

ESCUELA SUPERIOR DE COMPUTO

INSTITUTO POLITÉCNICO NACIONAL

PRACT ICA 5

**Comunicación interprocesos (IPC) en Linux y Windows**

S I S T E M A S O P E R A T I V O S

**Profesor:** Cortes Galicia Jorge

**Grupo:** 2CM9

**Integrantes:**

Beltran García Juan

Hernández Méndez Oliver Manuel

López López Rodrigo

Rangel Lozada Kevin Sebastián

Índice

[1 Introducción Teórica 1](#_Toc58185933)

[2 Desarrollo Experimental 2](#_Toc58185934)

[2.1 Ejercicio 1 2](#_Toc58185935)

[2.2 Ejercicio 2 5](#_Toc58185936)

[2.3 Ejercicio 3 6](#_Toc58185937)

[2.4 Ejercicio 4 8](#_Toc58185938)

[2.4.1 Codificación en Linux 8](#_Toc58185939)

[2.4.2 Codificación en Windows 18](#_Toc58185940)

[2.5 Ejercicio 5 24](#_Toc58185941)

[2.6 Ejercicio 6 26](#_Toc58185942)

[2.7 Ejercicio 7 29](#_Toc58185943)

[2.7.1 Codificación en Linux 29](#_Toc58185944)

[2.7.2 Codificación en Windows 35](#_Toc58185945)

[3 Conclusiones 41](#_Toc58185946)

# Introducción Teórica

Como hemos visto teóricamente y en otras prácticas, el manejo de la información entre procesos es algo fundamental e indispensable, hablando principalmente de la funcionalidad de los sistemas operativos y las aplicaciones que lo emplean.

Un proceso puede ser de dos tipos:

* Proceso independiente.
* Proceso de cooperación.

Un proceso independiente no se ve afectado por la ejecución de otros procesos, mientras que un proceso cooperativo puede verse afectado por otros procesos en ejecución. Aunque uno puede pensar que esos procesos, que se ejecutan de forma independiente, se ejecutarán de manera muy eficiente, en realidad, hay muchas situaciones en las que la naturaleza cooperativa se puede utilizar para aumentar la velocidad, la conveniencia y el modularidad computacional. La comunicación entre procesos (IPC) es un mecanismo que permite a los procesos comunicarse entre sí y sincronizar sus acciones. La comunicación entre estos procesos puede verse como un método de cooperación entre ellos. Los procesos pueden comunicarse entre sí a través de ambos:

1. Memoria compartida
2. Paso de mensajes

La IPC provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí, normalmente a través de un sistema de bajo nivel de paso de mensajes que ofrece la red subyacente.

Un sistema operativo puede implementar ambos métodos de comunicación. Primero, discutiremos los métodos de comunicación de memoria compartida y luego el paso de mensajes. La comunicación entre procesos que utilizan memoria compartida requiere que los procesos compartan alguna variable y depende completamente de cómo la implementará el programador. Una forma de comunicación que utiliza la memoria compartida se puede imaginar así: Supongamos que proceso1 y proceso2 se están ejecutando simultáneamente y comparten algunos recursos o usan información de otro proceso. Process1 genera información sobre ciertos cálculos o recursos que se utilizan y la mantiene como un registro en la memoria compartida. Cuando process2 necesita usar la información compartida, verificará el registro almacenado en la memoria compartida y tomará nota de la información generada por process1 y actuará en consecuencia.

Ahora, comenzaremos nuestra discusión sobre la comunicación entre procesos a través del paso de mensajes. En este método, los procesos se comunican entre sí sin utilizar ningún tipo de memoria compartida. Si dos procesos p1 y p2 quieren comunicarse entre sí, proceden de la siguiente manera:

* Establezca un enlace de comunicación (si ya existe un enlace, no es necesario volver a establecerlo).
* Empiece a intercambiar mensajes utilizando primitivas básicas.

Necesitamos al menos dos primitivas:

- enviar (mensaje, destino) o enviar (mensaje)

- recibir (mensaje, host) o recibir (mensaje)

# Desarrollo Experimental

## Ejercicio 1

A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de la función: pipe(), shmget(), shmat(). Explique los argumentos y retorno de la función.

**pipe()**

Esta función se emplea para crear una tubería o pipe

Conceptualmente, un tubo o pipe es una conexión entre dos procesos, de manera que la salida estándar de un proceso se convierte en la entrada estándar del otro proceso. En el sistema operativo UNIX, las tuberías son útiles para la comunicación entre procesos relacionados,

Una tubería tiene en realidad dos descriptores de fichero: uno para el extremo de escritura y otro para el extremo de lectura. Como los descriptores de fichero de UNIX son simplemente enteros, un pipe o tubería no es más que un array de dos enteros:

Para crear la tubería se emplea la función pipe(), que abre dos descriptores de fichero y almacena su valor en los dos enteros que contiene el array de descriptores de fichero.

El primer descriptor de fichero es abierto como O\_RDONLY, es decir, sólo puede ser empleado para lecturas. El segundo se abre como O\_WRONLY, limitando su uso a la escritura. De esta manera se asegura que el pipe sea de un solo sentido: por un extremo se escribe y por el otro se lee, pero nunca al revés. Ya hemos dicho que si se precisa una comunicación “full-duplex”, será necesario crear dos tuberías para ello.

**Sintaxis:**

int pipe(int fds[2]);

**Parámetros:**

fds[0] : Sera el descriptor de fichero para el extremo de lectura

fds[1] : Sera el descriptor de fichero para el extremo de escritura

**Retorno:**

Se retorna el entero 0 en caso de éxito y se retorna -1 en caso de algún error

**shmget()**

Esta función se emplea para la creación o acceso a una zona de memoria compartida.

La memoria compartida se crea por un proceso mediante una llamada al sistema, la zona que se reserva en memoria no está en el espacio de direcciones del proceso, es una zona de memoria gestionada por el sistema operativo. Después, otros procesos a los que se les dé permiso para acceder a esa zona de memoria podrán también leer o escribir de ella.

**Sintaxis:**

int shmget(key\_t key, int size, int shmflg);

**Parámetros:**

key: Es una clave que genera el sistema y que será un elemento esencial para poder acceder a la zona de memoria que crearemos. La clave se obtiene previamente utilizando la función ftok(); que produce siempre una clave fija con los mismos argumentos: Para usarla:

key = ftok(“.”, ‘S’);

La clave debe ser la misma para todos los procesos que quieran usar esta zona de memoria

size: Indicará el tamaño de la memoria compartida. Se suele utilizar la opción de sizeof (tipo\_variable) para que se tome el tamaño del tipo de dato que habrá en la memoria compartida.

shmflg: Indicará los derechos para acceder a la memoria, si se crea si no existe y caso de que exista, se da o no un error.

**Retorno:**

Se retorna el identificador de la memoria compartida si no ha habido error

Se retorna -1 en caso de un error

**shmat()**

Una vez ya creada la memoria compartida, pero para utilizarla necesitamos saber su dirección (observe que shmget no nos la indica). Para utilizar la memoria compartida debemos antes vincularla con alguna variable de nuestro código. De esta manera, siempre que usemos la variable vinculada estaremos utilizando la variable compartida. Para establecer un vínculo utilizamos la función shmat.

**Sintaxis:**

void \*shmat(int shmid, char \*shmaddr, int shmflag);

**Parámetros:**

shmid: Es el identificador de la memoria compartida. Lo habremos obtenido con la llamada shmget

shmaddr: Normalmente, valdrá 0 o NULL que indicará al sistema operativo que busque esa zona de memoria en una zona de memoria libre, no en una dirección absoluta

shmflg: permisos. Por ejemplo. Aunque hayamos obtenido una zona de memoria con permiso para escritura o lectura, podemos vincularla a una variable para solo lectura. Si no queremos cambiar los permisos usamos 0

**Retorno:**

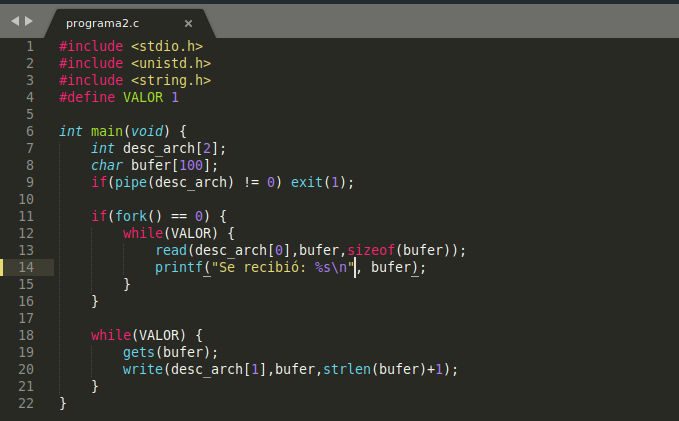
Se retorna un puntero a la zona de memoria compartida

Se retorna -1 en caso de algún error

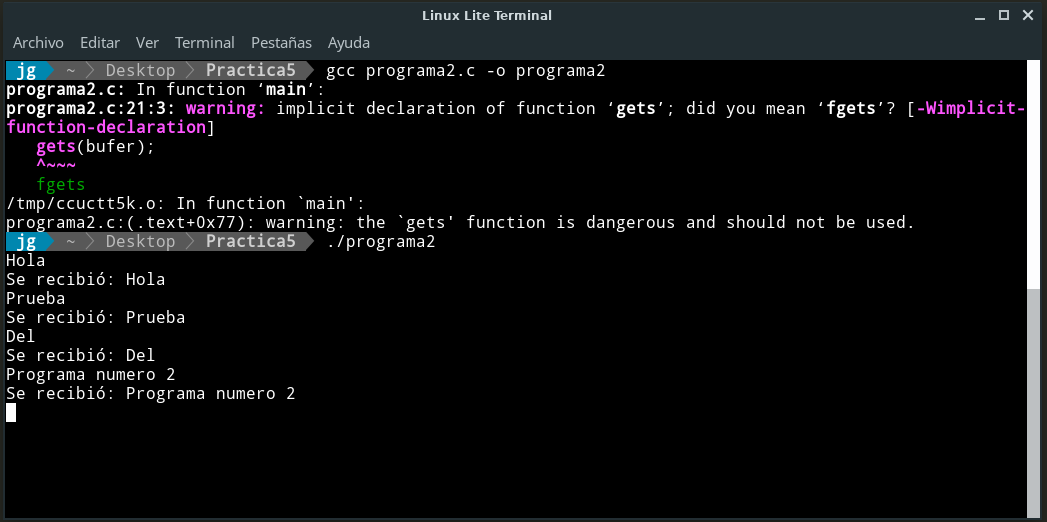
## Ejercicio 2

Capture, compile y ejecute el siguiente programa. Observe su funcionamiento.

**Codigo capturado**



**Compilación y ejecución del programa**

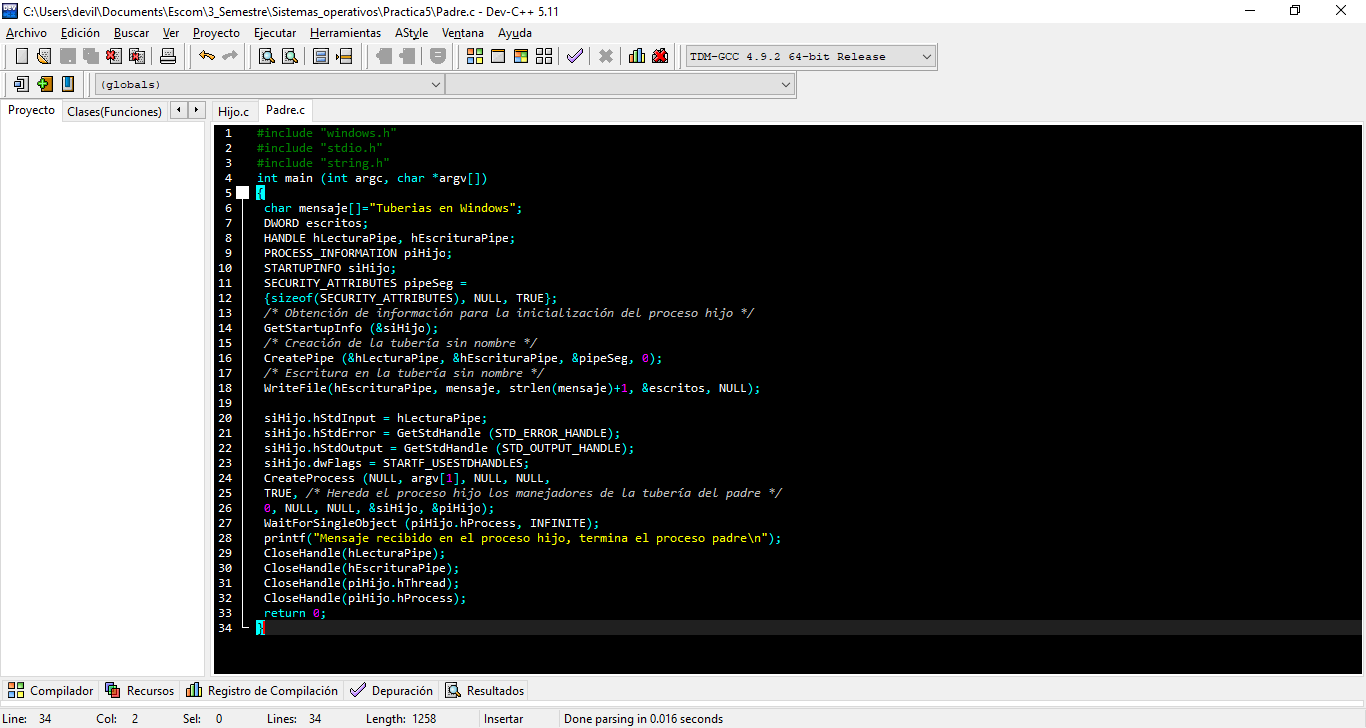


## Ejercicio 3

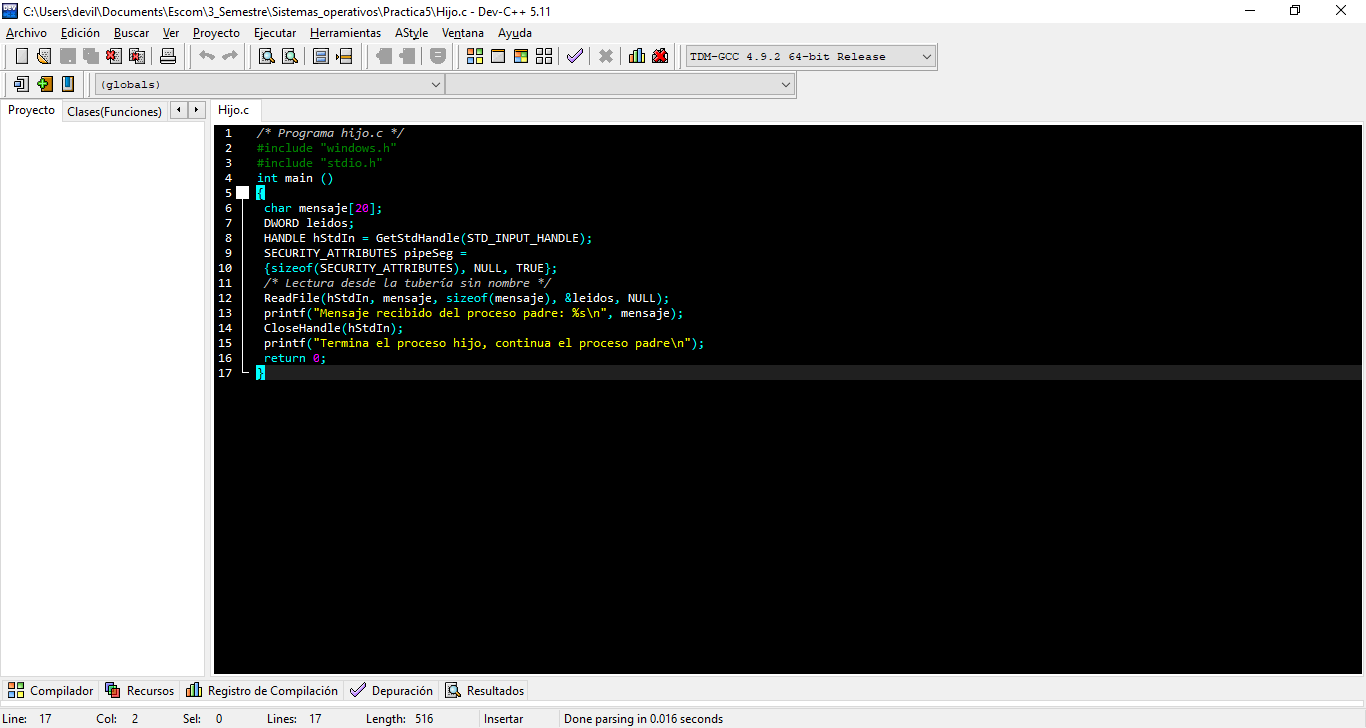
Capture, compile y ejecute los siguientes programas. Observe su funcionamiento. Ejecute de la siguiente forma: C:\>nombre\_programa\_padre nombre\_programa\_hijo

