**Ejercicio 1:**

El primer ciclo paralelo C, cuenta con 28 estudiantes, de los cuáles al finalizar el periodo, la Dirección de la carrera de Computación a solicitado las siguientes estadísticas de la materia INTRODUCCIÓN A LA PROGRAMACIÓN en función a los promedios por estudiante, dichos promedios se deben calcular (ponderar) de 3 calificaciones (ACD que representa el 35% de la nota, APE del 35% y la nota del AA con un peso del 30%). En resumen, los requerimientos son los siguientes:

\*Registre los nombres de cada estudiante de dicho paralelo.

\*Genere aleatoriamente las notas ACD, APE, AA, para cada uno de los 28 estudiantes de 0-10 pts.

\*Calcule el promedio de cada uno de los estudiantes del paralelo dada la siguiente ponderación: ACD->35%, APE->35%, y el AA->30%.

\*Obtenga el promedio del curso, del paralelo C.

\*Liste los nombres de los estudiantes y su nota, que hayan obtenido un promedio por encima del promedio del curso.

\*Liste los nombres de los estudiantes y su nota, que hayan obtenido un promedio por debajo del promedio del curso.

\*Muestre el estudiante con su calificación, si es el del mayor promedio (el más alto de la clase).

\*Muestre el estudiante con su calificación, si es el del menor promedio (el más bajo de la clase).

**ANALISIS:**

Primero realizar una ponderación de las tres aptitudes calificadas por la institución (ACD, APE y AA). Obtener los promedios estudiantiles mediante el uso de acumuladores y auxiliares para poder sacar las notas mayores al promedio y menores a la misma así mismo utilizar auxiliares para obtener la nota más alta y más baja del curso.

**PSEUDOCODIGO:**

// Declarar variables y arreglos

const int CANTIDAD\_ESTUDIANTES = 28

string nombres[CANTIDAD\_ESTUDIANTES]

float notasACD[CANTIDAD\_ESTUDIANTES]

float notasAPE[CANTIDAD\_ESTUDIANTES]

float notasAA[CANTIDAD\_ESTUDIANTES]

float promedios[CANTIDAD\_ESTUDIANTES]

float promedioCurso

// Función para generar aleatoriamente las notas de un estudiante

function generarNotasAleatorias()

// Implementar la generación aleatoria de notas en el rango de 0-10 para ACD, APE y AA

notasACD[i] = random(0, 10)

notasAPE[i] = random(0, 10)

notasAA[i] = random(0, 10)

// Función para calcular el promedio ponderado de un estudiante

function calcularPromedioPonderado(acd, ape, aa)

// Implementar el cálculo del promedio ponderado con los pesos dados

return (acd \* 0.35) + (ape \* 0.35) + (aa \* 0.30)

// Función para mostrar los resultados

function mostrarResultados()

// Calcular el promedio del curso

promedioCurso = sumarElementos(promedios) / CANTIDAD\_ESTUDIANTES

// Mostrar promedio del curso

Escribir ("Promedio del curso: " + promedioCurso)

// Mostrar estudiantes con promedio por encima del promedio del curso

for i = 0 to CANTIDAD\_ESTUDIANTES - 1

if promedios[i] > promedioCurso

Escribir ("Estudiante " + nombres[i] + ": " + promedios[i])

// Mostrar estudiantes con promedio por debajo del promedio del curso

for i = 0 to CANTIDAD\_ESTUDIANTES - 1

if promedios[i] < promedioCurso

Escribir ("Estudiante " + nombres[i] + ": " + promedios[i])

// Encontrar estudiante con el mayor promedio

maxPromedio = encontrarMaximo(promedios)

Escribir("Estudiante con el mayor promedio: " + nombres[maxPromedio] + ": " + promedios[maxPromedio])

// Encontrar estudiante con el menor promedio

minPromedio = encontrarMinimo(promedios)

output("Estudiante con el menor promedio: " + nombres[minPromedio] + ": " + promedios[minPromedio])

// Programa principal

for i = 0 to CANTIDAD\_ESTUDIANTES - 1

// Ingresar nombres de los estudiantes

input("Ingrese el nombre del estudiante " + (i + 1), nombres[i])

// Generar aleatoriamente las notas de cada estudiante

generarNotasAleatorias()

// Calcular el promedio ponderado de cada estudiante

promedios[i] = calcularPromedioPonderado(notasACD[i], notasAPE[i], notasAA[i])

// Mostrar los resultados llamando a la función mostrarResultados()

mostrarResultados()

**Ejercicio 2:**

Crea un programa que gestione el inventario de una tienda. Utiliza una matriz bidimensional para almacenar los productos disponibles en la tienda, con información como nombre, precio y cantidad. El programa debe permitir agregar nuevos productos, actualizar existencias, buscar productos y eliminarlos.

