UNIVERSIDAD DE COSTA RICA

ESCUELA DE INGENIERÍA ELÉCTRICA

IE0117 PROGRAMACIÓN BAJO PLATAFORMAS ABIERTAS

Reporte

Laboratorio 3

Prof. Carolina Trejos Quirós

Estudiante: Diego Pereira[1], B85938

27 de setiembre de 2025

ÍNDICE

 UCR © 2025

1. Introducción

A modo introductorio a la programación bajo plataformas abiertas, específicamente usando el lenguaje C para este laboratorio, se pretende repasar un poco la teoría acerca de los tipos de variables disponibles para almacenar valores numéricos, especialmente sus rangos para los límites mínimo y máximo, pero especialmente el máximo, ya que ambos ejercicios sólo aumentan en el rango de los positivos.

Para ambos ejercicios de este laboratorio se manipulan los argumentos de entrada para generar una salida con el resultado esperado, según lo requerido para cada ejercicio. Ambos ejercicios requieren recibir como argumento de entrada únicamente un valor numérico entero, sin embargo, se pueden recibir más de los requeridos, e incluso datos inválidos o no enteros como lo esperado, es por ello, que también se revisa y analizan los argumentos recibidos para validar si se pueden usar o no, en caso contrario, solicitar al usuario del ejercicio ingresar de nuevo un argumento válido según lo requerido.

A manera de buena práctica, se descompone la funcionalidad general en problemas más pequeños, donde las funciones ayudan a realizar una tarea específica, para no tener código mezclado realizando tareas distintas, sino separado en varias funciones que al final juntas son capaces de dar solución al problema general planteado.

2. Implementación

2.1. Ejercicio1: Factorial en C

Según la Figura 1 acerca de un algoritmo para el cálculo de factorial a un número dado, se solicita encontrar y arreglar posibles problemas en el código que podrían causar errores inesperados, tanto en fase de compilación como de ejecución.

```c

#include <stdio.h>

int factorial (int n) {
  int i = 1;
  while (n > 1) {
    i = i * n;
    int n = n - 1;
  }
  return i;
}

int main (int argc, char *argv[]) {
  int fac4 = factorial(4);
  int fac5 = factorial(5);
  printf("4! = %d, 5! = %d\n", fac4, fac5);
  return 0;
}
```

Figura 1. Elaboración propia. *Algoritmo de factorial a modificar*, 2025.

2.1.1. Modificación 1 al Factorial

Para la fase de compilación solamente se encuentra un problema que es la doble declaración del tipo "int" para la variable "n", y debido a que ya se encuentra declarada en los parámetros de la función, no es necesario declararla de nuevo dentro del iterador "while".

Para la fase de ejecución también se encuentra un posible problema que es el tipo numérico que retorna la función, y dado que el resultado de factorial puede crecer rápidamente, se recomienda cambiar del tipo "int" al tipo "long double", lo cual también requiere cambiar la declaración para la variable "i".

Aunque la no inclusión de la validación no interrumpe la ejecución, se recomienda incluir la validación adicional dentro del iterador para evitar un resultado no confiable, en caso de que el resultado exceda los límites permisibles de rango para el tipo "long double", ya que es el valor numérico máximo que se puede almacenar.

```
166 long double factorial_long_double(int n) {
167     if (n < 0) {
168         // Factorial no definido para negativos
169         return 0;
170     }
171     long double i = 1;
172     while (n > 1) {
173         i = i * n;
174         n = n - 1;
175         if ((errno == ERANGE || (i <= LDBL_MIN || LDBL_MAX <= i)))
            return -1;
176     }
177     return i;
178 }
```

Figura 2. Elaboración propia. *Modificación 1 al algoritmo de factorial iterando con "while"*, 2025.

