



Nombre:

1. (1 ptos) Especifica (incluyendo precondition y postcondición) la función máximo común divisor.

2. (1.5 ptos) Escribe las ecuaciones de la operación *max_pila*, que calcula el máximo de una pila de enteros.

Operaciones:

max_pila: pila → entero

Ecuaciones:

**Nombre:****3. (1.5 pts)** Las ecuaciones de la operación **concatenar** para pilas son:**conc:** pila, pila \rightarrow pila \forall p1, p2: pila, \forall x: elem1) **conc**(p_nula, p2) \equiv p22) **conc**(apilar(x, p1), p2) \equiv apilar(x, **conc**(p1, p2))Utilizando las ecuaciones del tipo **pila** y las ecuaciones anteriores demostrar el siguiente **teorema inductivo**: \forall p1, p2: pila**altura**(**conc**(p1, p2)) \equiv **altura**(p1) + **altura**(p2)



Nombre:

4. (1.5 ptos) Nos han dado la siguiente función diciéndonos que sirve para calcular el máximo común divisor de a y b para a y b naturales:

```
func maxcd(a, b: nat) dev m: nat
{Pre:  a > 0 ∧ b > 0; Dec: ??}
[ a = b → m := a
  a > b → m := maxcd(a - b, b)
  a < b → m := maxcd(a, b - a)
]
{Post: ??}
```

Escribe la función de cota de esta función recursiva y demuestra que acabará.

5. (2 ptos) 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) $\text{cuantas}(p_nula, e) \equiv 0$
 - 2) $(x = e) \Rightarrow \text{cuantas}(\text{apilar}(x, p), e) \equiv \text{cuantas}(p, e) + 1$
 - 3) $(x \neq e) \Rightarrow \text{cuantas}(\text{apilar}(x, p), e) \equiv \text{cuantas}(p, e)$

a) (1.7 ptos) Obtener a partir de la función anterior una función recursiva final con postcondición constante, que llamaremos **g_contar**.



Nombre:

b) (0.3 pts) Escribir una función **contar_c**, que hace lo mismo que contar pero haciendo una llamada a **g_contar** con los valores iniciales adecuados.

6. (2.5 pts) La siguiente función iterativa comprueba si alguno de los elementos del vector **a** indexado de 1 a N es igual a cero:

```
func algun_cero(a: vector) dev b: booleano
{Pre: cierto}
i := 0; b := falso;
{Inv: ?? ; Dec: ?? }
*[ i ≠ N → b := b ∨ (a[i + 1] = 0);
  i := i + 1
]
{Post: b = ∃ α: 1 ≤ α ≤ N: a[α] = 0 }
```

Encontrar un invariante y una función de cota para el bucle y verificar totalmente la función.