Taller multiplicación de matrices en C

Gustavo Adolfo Castillo Clavijo
Escuela de Ciencias Exactas e Ingeniería
Universidad Sergio Arboleda - Bogotá, Colombia
gustavoa.castillo@correo.usa.edu.co
CC: 1003559564

Felipe Uribe Guevara
Escuela de Ciencias Exactas e Ingeniería
Universidad Sergio Arboleda - Bogotá, Colombia
felipe.uribe@correo.usa.edu.co
CC: 1000970791

RESUMEN

El presente informe muestra la implementación de un programa que permite la multiplicación de matrices de igual tamaño, aleatorias para cada ejecución y datos de entrada (tamaño de las matrices e intervalo de aleatoriedad) antes del tiempo de ejecución en el lenguaje de programación C propuesto por el docente.

El informe es presentado en latex y se compila a través de la plataforma overleaf el cual es un editor de latex. Latex permite adjuntar un Script y presentarlo en el documento con 'Code listing', permitiendo que el lector pueda implementar y posteriormente probar el código, 'Code listing' se utilizó para mostrar todos los bloques de codigos en el documento

El programa realizado, fue escrito con el editor de código Visual Studio Code, compilado con GCC y se optimiza su construcción utilizando el Makefile.

Finalmente se obtuvo una correcta implementación para la multiplicación de matrices. Se utilizaron módulos para separar algunas tareas y de este modo tener un código limpio y fácil de entender.

1. MULTIPLICACIÓN DE MATRICES - EXPLICACIÓN TEORICA:

Para que dos matrices A y B puedan multiplicarse, A · B, es necesario que el número de columnas de la primera coincida con el número de filas de la segunda. Para este caso, debido a que son matrices cuadradas, no es necesario realizar la misma verificación, en cambio verificaremos que ambas matrices A y B sean cuadradas.

En tal caso, el producto A · B = R es otra matriz cuyos elementos se obtienen multiplicando cada vector fila de la primera por cada vector columna de la segunda, de la siguiente manera:

$$A = \begin{pmatrix} 2 & 3 & 5 \\ 7 & 2 & 4 \end{pmatrix}_{(2x3)}$$

$$B = \begin{pmatrix} 1 & 6 \\ 7 & 2 \\ 0 & -5 \end{pmatrix}_{(3x2)}$$

Figura 1: Matrices A y B.

$$A \cdot B = \begin{pmatrix} 2 & 3 & 5 \\ 7 & 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 & 6 \\ 7 & 2 \\ 0 & -5 \end{pmatrix} = \begin{pmatrix} 2\cdot1+3\cdot7+5\cdot0 & 2\cdot6+3\cdot3+5\cdot(-5) \\ 7\cdot1+2\cdot7+4\cdot0 & 7\cdot6+2\cdot2+4\cdot(-5) \end{pmatrix} = \begin{pmatrix} 23 & -7 \\ 21 & 26 \end{pmatrix}$$

Figura 2: Multiplicación de matrices.

2. IMPLEMENTACIÓN:

El algoritmo estándar de multiplicación de matrices tiene una complejidad de $O(n^3)$ en tiempo. Esto se debe a que se recorre la matriz posición por posición para luego recorrer cada una de las filas y columnas de matriz A y B respectivamente.

El programa consta de cinco funciones que se encargan de establecer el tamaño de una matriz, llenarla recogiendo datos, llenarla de forma aleatoria, imprimir una matriz y multiplicar dos matrices.

Para llenar la matriz se creó una interfaz y un módulo, la interfaz tiene dos funciones y una variable global. La variable global, la cual corresponde al tamaño de la matriz, una función para asignar el tamaño de la matriz y una última que llena la matriz con valores aleatorios. En la interfaz se encuentran las firmas de las funciones y en el módulo la implementación.

2.1. Numero aleatorio:

En C la función 'rand()' permite generar un numero aleatorio entre un intervalo de la siguiente manera:

Listing 1: Ejemplo para declarar un número aleatorio

```
// Numero Aleatorio
int intervalo = 10;
// Intervalo entre 0 y 10 para
int numeroAleatorio = (rand()  % intervalo);
```

Sin embargo, aunque es un número aleatorio dado un intervalo, para cada ejecución el número no cambia. Más adelante en el informe se soluciona el problema.

