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

Recuperatorio Segundo Parcial de Programación Imperativa

28/06/2024

	Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*
Calificación	**/4**	**/4**	**/2**	

- **<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 el registro de **estudiantes y sus notas en diferentes materias**. Los estudiantes pueden tener múltiples notas en la misma materia o en diferentes materias. Para identificar a un estudiante se usa un *string* corto, que podría contener letras o símbolos.

Para ello se definió la siguiente interfaz:

```
typedef struct studentsCDT * studentsADT;

/* Crea un sistema de administración de estudiantes
 */
studentsADT newStudents();

/* Ingresa una nota para el estudiante #studentName en la materia #subject.
 * Un mismo estudiante puede ingresar múltiples notas en la misma materia o en varias
 * materias distintas.
 * Se asume que los nombres de los estudiantes y las materias son cortos.
 * No hace nada si #studentName o #subject es NULL.
 */
void addGrade(studentsADT students, const char * studentName, const char * subject, float
grade);

/* Dado un estudiante #studentName retorna un vector de estructuras, donde cada
 * struct contiene el nombre de la materia y la nota del estudiante para esa materia,
 * ordenado alfabético por materia. En caso de haber más de una nota de una
 * misma materia, las mismas se muestran de acuerdo al orden en que fueron ingresadas
 * #studentName puede estar en mayúsculas o minúsculas
 * En #dim deja la cantidad de elementos del vector de respuesta
 * Si no existe el alumno o no tiene notas cargadas retorna NULL y deja dim en cero
 */
struct grade {
   char * subject;
   float grade;
}
struct grade * grades(const studentsADT students, const char * studentName, size_t * dim);
```

```
/* Funciones de iteración para poder consultar los nombres de los estudiantes
 * en orden alfabético
 */
void toBegin(studentsADT students);
size_t hasNext(const studentsADT students);

/* Retorna una copia del nombre del estudiante */
char * next(studentsADT students);

/* Retorna el promedio de notas del estudiante #studentName en la materia #subject.
 * Si el estudiante no tiene notas en esa materia, retorna -1.
 */
float getAverageGrade(const studentsADT students, const char * studentName, const char *
subject);

/* Libera los recursos utilizados por el TAD.
 */
void freeStudents(studentsADT students);
```

Implementar todas las estructuras necesarias

Implementar las siguientes funciones:

- **newStudents**
- **addGrade**
- **grades**
- **toBegin**
- **next**

Ejemplo de programa de prueba: En el ejemplo se cargan pocas notas, pero tener en cuenta que pueden estar todas las notas de todos los exámenes de un alumno durante la carrera.

```
int main(void) {
    // Crear un sistema de administración de estudiantes
    studentsADT school = newStudents();

    // Agregar notas para varios estudiantes
    char name[20] = "Alice";
    char subject[25] = "Discreta"
    addGrade(school, name, subject, 8.5);
    addGrade(school, name, "discreta", 9.0); // "Discreta" y "discreta" son lo mismo
    addGrade(school, name, subject, 7.0);    // Alice 8.5, 9 y 7 en Discreta
    strcpy(subject, "Quimica");
    addGrade(school, name, subject, 7.5);    // Alice 7.5 en Quimica
    strcpy(subject, "Fisica");
    addGrade(school, name, subject, 3.5);    // Alice 3.5 en Fisica
    strcpy(name, "Bob");
    addGrade(school, name, "Discreta", 6.0); // Bob 6 en Discreta
    addGrade(school, name, subject, 8.0);    // Bob 8 en Fisica

    // Obtener el promedio de notas de Alice en Discreta
    assert(getAverageGrade(school, "Alice", "Discreta") == 8.75);

    // Obtener el promedio de notas de Bob en Discreta
    assert(getAverageGrade(school, "Bob", "Discreta") == 6.0);

    // Obtener el promedio de notas de Alice en Filosofia (sin notas)
    assert(getAverageGrade(school, "Alice", "Filosofia") == -1);

    // Obtener las notas de Alice
    size_t dim;
    struct grade * aliceG = grades(school, "alice", &dim); // alice y Alice es lo mismo
```

```
    assert(dim == 5);
    assert(strcmp(aliceG[0].subject, "Discreta") == 0 && aliceG[0].grade == 8.5);
    assert(strcmp(aliceG[1].subject, "discreta") == 0 && aliceG[1].grade == 9.0);
    assert(strcmp(aliceG[2].subject, "Discreta") == 0 && aliceG[2].grade == 7.0);
    assert(strcmp(aliceG[3].subject, "Fisica") == 0 && aliceG[3].grade == 3.5);
    assert(strcmp(aliceG[4].subject, "Quimica") == 0 && aliceG[4].grade == 7.5);
    free(aliceG[0].subject);
    free(aliceG[1].subject);
    free(aliceG[2].subject);
    free(aliceG[3].subject);
    free(aliceG[4].subject);
    free(aliceG);

    // Agregamos a Anabella y al estudiante "123"
    // (está permitido identificar a un alumno con caracteres que no sean letras)

    addGrade(school, "anabella", "HCI", 10.0);
    addGrade(school, "123", "HCI", 9.5);

    // Iterar sobre los nombres de los estudiantes en orden alfabético
    toBegin(school);
    assert(hasNext(school) == 1);
    char * studentName = next(school);
    assert(strcmp(studentName, "123") == 0);
    free(studentName);

    studentName = next(school);
    assert(strcmp(studentName, "Alice") == 0);
    free(studentName);

    studentName = next(school);
    assert(strcmp(studentName, "anabella") == 0);
    free(studentName);

    studentName = next(school);
    assert(strcmp(studentName, "Bob") == 0);
    free(studentName);

    assert(hasNext(school) == 0);

    // Liberar los recursos utilizados por el TAD
    freeStudents(school);

    printf("Todos los tests pasaron correctamente.\n");
    return 0;
}
```

