REPORTE DE ESTADO

Trabajo Práctico “Laberintos y Fantasmas”

Grupo Pixelados

Cátedra:

Algoritmos y Estructuras de Datos

Integrantes:

- Ibarra Sánchez, Ludmila Daiana
- Maranzana, Rodrigo Ezequiel
- Ermasi, Franco
- Masino, Carlos Nicolas

Fecha: 07/10/2025

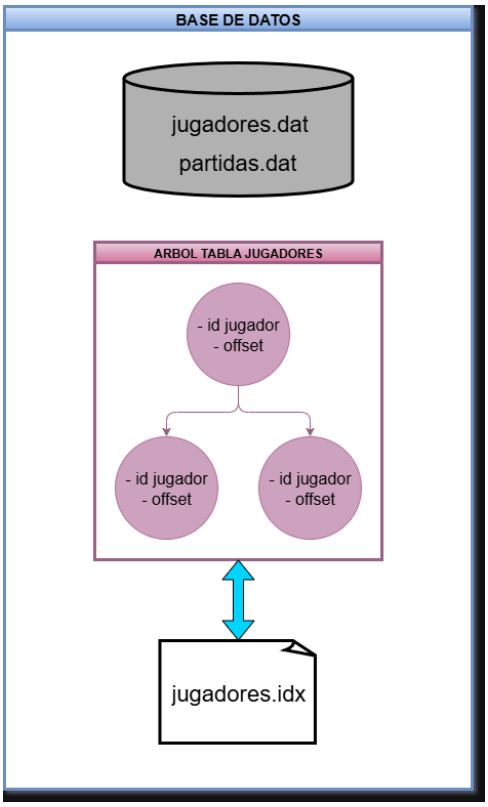
Funcionalidades actuales del Cliente:

Nuestro grupo tomó la decisión de utilizar la biblioteca gráfica SDL y sus derivadas SDL_mixer (mezclador de audio), SDL_ttf (soporte TrueType) y SDL_image (decodificador de imagenes).

Las siguientes funcionalidades se encuentran completas y funcionando:

- Lectura de configuraciones iniciales desde un archivo de texto.
- Generación del laberinto utilizando el algoritmo DFS (Deep-First Search) y post procesado, para asegurar la jugabilidad y controlar la dificultad.
- Colocación de puerta de entrada y salida, asegurando su conexión con el laberinto.
- Colocación de premios y vidas extra, puestas en lugares estratégicos y su recolección por parte del jugador, junto a su otorgamiento respectivo de punto/vida.
- Utilización de semillas que permiten generar partidas y sus sucesivas rondas exactamente iguales.
- Colocación de hasta la cantidad de fantasmas indicada en el archivo de configuración, variando ronda a ronda entre 1 fantasma y el máximo establecido.
- Cuatro tipos de fantasmas con sus respectivas variaciones de búsqueda del jugador, utilizando intercalación entre el algoritmo DFS (Deep-First Search) y BFS (Breadth-First Search), siendo el último mucho más agresivo aumentando así drásticamente la dificultad del juego. Cada fantasma posee su propio color para su fácil detección.
- Encolado de los movimientos del jugador y los fantasmas, con su posterior función de procesado de movimientos.
- Registro de los movimientos del jugador realizados en cada ronda y su posterior muestra por consola.
- Menú principal con las opciones de Nueva Partida, Continuar, Cambiar Usuario, Estadísticas y Salir (Cambiar Usuario y Estadísticas aún en progreso)
- Ingreso de nombre de Usuario, al iniciar el juego, mediante una ventana con un campo de texto realizado con SDL.
- Servicio de conexión socket tanto en cliente como el servidor.
- Servicio de base de datos en el servidor, que permite la creación de tablas, inserción de registros y selección de los mismos. Cada tabla se guarda en su respectivo archivo .dat e implementa la búsqueda de los registros mediante árboles de búsqueda binaria o escaneo completo de tabla (cuando se busca por un campo que no es clave primaria). Los árboles persisten en sus respectivos archivos índices. El servicio de base de datos acepta comandos de texto similares a los de SQL, y retorna respuestas de texto, y en caso de selección de registros un envío extra en formato binario. El cliente opera con dos tablas, jugadores y partidas.

