Unidad 2 Actividad 1

1.0

Generated by Doxygen 1.9.8

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

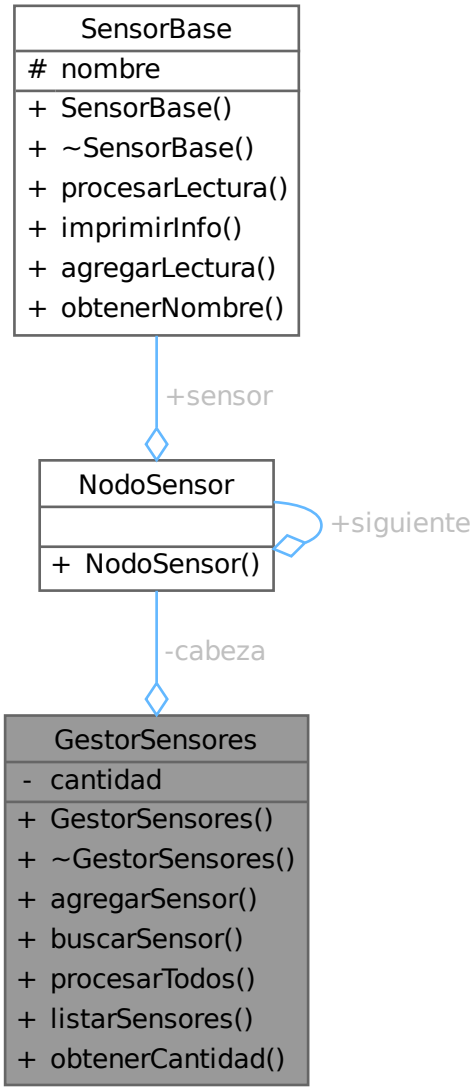
Class Documentation

4.1 GestorSensores Class Reference

Administrador central de todos los sensores del sistema.

`#include <GestorSensores.h>`

Collaboration diagram for GestorSensores:



Public Member Functions

- GestorSensores ()

  Constructor del gestor.
- ~GestorSensores ()

  Destructor - Libera TODOS los sensores y sus listas internas.
- void agregarSensor (SensorBase *sensor)

  Agrega un nuevo sensor a la lista de gestión.
- SensorBase * buscarSensor (const char *id)

  Busca un sensor por su ID.
- void procesarTodos ()

Procesa todos los sensores registrados.

- void listarSensores () const

  Imprime información de todos los sensores.

- int obtenerCantidad () const

  Obtiene el número de sensores registrados.

Private Attributes

- NodoSensor * cabeza

  Primer nodo de la lista de sensores.

- int cantidad

  Contador de sensores registrados.

4.1.1 Detailed Description

Administrador central de todos los sensores del sistema.

TAREAS:

1. Mantener una lista de todos los sensores activos
2. Permitir agregar nuevos sensores
3. Buscar sensores por ID
4. Procesar todos los sensores con iteración
5. Liberar memoria al destruirse

PATRÓN DE DISEÑO: Este es un ejemplo del patrón "Gestor" o "Manager":

- Centraliza la gestión de recursos (sensores)
- Proporciona una interfaz limpia para operaciones comunes

4.1.2 Constructor & Destructor Documentation

4.1.2.1 GestorSensores()

```
GestorSensores::GestorSensores ( )  [inline]
```

Constructor del gestor.

Inicializa la lista vacía y el contador en 0.

4.1.2.2 ~GestorSensores()

```
GestorSensores::~GestorSensores ( )  [inline]
```

Destructor - Libera TODOS los sensores y sus listas internas.

PROCESO CRÍTICO DE LIBERACIÓN EN CASCADA:

Para cada sensor en la lista:

1. Obtenemos el puntero SensorBase*
2. delete sensor llama a: a) ~SensorTemperatura() o ~SensorPresion()
   - Aquí se imprime el mensaje de liberación
   - Se destruye 'historial' (ListaSensor)
   - ~ListaSensor() libera TODOS los nodos internos b) ~SensorBase()
3. Liberamos el NodoSensor

RAZÓN del orden:

- Primero delete sensor (libera el sensor y su lista interna)
- Después delete nodoActual (libera el nodo de gestión)

Si no hiciéramos esto = FUGA MASIVA DE MEMORIA:

- Los sensores quedarían huérfanos
- Sus listas internas nunca se liberarían
- Todos los nodos de lecturas quedarían ocupando RAM

4.1.3 Member Function Documentation

4.1.3.1 agregarSensor()

```
void GestorSensores::agregarSensor (
            SensorBase * sensor )  [inline]
```

Agrega un nuevo sensor a la lista de gestión.

Parameters

sensor	Puntero al sensor (puede ser de cualquier tipo derivado)

CONCEPTO - Polimorfismo en Acción: Podemos recibir: agregarSensor(new SensorTemperatura("T-001")); agregarSensor(new SensorPresion("P-001"));

Ambos se almacenan como SensorBase*, pero mantienen su tipo real. Cuando llamamos sensor->procesar↩ Lectura(), se ejecuta la versión correcta según el tipo real del objeto.

PROCESO:

1. Crear un NodoSensor con el puntero
2. Insertarlo al final de la lista
3. Incrementar contador

4.1.3.2 buscarSensor()

```
SensorBase * GestorSensores::buscarSensor (
            const char * id )  [inline]
```

Busca un sensor por su ID.

Parameters

id	Identificador del sensor a buscar

Returns

Puntero al sensor encontrado, o nullptr si no existe

RAZÓN de retorno SensorBase∗:

- El código cliente puede trabajar con el sensor sin saber su tipo
- Puede llamar a métodos polimórficos directamente

4.1.3.3 listarSensores()

```
void GestorSensores::listarSensores ( ) const  [inline]
```

Imprime información de todos los sensores.

4.1.3.4 obtenerCantidad()

```
int GestorSensores::obtenerCantidad ( ) const  [inline]
```

Obtiene el número de sensores registrados.

Returns

Cantidad de sensores

4.1.3.5 procesarTodos()

```
void GestorSensores::procesarTodos ( )  [inline]
```

Procesa todos los sensores registrados.

DEMOSTRACIÓN DEL POLIMORFISMO:

Para cada sensor en la lista: sensor->procesarLectura();

Aunque todos son SensorBase*, cada uno ejecuta SU PROPIA versión:

- SensorTemperatura: Elimina mínimo + promedio
- SensorPresion: Solo promedio

El gestor no necesita saber qué tipo es cada sensor. El polimorfismo se encarga automáticamente.

4.1.4 Member Data Documentation

4.1.4.1 cabeza

```
NodoSensor* GestorSensores::cabeza  [private]
```

Primer nodo de la lista de sensores.

4.1.4.2 cantidad

```
int GestorSensores::cantidad  [private]
```

Contador de sensores registrados.

The documentation for this class was generated from the following file:

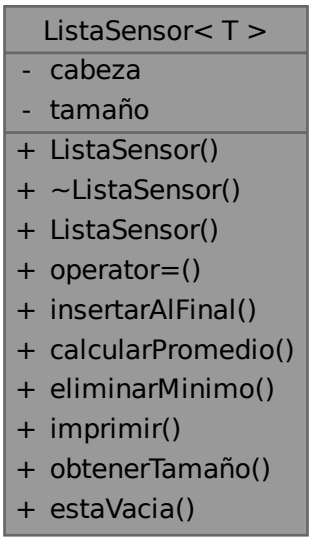
- /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/GestorSensores.h

4.2 ListaSensor< T > Class Template Reference

Lista Enlazada Simple Genérica.

`#include <ListaSensor.h>`

Collaboration diagram for ListaSensor< T >:



Public Member Functions

- ListaSensor ()
  Constructor por defecto.
- ~ListaSensor ()
  Destructor.
- ListaSensor (const ListaSensor &otra)
  Constructor de copia.
- ListaSensor & operator= (const ListaSensor &otra)
  Operador de asignación.
- void insertarAlFinal (T valor)
  Inserta un elemento al final de la lista.
- T calcularPromedio () const
  Calcula el promedio de todos los valores.
- T eliminarMinimo ()
  Encuentra y elimina el valor más bajo.
- void imprimir () const
  Imprime todos los elementos de la lista.
- int obtenerTamaño () const
  Obtiene el tamaño de la lista.
- bool estaVacia () const
  Verifica si la lista está vacía.

Private Attributes

- Nodo< T > * cabeza

  Puntero al primer nodo de la lista.
- int tamaño

  Contador de elementos en la lista.

4.2.1 Detailed Description

template<typename T>
class ListaSensor< T >

Lista Enlazada Simple Genérica.

Template Parameters

T	Tipo de dato que almacenará la lista

Ejemplo: En los vagones de tren:

- 'cabeza' es el primer vagón
- Cada vagón sabe cuál es el siguiente
- El último vagón apunta a nullptr (fin de la lista)

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ListaSensor() [1/2]

```
template<typename T >
ListaSensor< T >::ListaSensor ( )  [inline]
```

Constructor por defecto.

