

Matriz Dinámica

Mauricio Avilés

Contenido

- Conceptos
- Creación de Matrices en C++
- Operaciones
- Implementación de DynamicMatrix
- Ejemplo de utilización
- Ejercicios

Matrices

- Arreglos **multidimensionales**
- Se quiere hacer una clase que abstraiga una **matriz bidimensional** con cantidad de filas y columnas especificadas en tiempo de **ejecución**
- En C++ los arreglos **estáticos** deben tener dimensiones que se puedan determinar en tiempo de **compilación**

```
int m, n;  
  
cin >> m >> n;  
  
int arreglo[m][n]; // ¡no funciona!
```

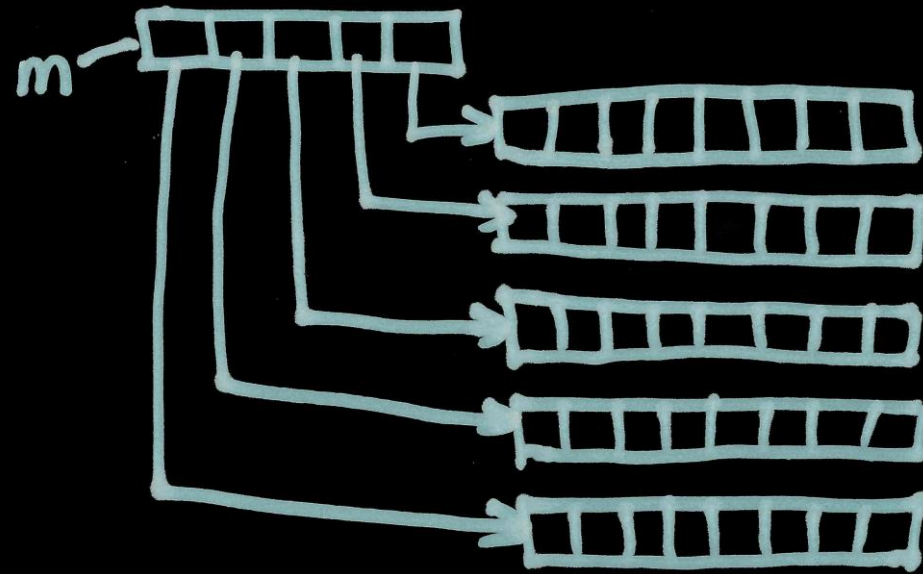
- Crear una matriz en memoria dinámica no es tan simple como especificar las dimensiones a la hora de crear el objeto dinámico

```
int m, n;
```

```
cin >> m >> n;
```

```
int *arreglo = new int[m][n]; // ¡tampoco funciona!
```

- La matriz va a estar formada por dos partes
 - Arreglo de punteros
 - Serie de arreglos de elementos
- Cada puntero del primer arreglo apunta hacia un arreglo de elementos



- El nombre de la matriz es un puntero al primer elemento del arreglo de punteros
- Por lo tanto es **un puntero a un puntero** de elementos
- Esta lógica puede extenderse a mayor cantidad de dimensiones

```
int m, n;  
cin >> m >> n;  
int **arreglo = new int*[m];  
for (int i = 0; i < m; i++) {  
    arreglo[i] = new int[n];  
}
```

- La memoria dinámica solicitada **no se encuentra inicializada**
- Los valores almacenados son indeterminados
- Es necesario **inicializar** la matriz con los valores deseados

```
for (int i = 0; i < m; i++) {  
    for (int j = 0; j < n; j++) {  
        arreglo[i][j] = 0;  
    }  
}
```

```
#include <stdexcept>
```

```
using namespace std;
```

```
template <typename E>
```

```
class DynamicMatrix
```

```
{
```

```
private:
```

```
    int rows;
```

```
    int columns;
```