**Codigo capturado**

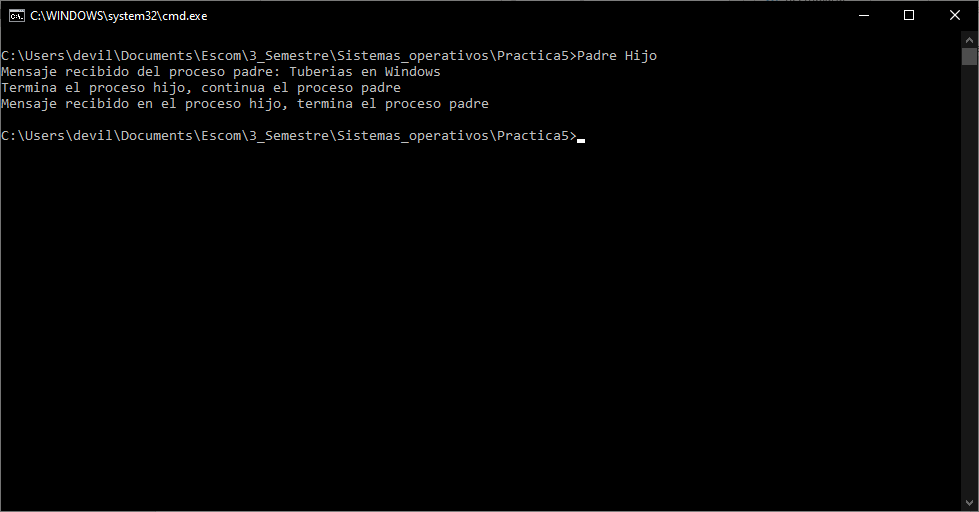
**Código de padre**



**Código Hijo**



**Compilación y ejecución del programa**



## Ejercicio 4

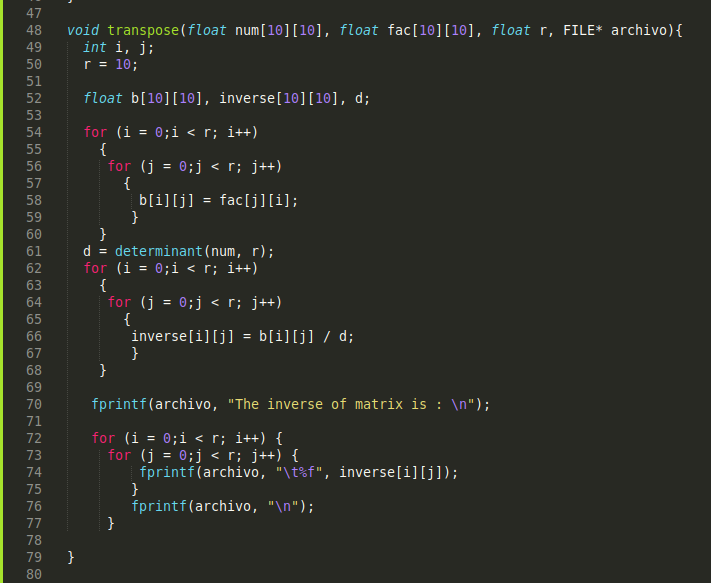
Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el proceso padre enviará al proceso hijo, a través de una tubería, dos matrices de 10 x 10 a multiplicar por parte del hijo, mientras tanto el proceso hijo creará un hijo de él, al cual enviará dos matrices de 10 x 10 a sumar en el proceso hijo creado, nuevamente el envío de estos valores será a través de una tubería. Una vez calculado el resultado de la suma, el proceso hijo del hijo devolverá la matriz resultante a su abuelo (vía tubería). A su vez, el proceso hijo devolverá la matriz resultante de la multiplicación que realizó a su padre. Finalmente, el proceso padre obtendrá la matriz inversa de cada una de las matrices recibidas y el resultado lo guardará en un archivo para cada matriz inversa obtenida.

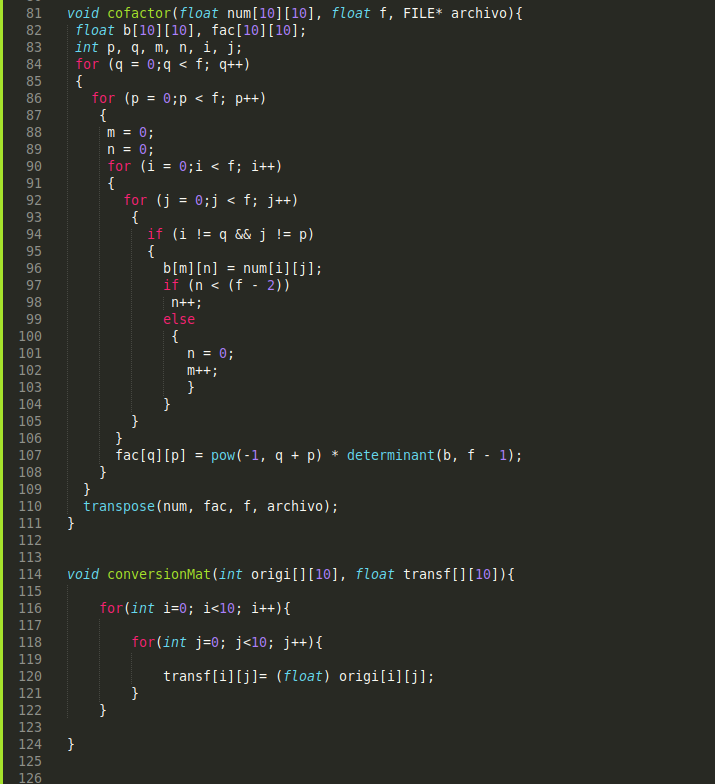
Programe esta aplicación tanto para Linux como para Windows utilizando las tuberías de cada sistema operativo

