- Unidad 4

Repaso
FUNCIONES

(Capitulo 6 bibliografía)

» Ventre, Luis O.

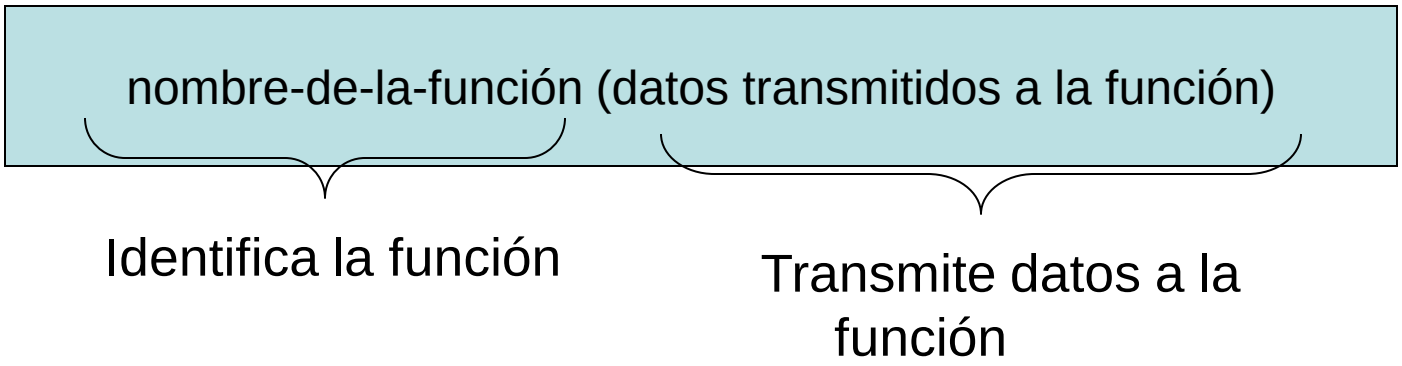
Intro

- El principal objetivo a cumplir, es entender la magnitud del problema a solucionar y poder **fraccionarlo o dividirlo** en problemas menores.

- **DIVIDE Y CONQUISTARAS**.

Declaración de Funciones y Parámetros

- **UNA FUNCION SE INVOCA o llama** dando su **NOMBRE** y transmitiéndole la lista de **parámetros** o valores a procesar, como argumentos, en los paréntesis que siguen al nombre de función.



- MAXIMO CUIDADO EN EL **ENVIO DE PARAMETROS** CUANDO SE INVOCA LA FUNCION Y EN LOS **VALORES DEVUELTOS**.

Declaración de Funciones y Parámetros

- Como usamos una FUNCION?...invocandola con su nombre:

sqrt (25);

pow (radio,2);

Identifica la funcion

Transmite datos a la funcion

- La función INVOCADA, debe ser capaz de ACEPTAR los datos que le son transmitidos.

Declaración de Funciones y Parámetros

- ANTES QUE UNA FUNCION PUEDA SER LLAMADA DEBE SER DECLARADA, **PROTOTIPO DE LA FUNCION**

tipo-de-datos-a-devolver nombre-de-función (lista de tipos de datos argumento)

- **Por ejemplo el siguiente prototipo:**

```
void encontrarMax(int, int);
```

Declara una función llamada encontrarMax, la cual recibe dos valores enteros como parámetro y no devuelve ningun valor(void).

Declaración de Funciones y Parámetros

- Definición a una función:

