Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Laboratorio de Simulación
Diego Sarceño 201900109
5 de mayo de 2022

Práctica 4

Índice

1. Problema 1

1.1. Enunciado

Debe de ingresar un vector de 10 elementos, llenarlo de números pares del 2 al 20. Al iniciar el programa debe preguntar al usuario como 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.

1.2. Metodología

Por un error de lectura del problema se realizó de modo que la única interacción con el usuario sea la parte de ”ascendente” o ”descendente”. De este modo, se genera una lista con los números pares del 2 al 20 y se realiza un método *shuffle* para randomizar la posición de los elementos de la lista, se utiliza como **seed** el ”time(0)”. Con esto, se utiliza la el método de ordenamiento merge sort. Ya dependiendo de la entrada del usuario se invierte el orden de la lista resultante o no.

1.3. Variables de entrada y salida

→: Caracter ”A” o ”D”.

←: lista de números.

1.4. Pseudocógido o Diagrama de Flujo

1.4.1. Algoritmo Merge Sort

Función MergeSort(vector, incio, final)

Paso 1: Definimos donde queremos ”partir” la lista: half = (inicio + final)/2

Paso 2: Se inicia la recursividad. Llamada a MergeSort(vector, inicio, half)

Paso 3: Llamada a MergeSort(vector, half + 1, final)

Paso 4: Llamada a Merge(vector, inicio, half, final)

Función Merge(vector, inicio, half, final)

Paso 1: Se define una lista auxiliar y los iteradores i = start, j = half + 1, k = 0 y t

Paso 2: Mientras $i \leq$ half && $j \leq$ final, hacer

- **Paso 2.1:** k++. Si vector[i] < vector[j] entonces auxiliar[k] = vector[i], i++.
- **Paso 2.2:** En otro caso auxiliar[k] = vector[j], j++.

Paso 3: Se agrega el elemento sobrante de alguna de las mitades al final de la lista auxiliar. (siempre habrá uno sobrante)

Paso 4: Reescribir todo en el vector original utlizando el contador ”t”.

1.4.2. Algoritmo Shuffle

Función shuffle(vector, length)

Paso 1: Se declaran variables locales "random_ position", "step" y "k = 0"

Paso 2: Mientras int $k <$ length hacer

- **Paso 2.1:** random_ position = numero entero random $\in [0, \text{length} - 1]$
- **Paso 2.2:** step = vector[k]
- **Paso 2.3:** vector[k] = vector[random_ position]
- **Paso 2.4:** vector[random_ position] = step

1.4.3. Main

Paso 1: Generación del vector con elementos pares del 2 al 20

Paso 2: Aplicamos *Shuffle* al vector para aleatorizarlo

Paso 3: Declaramos la variable de inicio del usuario y una variable control = 0

Paso 4: Mientras control == 0 hacer

- **Paso 4.1:** Solicitar el ingreso del caracter "A" o "D"
- **Paso 4.2:** Si es "A", aplicar MergeSort e imprimir el vector resultante. control = 1.
- **Paso 4.3:** Si es "D", aplicar MergeSort e imprimir desde el final del vector hasta el inicio. control = 1.
- **Paso 4.4:** Sino es ninguna, imprimir que no se ingresó ningún valor válido.

1.5. Código

2. Problema 2

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. (utilice un método de ordenación.)

2.2. Metodología

Al igual que el problema anterior se utiliza el algoritmo merge sort, aunque se realizó una implementación de quick-sort y bubble-sort.

2.3. Variables de entrada y salida

→: 5 números **enterios** por parte del usuario.

←: lista ordenada de números.

2.4. Pseudocógido o Diagrama de Flujo

Paso 1: Prototipado de funciones y definición de constantes, en este caso l = 5.

Paso 2: int vector[l]

Paso 3: Ingreso de los números por parte del usuario

Paso 4: Ordenar el vector con el algoritmo Merge Sort mostrado en el problema anterior.

