



IC2001 Estructuras de Datos

Grupo 2

Jose Pablo Aguero Mora 2021126372

Katerine Guzmán Flores 2019390523

Manual técnico – Proyecto 1A

I Semestre 2022

Índice:

Descripción	3
Implementación gráfica	3
Menú principal	5
Estructuras utilizadas.....	7
Inicialización de las estructuras	8
Función juego().....	13
Salir del programa	18

Manual técnico

Descripción:

Este proyecto constituye un programa que simula un sistema de ordenamiento automático en diferentes modelos de parqueo. Se presentan 3 modelos: estacionamiento de una sola planta con un solo piso y ordenamiento horizontal, estacionamiento de una planta, pero con dos posibles pisos y estacionamiento en forma de torre con espacios a la izquierda y derecha del elevador central.

En cada uno de los modelos se muestra una interfaz gráfica la cual muestra las animaciones de los carros y el estado del parqueo en todo momento. De igual forma se presentan los controles respectivos con los que se pueden ingresar nuevos vehículos al parqueo.

Se puede llevar un control de registros en la consola, en donde se van viendo los datos ingresados. Todos los modelos cuentan con las validaciones respectivas para impedir que se asignen dos carros a un solo espacio, se ingresen carros cuando el parqueo ya está lleno, entre otros. Además, se cuenta con un menú inicial que permite al usuario navegar entre los diferentes modelos de estacionamiento.

Implementación gráfica:

Para poder mostrar un entorno gráfico se utilizó la librería Allegro, en este caso se colocaron bitmaps que representaban diferentes elementos de la simulación, como por ejemplo los vehículos o el mapa que define la organización del estacionamiento para cada uno de los modelos.

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_ttf.h>
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_native_dialog.h>
#include <allegro5/allegro_primitives.h>
#include <allegro5/allegro_image.h>
#include <allegro5/allegro_audio.h>
#include <allegro5/allegro_acodec.h>
```

```
//Se dibuja el fondo animado y las opciones del menú en el display
al_clear_to_color(al_map_rgb(0, 0, 0));
al_draw_scaled_bitmap(Fondo, 0, 0, 1600, 1000, 0, 0, 800, 500, NULL); // 1920
al_draw_text(fuente1, al_map_rgb(53, 44, 25), X / 2, (RY * (50.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "ESTACIONAMIENTOS");
al_draw_text(fuente2, al_map_rgb(0, 0, 0), X / 2, (RY * (100.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "PARQUEO DE TORRE");
al_draw_text(fuente2, al_map_rgb(0, 0, 0), X / 2, (RY * (130.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "PARQUEO UN PISO");
al_draw_text(fuente2, al_map_rgb(0, 0, 0), X / 2, (RY * (160.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "PARQUEO DOS PISOS");
al_draw_text(fuente2, al_map_rgb(0, 0, 0), X / 2, (RY * (190.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "SALIR");
}
```

De esta forma se inicializaron cada uno de los servicios de allegro como por ejemplo los timers que son vitales para la ejecución del movimiento y la captura de entradas por teclado mientras el simulador se está ejecutando.

```
//Se crean todos los punteros propios de Allegro y se carga el multimedia y los timers
ALLEGRO_TIMER* timer = al_create_timer(1.0 / FPS);
ALLEGRO_TIMER* timerD = al_create_timer(1.0 / 15);

ALLEGRO_EVENT_QUEUE* cola_eventos = al_create_event_queue();

ALLEGRO_FONT* fuente1;
ALLEGRO_FONT* fuente2;
ALLEGRO_FONT* fuente3;

ALLEGRO_BITMAP* Fondo = al_load_bitmap("Imagenes/vectorP.png");

fuente1 = al_load_font("pixel.ttf", 80, NULL);
fuente2 = al_load_font("pixel.ttf", 50, NULL);
fuente3 = al_load_font("pixel.ttf", 30, NULL);

//Se registran las fuentes de eventos(timers y periféricos) en la cola de eventos
al_register_event_source(cola_eventos, al_get_timer_event_source(timer));
```

Esta implementación se hace por separado en 4 secciones separadas, la principal se encuentra en el archivo main, en donde se establece la ejecución del menú. Las otras son inicializaciones muy similares que permiten la presentación de los diferentes parqueos. Estas ejecuciones se hacen en las funciones juego() las cuales se encuentran en el archivo funciones.h, pero cuando termina la ejecución de estos procesos (al presionar la tecla escape) y termina el ciclo infinito dentro de la función juego(), se cierran esos procesos de allegro desde el menú.