UCR © 2025

2.1.2. Modificación 2 al Factorial

Otra modificación adicional, aunque no necesaria, es cambiar el tipo de iterador de “while” a “for” para comparar los resultados obtenidos.

```
166 long double factorial_long_double(int n) {
167     if (n < 0) {
168         // Factorial no definido para negativos
169         return 0;
170     }
171     long double i = 1;
172     for (int j = 1; j <= n; j++) {
173         i *= j;
174         if ((errno == ERANGE || (i <= LDBL_MIN || LDBL_MAX <= i)))
            return -1;
175     }
176     return i;
177 }
```

Figura 3. Elaboración propia. *Modificación 2 al algoritmo de factorial iterando con “for”*, 2025.

2.2. **Ejercicio2**: Matriz con Cuadrado Mágico en C

Según la Figura 4 acerca de matrices cuadradas, se solicita recorrer la matriz para chequear si se trata o no de una matriz con cuadrado mágico. Adicional a ello se solicita poder replicar la funcionalidad dado un tamaño diferente de matriz, llenando la misma con números aleatorios.

```
1  #include <stdio.h>
2
3  #define SIZE 5
4
5  int findLargestLine(int matrix[][SIZE]) {
6   // Su implementacion
7
8  }
9
10 int main() {
11     int matrix[SIZE][SIZE] = {
12         {2, 7, 6},
13         {9, 5, 1},
14         {4, 3, 8}
15     };
16
17     int largestLine = findLargestLine(matrix);
18     printf("El tamano de la secuencia de 1s mas grande es: %d\n", largestLine)
       ;
19
20     return 0;
21 }
```

Figura 4. Elaboración propia. *Ejemplo de matriz con cuadrado mágico de tamaño impar*, 2025.

2.2.1. Orden/Tamaño de número impar

El algoritmo que se muestra en la Figura 5 se utiliza tanto para llenar matrices de tamaño impar, como para completar sub cuadrantes de las matrices pares que no son múltiplo de cuatro. Para ello se utiliza el método conocido como el "método Siamese" de De la Laubera (De la Loubère).

```
339  void fill_magic_odd_v2(int n_size, int magic_square[n_size][n_size], int
     randomly) {
346      int i = 0; // Current row
347      int j = (n_size /2); // Current column (middle of top row)
348      for (int num = 1; num <= n_size * n_size; num++) {
349          if (randomly != 1) magic_square[i][j] = num;
350          else magic_square[i][j] = get_int_random(1, (n_size * n_size));
             // Place random
351          int next_i = (i - 1 + n_size) % n_size; // Move up, wrap around
352          int next_j = (j + 1) % n_size;     // Move right, wrap around
353          if (magic_square[next_i][next_j] != 0) { // If next cell is
             occupied
354              i = (i + 1) % n_size; // Move one cell down
355          } else {
356              i = next_i;
357              j = next_j;
358          }
359      }
360  }
```

Figura 5. Elaboración propia. *Parte del algoritmo para matrices de tamaño impar*, 2025.

2.2.2. Orden/Tamaño múltiplo de cuatro

A cada función que llena la matriz según el tipo, se agrega una bandera llamada "randomly" para diferenciar cuando llenarla con números aleatorios o construir la matriz con cuadrado mágico. La Figura 6 muestra el tipo múltiplo de cuatro.

```
362  void fill_magic_even_doubly(int n_size, int magic_square[n_size][n_size
     ], int randomly) {
363      // magic_doubly_v2 para multiplos de 4
364      // Fill the square with numbers 1 to n_size*n_size
365      // and then mark elements to be swapped
366      for (int i = 0; i < n_size; i++) {
367          for (int j = 0; j < n_size; j++) {
368              if (randomly != 1) magic_square[i][j] = (n_size * i) + j + 1
                 ;
369              else magic_square[i][j] = get_int_random(1, (n_size * n_size
                 )); // Place random
370          }
371      }
406  }
```

Figura 6. Elaboración propia. *Parte del algoritmo para matrices múltiplo de cuatro*, 2025.

