Trabajo Práctico 2 - Programación en C

Tecnología Digital II

Octubre 2025

1. Introducción

En la cátedra de TD2 empezamos a desarrollar una versión del juego "Plantas vs. Zombies"para demostrar algunas técnicas de programación en C con la librería SDL2. Creamos un prototipo funcional (ver *juego_base.c*) que incluye una grilla de juego, animaciones y la lógica básica.

Para poder compilar y ejecutar el juego base (y también los códigos necesarios para el trabajo práctico), se debe tener instalada en el sistema operativa la librería SDL2.

La instalación se puede hacer con el siguiente comando:

```
sudo apt-get install build-essential libsdl2-dev libsdl2-image-dev
```

Sin embargo, nuestro prototipo tiene serias limitaciones de diseño: utiliza arrays estáticos de tamaño fijo para manejar la grilla de plantas y los zombies del juego. Si bien es funcional, esta decisión dificulta la implementación de mecánicas más complejas, como por ejemplo la generación de plantas que ocupen más de un espacio en la fila, o el manejo de una lista de zombies arbitrariamente grande.

El objetivo de este trabajo práctico es tomar el prototipo y refactorizar sus estructuras de datos principales. Para ello, deberán cambiar los arrays estáticos por un manejo de estructuras basadas en memoria dinámica.

El problema principal se centra en dos tareas:

1. Reemplazar el array 2D de plantas por una nueva estructura que gestione cada fila del jardín como una *lista enlazada de segmentos*, representando los espacios libres y ocupados.

2. Reemplazar el array de zombies por una lista enlazada simple en cada fila, permitiendo un número ilimitado de enemigos.

Modalidad: El trabajo práctico debe realizarse en grupos de tres personas.

Fecha de Entrega: Domingo 9 de noviembre hasta las 23:59hs.

2. Tipo de Datos: `GameBoard` y `GardenRow`

Para solucionar las limitaciones del prototipo, se debe implementar un nuevo sistema de gestión de la zona de juego. La estructura principal será el `GameBoard`, que contendrá las 5 filas del jardín (`GardenRow`).

Se debe implementar la estructura `GardenRow`, que gestiona las plantas usando una **lista enlazada de segmentos**.

2.1. Estructuras a Implementar

```c
// Representa un segmento de la fila (puede estar vacio o tener una planta)
typedef struct RowSegment {
    int status; // Por ejemplo, 0 para VACIO, 1 para PLANTA
    int start_col; // Columna donde empieza el segmento
    int length; // Cuantas celdas de columna ocupa este segmento
    Planta* planta_data; // Puntero a los datos de la planta
    struct RowSegment* next;
} RowSegment;

// Gestiona una fila completa del jardin
typedef struct GardenRow {
    RowSegment* first_segment; // Puntero al primer segmento de la lista
    struct ZombieNode* first_zombie; // Puntero al primer zombie
} GardenRow;

// Nodo para la lista enlazada simple de zombies
typedef struct ZombieNode {
    Zombie zombie_data;
    struct ZombieNode* next;
} ZombieNode;

// Estructura principal que contiene todo el estado del juego
typedef struct GameBoard {
    GardenRow rows[GRID_ROWS];
} GameBoard;
```

2.2. Lógica de Segmentos

La lista de `RowSegment` no guarda solo las plantas, sino que describe la fila completa. Para eso describe en una lista enlazada los huecos y las plantas presentes, indicando el tamaño (en cantidad de columnas) de cada objeto de la lista.

A continuación se describe cómo se usa la estructura en diferentes casos.

Fila Vacía: Al inicio, una `GardenRow` contiene un único segmento en su lista: {`status: VACIO, start_col: 0, length: 9, ...`}.

Colocar una Planta: Si el jugador coloca una planta en la columna 3 de una fila vacía, el segmento original de 9 espacios se debe partir en tres:

1. {`status: VACIO, start_col: 0, length: 3`} →
2. {`status: PLANTA, start_col: 3, length: 1`} →
3. {`status: VACIO, start_col: 4, length: 5`}

Sacar una Planta: Al quitar una planta de una fila, se deben fusionar los nodos correspondientes para representar el hueco más grande que quedó.