```
ALLEGRO_DISPLAY* pantalla2 = al_create_display(ResX, ResY);
al_set_window_position(pantalla2, RX / 4 - ResX / 4, RY / 4 - ResY / 4);
al_set_window_title(pantalla2, "Proyecto 1A - Parqueo de torre");
```

```
void juego1(PtrTCarro& Lista, PtrTEspacio& ListaEspacio) {
    if (!al_init()) {
        al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede inicializar Allegro", NULL, NULL);
        return;
    }

    al_init_font_addon();
    al_init_ttf_addon();
    al_init_image_addon();

    al_init_primitives_addon();
    al_install_keyboard();
    int ResX = 800;
    int ResY = 500;

    ALLEGRO_EVENT_QUEUE* cola_eventos = al_create_event_queue();

    ALLEGRO_TIMER* timer = al_create_timer(1.0 / FPS);
    ALLEGRO_BITMAP* Personaje = al_load_bitmap("carroRojo.png");
    ALLEGRO_BITMAP* Personaje2 = al_load_bitmap("carroVerde.png");
```

Los recursos, referentes a estos servicios de allegro se encuentran en la carpeta del proyecto y están diseñados en dimensiones tales que los espacios del parqueo se acomoden con las rutas definidas que automatizan la organización del estacionamiento.

Menú principal:



Todos los estacionamientos son accedidos mediante un menú inicial que inicia las listas de carros y espacios además de los servicios básicos de allegro que se comentaron en la sección anterior. Para generar este menú se

utilizaron eventos que registren los movimientos del mouse y lo asigne a coordenadas en la ventana, de esta forma si se posiciona el cursor sobre la zona donde se encuentran las opciones de selección se imprime un texto con propiedades iguales por encima, pero con un diferente color, así da el efecto de botón interactivo.

Se programa esta misma zona para que realice acciones si recibe un click encima. Con todos estos elementos se logra generar un efecto de botones que permite navegar entre los diferentes parqueos y ofrece una opción extra para salir del programa.

```
//
if ((mousex >= X / 2 - 145 && mousex <= X / 2 + 145) && (mousey >= (RY * 100.0 / 768.0) && mousey <= (RY * 130.0 / 768.0))) {
    al_draw_text(fuente2, al_map_rgb(243, 123, 104), X / 2, (RY * (100.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "PARQUEO DE TORRE");
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
    {
        if (eventos.mouse.button & 1) { ... } Opción 1
    }
}

//
if ((mousex >= X / 2 - 140 && mousex <= X / 2 + 140) && (mousey >= (RY * (130.0 / 768.0)) && mousey <= (RY * 160.0 / 768.0))) {
    al_draw_text(fuente2, al_map_rgb(243, 123, 104), X / 2, (RY * (130.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "PARQUEO UN PISO");
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
    {
        if (eventos.mouse.button & 1) { ... } Opción 2
    }
}

//
if ((mousex >= X / 2 - 145 && mousex <= X / 2 + 145) && (mousey >= (RY * (160.0 / 768.0)) && mousey <= (RY * (190.0 / 768.0)))) {
    al_draw_text(fuente2, al_map_rgb(243, 123, 104), X / 2, (RY * (160.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "PARQUEO DOS PISOS");
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
    {
        if (eventos.mouse.button & 1) { ... } Opción 3
    }
}

if ((mousex >= X / 2 - 120 && mousex <= X / 2 + 120) && (mousey >= (RY * (190.0 / 768.0)) && mousey <= (RY * (220.0 / 768.0)))) {
    //Si se presiona la opción de salir se sale de la cola de eventos y termina la aplicación
    al_draw_text(fuente2, al_map_rgb(243, 123, 104), X / 2, (RY * (190.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "SALIR");
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
    {
        if (eventos.mouse.button & 1) {
            hecho2 = false;
            al_destroy_display(pantalla);
            system("CLS");
        }
    }
}
```

Si extendemos el contenido de el condicional que detecta el evento de click, podemos ver un contenido como el siguiente:

```
if ((mousex >= X / 2 - 140 && mousex <= X / 2 + 140) && (mousey >= (RY * (130.0 / 768.0)) && mousey <= (RY * 160.0 / 768.0))) {
    al_draw_text(fuente2, al_map_rgb(243, 123, 104), X / 2, (RY * (130.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "PARQUEO UN PISO");
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
    {
        if (eventos.mouse.button & 1) {
            sigo = true;
            system("CLS");
            al_destroy_display(pantalla);

            PtrEspacio ListaEspacios;
            PtrCarro ListaCarros;

            LlenarListaEnlazada2(ListaEspacios);
            LlenarListaEnlazadaCarro2(ListaCarros);
            LlenarPorcentajeOcupado2(ListaEspacios, ListaCarros);