Paso 5: Impresión del vector ordenado.

2.5. Código

3. Problema 3

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

3.2. Metodología

a) magnitud de cada vector: cálculo de magntidud de un vector

$$|\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}.$$

b) suma de los dos vectores: para dos vectores $\vec{A} = (x_1, y_1, z_1)$ y $\vec{B} = (x_2, y_2, z_2)$.

$$\vec{A} + \vec{B} = (x_1 + x_2, y_1 + y_2, z_1 + z_2).$$

c) producto escalar: para los vectores anteriores

$$\langle \vec{A}, \vec{B} \rangle = x_1x_2 + y_1y_2 + z_1z_2.$$

d) producto vectorial, dados los vectores anteriores

$$\begin{aligned} &(y_1z_2 - z_1y_2)\hat{\mathbf{x}} \\ \vec{A} \times \vec{B} = &-(x_1z_2 - z_1x_2)\hat{\mathbf{y}}. \\ &+(x_1y_2 - y_1x_2)\hat{\mathbf{z}} \end{aligned}$$

3.3. Variables de entrada y salida

→: Vectores (x_1, y_1z_1) y (x_2, y_2, z_2)

←: float para la magnitud, array para la suma, float para el producto escalar y array para el producto vectorial.

3.4. Pseudocógido o Diagrama de Flujo

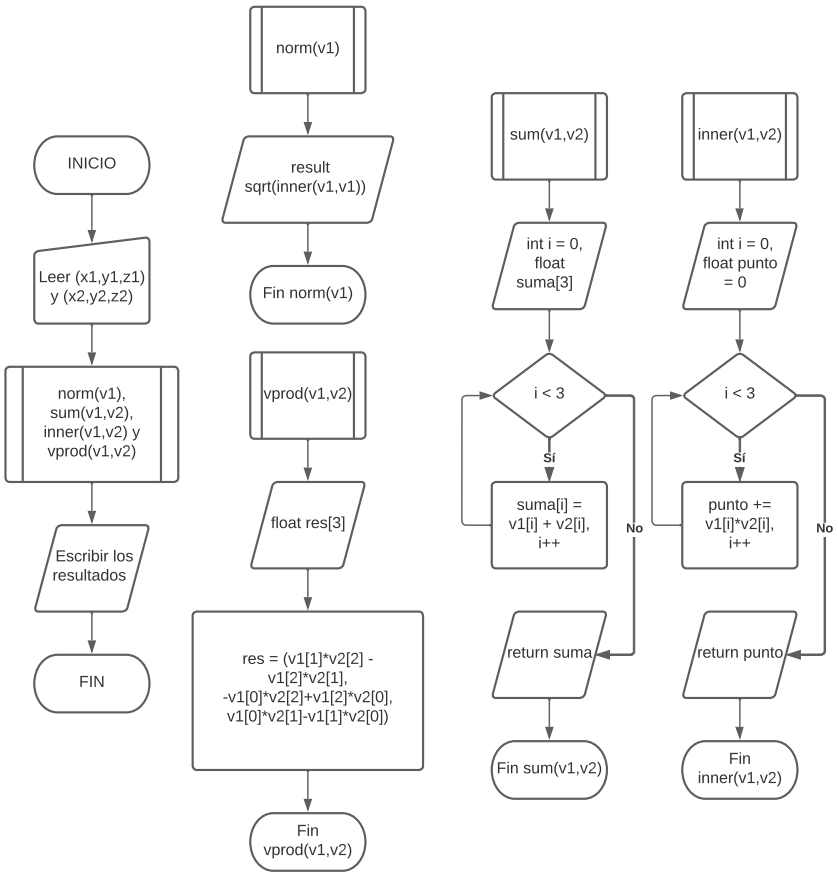


Figura 1: Diagrama de flujo de las diferentes operaciones con vectores.

