
**UNIVERSIDAD AUTÓNOMA DE SINALOA**

*Facultad de Informática Culiacán*

# Modularidad

# OBJETIVO EDUCACIONAL:

El alumno:

- Conocerá y utilizará la técnica de dividir un problema en módulos para facilitar su resolución y mantenimiento, así como la reutilización de Algoritmos ya declarados.

# Programación Modular

- Un problema difícil es más sencillo al dividirlo en pequeñas partes y tratar de buscar la solución de cada una de ellas y así resolver todo el problema general.

- En programación la mejor forma de elaborar y dar mantenimiento a un programa complejo, es construirlo a partir de bloques menores o módulos, pues de esta manera es más fácil analizar, programar y darle mantenimiento a cada uno de ellos, que si estuviera todo el código en el programa principal.

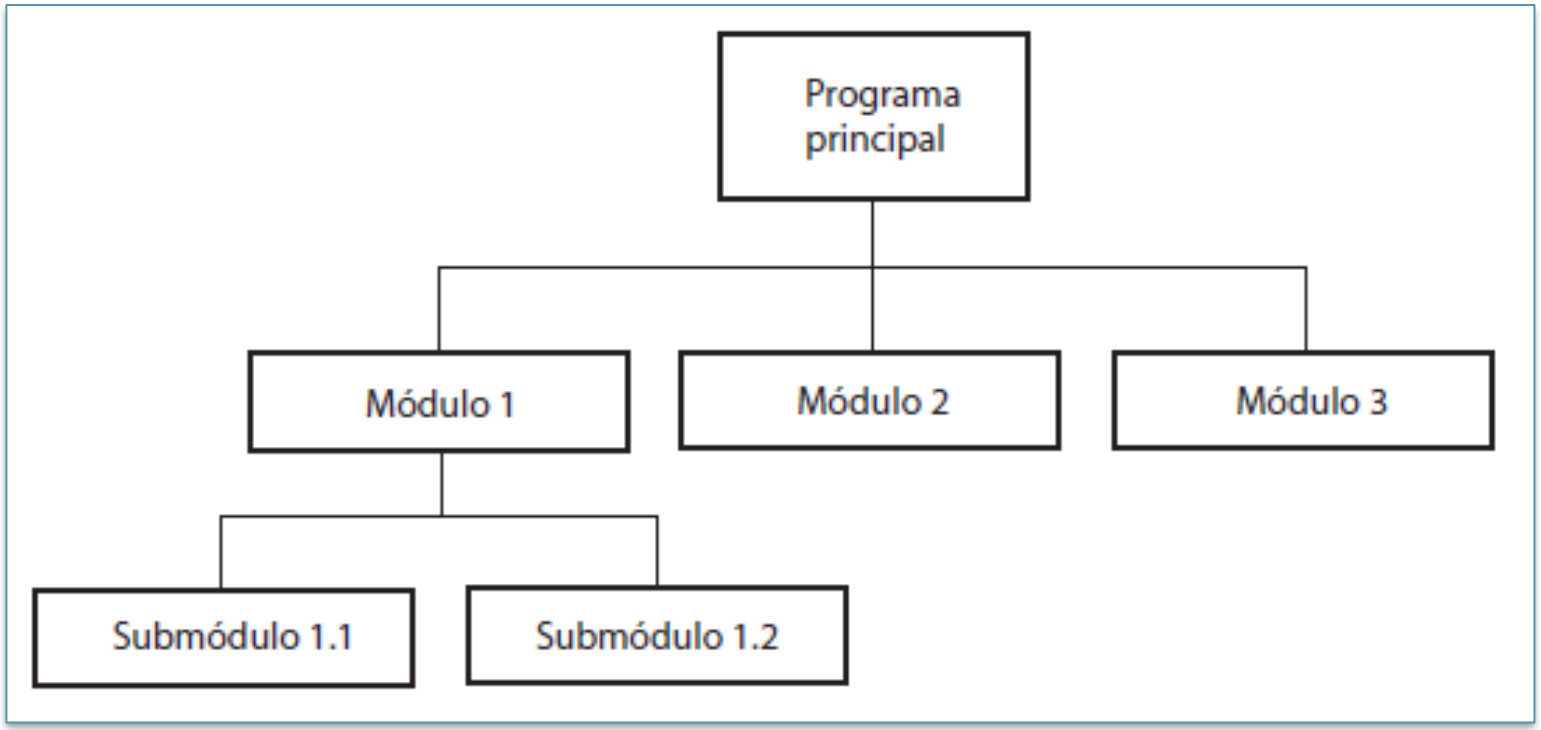
- Dichos módulos se escriben solamente una vez, pero pueden ser llamados en diferentes puntos del programa principal o de cualquier otro módulo.

