**Escuela de Ciencias Físicas y Matemáticas – Laboratorio de Simulación**

**Dayrin Idalia Cardona Crúz, 201804753, Laboratorio 4**

---

**Problema 1**

**1) Enunciado**

Ingresar un vector de 10 elementos, llenarlo de números pares del 2 al 20. Al iniciar el programa debe preguntar al usuario cómo quiere ver los números, el menú debe de ser por medio de caracteres: "a" verlos de forma ascendente, "d" descendente, en caso que el usuario escriba otro valor debe de decir que no es correcto y preguntarle el carácter nuevamente, hasta que este sea el correcto, al ingresar el valor correcto muestra el vector en pantalla y termina el programa.

**2) Metodología**

Para resolver este problema se emplean ciclos y sentencias de programación en lenguaje C. Para llenar el vector inicial con números pares se utilizará un ciclo for, cuya estructura es:

```
for(valor inicial; condición; valor siguiente){
        Procedimiento
}
```

A continuación, el usuario debe escoger entre dos tipos de orden diferente para observar su vector. Para obtener el valor de una letra, se utiliza la función **getchar()**. Después, para que mientras el usuario ingrese una letra que no sea a o d se utiliza un ciclo while, cuya estructura es:

```
while(condición) {
        Procedimiento
}
```

Cuando el usuario presione a o d, el programa tendrá que salir del bucle y entrar a evaluarse en dos condiciones que se realizarán por medio de una sentencia if, cuya estructura es:

```
if(condición) entonces {
}
```

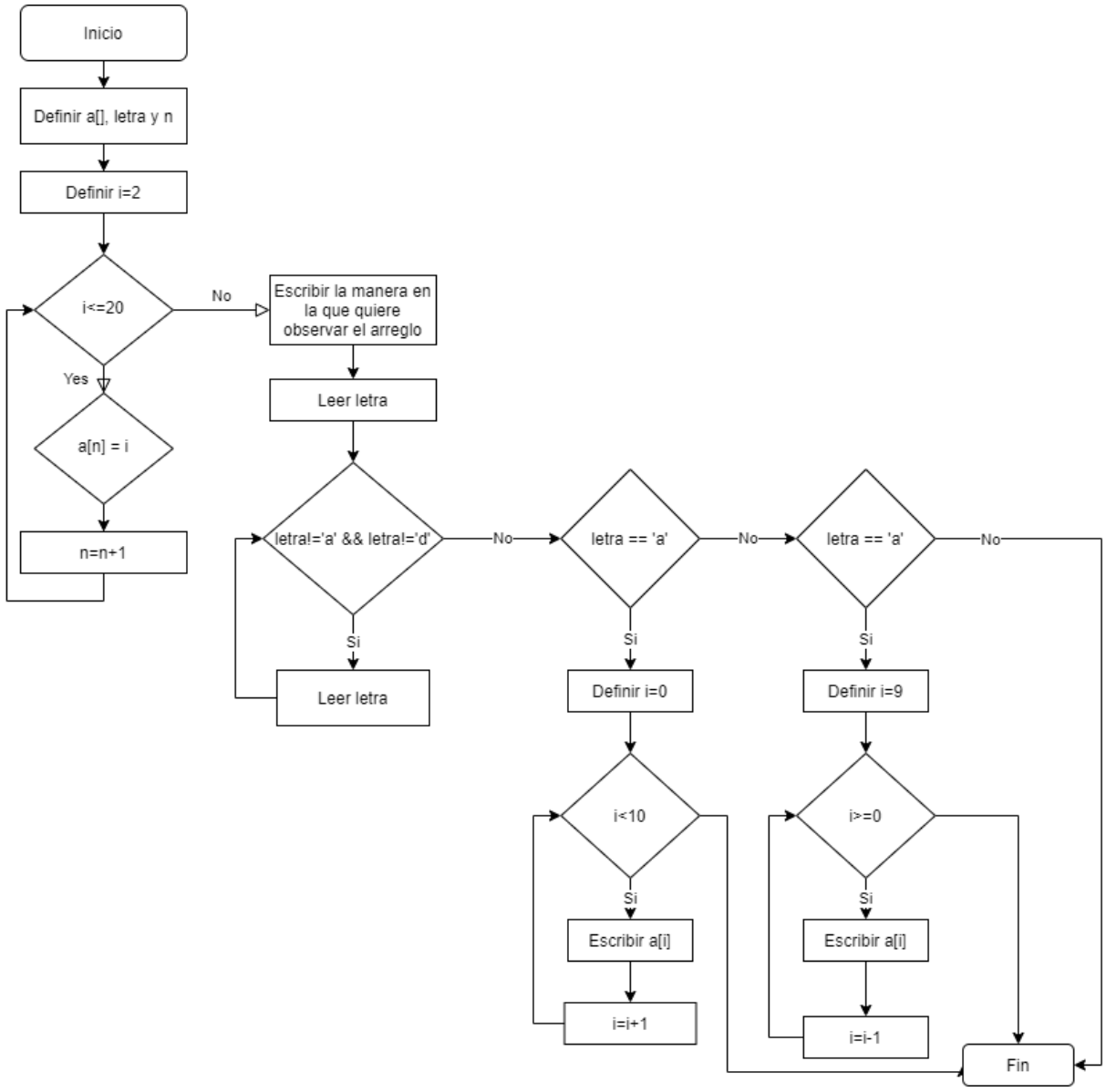
El orden ascendente muestra el vector de menor a mayor. Para mostrar el vector en orden ascendente, el ciclo for debe tener la estructura (int i=0;i<=10;i++). El orden descendente muestra el vector de mayor a menor. Para mostrar el vector en orden descendente, el ciclo for debe tener la estructura (int i=9;i>=0;i--), es decir, la estructura es lo opuesto a la estructura del ciclo for que muestra el vector en orden ascendente.

