Nombre y Apellido: ..... N° Legajo: .....

Segundo Parcial de Programación Imperativa

14/06/2024

	Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*
Calificación	**/3.5**	**/3.5**	**/3**	

- **Condición mínima de aprobación: Sumar 5 (cinco) puntos.**
- **Los ejercicios que no se ajusten estrictamente al enunciado, no serán aceptados.**
- **No usar variables globales ni static.**
- **No es necesario escribir los #include**
- **Escribir en esta hoja Nombre, Apellido y Legajo**

Ejercicio 1

Se desea implementar un TAD para **administrar los mostradores de *checkIn*** de un aeropuerto. La numeración de los mostradores es entera positiva, iniciando en 0.

Para ello se definió la siguiente interfaz:

```
typedef struct countersCDT * countersADT;

/* Crea un sistema de administración de mostradores a partir de la cantidad
 * máxima #maxCounters de mostradores que tendrá el aeropuerto.
 * Se asume que la gran mayoría de los mostradores serán utilizados por las aerolíneas.
 * Si maxCounters es cero retorna NULL.
 */
countersADT newCounters(size_t maxCounters);

/* Ingresa al pasajero #passenger en la cola de espera del mostrador #counterNumber
 * para luego poder hacer el checkIn de su vuelo.
 * Un mismo pasajero puede ingresar múltiples veces al mismo mostrador o a varios
 * mostradores distintos.
 * Se asume que los nombres de los pasajeros son cortos.
 * Falla si #counterNumber es mayor o igual a la cantidad máxima de mostradores.
 */
void enterCounter(countersADT counters, size_t counterNumber, const char * passenger);

/* Funciones de iteración para poder consultar para un mostrador los pasajeros que están
 * esperando, en ORDEN DE LLEGADA (ver ejemplo en programa de prueba)
 */
void toBeginByCounter(countersADT counters, size_t counterNumber);
size_t hasNextByCounter(const countersADT counters, size_t counterNumber);
const char * nextByCounter(countersADT counters, size_t counterNumber);

/* Para cada mostrador, realiza el checkIn del pasajero que está esperando primero en la
 * cola, si es que hay al menos un pasajero esperando en ese mostrador.
 * Retorna un arreglo donde, para cada mostrador donde se hizo checkIn, se indica:
 * - El número del mostrador
 * - El nombre del pasajero que hizo checkIn en el mostrador y salió de la cola de espera
 * - La cantidad de pasajeros esperando en el mostrador luego de la operación
 * respetando el ORDEN ASCENDENTE por número de mostrador
 * Deja en un parámetro de entrada/salida la dimensión del vector de retorno
 */
```

```
struct checkInResult {
    size_t counterNumber;
    const char * checkedInPassenger;
    size_t waitingPassengers;
};
struct checkInResult * checkInCounters(countersADT counters, size_t * checkInResultDim);

/* Libera los recursos utilizados por el TAD
 */
void freeCounters(countersADT counters);
```

Se pide:

- **Implementar <u>todas las estructuras</u> necesarias**, de forma tal que **<u>todas las funciones puedan ser implementadas de la forma más eficiente posible</u>** (no sólo las que les pedimos implementar)
- **Implementar las siguientes funciones:**
    - **newCounters**
    - **enterCounter**
    - **toBeginByCounter**
    - **checkInCounters**

