UNIVERSIDAD NEBRIJA

Escuela Politécnica Superior

Estructuras de datos y algoritmos

Práctica 1

Desarrolle las siguientes funciones de forma recursiva utilizando la declaración propuesta.

- La paridad de un número *n* mediante restas.

```
bool numPar(int n)
```

- La paridad de un número utilizando recursión indirecta. La función esPar llama a la función esImpar y viceversa.

```
bool esPar(int n)
bool esImpar(int n)
```

- El producto de dos números positivos *n* y *m* realizando sumas.

$$n \cdot m = \sum_{k=1}^{n} m$$

```
int producto(int n, int m)
```

- La potencia de dos números calculada con productos.

  La función *potencia*$_1$ definida con una sola relación de recurrencia que se aplica a todos los valores de *n* mayores de cero.

$$x^n = \begin{cases} 1 & si\ n = 0 \\ x \cdot x^{n-1} & si\ n > 0 \end{cases}$$

```
int potencia1(int x, int n)
```

- La función *potencia₂* definida con dos relaciones de recurrencia, una para valores pares y otra para impares.

$$x^n = \begin{cases} 1 & si\ n = 0 \\ x^{n/2} \cdot x^{n/2} & si\ n\ es\ par \\ x \cdot x^{n/2} \cdot x^{n/2} & si\ n\ es\ impar \end{cases}$$

```
int potencia2(int x, int n)
```

- La función factorial de un número *n* mayor o igual que cero.

$$n! = \begin{cases} 1 & si\ n = 0 \\ n\ (n-1)! & si\ n > 0 \end{cases}$$

```
int factorial(int n)
```

- La sucesión de Fibonacci.

$$fibonacci\ (n) = \begin{cases} 0, & si\ n = 0 \\ 1, & si\ n = 1 \\ fibonacci(n-2) + fibonacci(n-1), & si\ n > 1, \end{cases}$$

```
int fibonacci(int n)
```

- El cociente y el resto de la división de dos números enteros.

```
int cociente(int dividendo, int divisor)
int resto(int dividendo, int divisor)
```

- La función múltiplo determina si un número entero es divisible por otro.

```
bool multiplo(int dividendo, int divisor)
```

- La sumatoria de todos los números desde 1 hasta *n*, la sumatoria de los números pares y de los números impares desde 1 hasta *n*.

```
int sumatoria(int n)
int sumatoriaPar(int n)
int sumatoriaImpar(int n)
```

- El máximo común divisor calculado con el método de Euclides.

$$mcd(a,b) = \begin{cases} a, & si\ b = 0 \\ mcd(b, a\ mod\ b), & si\ b \neq 0 \end{cases}$$

```
int gcd(int a, int b)
```

- La evaluación de una función de grado *n* en un punto x.

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2 x^2 + a_1 x^1 + a_o x^0$$

La función f(x) se representa con un vector de coeficientes de *n*+1 elementos: {a_0, a_1, a_2, … , a_{n-2}, a_{n-1}, a_n}.

Por ejemplo, la función f(x) = $2x^4$ - $5x^3$ - $3x^2$ + 4x + 10 se representa con el vector de coeficientes: {10, 4, -3, -5, 2}. Si la función se evalúa en x = 2, se tiene que f(2) es -2. Otros ejemplos de evaluar esta función en otros puntos de x son: f(0)=10, f(1)=8, f(-1)=10, f(-2)=62.

Para calcular la potencia de un número x elevado a *n*, utilice la función pow(x, n).

```
double eval(int c[], int n, int valor)
```

- Funciones recursivas aplicadas a un array de números enteros.

```
int minimo(int v[], int n)
int maximo(int v[], int n)
int sumatoria(int v[], int n)
int sumatoriaPar(int v[], int n)
int sumatoriaImpar(int v[], int n)
double promedio(int v[], int n, int total)
bool existe(int v[], int n, int valor)
void invierte(int v[], int p, int n)
```

- Las torres de Hanoi.

  Las Torres de Hanoi es un juego inventado en 1883 por el matemático francés Éduard Lucas. Consiste en tres varillas verticales y un número finito de discos de distinto tamaño. Al inicio todos los discos están colocados en la primera varilla, ordenados de menor a mayor tamaño. El juego consiste en pasar todos los discos a la tercera varilla respetando el orden del tamaño de los discos. Las reglas son:

  a. Solo se puede mover un disco cada vez.
  b. Un disco de mayor tamaño no puede colocarse sobre uno más pequeño.
  c. Solo se puede mover el disco que se encuentra sobre el resto de discos de cada varilla.

```
hanoi(int n, int origen, int auxiliar, int destino)
```

Desarrolle las siguientes funciones de forma iterativa utilizando la declaración propuesta.

- La paridad de un número *n* mediante restas.

```
bool numPar2(int n)
```

- El cociente y el resto de la división de dos números enteros.

```
int cociente2(int dividendo, int divisor)
int resto2(int dividendo, int divisor)
```

- La función múltiplo determina si un número entero es divisible por otro.

```
bool multiplo2(int dividendo, int divisor)
```

- La sucesión de Fibonacci.

```
int fibonacci2(int n)
```

- La representación binaria de un número entero decimal sin signo.

```
void imprimeBinario(unsigned short n)
```

- El producto de dos números positivos *n* y *m* realizando sumas.

$$n \cdot m = \sum_{k=1}^{n} m$$

```
int producto2(int n, int m)
```

- La potencia de x elevado a *n* con productos.

$$x^n = \prod_{k=1}^{n} x$$

```
int potencia3(int x, int n)
```

- La evaluación de una función de grado *n* en un punto x.

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2 x^2 + a_1 x^1 + a_o x^0$$

La función f(x) se representa con un vector de coeficientes de *n*+1 elementos: $\{a_0, a_1, a_2, \ldots, a_{n-2}, a_{n-1}, a_n\}$.

```
double eval2(int c[], int n, int valor)
```

- Las funciones esIdentidad, esSimetrica para matrices de orden $m \times m$, la función transpuesta de una matriz de orden $m \times m$ y la función esIgual para comparar dos matrices de orden $m \times m$.

```
bool esIdentidad(int m[][], int m)
bool esSimetrica(int m[][], int m)
bool transpuesta(int m[][], int t[][], int m, int n)
bool esIgual(int m1[][], int m2[][], int m, int n)
```

- La función esPerfecto determina si un número *n* es perfecto. Un número es perfecto cuando es igual a la suma de sus divisores positivos sin incluirse a sí mismo.

  Por ejemplo, los números 6, 28 y 496 son perfectos porque son iguales a la suma de sus divisores: 6 = 1+2+3, 28 = 1+2+4+7+14 y 496 = 1+2+4+8+16+31+62+124+248.

```
bool esPerfecto(int n)
```