

## Tarea Programada 2

Generado por Doxygen 1.9.1



<b>1 Tarea Programada 2</b>	<b>1</b>
1.1 Compilación	1
1.1.1 Usando g++:	1
1.1.2 Usando Makefile:	1
1.2 Ejecución	2
<b>2 Índice jerárquico</b>	<b>3</b>
2.1 Jerarquía de la clase	3
<b>3 Índice de clases</b>	<b>5</b>
3.1 Lista de clases	5
<b>4 Índice de archivos</b>	<b>7</b>
4.1 Lista de archivos	7
<b>5 Documentación de las clases</b>	<b>9</b>
5.1 Referencia de la Clase Estado	9
5.1.1 Documentación del constructor y destructor	10
5.1.1.1 ~Estado()	10
5.1.2 Documentación de las funciones miembro	10
5.1.2.1 cargar()	11
5.1.2.2 clonar()	11
5.1.2.3 imprimir()	11
5.1.2.4 operator=()	11
5.1.2.5 operator==()	11
5.1.3 Documentación de las funciones relacionadas y clases amigas	11
5.1.3.1 operator<<	12
5.1.3.2 operator>>	12
5.2 Referencia de la Clase EstadoConRuta	12
5.2.1 Documentación del constructor y destructor	14
5.2.1.1 EstadoConRuta() [1/2]	14
5.2.1.2 EstadoConRuta() [2/2]	14
5.2.1.3 ~EstadoConRuta()	14
5.2.2 Documentación de las funciones miembro	14
5.2.2.1 clonar()	15
5.2.2.2 getEstado()	15
5.2.2.3 getRuta()	15
5.2.2.4 operator=()	15
5.2.2.5 operator==()	16
5.3 Referencia de la Clase EstadoTH	16
5.3.1 Descripción detallada	18
5.3.2 Documentación del constructor y destructor	18
5.3.2.1 EstadoTH()	18
5.3.3 Documentación de las funciones miembro	18

5.3.3.1 cargar()	18
5.3.3.2 clonar()	18
5.3.3.3 imprimir()	19
5.3.3.4 operator=()	19
5.3.3.5 operator==()	19
5.3.4 Documentación de las funciones relacionadas y clases amigas	20
5.3.4.1 ProblemaTH	20
5.4 Referencia de la Clase Fabrica	20
5.4.1 Documentación del constructor y destructor	21
5.4.1.1 ~Fabrica()	21
5.4.2 Documentación de las funciones miembro	21
5.4.2.1 produce()	22
5.4.2.2 producir()	22
5.4.2.3 setNombre()	22
5.4.3 Documentación de los datos miembro	22
5.4.3.1 nombre	22
5.5 Referencia de la Clase FabricaLightsOut	23
5.5.1 Descripción detallada	24
5.5.2 Documentación de las funciones miembro	24
5.5.2.1 producir()	25
5.6 Referencia de la Clase FabricaProblemaTH	25
5.6.1 Descripción detallada	26
5.6.2 Documentación de las funciones miembro	26
5.6.2.1 producir()	27
5.7 Referencia de la Clase FabricaSolucionadorCZ	27
5.7.1 Descripción detallada	28
5.7.2 Documentación de las funciones miembro	28
5.7.2.1 producir()	29
5.8 Referencia de la Clase FabricaSolucionadorSofia	29
5.8.1 Descripción detallada	30
5.8.2 Documentación de las funciones miembro	30
5.8.2.1 producir()	31
5.9 Referencia de la Clase GridLO	31
5.9.1 Descripción detallada	33
5.9.2 Documentación del constructor y destructor	33
5.9.2.1 GridLO() [1/2]	33
5.9.2.2 GridLO() [2/2]	33
5.9.2.3 ~GridLO()	33
5.9.3 Documentación de las funciones miembro	33
5.9.3.1 cargar()	34
5.9.3.2 clonar()	35
5.9.3.3 flip()	35

5.9.3.4 imprimir()	35
5.9.3.5 operator=()	36
5.9.3.6 operator==()	36
5.9.4 Documentación de las funciones relacionadas y clases amigas	36
5.9.4.1 LightsOut	37
5.10 Referencia de la Clase Lista::Iterador	37
5.10.1 Documentación del constructor y destructor	37
5.10.1.1 Iterador() [1/2]	38
5.10.1.2 Iterador() [2/2]	38
5.10.2 Documentación de las funciones miembro	38
5.10.2.1 operator=()	38
5.10.2.2 operator*()	38
5.10.2.3 operator++() [1/2]	38
5.10.2.4 operator++() [2/2]	38
5.10.2.5 operator==()	38
5.10.3 Documentación de las funciones relacionadas y clases amigas	39
5.10.3.1 Lista	39
5.11 Referencia de la Clase LightsOut	39
5.11.1 Descripción detallada	40
5.11.2 Documentación del constructor y destructor	41
5.11.2.1 LightsOut()	41
5.11.3 Documentación de las funciones miembro	41
5.11.3.1 esSolucion()	41
5.11.3.2 getEstadoInicial()	41
5.11.3.3 getSiguietes()	41
5.11.3.4 heuristica()	42
5.12 Referencia de la Clase Lista	42
5.12.1 Documentación del constructor y destructor	44
5.12.1.1 Lista() [1/2]	44
5.12.1.2 Lista() [2/2]	44
5.12.1.3 ~Lista()	44
5.12.2 Documentación de las funciones miembro	44
5.12.2.1 begin()	44
5.12.2.2 borrar()	44
5.12.2.3 buscar()	45
5.12.2.4 end()	45
5.12.2.5 imprimir()	45
5.12.2.6 insertar()	45
5.12.2.7 isEmpty()	45
5.12.2.8 operator=()	45
5.12.2.9 pop_back()	45
5.12.2.10 pop_front()	46

5.12.2.11 push_back()	46
5.12.2.12 push_front()	46
5.12.2.13 rbegin()	46
5.12.3 Documentación de las funciones relacionadas y clases amigas	46
5.12.3.1 Iterador	46
5.12.3.2 operator<<	46
5.13 Referencia de la Clase MatrixZ2	47
5.13.1 Descripción detallada	48
5.13.2 Documentación del constructor y destructor	48
5.13.2.1 MatrixZ2() [1/4]	48
5.13.2.2 MatrixZ2() [2/4]	48
5.13.2.3 MatrixZ2() [3/4]	49
5.13.2.4 MatrixZ2() [4/4]	49
5.13.2.5 ~MatrixZ2()	49
5.13.3 Documentación de las funciones miembro	49
5.13.3.1 cargar()	49
5.13.3.2 getColumns()	50
5.13.3.3 getRows()	50
5.13.3.4 imprimir()	50
5.13.3.5 operatori=()	51
5.13.3.6 operator+()	51
5.13.3.7 operator=() [1/2]	51
5.13.3.8 operator=() [2/2]	52
5.13.3.9 operator==(1/2)	52
5.13.3.10 operator==(2/2)	52
5.13.3.11 operator[]()	53
5.13.3.12 randomize()	53
5.14 Referencia de la Clase Matriz03021_BS	53
5.14.1 Documentación del constructor y destructor	55
5.14.1.1 Matriz03021_BS() [1/2]	55
5.14.1.2 Matriz03021_BS() [2/2]	55
5.14.1.3 ~Matriz03021_BS()	55
5.14.2 Documentación de las funciones miembro	55
5.14.2.1 contarMaxColumna()	56
5.14.2.2 contarMovimientosRestantes()	56
5.14.2.3 heuristica()	56
5.14.2.4 mover()	56
5.14.2.5 movimientoValido()	57
5.14.2.6 operatori=()	57
5.14.2.7 operator=()	58
5.14.2.8 operator==(1/2)	58
5.14.2.9 randomize()	58

5.14.2.10 solucionado()	58
5.14.3 Documentación de las funciones relacionadas y clases amigas	59
5.14.3.1 operator<<	59
5.14.3.2 operator>>	59
5.15 Referencia de la Clase Problem03021_BS	59
5.15.1 Documentación del constructor y destructor	61
5.15.1.1 Problem03021_BS()	62
5.15.2 Documentación de las funciones miembro	62
5.15.2.1 esSolucion()	62
5.15.2.2 getEstadoInicial()	62
5.15.2.3 getSiguietes()	62
5.15.2.4 heuristica()	63
5.16 Referencia de la Clase Problem03021Factory	63
5.16.1 Documentación del constructor y destructor	65
5.16.1.1 ~Problem03021Factory()	65
5.16.2 Documentación de las funciones miembro	66
5.16.2.1 producir()	66
5.17 Referencia de la Clase Problema	66
5.17.1 Documentación de las funciones miembro	67
5.17.1.1 esSolucion()	67
5.17.1.2 getEstadoInicial()	67
5.17.1.3 getSiguietes()	68
5.17.1.4 heuristica()	68
5.18 Referencia de la Clase ProblemaTH	68
5.18.1 Descripción detallada	70
5.18.2 Documentación de las funciones miembro	71
5.18.2.1 esSolucion()	71
5.18.2.2 getEstadoInicial()	71
5.18.2.3 getSiguietes()	71
5.18.2.4 heuristica()	72
5.19 Referencia de la Clase Producto	72
5.19.1 Documentación del constructor y destructor	73
5.19.1.1 ~Producto()	73
5.20 Referencia de la Clase Registro	73
5.20.1 Documentación del constructor y destructor	74
5.20.1.1 Registro()	74
5.20.1.2 ~Registro()	74
5.20.2 Documentación de las funciones miembro	74
5.20.2.1 getFabrica()	74
5.21 Referencia de la Clase Separador	74
5.21.1 Descripción detallada	75
5.21.2 Documentación de las funciones miembro	75

5.21.2.1 liberar()	75
5.21.2.2 separar()	75
5.22 Referencia de la Clase Solucion	76
5.22.1 Documentación del constructor y destructor	76
5.22.1.1 Solucion()	76
5.22.1.2 ~Solucion()	76
5.22.2 Documentación de las funciones relacionadas y clases amigas	77
5.22.2.1 operator<<	77
5.23 Referencia de la Clase Solucionador	77
5.23.1 Documentación de las funciones miembro	78
5.23.1.1 solucione()	78
5.24 Referencia de la Clase SolucionadorCarolina	79
5.24.1 Descripción detallada	80
5.24.2 Documentación de las funciones miembro	80
5.24.2.1 solucione()	80
5.25 Referencia de la Clase SolucionadorSofia	81
5.25.1 Descripción detallada	82
5.25.2 Documentación de las funciones miembro	82
5.25.2.1 solucione()	82
5.25.3 Documentación de los datos miembro	83
5.25.3.1 cantidadPasos	83
5.26 Referencia de la Clase Solver03021	83
5.26.1 Documentación de las funciones miembro	85
5.26.1.1 ordenarLista()	85
5.26.1.2 solucione()	86
5.27 Referencia de la Clase Solver03021Factory	86
5.27.1 Documentación del constructor y destructor	88
5.27.1.1 ~Solver03021Factory()	88
5.27.2 Documentación de las funciones miembro	89
5.27.2.1 producir()	89
5.28 Referencia de la Clase State03021_BS	89
5.28.1 Documentación del constructor y destructor	91
5.28.1.1 State03021_BS() [1/2]	91
5.28.1.2 State03021_BS() [2/2]	91
5.28.1.3 ~State03021_BS()	91
5.28.2 Documentación de las funciones miembro	91
5.28.2.1 cargar()	91
5.28.2.2 clonar()	92
5.28.2.3 imprimir()	92
5.28.2.4 operatori=()	93
5.28.2.5 operator==(())	93
5.28.3 Documentación de las funciones relacionadas y clases amigas	93



5.28.3.1 Problem03021_BS	93
<b>6 Documentación de archivos</b>	<b>95</b>
6.1 Referencia del Archivo Estado.h	95
6.2 Referencia del Archivo EstadoConRuta.cpp	96
6.2.1 Descripción detallada	96
6.3 Referencia del Archivo EstadoConRuta.h	97
6.4 Referencia del Archivo EstadoTH.cpp	98
6.5 Referencia del Archivo EstadoTH.h	98
6.5.1 Documentación de los 'defines'	99
6.5.1.1 LINESIZE	99
6.5.1.2 QPLATES	99
6.5.1.3 QTOWERS	100
6.6 Referencia del Archivo Fabrica.h	100
6.6.1 Documentación de los 'defines'	100
6.6.1.1 NOMBRE_MAX_SIZE	100
6.7 Referencia del Archivo FabricaLightsOut.cpp	101
6.8 Referencia del Archivo FabricaLightsOut.h	101
6.9 Referencia del Archivo FabricaProblemaTH.cpp	102
6.10 Referencia del Archivo FabricaProblemaTH.h	103
6.11 Referencia del Archivo FabricaSolucionadorCZ.cpp	105
6.12 Referencia del Archivo FabricaSolucionadorCZ.h	105
6.13 Referencia del Archivo FabricaSolucionadorSofia.cpp	107
6.14 Referencia del Archivo FabricaSolucionadorSofia.h	108
6.15 Referencia del Archivo GridLO.cpp	109
6.16 Referencia del Archivo GridLO.h	109
6.16.1 Documentación de los 'defines'	111
6.16.1.1 GRIDSIZE_LO	111
6.16.1.2 STREAMSIZE_LO	111
6.17 Referencia del Archivo LightsOut.cpp	111
6.18 Referencia del Archivo LightsOut.h	111
6.19 Referencia del Archivo Lista.cpp	112
6.20 Referencia del Archivo Lista.h	113
6.21 Referencia del Archivo main.cpp	114
6.21.1 Documentación de las funciones	115
6.21.1.1 main()	115
6.22 Referencia del Archivo MatrixZ2.cpp	115
6.23 Referencia del Archivo MatrixZ2.h	115
6.24 Referencia del Archivo Matriz03021_BS.cpp	117
6.24.1 Descripción detallada	117
6.25 Referencia del Archivo Matriz03021_BS.h	118
6.25.1 Documentación de los 'defines'	119

6.25.1.1 COLUMNASBS . . . . .	119
6.25.1.2 FILASBS . . . . .	119
6.26 Referencia del Archivo Problem03021_BS.cpp . . . . .	119
6.26.1 Descripción detallada . . . . .	120
6.27 Referencia del Archivo Problem03021_BS.h . . . . .	120
6.28 Referencia del Archivo Problem03021Factory.cpp . . . . .	121
6.28.1 Descripción detallada . . . . .	122
6.29 Referencia del Archivo Problem03021Factory.h . . . . .	123
6.30 Referencia del Archivo Problema.h . . . . .	124
6.31 Referencia del Archivo ProblemaTH.cpp . . . . .	124
6.32 Referencia del Archivo ProblemaTH.h . . . . .	125
6.33 Referencia del Archivo Producto.h . . . . .	127
6.34 Referencia del Archivo README.md . . . . .	127
6.35 Referencia del Archivo Registro.cpp . . . . .	127
6.36 Referencia del Archivo Registro.h . . . . .	128
6.36.1 Documentación de los 'defines' . . . . .	128
6.36.1.1 CAPACIDAD . . . . .	129
6.37 Referencia del Archivo Separador.cpp . . . . .	129
6.38 Referencia del Archivo Separador.h . . . . .	129
6.39 Referencia del Archivo Solucion.cpp . . . . .	130
6.40 Referencia del Archivo Solucion.h . . . . .	131
6.41 Referencia del Archivo Solucionador.h . . . . .	132
6.42 Referencia del Archivo SolucionadorCarolina.cpp . . . . .	133
6.43 Referencia del Archivo SolucionadorCarolina.h . . . . .	134
6.44 Referencia del Archivo SolucionadorSofia.cpp . . . . .	136
6.45 Referencia del Archivo SolucionadorSofia.h . . . . .	136
6.46 Referencia del Archivo Solver03021.cpp . . . . .	138
6.46.1 Descripción detallada . . . . .	138
6.47 Referencia del Archivo Solver03021.h . . . . .	139
6.48 Referencia del Archivo Solver03021Factory.cpp . . . . .	140
6.48.1 Descripción detallada . . . . .	141
6.49 Referencia del Archivo Solver03021Factory.h . . . . .	142
6.50 Referencia del Archivo State03021_BS.cpp . . . . .	143
6.50.1 Descripción detallada . . . . .	143
6.51 Referencia del Archivo State03021_BS.h . . . . .	144
<b>Índice alfabético</b>	<b>145</b>

# 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

# Índice jerárquico

### 2.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

Estado . . . . .	9
EstadoConRuta . . . . .	12
EstadoTH . . . . .	16
GridLO . . . . .	31
State03021_BS . . . . .	89
Fabrica . . . . .	20
FabricaLightsOut . . . . .	23
FabricaProblemaTH . . . . .	25
FabricaSolucionadorCZ . . . . .	27
FabricaSolucionadorSofia . . . . .	29
Problem03021Factory . . . . .	63
Solver03021Factory . . . . .	86
Lista::Iterador . . . . .	37
Lista . . . . .	42
MatrixZ2 . . . . .	47
Matriz03021_BS . . . . .	53
Producto . . . . .	72
Problema . . . . .	66
LightsOut . . . . .	39
Problem03021_BS . . . . .	59
ProblemaTH . . . . .	68
Solucionador . . . . .	77
SolucionadorCarolina . . . . .	79
SolucionadorSofia . . . . .	81
Solver03021 . . . . .	83
Registro . . . . .	73
Separador . . . . .	74
Solucion . . . . .	76



## Capítulo 3

# Índice de clases

### 3.1. Lista de clases

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

<a href="#">Estado</a>	9
<a href="#">EstadoConRuta</a>	12
<a href="#">EstadoTH</a>	
Clase derivada de <a href="#">Estado</a> , representa tres torres con platillos (matrix) del problema Torres de Hanoi	16
<a href="#">Fabrica</a>	20
<a href="#">FabricaLightsOut</a>	
Clase derivada de <a href="#">Fabrica</a> , para producir un puntero a <a href="#">LightsOut</a>	23
<a href="#">FabricaProblemaTH</a>	
Clase derivada de <a href="#">Fabrica</a> , produce un puntero a <a href="#">ProblemaTH</a>	25
<a href="#">FabricaSolucionadorCZ</a>	
Clase derivada de <a href="#">Fabrica</a> , para producir un puntero a <a href="#">SolucionadorCarolina</a>	27
<a href="#">FabricaSolucionadorSofia</a>	
Clase derivada de <a href="#">Fabrica</a> , produce un puntero a <a href="#">SolucionadorSofia</a>	29
<a href="#">GridLO</a>	
Clase derivada de <a href="#">Estado</a> , representa una cuadrícula (grid) del problema <a href="#">LightsOut</a>	31
<a href="#">Lista::Iterador</a>	37
<a href="#">LightsOut</a>	
Clase derivada de <a href="#">Problema</a> , representa el problema <a href="#">LightsOut</a>	39
<a href="#">Lista</a>	42
<a href="#">MatrixZ2</a>	
Esta clase representa una matriz con entradas en el cuerpo Z2 (F2)	47
<a href="#">Matriz03021_BS</a>	53
<a href="#">Problem03021_BS</a>	59
<a href="#">Problem03021Factory</a>	63
<a href="#">Problema</a>	66
<a href="#">ProblemaTH</a>	
Clase derivada de <a href="#">Problema</a> , representa problema Torres de Hanoi	68
<a href="#">Producto</a>	72
<a href="#">Registro</a>	73
<a href="#">Separador</a>	74
<a href="#">Solucion</a>	76
<a href="#">Solucionador</a>	77
<a href="#">SolucionadorCarolina</a>	
<a href="#">Solucionador</a> de Carolina, derivado de <a href="#">Solucionador</a>	79