            ALLEGRO_DISPLAY* pantalla2 = al_create_display(ResX, ResY);
            al_set_window_position(pantalla2, RX / 4 - ResX / 4, RY / 4 - ResY / 4);
            al_set_window_title(pantalla2, "Proyecto 1A - Parqueo de un piso");

            printf("----- Bienvenido al sistema de parqueos ----- \n");
            printf("Siga las instrucciones en pantalla para ingresar o retirar vehiculos \n");

            juego2(ListaCarros, ListaEspacios);

            if (!pantalla2)
            {
                al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede crear la pantalla", NULL, ALLEGRO_MESSAGEBOX_ERROR);
                return -1;
            }
            DestruirInventarioEspacios(ListaEspacios);
            DestruirInventarioCarros(ListaCarros);
            al_destroy_display(pantalla2);
            main();
            hecho2 = false;
        }
    }
}
```

Por lo tanto cada vez que se selecciona una de estas opciones se generan nuevas listas enlazadas para los carros y los espacios, se asignan dinámicamente las

coordenadas del parqueo y se inicializan los servicios de allegro correspondientes para proceder a invocar la función juego() correspondiente que contiene la interfaz y la lógica propia de cada modelo de parqueo.

Estructuras utilizadas:

Antes de analizar los algoritmos correspondientes al ordenamiento y movimiento de los carros es necesario tener presente cómo funcionan las estructuras utilizadas en el proyecto.

Para poder gestionar las coordenadas correctamente es necesario utilizar una lista enlazada de vehículos y otra lista separada que corresponde a espacios del parqueo.

El struct de los nodos espacio se generan de la siguiente manera:

```
typedef struct TEspacio // Nueva estructura de tipo TEspacio
{
    int CodigoP;
    int PosX;
    int PosY;
    bool Ocupado;
    int Columna;
    TEspacio* Siguiente;
}*PtrTEspacio;
```

El CodigoP representa el identificador de cada espacio, sirve para buscar entre la lista enlazada, se enlaza con la placa del vehículo registrado. También se encuentran las coordenadas en "X" y "Y", además de un booleano que indica si el espacio está ocupado por otro vehículo. Finalmente tiene un entero llamado Columna, que sirve para saber en cuál de los dos segmentos del mapa está el espacio del parqueo.

Por otro lado, el struct de los nodos carro se presentan:

```
typedef struct TCarro // Nueva estructura de tipo TCarro
{
    int CodigoP;
    int PosX;
    int PosY;
    int Skin;
    bool Parqueado;
    bool Sale;
    TCarro* Siguiente;
}*PtrTCarro;
```

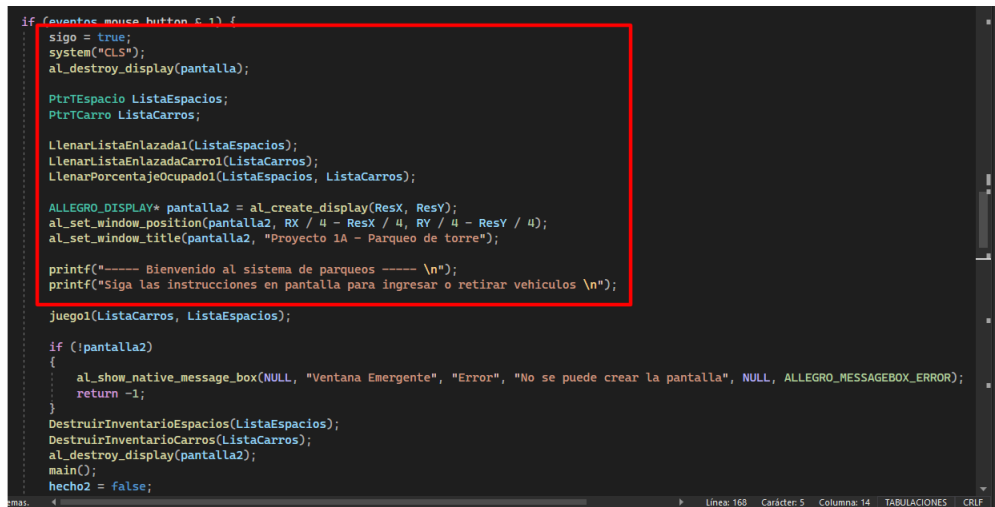
En este caso cuenta igualmente con un identificador entero, además de las coordenadas base. A partir de esto se encuentra el valor “Skin” que es un entero asignado de forma aleatoria para definir cuál es el bitmap de carro que se le va a asignar al nodo en particular.

El valor Parqueado define si el carro ya llegó a su destino (es un flag que permite controlar las animaciones), de igual forma sale cumple la misma función, pero en el movimiento inverso.

Inicialización de las estructuras:

Los 3 modelos de estacionamiento presentan una línea lógica muy similar, con diferencias en el movimiento, las cantidades y los sistemas de coordenadas. Por este motivo se va a explicar en detalle solo uno de estos modelos y se va a indicar en donde varían con respecto a los demás.

Esta ejecución inicia al hacer click en una de las opciones (según lo descrito en la sección del menú):



```
if (eventos_mouse.button == 1) {
    signo = true;
    system("CLS");
    al_destroy_display(pantalla);