2.2. Interfaz y Modulo:

Creamos una interfaz que nos permite crear módulos y posteriormente dividir algunas de las funciones del programa en cada uno de estos módulos.

Primero declaramos una variable global 'SizeC', un entero que corresponde al tamaño de la matriz.

Declaramos la firma para la función que se encarga de establecer el tamaño y la firma de la función que se encarga de llenar la matriz con valores aleatorios.

Listing 2: Script modulo.h (Completo)

```
#ifndef MODULO_H_INCLUDED
#define MODULO_H_INCLUDED
#include <stddef.h>

int sizeC;

// Se declara el tama o de la matriz
int setSizeMatriz(int size);

// Se declara ka firma del metodo
void MatrizAletoria(int matriz[][sizeC], int aleatorio);

#endif
```

Se crea un modulo para implementar las funciones propuestas en la interfaz 'modulo.h'. El módulo necesita importar la interfaz 'modulo.h'.

La función 'SizeMatriz()', recibe como parámetro un entero, el cual corresponde al tamaño que será asignado a ambas matrices, la función 'MatrizAletoria()', recibe como parámetros una matriz con un tamaño preestablecido y un entero como rango.

Listing 3: Script modulo.c (Completo)

```
#include "modulo.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Funci n para establecer el tama o de la matriz
int setSizeMatriz(int size){
    sizeC = size;
  return sizeC;
}

```

```
// Funcion de matriz aleatoria
void MatrizAletoria(int matriz[][sizeC], int aleatorio){

    for (int i = 0; i < sizeC; i++){
        for (int j = 0; j < sizeC; j++){
            matriz[i][j] = (rand()  % aleatorio) + 1;
        }
    }
}
```

2.3. Principal y sus funciones:

2.3.1. Funciones y operaciones de matriz. Se crear un 'principal.c' que tiene el restante de las funciones (Llenar matriz, Imprimir matriz y multiplicar matrices)

Primero se realizan las importaciones, dentro de ellas se debe incluir el 'modulo.h' para inicializar la matriz de forma aleatoria:

Listing 4: Script principal.c (Importaciones)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "modulo.h"
```

Se define una variable size como el tamaño, esta variable es necesaria para realizar todas la operaciones de matrices. Posteriormente se crea la función 'LLenarMatriz()' esta función recibe como parametro una matriz inicializada y permite que el usuario ingrese manualmente los datos de una matriz:

Listing 5: Script principal.c (Tamaño de las matrices y función para llenar una matriz)

```
int size;

void LlenarMatriz(int matriz[][size]){
    // Se ingresan los datos de la matriz
    printf("\nIngrese los valores de la matriz: \n\n");
    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
         scanf("%d",&matriz[i][j]);
        }

    }
}
```

La función 'ImprimirMatriz()' recibe una matriz y la recorre. Consta de dos estructuras for anidadas y dibuja la matriz con el objetivo de visualizarla de manera entendible en tiempo de ejecución:

Listing 6: Script principal.c (Función para imprimir una matriz)

```
void ImprimirMatriz(int matriz[][size]){
    // Se imprimen los datos de la matriz
    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }
}
```

La función 'MultipliacionMatrices()' se encarga de multiplicar dos matrices que recibe como parámetros. Debe recorrer la matriz resultado posición por posición, y después debe recorrer las filas y columnas de la matriz A y la matriz B respectivamente para multiplicarlas:

Listing 7: Script principal.c (Función para multiplicar dos matrices)