CONCEPTO: Inicializamos la lista vacía. cabeza = nullptr significa "no hay ningún nodo todavía"

4.2.2.2 ~ListaSensor()

```
template<typename T >
ListaSensor< T >::~ListaSensor ( )  [inline]
```

Destructor.

RAZÓN: Cuando la lista se destruye, debemos liberar TODOS los nodos que creamos con 'new'. Si no lo hacemos, la memoria queda ocupada y hay fuga de memoria.

PROCESO:

1. Empezamos desde la cabeza
2. Guardamos el siguiente antes de borrar el actual
3. Liberamos el nodo actual
4. Avanzamos al siguiente
5. Repetimos hasta llegar a nullptr

4.2.2.3 ListaSensor() [2/2]

```
template<typename T >
ListaSensor< T >::ListaSensor (
            const ListaSensor< T > & otra )  [inline]
```

Constructor de copia.

Parameters

otra	Lista a copiar

RAZÓN: Si hacemos ListaSensor<int> copia = original; necesitamos crear NUEVOS nodos, no copiar los punteros. Si copiáramos solo punteros, ambas listas compartirían los mismos nodos, lo que seria un PROBLEMA al destruir porque la eliminación se hara dos veces a el mismo elemento lo cual causará error

4.2.3 Member Function Documentation

4.2.3.1 calcularPromedio()

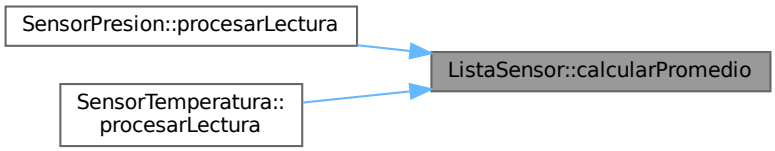
```
template<typename T >
T ListaSensor< T >::calcularPromedio ( ) const  [inline]
```

Calcula el promedio de todos los valores.

Returns

Promedio de tipo T

RAZÓN: Necesitamos procesar los datos, que pueden ser int o float. Esta es una operación común en sensores: obtener el valor promedio de las lecturas. Here is the caller graph for this function:



4.2.3.2 eliminarMinimo()

```
template<typename T >
T ListaSensor< T >::eliminarMinimo ( )  [inline]
```

Encuentra y elimina el valor más bajo.

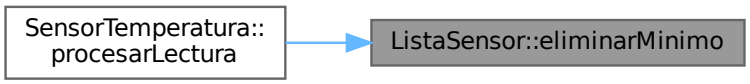
Returns

El valor eliminado

PROCESO:

1. Recorrer la lista para encontrar el mínimo
2. Guardar referencia al nodo anterior
3. Reenlazar: anterior->siguiente = minimo->siguiente
4. Liberar el nodo del mínimo

Here is the caller graph for this function:



4.2.3.3 estaVacia()

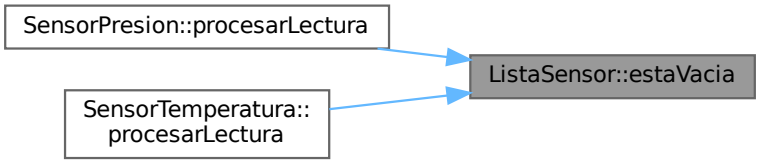
```
template<typename T >
bool ListaSensor< T >::estaVacia ( ) const  [inline]
```

Verifica si la lista está vacía.

Returns

true si está vacía, false en caso contrario

Here is the caller graph for this function:

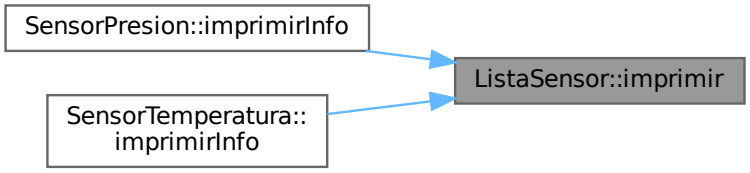


4.2.3.4 imprimir()

```
template<typename T >
void ListaSensor< T >::imprimir ( ) const  [inline]
```

Imprime todos los elementos de la lista.

RAZÓN: Para depuración y verificación visual de los datos Here is the caller graph for this function:



4.2.3.5 insertarAlFinal()

```
template<typename T >
void ListaSensor< T >::insertarAlFinal (
            T valor )  [inline]
```

Inserta un elemento al final de la lista.

Parameters

valor	Dato a insertar

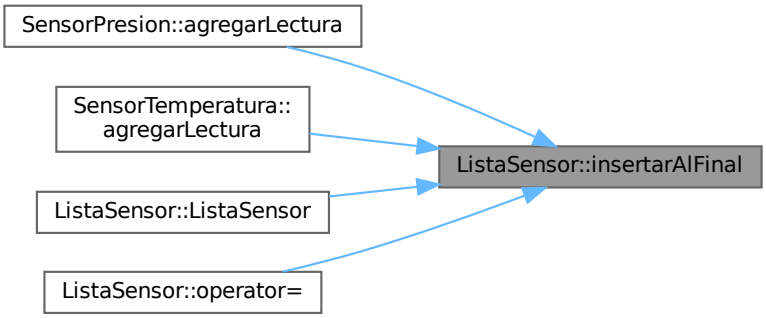
PROCESO: Caso 1: Lista vacía (cabeza == nullptr)

- El nuevo nodo se convierte en la cabeza

Caso 2: Lista con elementos

- Recorremos hasta el último nodo
- Enganchamos el nuevo nodo al final

Here is the caller graph for this function:



4.2.3.6 obtenerTamaño()

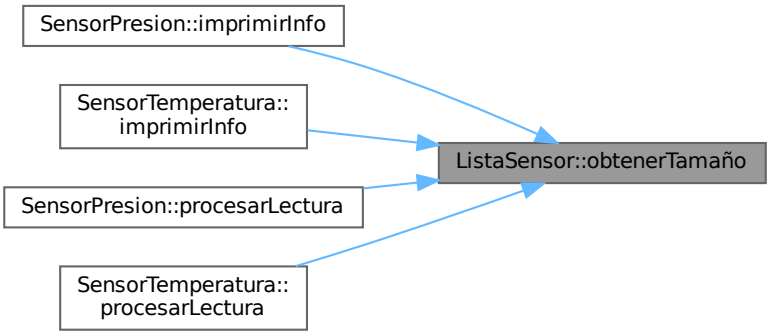
```
template<typename T >
int ListaSensor< T >::obtenerTamaño ( ) const  [inline]
```

Obtiene el tamaño de la lista.

Returns

Número de elementos

Here is the caller graph for this function:



4.2.3.7 operator=()

```
template<typename T >
ListaSensor & ListaSensor< T >::operator= (
            const ListaSensor< T > & otra )  [inline]
```

Operador de asignación.

Parameters

otra	Lista a asignar

Returns

Referencia a esta lista

RAZÓN: Para manejar lista1 = lista2; Debemos:

- Limpiar lista1
- Copiar contenido de lista2

4.2.4 Member Data Documentation

4.2.4.1 cabeza

```
template<typename T >
Nodo<T>* ListaSensor< T >::cabeza  [private]
```

Puntero al primer nodo de la lista.

4.2.4.2 tamaño

```
template<typename T >
int ListaSensor< T >::tamaño  [private]
```

Contador de elementos en la lista.

The documentation for this class was generated from the following file:

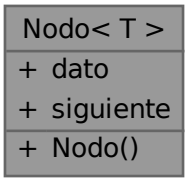
- /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/ListaSensor.h

4.3 Nodo< T > Struct Template Reference

Nodo genérico para la lista enlazada.

`#include <ListaSensor.h>`

Collaboration diagram for Nodo< T >:



Public Member Functions

- Nodo (T valor)

  Constructor del nodo.

Public Attributes

- T dato

  Valor almacenado en el nodo.
- Nodo< T > * siguiente

  Puntero al siguiente nodo.

4.3.1 Detailed Description

template<typename T>
struct Nodo< T >

Nodo genérico para la lista enlazada.

Template Parameters

T	Tipo de dato que almacenará el nodo: int, float, etc..

CONCEPTO: Un nodo es como una caja que contiene:

1. Un valor (dato)
2. Una flecha que apunta a la siguiente caja, la siguiente.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Nodo()

```
template<typename T >
Nodo< T >::Nodo (
            T valor )  [inline]
```

Constructor del nodo.

Parameters

valor	El dato a almacenar

¿Por qué? Inicializamos el "siguiente" como un nullptr porque aun no se sabe cual sera el nodo siguiente

4.3.3 Member Data Documentation

4.3.3.1 dato

```
template<typename T >
T Nodo< T >::dato
```

Valor almacenado en el nodo.

4.3.3.2 siguiente

```
template<typename T >
Nodo<T>* Nodo< T >::siguiente
```

Puntero al siguiente nodo.

