Programación 2

Tema 1. Especificación formal de algoritmos (I)

Problemas 01

En cada uno de los ejercicios que siguen se debe sustituir cada especificación no formal por una especificación formal escribiendo para ello un par de predicados matemáticos:

- su predicado **precondición** y
- su predicado **postcondición**

Ejercicio ilustrativo

```
/*
 *  Devuelve la suma de los dígitos del número natural
 *  <n>, es decir, la suma de sus cifras cuando se
 *  expresa en base 10
 */
int sumarCifras (const int n);
```

Una solución del ejercicio anterior:

```
/*
 *  Devuelve la suma de los dígitos del número natural
 *  <n>, es decir, la suma de sus cifras cuando se
 *  expresa en base 10
 */
int sumarCifras (const int n);
```

```
/*
 *  Pre: n ≥ 0
 *  Post: sumarCifras(n) = (Σα∈[1,∞]. n / 10^(α-1) % 10)
 */
int sumarCifras (const int n);
```

Ejercicio 1º

```
/*
 *  Devuelve el valor de la media aritmética de <x>, <y>
 *  y <z>
 */
double mediaAritmetica (const double x, const double y,
                        const double z);
```

```
/*
 *  Devuelve el valor de la media aritmética de <x>, <y>
 *  y <z>
 */
double mediaAritmetica (const double x, const double y,
                        const double z);
```

```
/*
 *  Pre: cierto
 *  Post: mediaAritmetica(x,y,z) = (x + y + z) / 3.0
 */
double mediaAritmetica (const double x, const double y,
                        const double z);
```

Ejercicio 2º

```
/*
 *  Devuelve un valor <cierto> si y sólo si el natural
 *  no nulo <b> es un divisor del natural <a>
 */
bool esDivisor (const int a, const int b);
```

```
/*
 *  Devuelve un valor <cierto> si y sólo si el natural
 *  no nulo <b> es un divisor del natural <a>
 */
bool esDivisor (const int a, const int b);
```

```
/*
 *  Pre: a ≥ 0 ∧ b > 0
 *  Post: esDivisor(a,b) = (a % b = 0)
 */
bool esDivisor (const int a, const int b);
```

Ejercicio 3º

```
/*
 *  Devuelve un valor del intervalo no vacío [a,b]
 */
int intermedio (const int a, const int b);
```

```
/*
 *  Devuelve un valor del intervalo no vacío [a,b]
 */
int intermedio (const int a, const int b);
```

```
/*
 *  Pre: a ≤ b
 *  Post: intermedio(a,b) ≥ a ∧ intermedio(a,b) ≤ b
 */
int intermedio (const int a, const int b);
```

```
/*
 *  Devuelve un valor del intervalo no vacío [a,b]
 */
int intermedio (const int a, const int b);
```

```
/*
 *  Pre: a ≤ b
 *  Post: intermedio(a,b) = X  ∧  X ≥ a  ∧  X ≤ b
 */
int intermedio (const int a, const int b);
```

Ejercicio 4º

```
/*
 * Devuelve el mayor de los valores de los parámetros
 * <a>, <b> y <c>
 */
int mayor (const int a, const int b, const int c);
```

```
/*
 * Devuelve el mayor de los valores de los parámetros
 * <a>, <b> y <c>
 */
int mayor (const int a, const int b, const int c);
```

```
/*
 *  Pre: cierto
 *  Post: (a ≥ b ∧ a ≥ c → mayor(a,b,c) = a) ∧
 *        (b ≥ a ∧ b ≥ c → mayor(a,b,c) = b) ∧
 *        (c ≥ a ∧ c ≥ b → mayor(a,b,c) = c)
 */
int mayor (const int a, const int b, const int c);
```

```
/*
 * Devuelve el mayor de los valores de los parámetros
 * <a>, <b> y <c>
 */
int mayor (const int a, const int b, const int c);
```

```
/*
 *  Pre: cierto
 *  Post: mayor(a,b,c) = (Máx α∈{a,b,c}.α)
 */
int mayor (const int a, const int b, const int c);
```

Ejercicio 5º

```
/*
 *  Devuelve el número de cifras del número natural <n>
 *  cuando se expresa en base 10
 */
int numCifras (const int n);
```

```
/*
 *  Devuelve el número de cifras del número natural <n>
 *  cuando se expresa en base 10
 */
int numCifras (const int n);
```

```
/*
 *  Pre: n ≥ 0
 *  Post: (n = 0 → numCifras(n) = 1) ∧
 *        (n > 0 → 10 numCifras(n)-1 ≤ n ∧ n < 10numCifras(n) )
 */
int numCifras (const int n);
```

```
/*
 *  Devuelve el número de cifras del número natural <n>
 *  cuando se expresa en base 10
 */
int numCifras (const int n);
```

```
/*
 *  Pre: n ≥ 0
 *  Post: numCifras(n) = R  ∧  (n = 0 → R = 1)  ∧
 *        (n > 0 →  10^(R-1) ≤ n  ∧  n < 10^R)
 */
int numCifras (const int n);
```

Ejercicio 6º

```
/*
 *  Devuelve la mayor de las cifras del número natural
 *  <n> cuando se expresa en base 10
 */
int mayorCifra (const int n);
```

```
/*
 *  Devuelve la mayor de las cifras del número natural
 *  <n> cuando se expresa en base 10
 */
int mayorCifra (const int n);
```

```
/*
 *  Pre: n ≥ 0
 *  Post: mayorCifra(n) = (Máx α∈[1,∝].(n / 10^(α-1)) % 10)
 */
int mayorCifra (const int n);
```

Ejercicio 7º

```
/*
 *  Devuelve la cifra más significativa del número
 *  natural <n> cuando se expresa en base 10
 */
int cifraMasSignificativa (const int n);
```

```
/*
 *  Devuelve la cifra más significativa del número
 *  natural <n> cuando se expresa en base 10
 */
int cifraMasSignificativa (const int n);
```

```
/*
 *  Pre: n ≥ 0
 *  Post: (n = 0 → R = 1) ∧
 *        (n > 0 → 10^(R-1) ≤ n ∧ n < 10^R) ∧
 *        cifraMasSignificativa(n) = n / 10^(R-1)
 */
int cifraMasSignificativa (const int n);
```

Ejercicio 8º

```
/*
 *  Devuelve la raíz cuadrada entera por defecto del
 *  número natural <n>
 */
int raizCuadrada (const int n);
```

```
/*
 *  Devuelve la raiz cuadrada entera por defecto del
 *  número natural <n>
 */
int raizCuadrada (const int n);
```

```
/*
 *  Pre: n ≥ 0
 *  Post: raizCuadrada(n)^2 ≤ n  ∧
 *        (raizCuadrada(n) + 1)^2 > n
 */
int raizCuadrada (const int n);
```

```
/*
 *  Devuelve la raíz cuadrada entera por defecto del
 *  número natural <n>
 */
int raizCuadrada (const int n);
```

```
/*
 *  Pre: n ≥ 0
 *  Post: raizCuadrada(n) = RC ∧  RC^2 ≤ n ∧ (RC + 1)^2 > n
 */
int raizCuadrada (const int n);
```