UCR © 2025

2.2.3. Orden/Tamaño diferente de los dos anteriores

Para el tipo de matriz que no es múltiplo de cuatro ni tampoco impar, se utiliza el método conocido como el "método LUX" de J.H. Conway.

```
410 void fill_magic_even_singly(int n_size, int magic_square[n_size][n_size
    ], int randomly) {
411     int temp_num, half_size = (n_size / 2), magic_odd[half_size][
        half_size];
412     int m_index = ((n_size - 2) / 4), swapped_cols = 0, magic_swap[
        n_size];
424     // Create the odd-order magic square
425     fill_magic_odd_v2(half_size, magic_odd, randomly);
426     // Fill the four sub-squares (L, U, X, and the fourth)
427     for (int i = 0; i < half_size; i++) {
428         for (int j = 0; j < half_size; j++) {
429             magic_square[i][j] = magic_odd[i][j]; // A=L
430             magic_square[i+half_size][j+half_size] = magic_odd[i][j] + (
                1*half_size*half_size); // D=1*X
431             magic_square[i][j+half_size] = magic_odd[i][j] + (2*
                half_size*half_size); // B=3*Fourth region
432             magic_square[i+half_size][j] = magic_odd[i][j] + (3*
                half_size*half_size); // C=2*U
433         }
434     }
460 }
```

Figura 7. Elaboración propia. *Parte del algoritmo para matrices diferentes de las dos anteriores*, 2025.

3. RESULTADOS

3.1. Ejercicio1: Factorial en C

```
----------------- Rango Decimal -----------------
---------------- Default: signed ----------------
Min       (short) : -32,768
Max       (short) : +32,767
Max     (u short) : +65,535
Min         (int) : -2,147,483,648
Max         (int) : +2,147,483,647
Max       (u int) : +4,294,967,295
Min        (long) : -9,223,372,036,854,775,808
Max        (long) : +9,223,372,036,854,775,807
Max      (u long) : +18,446,744,073,709,551,615
Min   (long long) : -9,223,372,036,854,775,808
Max   (long long) : +9,223,372,036,854,775,807
Max (u long long) : +18,446,744,073,709,551,615
----------------- Rango Decimal -----------------
--------------- Default: unsigned ---------------
Min       (float) : +0
Max       (float) : +340,282,346,638,528,859,811,704,183,484,516,925,440
```

Figura 8. Elaboración propia. *Límites en forma decimal de los diferentes tipos numéricos*, 2025.

```
--------------- Rango Exponencial ---------------
---------------- Default: signed ----------------
Min       (short) : -3.276800E+04
Max       (short) : +3.276700E+04
Max     (u short) : +6.553500E+04
Min         (int) : -2.147484E+09
Max         (int) : +2.147484E+09
Max       (u int) : +4.294967E+09
Min        (long) : -9.223372E+18
Max        (long) : +9.223372E+18
Max      (u long) : +1.844674E+19
Min   (long long) : -9.223372E+18
Max   (long long) : +9.223372E+18
Max (u long long) : +1.844674E+19
--------------- Rango Exponencial ---------------
--------------- Default: unsigned ---------------
Min       (float) : +1.175494E-38
Max       (float) : +3.402823E+38
Min      (double) : +2.225074E-308
Max      (double) : +1.797693E+308
Min (long double) : +3.362103E-4932
Max (long double) : +1.189731E+4932
```

