado.

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

- EstadoConRuta.h
- EstadoConRuta.cpp

5.3. Referencia de la Clase EstadoTH

Clase derivada de Estado, representa tres torres con platillos (matrix) del problema Torres de Hanoi.

```
#include <EstadoTH.h>
```

Diagrama de herencias de EstadoTH

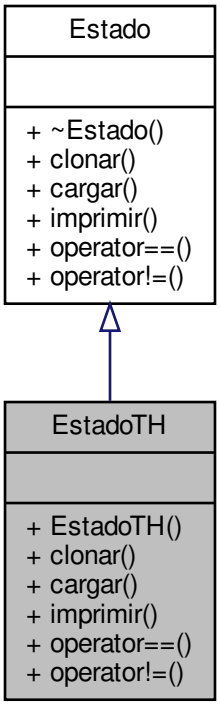
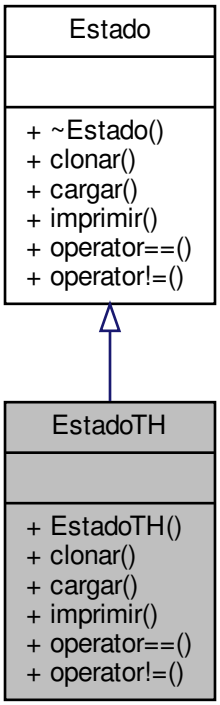


Diagrama de colaboración para EstadoTH:



Métodos públicos

- EstadoTH ()

  Constructor por omision (se crea matrix QPLATES x 3)
- EstadoTH ∗ clonar ()

  Clona this.
- istream & cargar (istream &entrada)

  Carga el estado desde un istream.
- ostream & imprimir (ostream &salida)

  Pasa la informacion del estado a un ostream.
- int operator== (Estado ∗e)

  Operador para determinar si dos estados son iguales.
- int operator!= (Estado ∗e)

  Operador para determinar si dos estados son distintos.

Amigas

- class ProblemaTH

5.3.1. Descripción detallada

Clase derivada de Estado, representa tres torres con platillos (matrix) del problema Torres de Hanoi.

5.3.2. Documentación del constructor y destructor

5.3.2.1. EstadoTH()

```
EstadoTH::EstadoTH ( )
```

Constructor por omision (se crea matrix QPLATES x 3)

5.3.3. Documentación de las funciones miembro

5.3.3.1. cargar()

```
istream & EstadoTH::cargar (
            istream & entrada )  [virtual]
```

Carga el estado desde un istream.

Parámetros

entrada	Istream del cual se carga el estado

Devuelve

istream modificado

Implementa Estado.

5.3.3.2. clonar()

```
EstadoTH * EstadoTH::clonar ( )  [virtual]
```

Clona this.

Devuelve

Puntero a EstadoTH (clon de this)

Implementa Estado.

5.3.3.3. imprimir()

```
ostream & EstadoTH::imprimir (
            ostream & salida )  [virtual]
```

Pasa la informacion del estado a un ostream.

Parámetros

salida	Ostream al cual se imprime el estado

Devuelve

Ostream modificado

Implementa Estado.

5.3.3.4. operatori=()

```
int EstadoTH::operator!= (
            Estado * e )  [virtual]
```

Operador para determinar si dos estados son distintos.

Parámetros

e	Puntero a estado

Devuelve

False solo si e apunta a un EstadoTH y sus matrix son iguales

Implementa Estado.

5.3.3.5. operator==()

```
int EstadoTH::operator== (
            Estado * e )  [virtual]
```

Operador para determinar si dos estados son iguales.

Parámetros

e	Puntero a estado

Devuelve

True solo si e apunta a un EstadoTH y sus matrix son iguales

Implementa Estado.

5.3.4. Documentación de las funciones relacionadas y clases amigas

5.3.4.1. ProblemaTH

```
friend class ProblemaTH  [friend]
```

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

- EstadoTH.h
- EstadoTH.cpp

5.4. Referencia de la Clase Fabrica

```
#include <Fabrica.h>
```

Diagrama de herencias de Fabrica

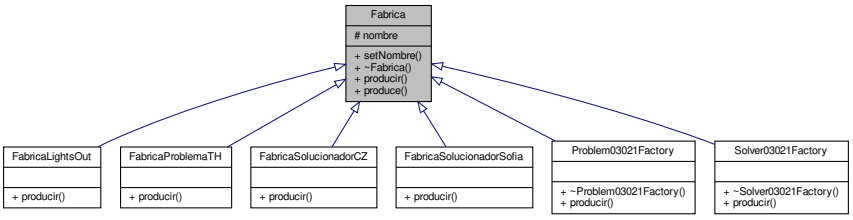
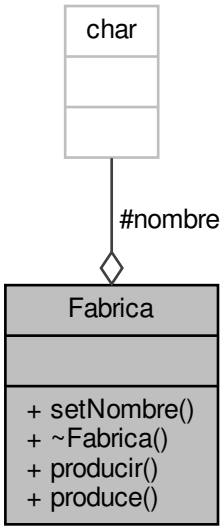


Diagrama de colaboración para Fabrica:



Métodos públicos

- virtual void setNombre (const char ∗nombre)
- virtual ∼Fabrica ()
- virtual Producto ∗ producir ()=0
- virtual int produce (const char ∗nombre)

Atributos protegidos

- char nombre [NOMBRE_MAX_SIZE]

5.4.1. Documentación del constructor y destructor

5.4.1.1. ∼Fabrica()

```
virtual Fabrica::~Fabrica ( )  [inline], [virtual]
```

5.4.2. Documentación de las funciones miembro

5.4.2.1. produce()

```
virtual int Fabrica::produce (
            const char * nombre )  [inline], [virtual]
```

5.4.2.2. producir()

```
virtual Producto* Fabrica::producir ( )  [pure virtual]
```

Implementado en Solver03021Factory, Problem03021Factory, FabricaSolucionadorSofia, FabricaSolucionadorCZ, FabricaProblemaTH y FabricaLightsOut.

5.4.2.3. setNombre()

```
virtual void Fabrica::setNombre (
            const char * nombre )  [inline], [virtual]
```

5.4.3. Documentación de los datos miembro

5.4.3.1. nombre

```
char Fabrica::nombre[NOMBRE_MAX_SIZE]  [protected]
```

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

- Fabrica.h

5.5. Referencia de la Clase FabricaLightsOut

Clase derivada de Fabrica, para producir un puntero a LightsOut.

`#include <FabricaLightsOut.h>`

Diagrama de herencias de FabricaLightsOut

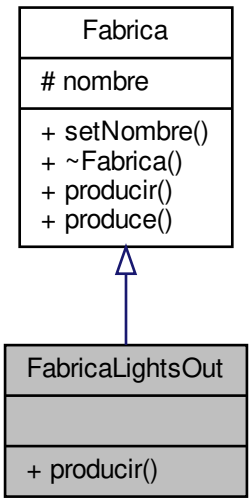
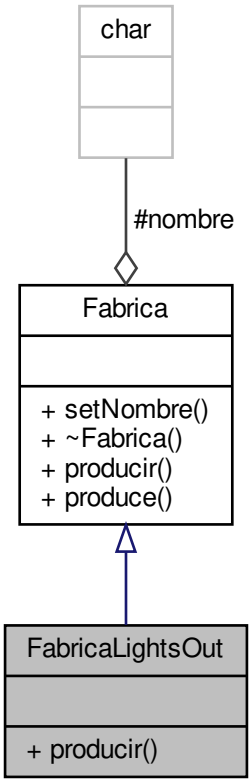


Diagrama de colaboración para FabricaLightsOut:



Métodos públicos

- LightsOut ∗ producir ()

  Retorna un Puntero a LightsOut.

Otros miembros heredados

5.5.1. Descripción detallada

Clase derivada de Fabrica, para producir un puntero a LightsOut.

5.5.2. Documentación de las funciones miembro

5.5.2.1. producir()

```
LightsOut * FabricaLightsOut::producir ( )  [virtual]
```

Retorna un Puntero a LightsOut.

Devuelve

Puntero a LightsOut construido por omisión

Implementa Fabrica.

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

- FabricaLightsOut.h
- FabricaLightsOut.cpp

5.6. Referencia de la Clase FabricaProblemaTH

Clase derivada de Fabrica, produce un puntero a ProblemaTH.

```
#include <FabricaProblemaTH.h>
```

Diagrama de herencias de FabricaProblemaTH

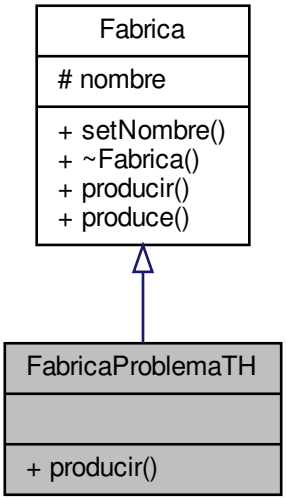
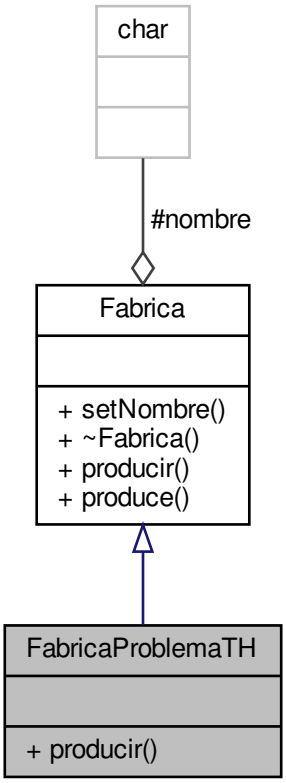


Diagrama de colaboración para FabricaProblemaTH:



Métodos públicos

- ProblemaTH * producir ()

  Retorna un puntero a ProblemaTH.

Otros miembros heredados

5.6.1. Descripción detallada

Clase derivada de Fabrica, produce un puntero a ProblemaTH.

5.6.2. Documentación de las funciones miembro

5.6.2.1. producir()

```
ProblemaTH * FabricaProblemaTH::producir ( )  [virtual]
```

Retorna un puntero a ProblemaTH.

Devuelve

Puntero a ProblemaTH construido por omision

Implementa Fabrica.

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

- FabricaProblemaTH.h
- FabricaProblemaTH.cpp

5.7. Referencia de la Clase FabricaSolucionadorCZ

Clase derivada de Fabrica, para producir un puntero a SolucionadorCarolina.

```
#include <FabricaSolucionadorCZ.h>
```

Diagrama de herencias de FabricaSolucionadorCZ

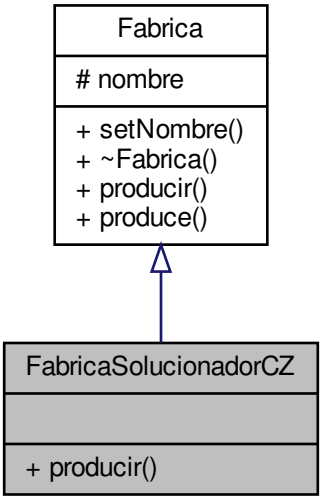
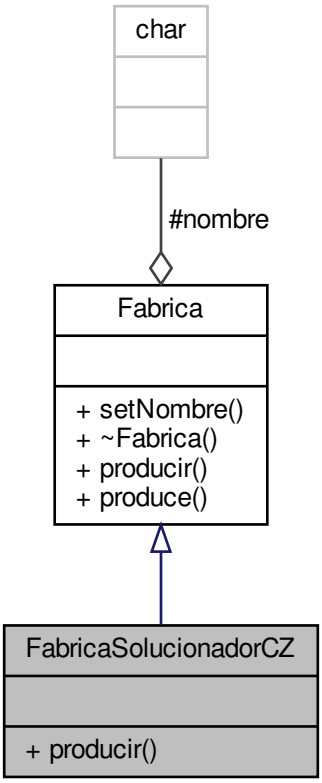


Diagrama de colaboración para FabricaSolucionadorCZ:



Métodos públicos

- SolucionadorCarolina ∗ producir ()

  Retorna un Puntero a SolucionadorCarolina.

Otros miembros heredados

5.7.1. Descripción detallada

Clase derivada de Fabrica, para producir un puntero a SolucionadorCarolina.

5.7.2. Documentación de las funciones miembro

5.7.2.1. producir()

```
SolucionadorCarolina * FabricaSolucionadorCZ::producir ( )  [virtual]
```

Retorna un Puntero a SolucionadorCarolina.

Devuelve

Puntero a SolucionadorCarolina construido por omisión

Implementa Fabrica.

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

- FabricaSolucionadorCZ.h
- FabricaSolucionadorCZ.cpp

5.8. Referencia de la Clase FabricaSolucionadorSofia

Clase derivada de Fabrica, produce un puntero a SolucionadorSofia.

```
#include <FabricaSolucionadorSofia.h>
```

Diagrama de herencias de FabricaSolucionadorSofia

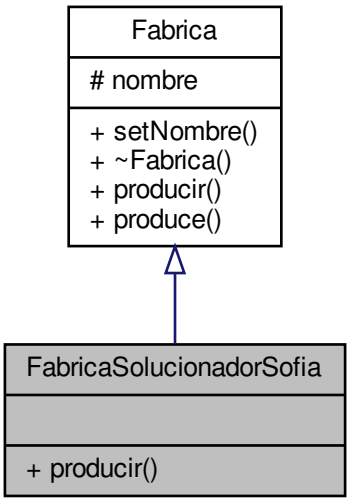
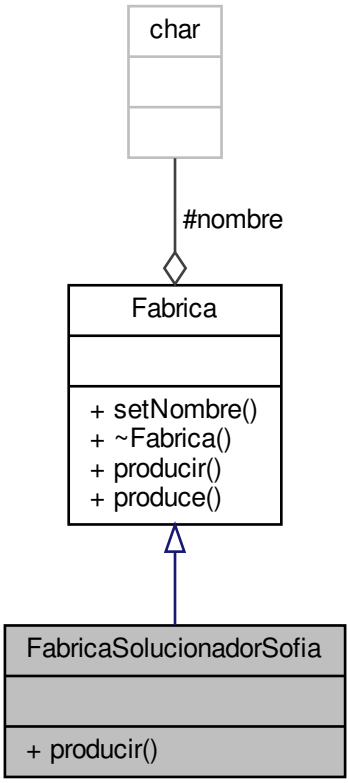


Diagrama de colaboración para FabricaSolucionadorSofia:



Métodos públicos

- SolucionadorSofia * producir ()

  Retorna un puntero a SolucionadorSofia.

Otros miembros heredados

5.8.1. Descripción detallada

Clase derivada de Fabrica, produce un puntero a SolucionadorSofia.

5.8.2. Documentación de las funciones miembro

5.8.2.1. producir()

```
SolucionadorSofia * FabricaSolucionadorSofia::producir ( )  [virtual]
```

Retorna un puntero a SolucionadorSofia.

Devuelve

Puntero a SolucionadorSofia construido por omision

Implementa Fabrica.

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

- FabricaSolucionadorSofia.h
- FabricaSolucionadorSofia.cpp

5.9. Referencia de la Clase GridLO

Clase derivada de Estado, representa una cuadrícula (grid) del problema LightsOut.

```
#include <GridLO.h>
```

Diagrama de herencias de GridLO

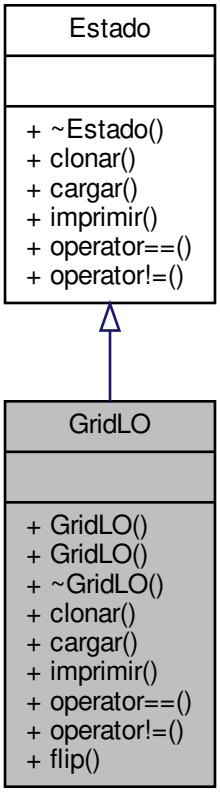
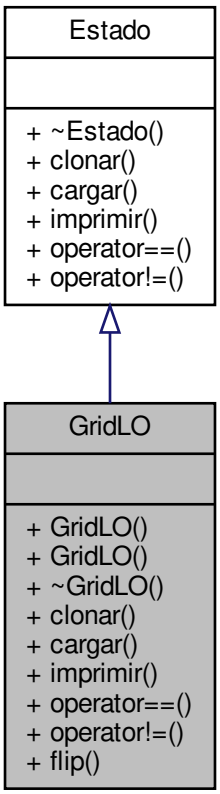


Diagrama de colaboración para GridLO:



Métodos públicos

- GridLO ()

  Constructor por omisión.
- GridLO (const GridLO &grid)

  Constructor por copia.
- ∼GridLO ()

  Destructor.
- GridLO ∗ clonar ()

  Clona this.
- std::istream & cargar (std::istream &in)

  Carga el estado desde un istream.
- std::ostream & imprimir (std::ostream &out)

  Pasa la información del estado a un ostream.
- int operator== (Estado ∗e)

  Operador para determinar si dos estados son iguales.
- int operator!= (Estado ∗e)

  Operador para determinar si dos estados son distintos.
- GridLO flip (unsigned int row, unsigned int column)

  Genera el resultado de invertir una casilla y las adyacentes.

Amigas

- class LightsOut

5.9.1. Descripción detallada

Clase derivada de Estado, representa una cuadrícula (grid) del problema LightsOut.

5.9.2. Documentación del constructor y destructor

5.9.2.1. GridLO() [1/2]

```
GridLO::GridLO ( )
```

Constructor por omisión.

Se crea grid de 4x4

5.9.2.2. GridLO() [2/2]

```
GridLO::GridLO (
            const GridLO & grid )
```

Constructor por copia.

Parámetros

grid	GridLO que se copia

5.9.2.3. ~GridLO()

```
GridLO::~GridLO ( )
```

Destructor.

5.9.3. Documentación de las funciones miembro

5.9.3.1. cargar()

```
istream & GridLO::cargar (
            std::istream & in )
```

Carga el estado desde un istream.

Parámetros

in	Istream del cual se carga el estado

Devuelve

Istream modificado

5.9.3.2. clonar()

```
GridLO * GridLO::clonar ( )  [virtual]
```

Clona this.

Devuelve

Puntero a GridLo clonado

Implementa Estado.

5.9.3.3. flip()

```
GridLO GridLO::flip (
            unsigned int row,
            unsigned int column )
```

Genera el resultado de invertir una casilla y las adyacentes.

Parámetros

row	Fila de la casilla invertida
column	Columna de la casilla invertida

Devuelve

GridLO resultado de hacer el movimiento

5.9.3.4. imprimir()

```
ostream & GridLO::imprimir (
            std::ostream & out )
```

Pasa la información del estado a un ostream.

Parámetros

out	Ostream al cual se imprime

Devuelve

Ostream modificado

5.9.3.5. operatori=()

```
int GridLO::operator!= (
            Estado * e )  [virtual]
```

Operador para determinar si dos estados son distintos.

Parámetros

e	Puntero a estado

Devuelve

False solo si e apunta a un GridLO y los grids son iguales

Implementa Estado.

5.9.3.6. operator==()

```
int GridLO::operator== (
            Estado * e )  [virtual]
```

Operador para determinar si dos estados son iguales.

Parámetros

e	Puntero a estado

Devuelve

True solo si e apunta a un GridLO y los grids son iguales

Implementa Estado.

5.9.4. Documentación de las funciones relacionadas y clases amigas

5.9.4.1. LightsOut

```
friend class LightsOut  [friend]
```

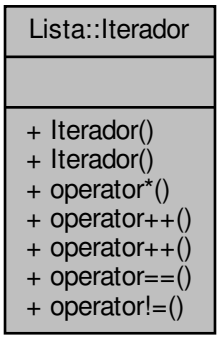
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- GridLO.h
- GridLO.cpp

5.10. Referencia de la Clase Lista::Iterador

```
#include <Lista.h>
```

Diagrama de colaboración para Lista::Iterador:



Métodos públicos

- Iterador ()
- Iterador (Celda ∗)
- Estado ∗& operator∗ ()
- Iterador & operator++ ()
- Iterador operator++ (int)
- int operator== (Iterador)
- int operator!= (Iterador)

Amigas

- class Lista

5.10.1. Documentación del constructor y destructor

5.10.1.1. Iterador() [1/2]

```
Lista::Iterador::Iterador ( )
```

5.10.1.2. Iterador() [2/2]

```
Lista::Iterador::Iterador (
            Celda * actual )
```

5.10.2. Documentación de las funciones miembro

5.10.2.1. operatori=()

```
int Lista::Iterador::operator!= (
            Iterador otro )
```

5.10.2.2. operator∗()

```
Estado *& Lista::Iterador::operator* ( )
```

5.10.2.3. operator++() [1/2]

```
Lista::Iterador & Lista::Iterador::operator++ ( )
```

5.10.2.4. operator++() [2/2]

```
Lista::Iterador Lista::Iterador::operator++ (
            int  )
```

5.10.2.5. operator==()

```
int Lista::Iterador::operator== (
            Iterador otro )
```

5.10.3. Documentación de las funciones relacionadas y clases amigas

5.10.3.1. Lista

```
friend class Lista  [friend]
```

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

- Lista.h
- Lista.cpp

5.11. Referencia de la Clase LightsOut

Clase derivada de Problema, representa el problema LightsOut.

```
#include <LightsOut.h>
```

Diagrama de herencias de LightsOut

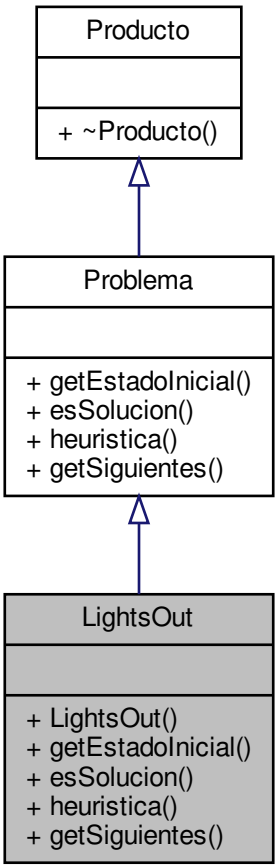
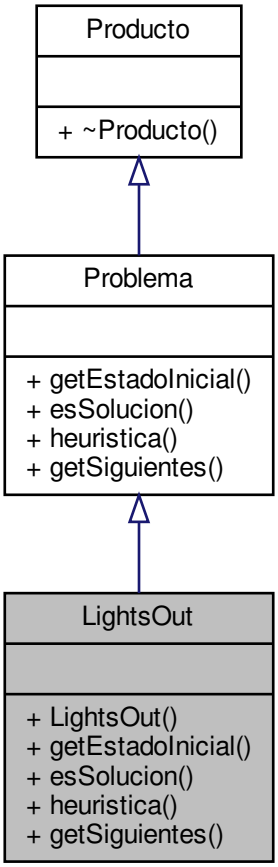


Diagrama de colaboración para LightsOut:



Métodos públicos

- LightsOut ()

  Constructor por omisión.
- GridLO ∗ getEstadoInicial ()

  Obtiene el estado inicial de GridLO.
- int esSolucion (Estado ∗e)

  Determina si un estado es solución.
- int heuristica (Estado ∗e)

  Determina la heurística de un estado.
- Lista ∗ getSiguientes (Estado ∗e)

  Obtiene los siguientes estados de un estado.

