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

Segundo Parcial de Programación Imperativa

16/06/2023

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

- **<u>Condición mínima de aprobación: Sumar 5 (cinco) puntos.</u>**
- **Los ejercicios que no se ajusten estrictamente al enunciado, <u>no serán aceptados</u>.**
- **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 las pistas y los despegues de aviones de un aeropuerto**. Cada pista se identifica con un número entero positivo, siendo la primera pista la número 1, la segunda la número 2, etc.

Para ello se definió la siguiente interfaz:

```
typedef struct airportCDT * airportADT;

/* Crea un sistema de administración de pistas y despegues de aviones
** de un aeropuerto.
** El sistema inicia sin pistas.
*/
airportADT newAirport(void);

/* Agrega una pista de despegue con el identificador runwayId.
** La pista inicia sin aviones.
** Retorna la cantidad actual de pistas en el sistema o -1 si falla.
** Falla si existe una pista con el identificador runwayId.
*/
int addRunway(airportADT airportAdt, size_t runwayId);

/* Agrega al final de la pista de despegue con el identificador
** runwayId al avión de matrícula registration
** y retorna la cantidad actual de aviones en la pista o -1 si falla.
** Falla si la pista no existe.
*/
int addPlaneToRunway(airportADT airportAdt, size_t runwayId, const char * registration);

/* Elimina al avión que se encuentra al principio de la pista de
** despegue con el identificador runwayId
** y retorna la matrícula del avión eliminado o NULL si falla.
** Falla si la pista no existe.
** Falla si no hay aviones en la pista.
*/
char * takeOff(airportADT airportAdt, size_t runwayId);
```

```
/* Retorna un arreglo con las matrículas de los aviones que se
** encuentran en la pista de despegue con el identificador
** runwayId en orden inverso al orden de despegue (el último elemento
** del arreglo debe coincidir con el valor de retorno de una
** invocación a la función takeOff sobre esa pista).
** El arreglo debe ** contar con una cadena vacía "" como marca de fin.
** Si la pista no existe retorna NULL.
*/
char ** pendingFlights(airportADT airportAdt, size_t runwayId);

/* Libera los recursos utilizados por el sistema de administración de
** pistas y despegues de aviones de un aeropuerto.
*/
void freeAirport(airportADT airportAdt);
```

Se pide:

- **Implementar todas las estructuras necesarias**, de forma tal que las funciones **`addRunway`**, **`addPlaneToRunway`** y **`takeOff`** puedan ser implementadas de la forma más eficiente posible.
- **Implementar las siguientes funciones:**
  - **`newAirport`**
  - **`takeOff`**
  - **`pendingFlights`**

Ejemplo de programa de testing

```
int
main(void) {
    // Crea un sistema sin pistas
    airportADT airportAdt = newAirport();

    // Agrega la pista con el id 5
    assert(addRunway(airportAdt, 5) == 1);
    assert(addRunway(airportAdt, 0) == 2);

    // Falla porque ya existe una pista con el id 5
    assert(addRunway(airportAdt, 5) == -1);

    // Falla porque no es válida la pista cero
    assert(addRunway(airportAdt, 0) == -1);

    // Agrega al final de la pista de id 5 al avión de matrícula AR01 y será
    // el primero en despegar
    char aux[20] = "AR01";
    assert(addPlaneToRunway(airportAdt, 5, aux) == 1);

    // Agrega al final de la pista de id 5 al avión de matrícula AC91 y será
    // el segundo en despegar
    strcpy(aux, "AC91");
    assert(addPlaneToRunway(airportAdt, 5, aux) == 2);

    strcpy(aux, "AC92");
    assert(addPlaneToRunway(airportAdt, 5, aux) == 3);
```

```
    // Obtiene los aviones por despegar de la pista de id 5 en orden inverso al
    // orden de despegue
    char ** aux = pendingFlights(airportAdt, 5);
    assert(!strcmp(aux[0], "AC92"));
    assert(!strcmp(aux[1], "AC91"));
    assert(!strcmp(aux[2], "AR01"));
    assert(!strcmp(aux[3], ""));
    free(aux);

    // Despega el primer avión de la pista de id 5
    assert(!strcmp(takeOff(airportAdt, 5), "AR01"));
    assert(!strcmp(takeOff(airportAdt, 5), "AC91"));
    assert(!strcmp(takeOff(airportAdt, 5), "AC92"));

    // Ya no quedan aviones por despegar en la pista de id 5
    assert(takeOff(airportAdt, 5) == NULL);

    // No existe la pista 3
    assert(takeOff(airportAdt, 3) == NULL);

    // No se puede agregar un avión a una pista que no existe
    assert(addPlaneToRunway(airportAdt, 3, "FL91") == -1);

    // No se puede agregar un avión a una pista que no existe
    assert(pendingFlights(airportAdt, 3) == NULL);

    freeAirport(airportAdt);
    puts("Despegue exitoso!");
    return 0;
}
```

