|  |
| --- |
| Programación Declarativa |
| Matrilog |
| Creación de una librería para manipular matrices en Prolog escrita en C |

|  |
| --- |
| Lucía Calzado  26-4-2020 |

# Índice

## Introducción

## Decisiones de diseño

## Librería SWI-Prolog

### Términos y tipos de llamadas utilizados

### Idea general de construcción

### Cómo compilar una librería

## Estructura de la librería Matrilog

## Conclusiones

## Anexo: ejecuciones

# Introducción

En este documento se muestra el desarrollo de Matrilog, una librería escrita en C para Prolog, con el objetivo de implementar en él algunas operaciones con matrices.

Se hablará de las decisiones que se han tenido que tomar para diseñar las matrices en C, de la librería que se ha usado, proporcionada por Prolog[1] y de la estructura de esta librería, que podría servir como guía para muchas otras.

Para conseguir esto, se ha hecho uso de las listas que brinda el lenguaje Prolog, las cuales se han pasado a matrices en C para poder realizar las operaciones requeridas (véase apartado 5. Funciones implementadas para echar un vistazo a las operaciones que se han codificado).

Las operaciones con matrices implementadas han sido: sumar todos los elementos de una matriz, multiplicar una matriz por un número entero, sumar dos matrices y multiplicar dos matrices.

Todas las capturas de pantalla de este documento, a excepción de las de los anexos, son del código fuente, que se puede encontrar junto a este documento y en el repositorio de GitHub <https://github.com/Geadalu/DeclarativeProg/tree/master/TrabajoC>.

# Decisiones de diseño

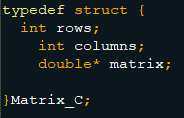
En este apartado se enumerarán las distintas decisiones que se han tomado para el diseño de la librería.

## Vectores unidimensionales

En lugar de usar una matriz bidimensional en C, declarada como matriz[][], se ha decidido usar una cadena (de aquí en adelante, vector) unidimensional. Gracias al control de este vector que podremos ver más adelante en el código fuente de la librería, a pesar de que dentro del programa se trata la lista de listas de Prolog como un vector, esto para el usuario es completamente transparente y simple de utilizar.

## Encapsulamiento del vector en un struct

También se ha decidido encapsular los atributos del vector en una estructura de C struct a la que se ha llamado Matrix\_C.



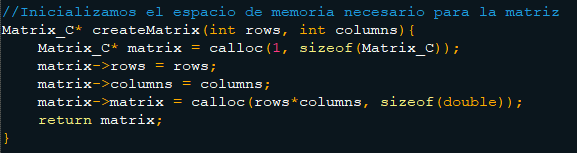
Esto permite acceder a ellos en cualquier parte del programa, y se evita tener que pasar constantemente por argumentos las filas y las columnas, así como la propia matriz.

## Operaciones en float

Una manera rápida y eficaz de evitar conflictos entre tipos de datos cuando el usuario meta enteros y números de punto flotante (de ahora en adelante, *float*) en una misma matriz es convertir todos los números al mismo tipo, y hacer así las operaciones. Debido a que, si al menos un número de la matriz fuera float, el resultado debiera ser también float, se ha decidido pasar todos los números a tipo float y hacer así las operaciones, independientemente de si es una matriz de enteros, de puntos flotantes o mixta.

## Restricciones

Debido a la decisión explicada en el punto a, al usuario se le impone la restricción de meter en los predicados el número de filas y de columnas que tienen las matrices bidimensionales que está introduciendo. Esto es necesario, en mayor parte, para la necesaria y correcta reserva de memoria en C que necesita cualquier vector o matriz para inicializarse.



# Librería SWI-Prolog

Para realizar este proyecto se ha hecho uso de la librería SWI-Prolog para C[1]. Para introducirla en la librería, se debe añadir como cabecera: #include <SWI-Prolog.h>.

Esta librería contiene herramientas para escribir predicados de Prolog en C, convertir términos de Prolog a tipos de datos de C y unificar tipos de datos de C a términos de Prolog, entre muchas otras operaciones. Aquí se describirán las que se han usado para escribir Matrilog.

## Términos y tipos de llamadas utilizados

### term\_t

Este tipo de variable representa cualquier tipo de término en Prolog. Puede ser un entero, un float, una lista… Si se necesita operar directamente con estos términos o hacer cualquier tipo de referencia a ellos, se deben usar las funciones PL\_new\_term\_ref() y PL\_copy\_term\_ref(+term). El primero crea una nueva referencia para un término nuevo, y el segundo copia una ya existente. Podemos ver un claro ejemplo del uso de estas llamadas en el manejo de las listas en la librería, donde a partir de una lista (en la librería, un dato term\_t llamado list), construimos referencias a su cabecera y su cola:



Para, posteriormente, crear la lista con la función PL\_get\_list():



### foreign\_t

Este tipo de definición de función es el que permite escribir el predicado para Prolog. Los métodos de la librería que lleven este tipo, son los que posteriormente se registrarán como predicados en la función install(), que veremos más adelante.