# Programación Modular

- **Definición**:
- La programación modular es:
  - Una técnica que consiste en dividir un programa en tareas y dar origen a la creación de pequeños programas llamados *módulos, subprogramas o subrutinas con la finalidad de simplificar* la elaboración y mantenimiento del mismo, donde cada módulo se codifica y se procesa de manera independiente, sin importar los detalles de otros módulos.
- Esto facilita la localización de un error cuando se produce.

# Programación Modular

- Este tipo de programación es uno de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa.



# Ventajas de la programación modular

1. Facilita el diseño descendente.
2. Se simplifica un algoritmo complejo.
3. Cada módulo se puede elaborar de manera independiente, lo que permite trabajar simultáneamente a varios programadores y con ello disminuir el tiempo de elaboración del algoritmo.
4. La depuración se lleva a cabo en cada módulo.
5. El mantenimiento es más sencillo.
6. Creación de bibliotecas con módulos específicos (se pueden utilizar en otros programas).

# Programa principal

- Es donde se inicia la ejecución del programa. El resto de los Módulos se llaman desde PRINCIPAL *y desde el interior de otros Módulos.*

- El papel más importante del *programa principal, es coordinar a los otros Módulos, mediante llamadas* o invocaciones.

# Módulo (Método o Función)

- Es un subprograma(SubAlgoritmo) que realiza una tarea específica que puede o no recibir valores (parámetros).
- El uso de Módulos es una práctica común y recomendable ya que permite dividir el código, simplificando así el desarrollo y la depuración del mismo.
- Para utilizar Módulos en un programa es necesario declararlos previamente.

# Ámbito de las variables

- ***Variable local.*** *Variable declarada en un determinado módulo, sólo se encuentra disponible durante el funcionamiento* del mismo, es decir está en memoria cuando dicho módulo está activo.

- ***Variable global.*** *Variable declarada fuera de cualquier módulo y que puede ser utilizada por los módulos* que se encuentran después de dicha declaración; por lo tanto se podrá utilizar en todo el programa.

# Estructura de un Algoritmo Modular

```
//Objetivo: Descripción breve, clara y precisa del problema
//Programador: Nombre completo del programador
//Fecha: Fecha de elaboración del algoritmo
INICIO
    Definición de Constantes  y  Variables Globales
    PRINCIPAL ()
    INICIO
        Definición de Constantes  y Variables Locales
        . . .
        ModuloA()
        ModuloB()                    } Llamada o invocación de Módulos
        . . .
        Instrucciones
    FIN
    SINVALOR ModuloA ()
    INICIO
        Definición de Constantes  y Variables Locales
        Instruccion1
        Instruccion2
    FIN
    SINVALOR ModuloB ()
    INICIO
        Definición de Constantes  y Variables Locales
        Instruccion1
        Instruccion2
    FIN
FIN
```

