Ingeniería Informática - Depto. de Informática e Ingeniería de Sistemas
Examen de Programación 2 - 8 de junio de 2016

- Disponed sobre la mesa en lugar visible un ***documento de identificación*** provisto de fotografía. Escribid ***nombre y dos apellidos*** en cada una de las hojas de papel que haya sobre la mesa.
- Comenzad a resolver cada una de las partes del examen ***en una hoja diferente*** para facilitar su corrección por profesores diferentes.
- El tiempo total previsto para realizar el examen es de ***dos horas y media***. No está permitido consultar libros ni apuntes, excepto los dos documentos señalados en la convocatoria del examen (Guía sintáxis C++ y Apartado 1.2 de los apuntes del curso).

Problema 1º (2 puntos)

A continuación se presenta un diseño iterativo de la función **`multiplicar`**`(a,b)`.

```
/*
 * Pre: b >= 0
 * Post: multiplicar (a,b) = a * b
 */
int multiplicar (const int a, const int b) {
    int multiplicando = a, multiplicador = b;        // linea a
    int r = 0;                                        // linea b
    while ( multiplicador != 0) {                     // linea c
        if ( multiplicador % 2 == 1) {                // linea d
            r = r + multiplicando;                    // linea e
        }
        multiplicador = multiplicador / 2;            // linea f
        multiplicando = 2 * multiplicando;            // linea g
    }
    return r;                                         // linea h
}
```

En este problema se pide analizar el coste de ejecutar una invocación **`multiplicar`**`(a,b)` de la función anterior. Para ello hay que:

- Indicar de qué parámetro de la función o combinación de parámetros depende su coste en tiempo.
- Explicar en qué circunstancias y para qué valores de los parámetros se presentan los casos asintóticamente peores y los casos asintóticamente mejores, en cuanto a su coste en tiempo.
- Deducir, paso a paso, las funciones de coste en los casos peores y mejores.
- Caracterizar asintóticamente cada una de las funciones de coste determinadas en el punto anterior.

Problema 2º (3 puntos)

A continuación se presenta un diseño iterativo de la función **`dividir`**`(a,b,cociente,resto)`.

```
/*
 * Pre:  a >= 0 AND b > 0
 * Post:  cociente  = a / b AND resto = a  % b
 */
void dividir (const int a, const int b, int& cociente, int& resto) {
    cociente = 0;   resto = a;
    while (resto >= b) {
        resto = resto - b;   cociente = cociente + 1;
    }
}
```

En este problema se pide demostrar formalmente la corrección del diseño. Se presentarán, anotadas sobre el propio código o escritas aparte, las pruebas que permiten concluir la corrección de la totalidad de su código, sin olvidar la prueba de la terminación del bucle.

Problema 3º (2.5 puntos)

El comportamiento de la función **`iesimo`**`(v,i)` se especifica a continuación. El dato **`DIM`** es una constante entera y positiva definida previamente.

```
// Número de componentes de los vectores con los que se va a trabajar (DIM > 0)
const int DIM = ...;

/*
 * Pre:  ((Núm alfa EN [0,DIM−1]. v[alfa] > 0) >= i) AND i > 0
 * Post:  iesimo(v, i) = INDICE AND INDICE >= 0 AND v[INDICE] > 0.0 AND
 *          ((NUM alfa EN [0,INDICE]. v[alfa] > 0.0) = i)
 */
int iesimo (const double v[], const int i);
```

En este problema se pide hacer un diseño sin bucles de la función **`iesimo`**`(v,i)`. En caso de ser precisa una inmersión, explicar la técnica de inmersión que ha sido aplicada y tener presente que la calificación del problema estará en función de la correccíon del diseño de la función o funciones que constituyan la solución.

