



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M.I. Edgar Tista García

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 1

No de Práctica(s): 4

Integrante(s): Díaz Hernández Marcos Bryan

*No. de Equipo de
cómputo empleado:*

No. de Lista o Brigada: 9

Semestre: 2020-2

Fecha de entrega: 07 de marzo de 2020

Observaciones:

CALIFICACIÓN: _____

Objetivo de la práctica

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

Introducción

Esta práctica consiste en el uso de memoria dinámica para poder aplicarla a conceptos previos como estructuras, y funciones, los programas están elaborados con la intención de ser comprensibles para mi y para el usuario.

Ejercicios de la guía de laboratorio

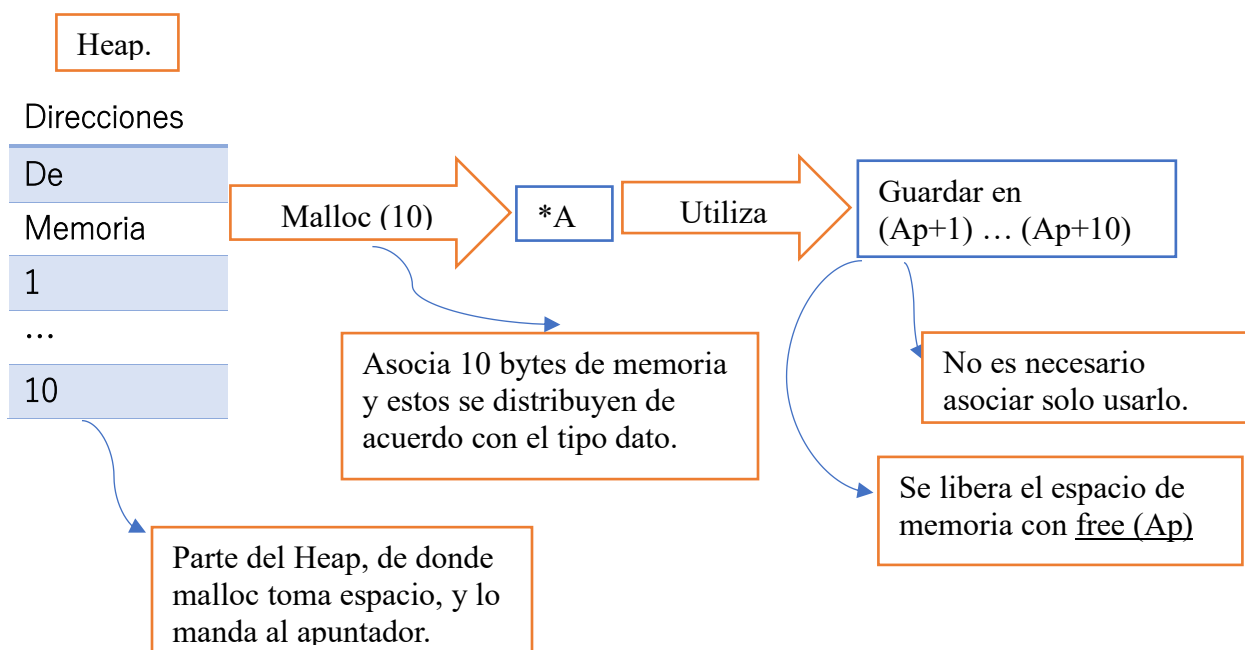
- Primer ejemplo: Código (malloc).

En este código se utiliza la función malloc que permite el uso de memoria dinámica en los programas de c, esta se caracteriza por utilizar un apuntador para apuntar a un conjunto de memoria disponible del heap. Es necesario para esta función el indicar la cantidad de bytes del tipo de dato e indicar cuantos bytes de ese dato son necesarios.

Además, se debe indicar la conversión del tipo de dato ya que malloc regresa un tipo de dato void, que no está asociado a algún tipo de dato.

```
scanf(" %d", &num);  
arreglo=(int *) malloc(num*sizeof(int));
```

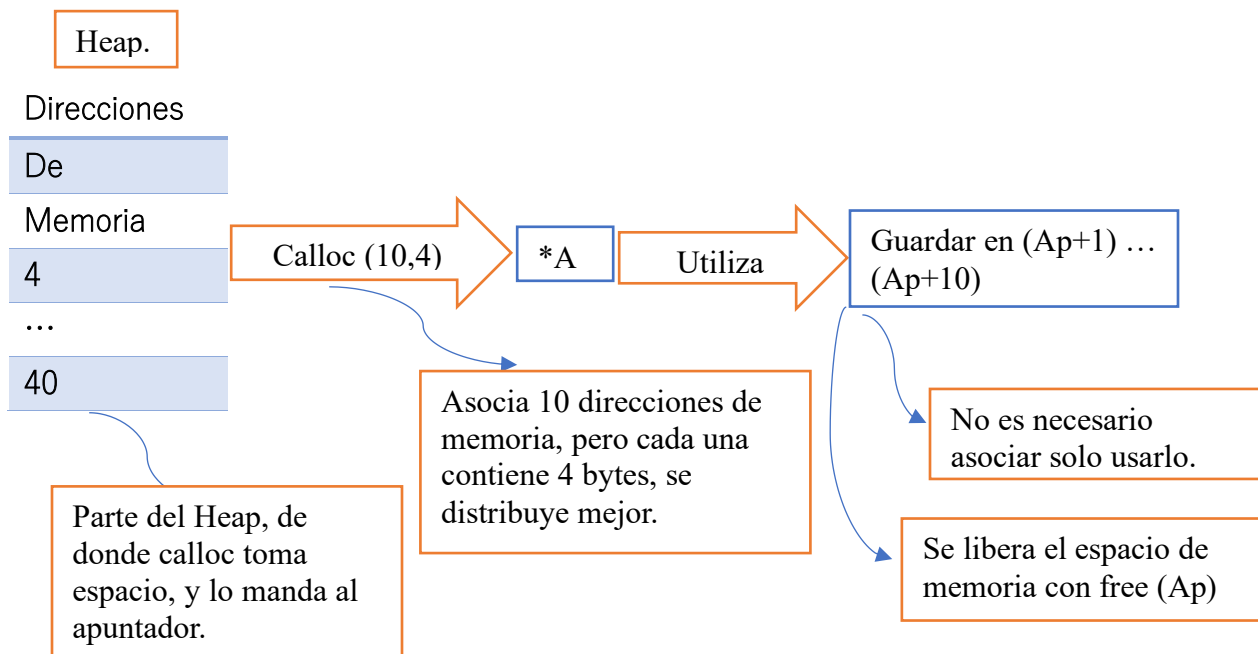
Diagrama de funcionamiento.



- Segundo ejemplo: Código (calloc).

En este código se emplea conceptualmente el mismo hecho de utilizar una función llamada calloc para poder asignar memoria dinámica a un apuntador y después utilizarlo, en este caso los argumentos de la función son dos, uno corresponde al número de bytes y el segundo corresponde a la cantidad de bytes por cada uno de los bytes que se colocaron primero o en otras palabras corresponden a la cantidad de bytes de acuerdo con el tipo de dato.

Diagrama de funcionamiento.

