.UBA fiuba

FACULTAD DE INGENIERÍA

Memoria Dinámica en C

Parte 2 - Funciones

Algoritmos y Programación I - Fundamentos de Programación

rev1.1

Diego Serra - Gustavo Bianchi

Funciones para gestión de memoria dinámica

Funciones	Descripción
`malloc( )`	Solicita una porción de memoria disponible, sin inicializar.
`realloc( )`	Reasigna una porción de memoria ya disponible, por otra de diferente tamaño.
`free( )`	Libera una porción de memoria previamente solicitada.
`calloc( )`	Solicita una porción de memoria disponible, inicializando la misma a cero.

Los prototipos de las funciones se encuentran en <stdlib.h>

Prototipos de las Funciones

```c
#include <stdlib.h>


void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

Aclaración: `size_t` estará definido según la plataforma como alguna variante de `unsigned integer`, la mayor que el sistema pueda operar.

Función malloc( )

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

Descripción:

- ➔ Reserva la memoria solicitada y retorna un puntero a dicha memoria.
- ➔ No se realiza ninguna inicialización de la misma.

Parámetros:

- ➔ **size** – tamaño del bloque de memoria en bytes.

Valor de retorno:

- ➔ retorna un puntero a la memoria reservada, o bien NULL si la solicitud falla.

Función free( )

```
void free(void *ptr);
```

Descripción:

- Libera el bloque de memoria previamente reservada por una invocación a malloc, calloc o realloc.

Parámetros:

- **ptr** – puntero al bloque de memoria que se quiere liberar que fue previamente reservado por una invocación a malloc, calloc o realloc.
  Si un puntero NULL se pasa como argumento, no se lleva a cabo acción alguna.

Valor de retorno:

- Esta función no retorna valor.

Aclaración: En el caso que **ptr** haya sido liberado con free previamente, un comportamiento indefinido puede llevarse a cabo.

Función calloc( )

```
void *calloc(size_t nmemb, size_t size);
```

Descripción:

- ➔ Reserva la memoria solicitada y retorna un puntero a dicha memoria. Todo el bloque es inicializado a cero.

Parámetros:

- ➔ **nmems** – cantidad de elementos para los cuales se quiere reservar.
- ➔ **size** – tamaño de cada elemento en bytes.

Valor de retorno:

- ➔ Retorna un puntero a la memoria reservada, o bien NULL si la solicitud falla.

Función realloc( )

```
void *realloc(void *ptr, size_t size);
```

Descripción:

➔ Intenta cambiar el tamaño del bloque de memoria señalado por ptr y que fue asignado previamente con una llamada a malloc( ) o calloc( ).

Parámetros:

➔ **ptr** – Puntero a un bloque de memoria previamente asignado con malloc, calloc o realloc que se va a reasignar. Si es NULL, se asigna un nuevo bloque y la función devuelve un puntero al mismo.

➔ **size** – Nuevo tamaño del bloque de memoria, en bytes. Si es 0 y ptr apunta a un bloque de memoria existente, el bloque de memoria señalado por ptr se desasigna y se devuelve un puntero NULL.

Valor de retorno:

➔ Retorna un puntero a la nueva memoria reservada, o bien NULL si la solicitud falla.

Función sizeof( )

Caso de uso con nombre de tipo:

