UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructuras de Datos I
Segundo parcial– 18/11/2020

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

Ejercicio 1. [35 puntos]

Sea el siguiente programa y su especificación:

```
    bool esEquilibrada(vector<int> s){
L1:     bool result = false;
L2:     int cant = 0;
L3:     int i = 0;
L4:     while(i < s.size()/2){
L5:         if (s[i] >= 0){
L6:             cant = cant + 1;
            } else {
L7:             cant = cant - 1;
            }
L8:         if (s[s.size()-1-i] >= 0){
L9:             cant = cant + 1;
            } else {
L10:            cant = cant - 1;
            }
L11:        i++;
        }
L12:    if (cant == 0){
L13:        result = true;
        }
L14:    return result;
    }
```

a) Describir el diagrama de control de flujo (control-flow graph) del programa.

b) ¿Es posible escribir para este programa un conjunto de casos de test (o *test suite*) que cubra todas las sentencias pero no cubra todas las decisiones? En caso afirmativo, escribir el test suite y mostrar qué líneas cubre cada test; en caso negativo, justificarlo.

c) Escribir un *test suite* que encuentre el defecto presente en el código (una entrada que cumple la precondición pero que el código no cumple la postcondición).

d) ¿Es posible escribir para este programa un *test suite* que cubra todas las decisiones pero que no encuentre el defecto en el código? En caso afirmativo, escribir el test suite; en caso negativo, justificarlo.

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

proc esEquilibrada (inout s: $seq\langle\mathbb{Z}\rangle$) {
Pre $\{True\}$
Post $\{result = True \leftrightarrow ((cantPositivas(s) - cantNegativas(s)) = 0)\}$
aux cantPositivas (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z} = \sum_{j=0}^{|s|-1}(\text{if } s[j] \geq 0 \text{ then } 1 \text{ else } 0 \text{ fi})$;
aux cantNegativas (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z} = \sum_{j=0}^{|s|-1}(\text{if } s[j] < 0 \text{ then } 1 \text{ else } 0 \text{ fi})$;
}

Ejercicio 2. [30 puntos]

Dada la siguiente especificación:

proc sumarConsecutivos (inout s: $seq\langle\mathbb{Z}\rangle$) {
Pre $\{s = S_0 \wedge |s| > 0\}$
Post $\{|s| = |S_0| \wedge_L (primeroIgual(s, S_0) \wedge seSumanDeAdos(s, S_0))\}$

pred $primeroIgual(s : seq\langle\mathbb{Z}\rangle, t : seq\langle\mathbb{Z}\rangle)\{(|s| > 0 \wedge |t| > 0) \longrightarrow_L s[0] = t[0]\}$
pred $seSumanDeAdos(s : seq\langle\mathbb{Z}\rangle, t : seq\langle\mathbb{Z}\rangle)\{(\forall j : \mathbb{Z})((1 \leq j < |s| \wedge |s| = |t|) \longrightarrow_L s[j] = t[j] + t[j-1])\}$
}

Dado el siguiente invariante:

$$|s| = |S_0| \wedge_L (0 \leq i < |s| \wedge_L ((\forall j : \mathbb{Z})(i < j < |s| \longrightarrow_L s[j] = S_0[j] + S_0[j-1]) \wedge (\forall k : \mathbb{Z})(0 \leq k \leq i \longrightarrow_L s[k] = S_0[k])))$$

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

b) Escribir un programa en C++ que cumpla la especificación y cuyo un ciclo **NO verifique el invariante**. Es decir, que no se pueda demostrar su correctitud utilizando el invariante.

Ejercicio 3. [35 puntos]

Se dice que una matriz M de dimensión $n \times n$ es *de zooms* si se cumplen las dos condiciones siguientes:

- $(\forall i : \mathbb{Z})(\forall j : \mathbb{Z})((0 \leq i < n \wedge 0 \leq j < n - 1) \longrightarrow_L M[i][j] < M[i][j + 1])$
- $(\forall i : \mathbb{Z})(1 \leq i < n \longrightarrow_L (\exists j : \mathbb{Z})(0 \leq j < n - 1 \wedge_L M[i - 1][j] < M[i][0] \wedge M[i][n - 1] < M[i - 1][j + 1]))$

a) Describir en castellano qué significa la propiedad enunciada, y dar un ejemplo de matriz *de zooms* para $n = 5$.

$$\begin{bmatrix} -11 & -8 & 9 \\ -5 & -1 & 6 \\ 1 & 2 & 5 \end{bmatrix}$$

Una matriz *de zooms* de 3×3.

b) Implementar en C++ la siguiente función:

```
bool buscarEnZooms(vector<vector<int>>& M, int elem, int& fi, int& co)
```

que toma una matriz *de zooms* y un elemento, y devuelve **`true`** en caso de que el elemento esté presente, y en ese caso además devuelve en los parámetros **`fi`** y **`co`** las coordenadas de ese elemento en la matriz (fila y columna, respectivamente). El algoritmo propuesto debe tener complejidad temporal de peor caso $O(n)$.

c) Justificar detalladamente por qué el algoritmo cumple con la complejidad pedida.