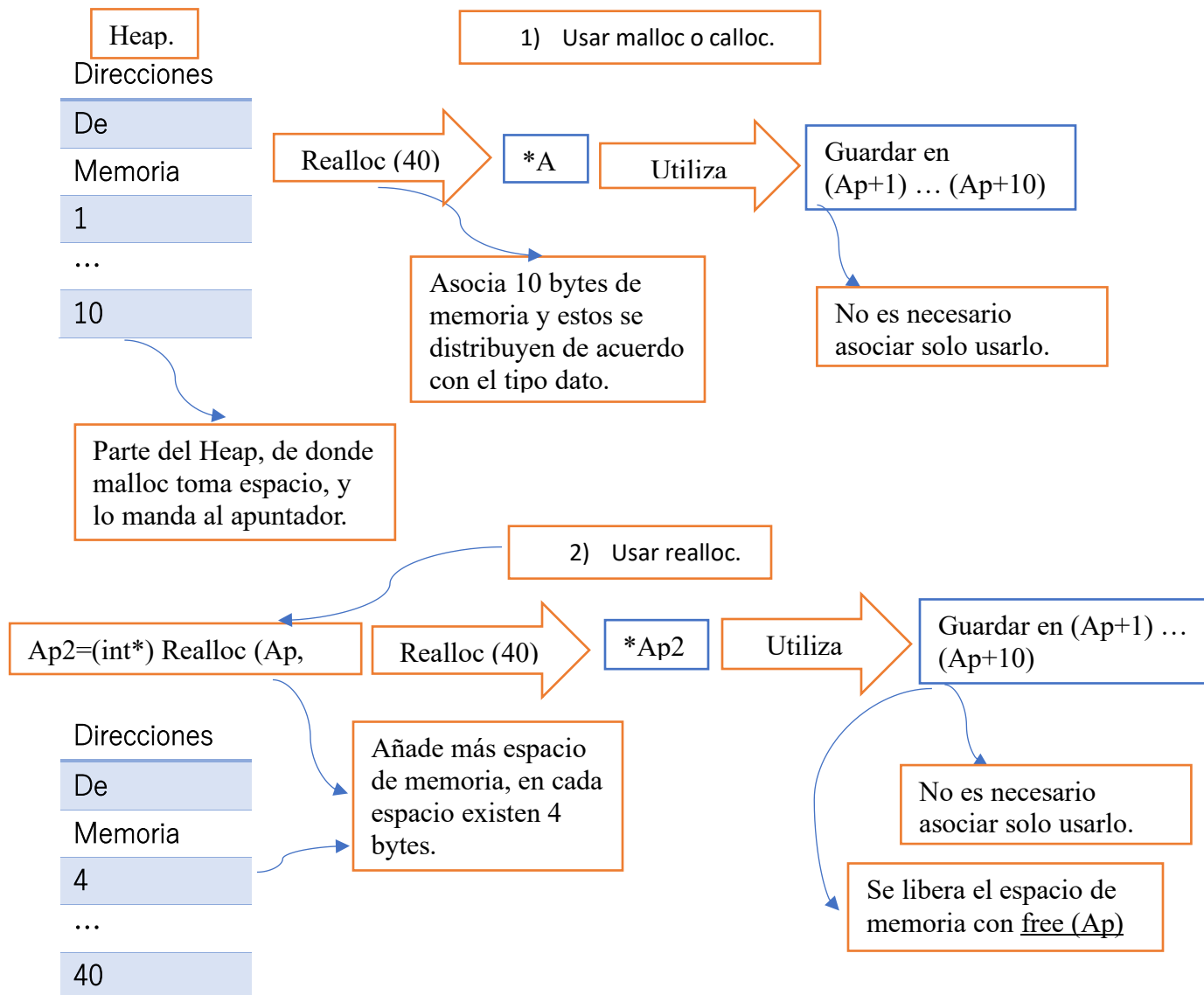


- Tercer ejemplo: Código (realloc).

En este código se utiliza una tercera función de memoria dinámica que permite aumentar el tamaño de bytes que se reservan con las funciones malloc y calloc, lo que ayuda a tener a disposición más memoria para trabajar, la forma en que funciona realloc es por medio de dos apuntadores, el primero corresponde al apuntador que asocio malloc o calloc a los bytes de memoria, el segundo es un apuntador que almacenara al primero, además el espacio extra asignado.

Se sigue una consideración para utilizar realloc es que necesita que se indique todos los bytes que se van a necesitar más los que ya se tenían apartados, de lo contrario podría reducir la cantidad de bytes de memoria.

Diagrama de funcionamiento.



Pregunta.

- Con la instrucción `(int *) malloc (sizeof(int)*3)`, sin el operador `(*)`, ¿funciona?

No funciona porque detecta únicamente el valor como constante y es necesario indicar que hace la constante.

```
scanf("%d", &num);
arreglo=(int *) malloc(sizeof(int)*3);
if(arreglo!=NULL) printf("Ver ");
public void *__cdecl malloc(size_t_Size)
```

Ejercicios del Laboratorio

- Ejercicio 1

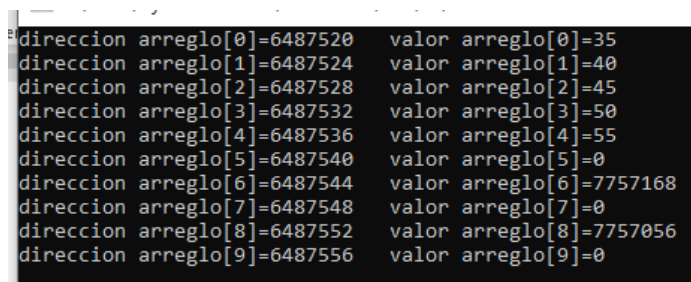
Aclaración.

Algunas de las capturas de pantalla son del laboratorio, debido a que considero que las instrucciones que coloqué en el programa no deberían ir en esa posición, por lo que, volví a realizar el código para ver correctamente el orden y lo que ocurre.

- Preguntas del ejercicio.

1.- Captura la pantalla con la ejecución del programa y explica de manera detallada, los resultados que se observan.

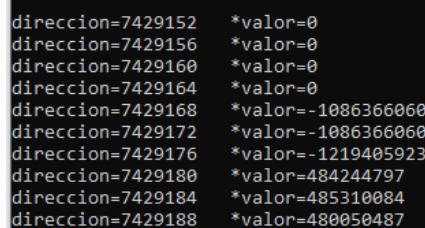
La primera sección de la salida tiene las direcciones de memoria del arreglo, son contiguas y cada una es de 4 bytes, además se imprimió el contenido de las direcciones de memoria. Se mostraron los valores con los que estaba inicializado el arreglo, y también se mostro el contenido de direcciones siguientes del arreglo que contenían valores desconocidos (Imagen 1).



```
direccion arreglo[0]=6487520 valor arreglo[0]=35
direccion arreglo[1]=6487524 valor arreglo[1]=40
direccion arreglo[2]=6487528 valor arreglo[2]=45
direccion arreglo[3]=6487532 valor arreglo[3]=50
direccion arreglo[4]=6487536 valor arreglo[4]=55
direccion arreglo[5]=6487540 valor arreglo[5]=0
direccion arreglo[6]=6487544 valor arreglo[6]=7757168
direccion arreglo[7]=6487548 valor arreglo[7]=0
direccion arreglo[8]=6487552 valor arreglo[8]=7757056
direccion arreglo[9]=6487556 valor arreglo[9]=0
```

Imagen 1.

La segunda sección imprimía las direcciones de memoria signadas al apuntador ptr, las cuales al estar asignadas por malloc. Mantienen el contenido de las direcciones de memoria, por ello cuando se imprimen mantiene valores desconocidos (Imagen 2).



```
direccion=7429152 *valor=0
direccion=7429156 *valor=0
direccion=7429160 *valor=0
direccion=7429164 *valor=0
direccion=7429168 *valor=-1086366060
direccion=7429172 *valor=-1086366060
direccion=7429176 *valor=-1219405923
direccion=7429180 *valor=484244797
direccion=7429184 *valor=485310084
direccion=7429188 *valor=480050487
```

Imagen 2.

2.- Modifica la asignación inicial de “*ptr” de tal modo que ahora se utilice la función calloc, compila, ejecuta y explica la diferencia con la ejecución previa.

La diferencia que existe entre la asignación con malloc y calloc tiene que ver con los parámetros que se necesitan para llevar a cabo la asignación de memoria dinámica, en el caso de malloc solo se necesita un parámetro, y en calloc se necesitan dos parámetros además que calloc limpia las localidades de memoria (Imagen 3).

```
int *ptr=(int *) malloc(10);  
int *ptr3=(int *) calloc(10, sizeof(int));  
int *Ap=(int *) calloc(10, sizeof(int));
```

Imagen 3.