```
sizeof(nombre-tipo)
```

Descripción:

- ➔ Permite obtener la cantidad de bytes que ocupa un tipo de dato en memoria

Parámetros:

- ➔ **nombre-tipo** nombre del tipo de dato del cual se quiere conocer el tamaño en bytes.

Valor de retorno:

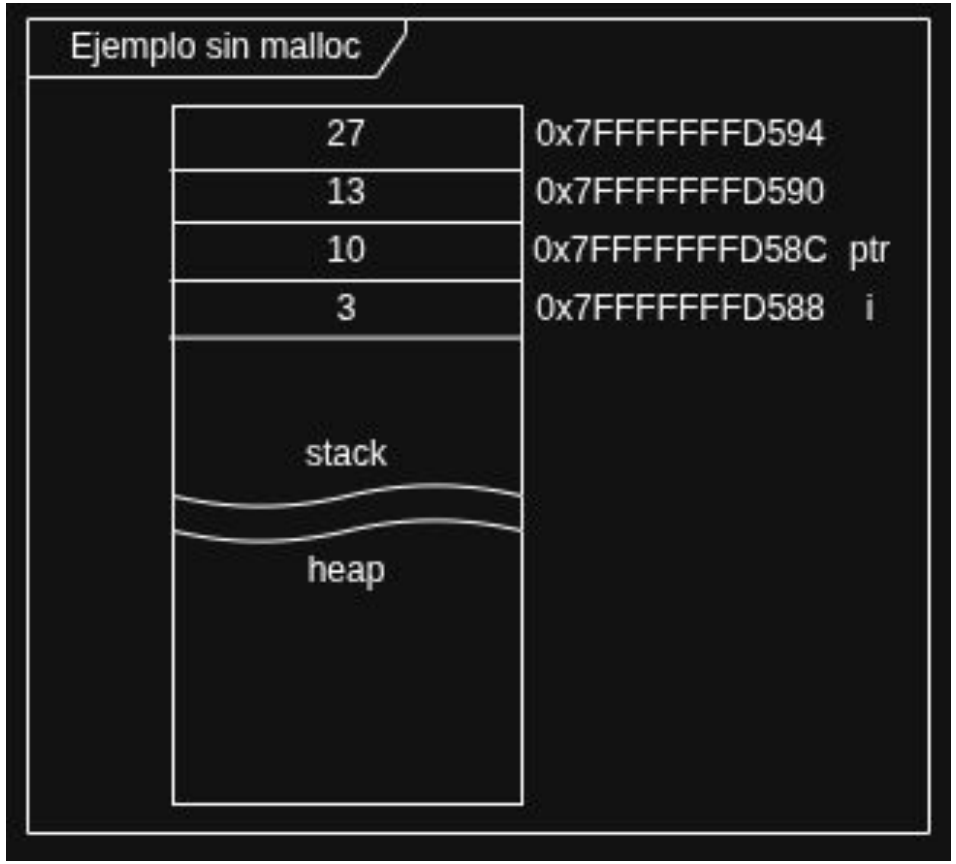
- ➔ retorna la cantidad de bytes del tipo de datos

Ejemplo de uso (reserva espacio para 10 enteros):

```
int * ptr = malloc(10 * sizeof(int));
```

Ejemplo sin uso de malloc( )

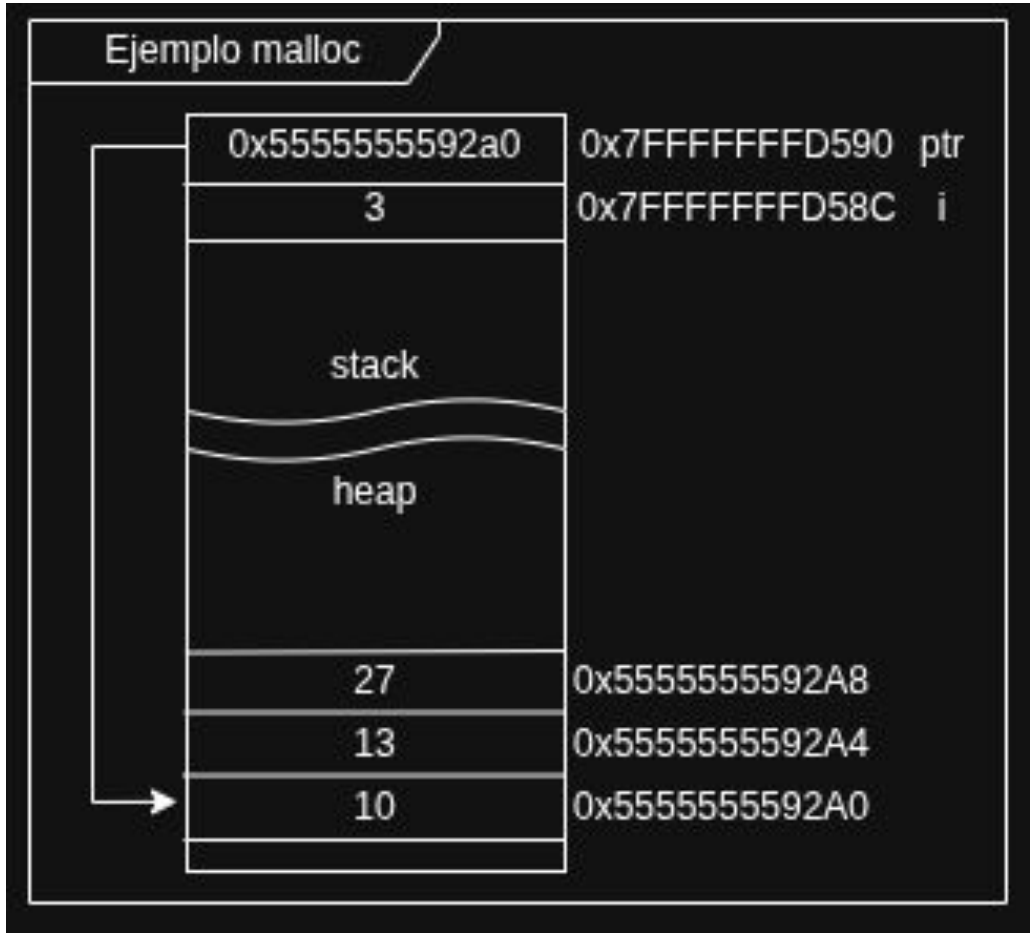
```c
#include <stdio.h>