5.11.1. Descripción detallada

Clase derivada de Problema, representa el problema LightsOut.

5.11.2. Documentación del constructor y destructor

5.11.2.1. LightsOut()

```
LightsOut::LightsOut ( )
```

Constructor por omisión.

5.11.3. Documentación de las funciones miembro

5.11.3.1. esSolucion()

```
int LightsOut::esSolucion (
            Estado * e )  [virtual]
```

Determina si un estado es solución.

Parámetros

e	Puntero a Estado

Devuelve

True solo si e apunta a un GridLO y el grid está todo apagado

Implementa Problema.

5.11.3.2. getEstadoInicial()

```
GridLO * LightsOut::getEstadoInicial ( )  [virtual]
```

Obtiene el estado inicial de GridLO.

Devuelve

Puntero a GridLO inicial

Implementa Problema.

5.11.3.3. getSiguientes()

```
Lista * LightsOut::getSiguientes (
            Estado * e )  [virtual]
```

Obtiene los siguientes estados de un estado.

Parámetros

e	Puntero a Estado

Devuelve

Puntero a Lista que contiene los estados siguientes a ∗e

Implementa Problema.

5.11.3.4. heuristica()

```
int LightsOut::heuristica (
            Estado * e )  [virtual]
```

Determina la heurística de un estado.

Parámetros

e	Puntero a Estado

Devuelve

Valor de la heurística (cantidad de luces encendidas en el GridLO)

Implementa Problema.

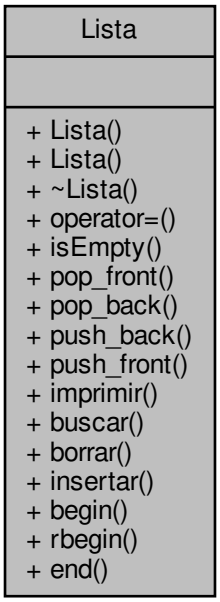
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- LightsOut.h
- LightsOut.cpp

5.12. Referencia de la Clase Lista

```
#include <Lista.h>
```

Diagrama de colaboración para Lista:



Clases

- class Iterador

Métodos públicos

- Lista ()
- Lista (Lista &)
- ∼Lista ()
- Lista & operator= (Lista &)
- int isEmpty ()
- Estado ∗ pop_front ()
- Estado ∗ pop_back ()
- Lista & push_back (Estado ∗)
- Lista & push_front (Estado ∗)
- ostream & imprimir (ostream &)
- Iterador buscar (Estado ∗)
- Lista & borrar (Iterador &)
- Lista & insertar (Iterador, Estado ∗)
- Iterador begin ()
- Iterador rbegin ()
- Iterador end ()

Amigas

- class Iterador
- ostream & operator<< (ostream &salida, Lista *lista)

5.12.1. Documentación del constructor y destructor

5.12.1.1. Lista() [1/2]

```
Lista::Lista ( )
```

5.12.1.2. Lista() [2/2]

```
Lista::Lista (
            Lista & lista )
```

5.12.1.3. ~Lista()

```
Lista::~Lista ( )
```

5.12.2. Documentación de las funciones miembro

5.12.2.1. begin()

```
Lista::Iterador Lista::begin ( )
```

5.12.2.2. borrar()

```
Lista & Lista::borrar (
            Iterador & i )
```

5.12.2.3. buscar()

```
Lista::Iterador Lista::buscar (
            Estado * elemento )
```

5.12.2.4. end()

```
Lista::Iterador Lista::end ( )
```

5.12.2.5. imprimir()

```
ostream & Lista::imprimir (
            ostream & salida )
```

5.12.2.6. insertar()

```
Lista & Lista::insertar (
            Iterador i,
            Estado * elemento )
```

5.12.2.7. isEmpty()

```
int Lista::isEmpty ( )
```

5.12.2.8. operator=()

```
Lista & Lista::operator= (
            Lista & lista )
```

5.12.2.9. pop_back()

```
Estado * Lista::pop_back ( )
```

5.12.2.10. pop_front()

```
Estado * Lista::pop_front ( )
```

5.12.2.11. push_back()

```
Lista & Lista::push_back (
            Estado * elemento )
```

5.12.2.12. push_front()

```
Lista & Lista::push_front (
            Estado * elemento )
```

5.12.2.13. rbegin()

```
Lista::Iterador Lista::rbegin ( )
```

5.12.3. Documentación de las funciones relacionadas y clases amigas

5.12.3.1. Iterador

```
friend class Iterador  [friend]
```

5.12.3.2. operator<<

```
ostream& operator<< (
            ostream & salida,
            Lista * lista )  [friend]
```

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

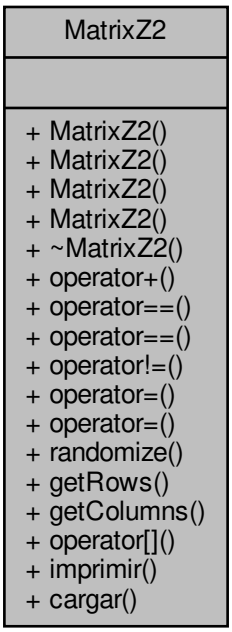
- Lista.h
- Lista.cpp

5.13. Referencia de la Clase MatrixZ2

Esta clase representa una matriz con entradas en el cuerpo Z2 (F2)

`#include <MatrixZ2.h>`

Diagrama de colaboración para MatrixZ2:



Métodos públicos

- MatrixZ2 ()

  Constructor por omisión.
- MatrixZ2 (unsigned int rows, unsigned int columns)

  Constructor por dimensiones (rectangular)
- MatrixZ2 (unsigned int rowsColumns)

  Constructor por dimensiones (cuadrada)
- MatrixZ2 (const MatrixZ2 &matrix)

  Constructor de copia.
- ~MatrixZ2 ()
- MatrixZ2 operator+ (MatrixZ2 &matrix)

  Suma de dos matrices.
- int operator== (MatrixZ2 &matrix)

  Determina si dos matrices son iguales.
- int operator== (bool b)

  Determina si todas las entradas son el valor que recibe.
- int operator!= (MatrixZ2 &matrix)

Determina si dos matrices son diferentes.

- void operator= (MatrixZ2 &matrix)

  Operador de asignación con otra matriz.
- void operator= (bool value)

  Operador de asignación con un valor.
- void randomize ()

  Rellena la matriz con valores aleatorios.
- int getRows ()

  Retorna cantidad de filas.
- int getColumns ()

  Retorna cantidad de columnas.
- bool * operator[ ] (int n)

  Permite acceder a una fila (para leer o modificar)
- std::ostream & imprimir (std::ostream &out)

  Pasa los contenidos de la matriz a un ostream.
- std::istream & cargar (std::istream &in)

  Carga los contenidos de la matriz desde un istream.

5.13.1. Descripción detallada

Esta clase representa una matriz con entradas en el cuerpo Z2 (F2)

5.13.2. Documentación del constructor y destructor

5.13.2.1. MatrixZ2() [1/4]

```
MatrixZ2::MatrixZ2 ( )
```

Constructor por omisión.

Crea una matriz 1x1 con entradas 0

5.13.2.2. MatrixZ2() [2/4]

```
MatrixZ2::MatrixZ2 (
            unsigned int rows,
            unsigned int columns )
```

Constructor por dimensiones (rectangular)

Crea una matriz rowsxcolumns con entradas 0

Parámetros

rows	Cantidad de filas
columns	Cantidad de columnas

5.13.2.3. MatrixZ2() [3/4]

```
MatrixZ2::MatrixZ2 (
            unsigned int rowsColumns )
```

Constructor por dimensiones (cuadrada)

Crea una matriz rowsColumnsxrowsColumns con entradas 0

Parámetros

rowsColumns	Cantidad de filas y de columnas

5.13.2.4. MatrixZ2() [4/4]

```
MatrixZ2::MatrixZ2 (
            const MatrixZ2 & matrix )
```

Constructor de copia.

Parámetros

matrix	Matriz que va a copiar

5.13.2.5. ~MatrixZ2()

```
MatrixZ2::~MatrixZ2 ( )
```

Destructor (libera la memoria ocupada por la matriz)

5.13.3. Documentación de las funciones miembro

5.13.3.1. cargar()

```
std::istream & MatrixZ2::cargar (
            std::istream & in )
```

Carga los contenidos de la matriz desde un istream.

Parámetros

in	Istream de donde se carga

Devuelve

Istream modificado

5.13.3.2. getColumns()

```
int MatrixZ2::getColumns ( )
```

Retorna cantidad de columnas.

Devuelve

Cantidad de columnas (int)

5.13.3.3. getRows()

```
int MatrixZ2::getRows ( )
```

Retorna cantidad de filas.

Devuelve

Cantidad de filas (int)

5.13.3.4. imprimir()

```
std::ostream & MatrixZ2::imprimir (
            std::ostream & out )
```

Pasa los contenidos de la matriz a un ostream.

Parámetros

out	Ostream donde se imprime

Devuelve

Ostream modificado

5.13.3.5. operatori=()

```
int MatrixZ2::operator!= (
            MatrixZ2 & matrix )
```

Determina si dos matrices son diferentes.

Parámetros

matriz	Matriz a comparar con this

Devuelve

False solo si todas las entradas respectivas son iguales

5.13.3.6. operator+()

```
MatrixZ2 MatrixZ2::operator+ (
            MatrixZ2 & matrix )
```

Suma de dos matrices.

Parámetros

matrix	Matriz que se va a sumar con this

Devuelve

Matriz del resultado de la suma

5.13.3.7. operator=() [1/2]

```
void MatrixZ2::operator= (
            bool value )
```

Operador de asignación con un valor.

Parámetros

value	Bool que se va a asignar a todas las entradas (mantiene la misma dimensión)

5.13.3.8. operator=() [2/2]

```
void MatrixZ2::operator= (
            MatrixZ2 & matrix )
```

Operador de asignación con otra matriz.

Parámetros

matriz	Matriz que se va a copiar y asignar a this

5.13.3.9. operator==() [1/2]

```
int MatrixZ2::operator== (
            bool b )
```

Determina si todas las entradas son el valor que recibe.

Parámetros

b	Bool contra el cual compara cada entrada de la matriz

Devuelve

True solo si todas las entradas son iguales a b

5.13.3.10. operator==() [2/2]

```
int MatrixZ2::operator== (
            MatrixZ2 & matrix )
```

Determina si dos matrices son iguales.

Parámetros

matriz	Matriz a comparar con this

Devuelve

True solo si todas las entradas respectivas son iguales

5.13.3.11. operator[]()

```
bool * MatrixZ2::operator[] (
            int n )
```

Permite acceder a una fila (para leer o modificar)

Parámetros

n	Número de fila

Devuelve

Puntero a bool correspondiente a la fila n de la matriz

5.13.3.12. randomize()

```
void MatrixZ2::randomize ( )
```

Rellena la matriz con valores aleatorios.

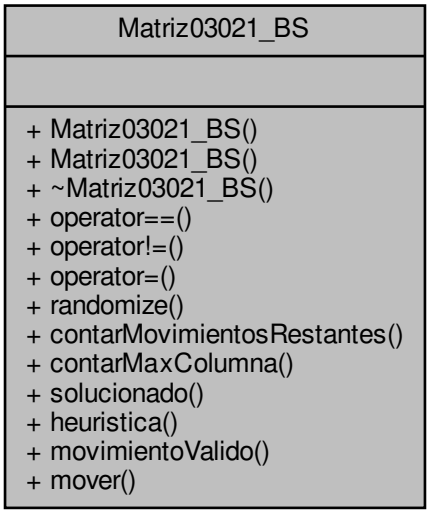
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- MatrixZ2.h
- MatrixZ2.cpp

5.14. Referencia de la Clase Matriz03021_BS

```
#include <Matriz03021_BS.h>
```

Diagrama de colaboración para Matriz03021_BS:



Métodos públicos

- Matriz03021_BS ()

  Constructor por omisión, invoca al método _init() que inicializa la creación de la matriz.
- Matriz03021_BS (Matriz03021_BS &)

  Constructor de copia, invoca métodos que se encargan de reservar memoria y copiar las entradas de la matriz así como su vector descenso;.
- ~Matriz03021_BS ()

  Destructor, invoca a un método que se encarga de la liberación de la memoria.
- int operator== (Matriz03021_BS &)

  Sobrecarga del operador ==, realiza una comparación entrada por entrada de la matriz.
- int operator!= (Matriz03021_BS &)

  Sobrecarga del operador !=, realiza una comparación entrada por entrada de la matriz.
- void operator= (Matriz03021_BS &)

  Sobrecarga del operador =, invoca a otros métodos que se encargan de asignar una matriz pasada por referencia en parámetro a la propia matriz.
- void randomize ()

  Metodo que se encarga de randomizar las primeras cuatro columnas de la matriz. Utiliza el método rand() que genera valores de 0 a 4, además del método srand() y time() que se encargan de crear la semilla aleatoria en tiempo de ejecución. La generación de número aleatorio permite elegir entra los indices del vector Colores(atributo de Matriz03021_BS) para ser asignados si estos tienen menos de 4 apariciones en la matriz.
- int contarMovimientosRestantes ()

  Metodo que cuenta la cantidad mínima de ordenaciones que se tienen que dar para que la matriz esté ordenada.
- int contarMaxColumna (int)

  Metodo que cuenta la cantidad de apariciones máxima de un caracter en una columna.
- int solucionado ()

  Metodo que verifica si la matriz está ordenada completamente (solucionada) o no.
- int heuristica ()

Metodo que calcula la heurística optimista según el contenido de la matriz.

- int movimientoValido (int, int)

  Metodo que verifica la validez de un movimiento para saber si puede realizarse. Se trabaja con el elemento que está más arriba (de fila 0 a fila 3) de cada columna de acuerdo al vector descensos[].

- Matriz03021_BS & mover (int, int)

  Método que realiza el movimiento de una de un caracter de una columna hacia otra dependiendo si es válido o no.

Amigas

- ostream & operator<< (ostream &salida, Matriz03021_BS &matriz)
- istream & operator>> (istream &entrada, Matriz03021_BS &matriz)

5.14.1. Documentación del constructor y destructor

5.14.1.1. Matriz03021_BS() [1/2]

```
Matriz03021_BS::Matriz03021_BS ( )
```

Constructor por omisión, invoca al método _init() que inicializa la creación de la matriz.

5.14.1.2. Matriz03021_BS() [2/2]

```
Matriz03021_BS::Matriz03021_BS (
            Matriz03021_BS & otra )
```

Constructor de copia, invoca métodos que se encargan de reservar memoria y copiar las entradas de la matriz así como su vector descenso;.

Parámetros

otra	Recibe un objeto Matriz03021_BS por referencia.

5.14.1.3. ~Matriz03021_BS()

```
Matriz03021_BS::~Matriz03021_BS ( )
```

Destructor, invoca a un método que se encarga de la liberación de la memoria.

5.14.2. Documentación de las funciones miembro

5.14.2.1. contarMaxColumna()

```
int Matriz03021_BS::contarMaxColumna (
            int columna )
```

Metodo que cuenta la cantidad de apariciones máxima de un caracter en una columna.

Parámetros

columna	Recibe el subindice de la columna que se desea saber la cantidad de apariciones máxima de un caracter.

Devuelve

int Devuelve un entero que representa la cantidad de apariciones máxima de un caracter.

5.14.2.2. contarMovimientosRestantes()

```
int Matriz03021_BS::contarMovimientosRestantes ( )
```

Metodo que cuenta la cantidad mínima de ordenaciones que se tienen que dar para que la matriz esté ordenada.

Devuelve

int Devuelve un entero que representa la cantidad mínima de ordenaciones que se tienen que dar para que la matriz esté ordenada.

5.14.2.3. heuristica()

```
int Matriz03021_BS::heuristica ( )
```

Metodo que calcula la heurística optimista según el contenido de la matriz.

Devuelve

int Devuelve un entero con la cantidad aproximada de movimientos mínimos que se deben realizar para llegar a la solución.

5.14.2.4. mover()

```
Matriz03021_BS & Matriz03021_BS::mover (
            int columnaOrigen,
            int columnaDestino )
```

Método que realiza el movimiento de una de un caracter de una columna hacia otra dependiendo si es válido o no.

Parámetros

columnaOrigen	Recibe el número de la columna de la que se quiere mover el elemento.
columnaDestino	Recibe el entero que representa la columna en la que se quiere colocar el elemento.

Devuelve

Matriz03021_BS Devuelve un tipo Matriz03021_BS por referencia (se devuelve así misma), con los cambios realizados o no.

5.14.2.5. movimientoValido()

```
int Matriz03021_BS::movimientoValido (
            int columnaOrigen,
            int columnaDestino )
```

Metodo que verifica la validez de un movimiento para saber si puede realizarse. Se trabaja con el elemento que está más arriba (de fila 0 a fila 3) de cada columna de acuerdo al vector descensos[].

Parámetros

columnaOrigen	Recibe el número de la columna de la que se quiere mover el elemento.
columnaDestino	Recibe el entero que representa la columna en la que se quiere colocar el elemento.

Devuelve

int Devuelve un entero distinto de 0 si el movimiento es válido; de forma contraria, retorna 0.

5.14.2.6. operatori=()

```
int Matriz03021_BS::operator!= (
            Matriz03021_BS & otra )
```

Sobrecarga del operador !=, realiza una comparación entrada por entrada de la matriz.

Parámetros

otra	Recibe un objeto Matriz03021_BS por referencia.

Devuelve

int Devuelve 1 si al menos una de las entradas es distinta; de manera contraria retorna 0.

5.14.2.7. operator=()

```
void Matriz03021_BS::operator= (
            Matriz03021_BS & otra )
```

Sobrecarga del operador =, invoca a otros métodos que se encargan de asignar una matriz pasada por referencia en parámetro a la propia matriz.

Parámetros

otra	Recibe un objeto Matriz03021_BS por referencia.

5.14.2.8. operator==()

```
int Matriz03021_BS::operator== (
            Matriz03021_BS & otra )
```

Sobrecarga del operador ==, realiza una comparación entrada por entrada de la matriz.

Parámetros

otra	Recibe un objeto Matriz03021_BS por referencia.

Devuelve

int Devuelve 0 si al menos una de las entradas es distinta; de manera contraria retorna 1.

5.14.2.9. randomize()

```
void Matriz03021_BS::randomize ( )
```

Metodo que se encarga de randomizar las primeras cuatro columnas de la matriz. Utiliza el método rand() que genera valores de 0 a 4, además del método srand() y time() que se encargan de crear la semilla aleatoria en tiempo de ejecución. La generación de número aleatorio permite elegir entra los indices del vector Colores(atributo de Matriz03021_BS) para ser asignados si estos tienen menos de 4 apariciones en la matriz.

5.14.2.10. solucionado()

```
int Matriz03021_BS::solucionado ( )
```

Metodo que verifica si la matriz está ordenada completamente (solucionada) o no.