Al usar calloc limpia el contenido de las localidades de memoria (Imagen 4).

```
direccion=10116128 *valor=0  
direccion=10116132 *valor=0  
direccion=10116136 *valor=0  
direccion=10116140 *valor=0  
direccion=10116144 *valor=0  
direccion=10116148 *valor=0  
direccion=10116152 *valor=0  
direccion=10116156 *valor=0  
direccion=10116160 *valor=0  
direccion=10116164 *valor=0
```

Imagen 4.

3.- Utiliza realloc para redimensionar “ptr” con un nuevo tamaño.

- ¿Si se usa realloc en el mismo apuntador funciona?

Al redimensionar el tamaño de “ptr” no hay ningún error, por ello el contenido que este ya tenía se conserva además de que agrega las localidades de memoria, al parecer el realloc asigna las localidades de nuevas, pero no las limpia, por ello estas se quedan con el contenido que ya tenían (Imagen 5).

- ¿Cuándo se asigna más memoria dinámica con realloc este busca dentro de las mismas localidades de memoria o se mueve a nuevas localidades?

Cuando se utiliza realloc este busca dentro de las mismas localidades para tener secuencia, entre las localidades de memoria ya que al apuntador le asigna espacios de memoria fijos a los cuales apunte y pueda acceder a ellos de forma secuenciada (Imagen 5).

```

Redimensionados
Direccion de memoria de Ptr:9133088, Contenido:0
Direccion de memoria de Ptr:9133092, Contenido:5
Direccion de memoria de Ptr:9133096, Contenido:10
Direccion de memoria de Ptr:9133100, Contenido:15
Direccion de memoria de Ptr:9133104, Contenido:20
Direccion de memoria de Ptr:9133108, Contenido:25
Direccion de memoria de Ptr:9133112, Contenido:30
Direccion de memoria de Ptr:9133116, Contenido:35
Direccion de memoria de Ptr:9133120, Contenido:40
Direccion de memoria de Ptr:9133124, Contenido:45
Direccion de memoria de Ptr:9133128, Contenido:0
Direccion de memoria de Ptr:9133132, Contenido:0
Direccion de memoria de Ptr:9133136, Contenido:-678424198
Direccion de memoria de Ptr:9133140, Contenido:-678424198
Direccion de memoria de Ptr:9133144, Contenido:-536995725
Direccion de memoria de Ptr:9133148, Contenido:1951642317
Direccion de memoria de Ptr:9133152, Contenido:1958262919
Direccion de memoria de Ptr:9133156, Contenido:1949545177
Direccion de memoria de Ptr:9133160, Contenido:1949545177
Direccion de memoria de Ptr:9133164, Contenido:1949545177

```

Imagen 5.

- Indica si hay diferencias entre asignar realloc al mismo apuntador (ptr) o a uno nuevo (ptr3)

En realidad, no existe mucha diferencia entre usar en el mismo apuntador o asignar más memoria en uno nuevo, pero si se busca la eficiencia en un programa, lo ideal es que sea sobre el mismo apuntador, además de que se hacen menos instrucciones y se consume menos memoria, aunque sea un poco menos de memoria podría ocuparse esa memoria en otras variable o procesos (Imagen 6).

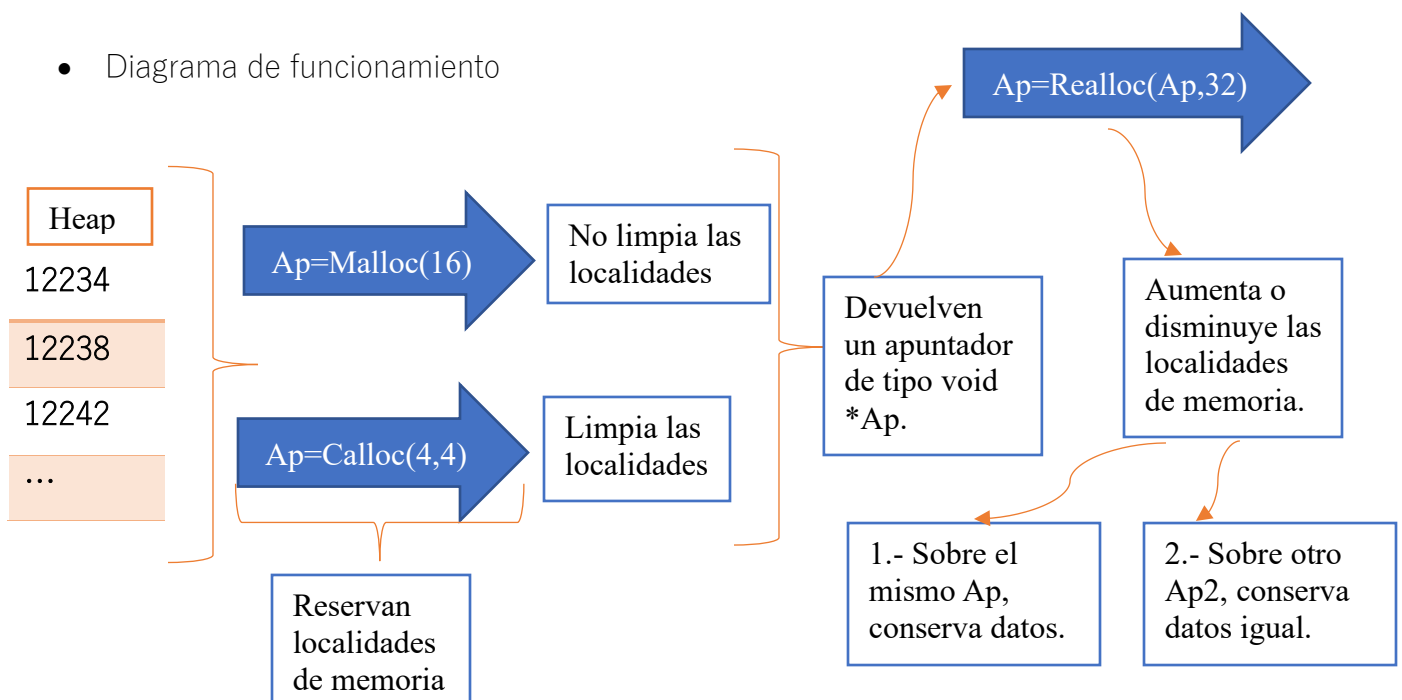
```

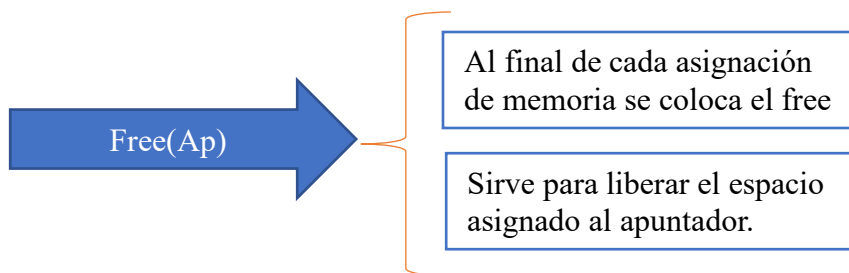
Direccion de memoria Ptr3:9133088, Contenido:0
Direccion de memoria Ptr3:9133092, Contenido:5
Direccion de memoria Ptr3:9133096, Contenido:10
Direccion de memoria Ptr3:9133100, Contenido:15
Direccion de memoria Ptr3:9133104, Contenido:20
Direccion de memoria Ptr3:9133108, Contenido:25
Direccion de memoria Ptr3:9133112, Contenido:30
Direccion de memoria Ptr3:9133116, Contenido:35
Direccion de memoria Ptr3:9133120, Contenido:40
Direccion de memoria Ptr3:9133124, Contenido:45
Direccion de memoria Ptr3:9133128, Contenido:0
Direccion de memoria Ptr3:9133132, Contenido:0
Direccion de memoria Ptr3:9133136, Contenido:-678424198
Direccion de memoria Ptr3:9133140, Contenido:-678424198
Direccion de memoria Ptr3:9133144, Contenido:-536995725
Direccion de memoria Ptr3:9133148, Contenido:1951642317
Direccion de memoria Ptr3:9133152, Contenido:1958262919
Direccion de memoria Ptr3:9133156, Contenido:1949545177
Direccion de memoria Ptr3:9133160, Contenido:1949545177
Direccion de memoria Ptr3:9133164, Contenido:1949545177

```

Imagen 6.

