Informática I

Claudio Paz

claudiojpaz@gmail.com

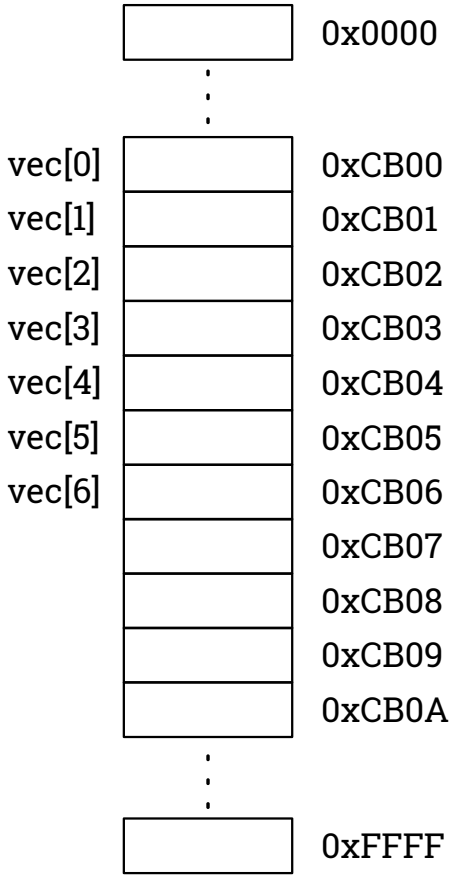
Junio 2019

Unidad 6

Arreglos en lenguaje C

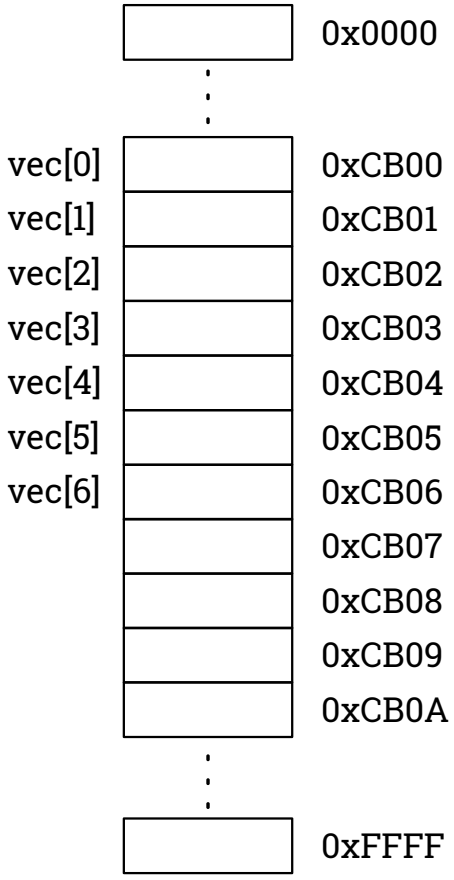
Arreglos

Arreglos

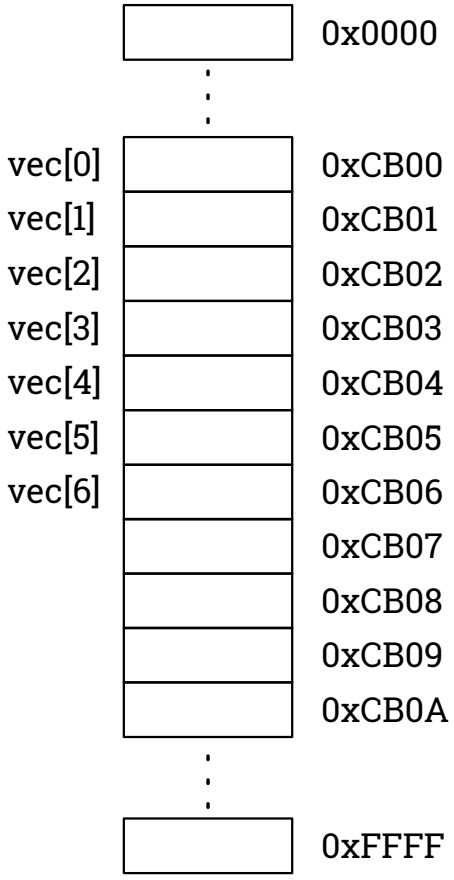

0x0000
vec[0]
0xCB00
vec[1]
0xCB01
vec[2]
0xCB02
vec[3]
0xCB03
vec[4]
0xCB04
vec[5]
0xCB05
vec[6]
0xCB06
0xCB07
0xCB08
0xCB09
0xCB0A
0xFFFF

Arreglos