The documentation for this struct was generated from the following file:

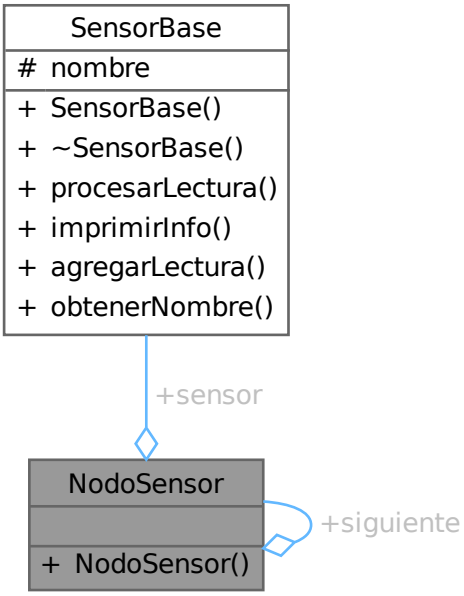
- /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/ListaSensor.h

4.4 NodoSensor Struct Reference

Nodo para la lista de gestión polimórfica.

```
#include <GestorSensores.h>
```

Collaboration diagram for NodoSensor:



Public Member Functions

- NodoSensor (SensorBase ∗s)

  Constructor del nodo.

Public Attributes

- SensorBase ∗ sensor

  Puntero polimórfico al sensor.
- NodoSensor ∗ siguiente

  Siguiente nodo en la lista.

4.4.1 Detailed Description

Nodo para la lista de gestión polimórfica.

DIFERENCIA CLAVE con Nodo<T>:

- Este nodo NO es genérico
- Almacena específicamente punteros a SensorBase
- Permite el polimorfismo: guardar SensorTemperatura∗ y SensorPresion∗ juntos

CONCEPTO - Puntero a Clase Base: SensorBase∗ puede apuntar a:

- SensorTemperatura
- SensorPresion
- Cualquier futuro sensor que herede de SensorBase

4.4.2 Constructor & Destructor Documentation

4.4.2.1 NodoSensor()

```
NodoSensor::NodoSensor (
            SensorBase * s )  [inline]
```

Constructor del nodo.

Parameters

s	Puntero al sensor a almacenar

4.4.3 Member Data Documentation

4.4.3.1 sensor

```
SensorBase* NodoSensor::sensor
```

Puntero polimórfico al sensor.

4.4.3.2 siguiente

```
NodoSensor* NodoSensor::siguiente
```

Siguiente nodo en la lista.

The documentation for this struct was generated from the following file:

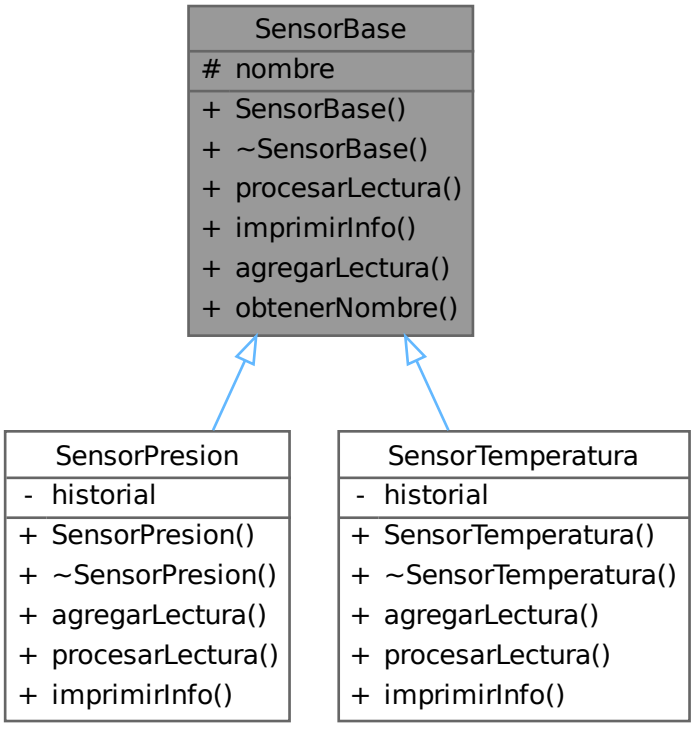
- /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/GestorSensores.h

4.5 SensorBase Class Reference

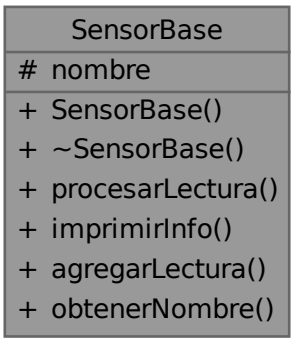
Clase abstracta que representa un sensor genérico.

```
#include <SensorBase.h>
```

Inheritance diagram for SensorBase:



Collaboration diagram for SensorBase:



Public Member Functions

- SensorBase (const char *id)
  Constructor que inicializa el nombre del sensor.
- virtual ~SensorBase ()
  Destructor virtual.
- virtual void procesarLectura ()=0
  Procesa las lecturas del sensor según su tipo.
- virtual void imprimirInfo () const =0
  Imprime información del sensor.
- virtual void agregarLectura (const char *valor)=0
  Agrega una lectura al sensor.
- const char * obtenerNombre () const
  Obtiene el nombre del sensor.

Protected Attributes

- char nombre [50]
  Identificador único del sensor.

4.5.1 Detailed Description

Clase abstracta que representa un sensor genérico.

CONCEPTO - Clase Abstracta: Es como una plantilla que establece lo que cualquier sensor DEBE poder:

1. Procesarse (procesarLectura)
2. Imprimir su información (imprimirInfo)
3. Agregar lecturas (agregarLectura)

RAZÓN del "= 0": Los métodos con "= 0" son "virtuales puros". Esto significa:

- NO tienen implementación aquí
- Las clases hijas DEBEN implementarlos
- NO se puede crear un objeto SensorBase directamente

4.5.2 Constructor & Destructor Documentation

4.5.2.1 SensorBase()

```
SensorBase::SensorBase (
            const char * id )  [inline]
```

Constructor que inicializa el nombre del sensor.

Parameters

id	Identificador del sensor (ej: "T-001", "P-105")

RAZÓN de usar strcpy:

- Copia cadenas de caracteres al estilo C
- strcpy(destino, origen)

4.5.2.2 ~SensorBase()

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

Destructor virtual.

REGLA DE ORO: Si una clase tiene métodos virtuales, su destructor DEBE ser virtual.

4.5.3 Member Function Documentation

4.5.3.1 agregarLectura()

```
virtual void SensorBase::agregarLectura (
            const char * valor )  [pure virtual]
```

Agrega una lectura al sensor.

Parameters

valor	Valor de la lectura

RAZÓN de recibir const char∗:

- Podemos recibir "45.3" o "80"
- La clase hija convierte al tipo correcto, float o int

Implemented in SensorPresion, and SensorTemperatura.

Here is the caller graph for this function:

4.5.3.2 imprimirInfo()

```
virtual void SensorBase::imprimirInfo ( ) const  [pure virtual]
```

Imprime información del sensor.

MÉTODO VIRTUAL PURO: Cada sensor mostrará su información específica

Implemented in SensorPresion, and SensorTemperatura.

Here is the caller graph for this function:

4.5.3.3 obtenerNombre()

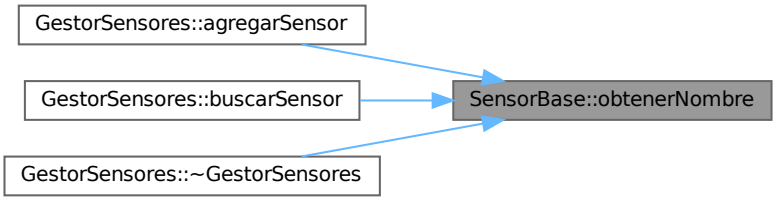
```
const char * SensorBase::obtenerNombre ( ) const  [inline]
```

Obtiene el nombre del sensor.

Returns

Puntero al nombre

MÉTODO NO VIRTUAL: Todos los sensores obtienen el nombre igual, no necesita polimorfismo. Here is the caller graph for this function:



4.5.3.4 procesarLectura()

```
virtual void SensorBase::procesarLectura ( )  [pure virtual]
```

Procesa las lecturas del sensor según su tipo.

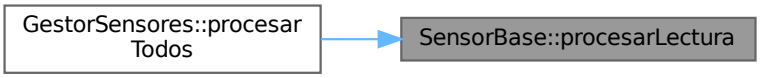
MÉTODO VIRTUAL PURO (= 0):

- Cada tipo de sensor lo implementa diferente:
  - Temperatura: Elimina el mínimo y calcula promedio
  - Presión: Calcula promedio directo

RAZÓN del 'virtual': Permite que en tiempo de ejecución se llame a la versión correcta: SensorBase* s = new SensorTemperatura(); s->procesarLectura(); // Llama a SensorTemperatura::procesarLectura()

Implemented in SensorPresion, and SensorTemperatura.

Here is the caller graph for this function:



4.5.4 Member Data Documentation

4.5.4.1 nombre

```
char SensorBase::nombre[50]  [protected]
```

Identificador único del sensor.