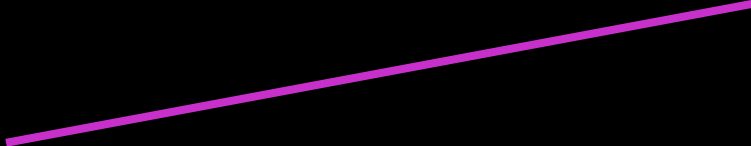
```
    E** matrix;
```

El tipo de elementos en la matriz es genérico, similar a las estructuras de lista vistas anteriormente.


```
#include <stdexcept>

using namespace std;

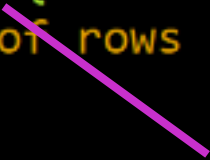
template <typename E>
class DynamicMatrix
{
private:
    int rows;
    int columns;
    E** matrix;
```



Cantidad de filas y de columnas.
La matriz de elementos se maneja por medio de un puntero a puntero de tipo E.

```
public:
    DynamicMatrix(int pRows, int pColumns) throw(runtime_error) {
        if (pRows <= 0 || pColumns <= 0) {
            throw runtime_error("Number of rows and columns must be greater than zero.");
        }
        rows = pRows;
        columns = pColumns;
        matrix = new E*[rows];
        for (int i = 0; i < rows; i++) {
            matrix[i] = new E[columns];
        }
    }

    ~DynamicMatrix() {
        for (int i = 0; i < rows; i++) {
            delete [] matrix[i];
        }
        delete [] matrix;
    }
}
```



La cantidad de filas y de columnas debe ser mayor que cero.

```
public:
    DynamicMatrix(int pRows, int pColumns) throw(runtime_error) {
        if (pRows <= 0 || pColumns <= 0) {
            throw runtime_error("Number of rows and columns must be greater than zero.");
        }
        rows = pRows;
        columns = pColumns;
        matrix = new E*[rows];
        for (int i = 0; i < rows; i++) {
            matrix[i] = new E[columns];
        }
    }

    ~DynamicMatrix() {
        for (int i = 0; i < rows; i++) {
            delete [] matrix[i];
        }
        delete [] matrix;
    }
```

Se inicializa la cantidad de filas y columnas.


```
public:
    DynamicMatrix(int pRows, int pColumns) throw(runtime_error) {
        if (pRows <= 0 || pColumns <= 0) {
            throw runtime_error("Number of rows and columns must be greater than zero.");
        }
        rows = pRows;
        columns = pColumns;
        matrix = new E*[rows];
        for (int i = 0; i < rows; i++) {
            matrix[i] = new E[columns];
        }
    }

    ~DynamicMatrix() {
        for (int i = 0; i < rows; i++) {
            delete [] matrix[i];
        }
        delete [] matrix;
    }
}
```

Creación del arreglo de punteros que representa las filas de la matriz.

```
public:
    DynamicMatrix(int pRows, int pColumns) throw(runtime_error) {
        if (pRows <= 0 || pColumns <= 0) {
            throw runtime_error("Number of rows and columns must be greater than zero.");
        }
        rows = pRows;
        columns = pColumns;
        matrix = new E*[rows];
        for (int i = 0; i < rows; i++) {
            matrix[i] = new E[columns];
        }
    }

    ~DynamicMatrix() {
        for (int i = 0; i < rows; i++) {
            delete [] matrix[i];
        }
        delete [] matrix;
    }
}
```

Por cada fila, crear un arreglo de elementos con la cantidad especifica en las columnas. Se asigna al elemento correspondiente en el arreglo de punteros.

```
public:
    DynamicMatrix(int pRows, int pColumns) throw(runtime_error) {
        if (pRows <= 0 || pColumns <= 0) {
            throw runtime_error("Number of rows and columns must be greater than zero.");
        }
        rows = pRows;
        columns = pColumns;
        matrix = new E*[rows];
        for (int i = 0; i < rows; i++) {
            matrix[i] = new E[columns];
        }
    }

    ~DynamicMatrix() {
        for (int i = 0; i < rows; i++) {
            delete [] matrix[i];
        }
        delete [] matrix;
    }
}
```

