Práctica 4

Estructuras de datos dinámicas

1. Responda:

   a. ¿En qué se diferencia la función **malloc** de la función **calloc**? ¿y de la función **realloc**?
   b. ¿Puede utilizarse la función **realloc** en lugar de la función **malloc**? ¿Se requiere alguna condición? ¿Qué sucede si **realloc** se invoca con el valor 0 como parámetro?
   c. ¿Qué utilidad tiene el operador **sizeof** a la hora de reservar memoria dinámicamente?

2. Analice, compile y ejecute el siguiente código. Indique imprime y por qué.

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

void f (int * p);

int main(){

    int * ptr = NULL;
    f(ptr);

    if (ptr == NULL)
        printf("ptr es NULL\n");
    else
        printf("ptr no es NULL\n");

    return 0;
}

void f (int * p) {
    p = (int *) malloc(10*sizeof(int));
}
```

3. Escriba un programa que lea un número entero ***n*** desde teclado y luego reserve memoria en forma dinámica para un arreglo de ***n*** enteros. Inicialícelo con valores aleatorios y a continuación calcule e imprima el máximo número almacenado. Por último, libere la memoria reservada dinámicamente. *Nota: Modularice la reserva de memoria, la inicialización y el cálculo del máximo.*

4. Escriba un programa que lea un número entero ***n*** desde teclado y luego reserve memoria en forma dinámica para un arreglo de ***n*** float. Inicialícelo con valores ingresados por teclado y a continuación calcule e imprima el promedio de todos ellos. Por último, libere la memoria reservada dinámicamente. Nota: Modularice la reserva de memoria, la inicialización y el cálculo del promedio.

5. Escriba un programa que reserve en forma dinámica un arreglo de 100 caracteres. A continuación, lea 10 oraciones utilizando la función **gets** e informe para cada una de ellas la cantidad de letras minúsculas y de letras mayúsculas que la componen. Utilice el arreglo creado como variable temporal para procesar cada oración. Por último, libere la memoria reservada dinámicamente.

6. Defina la estructura de una lista enlazada de enteros. Implemente las siguientes funciones:
   a. Inicializar la lista.
   b. Eliminar todos los elementos de la lista.
   c. Agregar un elemento al principio de la lista.
   d. Agregar un elemento al final de la lista.
   e. Calcular la cantidad de elementos de la lista.
   f. Imprimir todos los elementos separados por coma.

7. Utilizando la estructura y funciones del ejercicio anterior escriba un programa que lea números enteros positivos desde teclado hasta ingresar el número 0. Los números leídos deben ser almacenados en orden ingresado por teclado. Generar una nueva lista con el orden invertido. Imprimir los elementos de cada lista junto con la cantidad de elementos de cada una. Por último, libere la memoria reservada dinámicamente.

8. Utilizando la estructura y funciones de los ejercicios anteriores escriba un programa que lea números enteros desde teclado hasta ingresar el número 0. Luego, vuelva a leer otro número entero desde teclado y elimine de la lista a todos aquellos que sean múltiplos del mismo.

Reserva dinámica de memoria

9. Dado los siguientes programas, analice y responda las preguntas.

Código 1:

```c
#include <stdio.h>
#include <stdlib.h>
#define N 50

int main() {
    int arreglo1[N];
    return 0;
}
```

Código 2:

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

int main() {
    int n;
    scanf("%d",&n);
    /* el usuario ingresa 50 */
    int arreglo3[n];
    return 0;
}
```

Código 3:

```c
#include <stdio.h>
#include <stdlib.h>
#define N 50

int main() {
        int * arreglo2 = (int *) malloc (N*sizeof(int));
        return 0;
}
```

Indique para cada arreglo definido:
   a. ¿Qué tipo de arreglo se está utilizando (estático/dinámico/de longitud variable)?
   b. ¿En qué momento se determina cuánta memoria se va a reservar (compilación/ejecución)?
   c. ¿Cuál es el límite de tamaño de cada tipo de arreglo?
   d. ¿Se puede cambiar el tamaño en ejecución?
   e. ¿Se puede liberar la memoria reservada?
   f. ¿El nombre del arreglo puede apuntar a otra dirección?
   g. ¿Todos los códigos compilan de acuerdo a ANSI C90?

10. Escriba un programa que lea un número entero *n* y luego reserve memoria en forma dinámica para un arreglo de *n* elementos `double`. Inicialice las posiciones del arreglo a partir de valores ingresados por teclado y a continuación imprima el promedio de todos ellos.
    a. Empleando notación de arreglos.
    b. Empleando notación de punteros.

    Por último, libere la memoria reservada.

