

3.4 Descomposición de un número



En este ejercicio proponemos la definición de funciones sencillas que permitan descomponer un número de múltiples formas.

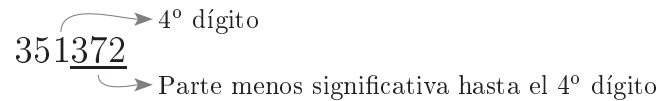
Parte más significativa Escribe una función que devuelva la parte más significativa desde el n -ésimo dígito de un número.

Ejemplo



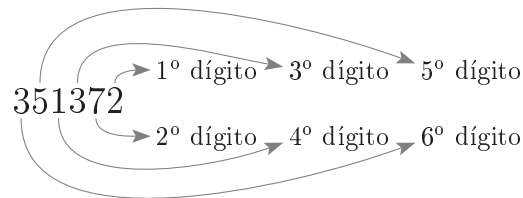
Parte menos significativa Escribe una función que devuelva la parte menos significativa hasta el n -ésimo dígito de un número.

Ejemplo



Dígito n -ésimo Escribe una función que nos devuelva el dígito n -ésimo de un número.

Ejemplo



Cualquier base Escribe funciones que permitan realizar los apartados anteriores sea cual sea la base con la que estemos trabajando.

Para este enunciado Consulta la pista 3.4a.

3.5 Palíndromos



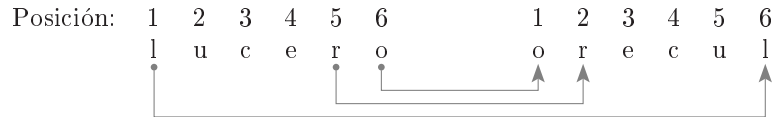
Un palíndromo es una palabra o frase que se lee igual de izquierda a derecha que de derecha a izquierda; por ejemplo, la palabra “anilina”, o las frases “anula la luz azul a la luna” y “sé verla al revés”, despreciando las diferencias entre mayúsculas y minúsculas, los espacios en blanco entre palabras y las tildes.

De igual forma, decimos que la expresión de un número en una determinada base (véase el ejercicio 2.21) es un palíndromo, o capicúa, si dicho número se lee igual de izquierda a derecha que de derecha a izquierda, por ejemplo 81318 en base 10, o el número nueve expresado en base 2, 1001. Obviamente, suponemos que los números siempre están expresados por su representación mínima, es decir, sin ceros redundantes. Por ejemplo, 100 en base 10 no es un palíndromo, ya que no se lee igual de derecha a izquierda de izquierda a derecha; no consideramos expresiones de 100 como 00100, que en principio también son el mismo número, ya que entonces podríamos decir que sí que es un palíndromo.

3.5.1 Reverso de caracteres

Escribe una función que tenga como parámetro de entrada una cadena de caracteres y devuelva su reverso.

Ejemplo El reverso de una cadena de caracteres es otra cadena pero cuyas letras están en orden inverso de aparición.



O sea, el reverso de la palabra *lucero* tiene por primera letra la *o*, que es la última de *lucero*, por segunda letra a *r* que es la penúltima, y así sucesivamente.

3.5.2 Comprobación de palíndromo

Escribe una función que tenga como parámetro de entrada una cadena de caracteres y nos indique si es un palíndromo o no. Observa que, si se trata de una frase, los espacios que separan las palabras no se tienen en cuenta.

3.5.3 Reverso de un número

Escribe una función que tenga como parámetro un número entero y devuelva el reverso de dicho número. (Véase la pista 3.5a.)

3.5.4 Número palíndromo

Escribe una función que tenga como parámetro de entrada un número y nos indique si éste es palíndromo o no.

3.6 Cuentalettras



Un *cuentalettras* es un programa que recibe como entrada una secuencia de palabras y va contando y escribiendo el número de letras de cada una de ellas. En este caso, queremos que el cuentalettras haga su trabajo con el contenido de un fichero de texto.

Supongamos que en un fichero de texto tenemos almacenado el siguiente poema de Manuel Golmayo:

*Soy y seré a todos definible.
Mi nombre tengo que daros;
cociente diametral siempre inmedible
soy de los redondos aros*

el cuentalettras devolvería como resultado: 31415926535897932384.

¿No te resulta familiar esta secuencia de dígitos? Efectivamente, son los 20 primeros dígitos del número π . ¿Qué te parece la siguiente reflexión en la lengua shona de Zimbabue?:

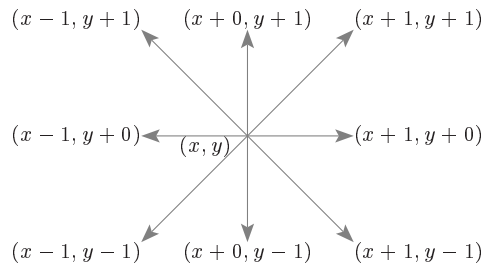
*Iye “P” naye “I” ndivo vadikanwi.
“Pi” achava mwana.*

Su autor es Martin Mugochi de la Universidad de Zimbabue y su traducción más o menos es: “*P*” e “*I*” son amantes. “*Pi*”, su hijo, será un cerebritito. Ante esta frase el cuentalettras debería responder: 314159265. ¡Qué casualidad!

Si quieres encontrar más π nemotécnicos y π emas puedes consultar la página: <http://www.cilea.it/~bottoni/www-cilea/F90/piph.htm>.

Proponemos la realización de programas que permitan estudiar los movimientos de una partícula dentro de una determinada superficie geométrica.

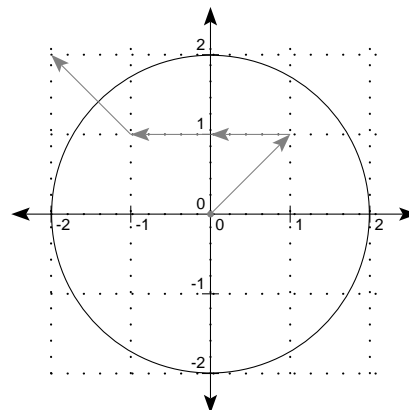
Movimientos dentro de un círculo Supongamos que tenemos una partícula que se mueve de forma aleatoria en el plano en las coordenadas enteras. La partícula puede pasar con la misma probabilidad a cualquiera de las posiciones en las que una o ambas de sus coordenadas actuales varíe de forma entera en -1 o 1 , como muestra el siguiente gráfico:



Queremos realizar un programa que, dado un círculo y una partícula en el interior de dicho círculo, calcule el número de movimientos que realiza la partícula antes de salirse del círculo.

Ejemplo Consideremos un círculo de radio 2 centrado en el origen. Queremos ver cuántos movimientos realiza una partícula, que inicialmente se halla situada en el origen y que se mueve de forma aleatoria, antes de salirse del círculo.

En el siguiente gráfico podemos ver una secuencia aleatoria de movimientos de la partícula:



La partícula parte del origen de coordenadas $(0,0)$, y realiza el movimiento aleatorio, que consiste en elegir dos números entre -1 y 1 que determinarán el movimiento de cada coordenada, en este caso $[1,1]$. La partícula se mueve y por tanto llega a la posición $(0 + 1, 0 + 1) = (1, 1)$. El segundo movimiento de la partícula viene dado por la elección de los números $[-1,0]$ y por tanto la posición final después de este segundo movimiento es $(1 - 1, 1 - 0) = (0, 1)$. La partícula realiza un tercer movimiento $[-1,0]$ que la lleva a la posición $(0 - 1, 1 - 0) = (-1, 1)$. Finalmente, el cuarto movimiento $[-1, 1]$ lleva a la partícula fuera del círculo, y ahí termina el experimento. Se han necesitado tres movimientos. (Véase la pista 3.7a.)

Media de movimientos Considera el apartado anterior y modifica ligeramente la solución para que podamos hacer, no un único experimento, sino la cantidad que deseemos, y así con ellos calcular la media de movimientos necesarios para salirse de un círculo con un determinado radio.

Otras figuras geométricas de dos dimensiones Escribe un programa general para movimientos en el plano que permita utilizar otras figuras geométricas para delimitar las superficies dentro de las cuales se mueven las partículas. Por ejemplo cuadrados, rectángulos, triángulos, etc.

Movimiento dentro de un cubo Escribe un programa que, dado un cubo y una partícula en su interior, calcule el número de movimientos que la partícula realiza antes de salirse del cubo.