Diseño de la Base de Datos del Servidor



Funcionalidades actuales del Servicio de Base de Datos:

El servicio de base de datos es un sistema que permite la creación de tablas genéricas, según las diferentes solicitudes que se realicen hacia el servidor. Todas las tablas cuentan con un índice implementado con árbol de búsqueda binaria. Este contiene como clave el valor del campo PK. Cuando se solicite una operación que necesite realizar una búsqueda, sobre un campo que no sea clave primaria, el servicio de base de datos realiza un escaneo completo de tabla. El sistema gestiona de manera automática la apertura y cierre de tablas existentes.

La comunicación entre cliente-servidor se lleva a cabo mediante comandos de texto. El servicio de base de datos tiene comandos (CREAR, INSERTAR, ACTUALIZAR, SELECCIONAR), y restricciones (PK, AI) y tipos de datos (ENTERO, TEXTO).

En el caso de SELECCIONAR, si la consulta arroja resultados, los registros se envían (luego de un mensaje de texto convencional) en formato binario con la misma composición y orden que el Cliente haya solicitado al crear la tabla.

La siguiente lista incluye ejemplos de las solicitudes admitidas por el servidor:

- CREAR:
    - CREAR jugadores (username TEXTO(16) PK, puntajeMax ENTERO, cantPartidas ENTERO)
    - CREAR partidas (idPartida ENTERO PK AI, puntaje ENTERO, cantMovs ENTERO)

- INSERTAR:
    - INSERTAR jugadores (nombre PEPE, puntajeMax 15)

- ACTUALIZAR:
    - ACTUALIZAR jugadores (puntajeMax 25) DONDE idJugador IGUAL 21

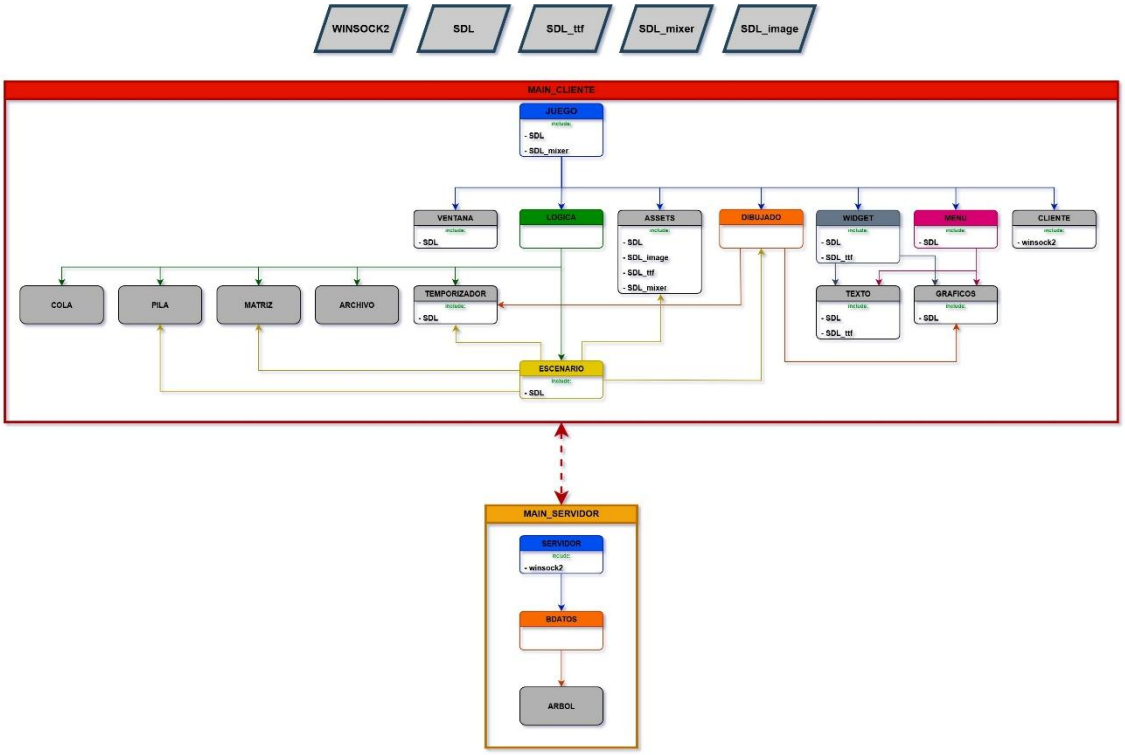
- SELECCIONAR:
    - SELECCIONAR jugadores DONDE username IGUAL PEPE
    - SELECCIONAR partidas DONDE username IGUAL PEPE