    Nota: modularice la reserva de memoria, la inicialización, el cálculo del promedio y la liberación de memoria.

11. Escriba un programa que lea desde teclado dos valores enteros *n* y *m*, y luego reserve memoria en forma dinámica para una matriz de enteros de *n* filas por *m* columnas. Inicialice la matriz creada con valores ingresados por teclado. Una vez inicializada, imprima las posiciones de todos aquellos valores múltiplos de 3. Por último, libere la memoria reservada.

    Nota: modularice la reserva de memoria, la inicialización, la impresión de las posiciones con valores múltiplo de 3 y la liberación de memoria.

12. Rehaga el ejercicio anterior utilizando un arreglo unidimensional en lugar de uno bidimensional.

13. Dado el siguiente código:

```
int x=2, y=3;
int * m= (int*) malloc(x*y*sizeof(int));
```

    a. ¿Qué estructura/s se puede/n modelar con la variable *m*?
    b. ¿Se puede utilizar a *m* como si fuera una matriz, simulando el acceso por filas y columnas? En ese caso, ¿cómo se podría acceder a sus elementos?
    c. Escriba un programa que trabaje con una matriz de enteros utilizando las declaraciones anteriores sin importar el orden de acceso a cada dimensión (fila y columna o columna y fila). Cargue a *m* con valores leídos desde teclado y luego imprima sus valores en pantalla para verificar que se cargó correctamente. Por último, libere la memoria reservada.

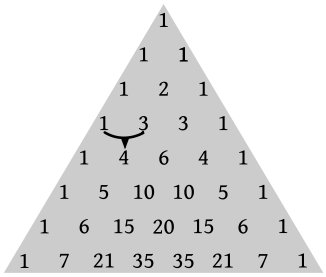
14. En álgebra lineal, una matriz triangular es un tipo especial de matriz cuadrada cuyos elementos por encima o por debajo de su diagonal principal son cero. Una matriz cuadrada de orden *n* se dice que es triangular inferior si es de la forma:

$$L = \begin{pmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{pmatrix}$$

    Escriba un programa que lea desde teclado un valor entero *n* y reserve memoria para una matriz triangular inferior de orden *n* de enteros. Como se desea ahorrar espacio de almacenamiento, no se deben almacenar los elementos cuyo valor es 0, es decir, sólo se reservará memoria para los valores del triángulo inferior de la matriz. Luego, inicialice la estructura con valores aleatorios entre 0 y 20 e imprímala en pantalla. Por último, libere la memoria reservada.

    Nota: modularice la reserva de memoria, la inicialización, la impresión y la liberación de memoria.

15. Escriba un programa que, dado un número entero ***n*** ingresado por teclado, construya una pirámide de pascal de ***n*** filas. Un triángulo de pascal es una serie de filas apiladas que comienza con un elemento y agrega un elemento más en cada fila. El primer y último elemento de cada fila es un 1. Los demás elementos se calculan sumando los 2 números superiores de la fila anterior. Por ejemplo, para obtener el segundo elemento de la cuarta fila (4) deben sumarse el primer (1) y segundo (3) elemento de la tercera fila de forma que 1+3 = 4. Tenga en cuenta las siguientes condiciones:



   a) Utilizar una estructura de datos eficiente. Contemple la información necesaria para recorrer la pirámide y liberar la memoria de forma correcta.
   b) Implemente una función que dado un entero ***n*** retorne una pirámide de pascal de ***n*** filas.
   c) Implemente una función para imprimir la pirámide.
   d) Implemente una función para destruir la pirámide.

16. Escriba un programa que permita conocer todos los divisores de un número entero *n* leído desde teclado. Los números deben almacenarse en un arreglo unidimensional. Como se desea optimizar el espacio a ocupar, la memoria debe reservarse a medida que se la va necesitando. Una vez que se almacenaron todos los divisores, imprímalos junto al número *n*. Por último, libere la memoria reservada.

17. Dado el siguiente código:

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int*** crearTensor(int t){
    int i,j;
    int*** a;

    a = (int***) calloc(t,sizeof(int**));
    for (i=0; i<t; i++){
        a[i] = (int**) calloc(t, sizeof(int*));
        for (j=0; j<t; j++){
            a[i][j] = (int*) calloc(t, sizeof(int));
        }
    }
    return a;
}

int main()
{
    int*** e;
    e = crearTensor(N);

    return 0;
}
```

Analice el código y realice un diagrama de cómo se reserva memoria para la variable *a* y cómo queda la variable *e* luego del llamado al módulo. ¿Por qué es necesario que la variable *a* sea de tipo *int****?