1. Escribir un programa que declare un *array* de tipo `int` que almacene los primeros diez números naturales. Imprimir el *array* un elemento por línea.
2. Escribir un programa que haga lo mismo que el ejercicio anterior pero para los primeros *n* números naturales. El usuario ingresa *n* como argumento de línea de comandos.
3. Escribir un programa que acepte dos argumentos *n* y *m*. El número *n* es el tamaño de un *array* de enteros. El programa llena el *array* con números aleatorios entre 0 y *m* y los imprime uno por línea.
4. Escribir un programa que genere un *array* de 100 elementos con enteros aleatorios entre 0 y 100. Calcular el promedio de los valores en el *array*.
5. Escribir un programa que genere un *array* de 100 elementos con enteros aleatorios entre 0 y 100. Encontrar el máximo y el mínimo del *array*.
6. Escribir un programa que declare el *array* de enteros 1, 2, 3, 4, 5, 6 e invierta el orden del mismo. Luego imprimirlo.
7. Escribir un programa similar al anterior pero que invierta un *array* con los primeros 20 números pares.
8. Escribir un programa que acepte como argumento un entero *n*. Generar una lista con los primeros *n* números de Fibonacci usando un *array* de dimensión *n*.
9. Escribir un programa que acepte dos argumentos enteros *s* y *n*, siendo *s* el valor semilla para `srand()`. El programa declara e inicializa un *array* de enteros de longitud 10 con números aleatorios módulo *n*. Buscar el mínimo y máximo del *array*. Imprimir la lista en una sola línea indicando con una flecha hacia arriba el máximo y una hacia abajo el mínimo.
10. Escribir un programa que haga lo mismo que el programa `echo` de Linux.
11. Escribir un programa que calcule el producto punto entre dos vectores de dimensión *N* y valores reales. Siendo *N* una constante conocida en tiempo de compilación al igual que los componentes de los vectores.
12. Escribir un programa que genere un *array* de 100 elementos con valores aleatorios enteros del 0 al 9. Contar la cantidad de veces que aparece cada dígito en el *array* e imprimir una tabla con las frecuencias.
13. Escribir un programa que reciba un *string* como argumento y cuente la cantidad de carácteres en la palabra.
14. Escribir un programa que reciba un *string* como argumento y cuente cuántas veces aparece cada letra en la cadena. Ignorar dígitos o signos de puntuación.
15. Escribir un programa que reciba un *string* como argumento y decida si la palabra es un palíndromo o no.
16. Escribir una función que reciba un argumento de tipo *string* y busque en un *array* conocido en tiempo de compilación si el *string* está en el *array*. Devolver el índice del elemento en el *array* si fue encontrado. De lo contrario devolver -1.
17. Escribir una función `to_lower_case()` que convierta una palabra a todas letras minísculas. Ignorar dígitos o signos de puntuación.
18. Escribir una función `to_upper_case()` que convierta una palabra a todas letras mayúsculas. Ignorar dígitos o signos de puntuación.
19. Escribir un programa en C que acepte dos argumentos. Un entero *n*, y un *string*. El programa debe realizar el cifrado del segundo argumento usando un cifrado de desplazamiento, siendo la cantidad a desplazar *n*. El segundo argumento es siempre una palabra que consiste solo de letras del alfabeto inglés, todas en minísculas. Imprimir la palabra cifrada.
20. Escribir una función llamada `parse_int()` que funcione de manera similar a `atoi`. Devolver el valor de un *string* como número entero, siempre y cuando los caracteres sean dígitos del 0 al 9. Debe funcionar para números enteros positivos y negativos.

21. Escribir una función que reciba como argumento un *string* y elimine el espacio en blanco al principio o al final del mismo.

22. Algunas palabras son muy largas. Por ejemplo *internationalization* o *localization*. Escribir un programa que acepte una palabra como argumento en la línea de comandos. Si la longitud de la palabra es menor o igual a 10 imprimir la palabra tal como fue ingresada. Si tiene más de 10 caracteres imprimir únicamente la primer letra seguida de un entero *n* seguido de la última letra, siendo *n* la cantidad de letras entre la primera y la última. Por ejemplo para *localization* el programa imprime `l10n` y para *internationalization* imprime `i18n`. Pero para *word* imprime `word`.

23. Escribir un programa que reciba un argumento entero e imprima el mes del año correspondiente o un error si el argumento es menor a 1 o mayor a 12.

24. Escribir un programa que traduzca una palabra ingresada como argumento a código Morse, separando cada letra con un espacio. El programa recibe su argumento en letras minísculas únicamente.