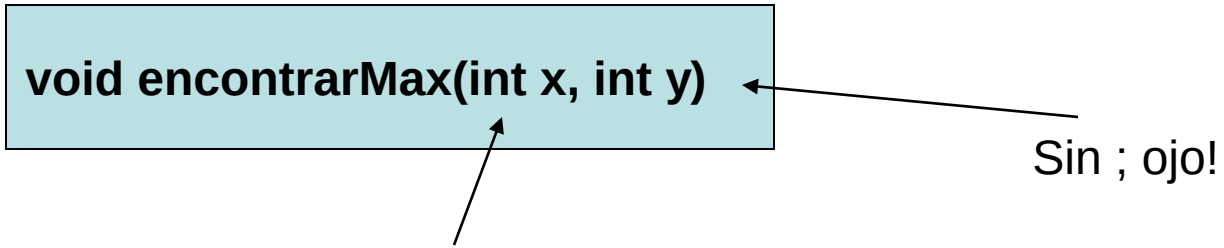
  Encabezado y el cuerpo de la función.

```
línea de encabezado de función
{
declaraciones de constante y
variable; cualquier
otra instrucción de C++
}
```

El encabezado es siempre la primer línea, contiene **tipo de valor devuelto, su nombre, y los nombres tipos y orden de datos de argumento**

Declaración de Funciones y Parámetros

El encabezado de la función encontrarMax seria:



Los nombres de argumento en el encabezado se conocen como **parámetros formales**, el **parámetro x** se usara para almacenar el primer valor transmitido y el **parámetro y** para el segundo.

El primer paso en la invocación, es buscar los valores de primernum y segundonum, y enviárselos a la función, en este momento se almacenan en los parámetros nombrados “x” e “y”.

Declaración de Funciones y Parámetros

Colocación de instrucciones:

```
directivas del preprocesador

prototipos de funcion

int main()
{
        constantes simbólicas
        declaraciones de variables

        otras instrucciones ejecutables

        return valor
}
```

Declaración de Funciones y Parámetros

Generalmente en el archivo fuente, primero se enlista MAIN, ya que es la función principal que dará una idea al lector del programa que hace el mismo antes de ver los detalles de cada función.

```
#include <iostream>
using namespace std;

void encontrarMax(int, int);  // decla

int main()
{
  int primernum, segundonum;

  cout << "\nIngrese un numero: ";
  cin  >> primernum;
  cout << "Bien! Por favor ingrese un
  cin  >> segundonum;

  encontrarMax(primernum, segundonum);

  system("PAUSE");
  return 0;
}
```

```
void encontrarMax(int x, int y)
{                       // inicio del cuerpo de la función
  int maxnum;           // declaracion de variable

  if (x >= y)           // encuentra el numero máximo
    maxnum = x;
  else
    maxnum = y;

  cout << "\nEl maximo de los dos numeros es: "
       << maxnum << endl;

  return;
}  // fin de la funcion y del cuerpo de la misma
```

Declaración de Funciones y Parámetros

Cabos de FUNCION:

Enfoque común de programación, terminar la función main y luego las demás funciones. Problema para pruebas intermedias, compilación.

Se puede hacer una función que actúe como si fuera la correcta y que acepte los datos e imprima un mensaje en pantalla.

Esta función FALSA se llama CABO.

Funciones con listas de parámetros vacías:

En su prototipo pueden llevar void o nada en el lugar de los argumentos.

Argumentos por omisión:

Estos se enlistan en el prototipo de la función, y son transmitidos a la función en forma automática cuando los argumentos correspondientes son omitidos.

Declaración de Funciones y Parámetros

Ejemplo de argumentos por omisión:

Si el prototipo de nuestra función fuera:

```
void ejemplo (int, int = 5, double = 6.78);
```

Este prototipo proporciona valores por omisión para los dos últimos argumentos. Por lo tanto las siguientes llamadas a función son validas:

```
ejemplo ( 7, 2, 9.3)  // no utiliza valores por omisión

ejemplo ( 7, 2)       // igual que llamado ejemplo( 7, 2, 6.78)

ejemplo (7)           // igual que llamado ejemplo (7, 5, 6.78)
```

Sobrecarga de funciones

Reutilización de nombres de FUNCION:

Único requisito, que el compilador pueda determinar que función utilizar basándose en los tipos de datos de los argumentos enviados a la función y no los tipos de datos del valor devuelto.

Cada función debe escribirse como una entidad separada.

El código puede tener leves cambios, aunque una buena practica de programación es que funciones sobrecargadas realicen en esencia las mismas operaciones.

