LexRis Logic Headers
Guía del Programador

LexRis Logic: Christian Benavides Castillo

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 Datos, Encriptador, Lector de Archivos, Separador de Cadenas y en especial de las librerías:

- Allegro v5.1.9 (WIP)
- ENet v1.3.13

2 Allegro (LexRisLogic/Allegro/LL_Allegro.h)

Cabecera que ofrece gran parte de todo el sistema de ALLEGRO para ser iniciado, variables importantes usadas para las operaciones de dibujo, control e información y algunas cabeceras con diferentes clases representado a las estructuras de ALLEGRO.

2.1 Tipos

- **typedef float pos_t**
  Tipo para las variables de posición en pantalla.
- **typedef unsigned int display_size_t**
  Tipo para variables de tamaño de pantalla.

2.2 Variables Globales

- **float scale_x**
  Variable usada para el tamaño de las imágenes, videos y figuras primitivas en el eje X.
- **float scale_y**
  Variable usada para el tamaño de las imágenes, videos y figuras primitivas en el eje y.
- **float text_scale**
  Variable usada para el tamaño del fuentes para dibujar texto.
- **float primitives_scale**
  Variable usada para el grosor de línea de las figuras primitivas.

- **bool exit_program**
  Variable que puede usarse para el control de salida de programa (Uso Recomendable).

- **display_size_t desktop_size_x**
  Variable que al iniciarse el sistema ALLEGRO contendrá la información del tamaño del eje x del escritorio.

- **display_size_t desktop_size_y**
  Variable que al iniciarse el sistema ALLEGRO contendrá la información del tamaño del eje y del escritorio.

2.3 Funciones Globales

- **void init_allegro()**
  Función Principal que inicia todo el sistema de ALLEGRO, halla la información del escritorio, y cuando se cierre el programa automáticamente se cerrará todo el sistema de ALLEGRO.

- **void rest(float _time)**
  Detiene el proceso durante un tiempo _time en segundos.

- **bool primitives_addon()**
  Instala el sistema de figuras primitivas para su uso.
  Retorna verdad en correcta instalación.

- **bool image_addon()**
  Instala el sistema de imágenes para ser cargadas, editadas y guardadas.
  Retorna verdad en correcta instalación.

- **bool text_addon()**
  Instala el sistema de fuentes para mostrar texto en pantalla.
  Retorna verdad en correcta instalación.

- **bool audio_addon()**
  Instala el sistema de Audio para reproducir muestras de formato *.wav, *.flac, *.ogg, *.it, *.mod, *.s3m, *.xm.
  Retorna verdad en correcta instalación.

- **void uninstall_audio()**
  Desinstala el sistema de audio.

- **void uninstall_primitives()**
  Desinstala el sistema de figuras primitivas.

2.4 Cabeceras

2.4.1 Pantalla (LexRisLogic/Allegro/LL_Display.h)

Contenido:
→ Clase: **Display** representando a **ALLEGRO_DISPLAY**.

1. **Clase Display y Funciones**

   - **Display**
     Clase para manejar las opciones de una ventana (pantalla) que sera objetivo de las operaciones de dibujo.
   - **Display(display_size_t SizeX, display_size_t SizeY)**
     Constructor, crea una pantalla de tamaño SizeX*SizeY, El tamaño de la pantalla en cualquier tipo de pantalla completa ha de ser específico como: 800*600, 640*480, 1366*768, etc.
   - **void set_title(string T)**
     El título de la pantalla creada es cambiado por la cadena T.
   - **void set_flag(int F)**
     Un flag indica de que tipo sera la pantalla, al ser modificado el flag, la pantalla se volverá a crear con los nuevos parámetros y F toma valores de las macros de ALLEGRO:
     (a) **ALLEGRO_WINDOWED**
     Para una pantalla de tamaño fijo en forma de ventana.
     (b) **ALLEGRO_FULLSCREEN**
     Utiliza el modo pantalla completa que el driver de pantalla brinda al sistema operativo, usar mas de una actualización seguida en este modo puede causar que el actualizado no sea óptimo (rapidez y calidad).
     (c) **ALLEGRO_FULLSCREEN_WINDOW**
     Genera una pantalla completa en forma de ventana, no genera errores con el actualizado seguido.
     (d) **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(display_size_t SizeX, display_size_t SizeY)**
     Redimensiona la pantalla a tamaño SizeX*SizeY, solo se puede usar en modo **ALLEGRO_WINDOWED** o **ALLEGRO_RESIZABLE**.
   - **display_size_t get_sizex()**
     Retorna el tamaño actual del eje X de la pantalla.
   - **display_size_t get_sizey()**
     Retorna el tamaño actual del eje Y de la pantalla.
   - **void set_target()**
     Actualiza la pantalla objetivo de las operaciones de dibujo, seleccionando a la pantalla que llame este método.
   - **void clean()**
     Limpia la pantalla con el color blanco por defecto.

- **void refresh()**
  Actualizar la pantalla y muestra la imagen generada en pantalla.
- **operator ALLEGRO_DISPLAY*& ()**
  Retorna el puntero **ALLEGRO_DISPLAY** para que pueda ser usado con las funciones restantes de ALLEGRO.
- **operator ALLEGRO_BITMAP* ()**
  Retorna un puntero a la imagen generada en pantalla, cuidado con manejar operaciones de dibujo, porque puede causar problemas en la visualización de pantalla.
- **~ Display()**
  Destructor, destruye la pantalla y la interfaz creada.

2.4.2 Cuadros de Diálogo Nativos (LexRisLogic/Allegro/LL_Native_Dialog.h)

Contenido:
→ Función: Cuadro de Diálogo.
→ Clase: **FileChooser** representando a **ALLEGRO_FILECHOOSER**.
→ Clase: **TextLog** representando a **ALLEGRO_TEXTLOG**.

1. **Función de Cuadro de Diálogo**
   bool show_native_message(ALLEGRO_DISPLAY*, string title, string header, string text, int flags)
   Función que se encarga de mostrar un cuadro de diálogo donde la cadena title es el título del cuadro, la cadena header es el título dentro del cuadro, la cadena text es el mensaje a mostrar al usuario, los flags vienen a ser el tipo de mensaje que pueden ser:

   (a) **ALLEGRO_MESSAGEBOX_WARN**
   Muestra un mensaje de tipo alerta.

   (b) **ALLEGRO_MESSAGEBOX_ERROR**
   Muestra un mensaje de tipo error.

   (c) **ALLEGRO_MESSAGEBOX_QUESTION**
   Muestra un mensaje.

   (d) **ALLEGRO_MESSAGEBOX_OK_CANCEL**
   Cuadro con opciones de aceptar o cancelar.

   (e) **ALLEGRO_MESSAGEBOX_YES_NO**
   Cuadro con opciones de si o no.

   La función retorna verdadero en caso de haber recibido éxito sea una respuesta si o aceptar, y falso si la ventana a recibido alguna cancelación, mensaje de no o cancelar, o a sido cerrada por el usuario.