    PtrTEspacio ListaEspacios;
    PtrTCarro ListaCarros;

    LlenarListaEnlazada1(ListaEspacios);
    LlenarListaEnlazadaCarro1(ListaCarros);
    LlenarPorcentajeOcupado1(ListaEspacios, ListaCarros);

    ALLEGRO_DISPLAY* pantalla2 = al_create_display(ResX, ResY);
    al_set_window_position(pantalla2, RX / 4 - ResX / 4, RY / 4 - ResY / 4);
    al_set_window_title(pantalla2, "Proyecto 1A - Parqueo de torre");

    printf("----- Bienvenido al sistema de parqueos ----- \n");
    printf("Siga las instrucciones en pantalla para ingresar o retirar vehiculos \n");

    juego1(ListaCarros, ListaEspacios);

    if (!pantalla2)
    {
        al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede crear la pantalla", NULL, ALLEGRO_MESSAGEBOX_ERROR);
        return -1;
    }
    DestruirInventarioEspacios(ListaEspacios);
    DestruirInventarioCarros(ListaCarros);
    al_destroy_display(pantalla2);
    main();
    hecho2 = false;
}
```

Lo primero a notar en esta sección es la conversión de variable “signo” a true, esta variable ayuda al sistema a reconocer cuando se pueden ingresar más vehículos al

estacionamiento. Posteriormente se limpia la consola y se cierra la ventana del menú principal, para pasar a declarar las listas respectivas: ListaEspacios y ListaCarros.

La función LlenarListaEnlazada permite unificar ciertas tareas necesarias como inicializar la lista (en NULL), además de generar automáticamente nuevos nodos (según el índice del ciclo) y agregarlos a la lista de espacios.

```
// Llenar parqueo torre
void LlenarListaEnlazada1(PtrTEspacio& Lista)
{
    PtrTEspacio Nuevo;
    InicializarInventario(Lista);
    for (int i = 1; i <= 18; i++)
    {
        Nuevo = CrearEspacio1(i, i);
        AgregarInicioInventario(Lista, Nuevo);
    }
}
```

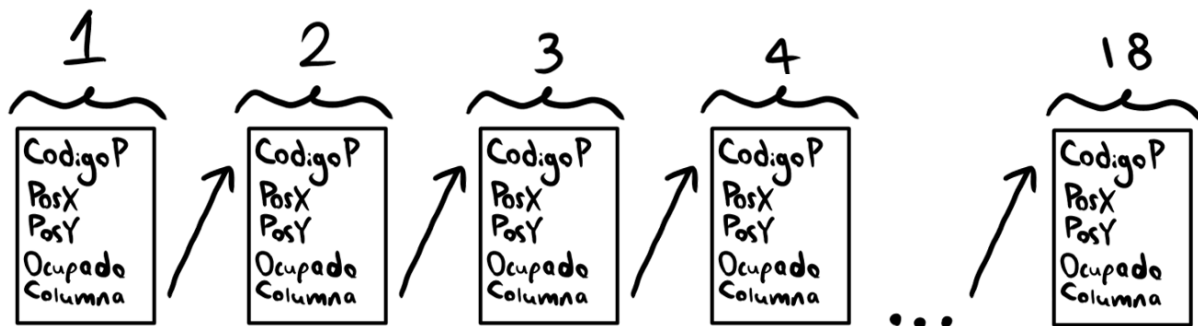
En este caso se repite el proceso 18 veces ya que esta es la cantidad de espacios del parqueo en forma de torre. A la hora de crear los espacios, lo más importante es diseñar un sistema que pueda asignar cada coordenada correctamente de manera dinámica.

```
PtrTEspacio CrearEspacio1(int NCodigo, int NNombre)
{
    PtrTEspacio Pieza = new(TEspacio);

    // Asignacion de valores según cada ciclo
    Pieza->CodigoP = NCodigo;
    if (NCodigo == 10) {
        pruebaTorre = 433;
    }
    if (NCodigo >= 10) {
        Pieza->PosX = 470;
    }
    else {
        Pieza->PosX = 310;
    }
    Pieza->PosY = pruebaTorre;
    pruebaTorre -= 50;
    Pieza->Ocupado = false;

    Pieza->Siguiete = NULL;
    return Pieza;
}
```

Para este modelo específico se inicializa en la coordenada “Y” de 433 que es el lugar más bajo donde puede haber un espacio. A partir de ahí se va restando 50 a esa coordenada hasta completar 9 ciclos (que representan una columna) y una vez que termine devuelve la coordenada “Y” a 433 y cambia la “X” a 470 que representa la segunda columna, en este punto se repite el proceso de restar 50. Esto finaliza cuando ya ingresa los 18 espacios (en el caso del segundo parqueo son 14 espacios y en el tercero 12).



Una vez que termine el llenado de la lista de espacios se procede a realizar un proceso similar, pero con los carros.

