

# Laboratorio de Arquitectura de Redes

---

Asignación dinámica de memoria en  
lenguaje C

# Asignación dinámica de memoria en lenguaje C

---

- Definición
- La memoria en los programas en ejecución
- Asignación y liberación dinámica de memoria
- Vectores dinámicos
  - Unidimensionales
  - Bidimensionales
- Reasignación de bloques de memoria

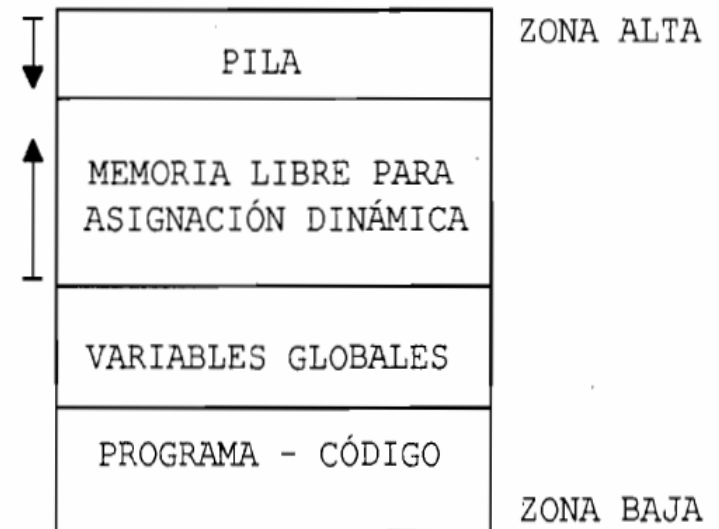
# Definición

---

- En la declaración de variables, el compilador de C reserva memoria para cada variable:
  - Si son globales, en la zona de datos asociada al propio programa
  - Si son locales, en la zona de pila (stack)
- La **asignación dinámica de memoria** consiste en la reserva de memoria para los datos en tiempo de ejecución, es decir, tomándola de la que el sistema tiene libre en ese momento.

# La memoria en los programas en ejecución

- Un programa en ejecución es un *proceso activo* que tiene a su disposición:
  - El tiempo utilización del procesador
  - La memoria que le asignó el sistema operativo
    - Segmento de código
    - Segmento de datos (variables globales)
    - Segmento de pila
      - Variables locales
      - Direcciones de regreso de las llamadas a funciones



**UBICACIÓN EN MEMORIA  
DE UN PROGRAMA EN C**

# Asignación y liberación dinámica de memoria (I)

---

- En tiempo de ejecución es posible *pedir* memoria al sistema operativo:

```
void *malloc(unsigned tamaño);
```

- Está declarada en `STDLIB.H`
- `tamaño` es un entero sin signo que indica el número de bytes que se solicitan al sistema operativo
- La función devuelve un puntero genérico que apunta a la dirección de comienzo del bloque de memoria asignado. En caso de error devuelve un puntero nulo (`NULL`)
  - En el prototipo se indica que devuelve un puntero de tipo `void` que puede apuntar a cualquier tipo válido.

# Asignación y liberación dinámica de memoria (II)

---

- ❑ Tras utilizar la memoria asignada dinámicamente, hay que *devolverla* al sistema operativo, liberándola de nuestro programa:

```
void free(void *nombrepuntero);
```

- ❑ Está declarada en `STDLIB.H`
- ❑ `nombrepuntero` es el identificador del puntero (de cualquier tipo) que apunta al bloque de memoria que se desea liberar
- ❑ La función no devuelve nada

# Asignación y liberación dinámica de memoria (III)

---

- Ejemplo: asignación dinámica de memoria para un entero y liberación posterior:

```
int *dato;    /* Puntero a int */
dato = (int *)malloc(sizeof(int));
if (dato==NULL)
    printf("Error en la asignación");
...
/* Operaciones de utilización de la
   memoria asignada */
free(dato);  /* Liberación de la memoria
             asignada dinámicamente */
```

# Vectores dinámicos (I)

---

- ❑ Los **vectores dinámicos** son aquellos cuyo tamaño se determina en un proceso de asignación dinámica de memoria
- ❑ **Vectores dinámicos unidimensionales:**

```
void * calloc(numelementos, tamañoelemento);
```

  - ❑ Está declarada en `STDLIB.H`
  - ❑ Devuelve un puntero convertible a cualquier tipo de dato que apunta a la primera posición del bloque de memoria asignado (o puntero nulo en caso de error)
  - ❑ `numelementos` es un entero sin signo que representa el número de elementos del vector
  - ❑ `tamañoelemento` es un entero sin signo que representa el tamaño de un elemento del vector
- Ejemplo: reserva de memoria para un vector de 10 elementos de tipo `double`:

```
punt =(double*)calloc(10, sizeof(double));
```