2. **Clase FileChooser y Funciones**

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

- **void set_display(ALLEGRO_DISPLAY* d)**
  Asigna la pantalla a bloquearse cuando se abra el explorador de archivos.
- **ALLEGRO_DISPLAY*** get_display()
  Retorna un puntero a la pantalla que será bloqueada por el explorador de archivos.
- **void set_path(string path)**
  Indica la ruta inicial del explorador de archivos desde donde iniciara la búsqueda para el usuario.
- **string get_path()**
  Retorna la ruta inicial del explorador de archivos.
- **void set_title(string title)**
  Cambia el título del explorador de archivos.
- **string get_title()**
  Retorna el título del explorador de archivos.
- **void set_mode(int mode)**
  Indica el modo de explorador de archivos y su finalidad, mediante los siguientes flags, pueden usarse combinaciones:
  (a) **ALLEGRO_FILECHOOSER_FILE_MUST_EXIST**
  Permite solo la visualización de los archivos existentes y no la entrada para nuevos archivos.
  (b) **ALLEGRO_FILECHOOSER_SAVE**
  El explorador entra en el modo de salvado de un archivo mediante la entrada de un nuevo nombre para el archivo.
  (c) **ALLEGRO_FILECHOOSER_FOLDER**
  Crea un explorador de archivos para búsqueda de carpetas.
  (d) **ALLEGRO_FILECHOOSER_PICTURES**
  Crea un explorador con visualización de imágenes.
  (e) **ALLEGRO_FILECHOOSER_SHOW_HIDDEN**
  Crea un explorador con la visualización de archivos ocultos.
  (f) **ALLEGRO_FILECHOOSER_MULTIPLE**
  Habilita la selección de varios archivos en el explorador de archivos.
- **unsigned int get_count_selected_files()**
  Retorna el número de archivos seleccionados.
- **void operator () ()**
  Abre el explorador de archivos con todos los parámetros dados.
- **string operator [] (unsigned int pos)**
  Obtiene la o las rutas de los archivos seleccionados por el explorador de archivos que estarán indexados y el parámetro es el índice del dato que se quiere recuperar.
- **~ FileChooser()**
  Destructor, destruye la interfaz creada para el explorador de archivos.

3. Clase TextLog y Funciones

- **TextLog**
  Clase que crea un registro de texto.

- **void set_title(string title)**
  Cambia el título del Registro.
- **string get_title()**
  Retorna el título del Registro.
- **void set_mode(int mode)**
  Indica el Comportamiento de la ventana del Registro donde se puede combinar los flags:
  (a) **ALLEGRO_TEXTLOG_NO_CLOSE**
  Evita que la ventana del registro tenga un botón de cerrado, de lado contrario generara un evento que podrá ser manejado por la clase **Input**.
  (b) **ALLEGRO_TEXTLOG_MONOSPACE**
  La ventana de Registro usara una fuente de espacio sencillo para mostrar el texto.
- **operator ALLEGRO_TEXTLOG*& ()**
  Retorna el puntero **ALLEGRO_TEXTLOG** para que pueda ser usado con las funciones restantes de ALLEGRO.
- **bool is_open()**
  Retorna verdadero si el Registro esta abierto.
- **bool open()**
  Abre una ventana de registro de acuerdo a los parámetros dados, si aún no ha sido abierta.
- **void write(string str)**
  Escribe la cadena str en el registro, si no existe la ventana de registro escribirá la cadena en la consola del programa.
- **void close()**
  Cierra la ventana de registro si ha sido abierta.
- **~ TextLog()**
  Destructor. llama a la función close() para liberar la memoria usada por la ventana de registro.

2.4.3 Entradas (LexRisLogic/Allegro/LL_Input.h)

Contenido:
→ Concepto: Evento
→ Estructura: **Key**.
→ Clase: **KeyControl**.
→ Clase: **Input** representando a:
ALLEGRO_EVENT, **ALLEGRO_EVENT_QUEUE** y **ALLEGRO_TIMER**

1. **Concepto Evento**

   - **Concepto**
     Un evento es cualquier tipo de información de entrada que puede darse a lo largo del tiempo, por lo que se debe estar a la espera de estos eventos que los manejará la clase **Input** el cual controla algunos tipos de eventos.
   - **Tipos**
     (a) Teclado: Estado de cada tecla, al ser presionada la tecla se genera un evento y al ser soltada del mismo modo.
     (b) Ratón: Estado de los botones del Ratón, posición en pantalla con coordenadas (x,y) , y la coordenada z que indica la posición de la rueda o eje vertical del ratón.
     (c) Ventana: Estado del botón cerrar de la ventana de registro o pantalla.
     (d) Paquetes: Más adelante hablaremos de paquetes para la parte de la librería ENET.

2. **Estructura Key, Variables y Funciones**

   - **Key**
     Clase que representa una tecla.
   - **string name**
     Nombre asignado a la tecla.
   - **int keycode**
     Código de la tecla. Puede ser hallado con una función de **KeyControl** llamada getkeycode() que retorna el código de la tecla presionada en el momento de espera de un evento.
   - **bool active**
     Variable que guarda el estado de la tecla, 1 presionada y 0 sin presionar que sera manejado por la clase **Input**.
   - **Key**(**string** n, **int** k)
     Constructor que recibe el nombre y el código de la tecla.

3. **Clase KeyControl y Funciones**

   - **KeyControl**
     Clase que guarda una collección de teclas(**Key**) sin conflictos de nombre y código, para que la clase **Input** pueda manejar solo las teclas que han sido insertadas en esta clase.

- **KeyControl(ALLEGRO_EVENT_QUEUE* NEQ)**
  Constructor, recibe la cola de eventos para poder usar la función getkeycode() y obtener el código de la tecla presionada.
- **Key& operator [] (unsigned int index)**
  Obtiene la llave de acuerdo a su índice, cuidado con modificar el nombre o código de tecla de la llave, para asegurarse de la modificación se crearon dos funciones de modificación por Nombre y Código de Tecla.
- **int find_key(string Name)**
  Obtiene el índice de la llave buscando por Nombre, retorna -1 si no existe.
- **int find_key(int keycode)**
  Obtiene el índice de la llave buscando por Código de Tecla, retorna -1 si no existe.
- **bool add_key(string Name)**
  Inserta una llave con el nombre dado solo sino existe ya una llave con ese nombre e inicia la función getkeycode() para obtener el código de tecla, y luego la inserta si no esta ese código usado.
- **bool add_key(string Name, int keycode)**
  Inserta una llave con el nombre y código solo si no existe una llave con tal nombre o con tal código de tecla.
- **bool mod_key(int index, string new_Name)**
  Modifica el nombre de una llave ubicado con su índice, el nombre solo puede ser modificado si el nuevo nombre no existe en la Collección.
- **bool mod_key(int index, int new_keycode)**
  Modifica el Código de tecla de una llave ubicado con su índice, el Código de Tecla solo puede ser modificado si el código no esta siendo usado por otra tecla.
- **int remove_key(string Name)**
  Remueve la LLave de la collección con el nombre indicado en el parámetro.
- **int remove_key(int keycode)**
  Remueve la llave de la collección con el código de tecla indicado en el parámetro.
- **int get_keycode()**
  Obtiene el código de tecla de un evento de teclado al presionar la tecla, solo funciona si la cola de eventos ha de tener registrado los eventos del teclado, sino no se podrá salir de la función.
- **∼ KeyControl()**
  Destructor, Limpia la collección de llaves.

4. **Clase Input y Funciones**

- **Input**
  Clase que maneja algunos tipos de eventos generados.
- **Input(float fps)**
  Constructor, instala el sistema para la entrada de teclado y ratón,

recibe los fps para crear un temporizador que es registrado en la cola de eventos para generar eventos cada cierto tiempo (Recomendado para saber cuando realizar las operaciones de dibujo).

- **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*& Display)**
  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 change_fps(float fps)**
  Cambia la respuesta de evento del temporizador.
- **void set_key_control(KeyControl* k)**
  Establece la collección de llaves a utilizar, se envía el puntero para evitar una copia de la collección.
- **KeyControl* get_key_control()**
  Retorna un puntero a la collección 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 usar eventos que han sido generados en un tiempo no deseado.
- **bool& operator [] (string Name)**
  Busca el estado de una tecla "Name" solo si se encuentra en la collección establecida y retorna por referencia para que el valor pueda ser modificado.
- **void operator () ()**
  Pregunta y consigue los datos de los eventos de teclado, de la pantalla, de la ventana de registro, del temporizador y del ratón.
- **void get_exit()**
  Pregunta y consigue solo los datos de los eventos de salida de la pantalla o de la ventana de registro.
- **void keyboard_on()** Registra en la cola de eventos al teclado para poder controlar sus eventos generados por cada tecla.
- **void keyboard_off()** Anula el registro del teclado, y deja de controlar sus eventos generados.
- **void mouse_on()** Registra en la cola de eventos al ratón para poder controlar sus eventos generados.
- **void mouse_off()** Anula el registro del ratón, y deja de controlar sus eventos generados.

