1. Escribir un programa en C que reciba un argumento entero n y acepte n enteros por entrada estándar. Al terminar de leer los números imprimir su suma.
2. Escribir un programa en C que acepte n enteros por entrada estándar hasta encontrar un cero. Al terminar de leer los números imprimir su suma.
3. Escribir un programa en C que lea un número n entero por entrada estándar. Leer una línea con n enteros a continuación. Al terminar de leer los números, imprimir su suma.
4. Escribir un programa en C que sume todos los números enteros que ingresen por entrada estándar. Finalizar de leer números al encontrar `EOF` (*end of file*). Imprimir la suma.
5. Escribir un programa en C que cuente la cantidad de caracteres ingresados por entrada estándar.
6. Escribir un programa en C que reciba dos argumentos enteros a y b. El programa lee por entrada estándar un número arbitrario de enteros e imprime los que están entre a y b, es decir en el intervalo $[a, b]$.
7. Escribir un programa que envíe a salida estándar una cantidad n de números aleatorios entre a y b. Ingresar los parámetros n, a y b como argumentos de línea de comandos.
8. Escribir un programa que elija al azar un número entre uno y mil. Pedir al usuario que ingrese un número intentando adivinar. Indicar si el número correcto es menor o mayor al número ingresado. Darle al usuario un número limitado de preguntas, cinco por ejemplo. ¿Cuántas preguntas necesita el usuario para poder adivinar siempre el número?
9. Mejorar el programa anterior validando lo que ingresa el usuario. En primer lugar si el usuario ingresa un número $1000 < n < 1$ pedir que ingrese de nuevo un número válido. En segundo lugar chequear que el usuario ingrese un número válido, es decir caracteres que sean dígitos y no algo como “abc”.
10. Escribir un programa en C que cuente la cantidad de líneas en `stdin`.
11. Escribir un programa en C que “recorte” los llamados *whitespace characters*. Los carácteres que se consideran espacio en blanco son `'\n'` (nueva línea), espacio y `'\t'` (tabulación). Hay otros pero son más raros de encontrar. Con recortar nos referimos a eliminar el espacio en blanco y copiar la entrada ya “recortada” a `stdout`.
12. Escribir un programa en C que imprima un menú con cinco opciones y una sexta para salir del programa. El programa imprime el menú y espera la entrada del usuario para elegir una de las seis opciones, imprime algo distinto según la opción y repite el menú.
13. Escribir un programa en C que funcione como el comando `cat` de Unix.
14. Escribir un programa en C que funcione como el comando `cp` de Unix.
15. Escribir un programa en C que funcione como el comando `nl` de Unix.
16. Escribir un programa en C que funcione como el comando `head` de Unix.
17. Escribir un programa en C que funcione como el comando `tail` de Unix.
18. Escribir un programa en C que funcione como el comando `wc` de Unix.
19. Escribir un programa en C para jugar al ahorcado por línea de comandos. Usar un archivo de texto con una palabra por línea como diccionario para el juego.
20. Escribir un programa en C para jugar al Tatetí. Indicar quién ganó o si hubo un empate.
21. Escribir un programa en C para jugar al Wordle. Usar un archivo de texto con una palabra por línea como diccionario para el juego.
22. Escribir en C un clon del Buscaminas. Leer por entrada estándar el número de filla, el número de columna y la acción (poner una bandera o descubrir el casillero).

23. Escribir un programa que lea un archivo por entrada estándar con un formato predefinido que consiste en `nombre nota_1 nota_2` dónde `nombre` es un *string* y `nota_1` y `nota_2` son enteros. Tabular por salida estándar los datos que ingresan agregando una cuarta columna con el promedio de las dos notas.

24. Escribir un programa en C para tomar notas. Usar un menú para elegir entre tomar notas, ver notas, borrar notas y salir. Almacenar las notas en un archivo de texto.

25. Escribir un programa en C que lea un archivo de texto en formato CSV y tabule los datos encontrados.

26. Escribir un programa en C que lea un archivo de texto en formato CSV y genere un enunciado `INSERT` para cargar los datos en una tabla de MySQL. Pasar dos argumentos al programa, el nombre de la tabla y el nombre del archivo.

27. Escribir un programa en C que reciba como argumento el nombre de un archivo de texto y un número n. Cifrar los contenidos del archivo usando rotación módulo n y escribir el resultado en un nuevo archivo con el nombre `out.txt`.

28. Escribir un programa en C que lea un archivo de texto con números al azar entre 1 y 10. Imprimir un histograma de los datos leídos.

