ARRAYS

EJERCICIO 1

Diseña un programa que lea un número indeterminado de caracteres, indicando el fin de la lectura con *, y muestre el número total de apariciones de cada letra.

Finalmente, indicará cuántos caracteres se han leído, y de ellos, cuántas letras y separadores.

EJERCICIO 2

Escribe un programa que reciba un número indeterminado de caracteres terminado por el carácter # Los caracteres leídos deben ser 0 ó 1 (dígitos binarios) ó cualquier separador. El programa deberá construir caracteres a partir de los dígitos binarios leídos y seleccionar algunos de ellos, copiándolos a un array de char.

Cada bloque de 8 dígitos binarios leídos sirve para reconstruir un carácter (un dato char). Los dígitos binarios se leen desde el más significativo al menos significativo.

Con el objetivo de minimizar el tiempo de cálculo, para formar un carácter se empleará un array de datos int que contiene el peso de cada dígito binario en un byte: 256, 128, 64, . . . , 1.

No todos los caracteres reconstruidos serán de nuestro interés. El programa seleccionará únicamente los caracteres alfabéticos, numéricos, la coma, el punto y el espacio en blanco. Los caracteres seleccionados se copiarán en una array (mensaje) que tendrá un tamaño máximo predeterminado de 200.

Finalmente mostrará el mensaje reconstruido.

EJERCICIO 3

Escribir un programa que lea un número indeterminado de números positivos (termina la lectura cuando se introduce un negativo) aunque nunca leerá más de 50. Conforme los va leyendo, los va almacenando en un vector, datos.

A continuación elimina del vector los valores repetidos, dejando una sola copia. No se dejan huecos en el vector y todos los números quedan agrupados en las posiciones más bajas del vector.

Implementar tres versiones del borrado:

a) Usar un vector auxiliar ***sin_repetidos*** en el que almacenamos una única aparición de cada componente
b) El problema del algoritmo anterior es que usa otro vector, lo que podría suponer una carga importante de memoria si trabajásemos con vectores grandes. Por lo tanto, vamos a resolver el problema sin usar vectores auxiliares. Si una componente está repetida, se borrará del vector copiando desde *p* hasta la penúltima posición el contenido de *p+1* en *p*.
c) El anterior algoritmo nos obliga a desplazar muchos componentes. Implementar el algoritmo que usa dos apuntadores, ***posicion_lectura*** y ***posicion_escritura***, que nos van indicando, en cada momento, la componente que se está leyendo y el sitio dónde tiene que escribirse. Por ejemplo, supongamos que en un determinado momento la variable ***posicion_lectura*** vale 6 y ***posicion_escritura*** 3. Si la componente en la posición 6 está repetida, simplemente avanzaremos ***posicion_lectura***. Por el contrario, si no estuviese repetida, la colocaremos en la posición 3 y avanzaremos una posición ambas variables.