PRACTICA – PROGRAMACIÓN I

Ejercicio 1: Matriz de Calificaciones
Enunciado:
Escribe un programa que maneje las calificaciones de una clase. Se debe utilizar un array bidimensional donde cada fila representa a un estudiante y cada columna representa una asignatura.

- Crea una función que reciba el array bidimensional y calcule el promedio de calificaciones de cada estudiante.
- Crea una función que ordene las calificaciones de cada estudiante de mayor a menor.
- Implementa un método de búsqueda que permita encontrar la calificación más alta en una asignatura específica.
- Muestra el promedio de calificaciones de cada estudiante y la calificación más alta de una asignatura específica ingresada por el usuario.

c++

```
#include <iostream>
#include <vector>
#include <algorithm>

std::vector<double> calcularPromedios(const std::vector<std::vector<int>>& calificaciones) {
  std::vector<double> promedios;
  for (const auto& estudiante : calificaciones) {
    double suma = 0;
    for (int calificacion : estudiante) {
      suma += calificacion;
    }
    promedios.push_back(suma / estudiante.size());
  }
  return promedios;
}


void ordenarCalificaciones(std::vector<std::vector<int>>& calificaciones) {
  for (auto& estudiante : calificaciones) {
    std::sort(estudiante.begin(), estudiante.end(), std::greater<int>());
  }
}


int encontrarCalificacionMaxima(const std::vector<std::vector<int>>& calificaciones, int asignatura)
{
  int maxCalificacion = -1;
  for (const auto& estudiante : calificaciones) {
    if (asignatura < estudiante.size()) {
      if (estudiante[asignatura] > maxCalificacion) {
        maxCalificacion = estudiante[asignatura];
```

```
            }
        }
    }
    return maxCalificacion;
}

int main() {
    std::vector<std::vector<int>> calificaciones = {
        {85, 90, 78},
        {88, 76, 92},
        {70, 80, 85}
    };


    std::vector<double> promedios = calcularPromedios(calificaciones);
    std::cout << "Promedios de cada estudiante:\n";
    for (size_t i = 0; i < promedios.size(); ++i) {
        std::cout << "Estudiante " << i + 1 << ": " << promedios[i] << "\n";
    }


    ordenarCalificaciones(calificaciones);


    std::cout << "\nCalificaciones ordenadas:\n";
    for (size_t i = 0; i < calificaciones.size(); ++i) {
        std::cout << "Estudiante " << i + 1 << ": ";
        for (int calificacion : calificaciones[i]) {
            std::cout << calificacion << " ";
        }
        std::cout << "\n";
    }


    int asignatura;
    std::cout << "\nIngrese el numero de la asignatura (0 a 2): ";
    std::cin >> asignatura;
    int calificacionMaxima = encontrarCalificacionMaxima(calificaciones, asignatura);
    std::cout << "La calificacion mas alta en la asignatura " << asignatura << " es: " <<
calificacionMaxima << "\n";

    return 0;
}
```

Python:

```
def calcular_promedios(calificaciones):
    promedios = []
    for estudiante in calificaciones:
```

```
        promedio = sum(estudiante) / len(estudiante)
        promedios.append(promedio)
    return promedios

def ordenar_calificaciones(calificaciones):
    for estudiante in calificaciones:
        estudiante.sort(reverse=True)

def encontrar_calificacion_maxima(calificaciones, asignatura):
    max_calificacion = -1
    for estudiante in calificaciones:
        if asignatura < len(estudiante):
            if estudiante[asignatura] > max_calificacion:
                max_calificacion = estudiante[asignatura]
    return max_calificacion

def main():
    calificaciones = [
        [85, 90, 78],
        [88, 76, 92],
        [70, 80, 85]
    ]


    promedios = calcular_promedios(calificaciones)
    print("Promedios de cada estudiante:")
    for i, promedio in enumerate(promedios):
        print(f"Estudiante {i + 1}: {promedio:.2f}")


    ordenar_calificaciones(calificaciones)


    print("\nCalificaciones ordenadas:")
    for i, estudiante in enumerate(calificaciones):
        print(f"Estudiante {i + 1}: {' '.join(map(str, estudiante))}")


    asignatura = int(input("\nIngrese el número de la asignatura (0 a 2): "))
    calificacion_maxima = encontrar_calificacion_maxima(calificaciones, asignatura)
    print(f"La calificación más alta en la asignatura {asignatura} es: {calificacion_maxima}")

if __name__ == "__main__":
    main()
```

Ejercicio 2: Inventario de Tienda
Enunciado:
Escribe un programa para gestionar el inventario de una tienda. Usa un array bidimensional donde cada fila representa un producto y las columnas representan el código del producto, nombre del producto y la cantidad en stock.

