

UNIVERSIDAD DE COSTA RICA

ESCUELA DE INGENIERÍA ELÉCTRICA

IE0117: PROGRAMACIÓN BAJO PLATAFORMAS ABIERTAS

---

# Reporte

## Laboratorio #

---

Prof. Carolina Trejos Quirós

Estudiante: David Abraham Vega Naranjo, C38344

3 de mayo de 2025

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Implementación</b>	<b>2</b>
2.1. Ejercicio 1 . . . . .	2
2.1.1. Pseudocódigo . . . . .	2
2.1.2. Código . . . . .	3
2.1.3. Explicacion . . . . .	3
2.2. Ejercicio 2 . . . . .	4
2.2.1. Código . . . . .	4
2.2.2. Explicacion . . . . .	4
2.3. Ejercicio 3 . . . . .	5
2.3.1. Pseudocódigo . . . . .	5
2.3.2. Codigos y explicación . . . . .	7
<b>3. Resultados</b>	<b>11</b>
3.1. Ejercicio 1 . . . . .	11
3.1.1. Ejemplos . . . . .	11
3.1.2. Salida . . . . .	11
3.2. Ejercicio 2 . . . . .	12
3.2.1. Salidas . . . . .	12
3.3. Ejercicio 3 . . . . .	13
3.3.1. Salidas . . . . .	13
<b>4. Conclusiones y recomendaciones</b>	<b>14</b>
<b>Referencias</b>	<b>15</b>

# 1. Introducción

En este laboratorio se llevó acabo diversas tareas orientas a formas de recorrer array normales o bidimensionales y tambien el uso dos bibliotecas nuevas nesacesarias para utilizar la funcion "rand" y "srand".

Mi repositorio [link](#).

## 2. Implementación

### 2.1. Ejercicio 1

#### 2.1.1. Pseudocódigo

```
1 // inicio
2 // definir arreglo_bidimensional[filas][columnas] de enteros
3 // definir variable suma = 0 como entero
4 // definir variable j = 0 como entero
5 // definir variable tamano igual a size matriz / size filas
6
7 // for i desde 0 hasta tamano
8 //     agregar valor de elemento de matriz a suma con "+="
9 //     ncrementar en 1 a j con "+= 1"
10
11 // si el residuo de tamano igual a 0
12 //     definir j como tamano - 1
13 //     for i desde 0 hasta tamano
14 //         agregar valor de elemento de matriz a suma con "+="
15 //         disminuir en 1 a j con "-="
16 // sino
17 //     definir j como tamano - 1
18 //     for i desde 0 hasta tamano
19 //         si i y j son iguales a tamano / 2
20 //             disminuir en 1 a j con "-="
21 //             saltar a siguiente iteracion
22 //         agregar valor de elemento de matriz a suma con "+="
23 //         disminuir en 1 a j con "-="
24 // imprimir resultado de suma
25 // fin
```

### 2.1.2. Código

```
1  int main() {
2      // int matriz[3][3] = {
3      //     {1, 2, 3},
4      //     {1, 5, 3},          // ejemplo #1
5      //     {1, 2, 8}
6      // };
7      // int matriz[4][4] = {
8      //     {1, 2, 3, 4},
9      //     {1, 7, 3, 4},          // ejemplo #2
10     //     {1, 2, 9, 4},
11     //     {1, 2, 3, 12}
12     // };
13     int matriz[5][5] = {
14         {1, 2, 3, 4, 5},
15         {1, 2, 3, 4, 5},          // ejemplo #3
16         {1, 2, 3, 4, 5},
17         {1, 2, 3, 4, 5},
18         {1, 2, 3, 4, 5}
19     };
20     int suma = 0;
21     int j = 0;
22     int tamaño = sizeof(matriz) / sizeof(matriz[0]);
23     for (int i = 0; i < tamaño; i++) {
24         suma += matriz[i][j];
25         j += 1;
26     }
27     if (tamaño % 2 == 0) {
28         j = tamaño - 1;
29         for (int i = 0; i < tamaño; i++) {
30             suma += matriz[i][j];
31             j -= 1;
32         }
33     } else {
34         j = tamaño - 1;
35         for (int i = 0; i < tamaño; i++) {
36             if (i == tamaño / 2 && j == tamaño / 2) {
37                 j -= 1;
38                 continue;
39             }
40             suma += matriz[i][j];
41             j -= 1;
42         }
43     }
44     printf("Suma de los elementos de las diagonales: %d\n", suma);
45     return 0;
46 }
```

