

Universidad católica Andrés Bello

Escuela de Ingeniería en Informática

Cátedra: Algoritmos y Programación II

**Gestión de memoria eficiente sobre matrices dispersas**

**Docente**: **Integrantes**:

Larez Jesus Patiño, Angel

González, Leonardo

Ciudad Guayana, Noviembre del 2022

**Análisis del Problema:**

**Premisas:**

1. Nunca se podra guardar 0 (Ceros) en las columnas de la matriz.

2. La solucion y las funciones a usar estan planteadas mediante el uso de listas simplemente enlazadas.

3. Se deben introducir matrices con dimensiones (Fila x Columna) mayores a 0 (Cero).

4. La opcion de llenado de las matrices por automatico estara definido por numeros en el conjunto N = {-1, 0, 1} con el fin de acelerar las pruebas y para poder trabajar con matrices de gran dimension.

**Requerimientos:**

1. Para la compilacion del programa, se debera usar el fichero makefile, el cual ya estara previamente definido en la solucion. Unicamente se debera introducir el comando **make** en la terminal, y todas los archivos seran compilados a la vez.

2. Recomendado el uso de la terminal nativa de los distros de Linux, sobre todo para mejor visualizacion del menu del programa.

**Analisis:**

|  |  |  |
| --- | --- | --- |
| **ENTRADAS** | **PROCESOS** | **SALIDAS** |
| Dimensiones de las matrices a trabajar:  \* Filas de Matrices 1 y 2  \* Columnas de Matrices 1 y 2 | 1. Llenado de la matriz mediante asignacion manual o automatica (En el caso de asignacion automatica leer la premisa numero 4).  2. Seleccionar el tipo de operación a realizar en las matrices previamente creadas  \*Imprimir = Mostrar las filas y columnas con valores diferentes de 0.  \*Buscar Elemento = Busqueda del elemento existente en la matriz (Se requerira la fila y columna a buscar).  \*Asignar Elemento = Asignar un elemento dado en la posicion deseada de la matriz.  \*Producto por Escalar = Multiplicar todos los elementos existentes de la matriz por un escalar dado.  \*Sumar = Sumar las dos matrices dadas por el usuario al inicio del programa.  \*Transpuesta = Crear la matriz transpuesta indicada por el usuario.  \*Multiplicacion = Mostrar el resultado del producto de dos matrices.  3. Mostrar la salida. | \* Resultados obtenidos por cada caso de operación especificado en el menu del programa.  \* Cierre del programa por decision del usuario. |

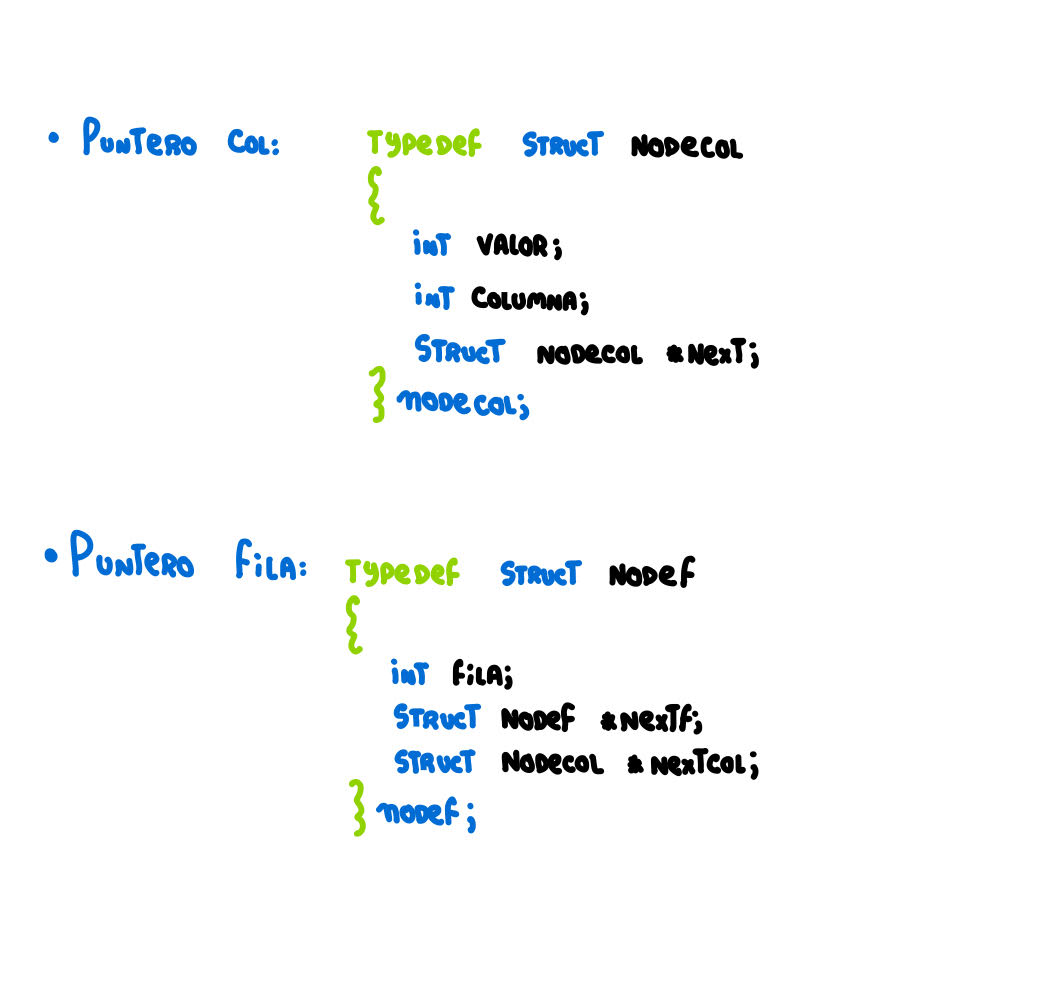
**Diseño de la Solucion:**

**Descripcion:**

Lograr implementar una matriz usando listas simplemente enlazadas podria considerarse algo no tan complejo, sin embargo, el “problema” llega a la hora de querer acceder eficazmente a los elementos para la realizacion de operaciones, pues el acceso a columnas es considerablemente mas complejo.

Es por ello que una correcta implementacion es necesaria para reducir las posibles complicaciones a la hora de realizar operaciones entre matrices (como las existentes en el programa). Por dicho motivo, se considero las siguientes estructuras para las nodos:

**Estructura de los nodos (Filas y Columnas de las matrices):**

****

La estructura **nodecol** sera utilizada para el manejo de las columnas, cuenta con:

**valor:** Usado para almacenar el elemento asignado a esa posicion de la matriz.

**columna:** Creado para almacenar el numero de columna en determinada fila.

**nodecol \*next:** autoreferencia a la estructura para el enlace de otros nodos (columnas).

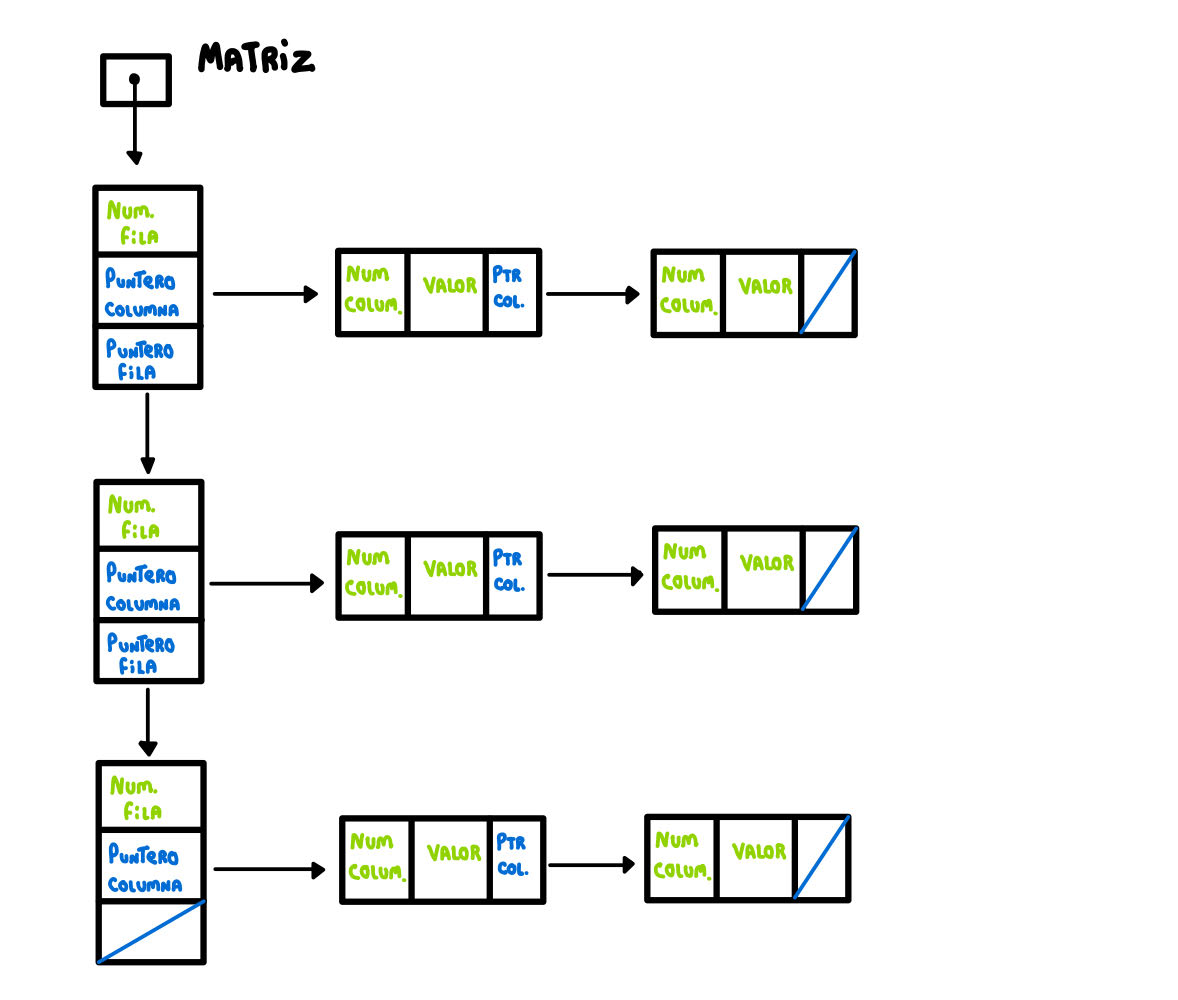
Por otro lado, se tiene la estructura **nodef**, la cual contiene:

**fila:** Almacena el numero de fila de la matriz.

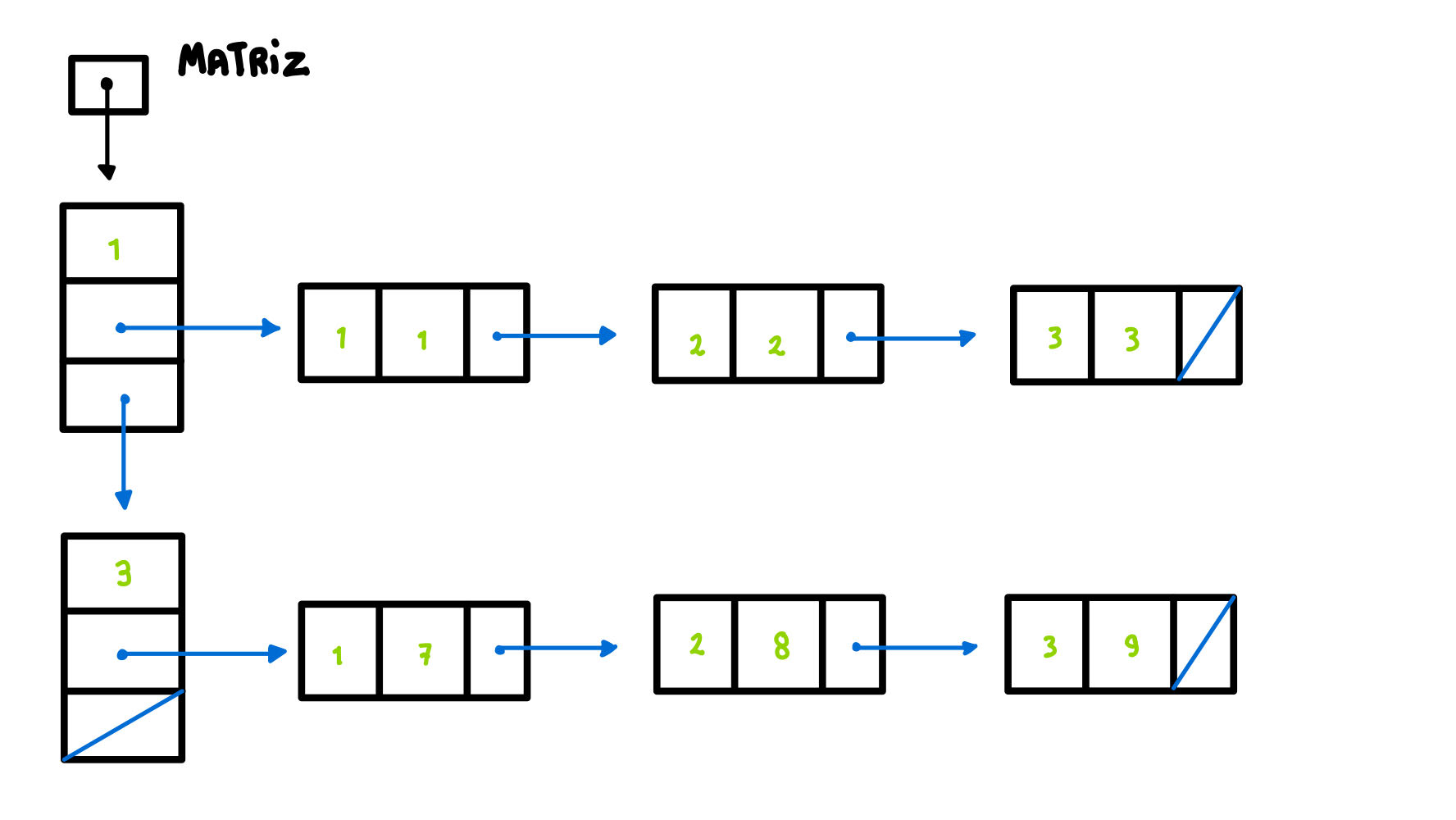
**nodef \*nextf:** autoreferencia a la estructura para el enlace de otros nodos (filas)

**nodecol \*nextcol:** referencia a la estructura **nodecol** para poder enlazar dos tipos de estructuras diferentes, permitiendo el enlace entre filas y columnas.

**Diagramas de la estructura base de las matrices:**



**Ejemplo ilustrativo de una matriz 3x3 creada bajo esta estructura:**



Como se puede observar, aquellas filas que contengan elementos inexistentes para efectos del programa (Elementos con valor igual a 0), simplemente no seran consideradas, reduciendo en gran cantidad el uso de memoria al no tener nodos “inservibles” que solo apuntan a columnas vacias.

**Justificacion de las Estructuras de Datos y Algoritmos utilizados:**

El principal motivo por el cual se considero esta estructura para la elaboracion de las matrices es por el simple hecho de que nunca se tendran nodos “irrelevantes” que apunten a columnas inexistentes, derivando en un uso mas eficiente de la memoria y facilitando asi ciertas operaciones en el programa.

Sin embargo, estas estructuras de nodos por si solas no son suficiente para lograr dicho uso eficiente de memoria, pues a la hora de crear las matrices la funcion encargada de realizar dicho proceso fue optimizada con el fin de lograr los objetivos deseados.

**Detalles de la Implementacion:**

**Descripcion de las funciones utilizadas en el programa**

**nodef \* new\_nodef(nodef \*m, int f);**

Esta funcion sera utilizada para la creacion de nuevos nodos de filas. Requerira un parametro de tipo nodef y un valor entero, el cual sera almacenado como el numero de cierta fila de la matriz.

**nodecol \*new\_nodecol(nodecol \*l, int col, int v);**

Funcion usada para la creacion de nuevos nodos de columnas, requiere un parametro de tipo nodecol y dos valores enteros. El primer valor entero sera el numero de cierta columna de la matriz y el segundo entero sera el valor del elemento a almacenar en dicha posicion.

**Nodecol \*add\_endj(nodecol \*link, nodecol \*new\_link);**

Esta funcion se encarga de enlazar los nuevos nodos de columnas creados. Tomara como parametro el nodo de filas en el cual se desea enlazar dicha columna. Dentro de la funcion se recorrera las columnas hasta que consegui el ultimo nodo con su proximo enlace apuntando a NULL.

**int valor\_aleatorio();**

Funcion creada con el fin de retornar valores aleatorios dentro del intervalor I = [-1, 1]. Unicamente utilizada para acelerar los procesos de prueba del programa.

**nodef \*new\_matrix(nodef \*M, nodecol \*tj, int i, int j, int op);**

Una de las funciones mas importantes del programa, pues esta es la que permitira la creacion de la matriz. Creara los nodos de filas y columnas, los enlazara y ademas verificara que ninguna fila “irrelevante” exista. Dependiendo de la decision del usuario, los valores de la matriz se asignaran de manera manual o en automatico.

**void Imprimir(nodef \*M);**

Funcion para imprimir todos los elementos existentes de determinada matriz. Para efectos del programa, solo se imprimira la fila, columna, y valor de dicha posicion, pues si se consideraran los ceros, seria casi imposible el visualizar matrices de gran tamaño.

**int ObtenerElemento(int i, int j, nodef \*M);**

Esta funcion buscara y retornara el elemento asignada a la posicion ingresada por el usuario.

**nodef \*AsignarElemento(int i, int j, int x, nodef \*M);**

Funcion utilizada para la asignacion de un elemento en determinada posicion de la matriz. Tomara como parametro la fila, columna, y el valor que se desea asignar en dicha posicion, ademas de la matriz a la cual se desea asignar el elemento.

**nodef \*ProductoPorEscalar(int e, nodef \*M);**

Establecida para retornar la matriz generada tras multiplicar cada elemento por el escalar “e”, dicho escalar sera ingresado por el usuario.

**nodef \*Suma(nodef \*M1, nodef \*M2);**

Esta funcion se encargara de sumar las matrices 1 y 2, se retornara una nueva matriz que tendra los elementos existentes de la matriz 1 y 2 sumados entre si.