Los arreglos son un conjunto de posiciones de memoria contiguos, en donde se pueden almacenar valores del mismo tipo.

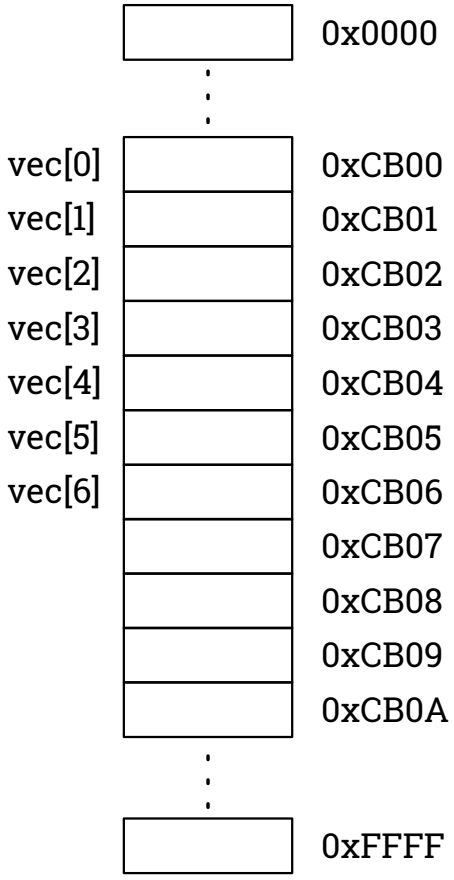


Arreglos

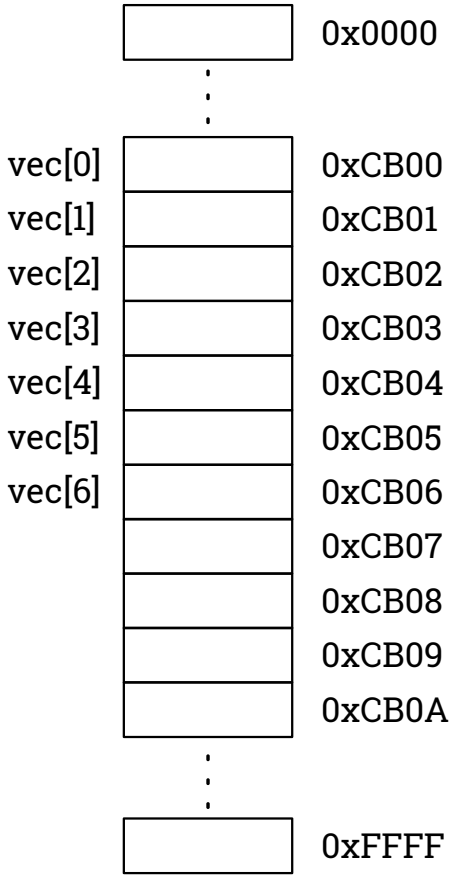

0x0000
vec[0]
0xCB00
vec[1]
0xCB01
vec[2]
0xCB02
vec[3]
0xCB03
vec[4]
0xCB04
vec[5]
0xCB05
vec[6]
0xCB06
0xCB07
0xCB08
0xCB09
0xCB0A
0xFFFF

Arreglos

Tienen un nombre que debe respetar las características de los identificadores.

