**Universidad Nacional Autónoma de México**

Facultad de Ingeniería

**Laboratorios de computación**

**salas A y B**

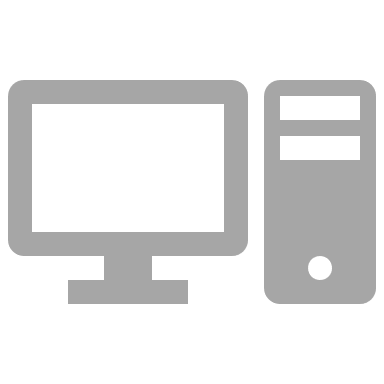
**PROFESOR:** M.I. Marco Antonio Martínez Quintana

**ASIGNATURA:** Estructura de Datos y Algoritmos I

**GRUPO:** 17

**NO DE PRÁCTICA:** 4

**NOMBRE:** Reyes Mendoza Miriam Guadalupe

**SEMESTRE:** 2020-2

**FECHA DE ENTREGA:** 22/02/2020

**OBSERVACIONES:**

**CALIFICACIÓN:**

**ALMACENAMIENTO EN TIEMPO DE EJECUCIÓN**

**OBJETIVO**

Utilizarás funciones en lenguaje C que permiten reservar y almacenar información de manera dinámica (en tiempo de ejecución).

**INTRODUCCIÓN**

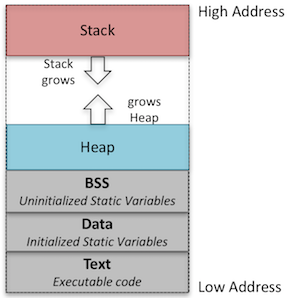
**MEMORIA DINAMICA**

La memoria dinámica se refiere al espacio de almacenamiento que se reserva en tiempo de ejecución, debido a que su tamaño puede variar durante la ejecución del programa.

El uso de memoria dinámica es necesario cuando a priori no se conoce el número de datos y/o elementos que se van a manejar.

Dentro de la memoria RAM, la memoria reservada de forma **dinámica** está alojada en el **heap** o **almacenamiento libre** y la memoria **estática** (como los arreglos o las variables primitivas) en el **stack** o **pila**.

La pila generalmente es una zona muy limitada. El heap, en cambio, en principio podría estar limitado por la cantidad de memoria disponible durante la ejecución del programa y el máximo de memoria que el sistema operativo permita direccionar a un proceso.

[](https://www.google.com/url?sa=i&url=https%3A%2F%2Fstudy.com%2Facademy%2Flesson%2Fdynamic-memory-allocation-definition-example.html&psig=AOvVaw021Bu1IjT0T8fpP43ORHTl&ust=1583115771488000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCJC2yvyb-OcCFQAAAAAdAAAAABBK)

**HEAP**

El montículo o heap (también llamado segmento de datos) es un medio de almacenamiento con más capacidad que la pila y que puede almacenar datos durante toda la ejecución de las funciones. Las variables globales y estáticas viven en el heap mientras la aplicación se esté ejecutando.

Para acceder a cualquier dato almacenado dentro del heap se debe tener una referencia o apuntador en la pila.

La memoria que se define de manera explícita (estática) tiene una duración fija, que se reserva (al iniciar el programa) y libera (al terminar la ejecución) de forma automática. La memoria dinámica se reserva y libera de forma explícita.

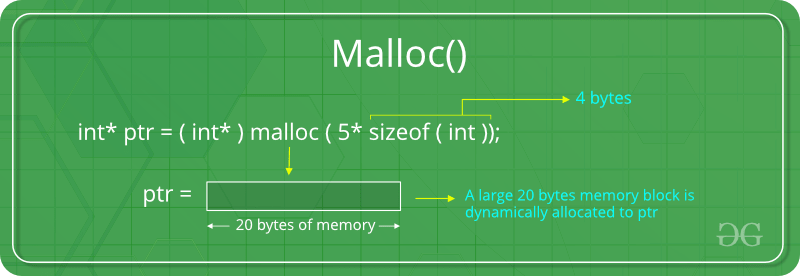
El almacenamiento dinámico puede afectar el rendimiento de una aplicación debido a que se llevan a cabo arduas tareas en tiempo de ejecución: buscar un bloque de memoria libre y almacenar el tamaño de la memoria asignada. Lenguaje C **permite el almacenamiento de memoria en tiempo de ejecución** a través de tres funciones: **malloc**, **calloc** y **realloc**.