**nodef \*Transponer(nodef \*M);**

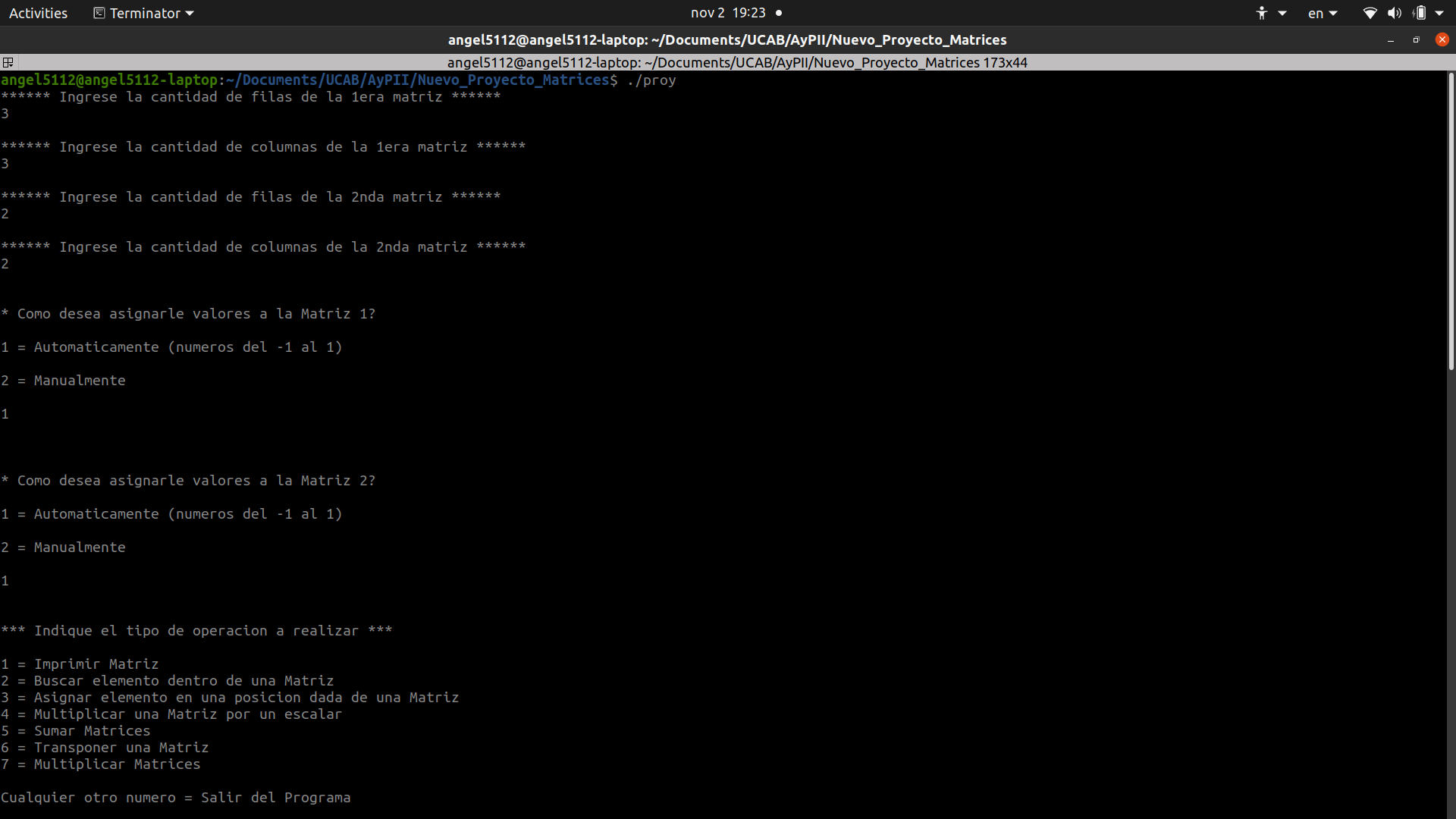
Funcion creada para transponer una matriz. Se retornara una nueva matriz que sera la transpuesta de la matriz ingresada.

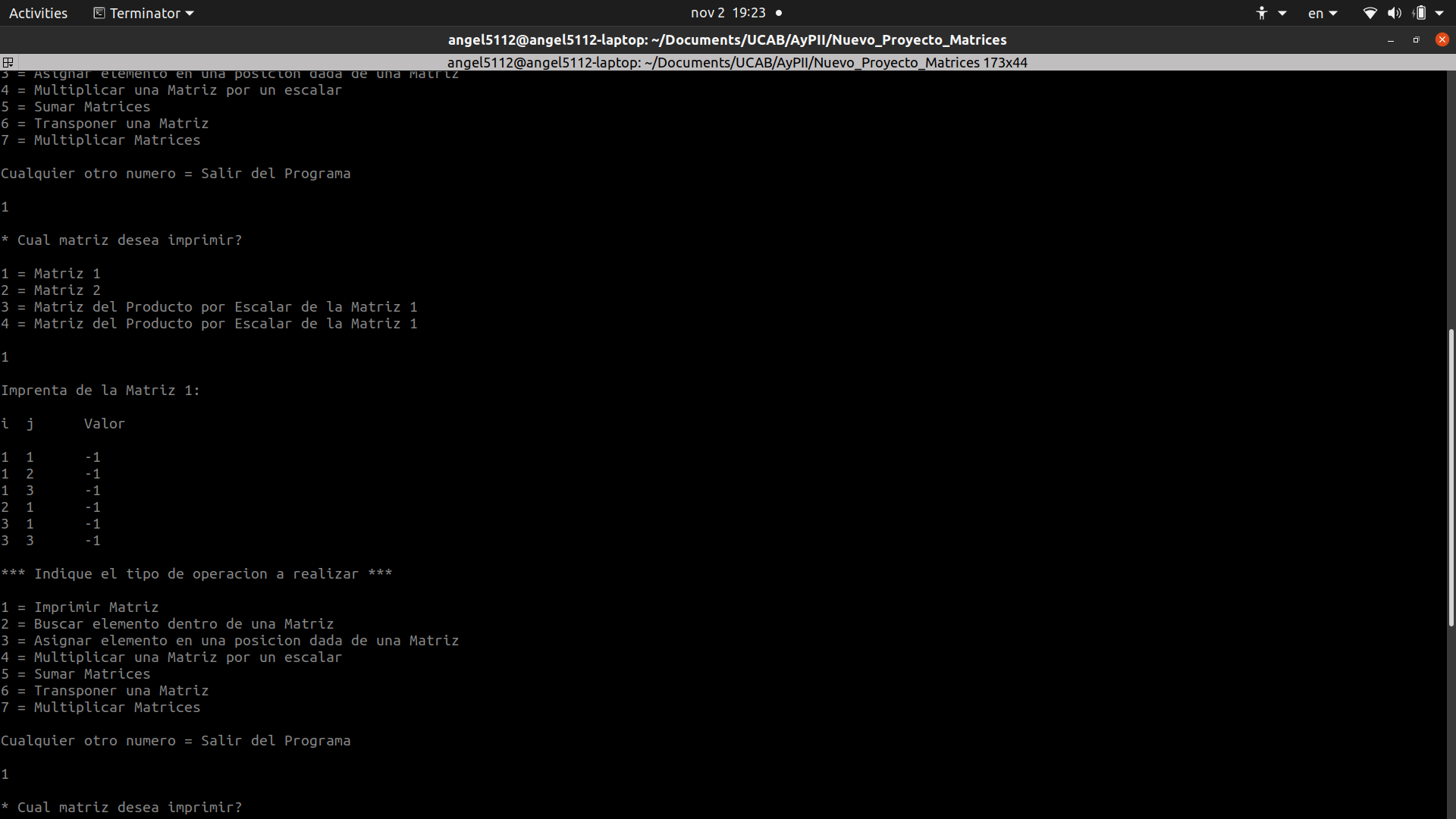
**nodef \*Multiplicar(nodef \*M1, nodef \*M2);**

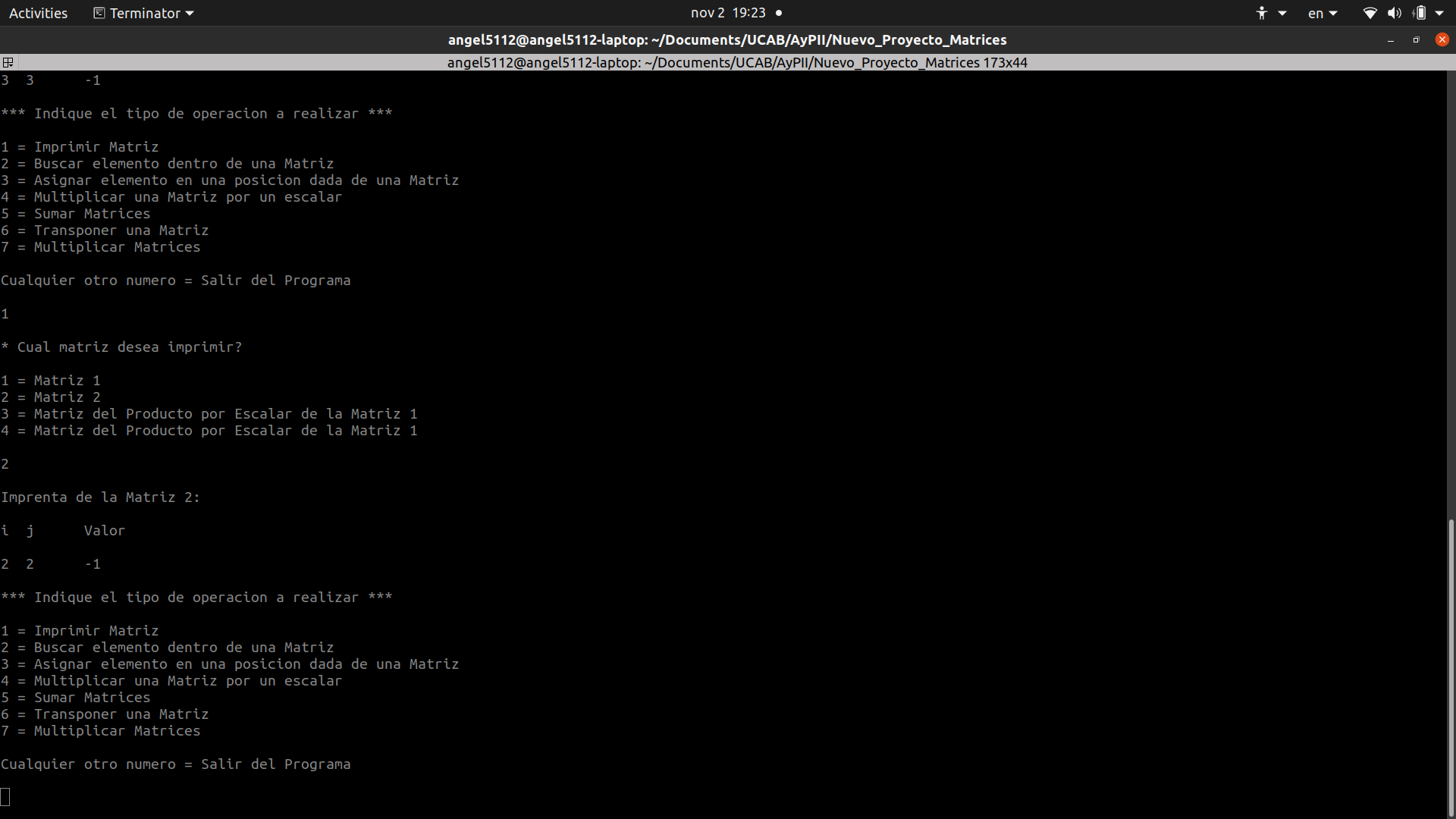
Funcion usada para realizar la multiplicacion entre 2 matrices. Es importante considerar que la cantidad de filas de la Matriz 1 debe ser igual al número de filas de la Matriz 2. Ademas la matriz resultante tiene el mismo numero de filas de la Matriz 1 y el mismo numero de columnas de la Matriz 2.

