Programación II, 2018-2019

Escuela Politécnica Superior, UAM

Práctica 3: TAD Cola y TAD Lista

OBJETIVOS

- Familiarización con los TADs Cola y Lista, aprendiendo su funcionamiento y potencial.
- Implementación en C de los TADs Cola y Lista.
- Utilización de los TADs Cola y Lista para resolver problemas.

NORMAS

Igual que en prácticas anteriores, los ***programas*** que se entreguen deben:

- Estar escritos en ***ANSI C,*** siguiendo las **normas de programación** establecidas.
- Compilar sin errores ni *warnings*, estableciendo las banderas "-ansi" y "-pedantic" al compilar.
- **Ejecutarse sin problema en una consola de comandos**.
- Incorporar un adecuado **control de errores**; es justificable que un programa no admita valores inadecuados, pero no que se comporte de forma anómala con dichos valores.
- La ***memoria*** que se entregue debe elaborarse sobre el modelo propuesto y entregado con la práctica.

PLAN DE TRABAJO

Semana 1: código de P3_E1. Cada profesor indicará en clase cómo se realizará la entrega: papel, e-mail, Moodle, etc.

Semana 2: código de todos los ejercicios.

La entrega final se realizará a través de Moodle, **siguiendo escrupulosamente las instrucciones** indicadas en el enunciado de la práctica 0 referentes a la organización y nomenclatura de ficheros y proyectos. Se recuerda que el fichero comprimido que se debe subir debe llamarse **Px_Prog2_Gy_Pz**, siendo '**x**' el número de la práctica, '**y**' el grupo de prácticas y '**z**' el número de pareja (ejemplo de entrega de la pareja 5 del grupo 2161: P1_Prog2_G2161_P05.zip).

Las **fechas de subida a Moodle del fichero** son las siguientes:

- Los alumnos de ***Evaluación Continua, la semana del 8 de abril*** *(cada grupo puede realizar la entrega hasta las 23:30 h. de la noche anterior a su clase de prácticas*).
- Los alumnos de ***Evaluación Final,*** según lo especificado en la normativa.

PARTE 1. EJERCICIOS

Ejercicio 1 (P3_E1). TAD Cola. Implementación de colas genéricas.

a) Definición del TAD Cola (Queue). Implementación del TAD utilizando un array de forma circular: estructura de datos y primitivas.

Implementad en el fichero **queue.c** las primitivas del TAD Cola definidas en el archivo de cabecera **queue.h** (ver anexo 1) utilizando punteros a funciones, tal y como se hizo en el ejercicio 3 de la práctica 2 con las pilas. Los tipos Status y Bool se encuentran definidos en **types.h**. La constante MAXQUEUE indica el tamaño máximo de la cola. Su valor debe permitir desarrollar los ejercicios propuestos.

La estructura de datos elegida para implementar el TAD COLA consiste en un array de punteros genéricos (void*), dos enteros (front y rear) que indican, respectivamente, la posición del elemento que se va a extraer primero de la cola y la posición del primer hueco donde se puede insertar en la cola; además, en la estructura se guardan los punteros a las funciones que se utilizarán para destruir, copiar e imprimir los elementos de la cola en cada caso. El array se utilizará de forma circular. El nuevo tipo de datos y la estructura de datos se muestran a continuación:

```
/* En queue.h */
typedef struct _Queue Queue;
/* En queue.c */
struct _Queue {
        void* items [MAXQUEUE];
        int front;
        int rear;
        destroy_element_function_type   destroy_element_function;
        copy_element_function_type      copy_element_function;
        print_element_function_type     print_element_function;
};
```

b) Comprobación de la corrección de la definición del tipo Queue y sus funciones.

Con el objetivo de comprobar el correcto funcionamiento de las funciones anteriores, compilad el programa **p3_testqueue.c incluido en el anexo 2**, enlazadlo con vuestra implementación de colas y ejecutadlo con Valgrind para comprobar que no se producen errores de memoria.

