

Apellido y Nombre:

Fecha:/...../.....

CONCEPTOS A TENER EN CUENTA

Un **arreglo** es un conjunto homogéneo de elementos que comparten un nombre común; y que podría entenderse como una variable capaz de almacenar más de un valor al mismo tiempo. Para acceder a los elementos de un arreglo es preciso utilizar índices que permitan identificar elementos individuales. En programación, un arreglo se define como una zona de almacenamiento contiguo (posiciones de memoria consecutivas) que contiene una serie de elementos, todos del mismo tipo (enteros, reales, lógicos, etc.) que pueden ser procesados individualmente o en conjunto.

Arreglos Unidimensionales o Vectores

Un vector es un arreglo que sólo requiere de un índice para acceder a sus elementos. Por ejemplo, los arreglos **Vector_1** y **Vector_2**, mostrados a continuación, ocupan 8 posiciones de memoria para almacenar, respectivamente, datos de tipo numérico y tipo alfanumérico. Cada uno de estos elementos se identifica con un número (índice) que permite utilizarlo de forma individual.

Vector_1 →	33	18	3	65	94	4	7	12
	1	2	3	4	5	6	7	8

Vector_2 →	'a'	'*'	'h'	'@'	'4'	'+'	'B'	'<'
	1	2	3	4	5	6	7	8

Índices de Vectores

Los índices permiten referenciar, directamente, posiciones específicas de un vector. Los índices pueden ser **valores constantes ordinales** (por ejemplo, **Vector_1[5]**), **variables ordinales** (por ejemplo, **Vector_1[i]**) o **expresiones de tipo ordinal** (por ejemplo, **Vector_1[i+2]**)

Rango

El número de elementos de un vector se conoce como **Rango del Vector**. Este se determina mediante la siguiente expresión:

$$\text{Rango de Vector} = \text{LS} - \text{LI} + 1$$

Límite inferior (LI): valor mínimo permitido para el índice

Límite superior (LS): el valor máximo permitido para el índice.

Por ejemplo, podría definirse un arreglo de 50 elementos considerando LI=1 y LS=50

$$\text{Rango de Vector} = \text{LS} - \text{LI} + 1 \quad \rightarrow \quad \text{Rango de Vector} = 50 - 1 + 1 \quad \rightarrow \quad \text{Rango de Vector} = 50.$$

Lectura y Escritura

Para acceder a un elemento específico de un vector debe indicarse el nombre del vector y el índice que identifica su posición. Por ejemplo, para mostrar el segundo elemento de **Vector_1** y el quinto elemento de **Vector_2** se procede como sigue:

Escribir Vector_1 [2] //nos mostrará el valor 18

Escribir Vector_2 [i] //con i=5, entonces, nos mostrará el valor '4'

Para almacenar valores en un vector es necesario especificar a qué elemento se accederá y el operador de asignación (\leftarrow). Por ejemplo, para almacenar el valor '\$' en la tercera posición de **Vector_2** se utiliza la instrucción **Vector_2[3] \leftarrow '\$'**. Es posible "cargar" vectores utilizando estructuras de control repetitivas que incluyan operaciones de asignación. El siguiente programa (en pseudocódigo y C/C++) ilustra la carga de vectores.

```

PROGRAMA uso_vectores
CONSTANTES
    MAX=10
TIPOS
    tvector=ARREGLO [1..MAX] de CARACTER
VARIABLES
    datos: tvector
    i: ENTERO
INICIO
    PARA i DESDE 1 HASTA MAX HACER
        ESCRIBIR "Ingrese un carácter:"
        LEER datos[i]
    FIN_PARA
    PARA i DESDE 1 HASTA MAX HACER
        Escribir datos[i]
    FIN_PARA
FIN

```

```

#include <iostream>
#include <stdlib.h>

using namespace std;

const int MAX=5;

typedef char tvector[MAX];

main ()
{
    tvector datos;
    int i;
    for(i=0; i < MAX; i++)
    {
        cout << "Ingrese un carácter: ";
        cin >> datos[i];
    }
    for(i=0; i < MAX; i++)
        cout << datos[i] << endl;
    system("pause");
}

```

Arreglos Bidimensionales o Matrices

Una matriz es un arreglo de 2 dimensiones, organizado en filas y columnas, cuyos elementos se acceden utilizando 2 índices. El primer índice identifica la fila a la que pertenece un elemento mientras que el segundo indica la columna en la que se encuentra dicho elemento. Las matrices se representan mediante tablas con dimensión M x N, donde M es el número de filas, y N el número de columnas.