- Diagrama de funcionamiento





- Relación con teoría.

Lo visto en clase es lo que hicimos en este primer ejercicio además todas las características son las vistas en clase, sin embargo, había que comprobar algunas, como el uso de realloc sobre el mismo apuntador. Todo lo visto en este ejercicio se relaciona con lo visto en clase teórica por ello este ejercicio es base para resolver los demás.

The screenshot shows a C++ IDE with a project named "Ejercicio1.c". The code in the editor is as follows:

```
25 for(variable=0; variable<10; variable++){
26     *(ptr+ variable)=variable*5;
27 }
28
29 for(cont=0; cont<10; cont++){
30     printf("Direccion de memoria de Ptr: %d, Contenido: %d\n", ptr, *(ptr+cont));
31 }
32 printf("\n");
33
34 ptr=(int *)realloc(ptr, 20);
35 ptr3=(int *)realloc(ptr, 20);
36
37 printf("Redimensionados\n");
38
39 for(cont=0; cont<20; cont++){
40     printf("Direccion de memoria de Ptr: %d, Contenido: %d\n", ptr, *(ptr+cont));
41 }
42 printf("\n");
43
44 for(cont=0; cont<20; cont++){
45     printf("Direccion de memoria Ptr3: %d, Contenido: %d\n", ptr3, *(ptr3+cont));
46 }
47
48 }
```

The output window shows the following results:

```
Redimensionados
Direccion de memoria de Ptr: 9133088, Contenido: 0
Direccion de memoria de Ptr: 9133092, Contenido: 5
Direccion de memoria de Ptr: 9133096, Contenido: 10
Direccion de memoria de Ptr: 9133100, Contenido: 15
Direccion de memoria de Ptr: 9133104, Contenido: 20
Direccion de memoria de Ptr: 9133108, Contenido: 25
Direccion de memoria de Ptr: 9133112, Contenido: 30
Direccion de memoria de Ptr: 9133116, Contenido: 35
Direccion de memoria de Ptr: 9133120, Contenido: 40
Direccion de memoria de Ptr: 9133124, Contenido: 45
Direccion de memoria de Ptr: 9133128, Contenido: 0
Direccion de memoria de Ptr: 9133132, Contenido: 0
Direccion de memoria de Ptr: 9133136, Contenido: -678424198
Direccion de memoria de Ptr: 9133140, Contenido: -678424198
Direccion de memoria de Ptr: 9133144, Contenido: -536995725
Direccion de memoria de Ptr: 9133148, Contenido: 1951642317
Direccion de memoria de Ptr: 9133152, Contenido: 1958262919
Direccion de memoria de Ptr: 9133156, Contenido: 1949545177
Direccion de memoria de Ptr: 9133160, Contenido: 1949545177
Direccion de memoria de Ptr: 9133164, Contenido: 1949545177
Direccion de memoria Ptr3: 9133088, Contenido: 0
Direccion de memoria Ptr3: 9133092, Contenido: 5
Direccion de memoria Ptr3: 9133096, Contenido: 10
Direccion de memoria Ptr3: 9133100, Contenido: 15
Direccion de memoria Ptr3: 9133104, Contenido: 20
Direccion de memoria Ptr3: 9133108, Contenido: 25
Direccion de memoria Ptr3: 9133112, Contenido: 30
Direccion de memoria Ptr3: 9133116, Contenido: 35
Direccion de memoria Ptr3: 9133120, Contenido: 40
Direccion de memoria Ptr3: 9133124, Contenido: 45
Direccion de memoria Ptr3: 9133128, Contenido: 0
Direccion de memoria Ptr3: 9133132, Contenido: 0
Direccion de memoria Ptr3: 9133136, Contenido: -678424198
Direccion de memoria Ptr3: 9133140, Contenido: -678424198
Direccion de memoria Ptr3: 9133144, Contenido: -536995725
Direccion de memoria Ptr3: 9133148, Contenido: 1951642317
Direccion de memoria Ptr3: 9133152, Contenido: 1958262919
Direccion de memoria Ptr3: 9133156, Contenido: 1949545177
Direccion de memoria Ptr3: 9133160, Contenido: 1949545177
Direccion de memoria Ptr3: 9133164, Contenido: 1949545177
```

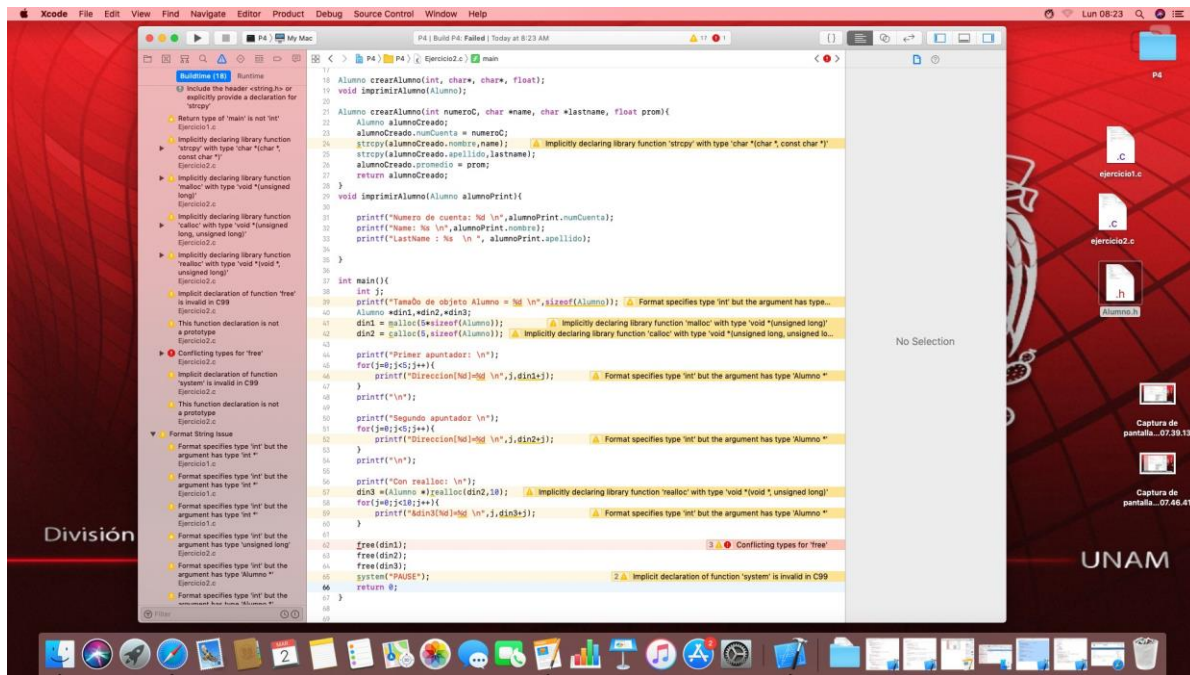
The compilation results window shows the following output:

```
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Bryan Hernández\Documents\EDA\P4\Díaz Hernández Marcos G1 P4 V2\Ejercicio1.c
- Output Size: 130.115234375 KiB
- Compilation Time: 0.55s
```

• Ejercicio 2

En el laboratorio surgió el siguiente problema:

Al instante de compilar el código se detectaba un error en el free, no lo identificaba el compilador, pero al correr el programa en casa, lo corrió sin problema, fue algo relacionado con la terminal de xcode, a veces pasa que no reconoce ciertas funciones, o incluso un código completo por lo que, recurro a copiar el código en otro archivo y normalmente esa solución funciona. Sin embargo, en el caso del laboratorio no sirvió, y me quede atorado en esa parte, por la falta de tiempo y por que no sabia como solucionar ese error, que al final tenia que ver mas con la terminal que con el código.