Devuelve

int Devuelve un entero distinto a 0 si la matriz está completamente ordenada; en caso contrario retorna 0;

5.14.3. Documentación de las funciones relacionadas y clases amigas

5.14.3.1. operator<<

```
ostream& operator<< (
            ostream & salida,
            Matriz03021_BS & matriz )  [friend]
```

5.14.3.2. operator>>

```
istream& operator>> (
            istream & entrada,
            Matriz03021_BS & matriz )  [friend]
```

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

- Matriz03021_BS.h
- Matriz03021_BS.cpp

5.15. Referencia de la Clase Problem03021_BS

```
#include <Problem03021_BS.h>
```

Diagrama de herencias de Problem03021_BS

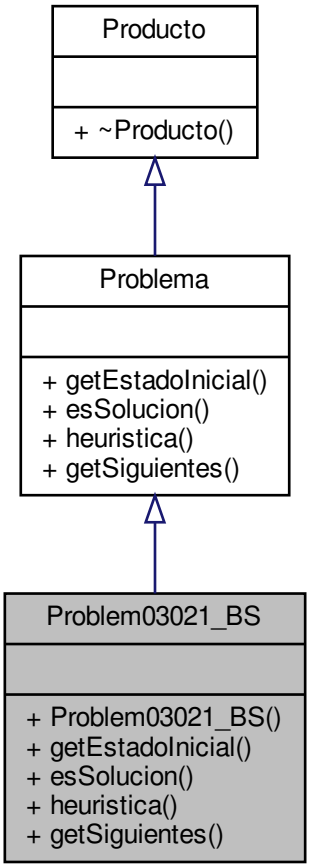
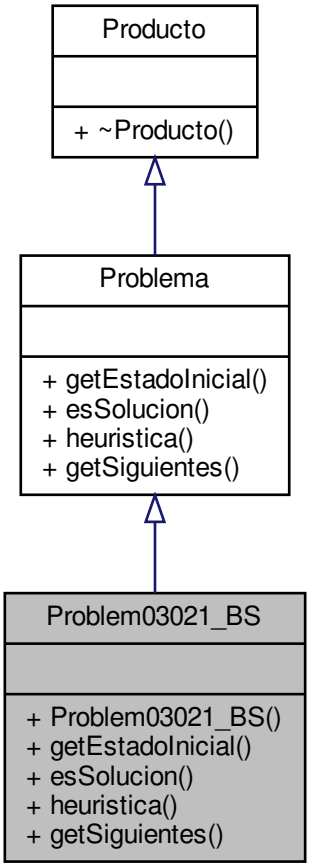


Diagrama de colaboración para Problem03021_BS:



Métodos públicos

- Problem03021_BS ()

  Constructor por omisión, inicializa puntero a State03021_BS con una instancia nueva de State03021_BS.
- Estado ∗ getEstadoInicial ()

  Método que retorna el estado contenido en el problema como atributo.
- int esSolucion (Estado ∗)

  Método que verifica si un estado del problema es solución.
- int heuristica (Estado ∗)

  Método que verifica la heurística de un estado del problema.
- Lista ∗ getSiguientes (Estado ∗)

  Método que verifica y retorna los siguientes estados posibles de un estado en particular contenidos en una lista.

5.15.1. Documentación del constructor y destructor

5.15.1.1. Problem03021_BS()

```
Problem03021_BS::Problem03021_BS ( )
```

Constructor por omisión, inicializa puntero a State03021_BS con una instancia nueva de State03021_BS.

5.15.2. Documentación de las funciones miembro

5.15.2.1. esSolucion()

```
int Problem03021_BS::esSolucion (
            Estado * estado )  [virtual]
```

Método que verifica si un estado del problema es solución.

Parámetros

estado	Recibe un puntero a Estado con el estado del que se quiere saber si está o no solucionado.

Devuelve

int Devuelve un 1 si la matriz del estado está completamente ordenada, en caso contrario, retorna 0.

Implementa Problema.

5.15.2.2. getEstadoInicial()

```
Estado * Problem03021_BS::getEstadoInicial ( )  [virtual]
```

Método que retorna el estado contenido en el problema como atributo.

Devuelve

Estado* Devuelve el atributo estadoInicial del problema.

Implementa Problema.

5.15.2.3. getSiguientes()

```
Lista * Problem03021_BS::getSiguientes (
            Estado * estado )  [virtual]
```

Método que verifica y retorna los siguientes estados posibles de un estado en particular contenidos en una lista.

Parámetros

estado	Recibe un puntero a Estado con el estado del que se quieren saber sus estados siguientes.

Devuelve

Lista* Devuelve un puntero a Lista que contiene los estados siguientes al actual.

Implementa Problema.

5.15.2.4. heuristica()

```
int Problem03021_BS::heuristica (
            Estado * estado )  [virtual]
```

Método que verifica la heurística de un estado del problema.

Parámetros

estado	Recibe un puntero a Estado con el estado del que se quiere saber su heurística.

Devuelve

int Devuelve un entero que representa la cantidad de ordenaciones mínimas que se deben realizar.

Implementa Problema.

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

- Problem03021_BS.h
- Problem03021_BS.cpp

5.16. Referencia de la Clase Problem03021Factory

```
#include <Problem03021Factory.h>
```

Diagrama de herencias de Problem03021Factory

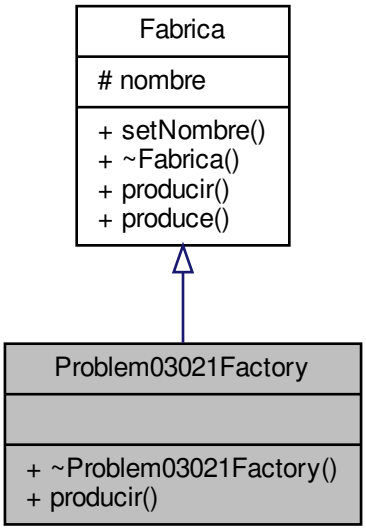
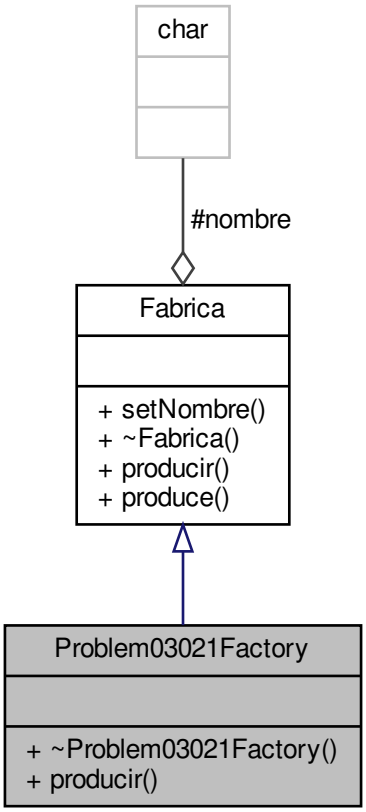


Diagrama de colaboración para Problem03021Factory:



Métodos públicos

- ∼Problem03021Factory ()
- Problem03021_BS ∗ producir ()

  Método que "produce"/crea y retorna una instancia de tipo Problem03021_BS (problema).

Otros miembros heredados

5.16.1. Documentación del constructor y destructor

5.16.1.1. ∼Problem03021Factory()

```
Problem03021Factory::~Problem03021Factory ( )
```

5.16.2. Documentación de las funciones miembro

5.16.2.1. producir()

```
Problem03021_BS * Problem03021Factory::producir ( )  [virtual]
```

Método que "produce"/crea y retorna una instancia de tipo Problem03021_BS (problema).

Devuelve

Problem03021_BS* Devuelve una instancia de tipo Problem03021_BS.

Implementa Fabrica.

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

- Problem03021Factory.h
- Problem03021Factory.cpp

5.17. Referencia de la Clase Problema

```
#include <Problema.h>
```

Diagrama de herencias de Problema

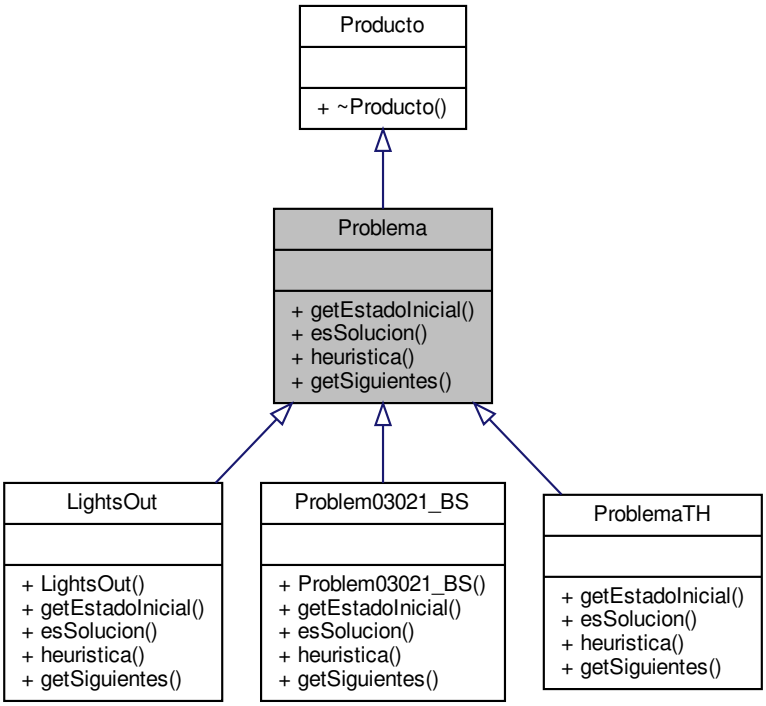
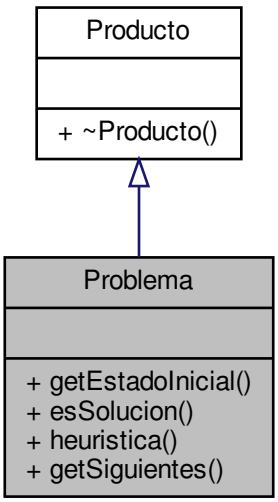


Diagrama de colaboración para Problema:



Métodos públicos

- virtual Estado * getEstadoInicial ()=0
- virtual int esSolucion (Estado *)=0
- virtual int heuristica (Estado *)=0
- virtual Lista * getSiguientes (Estado *)=0

5.17.1. Documentación de las funciones miembro

5.17.1.1. esSolucion()

```
virtual int Problema::esSolucion (
            Estado *  )  [pure virtual]
```

Implementado en ProblemaTH, LightsOut y Problem03021_BS.

5.17.1.2. getEstadoInicial()

```
virtual Estado* Problema::getEstadoInicial ( )  [pure virtual]
```

Implementado en ProblemaTH, Problem03021_BS y LightsOut.

5.17.1.3. getSiguientes()

```
virtual Lista* Problema::getSiguientes (
            Estado *  )  [pure virtual]
```

Implementado en ProblemaTH, LightsOut y Problem03021_BS.

5.17.1.4. heuristica()

```
virtual int Problema::heuristica (
            Estado *  )  [pure virtual]
```

Implementado en ProblemaTH, LightsOut y Problem03021_BS.

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

- Problema.h

5.18. Referencia de la Clase ProblemaTH

Clase derivada de Problema, representa problema Torres de Hanoi.

```
#include <ProblemaTH.h>
```

Diagrama de herencias de ProblemaTH

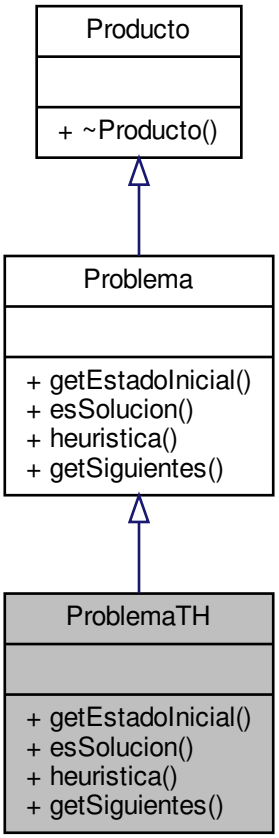
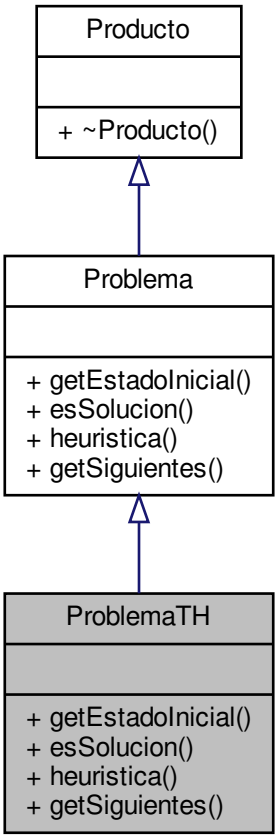


Diagrama de colaboración para ProblemaTH:



Métodos públicos

- Estado * getEstadoInicial ()

  Obtiene el estado inicial EstadoTH del problema.
- int esSolucion (Estado *e)

  Determina si un estado es solucion.
- int heuristica (Estado *e)

  Determina la heuristica de un estado.
- Lista * getSiguientes (Estado *e)

  Determina todos los posibles estados siguientes apartir de un estado.

5.18.1. Descripción detallada

Clase derivada de Problema, representa problema Torres de Hanoi.

5.18.2. Documentación de las funciones miembro

5.18.2.1. esSolucion()

```
int ProblemaTH::esSolucion (
            Estado * e )  [virtual]
```

Determina si un estado es solucion.

Parámetros

e	Puntero a Estado

Devuelve

True Solo si e apunta a un EstadoTH y todas las posiciones en la ultima torre(columna 2) poseen un valor de fila+1

Implementa Problema.

5.18.2.2. getEstadoInicial()

```
Estado * ProblemaTH::getEstadoInicial ( )  [virtual]
```

Obtiene el estado inicial EstadoTH del problema.

Devuelve

Puntero a Estado inicial

Implementa Problema.

5.18.2.3. getSiguientes()

```
Lista * ProblemaTH::getSiguientes (
            Estado * e )  [virtual]
```

Determina todos los posibles estados siguientes apartir de un estado.

Parámetros

e	Puntero a Estado

Devuelve

Puntero a Lista que contiene los estados siguientes a ∗e

Implementa Problema.

5.18.2.4. heuristica()

```
int ProblemaTH::heuristica (
            Estado * e )  [virtual]
```

Determina la heuristica de un estado.

Parámetros

e	Puntero a Estado

Devuelve

Valor de la heuristica

Implementa Problema.

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

- ProblemaTH.h
- ProblemaTH.cpp

5.19. Referencia de la Clase Producto

```
#include <Producto.h>
```

Diagrama de herencias de Producto

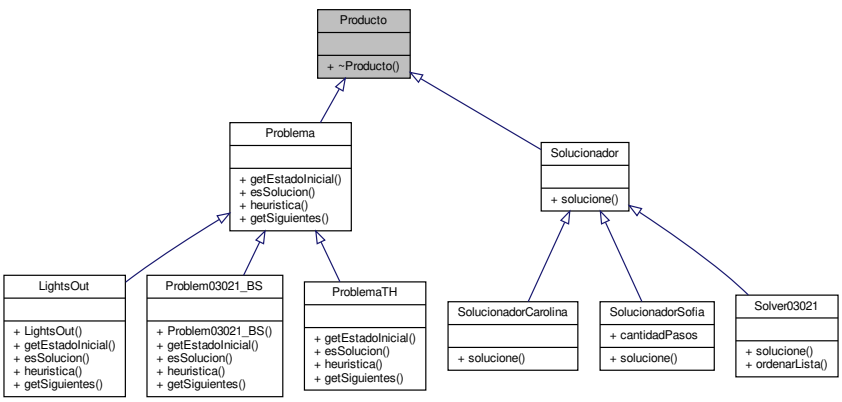
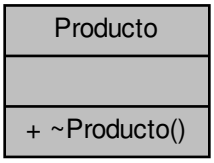


Diagrama de colaboración para Producto:



Métodos públicos

- virtual ∼Producto ()

5.19.1. Documentación del constructor y destructor

5.19.1.1. ∼Producto()

```
virtual Producto::~Producto ( )  [inline], [virtual]
```

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

- Producto.h

5.20. Referencia de la Clase Registro

```
#include <Registro.h>
```

Diagrama de colaboración para Registro:

Métodos públicos

- Registro ()
- ∼Registro ()
- Fabrica ∗ getFabrica (const char ∗)

5.20.1. Documentación del constructor y destructor

5.20.1.1. Registro()

```
Registro::Registro ( )
```

5.20.1.2. ∼Registro()

```
Registro::~Registro ( )
```

5.20.2. Documentación de las funciones miembro

5.20.2.1. getFabrica()

```
Fabrica * Registro::getFabrica (
            const char * nombre )
```

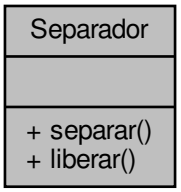
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- Registro.h
- Registro.cpp

5.21. Referencia de la Clase Separador

```
#include <Separador.h>
```

Diagrama de colaboración para Separador:



Métodos públicos

- char ** separar (const char *hilera)

  Separa una hilera de caracteres.
- void liberar (char **p)

  Libera la memoria ocupada por un array de hileras de chars.

5.21.1. Descripción detallada

Esta clase tiene métodos para separar hileras de caracteres y liberar la memoria ocupada por una hilera separada

5.21.2. Documentación de las funciones miembro

5.21.2.1. liberar()

```
void Separador::liberar (
            char ** p )
```

Libera la memoria ocupada por un array de hileras de chars.

Parámetros

p	Array de hileras de chars cuya memoria se libera

5.21.2.2. separar()

```
char ** Separador::separar (
            const char * hilera )
```

Separa una hilera de caracteres.

Separa un array de chars por grupos de espacios o tabulaciones

Parámetros

hilera	puntero al primer char de la hilera que se separa

Devuelve

array de punteros a char, cada uno apunta a una de las "palabras" de la hilera original, el último tiene una hilera formada por "\0"

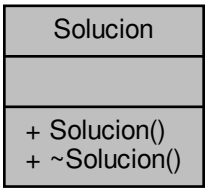
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- Separador.h
- Separador.cpp

5.22. Referencia de la Clase Solucion

`#include <Solucion.h>`

Diagrama de colaboración para Solucion:



Métodos públicos

- Solucion (Lista ∗)
- ∼Solucion ()

Amigas

- ostream & operator<< (ostream &salida, Solucion ∗solucion)

5.22.1. Documentación del constructor y destructor

5.22.1.1. Solucion()

```
Solucion::Solucion (
            Lista * listaPtr )
```

5.22.1.2. ∼Solucion()

```
Solucion::~Solucion ( )
```

5.22.2. Documentación de las funciones relacionadas y clases amigas

5.22.2.1. operator<<

```
ostream& operator<< (
            ostream & salida,
            Solucion * solucion )  [friend]
```

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

- Solucion.h
- Solucion.cpp

5.23. Referencia de la Clase Solucionador

```
#include <Solucionador.h>
```

Diagrama de herencias de Solucionador

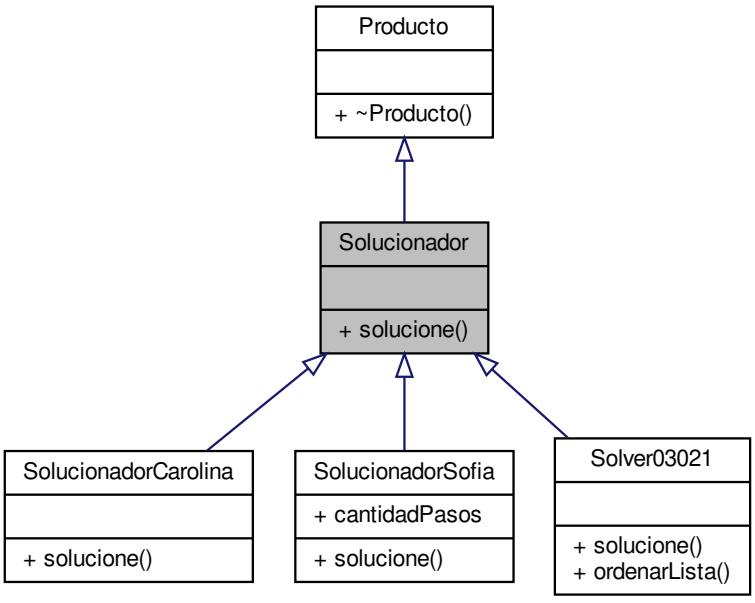
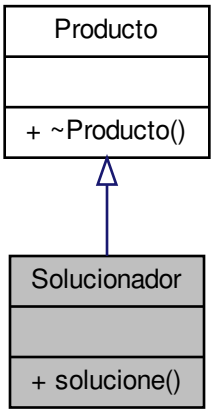


Diagrama de colaboración para Solucionador:



Métodos públicos

- virtual Solucion * solucione (Problema *)=0

5.23.1. Documentación de las funciones miembro

5.23.1.1. solucione()

```
virtual Solucion* Solucionador::solucione (
            Problema *  )  [pure virtual]
```

Implementado en SolucionadorSofia, SolucionadorCarolina y Solver03021.

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

- Solucionador.h

5.24. Referencia de la Clase SolucionadorCarolina

Solucionador de Carolina, derivado de Solucionador.

`#include <SolucionadorCarolina.h>`

Diagrama de herencias de SolucionadorCarolina

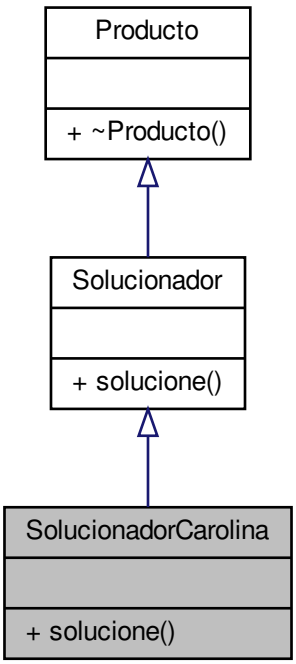
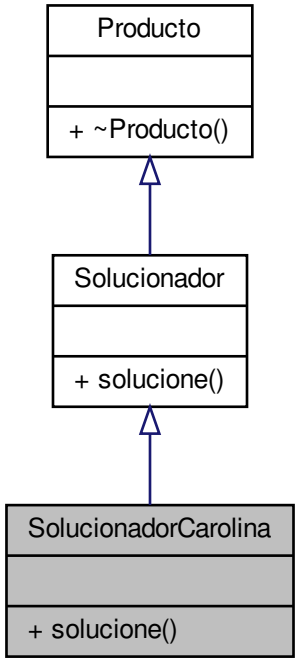


Diagrama de colaboración para SolucionadorCarolina:



Métodos públicos

- Solucion * solucione (Problema *problem)

  Encuentra la solución de un problema.