RAZÓN del protected:

- Las clases hijas pueden accederlo
- El mundo exterior NO puede modificarlo directamente

The documentation for this class was generated from the following file:

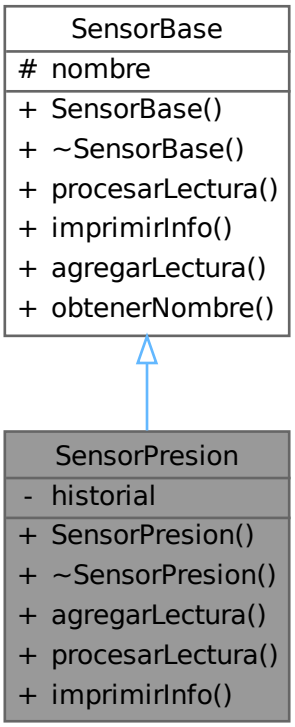
- /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/SensorBase.h

4.6 SensorPresion Class Reference

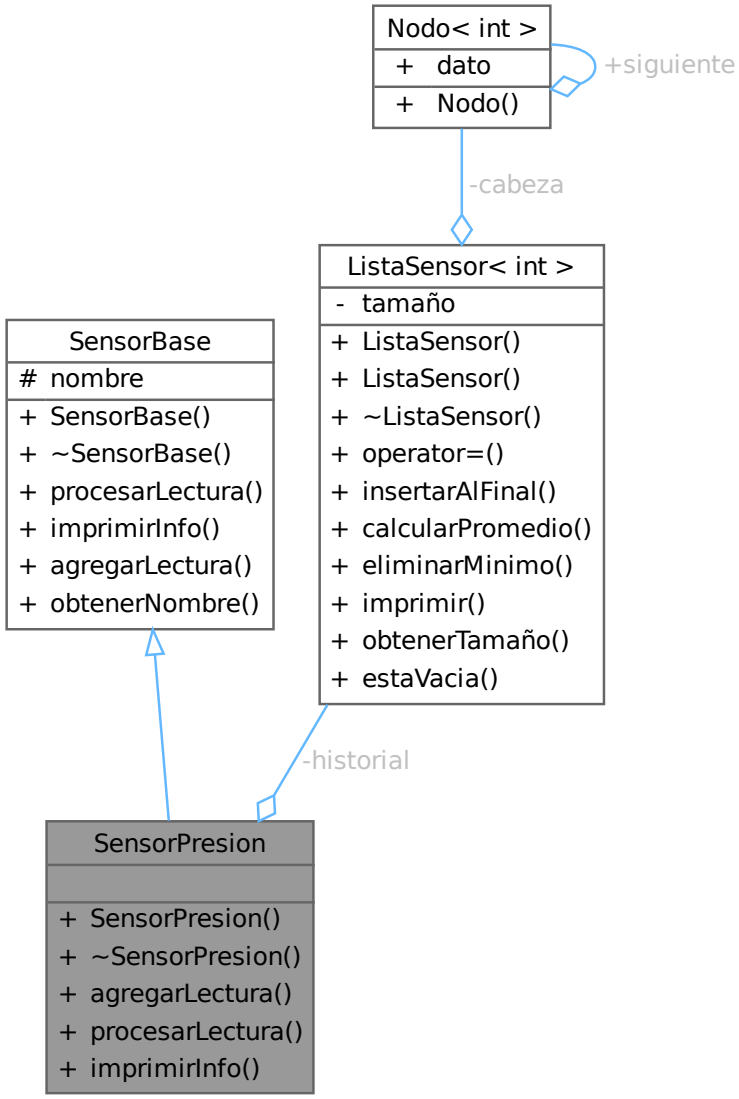
Sensor que maneja presiones en formato enteros.

`#include <SensorPresion.h>`

Inheritance diagram for SensorPresion:



Collaboration diagram for SensorPresion:



Public Member Functions

- SensorPresion (const char *id)

  Constructor del sensor de presión.
- ~SensorPresion ()

  Destructor que libera recursos.
- void agregarLectura (const char *valor) override

  Agrega una lectura de presión.
- void procesarLectura () override

  Procesa las lecturas calculando el promedio.
- void imprimirInfo () const override

  Imprime información detallada del sensor.

Public Member Functions inherited from SensorBase

- SensorBase (const char *id)

  Constructor que inicializa el nombre del sensor.
- virtual ~SensorBase ()

  Destructor virtual.
- const char * obtenerNombre () const

  Obtiene el nombre del sensor.

Private Attributes

- ListaSensor< int > historial

  Lista enlazada que almacena lecturas de presión.

Additional Inherited Members

Protected Attributes inherited from SensorBase

- char nombre [50]

  Identificador único del sensor.

4.6.1 Detailed Description

Sensor que maneja presiones en formato enteros.

DIFERENCIAS CON SensorTemperatura:

- Usa ListaSensor<int> en lugar de ListaSensor<float>
- El procesamiento es diferente, cuando promedia no elimina al chiquita

VENTAJA DEL POLIMORFISMO: Ambos sensores tienen la misma interfaz pero comportamientos distintos. El código cliente no necesita saber cuál es cuál

4.6.2 Constructor & Destructor Documentation

4.6.2.1 SensorPresion()

```
SensorPresion::SensorPresion (
            const char * id )  [inline]
```

Constructor del sensor de presión.

Parameters

id	Identificador único del sensor

4.6.2.2 ~SensorPresion()

```
SensorPresion::~SensorPresion ( )  [inline]
```

Destructor que libera recursos.

NOTA: Aunque no tiene código explícito, el destructor de 'historial' se llama automáticamente, liberando todos los nodos de la lista.

4.6.3 Member Function Documentation

4.6.3.1 agregarLectura()

```
void SensorPresion::agregarLectura (
            const char * valor )  [inline], [override], [virtual]
```

Agrega una lectura de presión.

Parameters

valor	String con el valor entero

RAZÓN de usar atoi():

- atoi = ASCII to Integer
- Convierte "1013" (char*) → 1013 (int)

Implements SensorBase.

4.6.3.2 imprimirInfo()

```
void SensorPresion::imprimirInfo ( ) const  [inline], [override], [virtual]
```

Imprime información detallada del sensor.

Implements SensorBase.

4.6.3.3 procesarLectura()

```
void SensorPresion::procesarLectura ( )  [inline], [override], [virtual]
```

Procesa las lecturas calculando el promedio.

LÓGICA ESPECÍFICA DE PRESIÓN:

- NO elimina valores porque todas las lecturas son validas
- Calcula y muestra el promedio directamente

CONTRASTE con SensorTemperatura:

- Temp: Elimina mínimo + promedio
- Presión: Solo promedio

Esto demuestra que cada sensor tiene su propia lógica de procesamiento, pero se acceden de forma uniforme.

Implements SensorBase.

4.6.4 Member Data Documentation

4.6.4.1 historial

```
ListaSensor<int> SensorPresion::historial  [private]
```

Lista enlazada que almacena lecturas de presión.

RAZÓN de usar int:

- Las presiones suelen medirse en valores enteros

The documentation for this class was generated from the following file:

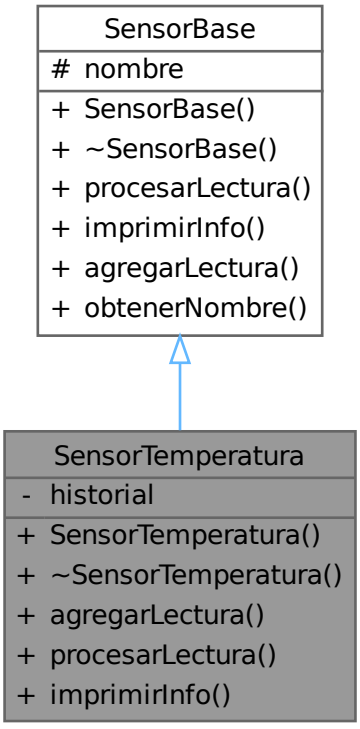
- /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/SensorPresion.h