Matriz_1		1	2	3	4
	1	4	7	5	9
	2	3	1	4	2
	3	5	8	3	1
	4	9	4	7	2

Matriz_2		1	2	3	4
	1	A	O	S	O
	2	B	M	L	M
	3	M	L	V	X
	4	G	P	H	C
	5	D	C	J	W

Matriz_1: Una matriz de tipo numérico con una dimensión de 4 x 4.

Matriz_2: Una matriz de tipo carácter con una dimensión de 5 x 4.

Si la matriz tiene igual número de filas que de columnas se dice que se trata de una matriz cuadrada. En las matrices cuadradas los elementos que ocupan las posiciones donde los índices de fila y columna son iguales reciben el nombre de diagonal principal. Por ejemplo, en la **Matriz_1** los elementos de la diagonal principal son: **4, 1, 3, 2**.

Índices de Matrices

Los índices permiten referenciar, directamente, posiciones específicas de los elementos de una matriz. Los índices pueden ser **valores constantes ordinales** (por ejemplo, **Matriz_1[1,1]**), **variables ordinales** (por ejemplo, **Matriz_1[i,j]**) o **expresiones de tipo ordinal** (por ejemplo, **Matriz_1[i+2,j]**)

Rango

El número o cantidad de elementos de una matriz se conoce como **Rango de la Matriz**. Éste se calcula como sigue:

$$\text{Rango de la matriz} = (\text{LSF} - \text{LIF} + 1) * (\text{LSC} - \text{LIC} + 1)$$

LSF: límite superior del índice de filas

LIF: límite inferior del índice de filas

LSC: límite superior del índice de columnas

LIC: límite inferior del índice de columnas

Por ejemplo, el rango para **Matriz_1**, **Rango de Matriz_1 = (LSF-LIF+1)*(LSC-LIC+1)**, entonces

$$\text{Rango de Matriz}_1 = (4-1+1)*(4-1+1)=16$$

Lectura y Escritura

Para acceder a un elemento específico de una matriz, se indica el nombre de la matriz y los índices que refieren a una posición. Por ejemplo, para mostrar el valor que se encuentra en la *fila 2, columna 3* de la **Matriz_2**, y el valor que se encuentra en la *fila 5, columna 1* de la misma matriz, se procede como sigue:

Escribir Matriz_1 [2,3] //nos mostrará el valor 4

Escribir Matriz_2 [i,j] //con i=5 y j=1 entonces, nos mostrará el valor 'D'

Para almacenar valores en una matriz es necesario especificar a qué elemento se accederá y el operador de asignación (\leftarrow). Por ejemplo, para almacenar el dato 'H' en la *cuarta fila, segunda columna* de **Matriz_2** se utiliza la instrucción **Matriz_2[4,2] \leftarrow 'H'**. Es posible "cargar" matrices utilizando estructuras de control repetitivas anidadas que incluyan operaciones de asignación. El siguiente programa (en pseudocódigo y C/C++) ilustra la carga de matrices.

```
PROGRAMA matriz_1
CONSTANTES
    FILA=4
    COL=4
TIPOS
    t_matriz=ARREGLO[1..FILA,1..COL] de CARACTER
VARIABLES
    letras: tmatriz
    i,j:ENTERO
INICIO
    PARA i DESDE 1 HASTA FILA HACER
        PARA j DESDE 1 HASTA COL HACER
            ESCRIBIR "Ingrese un caracter"
            LEER letras[i,j]
        FIN PARA
    FIN PARA
    PARA i DESDE 1 HASTA FILA HACER
        PARA j DESDE 1 HASTA COL HACER
            ESCRIBIR letras[i,j]
        FIN PARA
    FIN PARA
FIN

#include <iostream>
#include <stdlib.h>

using namespace std;

const int FILA=4, COL=4;

typedef char tmatriz [FILA][COL];

main ()
{
    tmatriz letras;
    int i,j;
    for(i=0;i < FILA;i++)
        for(j=0;j < COL;j++)
            { cout<<"Ingrese un caracter"<<endl;
              cin>>letras[i][j];
            }
    for(i=0;i < FILA;i++)
        for(j=0;j < COL;j++)
            cout<< letras[i][j] <<endl;
    system("pause");
}
```