### 3) Variables de entrada y salida

Existirán dos variables de entrada, una ingresada por el programa y la otra ingresada por el usuario. La variable de entrada ingresada por el programa es un arreglo de tipo entero llamado a[] que almacena todos los números pares del 2 al 20, además, se declara una variable n que indica la posición del arreglo en el for que llena el arreglo con los números pares. Mientras que la variable de entrada ingresada por el usuario es de tipo char llamada letra.

La variable de salida es el mismo vector a[] de tipo entero, pero ordenado de dos formas diferentes, ya sea ascendente o descendentemente, según indique el usuario.

### 4) Diagrama de flujo




### 5) Código

El código se puede encontrar en:
https://github.com/DayrinCardona/LabSimu1S2021DC/blob/main/Laboratorio4/Programa1.c

**Problema 2**

**1) Enunciado**

Crear un programa que solicite al usuario 5 números enteros, estos se deben de guardar en un vector, al terminar de guardar los valores, el programa debe de ordenarlos de forma ascendente y mostrar el vector ordenado.

**2) Metodología**

Un método de ordenación consiste en una operación que dispone un conjunto de datos en algún orden determinado, ya sea en orden ascendente o descendente. El método de selección se considera como un método de ordenación directa. Se busca el menor elemento de la lista y se intercambia con el primer elemento de la lista A[0]:

$A[0]A[1]A[2] \dots A[n-1]$

Después de la primera vuelta, el primer elemento de la lista está ordenado y el resto está desordenado. En la siguiente vuelta de la lista desordenada, se selecciona de nuevo el elemento más pequeño y se almacena en la posición A[1]. La sublista $A[2]A[3] \dots A[n-1]$ continúa desordenada. Se busca el elemento más pequeño y se intercambia con A[2]. Así sucesivamente hasta dar $n$-1 vueltas al vector. Con esto se consigue que el vector quede completamente ordenado.

**3) Variables de entrada y salida**

La variable de entrada es un vector, vector[], de tipo entero con cinco componentes que el usuario ingresa. Y la variable de salida es el mismo vector, vector[], de tipo entero con cinco componentes, pero ordenado de forma ascendente.