Problema 4º (2.5 puntos)

El listado que se presenta a continuación corresponde a un diseño iterativo de la función **`todosDiferentes`**`(v,n)`.

```
/*
 * Pre: n > 0
 * Post: todosDiferentes (v,n) =
 *           (PT alfa EN [0,n-2].(PT beta EN [alfa+1,n-1]. v[alfa] != v[beta]))
 */
template <typename T>
bool todosDiferentes (const T v[], const int n) {
    bool sonDistintos = true;
    int i = 0;
    // I1
    while ( sonDistintos && i != n - 1) {
        // I1
        int j = i + 1;
        // I2
        while ( sonDistintos && j != n) {
            // I2
            if (v[i] == v[j]) {
                sonDistintos = false;
            }
            else {
                j = j + 1;
            }
            // I2
        }
        i = i + 1;
        // I1
    }
    return sonDistintos;
}
```

Se pide escribir los predicados **`I1`** y **`I2`** que sean invariantes del bucle externo y del bucle interno, respectivamente, y que sean suficientemente fuertes (restrictivos) como para poder sustentar la prueba formal de la corrección de todo el código de la función.

Observación; No se pide aportar ninguna prueba formal de la corrección del código de la función.

Una solución del problema 1º

Analisis del coste de ejecutar una invocación **`multiplicar`**`(a,b)`:

El coste depende del valor del parámetro **`b`**.

Los casos mejores se presentan cuando el valor de del parámetro **`b`** es una potencia de 2 (un valor $100\ldots00$ expresado en base 2) ya que, en tales caso, la condición de la línea **`d`** solo se satisface en la última iteración del bucle ***`while`***.

Los casos peores se presentan cuando el valor de del parámetro **`b`** es una unidad inferior a una potencia de 2 (un valor $111\ldots11$ expresado en base 2) ya que, en tales caso, la condición de la línea **`d`** se satisface en todas las iteraciones del bucle ***`while`***.

- Función de coste en los casos mejores:

  $t(b) = t_a + t_b + t_c + (\Sigma\alpha \in [1, log_2\, b - 1].t_d + t_f + t_g + t_c) + t_d + t_e + t_f + t_g + t_c + t_h$

  $t(b) = t_a + t_b + t_c + (\Sigma\alpha \in [1, log_2\, b].t_d + t_f + t_g + t_c) + t_e + t_h$

  $t(b) = (t_d + t_f + t_g + t_c) \times log_2\, b + t_a + t_b + t_c + t_e + t_h$

  Caracterización asintótica de la función de coste en los casos mejores:

  $\boldsymbol{O}(t(b)) = \boldsymbol{O}((t_d + t_f + t_g + t_c) \times log_2\, b + t_a + t_b + t_c + t_e + t_h) = \boldsymbol{O}(log_2\, b) = \boldsymbol{O}(log\, b)$

- Función de coste en los casos peores:

  $t(b) = t_a + t_b + t_c + (\Sigma\alpha \in [1, log_2\, b].t_d + t_e + t_f + t_g + t_c) + t_h$

  $t(b) = (t_d + t_e + t_f + t_g + t_c) \times log_2\, b + t_a + t_b + t_c + t_h$

  Caracterización asintótica de la función de coste en los casos peores:

  $\boldsymbol{O}(t(b)) = \boldsymbol{O}((t_d + t_f + t_g + t_c) \times log_2\, b + t_a + t_b + t_c + t_e - t_f + t_h) = \boldsymbol{O}(log_2\, b) = \boldsymbol{O}(log\, b)$

Una solución del problema 2º

Demostración formal de la corrección del diseño anterior. Las pruebas que constituyen la demostración se han anotado sobre el código de la función.

```
/*
 * Pre: a >= 0 AND b > 0
 * Post: cociente = a / b AND resto = a % b
 */
void dividir (const int a, const int b, int& cociente, int& resto) {
    // a >= 0 AND b > 0
    // => [1]
    // a >= 0 AND b > 0 AND a >= 0 AND a = 0 * b + a
    cociente = 0;  resto = a;
    // Invariante del bucle: a >= 0 AND b > 0 AND resto >= 0 AND a = cociente * b + resto
    while (resto >= b) {
        // f_cota_antes = resto = A
        // a >= 0 AND b > 0 AND resto >= b AND resto >= 0 AND a = cociente * b + resto
        // => [2]
        // a >= 0 AND b > 0 AND resto - b >= 0 AND a = (cociente + 1) * b + resto - b
        resto = resto - b;  cociente = cociente + 1;
        // f_cota_despues = resto = A - b
        // Este bucle termina ya que:
        //    1) Decrece en cada iteración: Invariante => b > 0 => A > A - b y
        //
2) La función de cota propuesta está acotada inferiormente: Invariante => resto >= 0 => f_cota >= 0
    }
    // a >= 0 AND b > 0 AND resto < b AND resto >= 0 AND a = cociente * b + resto
    // => [3]
    // cociente = a / b AND resto = a % b
}
```