Ejercicio 2

Se desea implementar un TAD para administrar las **reservas de salas de un centro de conferencias**. La numeración de las salas es entera positiva, iniciando en 1.

Para ello se definió la siguiente interfaz:

```
typedef struct roomsCDT * roomsADT;

/* Crea un sistema de administración de salas a partir de la cantidad máxima
 * #maxRooms de salas que tendrá el centro de conferencias.
 * Se asume que la gran mayoría de las salas serán utilizadas por los asistentes.
 * Si maxRooms es cero retorna NULL.
 */
roomsADT newRooms(size_t maxRooms);
```

```
/* Reserva al asistente #attendee en la espera de la sala #roomNumber
 * para luego poder asistir a la conferencia.
 * Un mismo asistente puede reservar múltiples veces la misma sala o varias
 * salas distintas.
 * Se asume que los nombres de los asistentes son cortos.
 * No hace nada si #roomNumber es mayor o igual a la cantidad máxima de salas.
 */
void reserveRoom(roomsADT rooms, size_t roomNumber, const char * attendee);

/* Funciones de iteración para poder consultar para una sala los asistentes que están
 * esperando, en ORDEN DE RESERVA (ver ejemplo en programa de prueba)
 */
void toBeginByRoom(roomsADT rooms, size_t roomNumber);
size_t hasNextByRoom(const roomsADT rooms, size_t roomNumber);
const char * nextByRoom(roomsADT rooms, size_t roomNumber);

/* Para cada sala, realiza la admisión del asistente que está esperando primero,
 * si es que hay al menos un asistente esperando en esa sala.
 * Retorna un arreglo con los nombres de los asistentes que hicieron la admisión
 * y salieron de la espera
 * respetando el ORDEN ASCENDENTE por número de sala
 * Deja en un parámetro de entrada/salida la dimensión del vector de retorno
 * Si no hay asistentes retorna NULL
 */
char ** admitRooms(roomsADT rooms, size_t * admissionResultDim);

/* Para cada sala, realiza la admisión de #n asistentes que están esperando primero,
 * si es que hay al menos un asistente esperando en esa sala. Si en una sala hay menos
 * de #n que están esperando, entran todos
 * Retorna un arreglo con los nombres de los asistentes que hicieron la admisión
 * y salieron de la espera
 * respetando el ORDEN ASCENDENTE por número de sala
 * Deja en un parámetro de entrada/salida la dimensión del vector de retorno
 * Si no hay asistentes o n es cero retorna NULL
 */
char ** admitRoomsN(roomsADT rooms, size_t * admissionResultDim, size_t n);

/* Libera los recursos utilizados por el TAD
 */
void freeRooms(roomsADT rooms);
```

Implementar todas las estructuras necesarias, de forma tal que **todas las funciones** puedan ser implementadas de la forma **más eficiente posible**.