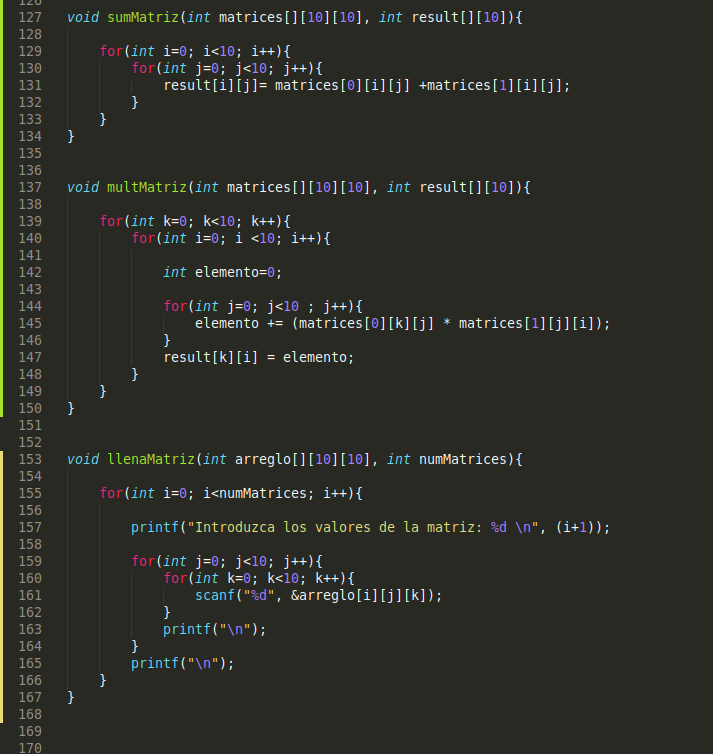
### Codificación en Linux

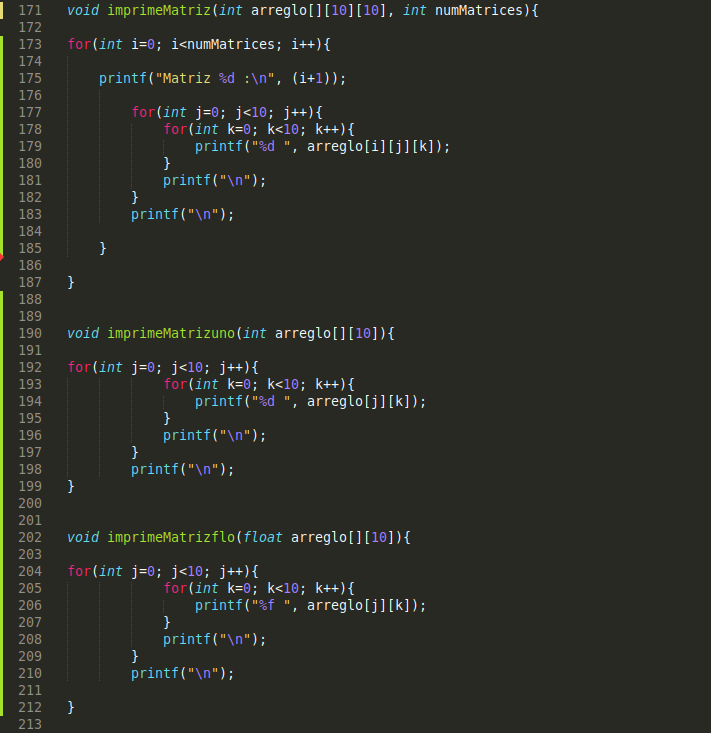
**Codigo fuente**

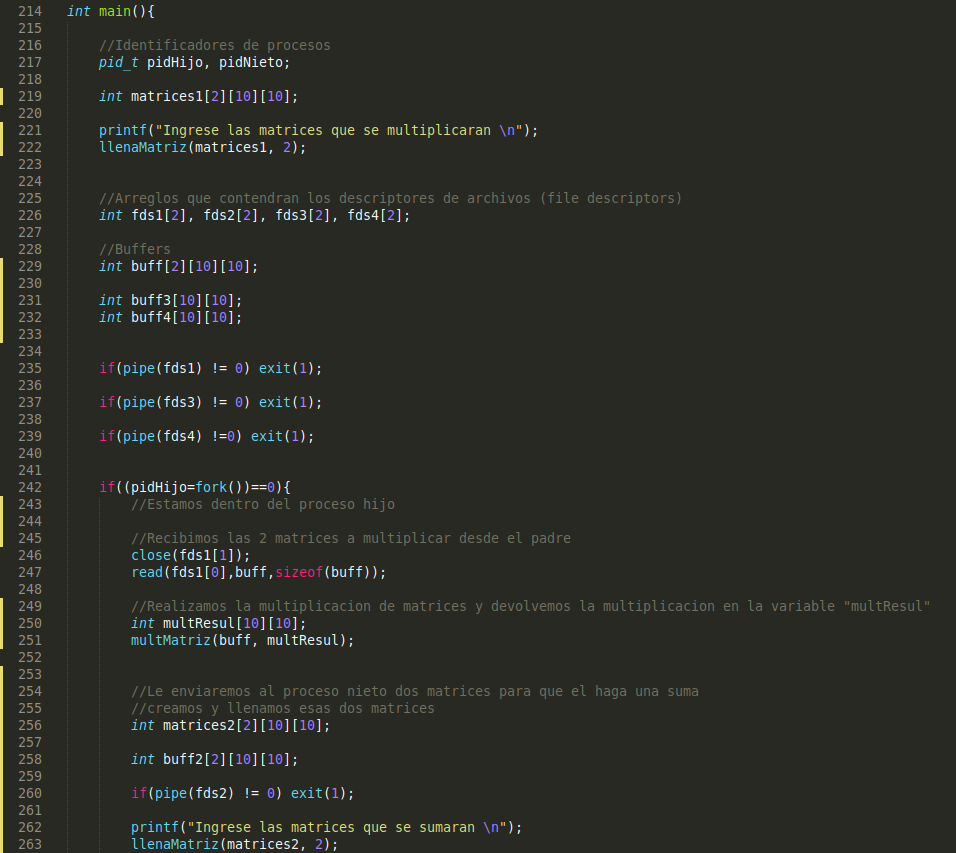








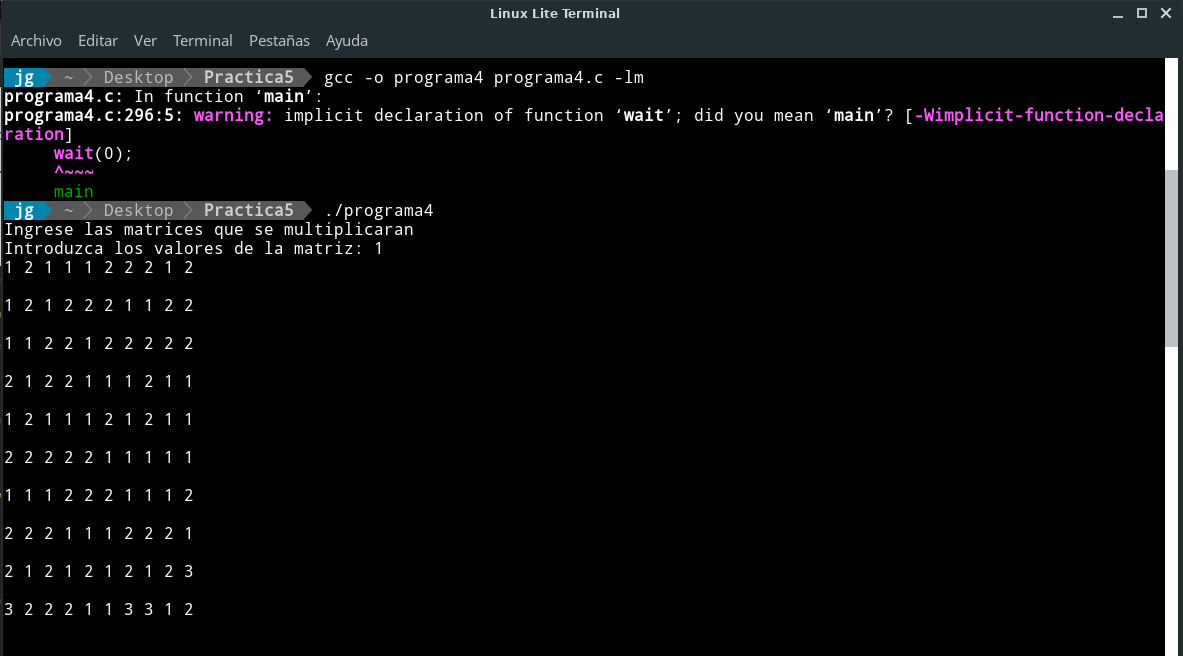


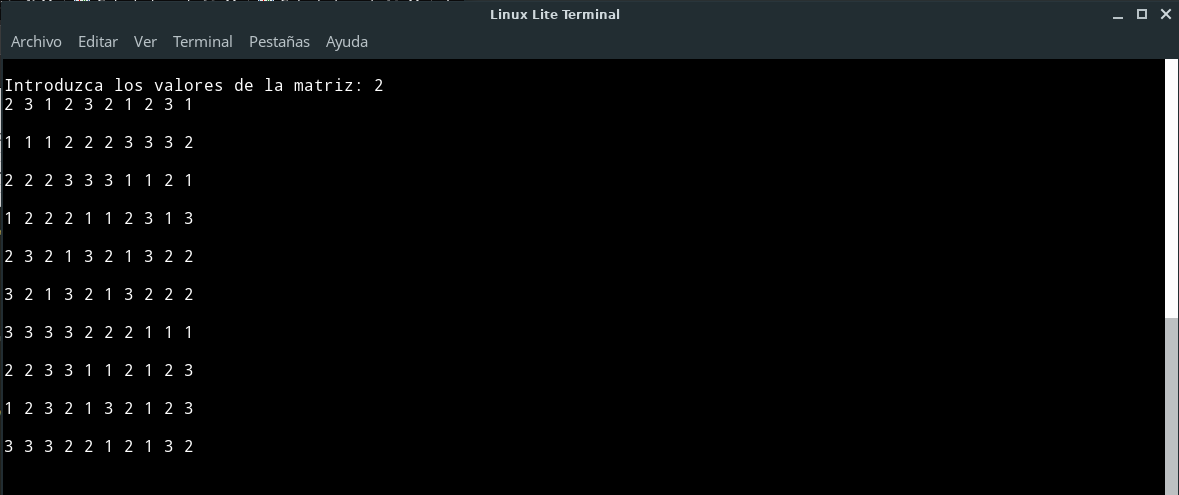


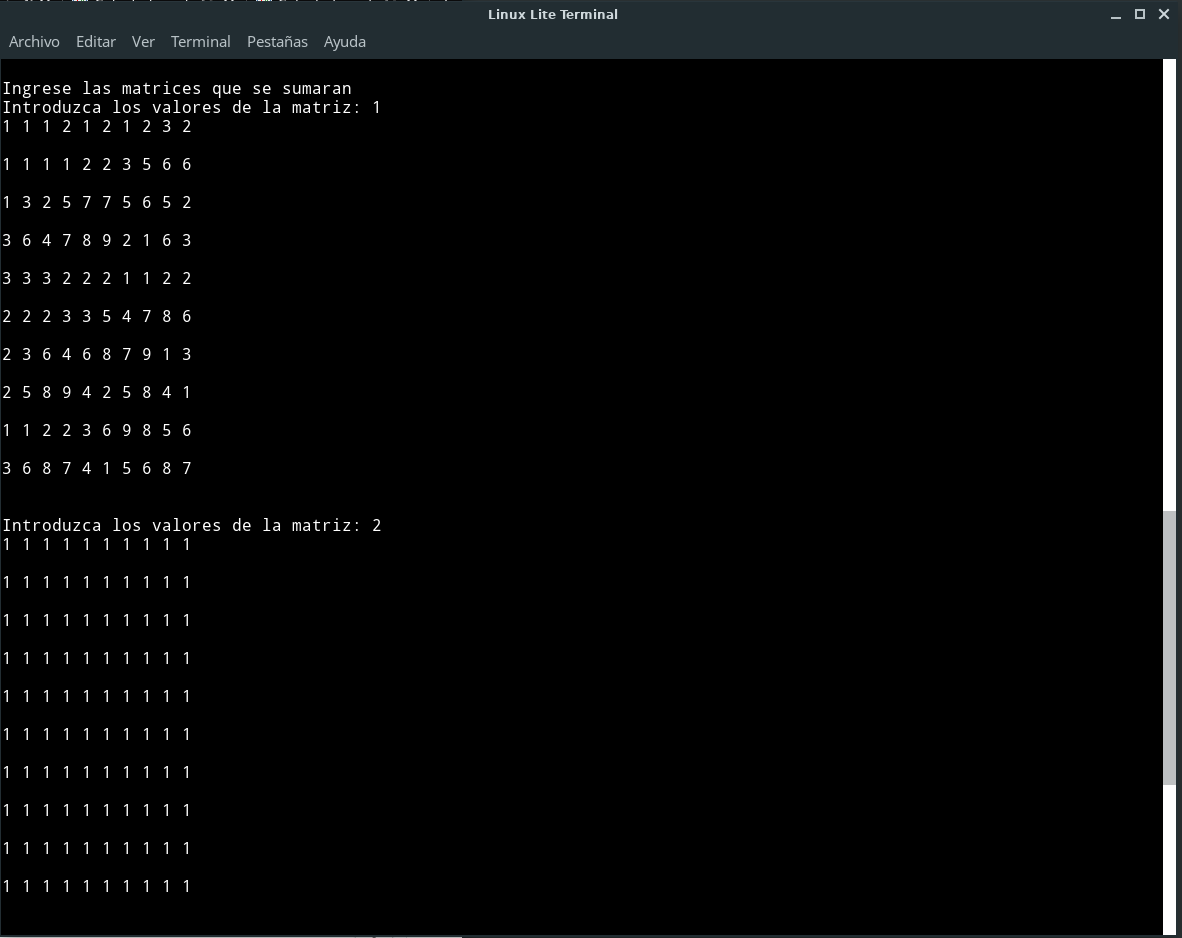


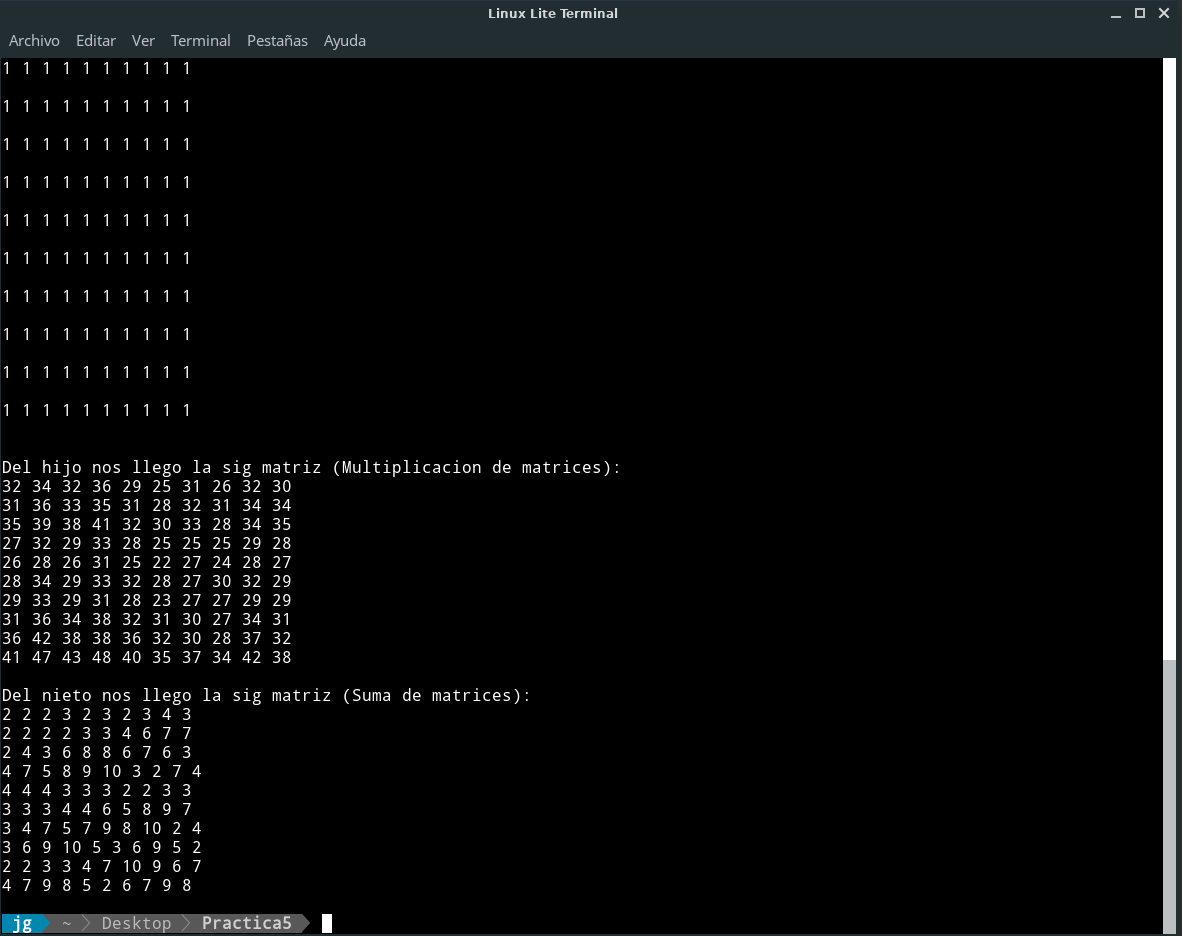


**Compilación y ejecución del programa**

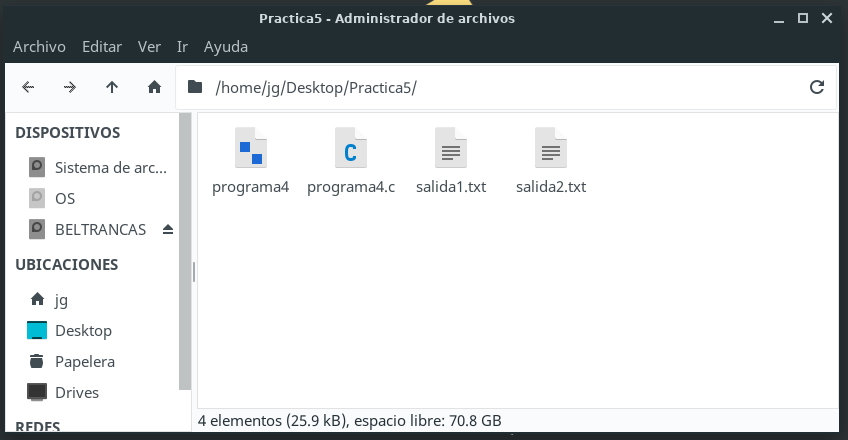




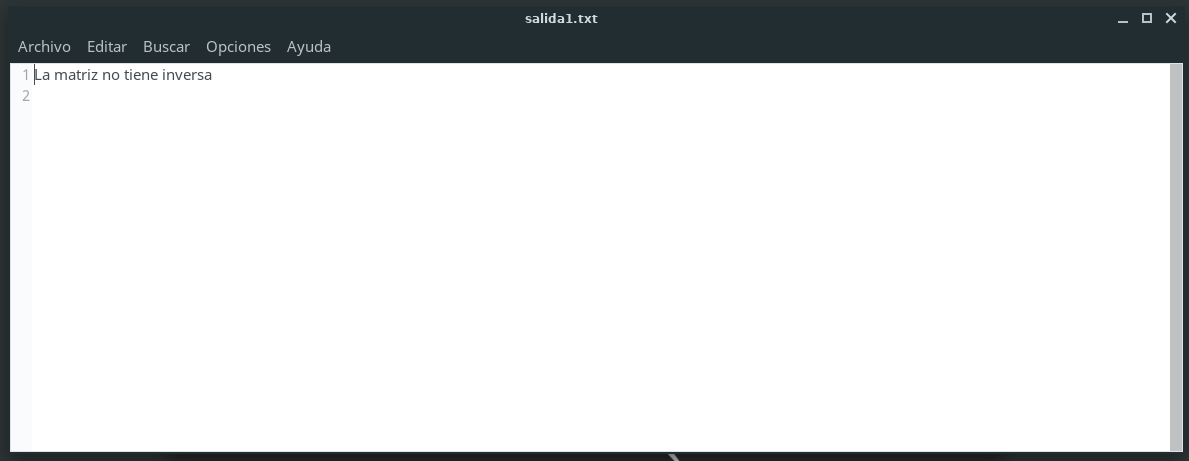


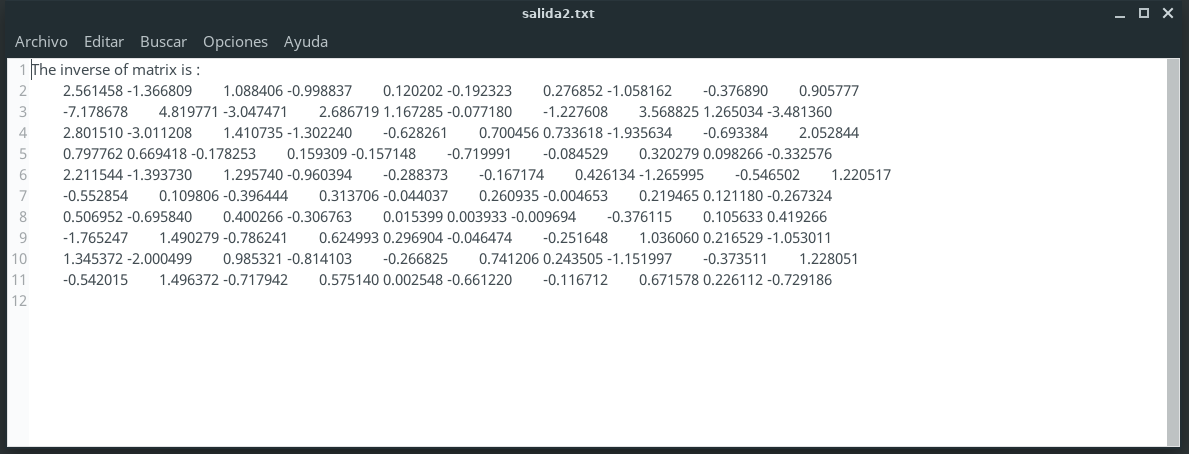


Observamos que dentro de la carpeta donde se encuentra el archivo llamado programa4.c se generaron los archivos de salida que contiene las matrices inversas, en caso de existir



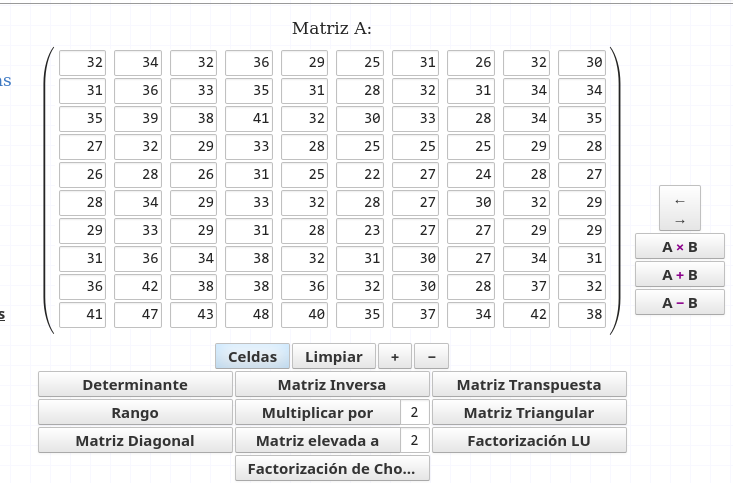
Observamos que el contenido de dichos archivos es el siguiente



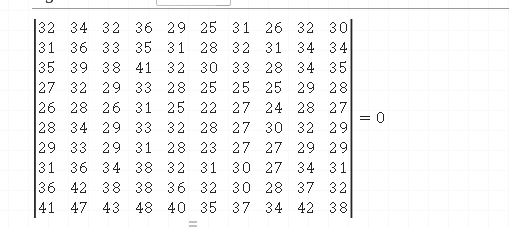


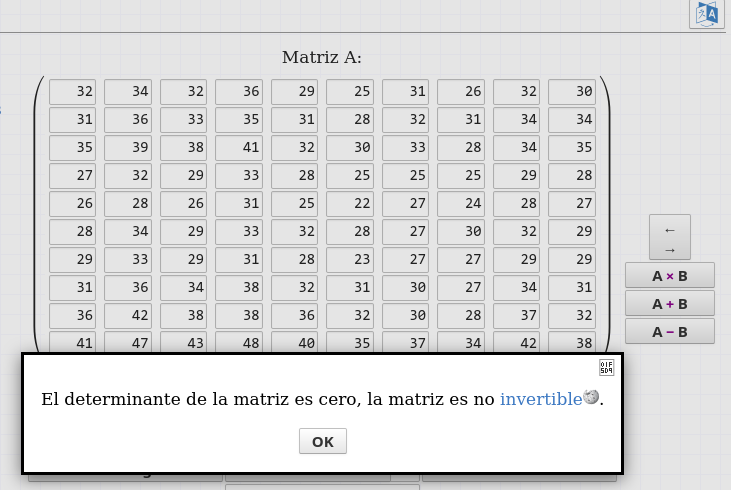
Para verificar las marices inversas se utilizo una calculadora de matrices online

Primero observaremos que la primera matriz no posee inversa, por lo que en el documento se imprime la leyenda “La matriz no tiene inversa”



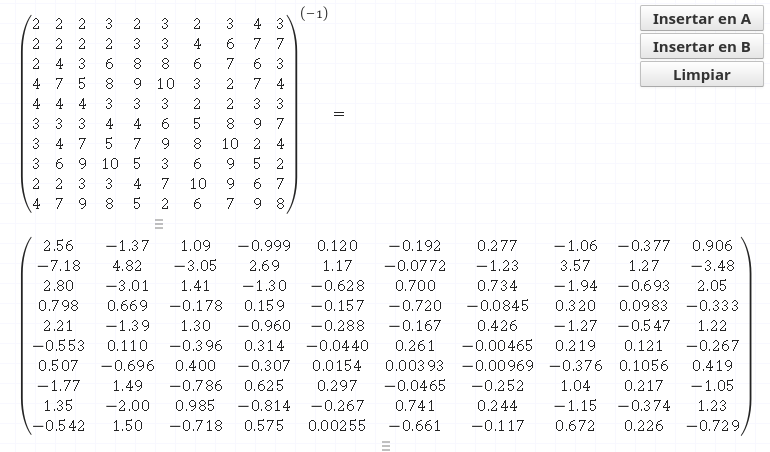
Debido a que el determinante es 0 la matriz no tiene inversa





Para la segunda matriz observamos que la matriz resultante por la calculadora es igual a la mostrada en el archivo llamado “salida2.txt”