Figura 9. Elaboración propia. *Límites en forma exponencial de los diferentes tipos numéricos,* 2025.

```
--------------- Factorial (short) ---------------
     Teorico 7! : 5,040
Experimental 7! : 5,040
     Teorico 8! : 40,320
Experimental 8! : -25,216
---------------- Factorial (int) ----------------
     Teorico 12! : 479,001,600
Experimental 12! : 479,001,600
     Teorico 13! : 6,227,020,800
Experimental 13! : 1,932,053,504
--------------- Factorial  (long) ---------------
     Teorico 20! : 2,432,902,008,176,640,000
Experimental 20! : 2,432,902,008,176,640,000
     Teorico 21! : 51,090,942,171,709,440,000
Experimental 21! : -4,249,290,049,419,214,848
------------- Factorial (long long) -------------
     Teorico 20! : 2,432,902,008,176,640,000
Experimental 20! : 2,432,902,008,176,640,000
     Teorico 21! : 51,090,942,171,709,440,000
Experimental 21! : -4,249,290,049,419,214,848
--------------- Factorial (float) ---------------
     Teorico 13! : 6,227,020,800
Experimental 13! : 6,227,020,800
     Teorico 14! : 87,178,291,200
Experimental 14! : 87,178,289,152
-------------- Factorial  (double) --------------
     Teorico 22! : 1,124,000,727,777,607,680,000
Experimental 22! : 1,124,000,727,777,607,680,000
     Teorico 23! : 25,852,016,738,884,976,640,000
Experimental 23! : 25,852,016,738,884,978,212,864
```

Figura 10. Elaboración propia. *Diferencias de factorial para los diferentes tipos numéricos,* 2025.

UCR © 2025

```
------------ Factorial (long double) ------------
Experimental 34! : 295,232,799,039,604,140,824,245,587,993,113,395,200
Experimental 35! : 1.033315E+40
------------ Factorial (long double) ------------
Experimental 1,754! : 1.979262E+4930
Experimental 1,755! : excede el rango
Ingrese un número para calcular el factorial : -1
Se requiere un entero igual/mayor que cero
Ingrese un número para calcular el factorial : 33
------------ Factorial (long double) ------------
Experimental 33! : 8,683,317,618,811,886,496,153,221,372,728,836,096
[diego@pereira Shared]$
```

Figura 11. Elaboración propia. *Diferencias de factorial usando el iterador "while"*, 2025.

```
------------ Factorial (long double) ------------
Experimental 34! : 295,232,799,039,604,140,861,139,076,140,532,498,432
Experimental 35! : 1.033315E+40
------------ Factorial (long double) ------------
Experimental 1,754! : 1.979262E+4930
Experimental 1,755! : excede el rango
Ingrese un número para calcular el factorial : -1
Se requiere un entero igual/mayor que cero
Ingrese un número para calcular el factorial : 33
------------ Factorial (long double) ------------
Experimental 33! : 8,683,317,618,811,886,496,153,221,372,728,836,096
[diego@pereira Shared]$
```

Figura 12. Elaboración propia. *Diferencias de factorial usando el iterador "for"*, 2025.

3.2. **Ejercicio2**: Matriz con Cuadrado Mágico en C

```
Ingrese el orden/tamaño de la matriz : 0
Se requiere un entero mayor que cero
Ingrese el orden/tamaño de la matriz : 7
---------------- Matriz Cuadrada ----------------
De Tamaño / Orden : 7
    30    39    48     1    10    19    28
    38    47     7     9    18    27    29
    46     6     8    17    26    35    37
     5    14    16    25    34    36    45
    13    15    24    33    42    44     4
    21    23    32    41    43     3    12
    22    31    40    49     2    11    20
Constante Mágica  : 175
---------------- Matriz Cuadrada ----------------
De Tamaño / Orden : 7
     6    10    10    16    18    24    49
    36    42    25     9    23    34    30
    46    11    24    18    48    11     9
    30    19    22    31    48    31    42
     7    21    38    46    19    15     6
     4     6     1     9    20    38    48
    22     6    17    24    40    10    24
Constante Mágica  : no es un cuadrado mágico
[diego@pereira Shared]$
```