- Crea una función para agregar un nuevo producto al inventario.
- Crea una función que ordene el inventario por el nombre del producto en orden alfabético.
- Implementa un método de búsqueda que permita encontrar un producto por su código.
- Desarrolla un proceso que permita reducir la cantidad en stock de un producto cuando se realiza una venta y otro que aumente la cantidad cuando se recibe nuevo stock.

c++:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

struct Producto {
  int codigo;
  std::string nombre;
  int cantidad;
};

void agregarProducto(std::vector<Producto>& inventario, int codigo, const std::string& nombre, int cantidad) {
  inventario.push_back({codigo, nombre, cantidad});
}

void ordenarInventario(std::vector<Producto>& inventario) {
  std::sort(inventario.begin(), inventario.end(), [](const Producto& a, const Producto& b) {
    return a.nombre < b.nombre;
  });
}

Producto* buscarProductoPorCodigo(std::vector<Producto>& inventario, int codigo) {
  for (auto& producto : inventario) {
    if (producto.codigo == codigo) {
      return &producto;
    }
  }
  return nullptr;
}

void venderProducto(std::vector<Producto>& inventario, int codigo, int cantidadVendida) {
  Producto* producto = buscarProductoPorCodigo(inventario, codigo);
  if (producto != nullptr) {
    if (producto->cantidad >= cantidadVendida) {
```

```
            producto->cantidad -= cantidadVendida;
        } else {
            std::cout << "Stock insuficiente para la venta.\n";
        }
    } else {
        std::cout << "Producto no encontrado.\n";
    }
}

void recibirNuevoStock(std::vector<Producto>& inventario, int codigo, int cantidadRecibida) {
    Producto* producto = buscarProductoPorCodigo(inventario, codigo);
    if (producto != nullptr) {
        producto->cantidad += cantidadRecibida;
    } else {
        std::cout << "Producto no encontrado.\n";
    }
}

int main() {
    std::vector<Producto> inventario;


    agregarProducto(inventario, 101, "Manzanas", 50);
    agregarProducto(inventario, 102, "Peras", 30);
    agregarProducto(inventario, 103, "Platanos", 100);

    ordenarInventario(inventario);
    std::cout << "Inventario ordenado por nombre:\n";
    for (const auto& producto : inventario) {
        std::cout << "Codigo: " << producto.codigo << ", Nombre: " << producto.nombre << ", Cantidad:
" << producto.cantidad << "\n";
    }

    int codigoBuscado = 102;
    Producto* productoBuscado = buscarProductoPorCodigo(inventario, codigoBuscado);
    if (productoBuscado != nullptr) {
        std::cout << "Producto encontrado: " << productoBuscado->nombre << " con cantidad: " <<
productoBuscado->cantidad << "\n";
    } else {
        std::cout << "Producto no encontrado.\n";
    }

    venderProducto(inventario, 101, 20);

    recibirNuevoStock(inventario, 103, 50);

    std::cout << "\nInventario actualizado:\n";
    for (const auto& producto : inventario) {
```

```
        std::cout << "Codigo: " << producto.codigo << ", Nombre: " << producto.nombre << ", Cantidad:
" << producto.cantidad << "\n";
    }

    return 0;
}
```

Python:

```
class Producto:
    def __init__(self, codigo, nombre, cantidad):
        self.codigo = codigo
        self.nombre = nombre
        self.cantidad = cantidad

def agregar_producto(inventario, codigo, nombre, cantidad):
    inventario.append(Producto(codigo, nombre, cantidad))

def ordenar_inventario(inventario):
    inventario.sort(key=lambda producto: producto.nombre)

def buscar_producto_por_codigo(inventario, codigo):
    for producto in inventario:
        if producto.codigo == codigo:
            return producto
    return None

def vender_producto(inventario, codigo, cantidad_vendida):
    producto = buscar_producto_por_codigo(inventario, codigo)
    if producto:
        if producto.cantidad >= cantidad_vendida:
            producto.cantidad -= cantidad_vendida
        else:
            print("Stock insuficiente para la venta.")
    else:
        print("Producto no encontrado.")