Cuando las funciones son idénticas y solo cambia el tipo de dato, es mejor implementar “plantillas de función” (ver pagina 314 libro)

Sobrecarga de funciones

Ejemplo:

```
void vabs(int x)
{if(x<0)
  x= -x;
  cout<<“ el valor absoluto del numero entero es”<<x<<endl;
}

void vabs(float x)
{if(x<0)
  x= -x;
  cout<<“ el valor absoluto del numero de punto flotante es”<<x<<endl;
}
void vabs(double x)
{if(x<0)
  x= -x;
  cout<<“ el valor absoluto del numero de doble precision es”<<x<<endl;
}
```

Devolver un solo valor

Al utilizar el método de transmitir datos a una función presentado, la función llamada solo recibe **copias de los valores** contenidos en los argumentos. Esto se conoce como **“llamada por valor” o “pasaje de datos por valor”.**

De esta forma, la función invocada puede procesar los datos y devolver **un y solo un valor legitimo** a la función que la invoco.

Para que esto no produzca errores indeseados debe manejarse con cautela y de manera correcta:

Desde el punto de vista de la función llamada esta debe devolver:

* El tipo de dato del valor devuelto (**encabezado 1º línea de la fx**)
* El valor real que se devuelve

Devolver un solo valor

Para que la función retorne el valor solo es necesario colocar la instrucción:

return expresion;

Debe cuidarse para evitar errores indeseados que el **tipo de dato devuelto por la función declarado en el encabezado** y el **tipo utilizado en la instrucción de return coincidan!.**

Desde el punto de vista del receptor, la función que llama debe:

* Ser alertada del tipo de valor a esperar (**prototipo de función**)
* Usar de manera apropiada el valor.

La variable utilizada para almacenar el valor devuelto debe ser del mismo tipo de dato.

```
max = encontrarMax(num1, num2);
```

```
cout<<encotnrarMax(num1, num2);
```

Devolver un solo valor

```
#include <iostream>
using namespace std;

int encontrarMax(int, int);  // prototipo de la funcion

int main()
{
  int primernum, segundonum, max;

  cout << "\nIngrese un numero: ";
  cin  >> primernum;
  cout << "Bien! Por favor ingrese un segundo numero: ";
  cin  >> segundonum;

  // la funcion es llamada aqui
  max = encontrarMax(primernum, segundonum);

  cout << "\nEl maximo de los dos numeros es: " << max << endl;

  system("PAUSE");
  return 0;
}
```

Alerta a main y demás fx sobre el valor devuelto

Asignación a la variable max del valor devuelto por la función

Devolver un solo valor

```
int encontrarMax(int x, int y)
{                       // inicio del cuerpo de la función
  int maxnum;           // declaracion de variable

  if (x >= y)           // encuentra el numero máximo
    maxnum = x;
  else
    maxnum = y;

  return maxnum;        // instrucción return
}
```

Encabezado declara que valor será devuelto

Ver en el libro ejemplo similar programa 6.6

Unidad 5
Tipos de datos Arreglos
(Capitulo 11 bibliografía)

» Ventre, Luis O.

Arreglos Unidimensionales

- Contexto: Todas las variables vistas hasta ahora, sin discriminar el tipo de dato que puedan almacenar, **todas** solo pueden almacenar un **UNICO** valor.

- En numerosas oportunidades es necesario manejar **conjuntos de valores de un mismo tipo**. Por ejemplo una lista de notas de parciales, una lista de valores de voltajes, etc.

- *Una **lista simple** que contiene elementos **individuales del mismo tipo** de datos se llama **arreglo unidimensional**.*

- *A continuación se vera como declarar, inicializar, almacenar y usar los arreglos unidimensionales.*

Arreglos Unidimensionales

- Lista de valores relacionadas con **“el mismo tipo de datos”** que se almacena bajo un **“nombre de grupo único”**.