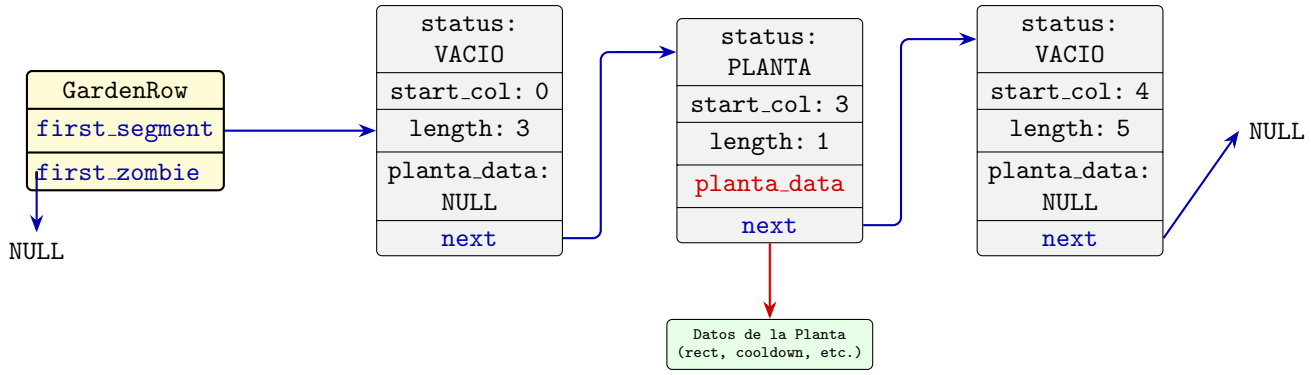

Figura 1: Diagrama detallado de una `GardenRow` originalmente vacía a la que se le colocó una planta en su columna 3

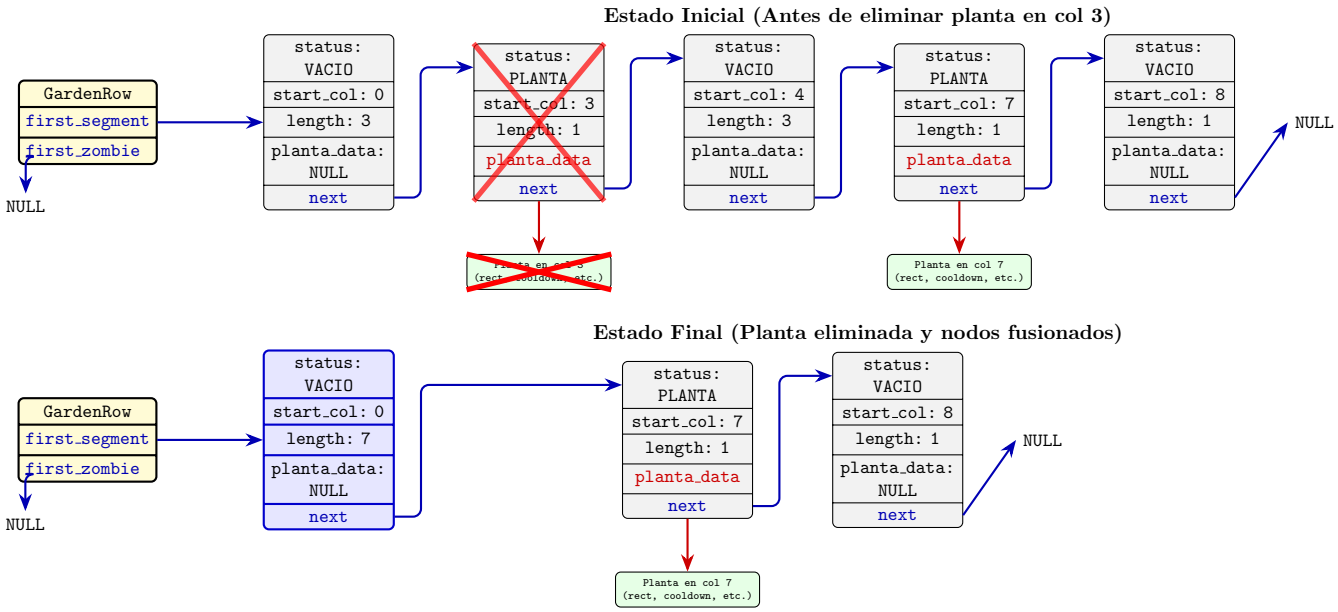

Figura 2: Diagrama detallado sobre la eliminación de una planta en la columna.

3. Enunciado

3.1. Ejercicio 1: Implementar las siguientes funciones sobre strings

Implementar las siguientes funciones:

- `char* strDuplicate(char* src)`: Duplica un string. Debe contar la cantidad de caracteres totales de src y solicitar la memoria equivalente. Luego, debe copiar todos los caracteres a esta nueva área de memoria. Además, como valor de retorno se debe retornar el puntero al nuevo string.
- `int strCompare(char* s1, char* s2)`: Compara dos strings lexicográficamente. Devuelve 0 si son iguales, 1 si el primer string es menor al segundo, -1 en otro caso.
- `char* strConcatenate(char* src1, char* src2)`: La función toma dos strings src1 y src2, y retorna un nuevo string que contiene una copia de todos los caracteres de src1 seguidos de los caracteres de src2. Además, la memoria de src1 y src2 debe ser liberada.