En el destructor se recorre el arreglo de punteros, liberando cada uno de los arreglos de elementos que contiene cada fila de la matriz.


```
public:
    DynamicMatrix(int pRows, int pColumns) throw(runtime_error) {
        if (pRows <= 0 || pColumns <= 0) {
            throw runtime_error("Number of rows and columns must be greater than zero.");
        }
        rows = pRows;
        columns = pColumns;
        matrix = new E*[rows];
        for (int i = 0; i < rows; i++) {
            matrix[i] = new E[columns];
        }
    }

    ~DynamicMatrix() {
        for (int i = 0; i < rows; i++) {
            delete [] matrix[i];
        }
        delete [] matrix;
    }
}
```

Se libera la memoria del arreglo de punteros.

```
E getValue(int pRow, int pColumn) throw(runtime_error) {  
    if (pRow < 0 || pRow >= rows) {  
        throw runtime_error("Invalid row.");  
    }  
    if (pColumn < 0 || pColumn >= columns) {  
        throw runtime_error("Invalid column.");  
    }  
    return matrix[pRow][pColumn];  
}
```

Se verifica que la fila y la columna indicadas sean válidas.

```
void setValue(int pRow, int pColumn, E value) throw(runtime_error) {  
    if (pRow < 0 || pRow >= rows) {  
        throw runtime_error("Invalid row.");  
    }  
    if (pColumn < 0 || pColumn >= columns) {  
        throw runtime_error("Invalid column.");  
    }  
    matrix[pRow][pColumn] = value;  
}
```

```
int getRows() {  
    return rows;  
}
```

```
int getColumns() {  
    return columns;  
}
```

```
};
```

```
E getValue(int pRow, int pColumn) throw(runtime_error) {  
    if (pRow < 0 || pRow >= rows) {  
        throw runtime_error("Invalid row.");  
    }  
    if (pColumn < 0 || pColumn >= columns) {  
        throw runtime_error("Invalid column.");  
    }  
    return matrix[pRow][pColumn];  
}
```

Se retorna el elemento en esa posición de la matriz.

```
void setValue(int pRow, int pColumn, E value) throw(runtime_error) {  
    if (pRow < 0 || pRow >= rows) {  
        throw runtime_error("Invalid row.");  
    }  
    if (pColumn < 0 || pColumn >= columns) {  
        throw runtime_error("Invalid column.");  
    }  
    matrix[pRow][pColumn] = value;  
}
```

```
int getRows() {  
    return rows;  
}
```

```
int getColumns() {  
    return columns;  
}
```

```
};
```



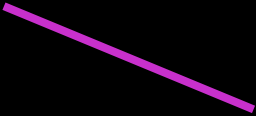
```
E getValue(int pRow, int pColumn) throw(runtime_error) {  
    if (pRow < 0 || pRow >= rows) {  
        throw runtime_error("Invalid row.");  
    }  
    if (pColumn < 0 || pColumn >= columns) {  
        throw runtime_error("Invalid column.");  
    }  
    return matrix[pRow][pColumn];  
}
```

```
void setValue(int pRow, int pColumn, E value) throw(runtime_error) {  
    if (pRow < 0 || pRow >= rows) {  
        throw runtime_error("Invalid row.");  
    }  
    if (pColumn < 0 || pColumn >= columns) {  
        throw runtime_error("Invalid column.");  
    }  
    matrix[pRow][pColumn] = value;  
}
```

```
int getRows() {  
    return rows;  
}
```

```
int getColumns() {  
    return columns;  
}
```

```
};
```



