

# LexRis Logic Headers V2.00

## Guía del Programador

LexRis Logic

## 1 Guía Inicial

Las Cabeceras de LexRis Logic brindan un código amigable orientado a objetos y funciones en C++ de Matemática Básica, Temporizador, Conversiones de Tipo, Encriptador, Lector de Archivos, Separador de Cadenas, Estructuras Matemáticas, Estructuras de Datos y en especial de las librerías:

- Allegro v5.2.0
- ENet v1.3.13
- irrKlang v1.5.0 (API)

## 2 namespace **LL\_AL5**

Espacio de nombres que incluye código amigable de la librería para programación de videojuegos: Allegro 5.

### 2.1 Allegro 5 (**LexRisLogic/Allegro5/Allegro.h**)

#### 2.1.1 Definición de Tipo: Posición.

**typedef float Type\_pos**

Tipo de dato usado para indicar posiciones en el eje x e y de todos los objetos a poderse dibujar en pantalla.

#### 2.1.2 Definición de Tipo: Tamaño de Pantalla.

**typedef unsigned int Type\_display\_size**

Tipo de dato usado para indicar valores de tamaño de pantalla objetivo o real.

#### 2.1.3 Variable: **extern float** bitmap\_scale\_x

Variable que contiene el valor actual de la escala de las imágenes en el eje x para ejecutar las operaciones de dibujo.

#### 2.1.4 Variable: **extern float** bitmap\_scale\_y

Variable que contiene el valor actual de la escala de las imágenes en el eje y para ejecutar las operaciones de dibujo.

#### 2.1.5 Variable: **extern float** text\_scale

Variable que contiene el valor actual de la escala de las fuentes de texto para ejecutar las operaciones de dibujo.

#### 2.1.6 Variable: `extern float primitives_scale`

Variable que contiene el valor actual de la escala de las figuras primitivas para ejecutar las operaciones de dibujo.

#### 2.1.7 Variable: `extern Type_display_size desktop_size_x`

Variable que contiene el valor del tamaño de pantalla del escritorio en el eje x que es actualizado cuando el sistema Allegro es iniciado, por defecto empieza con valor de 640.

#### 2.1.8 Variable: `extern Type_display_size desktop_size_y`

Variable que contiene el valor del tamaño de pantalla del escritorio en el eje y que es actualizado cuando el sistema Allegro es iniciado, por defecto empieza con valor de 480.

#### 2.1.9 Función: Iniciar Sistema Allegro.

`void init_allegro()`

Inicia el sistema Allegro y actualiza el tamaño de la pantalla de escritorio si se encuentra disponible.

#### 2.1.10 Función: Pausar.

`void sleep(float sleep_time)`

Pausa la aplicación durante un tiempo pasado en segundos.

## 2.2 Sistema de Archivos (**LexRisLogic/Allegro5/Path.h**)

### 2.2.1 Función: Obtener Directorio Actual.

**std::string** get\_current\_directory()

Retorna el directorio actual donde se esta ejecutando el programa.

### 2.2.2 Función: Cambiar Directorio de Ejecución.

**bool** change\_directory(**std::string** new\_path)

Cambia el directorio donde se ejecuta el programa, retorna verdadero en caso de éxito.

### 2.2.3 Función: Crear Directorio.

**bool** make\_directory(**std::string** new\_dir)

Crea un directorio nuevo de acuerdo a la ruta pasada por parámetro, retorna verdadero en caso de éxito.

### 2.2.4 Función: Revisar Existencia de una Ruta.

**bool** path\_exists(**std::string** path)

Retorna verdadero si la ruta pasada por parámetro existe en el sistema de archivos.

### 2.2.5 Función: Remover Ruta.

**bool** remove\_path(**std::string** path)

Elimina completamente la ruta pasada por parámetro del sistema de archivos, puede eliminar con éxito archivos y carpetas vacías.

## 2.3 Cuadros de Diálogo (LexRisLogic/Allegro5/NativeDialog.h)

### 2.3.1 Función: Iniciar Complemento Cuadros de Diálogo Nativo.

**bool** native\_dialog\_addon()

Inicia el complemento de cuadros de diálogo nativo para su uso completo, retorna verdadero en caso de éxito.

### 2.3.2 Función: Mostrar Cuadro de Diálogo.

**bool** show\_native\_message(**ALLEGRO\_DISPLAY\*** display, **std::string** title, **std::string** header, **std::string** message, **int** flag)

Muestra un cuadro de diálogo bloqueando el display enviado si es diferente de nulo, donde la cadena *title* es el título del cuadro, la cadena *header* es el título del mensaje, la cadena *message* es el mensaje a mostrar al usuario y el *flag* es el tipo de mensaje pudiendo ser los siguientes:

1. **ALLEGRO\_MESSAGEBOX\_WARN**  
Muestra un mensaje de tipo alerta.
2. **ALLEGRO\_MESSAGEBOX\_ERROR**  
Muestra un mensaje de tipo error.
3. **ALLEGRO\_MESSAGEBOX\_QUESTION**  
Muestra un mensaje simple.
4. **ALLEGRO\_MESSAGEBOX\_OK\_CANCEL**  
Muestra un cuadro con opciones de aceptar o cancelar.
5. **ALLEGRO\_MESSAGEBOX\_YES\_NO**  
Muestra un cuadro con opciones de si o no.

Retorna verdadero en caso de haber recibido como respuesta si o aceptar, y falso si la ventana a recibido alguna cancelación, una respuesta de no o cancelar, o fue cerrada por el usuario.

### 2.3.3 Clase: FileChooser

Clase que se encarga de crear un explorador de archivos donde el usuario indicara la ruta del o los archivos seleccionados.

**Funciones:**

- **void** set\_display(**ALLEGRO\_DISPLAY\*** display)  
Asigna la pantalla a bloquearse cuando se abra el explorador de archivos, puede ser nulo.
- **ALLEGRO\_DISPLAY\*** get\_display()  
Retorna un puntero a la pantalla que será bloqueada por el explorador de archivos.
- **void** set\_initial\_path(**std::string** new\_search\_path)  
Indica la ruta inicial del explorador de archivos por donde empezará la búsqueda.
- **std::string** get\_initial\_path()  
Retorna la ruta inicial del explorador de archivos.
- **void** set\_title(**std::string** new\_title)  
Cambia el título del explorador de archivos.
- **std::string** get\_title()  
Retorna el título del explorador de archivos.
- **void** set\_patterns(**std::string** new\_patterns)  
Cambia la lista de archivos admitidos (patrones) separado por ;.

- **std::string get\_patterns()**  
Retorna los patrones que serán permitidos por el explorador de archivos.
- **void set\_mode(int new\_mode)**  
Indica el modo del explorador de archivos y su finalidad con los siguientes flags o combinaciones:
  1. **ALLEGRO\_FILECHOOSER\_FILE\_MUST\_EXIST**  
Permite una visualización simple de búsqueda de un archivo.
  2. **ALLEGRO\_FILECHOOSER\_SAVE**  
El explorador entra en el modo de salvado de un archivo mediante la entrada de un nuevo nombre para el archivo.
  3. **ALLEGRO\_FILECHOOSER\_FOLDER**  
Crea un explorador de archivos para búsqueda de carpetas.
  4. **ALLEGRO\_FILECHOOSER\_PICTURES**  
Permite un explorador con visualización de imágenes.
  5. **ALLEGRO\_FILECHOOSER\_SHOW\_HIDDEN**  
Permite la visualización de archivos ocultos.
  6. **ALLEGRO\_FILECHOOSER\_MULTIPLE**  
Habilita la selección de varios archivos en el explorador de archivos.
- **bool start\_filechooser()**  
Inicia el explorador de archivos, retorna verdadero si se consiguió ruta alguna.
- **unsigned int get\_number\_of\_selected\_files()**  
Retorna el número de archivos conseguidos.

#### Operadores:

- **std::string operator [] (unsigned int index)**  
Obtiene una ruta que consiguió el explorador de archivos.

#### Destruyores:

- **~ FileChooser()**  
Destruye la interfaz creada para el explorador de archivos.

### 2.3.4 Clase: **TextLog**

Clase que crea un registro de texto.

#### Funciones:

- **void set\_title(std::string new\_title)**  
Cambia el título del Registro.
- **std::string get\_title()**  
Retorna el título del Registro.
- **void set\_mode(int new\_mode)**  
Indica el Comportamiento de la ventana de registro donde se puede combinar los siguientes flags:
  1. **ALLEGRO\_TEXTLOG\_NO\_CLOSE**  
Evita que la ventana del registro genere eventos de cerrado.
  2. **ALLEGRO\_TEXTLOG\_MONOSPACE**  
La ventana de Registro usara una fuente de espacio sencillo para mostrar el texto.
- **bool open\_textlog()**  
Abre la ventana de registro, retorna verdadero en caso de éxito.

- **bool** `write_text(std::string text)`  
Escribe una cadena en el registro, retorna verdadero en caso de éxito.
- **bool** `write_endl()`  
Escribe un salto de línea en el registro, retorna verdadero en caso de éxito.
- **bool** `close_textlog()`  
Cierra la ventana de registro, retorna verdadero en caso de éxito.
- **bool** `is_open()`  
Retorna verdadero si la ventana de registro esta abierta.

#### Operadores:

- **operator** `ALLEGRO_TEXTLOG*` `()`  
Retorna un puntero `ALLEGRO_TEXTLOG` para que pueda ser usado con las funciones restantes de Allegro.

#### Destruyores:

- `~` `TextLog()`  
Cierra y libera todo lo utilizado por la ventana de registro.

## 2.4 Pantalla (LexRisLogic/Allegro5/Display.h)

### 2.4.1 Clase: Display

Clase que provee toda una interfaz para manejar pantallas objetivos de operaciones de dibujo.

**Constructores:**

- **Display**(**Type\_display\_size** size\_x, **Type\_display\_size** size\_y)  
Crea una pantalla de tamaño  $size\_x * size\_y$  para ser objetivo de las operaciones de dibujo, el tamaño en el que trabajaran las posiciones sera igual al tamaño de la pantalla y las escalas globales serán normales.
- **Display**(**Type\_display\_size** size\_x, **Type\_display\_size** size\_y, **Type\_display\_size** real\_size\_x, **Type\_display\_size** real\_size\_y)  
Crea una pantalla de tamaño  $size\_x * size\_y$  para ser objetivo de las operaciones de dibujo, el tamaño en el que trabajaran las posiciones sera igual a una pantalla de tamaño  $real\_size\_x * real\_size\_y$  y en cuanto a las escalas serán actualizadas para trabajar con este tamaño de trabajo (Tamaño Real).

**Funciones:**

- **void set\_title(std::string new\_title)**  
Cambia el título de la pantalla creada.
- **Type\_display\_size get\_size\_x()**  
Retorna el tamaño del eje x de la pantalla.
- **Type\_display\_size get\_size\_y()**  
Retorna el tamaño del eje y de la pantalla.
- **void set\_real\_size(Type\_display\_size real\_size\_x, Type\_display\_size real\_size\_y)**  
Cambia el tamaño en el que se trabaja actualizando la posición de la cámara y las escalas.
- **Type\_display\_size get\_real\_size\_x()**  
Retorna el tamaño del eje x en el que se trabaja.
- **Type\_display\_size get\_real\_size\_y()**  
Retorna el tamaño del eje y en el que se trabaja.
- **void set\_flag(int new\_flag)**  
Un flag indica de que tipo sera la pantalla, al ser modificado el flag, la pantalla se volverá a crear con los nuevos parámetros, este flag toma combinaciones de las macros de Allegro:
  1. **ALLEGRO\_WINDOWED**  
Para una pantalla de tamaño fijo en forma de ventana.
  2. **ALLEGRO\_FULLSCREEN**  
Utiliza el modo pantalla completa del sistema operativo, usar mas de una actualización seguida en este modo puede causar que el renderizado no sea óptimo.
  3. **ALLEGRO\_FULLSCREEN\_WINDOW**  
Genera una pantalla completa en forma de ventana, no genera errores con la actualización seguida.
  4. **ALLEGRO\_RESIZABLE**  
Solo combinable con **ALLEGRO\_WINDOWED**, logra que el tamaño de la pantalla pueda ser manejado por el usuario, activando la opción maximizar.

- **void** `resize(Type_display_size size_x, Type_display_size size_y)`  
Redimensiona la pantalla a tamaño *size\_x\*size\_y* actualizando valores de escala con respecto al tamaño real, solo se puede usar en modo **ALLEGRO\_WINDOWED** o **ALLEGRO\_RESIZABLE**.
- **void** `set_cam(Type_pos new_pos_x, Type_pos new_pos_y)`  
Cambia la posición de la cámara, será visible todo lo dibujado en pantalla desde las coordenadas (x,y) hasta el tamaño real en el que se trabaja.
- **void** `set_cam_x(Type_pos new_pos_x)`  
Cambia la posición de la cámara en el eje x.
- **Type\_pos** `get_cam_x()`  
Retorna el posición de la cámara en el eje x.
- **void** `set_cam_y(Type_pos new_pos_y)`  
Cambia la posición de la cámara en el eje y.
- **Type\_pos** `get_cam_y()`  
Retorna el posición de la cámara en el eje y.
- **void** `plus_cam_x(Type_pos plus_x)`  
Aumenta la posición de la cámara en el eje x.
- **void** `plus_cam_y(Type_pos plus_y)`  
Aumenta la posición de la cámara en el eje y.
- **Type\_pos** `display_to_real_pos_x(Type_pos pos_x, bool references_axes=true)`  
Transforma la posición x pasada por parámetro que toma como referencia al tamaño de la pantalla y la transforma a una coordenada real, es decir, a una coordenada en la que se está trabajando; el segundo parámetro es para tomar en cuenta la posición de la cámara.
- **Type\_pos** `display_to_real_pos_y(Type_pos pos_y, bool references_axes=true)`  
Transforma la posición y pasada por parámetro que toma como referencia al tamaño de la pantalla y la transforma a una coordenada real, es decir, a una coordenada en la que se está trabajando; el segundo parámetro es para tomar en cuenta la posición de la cámara.
- **bool** `show_cursor()`  
Vuelve visible al cursor del ratón en la pantalla.
- **bool** `hide_cursor()`  
Oculta el cursor del ratón en la pantalla.
- **bool** `set_cursor(ALLEGRO_BITMAP* bitmap_cursor, int focus_x, int focus_y)`  
Asigna un mapa de bits al cursor del ratón indicando que punto de la imagen es el punto de enfoque del ratón.
- **bool** `destroy_cursor()`  
Destruye y libera la memoria utilizada por el cursor del ratón.
- **void** `clear()`  
Limpia la pantalla con el color blanco por defecto.
- **void** `clear_to_color(ALLEGRO_COLOR color)`  
Limpia la pantalla con el color enviado por parámetro.



