Programación II. Examen parcial. Marzo 2019.

Apellidos (mayúsculas):_____Nombre:_____

Ejercicio 2 Suponga que se dispone de las bibliotecas de los TADs, *Pila*, *Cola* y *Element* (esto es, se dispone de los correspondientes ficheros .h y .o pero **no** de los .c). Se desea crear el TAD *StackOrQueue* para, aprovechando las bibliotecas anteriores, gestionar una pila o una cola dependiendo de como se inicialice el TAD.

La interfaz delTAD *StackOrQueue* deberá incluir las siguientes funciones

```
/*Crea una pila o una cola en función del valor Type */
StackOrQueue * stackorqueue_ini(Type);
void stackorqueue_destroy(StackOrQueue *);

Status stackorqueue_push(StackOrQueue *, const Element *);
Element * stackorqueue_pop(StackOrQueue *);

int stackorqueue_print(FILE *, const StackOrQueue *;
```

donde Type se ha definido como:

```
typedef enum {LIFO=0, FIFO=1} Type;
```

Las funciones incluidas ende las interfaces de Pila, Cola son las habituales `(stack_ini, stack_destroy, stack_push, stack_pop, stack_isEmpty y stack_isFull` para la pila y `queue_ini, queue_destroy, queue_insert, queue_extract, queue_isEmpty y queue_isFull` para la cola). Todas las funciones siguen los mismos convenios que en prácticas. Para Element recordad su interfaz:

```
Element * element_ini(); // Reserva memoria
void element_destroy(Element *);
Status element_setInfo(Element *, void*); //Reserva memoria
void * element_getInfo(const Element *);
Element * element_copy(const Element *);  // Reserva memoria
int element_cmp(const Element *, const Element *);
```

a) Proporcione el encabezamiento de los ficheros StackOrQueue.h y StackOrQueue.c (el comienzo en C sin incluir los prototipos de las funciones)
b) Defina una estructura de datos adecuada para implementar el TAD StackOrQueue
c) Proporcione la implementación de las funciones StackOrQueue_ini y StackOrQueue_push

APARTADO A

```
// En stackorqueue.h
 #ifndef _STACKORQUEUE_H
 #define _STACKORQUEUE_H

 typedef struct  _StackOrQueue StackOrQueue
 typedef enum {LIFO=0, FIFO=1} Type;
// En  stackorqueue.c
 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <errno.h>

#include "pila.h"
#include "cola.h"
#include  "stackorqueue.h"

 extern int errno;
```

APARTADO B

```
struct _StackOrQueue {

  void *datos;
  Type tipo;
}
```

APARTADO C

```
StackOrQueue * StackOrQueue_ini(Type t) {
    StackOrQueue *p = NULL;

    p = (StackOrQueue *) malloc(sizeof(StackOrQueue));
    if (NULL == p) {
        fprintf(stderr, "%s", strerror(errno) );
        return NULL;
    }

    //se guarda el tipo de TAD (Pila/Cola) que se almacena
    p->t = t;

    // inicializo lo que corresponda
    if (FIFO == t)
        p->datos = queue_ini();
    else if (LIFO == t)
        p->datos = stack_ini();
    else
        p->datos = NULL;

    // Control de errores
    if (NULL == p->datos) {
       free(p);
       return NULL;
    }
    return p;
}

Status StackOrQueue_push (Element *ele, StackOrQueue *s) {
    if (!ele || !s) return ERROR;

    if (s->t == LIFO)
       return  stack_push(ele, (PILA *)s->datos) ;
    else
       return  queue_push(ele, (COLA *)s->datos) ;
}
```

ALTERNATIVA 1 a los apartados B y C (Se considerará válida, aunque es “un poquito peor ”)

```
StackOrQueue * StackOrQueue_ini(Type t) {
    StackOrQueue *p = NULL;

    p = (StackOrQueue *) malloc(sizeof(StackOrQueue));
    if (NULL == p) {
        fprintf(stderr, "%s", strerror(errno) );
        return NULL;
    }

    //se guarda el tipo de TAD (Pila/Cola) que se almacena
    p->t = t;

    // inicializo lo que corresponda
    if (FIFO == t) {
        p->cdatos = queue_ini();
        p→pdatos = NULL;
    }
    else if (LIFO == t) {
        p->pdatos = stack_ini();
        p->cdatos = NULL;
    }
    else   {
        p->pdatos = NULL;
        p→cdatos = NULL;
```

```
struct _StackOrQueue {
  Pila *pdatos;
  Cola *cdatos;
  Type tipo;
}
```

```
    }

    // Control de errores
    if (NULL == p->pdatos && NULL == p->cdatos) {
       free(p);
       return NULL;
    }
    return p;
}

Status StackOrQueue_push (Element *ele, StackOrQueue *s) {
    if (!ele || !s) return ERROR;

    if (s->t == LIFO)
       return  stack_push(ele, s->pdatos) ;
    else
       return  queue_push(ele, s->cdatos) ;
}
```

ALTERNATIVA 2 a los apartados B y C (También se considerará válida, aunque es "aun un poquito peor")

```
StackOrQueue * StackOrQueue_ini(Type t) {
    StackOrQueue *p = NULL;

    p = (StackOrQueue *) malloc(sizeof(StackOrQueue));
    if (NULL == p) {
        fprintf(stderr, "%s", strerror(errno) );
        return NULL;
    }

    // inicializo lo que corresponda
    if (FIFO == t) {
        p->cdatos = queue_ini();
        p→pdatos = NULL;
    }
    else if (LIFO == t) {
        p->pdatos = stack_ini();
        p->cdatos = NULL;
    }
    else   {
        p->pdatos = NULL;
        p→cdatos = NULL;
    }

    // Control de errores
    if (NULL == p->pdatos && NULL == p->cdatos) {
       free(p);
       return NULL;
    }
    return p;
}

Status StackOrQueue_push (Element *ele, StackOrQueue *s) {
    if (!ele || !s) return ERROR;

    if (NULL == s->pdatos)
       return  queue_push(ele, s->cdatos) ;
    else
       return  stack_push(ele, s->pdatos) ;
}
```

```
struct _StackOrQueue {
  Pila *pdatos;
  Cola *cdatos;
}
```