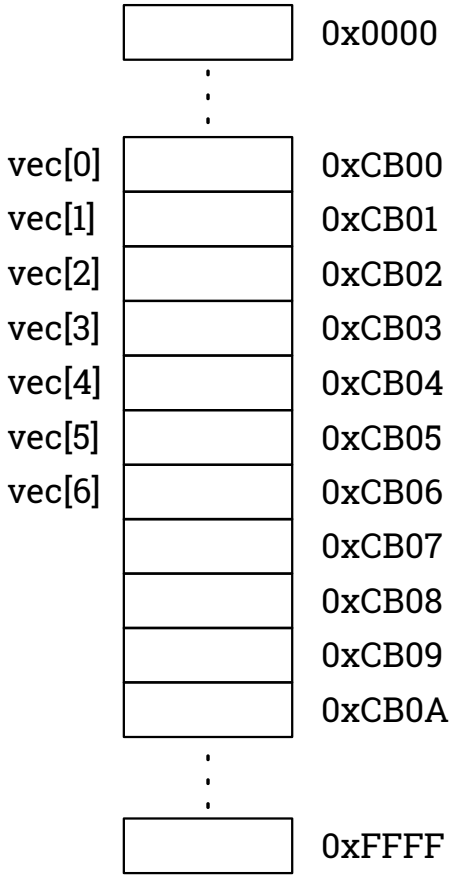


Arreglos


0x0000
vec[0]
0xCB00
vec[1]
0xCB01
vec[2]
0xCB02
vec[3]
0xCB03
vec[4]
0xCB04
vec[5]
0xCB05
vec[6]
0xCB06
0xCB07
0xCB08
0xCB09
0xCB0A
0xFFFF

Arreglos

Se puede acceder a cada valor mediante el nombre y el índice entre corchetes.