Arreglos Multidimensionales

Un arreglo multidimensional, puede definirse de tres, cuatro o N dimensiones. De acuerdo a la cantidad de dimensiones será necesario especificar la cantidad de índices para acceder a los elementos individuales del arreglo. Por ejemplo, si el arreglo tiene 3 dimensiones, se requieren 3 índices; si el arreglo tiene 4 dimensiones, serán necesarios 4 índices; y si el arreglo tiene n dimensiones, entonces se trabajará con n índices para acceder a los elementos.

Índices de Arreglos Multidimensionales

Los índices permiten trabajar con elementos o posiciones específicas de un arreglo. En el caso de los multidimensionales, se utilizarán tantos índices como dimensiones tenga el arreglo. Así, para acceder a los elementos del arreglo **Multi 4x5x30** (de 3 dimensiones) se requieran 3 referencias o índices, por ejemplo, **Multi[3,2,12]**.

Los índices pueden ser **valores constantes ordinales** (por ejemplo, **Multi[3,2,12]**), **variables ordinales** (por ejemplo, **Multi[i,j,k]**) o **expresiones de tipo ordinal** (por ejemplo, **Multi[3+2,j-3,5]**)

Rango

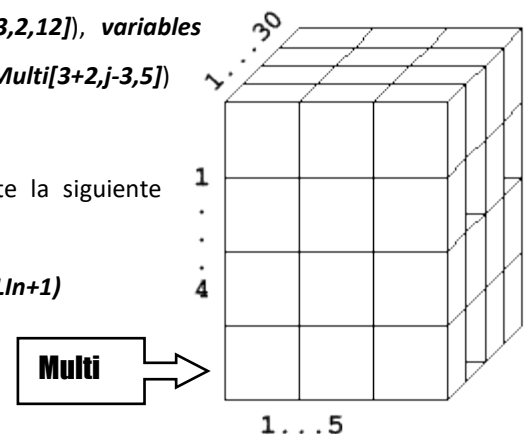
El número de elementos de arreglo multidimensional se determina mediante la siguiente expresión:

$$\text{Rango del arreglo multidimensional} = (LS1 - LI1 + 1) * (LS2 - LI2 + 1) * \dots * (LSn - LIn + 1)$$

LS1: límite superior del índice de la dimensión 1.

LI1: límite inferior del índice de la dimensión 1.

LS2: límite superior del índice de la dimensión 2.



LI2: límite inferior del índice de la dimensión 2.

...

LSn: límite superior del índice de la dimensión n.

Lin: límite inferior del índice de la dimensión n.

Por ejemplo, el rango para **Multi**, $\text{Rango de Multi} = (LS1-LI1+1) * (LS2-LI2+1) * (LS3-LI3+1)$, entonces

$$\text{Rango de Multi} = (4-1+1) * (5-1+1) * (30-1+1) = 600$$

Ejemplo: realizar la carga de valores a Multi, en pseudocódigo y en c++:

<pre> PROGRAMA Multidimensional CONSTANTES D1=4 D2=5 D3=30 TIPOS tmulti=ARREGLO [1..D1, 1..D2, 1..D3] de REAL VARIABLES num: tmulti i,j,k:entero INICIO PARA i DESDE 1 HASTA D1 HACER PARA j DESDE 1 HASTA D2 HACER PARA k DESDE 1 HASTA D3 HACER ESCRIBIR "Ingrese un valor: " LEER num[i,j,k] FIN_PARA FIN_PARA FIN_PARA PARA i DESDE 1 HASTA D1 HACER PARA j DESDE 1 HASTA D2 HACER PARA k DESDE 1 HASTA D3 HACER ESCRIBIR num[i,j,k] FIN_PARA FIN_PARA FIN_PARA FIN </pre>	<pre> #include <iostream> #include <stdlib.h> using namespace std; const int D1=4, D2=5, D3=30; typedef float tmulti[D1][D2][D3]; main () { tmulti num; int i,j,k; for(i=0;i < D1;i++) for(j=0;j < D2;j++) for(k=0;k < D3;k++) { cout<<"Ingrese un valor: "; cin>>num[i][j][k]; } for(i=0;i < D1;i++) for(j=0;j < D2;j++) for(k=0;k < D3;k++) cout<<num[i][j][k]<<endl; system("pause"); } </pre>
--	---