# Cómo se ejecuta un programa que contiene Módulos

1. Siempre inicia con la primera instrucción que encuentra en el programa PRINCIPAL y

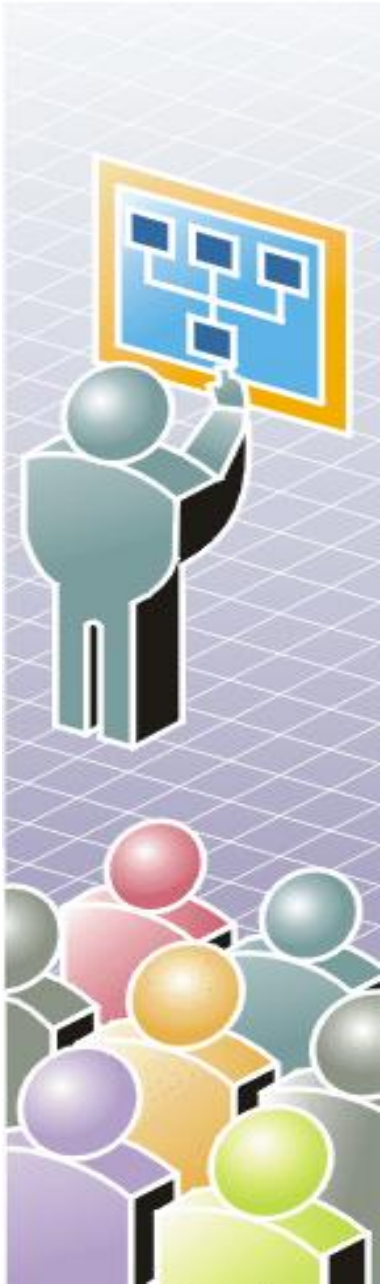
2. Continúa ejecutándolo según la estructura de control (secuencial, selectiva o repetitiva) del programa;

3. Cuando encuentra la llamada a un Módulo se traslada y ejecuta el cuerpo del Módulo desde la primera instrucción hasta encontrar el fin de la misma;

4. Cuando esto sucede regresa a la siguiente instrucción que llamó al Módulo y continúa de la misma manera.

# Problema #1

- Descripción:
  - Elaborar un pseudocódigo que solicite al usuario dos números enteros, luego los sume e imprima en pantalla el resultado.



# Análisis Modular del Problema

- Listado de Módulos

| | |
|---|---|
| Variables y constantes globales | • Datos No conocidos<br>  • Numero1<br>  • Numero2<br>• Datos de Salida<br>  • Suma |
| Lectura de Números | • Solicitar al usuario, los valores para cada número |
| Suma de Números | • Calcular la suma de los números indicados por el usuario |
| Impresión de la Suma | • Imprimir en pantalla el resultado de la suma |

# Algoritmo Modular

```
//Objetivo: Suma de dos números enteros
//Programador: X
//Fecha: ___de diciembre de 20

INICIO
    //Definición de Constantes  y  Variables Globales
    ENTERO Suma,Numero1,Numero2

    PRINCIPAL ()
    INICIO
        LeerNumeros()
        SumarNumeros()
        ImprimirSuma()
    FIN

    SINVALOR LeerNumeros()
    INICIO
         IMPRIMIR "Proporciona el valor del Número 1: "
         LEER Numero1
         IMPRIMIR "Proporciona el valor del Número 2: "
         LEER Numero2
    FIN

    SINVALOR  SumarNumeros()
    INICIO
          Suma=Numero1 + Numero2
    FIN

    SINVALOR ImprimirSuma()
    INICIO
          IMPRIMIR "La suma es: ",Suma
    FIN

FIN
```

# Plan de Prueba

| Número1 | Numero2 | Resultado |
|---|---|---|
| 4 | 8 | 12 |
| | | |
| | | |
| | | |