```
// Llenar lista carros parqueo torre
void LlenarListaEnlazadaCarro1(PtrTCarro& Lista)
{
    PtrTCarro Nuevo;
    InicializarInventarioCarro(Lista);
    for (int i = 1; i <= 5; i++)
    {
        Nuevo = CrearCarro1(i, i);
        AgregarInicioInventarioCarro(Lista, Nuevo);
    }
}
```

La diferencia principal es que, en el caso de los carros, el proceso solo se repite 5 veces, ya que inicialmente estos son los únicos vehículos registrados en el parqueo gracias al sistema de ocupación aleatorio.

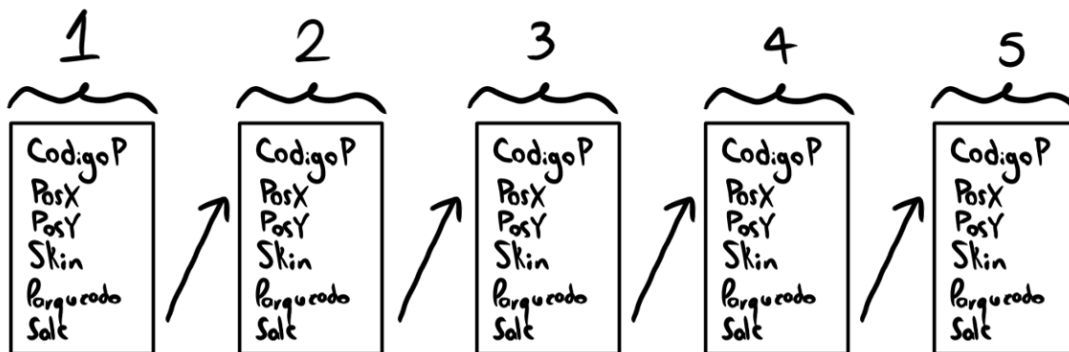
De igual forma a la hora de crear los nodos en esta lista se debe tener en cuenta la posición inicial del vehículo cuando va a ingresar al estacionamiento. Además, se inicializan valores como Parqueado y Sale en falso para poder controlar las animaciones correctamente. Y algo importante a resaltar es que la propiedad Skin es un número aleatorio de 1 a 4 que luego sirve para imprimir uno de los 4 modelos de carro registrados en cada modelo de parqueo.

```
PtrTCarro CrearCarro1(int NCodigo, int NNombre)
{
    PtrTCarro Pieza = new(TCarro);

    // Asignacion de valores según cada ciclo
    Pieza->CodigoP = NCodigo;
    Pieza->PosX = 390;
    Pieza->PosY = 478;
    Pieza->Parqueado = false;
    Pieza->Sale = false;

    int numCarro = crearRandom(1, 4);
    Pieza->Skin = numCarro;
    Pieza->Siguiente = NULL;
    return Pieza;
}
```

De esta forma, inicialmente se genera una estructura como la siguiente:



Y la última parte al inicializar estas estructuras es el asignar la ocupación aleatoria.

Esto se hace buscando un espacio aleatorio no ocupado para cada uno de los 5 carros iniciales, para así modificar sus coordenadas de inicio e igualarlas a las de los espacios seleccionados antes de que se abra la ventana. Esto para omitir la animación.

```

void LlenarPorcentajeOcupado1(PtrTEspacio& ListaEspacios, PtrTCarro& ListaCarros)
{
    PtrTEspacio EActual;
    PtrTCarro CActual;
    CActual = ListaCarros;
    int contadorA = 1;
    while (contadorA != 6) // Realiza el proceso con los 5 primeros carros de la lista
    {
        EActual = ListaEspacios; // Vuelve al inicio a buscar
        int numRand = crearRandom(1, 16);

        while (*find(numeros.begin(), numeros.end(), numRand) == numRand) {
            numRand = crearRandom(4, 17);
        }

        numeros.push_back(numRand); // Agrega al vector para no repetir random

        while (EActual->CodigoP != numRand)
        {
            EActual = EActual->Siguiente;
        }

        EActual->Ocupado = true;
        CActual->Parqueado = true;

        int posAuxX = EActual->PosX;
        int posAuxY = EActual->PosY;

        CActual->PosX = posAuxX;
        CActual->PosY = posAuxY;

        CActual = CActual->Siguiente;
        contadorA += 1;
    }
}

```

Para realizar el proceso se hace uso de un vector que va acumulando los identificadores de los espacios ya usados, de esta forma cada vez que se va a asignar un nuevo espacio se verifica este vector comprobando su estado, si no está el identificador se asigna ese espacio y si está entonces repite el proceso hasta encontrar un espacio aleatorio disponible.

Se igualan las coordenadas y de esta forma cuando se inicie la ventana no va a ejecutar las animaciones de movimiento, sino que los vehículos ya van a estar en sus posiciones respectivas.

Aquí finaliza la preparación de las estructuras previas a generar la ventana y los recursos visuales con Allegro.