- En la declaración de un arreglo de dimensión única es necesario indicar:
  - ***El tipo de datos del conjunto***
  - ***El nombre del arreglo o del grupo.***
  - ***La cantidad de elementos del grupo entre corchetes [ ]***

Sintaxis de declaración:

tipo-de-datos **nombreArreglo** [cantidad-elementos]

Arreglos Unidimensionales

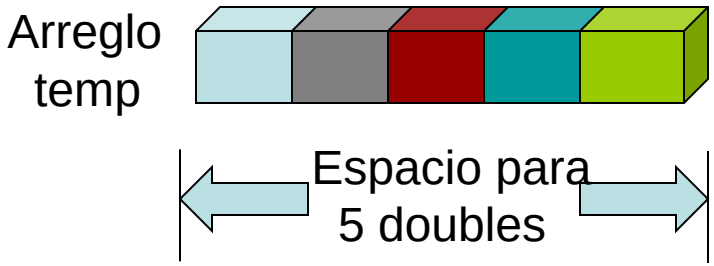
- Ej. de declaración de un arreglo de temperaturas llamado temp con 5 elementos:

```
double temp [5];
```

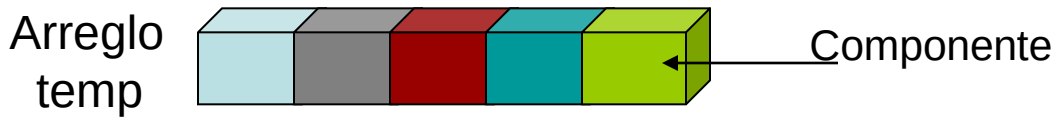
- Una buena práctica de programación es definir la **cantidad de elementos** como una **constante** antes de su declaración:
- Ej:

```
const int CANT = 5;

double temp[CANT];
```



Arreglos Unidimensionales



- **Cada elemento** del arreglo se llama *componente*. Estos se almacenan de manera secuencial en la memoria. Este **almacenamiento contiguo** es una ventaja para encontrar con facilidad cualquier elemento de la lista.

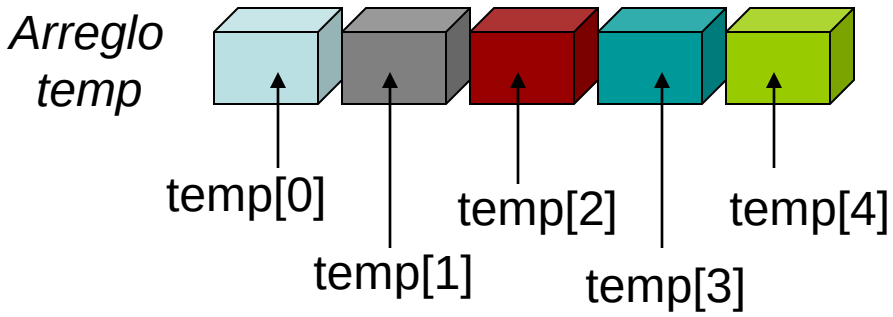
- ***Con esta característica dado el nombre del arreglo y su posición es posible acceder a cualquier elemento del arreglo.***

- *Esta posición se llama* ***valor índice o subíndice****.*

- ***Todo arreglo comienza en la posición 0.***

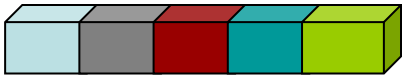
Arreglos Unidimensionales

- Continuando con el ejemplo anterior, los subíndices de los componentes serán:



- *Una vez declarado el array,* ***cada componente es una variable indexada*** *ya que debe darse su* ***nombre y su subíndice*** *para hacer referencia a ese elemento.*

Arreglos Unidimensionales



- Ejemplos de utilización de los elementos del arreglo:

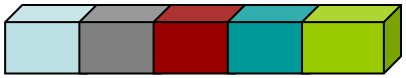
```
const int numero=5;
.
double temp[numero];
.
.
.
temp[0] = 95.75;
temp[1] = temp[0] – 11.0;
temp[4]= (temp[1] + temp[2] – 3.1) / 2.2:
```

