



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Edgar Tista García

Profesor:

Estructura de Datos y Algoritmos I

Asignatura:

1

Grupo:

Práctica #4

No de Práctica(s):

Santiago Díaz Edwin Jaret

Integrante(s):

*No. de Equipo de
cómputo empleado:*

Trabajo en casa

42

No. de Lista o Brigada:

2021-2

Semestre:

18/06/2021

Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____

Almacenamiento en tiempo de ejecución.

Objetivos

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

Desarrollo

Ejemplo 1

El programa solicita al usuario la cantidad de elementos que contiene su conjunto para reservar esa cantidad de elementos. Utiliza:

- La función *malloc()*, para reservar un bloque de memoria de manera dinámica. Esta reserva el número de bytes, en este caso, el tamaño del tipo de dato *int* (4 bytes) y los multiplica por la cantidad de elementos que ingresó el usuario.
- Una *sentencia if* para validar que el conjunto no esté vacío.
- Un *ciclo for* para imprimir los datos.
- La función *free()*, para liberar la memoria reservada por *malloc*.

Al final, imprime el contenido de cada espacio reservado.

```
¿Cuántos elementos tiene el conjunto?
6
Vector reservado:
[ 1932240 0 1900880 0 1599690831 1128613955 ]
Se libera el espacio reservado.
```

Ejemplo 2

El programa solicita al usuario la cantidad de elementos que contiene su conjunto, para reservar dicho espacio. Utiliza:

- La función *calloc()* con parámetro del número de elementos ingresados y el tamaño de los mismos, en este caso es el tamaño del tipo de dato *int* (4 bytes).
- Una *sentencia if* para validar que el conjunto no esté vacío.
- Un *ciclo for* para imprimir los datos.
- La función *free()* para liberar la memoria reservada por *calloc*

Al final, imprime el valor de cada elemento guardado por *calloc*

```
¿Cuántos elementos tiene el conjunto?
6
Vector reservado:
[ 0 0 0 0 0 0 ]
Se libera el espacio reservado.
```

Ejemplo 3

El programa le solicita al usuario que ingrese la cantidad de elementos que contiene su conjunto para reservar el espacio en la memoria.

Reserva la memoria con la función *malloc()* guardando el resultado de la cantidad por el tamaño del tipo de valor *int*, en bytes. Después, valida que el conjunto no esté vacío con una *sentencia if*.

Utiliza 2 ciclos *for*, para guardar el valor de cada elemento y para imprimirlos en forma de un vector. Se aumentará el espacio reservado con la función *realloc()* pasando como parámetro qué es lo que se va a reacomodar y el tamaño en bytes, en este caso, la multiplicación de el número de elementos por el tamaño del tipo de dato *int*.

El aumento del arreglo es duplicado y con otros 2 *ciclos for* va a guardar los nuevos valores de los elementos y va a imprimir el vector completo. SS

Ejercicio 1

El programa contiene una un arreglo con 5 elementos. Un *ciclo for* recorre el arreglo, para imprimir:

- El número de elemento a imprimir
- La dirección de memoria del arreglo
- El valor del arreglo por cada posición

```
direccion arreglo[0]=6487536  valor arreglo[0]=35
direccion arreglo[1]=6487540  valor arreglo[1]=40
direccion arreglo[2]=6487544  valor arreglo[2]=45
direccion arreglo[3]=6487548  valor arreglo[3]=50
direccion arreglo[4]=6487552  valor arreglo[4]=55
direccion arreglo[5]=6487556  valor arreglo[5]=0
direccion arreglo[6]=6487560  valor arreglo[6]=1403872
direccion arreglo[7]=6487564  valor arreglo[7]=0
direccion arreglo[8]=6487568  valor arreglo[8]=1403840
direccion arreglo[9]=6487572  valor arreglo[9]=0
```

```
direccion=1403840  *valor=1380944
direccion=1403844  *valor=0
direccion=1403848  *valor=1376592
direccion=1403852  *valor=0
direccion=1403856  *valor=0
direccion=1403860  *valor=0
direccion=1403864  *valor=96273416
direccion=1403868  *valor=201360360
direccion=1403872  *valor=1380944
direccion=1403876  *valor=0
```

Presione una tecla para continuar . . .

Además, hay otra variable que guarda el espacio de memoria con una función *malloc()*, en este caso, guarda 10 espacios.