Similar al método anterior. Se chequean restricciones y se asigna el valor enviado.

```

E getValue(int pRow, int pColumn) throw(runtime_error) {
    if (pRow < 0 || pRow >= rows) {
        throw runtime_error("Invalid row.");
    }
    if (pColumn < 0 || pColumn >= columns) {
        throw runtime_error("Invalid column.");
    }
    return matrix[pRow][pColumn];
}

void setValue(int pRow, int pColumn, E value) throw(runtime_error) {
    if (pRow < 0 || pRow >= rows) {
        throw runtime_error("Invalid row.");
    }
    if (pColumn < 0 || pColumn >= columns) {
        throw runtime_error("Invalid column.");
    }
    matrix[pRow][pColumn] = value;
}

int getRows() {
    return rows;
}

int getColumns() {
    return columns;
}
};

```

Los métodos getRows y getColumns consisten simplemente en retornar el valor de cada atributo.

Ejemplo de utilización de la clase DynamicMatrix

```
int main()
{
    DynamicMatrix<int> matriz(10, 15);
    for (int i = 0; i < matriz.getRows(); i++) {
        for (int j = 0; j < matriz.getColumns(); j++) {
            matriz.setValue(i, j, i+j);
        }
    }
    for (int i = 0; i < matriz.getRows(); i++) {
        for (int j = 0; j < matriz.getColumns(); j++) {
            cout << matriz.getValue(i, j) << "\t";
        }
        cout << endl;
    }

    return 0;
}
```

"C:\Users\Admin\Google Drive\Cursos\Estructuras de Datos\C3\4digo\Estructuras\Matriz\Matriz\bin\Debug\Matriz.exe"

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Process returned 0 (0x0) execution time : 0.225 s

Press any key to continue.

```
int main() {  
    int f, c;  
    cout << "Indique la cantidad de filas: ";  
    cin >> f;  
    cout << "Indique la cantidad de columnas: ";  
    cin >> c;  
    DynamicMatrix<int> *matriz2 = new DynamicMatrix<int>(f, c);  
  
    for (int i = 0; i < matriz2->getRows(); i++) {  
        for (int j = 0; j < matriz2->getColumns(); j++) {  
            matriz2->setValue(i, j, 0);  
        }  
    }  
    matriz2->setValue(0, 0, 9999);  
    matriz2->setValue(matriz2->getRows() - 1, matriz2->getColumns() - 1, 9999);  
  
    for (int i = 0; i < matriz2->getRows(); i++) {  
        for (int j = 0; j < matriz2->getColumns(); j++) {  
            cout << matriz2->getValue(i, j) << "\\t";  
        }  
        cout << endl;  
    }  
  
    delete matriz2;  
}
```


"C:\Users\Admin\Google Drive\Cursos\Estructuras de Datos\C3/4digo\Estructuras\Matriz\Matriz\bin\Debug\Matriz.exe"