- Dificultades en el código

No pude terminar el ejercicio por los errores que marcaba e intentar solucionarlos me llevaría mucho tiempo, y ya estaba por terminal el tiempo de la practica por lo que decidí intentar compilarlo el alumno.h con el ejercicio 2, pero aun así no funcionaba, por ello decidí enviar lo que tenia y resolver los dos problemas restantes en casa.

El dilema que me tomo más tiempo resolver se relaciona con las funciones que devuelven un valor, porque cuando lo intente aplicar a las estructuras regresa un arreglo de estructuras, para guardarlo en otro arreglo de estructuras, lo que sucedía era que no registraba ningún valor que insertaba, y al imprimir los alumnos registrados, en pantalla aparecían valores extrañísimos, por lo que opte por hacer paso por referencia.

Algo curioso que hice fue correr el programa “Alumno.h”, cuando lo hice la siguiente instrucción me impedía correr solo esa parte de Ejercicio 2, y si corría ejercicio 2, no volvía a aparecer (Imagen 1).

C:\Users\Bryan Hernández\Documents\EDA\P4\Díaz Her... [Error] 'strcpy' was not declared in this scope

Imagen 1.

- ¿Cómo lo resolví?

Como mencionaba en la sección de dificultades, opte por hacer paso por referencia. El paso por referencia lo usé para pasar las direcciones del apuntador “din3” y poder empezar a guardar los valores en cada una de las localidades asignadas (Imagen 2).

```
LlenarAlumnos(din3);  
Imprimir(din3);
```

Imagen 2.

- Preguntas del ejercicio.

1.- Explica los resultados de la salida del programa.

La primera línea de la salida corresponde a la cantidad de bytes que ocupa la estructura Alumno. Los bytes de almacenamiento no necesariamente son continuos por ello igual hacer igualaciones entre estructuras no es recomendable (Imagen 3).

```
C:\Users\Bryan Hernández\Documents\EDA\P4\Díaz Hernández A  
Tamaño de objeto Alumno = 88  
Primer apuntador:
```

Imagen 3.

El primer conjunto de líneas del primer apuntador muestra las direcciones de memoria que fueron asignadas al apuntador “din1”, por medio de la función malloc, donde cada elemento tiene 88 bytes, porque el tipo de dato corresponde a la estructura Alumno (Imagen 4).

```
Primer apuntador:  
Direccion[0]=1731360  
Direccion[1]=1731448  
Direccion[2]=1731536  
Direccion[3]=1731624  
Direccion[4]=1731712
```

Imagen 4.

El segundo conjunto de líneas del segundo apuntador muestra las direcciones de memoria que fueron asignadas al apuntador “din2”, por medio de la función calloc, donde cada elemento tiene 88 bytes, porque el tipo de dato corresponde a la estructura Alumno (Imagen 5).

```
Segundo apuntador  
Direccion[0]=1731808  
Direccion[1]=1731896  
Direccion[2]=1731984  
Direccion[3]=1732072  
Direccion[4]=1732160
```

Imagen 5.

El tercer conjunto de líneas es de un tercer apuntador donde se redimensiono al apuntador “din 2”, por medio de la función realloc, asignado 5 espacios más, donde se ve la linealidad de la búsqueda de realloc, que busca en las mismas localidades de memoria. Muestras las localidades de memoria a las que apunta “din3” (Imagen 6).

```

Con realloc:
&din3[0]=1731808
&din3[1]=1731896
&din3[2]=1731984
&din3[3]=1732072
&din3[4]=1732160
&din3[5]=1732248
&din3[6]=1732336
&din3[7]=1732424
&din3[8]=1732512
&din3[9]=1732600

```

Imagen 6.

Las localidades de memoria tienen saltos de 88 bytes, esto es debido a la forma en que se asignó la memoria dinámica en los apuntadores, donde se pedía reservar espacio para varias estructuras, debido a que cada estructura ocupa 88 bytes, se crea ese salto en las localidades de memoria (Imagen 7).

```

Alumno *din1, *din2, *din3;
din1 = malloc(5*sizeof(Alumno));
din2 = calloc(5, sizeof(Alumno));

```

Imagen 7.

2.- ¿Cuál es el tamaño de la estructura alumno?

Son 88 bytes de memoria, significa que cada variable del tipo Alumno ocupa 88 bytes (Imagen 3).

3.- Explica de que formase podría liberar la memoria reservada por la función calloc o malloc sin utilizar la función free.

Una forma en la que se puede liberar la memoria es utilizando realloc en el mismo apuntador, solo haciendo la asignación de bytes sobre 0, lo que borraría lo que existe y guardaría cero bytes en el mismo apuntador (Imagen 8).

Eso aplicaría para todos los apuntadores que se quisieran liberar, sin la necesidad de usar free, aunque ocupas más código, la función realloc sirve bastante bien para eliminar memoria asignada.

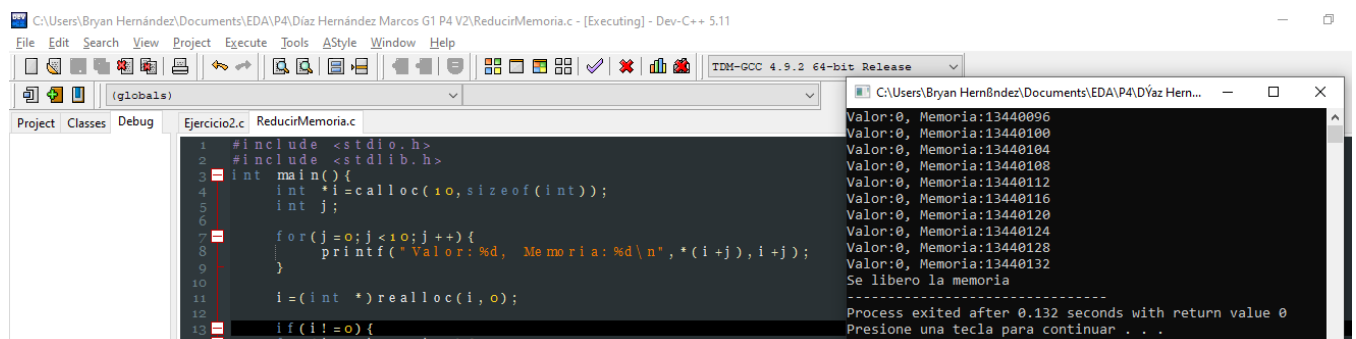
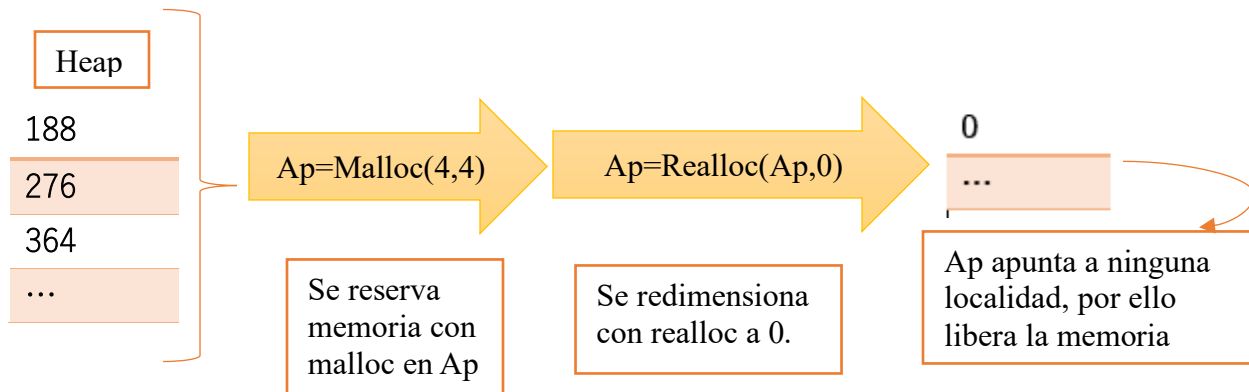


Imagen 8.