Otras figuras geométricas de tres dimensiones Escribe un programa general para movimientos en el espacio de tres dimensiones que permita utilizar diversas figuras geométricas para delimitar las regiones. Por ejemplo esferas, cilindros, poliedros, etc.

Para este enunciado Consulta la pista 3.7b.

Bibliografía El movimiento aleatorio de partículas se conoce como *movimiento browniano* y tiene interés en distintas áreas, desde la física a los modelos estadísticos de inventarios. Con la ayuda de [Tho00], por ejemplo, puedes ampliar tus conocimientos sobre los modelos estadísticos subyacentes.

Un poco de historia En 1637 René Descartes (1596–1650) publicó *La Géométrie*, donde presenta la geometría analítica, de importancia fundamental para el desarrollo posterior del cálculo por parte de Isaac Newton (1643–1727) y Gottfried Wilhelm Leibniz (1646–1716).

Descartes, matemático, filósofo y científico, escribió en 1637 *El discurso del método* (cuyo subtítulo era, *para bien dirigir la razón y buscar la verdad en las ciencias*), trabajo que marcó de forma decisiva la transición de la ciencia y la filosofía medievales a la edad moderna.

38 Generación de primos

Considera las siguientes propiedades relacionadas con la primalidad de un número:

- Un entero superior a 2 sólo puede ser primo si es impar.
- Un entero n superior a 3 sólo puede ser primo si verifica la propiedad $n^2 \bmod 24 = 1$.
- Un entero positivo n es primo si y sólo si no tiene divisores entre 2 y $\lfloor \sqrt{n} \rfloor$.

y, basándote en ellas, escribe las siguientes funciones:

Filtro: “mod24-1” Una función que indique si un entero verifica la propiedad siguiente:

$$n^2 \bmod 24 = 1$$

Filtro: divisores Una función que indique si un número es primo, buscando si tiene algún divisor entre 2 y $\lfloor \sqrt{n} \rfloor$.

Filtro: primos Una función que indique si un número es primo o no, descartando primero los pares, comprobando luego la propiedad “mod24-1,” y finalmente buscando sus posibles divisores.

Bibliografía La literatura sobre los números primos es muy abundante. La referencia [Pom83] es un interesante punto de partida para ampliar información sobre este tema.

39 Conjetura de Goldbach

En una carta escrita a Leonhard Euler en 1742, Christian Goldbach (1690–1764) afirmó (sin demostrarlo) que todo número par es la suma de dos números primos. Para poner a prueba esta conjetura (hasta cierto punto, claro está), basta con avanzar a través de los primeros n números pares hasta encontrar uno que no verifica esa propiedad, o hasta llegar al último, ratificándose la conjetura hasta ese punto,

```

int k = 0; bool seCumpleHastaK = true;
while (seCumpleHastaK && k <= n) {
    k = <el siguiente par>;
    <Tantear la descomposición de k>
    if (<falla el intento>) {
        <seCumpleHastaK se anota como falso>
    }
}

```

donde cada tanteo se puede expresar mediante un subprograma que responda a la siguiente llamada,

```
descomponer(numPar, conseguido, sumando1, sumando2);
```

esto es, un subprograma que, dado un entero `numPar` (supuestamente positivo y par), busca una descomposición del mismo en dos sumandos `sumando1` y `sumando2` primos, indicando además si lo ha conseguido o no. Esa descomposición de `numPar` se busca probando pares de sumandos,

$$(1, \text{numPar} - 1), (2, \text{numPar} - 2), \dots$$

hasta que ambos sean primos o bien el primero supere al segundo, para no repetir los tanteos finales,

$$\dots, (\text{numPar} - 2, 2), (\text{numPar} - 1, 1)$$

que son iguales a los iniciales.

Descomposición Escribe en C++ el subprograma anterior, suponiendo que existe una función que verifique la primalidad de un número. (De hecho, se ha desarrollado en el ejercicio 3.8.)

Conjetura de Goldbach Finalmente, desarrolla el programa correspondiente al algoritmo completo, siguiendo los pasos descritos.

Bibliografía Esta conjetura aparece recogida en las *Meditaciones algebraicas* de Edward Waring (1734–1793) junto con otros resultados interesantes. Te proponemos desarrollar un programa que permita comprobar los siguientes enunciados (pruébalos con los primeros millares de los números naturales):

- Conjetura: todo entero impar es un número primo o la suma de tres números primos.
- Teorema (Leonhard Euler, 1707–1783): todo entero positivo es la suma de, a lo más, cuatro cuadrados.
- Teorema (John Wilson, 1741–1793): para todo primo p , el número $(p - 1)! + 1$ es múltiplo de p .

En la novela [Dox00] se relata la historia de un matemático que únicamente vivió para intentar demostrar la conjetura de Goldbach.

310 El factorial en la sociedad del futuro



Ahora hay mucha policía; nos aseguran que es necesaria. Mi abuela me ha contado que antes sólo existían dos o tres policías, distintas pero iguales; ahora hay una sola, pero con tantas personas que necesitan vigilarse mutuamente. Por eso está organizada jerárquicamente, y los de un nivel vigilan a los del siguiente hasta que se llega a la ciudadanía. Nadie conoce el primer nivel; no se sabe cuántas personas lo forman; dicen que se llama CIA y que vigila al siguiente nivel. El segundo nivel se llama BICIA; tiene una sola persona (dicen que es un rey) que vigila a las dos personas del siguiente nivel. El tercer nivel se llama TRICIA; cada uno de sus dos miembros vigila a tres personas en el siguiente nivel. Así se sigue, por varios niveles más de los que poca gente conoce el nombre; siempre se cumple que una persona del nivel n vigila

a $n + 1$ personas del siguiente nivel, y que a cada persona del nivel $n + 1$ sólo la vigila una del nivel n ; se deduce que en cada nivel hay $(n - 1)!$ personas. Al último nivel lo llamamos POLICIA porque no sabemos exactamente a cuántas personas tocan. Las personas que quedamos como ciudadanos, aunque se está extendiendo el término CIADAANO. Mi abuela dice que antes se podía trabar conversación con cualquier persona desconocida; que antes los policías tenía un carnet y un traje especial. Ahora no es así; sólo hay una documentación, no hay una ropa especial. En busca de la prima de productividad, la policía para intempestivamente a la gente por la calle. En un principio hubo muchos altercados y tiroteos entre la misma policía; luego se impuso la costumbre de gritar el nivel para que los demás supiéramos por quién tomar partido; pero ahora, para hacerlo todo más sutil, se grita el número de la documentación. Afortunadamente la documentación está numerada consecutivamente por niveles; es fácil hacerse una idea de quién manda más, pero como los niveles son tan grandes, es difícil saber si dos números son del mismo nivel o los separan muy pocos niveles. El otro día me pararon el agente número 47335 y la agente número 80157; me trataron de muy malos modos; no sabía que sólo estaban un nivel por encima de mí y que les podía haber interpuesto una denuncia. Ahora voy a escribir un programa en mi microcomputadora portátil para auxiliarme la próxima vez; le daré dos números de documentación y me dirá cuántos niveles los separan.

3 1 1 Sucesión bicicleta



Consideremos las sucesiones cuyos términos están definidos del siguiente modo:

$$b_1, b_2 \in \mathbb{R}^+$$

$$b_n = \frac{b_{n-1} + 1}{b_{n-2}}, \quad \forall n \geq 3$$

Al igual que ocurre con la sucesión de Fibonacci se observa que, para avanzar a través de sus términos, basta con hacer *rodar* dos de ellos consecutivos. Por ejemplo, si tomamos los dos primeros términos 2 y 3, se obtienen los siguientes, 2, 3, 2, 1, 1, 2, 3, ... que, sólo por casualidad, son todos números enteros. Por otra parte, es fácil verificar que, para dos términos iniciales cualesquiera de \mathbb{R}^+ , la sucesión generada mediante la relación recurrente anterior es cíclica,

$$\forall b_1, b_2 \in \mathbb{R}^+, \exists p \in \mathbb{N} \text{ tal que } \forall n \geq 1, b_{n+p} = b_n$$

por lo que muy bien podría ser llamada *bicicleta*.

Periodo Escribe una función que calcule el período p de una sucesión dada por sus dos primeros elementos.

Serie Escribe ahora un programa que halle la suma de los n primeros términos de dicha sucesión.