29. Escribir un programa en C que acepte un argumento entero n y genere n oraciones aleatorias y las escriba en un archivo `out.txt`. Las oraciones no tienen que tener ningún sentido en particular. Pero en lo posible las oraciones generadas tienen que ser gramaticales.

30. Implementar en C un programa que reciba dos argumentos $n, m \in \mathbb{N}$. El programa genera una secuencia de n enteros aleatorios entre 0 y m. Enviar dicha secuencia a salida estándar separando cada número con un espacio.

    Conectar por medio de un *pipe* la salida de el primer programa a un segundo programa. Este programa recibe como argumento el número m (el mismo que el del programa anterior) y cuenta la cantidad de veces que aparece cada número en su entrada.

    Si interpretamos cada número como una figurita posible en el álbum del mundial el programa debe parar cuando hayamos completado el álbum, ignorando el resto de la entrada si es que hubiera. Informar al final la cantidad de figuritas que hicieron falta para completar el álbum y cuántas tenemos de cada una.

```
$ ./figus 50 20 | ./completar 20
no complete el album y compre 50 figus
figu →  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
cant →  3  1  4  0  1  5  5  3  1  0  4  2  2  2  3  2  5  5  1  1

$ ./figus 50 10 | ./completar 10
complete el album y compre 19 figus
figu →  0  1  2  3  4  5  6  7  8  9
cant →  2  4  1  1  2  1  2  3  1  2
```

31. Escribir un programa en C que reciba como argumento el nombre de un archivo de texto que contiene un número arbitrario de líneas. Cada línea puede contener letras del abecedario, tanto mayúsculas como minísculas. Cada vez que en una línea haya una secuencia con dos o más letras iguales reemplazar la secuencia por la letra y el número de veces que aparece. Por ejemplo si una línea es `AAaaaabcd` reemplazarla por `A2a4bcd`. Enviar el resultado de este proceso a salida estándar.

32. Escribir un programa que acepte de entrada estándar un número arbitrario de líneas conteniendo letras minísculas y mayúsculas. Invertir el proceso del programa anterior enviando a salida estándar el resultado de la descompresión. El resultado de conectar ambos programas con un *pipe* debe ser el archivo de texto original.

33. Escribir un programa que lea un archivo CSV (*comma separated values*) con los siguientes datos: Grupo, PJ, G, E, P, GF, GC, DG y Pts. Todos son números enteros y corresponden a los datos de los partidos de fase de grupos del Mundial 2022. El orden de las líneas en el archivo corresponde al orden de las 8 selecciones que llegaron a cuartos de final en orden alfabético.

    Almacenar los datos leídos en una matriz de 8×9. Imprimir la tabla tal como se cargó en la matriz agregando una columna al principio con el nombre del país.

34. Escribir un programa en C para realizar conversiones de temperatura de Fahrenheit a Celsius y viceversa. El programa debe aceptar como entrada un número y una letra: "F" o "C" y convertir la temperatura ingresada a Celsius si el usuario ingresó "F" o Fahrenheit si ingresó "C".

    Al finalizar la conversión preguntar al usuario si desea realizar otra operación o salir del programa.

35. ¿Cuáles son los argumentos de `fgets()`?

    - ☐ `fgets(char buffer, int bufsize, stdin)`
    - ☐ `fgets(char *buffer, int bufsize, FILE *fp)`
    - ☐ `fgets(char *s, int size, stdin)`
    - ☐ `fgets(char buffer[], FILE *stream)`

36. ¿Cuál es el primer argumento de `fscanf()`?

    - ☐ El *string* de formato que indica la conversión.
    - ☐ Un *buffer* donde se va a guardar el *string* leído.
    - ☐ El *stream* del cuál se va a leer, un `FILE *`.
    - ☐ Un variable de tipo `char` donde se va a guardar el dato leído.

37. ¿Qué número es el descriptor de archivo de `stderr`?

    - ☐ 0
    - ☐ 1
    - ☐ 2
    - ☐ 3

38. ¿Cuál de las siguientes funciones escribe a salida estándar?

    - ☐ `sprintf`
    - ☐ `fgets`
    - ☐ `puts`
    - ☐ `sscanf`

39. ¿Qué función de `stdio.h` se usa para leer un único caracter desde *standard input*?

    - ☐ `getchar()`
    - ☐ `scanf()`
    - ☐ `fgetc()`
    - ☐ `putc()`

40. ¿Cuál de las siguientes funciones usarías para abrir un archivo para lectura en C?

    - ☐ `fopen("myfile.txt", "w")`
    - ☐ `fclose()`
    - ☐ `fopen("myfile.txt", "r")`
    - ☐ `fwrite()`