**4) Pseudocódigo**

```
INICIO
voidmain()
    Definir vector[5], imenor, aux,
    Definir prototipo de función orden()
    Escribir ("Ingrese cinco números enteros")
    Desde i=0 hasta i=4 repetir
        Leer a[] en la posición i
    Fin Desde
    Ejecutar función orden()
Fin voidmain()
orden()
```

Desde i=0 hasta i=4 repetir

Desde j=i+1 hasta i=4 repetir

Si vector[j]<vector[imenor] entonces

Tomar imenor = j

Fin Desde

Tomar aux = vector[i]

Tomar vector[i] = vector[imenor]

Tomar vector[imenor] = aux

Fin Desde

Escribir ("El nuevo vector ordenado ascendentemente es")

Desde i=0 hasta i=4 repetir

Escribir vector[] en la posición i

Fin Desde

Fin orden()

FIN

**5) Código**

El código puede encontrarse en:
https://github.com/DayrinCardona/LabSimu1S2021DC/blob/main/Laboratorio4/Programa2.c

---

**Problema 3**

**1) Enunciado**

Crear un programa que solicite al usuario dos posiciones en coordenadas (x, y, z) al obtenerlas debe de almacenarlas en dos vectores, el programa automáticamente debe de mostrar los siguientes resultados:

a) magnitud de cada vector

b) suma de los dos vectores

c) producto escalar

d) producto vectorial

**2) Metodología**

La magnitud de un vector con coordenadas $(x,y,z)$ es:

$$|r| = \sqrt{x^2 + y^2 + z^2}$$

La suma de dos vectores con coordenadas $r_1 = (x_1, y_1, z_1)$ y $r_2 = (x_2, y_2, z_2)$, es:

$$r_1 + r_2 = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$

El producto escalar de dos vectores con coordenadas $r_1 = (x_1, y_1, z_1)$ y $r_2 = (x_2, y_2, z_2)$, es:

$$r_1 \cdot r_2 = x_1 x_2 + y_1 y_2 + z_1 z_2$$

El producto vectorial de dos vectores con coordenadas $r_1 = (x_1, y_1, z_1)$ y $r_2 = (x_2, y_2, z_2)$, es:

$$r_1 \times r_2 = (y_1 z_2 - z_1 y_2, z_1 x_2 - x_1 z_2, x_1 y_2 - y_1 x_2)$$

**3) Variables de entrada y salida**

Debido a que solicitaremos dos posiciones en coordenadas, el usuario ingresará dos vectores a1[], a2[] de tipo double. Estas son las variables de entrada.

Las variables de salida son cuatro:

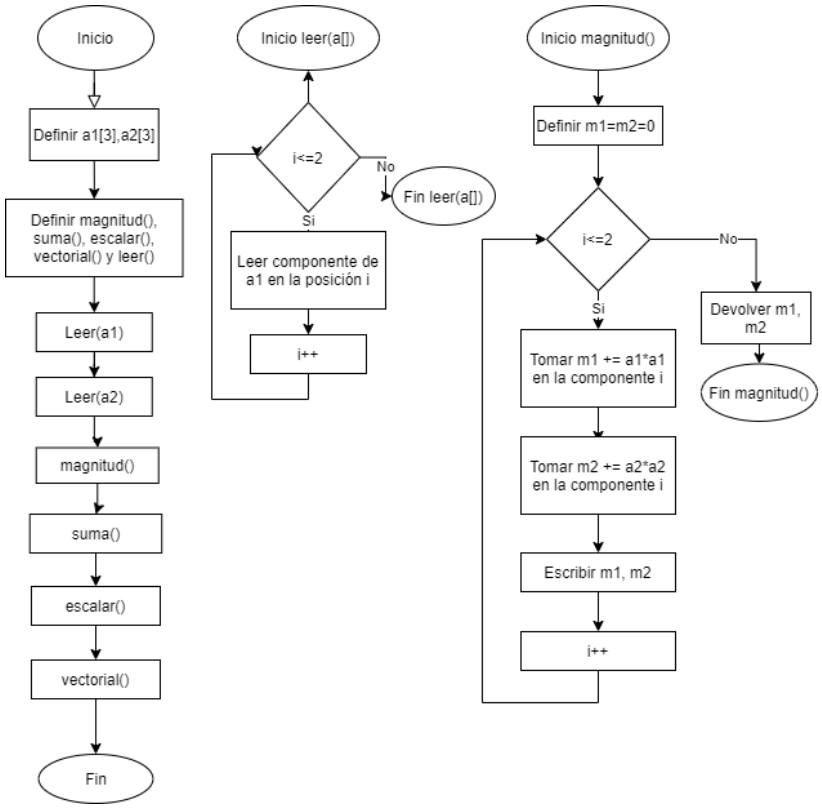
1. La magnitud de los vectores m1, m2, de tipo double, ya que los vectores son de tipo double.

