Aclaraciones: Se permite tener UNA hoja A4 con anotaciones durante el parcial. Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos.
Entregar cada ejercicio en una hoja separada, numerada y que incluya el nro. de orden.

Ejercicio 1. [25 puntos]

a) **[15 puntos]** Especificar el problema de decidir si, dados tres enteros positivos x, y y z, y un entero n, es cierto que existe entre x y z, un número p tal que $y = n + p$.

b) **[10 puntos]** Sean (P_1, Q_1) y (P_2, Q_2) especificaciones de dos algoritmos, tal que las variables libres de P_1 coinciden con las de P_2, y las de Q_1 con las de Q_2. Además sabemos que P_2 es más fuerte que P_1 y que Q_1 es equivalente a Q_2. ¿Dadas dos implementaciones I_1 e I_2, correspondientes a (P_1, Q_1) y (P_2, Q_2) respectivamente. Asumiendo que ninguna de las dos posee líneas de código redundantes, cuál de las dos implementaciones tendrá, a priori, más líneas de código.

Ejercicio 2. [25 puntos]

Especificar el problema de, dada una secuencia de enteros s y un natural n, determinar si s tiene al menos n elementos consecutivos entre sí, no necesariamente ubicados consecutivamente.

Por ejemplo, dados $n = 3$ y $s = \langle 3, 5, 1, 7, 4 \rangle$ la respuesta debería ser True (la secuencia tiene 3 elementos consecutivos entre sí: 3, 4 y 5), y dados $n = 4$ y $s = \langle 3, 5, 1, 7, 4 \rangle$ la respuesta debería ser False.

Ejercicio 3. [25 puntos]

Dada la siguiente especificación:

```
proc valorDeIndice (in l: seq(Z), in i: Z, out res : Bool) {
    Pre {0 ≤ i < length(l)}
    Post {res ↔ (∀j : Z)(j ≠ i ∧ 0 ≤ j < |l|) ⟶L l[j] = l[i] * j}
}
```

a) Implementar una función en imperativo que cumpla con la especificación.

b) Argumentar por qué el algoritmo propuesto implementa correctamente la especificación (no es necesario que propongan Pv, I, Fv y Qc).

Ejercicio 4. [25 puntos]

Dado la siguiente especificación y el siguiente programa:

Especificación

```
proc nivelDeAgenteFamoso (in a: Z, in b: Z, in c: Z, out res: Z) {
    Pre {a ≥ 0 ∧ b ≥ 0 ∧ c ≥ 0}
    Post {(todosCeros(a, b, c) → res = −1) ∧
          (esJamesBond(a, b, c) → res = 1) ∧
          (esMaxuelSmart(a, b, c) → res = 2) ∧
          (¬(todosCeros(a, b, c) ∨
          esJamesBond(a, b, c) ∨
          esMaxuelSmart(a, b, c)) → res = 0)}
    pred todosCeros (x: Z, y: Z, z: Z) {x = 0 ∧ y = 0 ∧ z = 0}
    pred esJamesBond (x: Z, y: Z, z: Z) {x = 0 ∧ y = 0 ∧ z = 7}
    pred esMaxuelSmart (x: Z, y: Z, z: Z) {
          (x = 0 ∧ y = 8 ∧ z = 6) ∨ (x = 8 ∧ y = 6 ∧ z = 0)}
}
```

Implementación

```
1  int nivelDeAgenteFamoso(int a, int b, int c){
2    int res = 0;
3    if (a == 0 && b == 0 && c == 7) {
4      res = 1;
5    } else {
6      if ((a == 0 && b == 8 && c == 6)
7            || (a == 8 && b == 6 && c == 0)) {
8        res = 2;
9      } else {
10       if (a + b + c == 0){
11         res = -1;
12       }
13     }
14   }
15   return res;
16 }
```

a) **[10 puntos]** Escribir un test suite que cubra todas las líneas del programa. Mostrar qué líneas cubre cada caso.

b) **[10 puntos]** Escribir un test suite que cubra todas las decisiones (branches) del programa. Mostrar qué decisiones cubre cada caso.

c) **[5 puntos]** ¿Es posible escribir para este programa un test suite que cubra todas las líneas pero no cubra todas las decisiones? En caso afirmativo, describirlo. En caso negativo, justificarlo.