75.41 Algortimos y Programación II Curso 4
TDA Cola

Implementada como nodos enlazados

15 de septiembre de 2019

1. Enunciado

Se pide implementar una Cola como nodos enlazados. Para ello se brindan las firmas de las funciones públicas a implementar y se deja a criterio del alumno la creación de las funciones privadas del TDA para el correcto funcionamiento de la Cola cumpliendo con las buenas prácticas de programación.

2. cola.h

```c
#ifndef __COLA_H__
#define __COLA_H__

#include <stdbool.h>

typedef struct nodo {
        void* elemento;
        struct nodo* siguiente;
} nodo_t;

typedef struct cola {
        nodo_t* nodo_inicio;
        nodo_t* nodo_fin;
        int cantidad;
} cola_t;

/*
 * Crea una cola reservando la memoria necesaria
 * para almacenar la estructura.
 * Devuelve un puntero a la estructura cola_t creada.
 */
cola_t* cola_crear();

/*
 * Encola un elemento.
 * Devuelve 0 si pudo encolar o -1 si no pudo.
 */
int cola_encolar(cola_t* cola, void* elemento);

/*
 * Desencola un elemento.
 * Devuelve 0 si pudo desencolar o -1 si no pudo.
 */
int cola_desencolar(cola_t* cola);

/*
 * Determina si la cola está vacia.
 * Devuelve true si está vacía y false si no.
 * Si la cola no existe devolverá true.
 */
bool cola_vacia(cola_t* cola);

/*
 * Devuelve la cantidad de elementos almacenados en la cola.
 * Si la cola no existe devolverá 0.
 */
int cola_cantidad(cola_t* cola);
```

```c

/*
 * Devuelve el primer elemento de la cola o NULL en caso de estar
 * vacía.
 * Si la cola no existe devolverá NULL.
 */
void* cola_primero(cola_t* cola);

/*
 * Destruye la cola liberando toda la memoria reservada
 * por la cola.
 */
void cola_destruir(cola_t*);

#endif /* __COLA_H__ */
```

3. Compilación y Ejecución

El TDA entregado deberá compilar y pasar las pruebas dispuestas por la cátedra sin errores, adicionalmente estas pruebas deberán ser ejecutadas sin pérdida de memoria.

Compilación:

```
gcc *.c -o cola_ne -g -std=c99 -Wall -Wconversion -Wtype-limits -pedantic -Werror -O0
```

Ejecución:

```
valgrind --leak-check=full --track-origins=yes --show-reachable=yes ./cola_ne
```

4. Minipruebas

Se les brindará un lote de minipruebas, las cuales recomendamos fuertemente sean ampliadas ya que no son exhaustivas y no prueban los casos borde, solo son un ejemplo de como encolar, desencolar y qué debería verse en la terminal en el **caso feliz**.

Minipruebas:

```c
#include "cola.h"
#include <stdio.h>

int main(){
        cola_t* cola = cola_crear();

        int a=1, b=2, c=3, d=4;

        printf("Encolo el elemento %i\n", a);
        cola_encolar(cola, (void*)&a);

        printf("Encolo el elemento %i\n", b);
        cola_encolar(cola, (void*)&b);

        printf("Encolo el elemento %i\n", c);
        cola_encolar(cola, (void*)&c);

        printf("Encolo el elemento %i\n", d);
        cola_encolar(cola, (void*)&d);

        printf("La cola tiene %i elementos\n", cola_cantidad(cola));

        printf("Desencolo un elemento\n");
        cola_desencolar(cola);

        printf("¿La cola está vacía?: %s\n", cola_vacia(cola)?"SI":"NO");

        cola_primero(cola);
        printf("El primer elemento de la cola es: %i\n", *(int*)cola_primero(cola));

        cola_destruir(cola);

        return 0;
}
```

La salida por pantalla luego de correrlas con valgrind debería ser:

```
==28629== Memcheck, a memory error detector
==28629== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28629== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==28629== Command: ./cola_ne
==28629==
Encolo el elemento 1
Encolo el elemento 2
Encolo el elemento 3
Encolo el elemento 4
La cola tiene 4 elementos
Desencolo un elemento
¿La cola está vacía?: NO
El primer elemento de la cola es: 2
==28629==
==28629== HEAP SUMMARY:
==28629==     in use at exit: 0 bytes in 0 blocks
==28629==   total heap usage: 6 allocs, 6 frees, 1,112 bytes allocated
==28629==
==28629== All heap blocks were freed -- no leaks are possible
==28629==
==28629== For counts of detected and suppressed errors, rerun with: -v
==28629== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

5. Entrega

La entrega deberá contar con todos los archivos necesarios para compilar y ejecutar correctamente el TDA.

Dichos archivos deberán formar parte de un único archivo **.zip** el cual será entregado a través de la plataforma de corrección automática **Kwyjibo**.

El archivo comprimido deberá contar, además del TDA con:

- El archivo con las pruebas agregadas para comprobar el correcto funcionamiento del TDA.
- Un **Readme.txt** donde se deberá explicar qué es lo entregado, como compilarlo (línea de compilación), como ejecutarlo (línea de ejecución) y todo lo que crea necesario aclarar.
- El enunciado.