# Programación 2

## Tema 1. Especificación formal de algoritmos (II)

## Problemas 02

Sustituir en cada uno de los ejercicios que siguen cada especificación no formal por una especificación formal, escribiendo como predicados matemáticos:

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

## *Ejercicio 1º*

```
/*
 *  Devuelve la suma de los elementos del vector
 *  valores[0,n-1] que almacena, al menos, un dato
 */
double sumaDatos (const double valores[], const int n);
```

```
/*
 *  Devuelve la suma de los elementos del vector
 *  valores[0,n-1] que almacena, al menos, un dato
 */
double sumaDatos (const double valores[], const int n);
```

```
/*
 *  Pre: n > 0  ∧  n <= #valores
 *  Post: sumaDatos(valores,n) = (Σα∈[0,n-1]. valores[α])
 */
double sumaDatos (const double valores[], const int n);
```

## *Ejercicio 2º*

```
/*
 *  Asigna al tercer parámetro la suma de los elementos
 *  del vector valores[0,n-1] que almacena, al menos,
 *  un dato
 */
void sumaDatos (const double valores[], const int n,
                double& suma);
```

```
/*
 *  Asigna al tercer parámetro la suma de los elementos
 *  del vector valores[0,n-1] que almacena, al menos,
 *  un dato
 */
void sumaDatos (const double valores[], const int n,
                double& suma);
```

```
/*
 *  Pre: n > 0  ∧  n <= #valores
 *  Post: suma = (Σα∈[0,n-1]. valores[α])
 */
void sumaDatos (const double valores[], const int n,
                double& suma);
```

## *Ejercicio 3º*

```
/*
 *  Devuelve el número de elementos positivos o nulos del
 *  vector v[0,n-1] que almacena, al menos, un dato
 */
int numPositivos (const double v[], const int n);
```

```
/*
 *  Devuelve el número de elementos positivos o nulos del
 *  vector v[0,n-1] que almacena, al menos, un dato
 */
int numPositivos (const double v[], const int n);
```

```
/*
 *  Pre: n > 0  ∧  n <= #v
 *  Post: numPositivos(v,n) = (Núm α∈[0,n-1]. v[α] ≥ 0.0)
 */
int numPositivos (const double v[], const int n);
```

## *Ejercicio 4º*

```
/*
 *  Se sabe que en el vector v[0,n-1] hay datos mayores
 *  que cero. Devuelve el valor de uno de ellos
 */
double unPositivo (const double v[], const int n);
```

```
/*
 *  Se sabe que en el vector v[0,n-1] hay datos mayores
 *  que cero. Devuelve el valor de uno de ellos
 */
double unPositivo (const double v[], const int n);
```

```
/*
 *  Pre: (∃α∈[0,n-1]. v[α] > 0.0)  ∧  n <= #v
 *  Post: unPositivo(v,n) > 0.0  ∧
 *        (∃α∈[0,n-1]. v[α] = unPositivo(v,n))
 */
double unPositivo (const double v[], const int n);
```

```
/*
 *  Se sabe que en el vector v[0,n-1] hay datos mayores
 *  que cero. Devuelve el valor de uno de ellos
 */
double unPositivo (const double v[], const int n);
```

```
/*
 *  Pre: (∃α∈[0,n-1]. v[α] > 0.0)  ∧  n <= #v
 *  Post: unPositivo(v,n) = P  ∧  P > 0.0  ∧
 *        (∃α∈[0,n-1]. v[α] = P)
 */
double unPositivo (const double v[], const int n);
```

## *Ejercicio 5º*

```
/*
 *  Se sabe que en el vector v[0,n-1] hay datos mayores
 *  que cero y datos menores que cero. Asigna al tercer
 *  parámetro uno de los datos mayores que cero y al
 *  cuarto parámetro uno de los datos menores que cero
 */
void selecciona (const double v[], const int n,
                 double& unPositivo, double& unNegativo);
```

```
/*
 *  Se sabe que en el vector v[0,n-1] hay datos mayores
 *  que cero y datos menores que cero. Asigna al tercer
 *  parámetro uno de los datos mayores que cero y al
 *  cuarto parámetro uno de los datos menores que cero
 */
void selecciona (const double v[], const int n,
                 double& unPositivo, double& unNegativo);
```

```
/*
 *  Pre: n <= #v  ∧  (∃α∈[0,n-1]. v[α] > 0.0)
 *       ∧ (∃α∈[0,n-1]. v[α] < 0.0)
 *  Post: unPositivo > 0.0  ∧  unNegativo < 0.0  ∧
 *        (∃α∈[0,n-1]. v[α] = unPositivo)  ∧
 *        (∃α∈[0,n-1]. v[α] = unNegativo)
 */
void selecciona (const double v[], const int n,
                 double& unPositivo, double& unNegativo);
```