3 1 2 Cotejo de n -gramas



Si quisiéramos determinar si dos palabras son iguales, podríamos plantearnos comparar uno a uno sus caracteres y, si todos coinciden, concluir que las palabras son *iguales*. Sin embargo, este mecanismo sólo nos indica si dos palabras o frases son iguales o no pero, en el caso de que no lo sean, no nos dice en qué medida son diferentes.

Una primera idea para determinar el grado de similitud de dos palabras puede ser contar el número de caracteres que tienen en común. Por ejemplo, las palabras *listo* y *clínico* comparten 2 caracteres. No obstante, esta idea no nos aporta información sobre si la distribución de esos 2 caracteres comunes es similar en las dos palabras. Para obtener este tipo de información podemos recurrir a compararlas cotejando n -gramas. Veamos de qué se trata.

Un n -grama es una secuencia de n caracteres. Si nos referimos a una secuencia de dos caracteres podemos hablar de bigrama; si la secuencia es de tres caracteres, trigramas. Comprobemos con el ejemplo anterior cuál es el resultado si comparamos las palabras mediante bigramas. Compararíamos los bigramas de *listo* (*li, is, st, to*) con los de *cínico* (*ci, in, ni, ic, co*) para concluir que no tienen ningún bigrama en común; luego los 2 caracteres comunes no dan lugar a ninguna secuencia de dos caracteres consecutivos en común.

Otro ejemplo puede ilustrar aún más la diferencia entre ambas estrategias. Las palabras *loco* y *comida* comparten también 2 caracteres y, además, 1 bigrama. Es decir, no sólo tienen caracteres en común sino que éstos tienen un cierto grado de similitud en su distribución en ambas palabras.

N -gramas Escribe un programa que dadas dos palabras las coteje utilizando n -gramas e indique, además, el número de caracteres que tienen en común. El valor de n lo propondrá el usuario teniendo en cuenta que no debería ser mayor que el número de caracteres de la palabra más corta.

Coefficiente de Dice Vamos a añadir una medida de similitud: el coeficiente de Dice. Este coeficiente se utiliza a menudo para determinar la similitud entre elementos con partes constituyentes. Se calcula así,

$$CD(p_1, p_2) = \frac{2 \times \text{comun}(p_1, p_2)}{\text{elementos}(p_1) + \text{elementos}(p_2)} \in [0, 1]$$

siendo $\text{comun}(p_1, p_2)$ el número de partes constituyentes (n -gramas o caracteres según la aproximación que utilicemos) que p_1 y p_2 tienen en común, y $\text{elementos}(p_1)$, $\text{elementos}(p_2)$ el número de partes constituyentes de p_1 y p_2 respectivamente.

La fórmula anterior proporciona un valor real en el rango $[0, 1]$. Si las dos palabras no tienen ningún n -grama en común obtendrán un coeficiente de Dice igual a 0, mientras que dos palabras idénticas tendrán un coeficiente de Dice igual a 1.

Amplía el programa desarrollado en el primer apartado para que incluya el cálculo del coeficiente de similitud en las dos estrategias que hemos considerado: caracteres y n -gramas.

Bibliografía El coeficiente de similitud de Dice [Dic45] fue ideado para medir el grado de asociación entre especies; sin embargo se utiliza con profusión en algunas tareas de procesamiento de lenguaje natural, como determinar el grado de asociación entre términos, oraciones o incluso documentos. También se utiliza para medir la similitud entre dos mapas de bits de igual longitud, comparar grafos conceptuales, moléculas, etc.

3.13 Simulación de variables aleatorias

Como todos los hombres en Babilonia, he sido próconsul; como todos, esclavo; también he conocido la omnipotencia, el oprobio, las cárceles. Miren: a mi mano derecha le falta el índice. [...] He conocido lo que ignoran los griegos: la incertidumbre. [...] Debo esta variedad casi atroz a una institución que otras repúblicas ignoran o que obra en ellas de un modo imperfecto: la lotería.

La lotería en Babilonia, Jorge Luis Borges

3.13.1 Introducción

Casi todos los lenguajes de programación proporcionan recursos para simular variables aleatorias. Normalmente, dichos recursos consisten en funciones para generar variables aleatorias uniformes, ya que con ellas se puede simular cualquier otra.

Aquí se resumen los recursos mínimos que proporciona C++, y se explica cómo manejar variables uniformes para construir otras. Justamente eso es lo que se propone al final: usar los recursos dados para implementar distintas variables aleatorias.

3.13.2 Los recursos en C++

En C++, existen distintos juegos de recursos para generar números aleatorios: unos son estándar y otros no; unos tienen más limitaciones que otros; algunos tienen un uso más cómodo que otros. Sin mayores discusiones, consideraremos la colección de funciones que tiene `stdlib.h` como fichero de cabeceras.

La función `int rand()` genera un número pseudoaleatorio entre 0 y `RAND_MAX`. Por ejemplo, la expresión `rand() % N` da un número natural aleatorio, entre 0 y $N-1$ (siempre que $N > 0$). La expresión `(double)rand()/RAND_MAX` da un número aleatorio real, del intervalo $[0, 1]$. (Nótese que la conversión a real de alguno de los miembros de la segunda expresión es necesaria para que la división se realice en \mathbb{R} .)

Aunque desde un punto de vista estrictamente teórico la primera expresión no se distribuye uniformemente, desde un punto de vista práctico, cuando N es mucho menor que `RAND_MAX`, se obtienen unos resultados de uniformidad aceptables.

En principio, la secuencia de números que proporciona la función `rand()` es siempre la misma; para que cambie de una ejecución a otra, se ha de dar un valor inicial distinto. Este valor se conoce como *semilla*; por ejemplo, con la instrucción siguiente:

```
srand(time(NULL));
```

El empleo del reloj del sistema para proporcionar una semilla es habitual. Para poder usarlo es necesario cargar el fichero de cabeceras `time.h`.

3.13.3 Uso de estos recursos a ojo

Jugando con la función dada en el apartado anterior, pueden obtenerse expresiones aleatorias variadísimas. Por ejemplo, con expresiones como las de la figura 3.1, se pueden simular la aleatoriedad de un dado justo (o sea, equiprobable) de seis caras; un dado de caras entre a y b ; un número real extraído uniformemente del intervalo $[a, b]$; una moneda justa, donde cara y cruz salen con igual probabilidad; una moneda trucada, donde la cara sale con una probabilidad distinta que la cruz, etc.

<code>rand()%2 + 1</code>	<code>(double)rand()/RAND_MAX < p</code>
<code>rand()%6 + 1</code>	<code>(3*rand()) % 5 + 10</code>
<code>rand()%(b-a+1) + a</code>	<code>(b - a) * ((double)rand()/RAND_MAX) + a</code>

Figura 3.1: Expresiones aleatorias

3.13.4 El método de la transformada inversa para variables aleatorias continuas

Pero generar una variable aleatoria arbitraria requiere cierta dosis de perspicacia. En este apartado se introduce el método de la *transformada inversa*. En primer lugar se presenta este método para variables aleatorias continuas.

Necesitamos el siguiente teorema, cuya demostración es trivial:

Si $F : \mathbb{R} \rightarrow [0, 1]$ es una función de distribución cualquiera, se tiene la siguiente equivalencia:

$$Y \sim U[0, 1] \Leftrightarrow F^{-1}(Y) \sim F$$

Es decir, si la variable aleatoria Y sigue una distribución uniforme en $[0, 1]$, la variable aleatoria $F^{-1}(Y)$ sigue una función de distribución F , y viceversa.

El método es entonces una mera aplicación de dicho teorema: para simular una variable aleatoria X de función de distribución F , podemos generar primero una $Y \sim U[0, 1]$, y con ella calcular $X = F^{-1}(Y)$. El teorema anterior nos garantiza que $X \sim F$. La figura 3.2 resume la situación:

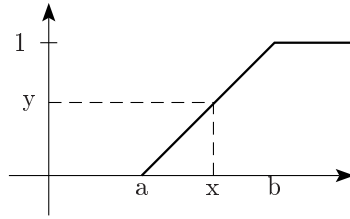


Figura 3.2: Función de distribución de una variable aleatoria continua

Supongamos que deseamos generar una *variable aleatoria uniforme* en el intervalo $[a, b]$. Como su función de distribución es,

$$F(x) = \begin{cases} 0 & \text{si } x \leq a \\ \frac{x-a}{b-a} & \text{si } a \leq x \leq b \\ 1 & \text{si } b \leq x \end{cases}$$

para $y \in [0, 1]$ se tiene $F^{-1}(y) = a + y * (b - a)$. Por lo tanto, si se genera $y \sim U[0, 1]$, basta con hallar $x = a + y * (b - a)$:

```
double y = (double)rand()/RAND_MAX;
double x = a + y*(b-a);
```

y se tiene $x \sim U[a, b]$.