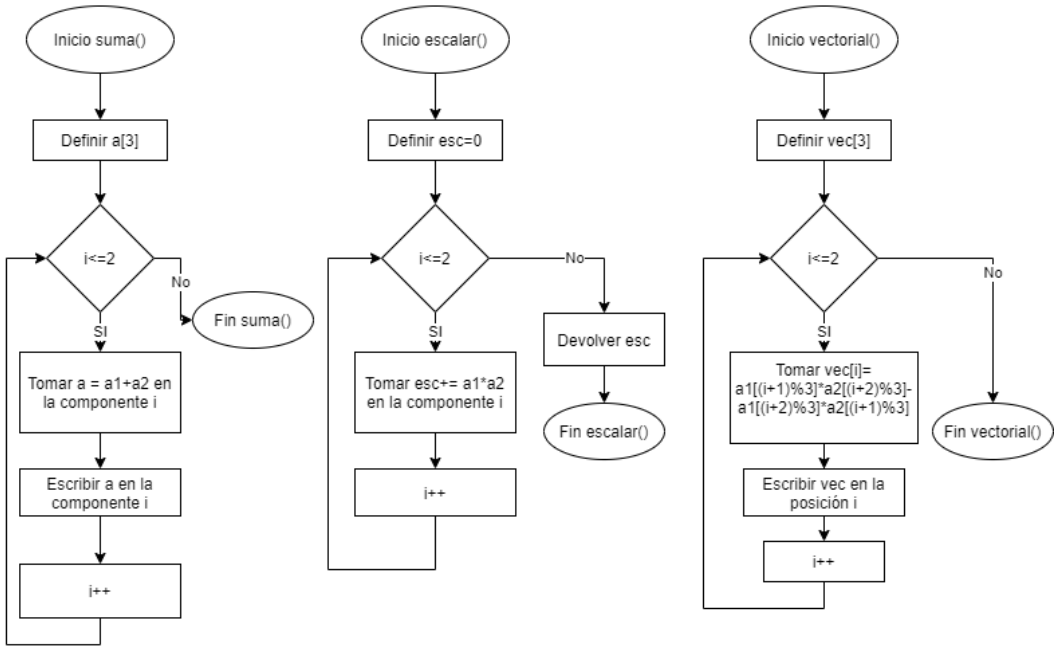
2. La suma de los dos vectores, el cual es otro vector a[] de tipo double.

3. El producto escalar de dos vectores, esc de tipo double.

4. El producto vectorial de dos vectores, otro vector vec[], de tipo double.

**4) Diagrama de flujo**






**5) Código**

El código se puede encontrar en:
https://github.com/DayrinCardona/LabSimu1S2021DC/blob/main/Laboratorio4/Programa3.c

---

**Programa 4**

**1) Enunciado**

Crear un programa que solicite al usuario dos matrices de 3X3 almacenarlas como (matA, matB) y una constante, el programa automáticamente debe de mostrar las los siguientes resultados:

a) matA por constante

b) suma de las dos matrices

c) resta de las dos matrices

d) multiplicación de las dos matrices

e) determinante de matA

f) transpuesta de matB

g) inversa de matA

h) reducción de Gauss de matA

i) reducción de Gauss Jordan matB

**2) Metodología**

Se realizarán varias operaciones con matrices utilizando la teoría matemática de operaciones con matrices. Una suma de matrices de $nxn$ se realiza sumando componente a componente.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{00} + b_{00} & a_{01} + b_{01} & a_{02} + b_{02} \\ a_{10} + b_{10} & a_{11} + b_{11} & a_{12} + b_{12} \\ a_{20} + b_{20} & a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

Similarmente sucede con la resta de matrices de $nxn$.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} - \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{00} - b_{00} & a_{01} - b_{01} & a_{02} - b_{02} \\ a_{10} - b_{10} & a_{11} - b_{11} & a_{12} - b_{12} \\ a_{20} - b_{20} & a_{21} - b_{21} & a_{22} - b_{22} \end{pmatrix}$$

