1. Escribir una función `void mas_dos(int x)` que reciba una variable *x* y le sume dos. Escribir una función `void mas_dos(int a[], int n)` que haga lo mismo pero con cada elemento del *array* `a[]`. ¿Qué diferencias encuentran entre ambas funciones?

2. Escribir una función `void swap(int a, int b)` en C que intercambie dos variables enteras. Hacer una función similar pero para un *array*. La función `void swap(int a[], int i, int j)` toma un *array* y dos índices. ¿Qué diferencias encuentran entre ambas funciones?

3. Implementar en C una función similar a `strcpy()` de `string.h`.

4. Implementar en C una función similar a `strcat()` de `string.h`.

5. ¿Cuál es la salida que producen las llamadas a `printf` del siguiente programa?

```c
#include <stdio.h>

int a = 10;

void f(int a) {
  a += 2;
  printf("%d\n", a);
}

int main(void) {
  f(a);
  printf("%d\n", a);
  f(66);
  a += 2;
  printf("%d\n", a);
  f(a+2);
}
```

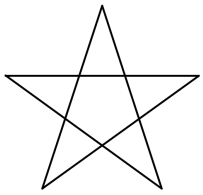
6. Implementar en C una librería para crear y manipular *arrays* de enteros con las siguientes funciones:
   (a) `void create_array(int size, int buffer[])`
   (b) `void create_array_with_value(int size, int buffer[], int value)`
   (c) `void create_random_array(int size, int buffer[], int lo, int hi)`
   (d) `void create_matrix(int n, int m, int buffer[][m])`
   (e) `void create_matrix_with_value(int n, int m, int buffer[][m], int value)`
   (f) `void create_random_matrix(int n, int m, int buffer[][m], int lo, int hi)`
   (g) `void print_array(int size, int buffer[])`
   (h) `void print_matrix(int n, int m, int buffer[][m])`
   (i) `void read_array(int size, int buffer[])`
   (j) `void read_matrix(int n, int m, int buffer[][m])`
   (k) `void fread_array(int size, int buffer[], FILE *fp)`
   (l) `void fread_matrix(int n, int m, int buffer[][m], FILE *fp)`
   (m) `void shuffle_array(int size, int buffer[])`
   (n) `void shuffle_matrix(int n, int m, int buffer[][m])`

7. Escribir en Processing una función `void turtle(char command)` para implementar lo que se conoce como *turtle graphics*. El argumento de tipo `char` puede ser uno de tres comandos: `'F'` por *forward* o adelante, `'R'` y `'L'` por *right* y *left* respectivamente (girar 90 grados a la derecha o a la izquierda). Usar la función `line()` de Processing para ir dibujando el camino de la tortuga.

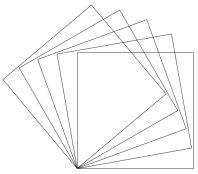
   También es conveniente tener una función `void rotateTurtle(int angle)` para girar a la tortuga con mayor precisión que los comandos *right* y *left*.

   Trabajar con variables globales para la posición, dirección y distancia de cada paso de la tortuga.

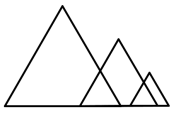
8. Realizar el siguiente dibujo usando gráficas tortuga.



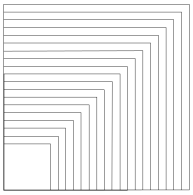
9. Realizar el siguiente dibujo usando gráficas tortuga.



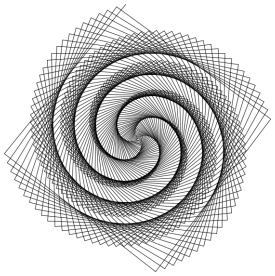
10. Realizar el siguiente dibujo usando gráficas tortuga. Agregar una función para que la tortuga avance un número determinado de pasos o píxeles.