3.13.5 El caso discreto

El método anterior se adapta a variables discretas fácilmente: sea la función de probabilidad $Prob(x = i) = p_i$, para $1 \leq i \leq n$; su función de distribución es $P(k) = \sum_{i=1}^k p_i$ para $1 \leq k \leq n$, y expresa la probabilidad de que $x \leq i$. Entonces, el método consiste en generar la variable aleatoria $y \sim U[0, 1]$, y hallar el mínimo k tal que $P(k) \geq y$. Supongamos que se desea simular un dado que dé los números entre 1

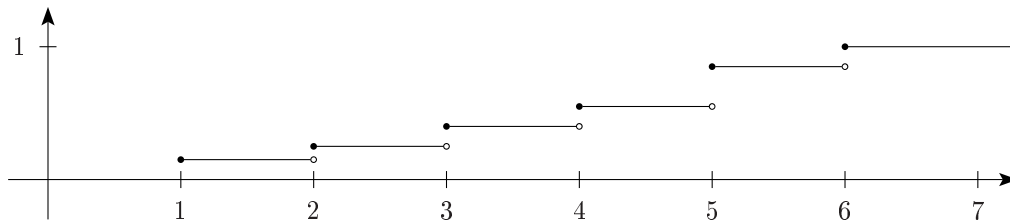


Figura 3.3: Función de distribución de una variable aleatoria discreta

y 6, y que la probabilidad p_i de obtener cada número sea 0.15, 0.1, 0.15, 0.15, 0.3 y 0.15 respectivamente. Para ello, se hallan las cantidades $P(i)$ por acumulación: 0.15, 0.25, 0.40, 0.55, 0.85 y 1.00. (Véase la figura 3.3.) Luego se genera $y \sim U[0, 1]$,

```
double y = (double)rand()/RAND_MAX;
```

y, finalmente, basta con hallar el $\min\{k \text{ tal que } P(k) \geq y\}$. Si las cantidades $P(i)$ se almacenaron en un array, esta búsqueda se puede realizar por inspección de la tabla, que tiene el contenido de la columna $P(i)$:

i	p_i	$P(i)$
1	0.15	0.15
2	0.1	0.25
3	0.15	0.40
4	0.15	0.55
5	0.3	0.85
6	0.15	1.00

Si, por ejemplo, la instrucción `double y = (double)rand()/RAND_MAX;` genera el valor 0.75, hay que buscar el menor número k que verifique que $P(k) \geq 0.75$. En este caso la cara k del dado es el cinco.

3.13.6 Expresiones anteriores

Calcula el conjunto de valores que puede producir cada una de las expresiones de la figura 3.1.

3.13.7 Uniforme continua

Deduce en C++ una función que genere una variable aleatoria uniforme real, del intervalo $[a, b]$.

3.13.8 Uniforme entera

Deduce en C++ una función que genere una variable aleatoria uniforme entera, del conjunto de números $\{a, \dots, b\}$.

3.13.9 Variable aleatoria continua

Escribe una función que genere una variable aleatoria continua de función de distribución F , definida así:

$$F(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ \sqrt{x} & \text{si } 0 \leq x \leq 1 \\ 1 & \text{si } x > 1 \end{cases}$$

3.13.10 Variable aleatoria exponencial continua

Escribe una función que genere una variable aleatoria continua de función de densidad exponencial de parámetro λ ; esto es, $f(x) = \lambda e^{-\lambda x}$. Recuerda que la función de distribución F de una función de densidad f viene dada por $F(x) = \int_{-\infty}^x f(t)dt$.

3.13.11 Dado infinito

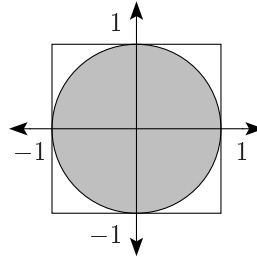
Escribe una función para simular un dado con infinitas caras, de forma que saque un 1 con probabilidad $1/2$; un 2 con probabilidad $1/4$; un 3 con probabilidad $1/8$; etc.

3.13.12 Bibliografía

El método de la transformada inversa se estudia en multitud de textos, como [Yak77, Mor84, PV87] entre otros, junto con otros métodos generales, como el del rechazo y el de la composición.

3.14 Aproximación hacia π con dardos

Considera el siguiente experimento: se dispone de una diana de radio unidad centrada en el origen de coordenadas y del cuadrado en el cual se inscribe dicha diana:



Si se efectúa un buen número de lanzamientos de dardos, uniformemente distribuidos en el cuadrado circunscrito $[-1, 1] \times [-1, 1]$, el número de los que caerán en la diana será aproximadamente proporcional a su superficie:

$$\frac{\text{número de disparos dentro}}{\text{número de disparos total}} \simeq \frac{\text{superficie de la diana}}{\text{superficie del cuadrado}}$$

y, como sabemos que

$$\frac{\text{superficie de la diana}}{\text{superficie del cuadrado}} = \frac{\pi}{4}$$

se puede estimar que

$$\pi \simeq 4 \frac{\text{número de disparos dentro}}{\text{número de disparos total}}$$

aproximación que será, probablemente, tanto más precisa cuanto mayor sea el número de lanzamientos.

Se pide un programa que efectúe un buen número de lanzamientos, que tantee los aciertos y deduzca de ahí una aproximación de π .

Bibliografía La idea de este enunciado y otras sobre simulación de variables aleatorias pueden ampliarse en [Ben84, Dew85a], entre otras muchas referencias.

3.15 Conjetura para la formación de palíndromos

Consideremos el siguiente procedimiento:

Dado un número, lo sumamos a su reverso. Si esta suma es un palíndromo, entonces paramos; y si no, repetimos el proceso con el número obtenido de dicha suma, hasta dar con un palíndromo.

Una curiosa conjetura de teoría de números afirma que, partiendo de cualquier número expresado en base 10, el procedimiento anterior para, y por tanto nos lleva a un palíndromo.

Ejemplo Aquí vemos un ejemplo de cómo funciona la conjetura. Supongamos que partimos del número 59; lo sumamos a su reverso y obtenemos 154. Repetimos la operación con 154: la suma con su reverso es 605. Por último, al sumar 605 a su reverso obtenemos un palíndromo:

$$\begin{array}{ccccccc} \rightarrow 59 & \rightarrow & \begin{array}{r} 59 \\ + 95 \\ \hline 154 \end{array} & \rightarrow & \begin{array}{r} 154 \\ + 451 \\ \hline 605 \end{array} & \rightarrow & \begin{array}{r} 605 \\ + 506 \\ \hline 1111 \end{array} & \rightarrow 1111 \leftarrow \end{array}$$

Conjetura Escribe un programa que lleve a cabo el procedimiento descrito por la conjetura para encontrar un palíndromo a partir de un número. (Véase la pista 3.15a.)

Terminación Una conjetura es una hipótesis no demostrada ni refutada. La conjetura que estamos considerando describe un método que, en caso de terminar, conduce a un palíndromo a partir de un número. Aunque el método en general termina, con algunos números no se sabe qué ocurre: por ejemplo, con el 196 se han realizado centenares de iteraciones pero no se ha conseguido llegar a un palíndromo.

Escribe un programa que lleve a cabo el procedimiento descrito por la conjetura para encontrar un palíndromo a partir de un número. El número de iteraciones tiene que limitarse para así evitar los desbordamientos y asegurar que el programa termina.

Bibliografía Desde que hay personas que piensan y demuestran ha habido conjeturas. Las conjeturas son misteriosas y siempre han generado leyendas y mitos. Muchas conjeturas en matemáticas son más populares que los teoremas más importantes. En el ejercicio 3.9 encontrarás una de las conjeturas más conocidas. La conjetura de la formación de palíndromos aparece en [Gar95], un libro ligero, ameno y divertido del maestro de la divulgación matemática Martin Gardner (1914–).

3.16 Juegos perdedores ganan



Queremos realizar un programa que permita simular juegos de azar y así observar su evolución a largo plazo. Para ello nos vamos a ir a nuestro casino particular en el que se encuentran los siguientes juegos:

Casi iguales pero no tanto El juego de *casi iguales pero no tanto* tiene las siguientes reglas: se lanza una moneda en la que sale cara con una probabilidad del 49.5% y cruz con una del 50.5%; el jugador gana un punto si sale cara y pierde un punto si sale cruz.