## *Ejercicio 6º*

```
/*
 *  Se sabe que en el vector v[0,n-1] hay datos mayores
 *  que cero. Asigna al tercer parámetro el valor de
 *  uno de ellos, concretamente, el de menor índice en
 *  el vector, y asigna al cuarto parámetro el valor
 *  de dicho índice
 */
void primerPositivo (const double v[], const int n,
                     double& primero, int& posicion);
```

```
/*
 *  Se sabe que en el vector v[0,n-1] hay datos mayores que
 *  cero. Asigna al tercer parámetro el valor de uno de ellos,
 *  concretamente, el de menor índice en el vector, y asigna
 *  al cuarto parámetro el valor de dicho índice
 */
void primerPositivo (const double v[], const int n,
                     double& primero, int& posicion);
```

```
/*
 *  Pre: n <= #v  ∧  (∃α∈[0,n-1]. v[α] > 0.0)
 *  Post: primero > 0.0  ∧  v[posicion] = primero ∧
 *        posicion ≥ 0  ∧ (∀α∈[0,posicion-1]. v[α] ≤ 0.0)
 */
void primerPositivo (const double v[], const int n,
                     double& primero, int& posicion);
```

## *Ejercicio 7º*

```
/*
 *  Se sabe que en el vector T[0,n-1] hay datos mayores
 *  que cero. Devuelve el índice más próximo a 0 de un
 *  elemento del vector T[0,n-1] que almacena uno de
 *  ellos
 */
int posicionPrimerPositivo (const double T[], const int n);
```

```
/*
 *  Se sabe que en el vector T[0,n-1] hay datos mayores
 *  que cero. Devuelve el índice más próximo a 0 de un
 *  elemento del vector T[0,n-1] que almacena uno de
 *  ellos
 */
int posicionPrimerPositivo (const double T[], const int n);
```

```
/*
 *  Pre: n <= #T  ∧  (∃α∈[0,n-1].T[α] > 0.0)
 *  Post: T[posicionPrimerPositivo(T,n)] > 0.0  ∧
 *        posicionPrimerPositivo(T,n)] ≥ 0  ∧
 *        (∀α∈[0,posicionPrimerPositivo(T,n)-1]. T[α] ≤ 0.0)
 */
int posicionPrimerPositivo (const double T[], const int n);
```

```
/*
 *  Se sabe que en el vector T[0,n-1] hay datos mayores
 *  que cero. Devuelve el índice más próximo a 0 de un
 *  elemento del vector T[0,n-1] que almacena uno de
 *  ellos
 */
int posicionPrimerPositivo (const double T[], const int n);
```

```
/*
 *  Pre: n <= #T  ∧  (∃α∈[0,n-1]. T[α] > 0.0)
 *  Post: posicionPrimerPositivo(T) = PPP  ∧  PPP ≥ 0  ∧
 *        T[PPP] > 0.0  ∧  (∀α∈[0,PPP-1]. T[α] ≤ 0.0)
 */
int posicionPrimerPositivo (const double T[], const int n);
```

## *Ejercicio 8º*

```
/*
 *  Devuelve la moda o dato más repetido de los elementos
 *  del vector v[0,n-1] que almacena, al menos, un dato
 */
int moda (const int v[], const int n);
```

```
/*
 *  Devuelve la moda o dato más repetido de los elementos
 *  del vector v[0,n-1] que almacena, al menos, un dato
 */
int moda (const int v[], const int n);
```

```
/*
 *  Pre: n > 0  ∧  n <= #v
 *  Post (∃α∈[0,n-1]. v[α] = moda(v,n))  ∧
 *       (∀α∈[0,n-1]. (Núm β∈[0,n-1]. v[β] = moda(v,n)) ≥
 *                    (Núm β∈[0,n-1]. v[β] = v[α]) )
 */
int moda (const int v[], const int n);
```

```
/*
 *  Devuelve la moda o dato más repetido de los elementos
 *  del vector v[0,n-1] que almacena, al menos, un dato
 */
int moda (const int v[], const int n);
```

```
/*
 *  Pre: n > 0  ∧  n <= #v
 *  Post: moda(v,n) = M  ∧  (∃α∈[0,n-1]. v[α] = M)  ∧
 *        (∀α∈[0,n-1]. (Núm β∈[0,n-1]. v[β] = v[α]) ≤
 *                     (Núm β∈[0,n-1]. v[β] = M) )
 */
int moda (const int v[], const int n);
```

```
/*
 *  Devuelve la moda o dato más repetido de los elementos
 *  del vector v[0,n-1] que almacena, al menos, un dato
 */
int moda (const int v[], const int n);
```

```
/*
 *  Pre: n > 0  ∧  n <= #v
 *  Post: moda(v,n) = M  ∧  (∃α∈[0,n-1]. v[α] = M)  ∧
 *        VECES_MODA = (Núm α∈[0,n-1]. v[α] = M)  ∧
 *        (∀α∈[0,n-1].
 *            (Núm β∈[0,n-1]. v[β] = v[α]) ≤ VECES_MODA) )
 */
int moda (const int v[], const int n);
```