Bibliotecas del Cliente:

- **JUEGO**: Controla la creación e inicialización de los componentes de SDL necesarios. Gestiona el game loop y los respectivos llamados a las funciones que conforman tanto a la lógica, detección de inputs, menús, comunicación con el socket, reproducción de efectos de audio y llamado a renderizado.
- **LOGICA**: Controla, como su nombre indica, la lógica que Laberintos y Fantasmas debe seguir. Gestiona los llamados a funciones generadoras de la partida, los movimientos del jugador, de los fantasmas, el otorgamiento de vidas, puntos, mientras regla las interacciones.
- **ASSETS**: Carga desde el disco los archivos de imagen, audio y fuentes TrueType necesarios para la ejecución del juego.
- **DIBUJADO**: Se encarga del Algoritmo del Pintor para la correcta muestra del mundo y las entidades que la componen. Como el juego utiliza perspectiva Top Down, se requiere un dibujado desde la fila menor, columna a columna, hasta la fila mayor, intercalando el dibujado de las entidades para la generación del efecto de 2.5D.
- **ESCENARIO**: Gestiona los algoritmos de generación del laberinto y la inicialización del escenario de juego.
- **CLIENTE**: Gestiona la comunicación con el servidor para la inserción y recupero de los jugadores y sus respectivas partidas utilizando la biblioteca winsock2.
- **VENTANA**: Permite la generación de ventanas graficas de SDL genéricas dentro de la propia ventana del juego.
- **WIDGET**: Permite la creación de utilidades como campos de texto, contadores numéricos gráficos, gráficos de barra, etc. para la muestra y/o recolección de información durante el juego y la partida.
- **MENU**: Permite la creación de menús gráficos interactivos y personalizables.
- **TEXTO**: Gestiona la creación de texturas a partir de strings y fuentes TrueType.
- **GRAFICOS**: Gestiona la escritura de texturas sobre el framebuffer de SDL.

- **TEMPORIZADOR**: Gestiona temporizadores (no son multihilo) que permiten ejecutar acciones de poca precisión como la animación de las entidades y entornos y la espera no bloqueante.
- **ARCHIVO**: Funciones para la lectura, parseo y escritura del archivo de configuración.
- **PILA, COLA, ARBOL**: Funciones de sus respectivos TDA.

Diagrama de Bloques



Nuestro proyecto se divide en módulos con responsabilidades únicas. Algunos módulos son APIs, otros un compendio de funciones relacionadas, y otros son TDAs vistos en la materia.

Como la cantidad de funciones que hemos desarrollado es significativa, realizar una descripción de cada una haría que este informe sea innecesariamente extenso. Hemos utilizado nombres de funciones autodocumentados y descriptivos por lo que, en conjunto con la descripción de cada módulo, esperamos sea suficientemente explicativo.

DESCRIPCIÓN MODULOS CLIENTE:

ARCHIVO:

Contiene funciones dedicadas a la manipulación (lectura, parseo, escritura) del archivo de configuración y contingencia.

ASSETS:

Contiene funciones dedicadas a la carga de assets (recursos) del juego como las imágenes, sonidos y fuentes, así como las funciones de destrucción de los mismos.

CLIENTE:

Contiene las funciones necesarias para establecer la comunicación con el servidor, desde la creación, conexión y cierre del socket, así como aquellas que emiten solicitudes y recepcionan las respuestas.

DIBUJADO:

Contiene funciones complementarias que se encargan de la interpretación del escenario del juego y sus entidades para luego renderizarlas según nuestro propio “Algoritmo del Pintor”.