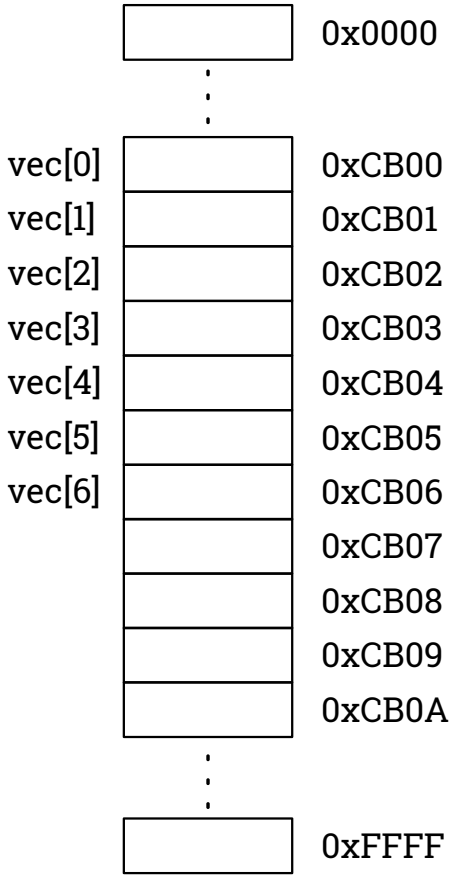


Arreglos

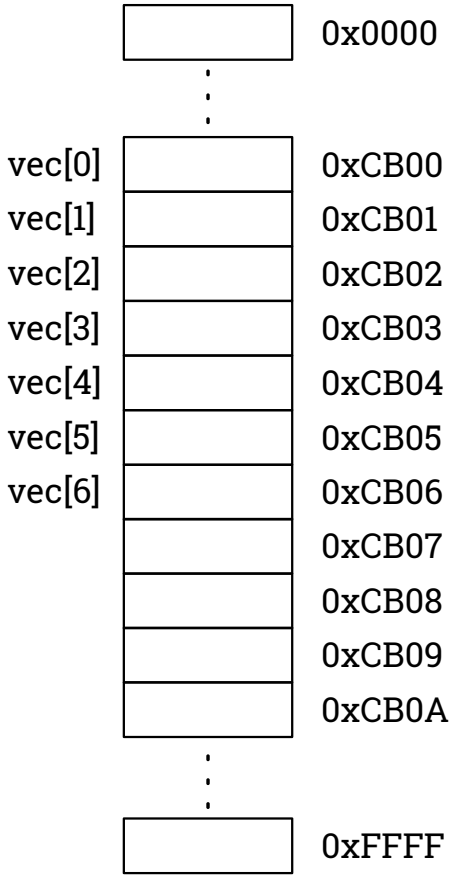
Se puede acceder a cada valor mediante el nombre y el índice entre corchetes.

Ej.

```
vec[3] = 15;
```

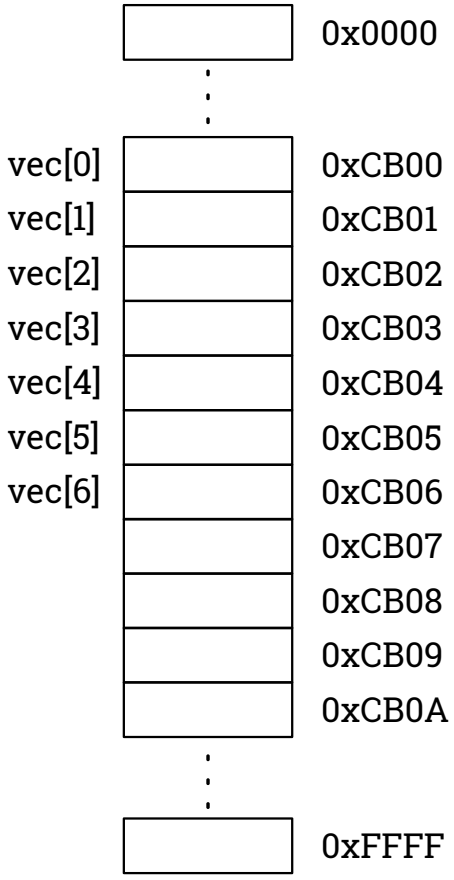


Arreglos


0x0000
vec[0]
0xCB00
vec[1]
0xCB01
vec[2]
0xCB02
vec[3]
0xCB03
vec[4]
0xCB04
vec[5]
0xCB05
vec[6]
0xCB06
0xCB07
0xCB08
0xCB09
0xCB0A
0xFFFF

Arreglos

El índice puede ser una variable o cualquier expresión

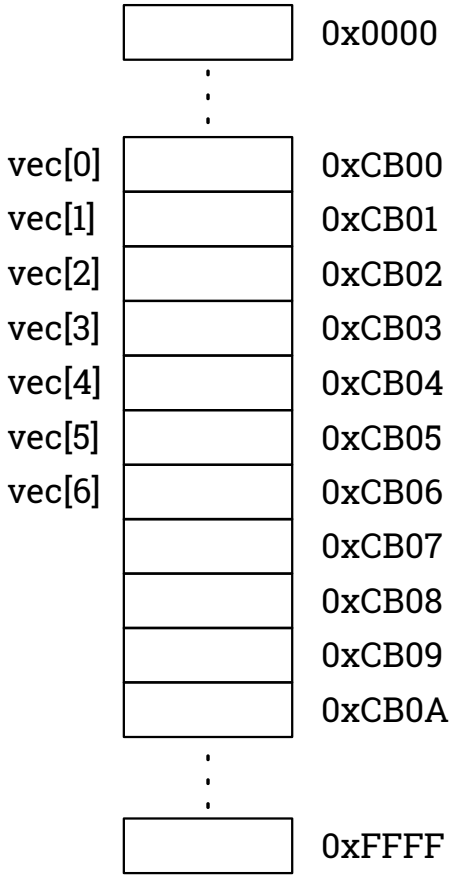


Arreglos

El índice puede ser una variable o cualquier expresión

Ej.

```
for (int i = 0; i < 7; i++)
  printf("%d\n", vec[i]);
```



Arreglos

Arreglos

Definición

Arreglos

Definición

```
tipo identificador[tamaño];
```

Arreglos

Definición

```
tipo identificador[tamaño];
```

`tipo` se refiere a cualquier tipo de dato (`int`, `char`, `float`, etc.).