Este programa recibe como argumento un fichero, cuya primera línea indica el número de nodos que se deben leer (siguiendo el formato que podéis ver a continuación), y los introduce uno a uno en una cola. A continuación, saca la mitad de los nodos introducidos, uno a uno, y los va introduciendo a su vez en una segunda cola. Finalmente, extrae uno a uno la otra mitad de los elementos que quedan en la cola y los introduce uno a uno en una tercera cola. Durante el proceso, va imprimiendo el contenido de las distintas colas como se muestra a continuación:

```
> cat nodos.txt
3
1 uno
2 dos
3 tres
```

```
> p3_testqueue nodos.txt
Inicialmente:
Cola 1: Queue vacia.
Cola 2: Queue vacia.
Cola 3: Queue vacia.
Añadiendo 3 elementos a q1:
Cola 1: Cola con 1 elementos:
[uno, 1, 0]
Cola 2: Queue vacia.
Cola 3: Queue vacia.

Cola 1: Cola con 2 elementos:
[uno, 1, 0][dos, 2, 0]
Cola 2: Queue vacia.
Cola 3: Queue vacia.

Cola 1: Cola con 3 elementos:
[uno, 1, 0][dos, 2, 0][tres, 3, 0]
Cola 2: Queue vacia.
Cola 3: Queue vacia.


<<<Pasando la primera mitad de Cola 1 a Cola 2
Cola 1: Cola con 2 elementos:
[dos, 2, 0][tres, 3, 0]
Cola 2: Cola con 1 elementos:
[uno, 1, 0]
Cola 3: Queue vacia.


<<<Pasando la segunda mitad de Cola 1 a Cola 3
Cola 1: Cola con 1 elementos:
[tres, 3, 0]
Cola 2: Cola con 1 elementos:
[uno, 1, 0]
Cola 3: Cola con 1 elementos:
[dos, 2, 0]

Cola 1: Queue vacia.
Cola 2: Cola con 1 elementos:
[uno, 1, 0]
Cola 3: Cola con 2 elementos:
[dos, 2, 0][tres, 3, 0]
```

c) Búsqueda de caminos en un grafo usando una cola (recorrido en anchura).

En la práctica anterior (P2_E4) se implementó un algoritmo para recorrer un grafo buscando el camino entre dos nodos del mismo (si lo había) utilizando una pila. En este ejercicio, se trata de explorar la utilización de colas con el mismo objetivo. Para ello, se sustituirá, en dicho algoritmo, la pila por una cola y se comprobarán las diferencias en el orden en que se exploran los nodos.

El programa se entregará en un fichero de nombre **p3_e1.c.** Sus argumentos de entrada y el formato de salida serán los mismos que en el ejercicio equivalente de la práctica 2.

Ejercicio 2 (P3_E2). Lista de enteros

a) Definición del TAD Lista Enlazada Circular (LEC). Implementación: estructura de datos y primitivas.

Implementad, en el fichero **list.c,** las primitivas del TAD Lista definidas en el archivo de cabecera **list.h**. La estructura de datos elegida para implementar el TAD Nodo (nodo de lista) consiste en una estructura con un campo que contiene un puntero a la información (sea del tipo que sea) y un puntero al siguiente elemento de la lista (es decir, un puntero al nodo siguiente). Por su parte, el TAD Lista se ha definido como una estructura que incluye un puntero al último nodo de la lista, el cual a su vez apuntará al primero, y los punteros a las funciones que se utilizarán para destruir, copiar, imprimir y comparar la información que alberguen los nodos de la lista en sus campos info (según del tipo que sean). Dichas estructuras se muestran a continuación:

```
/* En list.h */
typedef struct _List List;

/* En list.c */

typedef struct _NodeList { /* EdD privada, necesaria para implementar lista */
    void* info;
    struct _NodeList *next;
} NodeList;                /* Tipo NodeList privado */

struct _List {
    NodeList *last;  /*La LEC apunta al último nodo y el último al primero*/

    destroy_element_function_type destroy_element_function;
    copy_element_function_type copy_element_function;
    print_element_function_type print_element_function;
    cmp_element_function_type cmp_element_function;
};
```

Los prototipos de las primitivas de nodo y lista se incluyen en el anexo 3.

b) Comprobación de la corrección de la definición del tipo List y sus primitivas.