5.24.1. Descripción detallada

Solucionador de Carolina, derivado de Solucionador.

5.24.2. Documentación de las funciones miembro

5.24.2.1. solucione()

```
Solucion * SolucionadorCarolina::solucione (
            Problema * problem )  [virtual]
```

Encuentra la solución de un problema.

Parámetros

problema	Puntero a problema

Devuelve

Puntero a Solución con la solución del problema

Implementa Solucionador.

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

- SolucionadorCarolina.h
- SolucionadorCarolina.cpp

5.25. Referencia de la Clase SolucionadorSofia

Solucionador de Sofia, clase derivada de Solucionador.

`#include <SolucionadorSofia.h>`

Diagrama de herencias de SolucionadorSofia

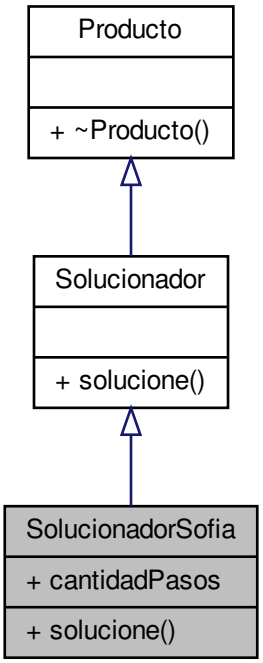
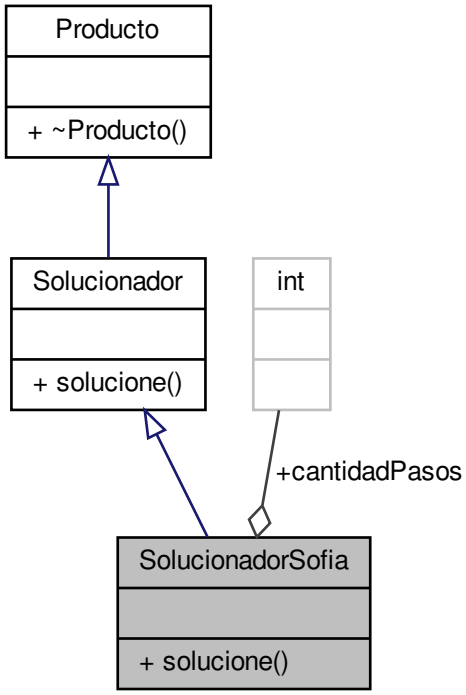


Diagrama de colaboración para SolucionadorSofia:



Métodos públicos

- Solucion * solucione (Problema *problema)

  Soluciona un problema.

Atributos públicos

- int cantidadPasos

  Contador para la cantidad de pasos en los cuales se soluciono el problema.

5.25.1. Descripción detallada

Solucionador de Sofia, clase derivada de Solucionador.

5.25.2. Documentación de las funciones miembro

5.25.2.1. solucione()

```
Solucion * SolucionadorSofia::solucione (
            Problema * problema )  [virtual]
```

Soluciona un problema.

Parámetros

problema	Puntero a problema

Devuelve

Puntero a Solucion con la solucion del problema

Implementa Solucionador.

5.25.3. Documentación de los datos miembro

5.25.3.1. cantidadPasos

```
int SolucionadorSofia::cantidadPasos
```

Contador para la cantidad de pasos en los cuales se soluciono el problema.

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

- SolucionadorSofia.h
- SolucionadorSofia.cpp

5.26. Referencia de la Clase Solver03021

```
#include <Solver03021.h>
```

Diagrama de herencias de Solver03021

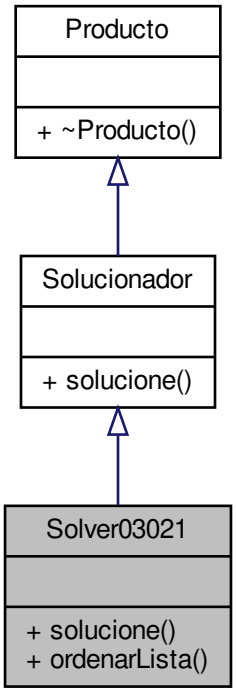
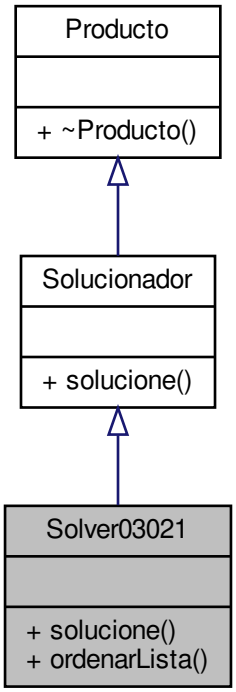


Diagrama de colaboración para Solver03021:



Métodos públicos

- Solucion ∗ solucione (Problema ∗)

  Método que recibe un problema y se encarga de explorarlos con ayuda de la clase EstadoConRuta y el método 'busqueSolucion' con una técnica de implementación de Listas.
- void ordenarLista (Lista ∗, Problema ∗)

  Ordena la lista ascendentemente por la heurística determinada por el problema.

5.26.1. Documentación de las funciones miembro

5.26.1.1. ordenarLista()

```
void Solver03021::ordenarLista (
            Lista * lista,
            Problema * problema )
```

Ordena la lista ascendentemente por la heurística determinada por el problema.

Parámetros

lista	Recibe una lista por puntero, representa la lista que se quiere ordenar.
problema	Recibe un Problema por puntero, representa el problema que se quiere solucionar.

5.26.1.2. solucione()

```
Solucion * Solver03021::solucione (
            Problema * problema )  [virtual]
```

Método que recibe un problema y se encarga de explorarlos con ayuda de la clase EstadoConRuta y el método 'busqueSolucion' con una técnica de implementación de Listas.

Busca la solución óptima de cada problema de la siguiente manera: Posee una listaDeEstados que sirve temporalmente para inicializar los estadosConRuta iniciales. Posee una frontera (COLA/FIFO) que representa los estados que deben verificarse si son solución y además que faltan de expandirse para obtener los siguientes. Posee una lista de visitados para evitar agregar duplicados a la frontera.

Llama al método auxiliar 'busqueSolucion' para que se encargue de manejar (por ciclo while) las listas en la exploración. De su manejo es prioridad la frontera ya que si encuentra solución, será el primer elemento de ella.

Parámetros

problema	Recibe un puntero a Problema que se quiere solucionar.

Devuelve

Solucion* o 0 Devuelve una instancia de Solución que contiene la lista de estados desde el inicio hasta llegar a la solución del problema, esto en caso de haber hallado la solución. Retorna 0 en caso contrario.

Implementa Solucionador.

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

- Solver03021.h
- Solver03021.cpp

5.27. Referencia de la Clase Solver03021Factory

```
#include <Solver03021Factory.h>
```

Diagrama de herencias de Solver03021Factory

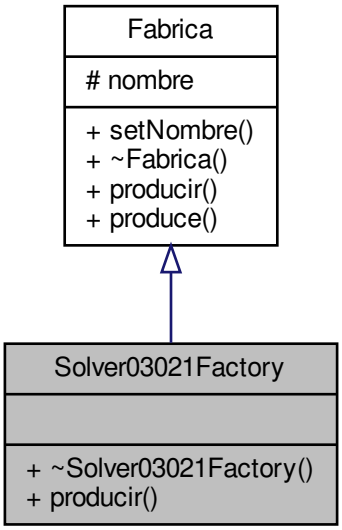
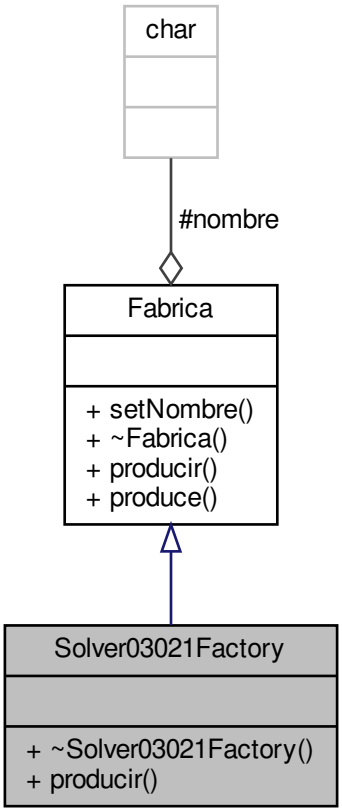


Diagrama de colaboración para Solver03021Factory:



Métodos públicos

- ∼Solver03021Factory ()
- Solver03021 ∗ producir ()

  Método que "produce"/crea y retorna una instancia de tipo Solver03021 (solucionador).

Otros miembros heredados

5.27.1. Documentación del constructor y destructor

5.27.1.1. ∼Solver03021Factory()

```
Solver03021Factory::~Solver03021Factory ( )
```

5.27.2. Documentación de las funciones miembro

5.27.2.1. producir()

```
Solver03021 * Solver03021Factory::producir ( )  [virtual]
```

Método que "produce"/crea y retorna una instancia de tipo Solver03021 (solucionador).

Devuelve

Solver03021* Devuelve una instancia de tipo Solver03021.

Implementa Fabrica.

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

- Solver03021Factory.h
- Solver03021Factory.cpp

5.28. Referencia de la Clase State03021_BS

```
#include <State03021_BS.h>
```

Diagrama de herencias de State03021_BS

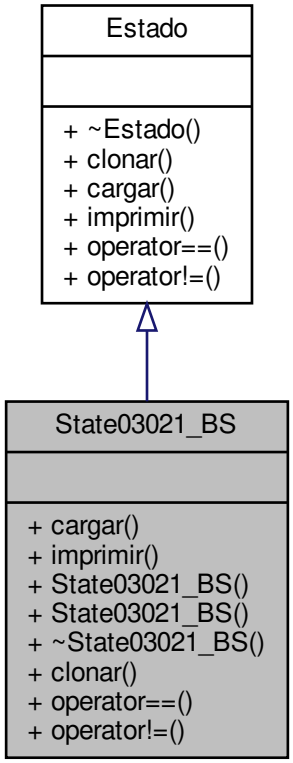
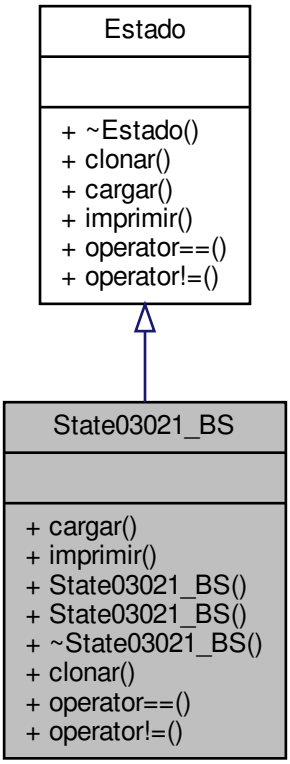


Diagrama de colaboración para State03021_BS:



Métodos públicos

- istream & cargar (istream &)

  Método que invoca la sobrecarga de >> de la matriz.
- ostream & imprimir (ostream &)

  Método que que invoca la sobrecarga de << de la matriz.
- State03021_BS ()

  Constructor por omisión, inicializa puntero a matriz con una instancia nueva de Matriz03021_BS.
- State03021_BS (Matriz03021_BS &)

  Constructor de copia, inicializa puntero a matriz con una instancia nueva de Matriz03021_BS clonada con la que pasan por referencia en parámetro.
- ∼State03021_BS ()

  Destructor, se encarga de la liberación de la memoria.
- State03021_BS ∗ clonar ()

  Método que invoca al constructor de copia del estado con la matriz propia y retorna su resultado.
- int operator== (Estado ∗)

  Sobrecarga del operador ==, realiza un dynamic casting para realizar la verificación de igualdad entre las matrices.
- int operator!= (Estado ∗)

  Sobrecarga del operador !=, invoca la sobrecarga de == y la retorna su negación.

Amigas

- class Problem03021_BS

5.28.1. Documentación del constructor y destructor

5.28.1.1. State03021_BS() [1/2]

```
State03021_BS::State03021_BS ( )
```

Constructor por omisión, inicializa puntero a matriz con una instancia nueva de Matriz03021_BS.

5.28.1.2. State03021_BS() [2/2]

```
State03021_BS::State03021_BS (
            Matriz03021_BS & otra )
```

Constructor de copia, inicializa puntero a matriz con una instancia nueva de Matriz03021_BS clonada con la que pasan por referencia en parámetro.

Parámetros

otra	Recibe un objeto Matriz03021_BS por referencia.

5.28.1.3. ~State03021_BS()

```
State03021_BS::~State03021_BS ( )
```

Destructor, se encarga de la liberación de la memoria.

5.28.2. Documentación de las funciones miembro

5.28.2.1. cargar()

```
istream & State03021_BS::cargar (
            istream & entrada )  [virtual]
```

Método que invoca la sobrecarga de >> de la matriz.

Parámetros

entrada	Recibe por referencia el flujo de entrada.

Devuelve

istream& Devuelve un flujo de entrada con lo que generó el método en su ejecución.

Implementa Estado.

5.28.2.2. clonar()

```
State03021_BS * State03021_BS::clonar ( )  [virtual]
```

Método que invoca al constructor de copia del estado con la matriz propia y retorna su resultado.

Devuelve

State03021_BS* Devuelve un nuevo objeto tipo State03021_BS creado con las carácteristicas del estado propio.

Implementa Estado.

5.28.2.3. imprimir()

```
ostream & State03021_BS::imprimir (
            ostream & salida )  [virtual]
```

Método que que invoca la sobrecarga de << de la matriz.

Parámetros

salida	Recibe por referencia el flujo de salida.

Devuelve

ostream& Devuelve un flujo de salida con lo que generó el método en su ejecución.

Implementa Estado.

5.28.2.4. operatori=()

```
int State03021_BS::operator!= (
            Estado * estado )  [virtual]
```

Sobrecarga del operador !=, invoca la sobrecarga de == y la retorna su negación.

Parámetros

estado	Recibe un puntero a Estado con el que se quiere comparar.

Devuelve

int Devuelve 1 si al menos una de las entradas de la matriz es distinta; de manera contraria retorna 0.

Implementa Estado.

5.28.2.5. operator==()

```
int State03021_BS::operator== (
            Estado * estado )  [virtual]
```

Sobrecarga del operador ==, realiza un dynamic casting para realizar la verificación de igualdad entre las matrices.

Parámetros

estado	Recibe un puntero a Estado con el que se quiere comparar.

Devuelve

int Devuelve 0 si al menos una de las entradas de la matriz es distinta; de manera contraria retorna 1.

Implementa Estado.

5.28.3. Documentación de las funciones relacionadas y clases amigas

5.28.3.1. Problem03021_BS

```
friend class Problem03021_BS  [friend]
```

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

- State03021_BS.h
- State03021_BS.cpp

Capítulo 6

Documentación de archivos

6.1. Referencia del Archivo Estado.h

`#include <iostream>`
Dependencia gráfica adjunta para Estado.h:

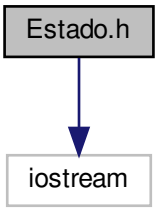
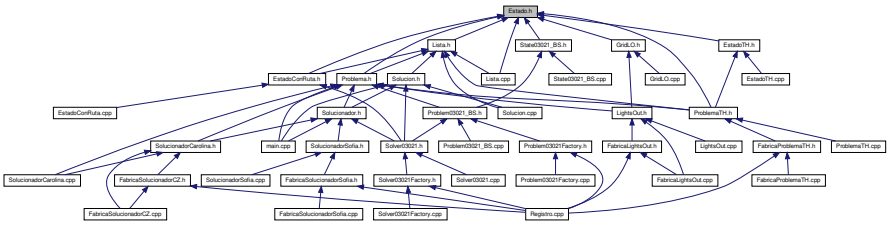


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

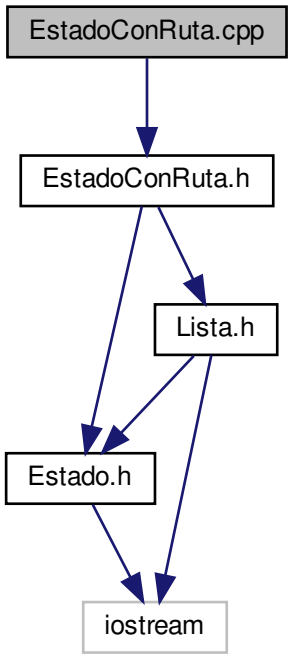
- class Estado

6.2. Referencia del Archivo EstadoConRuta.cpp

La Clase EstadoConRuta es una clase auxiliar del solucionador, se encarga de contener un estado su el camino por el cual se llega hasta él.

```
#include "EstadoConRuta.h"
```

Dependencia gráfica adjunta para EstadoConRuta.cpp:



6.2.1. Descripción detallada

La Clase EstadoConRuta es una clase auxiliar del solucionador, se encarga de contener un estado su el camino por el cual se llega hasta él.

Autor

Fabián Orozco, Sofía Shum y Carolina Zamora.

Versión

1.0

Fecha

2021-09-07

Copyright

Copyright (c) 2021

6.3. Referencia del Archivo EstadoConRuta.h

```
#include "Estado.h"
#include "Lista.h"
```

Dependencia gráfica adjunta para EstadoConRuta.h:

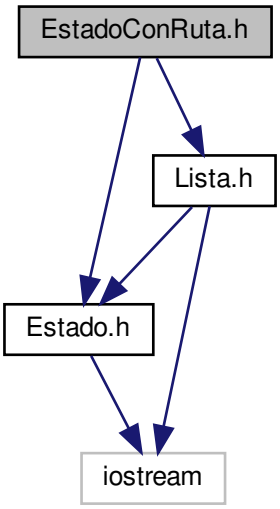
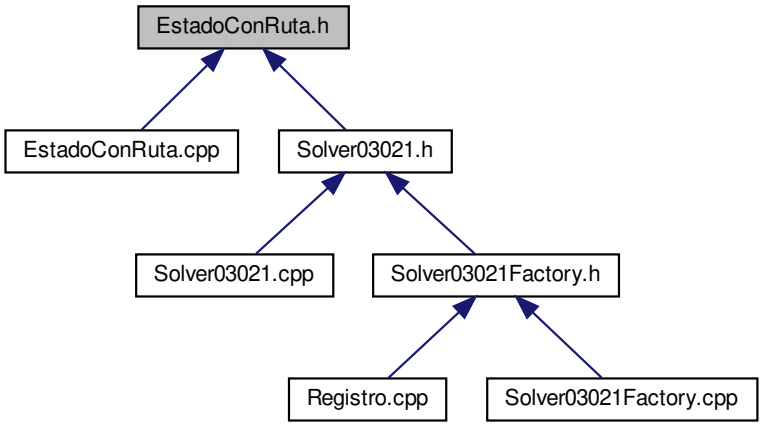


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

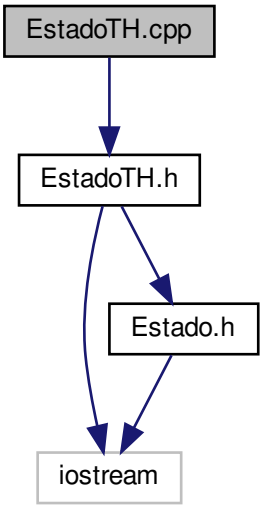


Clases

- class EstadoConRuta

6.4. Referencia del Archivo EstadoTH.cpp

```
#include "EstadoTH.h"
```
Dependencia gráfica adjunta para EstadoTH.cpp:



6.5. Referencia del Archivo EstadoTH.h

```
#include <iostream>
#include "Estado.h"
```
Dependencia gráfica adjunta para EstadoTH.h:

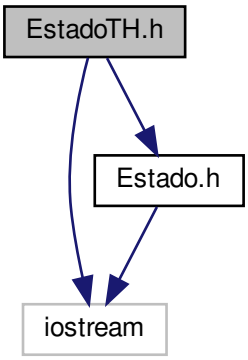
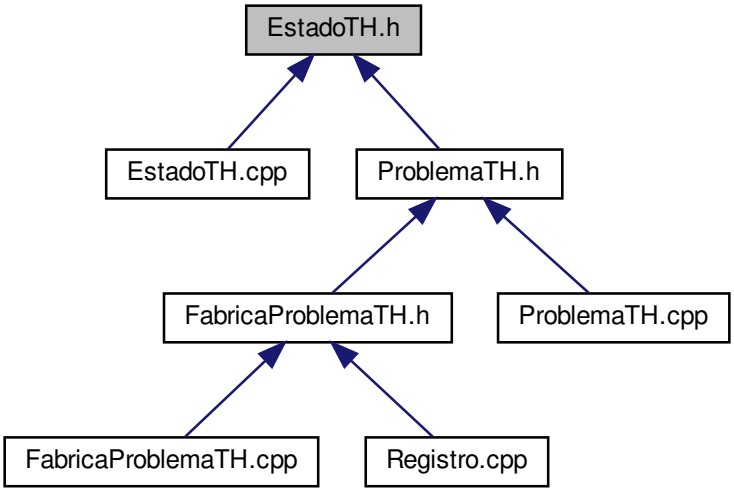


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class EstadoTH

  Clase derivada de Estado, representa tres torres con platillos (matrix) del problema Torres de Hanoi.