Con un *ciclo for* imprime:

- La dirección de memoria de cada espacio guardado.
- El valor de cada espacio guardado.

```
direccion=7695296  *valor=0
direccion=7695300  *valor=0
direccion=7695304  *valor=0
direccion=7695308  *valor=0
direccion=7695312  *valor=0
direccion=7695316  *valor=0
direccion=7695320  *valor=0
direccion=7695324  *valor=0
direccion=7695328  *valor=0
direccion=7695332  *valor=0
```

Al cambiar la función *malloc()* por *calloc()* , con el mismo parámetro de espacios reservados y pasándole el tamaño del tipo de dato int (4 bytes), se obtiene:

Dándonos a entender que la función *calloc()* guarda en cada espacio reservado un valor de **0**, siendo de distinta manera en la que guarda la función *malloc()*

Se modificó el programa para que se solicite al usuario que ingrese valores múltiplos de 4, esto se verifica con una *sentencia if* y estos valores se guardan en la variable con los espacios reservados por *calloc()*

Por último, se va a incrementar el espacio a 20 elementos con la función *realloc()*. El incremento se va a guardar en 2 variables, en **ptr** y en **ptr3**. Esto es para ver si se cambia la dirección de memoria al incrementar el espacio reservado

```
direccion=7564224  *valor=4
direccion=7564228  *valor=4
direccion=7564232  *valor=4
direccion=7564236  *valor=4
direccion=7564240  *valor=8
direccion=7564244  *valor=16
direccion=7564248  *valor=20
direccion=7564252  *valor=40
direccion=7564256  *valor=60
direccion=7564260  *valor=100
```

```
Sin realloc
direccion=1797056 *valor=4
direccion=1797060 *valor=4
direccion=1797064 *valor=4
direccion=1797068 *valor=44
direccion=1797072 *valor=4
direccion=1797076 *valor=4
direccion=1797080 *valor=4
direccion=1797084 *valor=4
direccion=1797088 *valor=8
direccion=1797092 *valor=8
```

```
Con realloc a la variable ptr
direccion=1797056 *valor=0
direccion=1797060 *valor=1
direccion=1797064 *valor=2
direccion=1797068 *valor=3
direccion=1797072 *valor=4
direccion=1797076 *valor=5
direccion=1797080 *valor=6
direccion=1797084 *valor=7
direccion=1797088 *valor=8
direccion=1797092 *valor=9
direccion=1797096 *valor=10
direccion=1797100 *valor=11
direccion=1797104 *valor=12
direccion=1797108 *valor=13
direccion=1797112 *valor=14
direccion=1797116 *valor=15
direccion=1797120 *valor=16
direccion=1797124 *valor=17
direccion=1797128 *valor=18
direccion=1797132 *valor=19
```

```
Con realloc a la variable ptr3
direccion=1797056 *valor=0
direccion=1797060 *valor=1
direccion=1797064 *valor=2
direccion=1797068 *valor=3
direccion=1797072 *valor=4
direccion=1797076 *valor=5
direccion=1797080 *valor=6
direccion=1797084 *valor=7
direccion=1797088 *valor=8
direccion=1797092 *valor=9
direccion=1797096 *valor=10
direccion=1797100 *valor=11
direccion=1797104 *valor=12
direccion=1797108 *valor=13
direccion=1797112 *valor=14
direccion=1797116 *valor=15
direccion=1797120 *valor=16
direccion=1797124 *valor=17
direccion=1797128 *valor=18
direccion=1797132 *valor=19
```

La dirección de memoria de los elementos guardados, no cambia entre las variables **ptr** y **ptr3** y además, utiliza las 10 direcciones de memoria antes de utilizar la función *realloc()*.

Ejercicio 2

Al principio el programa me marcaba error en la biblioteca “Alumno.h”, lo solucioné agregando

```
• #include <stdlib.h>
• #include <string.h>
```

El programa utiliza la biblioteca “*Alumno.h*” en donde se encuentra la estructura **Alumno** y **Dirección**, una función para llenar la información de las estructuras y una función en donde imprima la información de las estructuras.

Se imprime:

el tamaño en bytes de la estructura alumno. Esta se obtiene de la suma del tamaño de cada tipo de dato que contiene en bytes.

```
Tamaño de objeto Alumno = 88
```

- **Int:** $4(\text{bytes}) * 3(\text{variables}) = 12(\text{bytes})$
- **Char:** 1. En el archivo “Alumno.h” se especifica cuántos bytes es del tamaño de la variable, por lo tanto, se suma: $15+15+20+20 = 70$
- **Float:** $6(\text{bytes}) * 1(\text{variable}) = 6(\text{bytes})$
- La suma da **88 bytes**

Va a imprimir la dirección de memoria que se guarda con *malloc()*, cada dirección de memoria es un espacio reservado.

```
Primer apuntador:
Direccion[0]=10949200
Direccion[1]=10949288
Direccion[2]=10949376
Direccion[3]=10949464
Direccion[4]=10949552
```

Va a imprimir la dirección de memoria que guarda con *calloc()*, el espacio es de 5 elementos con tamaño de la estructura Alumno (88 bytes)

```
Segundo apuntador
Direccion[0]=10949648
Direccion[1]=10949736
Direccion[2]=10949824
Direccion[3]=10949912
Direccion[4]=10950000
```

Esto nos da de entender que cada función guarda en direcciones de memoria distintas.