- **template<typename T>**  
**void draw(T\* object, bool references\_axes=true)**

Función que recibe una clase **T** que debe tener los siguientes métodos implementados:

1. **Type\_pos get\_pos\_x()** → Usado para obtener la posición x del objeto teniendo en cuenta que trabaja conforme al tamaño real que se ha asignado.
2. **Type\_pos get\_pos\_y()** → Usado para obtener la posición y del objeto teniendo en cuenta que trabaja conforme al tamaño real que se ha asignado.
3. **void set\_pos(Type\_pos new\_pos\_x, Type\_pos new\_pos\_y)** → Usado para asignar la verdadera posición conforme al tamaño real que se ha asignado y al tamaño de la pantalla objetivo, terminada la operación de dibujo se reasignara su verdadera posición.
4. **void draw()** → Usado para dibujar al objeto, tener en cuenta que su posición será cambiada y aprovechar las escalas globales para una correcta operación de dibujo.

La operación de dibujo puede ser controlada con el parámetro `references_axes` para indicar si se tomará en cuenta la posición de la cámara, por defecto siempre se toma en cuenta esta posición.

- **void refresh()**  
Actualiza y muestra la imagen generada en pantalla.
- **void set\_target()**  
Asigna la pantalla como objetivo de las operaciones de dibujo.

#### Operadores:

- **operator ALLEGRO\_DISPLAY\* ()**  
Retorna un puntero **ALLEGRO\_DISPLAY** para que pueda ser usado con las funciones restantes de Allegro.
- **operator ALLEGRO\_BITMAP\* ()**  
Retorna una mapa de bits especial representando al buffer de la pantalla, tener mucho cuidado con el uso de otras operaciones de Allegro con este mapa de bits.
- **operator ALLEGRO\_MOUSE\_CURSOR\* ()**  
Retorna un puntero **ALLEGRO\_MOUSE\_CURSOR** para que pueda ser usado con las funciones restantes de Allegro.

#### Destruyores:

- **~ Display()**  
Destruye la pantalla y el cursor del ratón liberando la memoria utilizada.

## 2.5 Temporizador (LexRisLogic/Allegro5/Timer.h)

### 2.5.1 Clase: **Timer**

Clase que representa a un temporizador que genera eventos periódicamente.

**Funciones:**

- **bool** set\_speed\_seconds(**double** new\_speed\_seconds)  
Asigna cada cuanto tiempo en segundos el temporizador generará eventos, retorna verdadero en caso de éxito.
- **double** get\_speed\_seconds(**double** new\_speed\_seconds)  
Retorna el valor del tiempo que toma cada *tick* en segundos.
- **bool** create()  
Crea el temporizador inicialmente detenido.
- **bool** destroy()  
Destruye el temporizador, tener cuidado cuando se asigna el temporizador a los eventos.
- **bool** start()  
Inicia el temporizador empezando a generar eventos.
- **bool** resume()  
Resume el conteo de generación de eventos del temporizador.
- **bool** stop()  
Detiene la generación de eventos del temporizador.

**Operadores:**

- **operator ALLEGRO\_TIMER\*** ()  
Retorna un puntero **ALLEGRO\_TIMER** para que pueda ser usado con las funciones restantes de Allegro.

**Destructores:**

- **~ Timer()**  
Destruye el temporizador y libera la memoria utilizada.

## 2.6 Entradas (LexRisLogic/Allegro5/Input.h)

### 2.6.1 Clase: **KeyControl**

Clase que representa una estructura básica para controlar los eventos de teclas por llaves.

**Funciones:**

- **bool add\_key(std::string key\_name, int keycode)**  
Inserta una llave a la estructura asignándole un nombre y su código de tecla.
- **bool find\_key(std::string key\_name)**  
Busca si existe una llave con el nombre enviado por parámetro, retorna verdadero en caso de éxito.
- **bool find\_key(int keycode, std::string\* key\_name=NULLPTR)**  
Busca si existe una llave con el código de tecla enviado por parámetro, retorna verdadero en caso de éxito; puede obtenerse el nombre de la llave que tiene asignado el código de tecla enviando un puntero a una cadena.
- **bool mod\_key(std::string last\_key\_name, std::string new\_key\_name)**  
Cambia el nombre de una llave con otro, retorna verdadero en caso de éxito.
- **bool mod\_key(std::string key\_name, int new\_keycode)**  
Cambia el código de tecla de la llave.
- **bool remove\_key(std::string key\_name)**  
Remueve la llave que tenga el nombre enviado por parámetro.
- **bool remove\_key(int keycode)**  
Remueve la llave que tenga el código de tecla enviado por parámetro.
- **unsigned int size()**  
Retorna el número de llaves que están en la estructura.
- **void clear\_key\_status()**  
Cambia el estado de todas las llaves a falso, es decir, sin presionar.
- **void clear()**  
Vacía la estructura de llaves.
- **bool& get\_key\_down\_status(std::string key\_name)**  
Obtiene el estado de la llave que tenga el nombre pasado por parámetro, retorna por referencia para su modificación, si la llave no existe retorna un comodín evitando errores, revisar el nombre de las llaves cuando se use esta estructura.
- **int get\_keycode(std::string key\_name)**  
Retorna el código de tecla que se asigno a una llave que tiene el nombre pasado por parámetro.

**Destructores:**

- **~ KeyControl()**  
Destruye la estructura y libera la memoria utilizada.

### 2.6.2 Clase: **Input**

Clase que provee una interfaz para manejar la mayoría de eventos que nos permite controlar Allegro como el ratón, teclado, pantallas, ventanas de registro y temporizadores.

#### Constructores:

- **Input()**  
Instala el sistema para manejo del ratón y del teclado, creando también la cola de eventos.

#### Funciones:

- **bool unregister\_timer()**  
Anula el registro del temporizador, desactiva el control de eventos del temporizador, retorna verdadero en caso de éxito.
- **bool register\_timer(**ALLEGRO\_TIMER\*** new\_timer)**  
Registra un temporizador si no hay uno ya registrado, activa el control de eventos del temporizador, retorna verdadero en caso de éxito.
- **bool unregister\_display()**  
Anula el registro de la pantalla y desactiva el control de eventos de la pantalla, retorna verdadero en caso de éxito.
- **bool register\_display(**ALLEGRO\_DISPLAY\*** new\_display)**  
Registra una pantalla solo si no hay una ya registrada, activa el control de eventos de la pantalla, retorna verdadero en caso de éxito.
- **bool unregister\_textlog()**  
Anula el registro de la ventana de registro, desactiva el control de eventos de la ventana de registro, retorna verdadero en caso de éxito.
- **bool register\_textlog(**ALLEGRO\_TEXTLOG\*** new\_textlog)**  
Registra una ventana de registro si no hay una ya registrada, activa el control de eventos de la ventana de registro, retorna verdadero en caso de éxito.
- **void set\_key\_control(**KeyControl\*** new\_key\_control)**  
Establece la estructura de llaves a utilizar, se envía el puntero para evitar una copia de la estructura interna.
- **KeyControl\* get\_key\_control()**  
Retorna un puntero a la estructura de llaves que se esta usando actualmente.
- **void clear\_events()**  
Vacía la cola de eventos, si la cola de eventos esta activa y no esta siendo controlada, empezara a generar eventos en tiempos no deseados.
- **void clear\_key\_status()**  
Establece el estado de cada llave de la estructura de llaves utilizada actualmente a falso.
- **bool set\_wait\_time(float wait\_time)**  
Asigna el tiempo de espera de generación de eventos.
- **float get\_wait\_time()**  
Retorna el tiempo de espera asignado para la generación de eventos, por defecto la espera es 0, un tiempo inmediato.
- **bool keyboard\_on()**  
Registra los eventos del teclado.
- **bool keyboard\_off()**  
Anula el registro del teclado, y deja de controlar sus eventos generados.

- **bool input\_on(std::string\* input\_objetivo, unsigned int max\_input\_size, bool special\_is\_blocked=false)**  
Activa la entrada de una cadena mediante eventos, el cual se le asigna un tamaño máximo válido y la cadena generada se guardara en el contenido de la cadena objetivo, la variable final indica si se bloqueara la lectura para caracteres especiales como el salto de línea '\n' y tabulaciones '\t'.
- **bool input\_off(std::string\* input\_objetivo)**  
Desactiva la entrada de la cadena mediante eventos solo si se envía la cadena objetivo para poder asegurar la existencia del mismo.
- **bool mouse\_on(bool event\_type=false)**  
Activa la obtención de datos del ratón que puede ser mediante eventos u obtención directa de los datos, el parámetro enviado define la forma de obtención, por defecto es sin eventos por la gran cantidad de eventos generados por el ratón.
- **bool mouse\_off()**  
Anula el registro del ratón, y deja de controlar sus eventos generados si fue activada esta forma de obtención.
- **bool set\_mouse\_xy(int new\_mouse\_x, int new\_mouse\_y)**  
Mueve la posición del ratón a la posición (x,y) en la pantalla registrada, retorna verdadero en caso de éxito.
- **int get\_mouse\_x()**  
Retorna la posición actual del ratón en el eje x en la pantalla registrada.
- **int get\_mouse\_y()**  
Retorna la posición actual del ratón en el eje y en la pantalla registrada.
- **bool set\_mouse\_z(int new\_mouse\_z)**  
Cambia el valor del eje de la rueda del ratón, retorna verdadero en caso de éxito.
- **int get\_mouse\_z()**  
Retorna la posición actual del eje de la rueda del ratón.
- **bool& left\_click()**  
Retorna el estado del botón izquierdo del ratón, el valor puede ser modificado.
- **bool& right\_click()**  
Retorna el estado del botón derecho del ratón, el valor puede ser modificado.
- **bool& middle\_click()**  
Retorna el estado del botón central del ratón, el valor puede ser modificado.
- **bool get\_timer\_event()**  
Retorna el estado del temporizador, verdadero si el evento generado le pertenece al ratón.
- **bool& get\_display\_status()**  
Retorna el estado de la pantalla registrada, verdadero si se ha generado el evento de salida de la pantalla, el valor puede ser modificado.
- **bool& get\_textlog\_status()**  
Retorna el estado de la ventana de registro, verdadero si se ha generado el evento de salida de la ventana de registro, el valor puede ser modificado.
- **bool get\_event()**  
Consigue los eventos registrados añadiéndolos en la cola de eventos, retorna verdadero si se llega a obtener un evento en el tiempo de espera.
- **int get\_keycode()**  
Obtiene el código de tecla de un evento de teclado al presionar la tecla, es necesario tener una pantalla registrada para obtener los eventos del teclado, si se cierra la pantalla retorna -1.

#### Operadores:

- **bool& operator [] (std::string key\_name)**  
Retorna el estado de una llave con el nombre pasado por parámetro, si se encuentra en la estructura de llaves asignada se retorna por referencia su valor para que pueda ser modificado, retorna un comodín si no se encuentra.
- **operator ALLEGRO\_EVENT\_QUEUE\* ()**  
Retorna un puntero **ALLEGRO\_EVENT\_QUEUE** para que pueda ser usado con las funciones restantes de Allegro.

#### Destruyores:

- **~ Input()**  
Anula el registro de la pantalla, ventana de registro, temporizador, teclado, ratón, destruyendo la cola de eventos y desinstala el sistema del manejo del ratón y del teclado.

## 2.7 Color (LexRisLogic/Allegro5/Color.h)

### 2.7.1 Estructura: Color

Estructura básica de un color.

#### Constructores:

- **Color(unsigned char new\_red=0, unsigned char new\_green=0, unsigned char new\_blue=0, unsigned char new\_alpha=255)**  
Crea un color pasando su código en RGB y el nivel de opacidad, por defecto es un color negro completamente opaco.
- **Color(ALLEGRO\_COLOR color)**  
Crea un color a partir de la estructura **ALLEGRO\_COLOR** de Allegro.

#### Variables:

- **unsigned char red=0**  
Representa a la cantidad de color Rojo, valor entre 0 a 255.
- **unsigned char green=0**  
Representa a la cantidad de color Verde, valor entre 0 a 255.
- **unsigned char blue=0**  
Representa a la cantidad de color Azul, valor entre 0 a 255.
- **unsigned char alpha=255**  
Representa a la cantidad de opacidad del color, valor entre 0 a 255.

#### Operadores:

- **Color operator ! ()**  
Retorna el color opuesto en síntesis aditiva, pero con la misma opacidad.
- **Color operator = (ALLEGRO\_COLOR another\_color)**  
Copia el color de un **ALLEGRO\_COLOR** a esta estructura.
- **bool operator == (Color another\_color)**  
Compara la estructura con otra retornando verdadero si tienen el mismo color e igual opacidad.
- **bool operator != (Color another\_color)**  
Compara la estructura con otra retornando verdadero si tienen diferente color o diferente opacidad.
- **operator ALLEGRO\_COLOR ()**  
Retorna una estructura **ALLEGRO\_COLOR** para que pueda ser utilizado con las funciones restantes de Allegro.

## 2.8 Primitivos (**LexRisLogic/Allegro5/Primitives.h**)

### 2.8.1 Función: Iniciar Complemento Primitivos.

**bool** primitives\_addon()

Inicia el complemento de figuras primitivas para su uso completo, retorna verdadero en caso de éxito.

### 2.8.2 Función: Detener Complemento Primitivos.

**void** uninstall\_primitives()

Detiene el complemento de primitivos iniciado anteriormente, por defecto siempre se llama cuando termina la ejecución del programa.