**Análisis:**

Datos de entrada: nombres de productos, precio y cantidad.

Proceso: Funciones para agregar productos buscarlos o eliminarlos.

Salida: Presentar las actualizaciones de los productos.

**Pseudocódigo:**

// Declarar la matriz bidimensional para el inventario

const MAX\_PRODUCTOS = 100

const NUM\_COLUMNAS = 3 // Nombre, Precio, Cantidad

dimension inventario[MAX\_PRODUCTOS][NUM\_COLUMNAS]

// Función para agregar un nuevo producto al inventario

funcion agregarProducto(nombre, precio, cantidad)

// Buscar la primera fila vacía en el inventario

para i = 1 hasta MAX\_PRODUCTOS

si inventario[i][1] == "" entonces

// Agregar el nuevo producto

inventario[i][1] = nombre

inventario[i][2] = precio

inventario[i][3] = cantidad

mostrar("Producto agregado exitosamente.")

retornar

fin si

fin para

mostrar("No hay espacio para más productos en el inventario.")

fin funcion

// Función para actualizar existencias de un producto

funcion actualizarExistencias(nombre, cantidad)

// Buscar el producto en el inventario

para i = 1 hasta MAX\_PRODUCTOS

si inventario[i][1] == nombre entonces

// Actualizar la cantidad

inventario[i][3] = cantidad

mostrar("Existencias actualizadas exitosamente.")

retornar

fin si

fin para

mostrar("Producto no encontrado en el inventario.")

fin funcion

// Función para buscar un producto por nombre

funcion buscarProducto(nombre)

// Buscar el producto en el inventario

para i = 1 hasta MAX\_PRODUCTOS

si inventario[i][1] == nombre entonces

// Mostrar información del producto

mostrar("Nombre: " + inventario[i][1])

mostrar("Precio: " + inventario[i][2])

mostrar("Cantidad: " + inventario[i][3])

retornar

fin si

fin para

mostrar("Producto no encontrado en el inventario.")

fin funcion

// Función para eliminar un producto del inventario

funcion eliminarProducto(nombre)

// Buscar el producto en el inventario

para i = 1 hasta MAX\_PRODUCTOS

si inventario[i][1] == nombre entonces

// Eliminar el producto

inventario[i][1] = ""

inventario[i][2] = 0

inventario[i][3] = 0

mostrar("Producto eliminado exitosamente.")

retornar

fin si

fin para

mostrar("Producto no encontrado en el inventario.")

fin funcion

// Programa principal

// Aquí puedes agregar un menú interactivo para que el usuario elija las operaciones que desea realizar.

// Por ejemplo, agregar producto, actualizar existencias, buscar producto, eliminar producto, etc.

// Ejemplo de uso:

agregarProducto("Producto1", 10.99, 50)

actualizarExistencias("Producto1", 30)

buscarProducto("Producto1")

eliminarProducto("Producto1")

**Ejercicio 3:**

Crea un programa que gestione el inventario de una tienda, así como la emisión de facturas. Utiliza una matriz bidimensional para almacenar los productos disponibles en la tienda, con información como nombre, precio y cantidad. El programa debe permitir facturar un producto dado su código, y unidades deseadas. Adicional se debe agregar a la factura al 12% del IVA, y si la compra superar los $100, se debe aplicar un descuento.

Nota: Considere la alternativa de inexistencias en Stop, para el caso, muestre la alerta respectiva. Crea

**Análisis:**

Datos entrada: nombre productos, precio y cantidad.

Proceso: Iva, descuento

Salida: factura

**Pseudocodigo:**

// Declarar la matriz bidimensional para el inventario

const MAX\_PRODUCTOS = 100

const NUM\_COLUMNAS = 3 // Nombre, Precio, Cantidad

dimension inventario[MAX\_PRODUCTOS][NUM\_COLUMNAS]

// Declarar variables para la factura

dimension factura[MAX\_PRODUCTOS][4] // Código, Nombre, Precio Unitario, Cantidad

real totalFactura

const PORCENTAJE\_IVA = 12

const DESCUENTO\_MINIMO = 100

const PORCENTAJE\_DESCUENTO = 5

// Función para agregar un nuevo producto al inventario

funcion agregarProducto(nombre, precio, cantidad)

// Buscar la primera fila vacía en el inventario

para i = 1 hasta MAX\_PRODUCTOS

si inventario[i][1] == "" entonces

// Agregar el nuevo producto

inventario[i][1] = nombre

inventario[i][2] = precio

inventario[i][3] = cantidad

Escribir ("Producto agregado exitosamente.")

fin si

fin para

Escribir ("No hay espacio para más productos en el inventario.")