Justificación de las tres pruebas anotadas para demostrar la corrección de diferentes elementos del código anterior:

1 Resulta obvio que el primer predicado, $a >= 0 \wedge b > 0$, garantiza la satisfacción del segundo, $a >= 0 \wedge b > 0 \wedge a >= 0 \wedge a = 0 * b + a$.

2 Si simplificamos la condición $a = (cociente + 1) * b + resto - b \equiv a = cociente * b + resto$ del segundo predicado, identificamos la misma condición $a = cociente * b + resto$ que figura en el primer predicado. La satisfacción de las restantes condiciones del segundo predicado, $a >= 0 \wedge b > 0 \wedge resto - b >= 0$, resulta evidente sin más que observar las condiciones exigidas por el primer predicado, $a >= 0 \wedge b > 0 \wedge resto >= b$.

3 Las condiciones $resto < b \wedge resto >= 0 \wedge a = cociente * b + resto$ del primer predicado garantizan que el valor de $cociente$ es la división entera a/b y que el valor de $resto$ es el resto de dicha división entera, ya que el valor del dividendo, a, es cero o positivo y del divisor, b, es positivo.

Una solución del problema 3º

Solución resultante de un diseño recursivo por inmersión mediante debilitamiento de la postcondición de la función **iesimo**(v,i).

```
/*
 * Pre:  ((Núm alfa EN [desde,DIM−1]. v[alfa] > 0) >= i) AND i > 0
 * Post:  iesimo(v,desde,i)  = INDICE AND INDICE >= desde AND v[INDICE] > 0.0
 *          AND ((Núm alfa EN [desde,INDICE]. v[alfa]  <= 0.0) = i)
 */
int iesimo (const double v[], const int desde, const int i) {
    if (v[desde] > 0) {
        if (i == 1) {
            return desde;
        }
        else {
            return iesimo(v, desde + 1, i − 1);
        }
    }
    else {
        return iesimo(v, desde + 1, i);
    }
}

/*
 * Pre:  ((Núm alfa EN [0,DIM−1]. v[alfa] > 0) >= i) AND i > 0
 * Post:  iesimo(v,i)  = INDICE AND INDICE >= 0 AND v[INDICE] > 0.0
 *          AND ((NUM alfa EN [0,INDICE]. v[alfa] > 0.0)  = i)
 */
int iesimo (const double v[], const int i) {
    return iesimo(v, 0, i);
}
```

Una solución del problema 4º

En el código de la función **`todosDiferentes`**`(v,n)` se han anotado los predicados invariantes **I1** y **I2** más fuertes asociados a su bucle externo y a su bucle interno, respectivamente.

```
/*
 * Pre: n > 0
 * Post: todosDiferentes (v,n) =
 *          (PT alfa EN [0,n−2].(PT beta EN [alfa+1,n−1]. v[alfa] != v[beta]))
 */
template <typename T>
bool todosDiferentes (const T v[], const int n) {
    bool sonDistintos = true;
    int i = 0;
    while ( sonDistintos && i != n − 1) {
        // I1: i >= 0 AND i <= n − 1 AND
        //       sonDistintos =
        //          (PT alfa EN [0,i−1].(PT beta EN [alfa+1,n−1]. v[alfa] != v[beta]))
        int j = i + 1;
        while ( sonDistintos && j != n) {
            // I2: i >= 0 AND i < n − 1 AND j >= i + 1 AND j <= n AND
            //       sonDistintos =
            //          (PT alfa EN [0,i−1].(PT beta EN [alfa+1,n−1]. v[alfa] != v[beta]))
            //          AND (PT beta EN [i+1,j−1]. v[i] != v[beta])
            if (v[i] == v[j]) {
                sonDistintos = false;
            }
            else {
                j = j + 1;
            }
        }
        i = i + 1;
    }
    return sonDistintos;
}
```