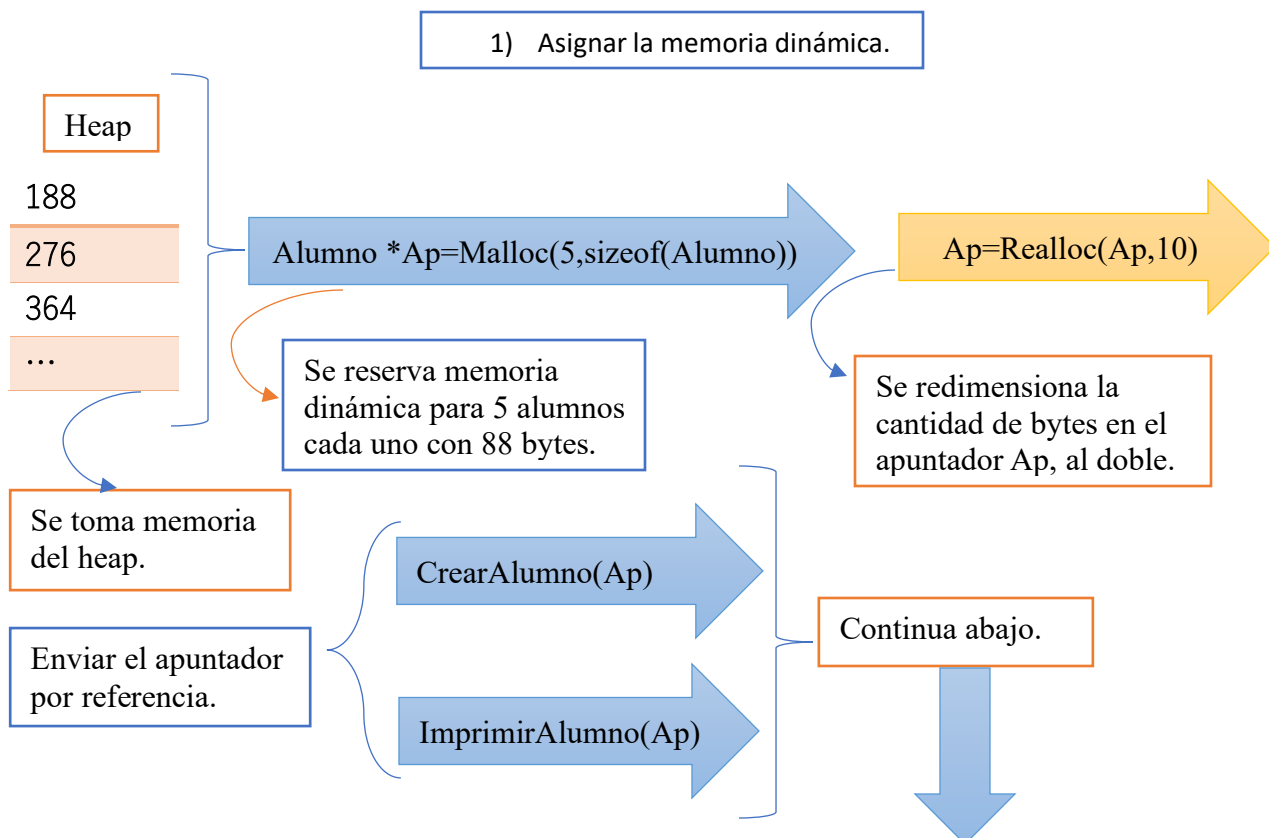
- Diagrama de funcionamiento Punto 3.

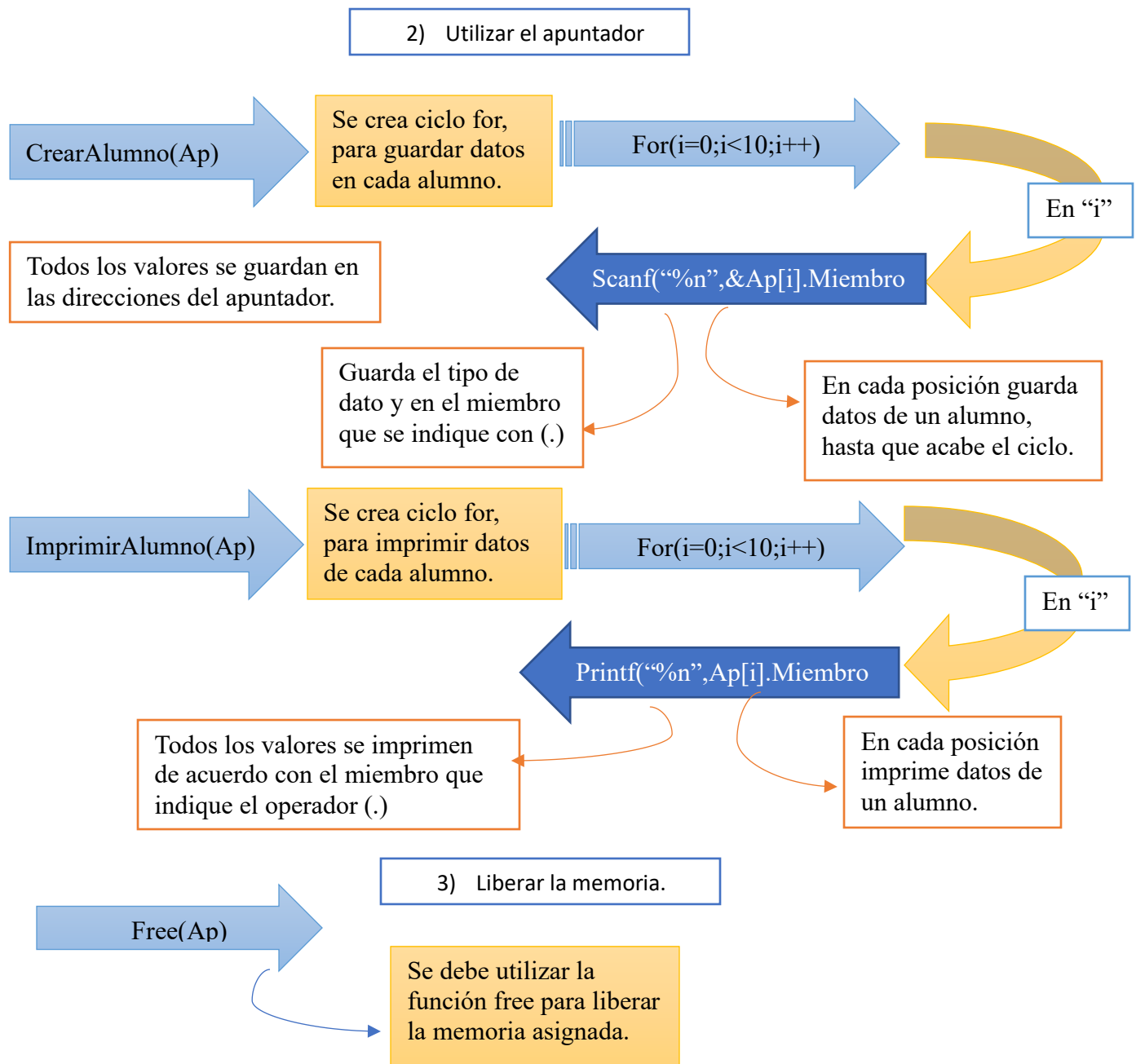


4.- Instrucciones para pedir al usuario los datos de los alumnos reservados.

Esta parte del programa fue la que me causo mayor problema por lo mencionado en el punto de dificultades, lo principal de esta parte del ejercicio consistió en utilizar el apuntador “din3”, el cual apuntaba a las direcciones de memoria antes descritas, y que permitía el guardar 10 alumnos, entonces las instrucciones fueron ejecutadas por medio de funciones, en las cuales pase valores por referencia, para guardar los datos que el usuario asignara a cada alumno y posteriormente imprimir los datos de cada alumno.

