	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS I

Grupo: 1

No de Práctica(s): 4] ALMACENAMIENTO EN TIEMPO DE EJECUCIÓN

Integrante(s): MURRIETA VILLEGAS ALFONSO

Semestre: 2018 - 2

Fecha de entrega: 12 DE MARZO DE 2018

Observaciones:

CALIFICACIÓN: _____

APLICACIONES DE APUNTADORES

1] Objetivo de la práctica

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

2] Desarrollo

INTRODUCCIÓN

El concepto de memoria dinámica hace referencia al espacio de memoria que es reservada al momento en que se ejecuta un programa, a su vez este también hace referencia a que puede variar durante la ejecución del programa.

El principal uso de la memoria dinámica es el optimizar el consumo de memoria o en dado caso el definir de manera más apropiada el consumo de memoria cuando no se sabe con exactitud la cantidad de elementos que se requerirán durante la ejecución de un programa.

MEMORIA DINÁMICA

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 o stack 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.

Las principales características del ***heap*** es que de manera explícita este tiene una duración fija (Cuando se inicia y termina un programa), además de que para acceder a un dato de este se hace uso de una referencia que en el caso de C es mediante un apuntador ubicado en la pila.

Para hacer emplear la memoria dinámica el Lenguaje C hace uso de 3 funciones, que es el caso de malloc, calloc y realloc.

2.1] Análisis de los problemas (Propuestos)

En el presente apartado se demostrarán el uso de las 3 funciones empleadas por el Lenguaje C en memoria dinámica.

EJERCICIO I | MALLOC

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

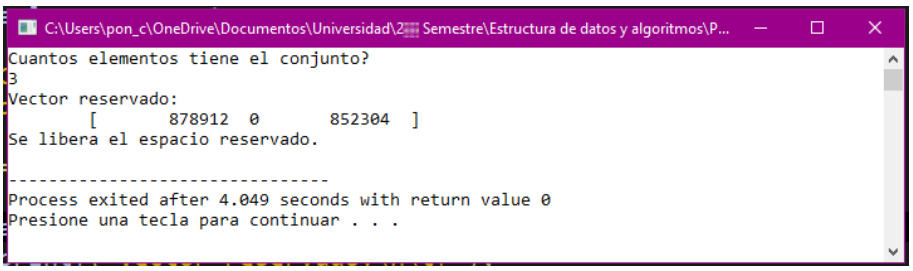
void *malloc(size_t size);

El primer ejercicio del manual hace uso de la función malloc para hacer una reserva de memoria, la manera en que hace esta reserva es la siguiente:

```
arreglo = (int *)malloc (num * sizeof(int));
```

Donde ***malloc*** recibe como parámetro el número de bytes que se desea reservar además de que se hace uso del ***sizeof*** para determinar la cantidad exacta de bytes.

Por último, la salida final del programa se muestra en la siguiente imagen.



NOTA: Como se puede ver dos de los datos reservados incluyen datos del sistema

NOTA 2: Una vez que se ha utilizado el espacio debe liberarse y esto se hace mediante la función free donde el parámetro que se le pasa es el apuntador al inicio de la memoria que se desea liberar. Si el apuntador es nulo la función no hace nada.

EJERCICIO II | 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:

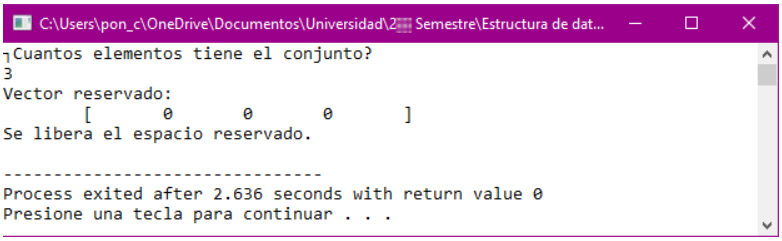
void *calloc (size_t nelem, size_t size);

El segundo ejercicio del manual hace uso de la función calloc para hacer una reserva de memoria, la manera en que hace esta reserva es la siguiente:

```
arreglo = (int *)calloc (num, sizeof(int));
```

Donde ***calloc*** recibe como parámetro el número de elementos que se van a reservar y mientras el sizeof indica el tamaño de cada elemento.

Por último, la salida final del programa se muestra en la siguiente imagen.



NOTA: Recordemos que a diferencia de malloc , calloc inicializa los valores en 0.