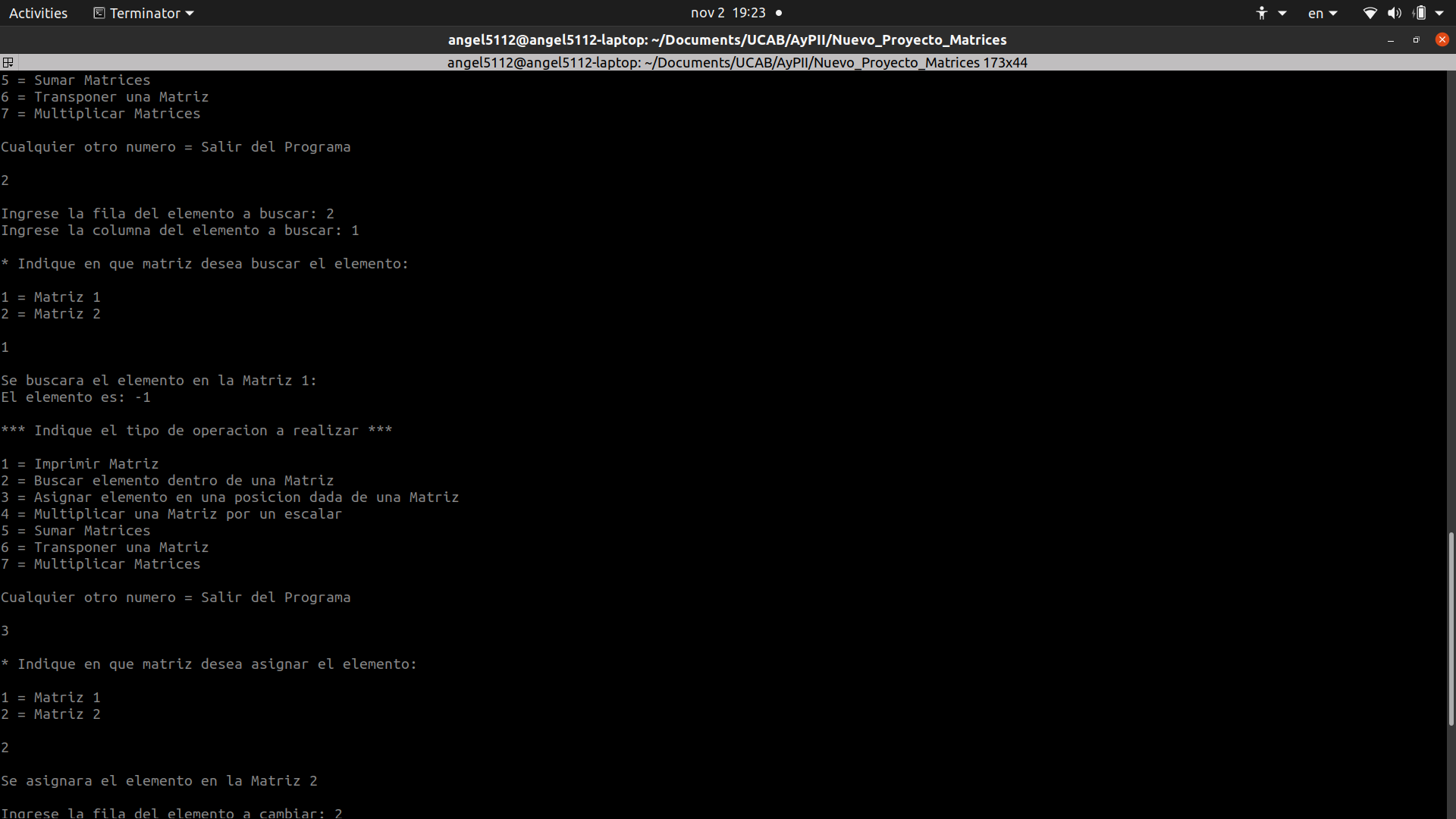
**Casos de Prueba:**

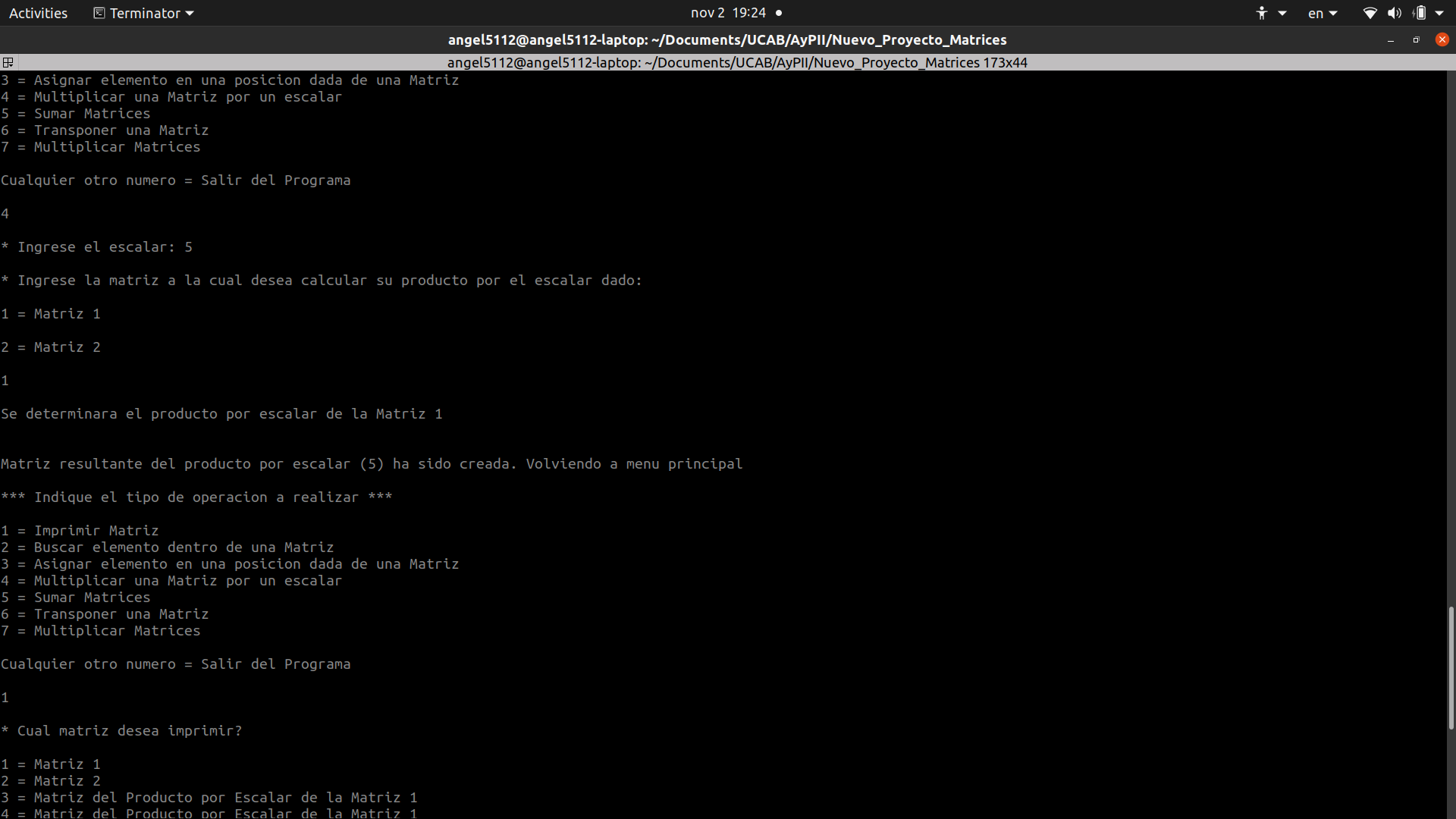
**Caso 1: Demostracion de Imprimir, Obtener Elemento, Producto por Escalar**