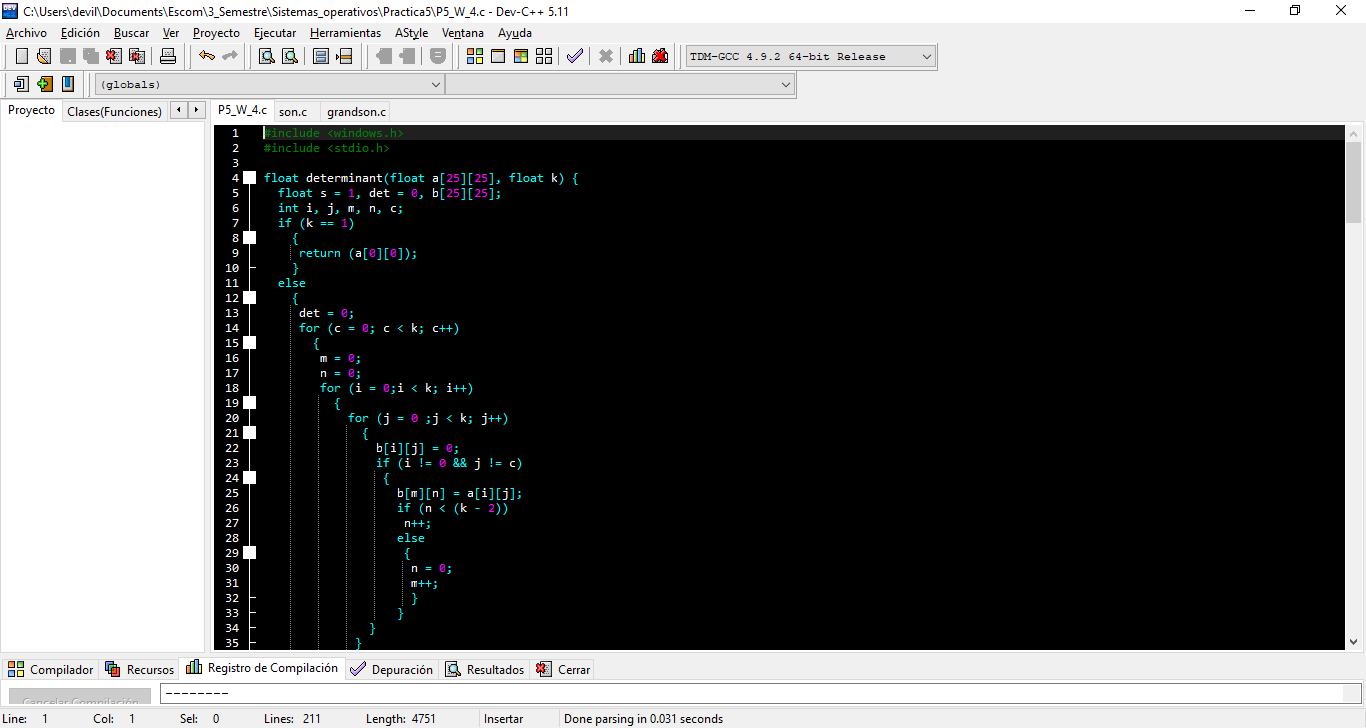


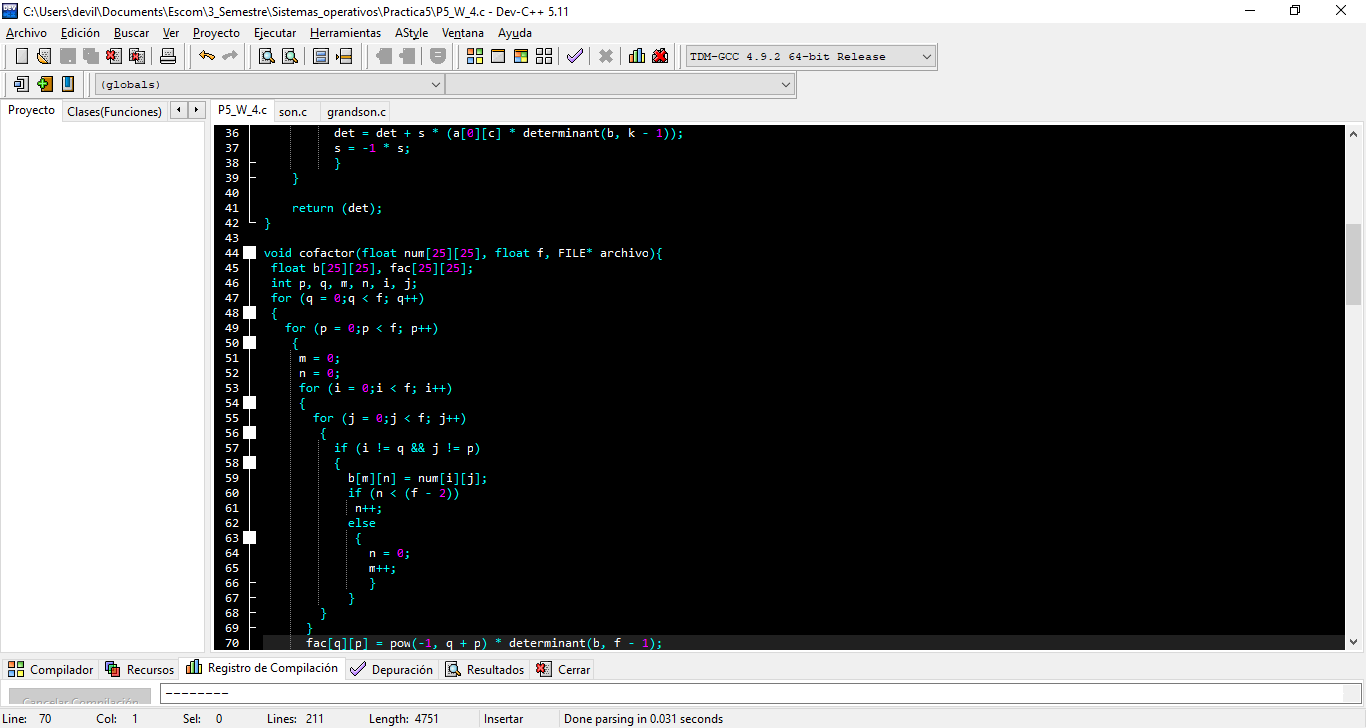


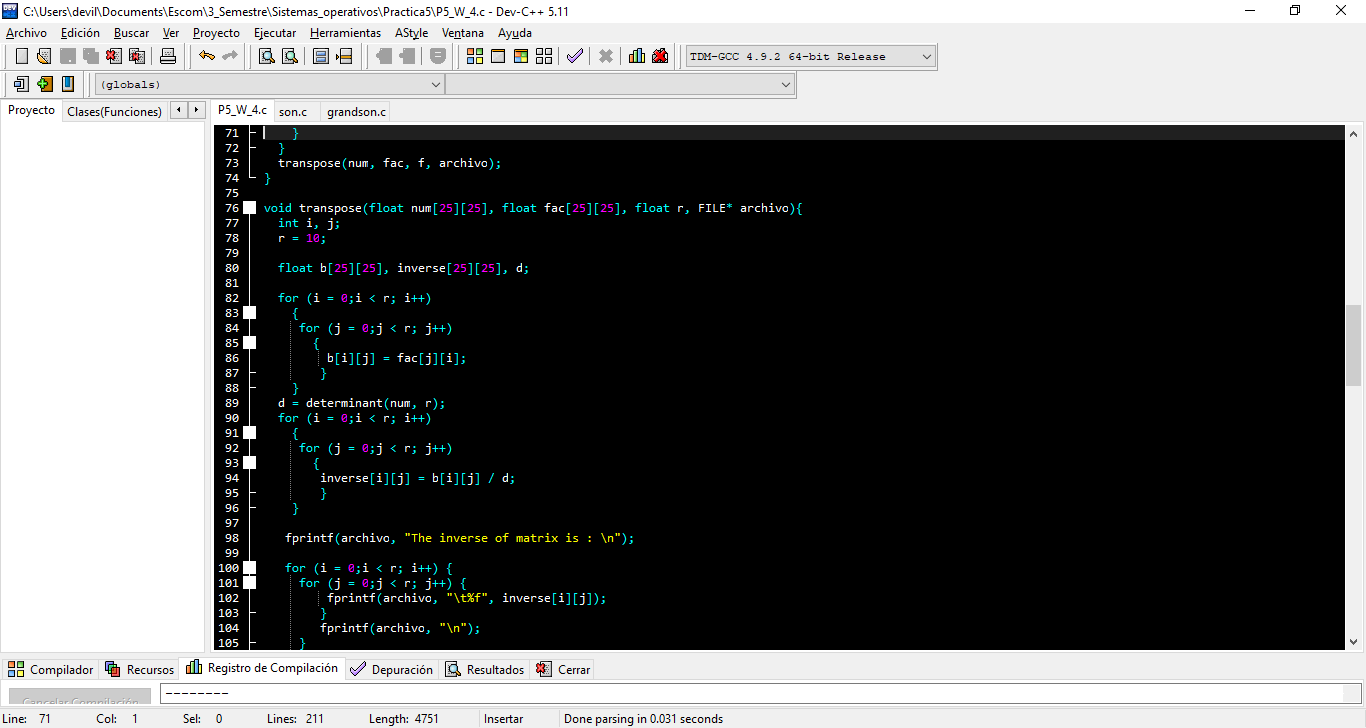
### Codificación en Windows

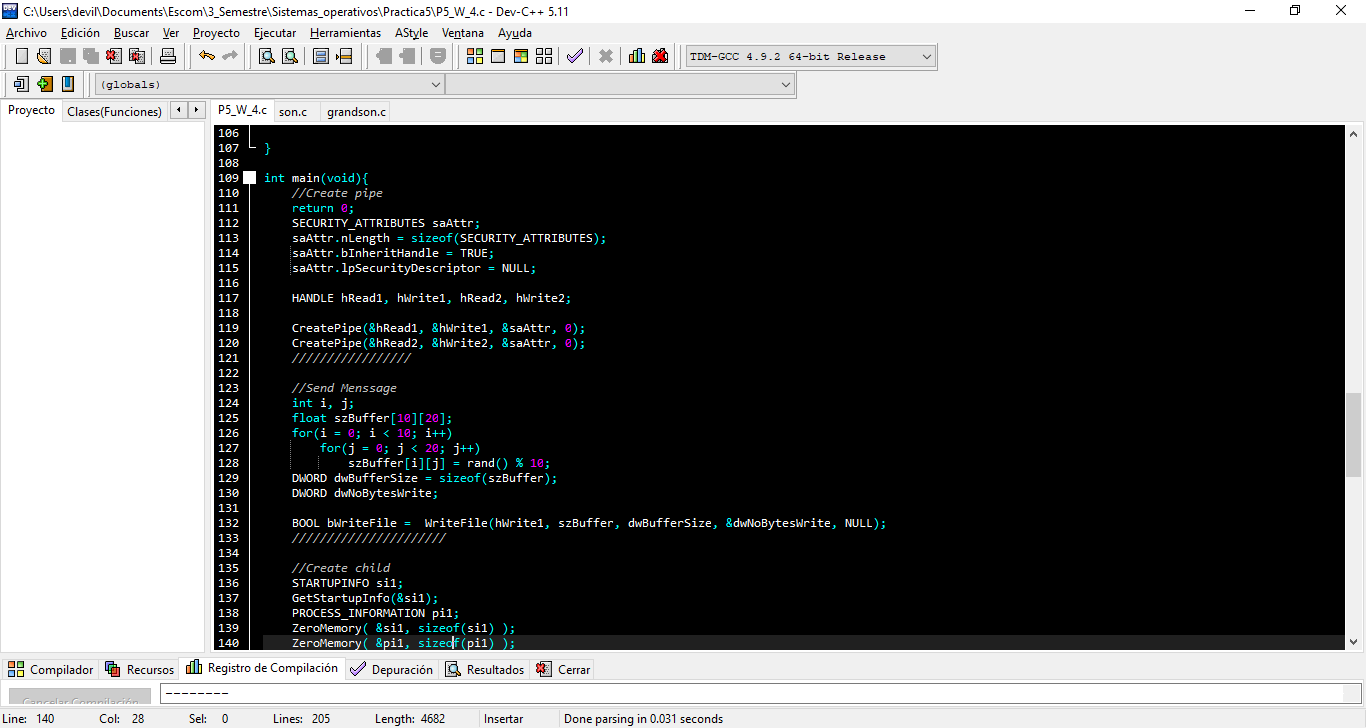
**Codigo fuente**

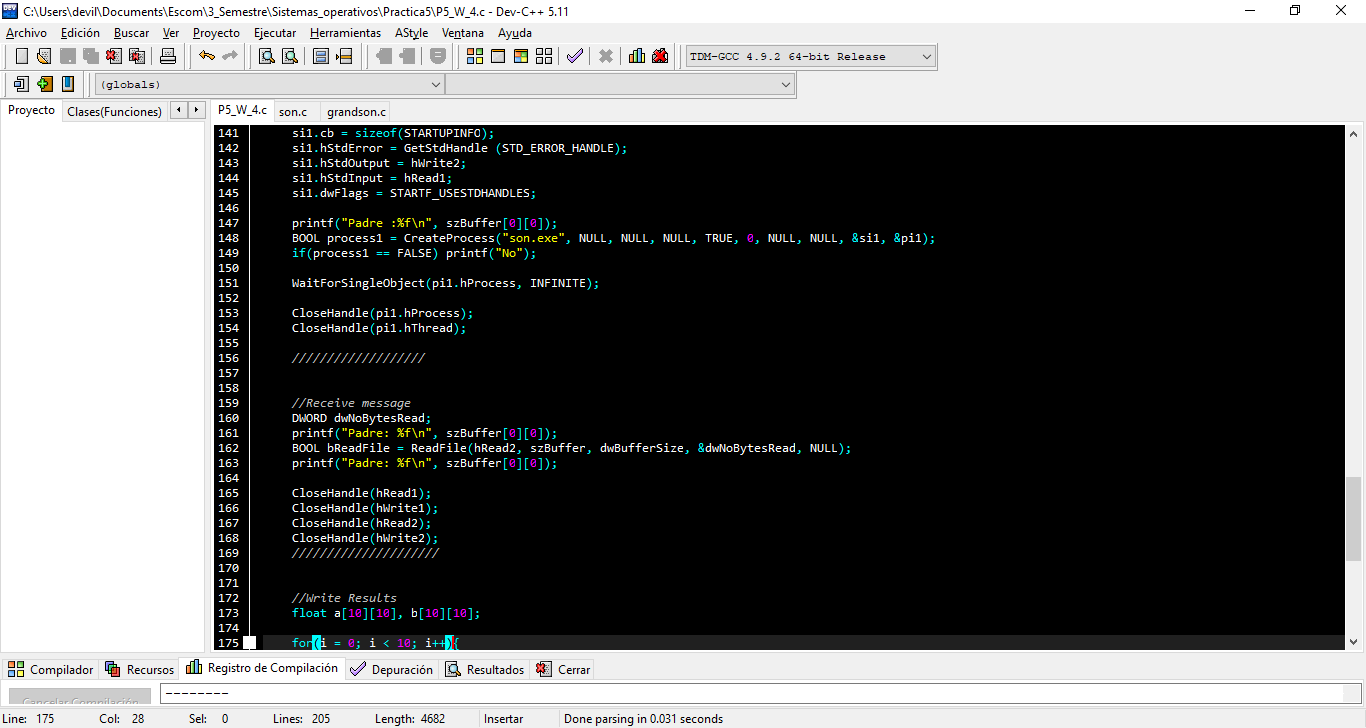
**Código de padre**

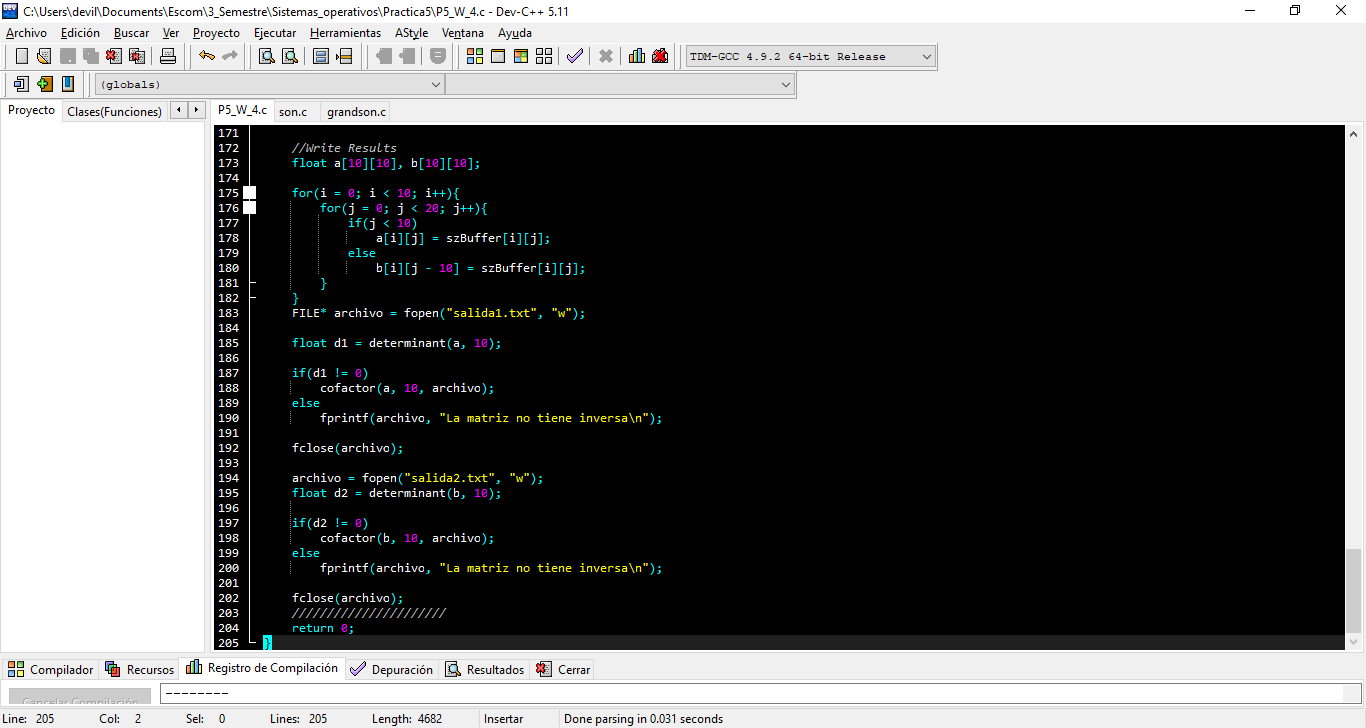




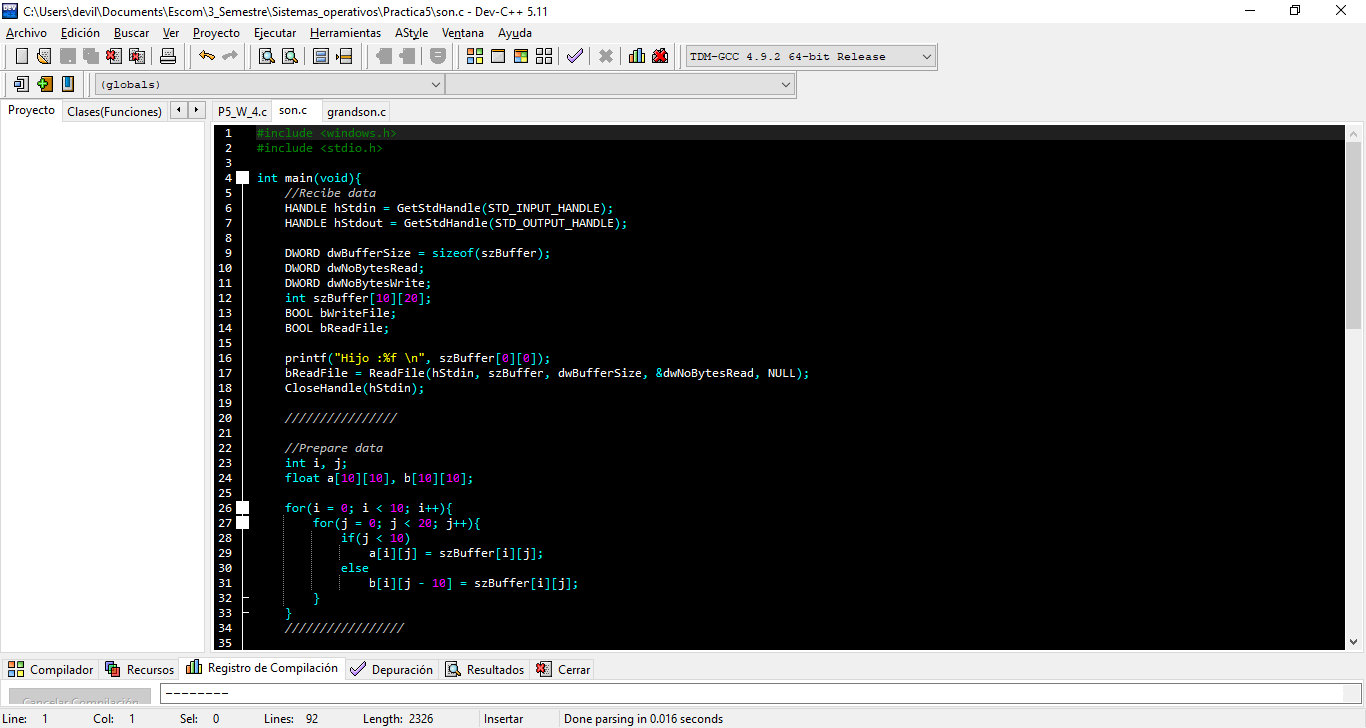


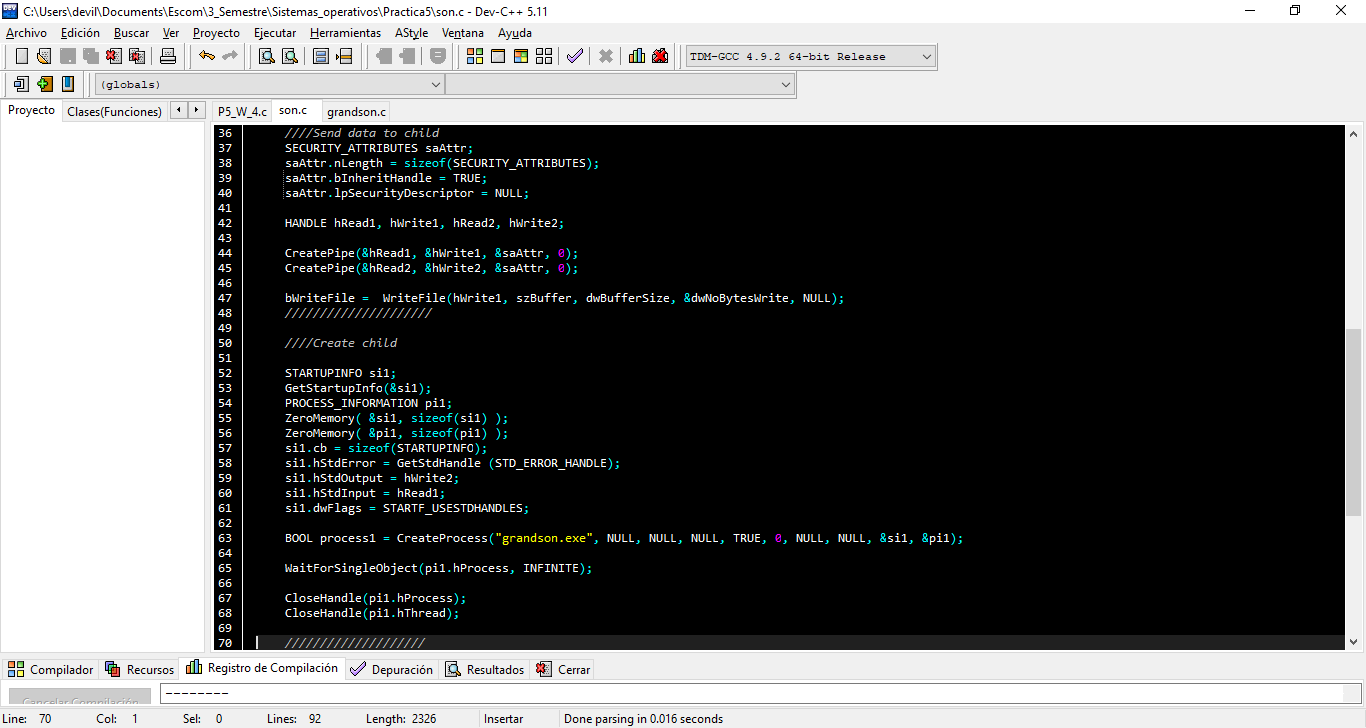


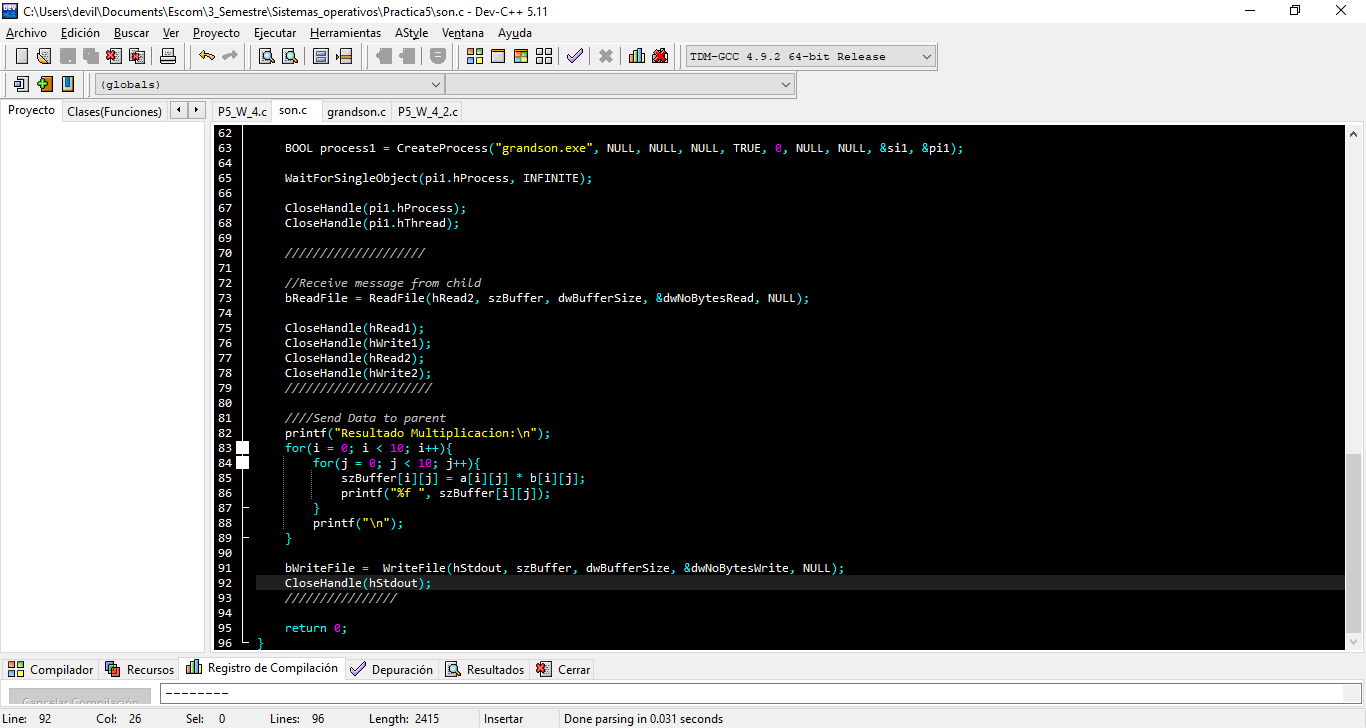