### 2.8.3 Clase: **Pixel**

Primitivo que consiste en un pixel de la pantalla.

**Constructores:**

- **Pixel()**  
Inicia los valores del pixel en la posición (0,0).
- **Pixel(Type\_pos pos\_x, Type\_pos pos\_y)**  
Inicia los valores del pixel en la posición (pos\_x, pos\_y).

**Funciones:**

- **void set\_pos(Type\_pos new\_pos\_x, Type\_pos new\_pos\_y)**  
Cambia la posición del Pixel a (new\_pos\_x, new\_pos\_y).
- **void set\_pos\_x(Type\_pos new\_pos\_x)**  
Cambia la posición del pixel en el eje x.
- **Type\_pos get\_pos\_x()**  
Retorna la posición actual del pixel en el eje x.
- **void set\_pos\_y(Type\_pos new\_pos\_y)**  
Cambia la posición del pixel en el eje y.
- **Type\_pos get\_pos\_y()**  
Retorna la posición actual del pixel en el eje y.
- **void set\_color(ALLEGRO\_COLOR new\_color)**  
Cambia el color del pixel.
- **ALLEGRO\_COLOR get\_color()**  
Retorna el color actual del pixel.
- **void draw()**  
Dibuja el pixel en su respectiva posición y su color tomando en cuenta las escalas globales.
- **void draw\_in\_another\_target()**  
Dibuja el pixel sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

#### 2.8.4 Clase: **Line**

Primitivo que consiste en un segmento de línea.

##### Constructores:

- **Line()**  
Inicia los valores del segmento de línea con (0,0) como posición inicial y (0,0) como posición final.
- **Line(Type\_pos pos\_x1, Type\_pos pos\_y1, Type\_pos pos\_x2, Type\_pos pos\_y2)**  
Inicia los valores del segmento de línea con (pos\_x1,pos\_y1) como posición inicial y (pos\_x2,pos\_y2) como posición final.

##### Funciones:

- **void set\_pos(Type\_pos new\_cam\_pos\_x, Type\_pos new\_cam\_pos\_y)**  
Cambia la posición que se toma por referencia para la cámara a (new\_cam\_pos\_x,new\_cam\_pos\_y).
- **void set\_pos\_x(Type\_pos new\_cam\_pos\_x)**  
Cambia la posición en el eje x que se toma por referencia para la cámara.
- **Type\_pos get\_pos\_x()**  
Retorna la posición tomada por referencia para la cámara en el eje x.
- **void set\_pos\_y(Type\_pos new\_cam\_pos\_y)**  
Cambia la posición en el eje y que se toma por referencia para la cámara.
- **Type\_pos get\_pos\_y()**  
Retorna la posición tomada por referencia para la cámara en el eje y.
- **void set\_points(Type\_pos new\_pos\_x1, Type\_pos new\_pos\_y1, Type\_pos new\_pos\_x2, Type\_pos new\_pos\_y2)**  
Cambia la posición del punto inicial y final del segmento.
- **void set\_pos\_1(Type\_pos new\_pos\_x1, Type\_pos new\_pos\_y1)**  
Cambia la posición del punto inicial del segmento.
- **Type\_pos get\_pos\_x1()**  
Retorna la posición del eje x del punto inicial.
- **Type\_pos get\_pos\_y1()**  
Retorna la posición del eje y del punto inicial.
- **void set\_pos\_2(Type\_pos new\_pos\_x2, Type\_pos new\_pos\_y2)**  
Cambia la posición del punto final del segmento.
- **Type\_pos get\_pos\_x2()**  
Retorna la posición del eje x del punto final.
- **Type\_pos get\_pos\_y2()**  
Retorna la posición del eje y del punto final.
- **void set\_color(ALLEGRO\_COLOR new\_color)**  
Cambia el color del segmento.
- **ALLEGRO\_COLOR get\_color()**  
Retorna el color actual del segmento.
- **void set\_thickness(float new\_thickness)**  
Cambia el grosor del segmento.
- **float get\_thickness()**  
Retorna el grosor actual del segmento.
- **void draw()**  
Dibuja el segmento tomando en cuenta la posición de la cámara como referencia, su posición real asignada y las escalas globales.
- **void draw\_in\_another\_target()**  
Dibuja el segmento sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.



### 2.8.5 Clase Base: **Primitive**

Clase Base que contiene la estructura base de una figura primitiva.

**Funciones:**

- **void set\_thickness(float new\_thickness)**  
Cambia el grosor de la primitiva.
- **float get\_thickness()**  
Retorna el grosor actual de la primitiva.
- **void set\_color(ALLEGRO\_COLOR new\_color)**  
Cambia el color de la primitiva.
- **ALLEGRO\_COLOR get\_color()**  
Retorna el color actual de la primitiva.
- **void set\_filled\_status(bool is\_filled)**  
Cambia la opción de relleno de la primitiva.
- **bool get\_filled\_status()**  
Retorna el estado de la opción de relleno de la primitiva.

### 2.8.6 Clase Base: **Figure**

Clase Base que hereda de la clase base **Primitive** aumentando en la estructura base la posición de una figura común.

**Funciones:**

- **void set\_pos(Type\_pos new\_pos\_x, Type\_pos new\_pos\_y)**  
Cambia la posición de la primitiva a (new\_pos\_x, new\_pos\_y).
- **void set\_pos\_x(Type\_pos new\_pos\_x)**  
Cambia la posición de la primitiva en el eje x.
- **Type\_pos get\_pos\_x()**  
Retorna la posición actual de la primitiva en el eje x.
- **void set\_pos\_y(Type\_pos new\_pos\_y)**  
Cambia la posición de la primitiva en el eje y.
- **Type\_pos get\_pos\_y()**  
Retorna la posición actual de la primitiva en el eje y.

### 2.8.7 Clase: **Circle**

Clase que hereda de la clase base **Figure** representando a la figura círculo.

**Constructores:**

- **Circle()**  
Inicia los valores del círculo en la posición (0,0) con radio 1.
- **Circle(Type\_pos pos\_x, Type\_pos pos\_y, float radius)**  
Inicia los valores del círculo en la posición (pos\_x, pos\_y) con el radio pasado por parámetro.

**Funciones:**

- **void set\_radius(float new\_radius)**  
Cambia el radio del círculo.
- **float get\_radius()**  
Retorna el radio actual del círculo.
- **void draw()**  
Dibuja el círculo tomando en cuenta su posición y las escalas globales para el tamaño real del radio.
- **void draw\_in\_another\_target()**  
Dibuja el círculo sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

### 2.8.8 Clase: **Ellipse**

Clase que hereda de la clase base **Figure** representando a la figura elipse.

**Constructores:**

- **Ellipse()**  
Inicia los valores de la elipse en la posición (0,0) con radio 1 en el eje x e y.
- **Ellipse(Type\_pos pos\_x, Type\_pos pos\_y, float radius\_x, float radius\_y)**  
Inicia los valores de la elipse en la posición (pos\_x, pos\_y) con los radios del eje x e y pasados por parámetros.

**Funciones:**

- **void set\_radius\_x(float new\_radius\_x)**  
Cambia el radio del eje x de la elipse.
- **float get\_radius\_x()**  
Retorna el radio actual del eje x de la elipse.
- **void set\_radius\_y(float new\_radius\_y)**  
Cambia el radio del eje y de la elipse.
- **float get\_radius\_y()**  
Retorna el radio actual del eje y de la elipse.
- **void draw()**  
Dibuja la elipse tomando en cuenta su posición y las escalas globales para el tamaño real de sus radios.
- **void draw\_in\_another\_target()**  
Dibuja la elipse sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

### 2.8.9 Clase: **Rectangle**

Clase que hereda de la clase base **Figure** representando a la figura rectángulo.

#### Constructores:

- **Rectangle()**  
Inicia los valores del rectángulo en la posición (0,0) con tamaño 0 en el eje x e y.
- **Rectangle(Type\_pos pos\_x, Type\_pos pos\_y, float size\_x, float size\_y)**  
Inicia los valores del rectángulo en la posición (pos\_x, pos\_y) con los tamaños del eje x e y pasados por parámetros.

#### Funciones:

- **void set\_size\_x(float new\_size\_x)**  
Cambia el tamaño del eje x del rectángulo.
- **float get\_size\_x()**  
Retorna el tamaño actual del eje x del rectángulo.
- **void set\_size\_y(float new\_size\_y)**  
Cambia el tamaño del eje y del rectángulo.
- **float get\_size\_y()**  
Retorna el tamaño actual del eje y del rectángulo.
- **void draw()**  
Dibuja el rectángulo tomando en cuenta su posición y las escalas globales para el tamaño real en cada eje.
- **void draw\_in\_another\_target()**  
Dibuja el rectángulo sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

### 2.8.10 Clase: **Triangle**

Clase que hereda de la clase base **Primitive** representando a la primitiva triángulo.

#### Constructores:

- **Triangle()**  
Inicia los valores del triángulo con (0,0) como posición de todos los vértices.
- **Triangle(Type\_pos pos\_x1, Type\_pos pos\_y1, Type\_pos pos\_x2, Type\_pos pos\_y2, Type\_pos pos\_x3, Type\_pos pos\_y3)**  
Inicia los valores del triángulo pasando por parámetro la posición de todos los vértice.

#### Funciones:

- **void set\_pos(Type\_pos new\_cam\_pos\_x, Type\_pos new\_cam\_pos\_y)**  
Cambia la posición que se toma por referencia para la cámara a (new\_cam\_pos\_x, new\_cam\_pos\_y).
- **void set\_pos\_x(Type\_pos new\_cam\_pos\_x)**  
Cambia la posición en el eje x que se toma por referencia para la cámara.
- **Type\_pos get\_pos\_x()**  
Retorna la posición tomada por referencia para la cámara en el eje x.
- **void set\_pos\_y(Type\_pos new\_cam\_pos\_y)**  
Cambia la posición en el eje y que se toma por referencia para la cámara.
- **Type\_pos get\_pos\_y()**  
Retorna la posición tomada por referencia para la cámara en el eje y.

- **void set\_points(Type\_pos new\_pos\_x1, Type\_pos new\_pos\_y1, Type\_pos new\_pos\_x2, Type\_pos new\_pos\_y2, Type\_pos new\_pos\_x3, Type\_pos new\_pos\_y3)**  
Cambia la posición de todos los vértices del triángulo.
- **void set\_pos\_1(Type\_pos new\_pos\_x1, Type\_pos new\_pos\_y1)**  
Cambia la posición del primer vértice del triángulo.
- **Type\_pos get\_pos\_x1()**  
Retorna la posición del eje x del primer vértice del triángulo.
- **Type\_pos get\_pos\_y1()**  
Retorna la posición del eje y del primer vértice del triángulo.
- **void set\_pos\_2(Type\_pos new\_pos\_x2, Type\_pos new\_pos\_y2)**  
Cambia la posición del segundo vértice del triángulo.
- **Type\_pos get\_pos\_x2()**  
Retorna la posición del eje x del segundo vértice del triángulo.
- **Type\_pos get\_pos\_y2()**  
Retorna la posición del eje y del segundo vértice del triángulo.
- **void set\_pos\_3(Type\_pos new\_pos\_x3, Type\_pos new\_pos\_y3)**  
Cambia la posición del tercer vértice del triángulo.
- **Type\_pos get\_pos\_x3()**  
Retorna la posición del eje x del tercer vértice del triángulo.
- **Type\_pos get\_pos\_y3()**  
Retorna la posición del eje y del tercer vértice del triángulo.
- **void draw()**  
Dibuja el triángulo tomando en cuenta la posición de la cámara como referencia, los vértices asignados y las escalas globales.
- **void draw\_in\_another\_target()**  
Dibuja el triángulo sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

#### 2.8.11 Clase: **Function**

Clase que representa a una función en dimensión 2 del tipo  $y = f(x)$  para visualizarse en pantalla.

**Constructores:**

- **Function()**  
Inicia el objeto sin asignarle una función para visualizar.
- **Function(Type\_pos (\*Function\_fx)(Type\_pos))**  
Asigna la función enviada por parámetro como la función en dimensión 2  $y = f(x)$  para visualizarse en pantalla, por defecto solo se dibuja una recta de  $(0, f(0))$  a  $(1, f(1))$ .

**Funciones:**

- **void set\_pos(Type\_pos new\_cam\_pos\_x, Type\_pos new\_cam\_pos\_y)**  
Cambia la posición que se toma por referencia para la cámara a  $(new\_cam\_pos\_x, new\_cam\_pos\_y)$ .
- **void set\_pos\_x(Type\_pos new\_cam\_pos\_x)**  
Cambia la posición en el eje x que se toma por referencia para la cámara.
- **Type\_pos get\_pos\_x()**  
Retorna la posición tomada por referencia para la cámara en el eje x.
- **void set\_pos\_y(Type\_pos new\_cam\_pos\_y)**  
Cambia la posición en el eje y que se toma por referencia para la cámara.

- **Type\_pos** `get_pos_y()`  
Retorna la posición tomada por referencia para la cámara en el eje y.
- **void** `set_thickness(float new_thickness)`  
Cambia el grosor de las rectas generadas por la función.
- **float** `get_thickness()`  
Retorna el grosor de las rectas generadas por la función.
- **void** `set_color(ALLEGRO_COLOR new_color)`  
Cambia el color de las rectas generadas por la función.
- **ALLEGRO\_COLOR** `get_color()`  
Retorna el color actual de las rectas generadas por la función.
- **bool** `set_step(float new_step)`  
Cambia el paso en el eje x para dibujar las rectas que genere la función, retorna verdadero si el paso es válido.
- **float** `get_step()`  
Retorna el paso en el eje x que se usa para dibujar la función.
- **void** `set_initial_x(Type_pos new_init)`  
Cambia la posición inicial en el eje x desde la cual se dibujara la función.
- **Type\_pos** `get_initial_x()`  
Retorna la posición inicial que se toma en cuenta para dibujar la función.
- **void** `set_final_x(Type_pos new_final)`  
Cambia la posición final en el eje x hasta donde se dibujara la función.
- **Type\_pos** `get_final_x()`  
Retorna la posición final que se toma en cuenta para dibujar la función.
- **void** `set_function(Type_pos (*Function_fx)(Type_pos))`  
Cambia la función en dimensión 2  $y = f(x)$  para visualizarse en pantalla.
- **void** `draw()`  
Dibuja la función de acuerdo al paso, desde la posición inicial hasta la posición final con todas las configuraciones hechas tomando en cuenta la posición de la cámara como referencia y las escalas globales.
- **void** `draw_in_another_target()`  
Dibuja la función sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

## 2.9 Mapa de Bits (**LexRisLogic/Allegro5/Bitmap.h**)

### 2.9.1 Función: Iniciar Complemento Imágenes.

**bool** image\_addon()

Inicia el complemento de imágenes para su uso completo, retorna verdadero en caso de éxito.

