Ingeniería Informática - Depto. de Informática e Ingeniería de Sistemas
Examen de Programación 2 - 4 de Junio de 2015

Problema 1º (4 puntos)

En este problema hay que probar la corrección del codigo de la función ***anular****(v,n)*. Para ello se pide:

1. Escribir un predicado invariante asociado al bucle suficientemente fuerte como para sustentar las demostraciones que se piden a continuación.
2. Demostrar la corrección del código que precede al bucle, del código que le sigue y del código que se ejecuta cada vez que se itera el bucle.
3. Demostrar la terminación del bucle.

```
/*
 * Pre:  v = Vo AND n >= 0
 * Post: anulados = (NUM alfa EN [0,n-1].Vo[alfa]<0)
 *         AND (PT alfa EN [0,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
 *                                 (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
 */
void anular (double v[], int n, int& anulados) {
    anulados = 0;
    int i = n - 1;
    while (i >= 0) {
        if (v[i] < 0.0) {
            v[i] = 0.0;
            anulados = anulados + 1;
        }
        i = i - 1;
    }
}
```

Problema 2º (2 puntos)

El coste de ejecutar la función ***esPrimo****(n)* depende del valor del parámetro ***n***. En este problema se pide:

1. Explicar claramente en qué circunstancias se presentan los casos mejores y los casos peores, en cuanto al coste en tiempo, al ejecutar la función ***esPrimo****(n)* para un valor de n suficientemente grande en valor absoluto (positivo o negativo).
2. Deducir la función de coste en tiempo, $t_{esPrimo(n)}(n)$, en los casos extremos, casos mejores y casos peores, asumiendo que el coste de ejecutar cada una de sus líneas de código etiquetadas por a, b, c, ... es igual a t_a, t_b, t_c, ..., respectivamente. Se valorará la claridad y el detalle con que se presenten los cálculos realizados.
3. Caracterizar asíntóticamente la función de coste, en tiempo, en los casos extremos, es decir, los casos mejores y peores, mediante la notación $\boldsymbol{O}(t_{esPrimo(n)}(n))$.

```
/*
 * Pre: cierto
 * Post: esPrimo(n) = (n >= 2) AND (PT alfa EN [2,n-1].n % alfa > 0)
 */
bool esPrimo (int n) {
    if (n==2) {                                          /* a */
        return true;                                     /* b */
    }
    else if (n<2 || n%2==0) {                            /* c */
        return false;                                    /* d */
    }
    else {
        int divisor = 3;                                 /* e */
        bool loParece = true;                            /* f */
        while (loParece && divisor*divisor<=n) {         /* g */
            loParece = n%divisor>0; divisor = divisor + 2;   /* h */
        }
        return loParece;                                 /* i */
    }
}
```

Problema 3º (2 puntos)

En un programa se ha desarrollado la función ***cuentaMenores****(v,n,dato,cuenta)* con la siguiente especificación.

```
/*
 * Pre: n >= 0 AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1])
 * Post: cuenta >= 0 AND cuenta <= n AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1]) AND
 *       (PT alfa EN [0,cuenta-1].v[alfa]<dato) AND (PT alfa EN [cuenta,n-1].v[alfa]>=dato)
 */
void cuentaMenores (const int v[], const int n, const int dato, int &cuenta)
```

Se pide probar formalmente la corrección del siguiente fragmento de código.

```
    /* n >= 500 AND dato % 2 = 0 AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1]) */
    cuentaMenores(v, n, dato, i);
    if (i<n) {
        v[i] = dato;
    }
    else {
        v[n-1] = dato;
    }
    /* (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1])) AND (EX alfa EN [0,n-1].v[alfa]=dato) */
```

Problema 4º (2 puntos)

Diseñar sin bucles la función ***esPrimo****(n)* con la misma especificación presentada en el segundo problema de este examen. El diseño de la función pedida y, en su caso de las funciones auxiliares, deberá ser correcto. La corrección de estas depende, esencialmente, de su adecuada especificación.