- El subíndice entre corchetes también puede ser una **expresión que evalúe a un numero entero**. Siempre dentro del rango de valores.

```
temp[i]                    temp[2*i]
temp[j-i]
```

Arreglos Unidimensionales

Arreglo temp



- ***De esta forma y con la gran ventaja de tener subíndices enteros podemos recorrer un arreglo con un ciclo for; utilizando la variable "i" como contador del ciclo y como subíndice:***

- Ej. Se desea guardar en una variable la suma de todos los elementos del arreglo temp.

```
suma = temp[0] + temp[1] + temp[2] + temp[3] + temp[4]
```

```
for (i=0 ; i<5 ; i++)
        suma = suma + temp[i];
```

Arreglos Unidimensionales

- ***Otro ejemplo:***

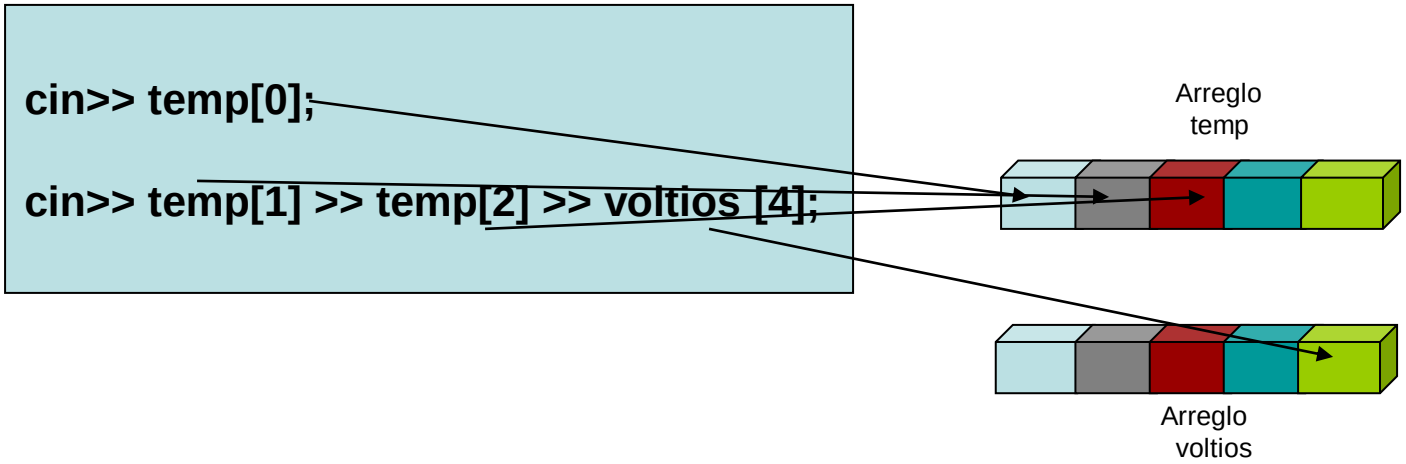
- Ej. Se desea recorrer un arreglo de mil valores y encontrar el mayor valor:

```
cons int numels = 1000;
.
.
maximo=voltios[0];

for (i=0 ; i<numels ; i++)
     if ( voltios[i] >  maximo)
        maximo = voltios[i];
```

Entrada y Salida de valores del arreglo

- ***A los objetos del arreglo se les puede asignar valores de manera interactiva usando cin.***
- ***Ej:***



Entrada y Salida de valores del arreglo

- ***Si generamos un arreglo temp, de 10 elementos…..como ingresaríamos valores por teclado?***

```
cin>> temp[0];

cin>> temp[1];

cin>> temp[2];

cin>> temp[3];

cin>> temp[4];

cin>> temp[5];

cin>> temp[6];

cin>> temp[7];

cin>> temp[8];

cin>> temp[9];
```

Los sub indices varían….

…y si el arreglo tuviera 1000 elementos?

….alguna forma de simplificarlo?