**MALLOC**

La función malloc **permite reservar un bloque de memoria de manera dinámica y devuelve un apuntador tipo void**. Su sintaxis es la siguiente:

**void \*malloc(size\_t size);**

La función recibe como parámetro el número de bytes que se desean reservar. En caso de que no sea posible reservar el espacio en memoria, se devuelve el valor nulo (NULL).

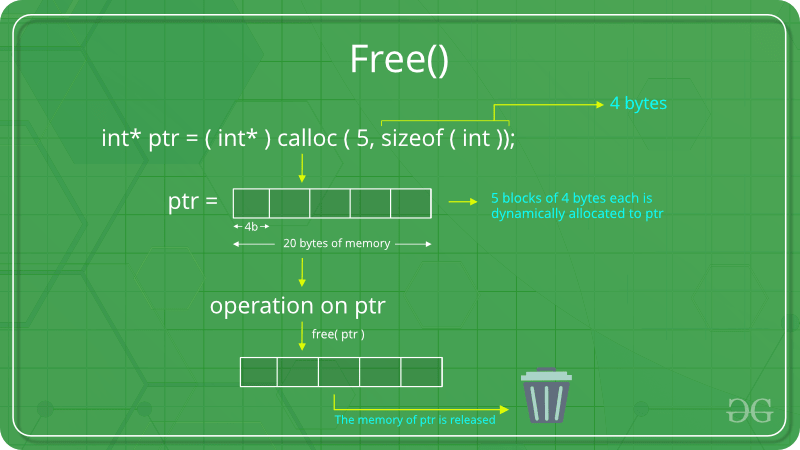


**FREE**

El almacenamiento en tiempo de ejecución se debe realizar de manera **explícita**, es decir, desde la aplicación se debe enviar la orden para reservar memoria. Así mismo,cuando la memoria ya no se utilice o cuando se termine el programa se debe liberar la memoria reservada. La función free **permite liberar memoria que se reservó de manera dinámica**. Su sintaxis es la siguiente:

**void free(void \*ptr);**

El parámetro ptr es el apuntador al inicio de la memoria que se desea liberar. Si el apuntador es nulo la función no hace nada.

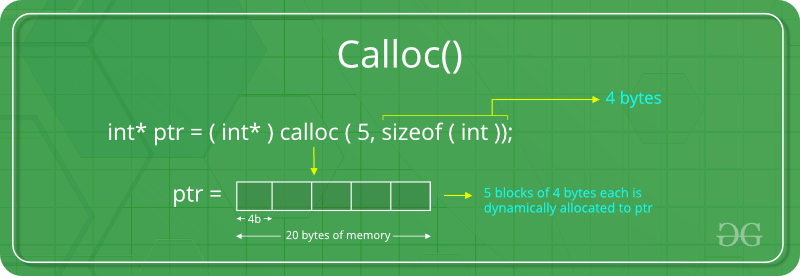


**CALLOC**

La función calloc funciona de manera similar a la función malloc pero, **además de reservar memoria en tiempo real, inicializa la memoria reservada con 0**. Su sintaxis es la siguiente:

**void \*calloc (size\_t nelem, size\_t size);**

El parámetro nelem indica el número de elementos que se van a reservar y size indica el tamaño de cada elemento.



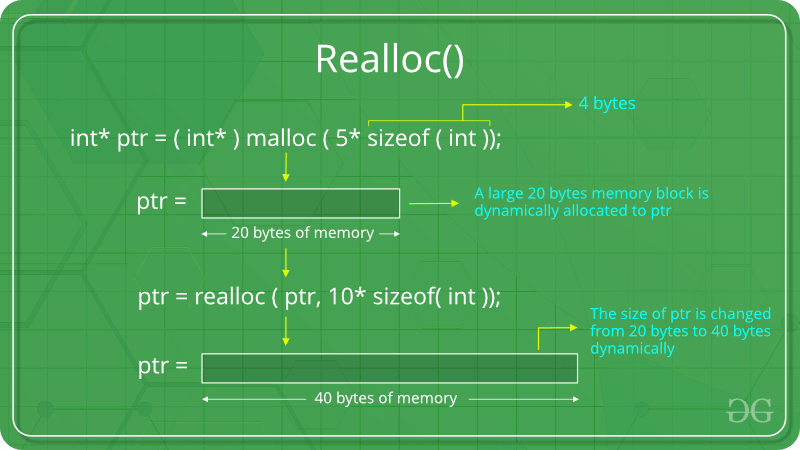
**REALLOC**

La función realloc **permite redimensionar el espacio asignado previamente de forma dinámica, es decir, permite aumentar el tamaño de la memoria reservada de manera dinámica**. Su sintaxis es la siguiente:

**void \*realloc (void \*ptr, size\_t size);**

Donde ptr es el apuntador que se va a redimensionar y size el nuevo tamaño, en bytes, que se desea aumentar al conjunto.