### 2.1.3. Explicación

Para este código, lo más importante fue obtener el tamaño de la matriz. Al ser una matriz cuadrada, la ventaja es que con solo conocer el número de filas o columnas ya se puede asumir su tamaño, y así evitar problemas

con matrices rectangulares. Para sumar la primera diagonal (de izquierda a derecha) fue mucho más sencillo, ya que se puede conocer de antemano la posición del primer elemento de un array bidimensional o matriz. Con una simple iteración sobre las filas, e incrementando en 1 el índice de las columnas, se puede ir sumando cada elemento de la diagonal. Para la segunda diagonal (de derecha a izquierda), se evalúa si el número de filas o columnas es impar o par. Esto se hace para que, cuando la matriz es impar, no se sume dos veces el elemento del centro. Para saber en qué iteración se debe evitar sumar ese elemento, se verifica si se está en el centro. Se aprovecha el hecho de que, al dividir un número impar entre dos y siendo la variable de tipo int, el operador genera un número sin su parte decimal. Entonces se puede asumir que ese valor representa la mitad de la matriz. Cuando eso se cumple, se salta a la siguiente iteración con un continue. En este caso del for, el índice de las columnas se debe restar en 1 en cada iteración, ya que se parte desde la última posición del índice de la columna de la primera fila.

## 2.2. Ejercicio 2

### 2.2.1. Código

```
1  #include <stdio.h>
2
3  int encontrarMaximo (int arr[], int n) {
4      int maximo = arr[0];
5
6      for (int i = 1; i < n; i++) {
7          if (arr[i] > maximo) {
8              maximo = arr[i];
9          }
10     }
11
12     return maximo;
13 }
14
15 int main() {
16     int numeros[ ] = {10, 20, 5, 40, 30};
17     int n = sizeof(numeros) / sizeof(numeros[0]);
18     int maximo = encontrarMaximo (numeros, n);
19     printf("El numero mas grande es: %d\n", maximo);
20     return 0;
21 }
```

### 2.2.2. Explicacion

Para la revisión de este código encontré que el contador "maximo", dentro del for donde se compara con el siguiente elemento del array, estaba verificando si ese elemento era menor que "maximo". En ese caso, "maximo" pasaba a ser ese elemento. Entonces, luego de todas las iteraciones, el

programa devolvía el elemento numéricamente menor del array.

El objetivo indicado en el documento pedía que, para este programa, se retornara el elemento mayor de un array. Buscando en el código proporcionado, encontré que en esa verificación del máximo estaba escrito "arr[i] < maximo" en lugar de "arr[i] > maximo". Con este reemplazo, al completarse las iteraciones, la función ahora sí retornaría el valor máximo de un elemento en el array.