EJERCICIOS RESUELTOS

1. Dado un vector de valores enteros de tamaño 20, diseñe un programa modular (los procedimientos y funciones necesarios) que permita cargar elementos en el vector y mostrar los valores almacenados.

```

programa vectores
constantes
    MAX=20
tipos
    vector=arreglo [1..MAX] de entero
variables
    num:vector
    opcion, ocupado:entero

procedimiento agregar (E/S n:vector,E/S ocup:entero)
inicio
    si ocup=MAX entonces
        escribir "Vector Completo"
    sino
        ocup←ocup+1
        escribir "Ingrese valor:"
        leer n[ocup]
    fin_si
fin

procedimiento mostrar (E m:vector, E ocup:entero)
variables
    i:entero
inicio
    para i desde 1 hasta ocup hacer
        escribir m[i]
    fin_para
fin
    
```

```

inicio
    ocupado<-0
    repetir
        escribir "1-cargar vector"
        escribir "2-mostrar vector"
        escribir "3-salir"
        escribir "ingrese opcion:"
        leer opcion
        según opcion hacer
            1: agregar(num,ocupado)
            2: mostrar(num,ocupado)
            3: escribir "fin del programa..."
        de otro modo: escribir "opcion incorrecta"
    fin_según
    hasta_que opcion=3
fin

```

2. Considerando una matriz 3X3 (matriz cuadrada) de valores reales, diseñe un programa (y los procedimientos y funciones necesarios) que permita cargar la matriz, calcular la suma de su diagonal principal y mostrar los elementos de la matriz.

```

programa matrices
constantes
    FILAS=3, COLUMNAS=3
tipos
    matriz=arreglo [1.. FILAS,1.. COLUMNAS] de real
variables
    numeros: matriz
    opcion:entero
procedimiento cargar_matriz (E/S num:matriz)
variables
    i,j:entero
inicio
    para i desde 1 hasta FILAS hacer
        para j desde 1 hasta COLUMNAS hacer
            escribir "ingrese elemento [",i,",",j,"]:"
            leer num[i,j]
        Fin_para
    Fin_para
fin
funcion diag_matriz (E num:matriz):real
variables
    i,j:entero
    suma:real
inicio
    suma<-0
    para i desde 1 hasta FILAS hacer
        para j desde 1 hasta COLUMNAS hacer
            si i=j entonces
                suma<-suma+num[i,j]
            fin_si
        fin_para
    fin_para
    diag_matriz<-suma
fin
procedimiento mostrar_matriz (E num:matriz)
variables
    i,j:entero
inicio
    escribir "valores almacenados en la matriz"
    para i desde 1 hasta FILAS hacer
        para j desde 1 hasta COLUMNAS hacer
            escribir "dato[",i,",",j,"]:", num[i,j]
        fin_para
    fin_para
fin
inicio
    repetir
        escribir "1-cargar matriz"
        escribir "2-suma de la diagonal principal"
        escribir "3-mostrar valores de la matriz"
        escribir "4-salir"
        escribir "ingrese opcion:"
        leer opcion

```

```

según opcion hacer
  1: escribir "cargar matriz"
    cargar_matriz(numeros)
  2: escribir "Suma de la diag. principal"
    escribir "suma=",diag_matriz(numeros)
  3: mostrar_matriz(numeros)
  4: escribir "programa finalizado"
    de otro modo: escribir "opcion incorrecta"
fin_según
hasta_que opcion=4
fin