25. Ampliar el programa anterior para traducir un mensaje de varias palabras. En la traducción a Morse cada letra está separada por un espacio y cada palabra con un `|`.

26. Escribir un programa en C que usando un único enunciado `printf` imprima los goles de la Selección Argentina en el Mundial de Catar como se muestra a continuación.

```
$ ./goles
22/11 - Gol de L. Messi a los 10 minutos!
26/11 - Gol de L. Messi a los 64 minutos!
26/11 - Gol de E. Fernandez a los 87 minutos!
30/11 - Gol de A. Mac Allister a los 46 minutos!
30/11 - Gol de J. Alvarez a los 67 minutos!
03/12 - Gol de L. Messi a los 35 minutos!
03/12 - Gol de J. Alvarez a los 57 minutos!
09/12 - Gol de N. Molina a los 35 minutos!
09/12 - Gol de L. Messi a los 73 minutos!
13/12 - Gol de L. Messi a los 34 minutos!
13/12 - Gol de J. Alvarez a los 39 minutos!
13/12 - Gol de J. Alvarez a los 69 minutos!
18/12 - Gol de L. Messi a los 23 minutos!
18/12 - Gol de A. Di Maria a los 36 minutos!
18/12 - Gol de L. Messi a los 108 minutos!
```

27. Escribir un programa que imprima tres *strings* representando una mano de truco aleatoria, por ejemplo: “Cuatro de copas”, “Ancho de espada”, “Tres de basto”.

28. Escribir un programa que declare un *array* de 40 posiciones representando un mazo de cartas para jugar al truco. El *array* debe estar ordenado por palo y número. Imprimir el mazo por consola.

29. Escribir un programa que use el mazo de cartas del ejercicio anterior para repartir tres cartas. Mezclar el mazo antes de repartir. Imprimir una cantidad *n* de manos donde *n* es un argumento del programa.

30. Escribir un programa que acepte un argumento p (probabilidad) y un argumento n. Crear una matriz que represente un tablero del juego Buscaminas de $n \times n$ marcando con un -1 las casillas con minas o cero de lo contrario. Usar la probabilidad p para determinar si un casillero tiene o no una mina.

31. Siguiendo el programa anterior, volver a recorrer el *array* para llenar los casilleros sin minas con el número de minas vecinas. Consideren usar un *array* de $(n + 2) \times (n + 2)$ para representar el tablero.

32. Escribir un programa que simule lo que se conoce como "caminos aleatorios". Usar una matriz de números enteros de 15×15 inicialmente llena de ceros. Poner un uno en el centro de la matriz e ir llenando con unos representando los pasos del "caminante". La caminata termina cuando se sale de la matriz original. El caminante puede dar un paso a la vez a la izquierda, derecha, arriba o abajo. Pero no puede volver sobre sus pasos. Imprimir la matriz que representa la caminata al terminar.

33. Escribir un programa en C que genere un Sudoku usando una matriz de 9×9. Poner los números iniciales al azar hasta que se genere un Sudoku válido. Usar una función para verificar si un Sudoku dado, completo o no, es válido.

34. Escribir un programa que declare una matriz (*array* bidimensional) de 3×3 de números enteros. Los elementos de la matriz se conocen en tiempo de compilación. Imprimir la matriz, una fila por línea separando con espacios los elementos.

35. Escribir un programa que acepte un argumento n y genere la matriz identidad de dimensión $n \times n$. Usar arrays bidimensionales.

36. Escribir un programa que declara una matriz de diez filas y cinco columnas. La matriz guarda en las primeras cuatro columnas números enteros entre 1 y 10 generados aleatoriamente en tiempo de ejecución. Representan notas de un alumno. Cada fila representa las notas de un alumno. La quinta columna de la matriz es igual al promedio de las cuatro notas en cada fila. El programa acepta dos argumentos, un valor semilla para `srand()` y el nombre de un alumno. Si el alumno se encuentra la lista el programa imprime si el alumno aprueba o no y con qué nota (promedio). El promedio debe ser un número de coma flotante. Si el alumno no está en la lista el programa imprime "Alumno no encontrado" y termina.

37. Escribir un programa que declare un array de enteros de 3×3 y guarde en cada elemento los primeros 9 argumentos recibidos por línea de comandos.

38. Escribir un programa que declare dos matrices de 3×3: A y B. Calcular y almacenar en una matriz C la suma de A y B. Al finalizar imprimir C.

39. Escribir un programa que calcule la traza de una matriz cuadrada. La traza es la suma de los elementos de la diagonal principal.

40. Escribir un programa que implemente la multiplicación de matrices cuadradas de $n \times n$. Dar valores a las matrices A y B en tiempo de compilación. Usar tres ciclos anidados.