- **bool input_on(string* X, unsigned int c, bool enter_is_blocked=0)**
  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 X, que es una cadena que existe en memoria, la variable final indica si se bloqueara la lectura para el carácter especial $'\backslash n'$.
- **bool input_off(string* X)**
  Desactiva la entrada de la cadena mediante eventos solo si se envía la cadena actual donde esta siendo guardado el dato para poder asegurar su existencia.
- **bool set_mouse_xy(pos_t x, pos_t y)**
  Cambia la posición del ratón a la posición (x,y) en la pantalla registrada, retorna verdadero en caso de éxito.
- **bool set_mouse_z(int z)**
  Cambia el valor del eje de la rueda del ratón, retorna verdadero en caso de en éxito.
- **pos_t get_mouse_x()**
  Retorna la posición actual del eje x del ratón.
- **pos_t get_mouse_y()**
  Retorna la posición actual del eje y del ratón.
- **int get_mouse_z()**
  Retorna la posición actual del eje z de la rueda del ratón.
- **bool& right_click()**
  Retorna el estado del botón derecho del ratón, el valor puede ser modificado.
- **bool& left_click()**
  Retorna el estado del botón izquierdo del ratón, el valor puede ser modificado.
- **bool& mid_click()**
  Retorna el estado del botón central del ratón, el valor puede ser modificado.
- **bool show_cursor()**
  Muestra el cursor en la pantalla registrada.
- **bool hide_cursor()**
  Oculta el cursor en la pantalla registrada.
- **bool get_timer_event()**
  Retorna el estado del temporizador, verdadero si ha generado un evento el temporizador.
- **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.
- **ALLEGRO_TIMER* get_timer()**
  Retorna el puntero del temporizador para que pueda ser usado con las demás funciones de ALLEGRO.
- **operator ALLEGRO_EVENT_QUEUE* ()**
  Retorna el puntero a la cola de eventos para que pueda ser usado con las funciones restantes de ALLEGRO.

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

2.4.4 Colores (LexRisLogic/Allegro/LL_Color.h)

Contenido:
→ Estructura: **Color** representando a **ALLEGRO_COLOR**.

1. **Estructura Color, Variables y Funciones**

   - **Color**
     Estructura que representa un color con el modelo RGBA (Red, Green, Blue, Alpha).
   - **char Red**
     Representa a la cantidad de color Rojo, 0 a 255.
   - **char Green**
     Representa a la cantidad de color Verde, 0 a 255.
   - **char Blue**
     Representa a la cantidad de color Azul, 0 a 255.
   - **char Alpha**
     Representa al nivel de opacidad del color, 0 a 255.
   - **Color**(**char r=0, char g=0, char b=0, char a=255**)
     Constructor, crea un color de tipo RGBA.
   - **operator ALLEGRO_COLOR**()
     Retorna una conversión de la clase a la estructura **ALLEGRO_COLOR** para que pueda ser utilizado con las funciones restantes de ALLEGRO.
   - **Color operator !** ()
     Retorna el color opuesto, pero con la misma opacidad.
   - **Color operator =** (**ALLEGRO_COLOR Ot**)
     Copia el color de un **ALLEGRO_COLOR** a esta estructura.

2.4.5 Cámara (LexRisLogic/Allegro/LL_Camera.h)

Contenido:
→ Clase: **Camera**.

1. **Clase Camera y Funciones**

   - **Camera**
     Clase que simula la posición de una cámara para las operaciones de dibujo, y controla las escalas a partir del tamaño real en el que se esta trabajando y la pantalla objetivo (**Display**).
   - **Camera**(**ALLEGRO_DISPLAY* display, int rsx, int rsy**)
     Constructor que recibe la pantalla objetivo y el tamaño real, es decir en el que se va a trabajar (rsx*rsy), y pasa a actualizar las escalas.

- **void set_display(ALLEGRO_DISPLAY* display)**
  Cambia la pantalla objetivo y actualiza las escalas.
- **ALLEGRO_DISPLAY*** get_display()
  Retorna un puntero a la pantalla objetivo que toma como referencia para las escalas.
- **void set_realsize(int rsx, int rsy)**
  Cambia el tamaño real en el que se trabaja y actualiza las escalas.
- **display_size_t get_realsizex()**
  retorna el valor del tamaño real del eje x en el que se esta trabajando.
- **display_size_t get_realsizey()**
  retorna el valor del tamaño real del eje y en el que se esta trabajando.
- **void set_cam(pos_t x, pos_t y)**
  Cambia la posición de la cámara a la nueva posición (x,y), será visible todo lo dibujado en pantalla desde las coordenadas (x,y) hasta (x+RealSizeX,y+RealSizeY).
- **void set_camx(pos_t x)**
  Cambia la posición de la cámara del eje x.
- **void set_camy(pos_t y)**
  Cambia la posición de la cámara del eje y.
- **pos_t get_camx()**
  Retorna el posición actual de la cámara en el eje x.
- **pos_t get_camy()**
  Retorna el posición actual de la cámara en el eje y.
- **void plus_x(pos_t v)**
  Aumenta en v la posición de la cámara del eje x.
- **void plus_y(pos_t v)**
  Aumenta en v la posición de la cámara del eje y.
- **void refresh()**
  Actualiza la posición de la cámara y las escalas; esta función debe ser usada si la pantalla objetivo cambia de propiedades sin cambiar el puntero a este.
- **template<typename T>**
  void draw(T* data, bool in=1)
  Función que recibe cualquier clase **T** pero ha de tener las funciones:
  (a) **pos_t get_posx()** → Usado para obtener la posición x del objeto teniendo en cuenta que trabaja conforme al tamaño real que se ha asignado.
  (b) **pos_t get_posy()** → Usado para obtener la posición y del objeto teniendo en cuenta que trabaja conforme al tamaño real que se ha asignado.
  (c) **void set_pos(pos_t nx,pos_t ny)** → Usado para asignarle su verdadera posición conforme al tamaño real que se ha asignado y al tamaño de la pantalla objetivo, y una vez termine de ser dibujado, reasignar su verdadera posición.

(d) **void draw()** → Usado para dibujar al objeto con su nueva posición generada.

La operación de dibujo se puede dar de dos formas con la variable 'in' que significa si el objeto sera dibujado dentro o fuera de la cámara, esto significa que si ignorara la posición actual de la cámara y dibujara donde se halla pedido, o de lo contrario buscarle su posición de acuerdo a la posición de la cámara, por defecto se toma en cuenta la posición de la cámara.

2.4.6 Primitivos (LexRisLogic/Allegro/LL_Primitives.h)

Contenido:
→ Clase: **Point** representando a la función **al_put_pixel()**.
→ Clase: **Figure** siendo una clase base.
→ Clase: **Circle** representando a la función **al_draw_circle()**.
→ Clase: **Ellipse** representando a la función **al_draw_ellipse()**.
→ Clase: **Rectangle** representando a la función **al_draw_rectangle()**.
→ Clase: **Function**.