4.7 SensorTemperatura Class Reference

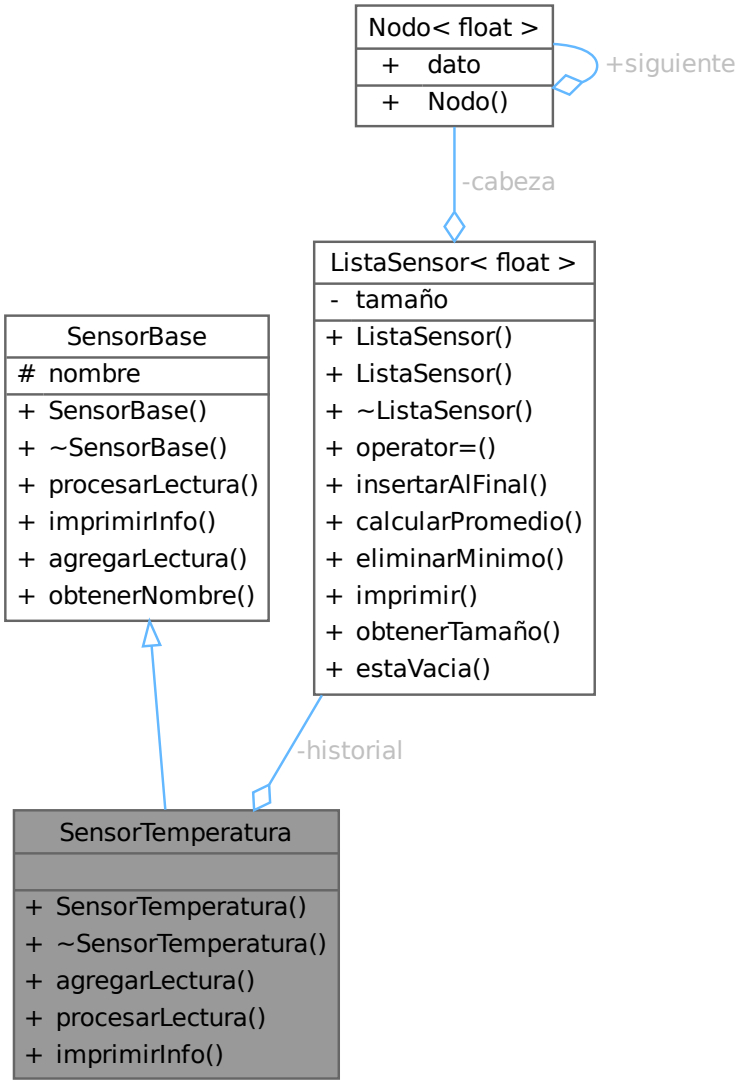
Sensor que maneja temperaturas en formato float.

```
#include <SensorTemperatura.h>
```

Inheritance diagram for SensorTemperatura:



Collaboration diagram for SensorTemperatura:



Public Member Functions

- SensorTemperatura (const char *id)

  Constructor que inicializa el sensor.
- ~SensorTemperatura ()

  Destructor que libera la lista interna.
- void agregarLectura (const char *valor) override

  Agrega una lectura de temperatura a la lista.
- void procesarLectura () override

  Procesa las lecturas: elimina mínimo y calcula promedio.
- void imprimirInfo () const override

  Imprime información del sensor y sus lecturas.

Public Member Functions inherited from SensorBase

- SensorBase (const char *id)

  Constructor que inicializa el nombre del sensor.
- virtual ~SensorBase ()

  Destructor virtual.
- const char * obtenerNombre () const

  Obtiene el nombre del sensor.

Private Attributes

- ListaSensor< float > historial

  Lista enlazada que almacena las lecturas de temperatura.

Additional Inherited Members

Protected Attributes inherited from SensorBase

- char nombre [50]

  Identificador único del sensor.

4.7.1 Detailed Description

Sensor que maneja temperaturas en formato float.

Hereda:

- El atributo 'nombre'
- La interfaz

Debe implementar:

- procesarLectura()
- imprimirInfo()
- agregarLectura()

4.7.2 Constructor & Destructor Documentation

4.7.2.1 SensorTemperatura()

```
SensorTemperatura::SensorTemperatura (
            const char * id )  [inline]
```

Constructor que inicializa el sensor.

Parameters

id	Identificador del sensor

SINTAXIS 'SensorBase(id)': Esto se llama "lista de inicialización de constructor"

- Llama al constructor de la clase padre -> SensorBase
- Le pasa el parámetro 'id'

PROCESO:

1. Se construye SensorBase con el id
2. Se construye el historial
3. Se ejecuta el cuerpo del constructor

4.7.2.2 ~SensorTemperatura()

```
SensorTemperatura::~SensorTemperatura ( )  [inline]
```

Destructor que libera la lista interna.

PROCESO DE DESTRUCCIÓN:

1. Se ejecuta este destructor
2. Se destruye "historial" -> llama a ~ListaSensor<float>()
3. Se llama a ~SensorBase()

RAZÓN de imprimir logs:

- Para verificar que la memoria se libera correctamente
- Para que yo entienda mejor mi codigo :3

4.7.3 Member Function Documentation

4.7.3.1 agregarLectura()

```
void SensorTemperatura::agregarLectura (
            const char * valor )  [inline], [override], [virtual]
```

Agrega una lectura de temperatura a la lista.

Parameters

valor	String con el valor

RAZÓN de usar atof():

- atof = ASCII to Float
- Convierte por ejemplo "23.5" (char*) -> 23.5 (float)

Implements SensorBase.

4.7.3.2 imprimirInfo()

```
void SensorTemperatura::imprimirInfo ( ) const  [inline], [override], [virtual]
```

Imprime información del sensor y sus lecturas.

MÉTODO const:

- No modifica el objeto
- Puede ser llamado en objetos const

Implements SensorBase.

4.7.3.3 procesarLectura()

```
void SensorTemperatura::procesarLectura ( )  [inline], [override], [virtual]
```

Procesa las lecturas: elimina mínimo y calcula promedio.

LÓGICA ESPECÍFICA DE TEMPERATURA:

1. Si hay lecturas, elimina la más baja porque podria ser un error o ruido
2. Calcula el promedio de las restantes
3. Muestra el resultado

RAZÓN de esta lógica: En sistemas reales, las lecturas anómalas bajas pueden indicar fallos del sensor, por eso las filtramos.

Implements SensorBase.

4.7.4 Member Data Documentation

4.7.4.1 historial

```
ListaSensor<float> SensorTemperatura::historial  [private]
```

Lista enlazada que almacena las lecturas de temperatura.

RAZÓN de usar ListaSensor<float>:

- Las temperaturas son decimales
- Necesitamos precisión decimal, por eso float

RAZÓN de ser private:

- Solo esta clase debe manipular su lista interna
- Encapsulamiento

The documentation for this class was generated from the following file:

- /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/SensorTemperatura.h

Chapter 5

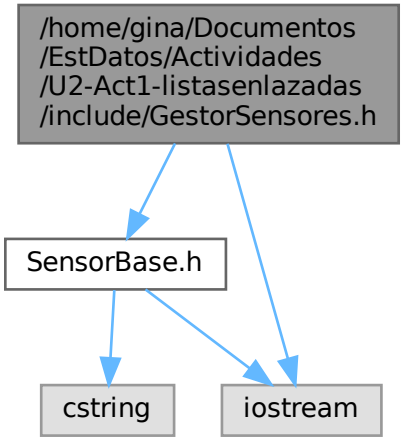
File Documentation

5.1 /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/GestorSensores.h File Reference

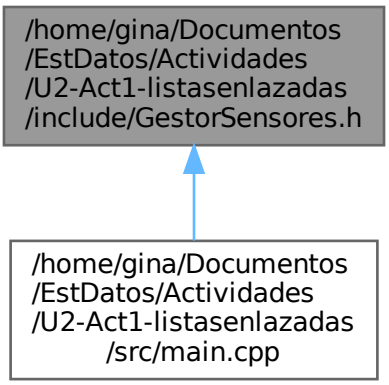
Sistema de gestión polimórfica de sensores.

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

Include dependency graph for GestorSensores.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct NodoSensor

  Nodo para la lista de gestión polimórfica.
- class GestorSensores

  Administrador central de todos los sensores del sistema.

5.1.1 Detailed Description

Sistema de gestión polimórfica de sensores.

Maneja una lista enlazada de punteros a SensorBase, permitiendo almacenar diferentes tipos de sensores juntos.

5.2 GestorSensores.h

Go to the documentation of this file.

```
00001
00008 #ifndef GESTOR_SENSORES_H
00009 #define GESTOR_SENSORES_H
00010
00011 #include "SensorBase.h"
00012 #include <iostream>
00013
00028 struct NodoSensor {
00029     SensorBase* sensor;
00030     NodoSensor* siguiente;
00031
00037     NodoSensor(SensorBase* s) : sensor(s), siguiente(nullptr) {}
00038 };
00039
00056 class GestorSensores {
00057 private:
00058     NodoSensor* cabeza;
00059     int cantidad;
00060
00061 public:
00067     GestorSensores() : cabeza(nullptr), cantidad(0) {
```