La multiplicación de matices de $nxn$ resulta en una matriz de $nxn$. La matriz multiplicación resulta de multiplicar las componentes de la fila *i* y la columna *i* para la componente *ii* de la matriz. La siguiente ecuación ilustra este último enunciado.

$$c_{ij} = \sum_{k=0}^{2} a_{ik} b_{kj}$$

El determinante de una matriz de *3x3*, se calcula de la siguiente manera:

$$\begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} = a_{00} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} - a_{01} \begin{vmatrix} a_{10} & a_{22} \\ a_{20} & a_{22} \end{vmatrix} + a_{02} \begin{vmatrix} a_{10} & a_{11} \\ a_{20} & a_{21} \end{vmatrix}$$

$$= a_{00}(a_{11}a_{22} - a_{12}a_{21}) - a_{01}(a_{10}a_{22} - a_{22}a_{20}) + a_{02}(a_{10}a_{21} - a_{11}a_{20})$$

La transpuesta de una matriz se calcula a partir de intercambiar las filas por las columnas y las filas por las columnas. La siguiente ecuación ilustra este último enunciado.

$$a_{ij}^{T} = a_{ji}$$

La inversa de una matriz se calcula a partir de las definiciones anteriores. Toma la adjunta de la matriz a, le calcula su matriz adjunta y lo divide por el determinante de la matriz.

$$a^{-1} = \frac{1}{|a|} adj(a^{T})$$

La adjunta de una matriz se puede encontrar de la siguiente manera:

$$adj(a) \begin{pmatrix} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & -\begin{vmatrix} a_{10} & a_{12} \\ a_{20} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{10} & a_{11} \\ a_{20} & a_{21} \end{vmatrix} \\ -\begin{vmatrix} a_{01} & a_{02} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{00} & a_{02} \\ a_{20} & a_{22} \end{vmatrix} & -\begin{vmatrix} a_{00} & a_{01} \\ a_{20} & a_{21} \end{vmatrix} \\ \begin{vmatrix} a_{01} & a_{02} \\ a_{11} & a_{12} \end{vmatrix} & -\begin{vmatrix} a_{00} & a_{02} \\ a_{10} & a_{12} \end{vmatrix} & \begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} \end{pmatrix}$$

Los métodos de Gauss y Gauss Jordan consisten en un conjunto de reglas y operaciones entre filas de matrices, reglas por medio de las cuales las matrices se reducen a matrices escalonadas y matrices

escalonadas reducidas. Las reglas son: multiplicar una fila por una constante, sumar y restar dos filas y dividir una fila por una constante.

La diferencia entre ambos métodos consiste en que el método de Gauss reduce una matriz a una matriz escalonada, triangular superior,

$$\begin{matrix} a_{00} & a_{01} & a_{02} \\ 0 & a_{11} & a_{12} \\ 0 & 0 & a_{22} \end{matrix}$$

mientras que el método de Gauss Jordan reduce una matriz a una matriz escalonada reducida, con unos en la diagonal principal, es decir, una matriz identidad.

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

**3) Variables de entrada y salida**

Las variables de entrada son las matrices de entrada matA y matB que ingresa el usuario.

Las variables de salida son las operaciones realizadas con las matrices:

1. La nueva matriz matA multiplicada multiplicada por una constante de tipo double.
2. La matriz que es la suma de las matrices matA y mat B de tipo double.
3. La matriz que es la resta de las matrices matA y mat B de tipo double.
4. La matriz que es la multiplicación de las matrices matA y mat B de tipo double.
5. El determinante de la matriz matA de tipo double.
6. La transpuesta de la matriz matB de tipo double.
7. La inversa de la matriz matA de tipo double.
8. La misma matriz matA reducida por el método de Gauss a una matriz triangular superior.
9. La misma matriz matB reducida por el método de Gauss Jordan a una matriz identidad.