```
void MultipliacionMatrices(int matrizA[][size], int matrizB[][size], int matrizR[][size]){
    // Se multpilican dos matrices
    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
            int R = 0;
            for (int z = 0; z < size; z++){
                R += matrizA[i][z] * matrizB[z][j];
            }
        matrizR[i][j] = R;
        }
    }
}
```

2.3.2. Función principal (Main). La función principal (main) se encarga de ejecutar todas las funciones e impresiones que tiene el programa, las cuales son, función para inicializar matriz de manera aleatoria, ingresando datos por pantalla, impresión de una matriz, y multiplicación de las matrices.

Es importante dar solución al problema que genera la función random a la hora de volver a ejecutarlo. La función 'rand()' al volver a compilarla escoge el mismo numero, por ello es necesario recurrir a la biblioteca '¡time.h¿' para solucionar el error.

El problema es que 'rand()', calcula los números aleatorios. Parte de un número inicial (llamado semilla), echa unas cuentas y saca un número aleatorio. Para el segundo número, echa unas cuentas con el resultado anterior y saca un segundo número y así sucesivamente.

Para evitar este problema tenemos la función 'srand()', a la que se le pasa de parámetro un número que se utilizará como número inicial para las cuentas. A esta función sólo debemos llamarla una vez en nuestro programa.

La fecha/hora del sistema. Este valor cambia si ejecutamos el programa en distinto instante de tiempo. Tendríamos que arrancar el programa dos veces en el mismo segundo para obtener la misma secuencia de números aleatorios. En C de linux esta fecha/hora se obtiene con la función 'time()'

La siguiente linea de codigo se pone una única vez al principio de nuestra función principal, esto soluciona el problema de los números aleatorios:

Listing 8: Script principal.c (Función srand con parametro de fecha/hora nulo)

```
    srand(time(NULL));
```

La función principal recibe dos parámetros ingresados por consola al momento de la ejecución estos parámetros son el tamaño de la matriz y el intervalo de aleatoriedad.

A continuación la función principal (main):

Listing 9: Script principal.c (Función principal (main))

```
int main(int _, char **entrada){
    srand(time(NULL));
    size = (int) atof(entrada[1]);
    int aleatorio = (int) atof(entrada[2]);
    setSizeMatriz(size);

    // Declaramos las futuras matrices
    int matrizA[size][size], matrizB[size][size], matrizR[size][size];

    // Se crea la primera matriz aleatoria
    MatrizAletoria(matrizA, aleatorio);
    //LlenarMatriz(matrizA, aleatorio);
    printf("\nMatriz A \n");

    ImprimirMatriz(matrizA);

    // Se crea la segunda matriz aleatoria
    MatrizAletoria(matrizB, aleatorio);
    //LlenarMatriz(matrizB, aleatorio);
    printf("\nMatriz B \n");
    ImprimirMatriz(matrizB);

    // Se multiplican amabas
```

```
24      MultipliacionMatrices(matrizA, matrizB, matrizR);
25
26      printf("\nEl resultado de la multiplicaci n: \n");
27      ImprimirMatriz(matrizR);
28
29      return 0;
30  }
```

2.4. MakeFile:

Se crea un make definiendo el compilador, los flags, el objetivo, la flag de programas y all esto con el objetivo de enlazar el nombre general de los archivos de programa.

Se definen dos comandos, principal y clean. Principal para compilar los archivos necesarios para el funcionamiento del programa y clean que permite borrar los archivos para volver a compilar con el principal.

A continuación el Makefile:

Listing 10: Makefile (Completo)