ESCENARIO:

Contiene las funciones que generan el escenario del juego. Realiza la inicialización de los elementos necesarios para la generación pseudoaleatoria del laberinto, su post procesamiento y la colocación de puertas.

GRAFICOS:

Contiene las funciones que permiten graficar en el renderer de SDL.

JUEGO:

Contiene las funciones principales que orquestan y ordenan el juego en sí, similar a un motor de juego. Sus funciones se encargan de determinar desde la inicialización del juego, su continua actualización y recepción de inputs, cambios de estado, así como su posterior destrucción.

LOGICA:

Contiene las funciones que controlan y dirigen los comportamientos y reglas que “Laberintos y Fantasmas” debe respetar. Se encarga de la inicialización de la lógica, así como el procesamiento de turnos, movimientos del jugador y los fantasmas, inicio de partida, cambio de ronda, y algoritmos de búsqueda por parte de los fantasmas.

MENU:

Contiene funciones que permiten la creación y manipulación de menús.

TEMPORIZADOR:

Contiene funciones que permiten la creación, actualización, y manipulación de temporizadores (no multihilos).

TEXTO:

Contiene funciones que permiten la creación de texturas a partir de fuentes TTF.

VENTANA:

Contiene funciones que permiten la creación de ventanas contenedoras genéricas, y su posterior manipulación, apertura, y cierre.

WIDGET:

Contiene funciones que permiten la creación, actualización y manipulación de widgets (utilidades principalmente prácticas para la interfaz de usuario HUD)

FUNCIONES MODULOS CLIENTE:

ARCHIVO:

Funciones Declaradas

- int archivo_leer_conf(FILE* arch, tConf *conf);
- int archivo_escribir_conf(FILE* arch, const tConf *conf);
- int archivo_escribir_escenario(tEscenario *escenario, int numRonda, long semillaRonda);

Funciones Internas:

- static int _archivo_parsear_linea_conf(char *buffer, tParam *param);

ASSETS:

Funciones Declaradas

- int assets_cargar_imagenes(SDL_Renderer *renderer, SDL_Texture **imagenes);
- int assets_cargar_sonidos(Mix_Chunk **sonidos);
- int assets_cargar_fuente(TTF_Font **fuente, int tamFuente);
- void assets_destuir_imagenes(SDL_Texture **imagenes);
- void assets_destuir_sonidos(Mix_Chunk **sonidos);
- void assets_destruir_fuente(TTF_Font *fuente);

CLIENTE:

Funciones Declaradas

- int cliente_inicializar();
- SOCKET cliente_conectar_servidor(const char *ipServidor, int puerto);
- int cliente_enviar_solicitud(SOCKET sock, const char *solicitud);
- void cliente_cerrar_conexion(SOCKET sock);
- int cliente_recibir_respuesta(SOCKET sock, char *respuesta, int tamBuffer);
- int cliente_recibir_datos(SOCKET sock, char *bufferDatos, int bytesEsperados);
- int cliente_ejecutar_solicitud(SOCKET sock, const char *solicitud, int* cantRegistros, char** bufferDatos);

DIBUJADO:

Funciones Declaradas

- void dibujado_escenario(SDL_Renderer *renderer, tEscenario *escenario, SDL_Texture **imagenes);

Funciones Internas:

- static void _dibujado_jugador_aturdido(SDL_Texture *textura);
- static void _dibujado_fantasma_aturdido(SDL_Texture *textura);
- static void _dibujado_entidad(SDL_Renderer *renderer, SDL_Texture **imagenes, tEntidad *entidad);
- static void _dibujado_jugador_potenciado(SDL_Texture *textura);

ESCENARIO:

Funciones Declaradas

- int escenario_crear(tEscenario *escenario, unsigned columnas, unsigned filas);
- void escenario_generar(tEscenario *escenario);
- void escenario_destruir(tEscenario *escenario);
- int escenario_calcular_mascara(tEscenario *escenario, int columna, int fila);
- eParedLimite escenario_ubic_es_pared_limite(const tEscenario *escenario, tUbicacion ubic);

Funciones Internas:

- static void _escenario_init_tiles(tTile *tiles);
- static void _escenario_generar_laberinto(tEscenario *escenario);
- static void _escenario_postprocesar(tEscenario *escenario);
- static void _escenario_calcular_puerta(tCasilla *puerta, tTile *tiles, int esEntrada, int pared, int cantFilas, int cantColumnas);

- static void _escenario_colocar_puertas(tEscenario *escenario);
- static void _escenario_despejar_puerta(tEscenario *escenario, tUbicacion ubicPuerta, int pared);
- static void _escenario_liberar_casilla(tEscenario *escenario, int fila, int columna);

GRAFICOS:

Funciones Declaradas

- SDL_Texture* graficos_crear_textura(SDL_Renderer *renderer, int ancho, int alto, int modoAcceso, unsigned formato);
- void graficos_dibujar_textura(SDL_Texture *textura, SDL_Renderer *renderer, SDL_Rect *rectFuente, SDL_Rect *rectDestino, tEfectoGrafico efecto);
- void graficos_dibujar_borde(SDL_Renderer *renderer, SDL_Rect rect, SDL_Color color);
- void graficos_dibujar_relleno(SDL_Renderer *renderer, SDL_Rect rect, SDL_Color color);

JUEGO:

Funciones Declaradas

- int juego_inicializar(tJuego *juego, const char *tituloVentana);
- int juego_ejecutar(tJuego *juego);
- void juego_destruir(tJuego *juego);

Funciones Internas:

- static int _juego_crear_ventana(SDL_Window **ventana, SDL_Renderer **renderer, unsigned anchoRes, unsigned altoRes, const char *tituloVentana);
- static void _juego_renderizar(SDL_Renderer *renderer, SDL_Texture **imagenes, tLogica *logica, tVentana *ventanaMenu, tVentana *ventanaUsername, tHud *hud);
- static void _juego_iniciar_partida(void* datos);
- static void _juego_salir_del_juego(void* datos);
- static void _juego_continuar_partida(void* datos);
- static int _juego_actualizar_hud(tHud *hud, tLogica *logica);
- static int _juego_crear_hud(tJuego *juego);
- static int _juego_cargar_assets(tJuego *juego);
- static void _juego_manejar_input(tJuego *juego, SDL_Keycode tecla);
- static void _juego_manejar_eventos(tJuego *juego);
- static int _juego_ventana_menu_crear(void *datos);
- static void _juego_ventana_menu_actualizar(SDL_Event *evento, void *datos);
- static void _juego_ventana_menu_dibujar(void *datos);
- static void _juego_ventana_menu_destruir(void *datos);
- static int _juego_ventana_usuario_crear(void *datos);
- static void _juego_ventana_usuario_actualizar(SDL_Event *evento, void *datos);
- static void _juego_ventana_usuario_dibujar(void *datos);
- static void _juego_ventana_usuario_destruir(void *datos);

LOGICA:

Funciones Declaradas

- int logica_inicializar(tLogica *logica);
- void logica_destruir(tLogica *logica);
- void logica_calc_min_res(const tLogica *logica, unsigned *anchoRes, unsigned *altoRes);
- int logica_procesar_turno(tLogica *logica, SDL_Keycode tecla);
- void logica_procesar_movimientos(tLogica *logica);
- int logica_iniciar_juego(tLogica *logica);
- int logica_nueva_ronda(tLogica *logica);
- void logica_fin_juego(tLogica *logica);
- void logica_mostrar_historial_movs(tCola *movsJugador);
- void logica_actualizar(tLogica *logica);

Funciones Internas:

- static int logica_ubicacion_valida(const tEscenario *escenario, tUbicacion ubic);
- static tUbicacion _logica_mover_fantasma_bfs(tEscenario *escenario, tEntidad *jugador, tEntidad *fantasma);
- static tUbicacion _logica_mover_fantasma_dfs(tEscenario *escenario, tEntidad *jugador, tEntidad *fantasma);
- static void _logica_colocar_fantasmas(tLogica *logica);
- static void _logica_inicializar_jugador(tEntidad *jugador);
- static void _logica_colocar_jugador(const tEscenario *escenario, tEntidad *jugador);
- static int _logica_fantasma_debe_usar_bfs(const tEntidad *fantasma);
- static void _logica_actualizar_entidad(tEntidad *entidad);
- static int _logica_encontrar_casilla_libre(tLogica *logica, tUbicacion *ubic);