def recibir_nuevo_stock(inventario, codigo, cantidad_recibida):
    producto = buscar_producto_por_codigo(inventario, codigo)
    if producto:
        producto.cantidad += cantidad_recibida
    else:
        print("Producto no encontrado.")

def main():
    inventario = []


    agregar_producto(inventario, 101, "Manzanas", 50)
```

```
    agregar_producto(inventario, 102, "Peras", 30)
    agregar_producto(inventario, 103, "Platanos", 100)


    ordenar_inventario(inventario)
    print("Inventario ordenado por nombre:")
    for producto in inventario:
        print(f"Codigo: {producto.codigo}, Nombre: {producto.nombre}, Cantidad: {producto.cantidad}")


    codigo_buscado = 102
    producto_buscado = buscar_producto_por_codigo(inventario, codigo_buscado)
    if producto_buscado:
        print(f"Producto encontrado: {producto_buscado.nombre} con cantidad: {producto_buscado.cantidad}")
    else:
        print("Producto no encontrado.")


    vender_producto(inventario, 101, 20)


    recibir_nuevo_stock(inventario, 103, 50)


    print("\nInventario actualizado:")
    for producto in inventario:
        print(f"Codigo: {producto.codigo}, Nombre: {producto.nombre}, Cantidad: {producto.cantidad}")

if __name__ == "__main__":
    main()
```

Ejercicio 3: Tablero de Juego
Enunciado:
Diseña un programa para un juego de mesa que use un tablero de 10x10 representado por un array bidimensional.

- Crea una función para inicializar el tablero con ceros.
- Crea una función que permita colocar fichas en el tablero, recibiendo la posición (fila, columna) y el tipo de ficha (1 o 2).
- Implementa un método de búsqueda que verifique si hay una línea horizontal, vertical o diagonal de 4 fichas del mismo tipo.
- Desarrolla un proceso que imprima el estado actual del tablero en consola.

c++:

```
#include <iostream>
#include <vector>
```

```
void inicializarTablero(std::vector<std::vector<int>>& tablero) {
    for (auto& fila : tablero) {
        std::fill(fila.begin(), fila.end(), 0);
    }
}

bool colocarFicha(std::vector<std::vector<int>>& tablero, int fila, int columna, int tipoFicha) {
    if (fila >= 0 && fila < 10 && columna >= 0 && columna < 10 && (tipoFicha == 1 || tipoFicha == 2))
{
        if (tablero[fila][columna] == 0) {
            tablero[fila][columna] = tipoFicha;
            return true;
        } else {
            std::cout << "La posición ya está ocupada.\n";
        }
    } else {
        std::cout << "Posición o tipo de ficha inválido.\n";
    }
    return false;
}

void imprimirTablero(const std::vector<std::vector<int>>& tablero) {
    for (const auto& fila : tablero) {
        for (int celda : fila) {
            std::cout << celda << " ";
        }
        std::cout << "\n";
    }
}

bool verificarLinea(const std::vector<std::vector<int>>& tablero, int tipoFicha) {
    for (const auto& fila : tablero) {
        for (int col = 0; col <= 6; ++col) {
            if (fila[col] == tipoFicha && fila[col + 1] == tipoFicha && fila[col + 2] == tipoFicha && fila[col +
3] == tipoFicha) {
                return true;
            }
        }
    }
    for (int col = 0; col < 10; ++col) {
        for (int fila = 0; fila <= 6; ++fila) {
            if (tablero[fila][col] == tipoFicha && tablero[fila + 1][col] == tipoFicha && tablero[fila + 2][col]
== tipoFicha && tablero[fila + 3][col] == tipoFicha) {
                return true;
            }
        }
    }
    for (int fila = 0; fila <= 6; ++fila) {
```