### install\_t

Dentro de esta función, que en la biblioteca se encuentra al final, por cada predicado escrito se tendrá que definir, mediante PL\_register\_foreign(), el nombre del predicado, el número de argumentos y cómo se llama el predicado dentro de la librería. De esta forma se queda registrado y listo para usar en la consola de Prolog. Como ejemplo, en esta función, se registra un solo predicado llamado sum\_matrix, al cual se le tienen que pasar 4 argumentos y que corresponde con el predicado de la librería pl\_sum\_matrix:



### PL\_fail y PL\_succeed

Estas dos llamadas permiten la comunicación para que Prolog devuelva false, en el caso de PL\_fail y true, en el caso de PL\_succeed. Su utilidad es muy simple: controlar si el programa, en algún punto debería fallar, y en ese caso, la operación entera. De esta forma, por ejemplo, hemos controlado que el usuario no pueda meter caracteres en vez de números para operar las matrices.

### Funciones de comprobación de tipos

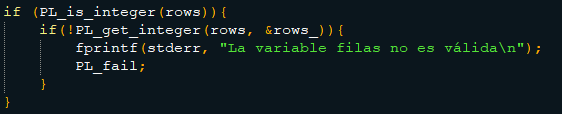
Hay varias funciones que proporciona la librería para comprobar el tipo de dato del término en cuestión. Las que se han usado han sido:

PL\_is\_number(+term\_t), que comprueba que un término es un número, pero sin comprobar el tipo del número.

PL\_is\_float(+term\_t), que comprueba que un término es un número de punto flotante.

PL\_is\_integer(+term\_t), que comprueba que un término es un número entero.

Todas estas funciones retornan un 0 en caso de fallo, y 1 en caso de terminar correctamente, por eso, en caso de estar dentro del método del predicado de Prolog, se debe controlar su salida y añadir PL\_fail en los que retornen un 0.



### Funciones de asignación de tipos de Prolog a C

Estas funciones pasan el término de Prolog elegido a un tipo de dato de C.

PL\_get\_float(+term\_t, -float\_number), que coge el término term\_t y lo intenta convertir a un float de C. El término float\_number tiene que ser una referencia a memoria.

PL\_get\_integer(+term\_t, -integer\_number), que coge el término term\_t y lo intenta convertir a un integer de C. El término integer\_number tiene que ser una referencia a memoria.

Hay una función especial que opera con términos en Prolog, pero no los pasa a C, y es esta:

PL\_get\_list(+term\_t\_tail, +term\_t\_head, -term\_t\_tail), que coge la lista y la divide en su cabeza y su cola.

### Colocación de datos en C a los términos de Prolog

De igual manera que se pueden pasar los términos de Prolog a datos en C con las funciones vistas en el apartado anterior, también se puede hacer lo contrario: pasar de datos en C a tipos term\_t de Prolog.

Esto es verdaderamente útil a la hora de la unificación del resultado final, por ejemplo.

PL\_put\_float(-term\_t\_r, +float\_number), donde float\_number es un número float de C, y term\_t\_r es un término de Prolog del tipo term\_t.

PL\_put\_integer(-term\_t\_r, +integer\_number), donde integer\_number es un número integer de C, y term\_t\_r es un término de Prolog del tipo term\_t.

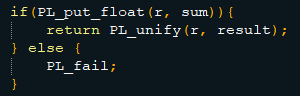
PL\_cons\_list(+term\_t\_List, +term\_t\_List2, -term\_t\_List), donde se añade la lista term\_t\_List2 a la lista term\_t\_List.

PL\_put\_nil(+term\_t\_List), que añade al término term\_t\_List la constante de terminación de una lista.

### Unificación

Para unificar el resultado y retornarlo en Prolog, usamos la función PL\_unify:

PL\_unify(-term\_t\_resultado, -term\_t\_variable), donde term\_t\_resultado es el resultado que da el programa, convertido a term\_t, y term\_t\_variable es la variable que se le pasa al predicado para obtener el resultado.



En esta imagen podemos ver cómo se intenta meter en el término r el float sum, y unificar este término r con result, que es la variable que se le pasa por argumentos al predicado.

## Idea general de construcción

Para construir una librería en C para Prolog, generalmente, se han de seguir tres pasos:

* Pasar de términos de Prolog a estructuras y datos en C

Todo lo que el usuario introduce mediante el predicado de Prolog, la librería necesita interpretarlo. Para ello, hacemos uso de las funciones anteriormente mencionadas para pasar tanto términos numéricos a integers o float, como listas a matrices.