Se utiliza un *realloc()* para aumentar el tamaño de **din2** (variable con función *calloc()*) a 10 espacios. Al aumentar, utiliza los mismos primeros 5 espacios que había reservado *calloc()* y los demás, son nuevos espacios reservados.

Para que el programa solicite los datos al usuario con el fin de llenar las estructuras, se utiliza:

- Una función que recopila los datos al usuario, con un *ciclo for* recorre los 10 espacios guardados por *calloc()*. Se guarda la información en la variable con los espacios reservados utilizando el operador miembro

```
Número de cuenta del Alumno #1: 9128
Nombre del Alumno #1: Edwin
Apellido del Alumno #1: San
Promedio del Alumno #1: 0
Calle del Alumno #1: Tecoh
Número de casa del Alumno #1: 12
Colonia del Alumno #1: Pedregal
Código Postal del Alumno #1: 14100
```

```
scanf("%d",&(alumno[j].numCuenta));
scanf("%s",&(alumno[j].domicilio.calle));
```

```
for (j = 0; j < 10; j++)
{
    limpio[j] = din4;
}
imprimirlos(limpio, j);
```

Por último, para liberar la memoria, se puede crear una variable con los datos vacíos de la estructura y con un *ciclo for()*, igualar las variables guardadas con la variable vacía. Se manda a llamar a la función encargada de imprimir los datos pasándole como parámetro la variable con los espacios guardados

```
Número de cuenta del Alumno #1: 0
Nombre del Alumno #1:
Apellido del Alumno #1:
Promedio del Alumno #1: 0
Calle del Alumno #1: j
Número de casa del Alumno #1: 0
Colonia del Alumno #1:
Código Postal del Alumno #1: 0
```

Ejercicio 3

El programa utiliza una biblioteca “computadora.h” en donde se encarga de crear una estructura de dato llamada Computadora, que contiene las variables:

1. De tipo carácter, **Marca** con un espacio de 15 bytes
2. De tipo carácter, **Modelo** con un espacio de 10 bytes.
3. De tipo carácter, **Procesador** con un espacio de 15 bytes
4. De tipo carácter, **Memoria** con un espacio de 10 bytes

Se le solicita al usuario el tamaño del arreglo, esto es para reservar un espacio en la memoria con la función *calloc()* y el tamaño de cada elemento es el tamaño de la estructura, que se obtiene con la función *sizeof()*.

```
Computadora *dir;
printf("Ingrese el tamaño del arreglo: ",an);
scanf("%d", &tam);
dir = calloc(tam, sizeof(Computadora));
```

Se declaran y utilizan 2 funciones para:

- Capturar los datos y guardarlos. Esto es posible con una *ciclo for()* que recorre el espacio reservado y se guarda en la variable con la notación punto.
- Se imprime los resultados obtenidos, con un *ciclo for()* se recorre los espacios guardados.

```
(relleno[i].marca)
```

```
La marca de la computadora #2 es: hP
El modelo de la computadora #2 es: LHN.1021
El procesador de la computadora #2 es: Intel Core i9
La memoria de la computadora #2 es de: 64 RAM
```

```
Ingrese la marca de la Computadora #2
hP
Ingrese el Modelo de la Computadora #2
LHN.1021
Ingrese el procesador de la Computadora #2
Intel Core i9
Ingrese la memoria de la Computadora #2
64 RAM
```

Conclusiones

Los Objetivos han sido cumplidos, aplicamos el desarrollo de la memoria dinámica en todos los ejercicios, as su vez, cada ejercicio se realizó de manera exitosa.

Reservar la memoria de manera dinámica genera un mejor flujo en la ejecución del programa, esto reduce el consumo de memoria innecesaria, fijar un límite y liberar la memoria que no necesitemos en el momento en que deseemos.

Si juntamos las variables de tipo estructura, funciones, memoria dinámica y arreglos, el programa llega a ser más eficiente, reduce las líneas de código, es más flexible y abre las puertas de las infinitas posibilidades de programas que se pueden hacer.

En lo personal,, uno de los problemas más difíciles es el desarrollo del programa, es el desarrollo y aplicación de la lógica, pues es utilizada para un generar mejor flujo del programa, y por eso, la práctica es de gran ayuda para crecer.