

ANÁLISIS Y DOCUMENTACIÓN DE LA MEMORIA DINÁMICA EN C



Alumno: Luis Daniel Mendoza Rodríguez
Materia: Introducción a la programación
Mtro: Dra. María de los Ángeles Salazar Olmos

Análisis y documentación de la memoria dinámica en C

Memoria dinámica

La memoria dinámica es un mecanismo que permite al programador reservar y liberar memoria durante la ejecución del programa en vez de hacerlo estáticamente en tiempo de compilación. Por ejemplo cuando asignas un array con determinado longitud, al momento de compilación, se reserva cuanta memoria se requerirá para ese array, en base al tipo de dato que puede almacenar, pero la memoria dinámica permite que el espacio reservado para las variables cambie durante la ejecución y esto es útil si no sabes exactamente cuánto espacios necesitaras porque depende de información proporcionada por el usuario.

La memoria dinámica se gestiona a través de la librería `<stdlib.h>` con funciones como `malloc()`, `calloc()`, `realloc()` y `free()`

Introducción al video

El video muestra como se utiliza las funciones `malloc()` y `free()` para la gestión de la memoria dinámica, con un ejemplo sencillo de un array, cuya longitud será entregada por el usuario a través de `scanf()`



#10



Proceso del ejercicio:

1.- Inicio del boilerplate del programa

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     return 0;
6 }
```

Compile Log | Debug | Find Results

Set: 0 | Lines: 6 | Length: 71 | Insert | Done parsing in 0.172 seconds

10:42 p. m.
18/10/2023

Aquí inicia el video diciendo que malloc sirve para solicitar asignar memoria, y recibe como parámetro la cantidad de memoria que quieres almacenar, en este caso se refiere al ejemplo, si quisieras almacenar un array con 4,000 espacios flotantes, no bastaría con pedirle 4,000 porque lo que recibe como parámetro es un valor en bytes, por lo que toma el tamaño de un float `sizeof(float)` y lo multiplica por la cantidad de espacios del array que quiere.

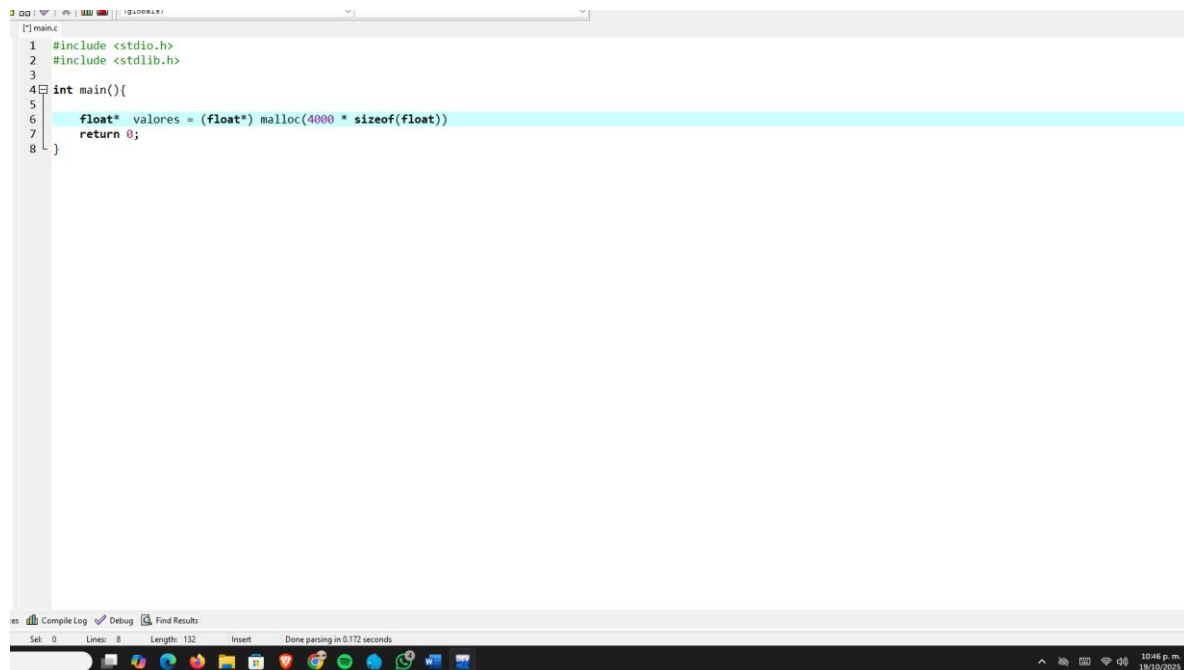
```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     malloc(4000 * sizeof(float))
6     return 0;
7 }
8
```

Compile Log | Debug | Find Results

Set: 0 | Lines: 8 | Length: 105 | Insert | Done parsing in 0.172 seconds

10:44 p. m.
18/10/2023

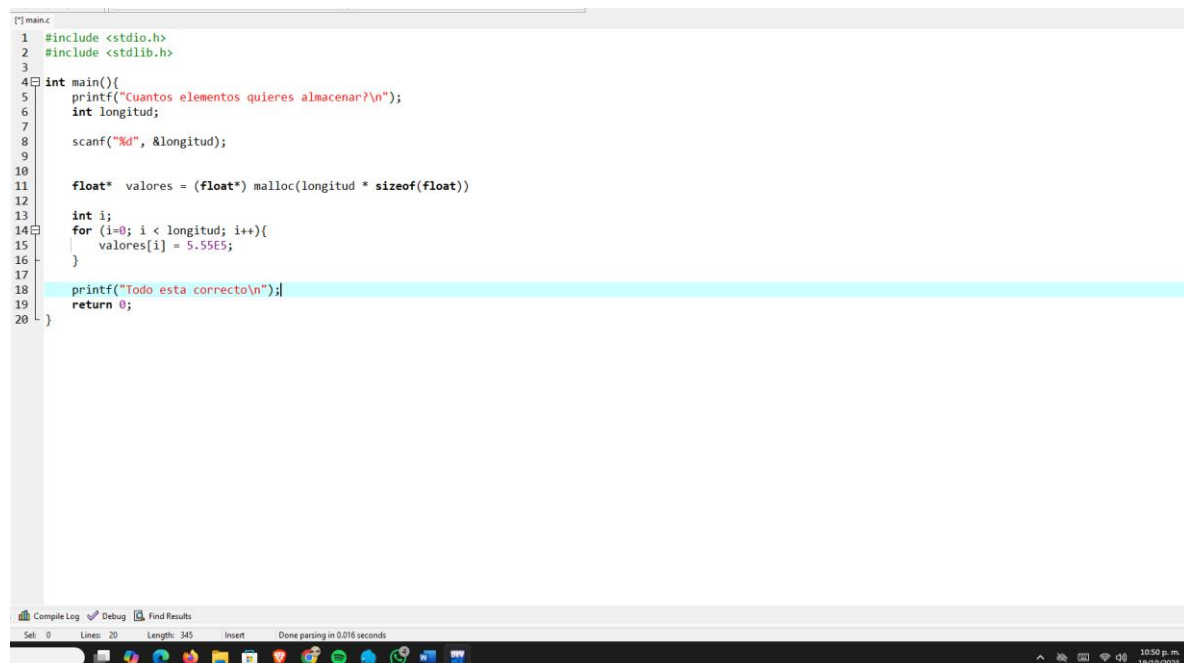
Luego explica que malloc regresa un puntero de void*, que puede ser cualquier cosa, entonces lo que hay que hacer es guardar lo que malloc regresa en un puntero del valor que necesitamos y antes hacerle un cast (una conversión de tipo) por eso es `(float*) malloc`....



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     float* valores = (float*) malloc(4000 * sizeof(float))
6     return 0;
7 }
8 }
```