```

3. Dado un arreglo 3x2x6 (tridimensional) de caracteres, diseñe un programa (y los procedimientos y funciones necesarios) que permitan cargar el arreglo, contar las veces que aparece una letra indicada por el usuario y mostrar los elementos del arreglo.

```

programa tridimensional
constantes
  D1=3, D2=2, D3=6
tipos
  tridim=arreglo [1.. D1,1..D2,1..D3] de carácter
variables
  alfab:tridim
  letra:carácter
  opcion:entero
procedimiento cargar_arreglo(E/S alfa:tridim)
variables
  i,j,k:entero
inicio
  para i desde 1 hasta D1 hacer
    para j desde 1 hasta D2 hacer
      para k desde 1 hasta D3 hacer
        escribir "ingrese elemento:"
        leer alfa[i,j,k]
      fin_para
    fin_para
  fin_para
fin
funcion contarletra(E alfa:tridim,E let:caracter):entero
variables
  i,j,k,contador:entero
inicio
  contador←0
  para i desde 1 hasta D1 hacer
    para j desde 1 hasta D2 hacer
      para k desde 1 hasta D3 hacer
        si let=alfa[i,j,k] entonces
          contador←contador+1
        fin_si
      fin_para
    fin_para
  fin_para
  contarletra←contador
fin
procedimiento mostrar_arreglo(E alfa:tridim)
variables
  i,j,k:entero
inicio
  escribir "valores almacenados en el arreglo"
  para i desde 1 hasta D1 hacer
    para j desde 1 hasta D2 hacer
      para k desde 1 hasta D3 hacer
        escribir "letra:",alfa[i,j,k]
      fin_para
    fin_para
  fin_para
fin
procedimiento menu (E/S opcion: entero)
inicio
  escribir "1-cargar arreglo"
  escribir "2-contar letra"
  escribir "3-mostrar valores del arreglo"
  escribir "4-salir"
  escribir "ingrese opcion:"

```

```

    leer opcion
fin
inicio
  repetir
    menu(opcion)
  según opcion hacer
    1: escribir "cargar arreglo"
      cargar_arreglo(alfab)
    2: escribir "ingrese letra a contar:"
      leer letra
      escribir "contadas:", contarletra(alfab, letra)
    3: mostrar_arreglo(alfab)
    4: escribir "programa finalizado"
  de otro modo: escribir "opcion incorrecta"
  fin_según
  hasta_que opcion=4
fin

```

EJERCICIOS A RESOLVER

1. Dadas las siguientes definiciones de arreglo, determine para cada uno: tipo de arreglo, número de dimensiones y calcule el *rango del arreglo* (cantidad de elementos):

tarreglo6=ARREGLO [1..10,1..15] de ENTERO

tarreglo1=ARREGLO [-5..33] de LÓGICO

tarreglo2=ARREGLO [-3..7,-9..2,-3..5,-4..9] de ENTERO

tarreglo3=ARREGLO [-1..6,-7..5,15..22] de CARÁCTER

tarreglo4=ARREGLO [-2..25,-3..14] de CARÁCTER

2. Diseñe un programa modular que gestione un vector de 20 valores enteros a través de un menú que presente las siguientes opciones:
 - a) **agregar** elementos al vector,
 - b) **determinar** los valores máximo y mínimo del vector,
 - c) **calcular** la suma de elementos del vector mediante un **algoritmo recursivo**.
 - d) **mostrar** los valores almacenados en el vector.
3. Diseñe un programa modular que gestione un vector de 15 valores reales a través de un menú que presente las siguientes opciones:
 - a) **insertar** elementos en el vector (orden decreciente),
 - b) **borrar** un valor especificado por usuario,
 - c) **calcular** el promedio de los elementos almacenados en el vector,
 - d) **mostrar** los valores del vector mediante un **algoritmo recursivo**.
4. Diseñe un programa modular que gestione un vector de 10 valores caracteres a través de un menú que presente las siguientes opciones:
 - a) **insertar** elementos en el vector (siempre por la primera posición),
 - b) **buscar** un valor en el vector aplicando el algoritmo secuencial
 - c) **contar** las mayúsculas, minúsculas y símbolos del vector.
 - d) **mostrar**, en orden inverso, los valores del vector mediante un **algoritmo recursivo**.

5. Realice las modificaciones, indicadas a continuación, de los algoritmos básicos de actualización:

```

PROCEDIMIENTO agregar(E/S a:tvector, E/S ocup:ENTERO, E nuevo:ENTERO)
INICIO
  SI ocup=MAX ENTONCES
    ESCRIBIR "VECTOR COMPLETO"
  SINO
    ocup ← ocup+1
    a[ocup] ← nuevo
  FIN_SI
FIN

```

Modificación 1: Los elementos deben agregarse siempre por la primera posición del arreglo.