MENU:

Funciones Declaradas

- tMenu* menu_crear(SDL_Renderer *renderer, unsigned capOpc, SDL_Point ubicacion, eMenuTipo menuTipo);
- void menu_destruir(tMenu* menu);
- int menu_agregar_opcion(tMenu *menu, int id, SDL_Texture *textura, unsigned tamAltura, tMenuAccion accion, eOpcionEstado estado);
- void menu_siguiente_opcion(tMenu *menu);
- void menu_anterior_opcion(tMenu *menu);
- tMenuAccion menu_confirmar_opcion(tMenu *menu);
- void menu_estado_opcion(tMenu *menu, int id, eOpcionEstado nuevoEstado);
- void menu_dibujar(tMenu* menu);

TEMPORIZADOR:

Funciones Declaradas

- void temporizador_inicializar(tTempor *tempor, float duracion);
- void temporizador_actualizar(tTempor* tempor);
- eTemporEstado temporizador_estado(const tTempor* tempor);
- void temporizador_iniciar(tTempor* tempor);
- void temporizador_pausar(tTempor* tempor);
- void temporizador_reanudar(tTempor* tempor);
- float temporizador_tiempo_restante(tTempor* temporizador);

TEXTO:

Funciones Declaradas

- SDL_Texture* texto_crear_textura(SDL_Renderer *renderer, TTF_Font *fuente, const char* texto, SDL_Color color);
- int texto_obtener_tam(TTF_Font *fuente, const char *texto, SDL_Point *tamTexto);

VENTANA:

Funciones Declaradas

- tVentana* ventana_crear(SDL_Renderer *renderer, tVentanaAccion accion, SDL_Rect dimensiones, SDL_Color cFondo, char conSombra);
- void ventana_destruir(tVentana *ventana);
- void ventana_abrir(tVentana *ventana);
- void ventana_cerrar(tVentana *ventana);
- void ventana_actualizar(tVentana *ventana, SDL_Event *event);
- void ventana_dibujar(tVentana *ventana);

Funciones Internas:

- static float _interpolacion_lineal(float valorInicial, float valorFinal, float factor);
- static SDL_Rect _ventana_dibujar(tVentana *ventana);

WIDGET:

Funciones Declaradas

- tWidget* widget_crear_campo_texto(SDL_Renderer *renderer, SDL_Point coords, TTF_Font *fuente, SDL_Color cTexto, SDL_Color cFondo, char visible);
- tWidget* widget_crear_boton_texto(SDL_Renderer *renderer, SDL_Point coords, const char* texto, TTF_Font *fuente, SDL_Rect rect, SDL_Color cTexto, SDL_Color cFondo, char visible);
- tWidget* widget_crear_contador(SDL_Renderer *renderer, SDL_Point coords, SDL_Texture *icono, TTF_Font *fuente, SDL_Color cTexto, int valor, char visible);
- tWidget* widget_crear_texto(SDL_Renderer *renderer, const char *texto, SDL_Point coords, TTF_Font *fuente, SDL_Color cTexto, SDL_Color cFondo, char visible);
- int widget_modificar_valor(tWidget *widget, void *valor);
- void widget_modificar_visibilidad(tWidget *widget, char visible);
- void widget_alternar_visibilidad(tWidget *widget);
- void widget_dibujar(tWidget *widget);
- void widget_destruir(tWidget *widget);

Funciones Internas:

- static tWidget* _widget_crear_base(SDL_Renderer *renderer, SDL_Point coords, TTF_Font *fuente, const char *texto, SDL_Color cTexto, eWidgetTipo tipo, char visible);
- static int _widget_campo_texto_actualizar(tWidget *widget);
- static void _widget_campo_texto_dibujar(tWidget *widget);
- static int _widget_boton_actualizar(tWidget *widget);
- static void _widget_boton_dibujar(tWidget *widget);
- static void _widget_boton_destruir(tWidget *widget);
- static int _widget_contador_actualizar(tWidget *widget);
- static void _widget_contador_dibujar(tWidget *widget);
- static int _widget_texto_actualizar(tWidget *widget);
- static void _widget_texto_dibujar(tWidget *widget);

DESCRIPCIÓN MÓDULOS SERVIDOR:

BDATOS:

Contiene las funciones de Base de Datos. Estas son utilizadas por el servidor para proporcionar el servicio, y permiten la creación, inserción, manipulación, actualización y selección en tablas (archivos binarios).