1. **Clase Point y Funciones**

   - **Point**
     Clase que contiene la información de un punto con su posición y color.
   - **Point()**
     Constructor, crea un punto con la posición inicial (0,0).
   - **Point(int a, int b)**
     Constructor, crea un punto con la posición inicial (a,b).
   - **void set_pos(pos_t xx, pos_t yy)**
     Cambia la posición del punto a (xx,yy).
   - **void set_posx(pos_t xx)**
     Cambia la posición del punto en el eje x.
   - **void set_posy(pos_t yy)**
     Cambia la posición del punto en el eje y.
   - **pos_t get_posx()**
     Retorna la posición del eje x del punto.
   - **pos_t get_posy()**
     Retorna la posición del eje y del punto.
   - **void set_color(ALLEGRO_COLOR Other)**
     Cambia el color del punto por el color Other.
   - **ALLEGRO_COLOR get_color()**
     Retorna el color actual del punto.
   - **void draw()**
     Dibujo el punto en la posición y el color dado.
   - **void draw_in_another_target()**
     Dibuja el punto pero sin uso de las escalas generadas por la clase **Camera**.

2. **Clase Figure y Funciones - Clase Base**

- **Figure**
  Clase base que contiene la información básica de una figura como posición, color, si será relleno y el grosor de línea.
- **void set_pos(pos_t xx, pos_t yy)**
  Cambia la posición de la figura a (xx,yy).
- **void set_posx(pos_t xx)**
  Cambia la posición de la figura en el eje x.
- **void set_posy(pos_t yy)**
  Cambia la posición de la figura en el eje y.
- **pos_t get_posx()**
  Retorna la posición del eje x de la figura.
- **pos_t get_posy()**
  Retorna la posición del eje y de la figura.
- **void set_thickness(float ot)**
  Cambia el grosor de línea.
- **float get_thickness()**
  Retorna el valor del grosor de línea.
- **void set_color(ALLEGRO_COLOR Other)**
  Cambia el color de la figura por el color Other.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual de la figura.
- **void set_is_filled(bool op)**
  Cambiar el valor sobre si la figura estará rellenada.
- **bool is_filled()**
  Retorna si la figura estará rellena.

3. **Clase Circle y Funciones**

- **Circle**
  Clase que hereda de la clase **Figure** y adiciona información de radio y la operación de dibujo para la representación de un círculo.
- **Circle()**
  Constructor, crea un círculo con centro en (0,0) y radio 1, la posición heredada de la clase figura viene a ser el centro del círculo.
- **Circle(pos_t a, pos_t b, float r)**
  Constructor, crea un círculo con centro en (a,b) y radio r.
- **void set_ratio(float ot)**
  Cambia el radio del círculo.
- **float get_ratio()**
  Retorna el radio actual del círculo.
- **void draw()**
  Dibujo el círculo con toda la información guardada.

- **void draw_in_another_target()**
  Dibuja el círculo pero sin uso de las escalas generadas por la clase **Camera**.

4. **Clase Ellipse y Funciones**

   - **Ellipse**
     Clase que hereda de la clase **Figure** y adiciona información de dos radios y la operación de dibujo para la representación de un elipse.
   - **Ellipse()**
     Constructor, crea un elipse con centro en (0,0) y los dos radios con valor 1, la posición heredada de la clase figura viene a ser el centro de la elipse.
   - **Ellipse(pos_t a, pos_t b, float rx, float ry)**
     Constructor, crea un elipse con centro en (a,b), rx como valor del radio en el eje x y ry como valor del radio en el eje y.
   - **void set_ratiox(float ot)**
     Cambia el radio del eje x de la elipse.
   - **float get_ratiox()**
     Retorna el radio actual del eje x de la elipse.
   - **void set_ratioy(float ot)**
     Cambia el radio del eje y de la elipse.
   - **float get_ratioy()**
     Retorna el radio actual del eje y de la elipse.
   - **void draw()**
     Dibujo la elipse con toda la información guardada.
   - **void draw_in_another_target()**
     Dibuja la elipse pero sin uso de las escalas generadas por la clase **Camera**.

5. **Clase Rectangle y Funciones**

   - **Rectangle**
     Clase que hereda de la clase **Figure** y adiciona información de tamaño de la linea en el eje x y el tamaño de la línea en el eje y, y la operación de dibujo para la representación de un rectángulo.
   - **Rectangle()**
     Constructor, crea un rectángulo con posición en (0,0), con 0 como tamaño en el eje x y 0 como tamaño en el eje y.
   - **Rectangle(pos_t a, pos_t b, float tamx, float tamy)**
     Constructor, crea un rectángulo con posición en (0,0), con tamx como tamaño en el eje x y tamy como tamaño en el eje y.
   - **void set_sizex(float ot)**
     Cambia el tamaño en el eje x del rectángulo.
   - **float get_sizex()**
     Retorna el tamaño actual en el eje x del rectángulo.
   - **void set_sizey(float ot)**
     Cambia el tamaño en el eje y del rectángulo.

- **float get_sizey()**
  Retorna el tamaño actual en el eje y del rectángulo.
- **void draw()**
  Dibujo el rectángulo con toda la información guardada.
- **void draw_in_another_target()**
  Dibuja el rectángulo pero sin uso de las escalas generadas por la clase **Camera**.

6. **Clase Function y Funciones**

- **Function**
  Clase que representa una función en R2, que debe tener la forma: **pos_t (*function_name)(pos_t)**; el cual recibe una coordenada en x, para retornar la coordenada y correspondiente.
- **Function(pos_t (*ofx)(pos_t))**
  Constructor, recibe un puntero a función donde evaluar una posición x para retornar f(x) que vendría a ser la coordenada y.
- **void set_pos(pos_t xx, pos_t yy)**
  Cambia la posición del eje principal a (xx,yy), por defecto el origen se encuentra en (0,0) pero con esta función puedes tomar otro punto (x,y) como el origen (0,0).
- **void set_posx(pos_t xx)**
  Cambia la posición x del origen.
- **void set_posy(pos_t yy)**
  Cambia la posición y del origen.
- **pos_t get_posx()**
  Retorna la posición x del origen.
- **pos_t get_posy()**
  Retorna la posición y del origen.
- **void set_thickness(float ot)**
  Cambia el grosor de línea.
- **float get_thickness()**
  Retorna el valor del grosor de línea.
- **void set_color(ALLEGRO_COLOR Other)**
  Cambia el color de la línea de la función por el color Other.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual de la línea de la función.
- **void set_pass(float p)**
  Cambia el paso de la función por p.
- **float get_pass()**
  Retorna el paso actual de la función.
- **void set_init(pos_t x)**
  Cambia el x inicial por el que la función se empezara a evaluar.
- **pos_t get_init()**
  Retorna la posición en el eje x por donde la función se empezara a evaluar.

- **void set_final(pos_t x)**
  Cambia el x final por el que la función se terminara de evaluar.
- **pos_t get_final()**
  Retorna la posición en el eje x por donde la función se terminara de evaluar.
- **void set_function(pos_t (*ofx)(pos_t)**
  Cambia la función con la que se evaluara un x para obtener un y.
- **void draw()**
  Dibuja la función de acuerdo al paso, desde la posición inicial hasta la posición final con todas las configuraciones hechas tomando como origen a la posición.
- **void draw_in_another_target()**
  Dibuja la función pero sin uso de las escalas generadas por la clase **Camera**.

2.4.7 Mapa de Bits (LexRisLogic/Allegro/LL_Bitmap.h)

Contenido:
→ Función: Guardar mapa de bits.
→ Clase: **Bitmap** representando a **ALLEGRO_BITMAP**.
→ Clase: **SubBitmap** representando a la función **al_create_sub_bitmap**.
→ Clase: **Image**.

1. **Función de Guardado de Mapas de Bits**
   bool save_bitmap(string name,ALLEGRO_BITMAP* bmp)
   La función guarda un mapa de bits en el directorio con el nombre y formato que se le den, en caso de éxito retorna verdadero.