Implementar las siguientes funciones:

- **newRooms**
- **reserveRoom**
- **toBeginByRoom**
- **admitRooms**
- **admitRoomsN**

Ejemplo de programa de prueba:

```
int main(void) {
    roomsADT conferenceCenter = newRooms(5);
    reserveRoom(conferenceCenter, 3, "Alice"); // Alice está en espera de la sala 3
    reserveRoom(conferenceCenter, 3, "Bob");
    reserveRoom(conferenceCenter, 3, "Bob");
    reserveRoom(conferenceCenter, 2, "Charlie");
    reserveRoom(conferenceCenter, 2, "David");
```

```
    // Se desea consultar los asistentes que están esperando en la sala 3
    toBeginByRoom(conferenceCenter, 3);
    // "Alice" es el primero que reservó la sala 3
    assert(hasNextByRoom(conferenceCenter, 3) == 1);
    assert(strcmp(nextByRoom(conferenceCenter, 3), "Alice") == 0);
    assert(hasNextByRoom(conferenceCenter, 3) == 1);
    assert(strcmp(nextByRoom(conferenceCenter, 3), "Bob") == 0);
    toBeginByRoom(conferenceCenter, 2);
    assert(hasNextByRoom(conferenceCenter, 2) == 1);
    assert(strcmp(nextByRoom(conferenceCenter, 2), "Charlie") == 0);

    // Se pueden usar en simultáneo las funciones de iteración para consultar
    // los asistentes que están esperando en distintas salas
    assert(hasNextByRoom(conferenceCenter, 3) == 1);
    assert(strcmp(nextByRoom(conferenceCenter, 3), "Bob") == 0);
    assert(hasNextByRoom(conferenceCenter, 2) == 1);
    assert(strcmp(nextByRoom(conferenceCenter, 2), "David") == 0);
    assert(hasNextByRoom(conferenceCenter, 3) == 0);
    assert(hasNextByRoom(conferenceCenter, 2) == 0);

    // Se realiza, para cada sala, la admisión del asistente que está esperando primero
    size_t admissionResultDim;
    char ** admissionResultVec;
    admissionResultVec = admitRooms(conferenceCenter, &admissionResultDim);
    assert(admissionResultDim == 2); // Se realizó la admisión en dos salas: 2 y 3
    // Resultado de la admisión en la sala 2
    assert(strcmp(admissionResultVec[0], "Charlie") == 0);
    // Resultado de la admisión en la sala 3
    assert(strcmp(admissionResultVec[1], "Alice") == 0);
    free(admissionResultVec);

    toBeginByRoom(conferenceCenter, 3);
    toBeginByRoom(conferenceCenter, 2);
    assert(strcmp(nextByRoom(conferenceCenter, 3), "Bob") == 0);
    assert(strcmp(nextByRoom(conferenceCenter, 2), "David") == 0);

    // Admitir hasta 2 personas por sala
    admissionResultVec = admitRoomsN(conferenceCenter, &admissionResultDim, 2);
    assert(admissionResultDim == 3); // Se hizo admisión de 3 personas
    // Resultado de la admisión en la sala 2
    assert(strcmp(admissionResultVec[0], "David") == 0);
    // Resultado de la admisión en la sala 3 (dos personas llamadas Bob)
    assert(strcmp(admissionResultVec[1], "Bob") == 0);
    assert(strcmp(admissionResultVec[2], "Bob") == 0);
    free(admissionResultVec);

    toBeginByRoom(conferenceCenter, 3);
    toBeginByRoom(conferenceCenter, 2);
    assert(hasNextByRoom(conferenceCenter, 3) == 0);
    assert(hasNextByRoom(conferenceCenter, 2) == 0);

    admissionResultVec = admitRooms(conferenceCenter, &admissionResultDim);
    assert(admissionResultDim == 0);
    assert(admissionResultVec == NULL);

    freeRooms(conferenceCenter);
    return 0;
}
```

Ejercicio 3

Escribir una función **recursiva** **`balance`** que reciba como **único parámetro** una lista que representa los **movimientos en la cuenta corriente de un cliente** (importes) y **retorne una nueva lista** donde tenga en cada nodo **el cliente y el balance de ese cliente** (la suma de todos sus importes).

La lista recibida está ordenada por el identificador entero del cliente (se asegura, no es necesario validarlo).

Los tipos de datos a usar son los siguientes:

```
typedef struct docNode {
    int id;              // identifica al cliente
    double amount;       // importe del comprobante (positivo o negativo)
    struct docNode * tail;
} docNode;

typedef docNode * docList;

typedef struct balanceNode {
    int id;             // identifica al cliente
    double balance;     // cuánto debe o se le debe al cliente
    struct balanceNode * tail;
} balanceNode;

typedef balanceNode * balanceList;
```

Ejemplo: si la función recibe una lista de tipo **`docList`** con los siguientes pares de **`id`** e **`importe`**

`{1, 100.0} → {1, -90.0} → {1, -10.0} → {3, 95.0} → {3, -10.0} → {5, -10.0}`

retorna una lista de tipo **`balanceList`** con los siguientes nodos

`{1, 0.0} → {3, 85.0} → {5, -10.0}`