### 2.9.2 Función: Guardar Mapa de Bits.

**bool** save\_bitmap(**std::string** bitmap\_file\_name,**ALLEGRO\_BITMAP\*** bitmap)

Guarda un mapa de bits en la ruta enviada por parámetro, retorna verdadero en caso de éxito.

### 2.9.3 Clase Base: **BitmapBase**

Clase Base que contiene la estructura base de una imagen en pantalla.

**Funciones:**

- **void set\_pos(**Type\_pos** new\_pos\_x,**Type\_pos** new\_pos\_y)**  
Cambia la posición del mapa de bits a (new\_pos\_x,new\_pos\_y).
- **void set\_pos\_x(**Type\_pos** new\_pos\_x)**  
Cambia la posición del mapa de bits en el eje x.
- ****Type\_pos** get\_pos\_x()**  
Retorna la posición actual del mapa de bits en el eje x.
- **void set\_pos\_y(**Type\_pos** new\_pos\_y)**  
Cambia la posición del mapa de bits en el eje y.
- ****Type\_pos** get\_pos\_y()**  
Retorna la posición actual del mapa de bits en el eje y.
- **void set\_angle(**float** new\_angle)**  
Cambia el ángulo de giro del mapa de bits, el ángulo trabaja en radianes.
- ****float** get\_angle()**  
Retorna el ángulo de giro del mapa de bits.
- **void set\_flag(**int** new\_flag)**  
Cambia el modo en como se realizara la operación de dibujo del mapa de bits, el modo puede tomar el valor de 0 para una operación por defecto o combinaciones de los siguientes flags de Allegro:
  1. **ALLEGRO\_FLIP\_HORIZONTAL**  
Dibuja el mapa de bits invirtiendo el eje y.
  2. **ALLEGRO\_FLIP\_VERTICAL**  
Dibuja el mapa de bits invirtiendo el eje x.
- **void set\_scale\_x(**float** new\_scale\_x)**  
Cambia la escala del mapa de bits en el eje x.
- ****float** get\_scale\_x()**  
Retorna la escala del mapa de bits en el eje x.
- **void set\_scale\_y(**float** new\_scale\_y)**  
Cambia la escala del mapa de bits en el eje y.
- ****float** get\_scale\_y()**  
Retorna la escala del mapa de bits en el eje y.
- **void set\_centering\_option(**bool** new\_centering\_option\_x,**bool** new\_centering\_option\_y)**  
Cambia la opción de centrado para la operación de dibujo, si la opción es verdadera, se tomara la posición en el eje indicado como central y no como inicial.

- **bool** `get_centering_option_x()`  
Retorna el estado de la opción de centrado en el eje x.
- **bool** `get_centering_option_y()`  
Retorna el estado de la opción de centrado en el eje y.

#### 2.9.4 Clase: **Bitmap**

Clase que hereda de la clase base **BitmapBase** representando a un mapa de bits simple.

##### Funciones:

- **float** `get_size_x()`  
Retorna el ancho del mapa de bits.
- **float** `get_size_y()`  
Retorna el alto del mapa de bits.
- **bool** `create(int size_x, int size_y)`  
Crea un mapa de bits con el ancho y alto pasados por parámetros, retorna verdadero en caso de éxito.
- **bool** `destroy()`  
Elimina el mapa de bits, liberando la memoria utilizada, retorna verdadero en caso de éxito.
- **void** `set_target()`  
Asigna al mapa de bits como objetivo de las operaciones de dibujo.
- **bool** `lock()`  
Asegura el mapa de bits en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void** `unlock()`  
Quita el bloqueo del mapa de bits liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO\_COLOR** `get_pixel_color(Type_pos pos_x, Type_pos pos_y)`  
Retorna el color actual que tiene un pixel en el mapa de bits pasando por parámetros su posición.
- **void** `draw()`  
Dibuja el mapa de bits tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void** `draw_in_another_target()`  
Dibuja el mapa de bits sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

##### Operadores:

- **operator** **ALLEGRO\_BITMAP\*** `()`  
Retorna un puntero **ALLEGRO\_BITMAP** que representa al mapa de bits para que pueda ser usado con las funciones restantes de Allegro.

##### Destruyores:

- **~** **Bitmap**`()`  
Libera la memoria utilizada por la clase y elimina el mapa de bits junto con todos los mapas de bits hijos generados.

### 2.9.5 Clase: **SubBitmap**

Clase que hereda de la clase base **BitmapBase** representando a una porción de otro mapa de bits.

**Funciones:**

- **void set\_parent\_bitmap(ALLEGRO\_BITMAP\* new\_parent\_bitmap)**  
Asigna el mapa de bits padre del cual se tomará porciones de él.
- **ALLEGRO\_BITMAP\* get\_parent\_bitmap()**  
Retorna el mapa de bits padre asignado.
- **void set\_sub\_x(int new\_sub\_x)**  
Asigna la posición inicial en x a seleccionarse del mapa de bits padre.
- **int get\_sub\_x()**  
Retorna la posición inicial de selección del mapa de bits en el eje x.
- **void set\_sub\_y(int new\_sub\_y)**  
Asigna la posición inicial en y a seleccionarse del mapa de bits padre.
- **int get\_sub\_y()**  
Retorna la posición inicial de selección del mapa de bits en el eje y.
- **void set\_size\_x(float new\_size\_x)**  
Asigna el ancho a seleccionarse del mapa de bits.
- **float get\_size\_x()**  
Retorna el ancho seleccionado del mapa de bits.
- **void set\_size\_y(float new\_size\_y)**  
Asigna el alto a seleccionarse del mapa de bits.
- **float get\_size\_y()**  
Retorna el alto seleccionado del mapa de bits.
- **bool create()**  
Crea un mapa de bits a partir de una porción del mapa de bits padre tomando en cuenta la configuración de las posiciones de selección y el ancho y alto a tomarse del mapa de bits, retorna verdadero en caso de éxito.
- **bool destroy()**  
Elimina la porción del mapa de bits, liberando la memoria utilizada, retorna verdadero en caso de éxito.
- **void set\_target()**  
Asigna la porción del mapa de bits como objetivo de las operaciones de dibujo.
- **bool lock()**  
Asegura la porción del mapa de bits en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void unlock()**  
Quita el bloqueo de la porción del mapa de bits liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO\_COLOR get\_pixel\_color(Type\_pos pos\_x, Type\_pos pos\_y)**  
Retorna el color actual que tiene un pixel en la porción del mapa de bits pasando por parámetros su posición.
- **void draw()**  
Dibuja la porción del mapa de bits tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void draw\_in\_another\_target()**  
Dibuja la porción del mapa de bits sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.



#### Operadores:

- **operator ALLEGRO\_BITMAP\*** ()  
Retorna un puntero **ALLEGRO\_BITMAP** que representa a la porción del mapa de bits padre para que pueda ser usado con las funciones restantes de Allegro.

#### Destruyores:

- **~ SubBitmap()**  
Libera la memoria utilizada por la clase y elimina solo el mapa de bits hijo generado.

### 2.9.6 Clase: **Image**

Clase que hereda de la clase base **BitmapBase** representando una imagen como un mapa de bits.

#### Funciones:

- **void set\_path(std::string new\_image\_path)**  
Se asigna la ruta de la imagen a cargarse por la clase.
- **std::string get\_path()**  
Retorna la ruta de la imagen a cargarse por la clase.
- **float get\_size\_x()**  
Retorna el ancho de la imagen.
- **float get\_size\_y()**  
Retorna el alto de la imagen.
- **bool load()**  
Carga el mapa de bits de la ruta indicada, retorna verdadero en caso de éxito.
- **bool save()**  
Guarda el mapa de bits en la ruta actual, retorna verdadero en caso de éxito, si la ruta no ha sido cambiada se sobrescribirá el archivo.
- **bool destroy()**  
Elimina la imagen de la memoria, retorna verdadero en caso de éxito.
- **void set\_target()**  
Asigna a la imagen como objetivo de las operaciones de dibujo.
- **bool lock()**  
Asegura la imagen en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void unlock()**  
Quita el bloqueo de la imagen liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO\_COLOR get\_pixel\_color(Type\_pos pos\_x, Type\_pos pos\_y)**  
Retorna el color actual que tiene un pixel en la imagen pasando por parámetros su posición.
- **void draw()**  
Dibuja la imagen tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void draw\_in\_another\_target()**  
Dibuja la imagen sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

**Operadores:**

- **operator ALLEGRO\_BITMAP\*** ()  
Retorna un puntero **ALLEGRO\_BITMAP** que representa al mapa de bits de la imagen para que pueda ser usado con las funciones restantes de Allegro.

**Destruyores:**

- **~ Image()**  
Libera la memoria utilizada por la clase y elimina el mapa de bits cargado junto con todos los mapas de bits hijos generados.

## 2.10 Texto (LexRisLogic/Allegro5/Text.h)

### 2.10.1 Función: Iniciar Complemento Texto.

**bool** text\_addon()

Inicia el complemento de texto y fuentes para su uso completo, retorna verdadero en caso de éxito.

### 2.10.2 Clase: Font

Clase que provee una interfaz para crear instancias de fuentes para el dibujado del texto en pantalla.

**Funciones:**

- **void** set\_path(**std::string** new\_font\_path)  
Se asigna la ruta de la fuente a cargarse por la clase.
- **std::string** get\_path()  
Retorna la ruta de la fuente a cargarse por la clase.
- **bool** set\_size(**float** new\_size)  
Asigna el tamaño de fuente, retorna verdadero si el tamaño es válido y aún no se haya cargado la fuente.
- **float** get\_size()  
Retorna el tamaño de fuente.
- **bool** load\_ttf\_font()  
Carga la fuente de un archivo *ttf* tomando en cuenta la escala global del texto, retorna verdadero en caso de éxito.
- **bool** load\_ttf\_font\_for\_another\_target()  
Carga la fuente de un archivo *ttf* sin tomar en cuenta la escala global del texto, retorna verdadero en caso de éxito.
- **bool** destroy()  
Destruye la fuente cargada actualmente, retorna verdadero en caso de éxito.

**Operadores:**

- **operator ALLEGRO\_FONT\*** ()  
Retorna un puntero **ALLEGRO\_FONT** para que pueda ser usado con las funciones restantes de Allegro.

**Destructores:**

- **~Font()**  
Libera la memoria utilizada por la clase y elimina la fuente cargada.

### 2.10.3 Clase: Text

Clase que dibuja una cadena de caracteres en pantalla.

**Funciones:**

- **void** set\_pos(**Type\_pos** new\_pos\_x,**Type\_pos** new\_pos\_y)  
Cambia la posición del texto a (new\_pos\_x,new\_pos.y).
- **void** set\_pos\_x(**Type\_pos** new\_pos\_x)  
Cambia la posición del texto en el eje x.
- **Type\_pos** get\_pos\_x()  
Retorna la posición actual del texto en el eje x.
- **void** set\_pos\_y(**Type\_pos** new\_pos\_y)  
Cambia la posición del texto en el eje y.

- **Type\_pos** `get_pos_y()`  
Retorna la posición actual del texto en el eje y.
- **void** `set_flag(int new_flag)`  
Cambia el modo en como se realizara la operación de dibujo del texto, puede tomar como valor los siguientes flags de Allegro:
  1. **ALLEGRO\_ALIGN\_LEFT**  
Dibuja el texto tomando la posición (x,y) como punto de partida, es decir el texto se extiende a la derecha (Igual que usar 0 como parámetro).
  2. **ALLEGRO\_ALIGN\_CENTRE** o **ALLEGRO\_ALIGN\_CENTER**  
Dibuja el texto tomando la posición (x,y) como punto central del texto.
  3. **ALLEGRO\_ALIGN\_RIGHT**  
Dibuja el texto tomando la posición (x,y) como punto final del dibujo, es decir el texto se extiende hacia la izquierda.
- **void** `set_color(ALLEGRO_COLOR new_color)`  
Cambia el color del texto.
- **ALLEGRO\_COLOR** `get_color()`  
Retorna el color actual del texto.
- **void** `set_font(Font* new_font)`  
Cambia la fuente que se utilizará para la operación de dibujo del texto.
- **Font\*** `get_font()`  
Retorna la fuente actual utilizada para la operación de dibujo del texto.
- **void** `draw()`  
Dibuja el texto con toda la configuración dada.

#### Operadores:

- **const char\*** `operator = (const char* new_text)`  
Operador para asignar una cadena constante como texto a dibujar.
- **std::string** `operator = (std::string new_text)`  
Operador para asignar una cadena como texto a dibujar.
- **operator const char\*** `()`  
Retorna una cadena constante del texto que se dibuja en pantalla.
- **operator std::string** `()`  
Retorna una cadena del texto que se dibuja en pantalla.

## 2.11 Mezclador (LexRisLogic/Allegro5/Mixer.h)

### 2.11.1 Función: Iniciar Complemento Audio.

**bool** audio\_addon()

Inicia el complemento de audio para su uso completo, retorna verdadero en caso de éxito.

### 2.11.2 Función: Detener Complemento Audio.

**void** uninstall\_audio()

Detiene el complemento de audio iniciado anteriormente, por defecto siempre se llama cuando termina la ejecución del programa.

### 2.11.3 Función: Restaurar Mezclador por defecto.

**bool** restore\_default\_mixer()

Restaura el mezclador y el dispositivo de salida de Allegro como mezclador por defecto para la reproducción de sonidos, retorna verdadero en caso de éxito; tener en cuenta que si se elimina un mezclador que se asigno como mezclador por defecto y aun se desea seguir usando la reproducción de sonidos, es necesario tener otro mezclador por defecto restaurando el mezclador de Allegro o creando un nuevo mezclador.

