Programación 2

Lección 2. Especificación formal de algoritmos que trabajan con tablas

1. Algunos tipos de datos que utilizaremos

```
struct Fecha {
   int dia, mes, anyo;  // dia/mes/anyo de la fecha
};

struct Nif{
   int dni;             // número del DNI
   char letra;          // letra asociada al DNI
};

struct Persona {
   Nif nif;             // NIF de la persona
   Fecha nacimiento;    // su fecha de nacimiento
   bool estaCasado;     // true (casado), false (soltero)
   bool esHombre;       // true (hombre), false (mujer)
};
```

2. Especificaciones y número de componentes de una estructura de datos indexada

```
// Pre: ?????
// Post: sumaComp(a, i, d)
//                      = (SIGMA alfa EN [i,d]. a[alfa])
int sumaComp (const int a[], const int i, const int d);
```

```
int v[10];          // reserva memoria para 10 datos int
double m[3][4];     // reserva memoria para 12 datos double

  …

int s = sumaComp(v, 0, 20);  // analizar su comportamiento
```

```
// Pre: 0 <= i ∧ i <= d ∧ d < #a
// Post: sumaComp(a, i, d)
//                    = (SIGMA alfa EN [i,j]. a[alfa])
int sumaComp (const int a[], const int i, const int d);
```

```
int v[10];          // reserva memoria para 10 datos int
double m[3][4];     // reserva memoria para 12 datos double
```

Donde (solo en predicados y aserciones, no en código C++):

- `#a` denota el número de componentes del vector **a**
- `#v` denota el número de componentes del vector **v**
- `#m(1)` denota el número de filas de la matriz **m**
- `#m(2)` denota el número de columnas de cada fila de la matriz **m**
- `#m` también denota el número de filas de la matriz **m**

3. Especificación de funciones de recorrido

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #v
 * Post: cuentaPares(v,n) = (Núm α∈[0,n-1].v[α] % 2 = 0)
 */
int cuentaPares (const double v[], const int n);
```

```
/*
 * Pre: desde ≥ 0  ∧ hasta < #v
 * Post: anonima(v,desde,hasta,minimo,maximo)
 *           = (Núm α∈[desde,hasta].v[α] ≥ minimo
 *                                       ∧ v[α] ≤ maximo)
 */
int anonima (const double v[],
             const int desde, const int hasta,
             const double minimo, const double maximo);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #TNif
 * Post: anonima(TNif,n) = NIF_MAS_BAJO  ∧
 *       (∃α∈[0,n-1].TNif[α] = NIF_MAS_BAJO)  ∧
 *       (∀α∈[0,n-1].TNif[α].dni ≥ NIF_MAS_BAJO.dni)
 */
Nif anonima (const Nif TNif[], const int n);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #TPersonas
 * Post: anonima(TPersonas,n,m) =
 *       (Núm α∈[0,n-1].TPersonas[α].nacimiento.mes = m)
 */
int anonima (const Persona TPersonas[], const int n,
             const int m);
```

4. Especificación de funciones de búsqueda

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #v
 * Post: anonima(v,n) = (∀α∈[0,n-2].v[α] ≤ v[α+1])
 */
bool anonima (const int v[], const int n);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #TP
 * Post: anonima(TP,n) =
 *          (∀α∈[0,n-2].TP[α].nif.dni ≤ TP[α+1].nif.dni)
 */
bool anonima (const Persona TP[], const int n);
```

```
/*
 * Pre: desde ≥ 0 ∧ hasta < #v
 * Post: anonima(v,desde,hasta,dato) =
 *                          (∃α∈[desde,hasta].v[α] = dato)
 */
bool anonima (const int v[], const int desde,
              const int hasta, const int dato);
```

```
/*
 * Pre: desde ≥ 0  ∧ hasta < #v
 *      ∧ (∀α∈[desde,hasta-1].v[α] ≤ v[α+1])
 * Post: anonima(v,desde,hasta,dato) =
 *                    (∃α∈[desde,hasta].v[α] = dato)
 */
bool anonima (const int v[], const int desde,
              const int hasta, const int dato);
```

```
/*
 * Pre:  n ≤ #TPersonas
 *       ∧ (∃α∈[0,n-1].¬ TPersonas[α].estaCasada)
 * Post: anonima(TPersonas,n) = S  ∧ ¬ S.estaCasada ∧
 *       (∃α∈[0,n-1].TPersonas[α] = S)
 */
Persona anonima (const Persona TPersonas[], const int n);
```

```
/*
 * Pre: n ≤ #TPersonas
 *      ∧ (∃α∈[0,n-1].TPersonas[α].esHombre)
 * Post: anonima(TPersonas,n) = INDICE  ∧
 *       INDICE ≥ 0 ∧ INDICE ≤ n-1  ∧
 *       TPersonas[INDICE].esHombre  ∧
 *       (∀α∈[0,INDICE-1].¬TPersonas[α].esHombre)
 */
int anonima (const Persona TPersonas[], const int n);
```