Arreglos

Definición

```
tipo identificador[tamaño];
```

`tipo` se refiere a cualquier tipo de dato (`int`, `char`, `float`, etc.).

`identificador` es el nombre que se usará para acceder a los elementos del arreglo. Debe cumplir los requisitos de cualquier identificador.

Arreglos
Definición

```
tipo identificador[tamaño];
```

`tipo` se refiere a cualquier tipo de dato (`int`, `char`, `float`, etc.).

`identificador` es el nombre que se usará para acceder a los elementos del arreglo. Debe cumplir los requisitos de cualquier identificador.

`tamaño` es una **constante** que indica cuantos elementos tendrá el arreglo.

Arreglos

Definición

Arreglos

Definición

```
int vec[100];
```

Arreglos
Definición

```
int vec[100];
```

Esto define un arreglo llamado `vec` de 100 enteros.

Arreglos
Definición

```
int vec[100];
```

Esto define un arreglo llamado `vec` de 100 enteros.

El estándar c99 determina que el tamaño máximo debe ser al menos 65536 elementos...

Arreglos
Definición

```
int vec[100];
```

Esto define un arreglo llamado `vec` de 100 enteros.

El estándar c99 determina que el tamaño máximo debe ser al menos 65536 elementos...

...pero el verdadero tamaño máximo depende de diversos factores relacionados a la memoria del programa (se verán más adelante en la carrera)

Arreglos

Arreglos

```c
#include <stdio.h>

int main (void) {
  int i;
  int vec[5];

  for ( i = 0; i < 5; i++ )
    vec[i] = 0;

  for ( i = 0; i < 5; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Arreglos

```c
#include <stdio.h>

int main (void) {
  int i;
  int vec[5];

  for ( i = 0; i < 5; i++ )
    vec[i] = 0;

  for ( i = 0; i < 5; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Si eventualmente se necesita un arreglo más grande, hay que cambiar el tamaño del arreglo...

Arreglos

```c
#include <stdio.h>

int main (void) {
  int i;
  int vec[5];

  for ( i = 0; i < 5; i++ )
    vec[i] = 0;

  for ( i = 0; i < 5; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Si eventualmente se necesita un arreglo más grande, hay que cambiar el tamaño del arreglo...

Arreglos

```c
#include <stdio.h>

int main (void) {
  int i;
  int vec[10];

  for ( i = 0; i < 5; i++ )
    vec[i] = 0;

  for ( i = 0; i < 5; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Si eventualmente se necesita un arreglo más grande, hay que cambiar el tamaño del arreglo...

Arreglos

```c
#include <stdio.h>

int main (void) {
  int i;
  int vec[10];

  for ( i = 0; i < 5; i++ )
    vec[i] = 0;

  for ( i = 0; i < 5; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Si eventualmente se necesita un arreglo más grande, hay que cambiar el tamaño del arreglo...

...y el control de las sentencias repetitivas

Arreglos

```c
#include <stdio.h>

int main (void) {
  int i;
  int vec[10];

  for ( i = 0; i < 5; i++ )
    vec[i] = 0;

  for ( i = 0; i < 5; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Si eventualmente se necesita un arreglo más grande, hay que cambiar el tamaño del arreglo...