```

if (eventos.mouse.button & 1) {
    siga = true;
    system("CLS");
    al_destroy_display(pantalla);

    PtrTEspacio ListaEspacios;
    PtrTCarro ListaCarros;

    LlenarListaEnlazada1(ListaEspacios);
    LlenarListaEnlazadaCarro1(ListaCarros);
    LlenarPorcentajeOcupado1(ListaEspacios, ListaCarros);

    ALLEGRO_DISPLAY* pantalla2 = al_create_display(ResX, ResY);
    al_set_window_position(pantalla2, RX / 4 - ResX / 4, RY / 4 - ResY / 4);
    al_set_window_title(pantalla2, "Proyecto 1A - Parqueo de torre");

    printf("----- Bienvenido al sistema de parqueos ----- \n");
    printf("Siga las instrucciones en pantalla para ingresar o retirar vehiculos \n");
    juego1(ListaCarros, ListaEspacios);

    if (!pantalla2)
    {
        al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede crear la pantalla", NULL, ALLEGRO_MESSAGEBOX_ERROR);
        return -1;
    }
    DestruirInventarioEspacios(ListaEspacios);
    DestruirInventarioCarros(ListaCarros);
    al_destroy_display(pantalla2);
    main();
    hecho2 = false;
}

```

En esta sección se genera la pantalla del modelo con sus dimensiones y características correspondientes, además se muestra un mensaje de bienvenida en la consola y la parte más importante es la invocación de la función juego(), ya que aquí se genera toda la lógica de los controles y el movimiento.

Función juego():

A esta función se le ingresan las dos listas previamente manipuladas y editadas (de espacios y carros), lo primero que se realiza en la función es iniciar los servicios de allegro necesarios para mostrar la interfaz.

```

void juego1(PtrTCarro& Lista, PtrTEspacio& ListaEspacio) {
    if (!al_init()) {
        al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede inicializar Allegro", NULL, NULL);
        return;
    }

    al_init_font_addon();
    al_init_ttf_addon();
    al_init_image_addon();

    al_init_primitives_addon();
    al_install_keyboard();
    int ResX = 800;
    int ResY = 500;

    ALLEGRO_EVENT_QUEUE* cola_eventos = al_create_event_queue();

    ALLEGRO_TIMER* timer = al_create_timer(1.0 / FPS);
    ALLEGRO_BITMAP* Personaje = al_load_bitmap("carroRojo.png");
    ALLEGRO_BITMAP* Personaje2 = al_load_bitmap("carroVerde.png");
    ALLEGRO_BITMAP* Personaje3 = al_load_bitmap("carroAzul.png");
    ALLEGRO_BITMAP* Personaje4 = al_load_bitmap("carroCafe.png");
    ALLEGRO_BITMAP* Npc = al_load_bitmap("coolbug.png");
    ALLEGRO_BITMAP* Fondo = al_load_bitmap("torre5.png"); // room.jpg
    //ALLEGRO_FONT* Fuente = al_load_font("bladeRunner.TTF", 40, NULL);

    al_register_event_source(cola_eventos, al_get_timer_event_source(timer));
    al_register_event_source(cola_eventos, al_get_keyboard_event_source());
}

```

Además, se inicializa el timer y la cola de eventos, que registra eventos de tipo timer (para las animaciones) y eventos de teclado (para los controles).

Posteriormente se generan los valores globales necesarios para la ejecución y se inicia un ciclo infinito (while true) que va a ser la base del simulador.

```
while (hecho) {  
    ALLEGRO_EVENT eventos;  
    al_wait_for_event(cola_eventos, &eventos);  
    if (eventos.type == ALLEGRO_EVENT_KEY_UP)  
    {  
        switch (eventos.keyboard.keycode) {  
            case ALLEGRO_KEY_ESCAPE:  
                hecho = false;  
                break;  
            case ALLEGRO_KEY_R:  
                int val;  
                break;  
        }  
    }  
}
```

Así como se ve en la imagen anterior, se usa un switch para ir indicando cómo el programa debe reaccionar a distintos eventos de teclado, como en ese caso el escape para salir del ciclo infinito y que se acabe la simulación. Pero se presentan otros como:

```
break;  
case ALLEGRO_KEY_R:  
    int val;  
    printf("Ingrese los datos: ");  
    cin >> val;  
  
    printf("\n");  
    printf("Informacion registrada, seleccione una opcion (A = Ingresar / W = Retirar)\n");  
    printf("\n");  
    break;
```

El caso de la R que activa una entrada por teclado en la consola mostrando un mensaje. Lo que se genera en ese input se guarda en la variable correspondiente para luego ser útil en las operaciones.