SERVIDOR:

Contiene las funciones que inicializan el socket y procesan solicitudes, derivando estas al servicio de base de datos interno.

FUNCIONES MÓDULOS SERVIDOR:

BDATOS:

Funciones Declaradas

- int bdatos_iniciar(tBDatos *bDatos);
- int bdatos_procesar_solcitud(tBDatos *bDatos, const char *solicitud, tLista *listaDatos, int *cantRegistrosDatos, int *tamRegistroDatos);
- eSimbolo bdatos_parsear_comando(tSecuencia *secuencia);
- int bdatos_apagar(tBDatos *bDatos);
- int bdatos_insertar(tBDatos *bDatos);
- int bdatos_actualizar(tBDatos *bDatos);
- int bdatos_seleccionar(tBDatos *bDatos);
- const char* bdatos_obtener_mensaje(eBDRetorno codigoError);
- int bdatos_cargar_idx(tBDatos *bDatos);

Funciones Internas:

- static int _bdatos_cerrar_tabla(tBDatos *bDatos);
- static int _bdatos_insertar(tBDatos *bDatos);
- static int _bdatos_seleccionar(tBDatos *bDatos, tLista *listaDatos, int *cantRegistrosDatos);
- static int _bdatos_tabla_existe(const char *nombreTabla);
- static int _bdatos_crear_tabla(tBDatos *bDatos, const char *nombreTabla);
- static int _bdatos_construir_encabezado(tEncabezado *encabezado, const char *nombreTabla, int cantCampos, tCampo *campos);
- static int _bdatos_buscar_campo(const tCampo *campos, int cantCampos, tCampo *campoEncontrado, const char *nombreCampoLeido);
- static int _bdatos_manejar_apertura_tabla(tBDatos *bDatos, const char *nombreTabla);
- static int _bdatos_cmp_indice(const void *a, const void *b);
- static int _bdatos_parsear(tSecuencia *secuencia, tParsear parsear, void *salida, unsigned tamSalida);

- static int _bdatos_parsear_identificador(tSecuencia *secuencia, void *salida, unsigned tamSalida);
- static int _bdatos_parsear_simbolo(tSecuencia *secuencia, void *salida, unsigned tamSalida);
- static int _bdatos_parsear_caracter(tSecuencia *secuencia, void *salida, unsigned tamSalida);
- static int _bdatos_parsear_numeros(tSecuencia *secuencia, void *salida, unsigned tamSalida);
- static int _bdatos_parsear_texto(tSecuencia *secuencia, void *salida, unsigned tamSalida);
- static void _bdatos_crear_secuencia(tSecuencia *secuencia, const char *buffer);
- static int _bdatos_parsear_declaracion_campos(tSecuencia *secuencia, tCampo *campos, int maxCampos, int *cantCamposLeidos);
- static int _bdatos_leer_campo_tipo(tSecuencia *secuencia, tCampo *campo);
- static int _bdatos_parsear_valores_insercion(tSecuencia *secuencia, const tEncabezado *encabezado, tDatoParseado *datosParseados, int *cantParseados);
- static eSimbolo _bdatos_comparar_simbolo(const char* simbolo);

SERVIDOR:

Funciones Declaradas

- int servidor_inicializar();
- SOCKET servidor_crear_socket();
- void servidor_procesar_solicitud(tBDatos *bDatos, SOCKET *sock, const char *solicitud);

MÓDULOS COMUNES:

ARBOL:

Funciones Declaradas

- void arbol_crear(tArbol *arbol);
- int arbol_buscar(const tArbol *arbol, void *dato, unsigned tamDato, tCmp cmp);
- int arbol_insertar_rec(tArbol *arbol, const void *dato, unsigned tamDato, tCmp cmp);
- void arbol_recorrer_preorden(const tArbol *arbol, void *extra, tAccion accion);
- void arbol_recorrer_posorden(const tArbol *arbol, void *extra, tAccion accion);
- int arbol_cargar_de_archivo(FILE *arch, tArbol *arbol, unsigned tamReg, tCmp cmp);
- int arbol_escribir_en_arch(FILE *arch, tArbol *arbol);
- void arbol_vaciar(tArbol *arbol);

Funciones Internas:

- static tNodoArbol** _arbol_buscar_nodo(const tArbol *arbol, const void *dato, tCmp cmp);
- static void _arbol_escribir_preorden(const tArbol *arbol, FILE *arch);

COLA:

Funciones Declaradas

- void cola_crear(tCola *cola);
- int cola_encolar(tCola *cola, const void *dato, unsigned tamDato);
- int cola_desencolar(tCola *cola, void *dato, unsigned tamDato);
- int cola_ver_primero(const tCola *cola, void *dato, unsigned tamDato);
- int cola_vacia(const tCola *cola);
- int cola_llena(const tCola *cola, unsigned tamDato);
- void cola_vaciar(tCola *cola);

LISTA:

Funciones Declaradas

- void lista_crear(tLista *lista);
- void lista_recorrer(const tLista *lista, tAccion accion, void *extra);
- int lista_insertar_final(tLista *lista, const void *dato, unsigned tamDato);
- int lista_insertar_comienzo(tLista *lista, const void *dato, unsigned tamDato);
- void lista_vaciar(tLista *lista);
- int lista_llena(const tLista *lista, unsigned tamDato);
- int lista_vacia(const tLista *lista);
- int lista_sacar_primero(tLista *lista, void *dato, unsigned tamDato);
- int lista_sacar_ultimo(tLista *lista, void *dato, unsigned tamDato);
- int lista_ver_primero(const tLista *lista, void *dato, unsigned tamDato);
- int lista_ver_ultimo(const tLista *lista, void *dato, unsigned tamDato);
- int lista_insertar_en_orden(tLista *lista, const void *dato, unsigned tamDato, int modo, tCmp cmp);

MATRIZ:

Funciones Declaradas

- void** matriz_crear(size_t columnas, size_t filas, size_t tamElem);
- void matriz_destruir(void** matriz, size_t filas);
- void matriz_inicializar(void** matriz, void *dato, size_t columnas, size_t filas, size_t tamElem);

PILA:

Funciones Declaradas

- void pila_crear(tPila *pila);
- int pila_llena(const tPila *pila, unsigned tamDato);
- int pila_apilar(tPila *pila, const void *dato, unsigned tamDato);
- int pila_tope(const tPila *pila, void *dato, unsigned tamDato);
- int pila_vacia(const tPila *pila);
- int pila_desapilar(tPila *pila, void *dato, unsigned tamDato);
- void pila_vaciar(tPila *pila);

VECTOR:

Funciones Declaradas

- int vector_crear(tVector* vector, size_t tamElem);
- void vector_vaciar(tVector *vector);
- void vector_destruir(tVector* vector);
- int vector_cargar_de_archivo(tVector* vector, const char* nombreArch, size_t tamElem);
- void vector_recorrer(tVector* vector, Accion accion, void* extra);
- void vector_ordenar(tVector* vector, int metodo, Cmp cmp);
- int vector_ord_buscar(tVector* vector, void* elem, Cmp cmp);
- int vector_ord_insertar(tVector* vector, void* elem, Cmp cmp, Actualizar actualizar);
- int vector_insertar_al_final(tVector* vector, void* elem);
- void vector_it_crear(tVectorIterador* it, tVector* vector);
- void* vector_it_primero(tVectorIterador* it);
- void* vector_it_siguiente(tVectorIterador* it);
- int vector_it_fin(tVectorIterador* it);

Funciones Internas:

- static void _ordenar_seleccion(tVector* vector, Cmp cmp);