Modificación 2: La operación debe verificar que no se agreguen valores repetidos.

```

PROCEDIMIENTO insertar(E/S a:tvector, E/S ocup:ENTERO, E nuevo:REAL)
VARIABLES
    i,j:ENTERO
INICIO
    SI ocup=MAX ENTONCES
        ESCRIBIR "VECTOR COMPLETO"
    SINO
        i←1
        MIENTRAS i <= ocup Y nuevo > a[i] HACER
            i←i+1
        FIN_MIENTRAS
        j←ocup
        MIENTRAS j >= i HACER
            a[j+1]←a[j]
            j←j-1
        FIN_MIENTRAS
        a[i]←nuevo
        ocup←ocup+1
    FIN_SI
FIN
PROCEDIMIENTO borrar(E/S a:tvector, E/S ocup:ENTERO, E buscado:CARACTER)
VARIABLES
    i,j:ENTERO
    existe:LÓGICO
INICIO
    SI ocup=0 ENTONCES
        ESCRIBIR "VECTOR VACÍO"
    SINO
        i←1
        existe←FALSO
        MIENTRAS i <= ocup Y existe=FALSO HACER
            SI buscado=a[i] ENTONCES
                existe←VERDADERO
            SINO
                i←i+1
            FIN_SI
        FIN_MIENTRAS
        SI existe=VERDADERO ENTONCES
            MIENTRAS i < ocup HACER
                a[i]←a[i+1]
                i←i+1
            FIN_MIENTRAS
            ocup←ocup-1
            ESCRIBIR "VALOR ELIMINADO"
        SINO
            ESCRIBIR "VALOR INEXISTENTE"
        FIN_SI
    FIN_SI
FIN

```

Modificación 1: La posición de inserción del nuevo valor será especificada por el usuario.

Modificación 2: La operación debe verificar que no se inserten valores repetidos.

Modificación 1: La posición borrada será especificada por el usuario.

Modificación 2: operación debe borrar todas las apariciones del valor en el arreglo.

6. Considerando el algoritmo de búsqueda binaria

a) Realice la prueba de escritorio del algoritmo para los valores 35 y 10 sobre el siguiente vector

5	9	10	14	19	32	39	47	53	65	68	74	88	91	97
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

b) Modifique el algoritmo de modo que trabaje sobre un arreglo ordenado en forma decreciente

c) Escriba una versión recursiva del algoritmo

7. Dados 2 vectores de enteros de 7 posiciones (cada uno), diseñe un programa modular que permita:

d) **cargar** valores en los vectores (carga completa),

e) **sumar**, posición a posición, los elementos de los vectores almacenando cada resultado obtenido en un tercer vector.

$C = A + B$, es igual a: $C[1] = A[1] + B[1]$, $C[2] = A[2] + B[2]$, ..., $C[i] = A[i] + B[i]$, ..., $C[7] = A[7] + B[7]$

f) **multiplicar**, posición a posición, los elementos de los vectores guardando cada resultado en el primer vector.

$A = A * B$, es igual a: $A[1] = A[1] * B[1]$, $A[2] = A[2] * B[2]$, ..., $A[i] = A[i] * B[i]$, ..., $A[7] = A[7] * B[7]$

- g) **calcular el cuadrado**, posición a posición, de los elementos del primer vector (reemplazando el contenido de cada posición por el valor calculado).

- h) mostrar el contenido de los vectores

Nota: Considere que el **producto** (sumas sucesivas) y el **cálculo de cuadrado** (suma de n primeros impares) se realizan mediante **algoritmos recursivos**.