* Operar con esos datos y estructuras directamente en lenguaje C

Cuando se tengan todos los datos en lenguaje C, se opera con ellos en cada predicado. Dependiendo del tipo o uso de ese predicado, se harán unas operaciones u otras, controlando siempre el flujo del programa con PL\_succeed o PL\_fail.

* Pasar de estructuras y datos en C a términos de Prolog

Finalmente, para poder devolver la respuesta del predicado (notada en todos los predicados de la librería como *result*), se necesita pasar el resultado a término en Prolog y unificarlo con el término del resultado.

## Cómo compilar una librería

Para compilar el archivo .c, usaremos este comando:

swipl-ld -c nombre\_archivo.c

Para convertir el archivo resultante en librería de Prolog, lanzaremos este comando:

swipl-ld -shared -o nombre\_libreria.so nombre\_archivo.o

Finalmente, dentro de la consola de Prolog solo tendríamos que cargar la librería:

load\_foreign\_library(nombre\_libreria).

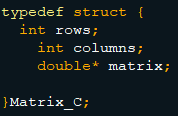
Y ya se podrían usar los predicados según se han escrito en la función install(). Se pueden ver ejecuciones de estos comandos en el anexo, ejecutados mediante un Makefile.

# Contenido de la librería

En este apartado se hablará de la estructura que ha seguido la librería: las funciones que tiene implementadas, tanto esenciales como de apoyo, y los predicados escritos para Prolog instalados.

## Estructura creada para la matriz

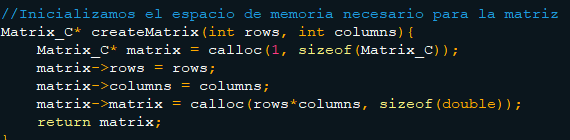
La estructura creada para la matriz es lo primero que se lee en la librería.



De esta estructura se ha hablado en el apartado de Decisiones de diseño.

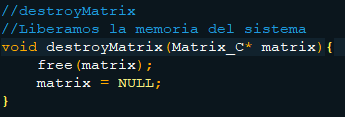
## Reserva de memoria para la matriz

En cualquier parte de la librería donde es necesario crear una matriz, se llama a este método, puesto que en C se necesita reservar memoria para las matrices (al no saber el compilador, en el momento de la creación, cuánta memoria tiene que reservar para ella).



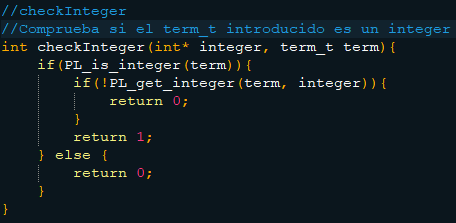
## Liberación de la memoria

La función destroyMatrix libera la memoria utilizada por la matriz. Se llama al final de cada predicado para liberar la memoria de todas las matrices que se han usado.



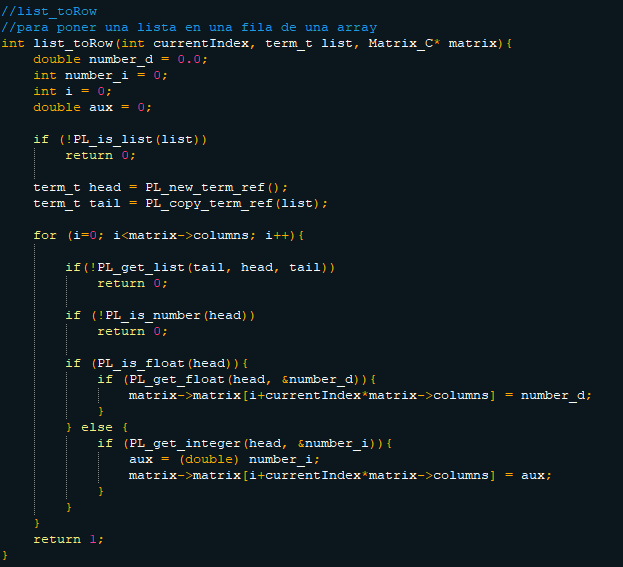
## Función checkInteger

Esta función comprueba si el term\_t del parámetro es un integer, en cuyo caso lo asigna al int integer. He decidido construirla para ahorrar líneas, puesto que hay que llamarla cada vez que se quieren recoger el número de filas y columnas de cada matriz.



## Función list\_toRow

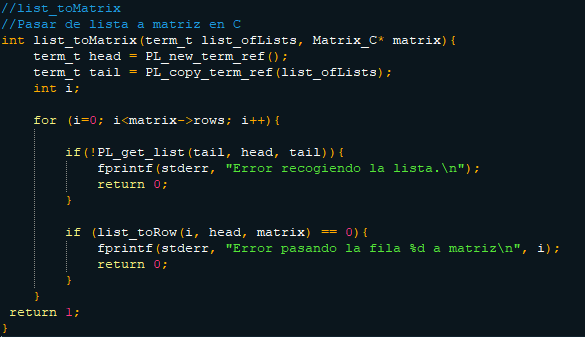
Esta función es muy importante, pues es la que pasa una fila de la lista (esto es, una de las listas dentro de la lista) a un vector de C, para posterior uso en la función list\_toMatrix(), que se explicará más adelante.



