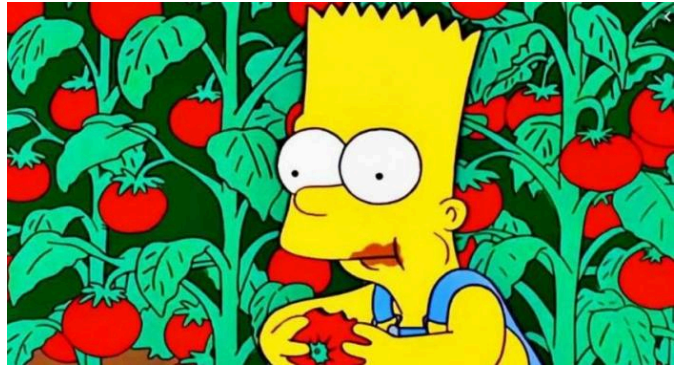


# Recursividad y labos

—

## Ejercicio 1

Los Simpsons empezaron una nueva vida como granjeros y su cultivo estrella es el Tomaco. Bart está comiendo una cesta de tomacos, pero entre ellos hay tomates que no quiere comer porque odia las verduras.



```
typedef struct fruto {  
    int peso; // en gramos  
    char nombre[MAX_NOMBRE];  
} fruto_t;
```

Se pide:

1. Hacer una función recursiva que dado un vector de frutos devuelva la cantidad de gramos de tomaco que comió Bart hasta encontrarse con un tomate.

## Ejercicio 2

Bart Simpson ha descubierto un nuevo y lucrativo negocio: el chef Luigi Risotto está pagando una fortuna por trufas silvestres para su restaurante. Sabiendo que el Bosque de Springfield está repleto de ellas, Bart decide adentrarse para recolectar tantas como pueda.

Sin embargo, el bosque es denso, caminar agota sus energías, y debe tener cuidado de no quedarse dormido de cansancio antes de juntar todo el botín.

Se solicita implementar un programa que simule la búsqueda de trufas en el Bosque de Springfield en forma de juego.

El juego consiste en un único nivel que se desarrolla en un terreno lógico de 5x5 (`MAX_FILAS` y `MAX_COLUMNAS`). En este terreno, Bart deberá recolectar trufas esparcidas aleatoriamente para venderlas al chef Luigi Risotto.

Al comenzar el juego, se deberán posicionar los elementos que lo componen en el terreno.

**IMPORTANTE**

- Ningún elemento puede inicializarse por fuera de los límites del terreno.

**Bart (Jugador)** Al comienzo del juego, Bart se inicializará en la primera coordenada del bosque (0, 0). El personaje contará con 100 movimientos\_restantes (energía) para usar durante toda la partida y 0 tesoros\_recolectados.

**Trufas (Tesoros)** Habrá hasta un máximo de 10 trufas (MAX\_TESOROS) inicializadas de forma random en el terreno. Bart las recolecta posicionándose sobre las mismas. Cuando Bart se topa con una trufa, se la guarda (sumando 1 a los tesoros recolectados). Al recolectarse, la trufa debe eliminarse de su vector, y se debe actualizar el tope\_tesoros correspondientemente.

Al moverse, Bart no puede pasarse de los límites del terreno. Por ejemplo, si Bart está en la fila 0 y el usuario lo quiere mover para arriba, ese movimiento queda sin efecto.

Bart se podrá mover en 4 direcciones gastando 1 movimiento por cada paso intentado mediante la función jugar:

- Arriba: W
- Abajo: S
- Derecha: D
- Izquierda: A

Luego de realizar una acción, en caso de estar en la misma coordenada que un tesoro, se activará la reacción relacionada al mismo que afectará a Bart y al juego (eliminación del tesoro, suma de recolección y aumento de movimientos restantes).

Para visualizar la partida, la función mostrar\_terreno deberá imprimir por pantalla una cuadrícula de 5x5. Se imprimirá el carácter 'B' para la posición del jugador, 'T' para los tesoros y un carácter a elección (por ejemplo -, \_ o ' ') para los espacios de tierra vacíos.

El estado del juego, evaluado por la función estado\_juego, se determinará de la siguiente manera:

- **El juego se dará por ganado (devuelve 1)** cuando Bart recolecta todas las trufas ocultas en el bosque (es decir, tesoros\_recolectados es 10).
- **El juego se dará por perdido (devuelve -1)** cuando Bart se queda sin movimientos restantes (0) antes de juntar todas las trufas.
- **El juego sigue en curso (devuelve 0)** mientras Bart tenga movimientos restantes y aún queden trufas en el terreno.

```

#ifndef RECOLECTOR_TESOROS_H
#define RECOLECTOR_TESOROS_H

#include <stdbool.h>

#define MAX_FILAS 5
#define MAX_COLUMNAS 5
#define MAX_TESOROS 10

typedef struct coordenada {
    int fil;
    int col;
} coordenada_t;

typedef struct jugador {
    coordenada_t posicion;
    int movimientos_restantes;
    int tesoros_recolectados;
} jugador_t;

typedef struct juego {
    jugador_t jugador;
    coordenada_t tesoros[MAX_TESOROS];
    int tope_tesoros;
} juego_t;

/*
 * Pre: -
 * Post:
 * - Inicializa el jugador en (0,0)
 * - tesoros_recolectados en 0
 * - movimientos_restantes en 100
 * - Inicializa los tesoros de forma aleatoria
 */
void cargar_juego(juego_t *juego);

/*
 * Pre: 'direccion' debe ser:
 * 'W' (arriba), 'S' (abajo), 'A' (izquierda), 'D' (derecha)
 * Post:
 * - Mueve al jugador si la jugada es válida (dentro del tablero)
 * - Si cae en un tesoro:
 *     - lo elimina del vector
 *     - suma 1 a tesoros_recolectados y 10 a movimientos_restantes
 */
void jugar(juego_t *juego, char direccion);

/*
 * Pre: juego inicializado
 * Post: Muestra el terreno por pantalla:
 * - 'J' para jugador
 * - 'T' para tesoros
 */
void imprimir_terreno(juego_t juego);

/*

```

```

* Pre: juego inicializado
* Post: Devuelve:
* 1 si el jugador recolectó todos los tesoros (gana)
* 0 si todavía quedan tesoros (sigue jugando)
* -1 si el jugador se quedó sin movimientos (pierde)
*/
int estado_juego(juego_t juego);

#endif

```

Para utilizar la biblioteca se deberá crear un 'juego.c' donde se llamará a la biblioteca y se le pedirá una acción al usuario.

### Ejercicio 3

El Alcalde Diamante anunció una inspección sorpresa a la Penitenciaría de Springfield. El Jefe Gorgory está en pánico porque estuvo durmiendo la siesta y no hizo la ronda.

La prisión está organizada en bloques, representados por una matriz de celda\_t. Gorgory no tiene tiempo de revisar a los ladrones de poca monta; su única prioridad es recorrer toda la matriz y verificar que todos los prisioneros de nivel de seguridad 3 (Máxima Seguridad) sigan en su celda.

```

typedef struct celda {
    int numero_celda;
    char nombre_recluso[MAX_NOMBRE];
    int nivel_seguridad; // 1: Baja, 2: Media, 3: Máxima
    bool esta_presente;
} celda_t;

```

Se pide:

1. Implementar una función que recorra la matriz completa de la prisión de forma recursiva. Si encuentra un prisionero de nivel 3, debe verificar su estado (esta\_presente). Si descubre que un preso de máxima seguridad se escapó, debe dejar de recorrer y dar aviso inmediato indicando quién es.