11. Realizar el siguiente dibujo usando gráficas tortuga.



12. Realizar el siguiente dibujo usando gráficas tortuga.



13. Escribir una versión recursiva en C de la función `int sum(int n)` que devuelva la suma de los primeros n números naturales. Comparar esta función con su versión iterativa. ¿Existe alguna forma más eficiente de escribir esta función?

14. Escribir una versión recursiva en C de la función `int factorial(int n)`.

15. Escribir una función recursiva que devuelva la suma de todos los dígitos de un número n.

16. Escribir una función recursiva en C `int power(int a, int n)` que devuelva a^n.

17. Escribir una versión recursiva en C de la función `int fibonacci(int n)` que devuelva el enésimo número de Fibonacci.

18. Escribir una función recursiva en C para invertir los dígitos de un número entero.

19. Escribir una función recursiva en C que reciba un entero n en binario y devuelva su equivalente en código Gray.

20. Escribir una función recursiva en C para pasar un número en decimal a binario.

21. Usando el algoritmo de Euclides, escribir una función recursiva en C `int mcd(int a, int b)` que devuelva el máximo común divisor de a y b.

22. Escribir la función `int catalan(int n)` que devuelva el enésimo número de Catalan. Los primeros doce números de Catalan son 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786. Estos números satisfacen la siguiente relación recursiva.

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \quad \text{y} \quad C_0 = 1$$

23. Escribir una función recursiva en C para imprimir un *array*.

24. Escribir una función recursiva en C que devuelva la suma de todos los elementos en un *array*.

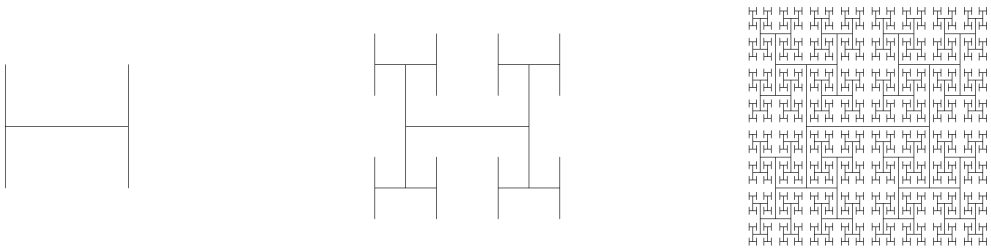
25. Escribir una función recursiva en C que devuelva el promedio de los elementos en un *array* de enteros.

26. Escribir una función recursiva en C `void hanoi(int n, int start, int end)` que imprima la secuencia de pasos para mover n discos del poste *start* al poste *end* (los tres postes del juego están numerados del 1 al 3) en el juego de ingenio conocido como "Las torres de Hanoi".

27. Implementar en C una función recursiva `void collatz(int n)` que imprima la secuencia de números que resulta de aplicar las operaciones de la conjetura de Collatz a n hasta llegar al número 1. La conjetura de Collatz dice que dado un número entero positivo n se puede transformar en el número 1 aplicando dos operaciones recursivamente. Si n es par dividirlo por dos y si es impar el próximo término es $3n + 1$.

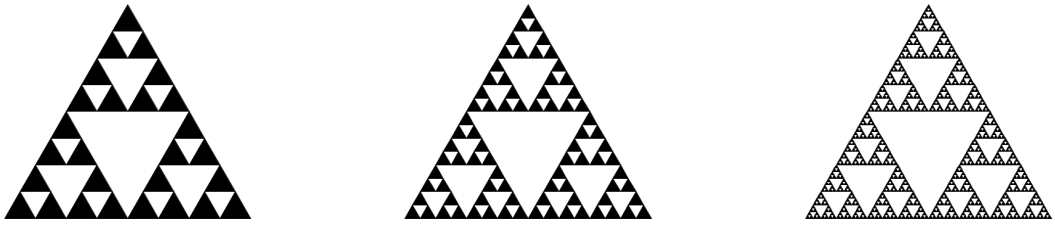
28. Escribir una función recursiva en C que devuelva el número de caminos posibles para recorrer una grilla de $n \times m$ yendo desde la esquina superior izquierda a la esquina inferior derecha. Restricciones: solo podemos movernos de izquierda a derecha o de arriba hacia abajo una unidad a la vez.