Escribe un programa que simule el juego a largo plazo, y que muestre los datos para que se aprecie la evolución de las ganancias o pérdidas del jugador.

Por tres es al revés El juego de *por tres es al revés* se juega con dos monedas M_1 y M_2 . Al lanzar la primera de ellas, M_1 , sale cara con un 9.5% de probabilidad y cruz con un 90.5%; al lanzar la segunda, M_2 , sale cara con una probabilidad del 75.5% y cruz con el 24.5%. El jugador gana un punto si sale cara y pierde un punto si sale cruz. Si el capital con el que cuenta el jugador es múltiplo de tres, se lanza la moneda M_1 , y si no se lanza la moneda M_2 . El jugador puede comenzar con un capital arbitrario, en puntos.

Escribe un programa que simule el juego a largo plazo y que muestre los datos para que se aprecie la evolución de las ganancias o pérdidas del jugador.

Mezclando juegos Si has realizado los ejercicios anteriores, te habrás dado cuenta de que nuestro casino es un negocio rentable. En los juegos que hemos propuesto el jugador lleva la peor parte y, a largo plazo, siempre acaba perdiendo.

Supongamos que abreviamos por C al juego *casi iguales pero no tanto* y por T al juego *por tres es al revés*. Se abre una nueva mesa de juego en nuestro casino que consiste en alternar jugadas a los dos juegos anteriores de la siguiente forma CCTTCCTTCCTT... , es decir dos jugadas al juego C, dos jugadas al juego T... ¿Apostarías tus puntos a este nuevo juego?

Escribe un programa que simule el juego a largo plazo y que muestre los datos para que se aprecie la evolución de las ganancias o pérdidas del jugador. El jugador puede comenzar con un capital arbitrario, en puntos. ¿Te sorprende el resultado?

Para este enunciado Consulta la pista 3.16a.

Bibliografía El resultado de mezclar juegos perdedores para encontrar un juego ganador es muy reciente y se conoce como *paradoja de Parrondo*. El propio autor, Juan Parrondo (1964–), lo explica muy claramente en el artículo [Par01]. En realidad, no se trata de una verdadera *paradoja* matemática, sino de un resultado sorprendente. El libro *Fotografiando las matemáticas* [Mar00] ofrece cuidadas fotografías que nos muestra cómo las matemáticas aparecen en muy diversos ámbitos de la vida. Uno de los cincuenta artículos que aglutina esta obra está dedicado al trabajo de Parrondo.

Rasgos independientes En una segunda versión, se buscaba facilitar la descripción, ofreciendo el programa diferentes peinados, ojos, orejas y expresión para que el testigo ocular los escogiese, fijando los rasgos del sujeto. Por ejemplo, los peinados disponibles son los siguientes:

```
Pelo denso      WWWWWWWW
Pelo escaso     |||||||||
Rapado          |"|"|"|"|"|"|"|"
A raya         \\\\/////
```

y residen en un archivo que tiene la siguiente disposición:

```
1 9 W
1 9 |
1 1 | 7 " 1 |
1 3 \ 6 /
```

Confecciona un programa que presente las distintas opciones para cada rasgo, para que el usuario elija una, y dibuje el individuo seleccionado.

```
Estilos de pelo disponibles:
1.- Pelo denso      WWWWWWWW
2.- Pelo escaso     |||||||||
3.- Rapado          |"|"|"|"|"|"|"|"
4.- A raya         \\\\/////
Escoja opción: _
```

Rasgos en caras Con todo, la identificación de rasgos separados no siempre resultaba fácil. Por eso se confeccionó la versión definitiva, que presentaba cuatro individuos, como los siguientes,

```
WWWWWWWWW      \\\\/////      |"|"|"|"|"|"|"|"      |||||||||
| 0 0 |        |-(. .)-|        |-(o o)-|        | \ / |
@  J  @        {  "  }        [  j  ]        <  -  >
|  ===  |        |  -  |        |  _ _  |        |  _ _  |
\ _ _ _ /        \ _ _ _ /        \ _ _ _ /        \ _ _ _ /
-- n. 1 --      -- n. 2 --      -- n. 3 --      -- n. 4 --
```

para que el testigo describiese al sospechoso combinando los ojos de un modelo, las orejas de otro y así sucesivamente. Desarrolla esta versión final, donde ahora el menú consiste en mostrar los retratos de referencia y en pedir las correspondientes elecciones:

```
Escoja los rasgos:
Pelo:      -
Ojos:      -
Orejas/nariz:  -
Boca:      -
```

Bibliografía La idea de este enunciado está tomada de [AR95].

3.19 Número de ceros en que termina un factorial



Para conocer el número de ceros finales del factorial de un número, $M! = 1 \times 2 \times 3 \times \dots \times M$, sin calcular dicho factorial, puede emplearse el siguiente método:

- Por cada factor $i \in \{1, \dots, M\}$ que sea múltiplo de 5, pero no de 5^2 , resulta un cero.
- Por cada factor $i \in \{1, \dots, M\}$ que sea múltiplo de 5^2 , pero no de 5^3 , resultan dos ceros.
- Y en general, por cada factor $i \in \{1, \dots, M\}$ que sea múltiplo de 5^i , pero no de 5^{i+1} , resultan i ceros.

Grado de multiplicidad Escribe una función (iterativa) que, dados los enteros $n \geq 1$ y $d \geq 2$, halla el grado de multiplicidad de d en n ; esto es, el número $i \geq 0$ tal que d^i es divisor de n , pero d^{i+1} no lo es. Por ejemplo, $\text{grado}(2, 24) = 3$, ya que 2^3 es divisor de 24, y en cambio 2^4 no lo es.

Ídem, recursiva

Multiplicidad de 5 Usando una cualquiera de las definiciones anteriores, escribe una función que, dado $n \geq 1$, halla el grado de multiplicidad de n respecto de 5. Por ejemplo, $\text{grado}_5(250) = 3$, ya que 5^3 es divisor de 250, y en cambio 5^4 no lo es.

Los ceros de un factorial, al fin Usando el apartado anterior, escribe una función que, conocido el entero $n \geq 1$, halla el número de ceros finales de su factorial.

3.20 Representación de un número con palabras



Se pide escribir un procedimiento que, dado un número natural, escriba por pantalla la *lectura* de ese número en castellano y en femenino. Por ejemplo, al llamar al procedimiento con el número 1204 obtendremos *mil doscientas cuatro*. Como muestra, los siguientes números: cero, una, dos, tres, cuatro, cinco, seis, siete, ocho, nueve, diez, once, doce, trece, catorce, quince, dieciséis, diecisiete, dieciocho, diecinueve, veinte, veintiuna, veintidós, veintitrés, veinticuatro, veinticinco, veintiséis, veintisiete, veintiocho, veintinueve, treinta, treinta y una, treinta y dos, treinta y tres, treinta y cuatro, treinta y cinco, treinta y seis, treinta y siete, treinta y ocho, treinta y nueve, cuarenta y siete, trescientas sesenta y cuatro, seiscientos ochenta y una, novecientos noventa y ocho, mil trescientas quince, mil seiscientos treinta y dos, mil novecientos cuarenta y nueve, dos mil doscientas sesenta y seis, dos mil quinientos ochenta y tres, dos mil novecientos, tres mil doscientas diecisiete, tres mil quinientos treinta y cuatro, tres mil ochocientos cincuenta y una, cuatro mil ciento sesenta y ocho, cuatro mil cuatrocientas ochenta y cinco, cuatro mil ochocientos dos, cinco mil ciento diecinueve, cinco mil cuatrocientas treinta y seis, cinco mil setecientos cincuenta y tres, seis mil setenta, seis mil trescientas ochenta y siete, seis mil setecientos cuatro, siete mil veintiuna, siete mil trescientas treinta y ocho, siete mil seiscientos cincuenta y cinco, siete mil novecientos setenta y dos, ocho mil doscientas ochenta y nueve, ocho mil seiscientos seis, ocho mil novecientos veintitrés, nueve mil doscientas cuarenta, nueve mil quinientos cincuenta y siete, nueve mil ochocientos setenta y cuatro, once mil cuatrocientas tres, ciento cuatro mil quinientos setenta y ocho, ciento noventa y siete mil setecientos cincuenta y tres, doscientas noventa mil novecientos veintiocho, trescientas ochenta y cuatro mil ciento tres, cuatrocientas setenta y siete mil doscientas setenta y ocho, quinientas setenta mil cuatrocientas cincuenta y tres, seiscientos sesenta y tres mil seiscientos veintiocho, setecientos cincuenta y seis mil ochocientos tres, ochocientos cuarenta y nueve mil novecientos setenta y ocho, novecientos cuarenta y tres mil ciento cincuenta y tres, un millón treinta y seis mil trescientas veintiocho, un millón ciento veintinueve mil quinientos tres, un millón doscientas veintidós mil seiscientos setenta y ocho, un millón trescientas quince mil ochocientos cincuenta y tres, un millón cuatrocientas

