UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructuras de Datos I
Segundo Recuperatorio– 9/12/2020

Aclaraciones: Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos.

Ejercicio 1. [30 puntos]

Sea el siguiente programa y su especificación:

```
    int contar(vector<vector<int> > ls){
L1:     int result = 0;
L2,3,4: for(int i=0; i < ls.size()-1; i++){
L5:         if(casiTodosDiez(ls[i], true)){
L6:             result = result + 1;
            }
        }
L7:     return result;
    }

    bool casiTodosDiez(vector<int> &v,
                       bool mirarPares){
L8:     int j;
L9:     if(mirarPares){
L10:        j = 0;
        } else {
L11:        j = 1;
        }
L12:    bool result = true;
L13:    while(j < v.size()){
L14:        if(v[j] != 10){
L15:            result = false;
            }
L16:        j = j + 2;
        }
L17:    return result;
    }
```

- **Cada caso de test propuesto debe contener la entrada y el resultado esperado.**

a) Describir el diagrama de control de flujo (control-flow graph) para la función contar y la función auxiliar casiTodosDiez.

b) Escribir un conjunto de casos de test (o *test suite*) que encuentre el defecto presente en el código (una entrada que cumple la precondición pero que el código no cumple la postcondición); y otro *test suite* que no lo encuentre. Explicar cuál es el defecto.

c) ¿Es posible escribir para la función contar un *test suite* que cubra todas las sentencias del programa (incluidas las de la función auxiliar)? En caso afirmativo, escribir el test suite y mostrar qué líneas cubre cada test; en caso negativo, justificarlo.

d) Escribir un *test suite* para la función casiTodosDiez que cubra todas sus sentencias pero no todas sus decisiones. Indicar explícitamente qué líneas cubre cada test.

proc contar (in ls: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$, out result: $\mathbb{Z}$) {
Pre $\{True\}$
Post $\{result = \sum_{j=0}^{|ls|-1}(\text{if } todosDiezEnPosicionesPares(ls[j]) \text{ then } 1 \text{ else } 0 \text{ fi})\}$
aux todosDiezEnPosicionesPares (s: $seq\langle\mathbb{Z}\rangle$) : Bool =
$(\forall i : \mathbb{Z})(0 \leq i < |s| \wedge\ i \mod 2 = 0) \longrightarrow_L s[i] = 10$;
}

Ejercicio 2. [30 puntos]

Dada la siguiente especificación:

proc esPiramide (in s: $seq\langle\mathbb{Z}\rangle$, out res: Bool) {
Pre $\{|s| \ mod\ 2 = 0 \wedge |s| > 0\}$
Post $\{res = True \Leftrightarrow (capicua(s) \wedge mitadCreciente(s))\}$

pred $capicua(s : seq\langle\mathbb{Z}\rangle)\{(\forall k : \mathbb{Z})(0 \leq k < |s| \longrightarrow_L s[k] = s[|s| - 1 - k])\}$
pred $mitadCreciente(s : seq\langle\mathbb{Z}\rangle)\{(\forall k : \mathbb{Z})(0 \leq k < |s|/2 - 1 \longrightarrow_L s[k] + 1 = s[k + 1])\}$
}

Dado el siguiente invariante:

$$0 \leq i \leq |s|/2 \wedge_L \Big(res = true \Leftrightarrow (\forall j : \mathbb{Z})(0 \leq j < i \longrightarrow_L (s[j] = s[|s|/2] - |s|/2 + j + 1 \wedge s[|s|/2 + j] = s[|s|/2] - j))\Big)$$

a) Escribir un programa en C++ que cumpla la especificación y que tenga un ciclo que **verifique el invariante**.

b) Explicar en palabras por qué el programa cumple la especificación y por qué el ciclo verifica el invariante.

Ejercicio 3. [40 puntos]

Dadas dos secuencias de enteros S y T sin repetidos y que no comparten valores, se dice que la posición i de S está *encerrada* por T si

$$encerrada(S,T,i) \equiv (\exists j,k:\mathbb{Z})(0 \leq j < |T| \wedge 0 \leq k < |T|) \wedge_L T[j] < S[i] < T[k]$$

Se desea contar la cantidad de posiciones de S encerradas por T, es decir,

$$\#encerradas(S,T) = \sum_{i=0}^{|S|-1} \textbf{if } encerrada(S,T,i) \textbf{ then } 1 \textbf{ else } 0$$

Por ejemplo,

$$\#encerradas([1,7,5,3],[6,2,4]) = 2$$

pues los valores 3 y 5 están *encerrados*: se cumple (por ejemplo) que $2 < 3 < 6$ y que $4 < 5 < 6$.

a) Dar un algoritmo que calcule $\#encerradas$ para dos secuencias S y T arbitrarias y calcular su complejidad algorítmica en función de las longitudes de S y T.

b) Suponiendo que S y T están ordenadas y tienen longitud n, dar un algoritmo alternativo que calcule $\#encerradas$ con complejidad **estrictamente menor** a $O(n)$. Jusitificar detalladamente la complejidad del algoritmo propuesto.