2. **Clase Bitmap y Funciones**

   - **Bitmap**
     Clase que se encarga de manejar toda la configuración para su dibujado; también asignarlo como mapa de bits objetivo de las operaciones de dibujo y usar la función **draw_in_another_target()** de todas las clases que tienen operaciones de dibujo a excepción de la clase **Video**, para así ignorar las escalas globales generada por la clase **Camera**.
   - **void set_pos(pos_t xx, pos_t yy)**
     Cambia la posición donde se dibujara el mapa de bits.
   - **void set_posx(pos_t xx)**
     Cambia la posición x del mapa de bits.
   - **void set_posy(pos_t yy)**
     Cambia la posición y del mapa de bits.
   - **pos_t get_posx()**
     Retorna la posición x del mapa de bits.
   - **pos_t get_posy()**
     Retorna la posición y del mapa de bits.

- **float get_sizex()**
  Retorna el tamaño actual en el eje x del mapa de bits.
- **float get_sizey()**
  Retorna el tamaño actual en el eje y del mapa de bits.
- **void set_angle(pos_t an)**
  Cambia el ángulo de la dirección de dibujo del mapa de bits desde el centro, el ángulo debe estar en radianes.
- **pos_t get_angle()**
  Retorna el ángulo actual usado por el mapa de bits.
- **void set_flag(int f)**
  Modo en como se realizara la operación de dibujo del mapa de bits, f puede tomar el valor de 0 para dibujar por defecto o combinaciones de las siguientes macros de ALLEGRO:
  (a) **ALLEGRO_FLIP_HORIZONTAL**
  Dibuja el mapa de bits girando sobre el eje y.
  (b) **ALLEGRO_FLIP_VERTICAL**
  Dibuja el mapa de bits girando sobre el eje x.
- **void set_scalex(pos_t sx)**
  Cambia la escala independiente en el eje x que manejara el mapa de bits.
- **void set_scaley(pos_t sy)**
  Cambia la escala independiente en el eje y que manejara el mapa de bits.
- **pos_t get_scalex()**
  Retorna la escala independiente actual en el eje x que maneja el mapa de bits.
- **pos_t get_scaley()**
  Retorna la escala independiente actual en el eje y que maneja el mapa de bits.
- **void set_target()**
  Cambia el mapa de bits objetivo de las operaciones de dibujo, seleccionando al mapa de bits que llame a este método.
- **bool create(int s_X, int s_Y)**
  Crea un mapa de bits vacío de dimensión s_X y s_Y, eliminando el mapa de bits anterior, retorna verdadero en caso de éxito.
- **bool destroy()**
  Destruye el mapa de bits que actualmente existe, retorna verdadero en caso de éxito.
- **void draw()**
  Dibuja el mapa de bits con toda la configuración dada.
- **void draw_in_another_target()**
  Dibuja el mapa de bits pero sin uso de las escalas generadas por la clase **Camera**.
- **operator ALLEGRO_BITMAP* ()**
  Retorna el puntero **ALLEGRO_BITMAP** para que pueda ser usado con las funciones restantes de ALLEGRO.

- ~ **Bitmap()**
  Destructor. llama a la función destroy() para liberar la memoria usada por el mapa de bits.

3. **Clase SubBitmap y Funciones**

   - **SubBitmap**
     Clase que hereda de la clase **Bitmap**, que aumenta funciones para establecer el padre de donde se creara un sub mapa de bits y funciones para su configuración y creación de sub mapa de bits.
   - **void set_bitmap_parent(ALLEGRO_BITMAP* ft)**
     Establece el mapa de bits padre de donde se creara el sub mapa de bits.
   - **ALLEGRO_BITMAP* get_bitmap_parent()**
     Retorna el mapa de bits padre de donde se creara el sub mapa de bits.
   - **void set_sub_x(pos_t xx)**
     Cambia la posición inicial x de donde se creara el sub mapa de bits a partir del padre.
   - **void set_sub_y(pos_t yy)**
     Cambia la posición inicial y de donde se creara el sub mapa de bits a partir del padre.
   - **pos_t get_sub_sx()**
     Retorna la posición inicial x de donde se creara el sub mapa de bits a partir del padre.
   - **pos_t get_sub_y()**
     Retorna la posición inicial y de donde se creara el sub mapa de bits a partir del padre.
   - **void set_sizex(pos_t xx)**
     Cambia el tamaño en el eje x que tomara el sub mapa de bits del padre.
   - **void set_sizey(pos_t yy)**
     Cambia el tamaño en el eje y que tomara el sub mapa de bits del padre.
   - **bool create_sub_bitmap()**
     Crea un sub mapa de bits a partir del padre asignado, con toda la configuración asignada, retorna verdadero en caso de éxito.

4. **Clase Image y Funciones**

   - **Image** Clase que hereda de la clase **Bitmap**, que aumenta funciones para poder cargar una imagen y convertirla en un mapa de bits para su dibujado.
   - **void set_path(string new_path)**
     Cambia el directorio, nombre y formato de la imagen.
   - **string get_path()**
     Retorna el directorio, nombre y formato de la imagen.

- **bool load()**
  Carga la imagen del directorio, creándola como un mapa de bits para ser usada.
- **bool save()**
  Guarda el mapa de bits sobrescribiendo la imagen anterior, puede cambiarse el directorio y luego guardar para evitar sobrescribir la imagen real.

2.4.8 Texto (LexRisLogic/Allegro/LL_Text.h)

Contenido:
→ Clase: **Font** representando a **ALLEGRO_FONT**.
→ Clase: **Text** representando a la función **al_draw_text()**.

1. **Clase Font y Funciones**

   - **Font**
     Clase que se encarga de cargar una fuente de formato *.ttf para que pueda ser usada para dibujar el texto.
   - **void set_path(string new_path)**
     Cambia el directorio y nombre donde se encuentra la fuente.
   - **string get_path()**
     Retorna el directorio y nombre donde se encuentra la fuente.
   - **void set_size(float Tsize)**
     Cambia el tamaño de la fuente que esta en px.
   - **float get_size()**
     Retorna el tamaño de la fuente que esta en px.
   - **bool load()**
     Carga la fuente con todas las configuraciones dadas.
   - **bool load_for_another_target()**
     Carga la fuente pero ignorando las escalas globales dadas por la clase **Camera**.
   - **void refresh()**
     Actualiza la fuente con las últimas escalas globales dadas por la clase **Camera**.
   - **bool destroy()**
     Destruye la fuente generada.
   - **operator ALLEGRO_FONT*** ()
     Retorna el puntero **ALLEGRO_FONT** para que pueda ser usado con las funciones restantes de ALLEGRO.
   - ∼ **Font()**
     Destructor. llama a la función destroy() para liberar la memoria usada por la fuente.

2. **Clase Text y Funciones**

- **Text**
  Clase el cual maneja la configuración de la posición, mensaje y modo en el que se dibujara el texto en el objetivo.
- **Text(Font* f)**
  Constructor, crea la estructura adicionando la fuente con cual se dibujara el texto.
- **void set_pos(pos_t xx, pos_t yy)**
  Cambia la posición donde se dibujara el texto dado.
- **void set_posx(pos_t xx)**
  Cambia la posición x del texto.
- **void set_posy(pos_t yy)**
  Cambia la posición y del texto.
- **pos_t get_posx()**
  Retorna la posición x del texto.
- **pos_t get_posy()**
  Retorna la posición y del texto.
- **void set_flag(int _flag)**
  Modo en como se realizara la operación de dibujo del texto, _flag pude tomar como valor las siguientes macros de ALLEGRO:
  (a) **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).
  (b) **ALLEGRO_ALIGN_CENTRE**
  Dibuja el texto tomando la posición (x,y) como punto central del texto.
  (c) **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 Other)**
  Cambia el color del texto por el color Other.
- **ALLEGRO_COLOR get_color()**
  Retorna el color actual del texto.
- **void set_font(Font* f)**
  Cambia la fuente que se utilizara para dibujar el texto.
- **void draw()**
  Dibuja el texto con toda la configuración dada.
- **operator const char* ()**
  Retorna una conversión del texto al tipo **const char***.
- **operator string ()**
  Retorna una conversión del texto al tipo **string**.
- **const char* operator = (const char* ot)**
  Operador para cambiar el texto que se mostrara, pasando como parámetro un **const char***.