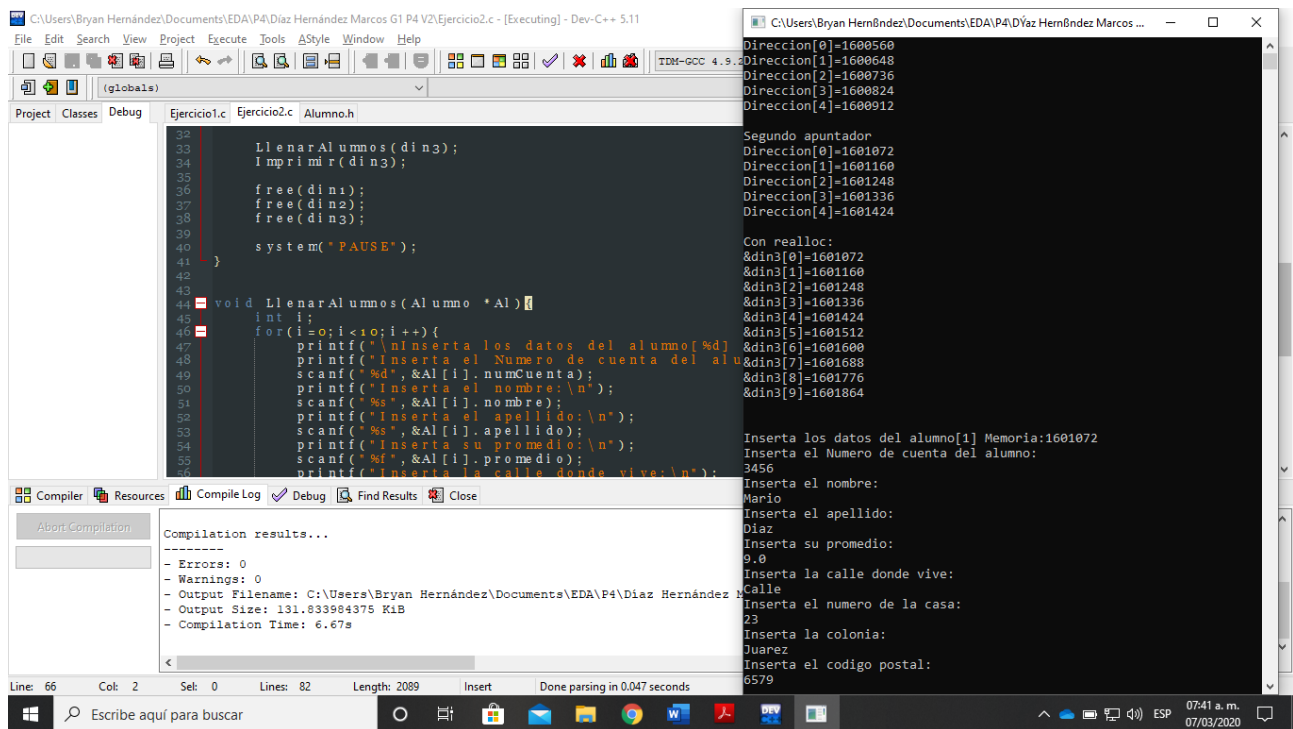
- Diagrama de funcionamiento.





- Relación con teoría.

La teoría envuelta en la resolución del tema se relaciona con los arreglos de estructuras, las funciones, las cuales sirven para poder trabajar en bloques de código, también se ocupa memoria dinámica que es el centro de esta práctica. En base a la memoria dinámica se comienza a realizar la programación del arreglo de estructuras, y con el arreglo de estructuras se trabajan con los conceptos de las funciones y de estructuras de datos abstractos.



• Ejercicio 3

El último ejercicio de la practica consistía en crear una estructura de tipo Automóvil, después crear un apuntador del tipo Automóvil al cual se le va a asignar memoria dinámica. Al asignarle memoria dinámica al apuntador se está asignando cuantos automóviles se van a registrar y ese valor lo va a dar el usuario.

Una vez asignado el número de automóviles a registrar el programa tiene que pedir los datos de cada auto, por medio de funciones y de ciclos, se llenan cada uno de los miembros de la estructura y se imprimen los datos de cada auto registrado.

• ¿Cómo lo resolví?

Lo primero que hice fue diseñar el dato Automóvil, después con funciones pedía al usuario la cantidad de autos que iba a registrar para después, por medio de otra función pasar el dato como parámetro y también pasar el apuntador de tipo automóvil al cual asigné memoria dinámica (Imagen 1).

```

11  int  Dato();
12  void CrearAutos(Automovil *Ap,int a);
13  void ImprimirAutos(Automovil *Az,int b);
14  void ImprimirMemoria(Automovil *Ac,int c);
15

```

Imagen 1.