Retorna 0 si ha fallado, y 1 si ha tenido éxito.

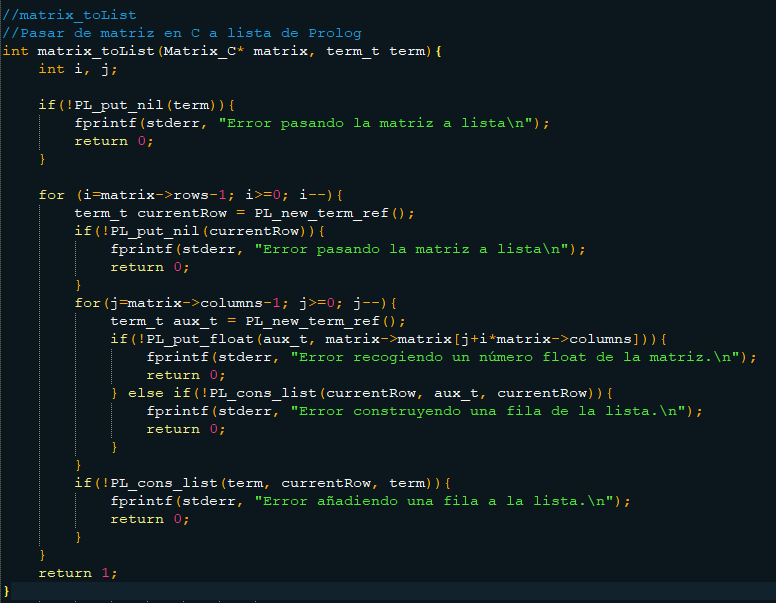
## Función list\_toMatrix

Esta función rellena un vector llamando varias veces a list\_toRow(). Se llama en cada predicado de Prolog, para pasar la lista que introduce el usuario a vector en C.



## Función matrix\_toList

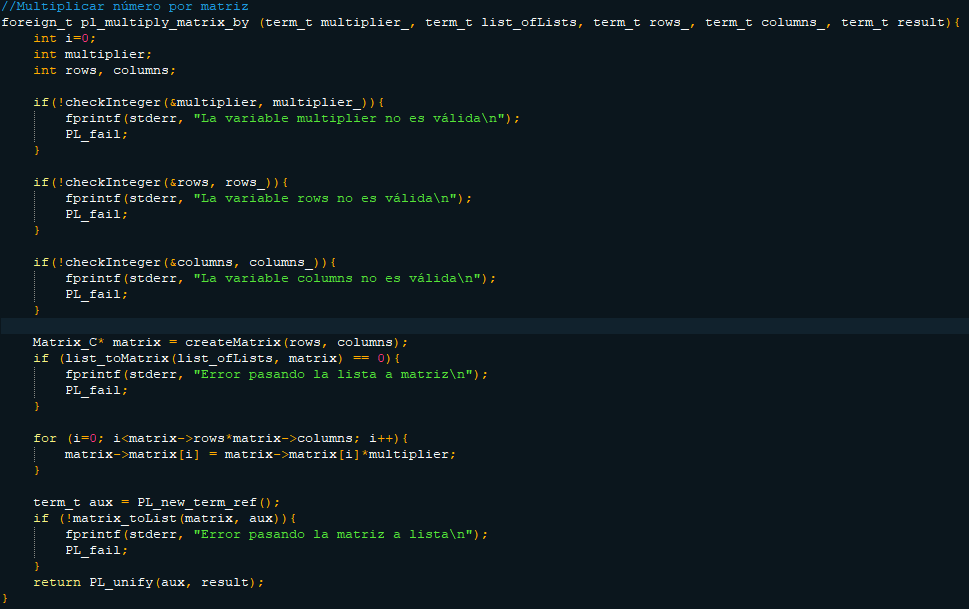
Esta función es la opuesta a la anterior. Se llama cuando se necesita pasar una matriz resultado a lista, para unificarla y devolverla en el predicado.