Ejercicio 2

Se desea implementar un TAD para **administrar estaciones de alquiler de bicicletas**. Cada estación cuenta con "docks" donde se estacionan las bicicletas. Un dock puede estar ocupado o libre:

- **Ocupado**: significa que el dock tiene una bicicleta estacionada lista para alquilar.
- **Libre**: significa que el dock está disponible para que le estacionen una bicicleta al devolver una alquilada.

La numeración de las estaciones es entera positiva, iniciando en 1.

La numeración de los docks es propia de cada estación, además de entera positiva e incremental, iniciando en 0.

```
typedef struct bikeSharingCDT * bikeSharingADT;

/* Crea un sistema de administración de estaciones de alquiler de bicicletas a partir de
** la cantidad máxima de estaciones que soportará.
** El sistema inicia sin estaciones.
** Cada estación tendrá un id entre 1 y maxStationsCount inclusive.
** Se asume que la cantidad de estaciones que se agregarán es cercana a maxStationsCount
*/
bikeSharingADT newBikeSharing(size_t maxStationsCount);
```

```
/* Agrega una estación de alquiler de bicicletas con el identificador stationId
** con una cantidad de docks docksCount. La estación inicia con bicicletas en
** todos sus docks
** Retorna la cantidad actual de estaciones en el sistema o -1 si falla.
** Falla si el identificador stationId no está en el rango [1, maxStationsCount].
** Falla si existe una estación con el identificador stationId.
*/
int addStation(bikeSharingADT bikeSharingAdt, size_t stationId, size_t docksCount);

/* Alguien retira la bicicleta del dock con identificador dockId de la estación con
** identificador stationId.
** Retorna 0 si se pudo hacer o -1 si falla.
** Falla si la estación no existe.
** Falla si el dock no existe.
** Falla si en el dock no había una bicicleta
*/
size_t rentBike(bikeSharingADT bikeSharingAdt, size_t stationId, size_t dockId);

/* Retorna la cantidad de docks ocupados en la estación con identificador stationId
** o -1 si la estación no existe
*/
size_t bikesAvailable(bikeSharingADT bikeSharingAdt, size_t stationId);

/* Se ocupa el dock con identificador dockId de la estación con identificador stationId,
** porque se coloca una bicicleta en el mismo
** Retorna 0 si se pudo ocupar el dock o -1 si falla.
** Falla si la estación no existe.
** Falla si el dock no existe.
** Falla si el dock está ocupado (ya había una bicicleta en el mismo)
*/
size_t returnBike(bikeSharingADT bikeSharingAdt, size_t stationId, size_t dockId);

/* Retorna la cantidad de docks libres en la estación con identificador stationId o -1
** si la estación no existe.
*/
size_t docksAvailable(bikeSharingADT bikeSharingAdt, size_t stationId);

/* Libera todos los recursos utilizados */
void freeBikeSharing(bikeSharingADT bikeSharingAdt);
```

Se pide:

- **Definir todas las estructuras necesarias** de forma tal que las funciones **addStation, rentBike, bikesAvailable, returnBike y docksAvailable** puedan ser implementadas de la forma más eficiente posible.
- **Implementar las siguientes funciones:**
  - **newBikeSharing**
  - **addStation**
  - **rentBike**

