Instituto Tecnológico Superior de San Andrés Tuxtla

Programación Básica.

Semestre 2.

CALIFICION TOTAL 80

Docente:

L.I. Sergio Pelayo Vaquero.

Alumno:

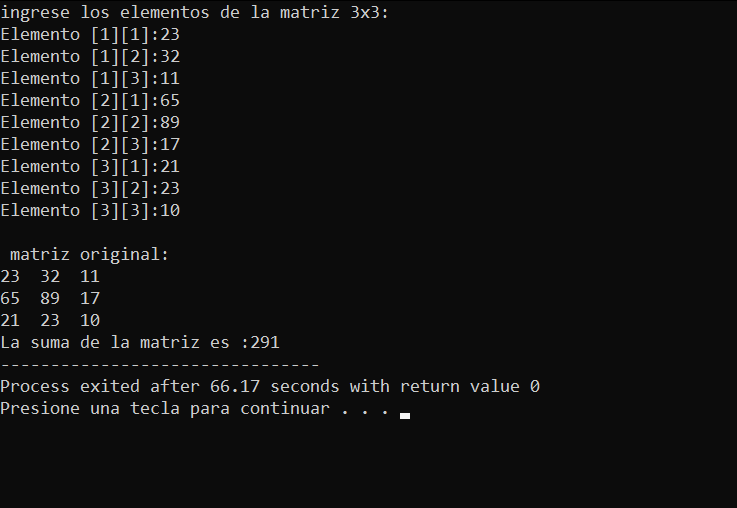
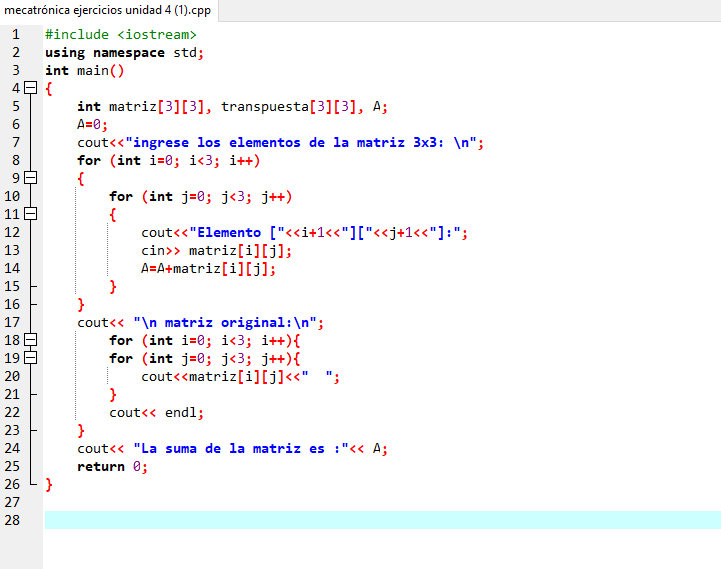
Ángel Emmanuel Pérez Dolores