## Predicado multiply\_matrix\_by

Este predicado multiplica una matriz por un número dado. Los argumentos son:

* term\_t multiplier\_: el número por el que se quiere multiplicar la matriz
* term\_t list\_ofLists: la matriz
* term\_t rows\_: las filas que tiene la matriz
* term\_t columns: las columnas que tiene la matriz
* term\_t result: el resultado que se devolverá

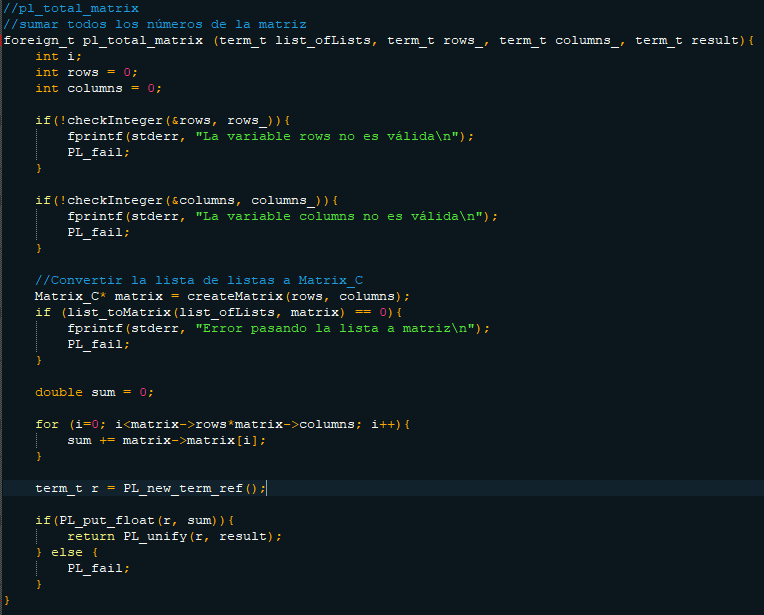


Se llama desde Prolog mediante mult\_matrix\_by.

## Predicado pl\_total\_matrix

Este predicado suma todos los números de la matriz. Los argumentos son:

* term\_t list\_ofLists: la matriz
* term\_t rows\_: las filas de la matriz
* term\_t columns\_: las columnas de la matriz
* term\_t result: el resultado que se devolverá

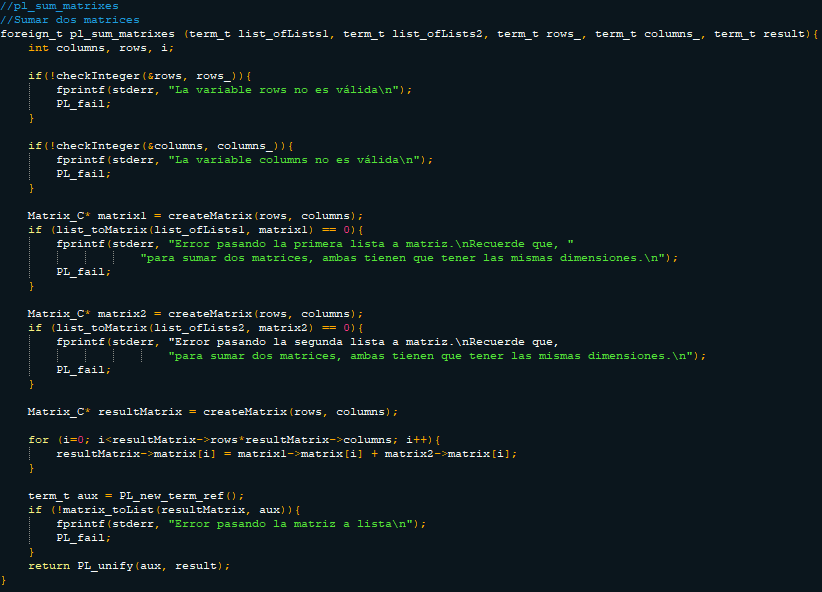


Se llama desde Prolog mediante total\_matrix.

## Predicado pl\_sum\_matrixes

Este predicado realiza la operación suma sobre dos matrices. Dado que la restricción para sumar dos matrices es que ambas tengan las mismas dimensiones, solo se pide al usuario que introduzca una vez las dimensiones. Los argumentos son:

* term\_t list\_ofLists1: la primera matriz
* term\_t list\_ofLists2: la segunda matriz
* term\_t rows\_: las filas de las matrices
* term\_t columns\_: las columnas de las matrices
* term\_t result: la matriz resultado que se devolverá

