**ARREGLOS**

Considerar hacer funciones en todos los ejercicios. Muchas situaciones se repiten y la buena utilización de funciones permitirá ahorrar tiempo.

**Arreglos unidimensionales numéricos**

# 1. Ejercicio

Realizar un programa que permita guardar hasta 1000 números enteros positivos en un arreglo. Dichos números se ingresarán por teclado hasta llenar el arreglo o bien hasta que el usuario ingrese cero (este valor siempre marca el final de los valores útiles del arreglo). A continuación, imprimir todos los valores útiles del arreglo separados por un espacio.

# 2. Ejercicio

Existe una serie de transformaciones frecuentes que pueden hacerse sobre un arreglo. Las mismas se listan y ejemplifican en la siguiente tabla:

| Transformación | Arreglo antes | Arreglo después |
| --- | --- | --- |
| ordenar | 5, 8, 3, 4, 2, 3, 0 | 2, 3, 3, 4, 5, 8, 0 |
| invertir | 5, 8, 3, 4, 2, 3, 0 | 3, 2, 4, 3, 8, 5, 0 |
| mezclar | 5, 8, 3, 4, 2, 3, 0 | 3, 4, 3, 8, 5, 2, 0 |

Por cada transformación, hacer una función que reciba como parámetro un arreglo (ya cargado) de números enteros positivos (el cero marca el final de los valores útiles). El tamaño del arreglo se desconoce pero todas las funciones deben poder tratar con arreglos de cualquier tamaño y de cualquier cantidad de valores útiles. Implementarlas en un programa de prueba.

# 3. Ejercicio

Existe una serie de operaciones básicas que pueden hacerse sobre 2 conjuntos representados en arreglos. Las mismas se listan y ejemplifican en la siguiente tabla:

| Transformación | Conjunto 1 | Conjunto 2 | Conjunto resultante |
| --- | --- | --- | --- |
| union | 5, 8, 3, 4, 2, 9, 0 | 1, 5, 8, 6, 0 | 5, 8, 3, 4, 2, 9, 1, 6, 0 |
| interseccion | 5, 8, 3, 4, 2, 9, 0 | 1, 5, 8, 6, 0 | 5, 8, 0 |
| diferencia (1-2) | 5, 8, 3, 4, 2, 9, 0 | 1, 5, 8, 6, 0 | 3, 4, 2, 9, 0 |

Por cada operación, hacer una función que reciba 3 arreglos de números enteros positivos como parámetros. Los dos primeros arreglos (ambos ya cargados) representan los operandos y el tercero (no inicializado) se utilizará para guardar el resultado. El tamaño de los arreglos se desconoce pero todas las funciones deben poder tratar con arreglos de cualquier tamaño y de cualquier cantidad de valores útiles. El cero marca el final de los valores útiles. Los arreglos operandos no deben modificarse. Implementarlas en un programa de prueba.

***4. Ejercicio***

Realizar las funciones cuyos prototipos y objetivos se detallan a continuación y luego implementarlas en un programa de ejemplo:

* **void** inicializar (**int** a[], **unisgned int** t, **int** n); Inicializa el arreglo a de tamaño t con el valor n.
* **void** inicializar\_azar (**int** a[], **unisgned int** t, **int** min, **int** max); Inicializa el arreglo a de tamaño t con valores al azar entre min y max (ambos inclusive).

# 5. Ejercicio

Existe una serie de situaciones frecuentes que involucran a un arreglo y un determinado valor. Las mismas se listan y ejemplifican en la siguiente tabla:

| Transformación | Arreglo | Valor | Resultado | Consideraciones |
| --- | --- | --- | --- | --- |
| agregar\_al\_final | 2, 5, 6, 1, 0 | 8 | 2, 5, 6, 1, 8, 0 | \*1 |
| agregar\_al\_inicio | 2, 5, 6, 1, 0 | 7 | 7, 2, 5, 6, 1, 0 | \*1 |
| agregar\_ordenado | 2, 4, 5, 7, 0 | 5 | 2, 4, 5, 5, 7, 0 | \*1 |
| insertar\_sin\_repetir | 2, 4, 5, 7, 0 | 3 | 2, 4, 5, 7, 3, 0 | \*1,2 |
| eliminar\_valor | 2, 4, 5, 7, 0 | 5 | 2, 4, 7, 0 | \*1,3 |
| posicion\_valor | 2, 4, 5, 7, 0 | 7 | Se retorna 3 | \*1,4,5 |

Por cada transformación, hacer una función que reciba como parámetros un arreglo (ya cargado) de números enteros positivos (el cero marca el final de los valores útiles) y un valor entero positivo. El tamaño del arreglo se desconoce pero todas las funciones deben poder tratar con arreglos de cualquier tamaño y de cualquier cantidad de valores útiles. Implementarlas en un programa de prueba.