...y el control de las sentencias repetitivas

Arreglos

```c
#include <stdio.h>

int main (void) {
  int i;
  int vec[10];

  for ( i = 0; i < 10; i++ )
    vec[i] = 0;

  for ( i = 0; i < 10; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Si eventualmente se necesita un arreglo más grande, hay que cambiar el tamaño del arreglo...

...y el control de las sentencias repetitivas

Arreglos

Arreglos

Si el programa tiene gran extensión, hacer estos cambios puede llevar a cometer errores...

Arreglos

Si el programa tiene gran extensión, hacer estos cambios puede llevar a cometer errores...

Para evitarlos se puede utilizar la directiva de preprocesador `#define`

Preprocesador

Preprocesador

Directiva `#define`

Preprocesador

Directiva `#define`

Se utiliza para **definir** *constantes simbólicas*

Preprocesador

Directiva `#define`

Se utiliza para **definir** *constantes simbólicas*

Preprocesador

Directiva `#define`

Se utiliza para **definir** *constantes simbólicas*

```c
#include <stdio.h>
#define N 5

int main (void) {
  int i;
  int vec[N];

  for ( i = 0; i < N; i++ )
    vec[i] = 0;

  for ( i = 0; i < N; i++ )
    printf("%d\n", vec[i]);

  return 0;
}
```

Preprocesador

Directiva `#define`

Preprocesador

Directiva `#define`

Antes de la compilación, el preprocesador *reemplaza* todas las constantes simbólicas por el valor que corresponde.

Preprocesador

Directiva `#define`

Antes de la compilación, el preprocesador *reemplaza* todas las constantes simbólicas por el valor que corresponde.

Los nombres de las constantes deben respetar las características de los identificadores, y se recomienda el uso de mayúsculas.

Preprocesador

Directiva `#define`

Antes de la compilación, el preprocesador *reemplaza* todas las constantes simbólicas por el valor que corresponde.

Los nombres de las constantes deben respetar las características de los identificadores, y se recomienda el uso de mayúsculas.

No deben usarse los punto y coma (`;`) en las directivas `#define` ya que luego, cuando se hagan los reemplazos podría haber errores de sintaxis.

Arreglos

Arreglos
Inicialización

Arreglos
Inicialización

Así como las variables comunes se podían inicializar en la definición

Arreglos
Inicialización

Así como las variables comunes se podían inicializar en la definición

```
int var = 3;
```

Arreglos
Inicialización

Así como las variables comunes se podían inicializar en la definición

```
int var = 3;
```

Los arreglos se pueden inicializar en la definición elemento por elemento usando llaves

Arreglos
Inicialización

Así como las variables comunes se podían inicializar en la definición

```
int var = 3;
```

Los arreglos se pueden inicializar en la definición elemento por elemento usando llaves

```
int vec[5] = {3, 5, 2, 10, 4};
```

Arreglos
Inicialización

Arreglos

Inicialización

Los elementos entre llaves se asignaran en orden desde el índice cero.

Arreglos
Inicialización

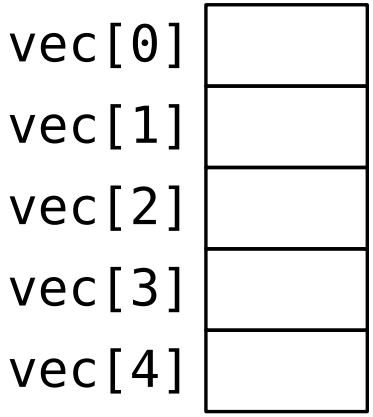
Los elementos entre llaves se asignaran en orden desde el índice cero.

```
int vec[5] = {3, 5, 2, 10, 4};
```

Arreglos
Inicialización

Los elementos entre llaves se asignaran en orden desde el índice cero.

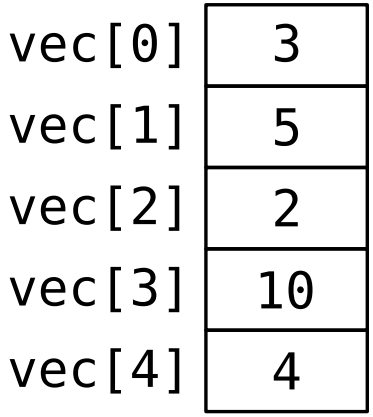
```
int vec[5] = {3, 5, 2, 10, 4};
```



Arreglos
Inicialización

Los elementos entre llaves se asignaran en orden desde el índice cero.