### 2.11.4 Clase: Mixer

Clase que representa al motor de sonido que usa Allegro para reproducir los sonidos de un archivo de audio o video conforme a la configuración de este.

**Funciones:**

- **bool** set\_frequency(**unsigned int** new\_frequency)  
Asigna la frecuencia con la que reproducirá la tarjeta de sonido, por defecto es 44100 Hz, retorna verdadero si el motor aun no ha sido iniciado para poder aplicar el cambio.
- **unsigned int** get\_frequency()  
Retorna la frecuencia con la que reproducirá la tarjeta de sonido.
- **bool** set\_audio\_depth(**ALLEGRO\_AUDIO\_DEPTH** new\_frequency)  
Asigna la profundidad, tipo y signo del sonido que se tomará en cuenta, por defecto usa 32 bits con precisión punto flotante, retorna verdadero si el motor aun no ha sido iniciado para poder aplicar el cambio, los valores que puede tomar son:
  1. **ALLEGRO\_AUDIO\_DEPTH\_INT8**
  2. **ALLEGRO\_AUDIO\_DEPTH\_INT16**
  3. **ALLEGRO\_AUDIO\_DEPTH\_INT24**
  4. **ALLEGRO\_AUDIO\_DEPTH\_FLOAT32**
  5. **ALLEGRO\_AUDIO\_DEPTH\_UNSIGNED**
  6. **ALLEGRO\_AUDIO\_DEPTH\_UINT8**
  7. **ALLEGRO\_AUDIO\_DEPTH\_UINT16**
  8. **ALLEGRO\_AUDIO\_DEPTH\_UINT24**
- **ALLEGRO\_AUDIO\_DEPTH** get\_audio\_depth()  
Retorna la macro de Allegro actualmente utilizada para definir la profundidad, tipo y signo de los sonidos.

- **bool** `set_channel_configuration(ALLEGRO_CHANNEL_CONF new_channel_conf)`  
Asigna la configuración de los altavoces, por defecto usa dos canales para sonido estéreo, retorna verdadero si el motor aun no ha sido iniciado para poder aplicar el cambio, los valores que puede tomar son:
  1. **ALLEGRO\_CHANNEL\_CONF\_1**
  2. **ALLEGRO\_CHANNEL\_CONF\_2**
  3. **ALLEGRO\_CHANNEL\_CONF\_3**
  4. **ALLEGRO\_CHANNEL\_CONF\_4**
  5. **ALLEGRO\_CHANNEL\_CONF\_5\_1**
  6. **ALLEGRO\_CHANNEL\_CONF\_6\_1**
  7. **ALLEGRO\_CHANNEL\_CONF\_7\_1**
- **ALLEGRO\_CHANNEL\_CONF** `get_channel_configuration()`  
Retorna la macro de Allegro actualmente usada para definir la configuración de los altavoces.
- **bool** `create()`  
Crea un mezclador con la configuración dada y una voz que representa un dispositivo de salida en el sistema con la misma configuración, excepto la profundidad que usa 16 bits con signo y precisión entera, retorna verdadero si se logro crear todo con éxito.
- **bool** `destroy()`  
Destruye el mezclador y la voz liberando la memoria utilizada.
- **bool** `set_quality(ALLEGRO_MIXER_QUALITY quality)`  
Asigna la calidad del mezclador de sonidos, retorna verdadero en caso de éxito, los valores que puede tomar son:
  1. **ALLEGRO\_MIXER\_QUALITY\_POINT**
  2. **ALLEGRO\_MIXER\_QUALITY\_LINEAR**
  3. **ALLEGRO\_MIXER\_QUALITY\_CUBIC**
- **ALLEGRO\_MIXER\_QUALITY** `get_quality()`  
Retorna la calidad del mezclador de sonidos, por defecto el valor de la calidad es **ALLEGRO\_MIXER\_QUALITY\_LINEAR**.
- **bool** `set_volume(float new_volume)`  
Asigna el volumen general del mezclador de sonidos donde 1.0 es el 100% pudiéndose duplicar el volumen al pasar 2.0 o disminuirlo con valores menores a 1.0, retorna verdadero en caso de éxito.
- **float** `get_volume()`  
Retorna el volumen general del mezclador de sonidos.
- **bool** `set_default_mixer()`  
Asigna al mezclador como mezclador por defecto para que las configuraciones realizadas se apliquen sobre la reproducción de sonido de archivos de audio y video.

#### Operadores:

- **operator** **ALLEGRO\_MIXER\*** `()`  
Retorna un puntero **ALLEGRO\_MIXER** para que pueda ser usado con las funciones restantes de Allegro.
- **operator** **ALLEGRO\_VOICE\*** `()`  
Retorna un puntero **ALLEGRO\_VOICE** para que pueda ser usado con las funciones restantes de Allegro.

#### Destructores:

- `~ Mixer()`  
Elimina el mezclador y la voz, liberando la memoria utilizada por toda la clase.

## 2.12 Audio (LexRisLogic/Allegro5/Audio.h)

### 2.12.1 Clase: `Audio`

Clase que representa a un archivo de audio. **Funciones:**

- `void set_path(std::string new_audio_path)`  
Se asigna la ruta del archivo de audio que va a ser reproducido.
- `std::string get_path()`  
Retorna la ruta del archivo de audio que va a ser reproducido.
- `bool set_speed(float new_speed)`  
Asigna la velocidad de reproducción del audio, retorna verdadero en éxito.
- `float get_speed()`  
Retorna la velocidad de reproducción actual del audio.
- `bool set_pan(float new_pan)`  
Cambia la dirección de la reproducción del audio, los valores que toma la variable `new_pan` están entre `[-1,1]` donde:
  - Reproducción normal es 0
  - Reproducción por parlante izquierdo es -1
  - Reproducción por parlante derecho es 1
- `float get_pan()`  
Retorna la dirección de la reproducción del audio.
- `void set_volume(float new_volume)`  
Cambia el volumen de la reproducción del audio a escala del volumen asignado actualmente en el mezclador por defecto.
- `float get_volume()`  
Retorna el volumen asignado a la reproducción del audio.
- `void set_mode(ALLEGRO_PLAYMODE playmode)`  
Cambia el modo de reproducción del audio, retorna verdadero en caso de éxito, puede tomar los siguientes flags de Allegro:
  1. `ALLEGRO_PLAYMODE_ONCE`  
Reproduce el audio una sola vez.
  2. `ALLEGRO_PLAYMODE_LOOP`  
Reproduce el audio una y otra vez hasta que la reproducción sea detenida.
  3. `ALLEGRO_PLAYMODE_BIDIR`  
Reproduce el audio normalmente, pero una vez que llegue al final, la reproducción será de manera inversa hasta llegar al principio y reproducirse normalmente, esto seguirá hasta que la reproducción sea detenida.
- `ALLEGRO_PLAYMODE get_playmode()`  
Retorna el modo de reproducción del audio, por defecto el modo de reproducción es una sola vez.
- `bool load()`  
Carga el archivo de audio y le aplica toda la configuración dada, la configuración puede ser modificada a pesar de haber cargado ya el archivo.

- **bool destroy()**  
Elimina de memoria la carga del archivo de audio y su instancia para reproducción.
- **unsigned int size()**  
Retorna la posición final de reproducción del archivo de audio.
- **float get\_time()**  
Retorna la duración del archivo de audio en segundos.
- **bool set\_audio\_position(unsigned int new\_position)**  
Cambia la posición de la reproducción del audio, retorna verdadero en caso de éxito.
- **unsigned int get\_audio\_position()**  
Retorna la posición actual de la reproducción del audio.
- **bool is\_playing()**  
Retorna verdadero si la reproducción esta activa.
- **void stop()**  
Detiene la reproducción.
- **void pause()**  
Pausa la reproducción.
- **void play()**  
Inicia o reanuda la reproducción.

#### Operadores:

- **operator ALLEGRO\_SAMPLE\* ()**  
Retorna un puntero **ALLEGRO\_SAMPLE** para que pueda ser usado con las funciones restantes de Allegro.
- **operator ALLEGRO\_SAMPLE\_INSTANCE\* ()**  
Retorna un puntero **ALLEGRO\_SAMPLE\_INSTANCE** para que pueda ser usado con las funciones restantes de Allegro.

#### Destruyores:

- **~ Audio()**  
Libera la memoria utilizada por la clase y elimina el archivo de audio cargado.



## 2.13 Video (LexRisLogic/Allegro5/Video.h)

### 2.13.1 Función: Iniciar Complemento Video.

**bool** video\_addon()

Inicia el complemento de video para su uso completo, retorna verdadero en caso de éxito.

### 2.13.2 Clase: Video

Clase que hereda de la clase base **BitmapBase** representando a un archivo de video.

**Funciones:**

- **void set\_path(std::string new\_video\_path)**  
Se asigna la ruta del archivo de video a cargarse por la clase.
- **std::string get\_path()**  
Retorna la ruta del archivo de video a cargarse por la clase.
- **float get\_size\_x()**  
Retorna el ancho del video.
- **float get\_size\_y()**  
Retorna el alto del video.
- **bool load()**  
Carga el video de la ruta indicada, retorna verdadero en caso de éxito.
- **bool destroy()**  
Elimina el video de la memoria, retorna verdadero en caso de éxito.
- **double get\_video\_position(ALLEGRO\_VIDEO\_POSITION\_TYPE reference\_position=ALLEGRO\_VIDEO\_POSITION\_ACTUAL)**  
Retorna la posición actual en segundos de la reproducción del video, el parámetro es para tomar la posición en referencia:
  1. **ALLEGRO\_VIDEO\_POSITION\_ACTUAL**  
A la posición en el sistema de Allegro.
  2. **ALLEGRO\_VIDEO\_POSITION\_VIDEO\_DECODE**  
A la posición a las imágenes del video.
  3. **ALLEGRO\_VIDEO\_POSITION\_AUDIO\_DECODE**  
A la posición del audio del video.
- **bool start()**  
Inicia la reproducción del video para poder obtener imágenes para dibujarlo en pantalla.
- **bool is\_playing()**  
Retorna verdadero si la reproducción esta activa.
- **void stop()**  
Detiene la reproducción.
- **void pause()**  
Pausa la reproducción.
- **void play()**  
Reanuda la reproducción.
- **void draw()**  
Dibuja las imágenes del video tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.

#### Operadores:

- **operator ALLEGRO\_VIDEO\*** ()  
Retorna un puntero **ALLEGRO\_VIDEO** para que pueda ser usado con las funciones restantes de Allegro.

#### Destructores:

- **~ Video()**  
Libera la memoria utilizada por la clase y elimina archivo de video cargado.

## 2.14 Iluminación (**LexRisLogic/Allegro5/Lighting.h**)

### 2.14.1 Clase: **Lighting**

Clase que representa a una iluminación básica usando un buffer que contiene intensidades de canal alfa en negro para crear zonas oscuras e iluminadas.

#### Constructores:

- **Lighting()**  
Obtiene las opciones de operación de dibujo como la operación de combinación, los valores de fuente y destino con el objetivo de retomar las verdaderas opciones de dibujo después de generar la iluminación.

#### Funciones:

- **void set\_pos(Type\_pos new\_pos\_x, Type\_pos new\_pos\_y)**  
Cambia la posición del buffer de iluminación a (new\_pos\_x, new\_pos\_y).
- **void set\_pos\_x(Type\_pos new\_pos\_x)**  
Cambia la posición del buffer de iluminación en el eje x.
- **Type\_pos get\_pos\_x()**  
Retorna la posición actual del buffer de iluminación en el eje x.
- **void set\_pos\_y(Type\_pos new\_pos\_y)**  
Cambia la posición del buffer de iluminación en el eje y.
- **Type\_pos get\_pos\_y()**  
Retorna la posición actual del buffer de iluminación en el eje y.
- **float get\_size\_x()**  
Retorna el tamaño actual en el eje x del buffer de iluminación.
- **float get\_size\_y()**  
Retorna el tamaño actual en el eje y del buffer de iluminación.
- **bool create(int size\_x, int size\_y)**  
Crea un mapa de bits con el ancho y alto pasados por parámetros para ser el buffer de iluminación, retorna verdadero en caso de éxito.
- **bool destroy()**  
Elimina el buffer de iluminación, liberando la memoria utilizada, retorna verdadero en caso de éxito.
- **void set\_target()**  
Asigna el buffer de iluminación como objetivo de las operaciones de dibujo, usar solo las funciones que brinda la clase para dibujar en el buffer con el objetivo de crear una correcta iluminación simple mediante figuras.
- **void clear(float intensity=0.0)**  
Limpia la buffer de iluminación con una intensidad de luz, 0 es totalmente opaco y 1 es totalmente iluminado.

- **template**<typename **T**>  
**void add**(**T**\* object\_shape, float intensity)

Función que recibe una clase **T** que debe tener los siguientes métodos implementados:

1. **ALLEGRO\_COLOR** **get\_color**() → Usado para obtener el color real del objeto y reponerlo terminada la llamada de la función.
2. **void set\_color**(**ALLEGRO\_COLOR** new\_color) → Usado para asignar la intensidad dada de iluminación mediante el uso de colores.
3. **void draw\_in\_another\_target**() → Usado para dibujar la al objeto en el buffer objetivo.

La operación de dibujo consiste en dibujar al objeto con otro color que represente intensidad de iluminación, puede mandar figuras primitivas o crear figuras más complejas.

- **float get\_pixel\_intensity**(**Type\_pos** pos\_x, **Type\_pos** pos\_y)  
Retorna la intensidad de luz que tiene un pixel en el buffer de iluminación pasando por parámetros su posición.
- **void draw**()  
Aplica el buffer de iluminación en la imagen objetivo tomando en cuenta toda las escalas globales para el tamaño real en cada eje.
- **void draw\_in\_another\_target**()  
Aplica el buffer de iluminación sin tomar en cuenta las escalas globales, usado para aplicar en otro objetivo.