Consideraciones:

(\*1) El arreglo recibido puede estar vacío (hay un 0 en la posición 0).

(\*2) Si el valor a ingresar ya está en el arreglo, el arreglo recibido no se modifica.

(\*3) Si el valor no se encuentra, ningún valor es eliminado quedando intacto el arreglo recibido.

(\*4) Si el valor no se encuentra, debe retornar -1 (menos uno).

(\*5) El arreglo recibido como parámetro, no puede modificarse.

# 6. Ejercicio

Realizar las funciones cuyos prototipos y objetivos se detallan a continuación y luego implementarlas en un programa de ejemplo:

* **int** maximo (**int** a[]); Retorna el valor máximo de un arreglo en el cual el cero marca la finalización de valores útiles. Si el arreglo está vacío, debe retornar cero.
* **int** minimo (**int** a[]); Retorna el valor mínimo de un arreglo en el cual el cero marca la finalización de valores útiles. Si el arreglo está vacío, debe retornar cero.
* **int** posicion\_maximo (**int** a[]); Retorna la posición del arreglo del valor máximo de un arreglo en el cual el cero marca la finalización de valores útiles. Si el arreglo está vacío, debe retornar -1 (menos uno).
* **int** posicion\_minimo (**int** a[]); Retorna la posición del arreglo del valor mínimo de un arreglo en el cual el cero marca la finalización de valores útiles. Si el arreglo está vacío, debe retornar -1 (menos uno).

***7. Ejercicio***

Las operaciones booleanas básicas pueden aplicarse a conjuntos de bits. Para arreglos booleanos de 8 elementos (cada uno contiene únicamente el valor 0 ó 1), programar las funciones de cada operación booleana como se describe a continuación:

* Función bit\_and: recibe 3 arreglos booleanos. Los 2 primeros son los operandos y el tercero es el resultado. Cada valor del primer operando se opera con el valor de la misma posición del segundo operando y se coloca el resultado en la misma posición del arreglo resultado. La operación AND da 1 (uno) como resultado cuando ambos valores operados son 1 (uno). En el resto de los casos el resultado es 0 (cero).
* Función bit\_or: recibe 3 arreglos booleanos. Los 2 primeros son los operandos y el tercero es el resultado. Cada valor del primer operando se opera con el valor de la misma posición del segundo operando y se coloca el resultado en la misma posición del arreglo resultado. La operación OR da 0 (cero) como resultado cuando ambos valores operados son 0 (cero). En el resto de los casos el resultado es 1 (uno).
* Función bit\_not: recibe 2 arreglos booleanos. El primero es el operando y el segundo es el resultado. Cada valor del operando se cambia por su contrario en el resultado (el uno pasa a ser cero y el cero pasa a ser uno).

Ejemplos:

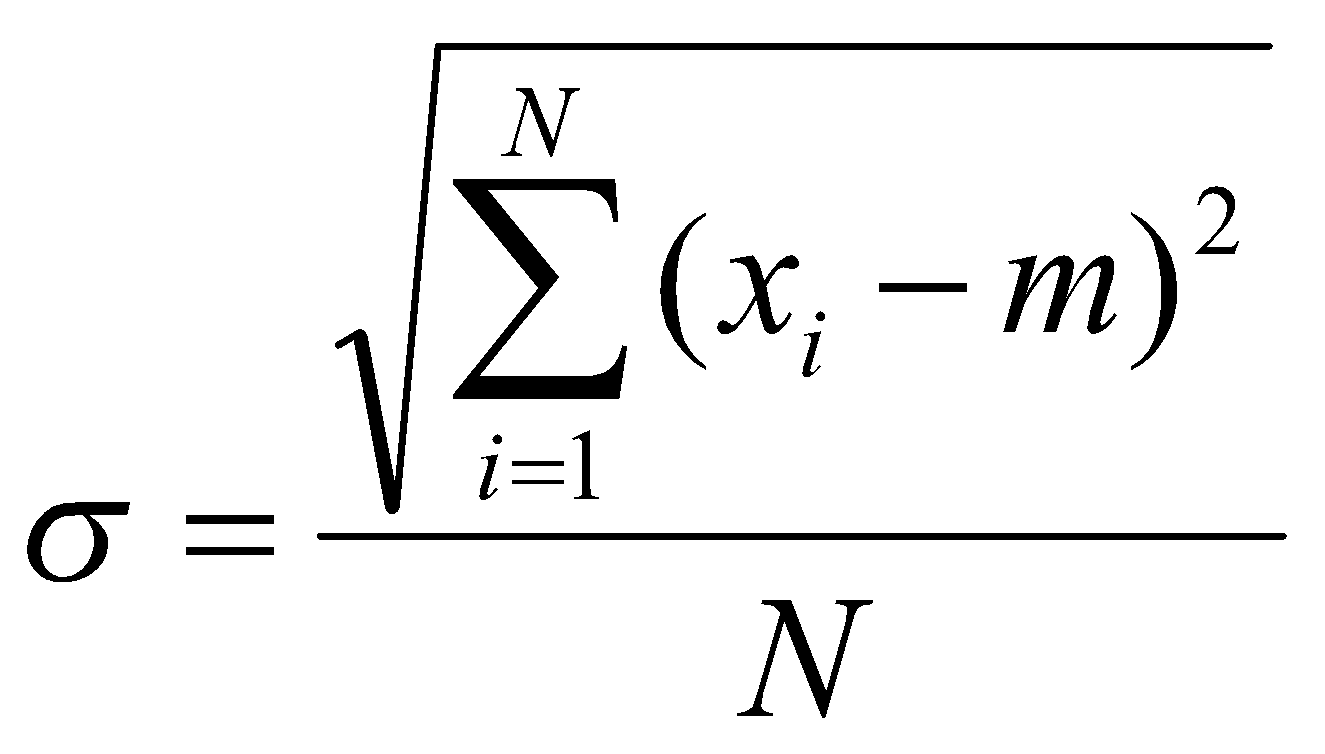
|  | AND | OR | NOT |
| --- | --- | --- | --- |
| Operando 1 | 1,1,1,1,0,0,0,0 | 1,1,1,1,0,0,0,0 | 1,1,0,1,0,0,1,0 |
| Operando 2 | 1,0,1,0,1,0,1,0 | 1,0,1,0,1,0,1,0 |  |
| Resultado | **1,0,1,0,0,0,0,0** | **1,1,1,1,1,0,1,0** | **0,0,1,0,1,1,0,1** |

# 8. Ejercicio

Realizar las funciones cuyos prototipos y objetivos se detallan a continuación:

* **double** sumatoria (**double** a[], **unsigned int** t); Retorna la sumatoria de todos los valores del arreglo a de tamaño t.
* **double** promedio (**double** a[], **unsigned int** t); Retorna el promedio de todos los valores del arreglo a de tamaño t.
* **double** desvio (**double** a[], **unsigned int** t); Retorna el desvío estándar de todos los valores del arreglo a de tamaño t.

Con estas funciones realizar un programa donde se ingresen por teclado los valores de un arreglo (de tamaño fijo y constante). A continuación mostrar el desvío estándar de los valores ingresados. Debe funcionar para arreglos de cualquier tamaño. El desvío estándar (σ) es una medida estadística de dispersión y la fórmula usual para su cálculo es:



***N =*** Cantidad de valores

***Xi*** = Cada uno de los valores

***m*** = Promedio de todos los valores

# 9. Ejercicio

Confeccionar un programa que solicite al usuario una cantidad de números al azar que se generará entre el 0 y el 36. Presentar luego un informe que indique qué cantidad de veces salió cada valor y el porcentaje que representa. No mostrar aquellos números que no hayan salido. Utilizar funciones en todos los aspectos posibles y eficientemente. Ejemplo:

Ingrese cantidad de valores: ***10***

Informe:

El numero 8 salió 1 vez (10%).

El numero 14 salió 2 veces (20%).

El numero 20 salió 4 veces (40%).

El numero 23 salió 1 veces (10%).

El numero 30 salió 2 veces (20%).

# 10. Ejercicio

Ingresar números enteros (incluido el 0) en un arreglo de hasta 20 elementos. Finalizar la carga con un número negativo. A continuación mostrar una gráfica de ‘\*’ representando al cantidad y mostrando el porcentaje del total acumulado de valores ingresados. Considerar que para el 100% se deben dibujar 60 asteriscos y el resto de los porcentajes, proporcionales a este valor. Utilizar funciones en todos los aspectos posibles y eficientemente. Ejemplo:

Ingrese número: ***4***

Ingrese número: ***8***

Ingrese número: ***15***

Ingrese número: ***16***

Ingrese número: ***23***

Ingrese número: ***42***

Ingrese número: ***-1***

Gráfica:

4 --> \*\* (3.70%)

8 --> \*\*\*\* (7.41%)

15 --> \*\*\*\*\*\*\*\* (13.89%)

16 --> \*\*\*\*\*\*\*\* (14.81%)

23 --> \*\*\*\*\*\*\*\*\*\*\*\* (21.30%)

42 --> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (38.89%)