#define MF 3

int main() {
    int ptr[MF];
    int i;

    for(i = 0; i < MF; i++) {
        printf("Ingrese numero positivo: ");
        scanf("%d", &ptr[i]);
    }

    for(i = 0; i < MF; i++) {
        printf("%d ", ptr[i]);
    }

    return 0;
}
```



Caso en donde el usuario ingrese los números: 10, 13, 27.

Ejemplo con uso de malloc( )

```c
#include <stdio.h>
#include <stdlib.h>

#define MF 3

int main() {
    // declaramos puntero
    int *ptr = NULL;
    int i;

    ptr = malloc(MF * sizeof(int));
    if (ptr != NULL) {
        for(i = 0; i < MF; i++) {
            printf("Ingrese numero positivo: ");
            scanf("%d", &ptr[i]);
        }
    } else {
        printf("Error de memoria.\n");
    }

    if (ptr != NULL) {
        for(i = 0; i < MF; i++) {
            printf("%d ", ptr[i]);
        }
    }

    free(ptr);

    return 0;
}
```



Caso en donde el usuario ingrese los números: 10, 13, 27.

Unidades de Byte

Multiple-byte units v·t·e

Decimal				Binary					
Value		Metric		Value		IEC		Memory	
1000	10^3	kB	kilobyte	1024	2^{10}	KiB	kibibyte	KB	kilobyte
1000^2	10^6	MB	megabyte	1024^2	2^{20}	MiB	mebibyte	MB	megabyte
1000^3	10^9	GB	gigabyte	1024^3	2^{30}	GiB	gibibyte	GB	gigabyte
1000^4	10^{12}	TB	terabyte	1024^4	2^{40}	TiB	tebibyte	TB	terabyte
1000^5	10^{15}	PB	petabyte	1024^5	2^{50}	PiB	pebibyte		–
1000^6	10^{18}	EB	exabyte	1024^6	2^{60}	EiB	exbibyte		–
1000^7	10^{21}	ZB	zettabyte	1024^7	2^{70}	ZiB	zebibyte		–
1000^8	10^{24}	YB	yottabyte	1024^8	2^{80}	YiB	yobibyte		–
1000^9	10^{27}	RB	ronnabyte	1024^9	2^{90}		–		–
1000^{10}	10^{30}	QB	quettabyte	1024^{10}	2^{100}		–		–

Orders of magnitude of data

En 1999, la Comisión Electrotécnica Internacional (IEC) publicó estándares para prefijos binarios

Perdida de memoria (Memory Leak)

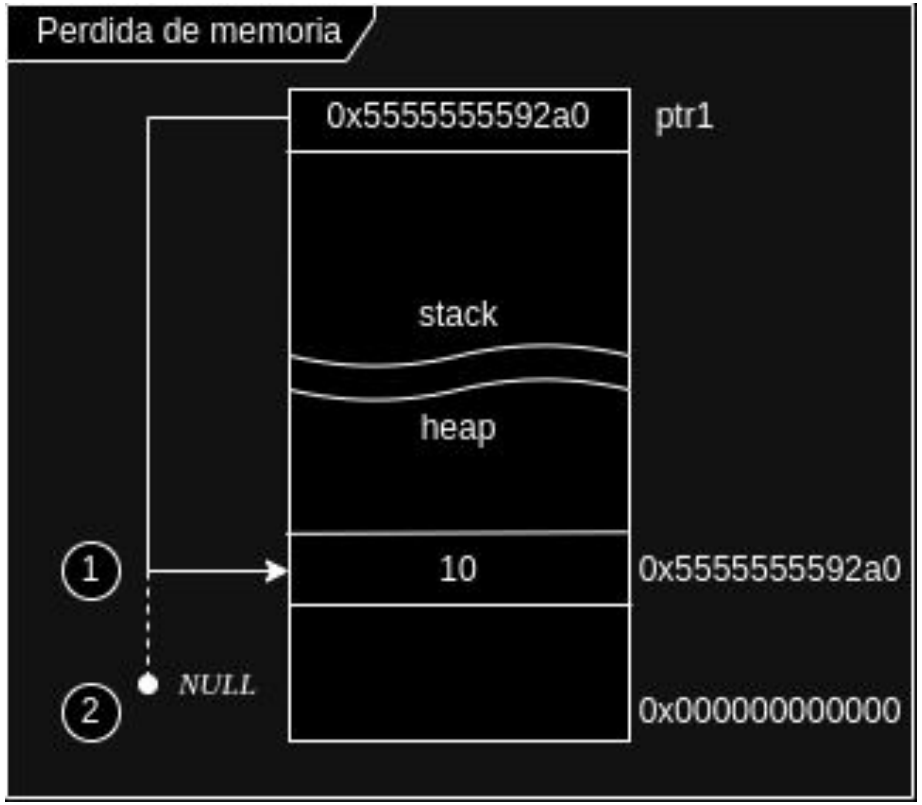
1) Cuando la **memoria asignada dinámicamente no se libera llamando a free**, **se producen pérdidas de memoria**.
   Asegúrese siempre que por cada asignación de memoria dinámica que utilice **malloc** o **calloc**, haya una llamada a **free** correspondiente.