```
int vec[5] = {3, 5, 2, 10, 4};
```



Arreglos
Inicialización

Arreglos
Inicialización

Si los valores para inicializar son **menos** que el tamaño del arreglo, el resto de los elementos son inicializados en cero

Arreglos
Inicialización

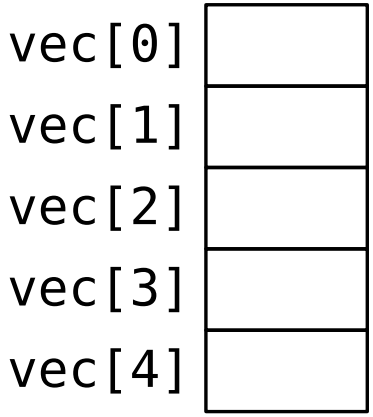
Si los valores para inicializar son **menos** que el tamaño del arreglo, el resto de los elementos son inicializados en cero

```
int vec[5] = {3, 5, 2};
```

Arreglos
Inicialización

Si los valores para inicializar son **menos** que el tamaño del arreglo, el resto de los elementos son inicializados en cero

```
int vec[5] = {3, 5, 2};
```



Arreglos
Inicialización

Si los valores para inicializar son **menos** que el tamaño del arreglo, el resto de los elementos son inicializados en cero

```
int vec[5] = {3, 5, 2};
```

vec[0]	3
vec[1]	5
vec[2]	2
vec[3]	0
vec[4]	0

Arreglos
Inicialización

Arreglos
Inicialización

Si los valores para inicializar son **más** que el tamaño del arreglo, el compilador da un `error` o `warning` dependiendo de si está la opción `-pedantic-errors`

Arreglos
Inicialización

Si los valores para inicializar son **más** que el tamaño del arreglo, el compilador da un `error` o `warning` dependiendo de si está la opción `-pedantic-errors`

```
int vec[5] = {3, 5, 2, 10, 4, 6};
```

Arreglos
Inicialización

Si los valores para inicializar son **más** que el tamaño del arreglo, el compilador da un `error` o `warning` dependiendo de si está la opción `-pedantic-errors`

```
int vec[5] = {3, 5, 2, 10, 4, 6};
```

vec[0]	3
vec[1]	5
vec[2]	2
vec[3]	10
vec[4]	4

Arreglos
Inicialización

Arreglos
Inicialización

Se puede inicializar **todo** el arreglo poniendo entre llaves un número cero

Arreglos
Inicialización

Se puede inicializar **todo** el arreglo poniendo entre llaves un número cero

```
int vec[5] = {0};
```

Arreglos
Inicialización

Se puede inicializar **todo** el arreglo poniendo entre llaves un número cero

```
int vec[5] = {0};
```

Estrictamente hablando de esta forma se inicializa el primer elemento en cero, y el resto de los elementos en cero.

Arreglos
Inicialización

Arreglos

Inicialización

Se puede omitir el tamaño entre los corchetes, siempre y cuando se usen las llaves para inicializar.

Arreglos
Inicialización

Se puede omitir el tamaño entre los corchetes, siempre y cuando se usen las llaves para inicializar.

```
int vec[] = {3, 5, 2, 10, 4};
```

Arreglos
Inicialización

Se puede omitir el tamaño entre los corchetes, siempre y cuando se usen las llaves para inicializar.