## 2.3. Ejercicio 3

### 2.3.1. Pseudocódigo

```
1 // definir size fijo de matriz cuadrada
2 // funcion de buscar secuencia mas larga de unos.
3 //     definir contator de tamaño en 0
4 //     for i desde 0 hasta size
5 //         definir inicio de la columna en 0
6 //         definir contador de 1s en 0
7 //         for f desde igual a i hasta f < size y c < size
8 //             si el elemento de la matriz es igual a 1
9 //                 se aumenta el contador de 1s
10 //                 si contador de 1s > contador de tamaño
11 //                     contador de tamaño = contador de 1s
12 //                 sino
13 //                     se reinicia contador de 1s
14 //             se incrementa c
15 //         finfor
16 //     finfor
17
18 // for i desde 0 hasta size - 1
19 //     definir inicio de la columna en 1 + i
20 //     definir contador de 1s en 0
21 //     for f desde 0 hasta f < size - 1 y c < size
22 //         si el elemento de la matriz es igual a 1
23 //             se aumenta el contador de 1s
24 //             si contador de 1s > contador de tamaño
25 //                 contador de tamaño = contador de 1s
26 //             sino
27 //                 se reinicia contador de 1s
28 //             se incrementa c
29 //         finfor
30 //     finfor
31
32 // for i desde 0 hasta size
33 //     definir inicio de la columna en size - 1
34 //     definir contador de 1s en 0
35 //     for f desde igual a i hasta f < size y c >= 0
36 //         si el elemento de la matriz es igual a 1
37 //             se aumenta el contador de 1s
38 //             si contador de 1s > contador de tamaño
39 //                 contador de tamaño = contador de 1s
40 //             sino
41 //                 se reinicia contador de 1s
```

```
42      //      se decrementa c
43      //      finfor
44  //      finfor
45
46
47      // for i desde 0 hasta size - 1
48      //      definir inicio de la columna en (size - 2) - i
49      //      definir contador de 1s en 0
50      //      for f desde 0 hasta f < size y c >= 0
51      //          si el elemento de la matriz es igual a 1
52      //              se aumenta el contador de 1s
53      //              si contador de 1s > contador de tamaño
54      //                  contador de tamaño = contador de 1s
55      //              sino
56      //                  se reinicia contador de 1s
57      //          se decrementa c
58      //      finfor
59  //      finfor
60
61      // retornar tamaño
62  // fin
63
64      // funcion void para hacer la matriz aleatoria
65      //      imprimir titulo
66      //      for f desde 0 hasta size
67      //          for c desde 0 hasta size
68      //              anadir elemento aleatorio entre 0 y 1
69      //              imprimir elemento
70      //              imprimir salto de linea
71
72  // inicio
73  //      cambiar la "semilla" conforme el tiempo
74  //      definir la matriz conforme size
75  //      definir el tamaño de la secuencia
76  //      imprimir la secuencia
77  //      retornar 0
78  // fin
```

### 2.3.2. Codigos y explicación

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define size 3
6
7  int findLargestLine(int matrix[][size]) {
8
9      int tamano = 0;
10
11
12     // para las diagonales de izquierda a derecha
13
14     for (int i = 0; i < size; i++) { // primera parte
15         int c = 0;
16         int newtamano = 0;
17         for (int f = i; f < size && c < size; f++) {
18             if (matrix[f][c] == 1) {
19                 newtamano ++;
20                 if (newtamano > tamano) {
21                     tamano = newtamano;
22                 }
23             } else {
24                 newtamano = 0;
25             }
26             c++;
27         }
28     }
29     for (int i = 0; i < size - 1; i++) { // segunda parte
30         int c = 1 + i;
31         int newtamano = 0;
32         for (int f = 0; f < size - 1 && c < size; f++) {
33             if (matrix[f][c] == 1) {
34                 newtamano ++;
35                 if (newtamano > tamano) {
36                     tamano = newtamano;
37                 }
38             } else {
39                 newtamano = 0;
40             }
41
42             c++;
43
44         }
45     }
46
47     // para las diagonales de derecha a izquierda
48
49     for (int i = 0; i < size; i++) { // tercera parte
50         int c = size - 1;
51         int newtamano = 0;
52         for (int f = i; f < size && c >= 0; f++) {
53             if (matrix[f][c] == 1) {
```