EJERCICIO III | 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);

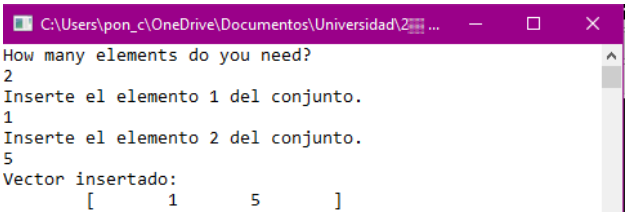
El tercer ejercicio del manual hace uso de la función realloc para redimensionar el tamaño de la reserva de la memoria, la manera en que hace esto es la siguiente:

```
arreglo2 = (int *)realloc (arreglo,num*sizeof(int));
```

Donde el arreglo 2 es el nuevo arreglo el cual se usará para redimensionar el arreglo 1, además cabe mencionar que para llevar acabo esto anteriormente se debe hacer una reserva de memoria.

Los parámetros que usa ***realloc*** son el apuntador que se va a redimensionar y ***sizeof*** es usado para declarar el nuevo tamaño en bytes que se desean aumentar al conjunto.

Por último, la salida del programa es la siguiente:



En esta imagen se ve la reserva e impresión del arreglo dinámico

```
Aumentando el tamaño del conjunto al doble.
Inserte el elemento 3 del conjunto.
5
Inserte el elemento 4 del conjunto.
6
Vector insertado:
        [       1       5       5       6       ]
```

En esta imagen se ve la reserva e impresión del arreglo dinámico una vez que se redimensiona (x2)

2.2] Análisis de los problemas (Práctica)

EJERCICIO I

PARTE A

El presente programa es un ejemplo del uso de memoria dinámica a través de la implementación de la función malloc, a continuación, se presenta parte por parte cómo se llegó o cómo es que salen los resultados impresos en la pantalla:

Es esta primera imagen lo que podemos ver en primera estancia es la posición de memoria de los datos de un arreglo, además de sus correspondientes valores.

En la segunda parte donde se menciona apuntador primero se muestra la posición de memoria de apuntador y posteriormente se muestra el valor que tiene.

NOTA: Recordemos que debido a que se usa malloc usualmente al momento de inicializar valores puede incluir datos del sistema.

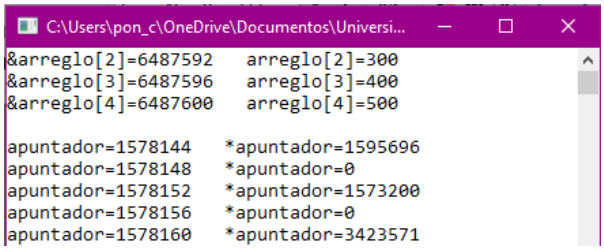


Imagen 1: Primera captura de pantalla

Posteriormente, a través del uso de la función ***strcpy*** (Esta función se emplea para el uso de cadenas de caracteres) y de un ciclo for imprimimos tanto la posición de memoria como los correspondientes caracteres.

A continuación, se muestra la salida del programa donde se puede ver la posición de memoria con su respectivo carácter.

Nótese que los espacios en blanco son los espaciados de la cadena de caracteres.

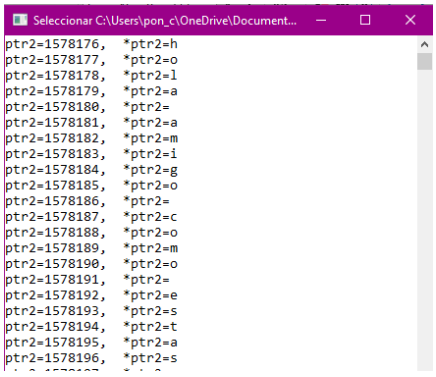
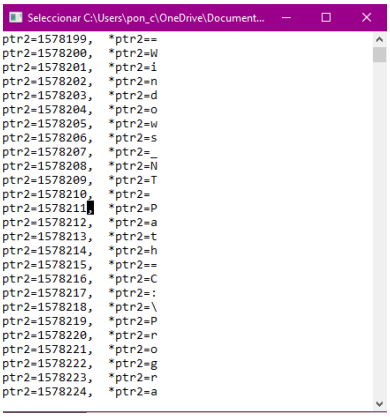
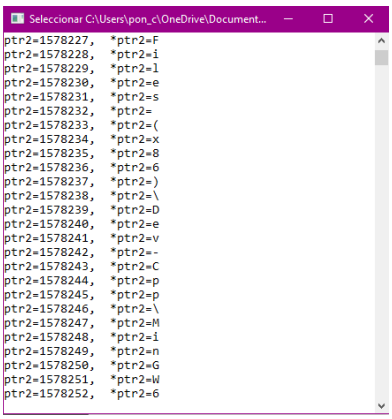


Imagen 2: Captura de pantalla que hace referencia a la impresión de la cadena

Sin embargo, debido a que se está utilizando la función ***malloc*** recordemos que esta puede incluir datos del sistema, en este caso concreto podemos ver como se imprime una dirección relacionada al compilado del programa.