SolucionadorSofia	
Solucionador de Sofia, clase derivada de Solucionador	81
Solver03021	83
Solver03021Factory	86
State03021_BS	89



## Capítulo 4

# Indice de archivos

### 4.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

<a href="#">Estado.h</a>	95
<a href="#">EstadoConRuta.cpp</a>	
La Clase <a href="#">EstadoConRuta</a> es una clase auxiliar del solucionador, se encarga de contener un estado su el camino por el cual se llega hasta él	96
<a href="#">EstadoConRuta.h</a>	97
<a href="#">EstadoTH.cpp</a>	98
<a href="#">EstadoTH.h</a>	98
<a href="#">Fabrica.h</a>	100
<a href="#">FabricaLightsOut.cpp</a>	101
<a href="#">FabricaLightsOut.h</a>	101
<a href="#">FabricaProblemaTH.cpp</a>	102
<a href="#">FabricaProblemaTH.h</a>	103
<a href="#">FabricaSolucionadorCZ.cpp</a>	105
<a href="#">FabricaSolucionadorCZ.h</a>	105
<a href="#">FabricaSolucionadorSofia.cpp</a>	107
<a href="#">FabricaSolucionadorSofia.h</a>	108
<a href="#">GridLO.cpp</a>	109
<a href="#">GridLO.h</a>	109
<a href="#">LightsOut.cpp</a>	111
<a href="#">LightsOut.h</a>	111
<a href="#">Lista.cpp</a>	112
<a href="#">Lista.h</a>	113
<a href="#">main.cpp</a>	114
<a href="#">MatrixZ2.cpp</a>	115
<a href="#">MatrixZ2.h</a>	115
<a href="#">Matriz03021_BS.cpp</a>	
La Clase <a href="#">Matriz03021_BS</a> representa una matriz con tamaño fijo de entradas tipo char. Posee métodos con enfoques propios al problema de Ball Sort	117
<a href="#">Matriz03021_BS.h</a>	118
<a href="#">Problem03021_BS.cpp</a>	
La Clase <a href="#">Problem03021_BS</a> representa en general el juego de Ball Sort, administra las opciones de éste mediante la clase <a href="#">State03021_BS</a>	119
<a href="#">Problem03021_BS.h</a>	120
<a href="#">Problem03021Factory.cpp</a>	
La Clase <a href="#">Problem03021Factory</a> representa una fabrica que produce problemas de Ball Sort	121

<a href="#">Problem03021Factory.h</a>	123
<a href="#">Problema.h</a>	124
<a href="#">ProblemaTH.cpp</a>	124
<a href="#">ProblemaTH.h</a>	125
<a href="#">Producto.h</a>	127
<a href="#">Registro.cpp</a>	127
<a href="#">Registro.h</a>	128
<a href="#">Separador.cpp</a>	129
<a href="#">Separador.h</a>	129
<a href="#">Solucion.cpp</a>	130
<a href="#">Solucion.h</a>	131
<a href="#">Solucionador.h</a>	132
<a href="#">SolucionadorCarolina.cpp</a>	133
<a href="#">SolucionadorCarolina.h</a>	134
<a href="#">SolucionadorSofia.cpp</a>	136
<a href="#">SolucionadorSofia.h</a>	136
<a href="#">Solver03021.cpp</a>	
La Clase <a href="#">Solver03021</a> representa un solucionador genérico que busca la solución óptima de un problema que se resuelve por algoritmos de búsqueda	138
<a href="#">Solver03021.h</a>	139
<a href="#">Solver03021Factory.cpp</a>	
La Clase <a href="#">Solver03021Factory</a> representa una fabrica que produce solucionadores	140
<a href="#">Solver03021Factory.h</a>	142
<a href="#">State03021_BS.cpp</a>	
La Clase <a href="#">State03021_BS</a> representa un posible estado o posición de una matriz enfocada en	
Ball Sort	143
<a href="#">State03021_BS.h</a>	144

## Capítulo 5

# Documentación de las clases

### 5.1. Referencia de la Clase Estado

```
#include <Estado.h>
```

Diagrama de herencias de Estado

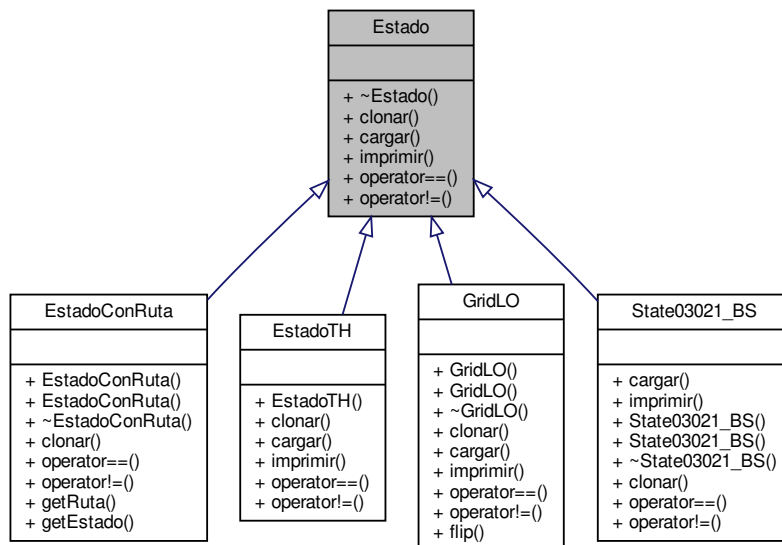
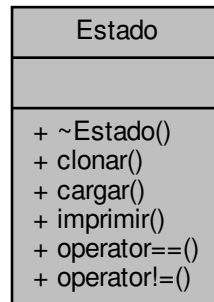


Diagrama de colaboración para Estado:



## 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. operator!=()

```
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

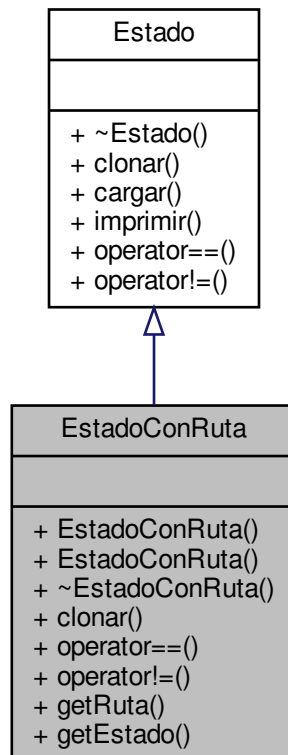
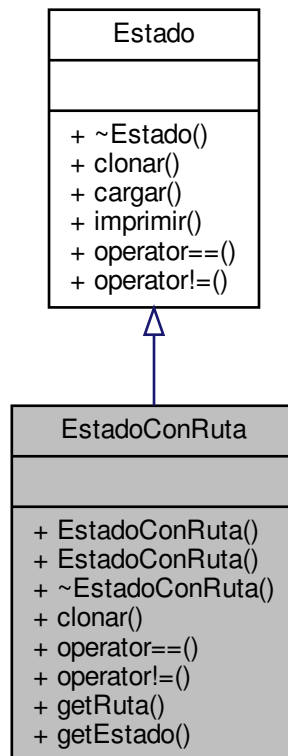


Diagrama de colaboración para EstadoConRuta:



## 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

<i>estado</i>	Recibe un puntero a <a href="#">Estado</a> con el estado que se quiere clonar y almacenar.
<i>lista</i>	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

<i>otro</i>	Recibe un <a href="#">EstadoConRuta</a> 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. 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](#).

### 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

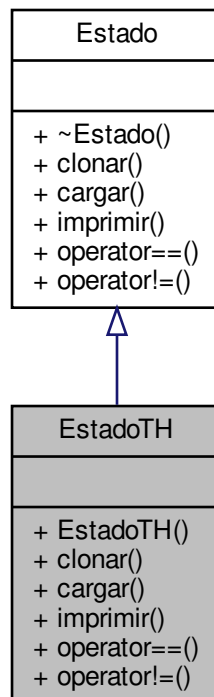
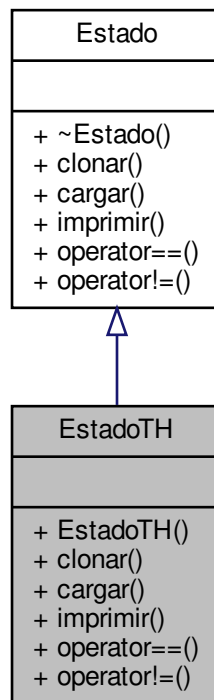


Diagrama de colaboración para EstadoTH:



## 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

<i>entrada</i>	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

<i>salida</i>	Ostream al cual se imprime el estado
---------------	--------------------------------------

##### Devuelve

Ostream modificado

Implementa [Estado](#).

#### 5.3.3.4. operator!=()

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

Operador para determinar si dos estados son distintos.

##### Parámetros

<i>e</i>	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

<i>e</i>	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

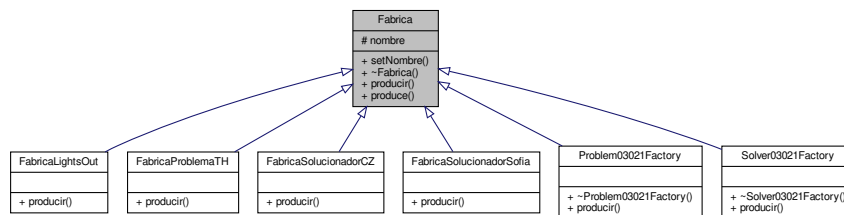
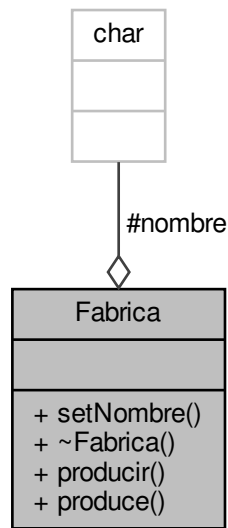


Diagrama de colaboración para Fabrica:



## 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

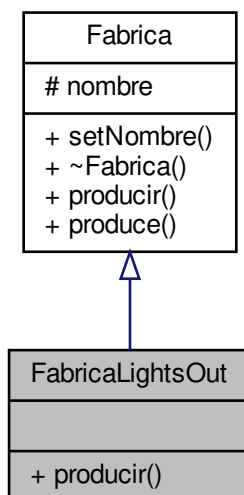
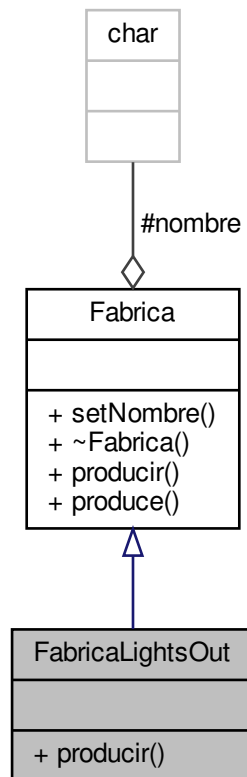


Diagrama de colaboración para FabricaLightsOut:



## 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

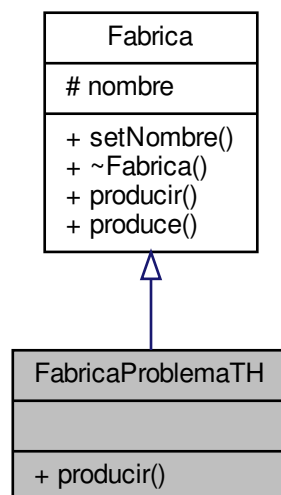
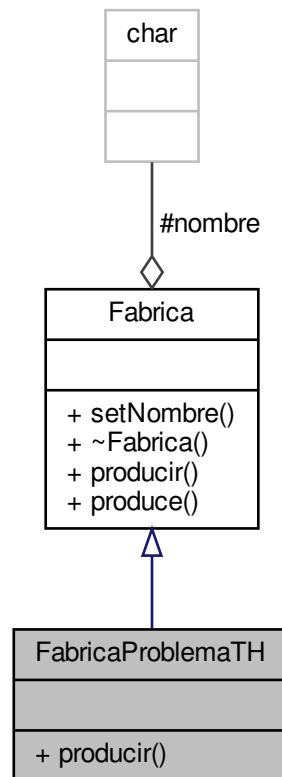


Diagrama de colaboración para `FabricaProblemaTH`:



## 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

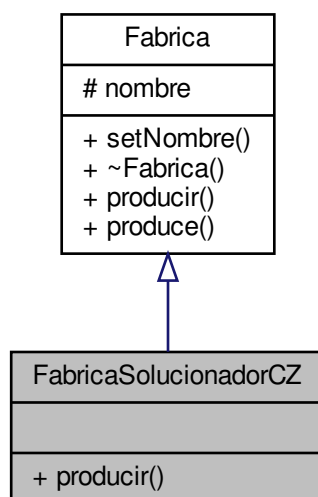
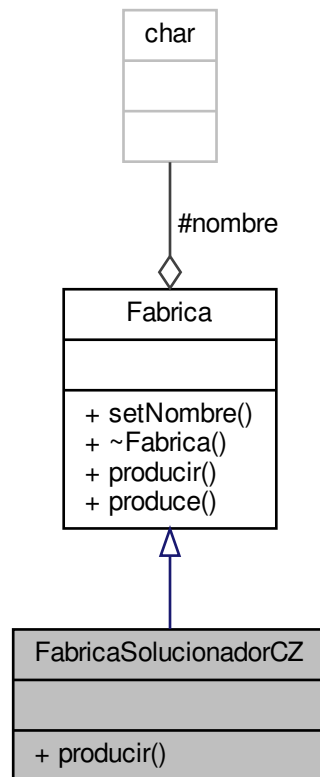


Diagrama de colaboración para FabricaSolucionadorCZ:



## 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

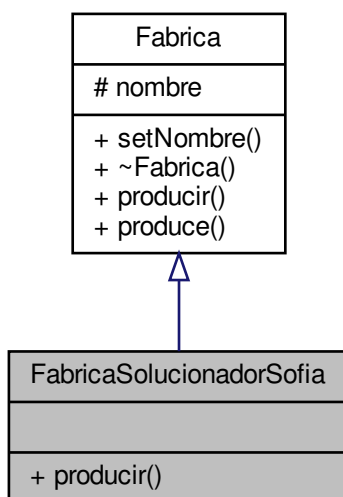
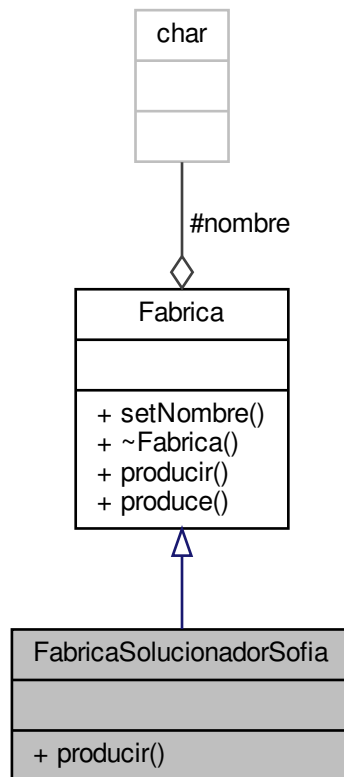


Diagrama de colaboración para `FabricaSolucionadorSofia`:



## 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

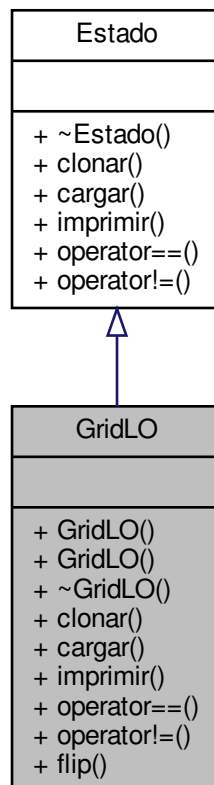
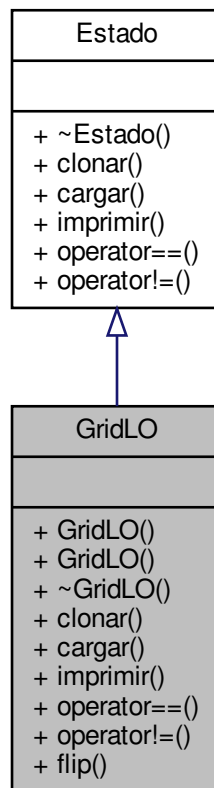


Diagrama de colaboración para GridLO:



## 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

<i>grid</i>	<a href="#">GridLO</a> 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

<i>in</i>	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

<i>row</i>	Fila de la casilla invertida
<i>column</i>	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**

<i>out</i>	Ostream al cual se imprime
------------	----------------------------

**Devuelve**

Ostream modificado

**5.9.3.5. operator!=()**

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

Operador para determinar si dos estados son distintos.

**Parámetros**

<i>e</i>	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**

<i>e</i>	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]
```