fin función

// Función para facturar un producto

funcion facturarProducto(codigo, unidades)

// Buscar el producto en el inventario

para i = 1 hasta MAX\_PRODUCTOS

si inventario[i][1] != "" y codigo == i entonces

// Verificar si hay suficientes unidades en el inventario

si inventario[i][3] >= unidades entonces

// Calcular el subtotal del producto

subtotal = inventario[i][2] \* unidades

// Calcular el monto del IVA

iva = subtotal \* (PORCENTAJE\_IVA / 100)

// Calcular el total antes de aplicar descuento

totalSinDescuento = subtotal + iva

// Verificar si se aplica descuento por superar los $100

si totalSinDescuento > DESCUENTO\_MINIMO entonces

descuento = totalSinDescuento \* (PORCENTAJE\_DESCUENTO / 100)

totalConDescuento = totalSinDescuento - descuento

sino

descuento = 0

totalConDescuento = totalSinDescuento

fin si

// Agregar el producto a la factura

factura[1][1] = codigo

factura[1][2] = inventario[i][1]

factura[1][3] = inventario[i][2]

factura[1][4] = unidades

// Actualizar el inventario restando las unidades facturadas

inventario[i][3] = inventario[i][3] - unidades

// Actualizar el total de la factura

totalFactura = totalFactura + totalConDescuento

Escribir ("Producto facturado exitosamente.")

sino

Escribir ("No hay suficientes unidades en el inventario.")

fin si

fin si

fin para

Escribir ("Producto no encontrado en el inventario.")

fin funcion

// Función para mostrar la factura

funcion mostrarFactura()

// Mostrar la información de cada producto en la factura

para i = 1 hasta MAX\_PRODUCTOS

si factura[i][1] != 0 entonces

Escribir ("Código: " + factura[i][1])

Escribir ("Nombre: " + factura[i][2])

Escribir ("Precio Unitario: " + factura[i][3])

Escribir ("Cantidad: " + factura[i][4])

fin si

fin para

// Mostrar totales

Escribir ("Total Factura: " + totalFactura)

fin funcion

// Programa principal

// Aquí puedes agregar un menú interactivo para que el usuario elija las operaciones que desea realizar.

// Por ejemplo, agregar producto, facturar producto, mostrar factura, etc.

// Ejemplo de uso:

agregarProducto("Producto1", 10.99, 50)

facturarProducto(1, 3)

facturarProducto(2, 2) // Este debería mostrar una alerta de producto no encontrado

mostrarFactura()

**Ejercicio 4:**

Crea un juego de tres en raya utilizando una matriz bidimensional de 3x3. Permita a dos jugadores marcar sus movimientos alternativamente. El juego debe verificar si alguno de los jugadores ha ganado o si hay un empate.

**Análisis:**

Entrada: movimientos

Proceso: encontrar posiciones

Salida: Mostrar empate o ganador

**Pseudocodigo:**

// Declarar la matriz bidimensional para el tablero de juego

dimension tablero[3][3]

const JUGADOR\_X = 'X'

const JUGADOR\_O = 'O'

const VACIO = ' '

// Función para inicializar el tablero

funcion inicializarTablero()

para fila = 1 hasta 3

para columna = 1 hasta 3

tablero[fila][columna] = VACIO

fin para

fin para

fin funcion

// Función para mostrar el tablero de juego

funcion mostrarTablero()

mostrar(" 1 2 3")

para fila = 1 hasta 3

mostrar("-------------")

mostrar(fila)

para columna = 1 hasta 3

mostrar("| " + tablero[fila][columna] + " ")

fin para

mostrar("|")

fin para

mostrar("-------------")

fin funcion

// Función para verificar si hay un ganador

funcion hayGanador(jugador)

// Verificar filas y columnas

para i = 1 hasta 3

si (tablero[i][1] == jugador y tablero[i][2] == jugador y tablero[i][3] == jugador) o

(tablero[1][i] == jugador y tablero[2][i] == jugador y tablero[3][i] == jugador) entonces

retornar verdadero

fin si

fin para

// Verificar diagonales

si (tablero[1][1] == jugador y tablero[2][2] == jugador y tablero[3][3] == jugador) o

(tablero[1][3] == jugador y tablero[2][2] == jugador y tablero[3][1] == jugador) entonces

retornar verdadero

fin si

retornar falso

fin funcion

// Función para verificar si hay un empate

funcion hayEmpate()

para fila = 1 hasta 3

para columna = 1 hasta 3

si tablero[fila][columna] == VACIO entonces

retornar falso

fin si

fin para

fin para

retornar verdadero

fin funcion

// Función para realizar un movimiento

funcion realizarMovimiento(jugador, fila, columna)

si fila >= 1 y fila <= 3 y columna >= 1 y columna <= 3 y tablero[fila][columna] == VACIO entonces

tablero[fila][columna] = jugador

retornar verdadero

sino

retornar falso

fin si

fin funcion

// Programa principal

inicializarTablero()

mostrarTablero()