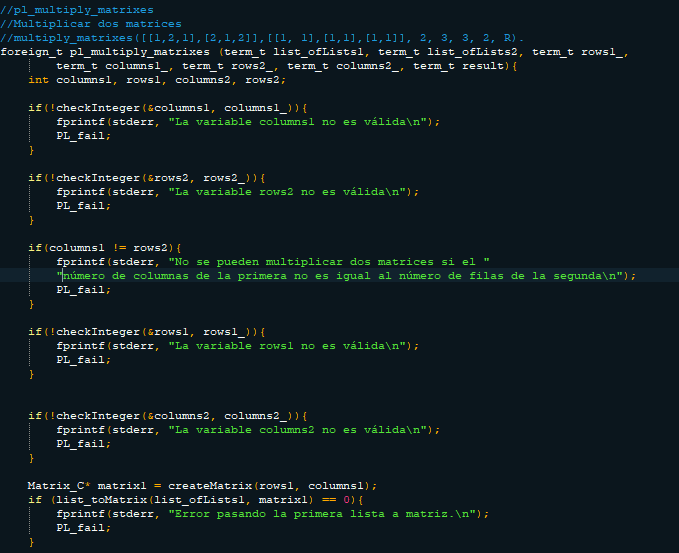
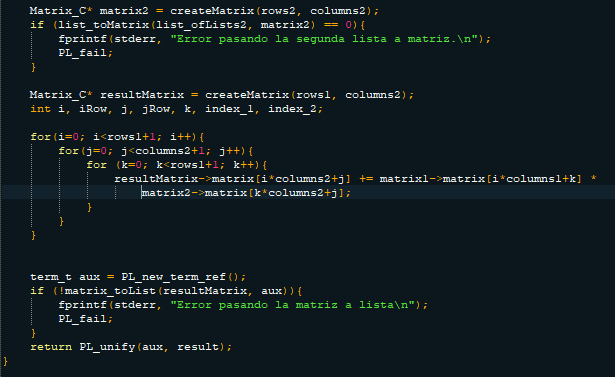


Se llama desde Prolog mediante sum\_matrixes.

## Predicado pl\_multiply\_matrixes

Este predicado multiplica dos matrices. Los argumentos son:

* term\_t list\_ofLists1: la primera matriz
* term\_t list\_ofLists2: la segunda matriz
* term\_t rows1\_: las filas de la primera matriz
* term\_t columns1\_: las columnas de la primera matriz
* term\_t rows2\_: las filas de la segunda matriz
* term\_t columns2\_: las columnas de la segunda matriz
* term\_t result: la matriz resultado que se devolverá

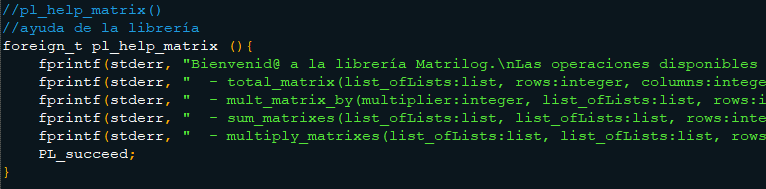
 

Se llama desde Prolog mediante multiply\_matrixes.

## Función pl\_help\_matrix

Esta función consiste en unos mensajes que muestran ayuda sobre la biblioteca.

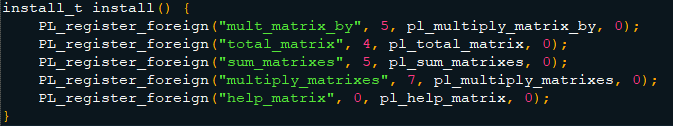
Esta pequeña ayuda incluye los nombres de las funciones y los argumentos que necesitan.



Se llama desde Prolog mediante help\_matrix.

## Función install

Esta función, como se ha explicado anteriormente, registra los predicados y les asigna un nombre para ejecutarlos desde Prolog.



# Conclusiones

Escribir esta librería ha sido tanto desafiante como interesante. Saber cómo se pueden escribir predicados para Prolog en otro lenguaje es muy útil y, en ocasiones como esta, prácticamente necesario, ya que Prolog, al ser un lenguaje declarativo, se puede usar perfectamente para realizar modelos y operaciones matemáticas con matrices. Sin embargo, al no tener ninguna estructura “matriz”, estas operaciones se hacen complicadas y tediosas de escribir, al contrario que en otro lenguaje que posea las facilidades de una estructura de datos.

Además, este trabajo me ha ayudado a reforzar tanto mis conocimientos en C como en Prolog, y hasta el momento en el que elegí hacerlo no sabía que era posible escribir librerías para un lenguaje de programación, en otro lenguaje de programación.