<u>Ejemplo de programa de prueba:</u>

```
int main(void) {
    countersADT terminalA = newCounters(10);
    enterCounter(terminalA, 5, "Foo");  // "Foo" ingresa a la cola del mostrador 5
    enterCounter(terminalA, 5, "Bar");
    enterCounter(terminalA, 5, "Bar");
    enterCounter(terminalA, 4, "Abc");
    enterCounter(terminalA, 4, "Xyz");

    // Se desea consultar los pasajeros que están esperando en el mostrador 5
    toBeginByCounter(terminalA, 5);
    // "Foo" es el primero que ingresó al mostrador 5
    assert(hasNextByCounter(terminalA, 5) == 1);
    assert(strcmp(nextByCounter(terminalA, 5), "Foo") == 0);
    assert(hasNextByCounter(terminalA, 5) == 1);
    assert(strcmp(nextByCounter(terminalA, 5), "Bar") == 0);
    toBeginByCounter(terminalA, 4);
    assert(hasNextByCounter(terminalA, 4) == 1);
    assert(strcmp(nextByCounter(terminalA, 4), "Abc") == 0);

    // Se pueden usar <u>en simultáneo</u> las funciones de iteración para consultar
    // los pasajeros que están esperando en distintos mostradores
    assert(hasNextByCounter(terminalA, 5) == 1);
    assert(strcmp(nextByCounter(terminalA, 5), "Bar") == 0);
    assert(hasNextByCounter(terminalA, 4) == 1);
    assert(strcmp(nextByCounter(terminalA, 4), "Xyz") == 0);
    assert(hasNextByCounter(terminalA, 5) == 0);
    assert(hasNextByCounter(terminalA, 4) == 0);

    // Se realiza, para cada mostrador, el checkIn del pasajero que está primero
    // esperando en la cola
    size_t checkInResultDim;
    struct checkInResult * checkInResultVec;
    checkInResultVec = checkInCounters(terminalA, &checkInResultDim);
    assert(checkInResultDim == 2); // Se realizó el checkIn en dos mostradores: 4 y 5
    // Resultado del checkIn en el mostrador 4
    assert(checkInResultVec[0].counterNumber == 4);
    assert(checkInResultVec[0].waitingPassengers == 1);
    assert(strcmp(checkInResultVec[0].checkedInPassenger, "Abc") == 0);
```

```
    // Resultado del checkIn en el mostrador 5
    assert(checkInResultVec[1].counterNumber == 5);
    assert(checkInResultVec[1].waitingPassengers == 2);
    assert(strcmp(checkInResultVec[1].checkedInPassenger, "Foo") == 0);
    free(checkInResultVec);

    toBeginByCounter(terminalA, 5);
    toBeginByCounter(terminalA, 4);
    assert(strcmp(nextByCounter(terminalA, 5), "Bar") == 0);
    assert(strcmp(nextByCounter(terminalA, 4), "Xyz") == 0);

    checkInResultVec = checkInCounters(terminalA, &checkInResultDim);
    assert(checkInResultDim == 2); // Se hizo checkIn en dos mostradores: 4 y 5
    // Resultado del checkIn en el mostrador 4
    assert(checkInResultVec[0].counterNumber == 4);
    assert(checkInResultVec[0].waitingPassengers == 0);
    assert(strcmp(checkInResultVec[0].checkedInPassenger, "Xyz") == 0);
    // Resultado del checkIn en el mostrador 5
    assert(checkInResultVec[1].counterNumber == 5);
    assert(checkInResultVec[1].waitingPassengers == 1);
    assert(strcmp(checkInResultVec[1].checkedInPassenger, "Bar") == 0);
    free(checkInResultVec);

    toBeginByCounter(terminalA, 5);
    toBeginByCounter(terminalA, 4);
    assert(strcmp(nextByCounter(terminalA, 5), "Bar") == 0);
    assert(hasNextByCounter(terminalA, 4) == 0);

    checkInResultVec = checkInCounters(terminalA, &checkInResultDim);
    assert(checkInResultDim == 1);
    // Resultado del checkIn en el mostrador 5
    assert(checkInResultVec[0].counterNumber == 5);
    assert(checkInResultVec[0].waitingPassengers == 0);
    assert(strcmp(checkInResultVec[0].checkedInPassenger, "Bar") == 0);
    free(checkInResultVec);

    toBeginByCounter(terminalA, 5);
    toBeginByCounter(terminalA, 4);
    assert(hasNextByCounter(terminalA, 5) == 0);
    assert(hasNextByCounter(terminalA, 4) == 0);

    checkInResultVec = checkInCounters(terminalA, &checkInResultDim);
    assert(checkInResultDim == 0); // Ningún mostrador hizo checkIn
    assert(checkInResultVec == NULL);

    freeCounters(terminalA);
    return 0;
}
```

Ejercicio 2

Se desea implementar un TAD para **administrar los muelles de los puertos** de una ciudad donde se pueden amarrar y desamarrar embarcaciones. La numeración de los puertos es entera positiva, iniciando en 0. La numeración de los muelles es entera positiva, iniciando en 0.