#### Operadores:

- **operator ALLEGRO\_BITMAP\*** ()  
Retorna un puntero **ALLEGRO\_BITMAP** que es el buffer de iluminación para que pueda ser usado con las funciones restantes de Allegro.

#### Destructores:

- **~ Lighting**()  
Destruye el buffer de iluminación y libera la memoria utilizada.

## 2.15 Interfaz (LexRisLogic/Allegro5/Special/Interface.h)

### 2.15.1 Clase Base: Interface

Clase Base que contiene la estructura base de una interfaz para manejar entradas del usuario en pantalla, clase especial que depende de las clases de **LL\_AL5**.

**Constructores:**

- **Interface()**  
Constructor para los valores constantes usados por las clases que heredan de él.

**Funciones:**

- **void set\_pos(Type\_pos new\_pos\_x, Type\_pos new\_pos\_y)**  
Cambia la posición de la interfaz a (new\_pos\_x, new\_pos\_y).
- **void set\_pos\_x(Type\_pos new\_pos\_x)**  
Cambia la posición de la interfaz en el eje x.
- **Type\_pos get\_pos\_x()**  
Retorna la posición actual de la interfaz en el eje x.
- **void set\_pos\_y(Type\_pos new\_pos\_y)**  
Cambia la posición de la interfaz en el eje y.
- **Type\_pos get\_pos\_y()**  
Retorna la posición actual de la interfaz en el eje y.
- **float get\_size\_x()**  
Retorna el ancho de la interfaz.
- **float get\_size\_y()**  
Retorna el alto de la interfaz.
- **void set\_unclick\_line\_color(ALLEGRO\_COLOR new\_unclick\_line\_color)**  
Cambia el color del borde de la interfaz cuando no esta enfocada.
- **ALLEGRO\_COLOR get\_unclick\_line\_color()**  
Retorna el color del borde de la interfaz cuando no esta enfocada.
- **void set\_unclick\_fill\_color(ALLEGRO\_COLOR new\_unclick\_fill\_color)**  
Cambia el color del relleno de la interfaz cuando no esta enfocada.
- **ALLEGRO\_COLOR get\_unclick\_fill\_color()**  
Retorna el color del relleno de la interfaz cuando no esta enfocada.
- **void set\_click\_line\_color(ALLEGRO\_COLOR new\_unclick\_line\_color)**  
Cambia el color del borde de la interfaz cuando esta enfocada.
- **ALLEGRO\_COLOR get\_click\_line\_color()**  
Retorna el color del borde de la interfaz cuando esta enfocada.
- **void set\_click\_fill\_color(ALLEGRO\_COLOR new\_unclick\_fill\_color)**  
Cambia el color del relleno de la interfaz cuando esta enfocada.
- **ALLEGRO\_COLOR get\_click\_fill\_color()**  
Retorna el color del relleno de la interfaz cuando esta enfocada.
- **void set\_thickness(float new\_thickness)**  
Cambia el grosor del borde de la interfaz.
- **float get\_thickness()**  
Retorna el grosor del borde de la interfaz.
- **Font\* get\_font()**  
Retorna la fuente utilizada para el texto de la interfaz.
- **void set\_text\_color(ALLEGRO\_COLOR new\_text\_color)**  
Cambia el color del texto de la interfaz.
- **ALLEGRO\_COLOR get\_text\_color()**  
Retorna el color del texto de la interfaz.

### 2.15.2 Clase: **Button**

Clase que hereda de la clase base **Interface** representando a un botón, clase especial que depende de las clases de **LL\_AL5**.

**Constructores:**

- **Button(Input\* input\_pointer)**  
Indica cual va a ser la interfaz de entrada a usarse para obtener los eventos de selección sobre el botón.

**Funciones:**

- **void set\_button\_text(std::string new\_button\_text)**  
Cambia el texto del botón.
- **std::string get\_button\_text()**  
Retorna el texto del botón.
- **void set\_font(Font\* new\_font)**  
Cambia la fuente utilizada para dibujar el texto del botón.
- **void draw()**  
Dibuja el botón en la pantalla objetivo tomando en cuenta las escalas globales, a su vez lee las entradas por si el usuario genero el evento de selección del botón.
- **bool is\_clicked()**  
Retorna el estado del botón, verdadero si ha sido seleccionado, tener en cuenta que el evento solo será transmitido una sola vez tornándose falso en su lectura.
- **bool button\_on()**  
Activa la generación del evento de selección del botón.
- **bool button\_off()**  
Desactiva la generación del evento de selección del botón.

### 2.15.3 Clase: **TextBox**

Clase que hereda de la clase base **Interface** representando a una caja de texto, clase especial que depende de las clases de **LL\_AL5**.

**Constructores:**

- **TextBox(Input\* input\_pointer)**  
Indica cual va a ser la interfaz de entrada a usarse para obtener los eventos de selección sobre la caja de texto.

**Funciones:**

- **void set\_hide\_option(bool hide\_option)**  
Cambia la opción para ocultar el texto ingresado por el carácter '\*'.
- **bool get\_hide\_option()**  
Retorna verdadero si la opción de texto oculto esta activada.
- **void set\_text\_length(unsigned int new\_text\_length)**  
Cambia el máximo tamaño que puede recibirse de entrada del usuario.
- **unsigned int get\_text\_length()**  
Retorna el máximo tamaño que puede recibirse de entrada del usuario.
- **void set\_font(Font\* new\_font)**  
Cambia la fuente utilizada para dibujar el texto de la interfaz.
- **void clear()**  
Limpia la entrada vaciando la cadena de texto.
- **bool set\_value(std::string new\_value)**  
Cambia el valor actual en la caja de texto, retorna verdadero en caso de éxito.

- **std::string get\_value()**  
Retorna la cadena que contiene la caja de texto.
- **void draw()**  
Dibuja la caja de texto en la pantalla objetivo tomando en cuenta las escalas globales, a su vez lee las entradas por si el usuario genero el evento de selección de la caja de texto y también cuando es enfocada se activa la entrada de una cadena en la clase **Input** para obtener la entrada del usuario en la cadena de la caja de texto.
- **bool is\_clicked()**  
Retorna el estado de la caja de texto, verdadero si esta enfocada.
- **bool textbox\_on()**  
Activa la generación del evento de selección de la caja de texto, así como la opción de ingresado de datos por la clase **Input**.
- **bool textbox\_off()**  
Desactiva la generación de eventos y desactiva la entrada de la cadena mediante eventos de la clase **Input**.

#### **Destructores:**

- **~ TextBox()**  
Desactiva la entrada de la cadena mediante eventos de la clase **Input** por si el programador olvido de desactivar la entrada y libera la memoria utilizada.

## 2.16 Sprite (LexRisLogic/Allegro5/Special/Sprite.h)

### 2.16.1 Clase: Sprite

Clase que hereda de la clase base **BitmapBase** representando a un conjunto de imágenes, clase especial que depende de las clases de **LL\_AL5**.

**Funciones:**

- **float get\_size\_x()**  
Retorna el ancho del sprite actualmente seleccionado.
- **float get\_size\_y()**  
Retorna el alto del sprite actualmente seleccionado.
- **unsigned int get\_size()**  
Retorna el número de sprites que existen almacenados.
- **bool set\_size(unsigned int new\_size)**  
Cambia el número de sprites que se almacenaran, retorna verdadero si aún no se ha separado memoria para la creación de los sprites y si el tamaño es válido.
- **bool set\_selection(unsigned int new\_selection)**  
Cambia el sprite seleccionado para las operaciones de dibujo, obtención de su tamaño y usarlo como objetivo otras operaciones de dibujo, retorna verdadero si el cambio es válido.
- **unsigned int get\_selection()**  
Retorna el índice del sprite seleccionado, el índice toma valores en el intervalo de [0:size>.
- **bool create()**  
Separa la memoria para poder crear los mapas de bits necesarios para todos los sprites, creando también arreglos para obtener el tamaño de cada uno de estos, retorna verdadero en caso de éxito.
- **bool create\_selection(int size\_x,int size\_y)**  
Crea un mapa de bits vacío de tamaño (*size\_x,size\_y*) en la posición seleccionada, si ya existe un mapa de bits este será eliminado para crear el nuevo mapa de bits, retorna verdadero en caso de éxito.
- **bool destroy\_selection()**  
Elimina el mapa de bits en la posición actualmente seleccionada, retorna verdadero en caso de éxito.
- **bool create\_data\_from\_directory(std::string sprites\_path,std::string bitmap\_format)**  
Crea todos los mapas de bits cargando los sprites del disco los cuales deberán tener su ruta completa de la siguiente manera:  
*sprites\_path (index\_number+1) bitmap\_format*  
Por ejemplo se cargaran dos sprites que tienen la siguiente ruta, */home/user/bitmaps/sprite* y en formato *.bmp*, entonces los dos sprites deberán tener como ruta completa:  
*/home/user/bitmaps/sprite1.bmp*  
*/home/user/bitmaps/sprite2.bmp*
- **bool destroy\_data()**  
Elimina todos los mapas de bits creados o cargados, retorna verdadero en caso de éxito.
- **bool destroy()**  
Elimina y libera la memoria separada para la creación y utilización de los mapas de bits.
- **void set\_target()**  
Asigna al mapa de bits actualmente seleccionado como objetivo de las operaciones de dibujo.

- **bool lock()**  
Asegura el mapa de bits actualmente seleccionado en memoria para prepararlo contra múltiples accesos de lectura y escritura, retorna verdadero en caso de éxito.
- **void unlock()**  
Quita el bloqueo del mapa de bits actualmente seleccionado liberando la memoria utilizada para los múltiples accesos.
- **ALLEGRO\_COLOR get\_pixel\_color(Type\_pos pos\_x, Type\_pos pos\_y)**  
Retorna el color actual que tiene un pixel en el mapa de bits actualmente seleccionado pasando por parámetros su posición.
- **void draw()**  
Dibuja el mapa de bits actualmente seleccionado tomando en cuenta toda la configuración dada y las escalas globales para el tamaño real en cada eje.
- **void draw\_in\_another\_target()**  
Dibuja el mapa de bits actualmente seleccionado sin tomar en cuenta las escalas globales, usado para dibujar en otro objetivo.

#### Operadores:

- **operator ALLEGRO\_BITMAP\* ()**  
Retorna un puntero **ALLEGRO\_BITMAP** que representa al mapa de bits actualmente seleccionado para que pueda ser usado con las funciones restantes de Allegro.

#### Destruyores:

- **~ Sprite()**  
Elimina todos los mapas de bits creados y libera toda la memoria utilizada por la clase.



## 3 namespace `LL_ENet`

Espacio de nombres que incluye código amigable para manejar la capa de comunicación red que nos brinda la librería ENet.

### 3.1 ENet (`LexRisLogic/ENet/ENet.h`)

#### 3.1.1 Función: Instalar Sistema ENet.

**bool** `install_enet()`

Instala el sistema ENet para poder ser utilizado por el software, retorna verdadero en caso de éxito.

#### 3.1.2 Función: Desinstalar Sistema ENet.

**void** `uninstall_enet()`

Desinstala el sistema ENet cerrando conexiones del software con el sistema.

### 3.2 Servidor (`LexRisLogic/ENet/Server.h`)

#### 3.2.1 Clase: `Server`

Clase que representa a un servidor con conexiones peer-to-peer con protocolo UDP.

**Funciones:**

- **bool** `set_ip(std::string new_ip)`  
Cambia la dirección IP a la que pertenece el servidor, retorna falso si la dirección IP no es válida o si el servidor ya esta corriendo.
- **std::string** `get_ip()`  
Retorna la dirección IP asignada para el servidor.
- **bool** `set_port(unsigned int new_port)`  
Cambia el puerto asignado al servidor, retorna falso si el servidor ya esta corriendo.
- **unsigned int** `get_port()`  
Retorna el puerto asignado al servidor.
- **bool** `set_max_peers_connections(unsigned int max_peers_connections)`  
Cambia el número de conexiones máximas permitidas por el servidor, retorna falso si el servidor ya esta corriendo.
- **unsigned int** `get_max_peers_connections()`  
Retorna el número de conexiones máximas permitidas por el servidor.
- **void** `set_wait_time(unsigned int wait_time)`  
Cambia el tiempo de espera en milisegundos para los eventos generados por el servidor.
- **unsigned int** `get_wait_time()`  
Retorna el tiempo de espera para los eventos.
- **bool** `start_server()`  
Inicia el servidor para recibir conexiones y enviar mensajes, retorna verdadero si se ha logrado crear un Host con la dirección IP y el puerto asignado.
- **bool** `get_event()`  
Revisa la cola de eventos del servidor, retorna verdadero si se encuentra un evento en la cola de eventos.
- **bool** `get_new_connection()`  
Retorna verdadero si el evento generado es una nueva conexión de un cliente.

- **bool** `get_disconnection()`  
Retorna verdadero si el evento generado es una desconexión de un cliente.
- **ENetPeer\*** `get_peer_connected()`  
Retorna un puntero a una conexión peer de un cliente que se ha conectado.
- **ENetPeer\*** `get_peer_disconnected()`  
Retorna un puntero a una conexión peer de un cliente que se ha desconectado.
- **ENetPeer\*** `get_issuer_peer()`  
Retorna un puntero a una conexión peer de un cliente propietario del actual mensaje en la cabeza de la cola de paquetes.
- **std::string** `get_message_received()`  
Retorna el actual mensaje en la cabeza de la cola de paquetes.
- **bool** `remove_message_received()`  
Elimina el mensaje en la cabeza de la cola de paquetes, retorna verdadero si el mensaje fue removido.
- **void** `clear()`  
Vacía la cola de paquetes.
- **bool** `empty()`  
Retorna verdadero si la cola de paquetes esta vacía.
- **bool** `send(ENetPeer* receiver, std::string message)`  
Envía un mensaje a una conexión peer de un cliente, retorna verdadero si se envió por la red con éxito.
- **bool** `broadcast(std::string message)`  
Envía un mensaje a todos los clientes conectados al servidor, retorna verdadero si se envió por la red con éxito.
- **bool** `stop_server()`  
Detiene el servidor, limpiando la cola de paquetes y cerrando todas las conexiones.