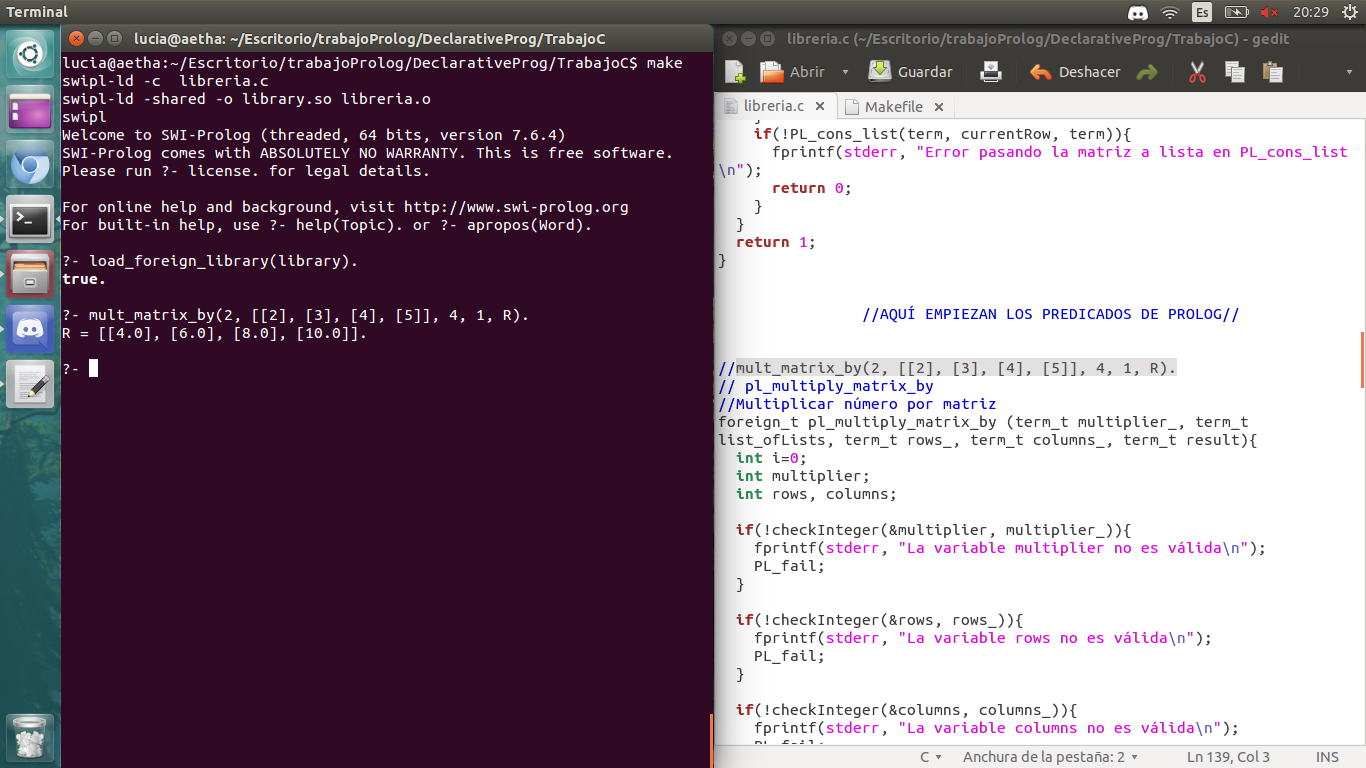
# Referencias

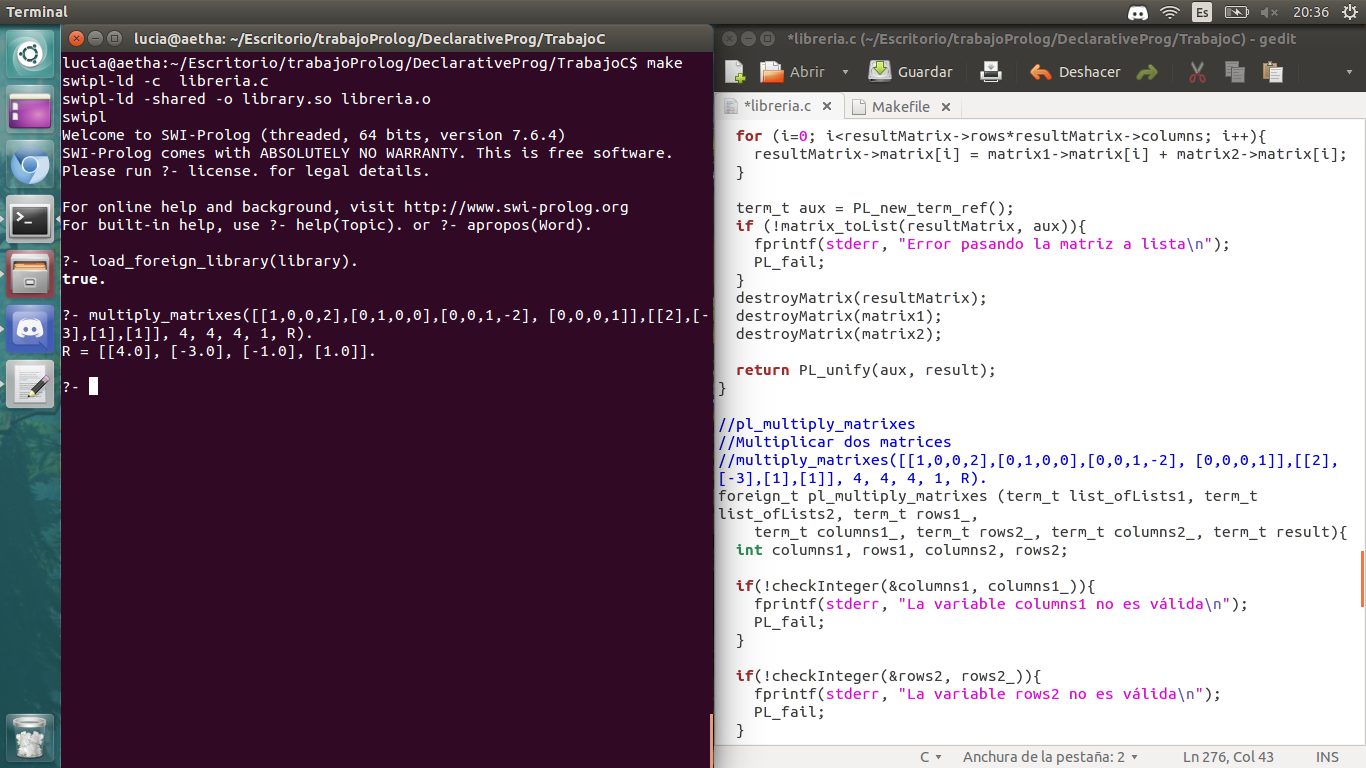
[1]. Foreign Language Interface – SWIProlog <https://www.swi-prolog.org/pldoc/man?section=foreign>