# *Actividad: Verificación*


MEMORIA RAM

***Pantalla de la PC***


*Estructura de 32 bits*

# Segunda propuesta

# Algoritmo Modular

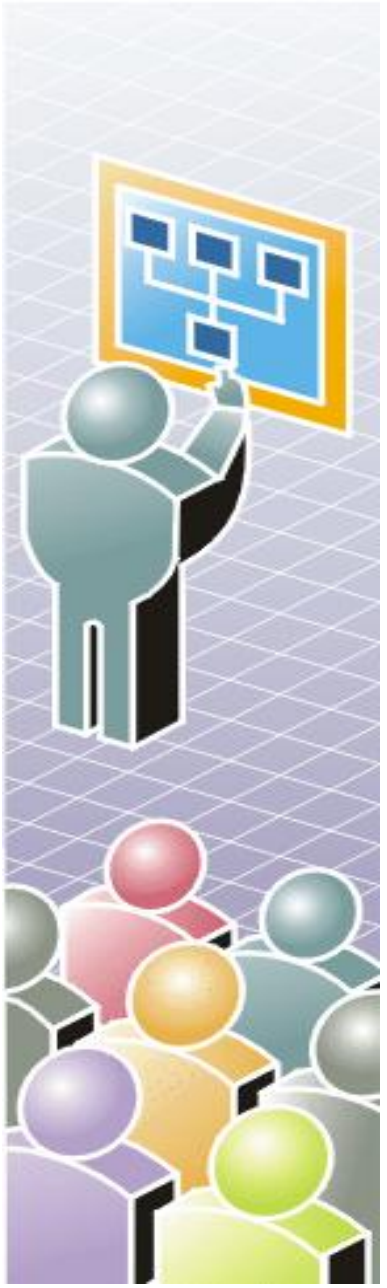
```
//Objetivo: Suma de dos números enteros
//Programador: X
//Fecha: ___de diciembre de 20

INICIO
    //Definición de Constantes  y  Variables Globales

    PRINCIPAL ()
    INICIO
        //Definición de Constantes  y  Variables Locales
        ENTERO Suma,Numero1,Numero2
        Numero1=LeerNumero()
        Numero2=LeerNumero()
        Suma=SumarNumeros(Numero1,Numero2)
        ImprimirSuma(Suma)
    FIN

    ENTERO LeerNumero()
    INICIO
          ENTERO Numero
         IMPRIMIR "Proporciona el valor del Número: "
         LEER Numero
         REGRESAR Numero
    FIN

    ENTERO  SumarNumeros(ENTERO Numero1,ENTERO Numero2)
    INICIO
          ENTERO Resultado
          Resultado=Numero1 + Numero2
          REGRESAR  Resultado
    FIN

    SINVALOR ImprimirSuma(ENTERO Valor)
    INICIO
          IMPRIMIR "La suma es: ",Valor
    FIN

FIN
```

# Actividades:

- Elaborar los pseudocódigos de los siguientes problemas, utilizando modularidad:

# Problema #2

- Descripción:
  - Elaborar un pseudocódigo que calcule e imprima:
    - El área de un triangulo (Base * Altura /2), y
    - El área de un circulo ( PI * $r^2$ ) .



# Propuesta Algoritmo Modular

```
//Objetivo: Suma de dos números enteros
//Programador: X
//Fecha: ___de diciembre de 20
INICIO
    //Definición de Constantes  y  Variables Globales


    PRINCIPAL ()
    INICIO

        _____

        _____

    FIN

    SINVALOR  AreaTriangulo()
    INICIO
        //Definición de Constantes  y  Variables Locales
        ENTERO Area,_____,Altura
        //Lectura de Datos
        IMPRIMIR "Proporciona el valor de la Base: "
        LEER _____
        IMPRIMIR "Proporciona el valor de la Altura: "
        _____ Altura
        //Proceso
        Area=_____ + _____ / 2
        //Salida
         IMPRIMIR "El Área es: ",_____
    FIN
```

# Propuesta Algoritmo Modular

**SINVALOR _____()**
**INICIO**
//Definición de Constantes y Variables Locales
**_____ REAL PI =_____**
ENTERO Area,_____
//Lectura de Datos
IMPRIMIR "Proporciona el valor del Radio: "
LEER _____
//Proceso
Area=_____
//Salida
IMPRIMIR "El Área es: ",_____
**FIN**

**FIN**


**UNIVERSIDAD AUTÓNOMA DE SINALOA**

*Facultad de Informática Culiacán*

# Arreglos

# Temas

- Descripción General
- Introducción a los arreglos
- Que es un arreglo?
- Notación para arreglos
- Rango de un arreglo
- Acceso a los elementos de un arreglo
- Creación de arreglos
- Creación de arreglos de tamaño calculado

# Problema

- Elaborar un pseudocódigo, que permita leer 10 números enteros , e imprima aquellos que son superiores a la media de los números.

- Tiempo: 5 min

# Descripción general

Los Arreglos proporcionan una forma importante de agrupar datos.