// Turnos de los jugadores

repetir

// Turno del Jugador X

mostrar("Turno del Jugador X:")

repetir

mostrar("Ingrese la fila (1-3):")

leer filaX

mostrar("Ingrese la columna (1-3):")

leer columnaX

hasta realizarMovimiento(JUGADOR\_X, filaX, columnaX)

mostrarTablero()

// Verificar si el Jugador X ha ganado o hay empate

si hayGanador(JUGADOR\_X) entonces

mostrar("¡Jugador X ha ganado!")

salir

sino si hayEmpate() entonces

mostrar("¡Empate!")

salir

fin si

// Turno del Jugador O

mostrar("Turno del Jugador O:")

repetir

mostrar("Ingrese la fila (1-3):")

leer filaO

mostrar("Ingrese la columna (1-3):")

leer columnaO

hasta realizarMovimiento(JUGADOR\_O, filaO, columnaO)

mostrarTablero()

// Verificar si el Jugador O ha ganado o hay empate

si hayGanador(JUGADOR\_O) entonces

mostrar("¡Jugador O ha ganado!")

salir

sino si hayEmpate() entonces

mostrar("¡Empate!")

salir

fin si

hasta falso

**Ejercicio 5:**

Desarrolla una solución que permita multiplicar dos matrices bidimensionales. El usuario debe ingresar las dimensiones y los elementos de ambas matrices. Realiza la multiplicación y muestra el resultado al final. Nota: Las matrices se deben llenar con número aleatorios únicos enteros positivos o negativos (-9-0, 0-9).

**Análisis:**

Entrada: Datos matrices y dimensiones.

Proceso: Multiplicación de las matrices.

Salida: Mostrar resultado.

**Pseudocodigo:**

// Declarar funciones

funcion llenarMatrizAleatoria(filas, columnas, minimo, maximo)

dimension matriz[filas][columnas]

numerosGenerados = {}

para fila = 1 hasta filas

para columna = 1 hasta columnas

repetir

numeroAleatorio = aleatorio(minimo, maximo)

hasta no contiene(numerosGenerados, numeroAleatorio)

// Almacenar el número generado en la lista para evitar repeticiones

agregarALista(numerosGenerados, numeroAleatorio)

matriz[fila][columna] = numeroAleatorio

fin para

fin para

retornar matriz

fin funcion

funcion mostrarMatriz(matriz, filas, columnas)

para fila = 1 hasta filas

para columna = 1 hasta columnas

mostrar(matriz[fila][columna] + " ")

fin para

mostrar("") // Nueva línea para separar las filas

fin para

fin funcion

funcion multiplicarMatrices(matriz1, filas1, columnas1, matriz2, filas2, columnas2)

// Verificar si las matrices son multiplicables

si columnas1 != filas2 entonces

mostrar("Error: Las matrices no son multiplicables.")

retornar

fin si

// Inicializar matriz resultado

dimension resultado[filas1][columnas2]

// Inicializar la matriz resultado con ceros

para fila = 1 hasta filas1

para columna = 1 hasta columnas2

resultado[fila][columna] = 0

fin para

fin para

// Realizar la multiplicación

para i = 1 hasta filas1

para j = 1 hasta columnas2

para k = 1 hasta columnas1

resultado[i][j] = resultado[i][j] + (matriz1[i][k] \* matriz2[k][j])

fin para

fin para

fin para

retornar resultado

fin funcion

// Programa principal

mostrar("Ingrese las dimensiones de la primera matriz:")

mostrar("Filas:")

leer filas1

mostrar("Columnas:")

leer columnas1

mostrar("Ingrese las dimensiones de la segunda matriz:")

mostrar("Filas:")

leer filas2

mostrar("Columnas:")

leer columnas2

// Llenar las matrices con números aleatorios únicos

matriz1 = llenarMatrizAleatoria(filas1, columnas1, -9, 9)

matriz2 = llenarMatrizAleatoria(filas2, columnas2, -9, 9)

// Mostrar las matrices generadas

mostrar("Primera matriz:")

mostrarMatriz(matriz1, filas1, columnas1)

mostrar("Segunda matriz:")

mostrarMatriz(matriz2, filas2, columnas2)

// Multiplicar las matrices

matrizResultado = multiplicarMatrices(matriz1, filas1, columnas1, matriz2, filas2, columnas2)

// Mostrar el resultado

mostrar("Matriz Resultado:")

mostrarMatriz(matrizResultado, filas1, columnas2)

**PSEUDOCODIGO:**