EJERCICIOS TEMA 4 (Verificación funciones recursivas)

1. Verificar la siguiente función recursiva que calcula la potencia de dos números enteros (a^b). Estudiar primero si hace falta alguna precondición.

```
int elev(int a, int b)
{
    int e;

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

2. Escríbase un programa recursivo que evalúe el número combinatorio ($^{m}_{n}$). Recordar que

$$\binom{m}{n} = \frac{m!}{n!(m-n)!}$$

Calcularlo mediante llamadas al factorial da lugar a valores intermedios exageradamente altos, que se deben evitar. Resolverlo utilizando únicamente recursividad lineal. Para ello habrá que expresar ($^{m}_{n}$) en función de ($^{m}_{n-1}$).

3. Encuéntrese un programa recursivo verificado que cuente el número de elementos de una pila de naturales (es decir, calcule la altura). Repetir el problema para el caso de una cola de naturales.

4. Escribir un programa recursivo verificado que decida si dos pilas de naturales son iguales.

5. Comprobar para que valores de los parámetros la siguiente función recursiva acaba.

```
int f(int a, int b)
{
    int r;

    if(a + b == 0)
        r = 0;
    else
    {
        r = f(2 * a, b - a - 1);
        r = r + 1;
    }
    return e;
}
```

6. Escribir un programa recursivo verificado que sume todos los elementos de una Lista_PI a partir del punto de interés (punto de interés incluido). La función de cota que se utilizará para poder aplicar el principio de inducción será la operación *pend*, que devuelve el número de elementos que quedan por recorrer y se define:

PEND1) pend(<l1, l2>) ≡ long(l2)

7. La siguiente función recursiva comprueba si alguno de los elementos del vector **a** desde 1 hasta la posición **i** es igual a cero:

```
func algun_cero(a: vector[1..N]; i: natural) dev b: booleano
{Pre: i ≤ N;      Dec: i}
[ i = 0 → b := ???
  i ≠ 0 →  b := algun_cero(a, i – 1);
           b := ???
  ]
{Post: b = ∃ α:1 ≤ α ≤ i: a[α] = 0 }
```

Completa y verifica totalmente la función.

8. Dada la siguiente función recursiva:

```
func f_igual (a: entero; p: pila) dev ig: bool
{Pre: cierto; Dec: altura(p) }
[ nula(p) → ig := cierto
 ~nula(p) → ig := f_igual(a, ??? );
            ig := ???
  ]
{Post:      ig = igual(a, p) }
```

Y dadas las siguientes ecuaciones de la operación igual:

```
∀ p: pila, ∀  a, x: entero
1) igual(a, p_nula) ≡ cierto
2) igual(a, apilar(x, p) ) ≡ (a = x) ∧ igual(a, p)
```

Completa y verifica la función.

9. Completa y verifica la siguiente función recursiva:

```
func f_mezcla(p1, p2: pila) dev q: pila
{Pre: cierto; Dec: ???}
[ nula(p1) → q := E1
  nula(p2) → q := E2
 ~nula(p1) ∧ ~nula(p2) → q := f_mezcla(E3, E4 );
                         {A1: ??? }
                           q := E5
  ]
{Post: q = mezcla(p1, p2) }
```

Dadas las siguientes ecuaciones de la operación mezcla:

```
∀ p1, p2: pila, ∀  x, y: elem
1) mezcla(p1, p_nula) ≡ p1
2) mezcla(p_nula, p1) ≡ p1
3) mezcla(apilar(x, p1), apilar(y, p2) ) ≡ apilar(x, apilar(y,
   mezcla(p1, p2) ) )
```