Para sacar el máximo partido a esta estructura, es fundamental entender:

- Creación
- Usos

# Introducción a los arreglos

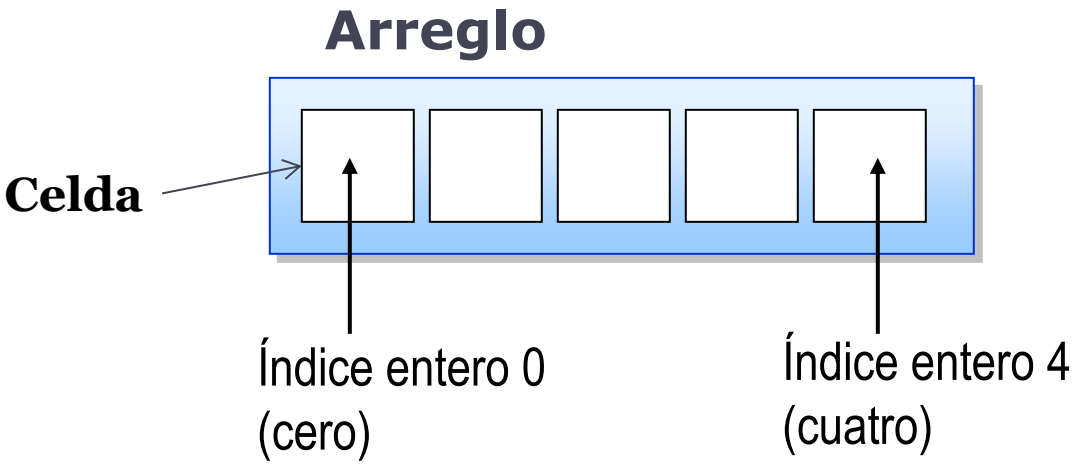
- ¿Qué es un arreglo?
- Notación para  arreglos
- Rango de un arreglo
- Acceso a los elementos de un arreglo

# ¿Qué es una arreglo?

Es un tipo de dato estructurado formado por un conjunto de datos.

Es una secuencia de elementos, donde:

- Todos sus elementos son del mismo tipo de dato (datos simples como: ENTERO, REAL, etc.),
- Se accede a elementos individuales usando índices enteros,

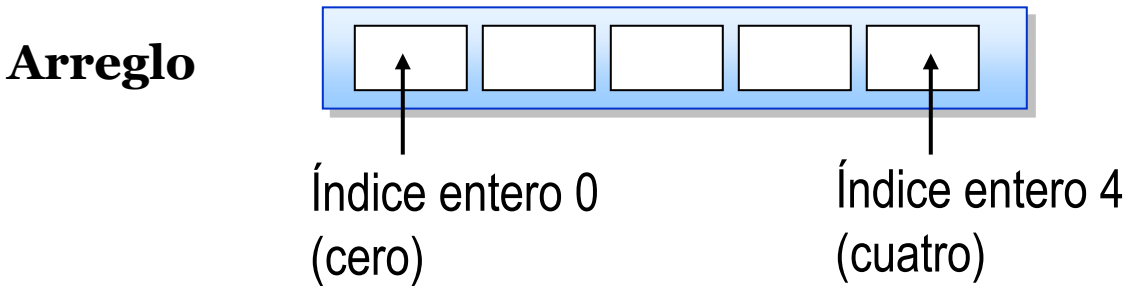


# ¿Qué es un arreglo?

Los arreglos permiten el acceso aleatorio.

Los elementos de un arreglo;

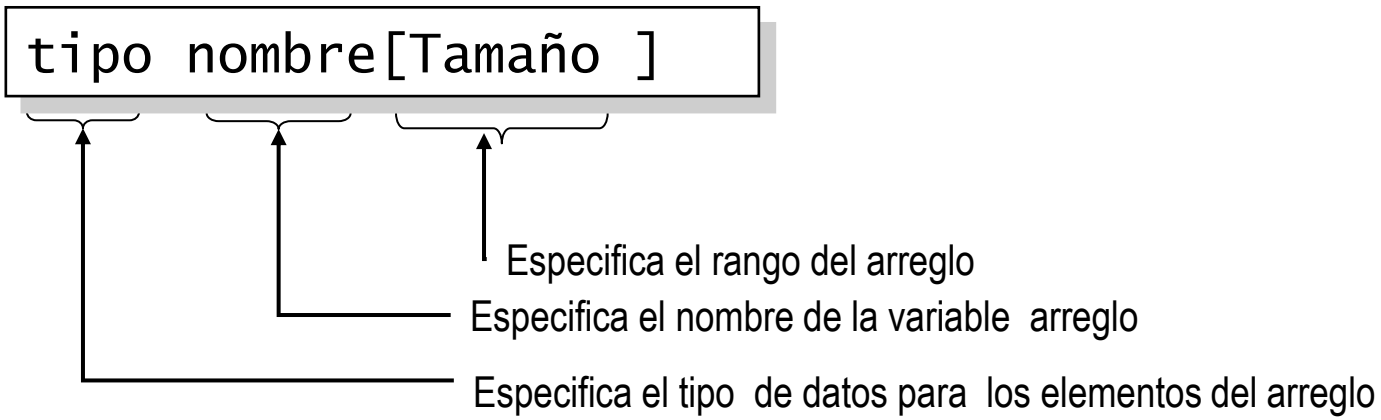
- Ocupan posiciones de memoria contiguas,
- Lo que significa que un programa puede acceder con la misma rapidez a todos los elementos de un arreglo.