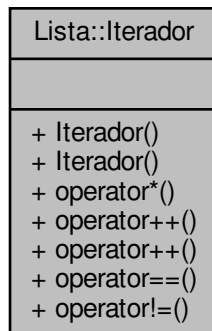
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [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. operator!=()

```
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

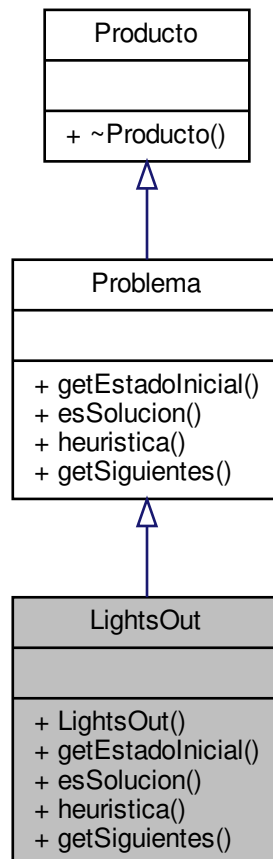
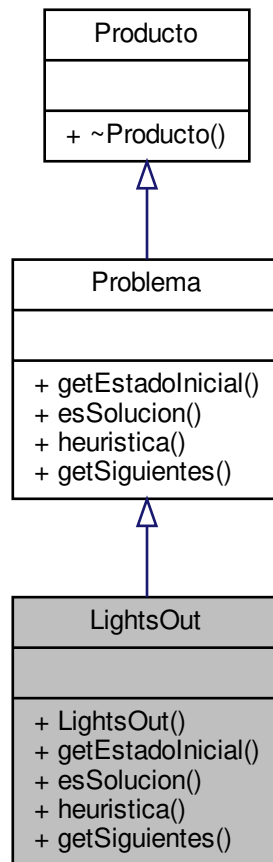


Diagrama de colaboración para LightsOut:



## 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

<i>e</i>	Puntero a <a href="#">Estado</a>
----------	----------------------------------

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**

<i>e</i>	Puntero a <a href="#">Estado</a>
----------	----------------------------------

**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**

<i>e</i>	Puntero a <a href="#">Estado</a>
----------	----------------------------------

**Devuelve**

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

Implementa [Problema](#).

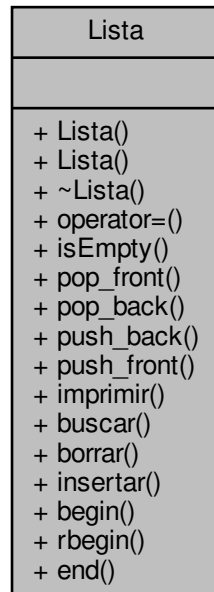
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [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](#)

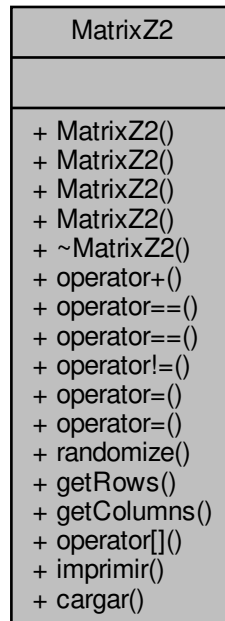


## 5.13. Referencia de la Clase MatrixZ2

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

<i>rows</i>	Cantidad de filas
<i>columns</i>	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

<i>rowsColumns</i>	Cantidad de filas y de columnas
--------------------	---------------------------------

### 5.13.2.4. MatrixZ2() [4/4]

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

Constructor de copia.

#### Parámetros

<i>matrix</i>	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**

<i>in</i>	Istream de donde se carga
-----------	---------------------------

**Devuelve**

Istream modificado

**5.13.3.2. getColumnns()**

```
int MatrixZ2::getColumnns ( )
```

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**

<i>out</i>	Ostream donde se imprime
------------	--------------------------

**Devuelve**

Ostream modificado

**5.13.3.5. operator!=()**

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

Determina si dos matrices son diferentes.

**Parámetros**

<i>matriz</i>	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**

<i>matrix</i>	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

<i>value</i>	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

<i>matriz</i>	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

<i>b</i>	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

<i>matriz</i>	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**

<i>n</i>	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.

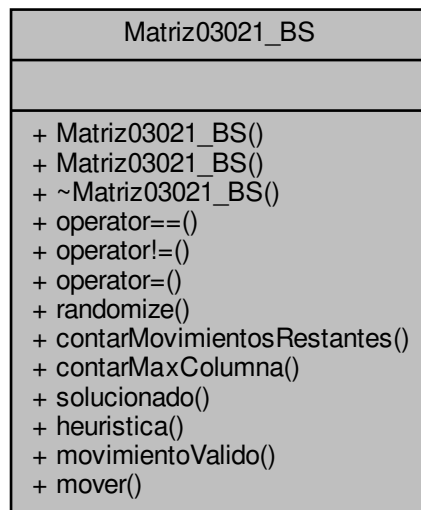
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [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](#) ()  
*Método 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 entre los índices del vector Colores(atributo de Matriz03021\_BS) para ser asignados si estos tienen menos de 4 apariciones en la matriz.*
- int [contarMovimientosRestantes](#) ()  
*Método que cuenta la cantidad mínima de ordenaciones que se tienen que dar para que la matriz esté ordenada.*
- int [contarMaxColumna](#) (int)  
*Método que cuenta la cantidad de apariciones máxima de un carácter en una columna.*
- int [solucionado](#) ()  
*Método que verifica si la matriz está ordenada completamente (solucionada) o no.*
- int [heuristica](#) ()



*Método que calcula la heurística optimista según el contenido de la matriz.*

- `int movimientoValido (int, int)`

*Método 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 carácter 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

<i>otra</i>	Recibe un objeto <code>Matriz03021_BS</code> 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 )
```

Método que cuenta la cantidad de apariciones máxima de un carácter en una columna.

##### Parámetros

<i>columna</i>	Recibe el subíndice de la columna que se desea saber la cantidad de apariciones máxima de un carácter.
----------------	--

##### Devuelve

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

#### 5.14.2.2. contarMovimientosRestantes()

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

Método 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 ( )
```

Método 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 uno de un carácter de una columna hacia otra dependiendo si es válido o no.

**Parámetros**

<i>columnaOrigen</i>	Recibe el número de la columna de la que se quiere mover el elemento.
<i>columnaDestino</i>	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 )
```

Método 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**

<i>columnaOrigen</i>	Recibe el número de la columna de la que se quiere mover el elemento.
<i>columnaDestino</i>	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. operator!=()**

```
int Matriz03021_BS::operator!= (
    Matriz03021\_BS & otra )
```

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

**Parámetros**

<i>otra</i>	Recibe un objeto <a href="#">Matriz03021_BS</a> 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

<i>otra</i>	Recibe un objeto <a href="#">Matriz03021_BS</a> 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

<i>otra</i>	Recibe un objeto <a href="#">Matriz03021_BS</a> 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 ( )
```

Método 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 entre los índices 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 ( )
```

Método 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

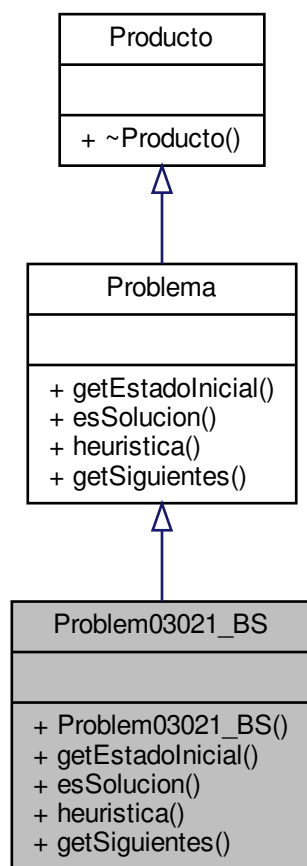
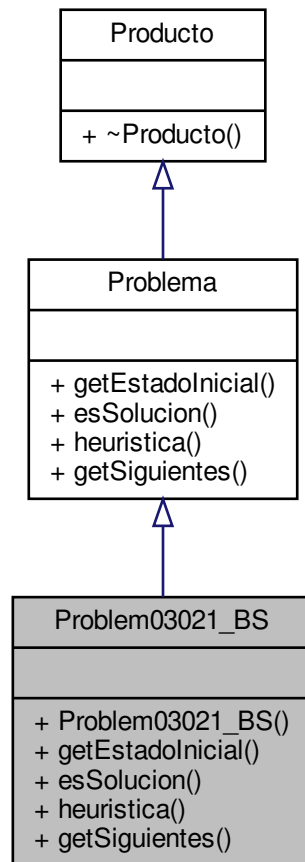


Diagrama de colaboración para Problem03021\_BS:



## 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

<i>estado</i>	Recibe un puntero a <a href="#">Estado</a> 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. getSiguietes()

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

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



## Parámetros

<i>estado</i>	Recibe un puntero a <a href="#">Estado</a> 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

<i>estado</i>	Recibe un puntero a <a href="#">Estado</a> 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

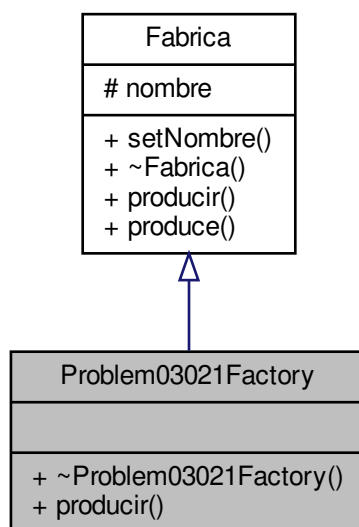
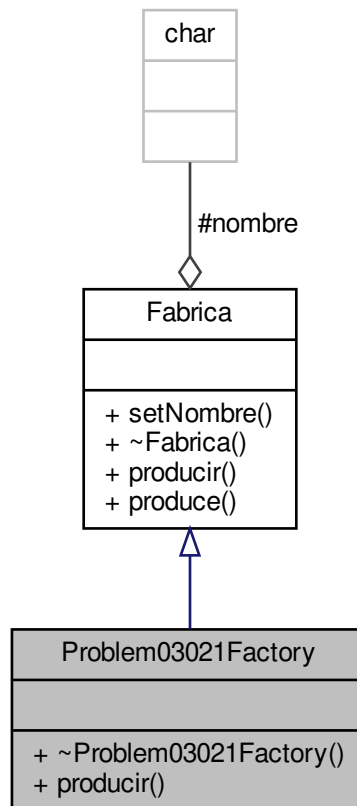


Diagrama de colaboración para Problem03021Factory:



## 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

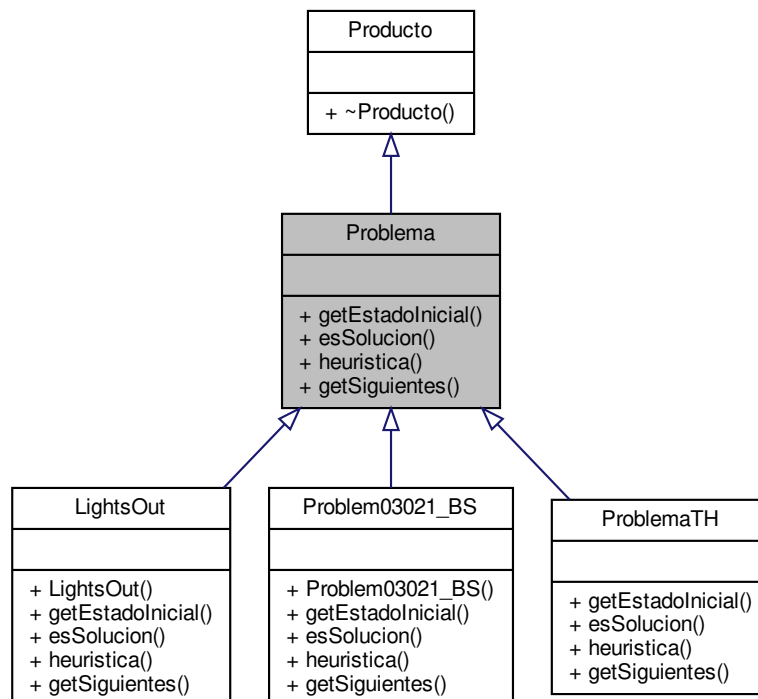
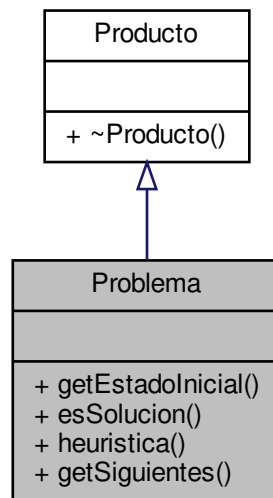


Diagrama de colaboración para Problema:



## Métodos públicos

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

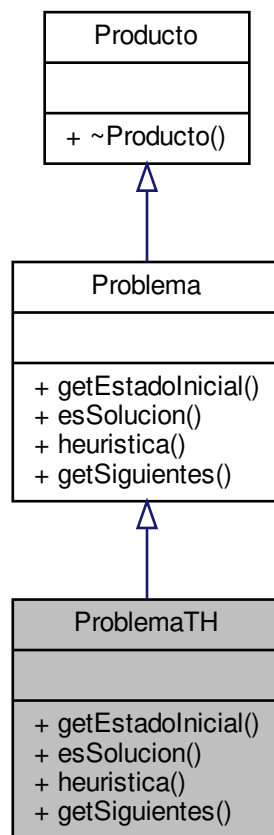
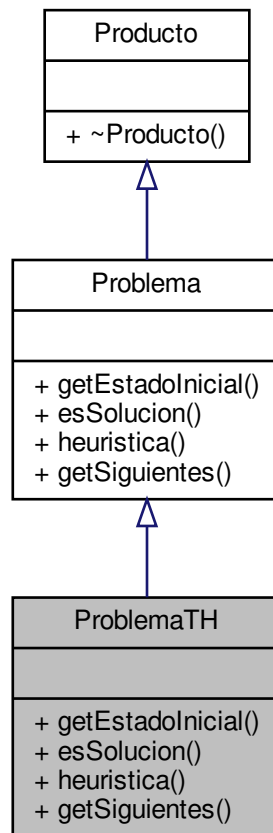


Diagrama de colaboración para ProblemaTH:



## 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

<i>e</i>	Puntero a <a href="#">Estado</a>
----------	----------------------------------

#### 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. getSiguietes()

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

Determina todos los posibles estados siguientes apartir de un estado.

#### Parámetros

<i>e</i>	Puntero a <a href="#">Estado</a>
----------	----------------------------------

**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 <a href="#">Estado</a>
---	----------------------------------

**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

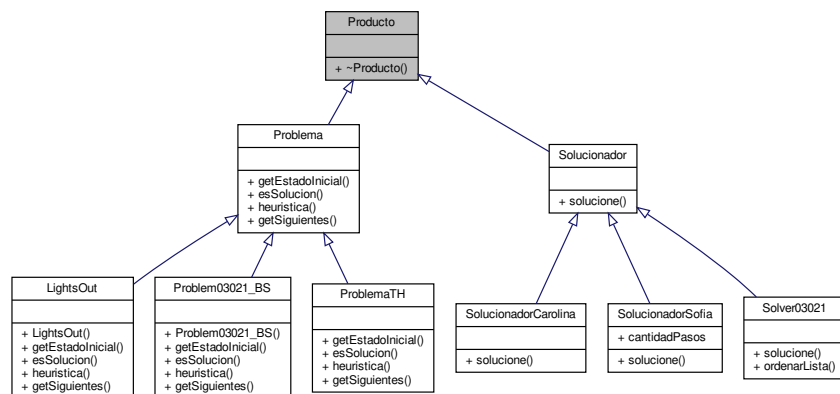
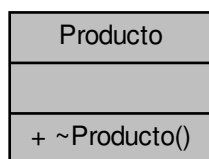


Diagrama de colaboración para Producto:



## 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

<i>p</i>	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

<i>hilera</i>	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"

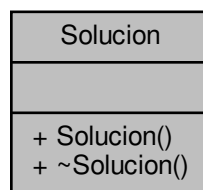
La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [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

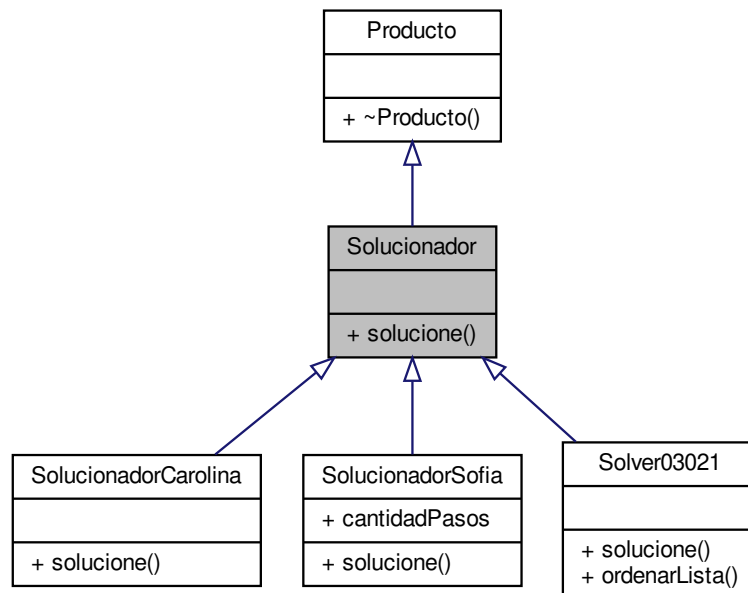
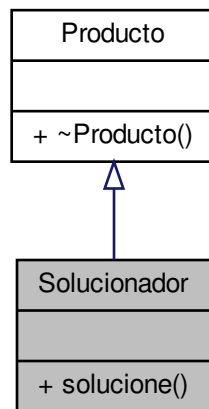


Diagrama de colaboración para Solucionador:



## 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

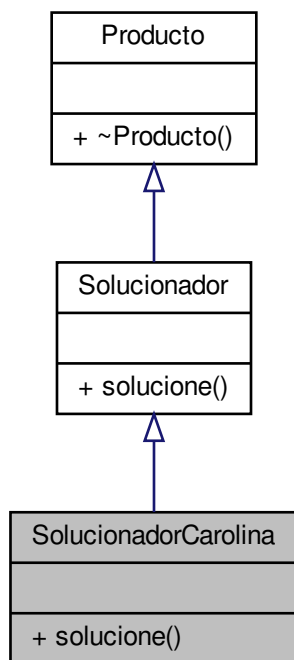
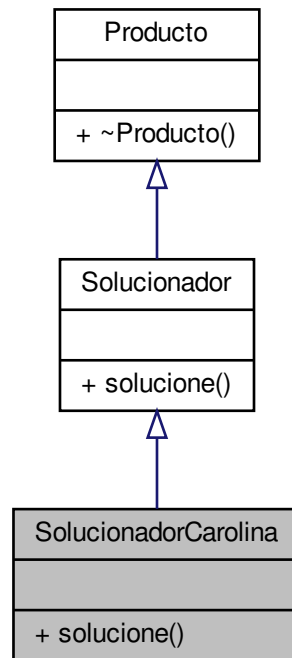


Diagrama de colaboración para SolucionadorCarolina:



## 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

<i>problema</i>	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

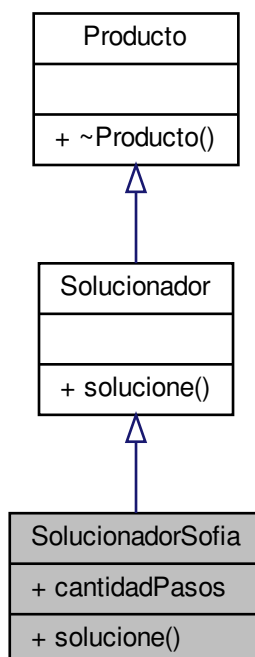
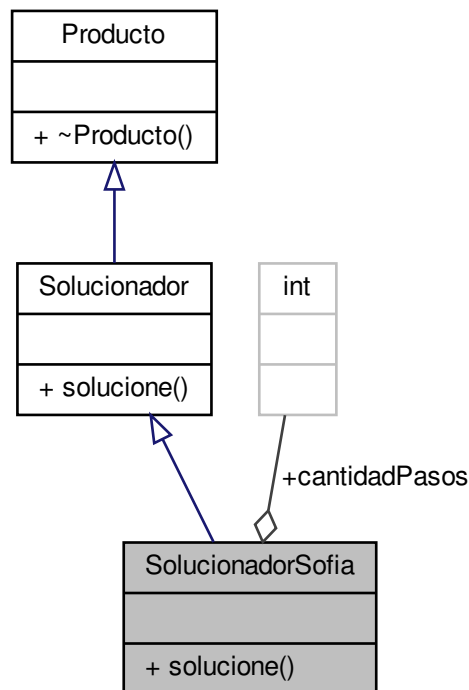