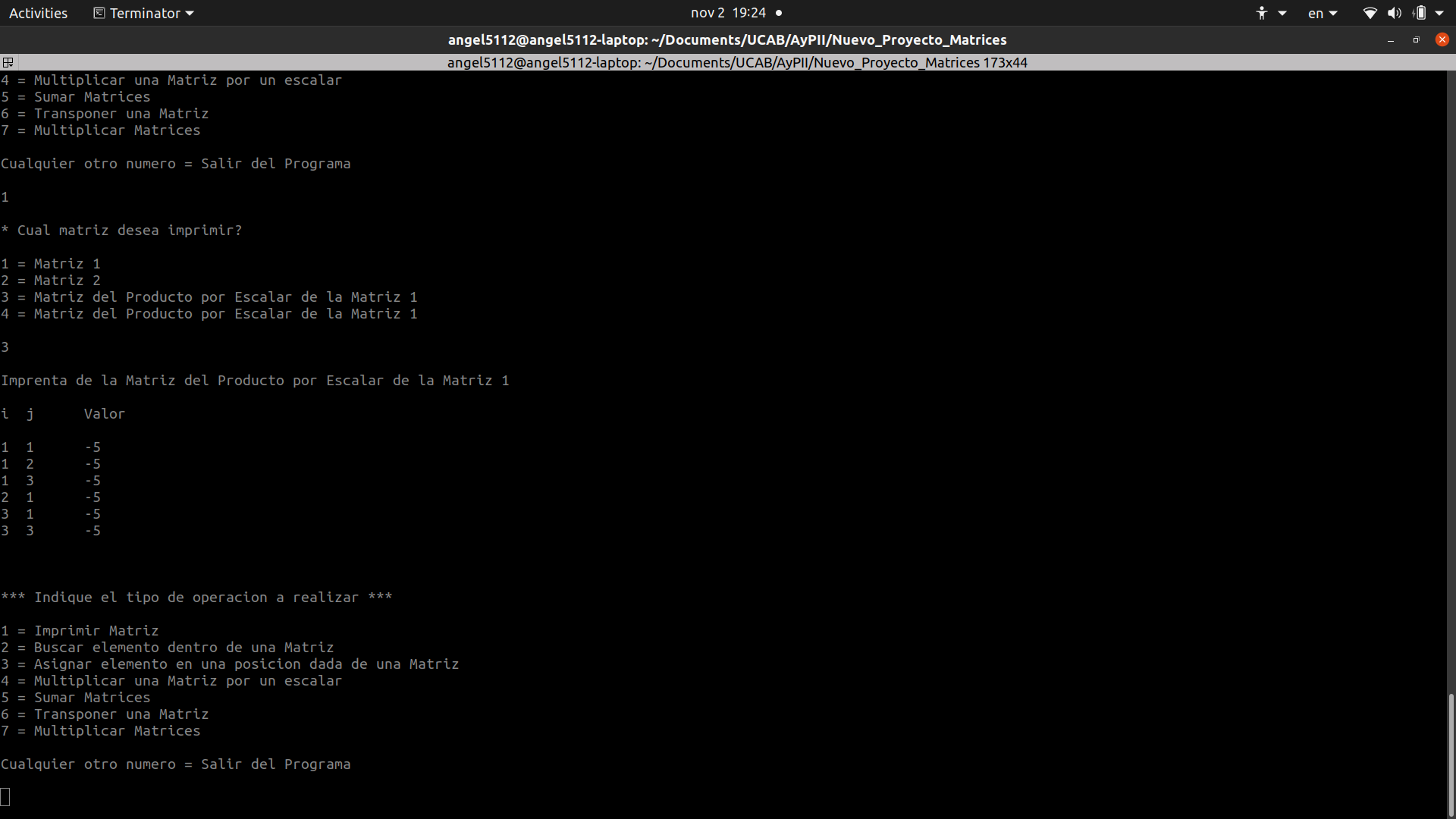
****

****

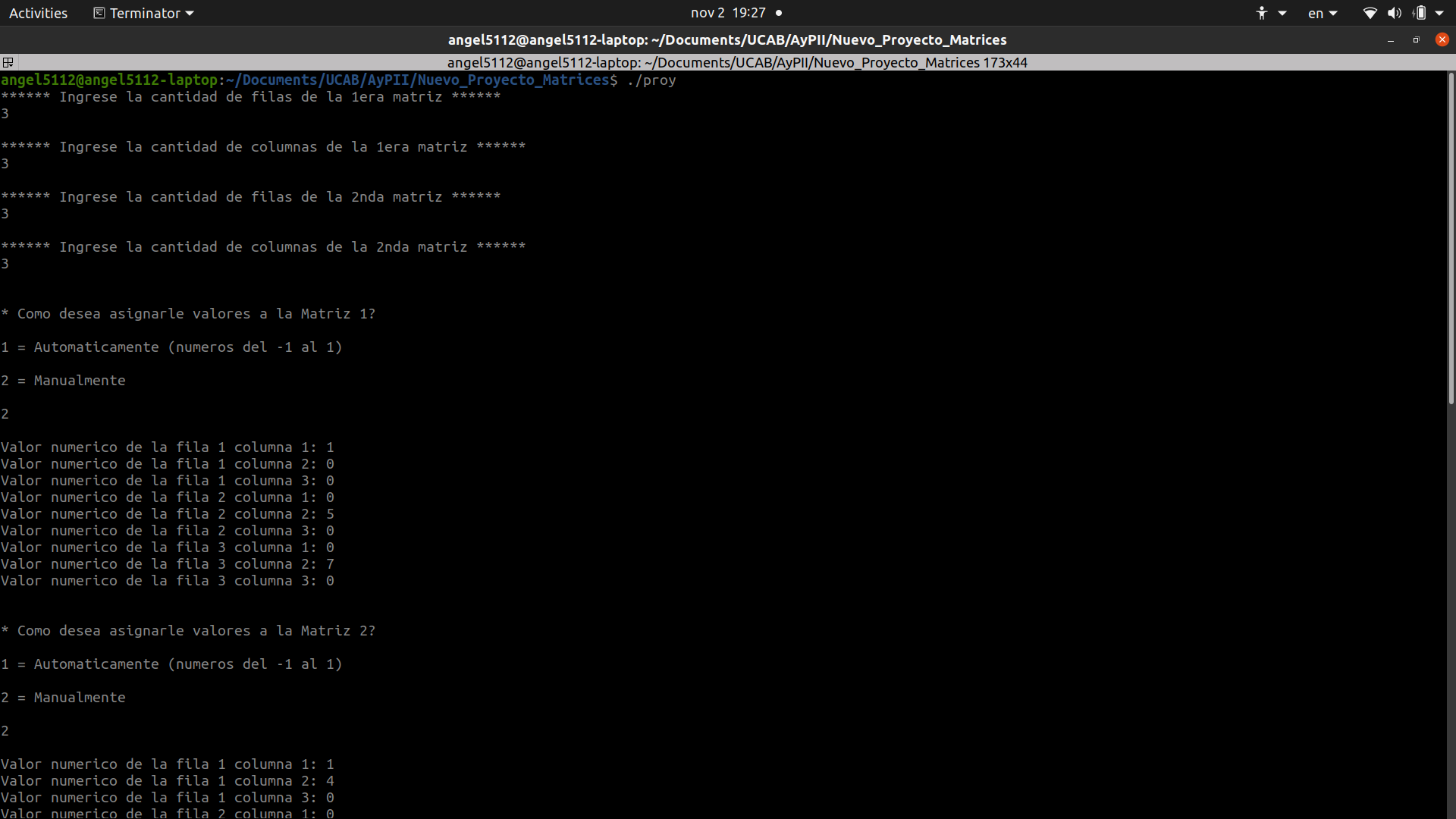
****

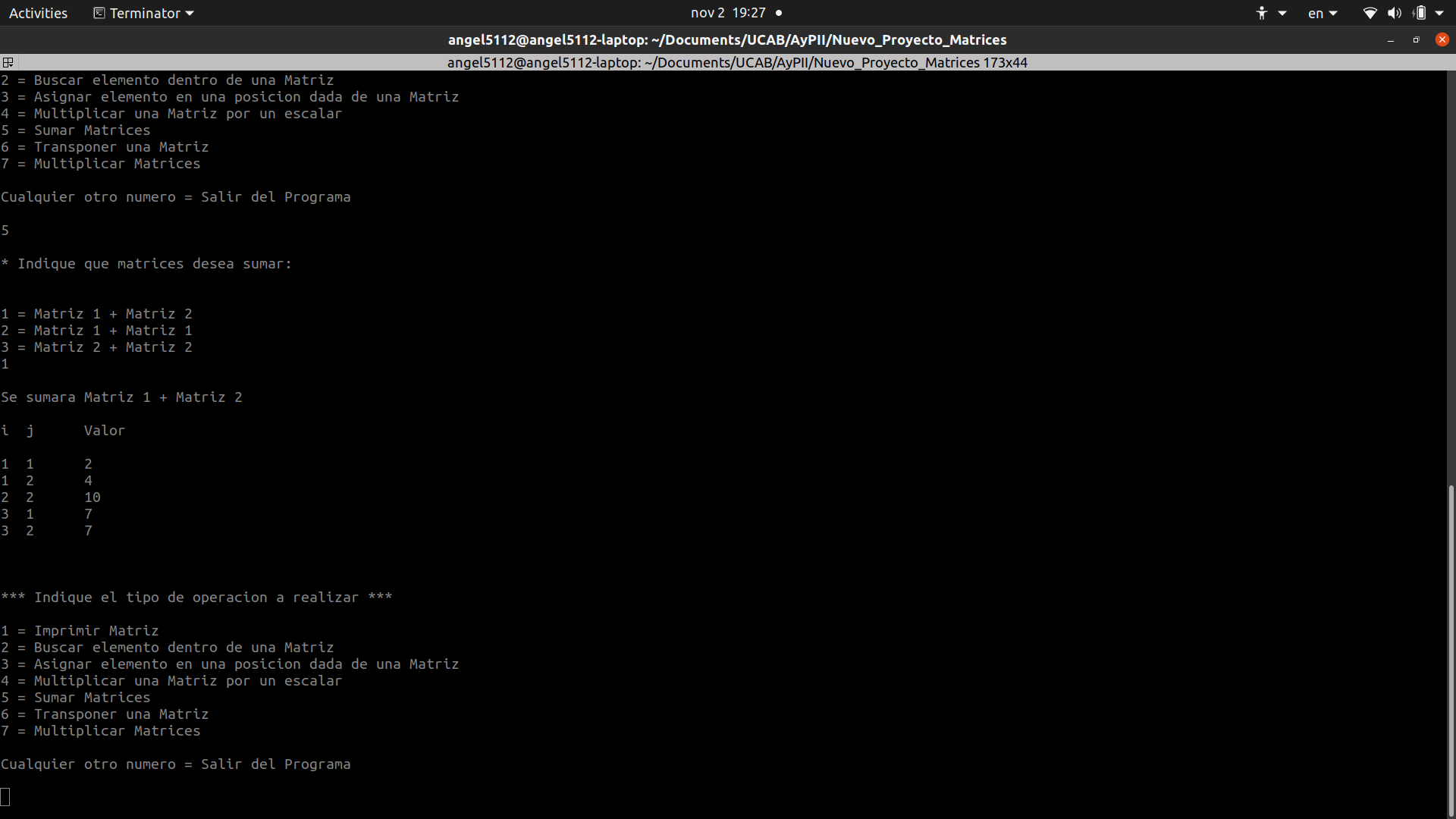
****

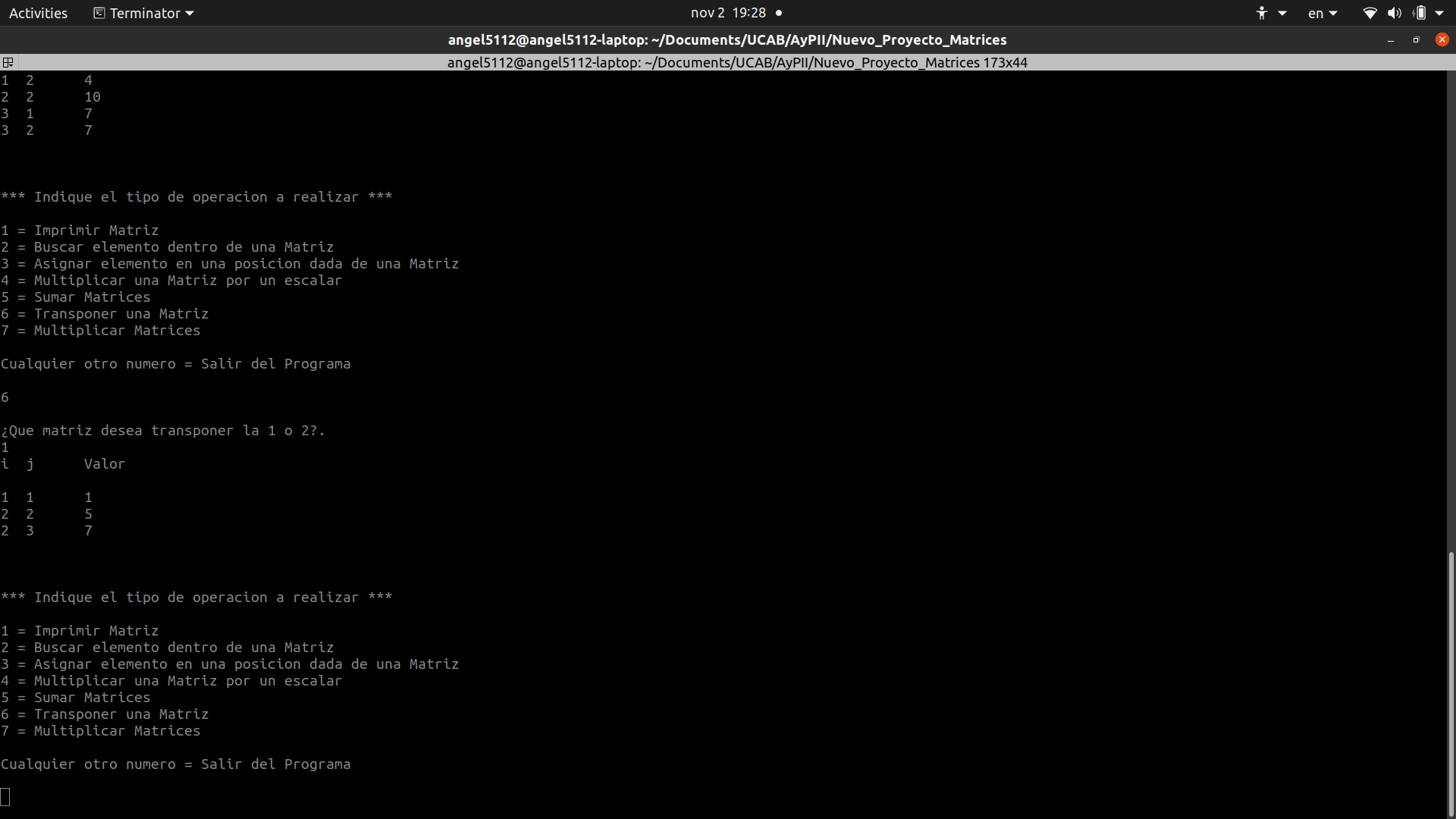
****

****

**Caso 2: Demostracion de Suma y Transpuesta**

****

****

****

**Codigo Fuente:**

***\* matriz.h \****

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Estructura de los nodos de la lista de columnas

typedef struct Nodecol

{

int valor;

int columna;

struct Nodecol \*next;

}nodecol;

// Estructura de los nodos de la lista de filas

typedef struct Nodef

{

int fila;

struct Nodef \*nextf;

struct Nodecol \*nextcol;

}nodef;

// Funcion para crear los nodos de las filas

nodef \*new\_nodef(nodef \*m, int f);

// Funcion para crear los nodos en la lista de columnas

nodecol \*new\_nodecol(nodecol \*l, int col, int v);

// Funcion para agregar un nodo al final (columnas)

nodecol\* add\_endj(nodecol\* link, nodecol\* l);

// Funcion para crear numeros aleatorios (Usar para asignar a la matriz)

int valor\_aleatorio();

// Funcion para crear una nueva matriz

nodef \*new\_matrix(nodef \*M, nodecol \*tj, int i, int j, int op);

// Funcion para imprimir una Matriz (1/7)

void Imprimir(nodef \*M);

// Funcion para buscar un elemento en una Matriz (2/7)

int ObtenerElemento(int i, int j, nodef \*M);

// Funcion para Asignar un elemento de una matriz en determinada posicion (3/7)

nodef \*AsignarElemento(int i, int j, int x, nodef \*M);

// Funcion para determinar la matriz resultante del producto por un escalar (4/7)

nodef \*ProductoPorEscalar(int e, nodef \*M);

// Funcion para determinar la matriz resultante de la suma de dos matrices dadas (5/7)

nodef \*Suma(nodef \*M1, nodef \*M2);

// Funcion para determinar la matriz transpuesta (6/7)

nodef \*Transponer(nodef \*M);

// Funcion para determinar el producto de dos matrices (7/7)

nodef \*Producto(nodef \*M1, nodef \*M2);

**/ \* matriz.c \*/**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "matriz.h"

int fila, columna, fila2, columna2;

// Funcion para crear los nodos de las filas

nodef \*new\_nodef(nodef \*m, int f)

{

nodef \*auxp;

auxp = (nodef\*)malloc(sizeof(nodef));

auxp->fila = f;

auxp->nextf = NULL;

auxp->nextcol = NULL;

return auxp;

}

// Funcion para crear los nodos en la lista de columnas

nodecol \*new\_nodecol(nodecol \*l, int col, int v)

{

nodecol\* auxp2;