3.2. Ejercicio 2: Implementación del juego

Partiendo del archivo `solucion_esqueleto.c`, deberán completar el cuerpo de las siguientes funciones:

- `void gameBoardDelete(GameBoard* board)`: función para liberar toda la memoria pedida para todo las estructuras del juego.
- `int gameBoardAddPlant(GameBoard* board, int row, int col)`: agrega una planta en la fila *row* y columna *col*. Si ya hay una planta en dicho lugar, no hace nada.
- `void gameBoardRemovePlant(GameBoard* board, int row, int col)`: Saca una planta de la fila *row* y columna *col*. Si no hay una planta en dicho lugar, no hace nada.
- `void gameBoardAddZombie(GameBoard* board, int row)`: crea un nuevo zombie correctamente inicializado y lo agrega a la lista de zombies del juego.
- `void gameBoardUpdate(GameBoard* board)`: realiza la misma lógica que la función *actualizarEstado* de la solución base. En resumen, avanza el juego un paso, haciendo que los zombies se muevan, que las plantas disparen, que las arvejas se muevan, chequeando si las arvejas colisionan con los zombies y, por último, creando nuevos zombies si hiciese falta.
- `void gameBoardDraw(GameBoard* board)`: realiza la misma lógica que la función *dibujar* de la solución base. Se encarga de dibujar las plantas, las arvejas y los zombies.
- Adicionalmente, en el main se debe replicar la lógica para ver si el juego terminó al haber llegado un zombie a la casa.

4. Casos de Test Mínimos

Dentro del mismo archivo main para el juego, se deben agregar casos de test que pruebe la correcta implementación de las nuevas estructuras y las diferentes funciones.

A continuación, se enumera un conjunto mínimo de tests.

`strDuplicate`:

1. String vacío.
2. String de un carácter.
3. String que incluya todos los caracteres válidos distintos de cero.

`strCompare`:

1. Dos string vacíos.
2. Dos string de un carácter.
3. Strings iguales hasta un carácter (hacer cmpStr(s1,s2) y cmpStr(s2,s1)).
4. Dos strings diferentes (hacer cmpStr(s1,s2) y cmpStr(s2,s1)).

`strConcatenate`:

1. Un string vacío y un string de 3 caracteres.
2. Un string de 3 caracteres y un string vacío.
3. Dos strings de 1 caracter.
4. Dos strings de 5 caracteres.

`gameBoardAddPlant:`

1. Agregar una planta en una fila vacía. Agregar tanto en el medio como en los extremos.
2. Llenar una fila completa de plantas.
3. Intentar agregar una planta en una celda ya ocupada.

`gameBoardRemovePlant:`

1. Plantar en las columnas 3, 4 y 5. Sacar la planta de la columna 4.
2. Siguiendo el caso anterior, sacar luego la planta en la columna 3.
3. Llenar una fila de plantas. Sacar una del medio.

`gameBoardAddZombie:`

1. Tomar una lista de 3 zombies y agregarle uno más.
2. Crear una lista de 10000 zombies.

5. Entregable

- No se deberá entregar un informe. Sin embargo, deben agregar comentarios en el código que expliquen su solución. No deben comentar qué es lo que hace cada una de las instrucciones sino cuáles son las ideas principales del código implementado y por qué resuelve cada uno de los problemas.
- La entrega debe consistir solamente de un archivo .c en donde se encuentre el juego, las funciones sobre strings y todos los tests correspondientes.

6. Uso de IA

Utilizando cualquier herramienta de inteligencia artificial, ya sea Gemini, Copilot, ChatGPT, Claude o LLaMa, responder las siguientes preguntas:

- Indique el porcentaje aproximado de líneas de código del trabajo práctico que fueron realizadas con asistencia de una IA.
- ¿Cómo verificaron que las sugerencias de la IA eran correctas?
- ¿Se enfrentaron a alguna dificultad al utilizar las herramientas de IA? ¿Cómo las resolvieron?
- ¿Consideran que el uso de la IA les ha permitido desarrollar habilidades de programación en C? ¿Por qué?

Nota: Estas preguntas fueron generadas con la asistencia de la IA, respondiendo al prompt: "Supone que sos un docente universitario y estás haciendo un trabajo práctico de programación en C. Tu intención es que tus estudiantes utilicen las herramientas de IA para favorecer el aprendizaje y no para que estas les resuelvan el trabajo. ¿Qué preguntas se les podrías hacer al final del tp para determinar si utilizaron correctamente las herramientas de IA."