— □ ×

Indique la cantidad de filas: 5

Indique la cantidad de columnas: 10

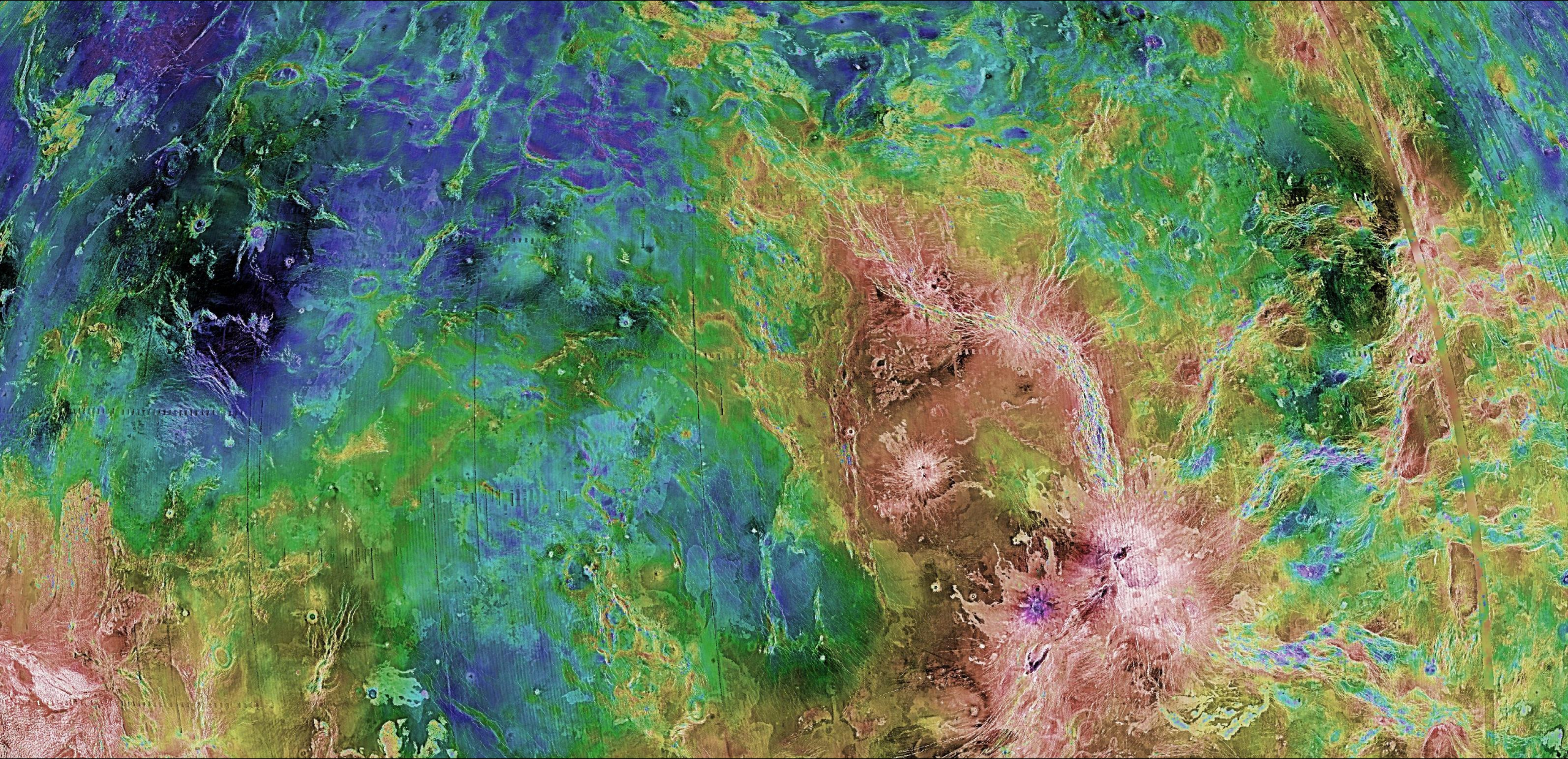
9999	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	9999

Process returned 0 (0x0) execution time : 7.590 s

Press any key to continue.

Ejercicios con la clase DynamicMatrix

- Implementar un método que sirva para **asignar a todos** los elementos de la matriz el mismo valor. Podría utilizarse para inicializar una matriz de enteros con ceros, por ejemplo.
- Implementar un método para **cambiar dinámicamente el tamaño** de la matriz con una nueva cantidad de filas y columnas.
 - Crear matriz con las nuevas dimensiones.
 - Mover los elementos a la nueva matriz.
 - Determine con qué valores se inicializan los elementos nuevos, podría ser un valor recibido por parámetro.
- Implementar una clase similar llamada **DynamicVector**
 - Implementar operaciones de suma de vectores, producto escalar, multiplicación de vectores.
- Implementar **operaciones** en la clase DynamicMatrix
 - Suma de matrices
 - Transpuesta
 - Multiplicación de vector por matriz
 - Multiplicación de matrices
 - Rotación derecha
 - Rotación izquierda



Matriz Dinámica

Mauricio Avilés