```
55         newtamano ++;
56         if (newtamano > tamano) {
57             tamano = newtamano;
58         }
59     } else {
60         newtamano = 0;
61     }
62     c--;
63 }
64 }
65 for (int i = 0; i < size - 1; i++) { // cuarta parte
66     int c = (size - 2) - i;
67     int newtamano = 0;
68     for (int f = 0; f < size && c >= 0; f++) {
69         if (matrix[f][c] == 1) {
70             newtamano ++;
71             if (newtamano > tamano) {
72                 tamano = newtamano;
73             }
74         } else {
75             newtamano = 0;
76         }
77         c--;
78     }
79 }
80 return tamano;
81 }
```

Se crea una función cuyo propósito es contar el tamaño de 1s consecutivos en las diagonales para ello se implementan una serie de for que se explican seguidamente:

Para las diagonales de izquierda a derecha, en la primera parte, el for de inicio marca el número de iteraciones, que equivale al número de diagonales desde 0 hasta size. Se ejecuta solo si es menor que size y se incrementa en 1. El siguiente for comienza en la primera fila y columna; se ejecuta solo si el número de filas y columnas es menor que size, y ambas aumentan en 1. En las siguientes iteraciones, el for de inicio indica desde qué fila se empieza, así no se toma en cuenta las mismas diagonales.

En la segunda parte, el for de inicio se ejecuta una vez menos que el anterior debido a que hay menos diagonales. Para el siguiente for, se empieza en la primera fila y en la columna uno, aumentando conforme al for de inicio, porque sino se tomaría en cuenta la diagonal del medio otra vez. Este for se ejecuta solo mientras se cumplan dos condiciones al mismo tiempo: que el número de iteraciones de filas sea menor en 1 con respecto a la primera parte y que el número de columnas sea menor que size.

Para las diagonales de derecha a izquierda, en la tercera parte, como en la anterior, el for de inicio también hace iteraciones dependiendo de size, ya que se comienza desde la diagonal central. En el siguiente for, se inicia desde la primera

fila y la última columna. Este se ejecuta solo si el número de filas es menor que size y el número de columnas es mayor o igual a 0, ya que la columna empieza desde el valor máximo. Es necesario tener un control para que no se ejecute indefinidamente.

En la cuarta parte, al for de inicio se le quita una iteración por los mismos motivos anteriores. En el siguiente for, se inicia en la fila 0, mientras que la columna comienza en "size - 2" y se le resta el número de iteraciones del for de inicio. Esto se hace para no tomar en cuenta diagonales ya revisadas.

Para cada una de las partes anteriores se añade la lógica necesaria para que, con base en unos contadores, se halle el verdadero tamaño máximo de unos consecutivos. Un contador "global" llamado "tamano", iniciado en 0, se utiliza para almacenar el valor máximo encontrado. Se verifica si el elemento de la matriz de esa iteración es igual a 1; si es así, se incrementa el contador "newtamano". Luego, se comprueba si "newtamano" es mayor que "tamano"; si lo es, se actualiza "tamano" con el valor de "newtamano". De no ser así, el contador de unos "newtamano" se reinicia. Esta variable "tamano" se retorna al final de la función.

```
1 void MakeMatrixAleatoria(int matrix[][size]) {
2     printf("La matriz utilizada corresponde a :\n");
3     for (int f = 0; f < size; f++) {
4         for (int c = 0; c < size; c++) {
5             matrix[f][c] = rand() % 2; // rellena la matriz con unos y ceros
6
7             printf("%d ", matrix[f][c]);
8         }
9         printf("\n");
10    }
11
12 }
13
14 int main() {
15     srand(time(NULL));
16
17     int matrix[size][size];
18     MakeMatrixAleatoria(matrix);
19
20     int largestLine = findLargestLine(matrix);
21     printf("El tamaño de la secuencia de 1s mas grande es: %d\n" ,
22     largestLine);
23
24     return 0;
25 }
```

Parte del resto del código es la implementación de la función void "MakeMatrixAleatoria". Se usa "void" porque la función no retorna nada. Básicamente, esta función lo que hace es ir iterando entre filas y columnas, reemplazando los elementos entre 0 y 1, gracias a la función "rand" de la biblioteca "stdlib", que proporciona la aleatoriedad.

Por último, la función principal "main", donde se ejecuta la función "srand(time(NULL))" es indispensable para que en cada ejecución del programa completo la matriz sea aleatoria. La función "rand" funciona con un algoritmo que crea una secuencia de números donde el primer número de esa secuencia se le llama "semilla", y con "time" de la biblioteca "time" se crea esa "semilla" a partir del tiempo, lo cual genera esa aleatoriedad buscada. También se define la matriz, se ejecuta la función creada "MakeMatrixAleatoria", se define el tamaño de la secuencia, y se imprime dicho número.

### 3. Resultados

#### 3.1. Ejercicio 1

##### 3.1.1. Ejemplos

