

Ejercicios de final

—

Ejercicio 1 parte práctica

Homero tiene un montonazo de ideas para supertrajes del Hombre Pie (*Pai*). Por lo general pone el color rojo como principal porque es su color favorito, pero en otras ocasiones intenta variar porque sino se aburre de usar siempre el mismo.

Mañana el Hombre Pie (*Pai*) debe encargarle un nuevo traje a su modista para su próxima misión que es muy importante y quiere que se su idea más reciente que incluya el color rojo.

La idea de un supertraje está representado por el struct ***supertraje_t***:

```
typedef struct supertraje {  
    int antiguedad;  
    char color[MAX_COLOR];  
    bool es_ganador;  
} supertraje_t;
```

Se pide:

Dado un vector de supertrajes que contiene todos los supertrajes del Hombre Pie (*Pai*) y su tope, crear una función ***recursiva*** que devuelva la posición de la idea de traje que le va a encargar a su modista.

Ejercicio 1 parte teórica

- ¿Qué elementos debe tener una función recursiva?
- ¿Qué tipo de dato es el campo ***color*** del struct *supertraje*? ¿Qué particularidad tiene ese tipo de dato?

Ejercicio 2 parte práctica

Bart y Lisa deciden realizar una broma a Homero en el hotel donde están pasando sus vacaciones, junto a Marge, Maggie, y sus tías Patty y Selma.

La broma resultará exitosa si:

- Bart y Lisa están cerca, es decir se encuentran en el mismo piso.
- No son rodeados por los demás adultos (Marge y Patty y Selma), es decir su madre (Marge) y sus tías (Patty y Selma) no se encuentran en los pisos contiguos del piso en que están ellas, uno en cada uno.
- Su padre (Marge) no ve la broma durante su ejecución, es decir Homero no se encuentra en la misma habitación que Marge.

Si el hotel puede ser representado como una matriz de habitaciones donde una fila representa un piso, y una habitación puede ser representada con la siguiente estructura:

```
typedef struct habitacion{
    int numero;
    char ocupantes[MAX_OCUPANTES]; // inicial del nombre
    int cantidad_ocupantes;
} habitacion_t;
```

Se pide:

Dada una matriz de habitaciones que representa el hotel en el instante que se ejecuta la broma, crear una función que devuelva **true** si la broma resulta exitosa o **false** en caso contrario.

Aclaraciones:

Patty y Selma cuentan como una y su inicial es 'T' de tías.

Ejercicio 2 parte práctica

- a. ¿Qué son las pre condiciones de una función? ¿Qué son las post condiciones de una función? Dar ejemplos.
- b. Explicar si la siguiente afirmación es Verdadera o Falsa y por qué: "Dado un vector y su tope, al acceder al elemento `vector[tope]` estoy accediendo a basura".

Ejercicio 3 parte práctica

En esta Navidad Homero quiere hacer una buena acción y se ofreció a repartir los regalos navideños. Va a repartirlos siguiendo el orden que le dieron representada en una cola de niños. El problema es que tiene todos los regalos apilados en su "trineo" (su clásico auto rosa).