- **string operator = (string ot)**
  Operador para cambiar el texto que se mostrara, pasando como parámetro un **string**.

2.4.9 Audio (LexRisLogic/Allegro/LL_Audio.h)

Contenido:
→ Clase: **Audio** representando a:
ALLEGRO_SAMPLE y **ALLEGRO_SAMPLE_INSTANCE**.

1. **Clase Audio y Funciones**

   - **Audio**
     Clase que se encarga de manejar toda la configuración para poder reproducir muestras de sonido.
   - **void set_path(string new_path)**
     Cambia el directorio y nombre donde se encuentra la muestra de sonido.
   - **string get_path()**
     Retorna el directorio y nombre donde se encuentra la muestra de sonido.
   - **void set_speed(float sp)**
     Cambia la velocidad de reproducción de la muestra, la velocidad normal toma el valor de 1.
   - **float get_speed()**
     Retorna la velocidad de reproducción de la muestra.
   - **void set_pan(float pn)**
     Cambia la dirección de la reproducción de la muestra. Dar de valor -1.0 para reproducir la muestra sólo a través del altavoz izquierdo; 1.0 para reproducir sólo a través del altavoz derecho; 0.0 para reproducir de forma centralizada. También puede usar la macro de ALLEGRO **ALLEGRO_AUDIO_PAN_NONE** para reproducir sin la técnica *panning*.
   - **float get_pan()**
     Retorna la dirección de la reproducción de la muestra.
   - **void set_volume(float vl)**
     Cambia el volumen con el que se reproducirá la muestra, el volumen normal toma el valor de 1.
   - **float get_volume()**
     Retorna el volumen con el que se reproduce la muestra.
   - **void set_mode(ALLEGRO_PLAYMODE flag)**
     Cambia el modo de reproducción de la muestra, la variable flag puede tomar los siguientes valores de las macros de allegro:
     (a) **ALLEGRO_PLAYMODE_ONCE**
     Reproduce la muestra una sola vez.

(b) **ALLEGRO_PLAYMODE_LOOP**
Reproduce la muestra una y otra vez hasta que la reproducción sea detenida.

(c) **ALLEGRO_PLAYMODE_BIDIR**
Reproduce la muestra normalmente, pero una vez que llegue al final la reproducirá al revés, y al regresar al principio volverá a reproducirse de la misma forma hasta que la reproducción sea detenida.

- **bool load()**
  Carga la muestra de sonido, y le aplica toda la configuración dada; la configuración puede ser modificada a pesar de haber cargado ya la muestra.
- **unsigned int get_size()**
  Retorna la posición final de la muestra.
- **void set_position(unsigned int vl)**
  Cambia la posición de donde se empezara a reproducir la muestra.
- **unsigned int get_position()**
  Retorna la posición actual de la reproducción de la muestra.
- **void stop()**
  Detiene la reproducción de la muestra, cambia la posición a 0.
- **void pause()**
  Pausa la reproducción de la muestra.
- **void play()**
  Reproduce la muestra desde la última posición en la que se quedo.
- **bool destroy()**
  Destruye la muestra y su instancia.
- **operator ALLEGRO_SAMPLE*** ()
  Retorna el puntero **ALLEGRO_SAMPLE** para que pueda ser usado con las funciones restantes de ALLEGRO.
- **operator ALLEGRO_SAMPLE_INSTANCE*** ()
  Retorna el puntero **ALLEGRO_SAMPLE_INSTANCE** para que pueda ser usado con las funciones restantes de ALLEGRO.
- ∼ **Audio**
  Destructor, llama a la función destroy() para liberar la memoria usada por la muestra y su instancia.

2.4.10 Vídeo (LexRisLogic/Allegro/LL_Video.h)

Contenido:
→ Clase: **Video** representando a **ALLEGRO_VIDEO**.

1. **Clase Video y Funciones**

   - **Video**
     Clase que se encarga de manejar toda la configuración para poder reproducir un vídeo y dibujarlo en pantalla.

- **void set_pos(pos_t xx, pos_t yy)**
  Cambia la posición donde se dibujara cada mapa de bits que genere el vídeo.
- **void set_posx(pos_t xx)**
  Cambia la posición x del vídeo.
- **void set_posy(pos_t yy)**
  Cambia la posición y del vídeo.
- **pos_t get_posx()**
  Retorna la posición x del vídeo.
- **pos_t get_posy()**
  Retorna la posición y del vídeo.
- **float get_sizex()**
  Retorna el tamaño actual en el eje x del vídeo.
- **float get_sizey()**
  Retorna el tamaño actual en el eje y del vídeo.
- **void set_angle(pos_t an)**
  Cambia el ángulo de la dirección de dibujo de cada mapa de bits que genere el vídeo, el ángulo debe estar en radianes.
- **pos_t get_angle()**
  Retorna el ángulo actual usado por cada mapa de bits que genere el vídeo.
- **void set_flag(int f)**
  Modo en como se realizara la operación de dibujo de cada mapa de bits que genere el vídeo, puede usar las macros para dibujar mapas de bits.
- **void set_scalex(pos_t sx)**
  Cambia la escala independiente en el eje x que manejara cada mapa de bits que genere el vídeo.
- **void set_scaley(pos_t sy)**
  Cambia la escala independiente en el eje y que manejara cada mapa de bits que genere el vídeo.
- **pos_t get_scalex()**
  Retorna la escala independiente actual en el eje x que maneja cada mapa de bits que genera el vídeo.
- **pos_t get_scaley()**
  Retorna la escala independiente actual en el eje y que maneja cada mapa de bits que genera el vídeo.
- **void set_path(string new_path)**
  Cambia el directorio y nombre donde se encuentra el vídeo.
- **string get_path()**
  Retorna el directorio y nombre donde se encuentra el vídeo.
- **bool load()**
  Carga el vídeo, y le aplica toda la configuración dada; la configuración puede ser modificada a pesar de haber cargado ya el vídeo.

- **void set_position(unsigned int vl)**
  Cambia la posición de donde se empezara a reproducir el vídeo.
- **unsigned int get_position()**
  Retorna la posición de actual de la reproducción del vídeo.
- **void draw()**
  Dibuja el mapa de bits que genero en el momento de la llamada de esta función para ser dibujada con toda la configuración dada.
- **void start()**
  Inicia la reproducción del vídeo.
- **void stop()**
  Detiene la reproducción del vídeo, cambia la posición a 0.
- **void pause()**
  Pausa la reproducción del vídeo.
- **void play()**
  Reproduce el vídeo desde la última posición en la que se quedo.
- **bool destroy()**
  Destruye el vídeo.
- **operator ALLEGRO_VIDEO* ()**
  Retorna el puntero **ALLEGRO_VIDEO** para que pueda ser usado con las funciones restantes de ALLEGRO.
- **∼ Video**
  Destructor, llama a la función destroy() para liberar la memoria usada por el vídeo.

3 ENet (LexRisLogic/ENet/LL_ENet.h)

Cabecera que ofrece una pequeña parte del sistema de ENET, Funciones de inicio y cerrado del sistema, y dos cabeceras para crear una conexión Cliente - Servidor.

3.1 Funciones Globales

- **bool install_enet()**
  Función Principal que inicia el sistema ENET.
- **void uninstall_enet()**
  Función Principal que cierra el sistema ENET.

3.2 Cabeceras

3.2.1 Servidor (LexRisLogic/ENet/LL_Server.h)

Contenido:
→ Clase: **Server** siendo un Host(**ENetHost**) de tipo servidor.