nueve mil veintiocho, un millón quinientas dos mil doscientas tres, un millón quinientas noventa y cinco mil trescientas setenta y ocho, un millón seiscientas ochenta y ocho mil quinientas cincuenta y tres, un millón setecientas ochenta y una mil setecientas veintiocho, un millón ochocientas setenta y cuatro mil novecientas tres, un millón novecientas sesenta y ocho mil setenta y ocho, dos millones sesenta y una mil doscientas cincuenta y tres, dos millones ciento cincuenta y cuatro mil cuatrocientas veintiocho, dos millones doscientas cuarenta y siete mil seiscientas tres, dos millones trescientas cuarenta mil setecientas setenta y ocho, dos millones cuatrocientas treinta y tres mil novecientas cincuenta y tres, dos millones quinientas veintisiete mil ciento veintiocho, dos millones seiscientas veinte mil trescientas tres, dos millones setecientas trece mil cuatrocientas setenta y ocho, dos millones ochocientas seis mil seiscientas cincuenta y tres, dos millones ochocientas noventa y nueve mil ochocientas veintiocho, dos millones novecientas noventa y tres mil tres, cien millones trescientas cuarenta y cinco mil seiscientos cinco, y trescientos veinticinco millones novecientas cuarenta y tres mil quinientas ochenta y seis... *y entró en el pueblo a pie, contando sus pasos, para que cada uno tuviera un nombre, y para no olvidarlos nunca más.*

Seda, Alessandro Baricco

3.21 ¿Cuál es el mejor orden para recibir los datos de un polinomio?



Junto con un amigo, que no ha leído este libro pero que sabe mucho Fortran, asistes a una entrevista con un pequeño empresario que necesita un programa para evaluar polinomios. Al empresario le gustaría un programa que, para evaluar el polinomio

$$a_n x^n + \dots + a_1 x + a_0$$

en un cierto valor de x , leyera todos los datos en una línea con el orden a_n, \dots, a_1, a_0, x . Intentáis convencerlo para que se conforme con un programa que lea primero el valor de la variable x y luego los coeficientes a_i , ya sea así,

$$x, a_0, a_1, \dots, a_n$$

o así:

$$x, a_n, a_{n-1}, \dots, a_1.$$

Al terminar la entrevista, tu amigo afirma que, si el empresario se empeña en exigir su orden, con el poco C++ que sabes, tendrás que abandonar el proyecto y las copiosas ganancias que te habría producido.

Pero tú, rebelde por naturaleza y no menos práctico, te resistes a perder los beneficios antedichos, y decides demostrar que tu amigo estaba equivocado. Para ello, haz un procedimiento que evalúe un polinomio leyendo de la entrada los coeficientes y el valor de la variable así,

$$a_0, a_1, \dots, a_n, x$$

y otro que los lea así:

$$a_n, a_{n-1}, \dots, a_1, x.$$

El procedimiento no escribirá *nada*: devolverá el resultado de la evaluación en un parámetro. (Véase la pista 3.21a.)

3.22 Calificación de oído

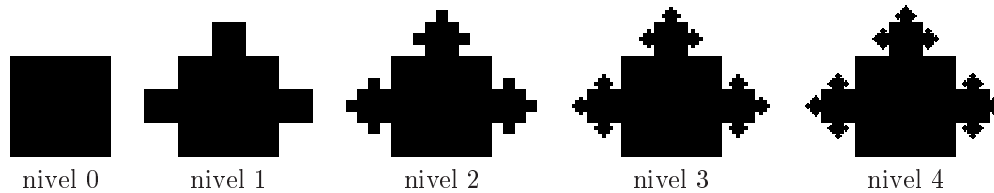
Un profesor, ya curtido por los años, se ha ido convenciendo por su experiencia de que el tamaño de las orejas está directamente relacionado con la inteligencia, y ésta con la nota que se obtiene en los exámenes. No entraré en los pormenores de su cuidadoso razonamiento, pero el caso es que ha llegado a la conclusión (inconfesable) de que le basta con medir las orejas de sus alumnos para aprobar a los que estén por encima de la media y suspender al resto. Así de rápido, y sin las engorrosas e innecesarias correcciones de exámenes.

Para llevar a cabo su ingenioso plan, ha escrito un programa que lee de la entrada estándar una lista formada por pares (nombre, tamaño) y escribe a continuación la lista de nombres con su calificación: apto o no apto. Naturalmente, la lista de datos sólo se ha de introducir una vez.

Y ahora sólo una cosa le resta por hacer: obtener los datos sin levantar sospechas. . .

3.23 Los cuadrados abominables y cáncer

El *cuadrado abominable* se define por niveles como se ve en las siguientes figuras:

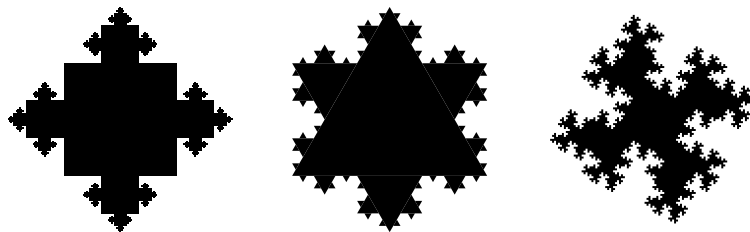


Es decir, en el nivel 0 tenemos un cuadrado de lado ℓ ; en el nivel 1 pegamos un cuadrado de lado $\ell/3$ en tres lados del cuadrado anterior; en el nivel 2 pegamos un cuadrado de lado $\ell/9$ en tres lados (los que quedan libres) de los cuadrados añadidos en el nivel anterior; etc. Tal vez sea más útil darse cuenta de que para hacer el nivel $n + 1$ hay que pegar tres cuadrados abominables de nivel n .

Pertenencia Escribe una función que, dado un nivel n y un punto del plano (x, y) , nos diga si dicho punto cae o no dentro del cuadrado abominable de nivel n . Se supondrá que el cuadrado abominable de nivel 0 tiene su extremo inferior izquierdo en el $(0, 0)$ y su lado mide 1.

En el límite Definimos el *cuadrado cáncer* como la unión de *todos* los niveles del cuadrado abominable. Escribe otra función que dado un punto (x, y) diga si dicho punto cae dentro del cuadrado cáncer. Haz las mismas suposiciones que para el caso anterior.

Otros entes abominables



Bibliografía Este insípido cuadrado cáncer es simplemente un fractal. Porque es *autosimilar*: si se mira de cerca cualquiera de las protuberancias, se observa un objeto similar al original. Tiene propiedades matemáticas harto curiosas; por ejemplo, su perímetro es infinito.

Benoît Mandelbrot (nacido el 20 de noviembre de 1924 en Varsovia, Polonia) acuñó el término fractal y se encargó de difundir la importancia y utilidad de las teorías matemáticas que circundan este concepto. Su libro más proselitista es [Man97].

Pero no es nada arriesgado afirmar que la aceptación de los fractales se debe más a su belleza visual que a su belleza elucubrativa. Las imágenes del libro [PR86] son una demostración prepotente de este hecho; a su lado, el cuadrado cáncer es un simple borrón de tinta.