auxp2 = (nodecol\*)malloc(sizeof(nodecol));

auxp2->columna = col;

auxp2->valor = v;

auxp2->next = NULL;

return auxp2;

}

// Funcion para agregar un nodo al final (columnas)

nodecol\* add\_endj(nodecol\* link, nodecol\* new\_link)

{

nodecol \*auxp;

if (link == NULL)

return new\_link;

for (auxp = link; auxp->next != NULL; auxp = auxp->next);

auxp->next = new\_link;

return link;

}

// Funcion para crear numeros aleatorios (Usar para asignar a la matriz)

int valor\_aleatorio()

{

int valor;

valor = rand() % 2 - 1;

return valor;

}

// Funcion para crear una nueva matriz

nodef \*new\_matrix(nodef \*M, nodecol \*tj, int i, int j, int op)

{

register int x, y;

int value;

nodef \*matp, \*matprev; // matp = Auxiliar a usar para recorrer la lista de filas

// Creacion y enlace de nodos del resto de la lista

M = new\_nodef(M, 1);

matp = matprev = M;

for (x = 1; x < i + 1; x++)

{

for (y = 1; y < j + 1; y++)

{

if (op == 1 || op < 1 || op > 2)

{

// Llamada a funcion para generar valores aleatorios (I = -1, 0, 1])

value = valor\_aleatorio();

}

else if (op == 2) // Operacion 2 = Asignacion manual

{

printf("Valor numerico de la fila %d columna %d: ", x, y);

scanf("%d", &value);

}

if (value != 0)

{

tj = new\_nodecol(tj, y, value);

matp->nextcol = add\_endj(matp->nextcol, tj);

}

}

if (matp->nextcol == NULL)

{

free(matp);

matp = matprev;

}

matp->nextf = new\_nodef(matp, x + 1);

matprev = matp;

matp = matp->nextf;

}

matp = M;

if (matp->nextcol == NULL)

{

M = NULL;

free(matp);

}

printf("\n");

return M;

}

// Funcion para imprimir una Matriz (1/7)

void Imprimir(nodef \*M)

{

nodef \*auxpf = M;

nodecol \*auxpc;

if (M == NULL)

{

printf("\nAdvertencia: La matriz es nula, por lo tanto no tiene imprenta. Volviendo al menu principal\n");

return;

}

printf("i j Valor\n\n");

while (auxpf != NULL)

{

auxpc = auxpf->nextcol;

while (auxpc != NULL)

{

printf("%d %d %d\n", auxpf->fila, auxpc->columna, auxpc->valor);

auxpc = auxpc->next;

}

auxpf = auxpf->nextf;

}

printf("\n");

}

// Funcion para buscar un elemento en una Matriz (2/7)

int ObtenerElemento(int i, int j, nodef \*M)

{

if (i < 0 || j < 0)

{

printf("\nError: fila o columna introducida es menor a 0. Elemento imposible de encontrar.\n");

return 0;

}

else if (M == NULL)

{

printf("\nAdvertencia: La matriz es nula, por lo tanto, el elemento a buscar es 0\n");

return 0;

