Ejercicios tema 1. Introducción al lenguaje imperativo de bajo nivel y estructuras básicas.

1. Implementar un subalgoritmo llamado **`contiene`** que reciba como parámetros una lista enlazada de números enteros y un número entero. El subalgoritmo deberá devolver un valor verdadero si la lista contiene un elemento con el mismo valor que el número recibido como parámetro y falso en caso contrario. Por ejemplo, para la lista [1, 5, 6] y el número 5 el subalgoritmo devolvería verdadero. Y para la misma lista y el número 7 el subalgoritmo devolvería falso.

2. Implementar un subalgoritmo llamado **`buscarPrimero`** que recibe como parámetros una lista enlazada de números enteros y un número entero. El subalgoritmo deberá devolver la posición de la primera aparición en la lista del número recibido como parámetro. Si el número no estuviese en la lista entonces devolvería el tamaño de la lista. Por ejemplo, para la lista [1, 5, 1] y el número 1 el subalgortmo devolvería el valor 0 (primera posición donde aparece el número 1). Y para la misma lista y el número 6 el subalgoritmo devolvería el valor 3 (número de elementos en la lista).

3. Implementar un subalgoritmo llamado **`eliminarPrimero`** que recibe como parámetros una lista enlazada de números enteros y un número entero. El subalgoritmo deberá eliminar de la lista la primera aparición del número recibido como parámetro. Si el número no estuviese en la lista entonces el subalgoritmo no haría nada. Por ejemplo, para la lista [1, 5, 1] y el número 1 el subalgortmo dejaría la lista de la forma [5, 1]. Y para la misma lista y el número 6 el subalgoritmo no modificaría la lista.

4. Implementar un subalgoritmo llamado **`eliminarUltimo`** que recibe como parámetros una lista enlazada de números enteros y un número entero. El subalgoritmo deberá eliminar de la lista la última aparición del número recibido como parámetro. Si el número no estuviese en la lista entonces el subalgoritmo no haría nada. Por ejemplo, para la lista [1, 5, 1, 6] y el número 1 el subalgortmo dejaría la lista de la forma [1, 5, 6]. Y para la misma lista y el número 7 el subalgoritmo no modificaría la lista.

5. Implementar un subalgoritmo llamado **`eliminarValor`** que recibe como parámetros una lista enlazada de números enteros y un número entero. El subalgoritmo deberá eliminar de la lista todas las aparciciones del número recibido como parámetro. Si el número no estuviese en la lista entonces el subalgoritmo no haría nada. Por ejemplo, para la lista [1, 5, 1, 6] y el número 1 el subalgortmo dejaría la lista de la forma [5, 6]. Y para la misma lista y el número 7 el subalgoritmo no modificaría la lista.

6. Implementar un subalgoritmo llamado `invertir` que recibe como parámetro una lista enlazada de números enteros y devuelve una lista con los mismos valores pero en orden inverso. Por ejemplo, para la lista [1, 2, 3] el subalgoritmo devolvería la lista [3, 2, 1].

7. Implementar un subalgoritmo llamado **`incluir`** que recibe como parámetro una lista enlazada de números enteros y un número entero. Si el número recibido no esta en la lista entonces el subalgoritmo lo añadirá al final de la misma. Si el número ya esta en la lista entonces el subalgoritmo no modificará la lista. Por ejemplo, para la lista [1, 3, 5] y el número 2 el subalgoritmo dejará la lista de la forma [1, 3, 5, 2]. Para la misma lista y el número 3 el subalgoritmo no modificará la lista.

8. Implementar un subalgoritmo llamado **repeticiones** que recibe como parámetros una lista enlazada de números enteros y un número entero. El subalgoritmo devolverá el número de veces que aparece en la lista el valor recibido como parámetro. Por ejemplo, para la lista [1, 4, 3, 6, 3, 7] y el número 3 el subalgoitmo devolverá el valor 2. Para la misma lista y el valor 8 el subalgoritmo devolverá 0.

9. Implementar un subalgoritmo llamado **eliminarRepeticiones** que recibe como parámetro una lista enlazada de números enteros. El subalgoritmo eliminará todas las repeticiones de los números de la lista dejando solamente la primera aparición de cada número. Por ejemplo, para la lista [2, 1, 4, 1, 5, 4, 4, 1, 6] el subalgoritmo dejará la lista en la forma [2, 1, 4, 5, 6]. Los valores que queden en la lista deberá mantener el mismo orden que en la lista original.

10. Implementar un subalgoritmo llamado **eliminarRepetidos** que recibe como parámetro una lista enlazada de números enteros. El subalgoritmo eliminará de la lista cualquier valor que se encuentre repetido. Por ejemplo, para la lista [2, 1, 4, 1, 5, 4, 4, 1, 6] el subalgoritmo dejará la lista en la forma [2, 5, 6]. Los valores que queden en la lista deberá mantener el mismo orden que en la lista original.

11. Implementar un subalgoritmo llamado **insertarOrdenado** que recibe como parámetros una lista enlazada de números reales (`double`) ordenada de menor a mayor y un número real (`double`). El subalgoritmo insertará el valor recibido en la lista de forma que la lista quede ordenada de menos a mayor después de la inserción. Por ejemplo, dada la lista [3.0, 4.5, 8.0, 20.3] y el valor 6.0 el subalgoritmo dejará la lista de la forma [3.0, 4.5, 6.0, 8.0, 20.3].

12. La siguiente estructura de datos representa una lista de números almacenada en un bloque de memoria dinámica usado parcialmente (no todo el bloque contiene datos válidos). En esta estructura `capacidad` almacena el tamaño del bloque de memoria dinámica y `nElementos` el número de elementos almacenados en el bloque:

```
typedef struct{
     unsigned capacidad, nElementos;
     int* buffer;
} Lista;
```

Implementar los siguientes algoritmos sobre esta estructura:
a) Subalgoritmo **vacia** que recibe como parámetro una estructura de tipo `Lista` y la inicializa a un bloque de tamaño 10 (`capacidad`) y 0 elementos almacenados (`nElementos`).
b) Subalgoritmo **almacenar** que recibe como paŕametros una estructura de tipo `Lista` y un número entero. Este subalgoritmo almacenará el número recibido detrás de la parte usada del buffer e incrementará el número de elementos almacenados (`nElementos`). Si no hubiese suficientes elementos entonces el subalgoritmo deberá duplicar la capacidad del buffer reservando más memoria.
c) Subalgoritmo **obtener** que recibe como parámetros una estructura de tipo `Lista` y un valor booleano (`ok`). El algoritmo devolverá y eliminará el primer elemento almacenado en el buffer y decrementará el número de elementos almacenados (`nElementos`). Si hay al menos un elemento en el buffer que devolver entonces el subalgoritmo devolverá verdadero en el parámetro `ok`. Si no hay elementos entonces el subalgoritmo devolverá falso en el parámetro `ok` y el valor 0. En caso de que el uso del buffer (`nElementos`) sea menor que la mitad de la capacidad y la capacidad sea mayor que 10 entonces el subalgoritmo deberá liberar la mitad del bloque de memoria.
d) Subalgoritmo **liberar** que recibe como parámetro una estructura de tipo Lista y libera la memoria reservada en el buffer.