Diagrama de colaboración para SolucionadorSofia:



### 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

<i>problema</i>	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

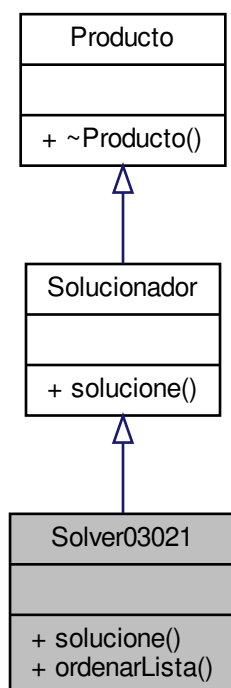
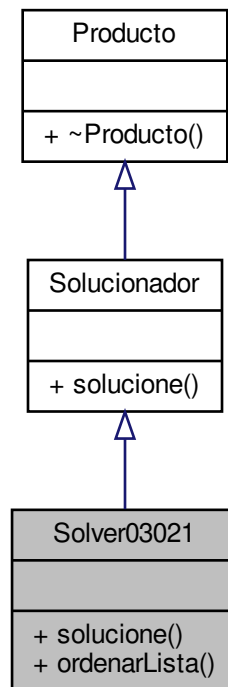


Diagrama de colaboración para Solver03021:



## 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

<i>lista</i>	Recibe una lista por puntero, representa la lista que se quiere ordenar.
<i>problema</i>	Recibe un <a href="#">Problema</a> 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

<i>problema</i>	Recibe un puntero a <a href="#">Problema</a> 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

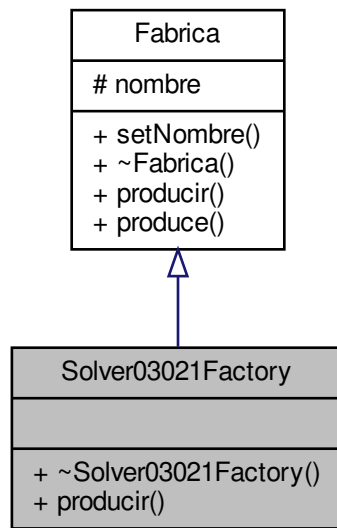
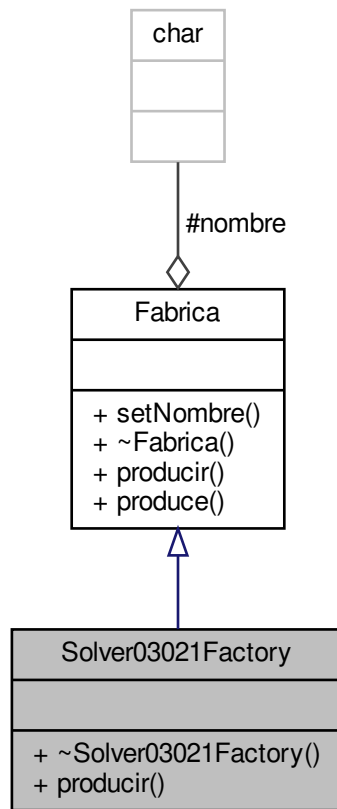


Diagrama de colaboración para Solver03021Factory:



## 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

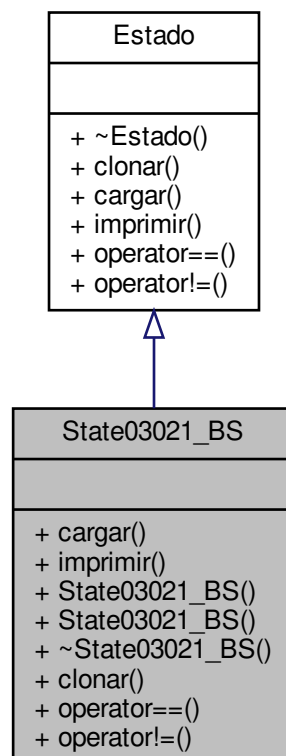
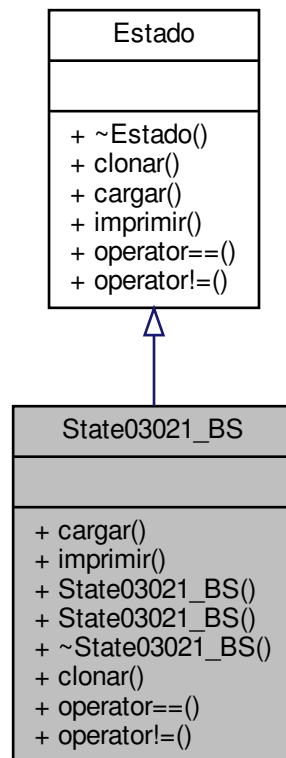


Diagrama de colaboración para State03021\_BS:



## 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

<i>otra</i>	Recibe un objeto <a href="#">Matriz03021_BS</a> 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**

<i>entrada</i>	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 características 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**

<i>salida</i>	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. operator!=()

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

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

##### Parámetros

<i>estado</i>	Recibe un puntero a <a href="#">Estado</a> 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

<i>estado</i>	Recibe un puntero a <a href="#">Estado</a> 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](#)





## Documentación de archivos

```
#include <iostream>
```

```
graph TD; Estado[Estado.h] --> iostream[iostream];
```

[illegible]

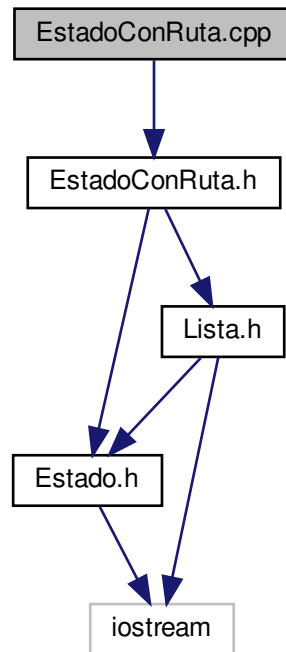
- class Estado

## 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:

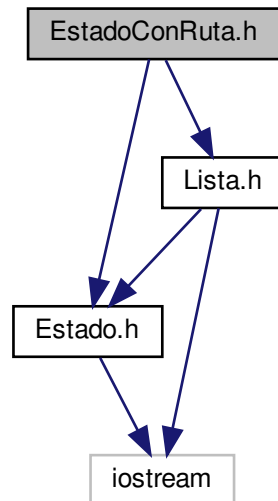
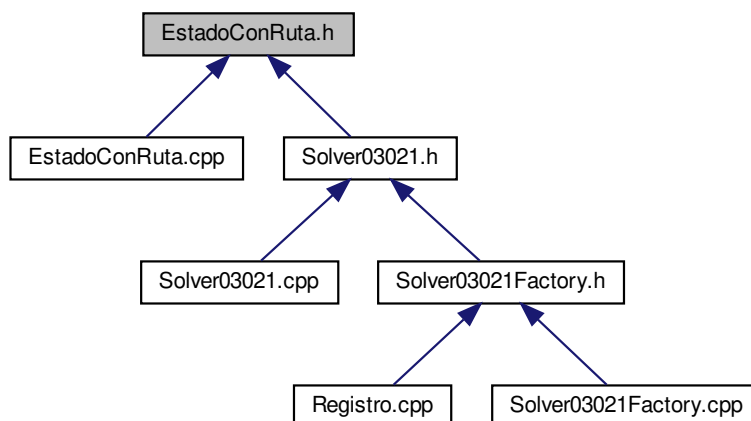


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



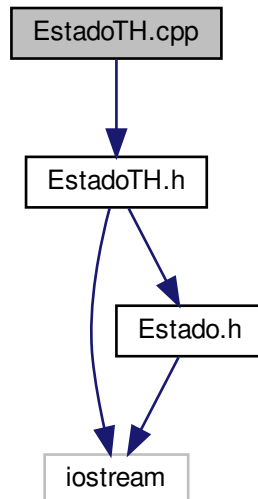
## Clases

- 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:

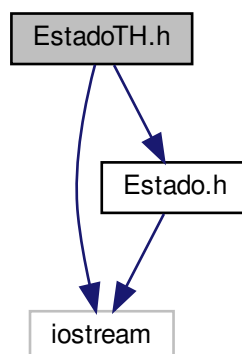
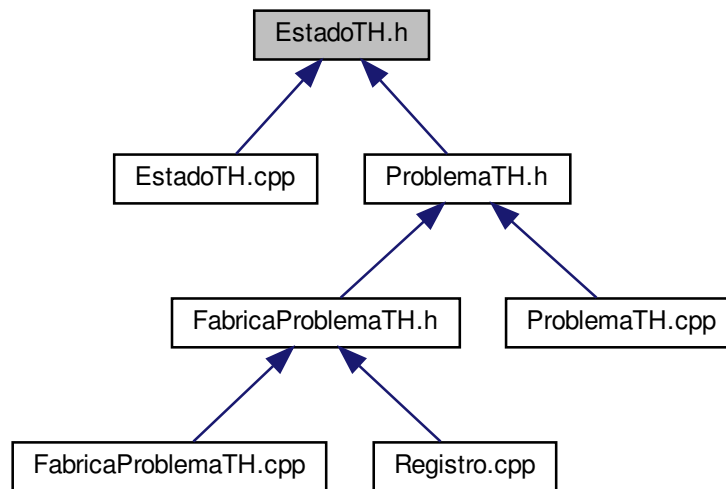


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## 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:

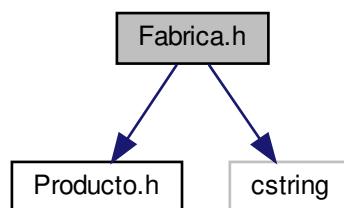
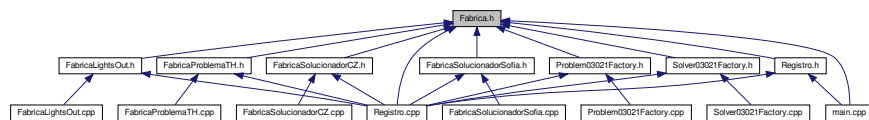


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [Fabrica](#)

## defines

- #define [NOMBRE\\_MAX\\_SIZE](#) 100

### 6.6.1. Documentación de los 'defines'

#### 6.6.1.1. NOMBRE\_MAX\_SIZE

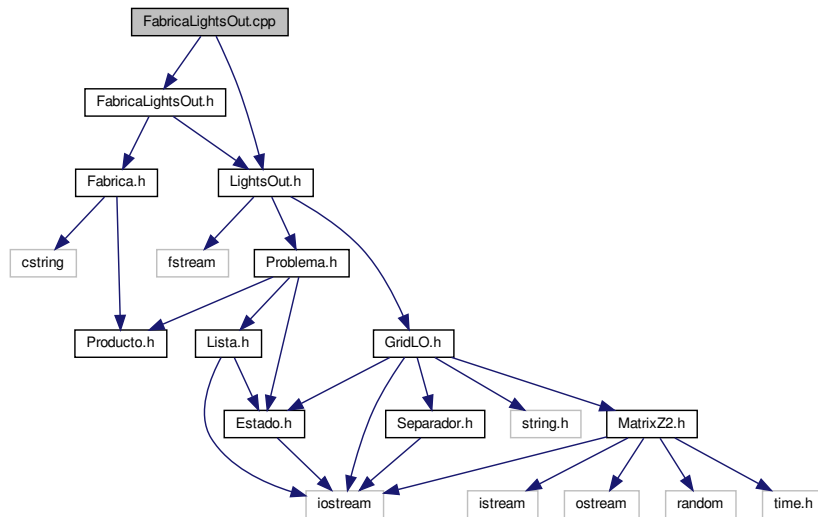
```
#define NOMBRE_MAX_SIZE 100
```

## 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:

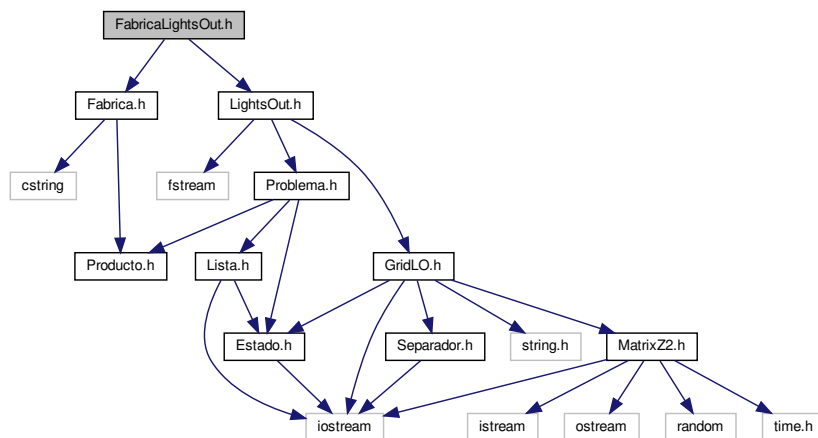
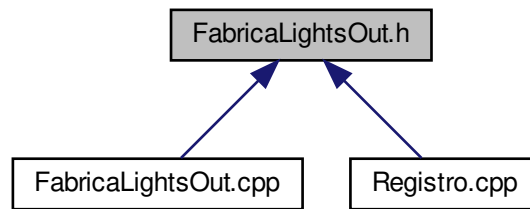


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [FabricaLightsOut](#)

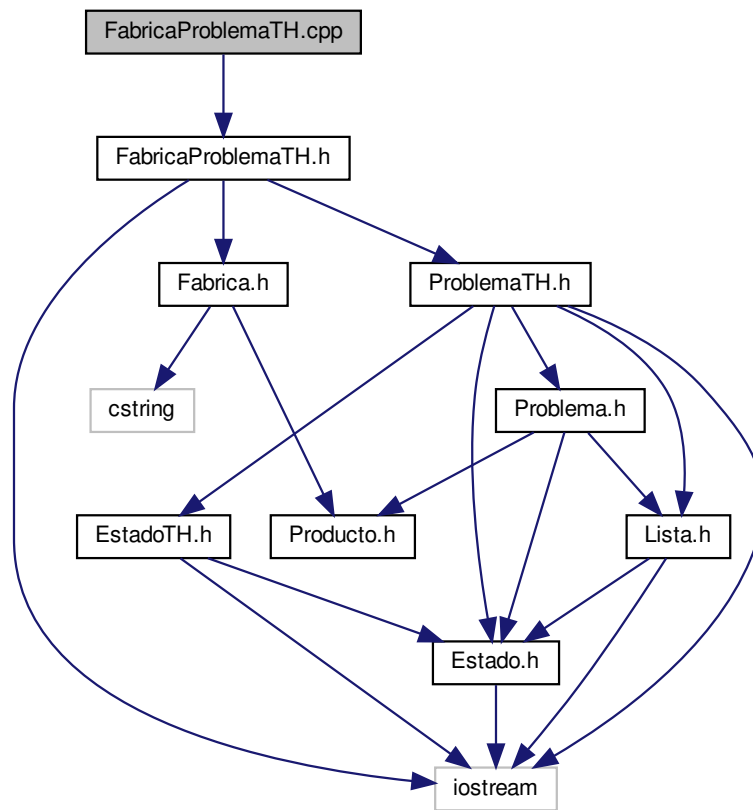
*Clase derivada de [Fabrica](#), para producir un puntero a [LightsOut](#).*

## 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:

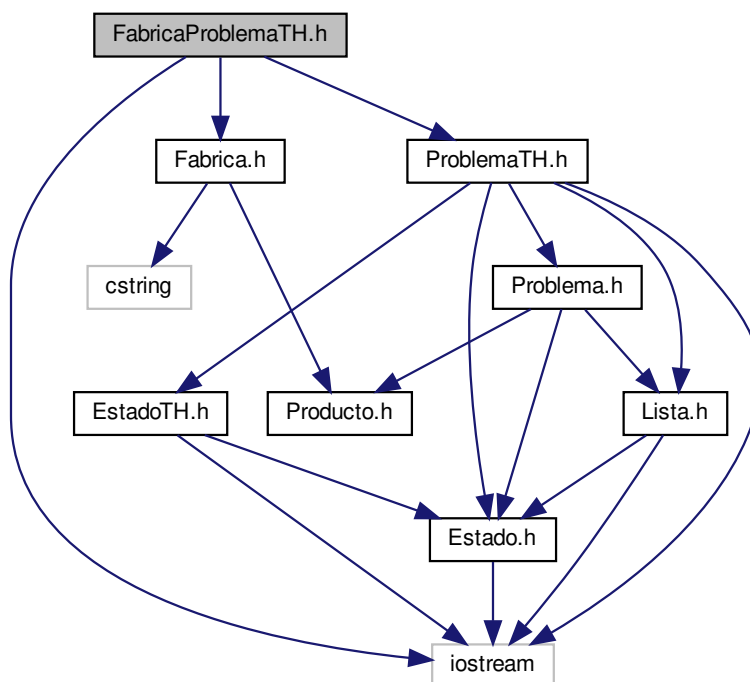
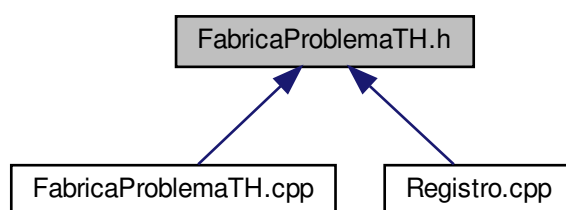


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class `FabricaProblemaTH`

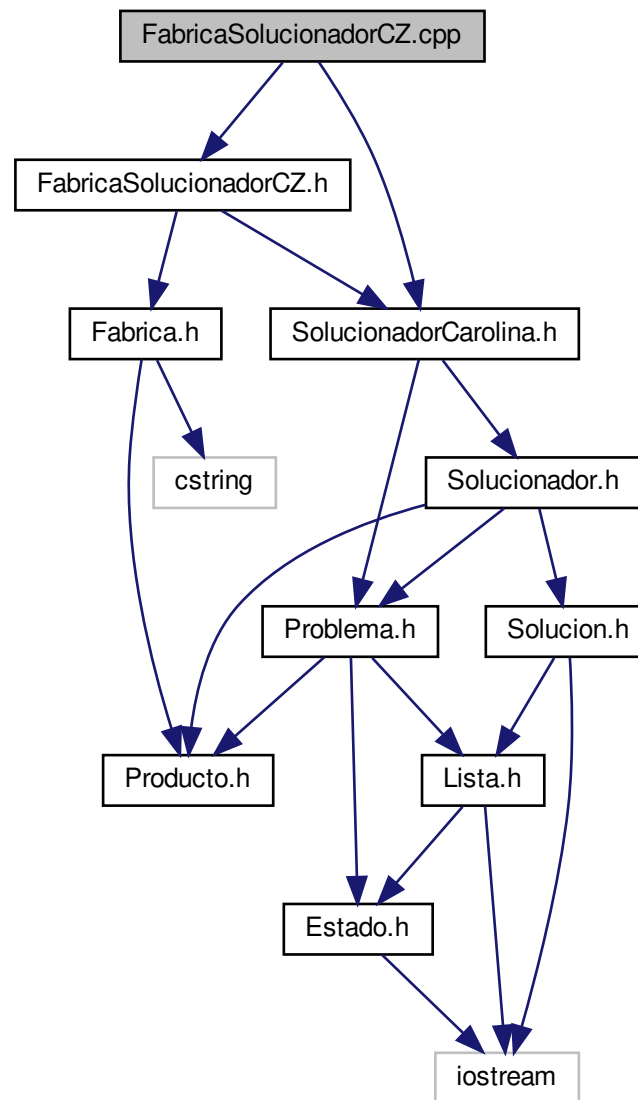
Clase derivada de `Fabrica`, produce un puntero a `ProblemaTH`.