Una solución del problema 1º

Invariante del bucle, prueba de la terminación del bucle y pruebas de la correción del código que precede y que sigue al bucle.

```
/*
 * Pre: v = Vo AND n >= 0
 * Post: anulados = (NUM alfa EN [0,n-1].Vo[alfa]<0)
 *        AND (PT alfa EN [0,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
 *                                  (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
 */
void anular (double v[], int n, int& anulados) {
    /*
     * (PT alfa EN [0,n-1].v[alfa]=Vo[alfa]) AND n >= 0
     * =>
     * n >= 0 AND n-1 >= -1 AND n-1 <= n-1 AND 0 = (NUM alfa EN [n,n-1].Vo[alfa]<0)
     * AND (PT alfa EN [0,n-1].v[alfa]=Vo[alfa])
     * AND (PT alfa EN [n,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
     *                              (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
     */
    anulados = 0;
    int i = n - 1;
    /*
     * Inv: n >= 0 AND i >= -1 AND i <= n-1
     *       AND anulados = (NUM alfa EN [i+1,n-1].Vo[alfa]<0)
     *       AND (PT alfa EN [0,i].v[alfa]=Vo[alfa])
     *       AND (PT alfa EN [i+1,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
     *                                  (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
     */
    while (i >= 0) {              /* f_cota(i) = i */
        /* i = A AND f_cota(antes) = A */
        if (v[i] < 0.0) {
            v[i] = 0.0;   anulados = anulados + 1;
        }
        i = i - 1;
        /* i = A - 1 AND f_cota(después) = A - 1 */
        /*
         * Prueba de la terminación del bucle:
         * - Condición 1: La función de cota decrece en cada iteración ya que
         *       f(antes) > f(después), es decir, A > A - 1
         * - Condición 2: El valor de la función de cota está acotado inferiormente por -1
         *       Inv: n >= 0 AND i >= -1 AND .... => f_cota(i) = i, es decir,
         *       f_cota(i) >= -1
         */
    }
    /*
     * n >= 0 AND i < 0 AND i >= -1 AND i <= n-1
     * AND anulados = (NUM alfa EN [i+1,n-1].Vo[alfa]<0)
     * AND (PT alfa EN [0,i].v[alfa]=Vo[alfa])
     * AND (PT alfa EN [i+1,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
     *                              (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
     * =>
     * anulados = (NUM alfa EN [0,n-1].Vo[alfa]<0)
     * AND (PT alfa EN [0,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
     *                              (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
     */
}
```

Prueba de la correción del código a iterar en el bucle.

```
    while (i >= 0) {
        /*
         * n >= 0 AND i >= -1 AND i <= n-1
         * AND anulados = (NUM alfa EN [i+1,n-1].Vo[alfa]<0)
         * AND (PT alfa EN [0,i].v[alfa]=Vo[alfa])
         * AND (PT alfa EN [i+1,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
         *                           (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
         */
        if (v[i] < 0.0) {
            /*
             * n >= 0 AND i >= -1 AND i <= n-1
             * AND anulados = (NUM alfa EN [i+1,n-1].Vo[alfa]<0)
             * AND (PT alfa EN [0,i].v[alfa]=Vo[alfa]) AND v[i] < 0
             * AND (PT alfa EN [i+1,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
             *                           (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
             * =>
             * n >= 0 AND i-1 >= -1 AND i-1 <= n-1
             * AND anulados + 1 = (NUM alfa EN [i,n-1].Vo[alfa]<0)
             * AND (PT alfa EN [0,i-1].v[alfa]=Vo[alfa])
             * AND (Vo[i]<=0 -> 0 = 0) AND (Vo[i]>=0 -> 0 = Vo[alfa])
             * AND (PT alfa EN [i+1,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
             *                           (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
             */
            v[i] = 0.0;
            anulados = anulados + 1;
        }
        else {
            /*
             * n >= 0 AND i >= -1 AND i <= n-1
             * AND anulados = (NUM alfa EN [i+1,n-1].Vo[alfa]<0)
             * AND (PT alfa EN [0,i].v[alfa]=Vo[alfa]) AND v[i] >= 0
             * AND (PT alfa EN [i+1,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
             *                           (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
             * =>
             * n >= 0 AND i-1 >= -1 AND i-1 <= n-1
             * AND anulados = (NUM alfa EN [i,n-1].Vo[alfa]<0)
             * AND (PT alfa EN [0,i-1].v[alfa]=Vo[alfa])
             * AND (PT alfa EN [i,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
             *                         (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
             */
        }
        i = i - 1;
        /*
         * Inv: n >= 0 AND i >= -1 AND i <= n-1
         *      AND anulados = (NUM alfa EN [i+1,n-1].Vo[alfa]<0)
         *      AND (PT alfa EN [0,i].v[alfa]=Vo[alfa])
         *      AND (PT alfa EN [i+1,n-1].(Vo[alfa]<=0 -> v[alfa] = 0) AND
         *                                (Vo[alfa]>=0 -> v[alfa] = Vo[alfa]) )
         */
    }
}
```