Figura 13. Elaboración propia. *Ejemplo de matriz con tamaño impar*, 2025.

```
Ingrese el orden/tamaño de la matriz : -1
Se requiere un entero mayor que cero
Ingrese el orden/tamaño de la matriz : 8
---------------- Matriz Cuadrada ----------------
De Tamaño / Orden : 8
    64    63     3     4     5     6    58    57
    56    55    11    12    13    14    50    49
    17    18    46    45    44    43    23    24
    25    26    38    37    36    35    31    32
    33    34    30    29    28    27    39    40
    41    42    22    21    20    19    47    48
    16    15    51    52    53    54    10     9
     8     7    59    60    61    62     2     1
Constante Mágica  : 260
---------------- Matriz Cuadrada ----------------
De Tamaño / Orden : 8
    25    58    42    52    18    64    54    20
    23    51    59    44    51    60    29    58
    61     3    44     8    37    24    40    14
    55    27    18    29    13    33    10    27
    39    51    51     9    15    40    36    27
    38    30    59    41    40    23    31    21
    21    14    14     7    28    53    44    47
    50    62    53     2    34    62    36    57
Constante Mágica  : no es un cuadrado mágico
[diego@pereira Shared]$
```

Figura 14. Elaboración propia. *Ejemplo de matriz múltiplo de cuatro,* 2025.