3.5. Código

4. Problema 4

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) Matriz A por constante

b) Suma de las matrices

c) Resta de las matrices

d) Multiplicación de las matrices

e) Determinante de la matriz A

f) Transpuesta de la matriz B

g) Inversa de la matriz A

h) Reducción de Gauss de la matriz A

i) Reducción de Gauss Jordan de la matriz B

4.2. Metodología

Sean $A = (a_{ij})$ y $B = (b_{ij})$, con ambas matrices 3×3 y $c \in \mathbb{R}$.

a) Matriz A por constante. Cada elemento de la matriz multiplicado por la constante c.

$$cA = (ca_{ij}).$$

b) Suma de las matrices. Suma elemento a elemento, es decir

$$A + B = (a_{ij} + b_{ij}).$$

c) Resta de las matrices. Resta elemento a elemento, i.e.

$$A - B = (a_{ij} - b_{ij}).$$

d) Multiplicación de las matrices. Cada elemento de la matriz resultante $\Gamma = (\gamma_{ij})$ es de la forma

$$\gamma_{ij} = \sum_{k=1}^{n=3} a_{ik} b_{kj}.$$

e) Determinante de la matriz A. Tomando el método de cofactores, se define el menor como el determinante de una matriz M_{ij} que resulta de quitar la $i-$ésima fila y la $j-$ésima columna de la matriz original. Dado esto, se define el cofactor asociacdo a ij, como $\mathcal{A}_{ij} = (-1)^{i+j} M_{ij}$. Y el determinante de la matriz se calcula multiplicando cada entrada de cualquier fila (o columna) por su respectivo cofactor, es decir (tomando la fila 1 como ejemplo)

$$\det\{A\} = \sum_{j=1}^{3} (-1)^{1+j} a_{1j} M_{1j}.$$

Esta fue la implementación que se quizo hacer, dado que no se pudo por problemas en el código, se utilizó la regla de Sarrus

$$\det\{A\} = (a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32})$$

$$-(a_{31}a_{22}a_{13} + a_{32}a_{23}a_{11} + a_{33}a_{12}a_{21}).$$

f) Transpuesta de la matriz B. Se intercambian filas por columnas.

$$B^t = (b_{ji}).$$

g) Inversa de la matriz A. Para calcular la inversa se utiliza la siguiente relación

$$A^{-1} = \frac{1}{\det\{A\}} \text{adj}(A).$$

En donde la matriz adj(A) es la matriz de cofactores transpuesta, en forma visual

$$\text{adj}(A) = \text{cof}(A)^t = \begin{pmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} & \mathcal{A}_{13} \\ \mathcal{A}_{21} & \mathcal{A}_{22} & \mathcal{A}_{23} \\ \mathcal{A}_{31} & \mathcal{A}_{32} & \mathcal{A}_{33} \end{pmatrix}^t.$$

h) Reducción de Gauss de la matriz A.

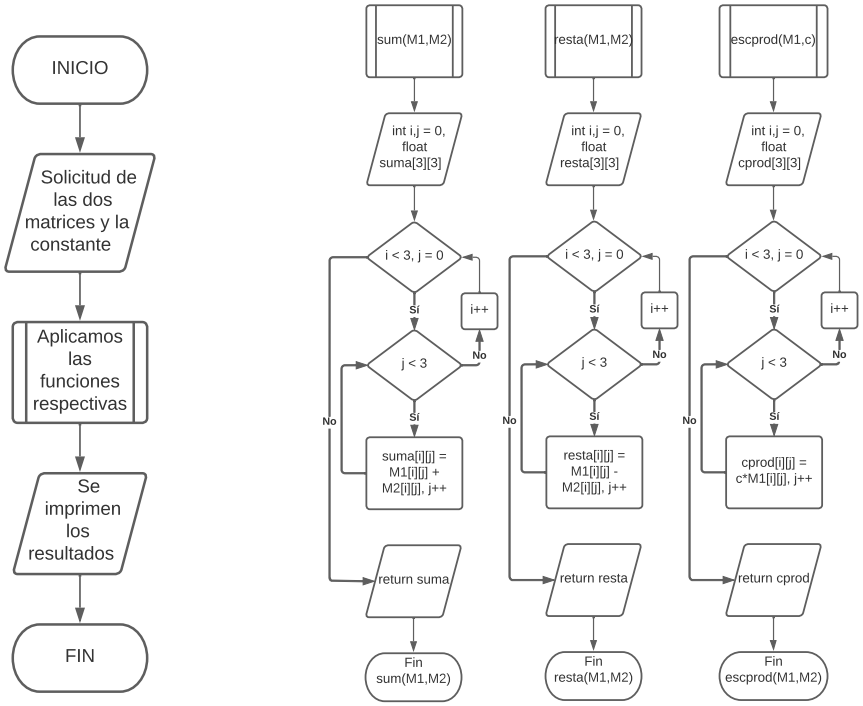
i) Reducción de Gauss Jordan de la matriz B

