Ejercicios: Hoja 3.2

9. Escribir un procedimiento que permita leer de teclado los diferentes valores de un vector de números

   void leerVector(int m[], int lon)

10. Escribir un procedimiento que permita escribir un vector en pantalla

    void escribirVector(int m[], int lon)

11. Realizar un procedimiento que permita realizar el producto por un escalar de un vector **void productoEscalar(int v[], int numero, int longitud)**

12. Implementar la función **int member(int elem, int [] arr, int lon)** que devuelve 1 si el elemento 'elem' de tipo 'int' está en el array 'arr'

13. Implementar la función **int paresAst(char v[], int lon)** que devuelve 1 si en todas las posiciones pares del array 'v' hay un carácter asterisco.

14. Implementar la función **int numVoc(char v[], int lon)** que devuelve el número de caracteres en 'v' que son vocales.

15. Implementar la función **int tresAst(char v[], int lon)** que devuelve 1 si el array de caracteres 'v' contiene al menos tres asteriscos. Implementar también la alternativa que indica si hay únicamente tres asteriscos.

16. Implementar la función **int numOcChar(char c, char v[], int lon)** que devuelve el número de caracteres en 'v' que son iguales al caracter 'c' (es decir, devuelve el número de apariciones de 'c' en 'v')

17. Implementar la función **int tresAstConsec(char v[], int lon)** que devuelve 1 si el array de caracteres 'v' contiene al menos tres asteriscos consecutivos.

18. Implementar la función **int nAstConsec(int n, char v[], int lon)** q devuelve 1 si el array de caracteres 'v' contiene al menos 'n' asteriscos consecutivos.

19. Implementar la función **int tresGrupAstConsec(char v[], int lon)** que indica cuántos grupos de tres asteriscos consecutivos hay en el array de caracteres 'v'.

20. Implementar la función **int nGrupAstConsec(int n, char [] v, int lon)** que indica cuántos grupos de ‘n’ asteriscos consecutivos hay en el array de caracteres 'v'

21. Implementar la función **int suma(int v[], int lon)** que devuelve el resultado de sumar todos los elementos del array, o -1 si v no es válido

22. Implementar la función **int mult(int v[], int lon)** que devuelve el resultado de sumar todos los elementos del array, o -1 si v no es válido

23. Implementar la función **int memberOrd(int elem, int arr[], int lon)** que devuelve 1 si el elemento 'elem' está en el array 'arr' el cuál está ordenado de menor a mayor

24. Implementar la función **int iguales(int v1[], int lon1, int v2[], int lon2)** que indica si los dos arrays almacenan los mismos elementos de izquierda a derecha.

25. Implementar el procedimiento **void reverse(int v[], int res[], int lon)** que devuelve en el array '*res*' los elementos de 'v' en orden inverso.

26. Implementar el procedimiento **void reverseSameVector(int v[], int lon)** que modifica 'v' dejando sus elementos en orden inverso.

27. Implementar el procedimiento **void removeCompact(int i, char v[], int lon)** que elimina del array 'v' el elemento en la posición 'i', desplazando el resto de elementos una posición a la izquierda para no dejar un hueco.

28. Escribir un procedimiento que permita leer de teclado los diferentes valores de una matriz de números

    void leerMatriz(int filas, int cols, int m[][cols])

29. Escribir un procedimiento que permita escribir matrices en pantalla

    void escribirMatriz(int filas, int cols, int m[][cols])

30. Escribir un procedimiento que permita sumar dos matrices (suponer que se pueden sumar)

    void sumarMatrices(int filas, int cols, int m1[][cols], int m2[][cols], int res[][cols])

31. Escribir un procedimiento que permita multiplicar dos matrices (suponer que se pueden multiplicar)

    void multiplicarMatrices(int filas1, int cols1, int m1[][cols1], int filas2, int cols2, int m2[][cols2], int filasR, int colsR, int[][colsR])

32. Escribir una función que devuelva 1 si una matriz es simétrica

    int esSimetrica(int filas, int cols, int m[][cols])