```
      for (int col = 0; col <= 6; ++col) {
        if (tablero[fila][col] == tipoFicha && tablero[fila + 1][col + 1] == tipoFicha && tablero[fila +
2][col + 2] == tipoFicha && tablero[fila + 3][col + 3] == tipoFicha) {
          return true;
        }
        if (tablero[fila + 3][col] == tipoFicha && tablero[fila + 2][col + 1] == tipoFicha && tablero[fila +
1][col + 2] == tipoFicha && tablero[fila][col + 3] == tipoFicha) {
          return true;
        }
      }
    }
    return false;
}

int main() {
    std::vector<std::vector<int>> tablero(10, std::vector<int>(10));
    inicializarTablero(tablero);
    colocarFicha(tablero, 0, 0, 1);
    colocarFicha(tablero, 0, 1, 1);
    colocarFicha(tablero, 0, 2, 1);
    colocarFicha(tablero, 0, 3, 1);
    imprimirTablero(tablero);
    if (verificarLinea(tablero, 1)) {
        std::cout << "Hay una línea de 4 fichas del tipo 1.\n";
    } else {
        std::cout << "No hay ninguna línea de 4 fichas del tipo 1.\n";
    }
    return 0;
}
Python
def inicializar_tablero():
    return [[0 for _ in range(10)] for _ in range(10)]

def colocar_ficha(tablero, fila, columna, tipo_ficha):
    if 0 <= fila < 10 and 0 <= columna < 10 and tipo_ficha in (1, 2):
        if tablero[fila][columna] == 0:
            tablero[fila][columna] = tipo_ficha
            return True
        else:
            print("La posición ya está ocupada.")
    else:
        print("Posición o tipo de ficha inválido.")
    return False

def imprimir_tablero(tablero):
    for fila in tablero:
        print(" ".join(map(str, fila)))
```

```
def verificar_linea(tablero, tipo_ficha):
    for fila in tablero:
        for col in range(7):
            if fila[col] == tipo_ficha and fila[col + 1] == tipo_ficha and fila[col + 2] == tipo_ficha and fila[col + 3] == tipo_ficha:
                return True
    for col in range(10):
        for fila in range(7):
            if tablero[fila][col] == tipo_ficha and tablero[fila + 1][col] == tipo_ficha and tablero[fila + 2][col] == tipo_ficha and tablero[fila + 3][col] == tipo_ficha:
                return True
    for fila in range(7):
        for col in range(7):
            if (tablero[fila][col] == tipo_ficha and tablero[fila + 1][col + 1] == tipo_ficha and
                    tablero[fila + 2][col + 2] == tipo_ficha and tablero[fila + 3][col + 3] == tipo_ficha):
                return True
            if (tablero[fila + 3][col] == tipo_ficha and tablero[fila + 2][col + 1] == tipo_ficha and
                    tablero[fila + 1][col + 2] == tipo_ficha and tablero[fila][col + 3] == tipo_ficha):
                return True
    return False

def main():
    tablero = inicializar_tablero()
    colocar_ficha(tablero, 0, 0, 1)
    colocar_ficha(tablero, 0, 1, 1)
    colocar_ficha(tablero, 0, 2, 1)
    colocar_ficha(tablero, 0, 3, 1)
    imprimir_tablero(tablero)
    if verificar_linea(tablero, 1):
        print("Hay una línea de 4 fichas del tipo 1.")
    else:
        print("No hay ninguna línea de 4 fichas del tipo 1.")

if __name__ == "__main__":
    main()
```

Ejercicio 4: Registro de Temperaturas

Enunciado:

Desarrolla un programa que registre las temperaturas diarias de una ciudad durante un mes (30 días) en diferentes horas del día (4 mediciones diarias: mañana, mediodía, tarde, noche) utilizando un array bidimensional.

- Crea una función que reciba el array bidimensional y calcule la temperatura promedio diaria.
- Crea una función que ordene las temperaturas de cada día de menor a mayor.
- Implementa un método de búsqueda que encuentre el día con la temperatura más alta y la más baja.
- Muestra las temperaturas promedio diarias y el día con la temperatura más alta y más baja.

c++:

```
#include <iostream>
#include <vector>
#include <algorithm>

void calcularPromedioDiario(const std::vector<std::vector<double>>& temperaturas,
std::vector<double>& promedios) {
  for (const auto& dia : temperaturas) {
    double suma = 0;
    for (double temp : dia) {
      suma += temp;
    }
    promedios.push_back(suma / dia.size());
  }
}

void ordenarTemperaturasDiarias(std::vector<std::vector<double>>& temperaturas) {
  for (auto& dia : temperaturas) {
    std::sort(dia.begin(), dia.end());
  }
}

void encontrarDiaTemperaturaExtrema(const std::vector<std::vector<double>>& temperaturas,
int& diaMax, int& diaMin) {
  double maxTemp = temperaturas[0][0];
  double minTemp = temperaturas[0][0];
  for (int i = 0; i < temperaturas.size(); ++i) {
    for (double temp : temperaturas[i]) {
      if (temp > maxTemp) {
        maxTemp = temp;
        diaMax = i;
      }
      if (temp < minTemp) {
        minTemp = temp;
        diaMin = i;
      }
    }
  }
}

int main() {
  std::vector<std::vector<double>> temperaturas(30, std::vector<double>(4));
  std::vector<double> promedios;


  for (int i = 0; i < 30; ++i) {
    for (int j = 0; j < 4; ++j) {
      temperaturas[i][j] = 15 + rand() % 15;
```

```
    }
  }

  calcularPromedioDiario(temperaturas, promedios);
  ordenarTemperaturasDiarias(temperaturas);

  int diaMax = 0, diaMin = 0;
  encontrarDiaTemperaturaExtrema(temperaturas, diaMax, diaMin);

  std::cout << "Promedios diarios:\n";
  for (double prom : promedios) {
    std::cout << prom << "\n";
  }

  std::cout << "Día con la temperatura más alta: " << diaMax + 1 << "\n";
  std::cout << "Día con la temperatura más baja: " << diaMin + 1 << "\n";

  return 0;
}
```

Python:

```
import random

def calcular_promedio_diario(temperaturas):
  promedios = []
  for dia in temperaturas:
    promedios.append(sum(dia) / len(dia))
  return promedios

def ordenar_temperaturas_diarias(temperaturas):
  for dia in temperaturas:
    dia.sort()

def encontrar_dia_temperatura_extrema(temperaturas):
  dia_max = dia_min = 0
  max_temp = min_temp = temperaturas[0][0]
  for i, dia in enumerate(temperaturas):
    for temp in dia:
      if temp > max_temp:
        max_temp = temp
        dia_max = i
      if temp < min_temp:
        min_temp = temp
        dia_min = i
  return dia_max, dia_min

def main():
  temperaturas = [[15 + random.randint(0, 14) for _ in range(4)] for _ in range(30)]
```

```
    promedios = calcular_promedio_diario(temperaturas)
    ordenar_temperaturas_diarias(temperaturas)
    dia_max, dia_min = encontrar_dia_temperatura_extrema(temperaturas)

    print("Promedios diarios:")
    for prom in promedios:
        print(prom)

    print(f"Día con la temperatura más alta: {dia_max + 1}")
    print(f"Día con la temperatura más baja: {dia_min + 1}")