## 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:

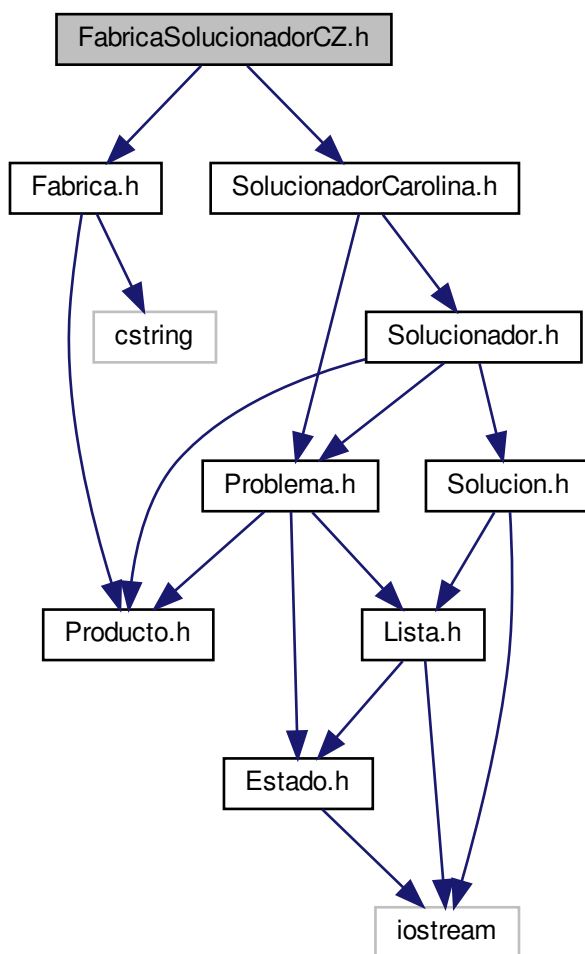
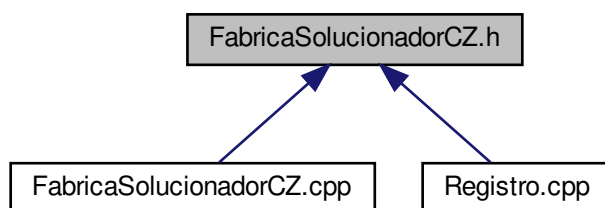


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

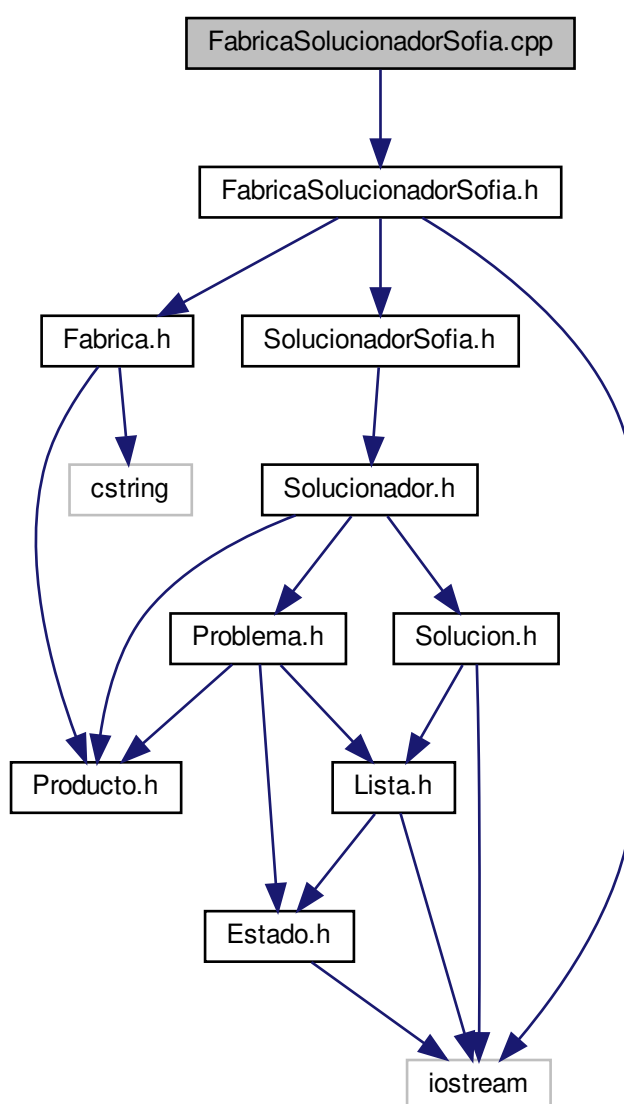
- class [FabricaSolucionadorCZ](#)

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:

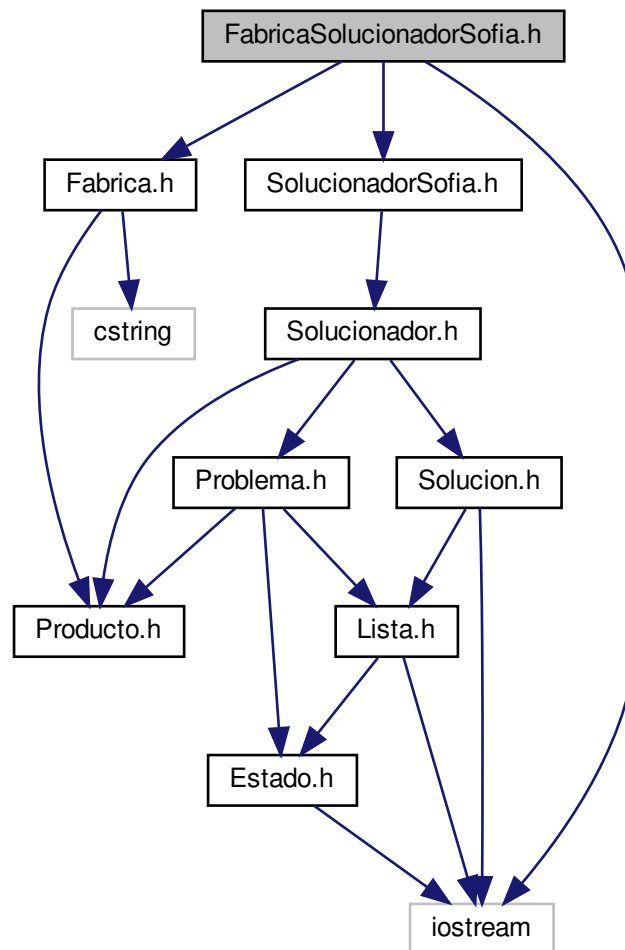
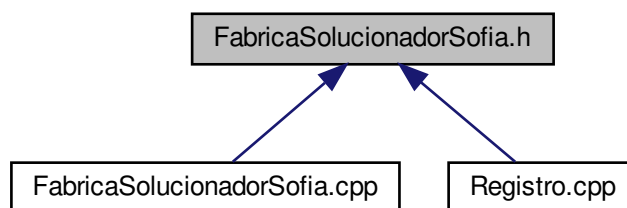


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

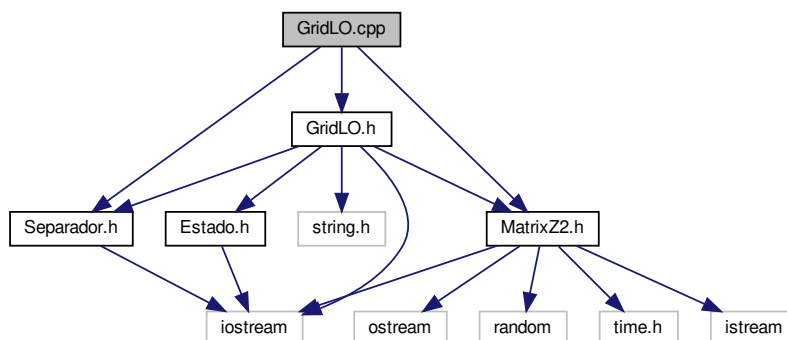
- class [FabricaSolucionadorSofia](#)

*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:

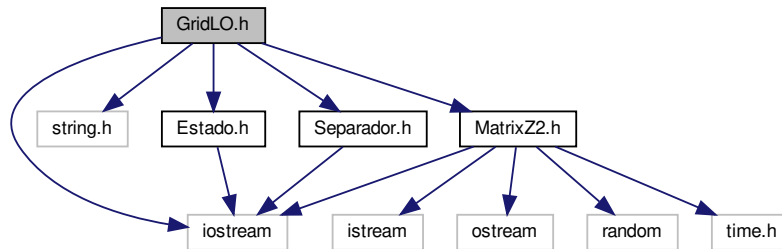
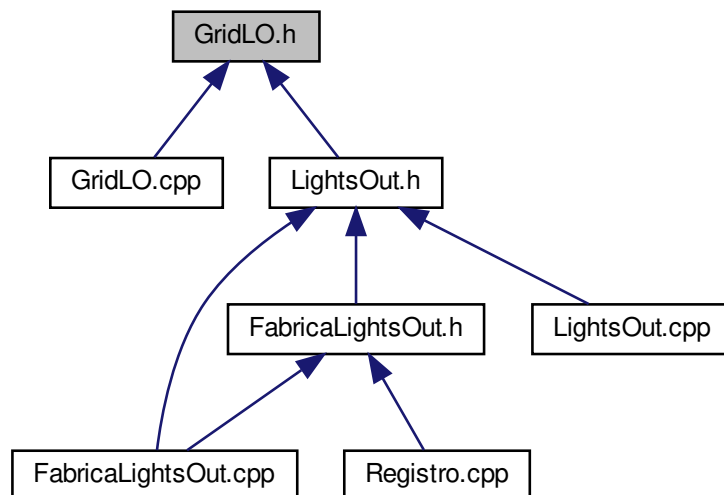


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## 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
```

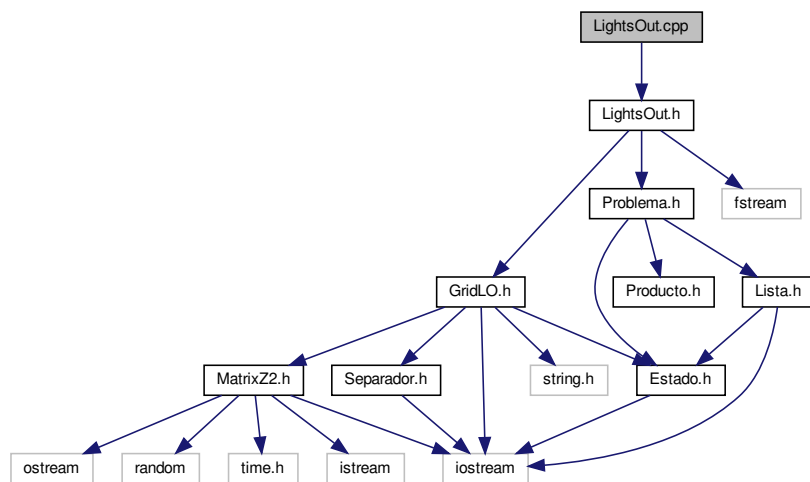
#### 6.16.1.2. STREAMSIZE\_LO

```
#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:

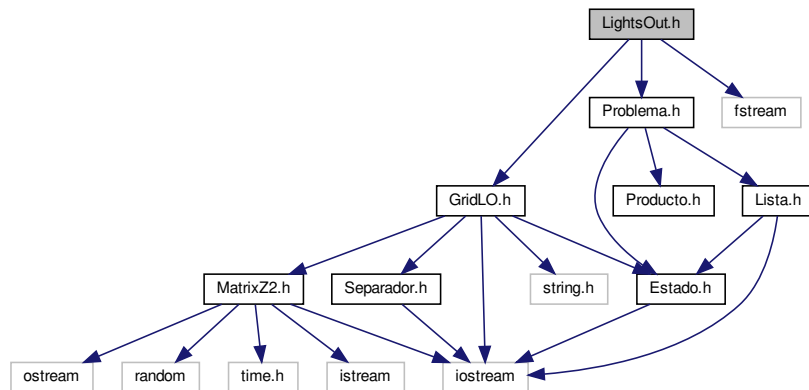
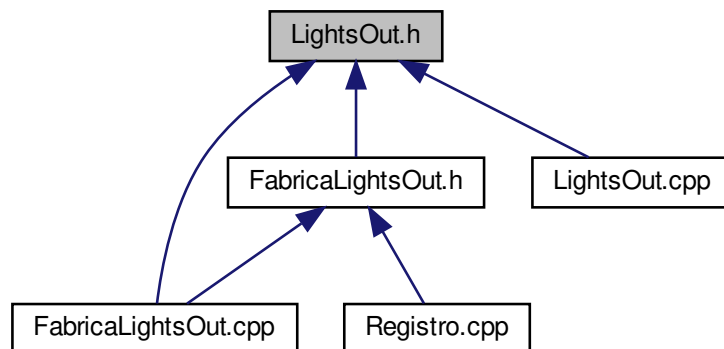


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [LightsOut](#)

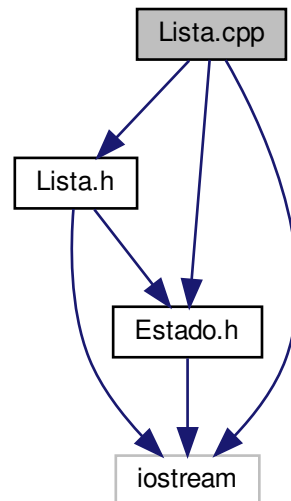
Clase derivada de [Problema](#), representa el problema [LightsOut](#).

## 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:

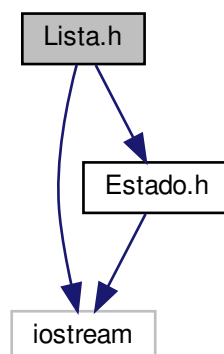
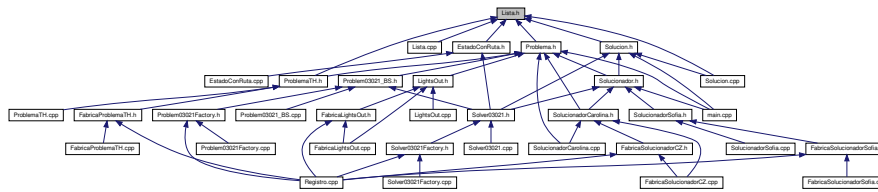