**STRINGS**

# 11. Ejercicio

Programar una función que reciba como parámetros un string y dos caracteres. La función debe retornar una copia del string recibido reemplazando en él todas las apariciones del primer carácter por el segundo carácter. Implementarla en un programa donde el usuario ingrese un string, un carácter al cual reemplazar y con qué carácter reemplazarlo. Luego debe mostrar el string modificado. Ejemplo:

Ingrese texto: ***La mar estaba serena***

Ingrese carácter a reemplazar: ***a***

Ingrese carácter de reemplazo: ***e***

Resultado: Le mer estebe serene

# 12. Ejercicio

Programar una función que reciba como parámetro un string, que lo modifique eliminando todo lo que no sean letras que encuentre al principio y al final del texto y, posteriormente lo retorne. Utilizarla en un programa de ejemplo. Dado el texto: “... La bella y graciosa moza marchose a lavar la ropa, $@”, aplicada la función debe obtenerse “La bella y graciosa moza marchose a lavar la ropa”.

# 13. Ejercicio

Realizar una función que reciba como parámetro un string (el cual no puede modificarse) y que lo imprima centrado en la pantalla y con los signos “<” y “>” a la izquierda y a la derecha completando el espacio en blanco. Tomar en cuenta que el ancho total de la pantalla de consola es de 80 caracteres (se asume que la función no recibirá strings más largos que el ancho total). Implementarla en un programa donde se ingrese un texto por teclado y luego se imprima utilizando la función pedida. Ejemplo:

Ingrese texto: ***Su voz tenía la sonoridad del rugido del león***

<<<<<<<<<<<<<<<<<Su voz tenía la sonoridad del rugido del león>>>>>>>>>>>>>>>>>

# 14. Ejercicio

Por cada uno de los siguientes prototipos, programar la implementación de las funciones de acuerdo a lo solicitado. Utilizarlas en un programa de prueba que verifique su funcionamiento correcto. En todos los casos se asume que los strings no contienen vocales acentuadas ni “ñ”. Las funciones deben servir para arreglos de cualquier tamaño, el cual se desconoce dentro de la función. En ningún caso se pide imprimir ni ingresar datos por teclado.

* **bool** es\_anterior\_i (**string s1**, **string s2**);

Compara los strings retornando **true** cuando el string s1 sea alfabéticamente anterior al string s2, considerando a las letras mayúsculas de sus respectivas minúsculas como iguales. Por ejemplo, si s1 = “hola” y s2 = “Mundo”, la función deberá retornar 1 (siendo que la "h“ es anterior a la "M", lo cual ocurre de manera inversa al comparar los strings con "<"). Los strings s1 y s2 no pueden modificarse.

* **unsigned int** contar\_palabras (**string s1**);

Debe retornar la cantidad de palabras que hay en el string s1. Se considera parte de una palabra a toda letra o carácter numérico, considerando a todos los demás símbolos como separadores de palabras. No puede modificar el contenido de s1.

* **unsigned int** contar\_apariciones (**string s1**, **string s2**);

Debe retornar la cantidad de veces que aparece el string s2 dentro del string s1. La función no debe considerar diferentes a las letras mayúsculas de sus respectivas minúsculas. Los strings s1 y s2 no pueden modificarse.

* **string** reemplazar (**string** buscar, **string** reemplazo, **string** sujeto);

Debe reemplazar todas las apariciones del string "buscar" por "reemplazo" dentro del string "sujeto" (que no puede modificarse) y retornar el string resultante. La función no debe considerar diferentes a las letras mayúsculas de sus respectivas minúsculas. Ejemplo: reeemplazar ("HOLA", "adios", “Hola don Pepito, hola don Jose”), debería retornar "adios don Pepito, adios don Jose”.

# 15. Ejercicio

Confeccionar una función booleana que reciba como parámetros dos frases (en dos strings independientes) y que determine si ambas frases son iguales. Es decir, si contienen las mismas palabras en el mismo orden. Por ejemplo: “Hola, como te va?” debe ser considerada igual a “hola… Como te va!”.

# 16. Ejercicio

En un arreglo de strings de N elementos(N como constantes), pueden almacenarse N strings. Para ello, es conveniente disponer de las siguientes funciones:

* **void** cargar\_lista (**string** [], **unsigned int**);

Permite ingresar por teclado strings, cada uno finalizando con ENTER y los guarda en el arreglo recibido como primer parámetro cuyo tamaño se recibe como segundo parámetro. Tomar como convención que un string nulo (con ‘\0’ como único carácter) sea la marca de finalización de strings útiles.

* **void** imprimir\_lista (**string** []);

Imprime la lista de strings separados por un salto de línea.

* **void** ordenar\_lista (**string** []);

Ordena la lista alfabéticamente.

Programar estas funciones y aplicarlas en un programa de ejemplo. Considere ayudarse de las funciones desarrolladas para strings (ejercicio 5.14).