	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	M.I. MARCO ANTONIO MARTINEZ QUINTANA
Asignatura:	EDA I
Grupo:	17
No de Práctica(s):	4
Integrante(s):	MAGALLANES GARCÍA ELVIRA VALENTINA
No. de Equipo de cómputo empleado:	34 EGIPTO
No. de Lista o Brigada:	
Semestre:	2020-2
Fecha de entrega:	01/03/2020
Observaciones:	

CALIFICACIÓN: _____

Objetivo:

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

Introducción:

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.

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 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.

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:

Código:

```c
//Practica 4. Memoria dinamica

#include <stdio.h>
#include <stdlib.h>

int main()
{
    //Un apuntador para poder acceder al dato almacenado
    int *arreglo, num, cont;
    printf("Cuantos elementos tiene el conjunto? \n");
    scanf("%d", &num);
    //Uso memoria dinamica en mi arreglo, voy guardando los datos en tiempo de ejecucion
    //malloc funciona para reservar un bloque de memoria
    arreglo = (int *)malloc (num * sizeof(int));
    if(arreglo!=NULL)
    {
        printf("Vector reservado: \n\t [");
        for (cont=0; cont<num ; cont++)
        {
            printf("\t %d", *(arreglo+cont));
        }
        printf("\t]\n");
        printf("Se libera el espacio reservado \n");
        free(arreglo);
    }
    return 0;
}
```

Ejecución:

```
Cuantos elementos tiene el conjunto?
3
Vector reservado:
         [       0       0       0       ]
```

```
Se libera el espacio reservado
Program ended with exit code: 0
```

Aquí dio datos con puros ceros porque como fue mi primera corrida la memoria estaba vacía

```
Cuantos elementos tiene el conjunto?
5
Vector reservado:
        [       0       0       0       0       3       ]
Se libera el espacio reservado
Program ended with exit code: 0
```

Aquí ya la memoria estaba llena y comenzó a arrojar datos que tenía pre guardados, por esto están esos números random porque es lo que tenía la memoria guardada.

```
Cuantos elementos tiene el conjunto?
10
Vector reservado:
        [       0       0       270009164       1610612736      463     0       31      0       0       0       ]
Se libera el espacio reservado
Program ended with exit code: 0
```

Código:

```c
//Practica 4. Memoria dinamica

#include <stdio.h>
#include <stdlib.h>

int main (){
    //Se inicializa las variables, dos enteros y un apuntador
    int *arreglo, num, cont;
    // Se imprime un comentario en pantalla
    printf("¿Cuantos elementos tiene el conjunto?\n");
    // Se guarda un dato de tipo entero
    scanf("%d",&num);
    //Se usa la funcion calloc
    /* La diferencia entre malloc y calloc, es que calloc inicializa la memoria reservada en cero
     */
    arreglo = (int *)calloc (num, sizeof(int));
    if (arreglo!=NULL) {
        printf("Vector reservado:\n\t[");
        for (cont=0 ; cont<num ; cont++){
            printf("\t%d",*(arreglo+cont));
        }
        printf("\t]\n");
        printf("Se libera el espacio reservado.\n");
        free(arreglo);
    }
    return 0;
}
```

Ejecución:

```
¿Cuantos elementos tiene el conjunto?
5
Vector reservado:
        [       0       0       0       0       0       ]
Se libera el espacio reservado.
Program ended with exit code: 0
```

Calloc se encarga de borrar todos los datos basuras que ya tenía la memoria, por lo tanto da puros ceros

```
¿Cuantos elementos tiene el conjunto?
10
Vector reservado:
        [       0       0       0       0       0       0       0       0       0       0       ]
Se libera el espacio reservado.
Program ended with exit code: 0
```