Comprueba el correcto funcionamiento de las primitivas anteriores implementando y ejecutando (paso a paso si es necesario) un programa **p3_e2.c** que contenga una función main que realice las siguientes acciones:

- Recibe un número entero como argumento de entrada (X).
- Inserta cada uno de los números desde X hasta 1 en dos listas, de la siguiente manera:
    - En la primera lista, si el número es par lo insertará por el comienzo y si es impar por el final.
    - En la segunda lista, inserta el número en orden (mantiene siempre la lista ordenada).
- Imprime ambas listas.
- Libera los recursos utilizados (se puede liberar directamente o bien utilizar las funciones de extracción para comprobar que funcionan bien).

Ejercicio 3 (P3_E3). Implementación del TAD Cola utilizando como estructura de datos una Lista Enlazada Circular

a) Implementa de nuevo el TAD Cola, ahora utilizando como estructura de datos una lista enlazada circular. Los prototipos de las funciones de Cola son los mismos que en ejercicios anteriores.

b) Prueba el correcto funcionamiento del TAD Cola implementado sobre una Lista utilizando esta nueva implementación para el ejercicio p3_testqueue.c incluido en el anexo 2. Entrega el makefile que contiene lo necesario para compilar, enlazar y ejecutar lo que corresponda para que p3_testqueue.c utilice esta nueva versión de colas.

Importante: Para evitar problemas de versiones al entregar todos los ejercicios juntos en la práctica, se pueden colocar estos ficheros en un **directorio aparte** o bien añadir una ele al nombre de cada uno de ellos: **queuel.h, queuel.c y p3_testqueuel.c**. (pregunta a tu profesor/a cómo lo prefiere).

PARTE 2. MEMORIA

2.1. Preguntas sobre la práctica

1. ¿Cuáles son las diferencias que se han encontrado entre utilizar pilas o utilizar colas para recorrer un grafo buscando si hay conexión o no entre dos nodos?
2. Sobre el ejercicio 3, si no hubiera sido necesario renombrar los ficheros para entregar las prácticas, suponiendo que se hubieran podido sobreescribir los ficheros que ya se tenían de ejercicios anteriores (queue.h, queue.c, p3_testqueue.c,...):
    a. ¿Qué ficheros se han tenido que modificar en el ejercicio 3a para pasar de la implementación de colas con el array a la implementación basada en listas? ¿Qué se ha modificado y por qué?
    b. ¿Qué ficheros se han tenido que modificar en el ejercicio 3b (con respecto al ejercicio 1b) para utilizar ahora las colas implementadas sobre listas? ¿Qué se ha modificado y por qué?

Anexo 1: queue.h

```
typedef struct _Queue Queue;
/* Tipos de los punteros a función soportados por la cola. Nota: podrían estar en elem_functions.h e incluirlo aquí   */
typedef void (*destroy_element_function_type)(void*);
typedef void (*copy_element_function_type)(const void*));
typedef int (*print_element_function_type)(FILE *, const void*);
/**-------------------------------------------------------------------
Inicializa la cola: reserva memoria para ella e inicializa todos sus elementos a NULL.
-------------------------------------------------------------------*/
Queue* queue_ini(destroy_element_function_type f1, copy_element_function_type f2, print_element_function_type f3);
/**-------------------------------------------------------------------
Libera la cola, liberando todos sus elementos.
-------------------------------------------------------------------*/
void queue_destroy(Queue *q);
/**-------------------------------------------------------------------
Comprueba si la cola está vacía.
-------------------------------------------------------------------*/
Bool queue_isEmpty(const Queue *q);
/**-------------------------------------------------------------------
Comprueba si la cola está llena.
-------------------------------------------------------------------*/
Bool queue_isFull(const Queue* queue);
/**-------------------------------------------------------------------
Inserta un nuevo nodo en la cola, reservando memoria nueva para él y haciendo una copia del elemento recibido.
-------------------------------------------------------------------*/
Status queue_insert(Queue *q, const void* pElem);
/**-------------------------------------------------------------------
Extrae un elemento de la cola. Devuelve directamente el puntero al elemento (no hace copia).
-------------------------------------------------------------------*/
void * queue_extract(Queue *q);
/**-------------------------------------------------------------------
Devuelve el número de elementos de la cola.
-------------------------------------------------------------------*/
int queue_size(const Queue *q);
/**-------------------------------------------------------------------
Imprime toda la cola, devolviendo el número de caracteres escritos.
-------------------------------------------------------------------*/
int queue_print(FILE *pf, const Queue *q);
```