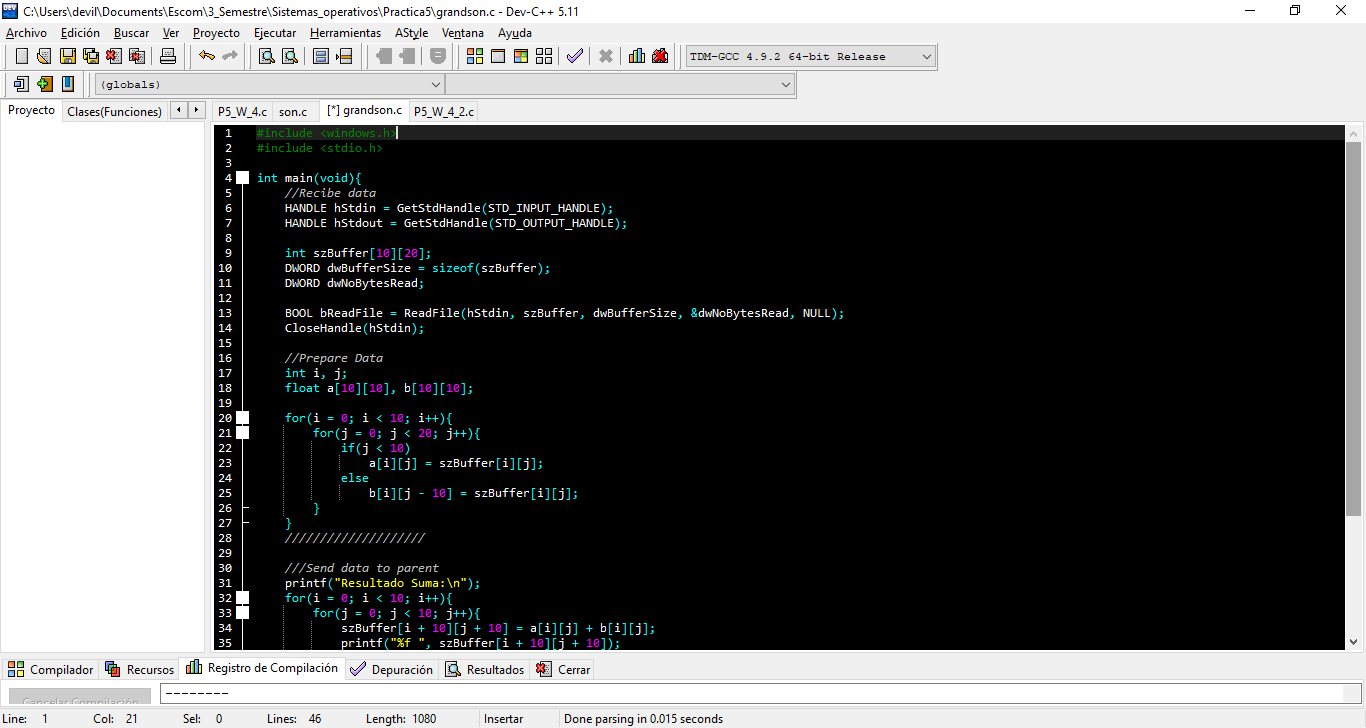
**Código Hijo**

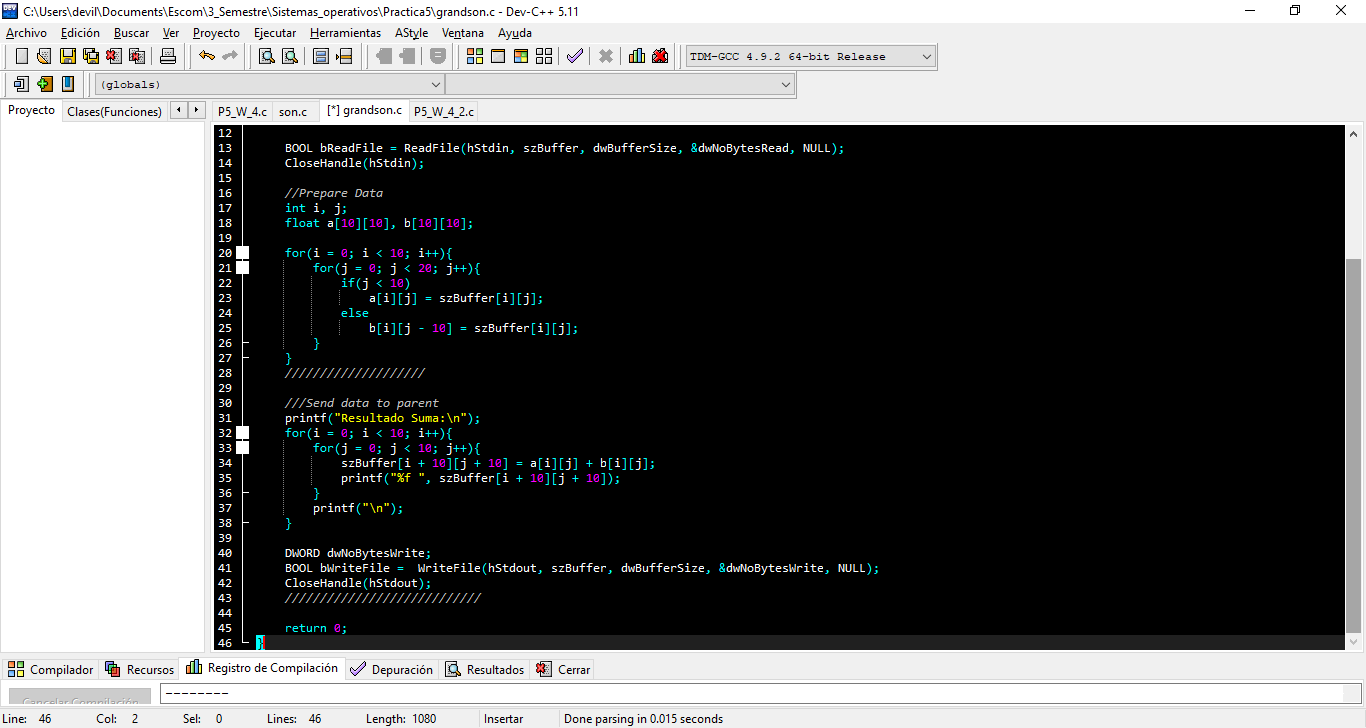




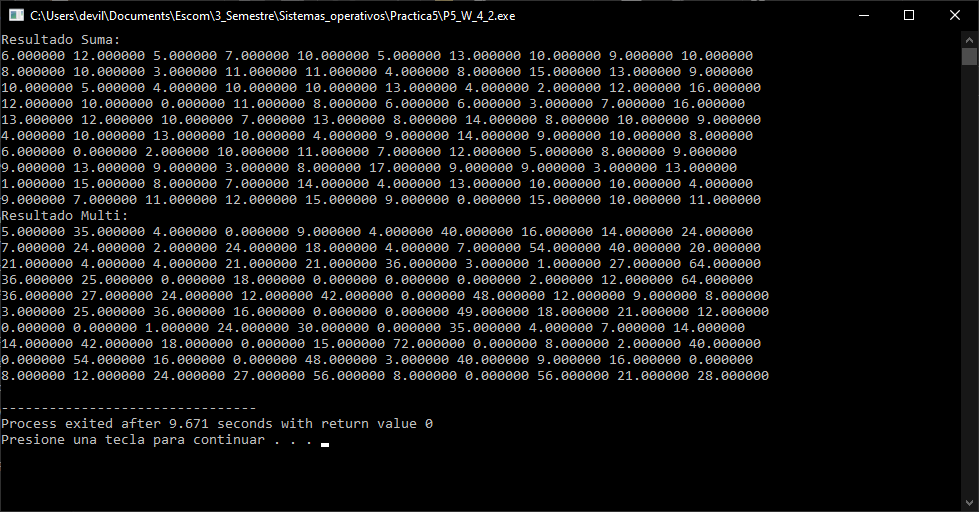


**Código Nieto**



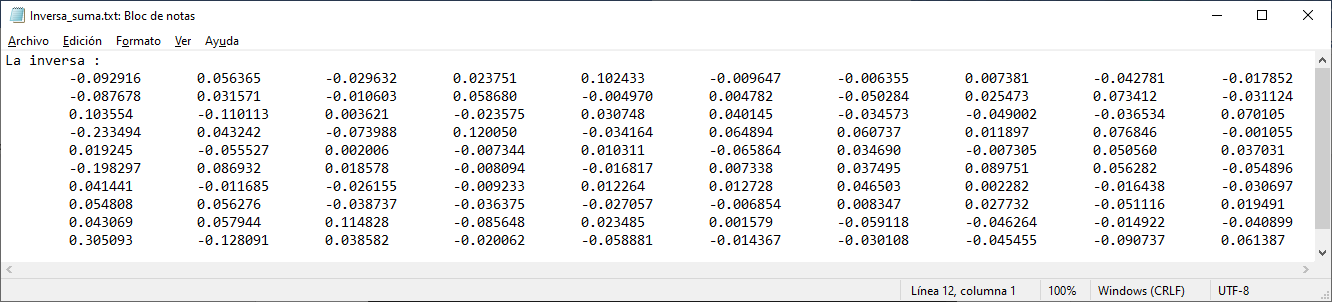


**Compilación y ejecución del programa**

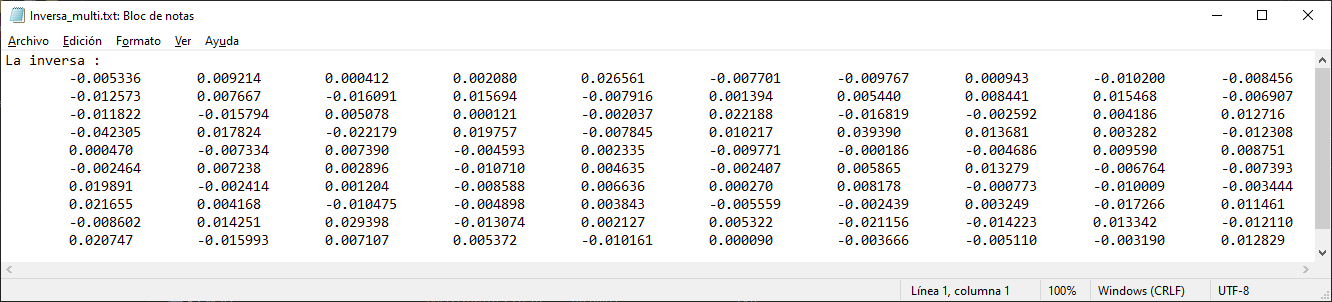


Los archivos generados por el programa y los cuales contendrán a las inversas de las matrices se muestran a continuación