Entrada y Salida de valores del arreglo

- *De manera análoga, puede utilizarse un ciclo for para la introducción interactiva de todos los datos del arreglo:*

```
const int numels=1000;

for ( i=0; i<numels; i++)
  {
    cout<<“Introduzca el elemento “<<i;
    cin>>temp[i];
  }
```

Entrada y Salida de valores del arreglo

- ***De igual manera puede utilizarse un <u>ciclo para imprimir en pantalla todos los valores del arreglo:</u>***

```
for ( i=0; i<numels; i++)
{
   cout<<“El valor del elemento “<<i;
   cout>>temp[i];
}
```

- *<u>Advertencia:</u> <u>C++ no tiene **verificación de limites**</u>. Esto implica que si el arreglo definido tiene 5 elementos y se accede a un elemento fuera del limite el **compilador no advertirá el error;** se accederá a esa posición de memoria y pueden producirse errores. A veces produce que el programa se caiga pero no siempre*.

Ejemplo

```
#include <iostream>
using namespace std;

int main()
{
  const int MAXTEMPS = 5;

  int i, temp[MAXTEMPS];

  for (i = 0; i < MAXTEMPS; i++)
  {
    cout << "Ingrese la temperatura: ";
    cin  >> temp[i];
  }

  cout << endl;

  for (i = 0; i < MAXTEMPS; i++)     // Imprime las temperaturas
    cout << "temperatura " << i << " es " << temp[i] << endl;

  system("PAUSE");
  return 0;
}
```

```
Ingrese la temperatura: 12
Ingrese la temperatura: 13
Ingrese la temperatura: 14
Ingrese la temperatura: 15
Ingrese la temperatura: 16

temperatura 0 es 12
temperatura 1 es 13
temperatura 2 es 14
temperatura 3 es 15
temperatura 4 es 16
Presione una tecla para continuar . . .
```

Inicialización de arreglos

- ***Al igual que las variables vistas, los arreglos pueden inicializarse cuando son declarados, la DIFERENCIA entre ambas declaraciones radica en que los valores del arreglo deben ir entre llaves { }***

```
int temp[5] = { 98, 87, 92, 79, 85};

char codigos[7] = { 'm', 'u', 'e', 's', 't', 'r', 'a'};
```

- ***La inicialización de valores puede extenderse a múltiples líneas:***

```
int voltios [9] = { 98, 87, 92,
                    79, 85, 66,
                    94, 55, 67};
```

- ***Si el numero de valores inicializados es menor al total de elementos, los restantes serán inicializados a "0".***

Inicialización de arreglos

- **En la inicialización puede omitirse el tamaño del arreglo si esta perfectamente definida su cantidad de elementos; los dos ejemplos a continuación son iguales:**

```
int temp[5] = { 98, 87, 92, 79, 85};

int temp[ ] = { 98, 87, 92, 79, 85};
```