29. Escribir una función recursiva en Processing para dibujar un árbol H de orden n.



Ejemplos con $n = 1$, $n = 2$ y $n = 5$

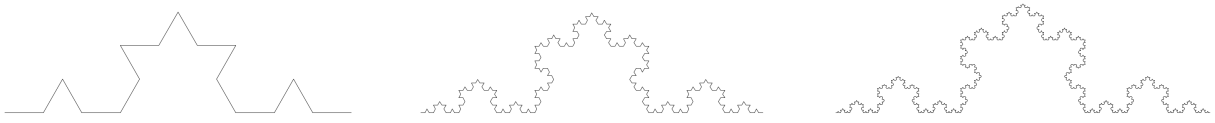
30. Escribir una función recursiva en Processing para dibujar el triángulo de Sierpinski. Usar un argumento n para controlar el orden de la recursión.



Ejemplos con $n = 3$, $n = 4$ y $n = 6$

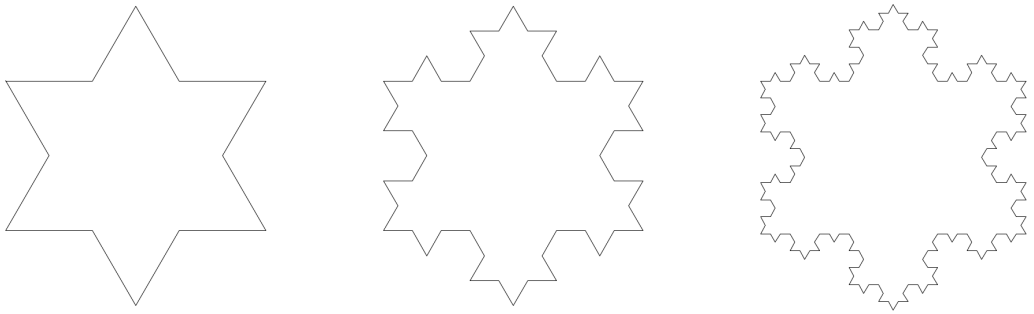
31. Escribir un programa en Processing para dibujar la espiral de Fibonacci usando recursión.

32. Escribir una función recursiva en Processing para dibujar la curva de Koch. Usar gráficas tortuga. Usar un argumento n para controlar el orden de la recursión.



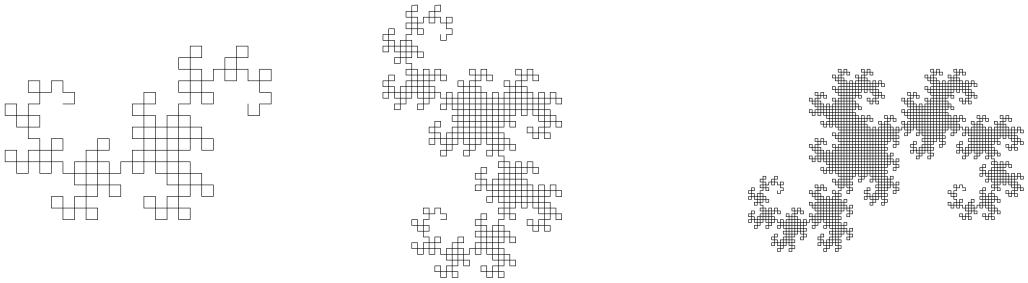
Ejemplos con $n = 2$, $n = 4$ y $n = 5$

33. Escribir una función recursiva en Processing para dibujar el copo de nieve de Koch. Usar gráficas tortuga. Usar un argumento *n* para controlar el orden de la recursión.



