Ejercicios Colas

Ejercicio 1. Número de elementos. Implementa una función derivada (sin acceder a la estructura de datos) que calcule el número de elementos de la cola:

```
int queue_num_elems(Queue *q){
  Queue *aux = NULL;
  int size = 0;

  if (!q) return -1;

  aux = queue_init();
  if (!aux) return -1;

  for (size=0, !stack_isEmpty(q), size++)
    queue_push(aux, queue_pop(q));

  while(!stack_isEmpty(aux))
    queue_push(q, queue_pop(aux));

  queue_free(aux);

  return size;
}
```

Ejercicio 2. Implementación de una cola utilizando dos pilas. Proporciona una implementación del TAD `Queue` a partir de dos pilas.

```
typedef struct _Queue{
  Stack *s;
  Stack *aux;
};

Queue *queue_init(){
  Queue *q = NULL;

  q = (Queue*)malloc(sizeof(Queue));
  if (!q) return NULL;

  q->s = stack_init();
  q->aux = stack_init();
  if (!(q->s) || !(q->aux)){
    queue_free(q);
    return NULL;
  }

void queue_free(Queue *q){
  if (!q) return;

  stack_free(q->s);
  stack_free(q->aux);
  free(q);
}

Status queue_push(Queue *q, void *elem){
  Status st = OK;
  void *e = NULL;
  if (!q || !elem) return ERROR;

  if (stack_isEmpty(q->s)) return ERROR;

  while(!stack_isEmpty(q->s) && st == OK){
    e = stack_pop(q->s);
    st = stack_push(q->aux, e);
```

```
  }

  if (st == ERROR)
    stack_push(q->s, e);
  else
    st = stack_push(q->s, elem);

  while(!stack_isEmpty(q->aux)){
    e = stack_pop(q->aux);
    stack_push(q->s, e);
  }

  return st;
}

void *queue_pop(Queue *q){
  if (!q || queue_isEmpty(q)) return NULL;

  return stack_pop(q->s);
}

Bool queue_isEmpty(const Queue *q){
  if (!q) return TRUE;

  return stack_isEmpty(q->s);
}

void *queue_getFront(const Queue *q){
  if (!q || queue_isEmpty(q)) return NULL;

  return stack_top(q->s);
}

void *queue_getBack(const Queue *q){
  void *elem = NULL;

  if (!q || queue_isEmpty(q)) return NULL;

  while (!stack_isEmpty(q->s))
    stack_push(aux, stack_pop(s));

  elem = stack_top(aux);

  while(!stack_isEmpty(q->aux))
    stack_push(s, stack_pop(aux));

  return elem;
}

size_t queue_size(const Queue *q){
  if (!q) return -1;

  return stack_size(q->s);
}
```

Ejercicio 3. Implementación de una pila usando una cola. Proporciona una implementación del TAD `Stack` a partir del TAD `Queue`. La estructura de datos contará con una única cola `q`, con los datos ordenados de modo que el `front` sea el `top` de la pila

```
typedef struct _Stack{
  Queue *q;
};

Stack *stack_init(){
  Stack *s = NULL;

  s = (Stack*)malloc(sizeof(Stack));
  if (!s) return NULL;
```

```
  s->q = queue_init();
  if (!(s->q)){
    free(stack);
    return NULL;
  }

  return s;
}

void stack_free(Stack *s){
  if (!s) return;

  queue_free(s->q);
  free(s);
}

Bool stack_isEmpty(Stack *s){
  if (!s) return TRUE;

  return queue_isEmpty(s->q);
}

Status stack_push(Stack *s, void *elem){
  void *e = NULL;
  int i;
  size_t size;

  if (!s || !elem) return ERROR;

  size = queue_size(s->q);
  queue_push(q->s, elem);
  for (i=0; i<size; i++){
    e = queue_pop(q->s);
    queue_push(q->s, e);
  }

  return OK;
}

void *stack_pop(Stack *s){
  if (!s || stack_isEmpty(s)) return NULL;

  return queue_pop(s->q);
}

void *stack_top(Stack *s){
  if (!s || stack_isEmpty(s)) return NULL;

  return queue_getFront(s->q);
}

Bool stack_isFull(Stack *s){
  if (!s) return TRUE;

  return queue_isFull(s->q);
}

size_t stack_size(Stack *s){
  if (!s) return -1;

  return queue_size(s->q);
}
```

Ejercicio 4. Intercalar elementos de una cola. Implementa la función `Queue *interleave(Queue *q)` que recibe una cola `q` y devuelve otra cola con los elementos de la primera mitad y la segunda mitad de `q` intercalados.

```
Queue *interleave(Queue *q){
  Queue *aux = NULL, *new = NULL;
  void *e = NULL;
  size_t size;
  int i

  if (!q || queue_isEmpty(q)) return NULL;

  size = queue_size(q);
  aux = queue_init();
  new = queue_init();
  if (!aux || !new){
    queue_free(aux);
    queue_free(new);
    return NULL;
  }

  for (i=0; i<size/2; i++){
    e = queue_pop(q);
    queue_push(aux);
  }

  for (i=0; i<size; i++){
    if (i%2 == 0)
      e = queue_pop(aux);
    else
      e = queue_pop(q);
    queue_push(new, e);
  }

  queue_free(q);
  queue_free(aux);

  return new;
}
```

Ejercicio 5. Concatenar dos colas. Implementa la función `Status concatenate(Queue *qa, Queue *qb)` que recibe dos colas `qa` y `qb` y modifica la primera de ellas, `qa`, situando a continuación de su último elemento los elementos de la segunda, `qb`, en su orden propio. La cola `qb` debe quedar vacía si la concatenación de las dos colas se realiza con éxito. En caso de producirse algún error las colas deben quedar como estaban inicialmente.

```
Status concatenate(Queue *qa, Queue *qb){
  void *e = NULL;
  size_t size_qa, size_qb;
  Status st = OK;
  int i, j;

  if (!qa || !qb) return ERROR;
  if (queue_isEmpty(qb)) return OK;

  size_qa = queue_size(qa);
  size_qb = queue_size(qb);

  for (i=0; i<size_qb, st == OK; i++){
    e = queue_pop(qb);
    st = queue_push(qa, e);
  }

  if (st == ERROR){
    for (j=0; j<size_qa; j++){
      e = queue_pop(qa);
      queue_push(qa, e);
    }
    for (j=0; j<i; j++){
      e = queue_pop(qa);
```

```
      queue_push(qb, e);
    }
  }

  return st;
```