Nombre	APELLIDOS (en mayúsculas)	Nota

TAD

Ejercicio 1. (3 ptos) Se desea implementar un TAD que represente un catálogo de coches (TAD *Catalogue*), que estará compuesta por el año del catálogo, y el conjunto de como máximo 50 marcas diferentes (TAD *Brand*) que hay dentro. Cada marca está definida por su nombre (máximo 1024 caracteres), y la lista de como máximo 20 modelos que fabrica (TAD *Model*). Cada modelo contiene su nombre (máximo 1024 caracteres), su longitud y anchura, y sus como máximo 10 versiones comercializadas (TAD *Version*). Por último, cada versión contiene su nombre (máximo 1024 caracteres), y su potencia.

(1 ptos) a) Escribir en C las estructuras de datos y las definiciones de nuevos tipos necesarias para implementar los TAD anteriores, garantizando al máximo la abstracción de los detalles de la implementación. Indicar en qué fichero se localizaría cada una de ellas.

(2 ptos) b) Escribir el código C de una función que, dado un catálogo, muestre todos los coches que contiene, devolviendo el número de caracteres impresos o `-1` en caso de error. En concreto, imprimirá el nombre de cada marca a modo de título, y para cada versión de cada modelo de esa marca, el nombre del modelo, sus dimensiones, el nombre de la versión y su potencia, como se muestra en el siguiente ejemplo:

```
Bord
    Siesta 400x173 Basic 75
    Siesta 400x173 Advanced 125
    Pocus 440x180 Access 90
    Pocus 440x180 Premium 150
Ceat
    Neon 430x179 Unique 275
```

Indicar en qué fichero se localizaría la nueva función.

Nota. *Se puede asumir la existencia de primitivas básicas en todos los TAD (acceso directo a los campos mediante* setters *y* getters*). Si se considera necesario, se pueden implementar primitivas auxiliares en cualquier TAD, indicando dónde se localizarían.*

Solución del Ejercicio 1.

a) **TAD *Version***

En `version.c`:

```
struct _Version {
  char name[MAX_WORD];
  int hp;
};
```

En `version.h`:

```
typedef struct _Version Version;
```

TAD *Model*

En `model.c`:

```
struct _Model {
  char name[MAX_WORD];
  int length;
  int width;
  Version *versions[MAX_VERSION];
  int n_versions;
};
```

En `model.h`:

```
typedef struct _Model Model;
```

TAD *Brand*

En `brand.c`:

```
struct _Brand {
  char name[MAX_WORD];
  Model *models[MAX_MODEL];
  int n_models;
};
```

En `brand.h`:

```
typedef struct _Brand Brand;
```

TAD *Catalogue*

En `catalogue.c`:

```
struct _Catalogue {
  int year;
  Brand *brands[MAX_BRAND];
  int n_brands;
};
```

En `catalogue.h`:

```
typedef struct _Catalogue Catalogue;
```

b) En `catalogue.c`:

```
int catalogue_print(Catalogue *c) {
  int i, n_char, n_char_total = 0;

  if (!(c)) {
    return INV_CHARS;
  }
```

```
  for (i = 0; i < c->n_brands; i++) {
    n_char = brand_print(c->brands[i]);
    if (n_char < 0) {
      return INV_CHARS;
    }
    n_char_total += n_char;
  }

  return n_char_total;
}
```

En `brand.c`:

```
int brand_print(Brand *b) {
  int i, n_char, n_char_total = 0;

  if (!(b)) {
    return INV_CHARS;
  }

  n_char_total = printf("%s\n", b->name);
  if (n_char_total < 0) {
    return INV_CHARS;
  }

  for (i = 0; i < b->n_models; i++) {
    n_char = model_print(b->models[i]);
    if (n_char < 0) {
      return INV_CHARS;
    }
    n_char_total += n_char;
  }

  return n_char_total;
}
```

En `model.c`:

```
int model_print(Model *m) {
  int i, n_char, n_char_total = 0;

  if (!(m)) {
    return INV_CHARS;
  }

  for (i = 0; i < m->n_versions; i++) {
    n_char = printf("\t%s %dx%d ", m->name, m->length, m->width);
    if (n_char < 0) {
      return INV_CHARS;
    }
    n_char_total += n_char;

    n_char = version_print(m->versions[i]);
    if (n_char < 0) {
      return INV_CHARS;
    }
    n_char_total += n_char;
  }

  return n_char_total;
}
```

En `version.c`:

```
int version_print(Version *v) {
  if (!(v)) {
    return INV_CHARS;
  }

  return printf("%s %d\n", v->name, v->hp);
}
```

Nombre	APELLIDOS (en mayúsculas)	Nota

Pilas y expresiones

Ejercicio 2. 4 ptos.