1. **Clase Server y Funciones**

   - **Server**
     Clase que crea un servidor para poder recibir mensajes de clientes y también trasmitir mensajes a los clientes.
   - **bool set_ip(string IP)**
     Cambia la IP donde se alojara el servidor, retorna verdadero en caso de éxito.
   - **string get_ip()**
     Retorna la IP actual donde el servidor está o estará alojado.
   - **bool set_port(unsigned int Port)**
     Cambia el puerto que se abrirá para el túnel de trasmisión de mensajes, retorna verdadero en caso de éxito.
   - **unsigned int get_port()**
     Retorna el puerto actual donde se abrirá o donde está abierto el túnel de transmisión de mensajes.
   - **void set_peer_count(unsigned int peers)**
     Cambia la cantidad de clientes que estarán conectados al servidor como máximo.
   - **unsigned int get_peer_count()**
     Retorna la cantidad de clientes que estarán conectados al servidor como máximo.
   - **void set_service_time(unsigned int ms)**
     Cambia el tiempo de servicio, es decir, el tiempo que esperara el servidor para recibir algún evento generado.
   - **unsigned int get_service_time()**
     Retorna el tiempo de servicio.
   - **bool start_server()**
     Inicia el servidor en la IP dada y abrirá el puerto asignado para el túnel de trasmisión de mensajes con la cantidad máxima de conexiones aceptadas.
   - **bool operator () ()**
     Pregunta durante el tiempo asignado si se generó un evento producidos por los **ENetPeer** que son los clientes del servidor, pueden haber tres tipos de eventos:
     (a) **Conexión**
     Evento generado cuando un cliente se ha conectado con el servidor.
     (b) **Desconexión**
     Evento generado cuando un cliente se ha desconectado del servidor.

(c) **Recepción**
Evento generado cuando un cliente ha enviado un mensaje al servidor, estos mensajes son guardados en una cola que guarda una dupla, el cliente y el mensaje enviado por dicho cliente.

- **bool event_connected()**
  Retorna verdadero si el evento generado es del tipo Conexión.
- **bool event_disconnected()**
  Retorna verdadero si el evento generado es del tipo Desconexión.
- **ENetPeer* get_peer_connected()**
  Retorna el cliente nuevo que se ha conectado al servidor.
- **ENetPeer* get_peer_disconnected()**
  Retorna el cliente que se ha desconectado del servidor.
- **ENetPeer* get_peer_received()**
  Retorna el cliente dueño del mensaje que se encuentra al frente de la cola, si la cola esta vacía retorna un puntero nulo.
- **string get_message_received()**
  Retorna el mensaje que se encuentra al frente de la cola, si la cola esta vacía retorna una cadena vacía.
- **bool remove_message_received()**
  Retira la dupla que se encuentra al frente de la cola, dando paso a la siguiente dupla; retorna verdadero en caso de éxito.
- **void clear()**
  Retira elementos en la cola hasta que quede vacía.
- **bool empty()**
  Retorna verdadero si la cola se encuentra vacía.
- **bool send_message_to_a_client(ENetPeer* Peer,string to_send)**
  Envía un mensaje al cliente representado por el puntero Peer, retorna verdadero si no hubo problema en crear el paquete y enviarlo, pero no es seguro la recepción del paquete.
- **bool send_message_to_all_clients(string to_send)**
  Envía un mensaje a todos los clientes que se encuentren actualmente conectados al servidor, retorna verdadero si no hubo problema en crear el paquete para ser enviado.
- **bool stop_server()**
  Limpia la cola y detiene el servidor iniciado en la IP dada y cierra el puerto asignado.
- **~ Server()**
  Destructor, llama a la función stop_server() para liberar memoria usada por el servidor.

3.2.2 Cliente (LexRisLogic/ENet/LL_Client.h)

Contenido:
→ Clase: **Client** siendo un Host(**ENetHost**) de tipo cliente.

1. **Clase Client y Funciones**

   - **Client**
     Clase que crea una conexión hacia un servidor para poder enviar mensajes a este.
   - **bool set_ip(string IP)**
     Cambia la IP del servidor donde se conectara el cliente.
   - **string get_ip()**
     Retorna la IP del servidor.
   - **bool set_port(unsigned int Port)**
     Cambia el puerto del servidor, es decir, que puerto abrió el servidor para que el cliente pueda trasmitir sus mensajes.
   - **unsigned int get_port()**
     Retorna el puerto del servidor.
   - **void set_service_time(unsigned int ms)**
     Cambia el tiempo de servicio, es decir, el tiempo que esperara el cliente para recibir algún evento generado.
   - **unsigned int get_service_time()**
     Retorna el tiempo de servicio.
   - **bool start_client()**
     Inicia el Host propio del cliente con una sola conexión permitida para el servidor donde se conectará.
   - **bool connect_to_server()**
     El Host se conectara a un servidor abierto con la IP dada y que el servidor tenga el puerto asignado abierto, que es por donde se enviarán los mensajes, retorna verdadero en caso de éxito.
   - **bool get_status()**
     Retorna el estado actual de la conexión al servidor, retorna verdadero si existe dicha conexión, si se pierde la conexión puede tratar de reconectar al servidor usando la anterior función.
   - **bool operator () ()**
     Pregunta durante el tiempo asignado si se generó un evento producido por el servidor, los eventos controlan el estado de la conexión y si se ha recibido un mensaje del servidor.
   - **string get_message_received()**
     Retorna el mensaje que se encuentra al frente de la cola, si la cola esta vacía retorna una cadena vacía.
   - **bool remove_message_received()**
     Retira el mensaje que se encuentra al frente de la cola, dando paso al siguiente mensaje; retorna verdadero en caso de éxito.

- **void clear()**
  Retira elementos en la cola hasta que quede vacía.
- **bool empty()**
  Retorna verdadero si la cola se encuentra vacía.
- **bool send_message_to_the_server(string to_send)**
  Envía al servidor un mensaje; retorna verdadero si no hubo problema en crear el paquete y enviarlo, pero no es seguro la recepción del paquete.
- **bool stop_client()**
  Limpia la cola, desconecta al cliente del servidor y detiene el Host creado para él.
- **∼ Client()**
  Destructor, llama a la función stop_client() para liberar memoria usada por el cliente.

4 Cabeceras Extras (LexRisLogic/LL_...)

4.1 Conversión de Datos (LexRisLogic/LL_Convert.h)

Contenido:
→ Función: Tipo **T** a **string**.
→ Función: **string** a **int**.
→ Función: **string** a **float**.
→ Función: **string** a **double**.

1. **Función de conversión de cualquier tipo de dato a cadena.**
   template<typename T>
   string to_string(T data)
   Convierte el objeto a una cadena, para poder convertir el objeto es necesario tener sobrecargado el operador <<(**ostream**, **T**).

2. **Función de conversión de cadena a entero.**
   Convierte la cadena a un entero, la cadena solo debe contener los caracteres numéricos para poder ser convertida.

3. **Función de conversión de cadena a punto flotante simple.**
   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.

4. **Función de conversión de cadena a punto flotante doble.**
   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.

4.2 Encriptador (LexRisLogic/LL_Encryptor.h)

Contenido:
→ Macro: Diccionario por defecto.
→ Clase: **Encryptor**.