Dados los siguientes TDAs:

```
struct pila_regalos;
typedef struct pila_regalos pila_regalos_t;

struct regalo;
typedef struct regalo regalo_t;

// Pre: -
// Post: Crea un TDA pila_regalos en el heap y devuelve un puntero al mismo.
// Devuelve NULL si no la pudo crear.
pila_regalos_t* pila_regalos_crear();

// Pre: - La pila fue previamente creada con `pila_regalos_crear`.
//       - La pila ya está completamente inicializado.
// Post: Agrega un regalo **al final** de la pila.
void guardar_regalo(pila_regalos_t* pila, regalo_t regalo);

// Pre: - La pila fue previamente creada con `pila_regalos_crear`.
// Post: - Quita el regalo **del final** de la pila y lo devuelve mediante la
// referencia `regalo_quitado` o NULL si la pila estaba vacia.
void quitar_regalo(pila_regalos_t* pila, regalo_t* regalo_quitado);

// Pre: - La pila fue previamente creada con `pila_regalos_crear`.
// Post: - Devuelve true si la pila no contiene regalos o false en caso contrario.
bool pila_regalos_esta_vacia(pila_regalos_t* pila);

// Pre: La pila recibida fue previamente creado con `pila_regalos_crear`.
// Post: Libera la memoria que se reservó en el heap para la pila.
void pila_regalos_destruir(pila_regalos_t* pila);
```

```
struct cola_ninios;
typedef struct cola_ninios cola_ninios_t;

struct ninio;
typedef struct ninio ninio_t;

struct regalo;
typedef struct regalo regalo_t;

// Pre: -
// Post: Crea un TDA cola_ninios en el heap y devuelve un puntero al mismo.
// Devuelve NULL si no la pudo crear.
cola_ninios_t* cola_ninios_crear();

// Pre: - La cola fue previamente creada con `cola_ninios_crear`.
//       - La cola ya está completamente inicializado.
// Post: Agrega un ninio **al final** de la cola.
void guardar_ninio(cola_ninios_t* cola, ninio_t ninio);

// Pre: - La cola fue previamente creada con `cola_ninios_crear`.
// Post: - Quita el ninio **del principio** de la cola y lo devuelve
mediante la
// referencia `ninio_quitado` o NULL si la cola estaba vacia.
void quitar_ninio(cola_ninios_t* cola, ninio_t* ninio_quitado);

// Pre: - La cola fue previamente creada con `cola_ninios_crear`.
// Post: Devuelve true si el ninio que se encuentra al principio de la cola
pidió
// ese regalo o false en caso contrario.
bool ninio_pidio_regalo(cola_ninios_t* cola, regalo_t regalo);

// Pre: - La cola fue previamente creada con `cola_ninios_crear`.
// Post: - Devuelve true si la cola no contiene ninios o false en caso
contrario.
bool cola_ninios_esta_vacia(cola_ninios_t* cola);

// Pre: La cola recibida fue previamente creado con `cola_ninios_crear`.
// Post: Libera la memoria que se reservó en el heap para la cola.
void cola_ninios_destruir(cola_ninios_t* cola);
```

Se pide:

Crear una función que reciba la cola de niños y una pila de regalos y elimine de la cola a todos los niños que tienen su regalo en la pila. Además, también se debe eliminar el regalo entregado de la pila.

Es importante que, al finalizar la función, tanto la pila como la cola queden en el mismo orden relativo en el que empezaron.

Ejemplo:

Si tenemos la siguiente pila y cola inicial:

Regalo1

Regalo2

Regalo3

Regalo4

NiñoA NiñoB NiñoC NiñoD

Y sabemos que el niño A pidió el regalo 11 (que no está en la pila), el B el 3, C el 1 y D el 8 (que no está en la pila). Al finalizar la pila y cola quedan:

Regalo2

Regalo4

NiñoA NiñoD

Ejercicio 3 parte teórica

¿Qué diferencia hay entre un puntero y una referencia en C?

Ejercicio 4 parte práctica

El autobús escolar se averió por lo que Bart y Lisa deben ir y volver de su casa a la escuela y viceversa caminando. A la ida van juntos, pero al tener distintos horarios de salida a la vuelta no.

Bart cuando vuelve solo va juntando municiones para su resortera, se distrae y siempre termina caminando de más.

Lisa le dejó un mapa para que vuelva a casa caminando lo menos posible. El mapa es una porción de Springfield en forma matriz del tipo de dato **edificio_t**, el cual tiene la siguiente estructura:

```
typedef struct edificio {
    char descripcion[MAX_DESCRIPCION]; // "escuela", "casa de ned",
    "kiosco", etc
    char movimiento; // 'W', 'S', 'D' o 'A'
    int cantidad_municiones;
} edificio_t;
```

Ella colocó la escuela en la posición (0,0) del mapa para que le sea más fácil ubicarse para arrancar.

Se pide:

Crear una función **recursiva** que recorra el mapa simulando el camino a casa que tiene que hacer Bart, devolviendo la descripción del **edificio_t** por el cual pasó y recolectó más municiones.

Aclaraciones:

- Bart termina su recorrido cuando el **edificio_t** en el que se encuentra es "**casa de la familia simpson**" y debe recolectar también las municiones que encuentre allí.
- Todos los **movimientos** seteados por Lisa son acordes, por lo que nunca le indicará moverse por fuera del mapa.

Ejercicio 4 parte teórica

Explicar las diferencias entre las operaciones de mezcla, unión, intersección y diferencia entre archivos. ¿Qué ocurre si uno de los archivos es más largo que el otro en cada una de esas operaciones?