3 24 Codificaciones de plantas con cadenas



Lindomayo ha descubierto que la estructura de las plantas planas está escrita con el abecedario de las letras T, I, D, B, C, H y A. Una palabra escrita con estas letras no describe a una especie, sino a un ejemplar. Para aprender a leer el lenguaje de Lindomayo basta con entender una correspondencia muy simple, a saber: T indica una pequeña prolongación, por un *Tramo*, de la rama en la que estamos; B describe un *Brote* que acaba la rama y C una rama que acaba bruscamente en un *Corte*; empieza una subrama a la *Izquierda* o la *Derecha* cuando leemos una I o una D; una H o una A representan una *Hoja* que cuelga a la izquierda o a la derecha. Por ejemplo, en TTHTTATHTTTATHTTTATTB leemos la larga rama de una cangorza recién comprada, adornada con hojas a la izquierda y a la derecha alternativamente; pero, en cuanto pasan unos días en nuestro higiénico urbanismo, pierde ansia, se le alargan los tallos y la describimos con TTTHTTTTATTTTB. Un níspero nos servirá para ilustrar el mecanismo de las subramas: TTTT⁶TTHTAT¹³ITHTATB²⁰DTHTATB²⁷DTHTAT²⁹ITHTAT³⁶ITHTATB⁴³DTHTATB⁵⁰ITHTAT⁵²ITHTAT⁵⁴ITHTATB⁶¹DTHTATB⁶⁸ITHTAT⁷⁰. El tronco del níspero está pelado (o podado) en sus 6 primeros tramos. Se bifurca por primera vez tras la etiqueta 6; desde aquí hasta 29, leemos la descripción de una rama que brota a la izquierda; de 29 hasta 52 tenemos otra rama, idéntica a la anterior, salvo en que brotó hacia la derecha. Cada una de estas dos ramas consta de otras dos subramas, descritas con las letras en los intervalos [13, 20], [20, 27], [36, 43] y [43, 50]. La lectura de estas palabras obliga a detener temporalmente la construcción mental de una rama cuando encontramos una bifurcación por culpa de I o D; pero en cuanto la subrama termina, la siguientes letras contribuyen a nuestra imagen de la rama original. Así, las letras en los intervalos [52, 54] y [68, 70] forman parte de la descripción del tronco del árbol.

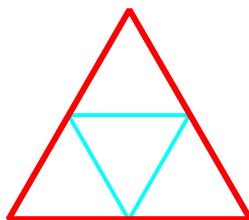
El *nivel* de una rama indica cuántas bifurcaciones hubo que tomar para llegar a ella. En el níspero, el tronco está en el nivel 0, la rama [6, 29] está en el nivel 1, y la rama [13, 20] está en el nivel 2. La *longitud* de una rama es el número de tramos que contiene.

En las dos horas que le quedan antes de que vuelva su marido, Lindomayo predente construir varios programas para insuflar vida a sus plantas. El programa *crece* añade n tramos delante de cada brote de una planta. El programa *nivel* calculará el nivel máximo de las ramas de un árbol (posiblemente también devuelva una de las subramas en ese nivel). El programa *es* concebirá un níspero saludable con subramas de hasta nivel n . El programa *compu* podará con un método que genera buenos beneficios de los árboles públicos: cada rama se cortará a la mitad de su longitud. El programa *beneficio* calculará el número de tramos de un árbol que se podan con el método del programa anterior.

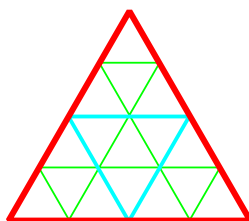
Un poco de historia Aristid Lindenmayer (1925–1989) presentó en 1968 una formalización matemática del crecimiento de las plantas que luego se ha dado en llamar sistemas de Lindenmayer o simplemente sistemas-L (*Lindenmayer Systems* o *L-systems* en inglés, para que lances una búsqueda en tu indexador favorito). El tema de este ejercicio se basa en este formalismo, convenientemente desforestado. El área de los lenguajes formales engloba a los sistemas-L y a otros formalismos como las gramáticas EBNF con las que usualmente se define la sintaxis de los lenguajes de programación. Por otro lado, los sistemas de Lindenmayer también tienen conexión con los fractales.

3.25.1 Pasatiempo

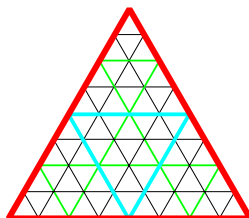
¿Eres capaz de dibujar la siguiente figura sin levantar el lápiz del papel y sin pasar dos veces por el mismo segmento de línea?



No es muy difícil. ¿Te atreves con este otro?

**3.25.2 Desafío**

Si has solucionado los pasatiempos anteriores, habrás notado que hay una manera *recurrente* de realizar el dibujo. Utilizando esta recurrencia podríamos hacer dibujos de triángulos *anidados* con tantos niveles de anidación como deseemos. Por ejemplo el siguiente dibujo tiene 3 niveles:



Te proponemos realizar un programa que dibuje una figura como las anteriores con el nivel de anidación en los triángulos que deseemos. (Véase la pista 3.25a.)

3.25.3 Un poco de ayuda

Para realizar este ejercicio contaremos con cierta ayuda a la hora de dibujar. Supondremos que tenemos un *trazador* al que daremos las órdenes para que dibuje lo que necesitamos.

El trazador dispone de una pluma multicolor que permite el dibujo de líneas de cierto tamaño en una dirección a partir del punto en el que está situado. El punto de origen del trazado de las líneas, la dirección del trazo, la longitud y color de las líneas se pueden establecer con los procedimientos y tipos que se detallan a continuación.

- Procedimiento

```
void construirTrazador(double const tamX, double const tamY, string const titulo);
```

que prepara un área de trazado de dimensión `tamX` × `tamY`. El título de la ventana se indica en el parámetro `titulo`. La pluma del trazador se establece en el origen de coordenadas (esquina inferior izquierda) y el ángulo de dibujo es cero.

- Procedimiento

```
void ponerCoordenadas(double const x, double const y);
```

que establece el punto de dibujo del trazador en las coordenadas (x, y) .

- Procedimiento

```
void girar(double const alpha);
```

que añade el ángulo de dibujo indicado en la variable `alpha` al ángulo actual de dibujo. El ángulo viene dado en radianes.

- Procedimiento

```
void dibujarLinea(double const longitud);
```

que dibuja una línea de la `longitud` indicada en el ángulo actual del dibujo, desde el punto en que está la pluma. El punto de dibujo de la pluma se establece al final de la línea dibujada.

- Tipo enumerado

```
typedef enum Colores {  
    blanco, amarillo, naranja, rosa, rojo, marron, verde, morado, azul, negro  
} Colores;
```

- Procedimiento

```
void ponerColor(Colores const c);
```

que establece el color de dibujo al indicado en la variable `c`.

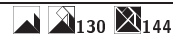
- Procedimiento

```
void esperar();
```

que espera hasta que se cierre la ventana de dibujo.

Todos estos procedimientos están en un paquete que puedes encontrar en <http://aljibe.sip.ucm.es/recursos/trazador/>.

3.26 Cálculo puntual de la matriz de mediotono de Judice-Jarvis-Ninke



La *matriz de mediotono de Limb* de nivel 0, que denotaremos con L_0 , es una matriz entera de tamaño 1×1 cuyo único elemento vale 0. La matriz de mediotono de Limb de nivel $n > 0$, L_n , es una matriz entera de tamaño $2^n \times 2^n$ que viene definida por

$$L_n = \begin{bmatrix} 4L_{n-1} & 4L_{n-1} + 3U_{n-1} \\ 4L_{n-1} + 2U_{n-1} & 4L_{n-1} + 1U_{n-1} \end{bmatrix}$$

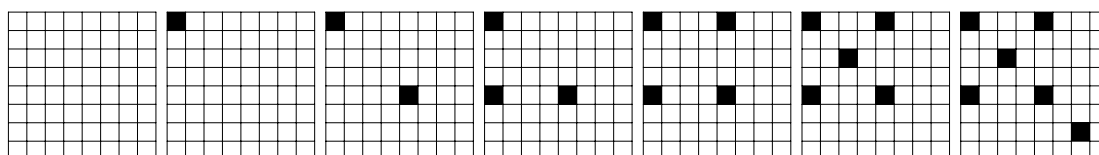
donde U_n es una matriz $2^n \times 2^n$ con todos sus elementos a 1.

Haz primero una función recursiva que, dada una fila i , una columna j y un nivel n , calcule el elemento de la posición (i, j) de la matriz de Limb de nivel n . Implementa luego una variante iterativa. (Véase la pista 3.26a.)