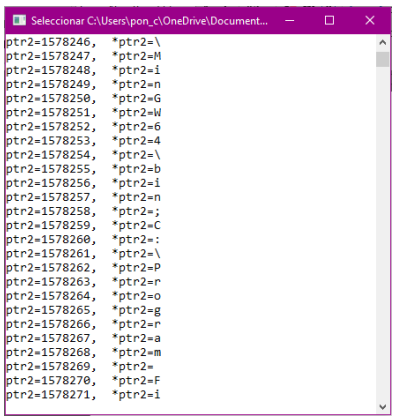


Imagen 3, 4, 5: Capturas de pantalla que hace referencia a la impresión de alguna dirección relacionada al compilado del programa

PARTE B

Al momento de remover los comentarios de las líneas 15 a 22 se puede apreciar que se están dando nuevos valores a los apuntadores sin cambiar la posición de memoria de estos, la manera en que cambia los valores es a través de la siguiente línea

Donde podemos ver que los correspondientes valores se dan a través de la suma del contador dentro de un ciclo for.

```
printf("\n");
for (cont=0;cont<5;cont++){
    *(ptr+cont)= (cont+1)*100;
}
```

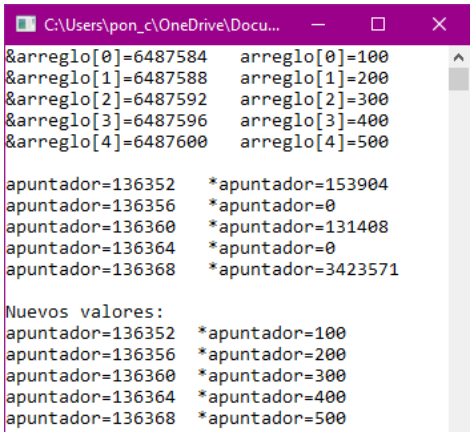
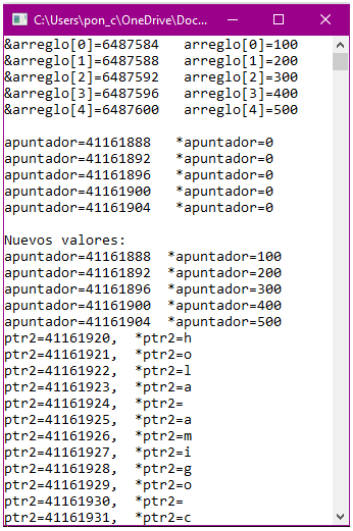
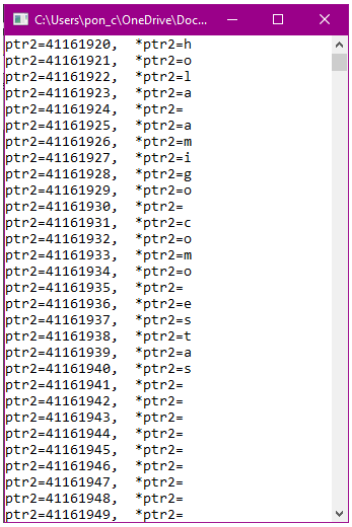


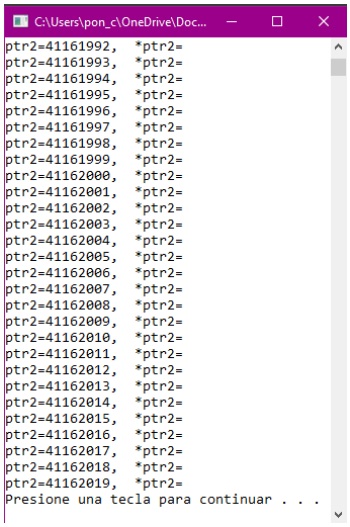
Imagen 6: Salida del programa cuna vez removidos los valores de las líneas 15 a 22

