

# Memoria dinámica y TDAs

—

## Ejercicio 1

Springfield entra en pánico después de que aparece un oso en la ciudad. Para tranquilizar a la población, el jefe Gorgory empieza a registrar reportes de posibles osos en distintas zonas.

Para esto cuenta con las siguientes estructuras:

```
typedef struct reporte {
    char descripcion[MAX_DESCRIPCION];
    int nivel_riesgo; // 1-10
    bool confirmado;
} reporte_t;

typedef struct zona {
    char nombre_zona[MAX_NOMBRE];
    reporte_t* reportes;
    int cantidad_reportes;
    bool vigilancia_activa;
} zona_t;
```

Se pide:

1. Crear un procedimiento que reserve memoria para UNA zona nueva. El procedimiento debe recibir el nombre de la zona y si tiene vigilancia activa, pero debe inicializar la zona SIN reportes (cantidad\_reportes en 0 y el puntero reportes en NULL).
2. Crear un procedimiento que permita registrar UN nuevo reporte en una zona existente.
3. Crear un procedimiento que permita crear un arreglo de zonas.
4. El jefe Gorgory quiere saber qué zona genera más preocupación. Crear una función que reciba un arreglo de zonas y devuelva el nombre de la zona con mayor cantidad de reportes confirmados con nivel de riesgo mayor a 5.
5. Crear un procedimiento que libere completamente la memoria de un arreglo de zonas (incluidos todos sus reportes).

## Ejercicio 2

Homero empieza a vender productos hechos con tomacco. Como el negocio crece rápido, necesita registrar los pedidos de distintos clientes.

```
typedef struct producto_tomacco {
    char nombre[MAX_NOMBRE];
    int nivel_adiccion; // 1-10
    bool aprobado_por_lisa;
} producto_tomacco_t;
```

Se tiene el siguiente TDA de vector dinámico de pedidos:

```
/* Interfaz de un vector dinámico de pedidos.
Las posiciones del vector se numeran desde 0. */
struct vector_din;
typedef struct vector_din vector_din_t;

// Post: Crea un vector dinámico de pedidos. Devuelve un puntero a un vector
vacío.
// Al terminar de usarlo este vector debe ser destruido con vec_destruir().
// Devuelve NULL si no se pudo crear el vector.
vector_din_t* vec_crear();

// Pre: `vec` fue creado usando vec_crear().
// Post: Destruye el vector.
void vec_destruir(vector_din_t* vec);

// Pre: `vec` fue creado usando vec_crear().
// Post: Agrega un pedido al final del vector.
// Devuelve true si se pudo agregar, false en caso contrario.
bool vec_guardar(vector_din_t* vec, producto_tomacco_t producto);

// Pre: `vec` fue creado usando vec_crear().
// Post: Devuelve un puntero al elemento en la posición `pos` del vector.
// Si `pos` es inválida, devuelve NULL.
// Una posición es inválida si es mayor o igual al largo del vector.
producto_tomacco_t* vec_obtener(vector_din_t* vec, size_t pos);

// Pre: `vec` fue creado usando vec_crear().
// Post: Elimina el pedido en la posición `pos` del vector.
// Devuelve true si se pudo eliminar, false si la posición es inválida.
bool vec_eliminar(vector_din_t* vec, size_t pos);

// Pre: `vec` fue creado usando vec_crear().
// Post: Devuelve la cantidad de elementos en el vector.
size_t vec_largo(vector_din_t* vec);

// Pre: `vec` fue creado usando vec_crear().
```

```
// Post: Imprime los elementos del vector en orden.  
void vec_imprimir(vector_din_t* vec);
```

Se pide:

1. Homero llega a la granja y encuentra el vector de productos lleno. Crear una función que reciba el vector y devuelva cuántos productos pueden venderse legalmente:
  - a. Deben estar aprobados por Lisa.
  - b. Deben tener nivel de adicción menor o igual a 3.
2. Apu quiere separar los productos peligrosos de los normales. Crear un procedimiento que reciba el vector original y elimine todos los productos cuyo nivel de adicción sea mayor al valor recibido por parámetro.
3. Crear un procedimiento que reciba DOS vectores de productos y genere UN NUEVO vector que contenga todos los productos de ambos vectores:
  - a. primero los productos del primer vector
  - b. luego los del segundo vector

Los vectores originales deben quedar intactos.