Inversa suma



Inversa Multiplicación

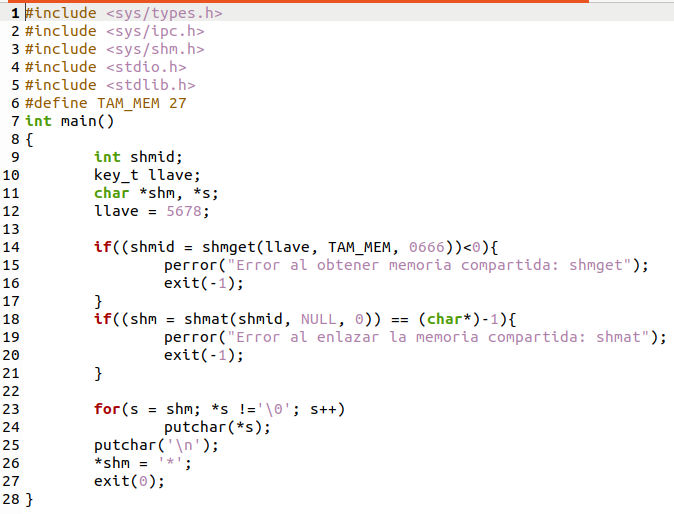


## Ejercicio 5

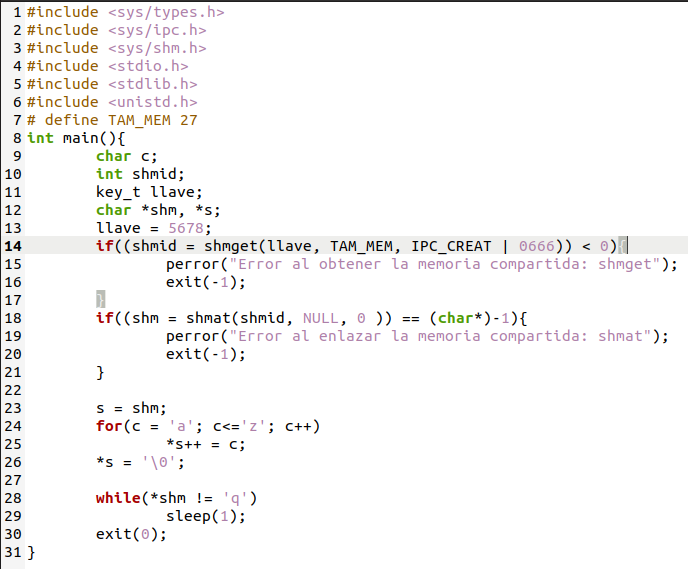
Capture, compile y ejecute los siguientes programas para Linux. Observe su funcionamiento.

**Codigo capturado**

**Código de Cliente de memoria compartida**

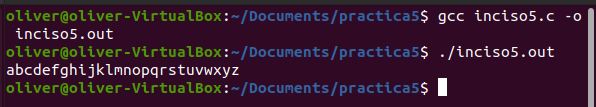


**Código de Servidor de la Memoria Compartida**

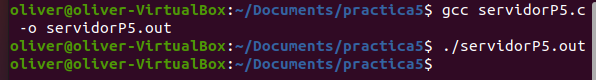


**Compilación y ejecución del programa**

Compilación y Ejecución: Cliente.



Compilación y Ejecución: Servidor.

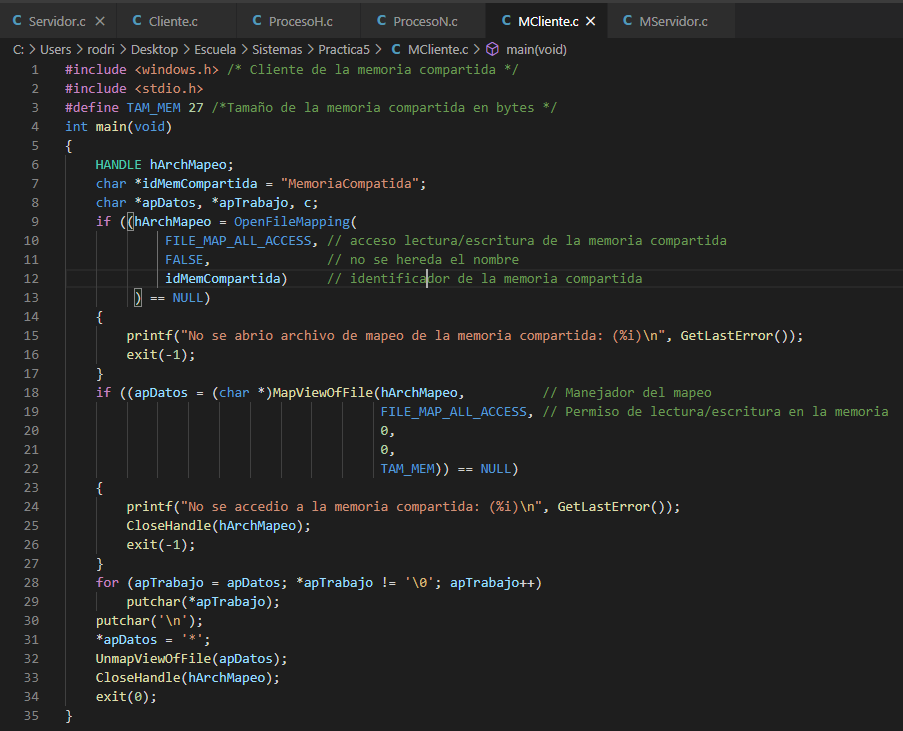


## Ejercicio 6

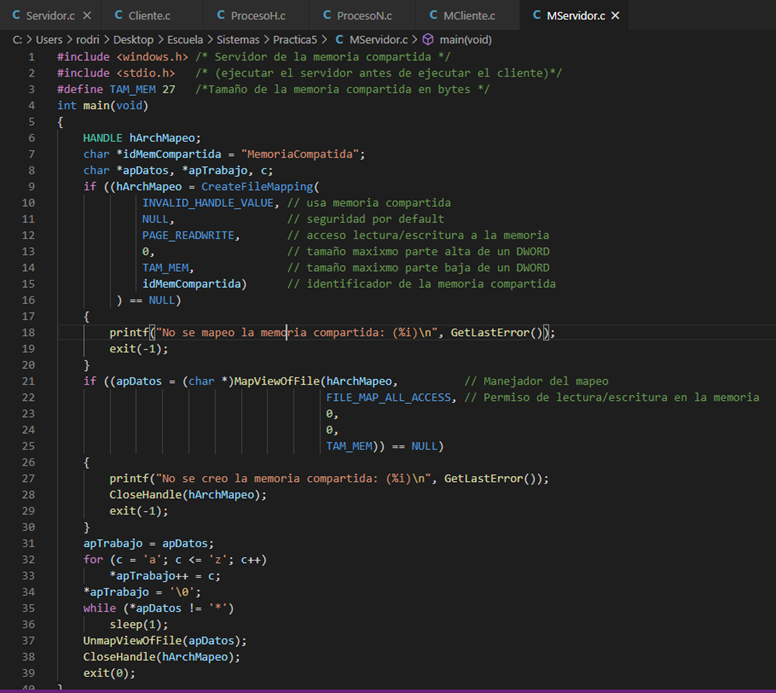
Capture, compile y ejecute los siguientes programas para Windows. Observe su funcionamiento.

**Codigo capturado**

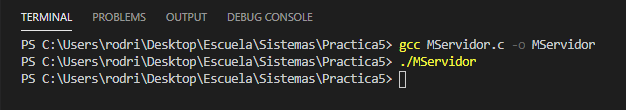
**Código del cliente de memoria compartida**

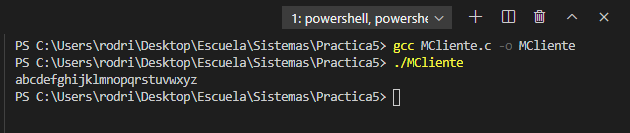


**Código del servidor de memoria compartida**



**Compilación y ejecución del programa**



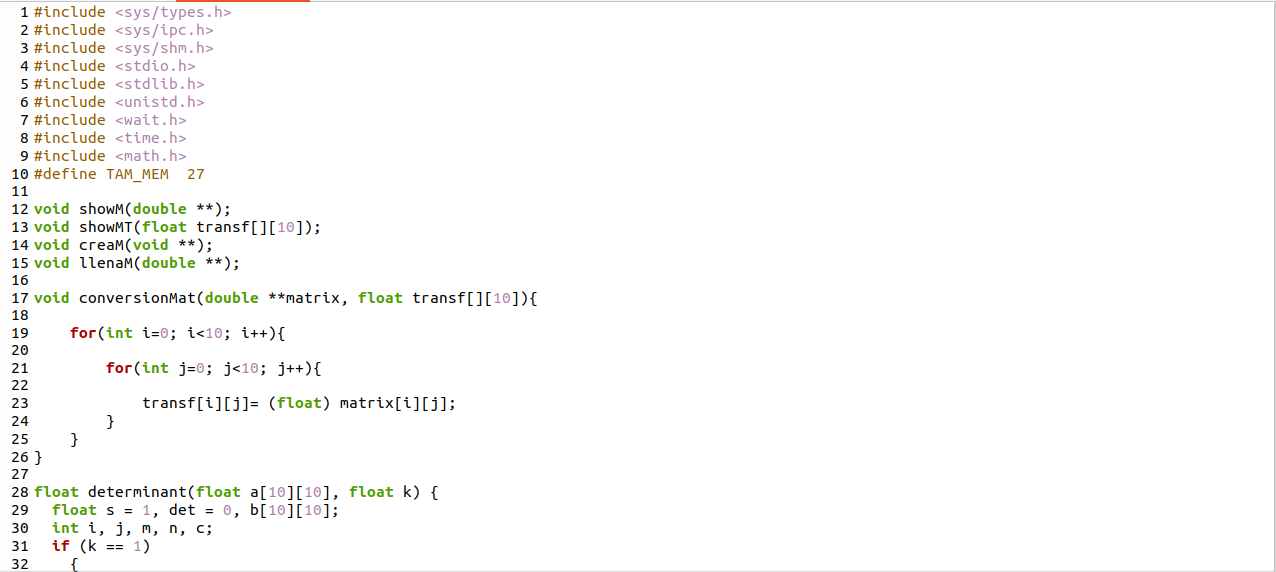


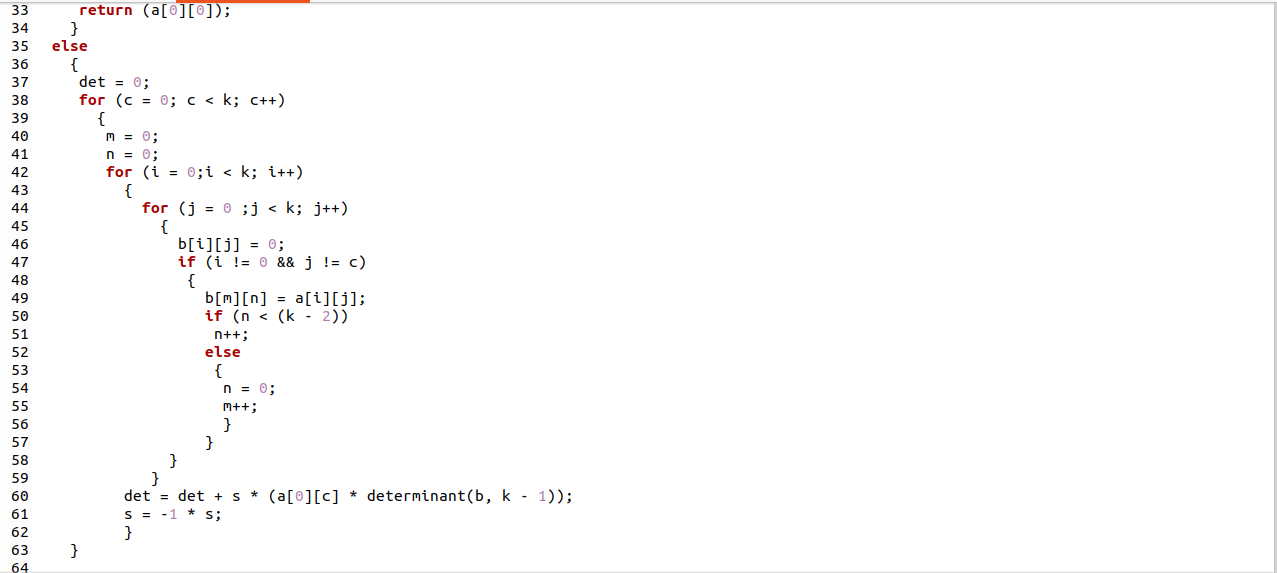
## Ejercicio 7

Programe nuevamente la aplicación del punto cuatro utilizando en esta ocasión memoria compartida en lugar de tuberías (utilice tantas memorias compartidas como requiera). Programe esta aplicación tanto para Linux como para Windows utilizando la memoria compartida de cada sistema operativo.