PARTE C

Al emplear la función calloc no cambiamos ningún aspecto relacionado a los datos de la salida excepto la parte donde se imprimían datos relacionados con el sistema, podemos observar este pequeño detallo primero al momento de inicializar los valores de lo apuntadores en cero, además de que podemos ver que al momento de terminar de imprimir la cadena de caracteres también en este apartado solamente se muestra la cadena inicializada en el programa (Se omite la dirección que se había impreso debido al uso de malloc)







Imágenes 7,8 y 9: Salida del programa usando la función ***calloc*** en vez de ***malloc.***

PARTE D

Como podemos observar en el anterior código el arreglo tiene los mismos valores que los que da la línea 17 por ende podemos sustituir esta línea por el uso del arreglo del programa.

```
printf("\n");
for (cont=0;cont<5;cont++){
    //*(ptr+cont)= (cont+1)*100;
    *(ptr+cont)= arreglo[cont];

}
```

Imagen 10: Se muestra la sustitución de la línea 17 mediante la implementación del

A continuación, se muestra la salida del programa empleando las líneas de código anteriores:

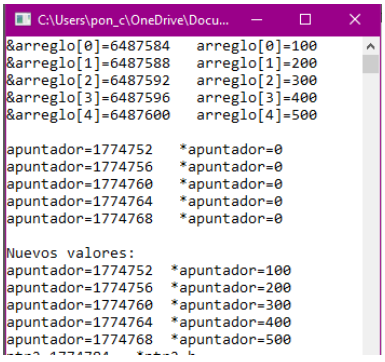


Imagen 11: Salida del programa

NOTA: Podemos ver como nuevamente se cambian las direcciones de memoria.

EJERCICIO II

PARTE A

El presente programa es un ejemplo del uso de memoria dinámica a través de la implementación de las funciones ***malloc*** y ***calloc***, a continuación, se presenta la salida del programa.

Notemos que los datos de salida son las respectivas direcciones de memoria donde fueron guardados los datos de los respectivos alumnos creados

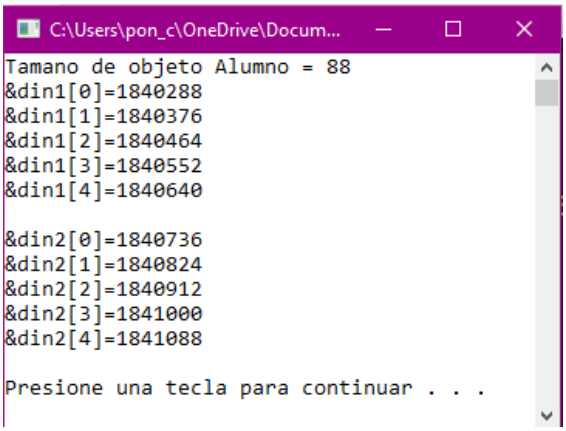


Imagen 12: Salida del programa

PARTE B

Para poder redimensionar el apuntador (En este caso es el de la función ***calloc*)** debemos usar ***realloc*** donde a través de la declaración de otro apuntador y ajustando al tamaño solicitado obtenemos lo pedido.

A continuación, se muestra la parte del código que se encarga de redimensionar el apuntador y también se muestra el ciclo for encargado de la impresión.

NOTA: Se usó un nuevo apuntador para poder redimensionar.

```
//PARTE PARA AUMENTAR
printf("Aumentando el tamaño del conjunto\n");
din3 = (Alumno*)realloc(din2,20*sizeof(Alumno));
printf("\n");

for(j=0;j<20;j++){
    printf("&din2[%d]=%d \n",j,din2+j);
}
printf("\n");
```

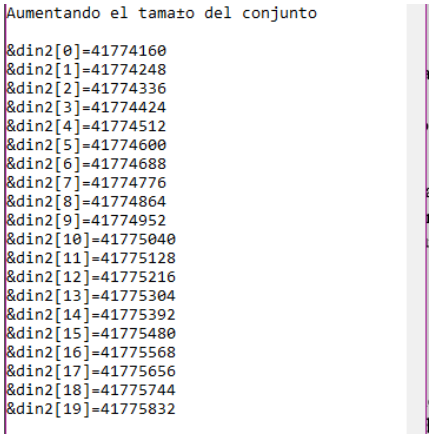


Imagen 13: Salida del programa una vez empleada la función ***realloc***

PARTE C

Al omitir el ***sizeof*** en la función ***calloc*** no obtenemos ningún error o efecto sobre la ejecución del programa y esto se debe a que anteriormente se usa la forma del elemento que se va a utilizar (Se usa el apuntador a alumno), como se puede ver en la siguiente captura de la línea empleada.

```
din1 = (Alumno*)malloc(5 /* *sizeof(Alumno) */);
```