Para ello se definió la siguiente interfaz:

```
typedef struct piersCDT * piersADT;

/* Crea un sistema de administración de muelles de los puertos de una ciudad
 */
```

```
piersADT newPiers(void);

/* Agrega el puerto #pierNumber y retorna 1
 * Falla si el puerto ya existe y retorna 0
 * Un puerto inicia sin muelles. Se asume un bajo porcentaje de puertos libres
 */
size_t addPier(piersADT piers, size_t pierNumber);

/* Agrega el muelle #dockNumber al puerto #pierNumber y retorna 1
 * Falla si el muelle ya existe en el puerto o si el puerto no existe y retorna 0
 * Un muelle inicia sin una embaración amarrada
 * Se asume un bajo porcentaje de muelles libres para cada puerto
 */
size_t addPierDock(piersADT piers, size_t pierNumber, size_t dockNumber);

/* Amarra una embaración en el muelle #dockNumber del puerto #pierNumber y retorna 1
 * Falla si el muelle ya estaba ocupado o si el muelle no existe en el puerto
 * o si el puerto no existe y retorna 0
 */
size_t dockShip(piersADT piers, size_t pierNumber, size_t dockNumber);

/* Indica si hay una embarcación amarrada en el muelle #dockNumber del puerto #pierNumber
 * Retorna:
 *  1 si hay una embarcación amarrada
 *  0 si no hay una embaración amarrada
 * -1 si el muelle no existe en el puerto o si el puerto no existe
 */
int shipInDock(const piersADT piers, size_t pierNumber, size_t dockNumber);

/* Indica la cantidad de embarcaciones amarradas en todos los muelles del puerto
 * #pierNumber
 * Falla si el puerto no existe y retorna -1
 */
size_t pierShips(const piersADT piers, size_t pierNumber);

/* Desamarra una embaración en el muelle #dockNumber del puerto #pierNumber y retorna 1
 * Falla si el muelle estaba libre o si el muelle no existe en el puerto
 * o si el puerto no existe y retorna 0
 */
size_t undockShip(piersADT piers, size_t pierNumber, size_t dockNumber);

/* Libera los recursos utilizados por el TAD
 */
void freePiers(piersADT piers);
```

Se pide:

- **Implementar <u>todas las estructuras</u> necesarias**, de forma tal que las funciones **`addPierDock`**, **`dockShip`**, **`shipInDock`**, **`pierShips`** y **`undockShip`** **<u>puedan ser implementadas de la forma más eficiente posible</u>**
- **Implementar las siguientes funciones:**
  - **`newPiers`**
  - **`addPierDock`**
  - **`dockShip`**
  - **`shipInDock`**