Si el apuntador que se desea redimensionar tiene el valor nulo, la función actúa como la función malloc. Si la reasignación no se pudo realizar, la función devuelve un apuntador a nulo, dejando intacto el apuntador que se pasa como parámetro (el espacio reservado previamente).



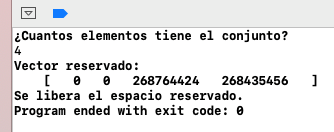
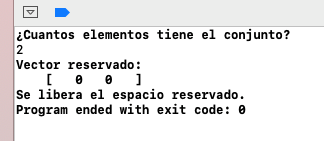
**DESARROLLO Y RESULTADOS**

1. **CÓDIGO: MALLOC**

**Imagen que contiene captura de pantalla

Descripción generada automáticamente**

**RESULTADOS**

****

**EXPLICACION**

Este programa hace un arreglo reservando espacios o bloques de memoria de manera dinámica. Este no ocupa funciones pues es un programa corto y sencillo para el cual no es tan necesario utilízalas dentro de la función principal.

Este empieza declarando 3 variables de tipo entero la primera es un apuntador dirigido a la localidad del arreglo; después una variable llamada num, la cual nos va a servir para definir el tamaño del arreglo y por último una variable llamada cont que es para que se recorra el arreglo y se vayan imprimiendo los valores de cada casilla. Se imprime en pantalla una pregunta para que se defina el numero de elemento del arreglo y el número que se lee se guarda en la variable num.

Después viene el uso de la función malloc esta se utiliza de la siguiente manera:

El arreglo va a ser igual a un apuntador void, pero se convierte en int para que no haya incompatibilidad en los tipos de datos. Después se escribe la función en la cual se multiplica el valor de la variable num por sizeof(int) para para hacer la reserva en cantidad de memoria de la cantidad de números.

Posteriormente se pone la condición donde que si el arreglos es diferente de NULL (si no se pudo reservar el espacio de memoria) se imprima en pantalla el vector reservado (arreglo) mediante un ciclo de repetición for que va a iniciar en la posición 0 del arreglo y lo va a ir recorriendo hasta que se llegue a la posición (num-1). Por último, el programa se termina usando una función más:

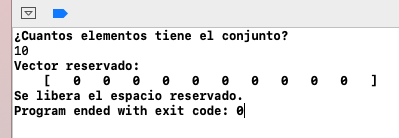
Free la cual libera la memoria reservada de manera dinámica.

1. **CÓDIGO: CALLOC**

**Imagen que contiene captura de pantalla

Descripción generada automáticamente**

**RESULTADOS**

****

**EXPLICACION**

Este programa hace un arreglo reservando espacios o bloques de memoria de manera dinámica, pero la inicializa en 0. Este es igual al anterior solamente que ocupa la función calloc en vez de malloc:

Esta iguala el arreglo a el apuntador void, pero se convierte en int para que no haya incompatibilidad en los tipos de datos. Después se escribe la función en la cual la variable num van a ser el número de elementos que se van a reservar y sizeof(int) el tamaño que va a tener cada uno de estos.

Después imprime el arreglo y por último limpia memoria que habíamos reservado.

Algo que a mí me hizo darme cuenta de la diferencia que tenia porque en su estructura son muy similares (en calloc se utiliza una coma para separar los elementos y su tamaño a comparación de malloc) en su ejecución podemos notar que si tu en la función malloc no vuelves a compilar el programa y solo lo ejecutas te empieza a imprimir basura pues a diferencia de calloc este no lo hace al momento y no se vuelva a inicializar en 0, cosa que la función calloc si hace y por eso jamás imprime basura.

