Informatics

Ingeniería en Electrónica y Automática Industrial

Dynamic Memory Allocation

V1.1

Dynamic Memory Allocation

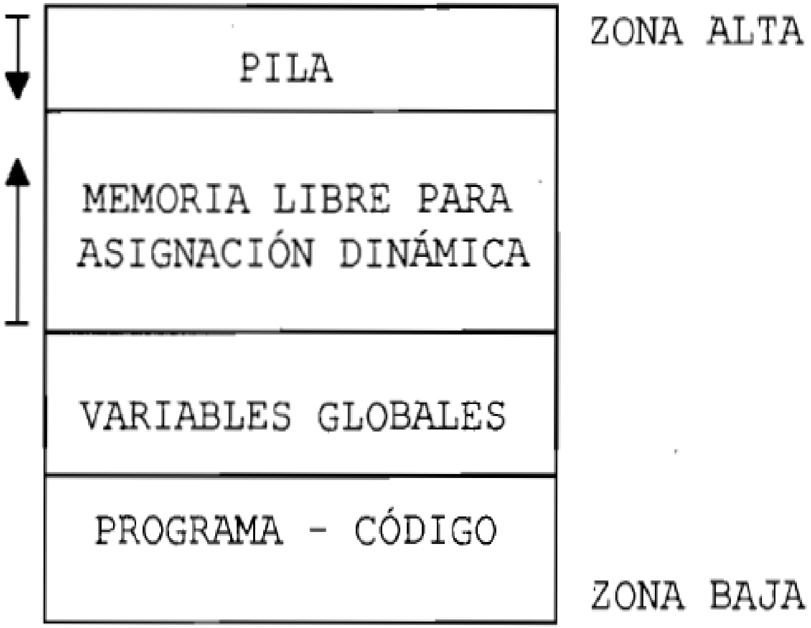
- Definition
- Memory map during program execution
- Dynamic memory allocation and release
- Dynamically allocated arrays
  - Unidimensional
  - Bidimensional
- Reallocation of memory blocks

Definition

- The compiler reserves memory for variables when they are declared, *before execution*
  - If global or static, in the data segment of the program
  - If local, in the stack
- The **dynamic memory allocation** is the assignment of memory space in ***execution time.***
  - The OS assigns the required memory from the available one in that moment
  - Very important tool when working with big multidimensional arrays to use memory efficiently

Memory map during program execution

- A program in execution is an *active process* that can use the memory assigned by OS:
  - Code segment
    - Program
  - Data segment
    - Global and static variables
  - Stack segment
    - Local variables
    - Return addresses in function calls
  - Free memory
    - **Dynamic allocation**



UBICACIÓN EN MEMORIA DE UN PROGRAMA EN C

Dynamic memory allocation and release (I)

- The program can ask for memory to the OS during execution time with **`malloc()`** function

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

  - Declared in `stdlib.h`
  - `size` indicates the number of requested bytes
  - It returns a generic pointer to the first address of the assigned memory block (`NULL` if error)

Dynamic memory allocation and release (II)

- After use, memory must be released with **`free()`** function

```
void free(void *pblock);
```

  - Declared in `stdlib.h`
  - `pblock` is the pointer to the block to be released
  - The function does not return anything

Dynamic memory allocation and release (III)

- Example:

```
int *dat;
dat = (int *)malloc(sizeof(int));  /*Assign*/
if (dat==NULL)
    printf(“Allocation error”);

...                            /*Using dat*/

free(dat);                        /*Release*/
```

Dynamically allocated arrays (I)

- Are the arrays whose size is fixed in execution time when they are allocated with **`calloc()`** function
- **Unidimensional dynamically allocated arrays**

```
void * calloc(numelements, elementsize);
```

  - Declared in `stdlib.h`
  - Returns a pointer to the first address of the assigned memory block (`NULL` if error)
  - `numelements` indicates the number of elements in the array
  - `elementsize` indicates the size of each element

Dynamically allocated arrays (II)