#### Destructores:

- **~ Server()**  
Detiene el servidor y libera toda la memoria utilizada por la clase.

### 3.3 Cliente (LexRisLogic/ENet/Client.h)

#### 3.3.1 Clase: **Client**

Clase que representa a un cliente con conexiones peer-to-peer con protocolo UDP.

#### Funciones:

- **bool** `set_ip(std::string new_ip)`  
Establece la dirección IP del servidor al cual se conectara el cliente.
- **std::string** `get_ip()`  
Retorna la dirección IP del servidor al cual se conectara el cliente.
- **bool** `set_port(unsigned int new_port)`  
Establece el puerto del servidor al cual se conectara el cliente.
- **unsigned int** `get_port()`  
Retorna el puerto del servidor al cual se conectara el cliente.
- **bool** `set_test_time(unsigned int new_test_time)`  
Cambia el tiempo de prueba en milisegundos para obtener la conexión al servidor, retorna falso si el tiempo es 0 o si ya se inicio el Host propio del cliente.

- **unsigned int get\_test\_time()**  
Retorna el tiempo de prueba de conexión.
- **void set\_wait\_time(unsigned int wait\_time)**  
Cambia el tiempo de espera en milisegundos para los eventos generados por el cliente.
- **unsigned int get\_wait\_time()**  
Retorna el tiempo de espera para los eventos.
- **bool start\_client()**  
Inicia el Host del cliente preparándolo para una futura conexión con un servidor, retorna verdadero si se ha logrado crear con éxito el Host del cliente.
- **bool connect\_to\_server()**  
Inicia la conexión con el servidor, si lo consigue probará la conexión para informar al servidor que se ha conectado un cliente, retorna verdadero en caso de éxito.
- **bool disconnect\_from\_server()**  
Vacía la lista de eventos del cliente y se desconecta del servidor cuando recibe la señal de desconexión por parte del servidor.
- **bool get\_connection\_status()**  
Retorna el estado de la conexión del cliente con el servidor.
- **bool get\_event()**  
Actualiza todas las variables del servidor y retorna verdadero si se generó algún evento en un tiempo definido anteriormente.
- **std::string get\_message\_received()**  
Retorna el actual mensaje recibido por el servidor en la cola de paquetes.
- **bool remove\_message\_received()**  
Elimina el mensaje en la cabeza de la cola de paquetes, retorna verdadero si el mensaje fue removido.
- **void clear()**  
Vacía la cola de paquetes.
- **bool empty()**  
Retorna verdadero si la cola de paquete está vacía.
- **bool send(std::string message)**  
Envía un mensaje al servidor, retorna verdadero si se envió por la red con éxito.
- **bool stop\_client()**  
Detiene la conexión con el servidor solo si se encuentra aún conectado, detiene el host del cliente y limpia la cola de paquetes.

#### Destructores:

- **~ Client()**  
Detiene el Host del cliente y libera toda la memoria utilizada por la clase.

## 4 namespace `LL_irrKlang`

Espacio de nombres que incluye un código amigable para manejar el motor de sonido irrKlang, usando la API de irrKlang.

### 4.1 Motor de Sonido (`LexRisLogic/irrKlang/irrKlang.h`)

#### 4.1.1 Clase: `SoundEngine`

Clase que representa al motor de sonido irrKlang.

**Funciones:**

- `bool create()`  
Crea un motor de sonido irrKlang para poder reproducir archivos de audio.
- `void set_default_engine()`  
Establece al objeto como motor de sonido por defecto.
- `void set_volume(float new_volume)`  
Establece el volumen principal del motor donde el valor de 1.0 es normal.
- `float get_volume()`  
Obtiene el volumen principal actual del motor.
- `bool destroy()`  
Destruye el motor de sonido irrKlang del objeto.

**Operadores:**

- `operator irrklang::ISoundEngine* ()`  
Retorna un puntero al motor de sonido irrKlang para manejarlo con las demás funciones de irrKlang.

**Destruyores:**

- `~ SoundEngine()`  
Destruye el motor de sonido irrKlang del objeto y libera la memoria utilizada por el objeto.

#### 4.1.2 Variable: `extern SoundEngine* default_engine`

Puntero al principal motor de sonido irrKlang, que es usado para poder reproducir archivos de audio.

## 4.2 Audio (`LexRisLogic/irrKlang/Audio.h`)

#### 4.2.1 Clase: `Audio`

**Funciones:**

- `void set_path(std::string new_audio_path)`  
Se asigna la ruta del archivo de audio que va a ser reproducido por el motor de sonido.
- `std::string get_path()`  
Retorna la ruta del archivo de audio que va a ser reproducido por el motor de sonido.
- `bool set_speed(float new_speed)`  
Asigna la velocidad de reproducción del audio, retorna verdadero en éxito.
- `float get_speed()`  
Retorna la velocidad de reproducción actual del audio.

- **bool set\_pan(float new\_pan)**  
Cambia la dirección de la reproducción del audio. Los valores que toma la variable new\_pan están entre [-1,1] donde:
  - Reproducción normal es 0
  - Reproducción por parlante izquierdo es -1
  - Reproducción por parlante derecho es 1
- **float get\_pan()**  
Retorna la dirección de la reproducción del audio.
- **void set\_volume(float new\_volume)**  
Cambia el volumen de la reproducción del audio a escala del volumen asignado actualmente en el motor de sonido irrKlang.
- **float get\_volume()**  
Retorna el volumen asignado a la reproducción del audio.
- **bool set\_loop\_mode(bool loop\_mode\_on)**  
Cambia el modo de reproducción por ciclos.
- **bool get\_loop\_mode()**  
Retorna verdadero si se el modo de la reproducción es por ciclos.
- **bool load()**  
Carga el archivo de audio como audio irrKlang para poder reproducirlo, retorna verdadero en caso de éxito.
- **unsigned int size()**  
Retorna la duración en milisegundos del archivo de audio.
- **bool set\_position(unsigned int new\_position)**  
Establece la posición de reproducción enviada por parámetro en milisegundos, retorna verdadero en caso de éxito.
- **unsigned int get\_position()**  
Retorna la posición actual de la reproducción del audio.
- **bool is\_playing()**  
Retorna verdadero si la reproducción esta activa.
- **void stop()**  
Detiene la reproducción.
- **void pause()**  
Pausa la reproducción.
- **void play()**  
Inicia o reanuda la reproducción.
- **bool destroy()**  
Destruye el audio irrKlang creado para la reproducción del archivo de audio.

#### Operadores:

- **operator irrklang::ISound\* ()**  
Retorna un puntero al audio irrKlang para manejarlo con las demás funciones de irrKlang.

#### Destruyores:

- **~ Audio()**  
Destruye el audio irrKlang y libera la memoria utilizada por el objeto.

## 5 namespace `LL_MathStructure`

Espacio de nombres que incluye un conjunto de estructuras matemáticas y funciones geométricas para su interacción.

### 5.1 Punto (`LexRisLogic/MathStructures/Point.h`)

#### 5.1.1 Clase: `Point`

Clase que representa a un punto en una dimensión.

**Constructores:**

- `Point()`  
Crea el punto sin definir la dimensión.
- `Point(unsigned int dimension)`  
Crea el punto definiendo la dimensión en que trabajara y dando como valor 0 en cada eje.
- `Point(const Point& another_point)`  
Constructor para copia profunda, usado para hacer una copia real de la estructura interna de la clase sin alterar información.

**Funciones:**

- `bool set_dimension(unsigned int new_dimension)`  
Cambia la dimensión del objeto, eliminando el anterior punto y creando otro con valor 0 en cada eje dimensional.
- `unsigned int get_dimension() const`  
Retorna el valor de la dimensión del punto.

**Operadores:**

- `float& operator [] (unsigned int index)`  
Retorna el eje dimensional indicado por el índice, por referencia para poder ser modificado.
- `const float operator [] (unsigned int index) const`  
Retorna el eje dimensional indicado por el índice, el objetivo de esta réplica es dar un error si se trata de cambiar el valor en una dimensión de un punto constante.
- `Point& operator = (const Point& another_point)`  
Copia la dimensión y todos los valores de cada eje dimensional de otro punto.
- `bool operator == (const Point& another_point) const`  
Retorna verdadero si el otro punto es igual al que invoca la función.
- `bool operator != (const Point& another_point) const`  
Retorna verdadero si el otro punto es diferente al que invoca la función.

**Destructores:**

- `~Point()`  
Elimina al punto y libera la memoria utilizada.

#### 5.1.2 Operador: `<<` para la clase `Point`

`std::ostream& operator << (std::ostream& output_stream, Point point)`

Operador de salida estándar en consola o terminal del punto.

#### 5.1.3 Función: Distancia Euclidiana

`double euclidean_distance(Point first_point, Point second_point)`

Retorna la distancia euclidiana entre dos puntos si pertenecen a la misma dimensión.

#### 5.1.4 Función: Crear punto 2D

**Point** create\_point(float x,float y)

Crea un punto de dimensión dos.

#### 5.1.5 Función: Crear punto 3D

**Point** create\_point(float x,float y,float z)

Crea un punto de dimensión tres.

### 5.2 Segmento (LexRisLogic/MathStructures/LineSegment.h)

#### 5.2.1 Clase: LineSegment

Clase que representa a un segmento de recta en cualquier dimensión.

**Constructores:**

- **LineSegment()**  
Crea un segmento sin definir la dimensión.
- **LineSegment(unsigned int dimension)**  
Crea un segmento donde el punto de inicio y final tienen el valor del punto de origen en la dimensión asignada, cuando un segmento tiene los dos puntos iguales hacen que no sea válido para las funciones de intersección.

**Funciones:**

- **bool set\_dimension(unsigned int new\_dimension)**  
Cambia la dimensión del segmento asignando al punto de inicio y final el valor del punto de origen.
- **unsigned int get\_dimension()**  
Retorna el valor de la dimensión del segmento.
- **bool set\_ini\_point(Point new\_ini\_point)**  
Cambia el punto inicial del segmento solo si pertenece a la misma dimensión, retorna verdadero en caso de éxito.
- **const Point get\_ini\_point()**  
Retorna el punto inicial del segmento.
- **bool set\_end\_point(Point new\_end\_point)**  
Cambia el punto final del segmento solo si pertenece a la misma dimensión, retorna verdadero en caso de éxito.
- **const Point get\_end\_point()**  
Retorna el punto final del segmento.
- **bool in\_range(unsigned int dimension,float number)**  
Retorna verdadero si un número se encuentra en el rango dimensional seleccionado del segmento.

#### 5.2.2 Función: Intersección de Lineas en Dos Dimensiones

**bool** intersection\_of\_lines\_in\_two\_dimensions(**LineSegment** first\_line,  
  **LineSegment** second\_line,  
  **float\*** x=**nullptr**,**float\*** y=**nullptr**)

Retorna la intersección entre dos rectas de dos dimensiones, retorna tres valores, Si existe intersección entre dos rectas y el punto de intersección; puede evitar mandar las variables x e y, para no obtener el valor de intersección; la función retorna falso si se envían rectas paralelas o incorrectas.

### 5.2.3 Función: Intersección de Segmentos en Dos Dimensiones

**bool** intersection\_of\_line\_segments\_in\_two\_dimensions(**LineSegment** first\_segment,  
   **LineSegment** second\_segment,  
   float\* x=nullptr,float\* y=nullptr)

Retorna la intersección entre dos rectas de dos dimensiones, retorna tres valores, Si existe intersección entre dos segmentos y el punto de intersección; puede evitar mandar las variables x e y, para no obtener el valor de intersección; la función retorna falso si se envían segmentos paralelos o incorrectos; los valores de x e y pueden haber sido modificados incluso aunque no exista intersección entre los segmentos.

## 5.3 Polígono ([LexRisLogic/MathStructures/Polygon.h](#))

Clase que representa un polígono de puntos de dos dimensiones.

### 5.3.1 Clase: **Polygon**

Funciones:

- **bool** add\_point(**Point** point)  
Añade al polígono un punto de segunda dimensión al final de la lista, retorna verdadero en caso de éxito.
- **bool** remove\_point(**unsigned int** index)  
Remueve el punto del polígono de la posición elegida, retorna verdadero en caso de éxito.
- **unsigned int** size()  
Retorna el número de puntos que tiene el polígono.
- **void** clear()  
Vacía la lista de puntos del polígono.
- **bool** set\_point(**unsigned int** Point,**Point** new\_point)  
Cambia el punto del polígono en la posición indicada si el nuevo punto pertenece a la segunda dimensión, retorna verdadero en caso de éxito.

Operadores:

- **const Point operator []** (**unsigned int** index)  
Retorna el punto del polígono en la posición indicada.

### 5.3.2 Función: Punto dentro de un Polígono

**bool** point\_into\_polygon(**Polygon** polygon,**Point** point)

Retorna verdadero si el punto de segunda dimensión enviado por parámetro se encuentra en el interior del polígono.

### 5.3.3 Función: Colisión de Polígonos

**bool** collision\_of\_polygons(**Polygon** first\_polygon,**Polygon** second\_polygon,  
   std::list<**Point**>\* points=nullptr)

Retorna dos valores, verdadero o falso si existe colisión entre dos polígonos enviados y un puntero a una lista donde se guardaran todos los puntos de intersección.



## 6 namespace **LL\_DataStructure**

Espacio de nombres que incluye estructuras de datos y funciones para su interacción.

### 6.1 Matriz Dispersa (**LexRisLogic/DataStructures/SparseMatrix.h**)

#### 6.1.1 Clase: **SparseMatrix**

**template<typename T>**

Matriz Dispersa que almacena los datos diferentes al valor nulo, es decir, valor que representa al vacío en la matriz con el objetivo de disminuir la memoria utilizada.

**Constructores:**

- **SparseMatrix(unsigned int size\_x, unsigned int size\_y, T null\_value)**  
Crea una Matriz de dimensión [size\_x, size\_y] vacía y se pasa el valor que representara el vacío en la matriz.

**Clases:**

- **Class\_Controller**  
Clase controladora de una posición de la matriz, se encarga de asignarle valor o eliminar el valor en dicha posición actualizando la matriz dispersa.