**4) Pseudocódigo**

INICIO

Declarar variables globales, matA[3][3], matB[3][3], matC[3][3], a[3][3],c

Declarar prototipos de funciones pedirmatrices(), constante(), suma(), resta(), multiplicación(), inversa(), determinante(), transpuesta(), imprimirmatrices()

Inicio void main()

Llamar pedirmatrices()

Llamar imprimirmatrices(matA)

Llamar imprimirmatrices(matB)

```
        Llamar constante()
        Llamar suma()
        Llamar resta()
        Llamar multiplicacion()
        Llamar determinante()
        Llamar transpuesta(matB)
        Llamar inversa()
        Llamar gauss()
        Llamar gaussjordan()
Fin void main()
Inicio void pedirmatrices()
Escribir("Ingrese los elementos de la primera matriz")
        Desde i=0 hasta i=2, repetir
                Desde j=0 hasta j=2, repetir
                        Leer matA en la posición i, j
                        j++
                Fin Desde
                i++
        Fin Desde
        Escribir("Ingrese los elementos de la segunda matriz")
        Desde i=0 hasta i=2, repetir
                Desde j=0 hasta j=2, repetir
                        Leer matB en la posición i, j
                j++
                Fin Desde
                i++
        Fin Desde
Fin void pedirmatrices()
Inicio void imprimirmatrices()
```

```
        Desde i=0 hasta i=2, repetir
                Desde j=0 hasta j=2, repetir
                        Imprimir  matA en la posición  i, j
                        j++
                Fin Desde
                i++
        Fin Desde
Fin void imprimirmatrices()
Inicio  void constante()
        Escribir("a)  matA por una constante")
        Desde i=0 hasta i=2, repetir
                Desde j=0 hasta j=2, repetir
                        Tomar matC en la posición  i,j  igual  a c*matA en la posición  i, j
                        j++
                Fin Desde
                i++
        Fin Desde
Fin void constante()
Inicio  void suma()
        Escribir("b)  Suma de las dos matrices")
        Desde i=0 hasta i=2, repetir
                Desde j=0 hasta j=2, repetir
                        Tomar matC en la posición  i,j  igual  a matA+matB en la posición  i, j
                        j++
                Fin Desde
                i++
        Fin Desde
Fin void suma()
Inicio  void resta()
```

```
        Escribir("c) Resta de las dos matrices")
        Desde i=0 hasta i=2, repetir
                Desde j=0 hasta j=2, repetir
                        Tomar matC en la posición i,j igual a matA-matB en la posición i, j
                        j++
                Fin Desde
                i++
        Fin Desde
Fin void resta()
Inicio void multiplicacion()
        Escribir("d) Multiplicación de las dos matrices")
        Desde i=0 hasta i=2, repetir
                Desde j=0 hasta j=2, repetir
                        Definir C en la posición ij igual a cero
                        Desde k=0 hasta k=2, repetir
                                Tomar C en la posición ij igual a matA en la posición ik*matB
                                en la posición k,j
                                k++
                        Fin Desde
                        j++
                Fin Desde
                i++
        Fin Desde
Fin void multiplicacion()
Inicio void determinante()
        Declarar det=0
        Escribir("e) Determinante de la matriz matA")
        Desde i=0 hasta i=2, repetir
                det+=matA[0][i]*(matA[1][(i+1)%3]*matA[2][(i+2)%3]-
                matA[1][(i+1)%3]*matA[2][(i+2)%3]
```

```
            i++
        Fin Desde
Fin void determinante(double  a[3][3])
 Inicio void transpuesta()
        Escribir("f)  La matriz transpuesta de matA es ")
        Desde i=0 hasta i=2, repetir
            Desde j=0 hasta j=2, repetir
                Tomar matC en la posición  j,i  igual  a a en la posición i,j
                j++
            Fin Desde
            i++
        Fin Desde
Fin void transpuesta()
Inicio  void inversa()
        Cálculo del determinante  de matA
        Definir matD[3][3], matE[3][3]
        Cálculo de la matriz transpuesta de matA, matD
        Cálculo manual de la matriz adjunta  de matD, matE
        Escribir("g)  La matriz inversa de matA es ")
        Desde i=0 hasta i=2, repetir
            Desde j=0 hasta j=2, repetir
                Tomar matE en la posición  i,j  igual a matE en la posición  i,j  dividido
                el determinante
                j++
            Fin Desde
            i++
        Fin Desde
Fin void inversa ()
Inicio  void gauss()
        Escribir("h)  La matriz matA reducida por Gauss es ")
```