Una solución del problema 2º

Los casos mejores y peores, en cuanto a coste, para un valor del parámetro n suficientemente grande se dan en los siguientes casos:

- **Casos mejores**. Cuando el valor de n es negativo o es par. En tales casos la función de coste es:

  $t(n) = t_a + t_c + t_d$

  La caracterización asintotica de la función de coste corresponde al orden de una función constante:

  $\boldsymbol{O}(t(n)) = \boldsymbol{O}(t_a + t_c + t_d) = \boldsymbol{O}(1)$

- **Casos peores**. Cuando el valor de n es un número primo. En tales casos la función de coste es:

  $t(n) = t_a + t_c + t_e + t_f + t_g + (\Sigma\alpha \in [1, k].t_h + t_g) + t_i$

  Donde el número k de iteraciones es, para n suficientemente grande, aproximadamente igual a $\frac{\sqrt{n}}{2}$. Por lo tanto.

  $t(n) = t_a + t_c + t_e + t_f + t_g + \frac{1}{2}(t_h + t_g)\sqrt{n} + t_i$

  La caracterización asintotica de la función de coste corresponde al orden de la función $\sqrt{n}$:

  $\boldsymbol{O}(t(n)) = \boldsymbol{O}(t_a + t_c + t_e + t_f + t_g + \frac{1}{2}(t_h + t_g)\sqrt{n} + t_i) = \boldsymbol{O}(\sqrt{n})$

Una solución del problema 3º

```
/*
 *  n >= 500 AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1])
 *  =>
 *  n >= 0 AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1])
 */
cuentaMenores(v, n, dato, i);
/*
 *  n >= 0 AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1]) AND i >= 0 AND i <= n AND
 * (PT alfa EN [0,i-1].v[alfa]<dato) AND (PT alfa EN [i,n-1].v[alfa]>=dato)
 */
if (i<n) {
    /*
     * n >= 0 AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1]) AND i >= 0 AND i < n AND
     * (PT alfa EN [0,i-1].v[alfa]<dato) AND (PT alfa EN [i,n-1].v[alfa]>=dato)
     * =>
     * (PT alfa EN [0,i-2].v[alfa]<=v[alfa+1]) AND v[i-1]<=dato AND dato<=v[i+1] AND
     * (PT alfa EN [i+1,n-2].v[alfa]<=v[alfa+1])
     */
    v[i] = dato;
}
else {
    /*
     * n >= 0 AND (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1]) AND i = n AND
     * (PT alfa EN [0,i-1].v[alfa]<dato) AND (PT alfa EN [i,n-1].v[alfa]>=dato)
     * =>
     * (PT alfa EN [0,n-3].v[alfa]<=v[alfa+1]) AND v[n-2]<=dato AND
     * ((EX alfa EN [0,n-2].v[alfa]=dato) OR dato = dato)
     */
    v[n-1] = dato;
}
/*
 *  (PT alfa EN [0,n-2].v[alfa]<=v[alfa+1]) AND (EX alfa EN [0,n-1].v[alfa]=dato)
 */
```

Una solución del problema 4º

Se va a plantear un diseño recursivo por inmersión mediante refuerzo de la precondición.

```
/*
 *  Pre: n > 2 AND divisor % 2 = 1 AND divisor >= 3
 *       AND (PT alfa EN [2, divisor −1].n % alfa > 0)
 *  Post: esPrimo(n, divisor ) = (n >= 2) AND (PT alfa EN [2,n−1].n % alfa > 0)
 */
bool esPrimo (int n, int divisor ) {
    if ( divisor * divisor <=n) {
        if (n % divisor==0) {
            return false ;
        }
        else {
            return esPrimo(n, divisor + 2);
        }
    }
    else {
        return true;
    }
}

/*
 *  Pre: cierto
 *  Post: esPrimo(n) = (n >= 2) AND (PT alfa EN [2,n−1].n % alfa > 0)
 */
bool esPrimo (int n) {
    if (n==2) {
        return true;
    }
    else if (n<2 || n%2==0) {
        return false ;
    }
    else {
        return esPrimo(n ,3);
    }
}
```