}

else

{

nodef \*auxp = M;

nodecol \*tempj = NULL;

while (auxp->fila != i)

{

if (auxp->fila > i)

{

return 0;

}

auxp = auxp->nextf;

}

if (auxp->nextcol == NULL)

return 0;

tempj = auxp->nextcol;

while (tempj->columna != j)

{

if (tempj->next == NULL && tempj->columna < j)

{

return 0;

}

if (tempj->columna < j)

tempj = tempj->next;

else if (tempj->columna > j)

{

return 0;

}

}

return tempj->valor;

}

}

// Funcion para Asignar un elemento de una matriz en determinada posicion (3/7)

nodef \*AsignarElemento(int i, int j, int x, nodef \*M)

{

nodef \*auxfp = NULL;

nodef \*auxfp2 = NULL;

nodef \*auxfprev = NULL;

nodecol \*auxcp = NULL;

nodecol \*auxcp2 = NULL;

nodecol \*auxcprev = NULL;

if (i <= 0 || j <= 0)

{

printf("\nError: Dimensiones no pueden ser menores o iguales a 0");

return M;

}

else if (M == NULL)

{

printf("\nAdvertencia: La matriz es nula, creando espacio para asignar el elemento\n\n");

M = new\_nodef(M, i);

auxfp = M;

auxcp = new\_nodecol(auxcp, j, x);

if (x != 0)

{

auxfp->nextcol = add\_endj(auxfp->nextcol, auxcp);

return M;

}

else

{

free(auxcp);

M = NULL;

return M;

}

}

else

{

auxfp = M;

while (auxfp->nextf != NULL && auxfp->fila != i) // Aqui hacer lo mismo, es auxfp->next != NULL segurisimo

{

auxfp = auxfp->nextf;

if (auxfp->fila > i)

{

printf("\nFila no encontrada (Valor intermedio), se creara espacio para asignarla\n");

auxfprev = M;

while (auxfprev->nextf != auxfp)

{

auxfprev = auxfprev->nextf;

}

auxfprev->nextf = NULL;

auxfp2 = new\_nodef(auxfp2, i);

auxcp2 = new\_nodecol(auxcp2, j, x);

auxfp2->nextcol = add\_endj(auxfp2->nextcol, auxcp2);

auxfprev->nextf = auxfp2;

auxfp2->nextf = auxfp;

return M;

}

if (auxfp->nextf == NULL && auxfp->fila < i)

{

printf("\nFila no encontrada, se creara espacio para asignarla\n");

auxfp2 = new\_nodef(auxfp2, i);

auxcp2 = new\_nodecol(auxcp2, j, x);

auxfp2->nextcol = add\_endj(auxfp2->nextcol, auxcp2);

auxfp->nextf = auxfp2;

return M;

}

}

auxcp = auxfp->nextcol;

while (auxcp->next != NULL && auxcp->columna != j)

{

auxcp = auxcp->next;

}

if (auxcp->columna < j)

{

printf("\nColumna no encontrada, se creara espacio para asignarla\n");

auxcp2 = new\_nodecol(auxcp2, j, x);

auxcp->next = auxcp2;

return M;

}

else if (auxcp->columna > j)

{

// Se necesita un prev en el anterior para crear un nodo intermedio y enlazarlos todos

printf("\nColumna no encontrada, se creara espacio para asignarla\n");

auxcprev = auxfp->nextcol;

while (auxcprev->next != auxcp)

{

auxcprev = auxcprev->next;

}

auxcprev->next = NULL;

auxcprev->next= new\_nodecol(auxcprev->next, j, x);

auxcprev = auxcprev->next;

auxcprev->next = auxcp;

return M;

}

}

return NULL;

}

// Funcion para determinar la matriz resultante del producto por un escalar (4/7)

nodef \*ProductoPorEscalar(int e, nodef \*M)

{

nodef \*Me = NULL;

if (e == 0) // Condicion de e = 0, resultado sera una matriz nula

{

return Me;

}

else

{

Me = new\_nodef(Me, M->fila); // Crear el primer nodo con la fila del 1er nodo de la Matriz original

nodef \*auxpM, \*auxpMe;

nodecol \*auxpcolM, \*auxpcolMe;

auxpM = M;

auxpMe = Me;

while (auxpM != NULL) // Recorriendo las filas de la matriz original

{

auxpcolM = auxpM->nextcol;

while (auxpcolM != NULL) // Recorriendo las columnas de la matriz original

{

auxpcolMe = new\_nodecol(auxpcolMe, auxpcolM->columna, (auxpcolM->valor \* e)); // Asigna los mismos valores de la matriz original en la matriz nueva, pero ya con el producto por escalar aplicado

auxpMe->nextcol = add\_endj(auxpMe->nextcol, auxpcolMe);

auxpcolM = auxpcolM->next;

}

auxpM = auxpM->nextf; // Cambiando a la siguiente fila de la matriz original

if (auxpM == NULL) // Cortafuegos para evitar violacion de segmento

break;

else

{

auxpMe->nextf = new\_nodef(auxpMe, auxpM->fila); // Copiando el siguiente nodo de fila en la Matriz escalar

auxpMe = auxpMe->nextf;

}

}

return Me;

}

}

// Funcion para determinar la matriz resultante de la suma de dos matrices dadas (5/7)

nodef \*Suma(nodef \*M1, nodef \*M2) {

nodef \*M3; // Matriz resultante

nodef \*auxpM1, \*auxpM2, \*auxpM3, \*auxpPrevM3; // Punteros Aux.

nodecol \*newCol = NULL;

auxpM1 = M1;

auxpM2 = M2;

int m = 0, n = 0, res = 0; // Valores a sumar

M3 = new\_nodef(M3, 1); // Agregar Nodo Fila inicial a Matriz resultante

auxpM3 = auxpPrevM3 = M3;

for (int i=1; i<=fila; i++) {

for (int j=1; j<=columna; j++) {

m = ObtenerElemento(i, j, auxpM1);

n = ObtenerElemento(i, j, auxpM2);

res = m + n;

if (res != 0) {

newCol = new\_nodecol(newCol, j, res);

auxpM3->nextcol = add\_endj(auxpM3->nextcol, newCol);

}

}

if (auxpM3->nextcol == NULL) { // Liberar espacio de fila de solo 0 ceros

free(auxpM3);

auxpM3 = auxpPrevM3;

}

auxpM3->nextf = new\_nodef(auxpM3, i + 1); // Agregar Siguiente fila a Matriz resultante

auxpPrevM3 = auxpM3;

auxpM3 = auxpM3->nextf;

}

return M3;

}

// Funcion para determinar la matriz transpuesta (6/7)

nodef \*Transponer(nodef \*M) {

int value, auxFila, auxColumna;

nodef \*MR; // Matriz resultante

nodef \*auxpM, \*auxpMR, \*auxpPrevMR; // Punteros Aux.

nodecol \*newCol = NULL;

auxpM = M;

MR = new\_nodef(MR, 1); // Agregar Nodo Fila inicial a Matriz resultante

auxpMR = auxpPrevMR = MR;

auxFila = fila;

auxColumna = columna;

if (fila != columna) {

value = auxColumna;

auxColumna = auxFila;

auxFila = value;

value = 0;

}

for (int i=1; i<=auxFila; i++) {

for (int j=1; j<=auxColumna; j++) {

if (fila > columna && i > columna)

value = ObtenerElemento(i, j, auxpM);

else

value = ObtenerElemento(j, i, auxpM);

if (value != 0) {

newCol = new\_nodecol(newCol, j, value);

auxpMR->nextcol = add\_endj(auxpMR->nextcol, newCol);

}

}

if (auxpMR->nextcol == NULL) { // Liberar espacio de fila de solo 0 ceros

free(auxpMR);

auxpMR = auxpPrevMR;

}

auxpMR->nextf = new\_nodef(auxpMR, i + 1); // Agregar Siguiente fila a Matriz resultante

auxpPrevMR = auxpMR;

auxpMR = auxpMR->nextf;

}

return MR;

}

// Funcion para determinar el producto de dos matrices (7/7)

nodef \*Producto(nodef \*M1, nodef \*M2) {

//

}

**/\* proy1.c \*/**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "matriz.h"

// Declaracion de variables globales a utilizar (Dimensiones de las matrices)

int fila, columna, fila2, columna2;

int main()

{

// Declaracion de variables a utilizar

int f, col, v, escalar, validacion, operacion;

nodef \*matriz\_escalar = NULL;

nodef \*matriz\_resultante;

nodef \*matriz\_suma = NULL;

validacion = 0; // Variable a usar como key del ciclo del menu de procedimientos y operaciones

// Solicitud de las dimensiones de la matriz

printf("\*\*\*\*\*\* Ingrese la cantidad de filas de la 1era matriz \*\*\*\*\*\*\n");

scanf("%d", &fila);

printf("\n");

printf("\*\*\*\*\*\* Ingrese la cantidad de columnas de la 1era matriz \*\*\*\*\*\*\n");

scanf("%d", &columna);

printf("\n");

printf("\*\*\*\*\*\* Ingrese la cantidad de filas de la 2nda matriz \*\*\*\*\*\*\n");

scanf("%d", &fila2);

printf("\n");

printf("\*\*\*\*\*\* Ingrese la cantidad de columnas de la 2nda matriz \*\*\*\*\*\*\n");

scanf("%d", &columna2);

printf("\n");

if (fila <= 0 || columna <= 0 || fila2 <= 0 || columna2 <= 0)

printf("Error: Filas o columnas no pueden ser menor o iguales a 0\n");

else

{

srand(time(NULL));

// Creacion de Matriz 1

printf("\n\* Como desea asignarle valores a la Matriz 1?\n");

printf("\n1 = Automaticamente (numeros del -1 al 1)\n");

printf("\n2 = Manualmente\n\n");

scanf("%d", &operacion); // Si operacion es un valor diferente de 1 o 2, se asignaran valores automaticos como prevencion

printf("\n");

nodef \*matriz1 = NULL;

nodecol \*tempcol = NULL;