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



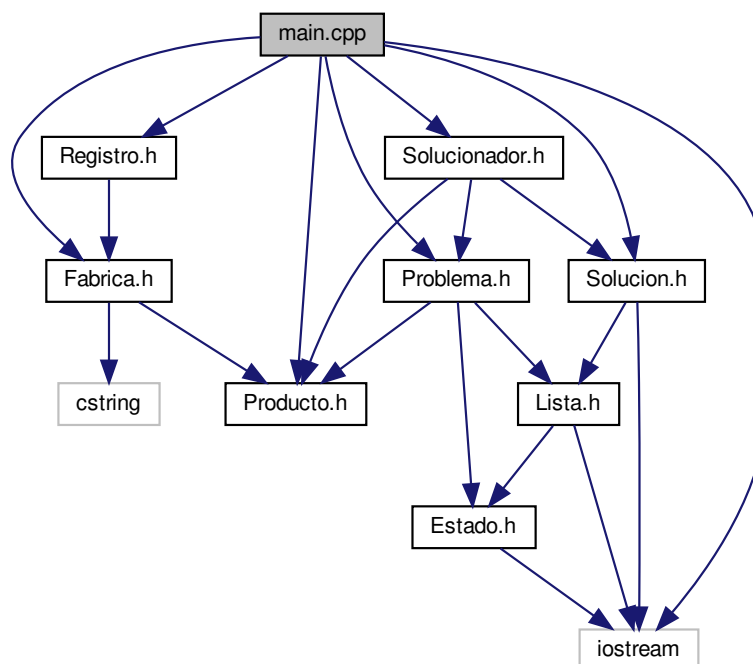
## Clases

- 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

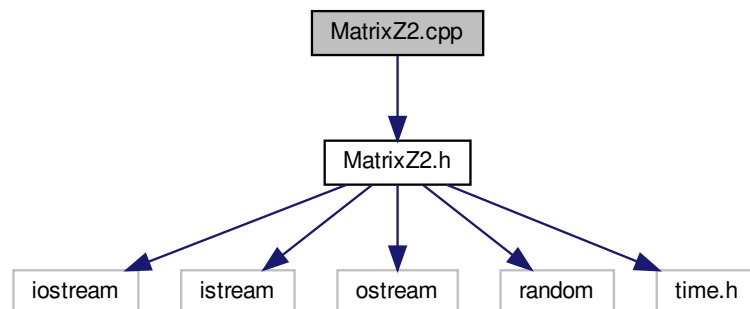
#### 6.21.1.1. `main()`

```
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:

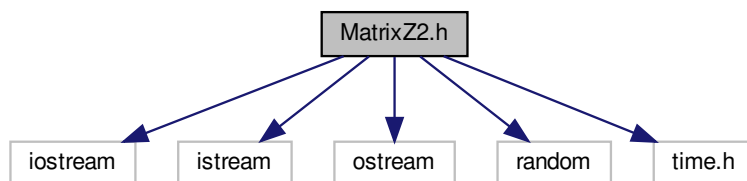
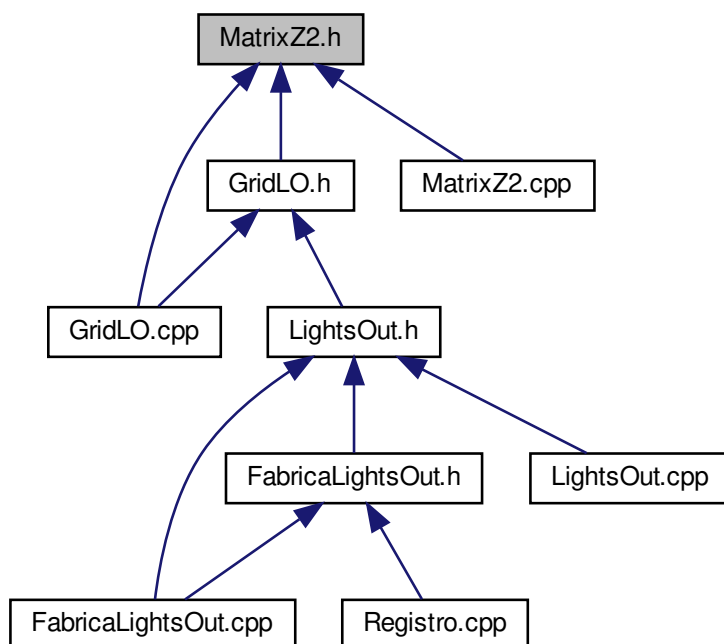


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [MatrixZ2](#)

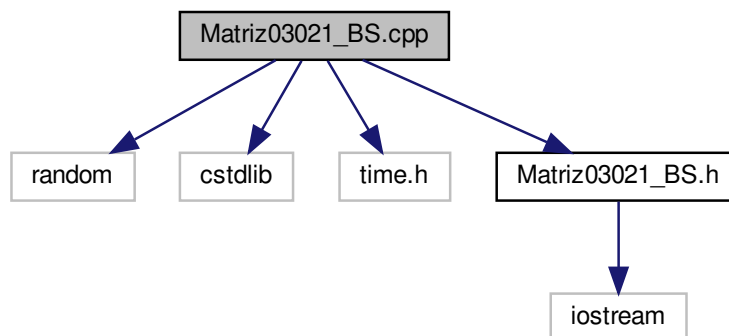
*Esta clase representa una matriz con entradas en el cuerpo Z2 (F2)*

## 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:

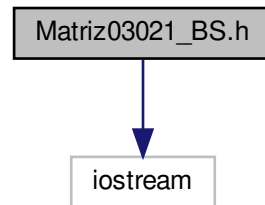
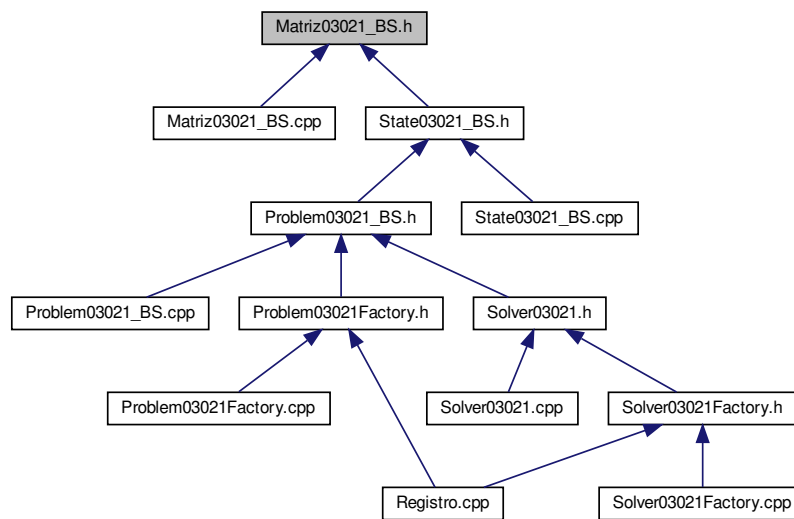


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



### 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

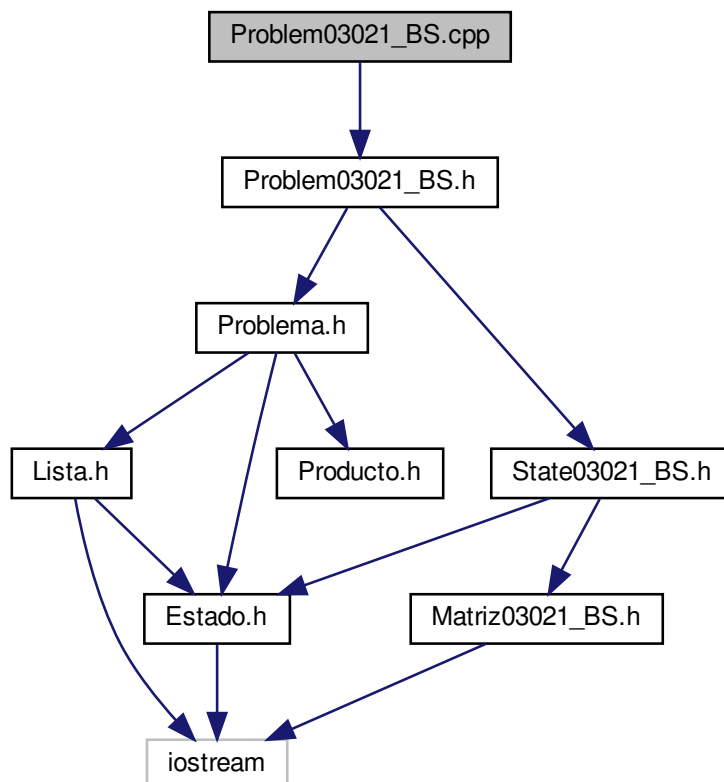
```
#define FILASBS 4
```

## 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:

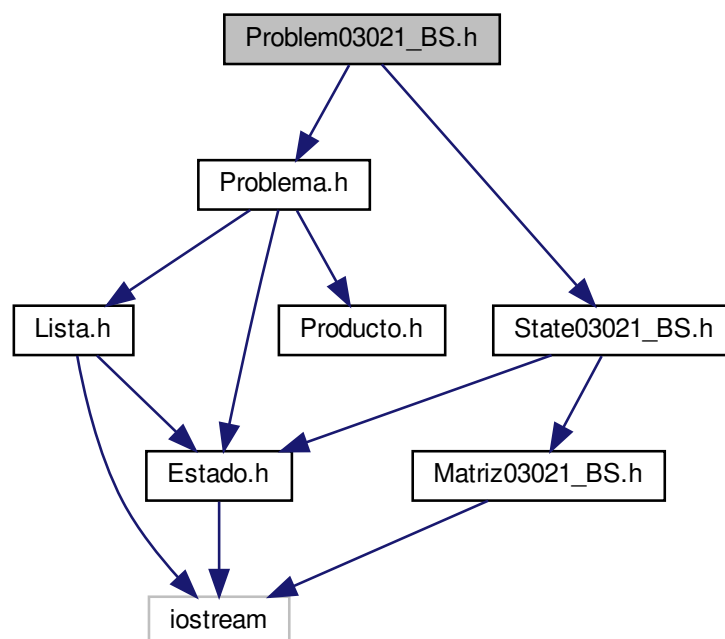
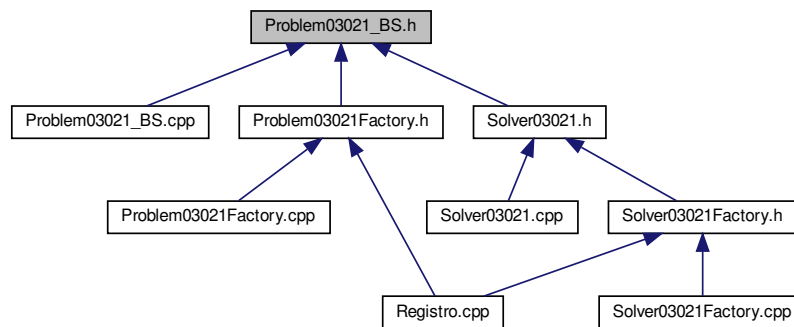


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

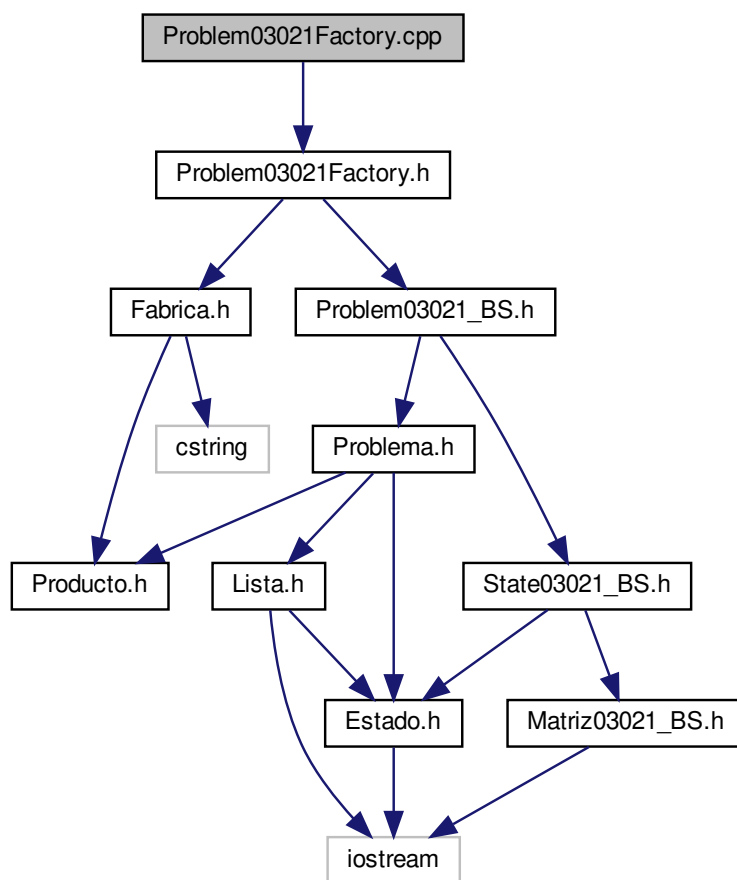
- class [Problem03021\\_BS](#)

## 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:

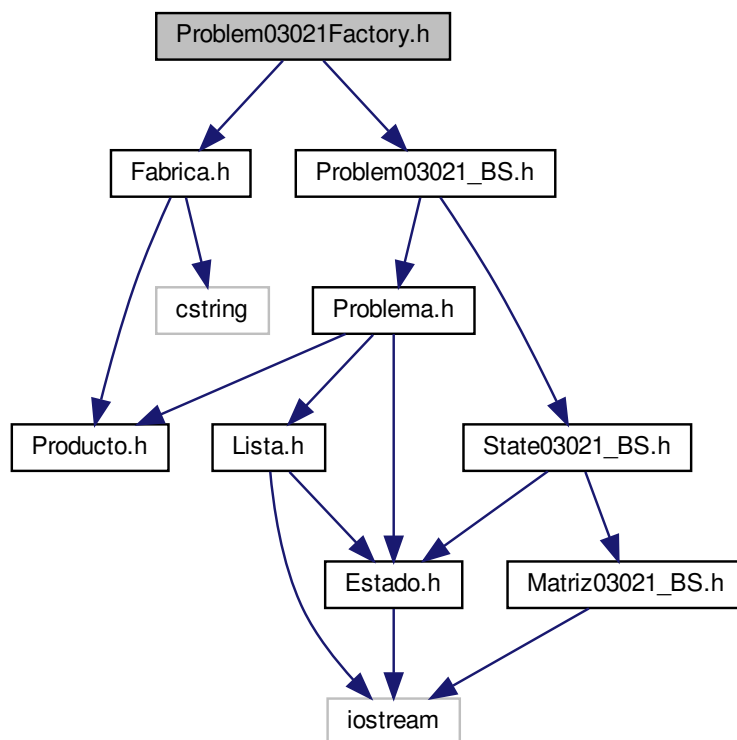
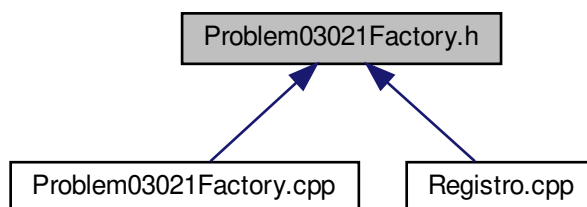


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

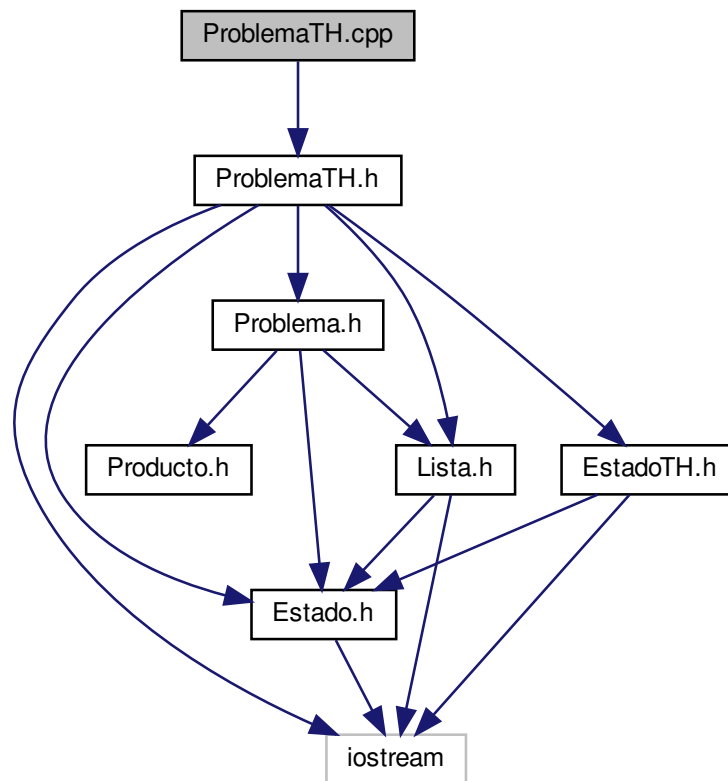


### Clases

- class [Problem03021Factory](#)



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:

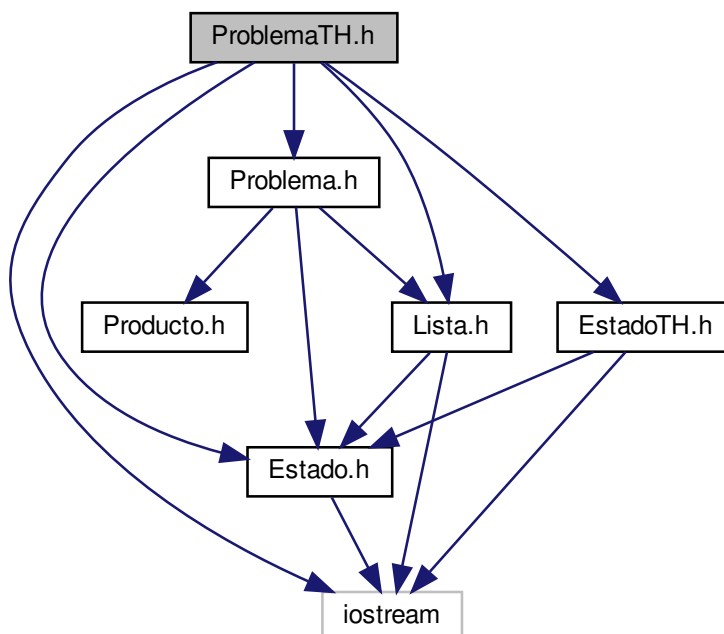
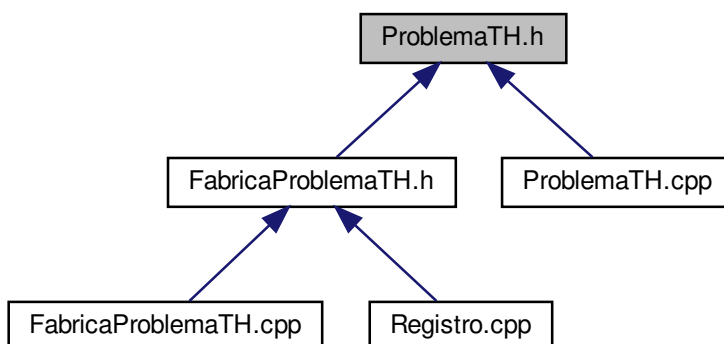


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

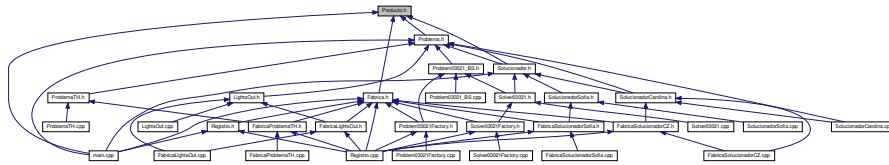
- class [ProblemaTH](#)

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

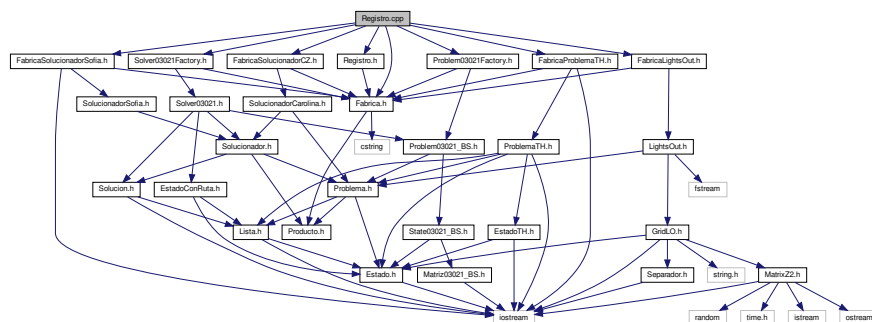
- class [Producto](#)

## 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:

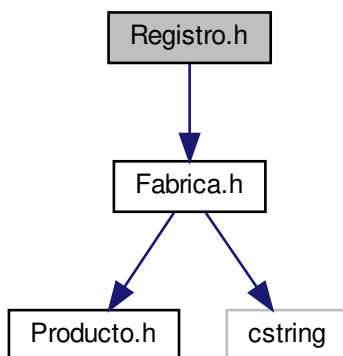
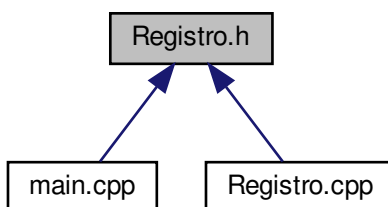


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



### Clases

- class [Registro](#)

### defines

- #define [CAPACIDAD](#) 120

#### 6.36.1. Documentación de los 'defines'

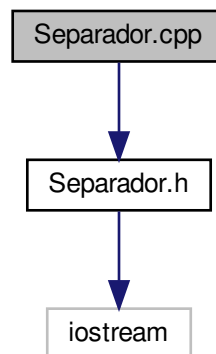
### 6.36.1.1. CAPACIDAD