defines

- #define QPLATES 4
- #define QTOWERS 3
- #define LINESIZE 25

6.5.1. Documentación de los 'defines'

6.5.1.1. LINESIZE

```
#define LINESIZE 25
```

6.5.1.2. QPLATES

```
#define QPLATES 4
```

6.5.1.3. QTOWERS

```
#define QTOWERS 3
```

6.6. Referencia del Archivo Fabrica.h

```
#include "Producto.h"
#include <cstring>
```

Dependencia gráfica adjunta para Fabrica.h:

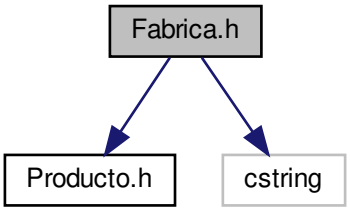
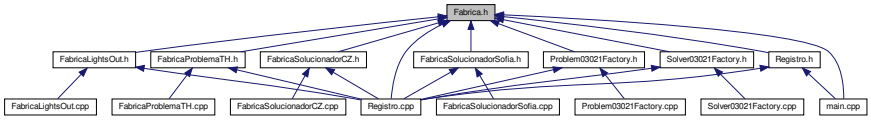


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class Fabrica

defines

- #define NOMBRE_MAX_SIZE 100

6.6.1. Documentación de los 'defines'

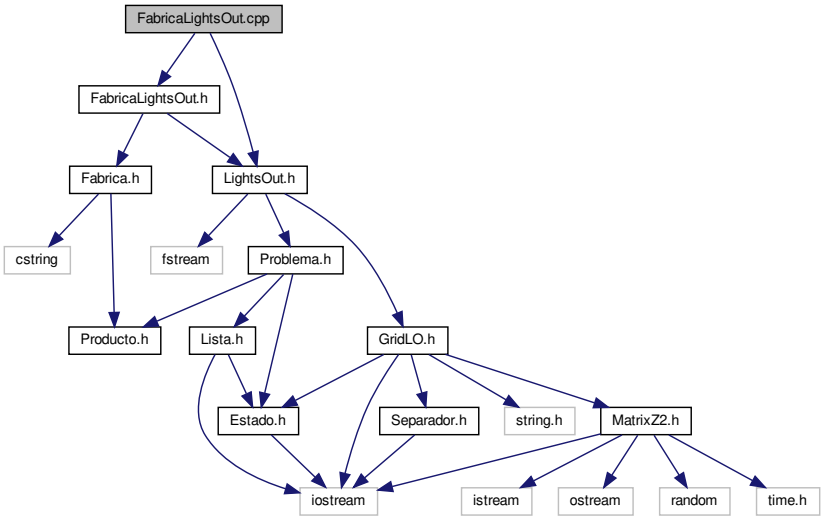
6.6.1.1. NOMBRE_MAX_SIZE

```
#define NOMBRE_MAX_SIZE 100
```

6.7. Referencia del Archivo FabricaLightsOut.cpp

```
#include "FabricaLightsOut.h"
#include "LightsOut.h"
```

Dependencia gráfica adjunta para FabricaLightsOut.cpp:



6.8. Referencia del Archivo FabricaLightsOut.h

```
#include "Fabrica.h"
#include "LightsOut.h"
```

Dependencia gráfica adjunta para FabricaLightsOut.h:

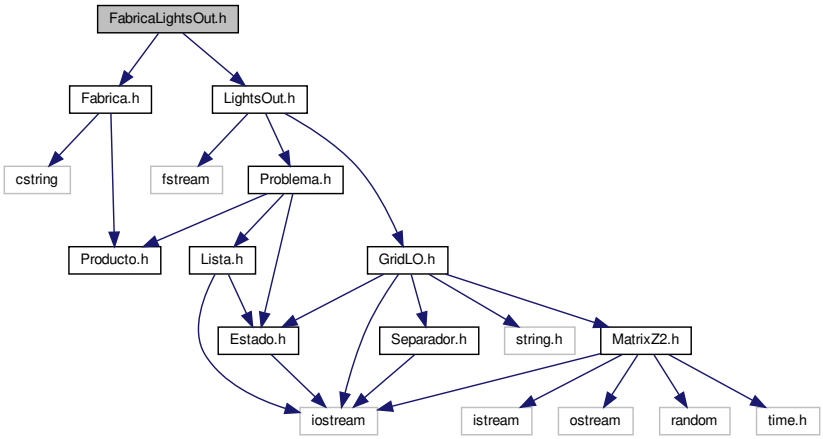
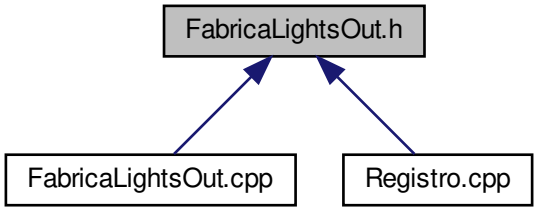


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

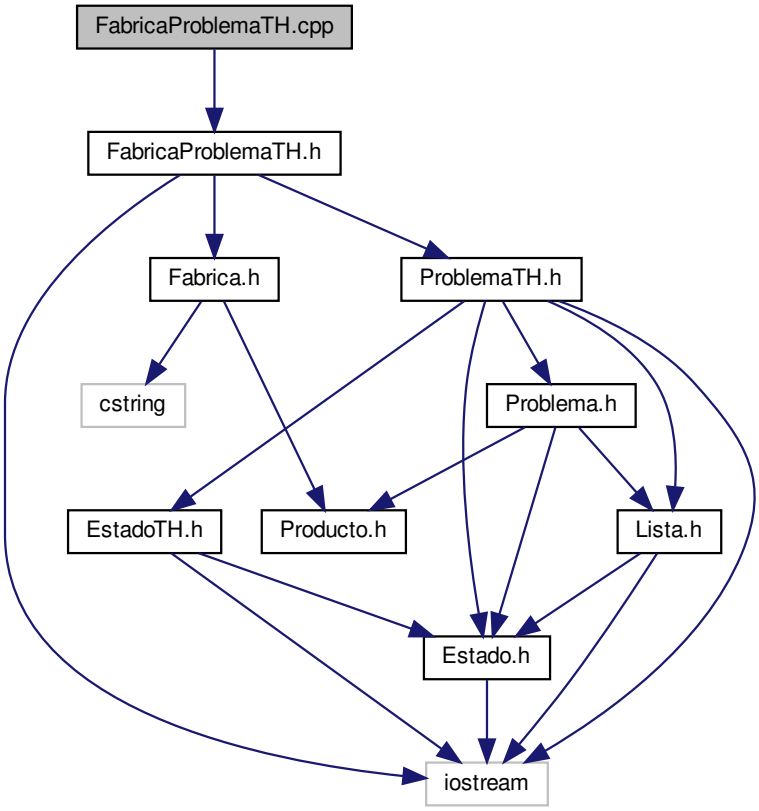
- class FabricaLightsOut

  Clase derivada de Fabrica, para producir un puntero a LightsOut.

6.9. Referencia del Archivo FabricaProblemaTH.cpp

```
#include "FabricaProblemaTH.h"
```

Dependencia gráfica adjunta para FabricaProblemaTH.cpp:



6.10. Referencia del Archivo FabricaProblemaTH.h

```
#include <iostream>
#include "Fabrica.h"
#include "ProblemaTH.h"
```

Dependencia gráfica adjunta para FabricaProblemaTH.h:

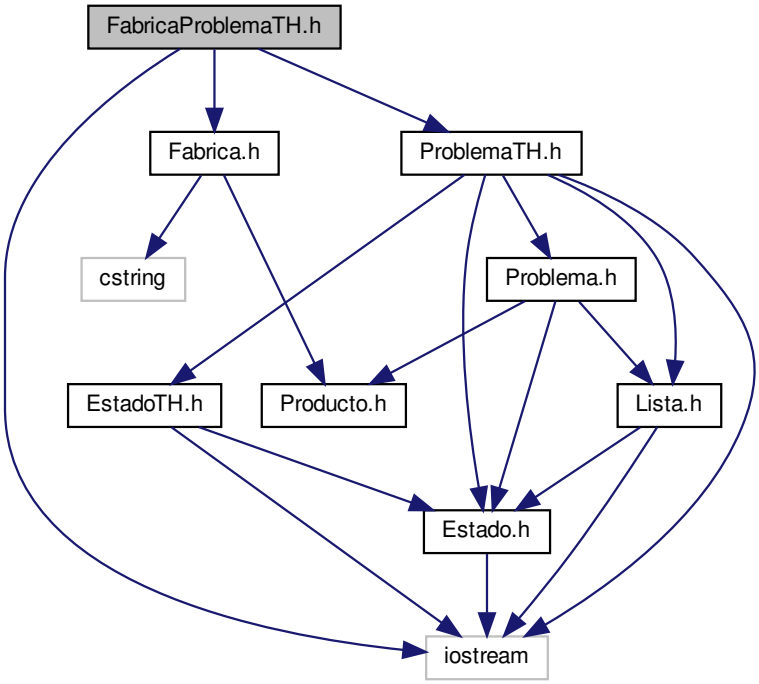
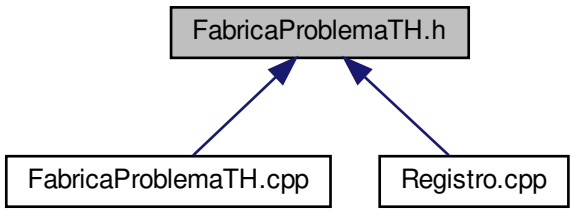


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

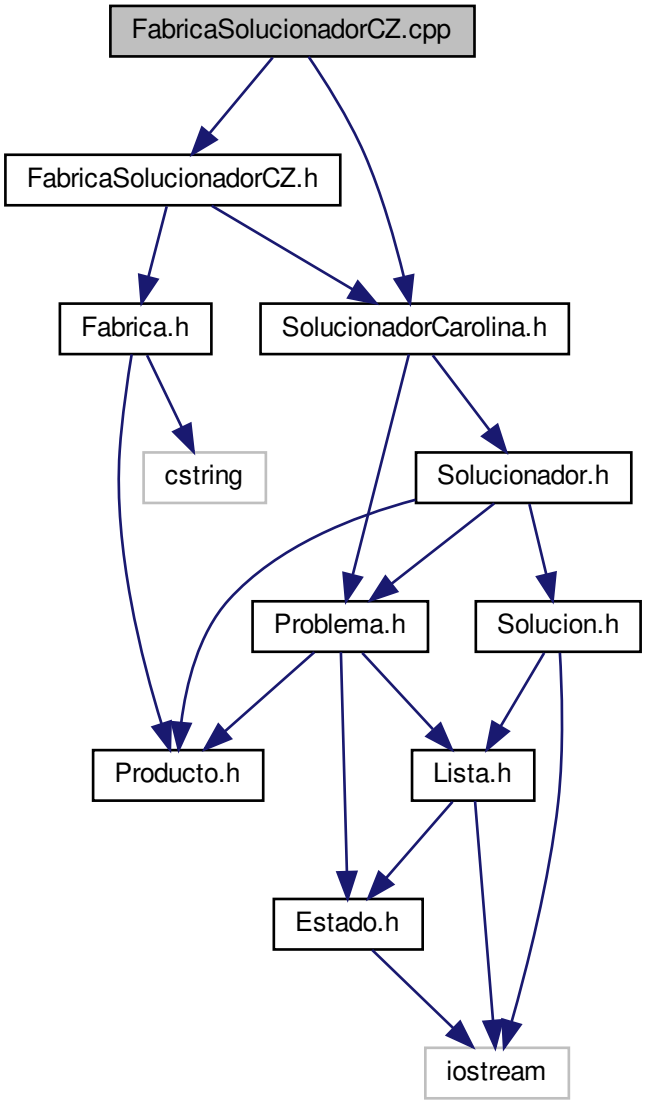
- class FabricaProblemaTH

  Clase derivada de Fabrica, produce un puntero a ProblemaTH.

6.11. Referencia del Archivo FabricaSolucionadorCZ.cpp

```
#include "FabricaSolucionadorCZ.h"
#include "SolucionadorCarolina.h"
```

Dependencia gráfica adjunta para FabricaSolucionadorCZ.cpp:



6.12. Referencia del Archivo FabricaSolucionadorCZ.h

```
#include "Fabrica.h"
#include "SolucionadorCarolina.h"
```

Dependencia gráfica adjunta para FabricaSolucionadorCZ.h:

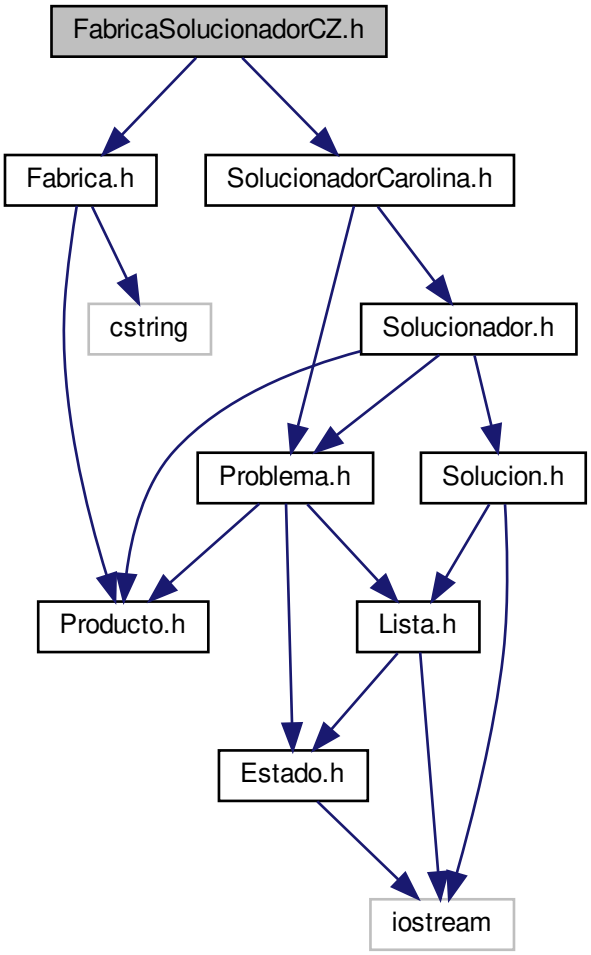
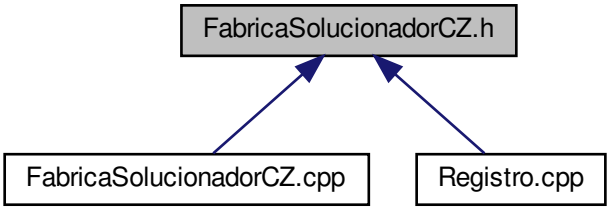


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

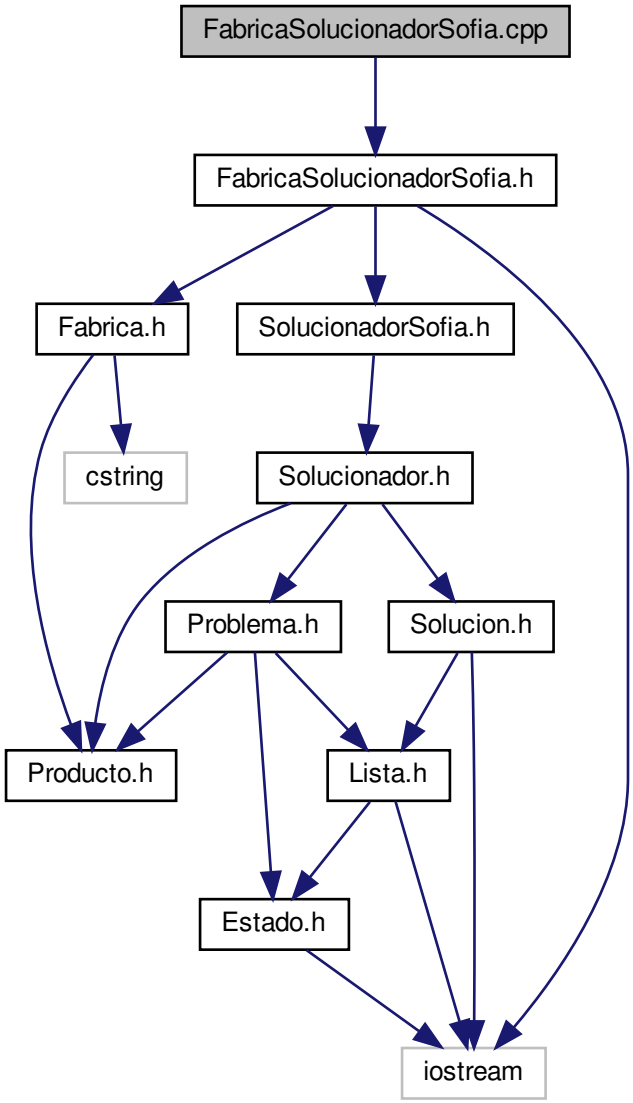
- class FabricaSolucionadorCZ

  Clase derivada de Fabrica, para producir un puntero a SolucionadorCarolina.

6.13. Referencia del Archivo FabricaSolucionadorSofia.cpp

```
#include "FabricaSolucionadorSofia.h"
```

Dependencia gráfica adjunta para FabricaSolucionadorSofia.cpp:



6.14. Referencia del Archivo FabricaSolucionadorSofia.h

```
#include "Fabrica.h"
#include "SolucionadorSofia.h"
#include <iostream>
```

Dependencia gráfica adjunta para FabricaSolucionadorSofia.h:

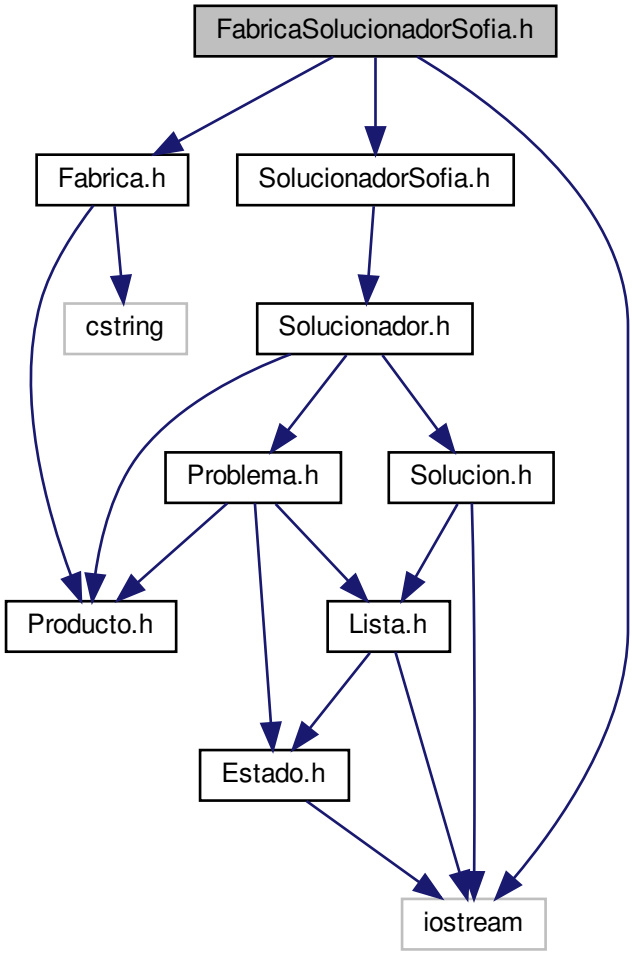
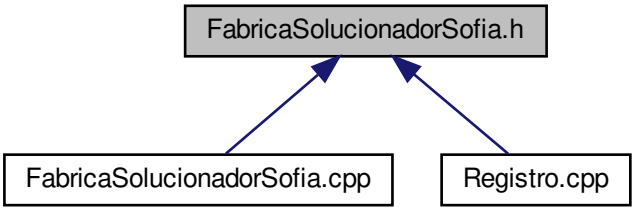


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

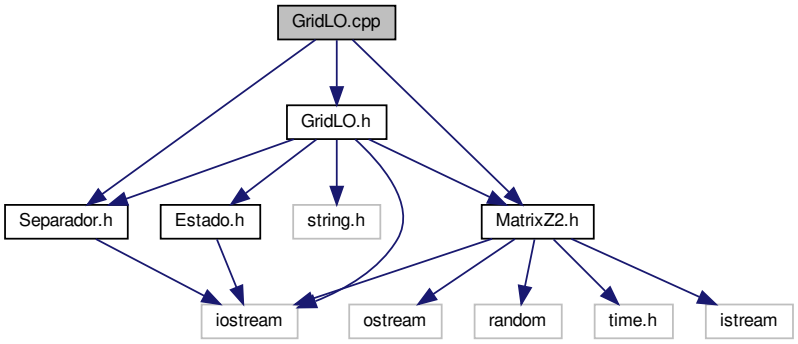
- class FabricaSolucionadorSofia

  Clase derivada de Fabrica, produce un puntero a SolucionadorSofia.