// Llamada a funcion para la creacion de la Matriz

matriz1 = new\_matrix(matriz1, tempcol, fila, columna, operacion);

operacion = 0;

// Creacion de Matriz 2

printf("\n\* Como desea asignarle valores a la Matriz 2?\n");

printf("\n1 = Automaticamente (numeros del -1 al 1)\n");

printf("\n2 = Manualmente\n\n");

scanf("%d", &operacion); // Si operacion es un valor diferente de 1 o 2, se asignaran valores automaticos como prevencion

printf("\n");

nodef \*matriz2 = NULL;

nodecol \*tempcol2 = NULL;

// Llamada a funcion para la creacion de la Matriz 2

matriz2 = new\_matrix(matriz2, tempcol2, fila2, columna2, operacion);

operacion = 0;

while (!validacion)

{

// Antes de esto se deben crear las matrices

// Menu de operaciones a utilizar en el programa

operacion = 0;

f = 0;

col = 0;

v = 0;

escalar = 0;

printf("\*\*\* Indique el tipo de operacion a realizar \*\*\* \n");

printf("\n1 = Imprimir Matriz\n");

printf("2 = Buscar elemento dentro de una Matriz\n");

printf("3 = Asignar elemento en una posicion dada de una Matriz\n");

printf("4 = Multiplicar una Matriz por un escalar\n");

printf("5 = Sumar Matrices\n");

printf("6 = Transponer una Matriz\n");

printf("7 = Multiplicar Matrices\n");

printf("\nCualquier otro numero = Salir del Programa\n\n");

scanf("%d", &operacion);

// Condiciones del menu

if (operacion == 1) // Imprimir una Matriz (1/7)

{

operacion = 0;

printf("\n\* Cual matriz desea imprimir?\n");

printf("\n1 = Matriz 1\n");

printf("2 = Matriz 2\n");

printf("3 = Matriz del Producto por Escalar de la Matriz 1\n");

printf("4 = Matriz del Producto por Escalar de la Matriz 1\n\n");

scanf("%d", &operacion);

// Condiciones de operacion en la seccion de Imprimir

if (operacion == 1)

{

printf("\nImprenta de la Matriz 1: \n\n");

Imprimir(matriz1);

}

else if (operacion == 2)

{

printf("\nImprenta de la Matriz 2: \n\n");

Imprimir(matriz2);

}

else if (operacion == 3)

{

printf("\nImprenta de la Matriz del Producto por Escalar de la Matriz 1\n\n");

if (matriz\_escalar == NULL)

{

printf("\nMatriz por escalar aun no ha sido creada o es nula, creando...\n\n");

printf("\n\* Ingrese el escalar: ");

scanf("%d", &escalar);

matriz\_escalar = ProductoPorEscalar(escalar, matriz1);

Imprimir(matriz\_escalar);

printf("\n\n");

}

else

{

Imprimir(matriz\_escalar);

printf("\n\n");

}

}

else if (operacion == 4)

{

printf("\nImprenta de la Matriz del Producto por Escalar de la Matriz 2\n\n");

if (matriz\_escalar == NULL)

{

printf("\nMatriz por escalar aun no ha sido creada o es nula, creando...\n\n");

printf("\n\* Ingrese el escalar: ");

scanf("%d", &escalar);

matriz\_escalar = ProductoPorEscalar(escalar, matriz2);

Imprimir(matriz\_escalar);

printf("\n\n");

}

else

{

Imprimir(matriz\_escalar);

printf("\n\n");

}

}

else

printf("\nError: Numero ingresado no corresponde a accion alguna, volviendo al menu principal\n");

}

else if (operacion == 2) // Busqueda de elemento (2/7)

{

operacion = 0;

printf("\nIngrese la fila del elemento a buscar: ");

scanf("%d", &f);

printf("Ingrese la columna del elemento a buscar: ");

scanf("%d", &col);

printf("\n\* Indique en que matriz desea buscar el elemento: \n");

printf("\n1 = Matriz 1\n");

printf("2 = Matriz 2\n\n");

scanf("%d", &operacion);

if (operacion == 1)

{

if (f > fila || col > columna)

printf("Error: Fila o Columna no puede ser mayor a la dimension original de la matriz. Volviendo a menu principal.\n");

else

{

printf("\nSe buscara el elemento en la Matriz 1:\n");

printf("El elemento es: %d\n\n", ObtenerElemento(f, col, matriz1));

}

}

else if (operacion == 2)

{

if (f > fila2 || col > columna2)

printf("Error: Fila o Columna no puede ser mayor a la dimension original de la matriz. Volviendo a menu principal.\n");

else

{

printf("\nSe buscara el elemento en la Matriz 2:\n");

printf("El elemento es: %d\n\n", ObtenerElemento(f, col, matriz2));

}

}

else

printf("\nError: Numero ingresado no corresponde a accion alguna. Volviendo a menu principal\n");

}

else if (operacion == 3) // Asignar Elemento (3/7)

{

printf("\n\* Indique en que matriz desea asignar el elemento: \n");

printf("\n1 = Matriz 1\n");

printf("2 = Matriz 2\n\n");

scanf("%d", &operacion);

if (operacion == 1)

{

if (f > fila || col > columna)

printf("Error: Fila o Columna no puede ser mayor a la dimension original de la matriz. Volviendo a menu principal.\n");

else

{

printf("\nSe asignara el elemento en la Matriz 1\n");

printf("\nIngrese la fila del elemento a cambiar: ");

scanf("%d", &f);

printf("Ingrese la columna del elemento a cambiar: ");

scanf("%d", &col);

printf("\nIngrese el valor a asignar en la posicion indicada anteriormente: ");

scanf("%d", &v);

matriz1 = AsignarElemento(f, col, v, matriz1);

printf("\nVolviendo al menu principal\n\n");

}

}

else if (operacion == 2)

{

if (f > fila2 || col > columna2)

printf("Error: Fila o Columna no puede ser mayor a la dimension original de la matriz. Volviendo a menu principal.\n");

else

{

printf("\nSe asignara el elemento en la Matriz 2\n");

printf("\nIngrese la fila del elemento a cambiar: ");

scanf("%d", &f);

printf("Ingrese la columna del elemento a cambiar: ");

scanf("%d", &col);

printf("\nIngrese el valor a asignar en la posicion indicada anteriormente: ");

scanf("%d", &v);

matriz2 = AsignarElemento(f, col, v, matriz2);

printf("\nVolviendo al menu principal\n\n");

}

}

else

printf("\nError: Numero ingresado no corresponde a accion alguna. Volviendo a menu principal\n");

}

else if (operacion == 4) // Producto por Escalar (4/7)

{

printf("\n\* Ingrese el escalar: ");

scanf("%d", &escalar);

printf("\n\* Ingrese la matriz a la cual desea calcular su producto por el escalar dado: \n");

printf("\n1 = Matriz 1\n");

printf("\n2 = Matriz 2\n\n");

scanf("%d", &operacion);

if (operacion == 1)

{

printf("\nSe determinara el producto por escalar de la Matriz 1\n\n");

matriz\_escalar = ProductoPorEscalar(escalar, matriz1);

printf("\nMatriz resultante del producto por escalar (%d) ha sido creada. Volviendo a menu principal\n\n", escalar);

}

else if (operacion == 2)

{

printf("\nSe determinara el producto por escalar de la Matriz 2\n\n");

matriz\_escalar = ProductoPorEscalar(escalar, matriz2);

printf("\nMatriz resultante del producto por escalar (%d) ha sido creada. Volviendo a menu principal\n\n", escalar);

}

else

printf("\nError: Numero ingresado no corresponde a accion alguna. Volviendo a menu principal\n");

}

else if (operacion == 5) // Sumar (5/7)

{

matriz\_resultante = NULL;

if (fila != fila2 || columna != columna2)

printf("\nError: Las matrices deben tener las mismas dimensiones.\n");

else

{

printf("\n\* Indique que matrices desea sumar: \n\n");

printf("\n1 = Matriz 1 + Matriz 2\n");

printf("2 = Matriz 1 + Matriz 1\n");

printf("3 = Matriz 2 + Matriz 2\n");

scanf("%d", &operacion);

if (operacion == 1)

{

printf("\nSe sumara Matriz 1 + Matriz 2\n\n");

matriz\_resultante = Suma(matriz1, matriz2);

Imprimir(matriz\_resultante);

printf("\n\n");

}

else if (operacion == 2)

{

printf("\nSe sumara Matriz 1 + Matriz 1\n\n");

matriz\_resultante = Suma(matriz1, matriz1);

Imprimir(matriz\_resultante);

printf("\n\n");

}

else if (operacion == 3)

{

printf("\nSe sumara Matriz 2 + Matriz 2\n\n");

matriz\_resultante = Suma(matriz2, matriz2);

Imprimir(matriz\_resultante);

printf("\n\n");

}

else

printf("\nError: Numero ingresado no corresponde a accion alguna. Volviendo a menu principal\n");

}

}

else if (operacion == 6) // Transponer (6/7)

{

matriz\_resultante = NULL;

operacion = 0;

printf("\n¿Que matriz desea transponer la 1 o 2?.\n");

scanf("%d", &operacion);

if (operacion == 1)

{

matriz\_resultante = Transponer(matriz1);

Imprimir(matriz\_resultante);

}

else if (operacion == 2)

{

matriz\_resultante = Transponer(matriz2);

Imprimir(matriz\_resultante);

}

else

printf("\nError: Numero ingresado no corresponde a accion alguna. Volviendo a menu principal\n");

printf("\n\n");

}

else if (operacion == 7) // Multiplicar (7/7)

{

//

}

else if (operacion < 1 || operacion > 7)

{

printf("\nHa decidido salir del programa.\n");

validacion = 1;

}

}

}

}

**Enlace al repositorio de GitHub del Proyecto:**

https://github.com/Angel5112/Proyecto\_Matriz.git