1. **Macro: DEFAULT_DICTIONARY**
   Diccionario de caracteres por defecto que usará el encriptador de cadenas.
2. **Clase Encryptor y Funciones**
   - **Encryptor**
     Clase que codifica una cadena en un mensaje encriptado que también con la misma clase puede ser decodificado.
   - **bool add_new_key(string nk)**
     Adiciona una nueva llave para poder codificar la cadena, se valida la llave de acuerdo al diccionario; 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(string dc)**
     Cambia el diccionario, esto eliminara la llaves generadas que tiene el encriptador.
   - **string get_dictionary()**
     Retorna el diccionario actual que esta usando el encriptador.
   - **string encrypt(string word)**
     Codifica el mensaje 'word' y retorna el mensaje encriptado.
   - **string decrypt(string encrypted_word)**
     Decodifica un mensaje encriptado y retorna el mensaje real, si se cambian las llaves puede que retorne otro mensaje encriptado del mensaje encriptado pero de manera inversa.
   - **~ Encryptor()**
     Destructor, llama a la función clear_keys() parar liberar la memoria usada por el encriptador.

4.3 Lector de Archivos (LexRisLogic/LL_FileStream.h)

Contenido:
→ Clase: **FileStream**.

1. **Clase FileStream y Funciones**
   - **FileStream**
     Clase que proporciona un lector de archivos de texto plano, y funciones para controlar línea a línea el archivo.
   - **void set_path(string new_path)**
     Cambia el directorio y nombre donde se encuentra el archivo.

- **string get_path()**
  Retorna el directorio y nombre donde se encuentra el archivo.
- **bool load()**
  Carga todas las lineas del archivo en la estructura para que pueda ser manejada con las funciones de control línea a línea.
- **bool reload()**
  Recarga el archivo eliminando todos los cambios hechos.
- **bool save()**
  Guarda el archivo modificado, puede cambiar el directorio y nombre del archivo para evitar sobrescribir el archivo original.
- **void clear_file()**
  Limpiar el archivo eliminando todas las líneas del archivo.
- **bool insert_line(unsigned int pos, unsigned int n_lines)**
  Inserta antes de la posición de línea dada un total de n_lines, es decir, n líneas vacías; retorna verdadero en caso de éxito.
- **bool remove_line(unsigned int line)**
  Remueve la línea que se encuentra en la posición dada en el parámetro line, retorna verdadero en caso de éxito.
- **unsigned int size()**
  Retorna el número de líneas que tiene actualmente el archivo.
- **string& operator [] (unsigned int line)**
  Retorna la línea que se encuentra en la posición dada por el parámetro line, retorna por referencia para que pueda ser manipulada la cadena que se encuentra en tal línea.
- **∼ FileStream()**
  Destructor, llama a clear_file() para liberar la memoria usada por el Lector de Archivos.

4.4 Matemática (LexRisLogic/LL_Math.h)

Contenido:
→ Función: Módulo Discreto.
→ Función: Módulo Continuo.
→ Macro: Valor de PI.
→ Función: Conversión de grados sexagesimales a radianes.
→ Función: Conversión de radianes a grados sexagesimales.
→ Función: Generador de Semillas.
→ Función: Cambiar semilla para generación de números aleatorios.
→ Función: Generador de números aleatorios en un rango dado.
→ Función: Intersección de dos segmentos en una dimensión.
→ Clase: **Dot**.
→ Función: Sobrecarga del Operador << para mostrar un **Dot** en consola.
→ Función: Distancia Euclidiana entre dos **Dot's**.

1. **Función para hallar el módulo (Matemática Discreta).**
   template<typename T>
   T mod(T a,T b)
   Retorna el valor de $amodb$, usado para saber el residuo de la operación a/b, o el valor que debería tomar a en un rango discreto de 0 a $b-1$.

2. **Función para hallar el módulo (Matemática Continua).**
   template<typename T>
   T range_mod(T a,T b)
   Se mantiene la idea del módulo para ser usado en rangos discretos pero ahora con rangos continuos, por ejemplo $3.4 mod 3.3$ 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 retornado que tomara a en base a b.

3. **Macro: MATH_PI**
   Macro que contiene el valor de PI.

4. **Función para convertir de grados sexagesimales a radianes**
   float sexagesimal_to_radian(float sexagesimal)
   Convierte los grados sexagesimales enviados por parámetro a radianes.

5. **Función para convertir de radianes a grados sexagesimales**
   float radian_to_sexagesimal(float radian)
   Convierte los radianes enviados por parámetro a grados sexagesimales.

6. **Función Generadora de Semillas**
   uint64_t rdtsc()
   Retorna una semilla usando la instrucción RDTSC de procesadores x86.

7. **Función para cambiar la semilla para la generación de números aleatorios**
   void set_new_seed()
   Cambia la semilla para generar números aleatorios con la función rand().

8. **Función para generar un número entero aleatorio en un rango**
   int random(int mini, int maxi)
   Retorna un número aleatorio X tal que $mini \leq X < maxi$.

9. **Función para manejar la intersección de dos segmentos**
   bool intersection_of_segments()
   Usado para conocer si dos segmentos en una dimensión tienen intersección, puede controlarse en varias dimensiones solo preguntado por los segmentos en cada eje y encontrar la intersección de estos para conocer la intersección de un bloque con otro en dimensión n.

10. **Clase Dot y Funciones**

    - **template<int N>**
      Dot
      Clase que representa a un punto de dimensión N.
    - **Dot()**
      Constructor, crea el punto con valor nulo, es decir todas las coordenadas en 0.
    - **int get_dimension()**
      Retorna la dimensión a la que pertenece el punto.
    - **float& operator [] (unsigned int i)**
      Retorna la i-ésima coordenada del punto para que pueda ser modificada.
    - **Dot<N>& operator = (Dot<N> ot)**
      Copia la información de un punto 'ot' de igual dimensión al punto que invoque este operador.
    - **bool& operator == (Dot<N> ot)**
      Retorna verdadero si el punto 'ot' de igual dimensión es igual al punto que invoca este operador.
    - **~ Dot()**
      Destructor, libera la memoria usada por el punto.

11. **Función para mostrar en consola un Dot**
    template<int N>
    ostream& operator << (ostream& os, Dot<N> dot)
    Muestra el punto en consola de la forma $'[a_0, a_1, a_2, ..., a_{N-1}]'$.

12. **Función para hallar la distancia euclidiana entre dos Dot's**
    template<int N>
    double euclidean_distance(Dot<N> one, Dot<N> two)
    Retorna la distancia euclidiana entre dos puntos que tengan la misma dimensión.

4.5 Separador de Cadenas (LexRisLogic/LL_String_Splitter.h)

Contenido:
→ Clase: **StringSplitter**.

1. **Clase StringSplitter y Funciones**

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

- **void set_string(string n_str)**
  Cambia la cadena que se separara por la clase.
- **string get_string()**
  Retorna la cadena que se separara o a 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 sub-cadenas son guardadas en la estructura interna de la clase, retorna verdadero en caso de éxito.
- **unsigned int size()**

- **string operator [] (unsigned int i)**
  Retorna la i-ésima separación hecha por el separador sobre la cadena asignada.
- **~ StringSplitter()**
  Destructor, destruye la estructura interna que guarda todas las sub-cadenas generadas al separar la cadena.

4.6 Temporizador (LexRisLogic/LL_Time.h)

Contenido:
→ Clase: **Chronometer**.

1. **Clase Chronometer y Funciones**

   - **Chronometer**
     Clase que representa al uso de un cronómetro para conocer el tiempo en segundos que pasa al realizar alguna actividad.
   - **bool play()**
     Inicia el Cronómetro.
   - **bool pause()**
     Pause el Cronómetro.
   - **bool stop()**
     Detiene el Cronómetro.
   - **double get_time()**
     Obtiene el tiempo acumulado por cada pausa que se hizo o si se detuvo poder obtener el tiempo final que se obtuvo.
   - **bool clear()**
     Reinicia el cronómetro limpiando los datos guardados, puede ser usado en cualquier estado del cronómetro.

5 Notas

Toda esta documentación es dada para que se pueda conocer sobre el funcionamiento que brindan de las Cabeceras de LexRis Logic, dirigida para la Versión v1.00 de estas cabeceras.