```
#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:

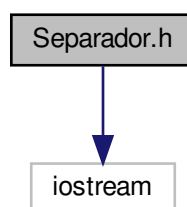
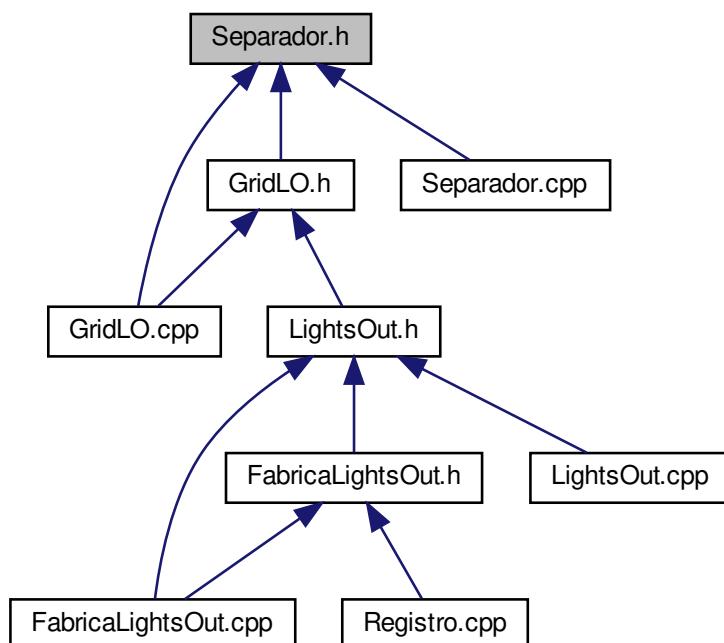


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



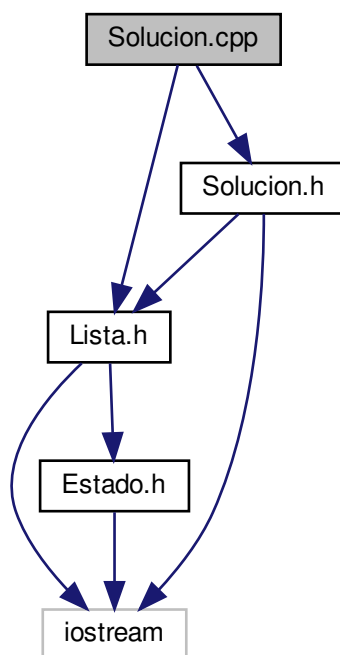
## Clases

- 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:

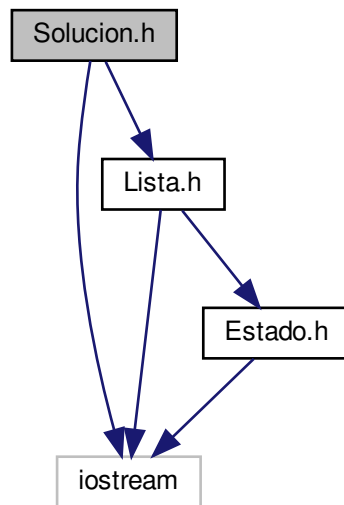
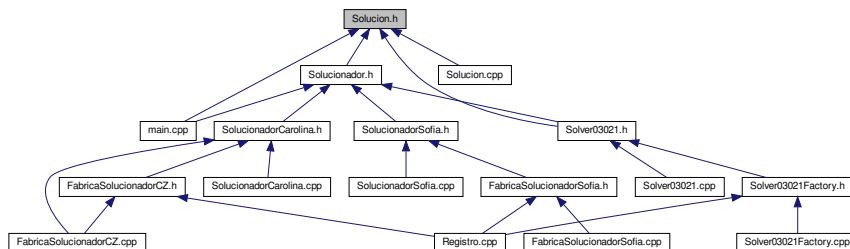


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## 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:

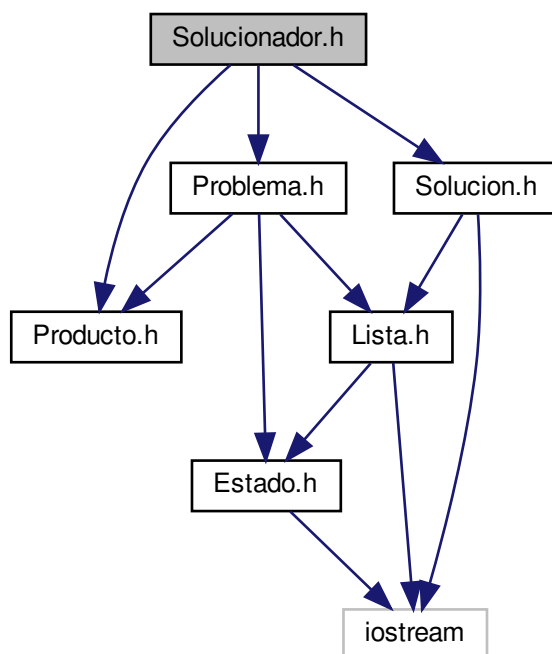
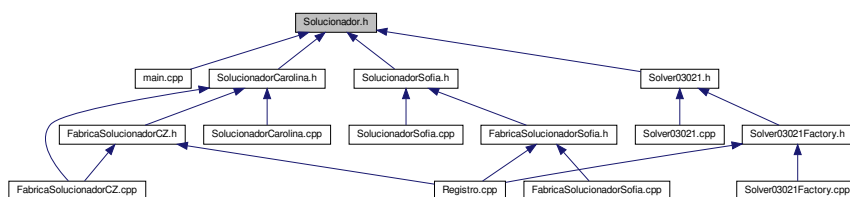


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



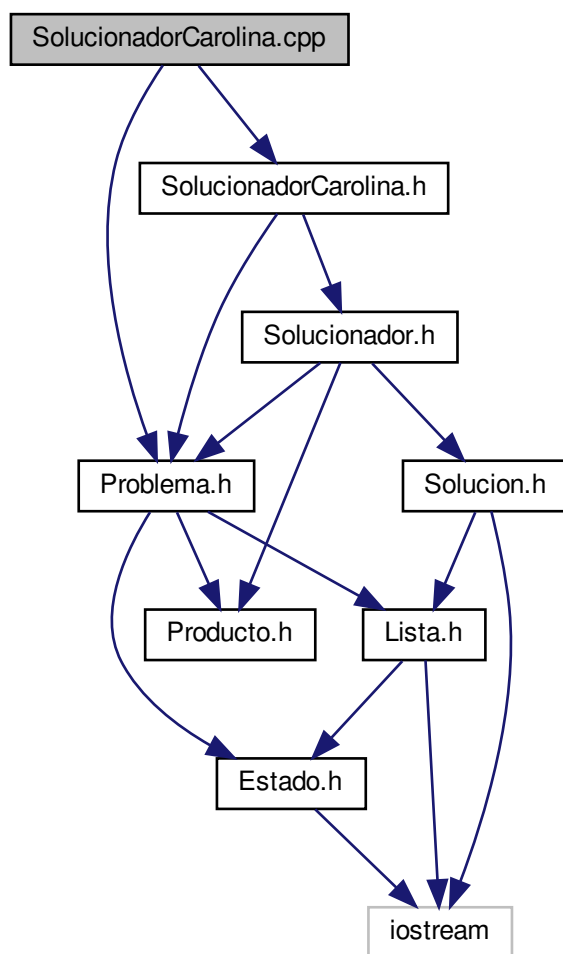
## Clases

- 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:

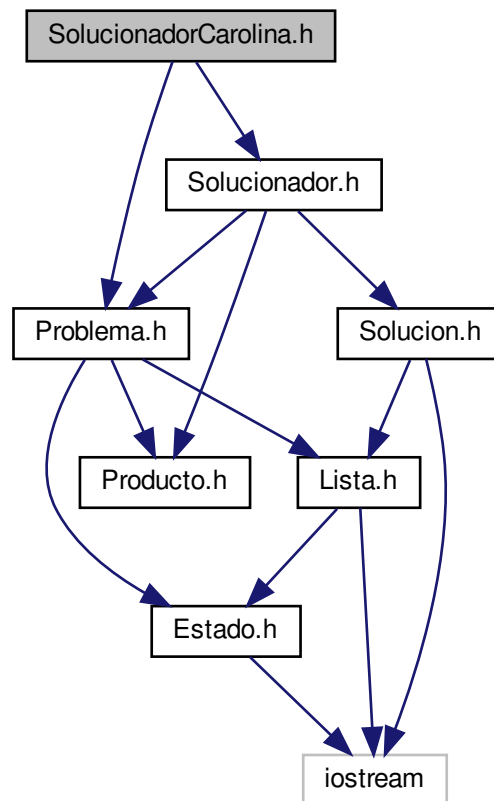
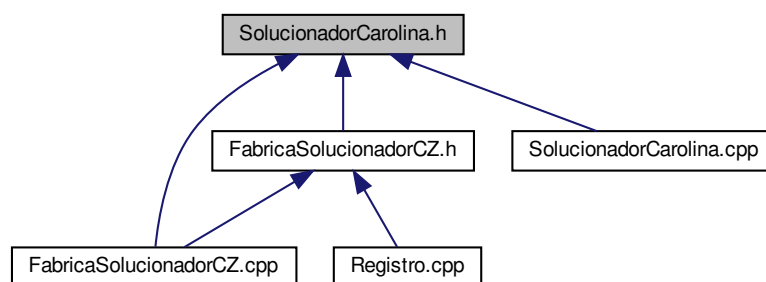


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



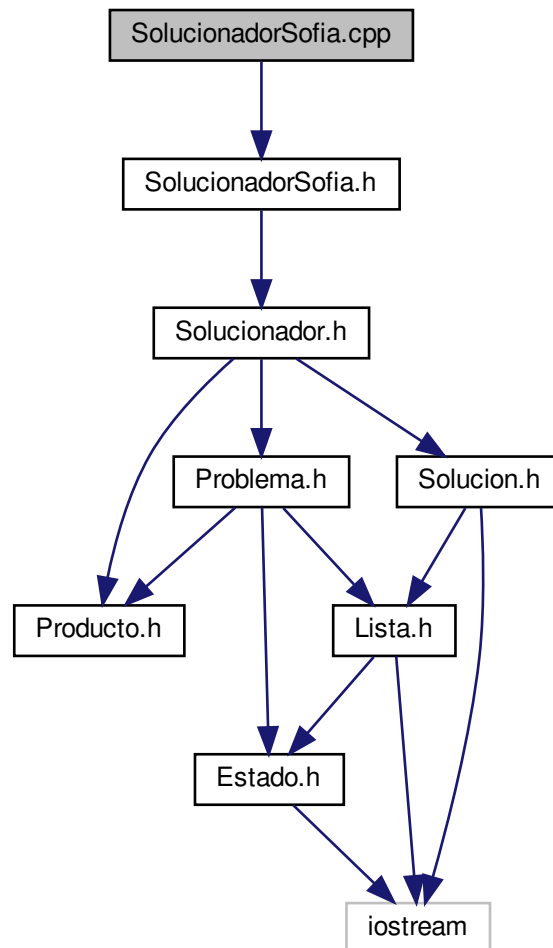
## Clases

- 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:

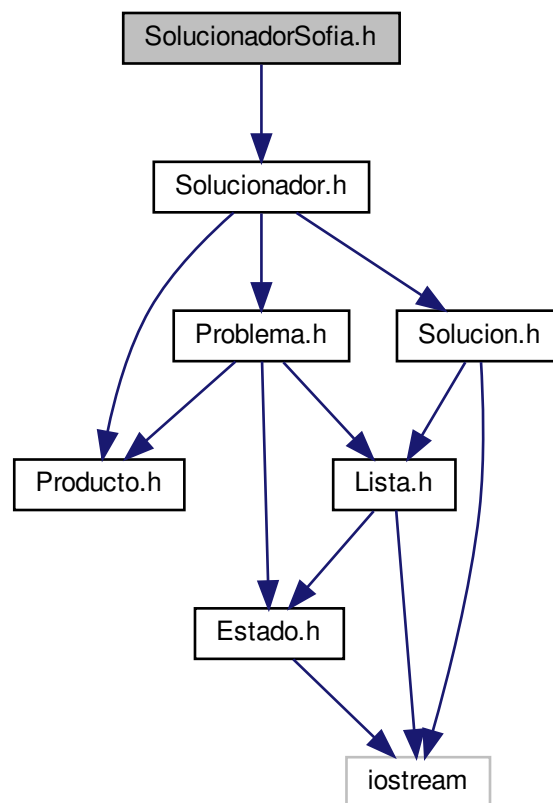
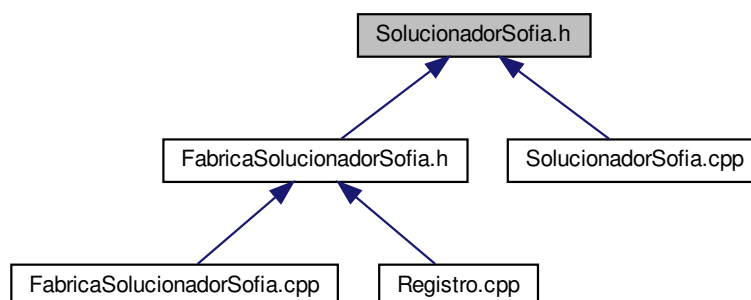


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [SolucionadorSofia](#)

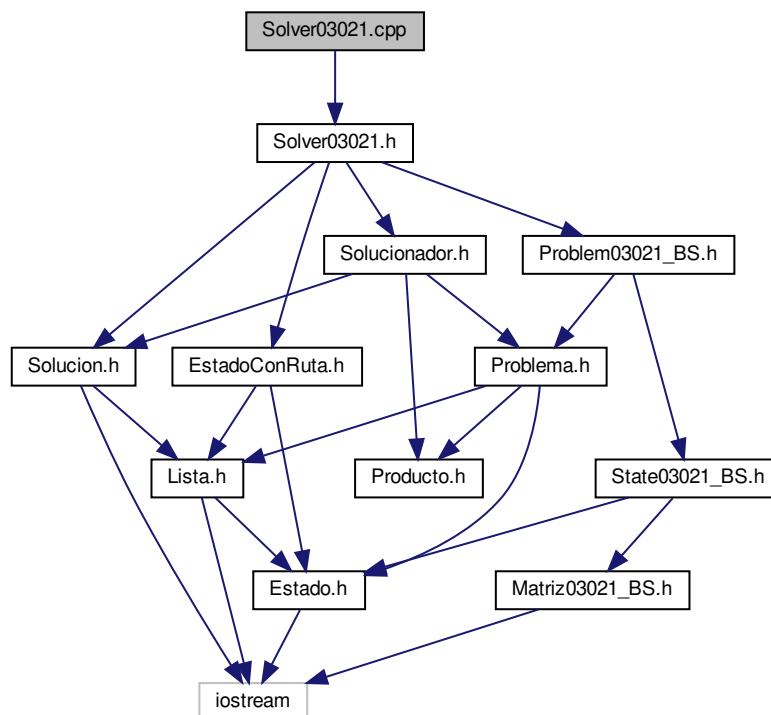
*Solucionador* de Sofia, clase derivada de *Solucionador*.

## 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 algoritmos 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 algoritmos 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:

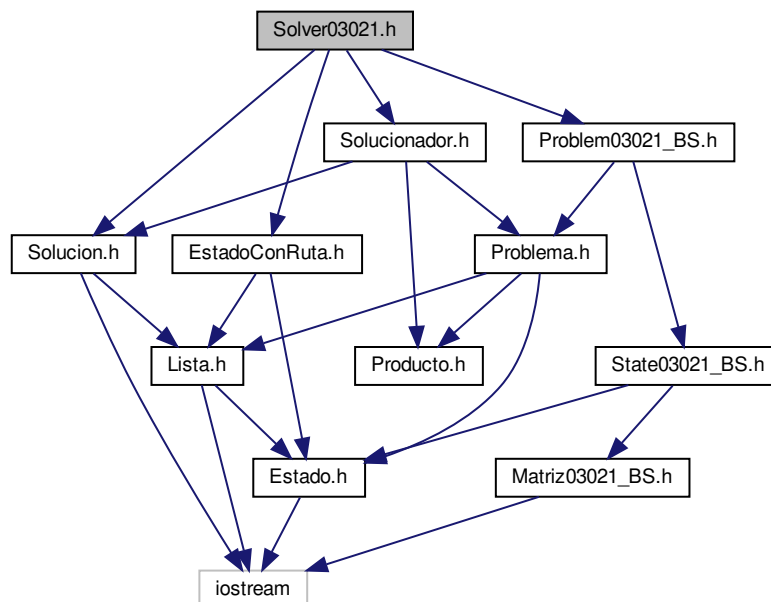
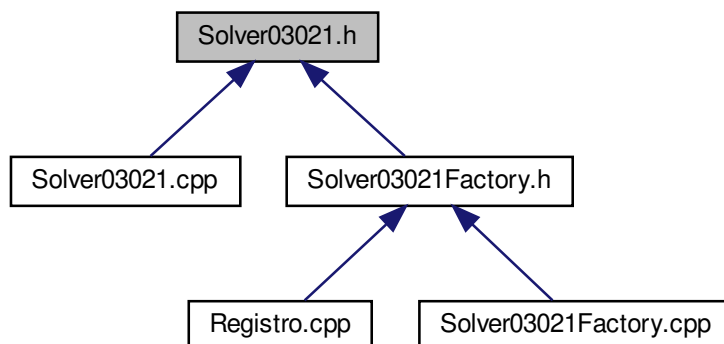


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

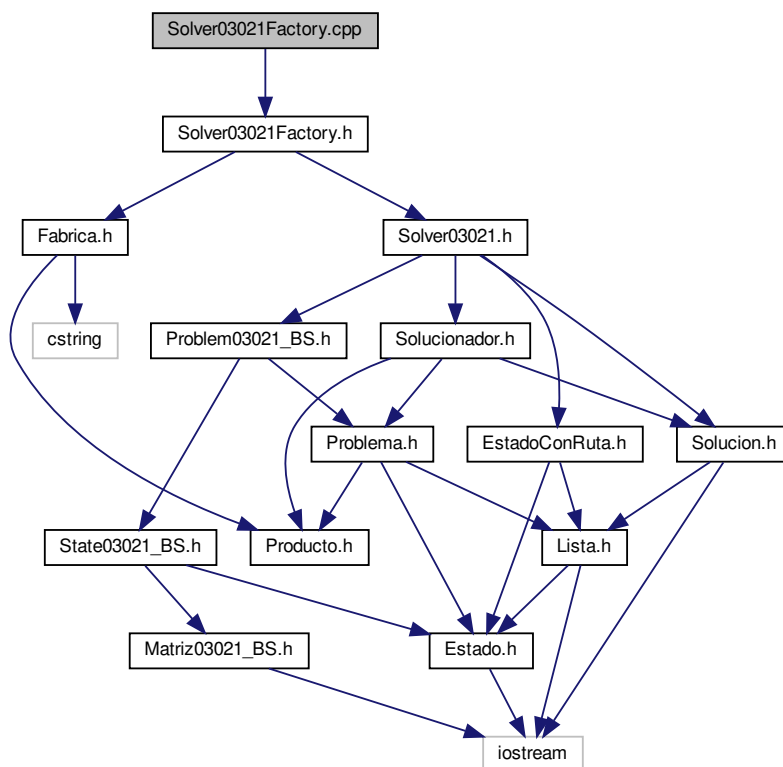
- class [Solver03021](#)