2) Cuando se pierde el seguimiento de los punteros que hacen referencia a la memoria asignada, puede suceder que la memoria no se libere. Por lo tanto, realice un seguimiento de todos los punteros y asegúrese de que se libere la memoria.

3) Cuando el programa finaliza abruptamente y la memoria asignada no se libera o si alguna parte del código impide la llamada a **free**, pueden ocurrir pérdidas de memoria.

Ejemplo de pérdida de memoria

```c
/*
Caso de perdida de memoria
(Memory Leak)
*/

#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr1 = NULL;

    ptr1 = malloc(sizeof(int));
    if (ptr1 != NULL) {
        *ptr1 = 10;
    }

    // 1
    ptr1 = NULL;
    // 2
    free(ptr1);
    return 0;
}
```



Al tener memoria reservada sin posibilidad de ser accedida para liberarla se produce una pérdida.

Buenas prácticas

- ✓ Utilizar el operador **sizeof** para determinar el tamaño de una estructura.
- ✓ Cuando se utiliza malloc, se debe validar que el puntero devuelto no sea NULL, y gestionar correctamente el caso.
- ✓ Cuando la memoria que se asigna dinámicamente ya no es necesaria, utilizar **free** para devolverla inmediatamente al sistema.

Errores de programación

- Liberar con **free()** memoria que no ha sido asignada dinámicamente con **malloc()** o **calloc()** es un error.
- Hacer referencia a memoria que ha sido liberada es un error.
- Intentar liberar memoria que ya ha sido liberada previamente (doble **free()**).

Bibliografía

1) *Programación en C - Metodología, algoritmos y estructura de datos [Joyanes Aguilar][Mcgraw-Hill] - Capítulo 11 - Asignación dinámica de memoria.*
2) *Computer Systems, A Programmer's Perspective - Capitulo 9.9 - Dynamic Memory Allocation [Bryant & O'Hallaron][3ed][Pearson]*
3) *Pointers in C Programming - Chapter 9 - Dynamic Memory Management [Thomas Mailund][Apress]*
4) *The C Programming Language [Kernighan & Ritchie][2ed][Prentice Hall]*

Links

1) *Linux Manual Page - malloc(3)*
   https://man7.org/linux/man-pages/man3/malloc.3.html
2) *Multiple-byte units*
   https://en.wikipedia.org/wiki/Byte#Multiple-byte_units