```
8
9     int matriz[3][3] = {
10         {1, 2, 3},
11         {1, 5, 3},          // ejemplo #1
12         {1, 2, 8}
13     };
14
15     int matriz[4][4] = {
16         {1, 2, 3, 4},
17         {1, 7, 3, 4},          // ejemplo #2
18         {1, 2, 9, 4},
19         {1, 2, 3, 12}
20     };
21
22     int matriz[5][5] = {
23         {1, 2, 3, 4, 5},
24         {1, 2, 3, 4, 5},          // ejemplo #3
25         {1, 2, 3, 4, 5},
26         {1, 2, 3, 4, 5},
27         {1, 2, 3, 4, 5}
28     };
```

Figura 1: ejemplos de matrices

##### 3.1.2. Salida

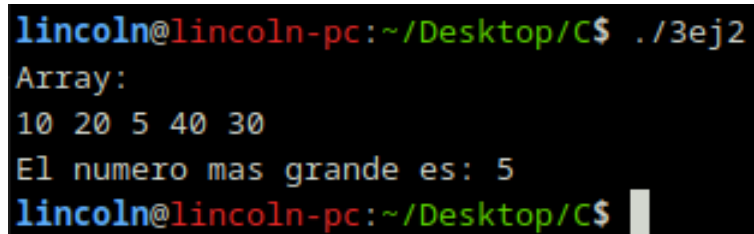
```
lincoln@lincoln-pc:~/Desktop/C$ gcc -Wall 3ejercicio1.c -o 3ej1
lincoln@lincoln-pc:~/Desktop/C$ ./3ej1
Suma de los elementos de las diagonales: 27
lincoln@lincoln-pc:~/Desktop/C$ gcc -Wall 3ejercicio1.c -o 3ej1
lincoln@lincoln-pc:~/Desktop/C$ ./3ej1
Suma de los elementos de las diagonales: 39
lincoln@lincoln-pc:~/Desktop/C$ gcc -Wall 3ejercicio1.c -o 3ej1
lincoln@lincoln-pc:~/Desktop/C$ ./3ej1
Suma de los elementos de las diagonales: 18
lincoln@lincoln-pc:~/Desktop/C$
```

Figura 2: salida de código

Con base a los ejemplos de las matrices y la salida del código se puede corroborar que se cumple con la suma de cada elemento de las matrices y que además se evita que se sume dos veces el mismo elemento en el centro de las matrices.

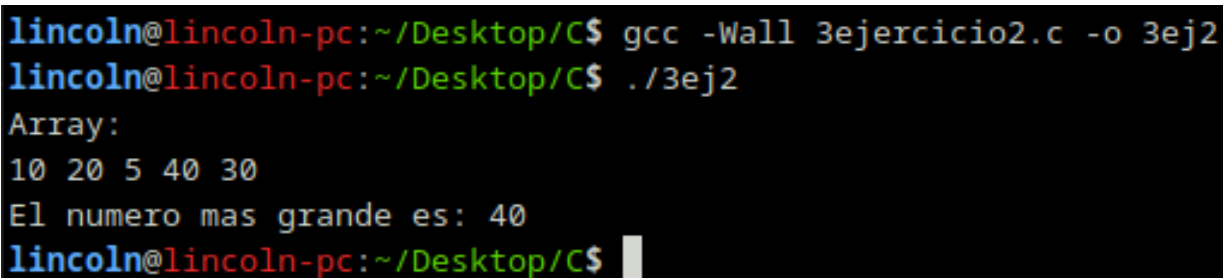
## 3.2. Ejercicio 2

### 3.2.1. Salidas

A terminal window with a black background and colored text. The prompt is 'lincoln@lincoln-pc:~/Desktop/C\$'. The user enters './3ej2'. The output is 'Array:', followed by '10 20 5 40 30' on the next line, and 'El numero mas grande es: 5' on the third line. The prompt is shown again at the end.