Por medio de calloc, asigne memoria a un apuntador tipo automóvil, donde coloque como parámetro de esta función el valor de autos que el usuario iba a insertar, y el otro parámetro fue el tamaño del tipo de dato automóvil (Imagen 2).

```

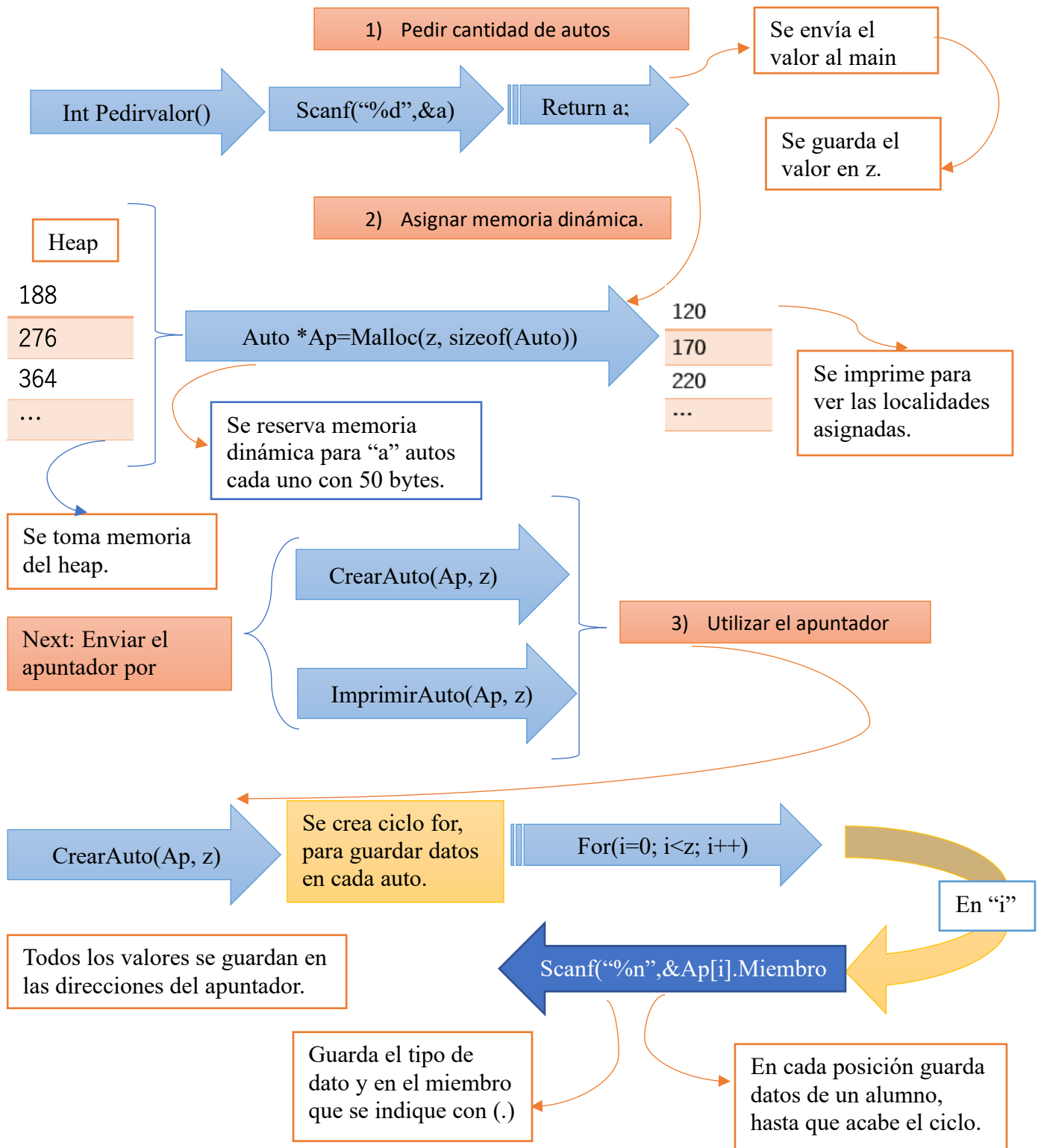
int main(){
    int z=Dato();
    Automovil *Autos=(Automovil *)calloc(z,sizeof(Automovil));
    ImprimirMemoria(Autos,z);
}

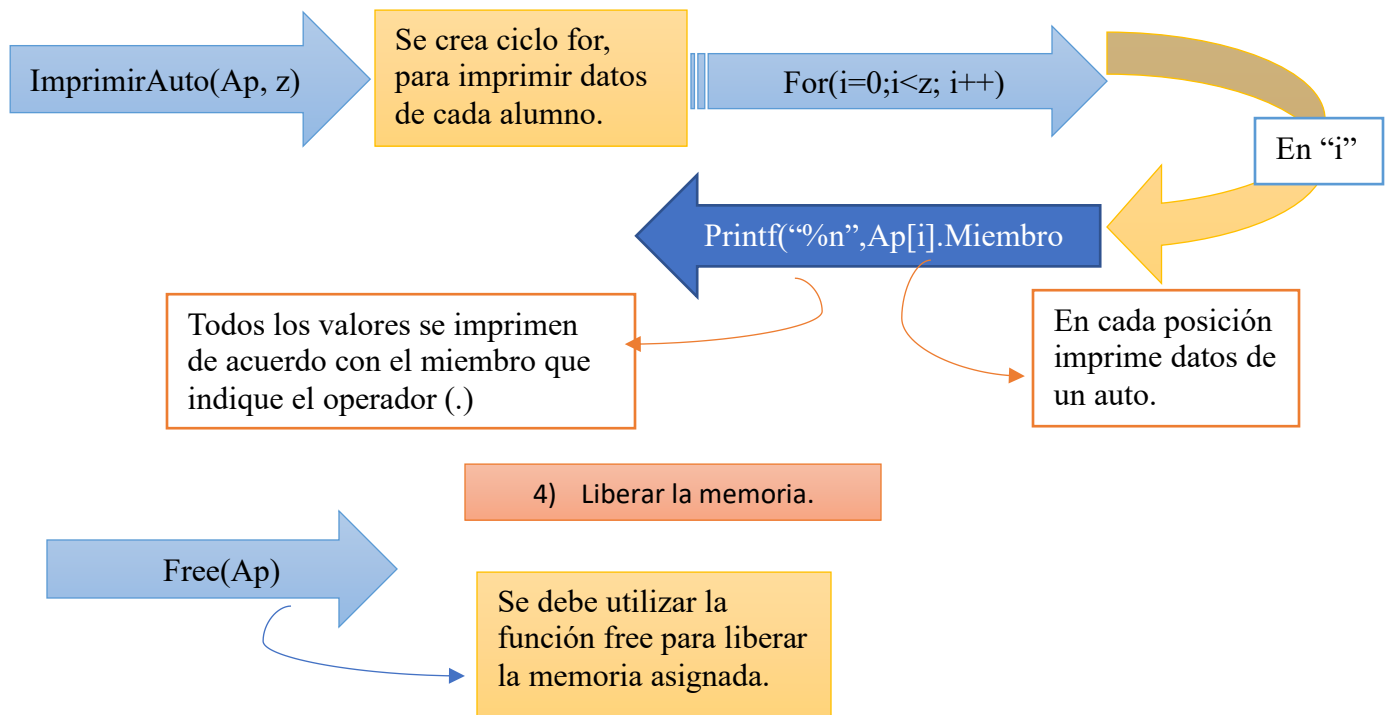
```

Imagen 2.

Por medio de funciones fui pidiendo al usuario los datos de cada carro, y fui guardando cada dato dentro de la memoria que fue asignada al apuntador, para después imprimir los datos que había insertado. (Imagen 1).

- Diagrama de funcionamiento





- Relación con teoría.

La parte teórica del ejercicio se fundamenta en memoria dinámica, arreglos de estructuras, además las funciones con mas de un parámetro, para poder hacer ciclos dentro de las funciones, y hacer el programa más eficiente, por otro lado, se usan los operadores de estructuras como es el (`.`) y (`->`) para acceder a los miembros de la estructura automóvil.

The screenshot shows a C++ IDE with the following code in `Ejercicio3.c`:

```

49 }
50
51
52 void ImprimirAutos(Automovil *Az, int b){
53     int i;
54     for(i=0; i<b; i++){
55         printf("Datos del automovil[%d]", i+1);
56         printf("Marca: %s\n", Az->Marca);
57         printf("Modelo: %s\n", Az->Modelo);
58         printf("Color: %s\n", Az->Color);
59         printf("Tipo: %s\n", Az->Tipo);
60     }
61 }
62
63
64
65 int Dato(){
66     int a;
67     printf("¿Cuántos autos registrara?:");
68     scanf("%d", &a);
69     return a;
70 }
71
72
  
```

The output window shows the results of the program execution:

```

Valor[1]:6487440, Memoria[1]:1862432
Valor[2]:6487440, Memoria[2]:1862482
Valor[3]:6487440, Memoria[3]:1862532

Inserta los datos del vehiculo[1], Memoria:1862432
Inserte la marca de auto:
Ferrari
Inserte el modelo del auto:
Spider
Inserte el color del auto:
Rojo
Inserte el tipo de auto:
Deportivo
Inserta los datos del vehiculo[2], Memoria:1862482
Inserte la marca de auto:
Ford
Inserte el modelo del auto:
Mustang
Inserte el color del auto:
Gris
Inserte el tipo de auto:
Deportivo
Inserta los datos del vehiculo[3], Memoria:1862532
Inserte la marca de auto:
Nissan
Inserte el modelo del auto:
Bicho
Inserte el color del auto:
Verde
Inserte el tipo de auto:
Normal

Datos del automovil[1]Marca:Ferrari
Modelo:Spider
Color:Rojo
Tipo:Deportivo

Datos del automovil[2]Marca:Ferrari
Modelo:Spider
  
```

The compilation results show 0 errors and 0 warnings, with an output size of 131.052734375 KiB and a compilation time of 0.59s.

Conclusiones.

Todos los ejercicios fueron resueltos de acuerdo con el objetivo, es decir, utilizar funciones para poder almacenar información en tiempos de ejecución, además de reservar memoria en apuntadores para utilizar la memoria, algo importante de la practica fue el uso de funciones para poder manejar los datos de forma más rápida y estructurada.

Un punto importante que no pude hacer se relacionó con las funciones que regresan un valor, debido que, al retornar un arreglo de estructuras, que se modificó en una función, y que ese arreglo se guardara en otro arreglo dentro del main. No lo pude hacer porque no estoy seguro de si es válido regresar un arreglo de estructuras. Por ello resolví los ejercicios mediante paso por valor, a pesar de buscar información la mayoría de la información indicaba funciones que regresaban estructuras simples, no arreglos de estructuras.

Para finalizar, se cumplió el objetivo, pero me hubiese gustado aplicar el retorno de valores mediante funciones para resolver los ejercicios, y tener otra forma de resolverlos.