Desde i=0 hasta i=2, repetir

Desde j=3 hasta j=0, repetir

matA en la posición i,j es igual a matA en la posición i,j dividido matA en la posición i,i

j--

Fin Desde

Desde k=i+1 hasta k=2, repetir

Desde j=3 hasta j=0, repetir

Tomar matA en la posición k, j igual a la matriz matA en la posición k,j – matA en la posición k,i por matA en la posición i,j

j--

Fin Desde

k++

Fin Desde

i++

Fin Desde

Escribir la matriz matA reducida por Gauss en la pantalla

Fin void gauss()

Inicio void gaussjordan()

Escribir(“i) La matriz matB reducida por Gauss Jordan es ”)

Desde i=0 hasta i=2, repetir

Desde j=3 hasta j=0, repetir

matB en la posición i,j es igual a matB en la posición i,j dividido matB en la posición i,i

j--

Fin Desde

Desde k=i+1 hasta k=2, repetir

Desde j=3 hasta j=0, repetir

Tomar matB en la posición k, j igual a la matriz matB en la posición k,j - matB en la posición k,i por matB en la posición i,j

j--

Fin Desde

k++

Fin Desde

Desde k=0 hasta k=i-1, repetir

Desde j=3 hasta j=0, repetir

Tomar matB en la posición k, j igual a la matriz matB en la posición k,j - matB en la posición k,i por matB en la posición i,j de nuevo

j--

Fin Desde

k++

Fin Desde

i++

Fin Desde

Escribir la matriz matB reducida por Gauss Jordan en la pantalla

Fin void gauss()

FIN

**5) Código**

El código se puede encontrar en:
https://github.com/DayrinCardona/LabSimu1S2021DC/blob/main/Laboratorio4/Programa4.c

---

**Problema 5**

**1) Enunciado**

Crear un programa que encuentre el factorial de un numero entero ingresado, debe de utilizar una funcion recursiva.

**2) Metodología**

Las funciones son conjuntos de instrucciones que pueden llamarse desde cualquier parte del programa. De esta manera, permiten que el programador tenga un grado de abstracción para resolver

el problema. Para resolver este problema, se puede utilizar una función recursiva que se invoque a sí misma en forma directa. El diseño de esta función recursiva debe contemplar la existencia de un caso base o condición de salida, en este caso, será el factorial del número 0, 0!=1.

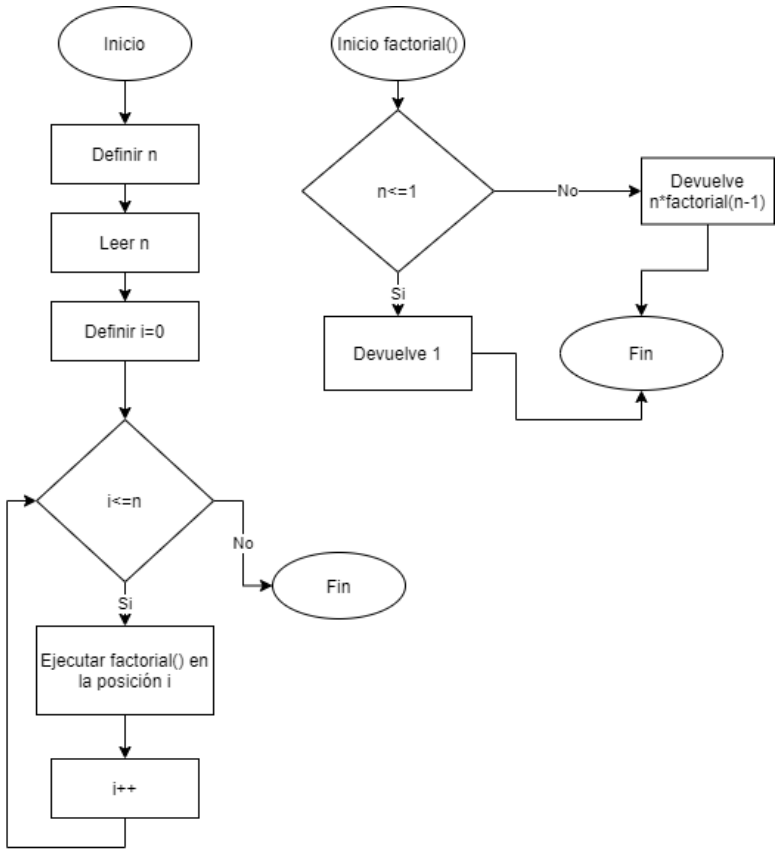
La llamada a esta función se puede realizar utilizando un for para que se ejecute la función tantas veces como indica el número que el usuario ingresa. Este for representa a grandes rasgos la función matemática para calcular el factorial de un número,