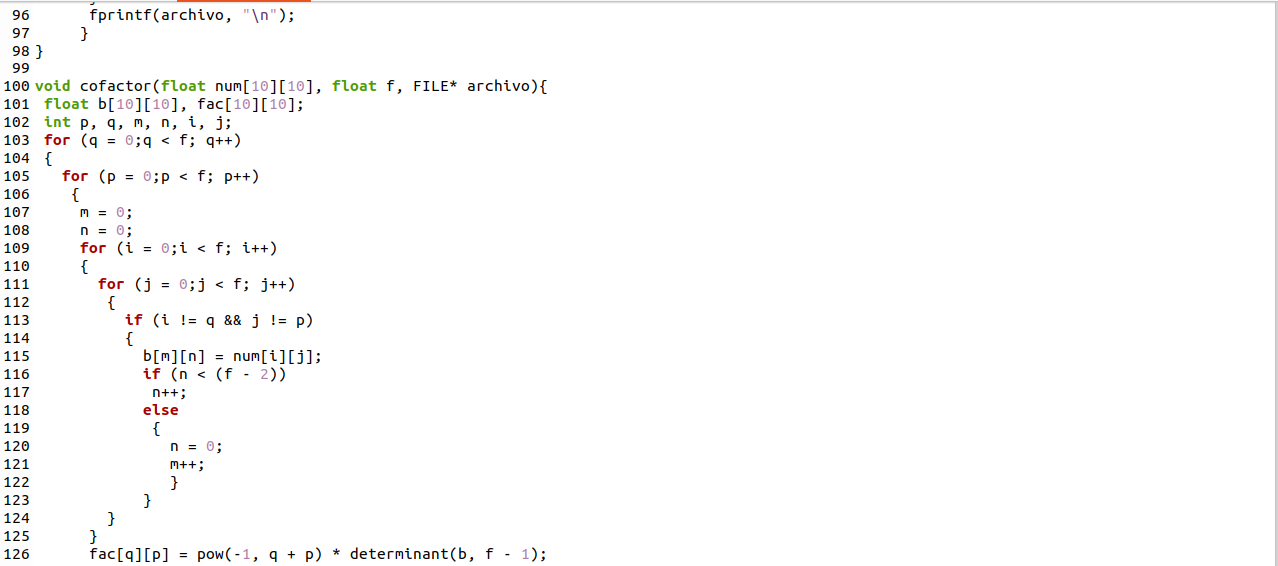
### Codificación en Linux

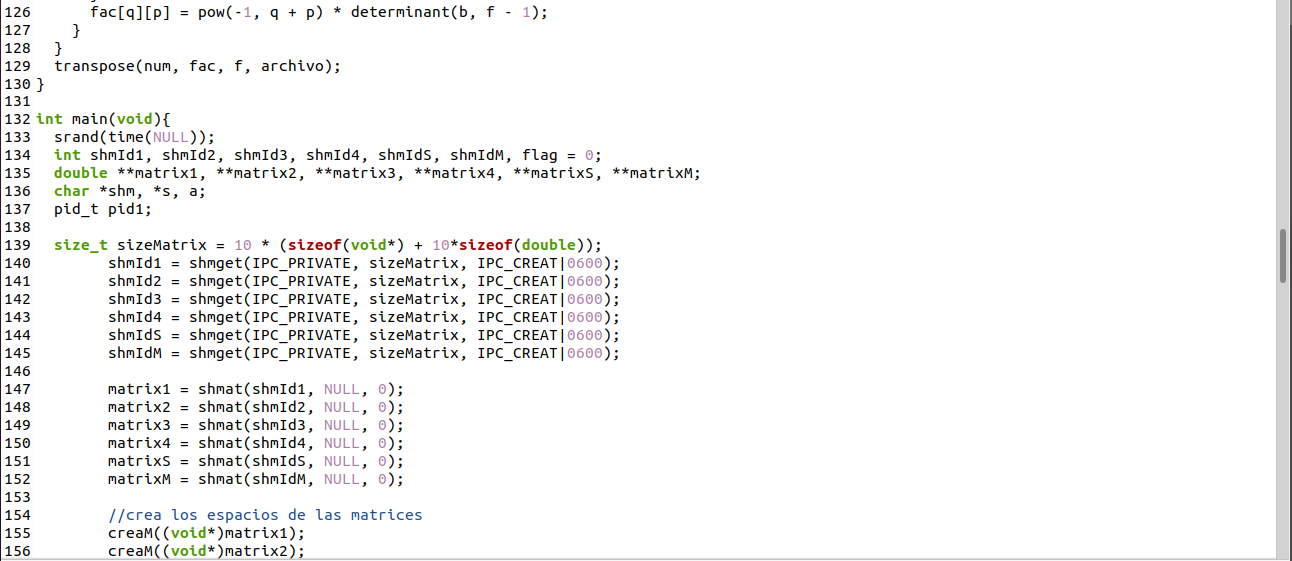
**Codigo fuente**



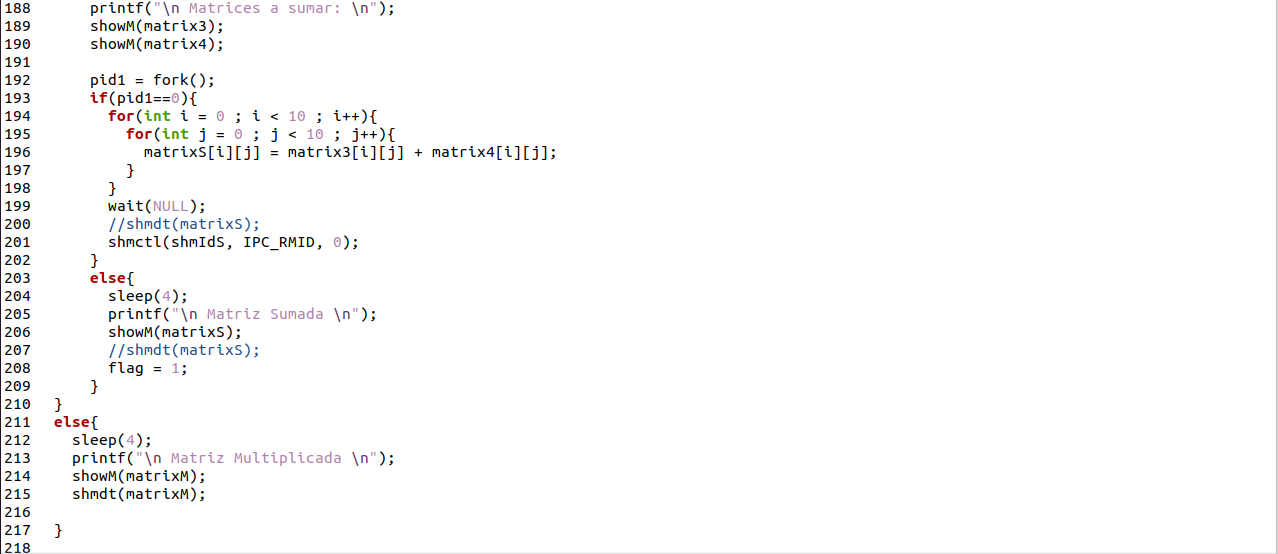


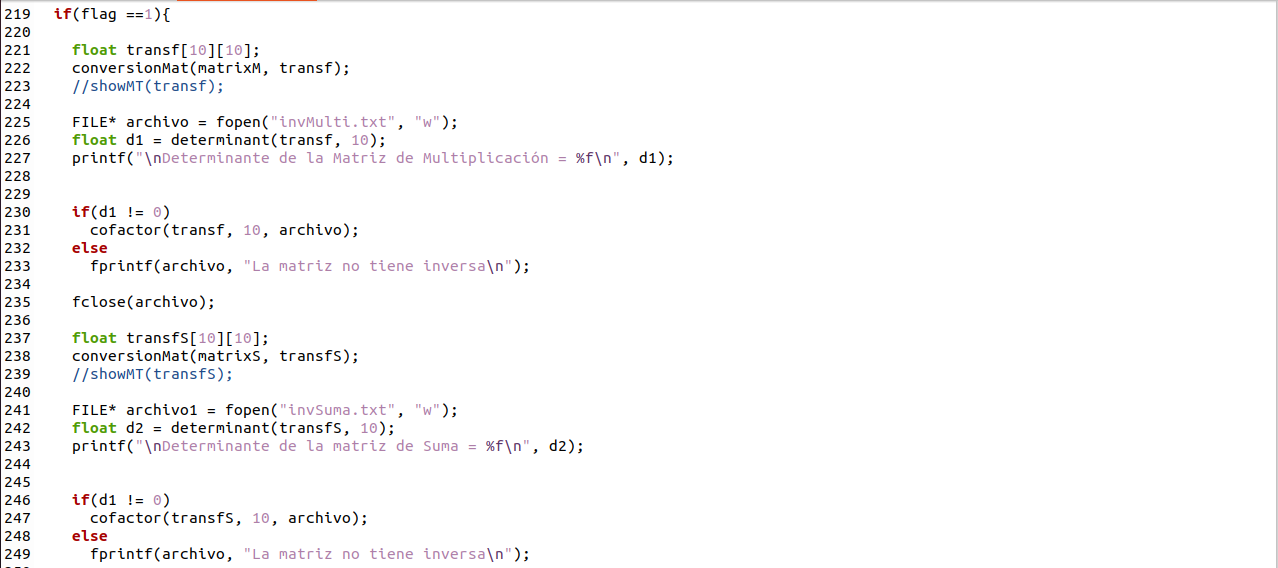


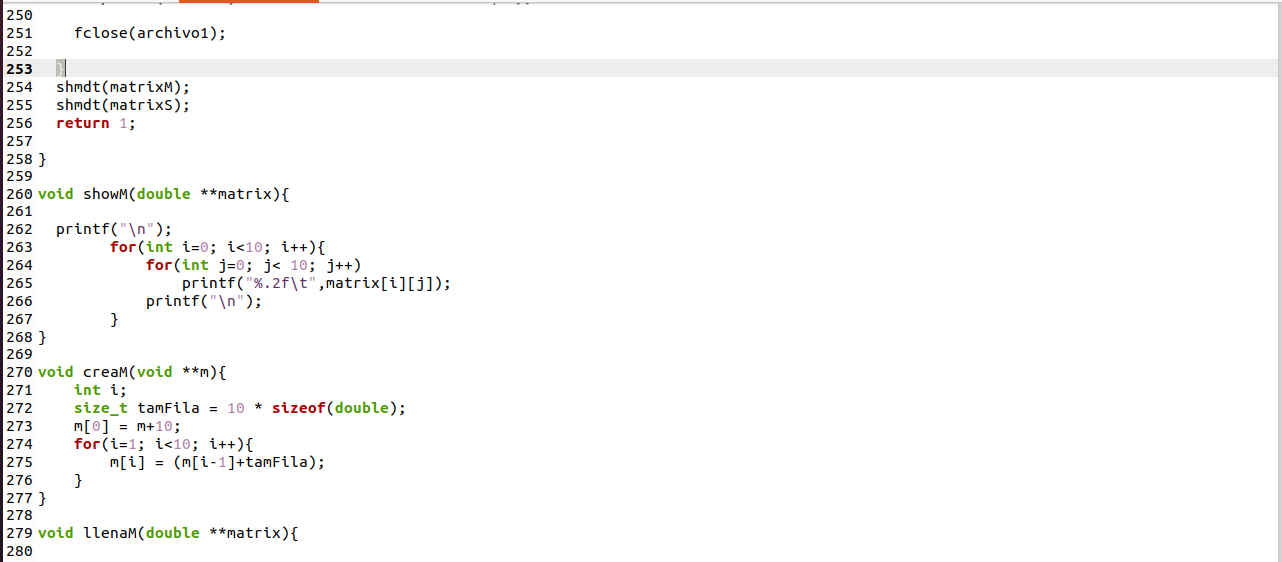


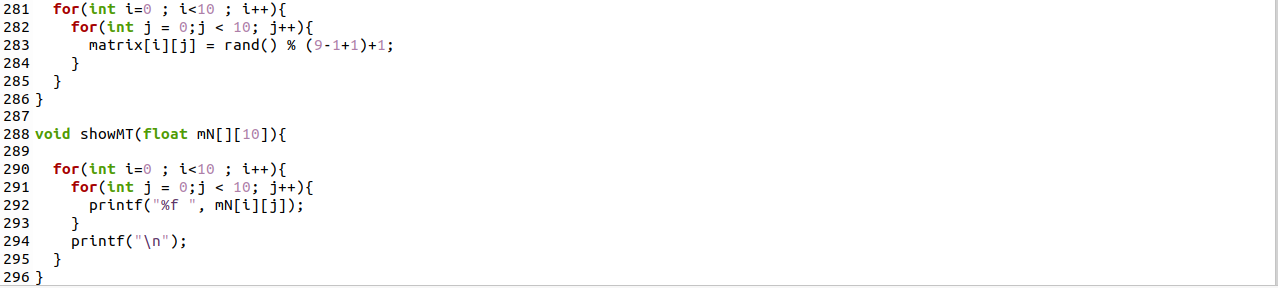




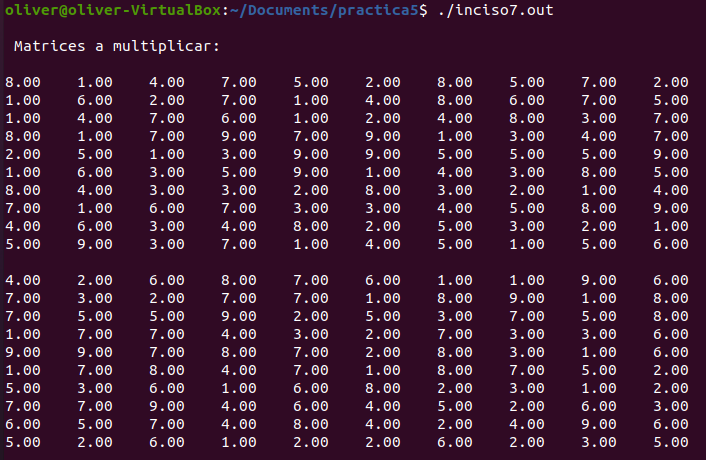


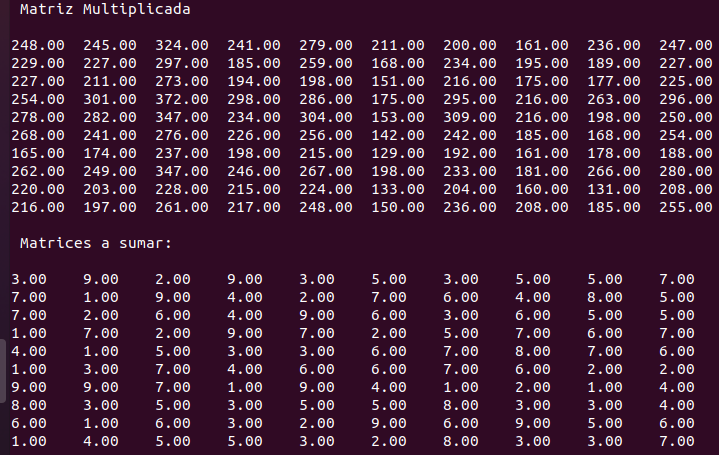


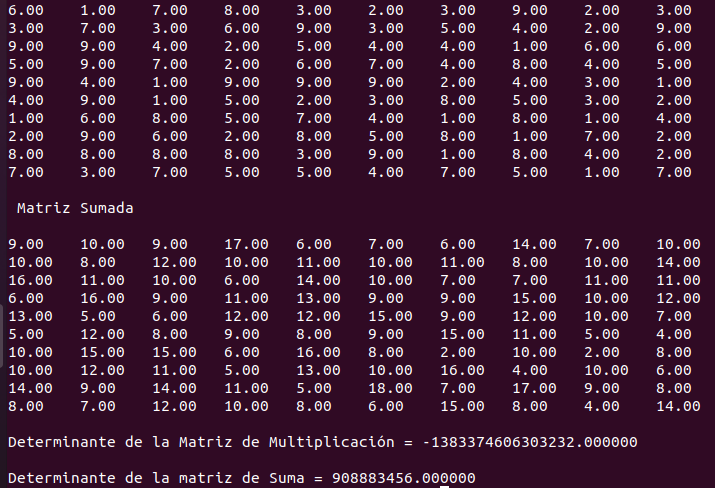




**Compilación y ejecución del programa**





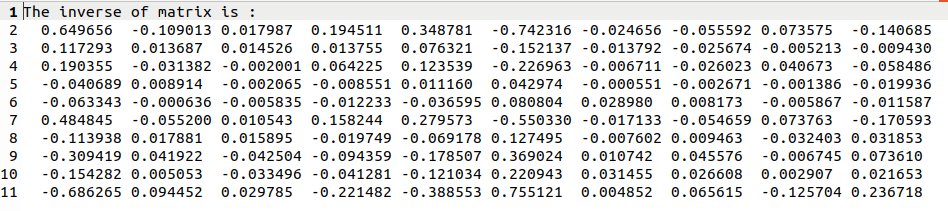


Archivos creados:



Contenido de los archivos, los cuales contienen la matriz inversa

InvMulti.txt

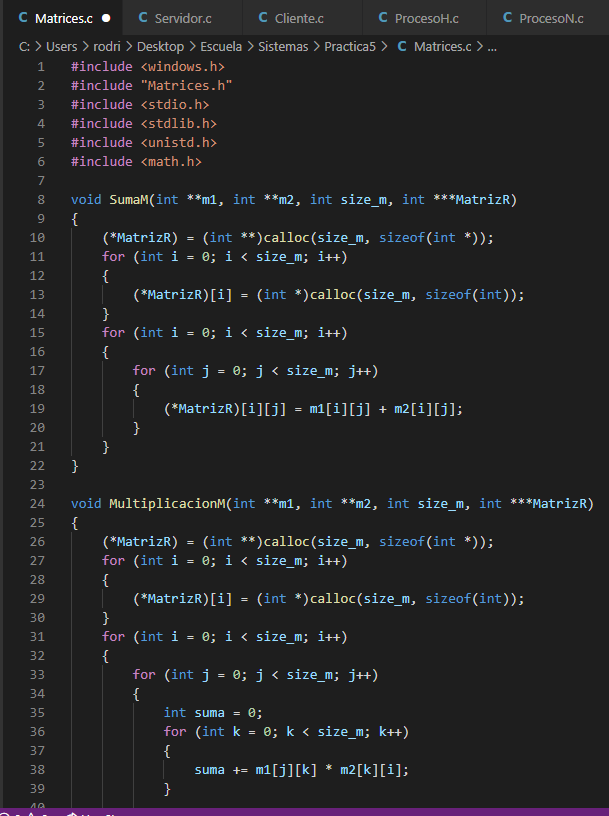


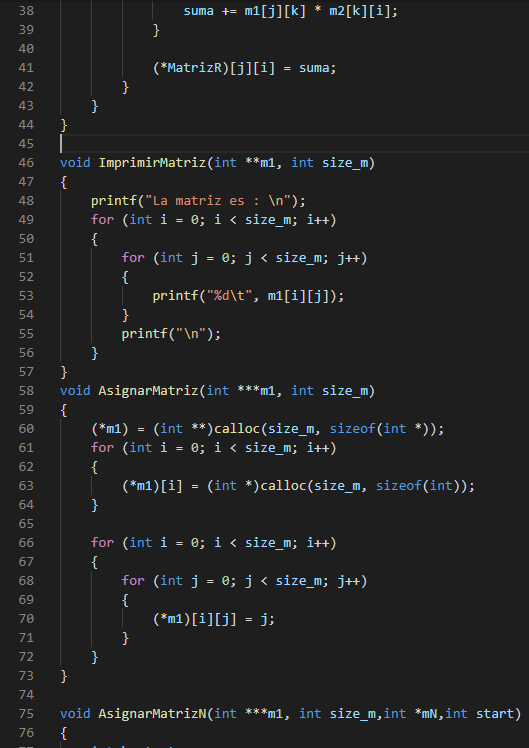
invSuma.txt

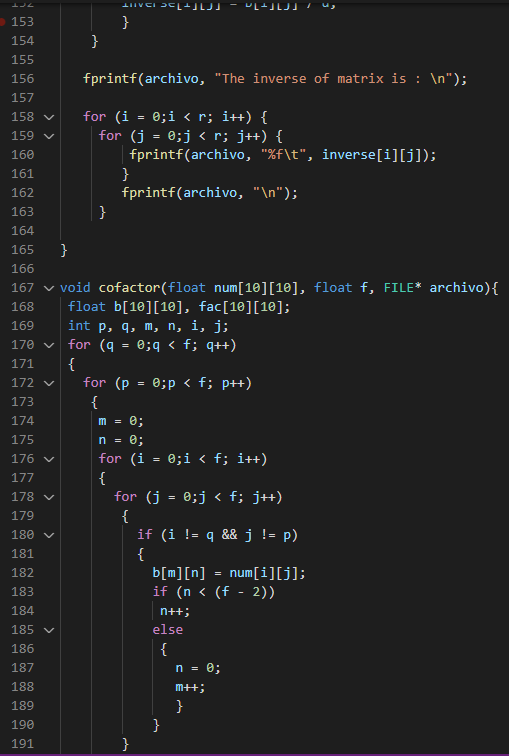
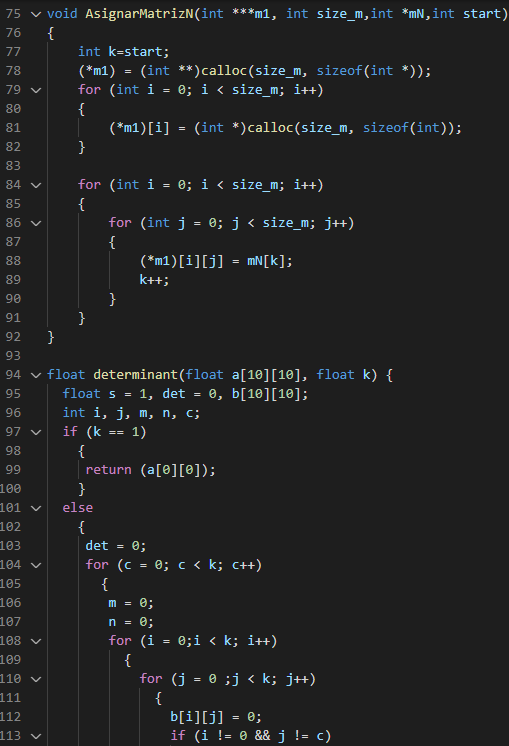