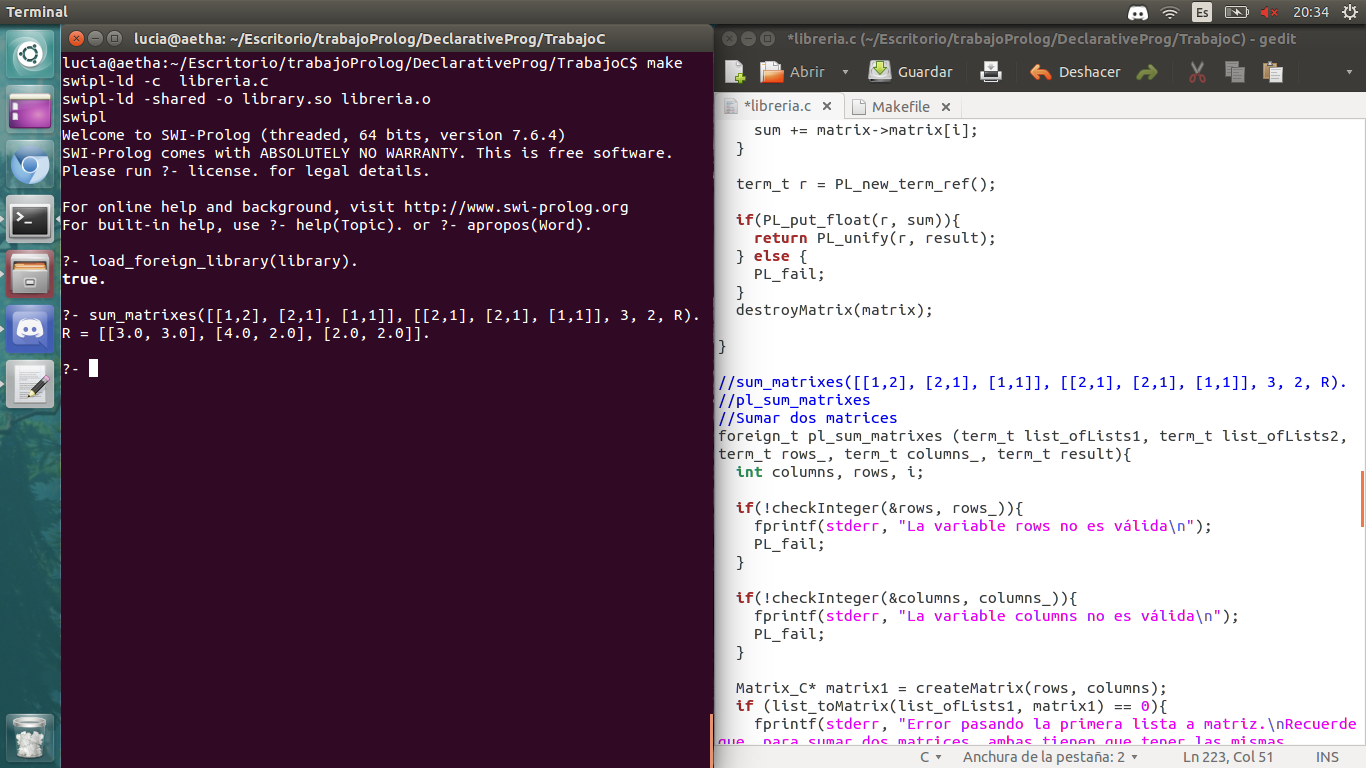
# Anexo: Ejecuciones

A continuación, se muestran capturas de pantalla con las ejecuciones de los diferentes predicados escritos.

* Multiplicar una matriz por un número: mult\_matrix\_by()



* Multiplicación de dos matrices: multiply\_matrixes()
* Suma de dos matrices: sum\_matrixes()



* Sumar todos los elementos de una matriz: total\_matrix()