The screenshot shows a code editor with a C program. The program includes `<stdio.h>` and `<stdlib.h>`. The `main` function declares a `float*` pointer named `valores` and assigns it the result of `(float*) malloc(4000 * sizeof(float))`. The line is highlighted in light blue. The editor's status bar at the bottom indicates 8 lines and 132 characters.

En esta segunda parte, remplazo el 4,000 por una variable llamada longitud que permite al usuario indicarle al programa cuantos valores quiere que almacene, porque después de que el usuario ingrese el valor que se guarda en “longitud” luego se remplaza en la función malloc el 4,000, para asignar solamente los espacios en memoria que el usuario solicita



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     printf("Cuantos elementos quieres almacenar?\n");
6     int longitud;
7
8     scanf("%d", &longitud);
9
10
11     float* valores = (float*) malloc(longitud * sizeof(float))
12
13     int i;
14     for (i=0; i < longitud; i++){
15         valores[i] = 5.55E5;
16     }
17
18     printf("Todo esta correcto\n");
19     return 0;
20 }
```

The screenshot shows a more complex C program. It prompts the user for the number of elements to store using `printf` and `scanf`. The `malloc` call now uses the `longitud` variable. A `for` loop initializes the array with the value `5.55E5`. The line `printf("Todo esta correcto\n");` is highlighted in light blue. The editor's status bar at the bottom indicates 20 lines and 345 characters.

Me faltó un ;

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     printf("Cuantos elementos quieres almacenar?\n");
6     int longitud;
7
8     scanf("%d", &longitud);
9
10    float* valores = (float*) malloc(longitud * sizeof(float))
11
12    int i;
13    for (i=0; i < longitud; i++){
14        valores[i] = 5.5555;
15    }
16
17    printf("Todo esta correcto\n");
18    return 0;
19 }
20 }
```

es Compile Log Debug Find Results Close

Message

\\Documents\\programas\\facultad\\USAI... In function 'main':
\\Documents\\programas\\facultad\\USAI... [Error] expected ';' or ':' before 'int'
\\Documents\\programas\\facultad\\USAI... [Error] undeclared (first use in this function)
\\Documents\\programas\\facultad\\USAI... [Note] each undeclared identifier is reported only once for each function it appears in