6.15. Referencia del Archivo GridLO.cpp

```
#include "GridLO.h"
#include "MatrixZ2.h"
#include "Separador.h"
```

Dependencia gráfica adjunta para GridLO.cpp:



6.16. Referencia del Archivo GridLO.h

```
#include <iostream>
#include <string.h>
#include "Estado.h"
```

```
#include "MatrixZ2.h"
#include "Separador.h"
```

Dependencia gráfica adjunta para GridLO.h:

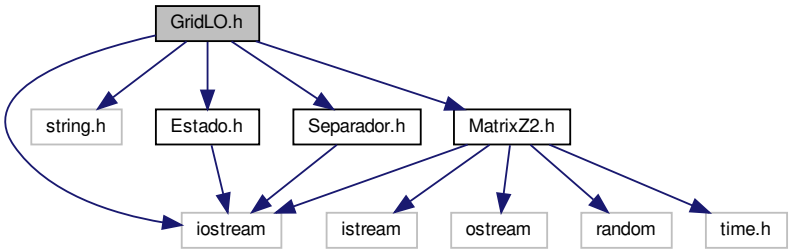
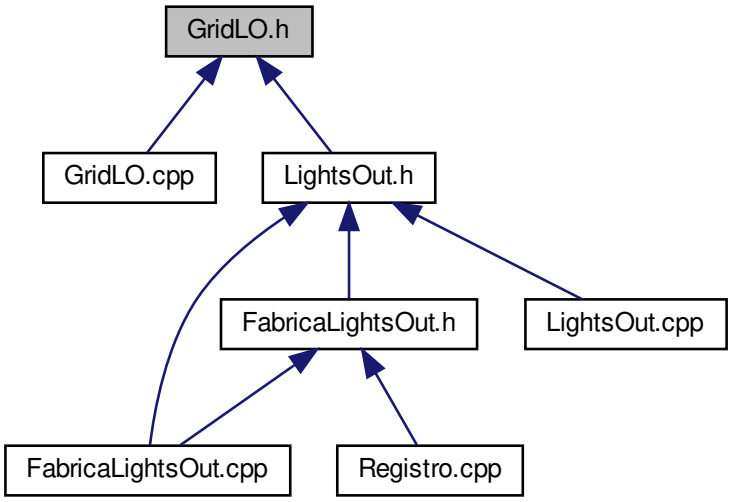


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class GridLO

  Clase derivada de Estado, representa una cuadrícula (grid) del problema LightsOut.

defines

- #define STREAMSIZE_LO 32
- #define GRIDSIZE_LO 4

6.16.1. Documentación de los 'defines'

6.16.1.1. GRIDSIZE_LO

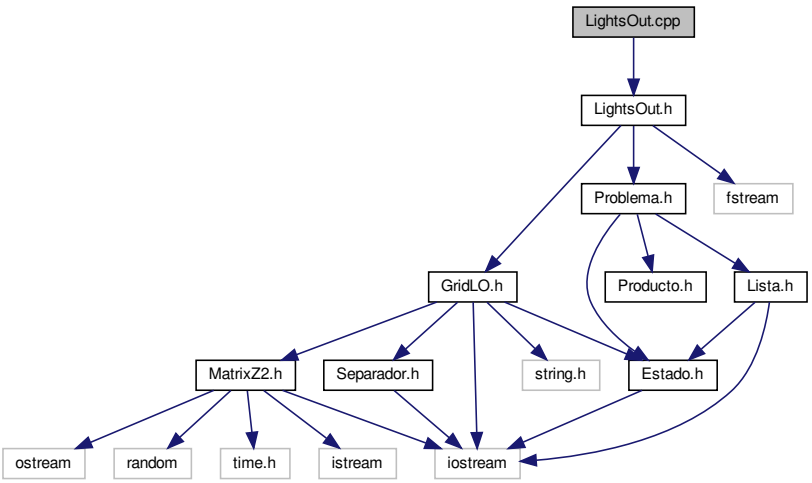
```
#define GRIDSIZE_LO 4
```

6.16.1.2. STREAMSIZE_LO

```
#define STREAMSIZE_LO 32
```

6.17. Referencia del Archivo LightsOut.cpp

```
#include "LightsOut.h"
```
Dependencia gráfica adjunta para LightsOut.cpp:



6.18. Referencia del Archivo LightsOut.h

```
#include "Problema.h"
#include "GridLO.h"
```

```
#include <fstream>
```

Dependencia gráfica adjunta para LightsOut.h:

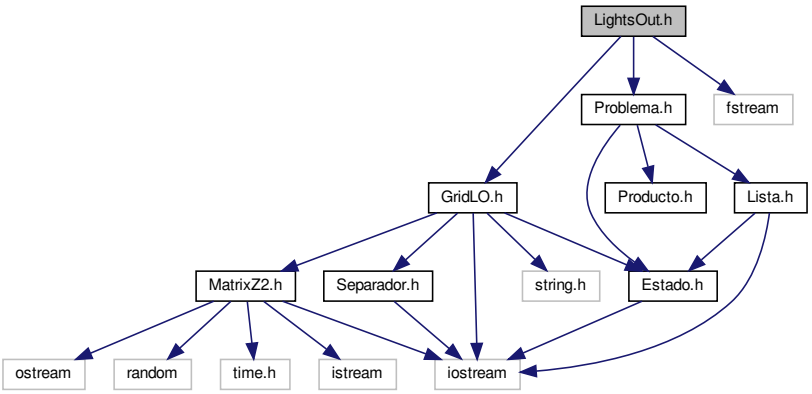
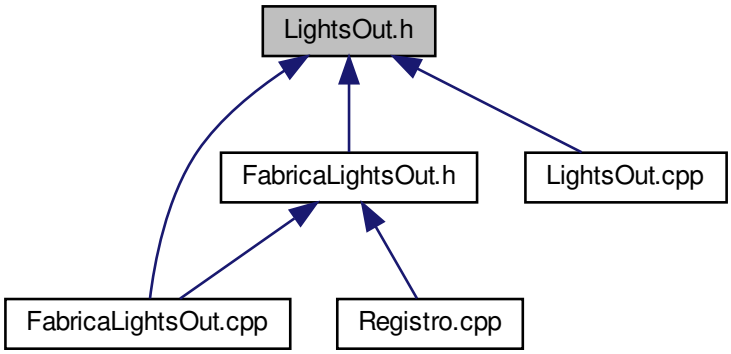


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

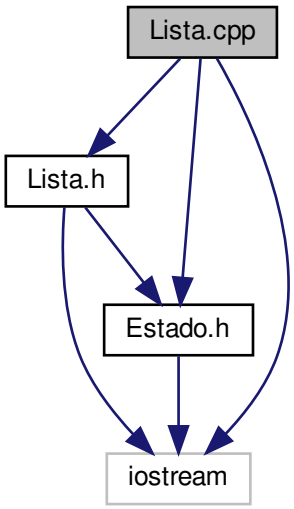
- class LightsOut

  Clase derivada de Problema, representa el problema LightsOut.

6.19. Referencia del Archivo Lista.cpp

```
#include "Lista.h"
#include "Estado.h"
```

```
#include <iostream>
```
Dependencia gráfica adjunta para Lista.cpp:



6.20. Referencia del Archivo Lista.h

```
#include <iostream>
#include "Estado.h"
```
Dependencia gráfica adjunta para Lista.h:

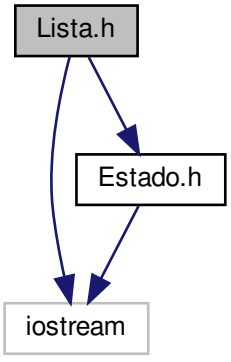
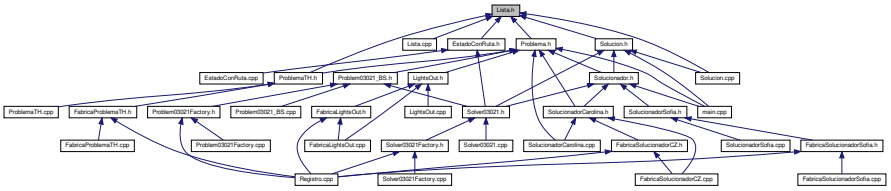


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



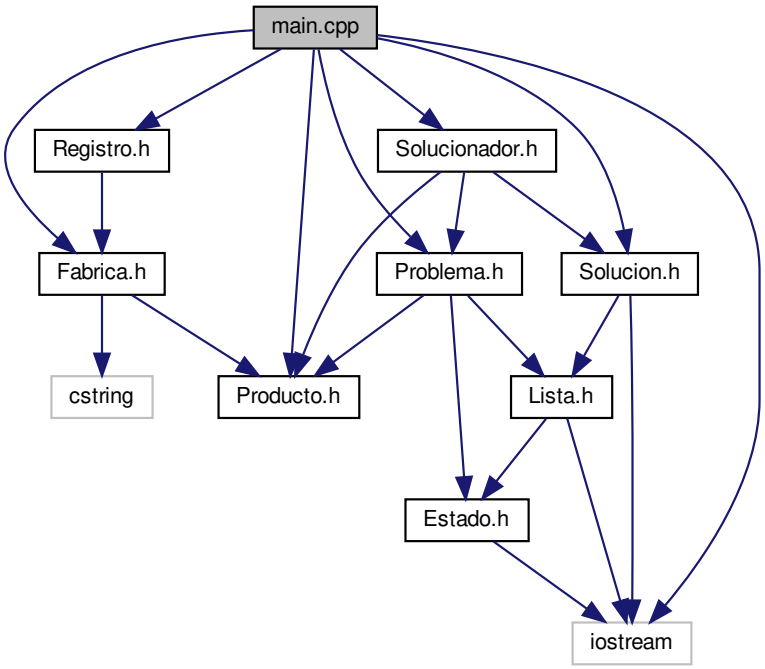
Clases

- class Lista
- class Lista::Iterador

6.21. Referencia del Archivo main.cpp

```
#include "Registro.h"
#include "Fabrica.h"
#include "Producto.h"
#include "Solucionador.h"
#include "Problema.h"
#include "Solucion.h"
#include <iostream>
```

Dependencia gráfica adjunta para main.cpp:



Funciones

- int main (int argc, char **argv)

6.21.1. Documentación de las funciones

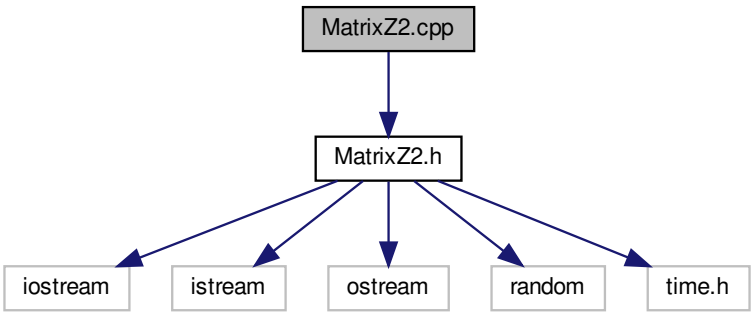
6.21.1.1. main()

```
int main (
            int argc,
            char ** argv )
```

6.22. Referencia del Archivo MatrixZ2.cpp

```
#include "MatrixZ2.h"
```

Dependencia gráfica adjunta para MatrixZ2.cpp:



6.23. Referencia del Archivo MatrixZ2.h

```
#include <iostream>
#include <istream>
#include <ostream>
#include <random>
```

```
#include <time.h>
```

Dependencia gráfica adjunta para MatrixZ2.h:

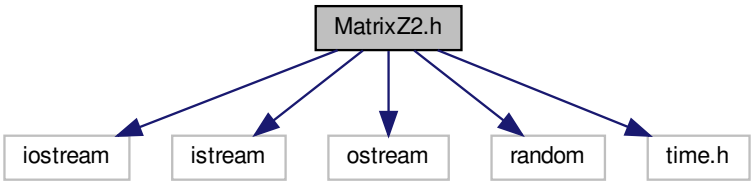
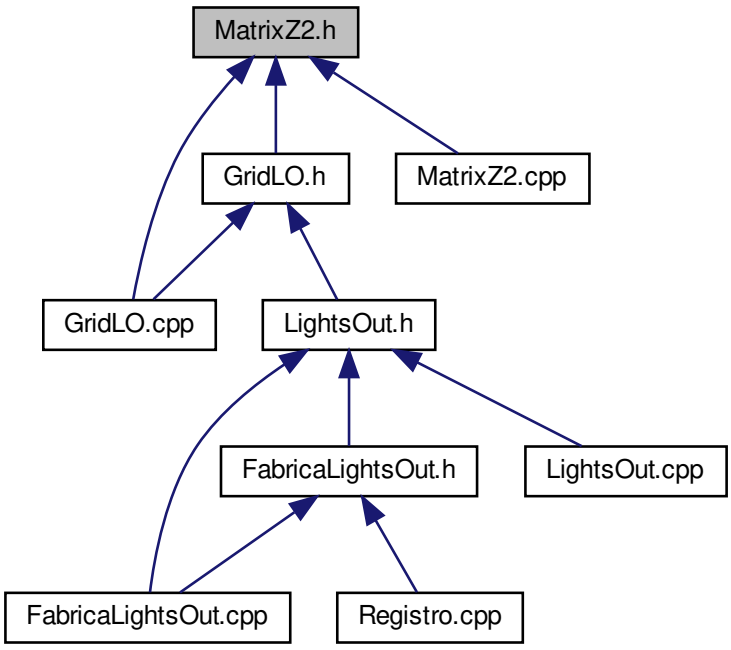


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class MatrixZ2

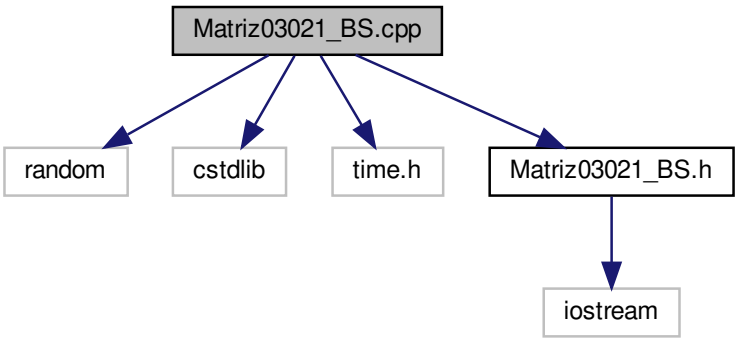
  Esta clase representa una matriz con entradas en el cuerpo Z2 (F2)

6.24. Referencia del Archivo Matriz03021_BS.cpp

La Clase Matriz03021_BS representa una matriz con tamaño fijo de entradas tipo char. Posee métodos con enfoques propios al problema de Ball Sort.

```
#include <random>
#include <cstdlib>
#include <time.h>
#include "Matriz03021_BS.h"
```

Dependencia gráfica adjunta para Matriz03021_BS.cpp:



6.24.1. Descripción detallada

La Clase Matriz03021_BS representa una matriz con tamaño fijo de entradas tipo char. Posee métodos con enfoques propios al problema de Ball Sort.

Autor

Fabián Orozco, Sofía Shum y Carolina Zamora.

Versión

1.0

Fecha

2021-09-07

Copyright

Copyright (c) 2021

6.25. Referencia del Archivo Matriz03021_BS.h

`#include <iostream>`
Dependencia gráfica adjunta para Matriz03021_BS.h:

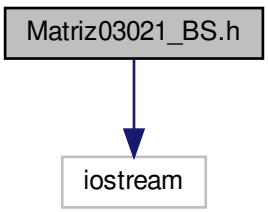
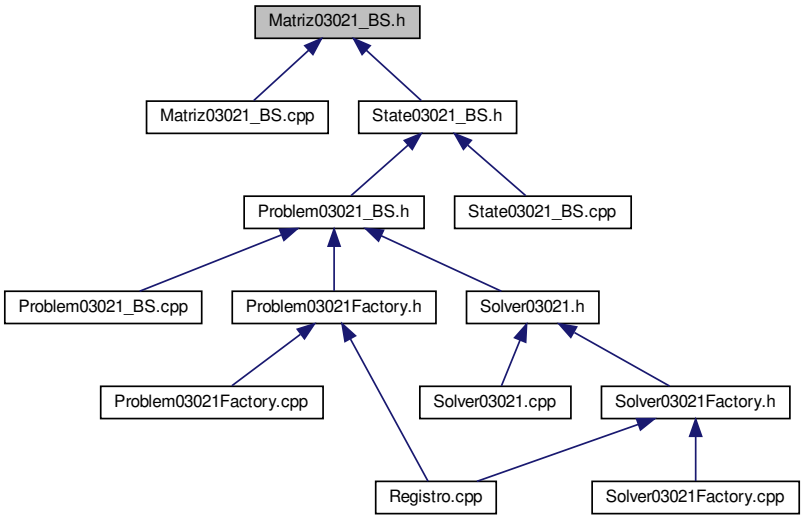


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class Matriz03021_BS

defines

- #define FILASBS 4
- #define COLUMNASBS 6

6.25.1. Documentación de los 'defines'

6.25.1.1. COLUMNASBS

```
#define COLUMNASBS 6
```

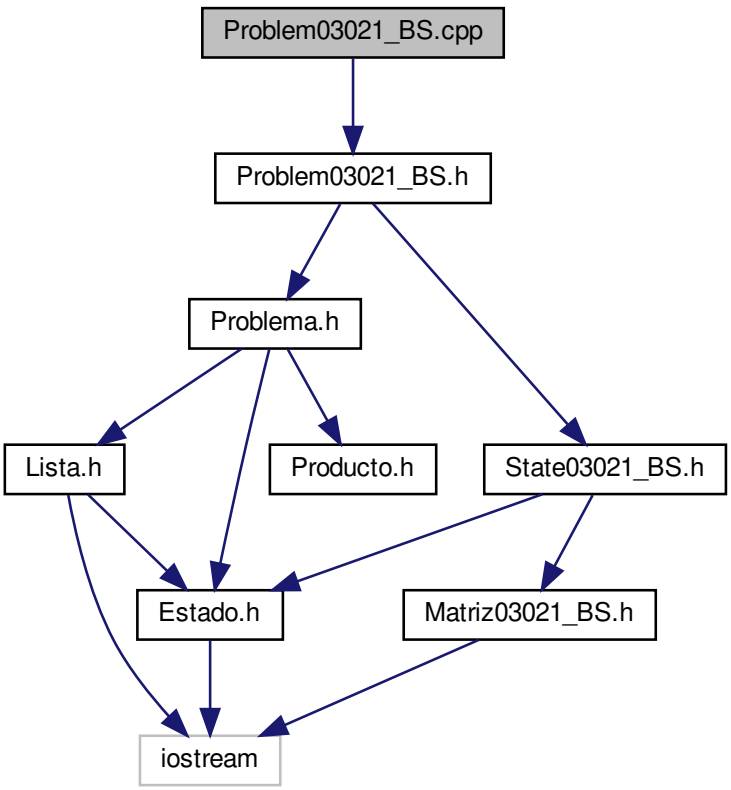
6.25.1.2. FILASBS

```
#define FILASBS 4
```

6.26. Referencia del Archivo Problem03021_BS.cpp

La Clase Problem03021_BS representa en general el juego de Ball Sort, administra las opciones de éste mediante la clase State03021_BS.

```
#include "Problem03021_BS.h"
```
Dependencia gráfica adjunta para Problem03021_BS.cpp:



6.26.1. Descripción detallada

La Clase Problem03021_BS representa en general el juego de Ball Sort, administra las opciones de éste mediante la clase State03021_BS.

Autor

Fabián Orozco, Sofía Shum y Carolina Zamora.