Código:

```c
//Practica 4. Memoria dinamica
#include <stdio.h>
#include <stdlib.h>
int main (){
    //inicializo mis variables
    int *arreglo, *arreglo2, num, cont;
    //Pregunto cuantos elemento tendra mi primer conjunto
    printf("¿Cuántos elementos tiene el conjunto?\n");
    scanf("%d",&num);
    arreglo = (int *)malloc (num * sizeof(int));
    if (arreglo!=NULL) {
        //Aqui ya puede el usario ingresar datos en el conjunto, no se
        //agregarian datos basura
        for (cont=0 ; cont < num ; cont++){
            printf("Inserte el elemento %d del conjunto.\n",cont+1);
            scanf("%d",(arreglo+cont));
        }
        //Me imprime el primer arreglo, con los datos agregados anteriormente
        printf("Vector insertado:\n\t[");
        for (cont=0 ; cont < num ; cont++){
            printf("\t%d",*(arreglo+cont));
        }
        printf("\t]\n");
        //Aqui se usara el segundo arreglo y aumentara el vector en tiempo de ejecucion
        printf("Aumentando el tamaño del conjunto al doble.\n");
        num *= 2;
        //Como voy a aumentar al doble, la variable antes dada por el usario donde
        //me indicaba de que tamaño quería mi vector, ahora la multiplico por 2 para
        //que sea el doble
        arreglo2 = (int *)realloc (arreglo,num*sizeof(int));
        //uso la funcion realloc que me ayuda a redimensionar mi vector en tiempo de ejecucion
        //está solo se debe usar luego del malloc o el calloc, porque necesita tener datos de memoria
        //anteriores para poder redimensionar
        if (arreglo2 != NULL) {
            arreglo = arreglo2;
            for (; cont < num ; cont++){
                //comienza el usario a agregar los datos faltantes
                printf("Inserte el elemento %d del conjunto.\n",cont+1);
                scanf("%d",(arreglo2+cont));
            }
            //imprime el vector final
            printf("Vector insertado:\n\t[");
            for (cont=0 ; cont < num ; cont++){
                printf("\t%d",*(arreglo2+cont));
            }
            printf("\t]\n");
        }
        free (arreglo);
    }
    return 0;
}
```

Ejecución:

```
¿Cuántos elementos tiene el conjunto?
4
Inserte el elemento 1 del conjunto.
1
Inserte el elemento 2 del conjunto.
2
Inserte el elemento 3 del conjunto.
3
Inserte el elemento 4 del conjunto.
4
Vector insertado:
        [       1       2       3       4       ]
Aumentando el tamaño del conjunto al doble.
Inserte el elemento 5 del conjunto.
5
Inserte el elemento 6 del conjunto.
6
Inserte el elemento 7 del conjunto.
7
Inserte el elemento 8 del conjunto.
8
Vector insertado:
        [       1       2       3       4       5       6       7       8       ]
Program ended with exit code: 0
```

```
¿Cuántos elementos tiene el conjunto?
-4
Program ended with exit code: 0
```

Como esto no es válido, el código termina desde ahí y no ejecuta lo demás

```
¿Cuántos elementos tiene el conjunto?
2.5
Inserte el elemento 1 del conjunto.
Inserte el elemento 2 del conjunto.
Vector insertado:
        [       0       0       ]
Aumentando el tamaño del conjunto al doble.
Inserte el elemento 3 del conjunto.
Inserte el elemento 4 del conjunto.
Vector insertado:
        [       0       0       268779433       -536870912      ]
Program ended with exit code: 0
```

Aquí debería hacer como en el código anterior y no ejecutar lo siguiente, pero si se ejecuta y lo que hace es que arroja valores basuras y el vector no está correctamente al doble, ya que solo está tomando el valor entero y no el decimal. Los valores basura que arroja son por el *malloc*

Conclusiones:

La práctica se realizó de manera sencilla y me ayudó a aprender de mejor manera como se usa la memoria dinámica, gracias a los ejercicios pude entender la diferencia entre malloc y calloc, además que entendí como se puede ir introduciendo datos en tiempo de ejecución e ir guardándolos en memoria al mismo tiempo.