Anexo 2: p3_testqueue.c

```
/*
 * File:   p3_testqueue.c
 * Author: Profesores de PROG2
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "queue.h"
#include "node.h"
#define MAX 255
extern int errno;

void mainCleanUp (Queue *q1, Queue *q2, Queue *q3, Node *pn, FILE *pf) {
    if (pn) node_destroy(pn);
    if (q1) queue_destroy (q1);
    if (q2) queue_destroy (q2);
    if (q3) queue_destroy (q3);
    if (pf) fclose (pf);
}
int main(int argc, char** argv) {

    FILE *pf=NULL;
    Queue *q1=NULL, *q2=NULL, *q3=NULL;
    Node *pn=NULL;
    int id, i, npoints, middle;
    char name[MAX];
    char s[MAX];

    if (argc < 2) {
        fprintf(stderr, "Error: Introduzca como argumento nombre del fichero con los nodos.\n");
        return EXIT_FAILURE;
    }
    pf = fopen(argv[1], "r");
    if (pf==NULL){
        fprintf (stderr, "Error apertura fichero: %s\n", strerror(errno));
        return EXIT_FAILURE;
    }
    if (fgets(s, MAX, pf)==NULL) {
       fprintf(stderr, "Error lectura de fichero: %s\n", strerror(errno));
       mainCleanUp (q1, q2, q3, pn, pf);
       return EXIT_FAILURE;
    }
    if (sscanf(s, "%d\n", &npoints) !=1 ){
       fprintf(stderr, "Error formato datos en fichero: %s\n", strerror(errno));
       mainCleanUp (q1, q2, q3, pn, pf);
       return EXIT_FAILURE;
    }
    pn = node_ini();
    if (pn==NULL) {
       fprintf(stderr, "Error: inicializacion de nodo.\n");
       mainCleanUp (q1, q2, q3, pn, pf);
       return EXIT_FAILURE;
    }
    q1 = queue_ini(node_destroy, node_copy, node_print);
    q2 = queue_ini(node_destroy, node_copy, node_print);
    q3 = queue_ini(node_destroy, node_copy, node_print);

    if (!q1 || !q2 || !q3) {
        fprintf(stderr, "Error: inicializacion de cola.\n");
        mainCleanUp (q1, q2, q3, pn, pf);
        return EXIT_FAILURE;
    }
```

```
    printf("Inicialmente:\n");
    printf("Cola 1: ");queue_print(stdout, q1);
    printf("Cola 2: ");queue_print(stdout, q2);
    printf("Cola 3: ");queue_print(stdout, q3);

    printf("Añadiendo %d elementos a q1:\n", npoints);
    for (i=0; i<npoints; i++){
       if (fgets(s, MAX, pf) == NULL){
            fprintf(stderr, "Error lectura de fichero: %s\n", strerror(errno));
            mainCleanUp (q1, q2, q3, pn, pf);
            return EXIT_FAILURE;
        }
       if(sscanf(s, "%d %s \n", &id, name) !=2 ){
            fprintf(stderr, "Error formato datos en fichero: %s\n", strerror(errno));
            mainCleanUp (q1, q2, q3, pn, pf);
            return EXIT_FAILURE;
        }
       node_setId(pn, id);
       node_setName(pn, name);
       if (queue_insert(q1, pn) == ERROR) {
            fprintf(stderr,"Error: inserción en cola.\n");
            mainCleanUp (q1, q2, q3, pn, pf);
            return EXIT_FAILURE;
        }
        printf("Cola 1: ");queue_print(stdout, q1);
        printf("Cola 2: ");queue_print(stdout, q2);
        printf("Cola 3: ");queue_print(stdout, q3);
        printf("\n");
    }

    node_destroy(pn); pn = NULL;
    printf("\n<<<Pasando la primera mitad de Cola 1 a Cola 2\n");
    npoints = queue_size(q1);
    middle = npoints/2;
    for(i=0; i< middle; i++){
       pn = queue_extract(q1);
       if (queue_insert(q2, pn)== ERROR) {
           fprintf(stderr, "Error: inserción en cola 2.\n");
           mainCleanUp (q1, q2, q3, pn, pf);
           return EXIT_FAILURE;
        }
       node_destroy(pn); pn = NULL;
       printf("Cola 1: ");queue_print(stdout, q1);
       printf("Cola 2: ");queue_print(stdout, q2);
       printf("Cola 3: ");queue_print(stdout, q3);
       printf("\n");
    }
    printf("\n<<<Pasando la segunda mitad de Cola 1 a Cola 3\n");
    for(i=npoints/2; i< npoints; i++){
       pn = queue_extract(q1);
       if (queue_insert(q3, pn) == ERROR) {
            fprintf(stderr,"Error: inserción en cola 3.\n");
            mainCleanUp (q1, q2, q3, pn, pf);
            return EXIT_FAILURE;
        }
       node_destroy(pn); pn = NULL;
       printf("Cola 1: ");queue_print(stdout, q1);
       printf("Cola 2: ");queue_print(stdout, q2);
       printf("Cola 3: ");queue_print(stdout, q3);
       printf("\n");
    }
    /* Liberar todo*/
    mainCleanUp (q1, q2, q3, pn, pf);
    return EXIT_SUCCESS;
}
```