```
1  GCC = gcc
2  FLAGS = -pedantic -Wall
3  OBJ = -c
4
5  PROGS = principal
6
7  all: principal
8  principal:
9      $(GCC) $(FLAGS) $(OBJ) modulo.c
10     $(GCC) $(FLAGS) $(OBJ) $@.c
11     $(GCC) $(FLAGS) -o $@ $@.o modulo.o
12 clean:
13     $(RM) $(PROGS) *.o
```

3. EJECUCIÓN:

Para la ejecución se utiliza la terminal de linux para este caso en la distribución PopOs, para compilar el programa se utiliza la estructura establecida en el Makefile.

Se realizan dos ejecuciones por tamaño de la matriz para evidenciar el correcto funcionamiento del programa. A continuación la ejecución:

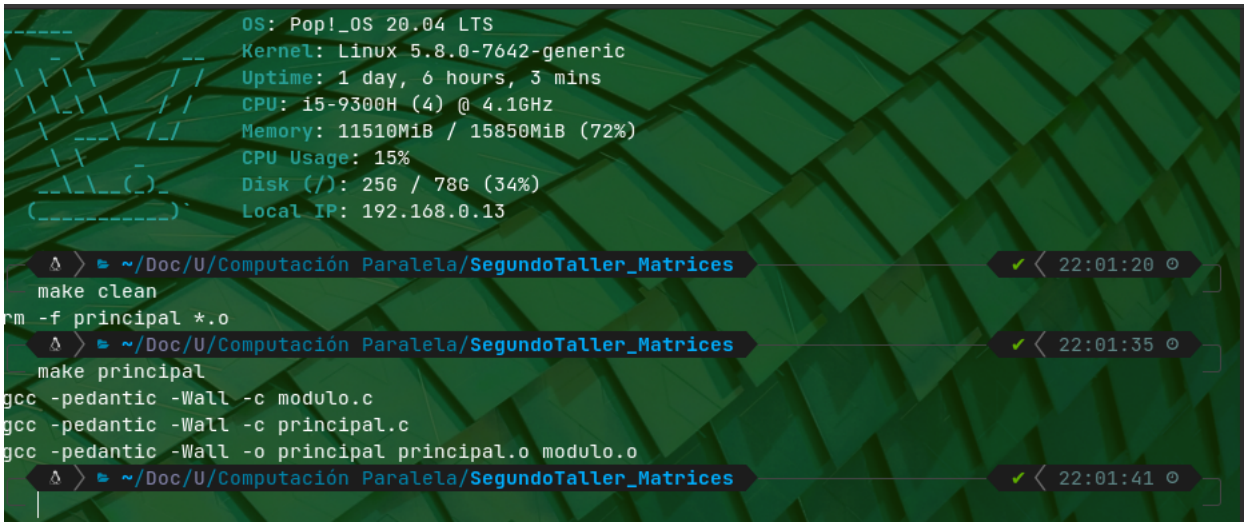

Figura 3: Construcción del programa con el Makefile.

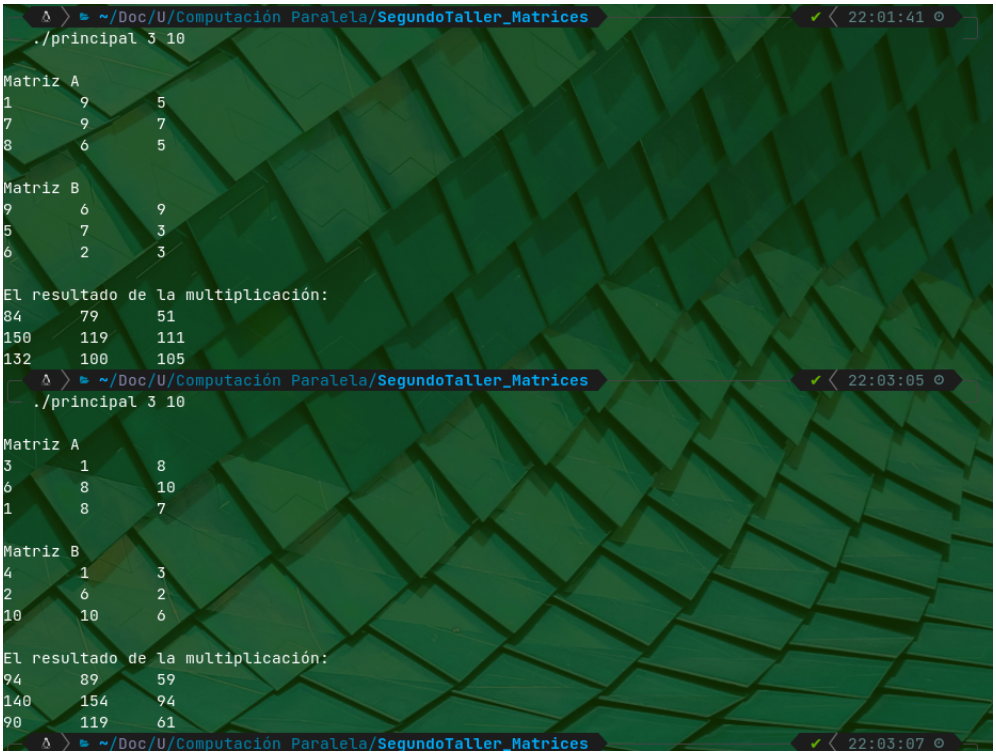

Figura 4: Ejecución con matrices de dimensión 3*3.

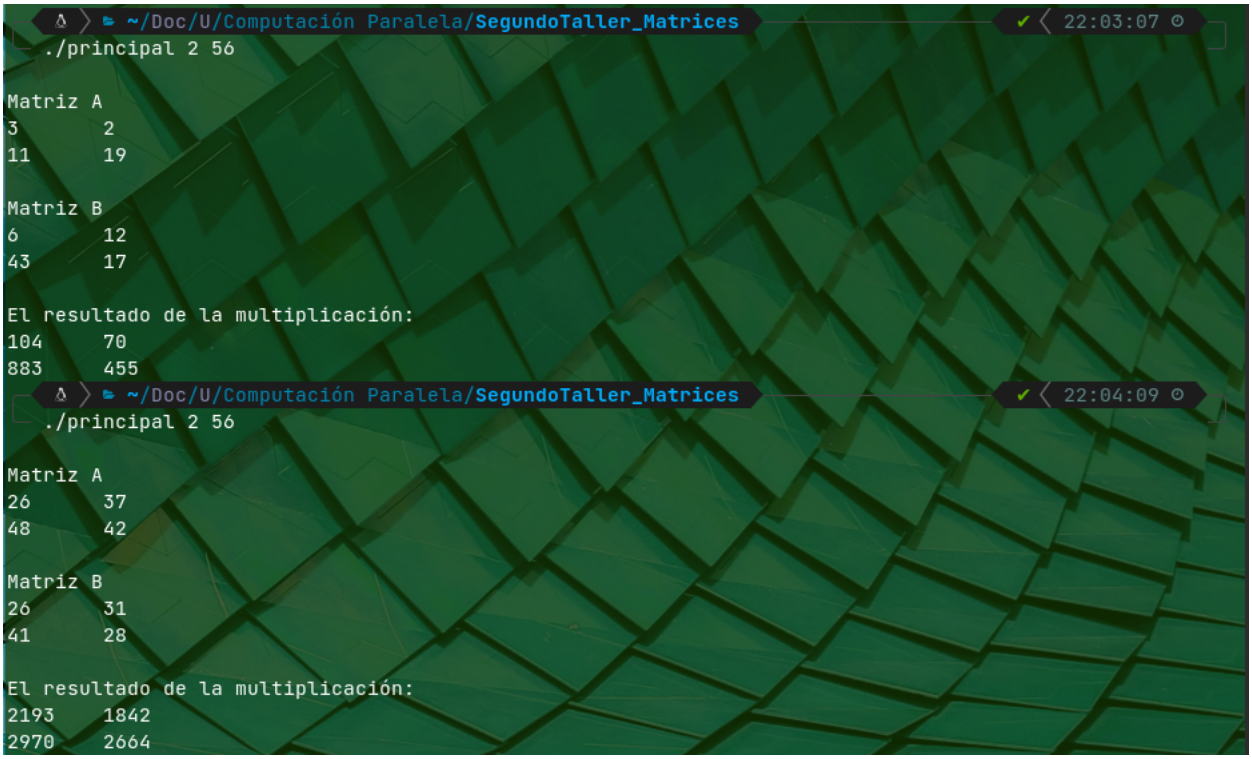

Figura 5: Ejecución con matrices de dimensión 2*2.

4. CONCLUSIONES:

Las herramientas de automatización como el Makefile, permiten agilizar el proceso de desarrollo de cualquier programa debido a que no se debe gastar tiempo ingresando comandos innecesarios en la consola, se establece todo en el Makefile.

La programación en C por módulos permite implementar bibliotecas que podrán ser utilizadas en cualquier momento si así se desea, esto con el objetivo de no repetir código y hacerlo más legible para cualquier otro, esto sin mencionar que un código limpio es sostenible por un gran equipo si se necesita para algún proyecto de mayores proporciones debido a la facilidad de compresión que permite.

El manejo de números aleatorios en C requiere de un poco mas atención para que sea completamente aleatorio sin importar la ejecución, esto debido a la semilla o número inicial que se establece para ello se implementan otras bibliotecas, en este caso se utiliza la función time y la función srand() que nos permite la correcta implementación de los números aleatorios en este programa.

Finalmente, las interfaces en C permiten crear variables que pueden ser utilizadas en los módulos, de esta manera se facilita el proceso de implementación de algunas funciones sin mencionar que la firma presenta una flexibilidad a la hora de jugar con los parámetros en los módulos que se creen posteriormente, similar a las interfaces, la sobrecarga de funciones y el polimorfismo visto en otros lenguajes de programación.