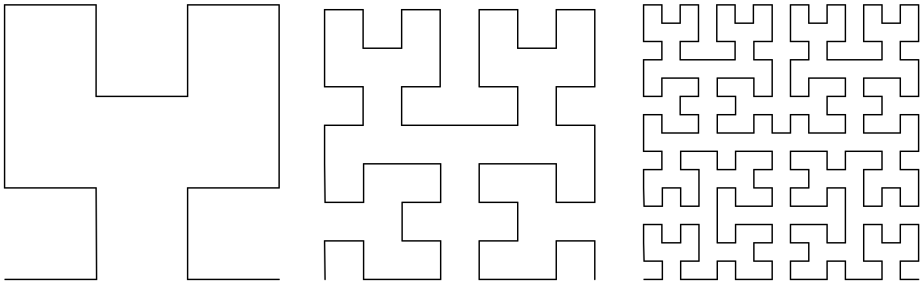
Ejemplos con $n = 1$, $n = 2$ y $n = 3$

34. Escribir una función recursiva en Processing para dibujar la curva dragón. Usar gráficas tortuga. Usar un argumento *n* para controlar el orden de la recursión.



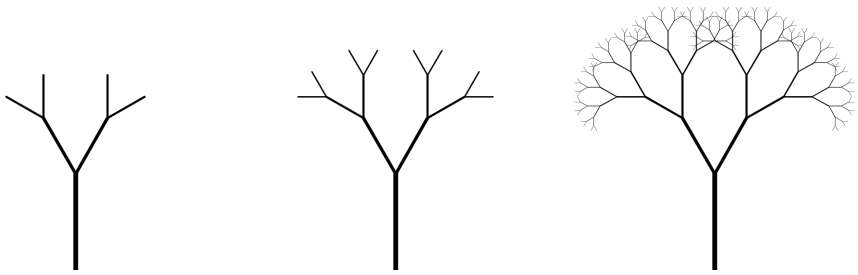
Ejemplos con $n = 8$, $n = 10$ y $n = 12$

35. Escribir una función recursiva en Processing para dibujar la curva de Hilbert. Usar gráficas tortuga. Usar un argumento *n* para controlar el orden de la recursión.



Ejemplos con $n = 2$, $n = 3$ y $n = 4$

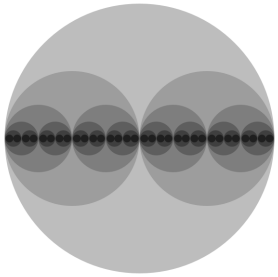
36. Escribir una función recursiva en Processing para dibujar el "Árbol Y" de orden n.



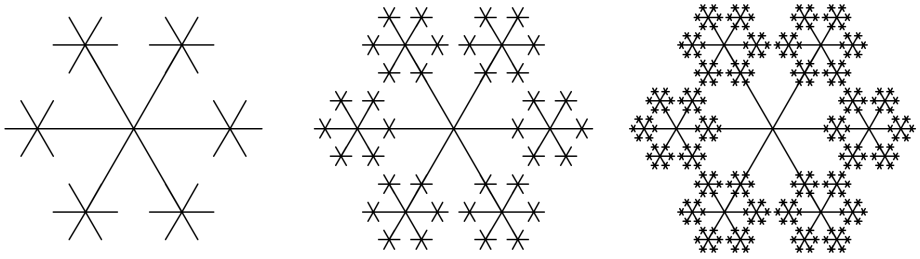
Ejemplos con $n = 3$, $n = 4$ y $n = 8$

37. Escribir una función recursiva en Processing para dibujar el conjunto de Cantor. Usar un argumento n para controlar el orden de la recursión.

38. Escribir una función recursiva en Processing que produzca el siguiente dibujo.



39. Escribir una función recursiva en Processing que produzca el siguiente dibujo similar a un copo de nieve. Usar un argumento n para controlar el orden de la recursión.



Ejemplos con $n = 2$, $n = 3$ y $n = 4$

40. Realizar una animación en Processing demostrando la solución del ejercicio 26 (Torres de Hanoi) para cinco discos.