if __name__ == "__main__":
    main()
```

Ejercicio 5: Matriz de Distancias
Enunciado:
Escribe un programa para calcular las distancias entre varias ciudades. Usa un array bidimensional donde cada celda [i][j] representa la distancia de la ciudad i a la ciudad j.

- Crea una función para ingresar las distancias en la matriz.
- Crea una función que ordene las ciudades por su distancia total a todas las demás (sumando las distancias en cada fila).
- Implementa un método de búsqueda que encuentre la distancia mínima entre dos ciudades específicas ingresadas por el usuario.
- Desarrolla un proceso que imprima la matriz de distancias y las ciudades ordenadas por su distancia total.

c++

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

vector<vector<int>> ingresarDistancias(int numCiudades) {
    vector<vector<int>> matriz(numCiudades, vector<int>(numCiudades, 0));
    for (int i = 0; i < numCiudades; ++i) {
        for (int j = 0; j < numCiudades; ++j) {
            cout << "Distancia de la ciudad " << i+1 << " a la ciudad " << j+1 << ": ";
            cin >> matriz[i][j];
        }
    }
    return matriz;
}

vector<int> ordenarCiudadesPorDistancia(const vector<vector<int>>& matriz) {
    vector<pair<int, int>> distanciasTotales;
    for (int i = 0; i < matriz.size(); ++i) {
```

```
        int distanciaTotal = 0;
        for (int j = 0; j < matriz[i].size(); ++j) {
            distanciaTotal += matriz[i][j];
        }
        distanciasTotales.push_back({distanciaTotal, i});
    }
    sort(distanciasTotales.begin(), distanciasTotales.end());
    vector<int> ciudadesOrdenadas;
    for (const auto& ciudad : distanciasTotales) {
        ciudadesOrdenadas.push_back(ciudad.second);
    }
    return ciudadesOrdenadas;
}

int distanciaEntreCiudades(const vector<vector<int>>& matriz, int ciudadOrigen, int ciudadDestino)
{
    return matriz[ciudadOrigen][ciudadDestino];
}

void imprimirMatrizYCiudadesOrdenadas(const vector<vector<int>>& matriz, const vector<int>&
ciudadesOrdenadas) {
    cout << "\nMatriz de distancias:\n";
    for (const auto& fila : matriz) {
        for (int distancia : fila) {
            cout << distancia << " ";
        }
        cout << endl;
    }
    cout << "\nCiudades ordenadas por distancia total:\n";
    for (int ciudad : ciudadesOrdenadas) {
        cout << "Ciudad " << ciudad+1 << endl;
    }
}

int main() {
    int numCiudades;
    cout << "Ingrese el número de ciudades: ";
    cin >> numCiudades;

    vector<vector<int>> matrizDistancias = ingresarDistancias(numCiudades);
    vector<int> ciudadesOrdenadas = ordenarCiudadesPorDistancia(matrizDistancias);
    imprimirMatrizYCiudadesOrdenadas(matrizDistancias, ciudadesOrdenadas);

    return 0;
}
Python
def ingresar_distancias(num_ciudades):
    matriz = []
```

```
    for i in range(num_ciudades):
        distancia_ciudad = []
        for j in range(num_ciudades):
            distancia = int(input(f"Distancia de la ciudad {i+1} a la ciudad {j+1}: "))
            distancia_ciudad.append(distancia)
        matriz.append(distancia_ciudad)
    return matriz

def ordenar_ciudades_por_distancia(matriz):
    distancias_totales = [(sum(distancias), indice) for indice, distancias in enumerate(matriz)]
    distancias_totales.sort()
    ciudades_ordenadas = [x[1] for x in distancias_totales]
    return ciudades_ordenadas

def distancia_entre_ciudades(matriz, ciudad_origen, ciudad_destino):
    distancia = matriz[ciudad_origen][ciudad_destino]
    return distancia

def imprimir_matriz_y_ciudades_ordenadas(matriz, ciudades_ordenadas):
    print("\nMatriz de distancias:")
    for fila in matriz:
        print(fila)
    print("\nCiudades ordenadas por distancia total:")
    for ciudad in ciudades_ordenadas:
        print(f"Ciudad {ciudad+1}")

num_ciudades = 3
matriz_distancias = ingresar_distancias(num_ciudades)
ciudades_ordenadas = ordenar_ciudades_por_distancia(matriz_distancias)
imprimir_matriz_y_ciudades_ordenadas(matriz_distancias, ciudades_ordenadas)
```

Ejercicio 6: Gestión de Notas de Estudiantes
Enunciado:

Desarrolla un programa para gestionar las notas de los estudiantes en diferentes materias. Usa un array bidimensional donde cada fila representa a un estudiante y cada columna representa una materia.

- Inicialización de Datos:

  Crea una función para inicializar el array bidimensional con notas aleatorias (entre 0 y 100) para 5 estudiantes y 3 materias.

- Cálculo de Promedios:

  Crea una función que reciba el array y calcule el promedio de notas para cada estudiante.

- Ordenamiento de Notas:

  Implementa una función que ordene las notas de cada estudiante de mayor a menor utilizando el algoritmo de Bubble Sort.

- Búsqueda de Notas:

  Implementa un método de búsqueda que permita encontrar la nota más alta y más baja en una materia específica.