4.3. Variables de entrada y salida

→: Matrices 3×3 A y B, y constante $c \in \mathbb{R}$.

←: Arrays bidimensionales y constantes, cada una en relación a cada inciso.

4.4. Pseudocógido o Diagrama de Flujo



(a) "Main" (b) Suma, resta y producto por escalar

Figura 2: (a) Diagrama de flujo de la función main. (b) Implementación de la suma y resta de matrices así como el producto por escalar.

4.5. Código

5. Problema 5

5.1. Enunciado

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

5.2. Metodología

La función factorial es una función recursiva definida a tramos

$$n! = \begin{cases} n * (n-1)! & n \geq 2 \\ 1 & n = 0, 1 \\ 0 & \text{en otro caso.} \end{cases}$$

Existen muchas otras definiciones, como la función gamma, pero para interés del problema, utilizamos la definición recursiva.

5.3. Variables de entrada y salida

→: Número entero n.

←: Número entero. (Dado el rápido crecimiento de la función factorial, es probable que para valores, aparentemente, normales de n, ni siquiera los `unsigned long long int` logren poder almacenar el número.)

5.4. Pseudocógido o Diagrama de Flujo

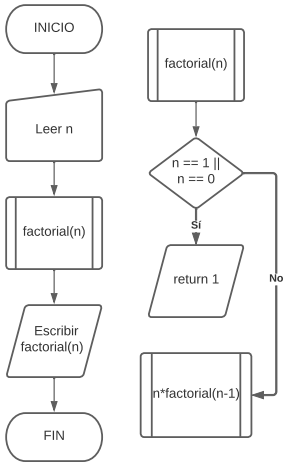


Figura 3: Diagrama de flujo de la implementación recursiva de la función factorial.

5.5. Código

6. Problema 6

6.1. Enunciado

Crear un programa que realice la sumatoria desde 1 hasta un numero n que ingrese el usuario de las siguientes funciones.

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

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

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

6.2. Metodología

No hay mucho que añadir, la sumatoria es un bucle con almacenamiento en una variable que agrega el valor del paso anterior más el de la sucesión valuada en el k correspondiente.

6.3. Variables de entrada y salida

→: Número entero n.

←: Número entero o flotante, según sea el caso de la serie.

6.4. Pseudocógido o Diagrama de Flujo

Paso 1: Prototipado de funciones para cada inciso (por simplicidad fueron nombradas con el inciso correspondiente), cuyas entradas son un número entero.

Paso 2: Ingreso del numero entero n (limite superior de las series).

Paso 3: Para cada función se realizó exactamente el mismo procedimiento, solo que con diferente término de sumatoria, de modo que el ciclo for general es

```
Definir la suma como un float, suma
Mientras un iterador sea menor igual que n, hacer
suma = suma + f_k
aumentar en 1 el iterador
Devolver "suma"
```

Paso 4: Realizar esto para cada función e imprimir los respectivos resultados.

6.5. Código