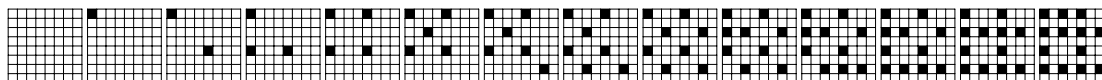
Un poco de historia Estas matrices pretenden solventar el problema de plasmar una imagen con múltiples tonos de gris en un dispositivo que sólo admite blanco o negro. Para articular esta exposición utilizaremos L_3 , que es una matriz 8×8 :

$$L_3 = \begin{bmatrix} 0 & 48 & 12 & 60 & 3 & 51 & 15 & 63 \\ 32 & 16 & 44 & 28 & 35 & 19 & 47 & 31 \\ 8 & 56 & 4 & 52 & 11 & 59 & 7 & 55 \\ 40 & 24 & 36 & 20 & 43 & 27 & 39 & 23 \\ 2 & 50 & 14 & 62 & 1 & 49 & 13 & 61 \\ 34 & 18 & 46 & 30 & 33 & 17 & 45 & 29 \\ 10 & 58 & 6 & 54 & 9 & 57 & 5 & 53 \\ 42 & 26 & 38 & 22 & 41 & 25 & 37 & 21 \end{bmatrix}$$

Con un poco de paciencia se verá que en esta matriz aparecen todos los enteros entre el 0 y el 63. Y con un poco de ingenio se podrá demostrar que una matriz L_n tiene todos los enteros entre el 0 y el $2^{2^n} - 1$. Tanto esta propiedad, como que haya que recorrer la matriz a saltos para seguirlos, forma parte de lo que querían conseguir sus autores. Se pretende que la matriz L_3 coordine la generación de las siguientes 64 figuras (una imagen vale más que mil palabras):



¡No cabe! Un poco más pequeño:



¡Vaya! Lo volvemos a intentar:



El resultado semejará una secuencia de tonos de gris progresivamente más oscuros. Tal vez, a esta escala, los puntos individuales de cada figura sean todavía ostensibles; pero con una reducción adicional, el efecto es mucho mejor:



Bibliografía El proceso de construcción de esta matriz se explica en [JJN74], fuente que es prácticamente imposible consultar. Pero, como casi todo lo relativo a informática gráfica, se puede encontrar una breve descripción en el fabuloso compendio [FvDFH97]. El libro [Rim93] se dedica exclusivamente al problema de plasmar, en un dispositivo monocromático, imágenes con múltiples tonos o colores; el tiempo lo ha castigado con creces por la arrogancia de querer ser muy práctico y moderno en su tiempo.

3.27 Dibujo de árboles mediante fractales



Muchas estructuras de la naturaleza se pueden dibujar en una computadora mediante el uso de fractales, no sólo *cánceres* como en el ejercicio 3.23. En este ejercicio usaremos un fractal sencillo que sirve para dibujar árboles similares a los siguientes:



Cómo dibujar un árbol El dibujo básico del fractal será un tronco con una hoja:



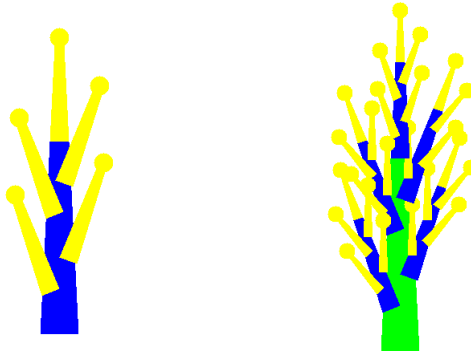
Y se procederá de forma recursiva con un algoritmo similar al siguiente

```

<Dibujar el tronco>
if (<nivel básico>) {
  <Dibujar la hoja>
} else {
  <Dibujar cinco subárboles de tamaño la mitad>
}

```

para obtener árboles como los siguientes, dependiendo del nivel de profundidad que elijamos:



Obviamente, para obtener resultados que se parezcan a árboles, habrá que escoger los colores de forma apropiada. Si quieres obtener mayor realismo en el dibujo de árboles tendrás que pensar un poco y hacer algunas pruebas. (Véase la pista 3.27a.)

Un poco de ayuda Para simplificar las tareas de dibujo, supondremos que disponemos del trazador que definimos en el ejercicio 3.25 teniendo en cuenta que se han añadido las siguientes funcionalidades:

- Procedimiento

```
void ponerColor(double const rojo, double const verde, double const azul);
```

que establece el color de dibujo en el espacio RGB (*red*, *green*, *blue*; rojo, verde, azul en inglés), siendo $0 \leq \text{rojo}, \text{verde}, \text{azul} \leq 1$. (Véase el ejercicio 1.1.)

- Procedimiento

```
void dibujarDisco(double const centroX, double const centroY, double const radio);
```

que dibuja un disco en la posición (centroX, centroY) con el radio indicado, obviamente $\text{radio} > 0$.

- Procedimiento

```
void dibujarPoligono(double const x[], double const y[], int const numPuntos);
```

que dibuja un polígono cuyos vértices son $(x[i], y[i])$ con $0 \leq i < \text{numPuntos}$.

3.28 Dragones y teselas



¿Te gusta el dibujo que aparece en la figura 3.5? Es un fractal, llamado la *curva del dragón* debido a

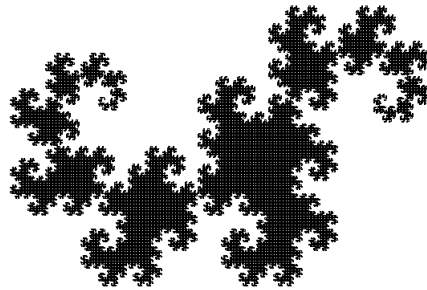


Figura 3.5: Curva del dragón

que, echándole bastante imaginación, se asemeja a un dragón.

En este ejercicio tendrás que desarrollar un programa que dibuje *dragones*. No te preocupes, es más fácil de lo que crees.

3.28.1 Técnica de construcción

Existen diversas formas de construir este fractal, pero la que nos va a servir de más ayuda fue la que describió el físico Bruce A. Banks:

Empezamos con un gran ángulo recto, y en cada paso sustituimos cada segmento por un nuevo ángulo recto de dimensiones menores tal y como se ilustra en la figura 3.6.

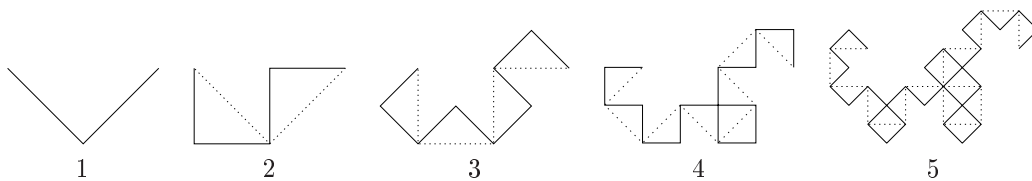


Figura 3.6: Pasos en la generación de una curva dragón

El número de veces que realicemos esa transformación de un lado por un par de lados nos indicará el grado de la curva; en la figura 3.6 tenemos las curvas desde el grado uno al grado cinco.

3.28.2 Un poco de ayuda

Para ayudarnos a dibujar, contamos con el trazador que se define en el ejercicio 3.25.

3.28.3 Un poco de historia

Al fractal que presenta el ejercicio se le suele llamar el dragón de Harter-Heighway, en honor a John Heighway y William Harter, que junto con su colega Bruce A. Banks, fueron los primeros en estudiarlo. En [Gar86] puedes encontrar más detalles.

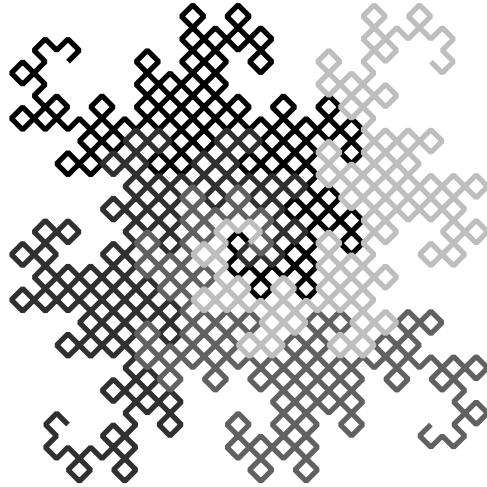


Figura 3.7: Cuatro dragones acoplados entre sí

El dragón tiene varias propiedades interesantes: se puede dibujar de un solo trazo sin que en ningún momento se cruce la línea ya dibujada; dos fractales coinciden exactamente en el borde de forma que se pueden acoplar unos con otros y se puede llegar a recubrir todo el plano con fractales de este tipo. Esta última característica le sirvió al afamado informático Donald E. Knuth (1938–) para alicatar un muro de su casa.