Señal: 0 Lines: 20 Length: 345 Insert Done parsing in 0.015 seconds

10:52 p. m.
19/10/2023

Ejecución una vez arreglado el problema:

Formato Fuente Párrafo Estilos Edición Voz Confiden

Normal Sin espaciado TITULO 1 TITULO 2 TITULO 3 Reemplazar Seleccionar Dictar Confiden

C:\Users\juism\Documents\pr... + -

Cuantos elementos quieres almacenar?
100
Todo esta correcto

Process exited after 2.269 seconds with return value 0
Presione una tecla para continuar . . .

\\Documents\\programas\\facultad\\USAI... In function 'main':
\\Documents\\programas\\facultad\\USAI... [Error] expected ';' or ':' before 'int'
\\Documents\\programas\\facultad\\USAI... [Error] undeclared (first use in this function)
\\Documents\\programas\\facultad\\USAI... [Note] each undeclared identifier is reported only once for each function it appears in

Señal: 0 Lines: 20 Length: 345 Insert Done parsing in 0.015 seconds

En esta ultima parte, agrega al final la liberación de memoria con la función free, aclara que una vez liberada la memoria no se puede acceder más delante de la historia.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     printf("Cuantos elementos quieres almacenar?\n");
6     int longitud;
7
8     scanf("%d", &longitud);
9
10
11     float* valores = (float*) malloc(longitud * sizeof(float));
12     if(valores == NULL){
13         printf("No tienes suficiente memoria\n");
14         return 1;
15     }
16     int i;
17     for (i=0; i < longitud; i++){
18         valores[i] = 5.55E5;
19     }
20
21     printf("Todo esta correcto\n");
22     free(valores);
23     return 0;
24 }
```

Compile Log | Debug | Find Results | Close

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\luism\Documents\programas\facultad\USAI\main.exe
- Output Size: 128.096609375 KiB
- Compilation Time: 0.64s

Set 0 Lines: 24 Length: 445 Insert Done parsing in 0.016 seconds

10:55 p.m. 19/10/2023

Agregue comentario para organizar mejor. Y al final

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     //solicitar una longitud dinámica al usuario
6     printf("Cuantos elementos quieres almacenar?\n");
7     int longitud;
8     scanf("%d", &longitud);
9
10    //asignar la memoria dinamica, en este caso un array de flotantes del tamaño que el usuario solicito
11    float* valores = (float*) malloc(longitud * sizeof(float));
12    //validación para ver si malloc devolvio algo valida
13    if(valores == NULL){
14        printf("No tienes suficiente memoria\n");
15        return 1;
16    }
17
18    //probar la nueva longitud del méxcio
19    int i;
20    for (i=0; i < longitud; i++){
21        valores[i] = 5.55E5;
22    }
23
24    printf("Todo esta correcto\n");
25    //limpiar memoria
26    free(valores);
27    return 0;
28 }
```

Compile Log | Debug | Find Results | Close

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\luism\Documents\programas\facultad\USAI\main.exe
- Output Size: 128.096609375 KiB
- Compilation Time: 0.64s

Set 0 Lines: 28 Length: 707 Insert Done parsing in 0.016 seconds

10:58 p.m. 19/10/2023

Comentarios finales

Respecto al ejercicio creo que es bastante claro y sencillo, lo que me llamo más la atención es que en lenguajes más modernos y de alto nivel como javascript, php y Python, esta gestión de la memoria corre por parte del interprete (que si es Python por ejemplo, es c) basta con declarar una variable de tipo array

En PHP:

```
$usuarios;
```

```
$usuarios[] = ["nombre" => Luis Daniel, "apellido" => Mendoza]
```

Al declarar `$usuarios`; se reservar un espacio en memoria que un no se conoce nada, y al hacer la asignación `$usuarios[]`, le indicas que quieres que `$usuarios` sea un array y que guardaras algo en la siguiente posición disponible, en este caso 1.

Sin embargo en C es necesario asignar el tamaño de la memoria en bytes y gestionarla porque la memoria dinámica se almacena en el heap, que no es especialmente grande.