UCR © 2025

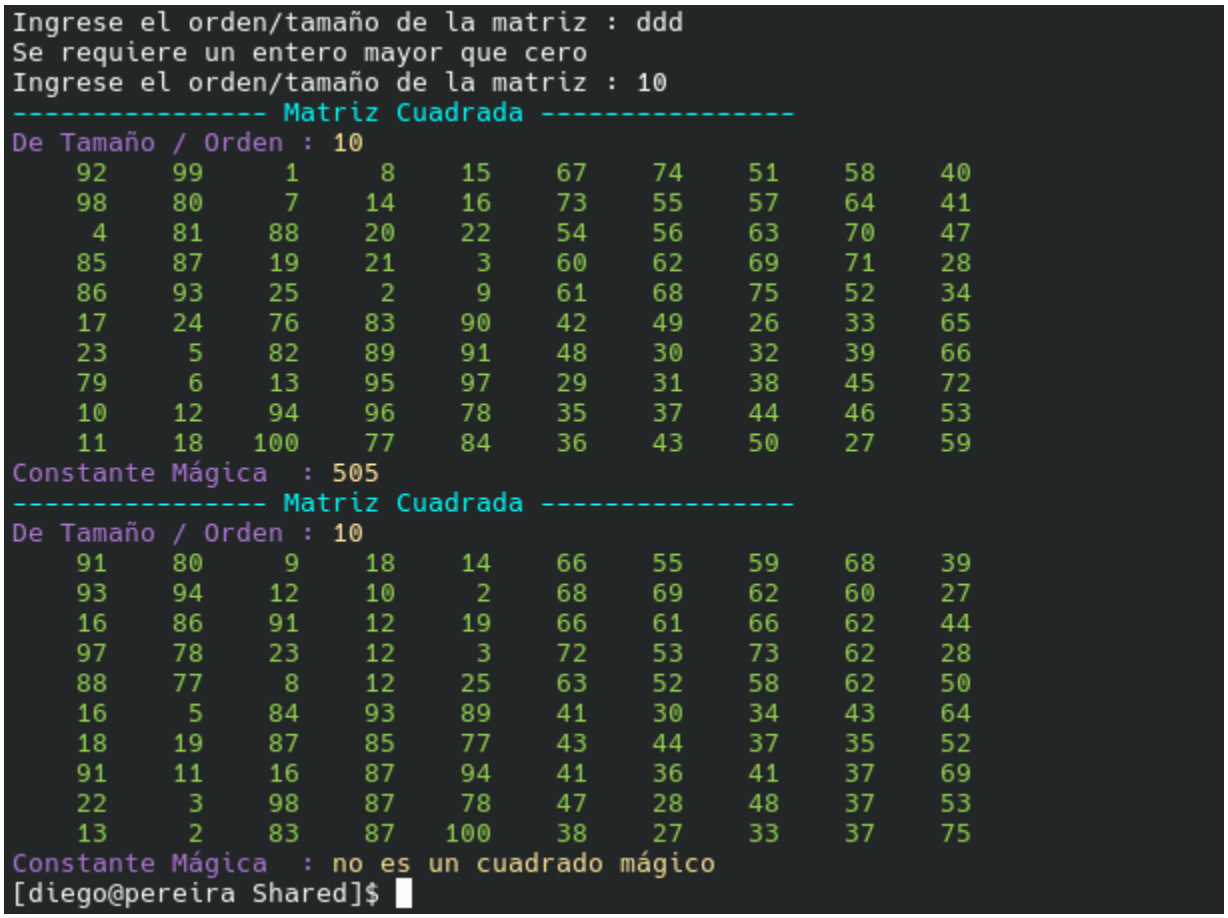

Figura 15. Elaboración propia. *Ejemplo de matriz diferente a las dos anteriores*, 2025.

4. Análisis de Resultados

Para la sección 3.1 acerca del cálculo de factorial a un número dado, los resultados pueden diferir según el tipo numérico empleado para almacenar el resultado, es por ello que de manera informativa primero se muestran los límites de rango mínimo y máximo según cada tipo numérico disponible en el lenguaje C.

La Figura 8 muestra los límites numéricos en forma decimal, sin embargo, no se incluyen los tipos “double” y “long double” porque son valores con excesiva cantidad de dígitos para imprimir en pantalla, por lo que la Figura 9 los muestra en forma exponencial, y por dicha razón es importante mencionar que todos los resultados usando esos dos tipos numéricos se muestran en la forma exponencial.

La Figura 10 muestra para cual factorial los resultados difieren de un tipo numérico a otro, siendo especialmente importantes las Figuras 11 y 12, donde para el número 35 la cantidad de dígitos del resultado factorial excede la cantidad de dígitos disponibles en el tipo numérico de “float”, por lo que a partir del límite de “float” todos los resultados se muestran en modo exponencial.

Las Figuras 11 y 12, además de mostrar las validaciones realizadas, también muestran el límite para el tipo “float” entre los números 34 y 35, donde la 11 muestra el resultado para el iterador “while” mientras que la 12 para el iterador “for”, según las cuales se observa que hasta el número 33 ambos resultados son iguales, sin embargo, a partir del número 34 se empiezan a notar algunas diferencias en los dígitos del resultado a partir del dígito 14.

Para la sección 3.2 acerca de matrices con cuadrado mágico, las Figuras 13, 14 y 15 muestran los resultados obtenidos para los tres tipos de matrices. Además de las validaciones realizadas para un tamaño mayor a cero, el cuadrado mágico solamente se muestra si al sumar cada fila todas resultan con la misma suma, lo mismo que para todas las columnas, y las dos diagonales, el resultado de sumar cada una por separado debe ser el mismo para todas, y solamente si ello se cumple se dice que la matriz es un cuadrado mágico. Adicional al ejemplo de cuadrado mágico se muestra otra matriz con el mismo tamaño dado pero llenada con números aleatorios, lo cual es muy poco probable que resulte en un cuadrado mágico por ser aleatorio, donde también cabe la posibilidad que se repitan números, lo cual no sucede en las matrices con constante mágica.

5. Conclusiones y recomendaciones

Según los resultados obtenidos para el primer ejercicio de factorial, se puede recomendar siempre el uso del tipo de dato numérico con mayor rango que es el “long double”, debido a que el resultado para factoriales suele crecer exponencialmente, sin embargo, aun así, aunque es el tipo de dato numérico de mayor capacidad para almacenar números, se observa que siempre existe la probabilidad de exceder la capacidad máxima del rango a partir de factorial 1755.

No se encuentra una razón definitiva para recomendar un tipo de iterador u otro, pero puede ser útil mencionar que, al comparar resultados con otras aplicaciones externas a este laboratorio, las diferencias obtenidas son poco menores al usar el iterador “for” en lugar del “while”.

UCR © 2025