3 ptos. a) La función `stack_count_if()` devuelve el número de items de una pila que satisfacen una determinada condición. Por tanto, `stack_count_if()` debe recibir dos parámetros de entrada: la pila y una función que se evalúa como `True` si el elemento de pila satisface la condición o `False` en caso contrario.
Proporcione el código, con control de errores, de la función `stack_count_if()`. La función, además, debe devolver `-1` si se produce algún error. En cualquier caso, al finalizar la función, la pila debe quedar en su estado original. Asuma la interfaz habitual del TAD Stack con sus condiciones de error. *No se permite acceder a la estructura de datos, deben usarse solo las funciones de la interfaz pública.* No se considerarán válidas soluciones no eficientes.

Ejemplo

Suponga, por ejemplo, que se ha implementado la siguiente función:

```
Boolean es_par(void *item) {
    if (*(int*)item % 2 == 0)
        return True;
    return False;
}
```

entonces cuando se invocase a `stack_count_if(s, es_par)`, donde `s` fuese una pila que almacenase referencias a enteros, se obtendría el total de enteros pares presentes en la pila.

1 ptos. b) Transformar la siguiente expresión infija a forma postfija, mostrando cada paso de la conversión: `(A + B * C + D)/ F;`

Símbolo	Expresion	Pila

Solución del Ejercicio 2.

a)

```
int stack_count_if(Stack *s, Boolean (*fun)(void *)) {
  Status st = OK;
  int contador = 0;
  Stack *s_aux = NULL;

  if (!s || !fun) return -1;

  s_aux = stack_ini();
  if (!s_aux)
      return -1;

  while (is_empty(s) == False && st == OK){
      ele = stack_pop(s);
      st = stack_push(s_aux, ele);
      if (fun(ele) == True)
          contador++;
  }

  if (st == ERROR)
      stack_push(s, ele);

  while (is_empty(s_aux) == False){
      ele = stack_pop(s_aux);
      stack_push(s, ele);
  }

  stack_free(s_aux);

  if (st == ERROR)
      return -1;
  return contador;
}
```

b)

Símbolo	Expresion	Pila
	-	-
(	-	(
A	A	(
+	A	(+
B	A B	(+
*	A B	(+*
C	A B C	(+*
+	A B C * +	(+
D	A B C * + D	(+
)	A B C * + D +	-
/	A B C * + D +	/
F	A B C * + D + F	/
;	A B C * + D + F /	-

Cuando se lee el símbolo de fin de cadena `';'` se vacía la pila volcando sus elementos en la expresión de salida. La expresión final es por tanto `A B C * + D + F / .`

Nombre	APELLIDOS (en mayúsculas)	Nota

Colas

Ejercicio 3. Dada la implementación de cola vista en clase:

```
#define MAX_QUEUE 8
struct _Queue {
  void *data[MAX_QUEUE];
  int front;
  int rear;
};
typedef struct _Queue Queue;
```

1 ptos a) Define una función `int queue_size(Queue *q)` que, accediendo a esta EdD, calcule *de forma eficiente* el número de elementos presentes en la cola. Justifica tu implementación con dibujos para el caso `q->front < q->rear` y el caso `q->front > q->rear`.

2 ptos b) Dada la EdD anterior, y el siguiente tipo de función:

```
typedef int (*print_elem_fn) (FILE *fp, const void *elem);
```

para imprimir un elemento `elem` en `fp`, define la función `void queue_print(FILE *fp, Queue *q, print_elem_fn print_elem)` que imprime todos los elementos de la cola `q` en `fp` usando la función `print_elem` que se le pasa como argumento. La implementación de `void queue_print` **debe** acceder a la EdD de queue.

Solución del Ejercicio 3.

1 ptos a)

```
int queue_size(Queue *q) {
    if (q == NULL) return 0;
    return (MAX_QUEUE + q->rear - q->front) % MAX_QUEUE;
}
```

2 ptos b)

```
void queue_print(FILE *fp, Queue *q, print_elem_fn print_elem) {
  int i;

  if (!q || !print_elem) return;

  i = q->front;
  while (i != q->rear) {
    print_elem(fp, q->data[i]);
    i = (i+1) % MAX_QUEUE;
  }

}
```