$$n! = 1 \times 2 \times 3 \times 4 \times \ldots \times (n-1) \times n.$$

**3) Variables de entrada y salida**

Se calculará el factorial de un número entero, entonces la variable *n* debe ser de tipo entero. Como se utilizará una función recursiva, la salida, es decir el factorial del número, también debe ser de tipo entero.

**4) Diagrama de flujo**




**5) Código**

El código se puede encontrar en:
https://github.com/DayrinCardona/LabSimu1S2021DC/blob/main/Laboratorio4/Programa5.c

**Problema 6**

**1) Enunciado**

Crear un programa que realice la sumatoria desde 1 hasta un número *n* que ingrese el usuario de las siguientes funciones:

**2) Metodología**

El programa se realizará utilizando ciclos for separados por funciones que calculen cada sumatoria para un n ingresado por el usuario y muestre los resultados en pantalla.

a) $\sum_{k=1}^{n} k^2(k-3)$

b) $\sum_{k=2}^{n} \frac{3}{k-1}$

Para esta segunda sumatoria se comenzará en $k$=2 porque, de lo contrario, la función tiene un denominador igual a 0 y, por lo tanto, tiende a infinito.

c) $\sum_{k=1}^{n} \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^k$

d) $\sum_{k=2}^{n} 0.1(3 * 2^{k-2} + 4)$

En programación, una sumatoria se puede representar perfectamente utilizando un ciclo for y una variable de almacenamiento. Para calcular cada sumatoria, es posible definir una variable de almacenamiento y valuar la función de la sumatoria en cada número de la sumatoria en cada ciclo del for.

**3) Variables de entrada y salida**

**4) Pseudocódigo**

```
INICIO
    Definir n
    Definir prototipos funciona(), funcionb(), funcionc(), funciond()
    Inicio int main()
        Escribir "Ingrese número n para realizar las sumatorias"
        Leer n
        Escribir "La sumatoria a es:"
        Llamar funciona()
        Escribir "La sumatoria b es:"
```

```
        Llamar funcionb()
        Escribir “La sumatoria c es:”
        Llamar funcionc()
        Escribir “La sumatoria d es:”
        Llamar funciond()
Fin int main()
Inicio void funciona()
        Definir a=0
        Desde k=1 hasta k<=n, tomar los siguientes pasos
                Tomar a+=k*k*(k-3)
                Tomar k++
        Fin Desde
        Escribir a
Fin void funciona()
Inicio void funcionb()
        Definir b=0
        Desde k=2 hasta k<=n, tomar los siguientes pasos
                Tomar b+=3/(k-1)
                Tomar k++
        Fin Desde
        Escribir b
Fin void funcionb()
Inicio void funcionc()
        Definir c=0
        Desde k=1 hasta k<=n, tomar los siguientes pasos
                Definir exp, exp1, exp2, exp3
                Inicializar  exp1 = sqrt(5)
                Inicializar  exp2 = (1+exp1)/2
                Inicializar  exp3 = (1-exp1)/2
```

Tomar c+=(1/exp1)*(pow(exp2,k)- pow(exp3,k))

Tomar k++

Fin Desde

Escribir c

Fin void funcionc()

Inicio void funciond()

Definir d=0

Desde k=2 hasta k<=n, tomar los siguientes pasos

Definir exp

Tomar exp= pow(2,k-2)

Tomar d+=3*exp+1

Fin Desde

Tomar d=d*0.1

Escribir d

Fin void funciond()

FIN

**5) Código**

El código se puede encontrar en:
https://github.com/DayrinCardona/LabSimu1S2021DC/blob/main/Laboratorio4/Programa6.c