1. **CÓDIGO: REALLOC**

**Imagen que contiene captura de pantalla

Descripción generada automáticamente**

**RESULTADOS**

**Imagen que contiene captura de pantalla

Descripción generada automáticamente**

**EXPLICACION**

Este programa no ocupa funciones y la función principal inicia declarando 4 variables todas de tipo entero la primeras dos son apuntadores que dirigen a la localidad de su respectivo arreglo y las cuales nos sirven para imprimir, definir y aumentar el tamaño del arreglo. Las otras dos variables se llaman num y cont, la primera nos sirve para dar el número de elementos que va a tener y la segunda para recorrer e imprimir los datos correspondientes a cada casilla de del mismo.

Se piden los numero de elementos que va a tener y mediante un malloc el arreglo va a ser igual a un puntero int. Después se escribe la función en la cual se multiplica el valor de la variable num por sizeof(int) para para hacer la reserva en cantidad de memoria.

Posteriormente se pone la condición donde que si el arreglos es diferente de NULL (si no se pudo reservar el espacio de memoria) se pidan los números que van a ir en cada casilla, los cuales lee y va guardando hasta que este quede lleno (de acuerdo con la condición) y cuando se termine el ciclo se imprima en pantalla el vector reservado (arreglo) mediante otro ciclo de repetición for.

Al terminar esto el tamaño del arreglo aumenta al doble, esto se hace declarando que la variable num\* va a ser igual a 2 veces esta, con esto entra la función realloc:

Lo que hace la función realloc es que iguala a otro arreglo a un apuntador int en el cual se escribe la función donde arreglo es el apuntador que se va a redimensionar y num\*size(off) es el nuevo tamaño (en bytes) que va a aumentar al conjunto.

Posteriormente se pone la condición donde que si el arreglos es diferente de NULL (si no se pudo reservar el espacio de memoria) se iguala el primer arreglo con el segundo. Lo que hace después para que se siga escribiendo en el mismo arreglo es que al igualarlos los elementos pasan a ser del otro y se continúan pidiendo hasta que la variable cont con su valor definido anteriormente no deje de ser menor a 2 veces la variable num (como se definió antes) y así va guardando los números que ingresemos en cada casilla del arreglo. Para terminar el programa se imprime el arreglo mediante otro ciclo de repetición for que va mostrando el valor contenido en cada espacio y cuando esto se acaba, el programa concluye liberando la memoria reservada de manera dinámica con la función free.

Con esto podemos ver que justamente gracias a estas funciones podemos ocupar solo el espacio de memoria que vayamos a utilizar y agregar más solo en caso de que se requiera, lo cual nos dará muchos beneficios en el tiempo de ejecución de un programa pues muchas veces sin saberlo al ejecutar el programa se reservan espacios de memoria que muy pocas veces o jamás se utilizan, lo hace que ya no sea tan eficiente.

**CONCLUSIONES**

Estos programas me sirvieron muchos para conocer que es la memoria dinámica y como reservarla en lenguaje C con tres funciones (malloc, calloc y reallooc) las cuales tienen diferentes usos, pero con las que también se puede almacenar información dentro de estos elementos reservados, en pocas palabras, sirven para la asignación de memoria. **Aquí pudimos ver que malloc imprime basura si no es la primera vez que se inicializa, sin embargo, calloc no lo hace pues siempre se inicializa en 0.**

Esta es una de las practicas que no ha tenido errores en su código lo cual nos facilita mucho su compilación. En el caso de esta practica vemos como podemos mejorar el tiempo de ejecución o hacer una optimización en la memoria, lo cual nos vas a permitir limpiar un espacio de memoria, reservarlo e inicializarlo en cero o redimensionar un espacio aumentándolo todo esto de manera dinámica para no desperdiciar espacios de memoria.

Estas funciones y todo lo hemos estado viendo nos ayuda nos ayuda muchos al momento de querer optimizar una programa o ente caso la memoria que se ocupa durante la ejecución del mismo, lo cual hace que este sea mucho más eficiente y así nosotros vamos adquiriendo más conocimientos y más habilidades en el desarrollo del lenguaje C, pero también en otros tipos de lenguajes, lo cual nos va a permitir ir avanzando paso a paso en su desarrollo pues muchas personas ya no llegan a este nivel de programación.

**BIBLIOGRAFÍA**

* Khan, R. (s.f.). *DYNAMIC MEMORY ALLOCATION: DEFINITION & EXAMPLE*. Recuperado el 29 de febrero de 2020, de Study: <https://study.com/academy/lesson/dynamic-memory-allocation-definition-example.html#lesson>

**GITHUB:**

https://github.com/miriamyi01/EDA-I