Versión

1.0

Fecha

2021-09-07

Copyright

Copyright (c) 2021

6.27. Referencia del Archivo Problem03021_BS.h

```
#include "Problema.h"
#include "State03021_BS.h"
```

Dependencia gráfica adjunta para Problem03021_BS.h:

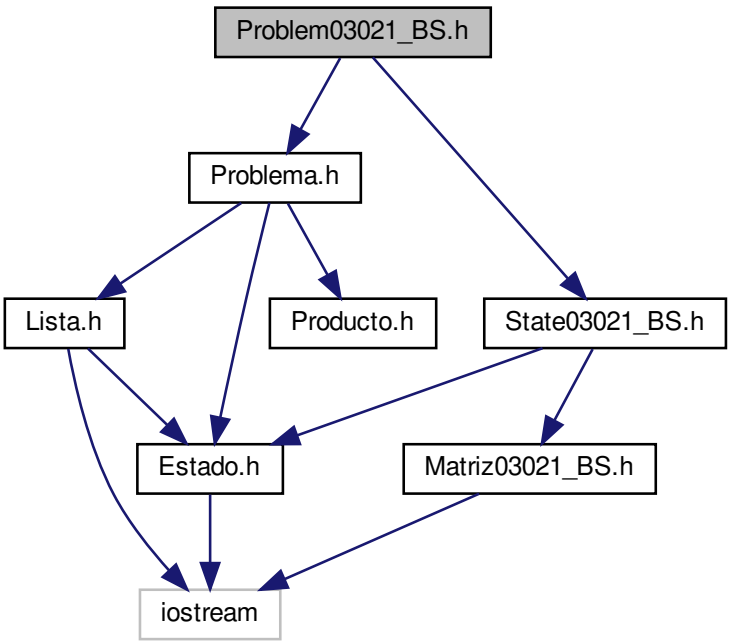
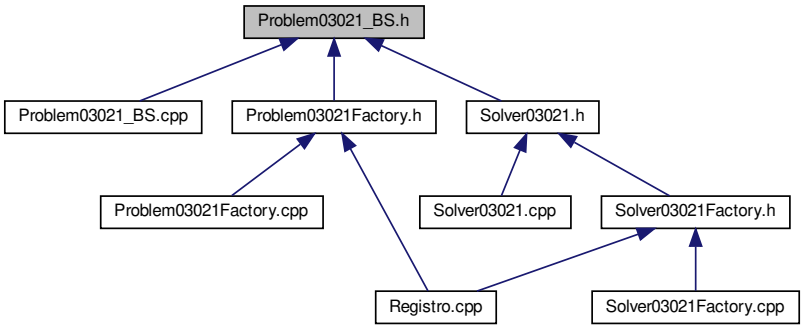


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



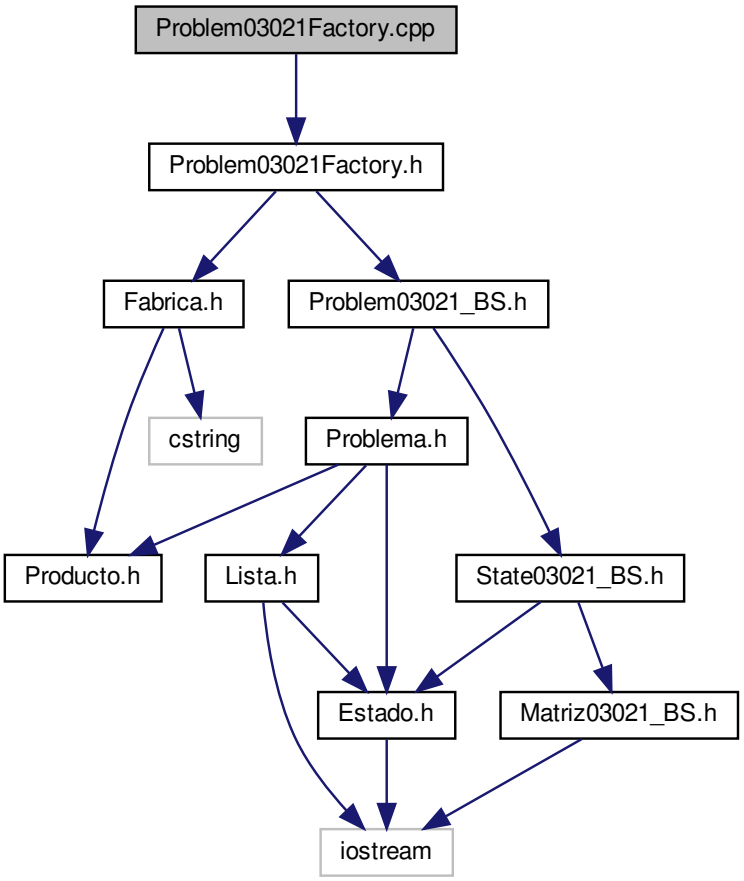
Clases

- class Problem03021_BS

6.28. Referencia del Archivo Problem03021Factory.cpp

La Clase Problem03021Factory representa una fabrica que produce problemas de Ball Sort.

```
#include "Problem03021Factory.h"
```
Dependencia gráfica adjunta para Problem03021Factory.cpp:



6.28.1. Descripción detallada

La Clase Problem03021Factory representa una fabrica que produce problemas de Ball Sort.

Autor

Fabián Orozco, Sofía Shum y Carolina Zamora.

Versión

1.0

Fecha

2021-09-07

Copyright

Copyright (c) 2021

6.29. Referencia del Archivo Problem03021Factory.h

```
#include "Fabrica.h"
#include "Problem03021_BS.h"
```

Dependencia gráfica adjunta para Problem03021Factory.h:

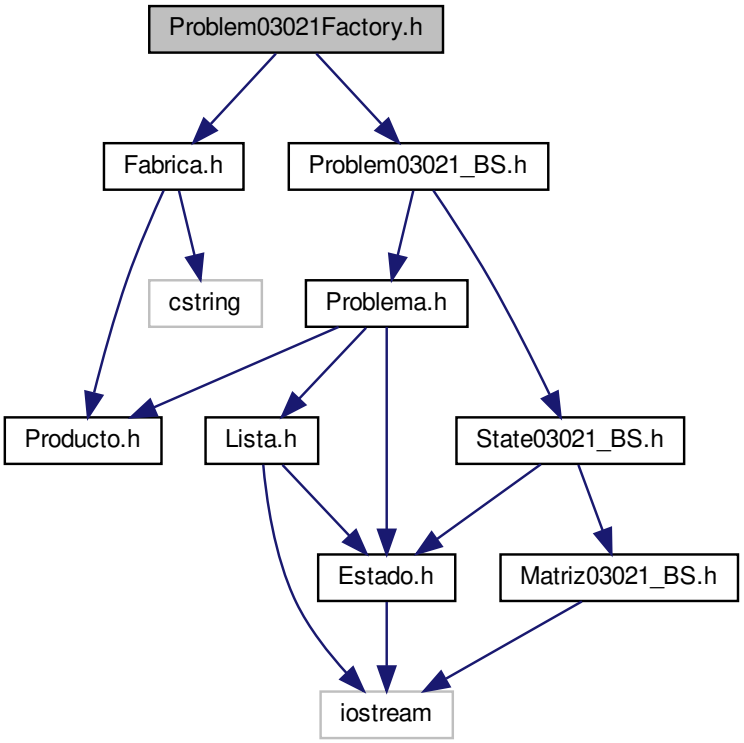
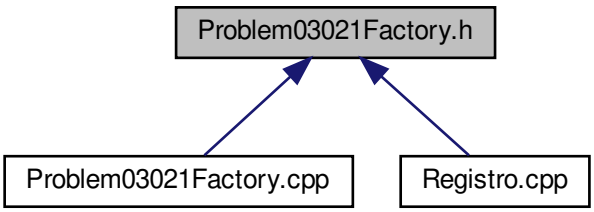


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class Problem03021Factory

6.30. Referencia del Archivo Problema.h

```
#include "Producto.h"
#include "Estado.h"
#include "Lista.h"
```

Dependencia gráfica adjunta para Problema.h:

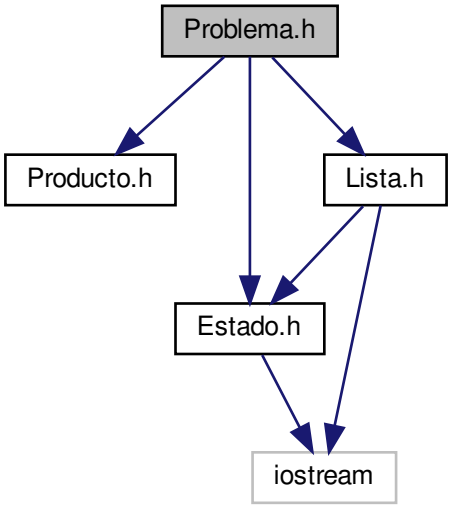
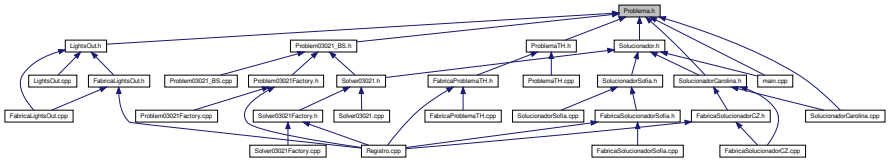


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



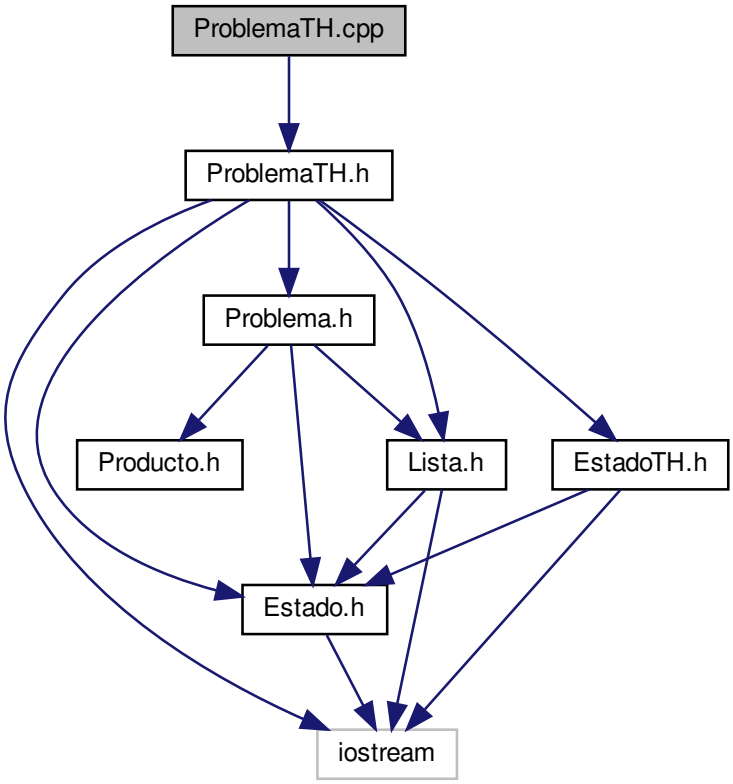
Clases

- class Problema

6.31. Referencia del Archivo ProblemaTH.cpp

```
#include "ProblemaTH.h"
```

Dependencia gráfica adjunta para ProblemaTH.cpp:



6.32. Referencia del Archivo ProblemaTH.h

```
#include "Problema.h"
#include "EstadoTH.h"
#include "Estado.h"
#include <iostream>
#include "Lista.h"
```

Dependencia gráfica adjunta para ProblemaTH.h:

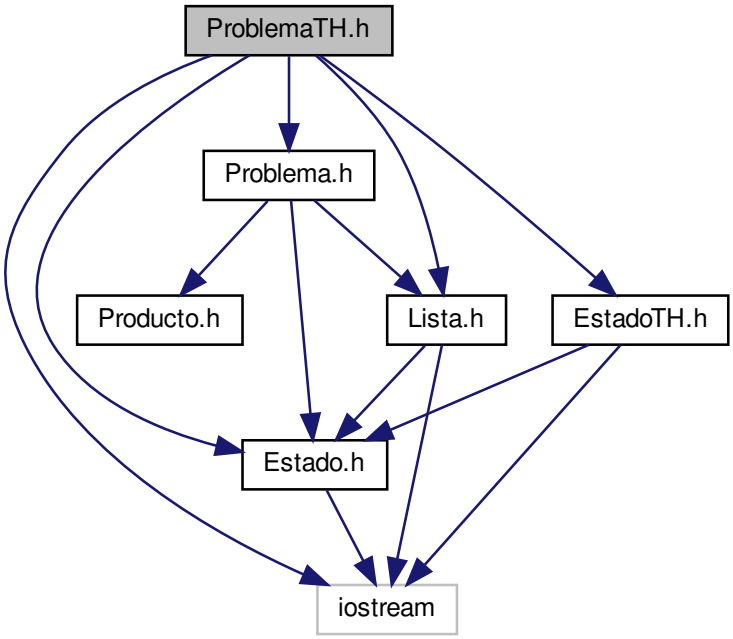
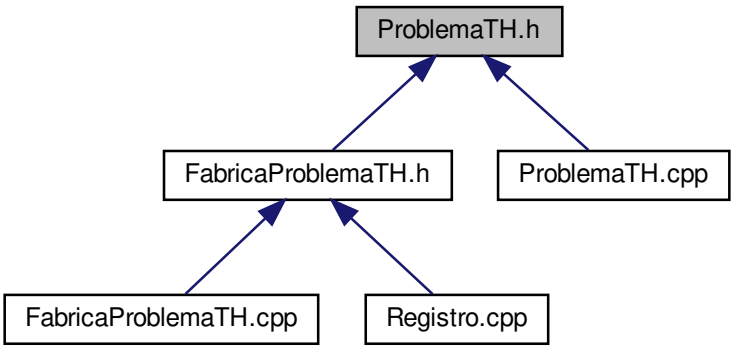


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



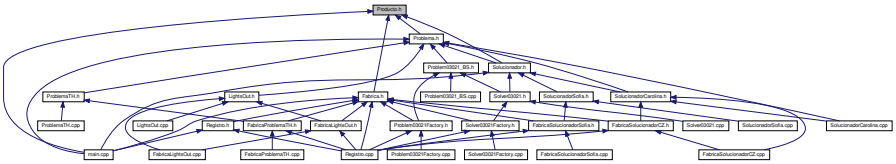
Clases

- class ProblemaTH

  Clase derivada de Problema, representa problema Torres de Hanoi.

6.33. Referencia del Archivo Producto.h

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

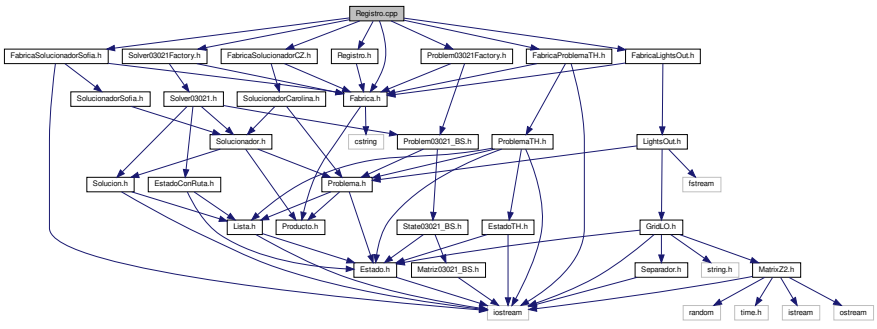
- class Producto

6.34. Referencia del Archivo README.md

6.35. Referencia del Archivo Registro.cpp

```
#include "Registro.h"
#include "Fabrica.h"
#include "FabricaProblemaTH.h"
#include "FabricaSolucionadorSofia.h"
#include "FabricaLightsOut.h"
#include "FabricaSolucionadorCZ.h"
#include "Problem03021Factory.h"
#include "Solver03021Factory.h"
```

Dependencia gráfica adjunta para Registro.cpp:



6.36. Referencia del Archivo Registro.h

```
#include "Fabrica.h"
```

Dependencia gráfica adjunta para Registro.h:

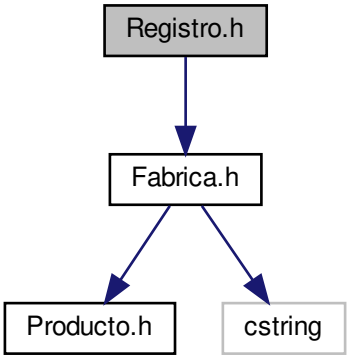
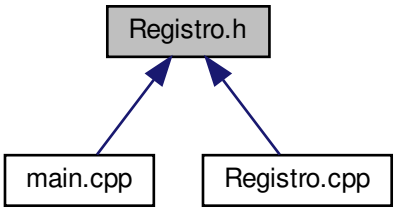


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class Registro

defines

- #define CAPACIDAD 120

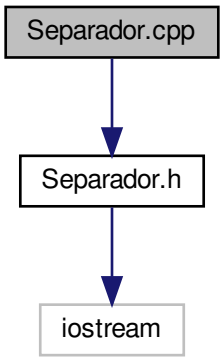
6.36.1. Documentación de los 'defines'

6.36.1.1. CAPACIDAD

```
#define CAPACIDAD 120
```

6.37. Referencia del Archivo Separador.cpp

```
#include "Separador.h"
```
Dependencia gráfica adjunta para Separador.cpp:



6.38. Referencia del Archivo Separador.h

```
#include <iostream>
```
Dependencia gráfica adjunta para Separador.h:

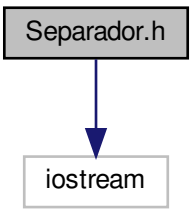
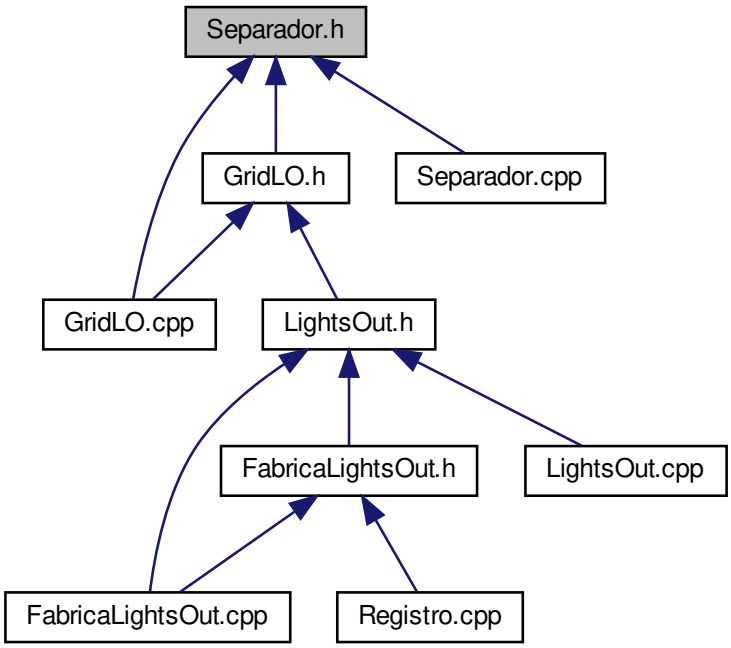


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



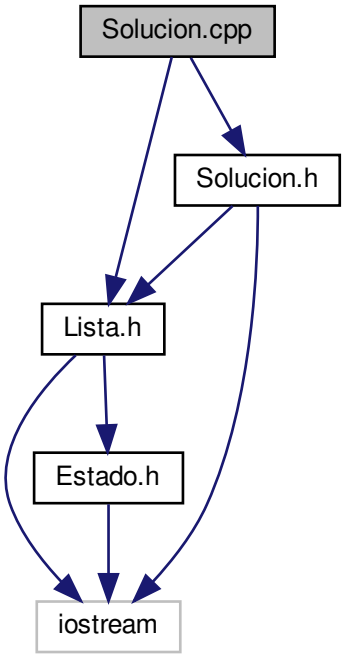
Clases

- class Separador

6.39. Referencia del Archivo Solucion.cpp

```
#include "Lista.h"
#include "Solucion.h"
```

Dependencia gráfica adjunta para Solucion.cpp:



6.40. Referencia del Archivo Solucion.h

```
#include <iostream>
#include "Lista.h"
```

Dependencia gráfica adjunta para Solucion.h:

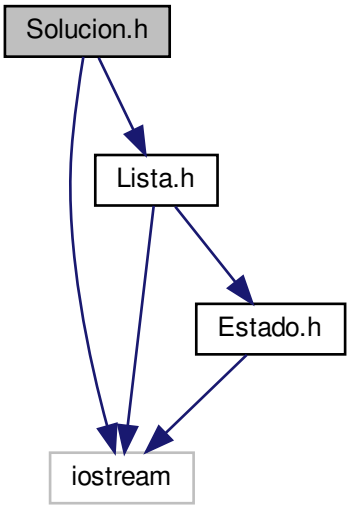
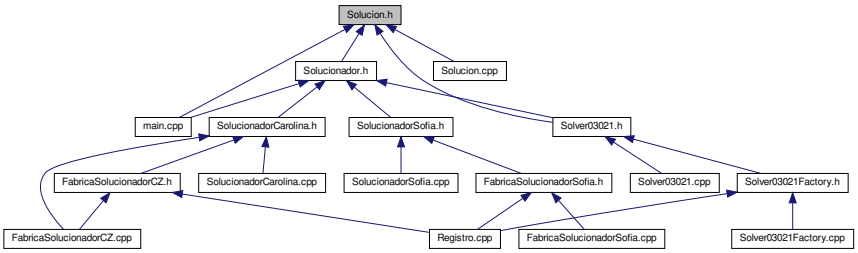


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class Solucion

6.41. Referencia del Archivo Solucionador.h

```
#include "Producto.h"
#include "Problema.h"
```

```
#include "Solucion.h"
```

Dependencia gráfica adjunta para Solucionador.h:

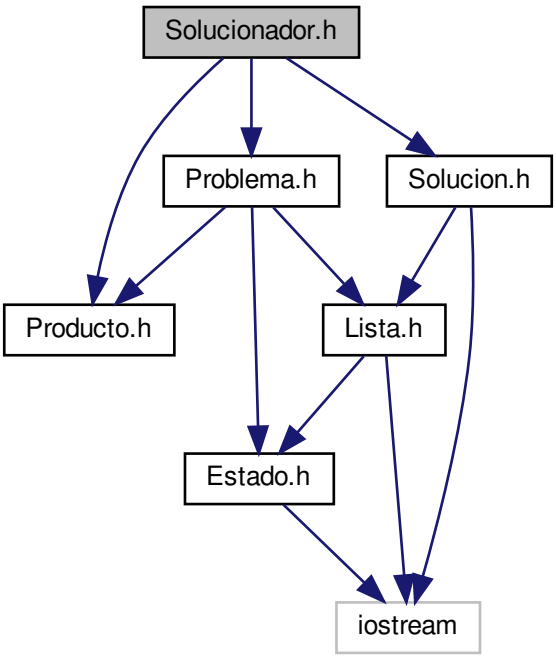
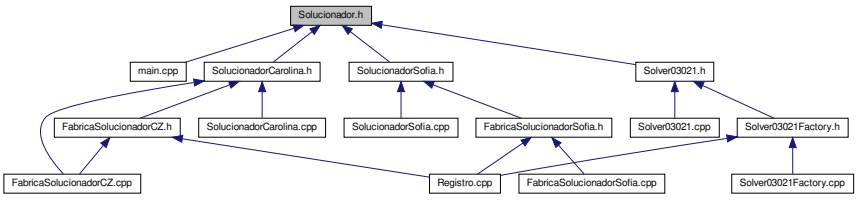