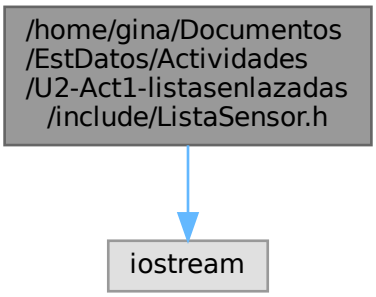
```
00068         std::cout « "\n[GestorSensores] Sistema inicializado." « std::endl;
00069     }
00070
00095     ~GestorSensores() {
00096         std::cout « "\n--- Liberación de Memoria en Cascada ---" « std::endl;
00097         NodoSensor* actual = cabeza;
00098
00099         while (actual != nullptr) {
00100             NodoSensor* siguiente = actual->siguiente;
00101
00102             std::cout « "[Destructor General] Liberando Nodo: " « actual->sensor->obtenerNombre() «
      std::endl;
00103
00104             // Llamamos al destructor polimórfico
00105             delete actual->sensor;  // Libera el sensor virtual
00106             delete actual;          // Libera el nodo de gestión
00107
00108             actual = siguiente;
00109         }
00110
00111         std::cout « "Sistema cerrado. Memoria limpia." « std::endl;
00112     }
00113
00132     void agregarSensor(SensorBase* sensor) {
00133         NodoSensor* nuevoNodo = new NodoSensor(sensor);
00134
00135         if (cabeza == nullptr) {
00136             cabeza = nuevoNodo;
00137             std::cout « "[Gestor] Primer sensor registrado: " « sensor->obtenerNombre() « std::endl;
00138         } else {
00139             NodoSensor* actual = cabeza;
00140             while (actual->siguiente != nullptr) {
00141                 actual = actual->siguiente;
00142             }
00143             actual->siguiente = nuevoNodo;
00144             std::cout « "[Gestor] Sensor agregado: " « sensor->obtenerNombre() « std::endl;
00145         }
00146         cantidad++;
00147     }
00148
00160     // Aquí es donde se realiza la busqueda
00161     SensorBase* buscarSensor(const char* id) {
00162         NodoSensor* actual = cabeza;
00163
00164         // Iteracion para busqueda
00165         while (actual != nullptr) {
00166             // Compara los valores y retorna 0 si es verdadero
00167             if (strcmp(actual->sensor->obtenerNombre(), id) == 0) {
00168                 return actual->sensor;
00169             }
00170
00171             // Si no lo es itera con el siguiente elemento de la lista
00172             actual = actual->siguiente;
00173         }
00174
00175         // No se encontro
00176         return nullptr;
00177     }
00178
00194     void procesarTodos() {
00195         if (cabeza == nullptr) {
00196             std::cout « "[Gestor] No hay sensores para procesar." « std::endl;
00197             return;
00198         }
00199
00200         std::cout « "\n--- Ejecutando Polimorfismo ---" « std::endl;
00201         NodoSensor* actual = cabeza;
00202
00203         while (actual != nullptr) {
00204             // Aqui se aplica el polimorfismo llamando la función correcta
00205             actual->sensor->procesarLectura();
00206             actual = actual->siguiente;
00207         }
00208     }
00209
00213     void listarSensores() const {
00214         if (cabeza == nullptr) {
00215             std::cout « "[Gestor] No hay sensores registrados." « std::endl;
00216             return;
00217         }
00218
00219         std::cout « "\n=== Lista de Sensores Registrados ===" « std::endl;
00220         std::cout « "Total de sensores: " « cantidad  « std::endl « std::endl;
00221
00222         NodoSensor* actual = cabeza;
00223         int index = 1;
00224
```

```
00225         while (actual != nullptr) {
00226             std::cout « index++ « ". ";
00227             actual->sensor->imprimirInfo();
00228             actual = actual->siguiente;
00229         }
00230     }
00231
00236     int obtenerCantidad() const {
00237         return cantidad;
00238     }
00239 };
00240
00241 #endif
```

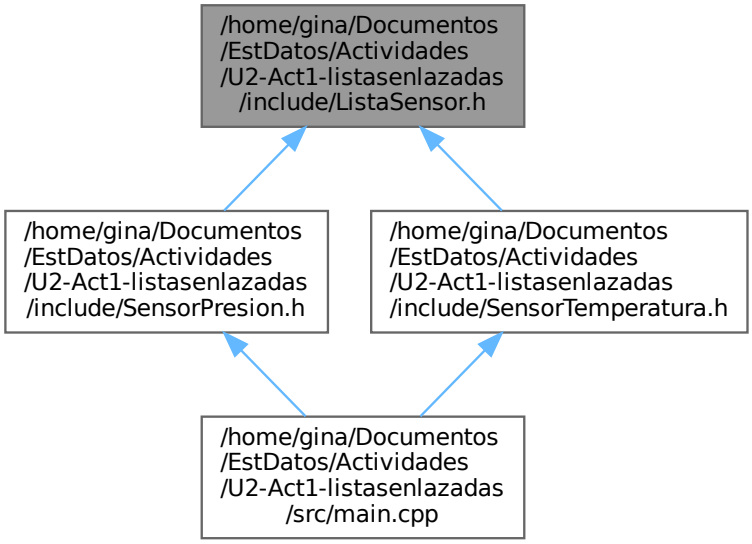
5.3 /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/ListaSensor.h File Reference

Implementación de la Lista Enlazada Simple Genérica para sensores IoT.

`#include <iostream>`
Include dependency graph for ListaSensor.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Nodo< T >

  Nodo genérico para la lista enlazada.
- class ListaSensor< T >

  Lista Enlazada Simple Genérica.

5.3.1 Detailed Description

Implementación de la Lista Enlazada Simple Genérica para sensores IoT.

Esta clase maneja la memoria dinámicamente usando punteros. Implementa la Regla de los Tres para evitar fugas de memoria.

5.4 ListaSensor.h

Go to the documentation of this file.

```
00001
00009 #ifndef LISTA_SENSOR_H
00010 #define LISTA_SENSOR_H
00011
00012 #include <iostream>
00013
00023 template <typename T>
00024 struct Nodo {
00025     T dato;
00026     Nodo<T>* siguiente;
00027
00036     Nodo(T valor) : dato(valor), siguiente(nullptr) {}
00037 };
```

```
00038
00049 template <typename T>
00050 class ListaSensor {
00051 private:
00052     Nodo<T>* cabeza;
00053     int tamaño;
00054
00055 public:
00062     ListaSensor() : cabeza(nullptr), tamaño(0) {
00063         std::cout « "[LOG] Lista genérica creada" « std::endl;
00064     }
00065
00081     ~ListaSensor() {
00082         Nodo<T>* actual = cabeza;
00083         while (actual != nullptr) {
00084             // Guardamos el siguiente
00085             Nodo<T>* siguiente = actual->siguiente;
00086             std::cout « "\t[LOG] Nodo<T> " « actual->dato « " liberado" « std::endl;
00087             // Liberamos memoria
00088             delete actual;
00089             // Avanzamos
00090             actual = siguiente;
00091         }
00092         std::cout « "\t[LOG] Lista genérica destruida" « std::endl;
00093     }
00094
00107     ListaSensor(const ListaSensor& otra) : cabeza(nullptr), tamaño(0) {
00108         Nodo<T>* actual = otra.cabeza;
00109         while (actual != nullptr) {
00110             // Copiamos cada dato
00111             insertarAlFinal(actual->dato);
00112             actual = actual->siguiente;
00113         }
00114     }
00115
00126     ListaSensor& operator=(const ListaSensor& otra) {
00127         // Verificamos que no haya una autoinsección
00128         if (this != &otra) {
00129             // Limpiar lista actual
00130             while (cabeza != nullptr) {
00131                 Nodo<T>* temp = cabeza;
00132                 cabeza = cabeza->siguiente;
00133                 delete temp;
00134             }
00135             tamaño = 0;
00136
00137             // Copiar nueva lista
00138             Nodo<T>* actual = otra.cabeza;
00139             while (actual != nullptr) {
00140                 insertarAlFinal(actual->dato);
00141                 actual = actual->siguiente;
00142             }
00143         }
00144         return *this;
00145     }
00146
00159     void insertarAlFinal(T valor) {
00160         // Creamos un nodo en memoria dinámica
00161         Nodo<T>* nuevoNodo = new Nodo<T>(valor);
00162
00163         // Verificamos si la lista esta vacía
00164         if (cabeza == nullptr) {
00165             cabeza = nuevoNodo;
00166             std::cout « "[LOG] Primer nodo insertado: " « valor « std::endl;
00167         } else {
00168             // Lista con elementos
00169             Nodo<T>* actual = cabeza;
00170
00171             // Avanzar hasta el último nodo
00172             while (actual->siguiente != nullptr) {
00173                 actual = actual->siguiente;
00174             }
00175
00176             // Enganchamos al nuevo nodo
00177             actual->siguiente = nuevoNodo;
00178             std::cout « "[LOG] Nodo insertado al final: " « valor « std::endl;
00179         }
00180
00181         tamaño++;
00182     }
00183
00192     T calcularPromedio() const {
00193
00194         // Mandamos una advertencia si la lista esta vacía
00195         if (cabeza == nullptr) {
00196             std::cout « "[ADVERTENCIA] Lista vacía, retornando 0." « std::endl;
00197             return static_cast<T>(0);
```