5. Especificación de algoritmos de comparación de dos vectores o tablas

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #v1 ∧ n ≤ #v2
 * Post: anonima(v1,v2,n) = (∀α∈[0,n-1].v1[α] = v2[α])
 */
bool anonima (const int v1[], const int v2[], const int n);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #v1 ∧ n ≤ #v2
 * Post: anonima(v1,v2,n) =
 *           (∀α∈[0,n-1].(∃β∈[0,n-1]. v2[β] = v1[α]))  ∧
 *           (∀α∈[0,n-1].(∃β∈[0,n-1]. v1[β] = v2[α]))
 */
bool anonima (const int v1[], const int v2[], const int n);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #v1 ∧ n ≤ #v2
 * Post: anonima(v1,v2,n) =
 *            (∀α∈[0,n-1].
 *                 (Núm β∈[0,n-1]. v1[β] = v1[α])
 *                 = (Núm β∈[0,n-1]. v2[β] = v1[α]))
 */
bool anonima (const int v1[], const int v2[], const int n);
```

6. Especificación de algoritmos que modifican un vector o una tabla

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #v ∧ v = Vo
 * Post: (∀α∈[0,n-1].(Vo[α] < 0.0 → v[α] = 0.0)  ∧
 *                    Vo[α] ≥ 0.0 → v[α] = Vo[α]) )
 */
void anonima (double v[], const int n);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #v ∧ v = Vo
 * Post: (∀α∈[0,n-1].(Vo[α] = x → v[α] = y)
 *                    ∧ Vo[α] ≠ x → v[α] = Vo[α]))
 */
void anonima (int v[], const int n,
              const int x, const int y);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #TP ∧ TP = TPo
 * Post: (∀α∈[0,n-1].(Núm β∈[0,n-1].TP[α] = TP[β]) ∧
 *                    (Núm β∈[0,n-1].TP[α] = TPo[β]) ) ∧
 *       (∀α∈[0,n-1].
 *          (¬TP[α].estaCasada
 *               → (∀β∈[0,α].¬TP[β].estaCasada) ) ∧
 *          (TP[α].estaCasada
 *               → (∀β∈[α,n-1].TP[β].estaCasada) ) )
 */
void anonima (Persona TP[], const int n);
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #TP ∧ TP = TPo
 * Post: sonPermutacion(TP,TPo,0,n-1) ∧
 *       (∀α∈[0,n-1].
 *          (¬TP[α].estaCasada
 *               → (∀β∈[0,α].¬TP[β].estaCasada) ) ∧
 *          (TP[α].estaCasada
 *               → (∀β∈[α,n-1].TP[β].estaCasada) ) )
 */
void anonima (Persona TP[], const int n);
```

Donde:

```
sonPermutacion(Ta,Tb,desde,hasta) ≡
    (∀α∈[desde,hasta].
          (Núm β∈[desde,hasta].Ta[α] = Ta[β]) ∧
          (Núm β∈[desde,hasta].Ta[α] = Tb[β]) )
```

```
/*
 * Pre: n ≥ 0 ∧ n ≤ #TP ∧ v = Vo
 * Post: sonPermutacion(v,Vo,0,n-1) ∧ ordenado(v,0,n-1)
 */
void anonima (int v[], const int n);
```

Donde:

sonPermutacion(Ta,Tb,desde,hasta) ≡
(∀α∈[desde,hasta].
(Núm β∈[desde,hasta].Ta[α] = Ta[β]) ∧
(Núm β∈[desde,hasta].Ta[α] = Tb[β]))

ordenado(T,desde,hasta) ≡
(∀α∈[desde,hasta-1].T[α] ≤ T[α+1])