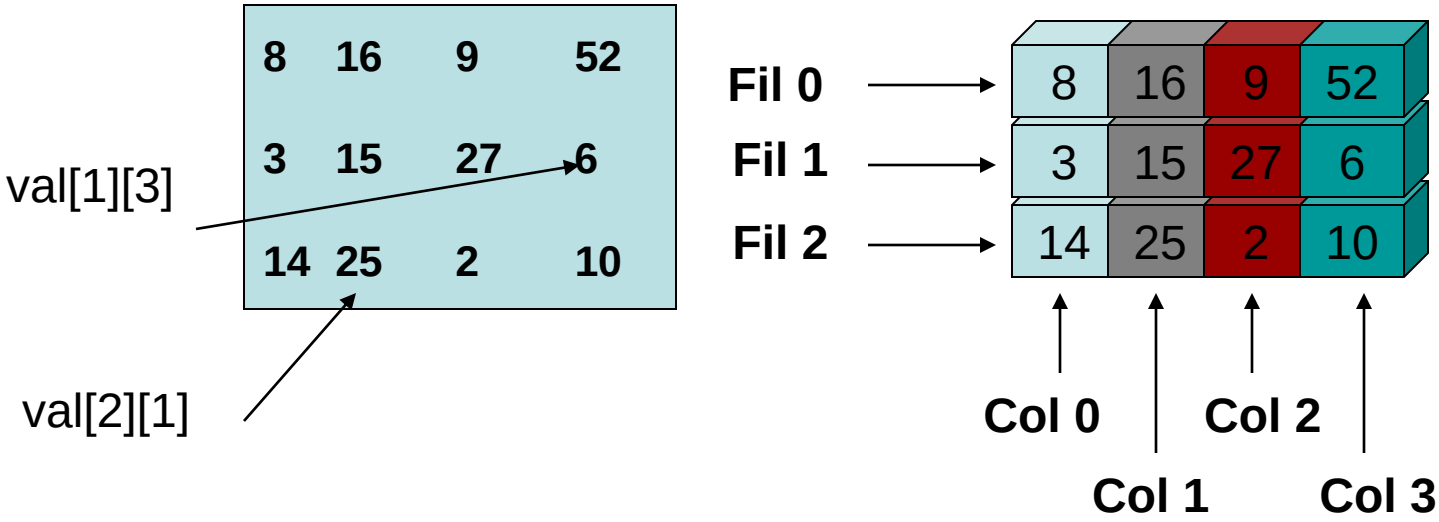
- **Otro uso interesante es la declaración de un arreglo de caracteres haciendo uso de las “ “.**

```
char codigos[ ] = “muestra”; // sin llaves ni comas
```

- ***Advertencia:*** *Esta declaración crea un arreglo con 8 caracteres. El ultimo es la secuencia de escape “carácter nulo” \0. Este se agrega de manera automática a todas las cadenas e “indica el fin”.*

Arreglos BIDIMENSIONALES

- **Un arreglo bidimensional, a veces llamado tabla, es un arreglo de elementos que posee filas y columnas.** Por ejemplo un arreglo bidimensional de números enteros se observa a continuación**:**

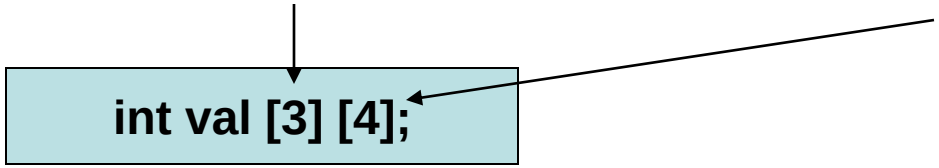


- **Para reservar los lugares de almacenamiento en su declaración deben incluirse el numero de filas y el numero de columnas**

```
int val [3] [4];
```

Arreglos BIDIMENSIONALES

- Es importante recordar que en un arreglo bidimensional **el primer subíndice hace referencia a la FILA y el segundo a la COLUMNA**



- **Al igual que los arreglos unidimensionales se puede hacer uso de cualquier elemento del arreglo:**

```
val[0][0]=62.54;

nuevonum= val[0][0] + val[0][1] + 4*(val[1][0] – 5);
```

Inicialización de Arreglos BIDIMENSIONALES

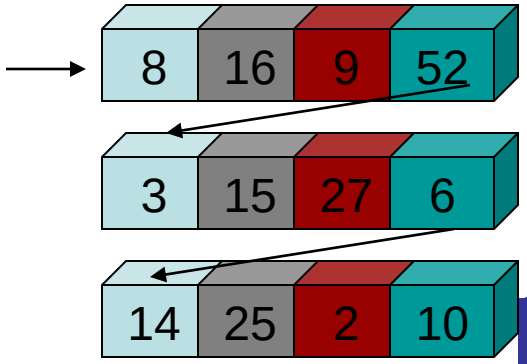
- De manera análoga con los arreglos unidimensionales puede inicializarse un arreglo bidimensional **enlistando los valores entre llaves y separándolos entre comas**. Pueden usarse llaves para las filas.

```
int val[3][4] = { {8,16,9,52},
                  {3,15,27,6},
                  {14,25,2,10} }
```

```
int val[3][4] = { 8,16,9,52,
                  3,15,27,6,
                  14,25,2,10}
```