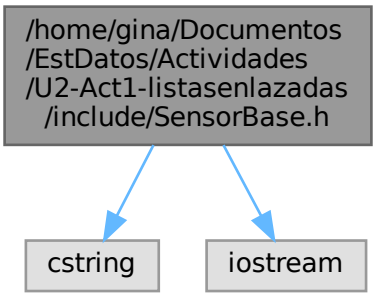
```
00198         }
00199
00200         T suma = 0;
00201         Nodo<T>* actual = cabeza;
00202
00203         // Sumar todos los valores
00204         while (actual != nullptr) {
00205             suma += actual->dato;
00206             actual = actual->siguiente;
00207         }
00208
00209         return suma / tamaño;
00210     }
00211
00223     T eliminarMinimo() {
00224
00225         // Avisamos de error si no hay dato
00226         if (cabeza == nullptr) {
00227             std::cout << "[ERRROR] No hay elementos para eliminar." << std::endl;
00228             return static_cast<T>(0);
00229         }
00230
00231         // Buscar el nodo con valor mínimo
00232         Nodo<T>* minNodo = cabeza;
00233         Nodo<T>* anterior = nullptr;
00234         Nodo<T>* anteriorMin = nullptr;
00235         Nodo<T>* actual = cabeza;
00236
00237         // Recorremos para encontrar el valor mìnimo
00238         while (actual != nullptr) {
00239             if (actual->dato < minNodo->dato) {
00240                 minNodo = actual;
00241                 anteriorMin = anterior;
00242             }
00243             anterior = actual;
00244             actual = actual->siguiente;
00245         }
00246
00247         T valorMin = minNodo->dato;
00248
00249         // Caso especial si el mínimo es la cabeza
00250         if (minNodo == cabeza) {
00251             cabeza = cabeza->siguiente;
00252         } else {
00253             anteriorMin->siguiente = minNodo->siguiente;
00254         }
00255
00256         delete minNodo;
00257         tamaño--;
00258
00259         std::cout << "[LOG] Valor mínimo eliminado: " << valorMin << std::endl;
00260         return valorMin;
00261     }
00262
00268     void imprimir() const {
00269
00270         // Avisamos si la lista esta vacía
00271         if (cabeza == nullptr) {
00272             std::cout << "[WARNING] Lista vacía\n";
00273             return;
00274         }
00275
00276         Nodo<T>* actual = cabeza;
00277         std::cout << "[Lista: ";
00278         while (actual != nullptr) {
00279             std::cout << actual->dato;
00280             if (actual->siguiente != nullptr) {
00281                 std::cout << " → ";
00282             }
00283             actual = actual->siguiente;
00284         }
00285         std::cout << "]" << std::endl;
00286     }
00287
00292     int obtenerTamaño() const {
00293         return tamaño;
00294     }
00295
00301     bool estaVacia() const {
00302         return cabeza == nullptr;
00303     }
00304 };
00305
00306 #endif
```

5.5 /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/SensorBase.h File Reference

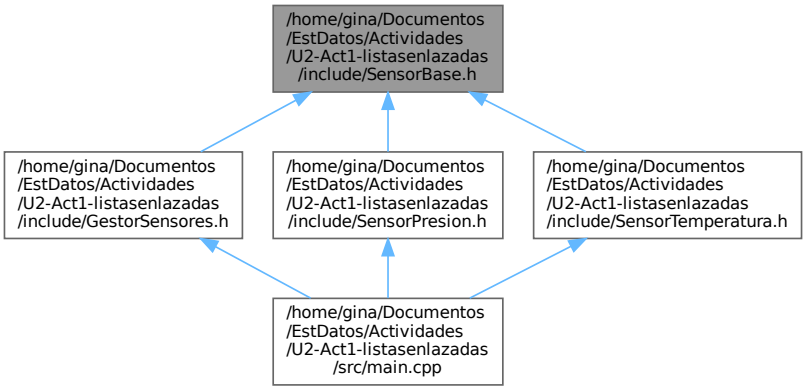
Clase base abstracta para todos los tipos de sensores.

```
#include <cstring>
#include <iostream>
```

Include dependency graph for SensorBase.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SensorBase

  Clase abstracta que representa un sensor genérico.

5.5.1 Detailed Description

Clase base abstracta para todos los tipos de sensores.

Define la interfaz común que todos los sensores deben implementar. Usa métodos virtuales puros para forzar la implementación en clases derivadas.

5.6 SensorBase.h

Go to the documentation of this file.

```
00001
00009 #ifndef SENSOR_BASE_H
00010 #define SENSOR_BASE_H
00011
00012 #include <cstring>
00013 #include <iostream>
00014
00015
00033 class SensorBase {
00034 protected:
00042     char nombre[50];
00043
00044 public:
00053     SensorBase(const char* id) {
00054         strcpy(nombre, id);
00055         std::cout « "[SensorBase] Sensor '" « nombre « "' creado" « std::endl;
00056     }
00057
00064     virtual ~SensorBase() {
00065         std::cout « "[SensorBase] Destructor base llamado para '" « nombre « "'.\n";
00066     }
00067
00081     virtual void procesarLectura() = 0;
00082
00089     virtual void imprimirInfo() const = 0;
00090
00099     virtual void agregarLectura(const char* valor) = 0;
00100
00108     const char* obtenerNombre() const {
00109         return nombre;
00110     }
00111 };
00112
00113 #endif
```

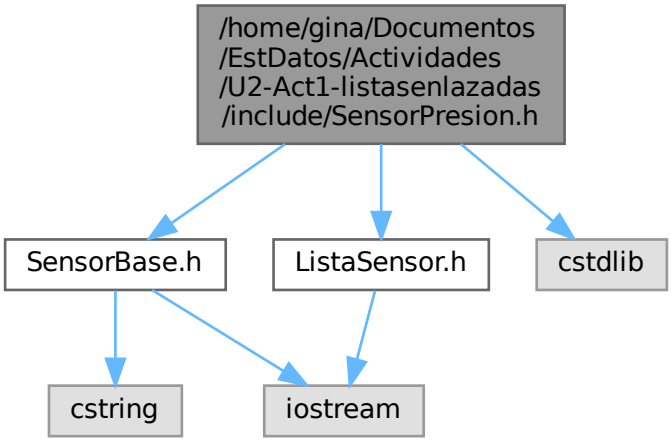
5.7 /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/SensorPresion.h File Reference

Implementación concreta de un sensor de presión.

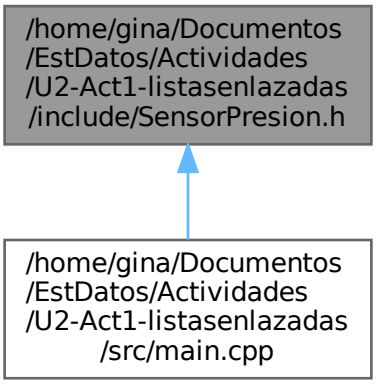
```
#include "SensorBase.h"
#include "ListaSensor.h"
```

```
#include <cstdlib>
```

Include dependency graph for SensorPresion.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SensorPresion

  Sensor que maneja presiones en formato enteros.

5.7.1 Detailed Description

Implementación concreta de un sensor de presión.

Maneja lecturas de tipo int valores de presion

5.8 SensorPresion.h

Go to the documentation of this file.

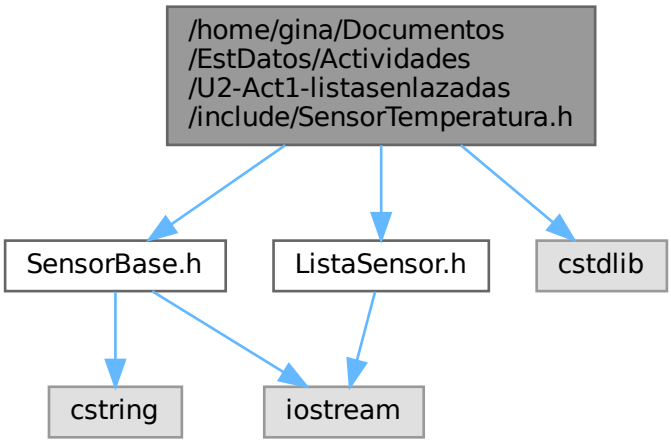
```
00001
00007 #ifndef SENSOR_PRESION_H
00008 #define SENSOR_PRESION_H
00009
00010 #include "SensorBase.h"
00011 #include "ListaSensor.h"
00012
00013 // para convertir de string a int
00014 #include <cstdlib>
00015
00029 class SensorPresion : public SensorBase {
00030 private:
00037     ListaSensor<int> historial;
00038
00039 public:
00044     SensorPresion(const char* id) : SensorBase(id) {
00045         std::cout << "[SensorPresion] Sensor de presión '" << nombre << "' inicializado." << std::endl;
00046     }
00047
00055     ~SensorPresion() {
00056         std::cout << "  [Destructor Sensor " << nombre << "] Liberando Lista Interna..." << std::endl;
00057     }
00058
00068     void agregarLectura(const char* valor) override {
00069         // String a entero
00070         int presion = atoi(valor);
00071         historial.insertarAlFinal(presion);
00072         std::cout << "[SensorPresion " << nombre << "] Lectura agregada: " << presion << " hPa" <<
      std::endl;
00073     }
00074
00089     void procesarLectura() override {
00090         std::cout << "\n-> Procesando Sensor " << nombre << "...";
00091
00092         // Si no hay registros
00093         if (historial.estaVacia()) {
00094             std::cout << "[SensorPresion] No hay lecturas para procesar." << std::endl;
00095             return;
00096         }
00097
00098         int promedio = historial.calcularPromedio();
00099         std::cout << "[Sensor Presion] Promedio calculado sobre " << historial.obtenerTamaño()
00100                   << " lectura(s): " << promedio << " hPa." << std::endl;
00101     }
00102
00106     void imprimirInfo() const override {
00107         std::cout << "\n=== Sensor de Presión ===" << std::endl;
00108         std::cout << "ID: " << nombre << std::endl;
00109         std::cout << "Tipo: Presión (int)" << std::endl;
00110         std::cout << "Lecturas almacenadas: " << historial.obtenerTamaño() << std::endl;
00111         historial.imprimir();
00112         std::cout << "=========================" << std::endl;
00113     }
00114 };
00115
00116 #endif
```

5.9 /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/include/SensorTemperatura.h File Reference

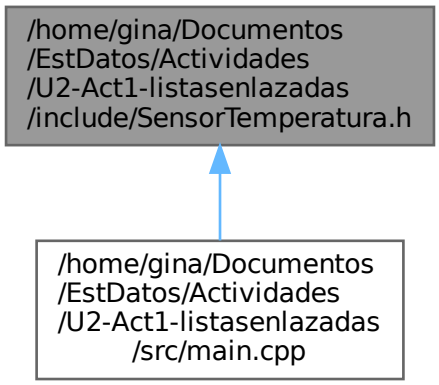
Implementación concreta de un sensor de temperatura.

```
#include "SensorBase.h"
#include "ListaSensor.h"
#include <cstdlib>
```

Include dependency graph for SensorTemperatura.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SensorTemperatura

  Sensor que maneja temperaturas en formato float.