A continuación, se muestra la salida del programa:

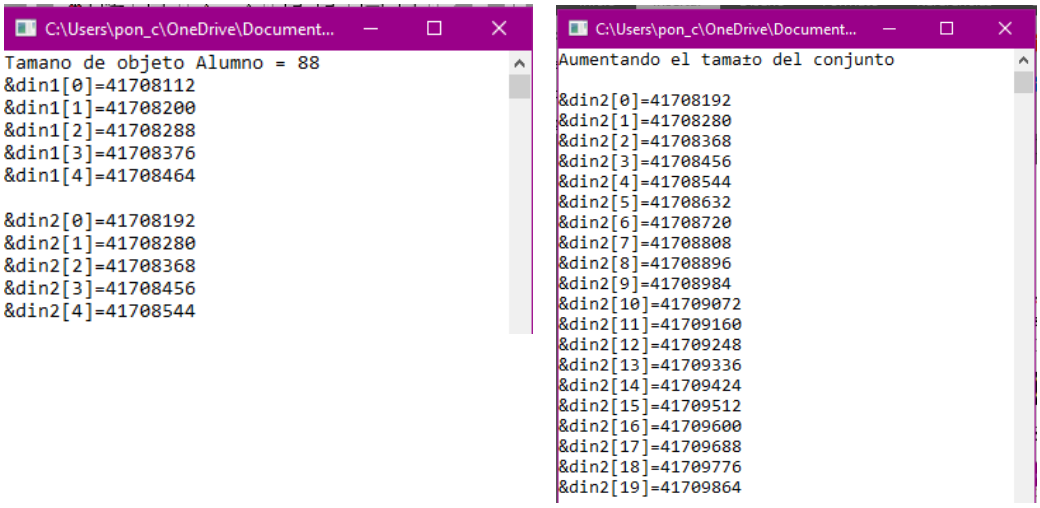


Imagen 14: Salida del programa una vez cambiada la línea 24

PARTE D

Una posible forma de liberar la memoria sin utilizar el ***free*** es usando la función ***realloc*** pero en vez de redimensionar de manera que se amplié la memoria lo que tendríamos que hacer es reducir el tamaño a cero o uno para de esta forma se pueda liberar toda o la mayor cantidad posible de memoria utilizada.

EJERCICIO III

Para poder llevar acabo todo lo que se pedía en el programa, tuve que abordar cada requisito de manera sistemática, primero tuve que realizar un tipo de dato abstracto empleado una estructura a la cual llame libro, a continuación, muestro la composición y la declaración de esta:

```
typedef struct {
    //Se omite los arreglos de caracteres mediante apuntadores

    //char titulo[20];
    //char autor[20];
    //char editorial[20];
    char *titulo;
    char *autor;
    char *editorial;
    int id;
}libro;
```

NOTA: Se utilizaron apuntadores a arreglos

Posteriormente para poder llevar más ordenado el código, además de fragmentarlo en segmentos más óptimos, utilice 3 funciones, ***main***, ***llenarDatos*** e ***imprimirLibro***

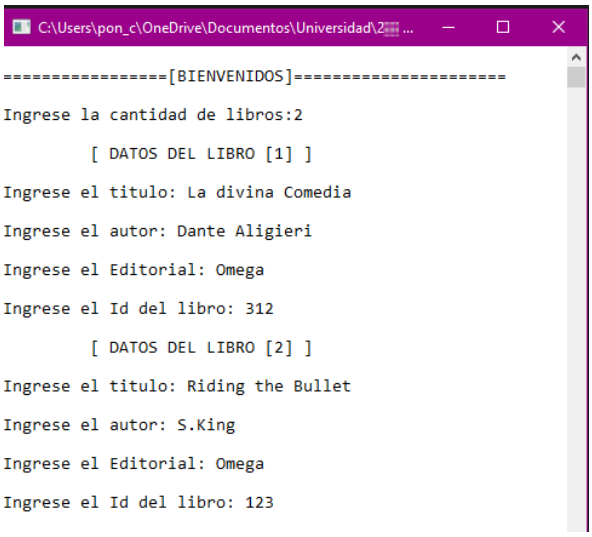
En el caso de la función ***main*** solamente se emplea para pedir la cantidad de libros que serán ingresados, además de que utiliza la función calloc para hacer dinámico todos los libros que serán ingresados.