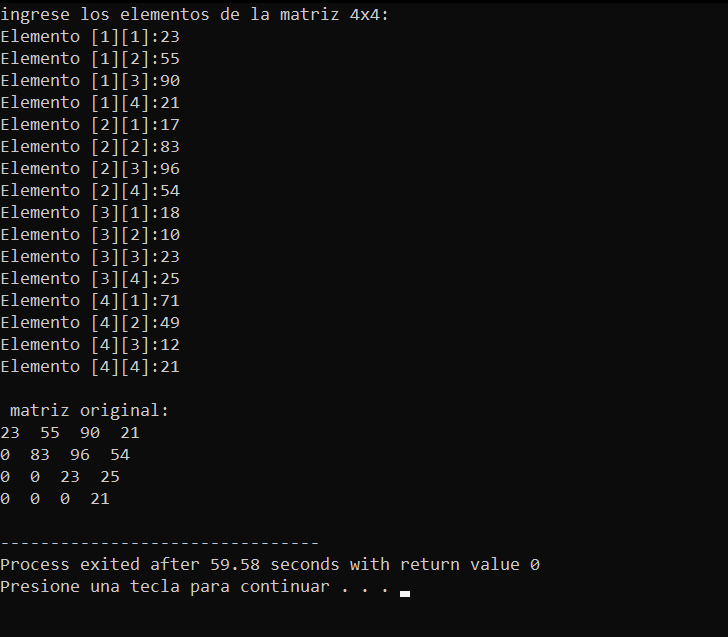
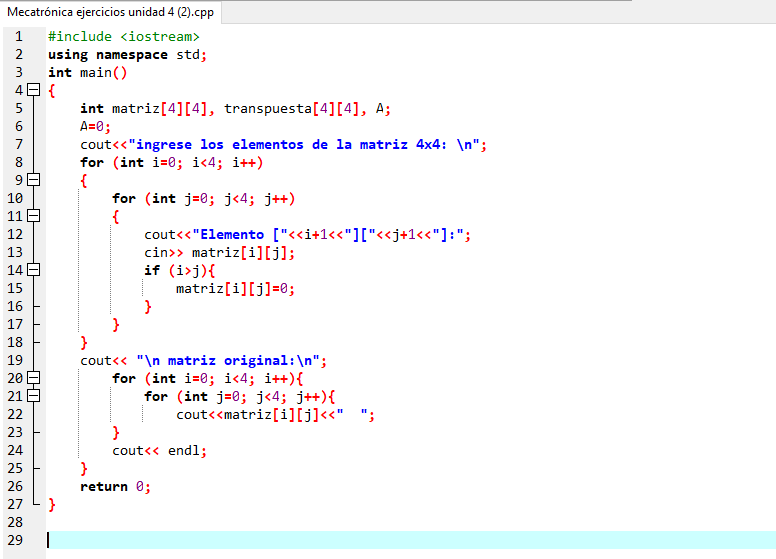
UNIDAD IV:

Cuadernillo de Ejercicios Unidad 4.

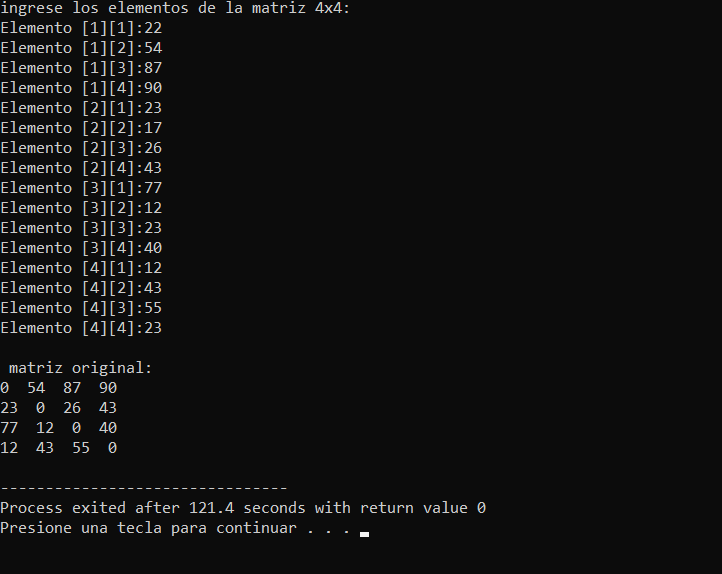
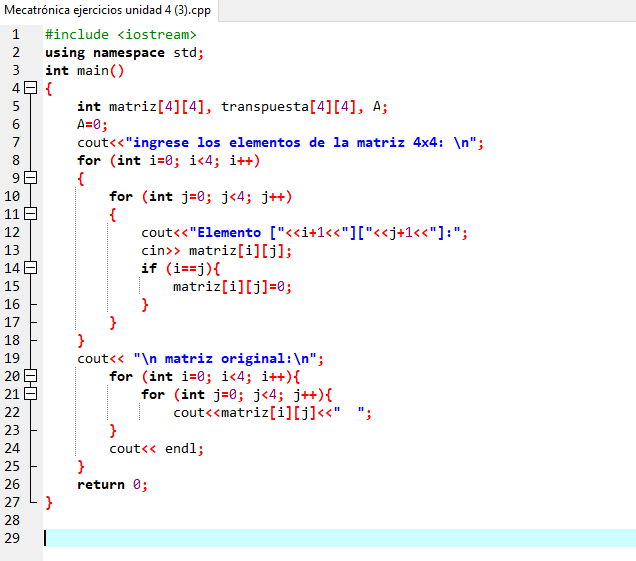
Sán Andrés Tuxtla Ver, a 06 de Junio del 2025.