- Visualización de Datos:

  Desarrolla un proceso que imprima las notas de todos los estudiantes, los promedios y las notas más altas y más bajas de una materia específica.
  c++:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>

using namespace std;

vector<vector<int>> inicializarNotas(int numEstudiantes, int numMaterias) {
  vector<vector<int>> notas(numEstudiantes, vector<int>(numMaterias, 0));
  random_device rd;
  mt19937 gen(rd());
  uniform_int_distribution<> dis(0, 100);
  for (int i = 0; i < numEstudiantes; ++i) {
    for (int j = 0; j < numMaterias; ++j) {
      notas[i][j] = dis(gen);
    }
  }
  return notas;
}

vector<float> calcularPromedios(const vector<vector<int>>& notas) {
  vector<float> promedios;
  for (const auto& estudiante : notas) {
    float promedio = accumulate(estudiante.begin(), estudiante.end(), 0.0) /
estudiante.size();
    promedios.push_back(promedio);
  }
  return promedios;
}

void ordenarNotas(vector<vector<int>>& notas) {
  for (auto& estudiante : notas) {
```

```
        sort(estudiante.rbegin(), estudiante.rend());
    }
}

int buscarNotaExtrema(const vector<vector<int>>& notas, int materia, string extrema) {
    if (extrema == "max") {
        int maxNota = INT_MIN;
        for (const auto& estudiante : notas) {
            maxNota = max(maxNota, estudiante[materia]);
        }
        return maxNota;
    } else if (extrema == "min") {
        int minNota = INT_MAX;
        for (const auto& estudiante : notas) {
            minNota = min(minNota, estudiante[materia]);
        }
        return minNota;
    }
}

void imprimirNotasPromediosExtremas(const vector<vector<int>>& notas, const
vector<float>& promedios, int materia) {
    cout << "Notas de los estudiantes:\n";
    for (const auto& estudiante : notas) {
        for (int nota : estudiante) {
            cout << nota << " ";
        }
        cout << endl;
    }
    cout << "Promedios de notas de los estudiantes:\n";
    for (float promedio : promedios) {
        cout << promedio << " ";
    }
    cout << endl;
    cout << "Nota más alta en la materia " << materia+1 << ": " << buscarNotaExtrema(notas,
materia, "max") << endl;
    cout << "Nota más baja en la materia " << materia+1 << ": " << buscarNotaExtrema(notas,
materia, "min") << endl;
}

int main() {
    int numEstudiantes = 5;
    int numMaterias = 3;

    vector<vector<int>> notas = inicializarNotas(numEstudiantes, numMaterias);
    vector<float> promedios = calcularPromedios(notas);
    ordenarNotas(notas);
```

```
    int materiaEspecifica = 0;
    imprimirNotasPromediosExtremas(notas, promedios, materiaEspecifica);

    return 0;
}
```

Python

```
import random

def inicializar_notas(num_estudiantes, num_materias):
    notas = []
    for _ in range(num_estudiantes):
        fila_notas = [random.randint(0, 100) for _ in range(num_materias)]
        notas.append(fila_notas)
    return notas

def calcular_promedios(notas):
    promedios = [sum(estudiante) / len(estudiante) for estudiante in notas]
    return promedios

def ordenar_notas(notas):
    for estudiante in notas:
        estudiante.sort(reverse=True)

def buscar_nota_extrema(notas, materia, extrema):
    if extrema == "max":
        return max(estudiante[materia] for estudiante in notas)
    elif extrema == "min":
        return min(estudiante[materia] for estudiante in notas)

def imprimir_notas_promedios_extremas(notas, promedios, materia):
    print("Notas de los estudiantes:")
    for estudiante in notas:
        print(estudiante)
    print("Promedios de notas de los estudiantes:", promedios)
    print(f"Nota más alta en la materia {materia+1}: {buscar_nota_extrema(notas, materia,
'max')}")
    print(f"Nota más baja en la materia {materia+1}: {buscar_nota_extrema(notas, materia,
'min')}")

num_estudiantes = 5
num_materias = 3

notas = inicializar_notas(num_estudiantes, num_materias)
promedios = calcular_promedios(notas)
ordenar_notas(notas)

materia_especifica = 0  # Cambiar a la materia específica que desees
imprimir_notas_promedios_extremas(notas, promedios, materia_especifica)
```