UBA Facultad de Ciencias Exactas y Naturales Departamento de Computación Algoritmos y Estructura de Datos I

Primer parcial 12/05/2017

1		2	3		4			TOTAL
a	b		a	b	a	b	c	
15		23	25		25			88

(A)

Nro. de orden:
LU:
Apellidos:
Nombres:

Aclaraciones: Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos. **Entregar cada ejercicio en hoja separada.**

Ejercicio 1. [25 puntos]

a) **[15 puntos]** Especificar el problema de decidir si, dados tres enteros x, y y z, y un entero $n > 2$, es cierto que para todo entero positivo j menor que n se cumple que la suma de x^j e y^j es igual a z^j.

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. ¿Es cierto que todo algoritmo que satisface la especificación (P_1, Q_1) satisface también (P_2, Q_2)? ¿Y al revés? Justificar.

Ejercicio 2. [25 puntos]

Especificar el problema de, dada una secuencia de enteros s y un natural n, determinar si s tiene al menos 2 picos de altura n. Un pico de altura n es una posición de la secuencia que tiene al menos n elementos ordenados de manera creciente anteriores a él, y al menos n elementos ordenados de manera decreciente luego de él. ESTRICTAM.

Por ejemplo, dados $n = 1$ y $s = \langle 3, 5, 1, 7, 5 \rangle$ la respuesta debería ser True (la secuencia tiene dos picos de altura 1 en las posiciones 1 y 3), y dados $n = 3$ y $s = \langle 1, 2, 3, 5, 4, 1, 0, 3, 5, 6 \rangle$ la respuesta debería ser False (ya que la secuencia tiene un único pico de altura 3 en la posición 3).

Ejercicio 3. [25 puntos]

Dada la siguiente especificación:

```
proc elMenosRepetido (in l: seq⟨ℤ⟩, out res : ℤ) {
    Pre {length(l) > 0}
    Post {res ∈ l ∧ (∀x : ℤ)(x ∈ l → #apariciones(l, res) ≤ #apariciones(l, x))}
}
```

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

b) Argumentar por qué el algoritmo propuesto implementa correctamente la especificación.

Ejercicio 4. [25 puntos]

Dado la siguiente especificación y el siguiente programa:

Especificación

```
proc superMaximo (in a: ℤ, in b: ℤ, in c: ℤ, out res: ℤ) {
    Pre {True}
    Post {esMayor(a, b, c) → res = a ∧
          esMayor(b, a, c) → res = b ∧
          esMayor(c, a, b) → res = c}
    pred esMayor (m: ℤ, x: ℤ, y: ℤ) {m ≥ x ∧ m ≥ y}
}
```

Implementación

```
1  int superMaximo(int a, int b, int c){
2    int res = c;
3    if (a >= b && a >= c) {
4      res = a;
5    } else {
6      if (b > c) {
7        res = b;
8      }
9    }
10   return res;
11 }
```

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.

EJERCICIO 1

a) proc exponentesMenoresCumplenIgualdad(in x,y,z,m:$\mathbb{Z}$, out result : bool){

Pre{ $m > 2$ }

Post{ $result = (\forall j : \mathbb{Z})\ 0 < j < m \rightarrow_L x^j + y^j = z^j$ }

Aclaración: decidí que j no fuera 0 (cero) ya que debe ser "entero positivo" y no considero positivo al 0.

b) No es cierto que todo algoritmo que cumpla la especificación de (P_1, Q_1) satisfaga (P_2, Q_2), ya que P_1 es más débil que P_2. En cambio, sí vale al revés. Como Q_1 y Q_2 son equivalentes y P_2 siempre implica a P_1, todo algoritmo que satisfaga (P_2, Q_2) cumplirá con (P_1, Q_1).

Al revés.

EJERCICIO 2:

```
proc alMenosDosPicos (in s: seq<Z>, in n:Z, out result: bool){
  Pre{ n>0}
  Post{ result = (cantidadDePicos(s,n) ≥ 2)}
}

fun cantidadDePicos ( s: seq<Z>, n:Z ): Z =
  Σ_{i=n}^{|s|-1-n}  if ( esPico(s, n, i)) then 1 else 0 fi;

pred esPico( s:seq<Z>, n:Z, i:Z){
  esCreciente( subseq(s, i-n, i)) ∧
  esDecreciente( subseq(s, i+1, i+1+n))
}
pred esCreciente (l:seq<Z>){
  (∀j:Z) 0≤j<|l|-1 →L l[j] < l[j+1]
}

pred esDecreciente (l:seq<Z>){
  (∀j:Z) 0≤j<|l|-1 →L l[j] > l[j+1]
}
```

i podría no ser mayor que los elementos de al lado.

EJERCICIO 3:

a)
```
int elMenosRepetido (vector<int> l){
    int menosRepetidoParcial = l[0];
    int i = 0;
    while (i<l.size()){
        if (apariciones(l, menosRepetidoParcial) > apariciones(l,l[i]))
        {
        menosRepetidoParcial = l[i];
        }
        i++;
    }
    return menosRepetidoParcial;
}
int apariciones(vector<int> l, int e){
    int aparicionesParcial = 0;
    int i = 0;
    while (i<l.size()){
        if (l[i] == e){
            aparicionesParcial++;
        }
        i++;
    }
    return aparicionesParcial;
}
```

b) En la primera función, ala variable que se devuelve siempre se le asigna un valor perteneciente a la lista, por lo tanto cumple que "res ∈ l". Además, el ciclo recorre todas las posiciones y compara las apariciones de sus elementos correspondientes en la lista con las del elemento menos repetido hasta el momento.

La segunda función se encarga de aumentar de a 1 una variable acumuladora cada vez que encuentra el valor ingresado en la lista.

Las funciones en conjunto aseguran que para el valor devuelto, las apariciones de los otros elementos sean más o la misma cantidad que las de él.

EJERCICIO 4:

Líneas

a)
```
TEST(superMaximoTest, ramaPositiva){
    int a = 4;
    int b = 3;
    int c = 2;
    ASSERT_EQ( a , superMaximo(a,b,c));
}
```
✓

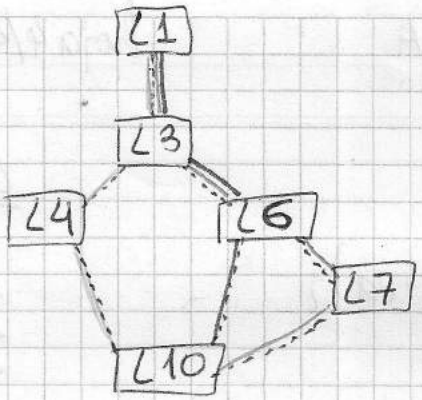
```
TEST (superMaximoTest, ramaNegativaConIf){
    int a = 3;
    int b = 4;
    int c = 2;
    ASSERT_EQ( b, superMaximo (a, b, c));
}
```
✓

Los "{" no son necesarios porque no se ejecutan.

b) A los anteriores, agrego:

```
TEST(superMaximoTest, ramaNegativaSinIf){
    int a = 2;
    int b = 3;
    int c = 4;
    ASSERT_EQ (c, superMaximo(a,b,c))
}
```
✓

DIBUJO LAS DECISIONES DEL OTRO LADO DE LA HOJA →



— rama Positiva
— rama Negativa con If
— rama Negativa Sin If.

✓

c) Es posible escribir un test suite que cubra las líneas pero no las decisiones. Un ejemplo está dado en (a). En ese caso falta la decisión correspondiente a no cumplir la guarda de la línea 6. (b>c) ✓