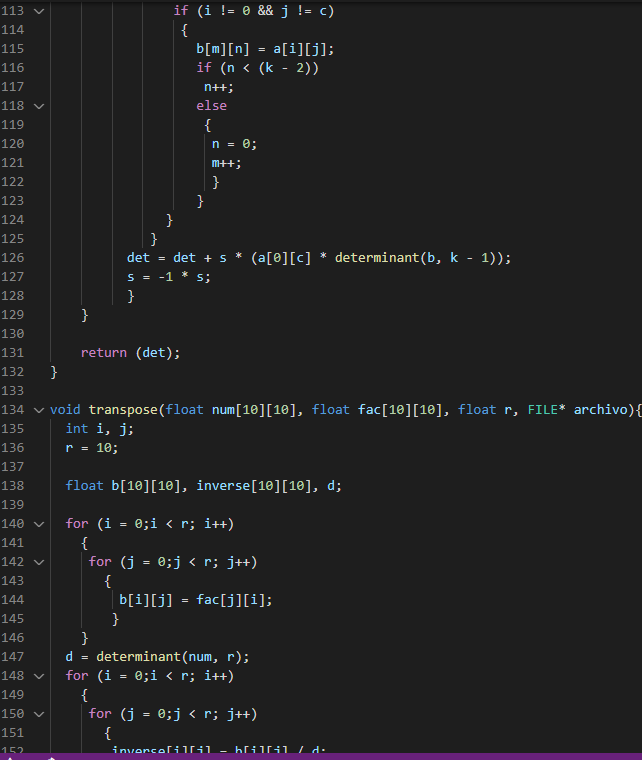
### Codificación en Windows

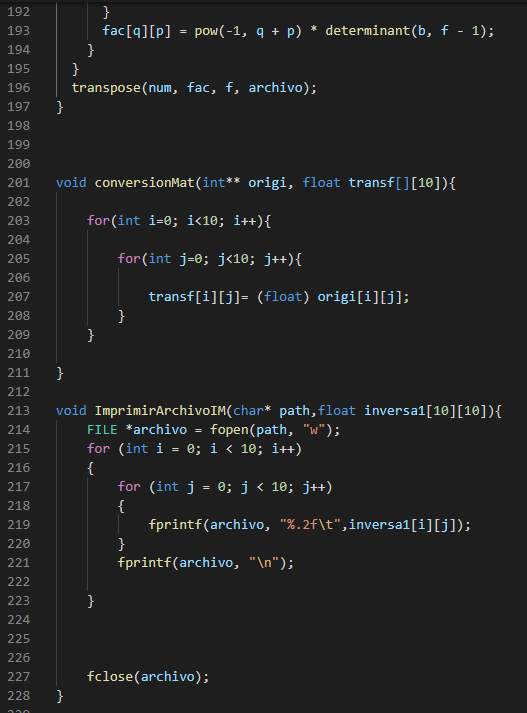
**Codigo fuente**

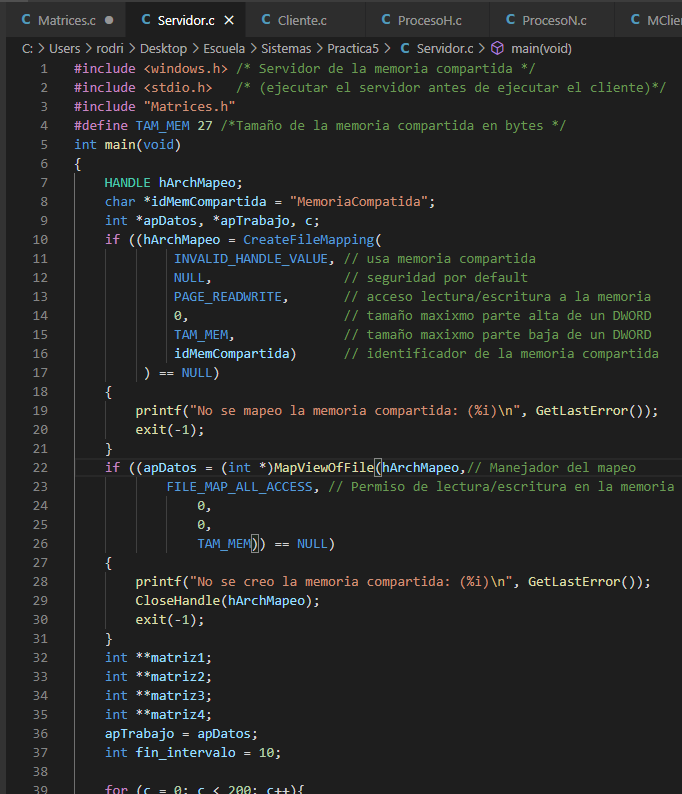


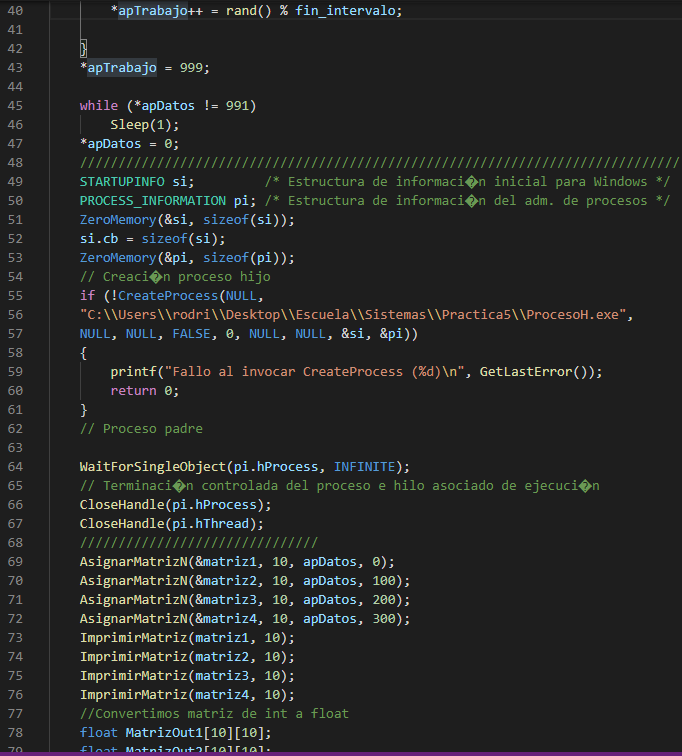


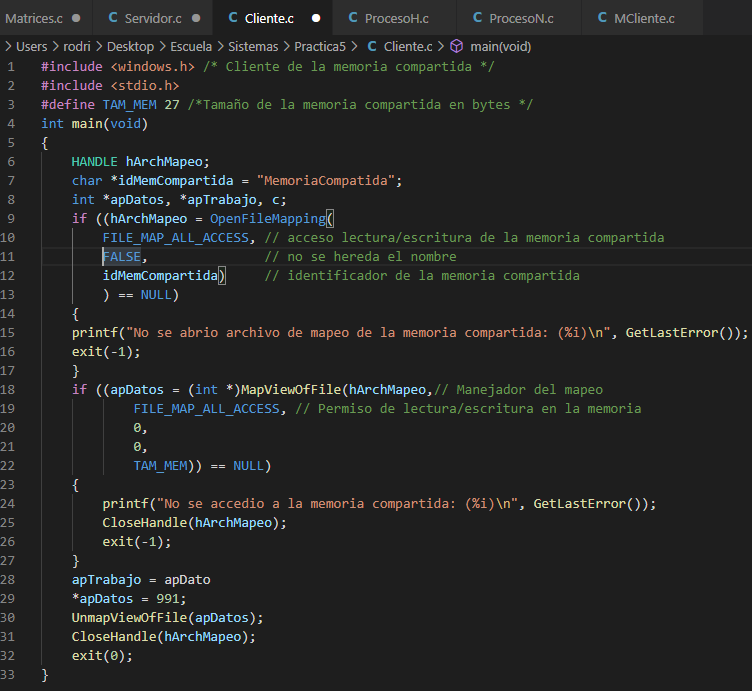


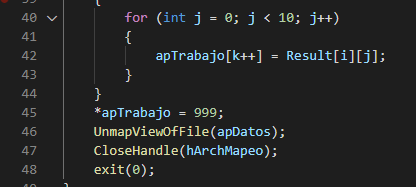
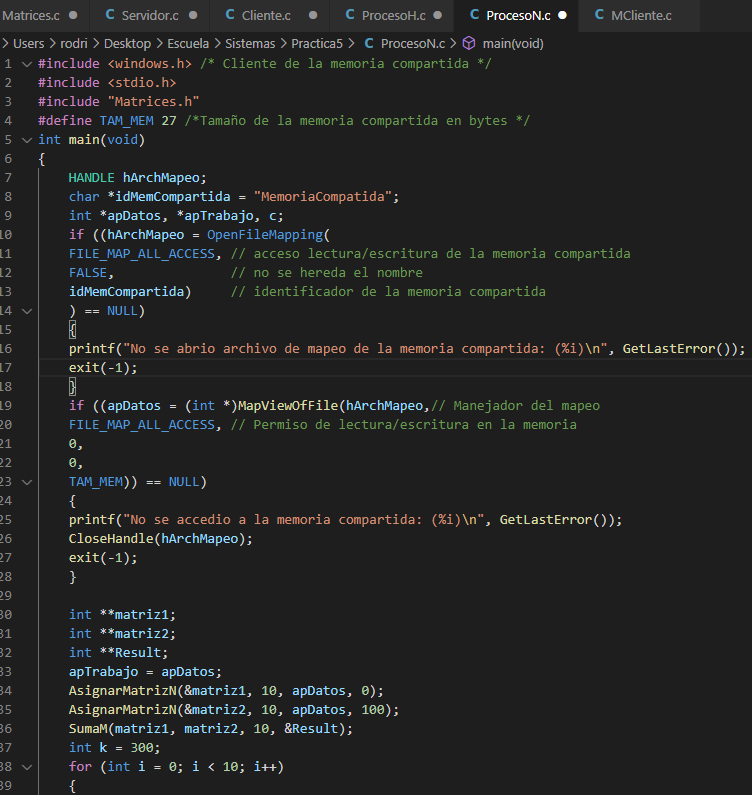
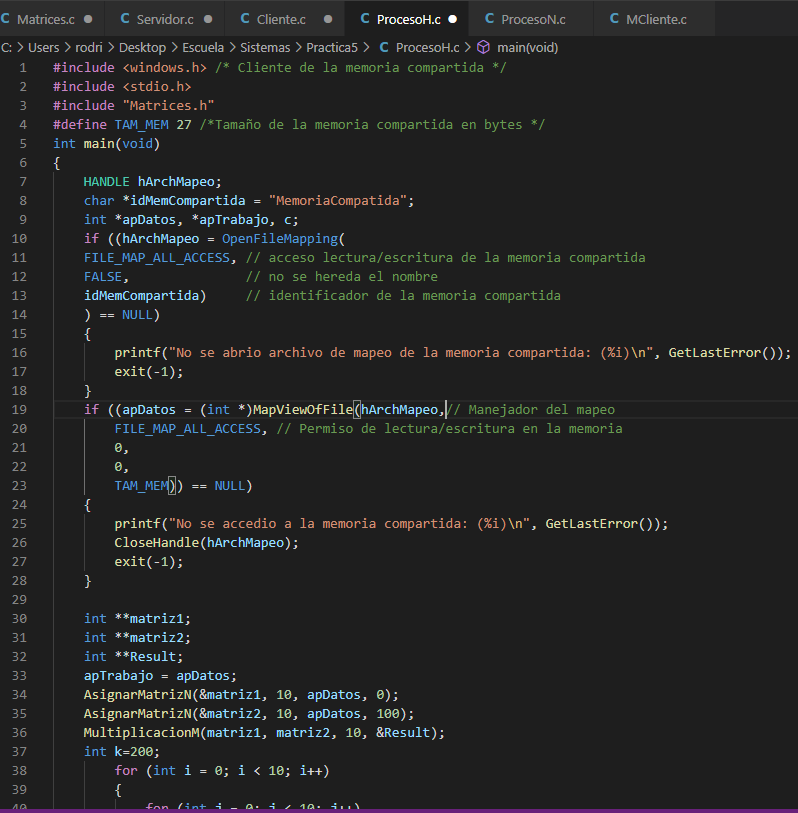




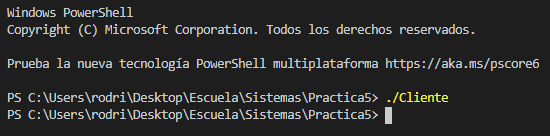


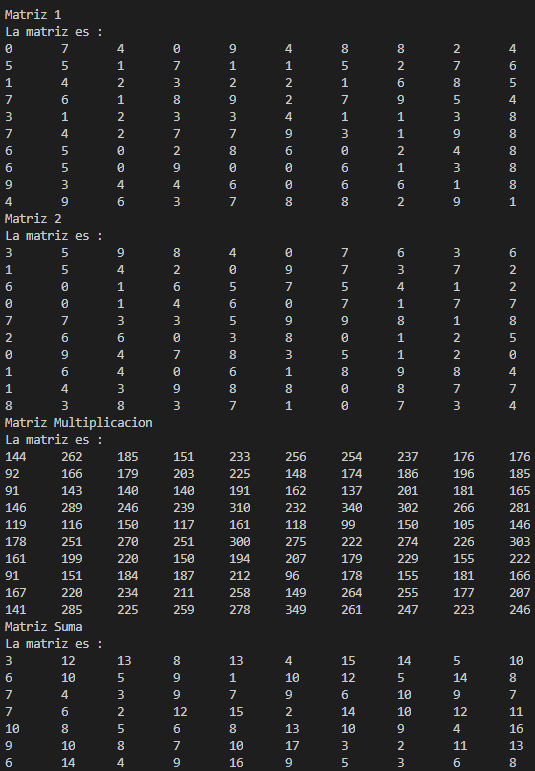






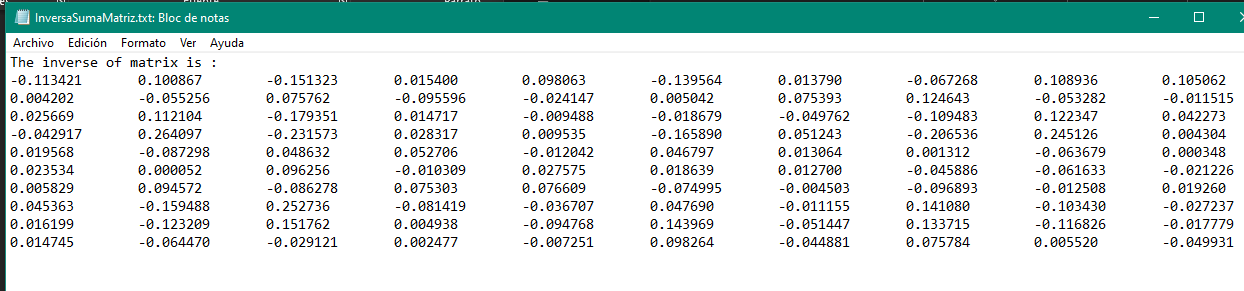
**Compilación y ejecución del programa**

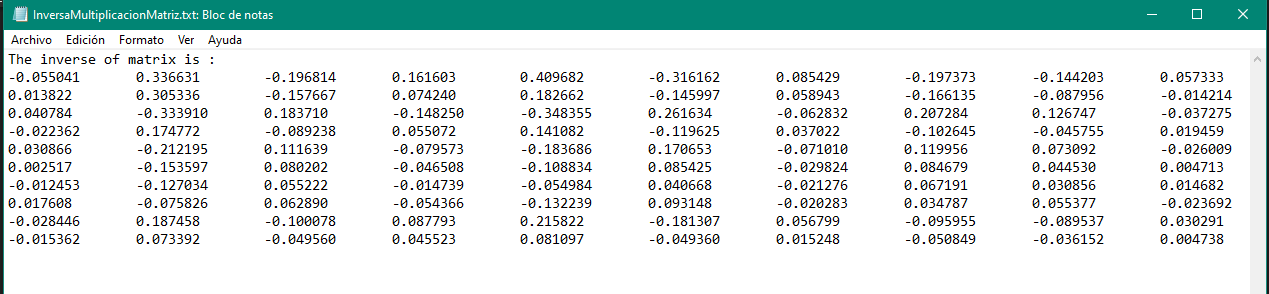






Archivos generados donde se encuentran las inversas de la matriz multiplicación y de la matriz suma





# Conclusiones