```
case ALLEGRO_KEY_A:  
    if (sigo == true) {  
        if ((VerificaMismo(Lista, val)) == false) { // Solo si no hay otra placa igual  
            PtrTCarro Nuevo;  
  
            Nuevo = CrearCarro1(val, general);  
            printf("\n");  
            printf(" Ingresando el vehiculo con la Placa: %i \n", val);  
            printf("\n");  
            AgregarInicioInventarioCarro(Lista, Nuevo);  
            Aux1 = Lista;  
        }  
        else {  
            printf("\n");  
            printf("Esta placa ya se encuentra registrada, ingrese una nueva \n");  
            printf("\n");  
        }  
    }  
    break;
```

El caso de la A que primero valida si no existe otra placa con ese registro y si eso se cumple entonces crea un nodo con la información agregada en R que inicializa el vehículo con los valores base, indica al usuario que el carro está ingresando y lo ingresa a la lista enlazada.

```
case ALLEGRO_KEY_W:
    AuxCarro = Lista;
    while (AuxCarro->CodigoP != val && AuxCarro->Siguiente != NULL)
    {
        AuxCarro = AuxCarro->Siguiente;
    }

    if (AuxCarro->CodigoP != val) {
        printf("Esta placa no se encuentra registrada \n");
    }
    else {
        if (AuxCarro->Parqueado == true) {
            printf("Sacando el vehiculo del edificio... \n");

            AuxCarro->Sale = true;
            int cual = AuxCarro->CodigoP;

            LiberarEspacio(ListaEspacio, cual);
        }
    }
    break;
}
```

Finalmente, en el caso de la W, se busca el nodo con el identificador ingresado en R, si no lo encuentra muestra un mensaje, pero en caso de que sí lo encuentre entonces cambia su valor de Sale a true (para que el sistema sepa que debe hacer la animación inversa) y libera espacio (esta función solamente cambia el valor de ocupado de ese espacio para que luego otro carro pueda entrar).

Esos son todos los controles registrados, luego se presenta el código respecto a los eventos de timer, en donde se van a tomar los elementos nuevos en la lista enlazada para asignarles espacio y luego moverlos (animación) hasta que sus coordenadas se igualen a las del espacio correspondiente.

```

if (eventos.type == ALLEGRO_EVENT_TIMER)
{
    if (eventos.timer.source == timer)
    {
        PtrTEspacio AuxEspacio;
        AuxEspacio = ListaEspacio;

        VerificaSigo(ListaEspacio); // Función para modificar el valor sigo

        while (AuxEspacio->Ocupado == true && sigo == true) // Solo si sigo está activo
        {
            if (AuxEspacio->Siguiente != NULL) {
                AuxEspacio = AuxEspacio->Siguiente; // se pasa al siguiente nodo
            }
        }

        // Aquí ya se encontró el espacio disponible en AuxEspacio
    }
}

```

En el código anterior se obtiene un espacio disponible y se valida que aún exista campo en el parqueo para nuevos carros.

```

// Aquí ya se encontró el espacio disponible en AuxEspacio
if (Aux1->Parqueado == false) {
    if (Aux1->PosY != AuxEspacio->PosY) {
        Aux1->PosY = Aux1->PosY - 5;
    }
    else {
        if (Aux1->PosX != AuxEspacio->PosX) {
            if (AuxEspacio->PosX > Aux1->PosX) {
                Aux1->PosX = Aux1->PosX + 5;
            }
            else {
                Aux1->PosX = Aux1->PosX - 5;
            }
        }
        else {
            if (Aux1->Siguiente != NULL) {
                AuxEspacio->Ocupado = true;
                Aux1->Parqueado = true;
                AuxEspacio->CodigoP = Aux1->CodigoP; // Se le asigna la placa al código de espacio
                Aux1 = Aux1->Siguiente;
            }
        }
    }
}
else {
    if (Aux1->Siguiente != NULL) {
        Aux1 = Aux1->Siguiente;
    }
}
}

```

De esta forma se busca modificar primero la coordenada “Y” (en este modelo en particular, funciona diferente en los demás) hasta que esa posición del carro sea igual a la del espacio asignado. Luego se realiza el mismo proceso, pero con la coordenada en “X”, cuando todo esto se cumple el espacio se pone como ocupado, el carro como parqueado y se guarda la placa del vehículo en el identificador del espacio.

Para el proceso de salida o retiro de vehículo se hace la misma lógica, pero a la inversa dentro del registro de eventos timer.