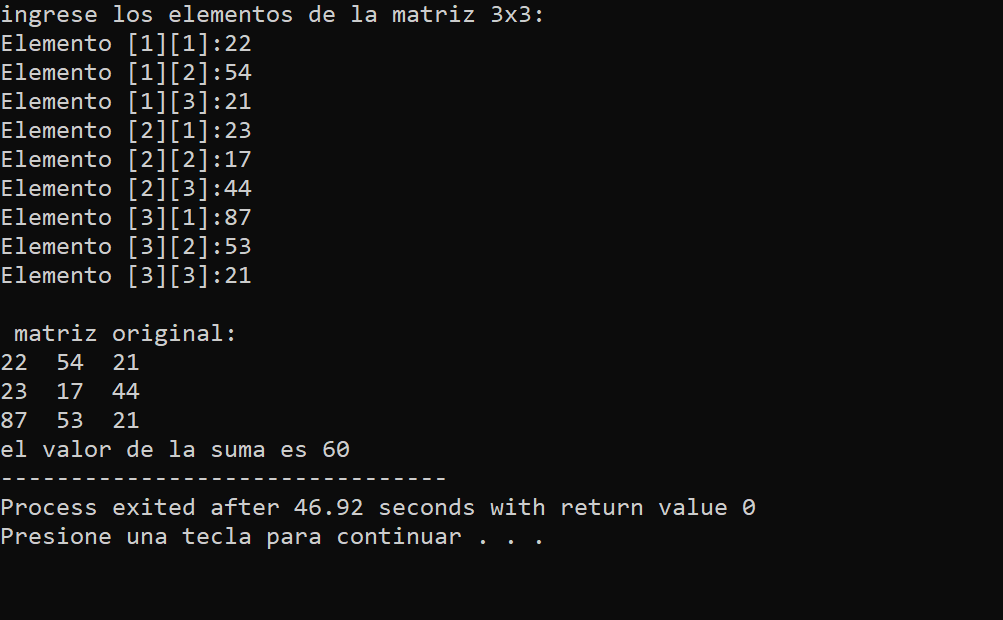
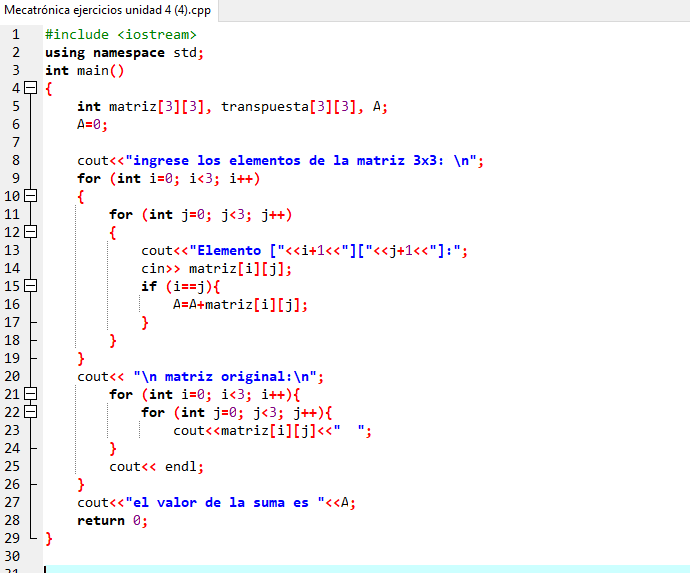
Considere que la matriz ya está creada, deberá agregar un proceso que sume todos los elementos almacenados, deberá mostrar en la pantalla la suma total de todos los elementos:



Realizar un programa en el que se cree una matriz de 4 filas y 4 columnas en el que la parte inferior a la diagonal principal sean siempre 0

Realizar un programa en C++ que realice una matriz de 4 filas y 4 columnas en el que la diagonal principal de la matriz sea siempre 0:



Se tiene una matriz con elementos numéricos almacenados, las dimensiones de la matriz son de 3 filas y 3 columnas, como lo anterior se solicita un programa que sume los elementos que se encuentran en la diagonal principal y muestre la suma en pantalla:

Instrumentos de Evaluación - Ingeniería Informática

# Lista de Cotejo: Cuadernillo de Ejercicios de Programación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N.º | Criterio a Evaluar | Cumple (✓) | No Cumple (✗) | Observaciones |
| 1 | Incluye todos los ejercicios solicitados en la consigna. | 10% |  |  |
| 2 | La sintaxis de los programas está correcta y sin errores. | 10% |  |  |
| 3 | Se evidencia comprensión en el uso de estructuras de control. | 10% |  |  |
| 4 | Cada ejercicio incluye comentarios explicativos del código. | 10% |  |  |
| 5 | Presentación clara, ordenada y con formato adecuado. | 10% |  |  |
| TOTAL | | | | 50% |

Instituto Tecnológico Superior de San Andrés Tuxtla

PROGRAMACIÓN BÁSICA.

Semestre 2.

Docente:

L.I. Sergio Pelayo Vaquero.

Alumno:

Ángel Emmanuel Pérez Dolores

UNIDAD IV:

Síntesis de la investigación.

Calificación 40%

# Vectores y Matrices Unidad 4.

En la programación, el manejo de grandes cantidades de datos estructurados es esencial. Las estructuras de datos como los vectores y matrices permiten organizar, acceder y manipular información de forma eficiente.

## Vector.

Un vector es una estructura de datos que permite almacenar y manipular una colección de elementos del mismo tipo, de manera ordenada. En C++, el tipo vector pertenece a la biblioteca estándar STL (Standard Template Library) y se caracteriza por poder cambiar su tamaño dinámicamente. Un vector es una estructura de datos unidimensional que almacena una secuencia de elementos del mismo tipo. Cada elemento del vector se identifica mediante un índice, que comienza desde cero.

Como tal, los vectores son más flexibles que los arreglos tradicionales, ya que pueden crecer o disminuir en tiempo de ejecución y permiten acceso a los elementos por medio de un índice.

Un ejemplo de un vector es:

*Vector<int> numeros ={10, 20, 30};*

Cada nuevo vector debe ser declarado usando la palabra clave “Vector”, seguido por el tipo de dato que queremos almacenar encerrando entre paréntesis angulares, por último debemos especificar el nombre del vector.

Los vectores cuentan con métodos que nos permiten evaluar sus condiciones como:

## push\_back()

Agrega un elemento al final.

*vector<int> numeros; numeros.push\_back(10); numeros.push\_back(20);*

Ahora el vector contiene: [10, 20]

## pop\_back()

Elimina el último elemento del vector.

*vector<int> numeros = {1, 2, 3};*

*numeros.pop\_back();*

Ahora el vector contiene: [1, 2]

## size().

Devuelve la cantidad de elementos en el vector.

vector<int> v = {4, 5, 6}; cout << v.size();

imprime: 3.

## clear()

Elimina todos los elementos del vector, dejándolo vacío.

vector<int> v = {7, 8, 9}; v.clear();

Ahora V está vacío.

## empty()

Retorna true si el vector está vacío, de lo contrario false.

vector<int> v; if (v.empty()) {

cout << "El vector está vacío.";

}

## at(pos)

Devuelve el elemento en la posición pos, con verificación de límites (más seguro que usar []).

vector<int> v = {10, 20, 30}; cout << v.at(1);

Imprime 20.