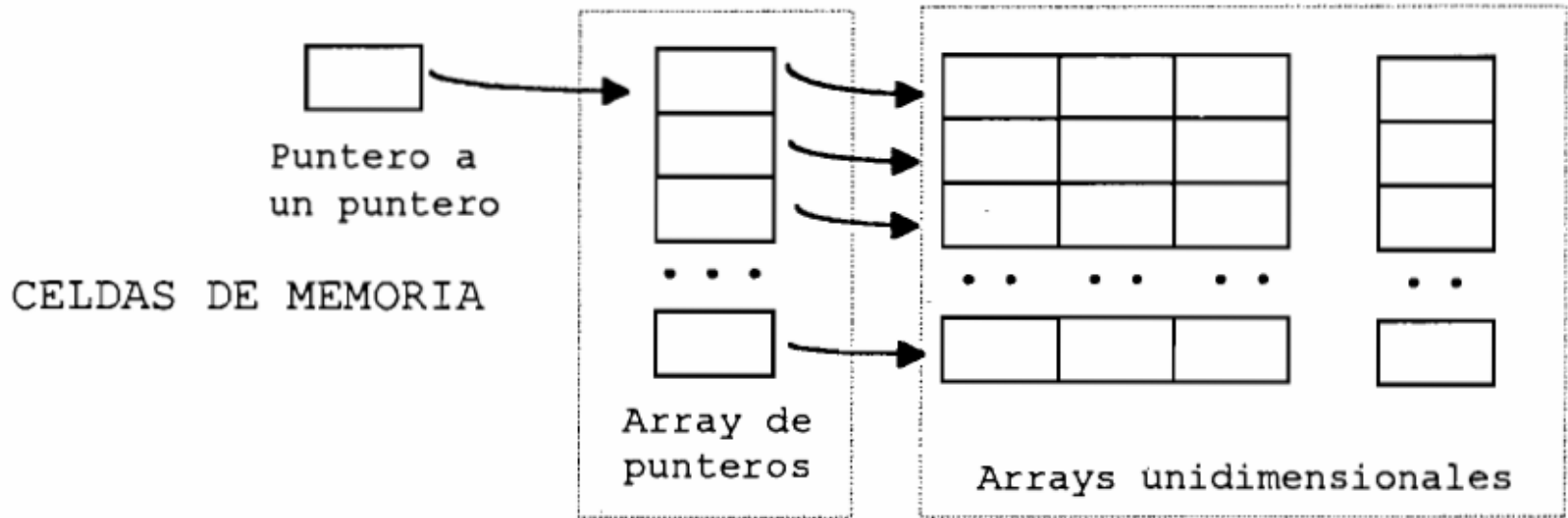
# Vectores dinámicos (II)

---

## □ **Vectores dinámicos bidimensionales:**

1. Debemos crear un puntero a puntero al tipo de datos del array bidimensional
2. Debe asignarse dinámicamente un vector de punteros
3. Debe asignarse dinámicamente un vector para datos a cada uno de los punteros del vector anterior
4. Tras su utilización, debe liberarse la memoria en orden inverso al orden en el que fue asignada:
  1. Primero, mediante un bucle, se libera cada uno de los vectores unidimensionales del tipo de dato utilizado
  2. Después, mediante otro bucle, se libera el vector de punteros

# Vectores dinámicos (III)



**ARRAY BIDIMENSIONAL CREADO MEDIANTE  
ASIGNACIÓN DINÁMICA DE MEMORIA**

# Vectores dinámicos (IV)

---

- Ejemplo: asignación dinámica de memoria para un vector unidimensional de  $N$  enteros y liberación posterior:

```
int *lista; /* Puntero a int */
lista = (int *)calloc(N, sizeof(int));
if (lista==NULL)
    printf("Error en la asignación");
...
/* Operaciones de utilización de la
   memoria asignada */
free(lista); /* Liberación de la memoria
             asignada dinámicamente */
```

# Vectores dinámicos (V)

---

- Ejemplo: reserva dinámica de memoria para un vector bidimensional de números reales (matriz de NFIL filas y NCOL columnas) y su liberación posterior:

```
float **punt;
                // Puntero a puntero a float
int i,j;
punt = (float **)calloc(NFIL , sizeof(float *))
        // Array de NFIL punteros a float
for (i=0 ; i<NFIL ; i++)
    punt[i] = (float *)calloc(NCOL , sizeof(float));
        // Se dispone de NFIL arrays de NCOL números reales.
        // Los datos pueden ser referenciados mediante expresiones
        // del tipo punt[i][j]
for (i=0 ; i<NFIL ; i++) free(punt[i]);
        // Se liberan los arrays unidimensionales (las filas de la
        // matriz de números reales)
free(punt);
        // Se libera la memoria del array unidimensional de
        // punteros a float
```

# Reasignación de bloques de memoria

---

- Es posible *reassignar* un bloque de memoria previamente asignado dinámicamente (para redimensionar ese bloque)

```
void * realloc(void *puntbloc, numbytes);
```

- La función está definida en `STDLIB.H`
- Devuelve un puntero genérico al bloque de memoria asignado, cuya dirección de comienzo puede ser diferente de la dirección inicial. En caso de error devuelve un puntero nulo (`NULL`)
- No se pierden los datos almacenados en el bloque de partida, se trasladan su ubicación
- `puntbloc` es un puntero que apunta al bloque de memoria que se quiere reasignar
- `numbytes` es un número entero sin signo que indica el nuevo tamaño en bytes que deberá tener el bloque de memoria