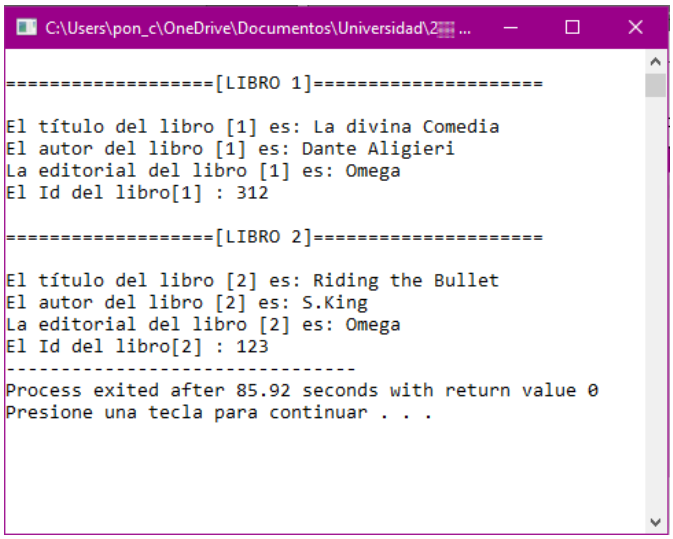
Por otro lado, la función ***llenarDatos*** incluye un apartado dedicado a los arreglos de caracteres los cuales usan apuntadores, también en este apartado se piden y guardan los datos de cada libro.

Por último, la función ***imprimirLibro*** se encarga de imprimir cada uno de los libros ingresados.

NOTA: Dentro de la función main a través del uso de ciclos for se realizan cada caso de las funciones ***llenarDatos*** e ***imprimirLibro.***

A continuación, se muestra tanto el llenado como la impresión de los datos usados:





Imágenes 15 y 16: En la imagen 15 se puede ver el llenado de los datos de cada libro, mientras que en la imagen 16 se puede ver la salida de los datos ingresados

2.3] Relación entre los ejercicios y el tema

Los ejercicios tanto del manual como los propuesto por el profesor son un acercamiento más a temas más complejos como son pilas, colas, etc, sin embargo, también este tema es un ejemplo de cómo podemos manipular de una mejor manera la memoria que consume nuestro programa, pese en nuestras computadoras disponemos por lo regular de muchísima memoria además de que nuestros programas consumen muy pocos recursos, consideremos que esto no siempre fue así o incluso no siempre es así (Hablando de casos actuales), tomemos en cuenta que hay muchos microcontroladores que no disponen de GB de memoria RAM en su lugar consumen cantidades inferiores las cuales deben ser usadas de manera más óptima.

El saber y disponer de este tipo de herramientas es sin duda una opción y alternativa al momento de programar con los mínimos recursos necesarios.

3] Conclusiones

La presente práctica está pensada en que como futuros ingenieros en computación siempre tengamos en mente que el mejor código no es el que tiene menos líneas sino el que realiza o cumple con el objetivo con la menor cantidad de recursos empleados y además en el menor tiempo de ejecución posible. En este particular caso lo que se buscaba hacer es lograr una mayor optimización en nuestros programas a través del uso de memoria dinámica la cual nos ayuda a poder emplear de una mejor manera la memoria RAM durante el tiempo de ejecución de un programa.

Con respecto a los ejercicios del manual de práctica los programas que se hicieron a diferencia de los plateados por el profesor solo tuvieron como propósito el enseñar el uso y sintaxis de las funciones empeladas para la memoria dinámica.

En cambio, los ejercicios planteados por el profesor (En el caso del ejercicio 1) son más que nada para enseñarnos y demostrarnos de manera visual las diferencias de las funciones al momento de emplearlas en un código algo más complejo.

En el caso del ejercicio 2 podemos notar que como principal propósito se encuentra el demostrar algunas características de las funciones malloc y calloc que usualmente no vemos, además de la parte reflexiva en los puntos C y D.

Por último, el ejercicio 3 notamos que mediante todo el conocimiento aprendido debemos emplearlo para poder lograr un arreglo dinámico, este ejercicio es el claro ejemplo de como a través del uso de estructuras, arreglos, apuntadores y memoria dinámica podemos hacer un programa más organizado, modular y óptimo.

4] Referencias

1) El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

2) Curso de Programación C7 C++. Javier Ceballos, tercera edición. USA, Ra – Ma. Educación 2007