5.9.1 Detailed Description

Implementación concreta de un sensor de temperatura.

Maneja lecturas de tipo float y las almacena en una lista enlazada genérica

5.10 SensorTemperatura.h

Go to the documentation of this file.

```
00001
00007 #ifndef SENSOR_TEMPERATURA_H
00008 #define SENSOR_TEMPERATURA_H
00009
00010 #include "SensorBase.h"
00011 #include "ListaSensor.h"
00012 #include <cstdlib>  // Para convertir string a float
00013
00029 class SensorTemperatura : public SensorBase {
00030 private:
00042     ListaSensor<float> historial;
00043
00044 public:
00059     SensorTemperatura(const char* id) : SensorBase(id) {
00060         std::cout « "[SensorTemp] Sensor de temperatura '" « nombre « "' inicializado." « std::endl;
00061     }
00062
00075     ~SensorTemperatura() {
00076         std::cout « "\t[Destructor Sensor " « nombre « "] Liberando Lista Interna..." « std::endl;
00077     }
00078
00089     void agregarLectura(const char* valor) override {
00090         // Convertir string a float
00091         float temp = atof(valor);
00092         historial.insertarAlFinal(temp);
00093         std::cout « "[SensorTemp " « nombre « "] Lectura agregada: " « temp « "°C" « std::endl;
00094     }
00095
00108     void procesarLectura() override {
00109         std::cout « "\n-> Procesando Sensor " « nombre « "..." « std::endl;
00110
00111         // Si no hay nada solo retornamos
00112         if (historial.estaVacia()) {
00113             std::cout « "[SensorTemp] No hay lecturas para procesar." « std::endl;
00114             return;
00115         }
00116
00117         // Si hay mas de de un registro se elimina al mas chiquito
00118         if (historial.obtenerTamaño() > 1) {
00119             float minimo = historial.eliminarMinimo();
00120             std::cout « "[Sensor Temp] Lectura más baja (" « minimo « "°C) eliminada." « std::endl;
00121             return;
00122         }
00123
00124         float promedio = historial.calcularPromedio();
00125         std::cout « "[Sensor Temp] Promedio calculado sobre "
00126                   « historial.obtenerTamaño() « " lectura(s): "  « promedio « "°C." « std::endl;
00127     }
00128
00136     void imprimirInfo() const override {
00137         std::cout « "\n=== Sensor de Temperatura ===\n";
00138         std::cout « "ID: " « nombre « std::endl;
00139         std::cout « "Tipo: Temperatura (float)" « std::endl;
00140         std::cout « "Lecturas almacenadas: " « historial.obtenerTamaño() « std::endl;
00141         historial.imprimir();
00142         std::cout « "============================\n";
00143     }
00144 };
00145
00146 #endif
```

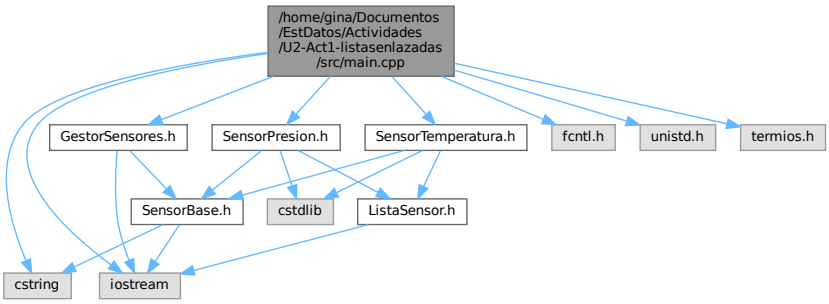
5.11 /home/gina/Documentos/EstDatos/Actividades/U2-Act1-listasenlazadas/src/main.cpp File Reference

Programa principal del Sistema de Gestión Polimórfica de Sensores para IoT.

```
#include "GestorSensores.h"
#include "SensorTemperatura.h"
#include "SensorPresion.h"
#include <iostream>
#include <fcntl.h>
```

```
#include <unistd.h>
#include <termios.h>
#include <cstring>
```

Include dependency graph for main.cpp:



Functions

- int configurarSerial (const char *puerto, int baudrate)

  Configuración del puerto serial para ESP32.
- bool leerLineaSerial (int fd, char *buffer, int maxLen)

  Lee una línea completa desde el puerto serial.
- void procesarLinea (char *linea, GestorSensores &gestor)

  Procesa una línea recibida del ESP32.
- int main ()

  Función principal del programa.

5.11.1 Detailed Description

Programa principal del Sistema de Gestión Polimórfica de Sensores para IoT.

Integra comunicación serial con ESP32 y gestión polimórfica de sensores

FLUJO DEL PROGRAMA:

1. Inicializar el gestor de sensores
2. Configurar comunicación serial con ESP32
3. Crear sensores de diferentes tipos
4. Recibir lecturas desde el puerto serial
5. Procesar datos polimórficamente
6. Liberar memoria automáticamente

5.11.2 Function Documentation

5.11.2.1 configurarSerial()

```
int configurarSerial (
            const char * puerto,
            int baudrate )
```

Configuración del puerto serial para ESP32.

Returns

Descriptor de archivo del puerto, o -1 si falla

CONCEPTO: Puerto Serial: Es un canal de comunicación entre la computadora y el ESP32. Funciona como un "cable virtual" por donde viajan datos.

PARÁMETROS CLAVES:

- Baudrate: Velocidad de transmisión (bits por segundo)
  - 9600: Lento pero confiable
  - 115200: Rápido, ideal para sensores

Here is the caller graph for this function:

5.11.2.2 leerLineaSerial()

```
bool leerLineaSerial (
            int fd,
            char * buffer,
            int maxLen )
```

Lee una línea completa desde el puerto serial.

Parameters

fd	Descriptor del puerto serial
buffer	Buffer donde se almacenará la línea
maxLen	Tamaño máximo del buffer

Returns

true si se leyó una línea completa, false en caso contrario

CONCEPTO - Lectura Serial: Los datos llegan byte por byte. Debemos acumularlos hasta encontrar un carácter de nueva línea ('
').

FORMATO ESPERADO DEL ESP32: "T,T-001,23.5\n" "P,P-105,1013\n"

Donde:

- Primer campo: Tipo (T=Temperatura, P=Presión)
- Segundo campo: ID del sensor
- Tercer campo: Valor de la lectura

PROCESO:

1. Leer caracteres uno por uno
2. Acumular en buffer
3. Si encontramos '
   ', terminamos
4. Retornar true si la línea está completa

Here is the caller graph for this function:

5.11.2.3 main()

```
int main ( )
```

Función principal del programa.

FLUJO:

1. Crear el gestor de sensores
2. Configurar puerto serial
3. Bucle principal:
   - Leer datos del ESP32
   - Crear sensores dinámicamente
   - Acumular lecturas
4. Procesar todos los sensores
5. Limpiar memoria (automático por destructores)

5.11.2.4 procesarLinea()

```
void procesarLinea (
            char * linea,
            GestorSensores & gestor )
```

Procesa una línea recibida del ESP32.

Parameters

linea	String con formato "TIPO,ID,VALOR"
gestor	Referencia al gestor de sensores

FORMATO DE LÍNEA: "T,T-001,23.5" -> Temperatura, ID=T-001, Valor=23.5 "P,P-105,1013" -> Presión, ID=P-105, Valor=1013

PROCESO DE PARSING:

1. Dividir la línea en campos usando ',' como separador
2. Campo 0: Tipo de sensor (T o P)
3. Campo 1: ID del sensor
4. Campo 2: Valor de la lectura
5. Buscar el sensor en el gestor
6. Si no existe, crearlo
7. Agregar la lectura al sensor

TÉCNICA - strtok: strtok divide un string usando delimitadores. Primera llamada: strtok(linea, ",") Siguientes: strtok(nullptr, ",") Here is the caller graph for this function:

Index