Si accedes a un índice fuera de rango, at() lanza una excepción. [] no lo hace y puede causar errores.

## front() y back()

front() devuelve el primer elemento. back() devuelve el último elemento.

vector<int> v = {100, 200, 300};

cout << v.front(); // Imprimirá: 100 cout << v.back(); // Imprimirá: 300

## insert(pos, valor)

Inserta un valor en una posición específica. Usa iteradores.

vector<int> v = {1, 2, 4};

v.insert(v.begin() + 2, 3);

Inserta 3 en la posición 2 y ahora queda como [1, 2, 3, 4]

## erase(pos)

Elimina el elemento en la posición indicada. También usa iteradores.

vector<int> v = {5, 6, 7};

v.erase(v.begin() + 1);

Elimina el 6 del vector, quedando como : [5, 7]

## resize(n)

Cambia el tamaño del vector a n elementos. Si se agranda, los nuevos elementos se inicializan en 0 (o valor predeterminado).

vector<int> v = {1, 2}; v.resize(4);

Ahora v será un vector de 4 elementos v = [1, 2, 0, 0]

Usar los métodos integrados de los vectores en C++ nos permite trabajar de manera más simple, permitiéndonos modificar los vectores a necesidad, de manera segura y eficiente, sin preocuparnos por detalles técnicos complicados.

# Matriz.

Una matriz es una estructura de datos que almacena información en forma de filas y columnas . Se utiliza para organizar datos bidimensionales, como una tabla y tienen un tamaño fijo al declararlas y se acceden a sus elementos con dos índices. uno para la fila y otro para la columna. La sintaxis para declarar una matriz es.

Int matriz[2][3];

Esto crea una matriz con 2 filas y 3 columnas. Las matrices tienen un tamaño fijo que se debe definir al momento de declararlas. Todos los elementos deben ser del mismo tipo: int, float, char, etc

Ejemplo de Matriz:

*Int matriz [2][2]={*

*{12, 23},*

*{31, 15}*

*}*

Este código declara una matriz de 2 filas y 2 columnas. A diferencia de los vectores, una matriz tradicional no puede cambiar su tamaño en tiempo de ejecución.

Aunque parece una estructura en dos dimensiones, en realidad se guarda de forma lineal en memoria (fila por fila).

Para procesar una matriz completa se usan dos bucles “for” uno para las filas y otro para las columnas.

for (int i = 0; i < filas; i++) {

for (int j = 0; j < columnas; j++) { cout << matriz[i][j] << " ";

}

}

También puedes usar **matrices tridimensionales** (o más), aunque son más complejas de manejar.

Ejemplo:

int cubo[2][3][4];

esto creará una forma 3D: 2 bloques de 3 filas x 4 columnas.

Los vectores y matrices son estructuras fundamentales para la programación, ya que nos permiten **organizar y manejar grandes cantidades de datos de forma eficiente**. Los vectores, como ya se había mencionado, al ser estructuras dinámicas, facilitan el almacenamiento y la manipulación de datos cuyo tamaño puede cambiar durante la ejecución del programa, mientras que las matrices permiten representar datos de manera más gráfica, como en forma de tablas, lo cual es esencial en aplicaciones como procesamiento de imágenes, simulaciones, bases de datos y cálculos matemáticos.

# Lista de Cotejo: Resumen sobre Vectores y Matrices de Programación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N.º | Criterio a Evaluar | Cumple (✓) | No Cumple (✗) | Observaciones |
| 1 | El resumen incluye definiciones claras de vectores y matrices. | 10% |  |  |
| 2 | Contiene ejemplos explicativos sobre su uso en programación. | 10% |  |  |
| 3 | Usa un lenguaje técnico y apropiado. | 10% |  |  |
| 4 | La información está bien organizada y sintetizada. | 10% |  |  |
| 5 | No presenta errores conceptuales. | 10% |  |  |
| TOTAL | | | | 50% |