# Notación para arreglos

Una variable de tipo arreglo se define especificando:

- El tipo de elementos del arreglo.
- El rango del arreglo.
- El nombre de la variable.

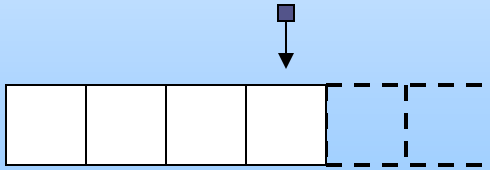


# Rango de un arreglo

- El rango se conoce también como dimensión del arreglo
- El número de índices asociados con cada elemento

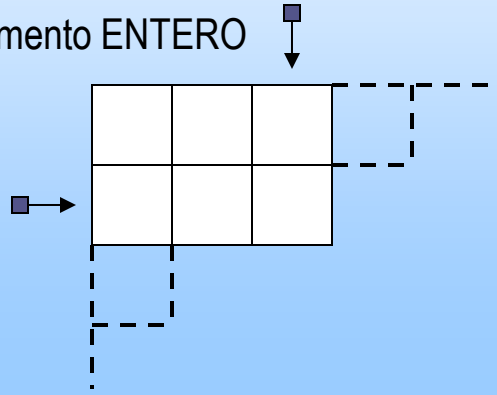
```
ENTERO Fila[ ]
```

Rango 1: Unidimensional
Un solo índice asociado con cada elemento **ENTERO**



```
ENTERO matriz[ ][ ]
```

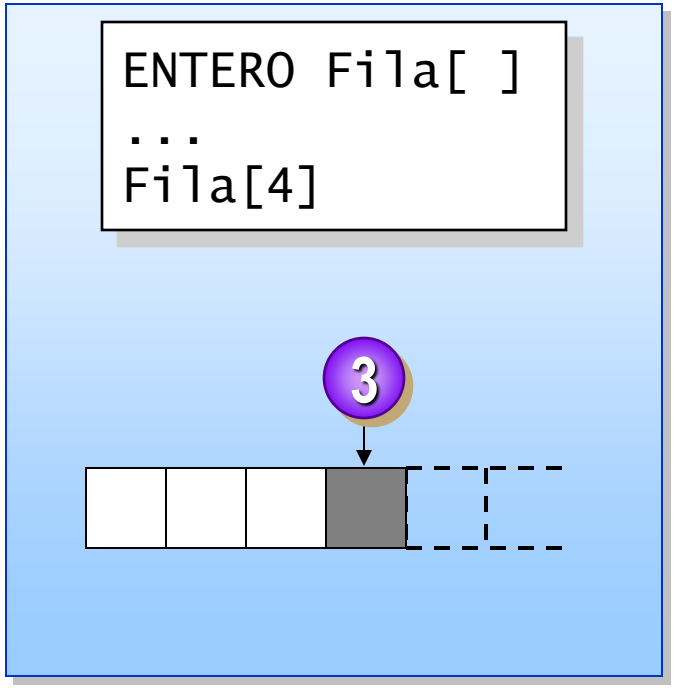
Rango 2: Bidimensional
Dos índices asociados con cada elemento ENTERO