8. Dada una matriz 4x4 de caracteres, diseñe un programa modular que permita:

- a) **cargar** los elementos de la matriz (desde la última posición hacia la primera),
- b) **buscar** un valor en la matriz
- c) **mostrar**, por filas, el contenido de la matriz,
- d) **mostrar**, por columnas, el contenido de la matriz
- e) **determinar** los valores máximo y mínimo de la diagonal principal

9. Considerando una matriz 3x4 de valores enteros, diseñe un programa modular que permita:

- a) **cargar** valores en la matriz,
- b) **determinar** las coordenadas (fila y columna) de un valor solicitado por el usuario
- c) **copiar** el contenido de la matriz a un vector
- d) **reemplazar** cada posición de la matriz por su correspondiente valor factorial (**algoritmo recursivo**)
- e) **generar** una nueva matriz cuyo contenido sea el **cubo** de cada posición de la matriz original.

Nota: Considere que el cálculo de **cubo** se realizará aplicando el método basado en sumas visto en prácticos anteriores.

10. Considerando un arreglo 2x3x4 de enteros, diseñe un programa modular que permita:

- a) **cargar** elementos en el arreglo,
- b) **contar** los números múltiplos de 3 del arreglo (aplicando el método basado en la suma de los dígitos de un número)
- c) **mostrar** los elementos impares del arreglo.

11. Considerando 3 arreglos A, B y C de 3x2x4 de enteros, diseñe un programa modular que permita:

- a) **cargar** valores en los arreglos A y B,
- b) **determinar** si los arreglos A y B son iguales o no.
- c) **calcular la potencia**, posición a posición, de los elementos de A elevados a los elementos de B, guardando cada resultado en el arreglo C,
- d) **mostrar** el contenido de los arreglos

Nota: Considere que la **potencia** se calcula mediante productos sucesivos.

12. Dado un arreglo 4x5x2x3 de caracteres diseñe un programa (y las subrutinas) que permita:

- a) **cargar** elementos en el arreglo,
- b) **copiar** el contenido del arreglo a un vector,
- c) **contar** las minúsculas almacenadas en el arreglo multidimensional.
- d) **buscar** un valor en el arreglo y mostrar sus coordenadas.

13. Considerando las definiciones a) y b), presentadas a continuación, determine el propósito del módulo misterio. ¿Cuál será el resultado en cada caso?

```
a. const int D1=4, D2=3, D3=5, D4=2;
b. const int D1=5, D2=9, D3=6, D4=10;

typedef char tmulti[D1][D2][D3][D4];
```

```
int misterio(tmulti datos)
{ int i,j,k,l;
  int cont=0;
  for(i=0;i < D1 ;i++)
    for(j=0;j < D2 ;j++)
      for(k=0;k < D3 ;k++)
        for(l=0;l < D4 ;l++)
          cont++;
  return cont;
}
```

14. Dada la siguiente definición de datos:

- a) realice la prueba de escritorio del módulo *nn*
- b) determine los valores adecuados para Z y Q de modo que se realice un uso eficiente de la memoria de la computadora

```
const int X=3, Y=3, Z=?, Q=?;
typedef float tarreglo1[X][Y];
typedef float tarreglo2[Z];
typedef float tarreglo3[Q];

void nn(tarreglo1 a, tarreglo2 &b, tarreglo3 &c)
{ int i,j,k=0, l=0;
  for(i=0;i<=X-1;i++)
    for(j=0;j<=Y-1;j++)
    {
      if (i==j)
      { b[l]=a[i][j];
        l++; }
      else
      { c[k]=a[i][j];
        k++; }
    }
}
```

15. Dada la siguiente definición, determine el propósito de los algoritmos presentados a continuación:

```
const int MAX=10;
typedef int tvector[MAX];
typedef char tmat[MAX][MAX];
typedef float tmul[MAX][MAX][MAX];

int enigma(tvector a, int ocup)
{int aux;
  if (ocup==1)
    return a[ocup];
  else
  { aux=enigma(a,ocup-1);
    if (aux > a[ocup])
      return aux;
    else
      return a[ocup];
  }
}

bool oculta(tmat m, tmat n)
{ int i,j;
  bool b=true;
  for(i=0;i<MAX && b==true;i++)
    for(j=0;j<MAX && b==true;j++)
      if (m[i][j]!=n[i][j])
        b=false;
  return b;
}

int misterio(tmul q, float n)
{ int i, j,k,r=0;
  for(i=0;i<MAX;i++)
    for(j=0;j<MAX;j++)
      for(k=0;k<MAX;k++)
        if (q[i][j][k]!=n)
          r++;
  return r;
}
```

■ ■ ■