```
if (AuxCarro->Sale == true) {
    if (AuxCarro->PosX != 390)
    {
        if (390 > AuxCarro->PosX) {
            AuxCarro->PosX = AuxCarro->PosX + 5;
        }
        else {
            AuxCarro->PosX = AuxCarro->PosX - 5;
        }
    }
    else {
        if (AuxCarro->PosY != 40) {
            AuxCarro->PosY = AuxCarro->PosY + 5;
        }
        else {
            int codeCual = AuxCarro->CodigoP;
        }
    }
}
```

Se modifica primero la posición en “X”, luego la de “Y” hasta que las coordenadas del carro sean las mismas a cuando se creó el nodo, en otras palabras: hasta que salga del estacionamiento.

```
al_clear_to_color(al_map_rgb(0, 0, 0));
al_draw_scaled_bitmap(Fondo, 0, 0, 1920, 1100, 0, 0, 800, 500, NULL);
PtrTCarro Aux;
Aux = Lista;
while (Aux != NULL)
{
    if (Aux->Skin == 1) {
        al_draw_bitmap(Personaje, Aux->PosX, Aux->PosY, NULL);
    }
    if (Aux->Skin == 2) {
        al_draw_bitmap(Personaje2, Aux->PosX, Aux->PosY, NULL);
    }
    if (Aux->Skin == 3) {
        al_draw_bitmap(Personaje3, Aux->PosX, Aux->PosY, NULL);
    }
    if (Aux->Skin == 4) {
        al_draw_bitmap(Personaje4, Aux->PosX, Aux->PosY, NULL);
    }

    Aux = Aux->Siguiente; // se pasa al siguiente nodo
}
```

Una vez que se definen estos algoritmos de movimiento, se muestran todos los recursos visuales en pantalla por medio de allegro. Es importante resaltar que, para mostrar los carros, se lee su valor de Skin y según eso se elige el modelo visual.

```

pruebaPiso = 285;
pruebaTorre = 433;
pruebaTercero = 85;
numeros.clear();
al_destroy_event_queue(cola_eventos);
al_destroy_timer(timer);

```

Y cuando el usuario decide salir del ciclo infinito presionando la tecla escape en cualquier momento, se ejecutan las instrucciones anteriores, las cuales devuelven los valores globales a como estaban en un inicio y cierran ciertos servicios de allegro.

```

if (eventos.mouse.button & 1) {
    sigo = true;
    system("CLS");
    al_destroy_display(pantalla);

    PtrEspacio ListaEspacios;
    PtrCarro ListaCarros;

    LlenarListaEnlazada1(ListaEspacios);
    LlenarListaEnlazadaCarro1(ListaCarros);
    LlenarPorcentajeOcupado1(ListaEspacios, ListaCarros);

    ALLEGRO_DISPLAY* pantalla2 = al_create_display(ResX, ResY);
    al_set_window_position(pantalla2, RX / 4 - ResX / 4, RY / 4 - ResY / 4);
    al_set_window_title(pantalla2, "Proyecto 1A - Parqueo de torre");

    printf("----- Bienvenido al sistema de parqueos ----- \n");
    printf("Siga las instrucciones en pantalla para ingresar o retirar vehiculos \n");

    juego1(ListaCarros, ListaEspacios);

    if (!pantalla2)
    {
        al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede crear la pantalla", NULL, ALLEGRO_MESSAGEBOX_ERROR);
        return -1;
    }

    DestruirInventarioEspacios(ListaEspacios);
    DestruirInventarioCarros(ListaCarros);
    al_destroy_display(pantalla2);
    main();
    hecho2 = false;
}

```

Finalmente, cuando se cierra el ciclo respectivo del modelo se realizan todos los cierres de los servicios allegro para cerrar la ventana, además de eliminar borrar las listas enlazadas de espacios y de carros.

Todos estos algoritmos descritos cambian ligeramente en los otros dos modelos en cuando a las cifras y sistemas de coordenadas, pero la lógica se mantiene igual.

Salir del programa:

Para salir del programa es necesario estar en el menú principal, aquí se programaron dos formas:

```

if ((mousex >= X / 2 - 120 && mousex <= X / 2 + 120) && (mousey >= (RY * (190.0 / 768.0)) && mousey <= (RY * (220.0 / 768.0)))) {
    //Si se presiona la opción de salir se sale de la cola de eventos y termina la aplicación
    al_draw_text(fuente2, al_map_rgb(243, 123, 104), X / 2, (RY * (190.0 / 768.0)), ALLEGRO_ALIGN_CENTRE, "SALIR");
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
    {
        if (eventos.mouse.button & 1) {
            hecho2 = false;
            al_destroy_display(pantalla);
            system("CLS");
        }
    }
}

if (eventos.type == ALLEGRO_EVENT_KEY_DOWN)
{
    //Si se presiona la tecla escape se sale del juego
    switch (eventos.keyboard.keycode) {
        case ALLEGRO_KEY_ESCAPE:
            hecho2 = false;
        }
    }
}

```

Es decir, usando la tecla escape o por otro lado haciendo click en la opción SALIR, ambos métodos cambian el valor de hecho2 a false, el cual mantiene el ciclo del menú.