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



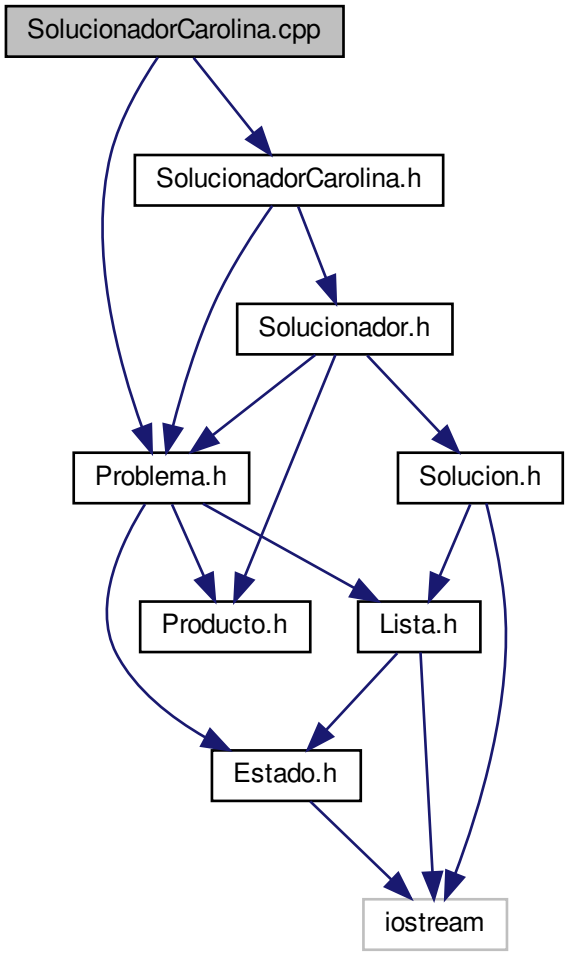
Clases

- class Solucionador

6.42. Referencia del Archivo SolucionadorCarolina.cpp

```
#include "SolucionadorCarolina.h"
#include "Problema.h"
```

Dependencia gráfica adjunta para SolucionadorCarolina.cpp:



6.43. Referencia del Archivo SolucionadorCarolina.h

```
#include "Problema.h"
#include "Solucionador.h"
```

Dependencia gráfica adjunta para SolucionadorCarolina.h:

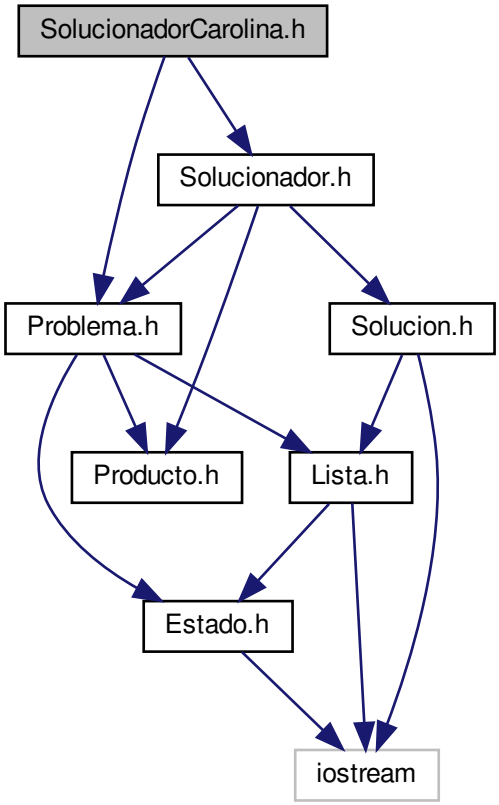
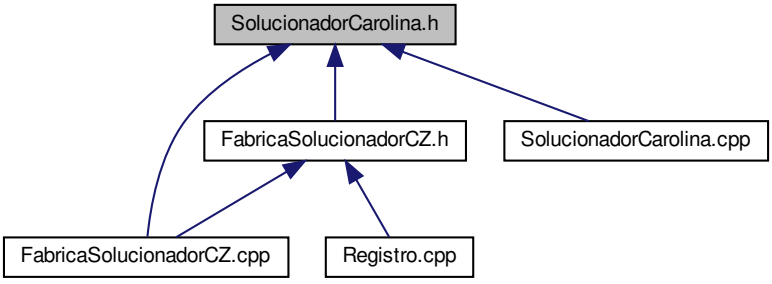


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

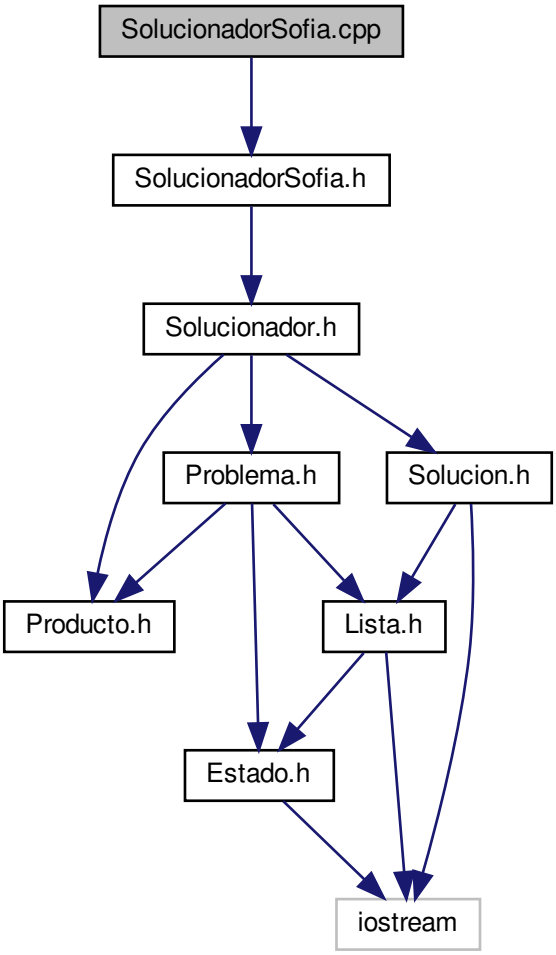
- class SolucionadorCarolina

  Solucionador de Carolina, derivado de Solucionador.

6.44. Referencia del Archivo SolucionadorSofia.cpp

```
#include "SolucionadorSofia.h"
```

Dependencia gráfica adjunta para SolucionadorSofia.cpp:



6.45. Referencia del Archivo SolucionadorSofia.h

```
#include "Solucionador.h"
```

Dependencia gráfica adjunta para SolucionadorSofia.h:

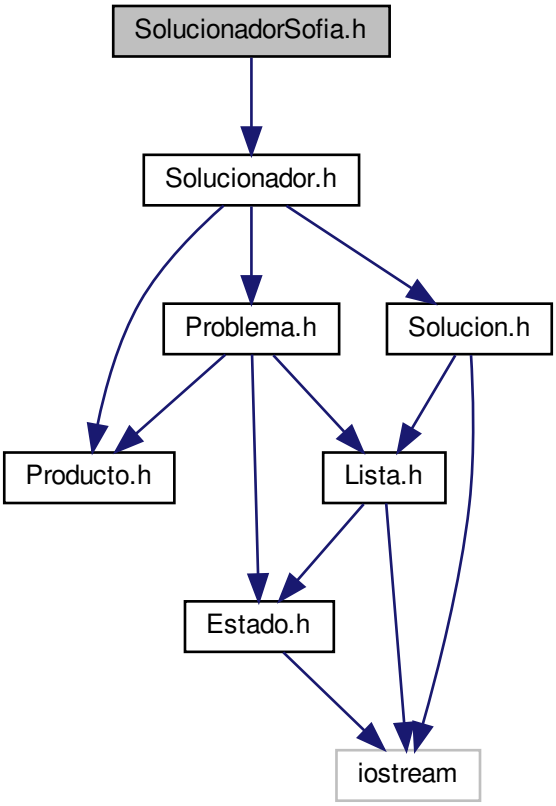
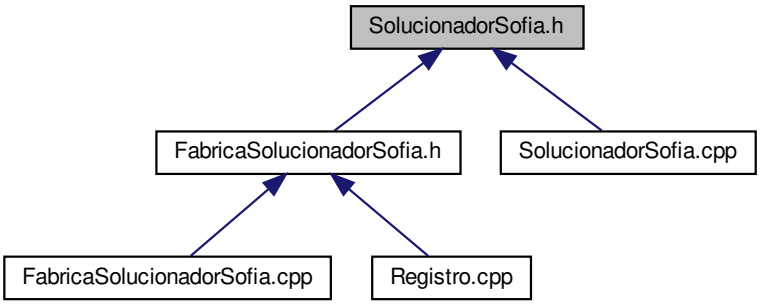


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class SolucionadorSofia

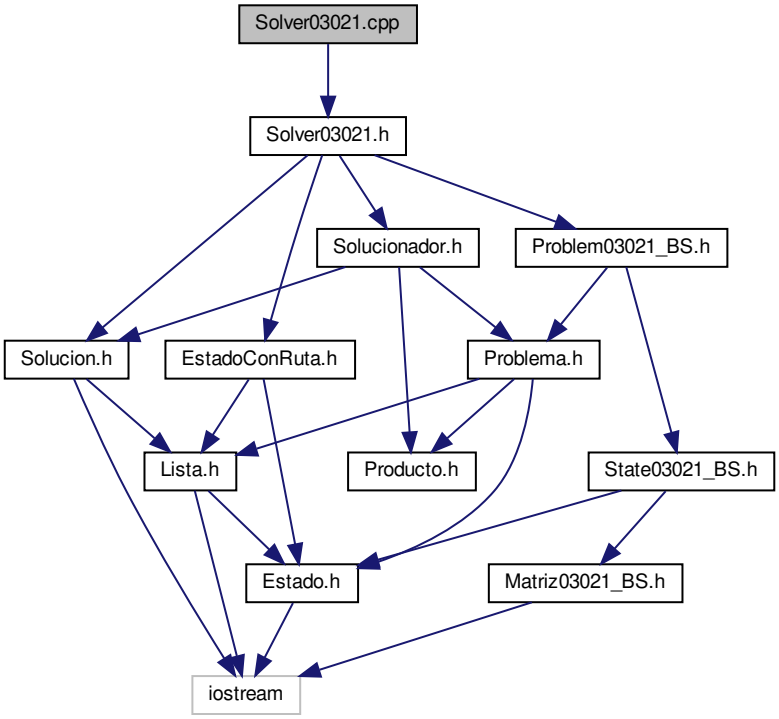
Solucionador de Sofia, clase derivada de Solucionador.

6.46. Referencia del Archivo Solver03021.cpp

La Clase Solver03021 representa un solucionador genérico que busca la solución óptima de un problema que se resuelve por algorítmos de búsqueda.

```
#include "Solver03021.h"
```

Dependencia gráfica adjunta para Solver03021.cpp:



6.46.1. Descripción detallada

La Clase Solver03021 representa un solucionador genérico que busca la solución óptima de un problema que se resuelve por algorítmos de búsqueda.

Autor

Fabián Orozco, Sofía Shum y Carolina Zamora.

Versión

1.0

Fecha

2021-09-07

Copyright

Copyright (c) 2021

6.47. Referencia del Archivo Solver03021.h

```
#include "Solucionador.h"
#include "Problem03021_BS.h"
#include "EstadoConRuta.h"
#include "Solucion.h"
```

Dependencia gráfica adjunta para Solver03021.h:

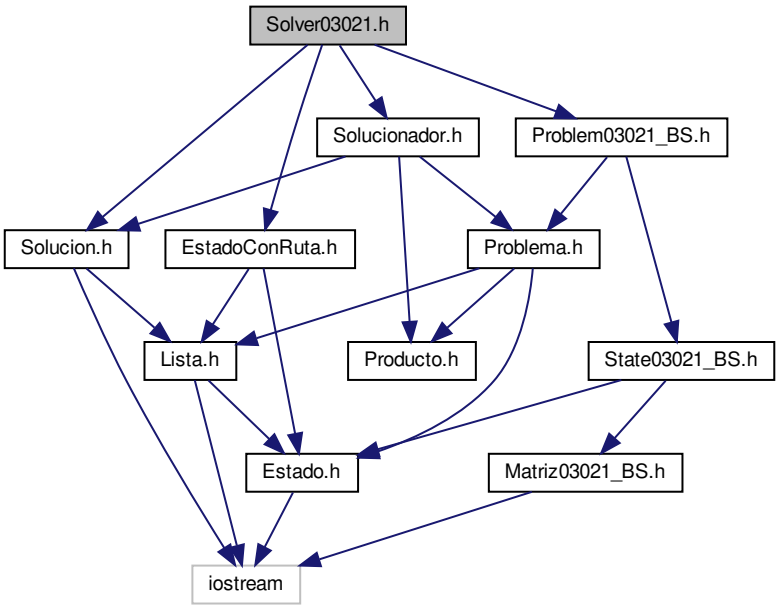
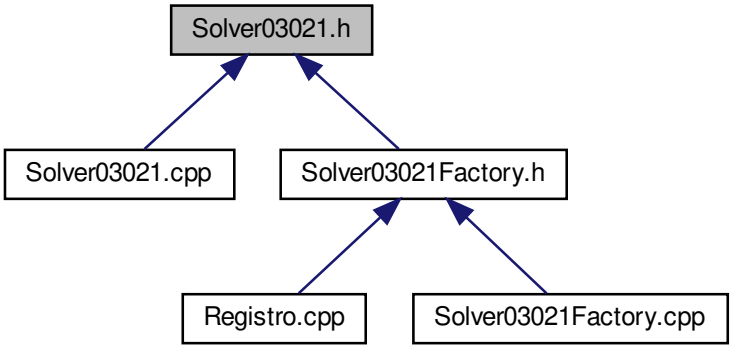


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

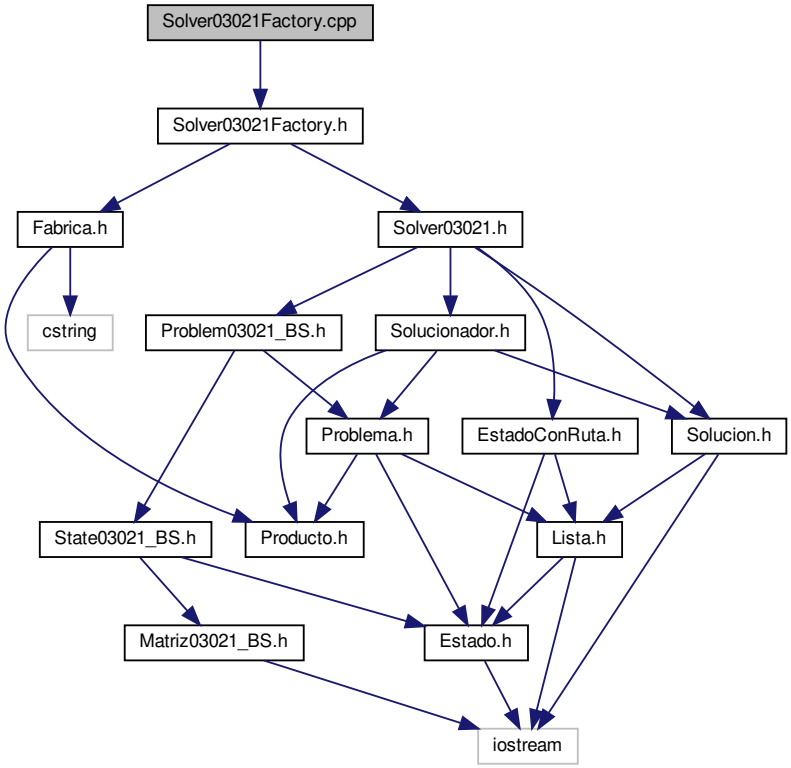
- class Solver03021

6.48. Referencia del Archivo Solver03021Factory.cpp

La Clase Solver03021Factory representa una fabrica que produce solucionadores.

```
#include "Solver03021Factory.h"
```

Dependencia gráfica adjunta para Solver03021Factory.cpp:



6.48.1. Descripción detallada

La Clase Solver03021Factory representa una fabrica que produce solucionadores.

Autor

Fabián Orozco, Sofía Shum y Carolina Zamora.

Versión

1.0

Fecha

2021-09-07

Copyright

Copyright (c) 2021

6.49. Referencia del Archivo Solver03021Factory.h

```
#include "Fabrica.h"
#include "Solver03021.h"
```

Dependencia gráfica adjunta para Solver03021Factory.h:

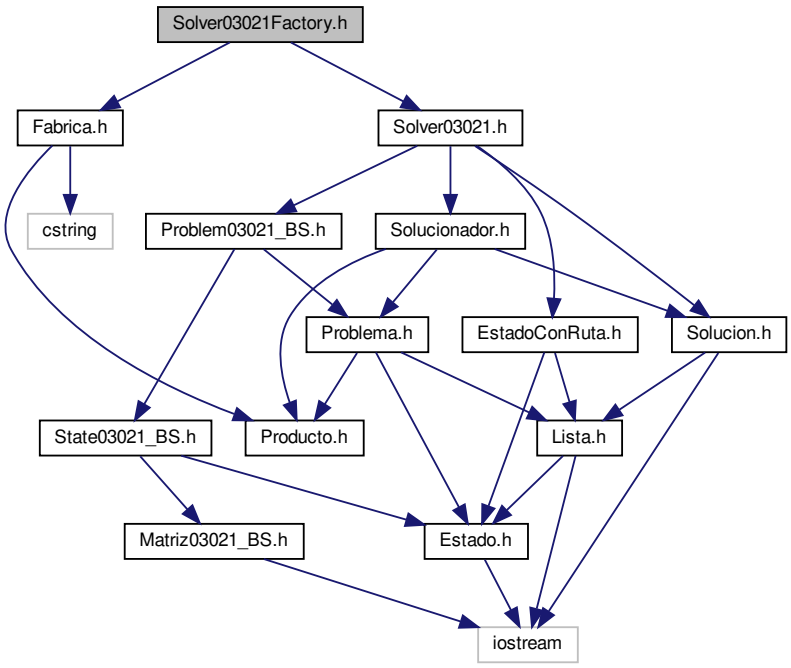
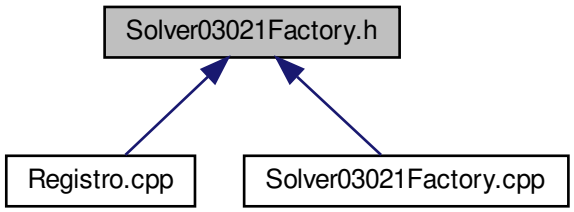


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



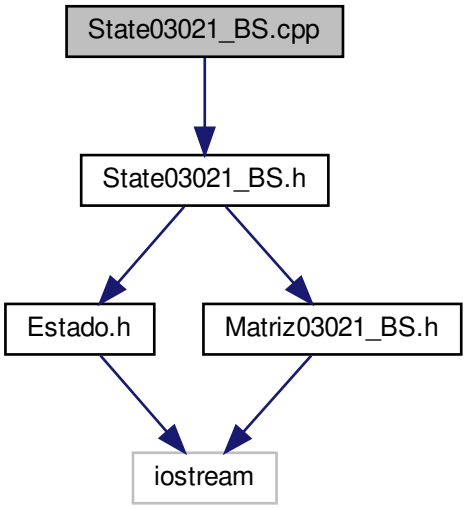
Clases

- class Solver03021Factory

6.50. Referencia del Archivo State03021_BS.cpp

La Clase State03021_BS representa un posible estado o posición de una matriz enfocada en Ball Sort.

```
#include "State03021_BS.h"
```
Dependencia gráfica adjunta para State03021_BS.cpp:



6.50.1. Descripción detallada

La Clase State03021_BS representa un posible estado o posición de una matriz enfocada en Ball Sort.

Autor

Fabián Orozco, Sofía Shum y Carolina Zamora.

Versión

1.0

Fecha

2021-09-07

Copyright

Copyright (c) 2021

6.51. Referencia del Archivo State03021_BS.h

```
#include "Estado.h"
#include "Matriz03021_BS.h"
```

Dependencia gráfica adjunta para State03021_BS.h:

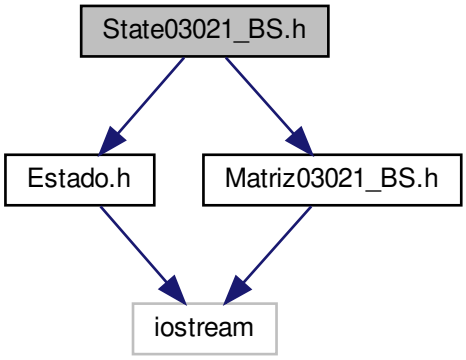
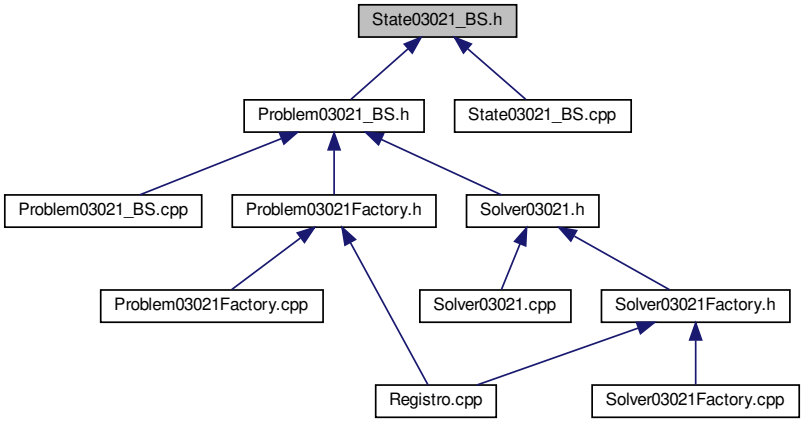


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class State03021_BS

Índice alfabético