```
lincoln@lincoln-pc:~/Desktop/C$ ./3ej2
Array:
10 20 5 40 30
El numero mas grande es: 5
lincoln@lincoln-pc:~/Desktop/C$
```

Figura 3: Salida incorrecta

A terminal window with a black background and colored text. The prompt is 'lincoln@lincoln-pc:~/Desktop/C\$'. The user enters 'gcc -Wall 3ejercicio2.c -o 3ej2'. The prompt is shown again. The user enters './3ej2'. The output is 'Array:', followed by '10 20 5 40 30' on the next line, and 'El numero mas grande es: 40' on the third line. The prompt is shown again at the end.

```
lincoln@lincoln-pc:~/Desktop/C$ gcc -Wall 3ejercicio2.c -o 3ej2
lincoln@lincoln-pc:~/Desktop/C$ ./3ej2
Array:
10 20 5 40 30
El numero mas grande es: 40
lincoln@lincoln-pc:~/Desktop/C$
```

Figura 4: Salida correcta

Observando las salidas anteriores del código, se confirma que, haciendo el cambio mencionado anteriormente en la explicación del código, ahora sí se está imprimiendo el valor del elemento mayor del array.

### 3.3. Ejercicio 3

#### 3.3.1. Salidas

```
lincoln@lincoln-pc:~/Desktop/C$ gcc -Wall 3ejercicio3.c -o 3ej3
lincoln@lincoln-pc:~/Desktop/C$ ./3ej3
La matriz de tamaño 4 utilizada corresponde a :
1 0 0 1
1 0 0 1
1 1 1 0
1 0 0 1
El tamaño de la secuencia de 1s mas grande es: 2
lincoln@lincoln-pc:~/Desktop/C$
```

Figura 5: Salida matriz tamaño 4

```
lincoln@lincoln-pc:~/Desktop/C$ ./3ej3
La matriz de tamaño 7 utilizada corresponde a :
0 1 1 0 1 1 0
1 1 1 0 1 1 0
1 1 0 1 0 0 1
1 0 1 0 0 0 1
0 1 0 0 0 1 0
0 0 1 1 0 0 1
1 0 1 0 1 1 1
El tamaño de la secuencia de 1s mas grande es: 5
lincoln@lincoln-pc:~/Desktop/C$
```

Figura 6: Salida matriz tamaño 7

Con respecto a las salidas anteriores del código, se confirma que funciona para diferentes tamaños de matrices y se conserva la aleatoriedad de los elementos entre 0 y 1. Además, se observan ambos casos de las diagonales: de izquierda a derecha y viceversa.

## 4. Conclusiones y recomendaciones

Gracias a la implementación de cada uno de los programas anteriores, pude aprender una buena base sobre cómo recorrer los arrays y, en mi opinión, una manera bastante curiosa de hacerlo siendo de forma diagonal, tanto de izquierda a derecha como viceversa. También comprendí el cuidado que hay que tener al generar la lógica de los "for": desde dónde iniciar y terminar, y hasta qué condición se ejecutan. Además, fue indispensable el uso de contadores locales y globales para llevar el control de los "for" y, sobre todo, para cumplir el objetivo en algunos ejercicios.

Asimismo, para el último ejercicio se utilizó la función "rand" y "srand" en combinación con las bibliotecas "stdlib" y "time", ya que sin ellas no se completaría el programa. Se aprendió cierta lógica detrás de la función "rand", la cual, sin "srand(time(NULL))", no podría cambiar la semilla ni otorgar la aleatoriedad buscada.

## Referencias

@manualcodigazoRand, author = Codigazo, title = Cómo usar la función rand en C, year = s.f., url = <https://www.codigazo.com/en-c/como-usar-funcion-rand-en-c>, note = Consultado en mayo de 2025