**Constructores:**

- **Class\_Controller(\_\_SparseMatrixNodeBase\*\* pointer\_x, \_\_SparseMatrixNodeBase\*\* pointer\_y, unsigned int pos\_x, unsigned int pos\_y, T null\_value)**

Solo puede ser usado por la matriz dispersa de manera correcta, por lo que ya existe una función que nos retorna esta clase ya configurada, no existen controladores vacíos.

**Funciones:**

- **T get\_value()**  
Retorna una copia del dato que contiene en la matriz en tal posición.

**Operadores:**

- **T operator = (T new\_data)**  
Operador para asignar el dato que guardara la matriz en la posición requerida, si el dato es el valor nulo se eliminará de memoria el objeto en esa posición.
- **operator T ()**  
Operador de conversión automática para evitar llamar a get\_Value() y tener un código más legible.

**Funciones:**

- **unsigned int get\_size\_x()**  
Retorna el tamaño de la matriz en el eje x.
- **unsigned int get\_size\_y()**  
Retorna el tamaño de la matriz en el eje y.
- **T get\_null\_value()**  
Retorna el valor que representa al vacío en la matriz dispersa.
- **void clear()**  
Elimina todos los datos almacenados en la matriz dispersa y libera la memoria utilizada por estos.

#### Operadores:

- **Class\_Controller operator () (unsigned int pos\_x,unsigned int pos\_y)**  
Retorna el controlador de la matriz dispersa en para posición pasada por parámetro.

#### Destructores:

- **~ SparseMatrix()**  
Destruye todos los objetos en la matriz y libera la memoria utilizada por la estructura.

## 6.2 Árbol-R (LexRisLogic/DataStructures/RTree.h)

### 6.2.1 Estructura: MBB

Estructura que contiene dos puntos en una dimensión para representar el cuadro mínimo delimitador de un objeto en el espacio.

#### Constructor:

- **MBB()**  
Crea los puntos representantes del cuadro mínimo delimitador sin asignarles una dimensión.
- **MBB(unsigned int new\_dimension)**  
Inicia los puntos representantes del cuadro mínimo delimitador pasando la dimensión a la que pertenece.

#### Variables:

- **unsigned int dimension=0**  
Dimensión del cuadro mínimo delimitador.
- **LL\_MathStructure::Point first\_point**  
Primer Punto representante del cuadro mínimo delimitador.
- **LL\_MathStructure::Point second\_point**  
Segundo Punto representante del cuadro mínimo delimitador.

#### Funciones:

- **bool set\_dimension(unsigned int new\_dimension)**  
Actualiza la dimensión del cuadro mínimo delimitador y los puntos representantes.

#### Operadores:

- **bool operator == (MBB another\_mbb)**  
Compara los dos puntos representantes de otro cuadro mínimo delimitador, retorna verdadero si ambos puntos son iguales.
- **bool operator != (MBB another\_mbb)**  
Compara los dos puntos representantes de otro cuadro mínimo delimitador, retorna verdadero si algún punto es diferente.

### 6.2.2 Función: Distancia entre Cuadros Mínimos Delimitadores

**double mbb\_distance(MBB first\_mbb,MBB second\_mbb)**

Retorna la distancia mínima entre dos Cuadros Mínimos delimitadores.

### 6.2.3 Clase: **RTree**

**template<typename T,int DIMENSION,unsigned int NODE\_SIZE>**

Estructura de Datos Espacial, que guarda objetos con igual tipo de dato, se pasa una función para hallar el Cuadro Mínimo Delimitador de estos para poder ubicarlos en el espacio, y también se pasa el tamaño de los nodos para evitar la sobre-posición.

**Constructores:**

- **RTree(MBB (\*Function\_to\_mbb)(T))**

Recibe un puntero a una función que reciba el tipo de dato que guardará la estructura y devuelva su respectivo Cuadro Mínimo Delimitador, la estructura usara esta función para encontrar su posición en el espacio e insertarlos en la estructura, tener en cuenta que la función debe ser biyectiva porque es usada para las operaciones de búsqueda y eliminado.

**Clases:**

- **iterator**

Iterador para recorrer por todos los elementos de la estructura, retorna una copia del objeto usado para solo lectura, tener cuidado con el destructor del tipo de dato almacenado.

**Constructores:**

- **iterator(\_RTreeNodeBase\_\* root\_node=nullptr)**

Recibe un puntero al nodo del Árbol-R, este nodo sera tratado como el nodo raíz y generara la estructura del iterador para poder recorrer los datos del nodo; por defecto se pasa un puntero nulo para crear iteradores vacíos.

**Operadores:**

- **iterator operator ++ (int)**

Crea una copia del iterador actual, luego itera al siguiente elemento actualizando la estructura y retorna la copia.

- **iterator operator ++ ()**

Itera al siguiente elemento y actualiza la estructura sin crear una copia, se retorna a si mismo con el cambio realizado.

- **const T operator \* ()**

Accede a una copia del dato que actualmente guarda el iterador.

- **bool operator != (iterator another\_iterator)**

Retorna verdadero si dos iteradores son diferentes, es decir, que no se refieran al mismo objeto.

**Funciones:**

- **iterator begin()**

Retorna un iterador al inicio de la estructura.

- **iterator end()**

Retorna un iterador vacío que representa el final de la estructura.

- **unsigned int size()**

Retorna el número de elementos que actualmente guarda la estructura.

- **void clear()**

Elimina todos los datos almacenados en la estructura.

- **bool find(T data)**

Busca si un dato ya ha sido insertado.

- **bool insert(T new\_data)**

Inserta un nuevo elemento a la estructura, retorna falso si el cuadro mínimo delimitador no pertenece a la dimensión o si ya se encuentra un dato con igual cuadro mínimo delimitador.

- **bool** `remove(T data_to_remove)`  
Remueve un dato de la estructura, retorna verdadero si lo encuentra y logra eliminar.
- **std::list<T>** `range_query(MBB mbb)`  
Retorna todos los elementos que se encuentren dentro del Cuadro Mínimo Delimitador pasado por parámetro.

#### **Destructores:**

- `~ RTree()`  
Elimina todos los datos de la estructura y libera toda la memoria utilizada por la estructura.

## 7 namespace **LL**

Espacio de Nombres que incluye funciones y clases básicas listadas más adelante.

### 7.1 Conversión de Tipo (**LexRisLogic/Convert.h**)

#### 7.1.1 Función: Conversión de Cualquier Tipo a Cadena.

```
template<typename T>  
std::string to_string(T data)
```

Convierte el objeto a una cadena, para poder convertir el objeto es necesario tener sobrecargado el operador <<(**ostream**, **T**).

#### 7.1.2 Función: Conversión de Cadena a Entero.

```
int to_int(std::string data)
```

Convierte la cadena a un entero, la cadena solo debe contener los caracteres numéricos para poder ser convertida.

#### 7.1.3 Función: Conversión de Cadena a Punto Flotante Simple.

```
float to_float(std::string data)
```

Convierte la cadena a un punto flotante simple, la cadena solo debe tener caracteres numéricos y un '.' separando la parte entera de la parte flotante.

#### 7.1.4 Función: Conversión de Cadena a Punto Flotante Doble.

```
double to_double(std::string data)
```

Convierte la cadena a un punto flotante doble, la cadena solo debe tener caracteres numéricos y un '.' separando la parte entera de la parte flotante.

## 7.2 Encriptador (LexRisLogic/Encryptor.h)

### 7.2.1 Constante: `const char* const DEFAULT_DICTIONARY`

Diccionario de caracteres por defecto que usará el encriptador de cadenas.

### 7.2.2 Clase: `Encryptor`

Clase que encripta una cadena en un mensaje que también puede ser descryptado con la misma clase.

**Funciones:**

- `bool add_new_key(std::string new_key)`  
Adiciona una nueva llave para poder encriptar las cadenas, se valida la llave de acuerdo al diccionario actual; se pueden adherir más llaves generando así una nueva llave.
- `void clear_keys()`  
Elimina todas las llaves que tiene actualmente el encriptador.
- `void set_dictionary(std::string new_dictionary)`  
Cambia el diccionario actual, esto eliminara la llaves que tiene actualmente el encriptador.
- `std::string get_dictionary()`  
Retorna el diccionario que esta usando el encriptador actualmente.
- `std::string encrypt(std::string message)`  
Codifica el mensaje y retorna el mensaje encriptado.
- `std::string decrypt(std::string encrypted_message)`  
Decodifica un mensaje encriptado y retorna el mensaje real.

**Destruyores:**

- `~Encryptor()`  
Elimina las llaves actuales y libera la memoria utilizada por el encriptador.

## 7.3 Lector de Archivos (LexRisLogic/FileStream.h)

### 7.3.1 Clase: FileStream

Clase que proporciona un lector de archivos texto plano, y funciones para controlar cada línea del archivo.

**Funciones:**

- **void set\_path(std::string new\_path)**  
Se asigna la ruta del archivo que sera creado o se leerá.
- **std::string get\_path()**  
Retorna la ruta del archivo.
- **bool load()**  
Carga todas las lineas del archivo, que se encuentra en la ruta dada, en la estructura para que pueda ser manejada con las funciones de control línea a línea, retorna falso si no existe el archivo requerido; si se necesita borrar los cambios y recargar el archivo se puede llamar otra vez esta función.
- **bool save()**  
Guarda el archivo modificado, puede cambiarse la ruta del archivo antes de guardar, para evitar sobrescribir el archivo original.
- **void clear\_file()**  
Limpia el archivo eliminando todas las líneas del archivo.
- **bool insert\_line(unsigned int insert\_position, unsigned int total\_of\_new\_lines)**  
Inserta un número de líneas antes de la línea en la posición pasada por parámetro, retorna verdadero en caso de éxito.
- **bool remove\_line(unsigned int remove\_position)**  
Remueve la línea que se encuentra en la posición dada, retorna verdadero en caso de éxito.
- **unsigned int size()**  
Retorna el número de líneas que tiene actualmente el archivo.

**Operadores:**

- **std::string& operator [] (unsigned int line\_position)**  
Retorna la línea que se encuentra en la posición dada, retorna por referencia para que pueda ser manipulada la línea del archivo.

**Destructores:**

- **~ FileStream()**  
Libera la memoria utilizada por el lector de archivos.

## 7.4 Matemática (LexRisLogic/Math.h)

### 7.4.1 Constante: `const double MATH_PI`

Valor de PI.

### 7.4.2 Función: Módulo (Matemática Discreta).

`int mod(int dividend, int divisor)`

Retorna el valor de  $a \bmod b$ , es decir, el valor que debería tomar  $a$  en un rango discreto de 0 a  $b - 1$ .

### 7.4.3 Función: Módulo (Matemática Continua).

`double range_mod(double dividend, double divisor)`

Se mantiene la idea del módulo para ser usado en rangos discretos pero ahora con rangos continuos, por ejemplo  $3.4 \bmod 3.3$  es igual a 0.1, hacer trabajar a los números en un rango continuo de  $0 \leq X < 3.3$ , donde  $X$  sera el valor que tomaría  $a$  en base a  $b$ .

### 7.4.4 Función: Conversión de Grados Sexagesimales a Radianes

`float sexagesimal_to_radian(float sexagesimal)`

Convierte los grados sexagesimales enviados a radianes.

### 7.4.5 Función: Conversión de Radianes a Grados Sexagesimales

`float radian_to_sexagesimal(float radian)`

Convierte los radianes enviados a grados sexagesimales.

### 7.4.6 Función: Generadora de Semillas

`void random_generate_new_seed()`

Genera una semilla nueva para el generador de números aleatorios.

### 7.4.7 Función: Generar un Número Aleatorio

`int random(int min_value, int max_value, bool include_max_value=false)`

Retorna un número aleatorio que se encuentre en el rango de  $[\text{min\_value} : \text{max\_value}]$ , con el valor de `include_max_value` se puede incluir la posibilidad de que también se genere el número `max_value`.

### 7.4.8 Función: Colisión entre Segmentos

`bool segment_collision(float ini_segment_1, float fin_segment_1,  
float ini_segment_2, float fin_segment_2)`

Usado para conocer si dos segmentos presentan colisión.

### 7.4.9 Función: Máximo Entero de un Número

`int max_integer(float number)`

Retorna el máximo entero del número flotante.



## 7.5 Separador de Cadenas (LexRisLogic/StringSplitter.h)

### 7.5.1 Clase: StringSplitter

Clase que representa a una función de separación de cadenas mediante un carácter.

**Funciones:**

- **void set\_string(std::string new\_string)**  
Cambia la cadena que será separada por la clase.
- **std::string get\_string()**  
Retorna la cadena que será o ya ha sido separada por la clase.
- **bool split(char character='\n')**  
Separa la cadena de acuerdo al carácter enviado, por defecto se separa por saltos de líneas; todas estas subcadenas son guardadas en la estructura interna de la clase, retorna verdadero en caso de éxito.
- **unsigned int size()**  
Retorna el número de elementos que generó la separación de la cadena.
- **void clear()**  
Elimina todos los elementos que tiene la estructura interna de la clase.

**Operadores:**

- **const std::string operator [] (unsigned int index)**  
Retorna la i-ésima separación hecha por el separador sobre la cadena asignada.

**Destructores:**

- **~ StringSplitter()**  
Libera la memoria utilizada por la estructura interna que guarda todas las subcadenas generadas al separar la cadena.

## 7.6 Tiempo (LexRisLogic/Time.h)

### 7.6.1 Clase: Chronometer

Clase que representa al uso de un cronómetro para conocer el tiempo en segundos que pasa al realizar alguna actividad.

**Funciones:**

- **bool play()**  
Inicia el Cronómetro.
- **bool pause()**  
Pausa el Cronómetro.
- **bool stop()**  
Detiene el Cronómetro.
- **const double get\_time()**  
Obtiene el tiempo acumulado por el cronómetro.
- **void clear()**  
Reinicia el cronómetro limpiando los datos guardados, puede ser usado en cualquier estado del cronómetro.

## 8 Notas

Toda esta documentación es dada para que se pueda conocer sobre el funcionamiento que brindan las Cabeceras de LexRis Logic, dirigida para la Versión v2.00.

## 9 License

Copyright (c) 2016 copyright LexRis Logic

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.