Ejemplo de programa de testing

```
int
main(void) {
    // Crea un sistema sin estaciones con una cantidad máxima de 10 estaciones
    bikeSharingADT bsADT = newBikeSharing(10);

    // Falla porque el id 0 no está en el rango [1, 10]
    assert(addStation(bsADT, 0, 5) == -1);

    // Se agrega la estación de id 3 con 5 docks ocupados
    assert(addStation(bsADT, 3, 5) == 1);

    // Falla porque ya existe una estación con id 3
    assert(addStation(bsADT, 3, 10) == -1);
    assert(addStation(bsADT, 4, 10) == 2);
    assert(docksAvailable(bsADT, 3) == 0);
    assert(bikesAvailable(bsADT, 3) == 5);

    // Se alquila la bicicleta del dock 4 de la estación de id 3
    assert(rentBike(bsADT, 3, 4) == 0);

    // Los docks disponibles en la estación de id 3 ahora es 1
    assert(docksAvailable(bsADT, 3) == 1);

    // Los docks ocupados en la estación de id 3 ahora son 1 menos
    assert(bikesAvailable(bsADT, 3) == 4);

    // Falla porque no existe un dock 5 en la estación con id 3
    assert(rentBike(bsADT, 3, 5) == -1);
    assert(rentBike(bsADT, 4, 0) == 0);

    // Se devuelve una bicicleta al dock 4 de la estación de id 3
    assert(returnBike(bsADT, 3, 4) == 0);

    // Los docks disponibles en la estación de id 3 ahora son 1 menos
    assert(docksAvailable(bsADT, 3) == 0);

    // Los docks ocupados en la estación de id 3 ahora son 1 más
    assert(bikesAvailable(bsADT, 3) == 5);

    freeBikeSharing(bsADT);
    puts("OK!");
    return 0;
}
```

Ejercicio 3

Se desea implementar un TAD para registrar la cantidad de apariciones de elementos, de cualquier tipo

```
typedef struct elemCountCDT * elemCountADT;

typedef ______ elemtype;

typedef int (*compare)(elemtype e1, elemtype e2);

/* Crea todos los recursos para el TAD
** Arranca inicialmente sin elementos.
*/
elemCountADT newElemCount(compare cmp);

/* Registra una aparición de elem y retorna la cantidad actual de
** apariciones registradas.
*/
size_t countElem(elemCountADT elemCountAdt, elemtype elem);

/* Retorna la cantidad de elementos distintos registrados. */
size_t distinctElems(elemCountADT elemCountAdt);

/* Funciones de iteración para que se puedan consultar todos los elementos
** registrados en forma ordenada
** junto con la cantidad de apariciones de cada uno.
*/
void toBegin(elemCountADT elemCountAdt);

size_t hasNext(elemCountADT elemCountAdt);

elemtype next(elemCountADT elemCountAdt, size_t * count);

/* Libera los recursos utilizados por el TAD */
void freeElemCount(elemCountADT elemCountAdt);
```

Se pide:

- **Definir todas las estructuras necesarias**
- **Implementar las siguientes funciones:**
  - **newElemCount**
  - **countElem**
  - **freeElemCount**
  - **toBegin**
  - **hasNext**

Ejemplo de programa de testing, con elemtype un alias para int

```
int cmpInts(int a, int b) {
   return a - b;
}

int
main(void) {
    elemCountADT ecADT = newElemCount(cmpInts);
    assert(countElem(ecADT, 10) == 1);
    assert(countElem(ecADT, 5+5) == 2);

    assert(distinctElems(ecADT) == 1);

    assert(countElem(ecADT, 5) == 1);
    assert(countElem(ecADT, 10) == 3);
    assert(countElem(ecADT, 5) == 2);
    assert(countElem(ecADT, 30) == 1);

    assert(distinctElems(ecADT) == 3);

    // Se itera por todos los elementos
    size_t aux;
    toBegin(ecADT);

    assert(hasNext(ecADT) == 1);
    assert(next(ecADT, &aux) == 5 && aux == 2);
    assert(hasNext(ecADT) == 1);
    assert(next(ecADT, &aux) == 10 && aux == 3);
    assert(next(ecADT, &aux == 30 && aux == 1);

    assert(hasNext(ecADT) == 0);
    freeWordCount(ecADT);
    puts("OK!");
    return 0;
}
```