- **El compilador asigna valores iniciando en val[0][0], y recorre por filas por lo tanto podría inicializarse con valores corridos pero no seria una ilustración clara.**

```
int val[3][4] = { 8,16,9,52,3,15,27,6,14,25,2,10 }
```

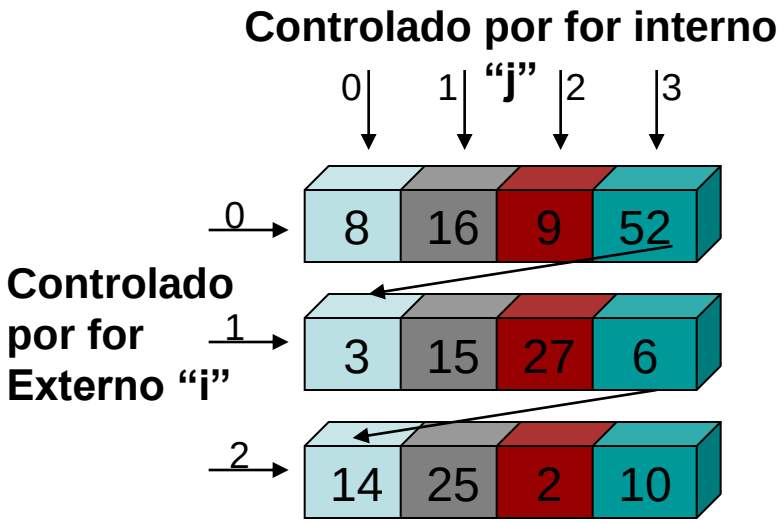


Entrada y salida de valores Arreglos BIDIMENSIONALES

- Para asignar valores a un arreglo bidimensional y/o imprimir sus valores es necesario implementar dos ciclos **for anidados**.
- Un **ciclo for** exterior recorrerá **las filas** o renglones.
- Un **ciclo for** interior recorrerá **las columnas**.
- Por cada ciclo exterior se recorrerán todas las columnas interiores.

```
for ( i=0; i<FILAS; i++) // ciclo externo
{
    for( j=0; j<COLUMNAS; j++) //ciclo interno
    {
      cout<<“Ingrese el elemento temp“<<i<<j;
      cin>>temp[i][j];
}
```

```
for ( i=0; i<FILAS; i++) // recorre filas
{   cout<<endl;
    for( j=0; j<COLUMNAS; j++) // recorre colum.
    {
      cout<<“El elemento temp“<<i<<j<<“es: “;
      cout>>temp[i][j];
}
```



Arreglos BIDIMENSIONALES

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{

  const int NUMRENG = 3;
  const int NUMCOLS = 4;
  int i, j;
  int val[NUMRENG][NUMCOLS] = {8,16,9,52,3,15,27,6,14,25,2,10};

  cout << "\nMuestra los valores del arreglo val por elemento"
       << endl << setw(4) << val[0][0] << setw(4) << val[0][1]
       << setw(4) << val[0][2] << setw(4) << val[0][3]
       << endl << setw(4) << val[1][0] << setw(4) << val[1][1]
       << setw(4) << val[1][2] << setw(4) << val[1][3]
       << endl << setw(4) << val[2][0] << setw(4) << val[2][1]
       << setw(4) << val[2][2] << setw(4) << val[2][3];

  cout << "\n\nMuestra los valores del arreglo val con for anidados";

  for (i = 0; i < NUMRENG; i++)
  {
    cout << endl;       // imprime una nue
    for (j = 0; j < NUMCOLS; j++)
      cout << setw(4) <<  val[i][j];
  }
  cout << endl;
  system("PAUSE");
  return 0;
```

```
Muestra los valores del arreglo val por elemento
   8  16   9  52
   3  15  27   6
  14  25   2  10

Muestra los valores del arreglo val con for anidados
   8  16   9  52
   3  15  27   6
  14  25   2  10
Presione una tecla para continuar . . .
```