<u>Ejemplo de programa de prueba:</u>

```
int main(void) {
    piersADT piersAdt = newPiers();
    assert(addPier(piersAdt, 5) == 1); // Agrega el puerto 5
```

```
    assert(addPierDock(piersAdt, 5, 4) == 1); // Agrega el muelle 4 del puerto 5
    assert(addPierDock(piersAdt, 5, 7) == 1);
    assert(addPier(piersAdt, 10) == 1);
    assert(addPierDock(piersAdt, 10, 1) == 1);
    assert(addPierDock(piersAdt, 10, 7) == 1);

    // No hay una embarcación amarrada en el muelle 4 del puerto 5
    assert(shipInDock(piersAdt, 5, 4) == 0);
    assert(shipInDock(piersAdt, 5, 7) == 0);
    assert(pierShips(piersAdt, 5) == 0);
    assert(shipInDock(piersAdt, 10, 1) == 0);
    assert(shipInDock(piersAdt, 10, 7) == 0);
    assert(pierShips(piersAdt, 10) == 0);

    // Se amarra una embaración en el muelle 4 del puerto 5
    assert(dockShip(piersAdt, 5, 4) == 1);
    assert(shipInDock(piersAdt, 5, 4) == 1);
    assert(shipInDock(piersAdt, 5, 7) == 0);
    assert(pierShips(piersAdt, 5) == 1);
    assert(dockShip(piersAdt, 5, 7) == 1);
    assert(shipInDock(piersAdt, 5, 4) == 1);
    assert(shipInDock(piersAdt, 5, 7) == 1);
    assert(pierShips(piersAdt, 5) == 2);

    // Se desamarra la embaración del muelle 7 del puerto 5
    assert(undockShip(piersAdt, 5, 7) == 1);
    assert(shipInDock(piersAdt, 5, 4) == 1);
    assert(shipInDock(piersAdt, 5, 7) == 0);
    assert(pierShips(piersAdt, 5) == 1);

    freePiers(piersAdt);
    return 0;
}
```

Ejercicio 3

Se desea implementar un TAD que modela una **colección de elementos genéricos**.

El TAD debe permitir agregar elementos como así también borrarlos y contar con funciones para recuperar elementos en forma individual o grupal según ciertos requerimientos.

Para ello se definió la siguiente interfaz:

```
typedef struct dataCDT * dataADT;

typedef __________ elemType;

/* Crea un TAD de elementos genéricos
 */
dataADT newData(¿?);

/* Agrega el elemento #elem, si no estaba. Retorna 1 si lo agrega, 0 si no
 */
int addElement(dataADT data, elemType elem);

/* Elimina el elemento #elem, si estaba. Retorna 1 si lo elimina, 0 si no
 */
int deleteElement(dataADT data, elemType elem);

/* Retorna la cantidad de elementos
 */
size_t countElement(const dataADT data);
```

```
/* La función #filter retorna 1 si el elemento debe incluirse en la respuesta y 0 si no
 * Retorna un vector ordenado con aquellos elementos que cumplan con el criterio
 * Si no hay elementos que cumplan con el criterio retorna NULL y setea *dim en cero
 */
elemType * elems(const dataADT data, int (*filter) (elemType), size_t * dim);

/* Retorna 1 si el elemento #elem está entre los datos, 0 si no
 */
int isElement(const dataADT data, elemType elem);

/* Libera los recursos utilizados por el TAD
 */
void freeData(dataADT data);
```

donde **¿?** en el prototipo indica que **usted debe indicar qué parámetro/s es necesario agregar**.

Se pide:

- **Implementar todas las estructuras necesarias**
- **Implementar las siguientes funciones:**
  - **newData**
  - **addElement**
  - **deleteElement**
  - **elems**

Ejemplo de programa de prueba, asumiendo que **elemType** es **int**:

```
int even(int n) {
   return n % 2 == 0;
}

int main(void) {
   dataADT data = newData(¿?); // Almacenaremos enteros ordenados en forma ascendente
   assert(addElement(data, 10) == 1);
   assert(addElement(data, 10) == 0);
   assert(addElement(data, 1) == 1);
   assert(addElement(data, 100) == 1);
   assert(addElement(data, 5) == 1);
   assert(addElement(data, 15) == 1);
   assert(addElement(data, 20) == 1);
   assert(countElement(data) == 6);

   size_t dim;
   int * v = elems(data, even, &dim);
   assert(dim == 3);
   assert(v[0] == 10 && v[1] == 20 && v[2] == 100);
   free(v);
   assert(countElement(data) == 6);

   assert(deleteElement(data, 20) == 1);
   assert(countElement(data) == 5);
   assert(deleteElement(data, 20) == 0);

   v = elems(data, even, &dim);
   assert(dim == 2);
   assert(v[0] == 10 && v[1] == 100);
   free(v);

   freeData(data);
   return 0;
}
```