Apellido, Nombre:
Padrón:

Fundamentos de Programación

Exámen final - 13/02/2025

Condición de aprobación: Tener 2 ejercicios **Bien**.

Aclaraciones:

- Un ejercicio práctico se considera **Bien** cuando además de cumplir con lo pedido, aplica las buenas prácticas profesadas por la cátedra.
- En un ejercicio la parte práctica equivale al 75% de la calificación de ese punto mientras que la parte teórica equivale al 25%.

Ejercicio 1

Parte práctica

Mickey y Donald juegan con cartas Pokemouse, y les gustaría poder intercambiarlas cuando tienen repetidas. Para esto, necesitan saber qué cartas tiene uno y no el otro.
Las cartas Pokemouse que tiene cada uno están representados por un archivo csv con el siguiente formato:

```
NOMBRE_POKEMOUSE;ID_POKEMOUSE;EVOLUCION;ATAQUE;
```

Saben que es la misma carta cuando tienen el mismo nombre **y** mismo id.

Se pide

Dado dos archivos de cartas, uno de Mickey (**cartas_mickey.csv**) y otro de Donald (**cartas_donald.csv**), crear otro archivo **cartas_diferentes.csv** que tenga las cartas que tiene Mickey pero que no tiene Donald para que las puedan intercambiar.

Aclaración

- El archivo está ordenado por nombre y después por id.

Parte teórica

Explicar las diferencias entre el for y while. Cuando conviene usar uno y cuando otro.

Ejercicio 2

Parte práctica

Aladdin está buscando la lámpara del genio, que se escondió porque se pelearon. Para encontrarlo, se tiene un mapa representado por una matriz de movimientos:

```
typedef struct indicacion {
    char direccion[MAX_DIRECCION];
    int cantidad_pasos;
} indicacion_t;
```

La dirección de cada indicación puede ser:

- **"Arriba"**: Moverse hacia arriba
- **"Izquierda"**: Moverse a la izquierda
- **"Abajo"**: Moverse hacia abajo
- **"Derecha"**: Moverse a la derecha

Además, en la dirección puede haber estos dos elementos:

- **"Trampa"**: Trampa
- **"Lámpara"**: Lámpara

Para recorrer el mapa, Aladdin se mueve la cantidad de pasos correspondiente en la dirección que se indica en cada posición de la matriz. Aladdin no puede pasarse de los límites de la matriz, así que él siempre frena en el borde aunque la indicación diga que tiene que seguir.

Se pide

Dada una matriz que representa el mapa, crear una función que la recorra siguiendo las indicaciones. Devolver true si Aladdin puede encontrar la lámpara del genio y false si cae en una trampa.

Aclaraciones

- Empieza recorriendo la matriz desde el (0,0).
- Se puede asumir que siempre o encontrará la lámpara o una trampa.
- El campo de `cantidad_pasos` se puede ignorar si la direccion es "Lámpara" o "Trampa".

Parte teórica

Al ejecutar el siguiente programa tiene un error en el manejo de memoria dinámica, explicar donde ocurre y porque es un error:

```c
// El struct guarda los tiempos de un corredor en cada una de sus carreras
typedef struct corredor {
    int* tiempos;
    int cantidad_carreras;
} alumno_t;

int main () {
    srand((unsigned)time(NULL));

    int cantidad_corredores = 100;
    corredor_t* corredores = malloc(sizeof(corredor_t) * cantidad_corredores);

    for(int i = 0; i < cantidad_corredores; i++) {
        int cantidad_carreras = rand() % 10 + 1; // una cantidad entre 1 y 10
        corredores[i].cantidad_carreras = cantidad_carreras;
        *(corredores + i).tiempos = malloc(sizeof(int) * cantidad_carreras);
    }

    // Procesamiento...

    free(corredores);

    for(int i = 0; i < cantidad_corredores; i++){
        free(corredores[i].tiempos);
    }

    return 0;
}
```

Ejercicio 3 (para 2C2024)

Parte práctica

En la mente de Riley se organizan los recuerdos a medida que van llegando, y se los va poniendo en un tubo para que lleguen a la memoria a largo plazo.
Un recuerdo se representa con el siguiente struct:

```c
typedef struct recuerdo {
    int id;
    char tipo[MAX_NOMBRE]; //"trauma", "feliz", "triste", "indiferente"
} recuerdo_t;
```

El tubo se puede pensar como que tiene un principio y un final. Los recuerdos entran por el principio de a uno y salen por el final en el mismo orden en el que entraron, no se puede sacar un recuerdo por el principio del tubo.

Para representar esto, se tiene un TDA "tubo" que guarda los recuerdos. El TDA cuenta con las siguientes primitivas:

```
// Pre: -
// Post: Crea un TDA tubo en el heap y devuelve un puntero al
// mismo. Devuelve NULL si no lo pudo crear.
tubo_t *tubo_crear();

// Pre: El tubo recibido fue previamente creado con `tubo_crear`.
// Post: Libera la memoria que se reservó en el heap para el tubo.
void tubo_destruir(tubo_t *tubo);

// Pre: El tubo recibido fue previamente creado con `tubo_crear`.
// Post: Agrega un reporte al principio del tubo.
bool tubo_agregar_al_principio(tubo_t *vector, recuerdo_t nuevo_elemento);

// Pre: El tubo recibido fue previamente creado con `tubo_crear`.
// Post: Se eliminará el primer recuerdo en el final del tubo.
// Devuelve true si se eliminó correctamente o false si el tubo
// estaba vacío y no se pudo eliminar nada. En caso de devolver true,
// el elemento que se acaba de eliminar será devuelto por referencia
// mediante el parámetro `elemento_eliminado`.
bool tubo_eliminar_al_final(tubo_t *tubo, recuerdo_t *elemento_eliminado);
```

Hubo un problema, y se infiltró un trauma en el tubo de los recuerdos. Alegría quiere sacarlo sí o sí.

Se pide

Hacer una función que reciba un `tubo_t` creado que encuentre el trauma y lo elimine. Al finalizar, el tubo tiene que **quedar con los mismos recuerdos que con los que arrancó pero sin el trauma**.

Aclaraciones

- Solo hay un trauma en el tubo.

Parte teórica

Ejemplificar, explicando paso por paso, como se realizaría una búsqueda binaria en el siguiente vector:

```
[3, 6, 10, 11, 20, 24, 42]
```

Suponga que se quiere buscar la posición del número 9.