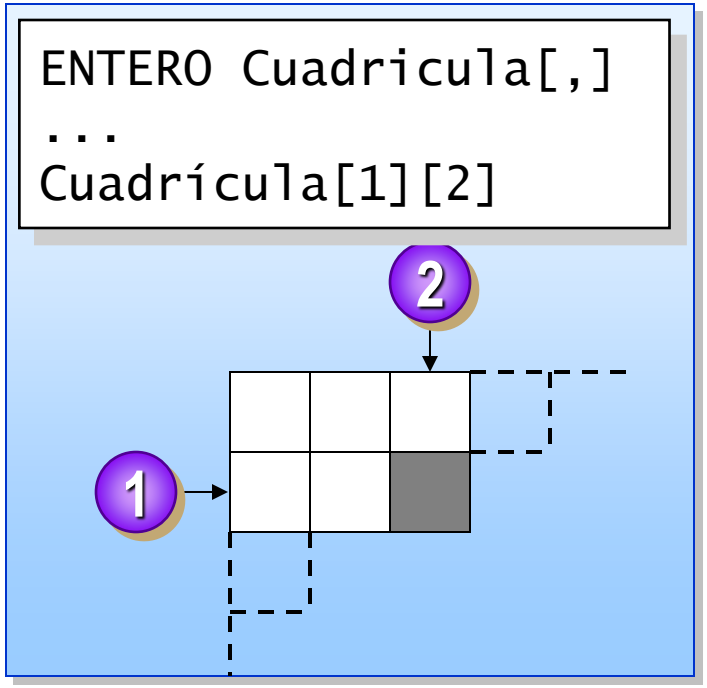


# Acceso a los elementos de un arreglo

Se indica un índice entero para cada rango

- Los índices se cuentan a partir de uno

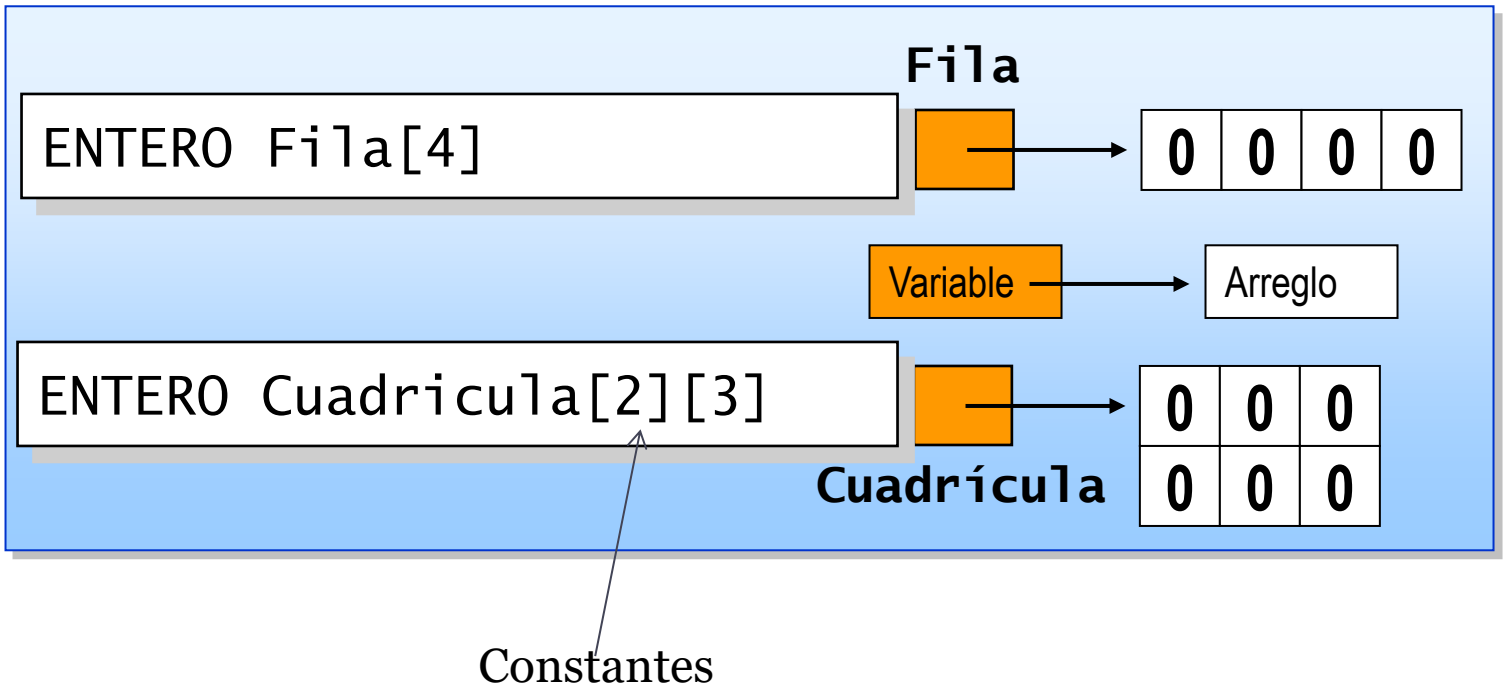


# Creación de arreglos

- Creación de un arreglo
- Creación de un arreglo de tamaño calculado

# Creación de un arreglo



# Creación de un arreglo de tamaño calculado

No es necesario que el tamaño de un arreglo sea una constante de tiempo de compilación:

- Se puede usar cualquier expresión entera válida
- El acceso a los elementos es igualmente rápido en todos los casos

# Creación de un arreglo de tamaño calculado

- Tamaño del arreglo especificado por constante entera de tiempo de compilación:

```
CONST ENTERO Tamaño=4
ENTERO Fila[Tamaño]
```

- Tamaño de arreglo especificado por valor entero de tiempo de ejecución:

```
ENTERO Tamaño
IMPRIMIR “Indique el tamaño del arreglo:”
LEER Tamaño
ENTERO Fila[Tamaño]
```

# Creación de un arreglo de tamaño calculado

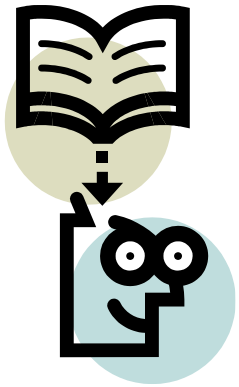
- Tamaño del arreglo especificado por constante entera de tiempo de compilación:

```
CONST ENTERO Renglones=2
CONST ENTERO Columnas=3
ENTERO Cuadricula[Renglones][Columnas]
```

- Tamaño de arreglo especificado por valor entero de tiempo de ejecución:

```
ENTERO Renglones, Columnas
IMPRIMIR “Indique el tamaño de Renglones para el arreglo:”
LEER Renglones