## 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:

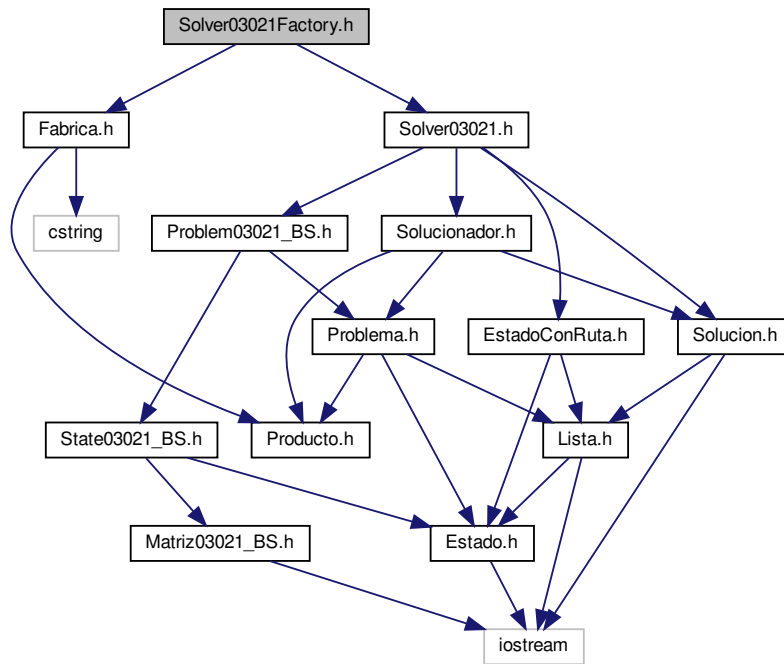
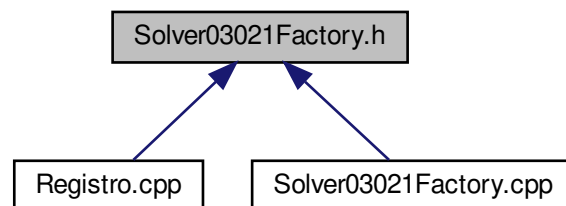


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [Solver03021Factory](#)

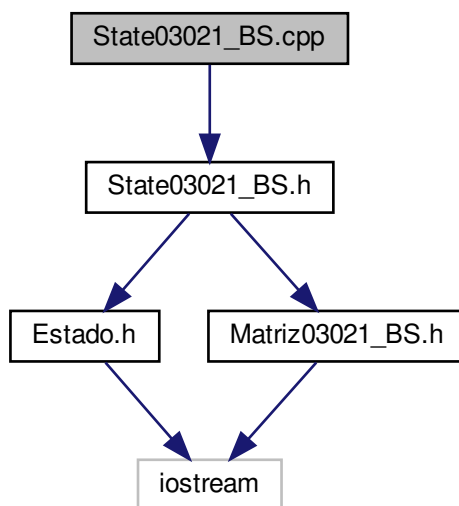


## 6.50. Referencia del Archivo State03021\_BS.cpp

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:

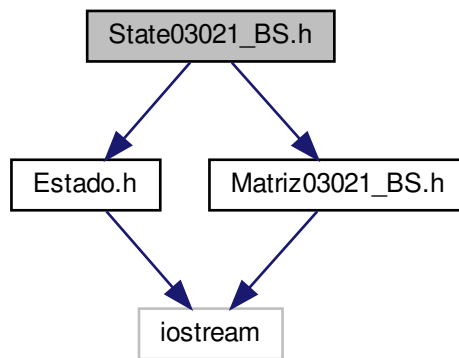
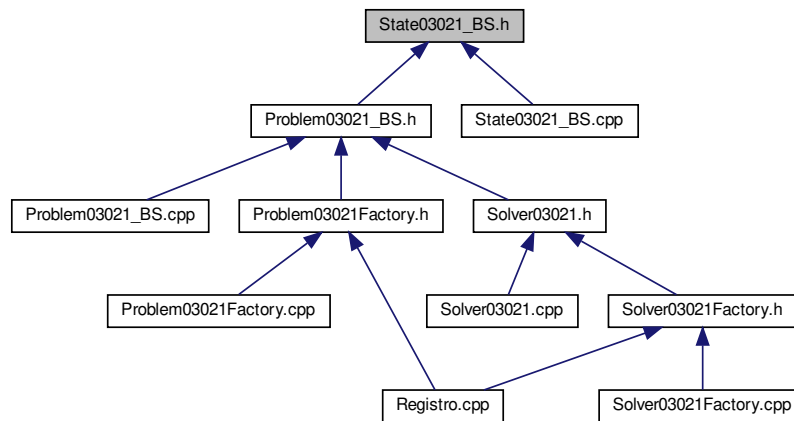


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [State03021\\_BS](#)

# Índice alfabético

- ~Estado
  - Estado, [10](#)
- ~EstadoConRuta
  - EstadoConRuta, [14](#)
- ~Fabrica
  - Fabrica, [21](#)
- ~GridLO
  - GridLO, [33](#)
- ~Lista
  - Lista, [44](#)
- ~MatrixZ2
  - MatrixZ2, [49](#)
- ~Matriz03021\_BS
  - Matriz03021\_BS, [55](#)
- ~Problem03021Factory
  - Problem03021Factory, [65](#)
- ~Producto
  - Producto, [73](#)
- ~Registro
  - Registro, [74](#)
- ~Solucion
  - Solucion, [76](#)
- ~Solver03021Factory
  - Solver03021Factory, [88](#)
- ~State03021\_BS
  - State03021\_BS, [91](#)
- begin
  - Lista, [44](#)
- borrar
  - Lista, [44](#)
- buscar
  - Lista, [44](#)
- cantidadPasos
  - SolucionadorSofia, [83](#)
- CAPACIDAD
  - Registro.h, [128](#)
- cargar
  - Estado, [10](#)
  - EstadoTH, [18](#)
  - GridLO, [33](#)
  - MatrixZ2, [49](#)
  - State03021\_BS, [91](#)
- clonar
  - Estado, [11](#)
  - EstadoConRuta, [14](#)
  - EstadoTH, [18](#)
  - GridLO, [35](#)
  - State03021\_BS, [92](#)

- COLUMNASBS
  - Matriz03021\_BS.h, [119](#)
- contarMaxColumna
  - Matriz03021\_BS, [55](#)
- contarMovimientosRestantes
  - Matriz03021\_BS, [56](#)
- end
  - Lista, [45](#)
- esSolucion
  - LightsOut, [41](#)
  - Problem03021\_BS, [62](#)
  - Problema, [67](#)
  - ProblemaTH, [71](#)
- Estado, [9](#)
  - ~Estado, [10](#)
  - cargar, [10](#)
  - clonar, [11](#)
  - imprimir, [11](#)
  - operator!=, [11](#)
  - operator<<, [11](#)
  - operator>>, [12](#)
  - operator==, [11](#)
- Estado.h, [95](#)
- EstadoConRuta, [12](#)
  - ~EstadoConRuta, [14](#)
  - clonar, [14](#)
  - EstadoConRuta, [14](#)
  - getEstado, [15](#)
  - getRuta, [15](#)
  - operator!=, [15](#)
  - operator==, [15](#)
- EstadoConRuta.cpp, [96](#)
- EstadoConRuta.h, [97](#)
- EstadoTH, [16](#)
  - cargar, [18](#)
  - clonar, [18](#)
  - EstadoTH, [18](#)
  - imprimir, [18](#)
  - operator!=, [19](#)
  - operator==, [19](#)
  - ProblemaTH, [20](#)
- EstadoTH.cpp, [98](#)
- EstadoTH.h, [98](#)
  - LINESIZE, [99](#)
  - QPLATES, [99](#)
  - QTOWERS, [99](#)
- Fabrica, [20](#)
  - ~Fabrica, [21](#)

- nombre, [22](#)
  - produce, [21](#)
  - producir, [22](#)
  - setNombre, [22](#)
- Fabrica.h, [100](#)
  - NOMBRE\_MAX\_SIZE, [100](#)
- FabricaLightsOut, [23](#)
  - producir, [24](#)
- FabricaLightsOut.cpp, [101](#)
- FabricaLightsOut.h, [101](#)
- FabricaProblemaTH, [25](#)
  - producir, [26](#)
- FabricaProblemaTH.cpp, [102](#)
- FabricaProblemaTH.h, [103](#)
- FabricaSolucionadorCZ, [27](#)
  - producir, [28](#)
- FabricaSolucionadorCZ.cpp, [105](#)
- FabricaSolucionadorCZ.h, [105](#)
- FabricaSolucionadorSofia, [29](#)
  - producir, [30](#)
- FabricaSolucionadorSofia.cpp, [107](#)
- FabricaSolucionadorSofia.h, [108](#)
- FILASBS
  - Matriz03021\_BS.h, [119](#)
- flip
  - GridLO, [35](#)
- getColumns
  - MatrixZ2, [50](#)
- getEstado
  - EstadoConRuta, [15](#)
- getEstadoInicial
  - LightsOut, [41](#)
  - Problem03021\_BS, [62](#)
  - Problema, [67](#)
  - ProblemaTH, [71](#)
- getFabrica
  - Registro, [74](#)
- getRows
  - MatrixZ2, [50](#)
- getRuta
  - EstadoConRuta, [15](#)
- getSiguientes
  - LightsOut, [41](#)
  - Problem03021\_BS, [62](#)
  - Problema, [67](#)
  - ProblemaTH, [71](#)
- GridLO, [31](#)
  - ~GridLO, [33](#)
  - cargar, [33](#)
  - clonar, [35](#)
  - flip, [35](#)
  - GridLO, [33](#)
  - imprimir, [35](#)
  - LightsOut, [36](#)
  - operator!=, [36](#)
  - operator==, [36](#)
- GridLO.cpp, [109](#)
- GridLO.h, [109](#)
  - GRIDSIZE\_LO, [111](#)
  - STREAMSIZE\_LO, [111](#)
- GridSIZE\_LO
  - GridLO.h, [111](#)
- heuristica
  - LightsOut, [42](#)
  - Matriz03021\_BS, [56](#)
  - Problem03021\_BS, [63](#)
  - Problema, [68](#)
  - ProblemaTH, [72](#)
- imprimir
  - Estado, [11](#)
  - EstadoTH, [18](#)
  - GridLO, [35](#)
  - Lista, [45](#)
  - MatrixZ2, [50](#)
  - State03021\_BS, [92](#)
- insertar
  - Lista, [45](#)
- isEmpty
  - Lista, [45](#)
- Iterador
  - Lista, [46](#)
  - Lista::Iterador, [37, 38](#)
- liberar
  - Separador, [75](#)
- LightsOut, [39](#)
  - esSolucion, [41](#)
  - getEstadoInicial, [41](#)
  - getSiguientes, [41](#)
  - GridLO, [36](#)
  - heuristica, [42](#)
  - LightsOut, [41](#)
- LightsOut.cpp, [111](#)
- LightsOut.h, [111](#)
- LINESIZE
  - EstadoTH.h, [99](#)
- Lista, [42](#)
  - ~Lista, [44](#)
  - begin, [44](#)
  - borrar, [44](#)
  - buscar, [44](#)
  - end, [45](#)
  - imprimir, [45](#)
  - insertar, [45](#)
  - isEmpty, [45](#)
  - Iterador, [46](#)
  - Lista, [44](#)
  - Lista::Iterador, [39](#)
  - operator<<, [46](#)
  - operator=, [45](#)
  - pop\_back, [45](#)
  - pop\_front, [45](#)
  - push\_back, [46](#)
  - push\_front, [46](#)
  - rbegin, [46](#)

Lista.cpp, 112  
 Lista.h, 113  
 Lista::Iterador, 37  
     Iterador, 37, 38  
     Lista, 39  
     operator!=, 38  
     operator\*, 38  
     operator++, 38  
     operator==, 38  
  
 main  
     main.cpp, 115  
 main.cpp, 114  
     main, 115  
 MatrixZ2, 47  
     ~MatrixZ2, 49  
     cargar, 49  
     getColumns, 50  
     getRows, 50  
     imprimir, 50  
     MatrixZ2, 48, 49  
     operator!=, 51  
     operator+, 51  
     operator=, 51, 52  
     operator==, 52  
     operator[], 53  
     randomize, 53  
 MatrixZ2.cpp, 115  
 MatrixZ2.h, 115  
 Matriz03021\_BS, 53  
     ~Matriz03021\_BS, 55  
     contarMaxColumna, 55  
     contarMovimientosRestantes, 56  
     heuristica, 56  
     Matriz03021\_BS, 55  
     mover, 56  
     movimientoValido, 57  
     operator!=, 57  
     operator<<, 59  
     operator>>, 59  
     operator=, 57  
     operator==, 58  
     randomize, 58  
     solucionado, 58  
 Matriz03021\_BS.cpp, 117  
 Matriz03021\_BS.h, 118  
     COLUMNASBS, 119  
     FILASBS, 119  
 mover  
     Matriz03021\_BS, 56  
 movimientoValido  
     Matriz03021\_BS, 57  
  
 nombre  
     Fabrica, 22  
 NOMBRE\_MAX\_SIZE  
     Fabrica.h, 100  
  
 operator!=  
     Estado, 11  
     EstadoConRuta, 15  
     EstadoTH, 19  
     GridLO, 36  
     Lista::Iterador, 38  
     MatrixZ2, 51  
     Matriz03021\_BS, 57  
     State03021\_BS, 92  
 operator<<  
     Estado, 11  
     Lista, 46  
     Matriz03021\_BS, 59  
     Solucion, 77  
 operator>>  
     Estado, 12  
     Matriz03021\_BS, 59  
 operator\*  
     Lista::Iterador, 38  
 operator+  
     MatrixZ2, 51  
 operator++  
     Lista::Iterador, 38  
 operator=  
     Lista, 45  
     MatrixZ2, 51, 52  
     Matriz03021\_BS, 57  
 operator==  
     Estado, 11  
     EstadoConRuta, 15  
     EstadoTH, 19  
     GridLO, 36  
     Lista::Iterador, 38  
     MatrixZ2, 52  
     Matriz03021\_BS, 58  
     State03021\_BS, 93  
 operator[]  
     MatrixZ2, 53  
 ordenarLista  
     Solver03021, 85  
  
 pop\_back  
     Lista, 45  
 pop\_front  
     Lista, 45  
 Problem03021\_BS, 59  
     esSolucion, 62  
     getEstadoInicial, 62  
     getSiguietes, 62  
     heuristica, 63  
     Problem03021\_BS, 61  
     State03021\_BS, 93  
 Problem03021\_BS.cpp, 119  
 Problem03021\_BS.h, 120  
 Problem03021Factory, 63  
     ~Problem03021Factory, 65  
     producir, 66  
 Problem03021Factory.cpp, 121  
 Problem03021Factory.h, 123  
 Problema, 66

- esSolucion, [67](#)
  - getEstadoInicial, [67](#)
  - getSiguientes, [67](#)
  - heuristica, [68](#)
- Problema.h, [124](#)
- ProblemaTH, [68](#)
  - esSolucion, [71](#)
  - EstadoTH, [20](#)
  - getEstadoInicial, [71](#)
  - getSiguientes, [71](#)
  - heuristica, [72](#)
- ProblemaTH.cpp, [124](#)
- ProblemaTH.h, [125](#)
- produce
  - Fabrica, [21](#)
- producir
  - Fabrica, [22](#)
  - FabricaLightsOut, [24](#)
  - FabricaProblemaTH, [26](#)
  - FabricaSolucionadorCZ, [28](#)
  - FabricaSolucionadorSofia, [30](#)
  - Problem03021Factory, [66](#)
  - Solver03021Factory, [89](#)
- Producto, [72](#)
  - ~Producto, [73](#)
- Producto.h, [127](#)
- push\_back
  - Lista, [46](#)
- push\_front
  - Lista, [46](#)
- QPLATES
  - EstadoTH.h, [99](#)
- QTOWERS
  - EstadoTH.h, [99](#)
- randomize
  - MatrixZ2, [53](#)
  - Matriz03021\_BS, [58](#)
- rbegin
  - Lista, [46](#)
- README.md, [127](#)
- Registro, [73](#)
  - ~Registro, [74](#)
  - getFabrica, [74](#)
  - Registro, [74](#)
- Registro.cpp, [127](#)
- Registro.h, [128](#)
  - CAPACIDAD, [128](#)
- Separador, [74](#)
  - liberar, [75](#)
  - separar, [75](#)
- Separador.cpp, [129](#)
- Separador.h, [129](#)
- separar
  - Separador, [75](#)
- setNombre
  - Fabrica, [22](#)
- Solucion, [76](#)
  - ~Solucion, [76](#)
  - operator<<, [77](#)
  - Solucion, [76](#)
- Solucion.cpp, [130](#)
- Solucion.h, [131](#)
- solucionado
  - Matriz03021\_BS, [58](#)
- Solucionador, [77](#)
  - solucione, [78](#)
- Solucionador.h, [132](#)
- SolucionadorCarolina, [79](#)
  - solucione, [80](#)
- SolucionadorCarolina.cpp, [133](#)
- SolucionadorCarolina.h, [134](#)
- SolucionadorSofia, [81](#)
  - cantidadPasos, [83](#)
  - solucione, [82](#)
- SolucionadorSofia.cpp, [136](#)
- SolucionadorSofia.h, [136](#)
- solucione
  - Solucionador, [78](#)
  - SolucionadorCarolina, [80](#)
  - SolucionadorSofia, [82](#)
  - Solver03021, [86](#)
- Solver03021, [83](#)
  - ordenarLista, [85](#)
  - solucione, [86](#)
- Solver03021.cpp, [138](#)
- Solver03021.h, [139](#)
- Solver03021Factory, [86](#)
  - ~Solver03021Factory, [88](#)
  - producir, [89](#)
- Solver03021Factory.cpp, [140](#)
- Solver03021Factory.h, [142](#)
- State03021\_BS, [89](#)
  - ~State03021\_BS, [91](#)
  - cargar, [91](#)
  - clonar, [92](#)
  - imprimir, [92](#)
  - operator!=, [92](#)
  - operator==, [93](#)
  - Problem03021\_BS, [93](#)
  - State03021\_BS, [91](#)
- State03021\_BS.cpp, [143](#)
- State03021\_BS.h, [144](#)
- STREAMSIZE\_LO
  - GridLO.h, [111](#)