Anexo 3: list.h

```
typedef struct _List List;

/* Tipos de los punteros a función soportados por la lista. Nota: podrían estar en elem_functions.h e incluirlo aquí  */
typedef void (*destroy_element_function_type)(void*);
typedef void (*copy_element_function_type)(const void*);
typedef int (*print_element _function_type)(FILE *, const void*);
typedef int (*cmp_element_function_type)(const void*, const void*);
/*El último tipo de funciones, cmp, serán aquellas que sirvan para comparar dos elementos, devolviendo un número positivo, negativo o cero según sea el primer argumento mayor, menor o igual que el segundo */

/* Inicializa la lista reservando memoria e inicializa todos sus elementos. */
List* list_ini (destroy_element_function_type f1, copy_element_function_type f2, print_element_function_type f3,
                cmp_element_function_type f4);

/* Libera la lista, liberando todos sus elementos. */
void list_destroy (List* list);

/* Inserta al principio de la lista realizando una copia de la información recibida. */
Status list_insertFirst (List* list, const void *pelem);

/* Inserta al final de la lista realizando una copia de la información recibida. */
Status list_insertLast (List* list, const void *pelem);

/* Inserta en orden en la lista realizando una copia del elemento. */
Status list_insertInOrder (List *list, const void *pelem);

/* Extrae del principio de la lista, devolviendo directamente el puntero al campo info del nodo extraído, nodo que finalmente es liberado. OJO: tras guardar la dirección del campo info que se va a devolver y antes de liberar el nodo, pone el campo info del nodo a NULL, para que no siga apuntando a la info a devolver y, por tanto, no la libere al liberar el nodo */
void * list_extractFirst (List* list);

/* Extrae del final de la lista, devolviendo directamente el puntero al campo info del nodo extraído, nodo que finalmente es liberado. OJO: tras guardar la dirección del campo info que se va a devolver y antes de liberar el nodo, pone el campo info del nodo a NULL, para que no siga apuntando a la info a devolver y, por tanto, no la libere al liberar el nodo */
void * list_extractLast (List* list);

/* Comprueba si una lista está vacía o no. */
Bool list_isEmpty (const List* list);

/* Devuelve la información almacenada en el nodo i-ésimo de la lista. En caso de error, devuelve NULL. */
const void* list_get (const List* list, int index);

/* Devuelve el número de elementos que hay en una lista. */
int list_size (const List* list);

/* Imprime una lista devolviendo el número de caracteres escritos. */
int list_print (FILE *fd, const List* list);
```