41. Escribir un programa que declare y asigne valores a una matriz de 4×4. Imprimir el promedio de cada fila y cada columna.

42. Escribir funciones para calcular el determinante para matrices de 2×2 y 3×3.

43. Escribir un programa que genere una matriz aleatoria A de $m \times n$. Imprimir la matriz A y A^t (la matriz transpuesta de A).

44. Escribir un programa que calcule la matriz adjunta para matrices de 2×2.

45. Escribir un programa que calcule la matriz adjunta para matrices de 3×3.

46. Escribir un programa que calcule la inversa de una matriz de 2×2 con elementos enteros. Poner números aleatorios en la matriz. Si no es posible obtener la inversa imprimir un mensaje al usuario.

    Usar la fórmula

    $$A^{-1} = \frac{1}{\det A} \operatorname{adj} A$$

47. Escribir un programa similar al anterior pero para matrices de 3×3.

48. Escribir un programa que genere una matriz de 3×3 con números del 1 al 9 sin repetirse. Los números pueden aparecer en cualquier lugar de la matriz. Generar la matriz de manera aleatoria. Decidir si la matriz generada es un cuadrado mágico de 3×3. Un cuadrado mágico de 3×3 es un arreglo de los 9 números del 1 al 9, sin repetirse, tal que las filas, las columnas y las dos diagonales suman 15. Existen 8 combinaciones posibles.

49. Escribir un programa en Processing que use el *array* `pixels[]` y las funciones `loadImage()` e `image()`. Convertir la imagen mostrada a escala de grises.

50. Escribir un programa en Processing que cargue una imagen y realice una detección de bordes. Poner la imagen original y la filtrada lado a lado.

51. Completar los espacios en blanco.
    (a) Una matriz puede representarse con un *array* _____. Para recorrer una matriz usamos dos *loops* _____.
    (b) En un *array* tradicional como los del lenguaje C, el primer elemento se encuentra en el índice _____ y la longitud del mismo es _____ luego de su creación.
    (c) Para acceder a un elemento en un *array* usamos su _____.
    (d) Para recorrer un *array* de *n* elementos podemos usar `for (` _____`; i <` _____ `; i++)`.
    (e) Cuando procesamos *arrays* de dos dimensiones usamos la variable de control `i` para las _____ y `j` para las _____.

52. ¿Cuál es la manera correcta de acceder al tercer elemento de un *array* llamado "numeros" en C?
    - ☐ `numeros(2)`
    - ☐ `numeros[2]`
    - ☐ `numeros.3`
    - ☐ `numeros[3]`

53. ¿Cuál es el máximo para la dimensión de un *array* en C?
    - ☐ 10
    - ☐ 100
    - ☐ 10000
    - ☐ No hay un máximo.

54. ¿Qué le pasamos a una función cuándo usamos un *array* como argumento?
    - ☐ Los valores del *array*.
    - ☐ El índice del *array*.
    - ☐ La dirección de memoria del primer elemento del *array*.
    - ☐ El tamaño del *array*.

55. ¿Cuál es el índice del último elemento de un *array* de diez elementos?
    - ☐ 9
    - ☐ 10
    - ☐ 11
    - ☐ Depende del tipo de dato usado.

56. ¿Cuál de las siguientes es la manera correcta en C de inicializar un *array* llamado `a` y que todos sus elementos valgan 5?

    - ☐ `int a[] = {5};`
    - ☐ `int a[5] = {5};`
    - ☐ `int a[5]; a =  5;`
    - ☐ `int a[5] = {5, 5, 5, 5, 5};`

57. ¿Cuál de las siguientes expresiones sirven para encontrar el número de elementos de un *array* llamado `data` en C?

    - ☐ `sizeof(data) / sizeof(data[0])`
    - ☐ `sizeof(data) - 1`
    - ☐ `count(data)`
    - ☐ `data.length()`

58. ¿Cuál es la salida que produce el siguiente código?

```
char characters[] = {'H', 'o', 'l', 'a'};
printf("%s", characters);
```

    - ☐ Hola
    - ☐ H
    - ☐ Hol
    - ☐ No podemos saber.

59. ¿Cuántos elementos en total pueden almacenarse en un *array* declarado como `int a[3][4]`?

    - ☐ 3
    - ☐ 4
    - ☐ 12
    - ☐ 7

60. ¿Cuál es la salida que produce el siguiente código?

```
int nums[5] = {1, 2, 3, 4, 5};
printf("%d", nums[5]);
```

    - ☐ 0
    - ☐ 1
    - ☐ 5
    - ☐ No podemos saber.