```
int vec[] = {3, 5, 2, 10, 4};
```

...pero el tamaño del arreglo siempre corresponderá a la cantidad de elementos inicializados entre llaves

Arreglos
Inicialización

Arreglos
Inicialización

Si se omiten tanto el tamaño entre corchetes como los inicializadores el compilador da error

Arreglos
Inicialización

Si se omiten tanto el tamaño entre corchetes como los inicializadores el compilador da error

```
int vec[];
```

Arreglos
Inicialización

Si se omiten tanto el tamaño entre corchetes como los inicializadores el compilador da error

```
int vec[];
```

```
error: array size missing in ‘vec’
   int vec[];
       ^~~
```

Arreglos

Arreglos de 2 Dimensiones

Arreglos

Arreglos de 2 Dimensiones

En C se pueden definir arreglos multidimensionales

Arreglos

Arreglos de 2 Dimensiones

En C se pueden definir arreglos multidimensionales

```
int mat[3][4];
```

Arreglos

Arreglos de 2 Dimensiones

En C se pueden definir arreglos multidimensionales

```
int mat[3][4];
```

El número entre corchetes a la izquierda determina la cantidad de filas.

Arreglos

Arreglos de 2 Dimensiones

En C se pueden definir arreglos multidimensionales

```
int mat[3][4];
```

El número entre corchetes a la izquierda determina la cantidad de filas.

El número entre corchetes a la derecha determina la cantidad de columnas.

Arreglos

Arreglos de 2 Dimensiones

Arreglos

Arreglos de 2 Dimensiones

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

Arreglos

Arreglos de 2 Dimensiones

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

Para acceder a cualquier elemento hay que utilizar los índices de filas y columnas.

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Como los arreglos de 1 dimensión, se utilizan llaves por cada fila y todas las filas también entre llaves.

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Como los arreglos de 1 dimensión, se utilizan llaves por cada fila y todas las filas también entre llaves.

```
int mat[2][3] = {{8,5,3},{4,6,7}};
```

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Como los arreglos de 1 dimensión, se utilizan llaves por cada fila y todas las filas también entre llaves.

```
int mat[2][3] = {{8,5,3},{4,6,7}};
```

8 mat[0][0]	5 mat[0][1]	3 mat[0][2]
4 mat[1][0]	6 mat[1][1]	7 mat[1][2]

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Se pueden inicializar de forma incompleta, donde los elementos faltantes se ponen en cero.

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Se pueden inicializar de forma incompleta, donde los elementos faltantes se ponen en cero.

```
int mat[2][3] = {{8,5},{4}};
```

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Se pueden inicializar de forma incompleta, donde los elementos faltantes se ponen en cero.

```
int mat[2][3] = {{8,5},{4}};
```

8 mat[0][0]	5 mat[0][1]	0 mat[0][2]
4 mat[1][0]	0 mat[1][1]	0 mat[1][2]

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Arreglos

Arreglos de 2 Dimensiones - Inicialización

...incluso pueden faltar filas.

Arreglos

Arreglos de 2 Dimensiones - Inicialización

...incluso pueden faltar filas.

```
int mat[2][3] = {{8,5}};
```

Arreglos

Arreglos de 2 Dimensiones - Inicialización

...incluso pueden faltar filas.

```
int mat[2][3] = {{8,5}};
```

8 mat[0][0]	5 mat[0][1]	0 mat[0][2]
0 mat[1][0]	0 mat[1][1]	0 mat[1][2]

Arreglos

Arreglos de 2 Dimensiones - Inicialización

Arreglos

Arreglos de 2 Dimensiones - Inicialización

O todos los elementos, salvo el primero. Sirve para inicializar en cero todo el arreglo.

Arreglos

Arreglos de 2 Dimensiones - Inicialización

O todos los elementos, salvo el primero. Sirve para inicializar en cero todo el arreglo.

```
int mat[2][3] = {{0}};
```

Arreglos

Arreglos de 2 Dimensiones - Inicialización

O todos los elementos, salvo el primero. Sirve para inicializar en cero todo el arreglo.

```
int mat[2][3] = {{0}};
```

0 mat[0][0]	0 mat[0][1]	0 mat[0][2]
0 mat[1][0]	0 mat[1][1]	0 mat[1][2]