EJERCICIOS TEMA 4 (Inmersión de Programas)

1. Obtener una función recursiva final mediante desplegado y plegado que calcule la altura de una pila. A partir de esta función obtener una función con postcondición constante.

2. Una función recursiva que calcula la potencia de dos números naturales (a^b) puede ser:

```
int elev(int a, int b)
{   assert(b >= 0);
    int e;

    if(b == 0)
        e = 1;
    else
        e = elev(a, b - 1) * a;
    return e;
}
```

b) Obtener a partir de la función anterior una función recursiva final, que llamaremos **g_elev**, mediante la técnica de desplegado y plegado.

c) Modificar la función anterior para que antes de cada llamada recursiva indique el valor actual de b y que potencia se está calculando:

```
Valor actual b: 4   Calculando: 5 ^ 4
Valor actual b: 3   Calculando: 5 ^ 4
........
```

d) Escribir una función **elev2** que calcule la potencia de dos naturales haciendo una llamada a **g_elev** con los valores iniciales adecuados.

3. Una función recursiva que cuenta cuantas veces está un elemento **e** en una pila **p** puede ser:

func contar(p: pila; e: elem) dev n: nat
{Pre: cierto} {Dec: altura(p)}
[(nula(p)) ⟶ n := 0;
(no(nula(p))) ⟶
[(cima(p) =e) ⟶ n := contar(desapilar(p),e);
n := n + 1;
(cima(p) ≠ e) ⟶ n := contar(desapilar(p),e);
]
]
{Post: n = cuantas(p, e)}

Las ecuaciones de la operación de **cuantas** son:

∀ p: pila, ∀ x, e: elem
1) cuantas(p_nula, e) ≡ 0
2) (x = e) ⇒ cuantas(apilar(x, p),e) ≡ cuantas(p, e) + 1
3) (x ≠ e) ⇒ cuantas(apilar(x, p),e) ≡ cuantas(p, e)

a) Obtener a partir de la función anterior una función recursiva final, que llamaremos **g_contar**, mediante la técnica de desplegado y plegado.

b) Escribir una función **contar_c**, que hace lo mismo que contar pero haciendo una llamada a **g_contar** con los valores iniciales adecuados.

4. Una función recursiva que indica si dos pilas **p1, p2** son iguales puede ser:

```
func es_igual(p1, p2: pila) dev b: boolean
{Pre:  cierto} {Dec: minima(altura(p1), altura(p2))}
[      ((nula(p1)∧(nula(p2))        ⟶    b := cierto;
       ((nula(p1)∧(no(nula(p2)))    ⟶    b := falso;
       ((no(nula(p1))∧(nula(p2))    ⟶    b := falso;
       ((no(nula(p1))∧(no(nula(p2))) ⟶
                                    b := es_igual(desapilar(p1), desapilar(p2));
                                    b := b ∧ (cima(p1) = cima(p2));
]
{Post: b = igual(p1, p2);}
```

Las ecuaciones de la operación de **igual** son:

∀ **p1, p2: pila,** ∀ **x, y: elem**

1) igual(p_nula, p_nula) ≡ cierto
2) igual(p_nula, apilar(y, p2)) ≡ falso
3) igual(apilar(x, p1), p_nula) ≡ falso
4) igual(apilar(x, p1), apilar(y, p2)) ≡ (x = y) ∧ igual(p1, p2)

a) Obtener a partir de la función anterior una función recursiva final, que llamaremos **g_igual**, mediante la técnica de desplegado y plegado.

b) Escribir una función **es_igual_g**, que hace lo mismo que contar pero haciendo una llamada a **g_igual** con los valores iniciales adecuados.

Inmersión de especificaciones:

5.

a) Escribir la especificación de una función que calcule el producto escalar de dos vectores de enteros. Esta función no se puede implementar directamente. Probar a realizar una inmersión de la especificación. Probar primero debilitando la postcondición y después reforzando la precondición. Escribir para cada uno cómo sería la función de cota y cuales serían las condiciones de los casos directo y recursivo (no hace falta implementar las funciones).

b) A partir de estas funciones escribir funciones que calculen el producto escalar.

6. Repetir el ejercicio anterior para una función que decida si un vector de naturales es capicúa.