- Example: Dynamic allocation of an array of `N` integers

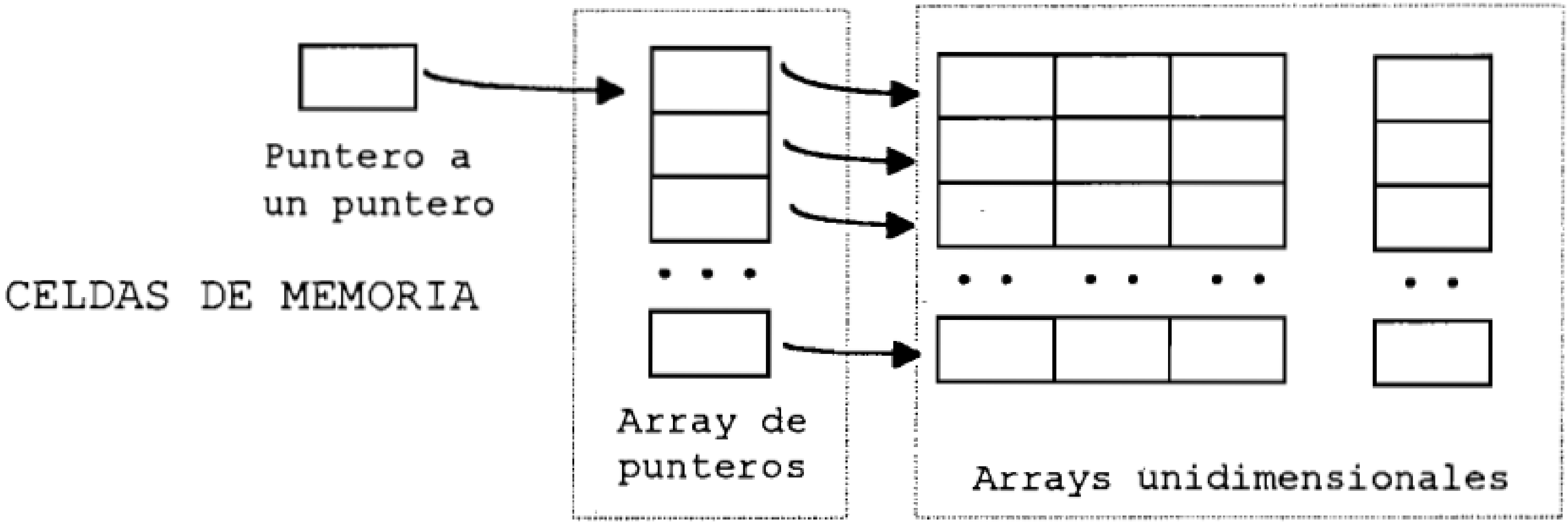
```
int *arr10;                 // Pointer to int
arr10= (int *)calloc(N, sizeof(int)); //Assign
if (arr10==NULL)
    printf(“Allocation error”);

  ...                          // Using arr10

free(arr10);                // Memory release
```

Dynamically allocated arrays (III)

- **Bidimensional dynamically allocated arrays**:

1. Declare a ***pointer to pointer*** to the data type of the 2D-array
2. Assign dynamically a 1D-array of pointers
3. Assign dynamically a 1D-array of data to each of the pointers of the previous array of pointers
4. Normal use of the 2D-array
5. Release memory in inverse order:
    1. With a loop release every 1D-array of data
    2. Release the 1D-array of pointers

Dynamically allocated arrays (IV)



ARRAY BIDIMENSIONAL CREADO MEDIANTE ASIGNACIÓN DINÁMICA DE MEMORIA

Dynamically allocated arrays (V)

- Example: 2D-array (NROW, NCOL) of real numbers

```
float **arr2D;            // Pointer to pointer to float
int i,j;

arr2D = (float **)calloc(NROW , sizeof(float *))
    //Assign mem for 1D-array of NROW pointers to float

for (i=0 ; i<NROW ; i++;)
    arr2D[i] = (float *)calloc(NCOL , sizeof(float));
     // Assign mem for each 1D-array of NCOL float numb

 ... // Use arr2D, elements can be accessed arr2D[i][j]

for (i=0 ; i<NROW ; i++) free(arr2D[i]);
                  // Release the 1D-arrays of real data
free(arr2D);            // Release the 1D-array of pointers
```

Reallocation of memory blocks

- In execution it is possible to **change the size assigned to an array** by reallocating the memory block it occupies

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

  - Declared in `stdlib.h`
  - Returns a pointer to the new memory block that might be different to the previous one (`NULL` if error)
  - Data of the original block are not lost
  - `ptoldblock` points to the original block to reallocate
  - `numbytes` indicates the size in bytes of the new block