

Práctica 4

Reserva dinámica de memoria

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

Código 1:

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

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

Código 2:

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

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

Código 3:

```
#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:

- ¿Qué tipo de arreglo se está utilizando (estático/dinámico/de longitud variable)?
 - ¿En qué momento se determina cuánta memoria se va a reservar (compilación/ejecución)?
 - ¿Cuál es el límite de tamaño de cada tipo de arreglo?
 - ¿Se puede cambiar el tamaño en ejecución?
 - ¿Se puede liberar la memoria reservada?
 - ¿El nombre del arreglo puede apuntar a otra dirección?
 - ¿Todos los códigos compilan de acuerdo a ANSI C90?
2. 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.
- Empleando notación de arreglos.
 - 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.

3. 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.

4. Dado el siguiente código:

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

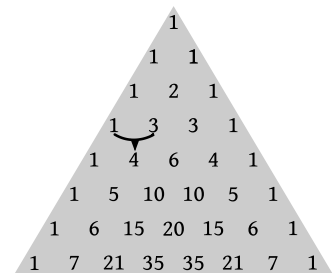
- ¿Qué estructura/s se puede/n modelar con la variable m ?
 - ¿Se puede utilizar a m como si fuera una matriz, simulando el acceso por filas y columnas? En ese caso, ¿cómo se podría acceder a sus elementos?
 - Escriba un programa que trabaje con una matriz de enteros utilizando las declaraciones anteriores 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.
5. Rehaga el ejercicio 3 utilizando un arreglo unidimensional en lugar de uno bidimensional:
- Almacenando la matriz por filas.
 - Almacenando la matriz por columnas.
6. 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:

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

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.

7. 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 tercer fila de forma que $1+3 = 4$. Tenga en cuenta las siguientes condiciones:



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

8. Se desea modelar la información de la población que vive en una ciudad. La información se organiza del siguiente modo: en la ciudad existen barrios y cada barrio tiene una cierta cantidad de personas (no siempre la misma). De cada persona se conoce: DNI, nombre y apellido y sexo. Para optimizar el espacio ocupado, sólo se reserva la memoria que cada barrio necesita para almacenar la información de sus habitantes.

Realice un programa que cargue la información de todas las personas de la ciudad, leyendo de a un barrio por vez. En primer lugar, se lee desde teclado un número entero n , el cual representa la cantidad de barrios de los cuales se leerá información. Por cada barrio se lee primero la cantidad de personas que viven en él y luego la información de sus habitantes. Una vez inicializada la estructura, realice un módulo que imprima el porcentaje de mujeres por cada barrio. Por último, libere la memoria reservada.

Nota: modularice la reserva de memoria, la carga de información, el cálculo del porcentaje y la liberación de memoria.

9. 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.

10. Dado el siguiente código:

```
#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^{***} ?