IMPRIMIR “Indique el tamaño de Columnas para el arreglo:”
LEER Columnas
ENTERO Cuadricula[Renglones, Columnas]
```

# Ejercicios Arreglos Unidimensionales



# Problema #1

- Elaborar un pseudocódigo, que permita:
  - Leer 10 números enteros proporcionados por el usuario, para:
    - Imprimir aquellos números que son superiores a la media de los números.

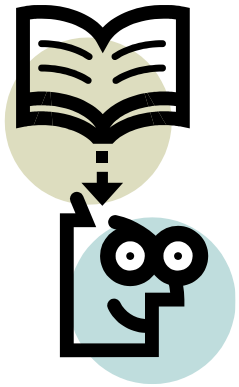
# Problema #2

- Elaborar un pseudocódigo que:
  - Permita leer valores enteros para dos arreglos unidimensionales de 10 elementos cada uno, luego:
    - Sumar el elemento uno del primer arreglo con el elemento uno del segundo arreglo y así sucesivamente hasta 10, almacenar el resultado en un tercer arreglo,
    - Imprimir el arreglo resultante.

# Problema #3

- Elaborar un pseudocódigo que:
  - Permita leer 10 valores enteros y guardarlos en un arreglo, luego
    a) Imprimirlos en forma inversa a como fueron proporcionados.
    b) Imprimir el mayor
    c) Imprimir el menor
    d) Determinar si un elemento X se encuentra en el arreglo.

# Ejercicios Arreglos Bidimensionales



# Problema #1

- Elaborar un pseudocódigo, que permita:
  - Leer y guardar datos numéricos en un arreglo bidimensional de 2 x 3, y determinar:
    - Cual es el número mayor
    - Cual es el promedio
    - Cuantos números son mayores al promedio
    - Imprimir los valores por renglón
    - Imprimir los valores por columna

# Preguntas

***F I N***