Programación 2

Diseño recursivo de algoritmos por inmersión

Problemas 05

Problema 1. Diseñar sin bucles:

```
/*
 * Pre: v = Vo AND n >= 1 AND n <= #v
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1]. v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n);
```

Se propone intentar dos diseños diferentes:

1. Aplicando una inmersión mediante debilitamiento de la postcondición de **`desplazar`**`(v,n)`
2. Aplicando una inmersión mediante refuerzo de la precondición de **`desplazar`**`(v,n)`

Formas equivalentes de escribir su especificación:

```
/*
 * Pre: v = Vo AND n >= 1 AND n <= #v
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1]. v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n);
```

```
/*
 * Pre: n >= 1 AND n <= #v AND
 *      (PT alfa EN [0,n-1]. v[alfa] = Vo[alfa])
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1]. v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n);
```

Se va a plantear un primer diseño sin bucles de la función **`desplazar`**`(v,n)` programando una función auxiliar **`moverDcha`**`(v,hasta)` mediante **debilitamiento de la postcondición** de la primera función.

```
/*
 * Pre: n >= 1 AND n <= #v AND v = Vo
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1]. v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n);
```

```
/*
 * Pre: hasta >= 1 AND hasta < #v AND v = V1
 * Post: (PT alfa EN [1,hasta]. v[alfa] = V1[alfa-1])
 */
template <typename T>
void moverDcha (T v[], const int hasta);
```

Otra forma de escribir ambas especificaciones:

```
/*
 * Pre: n >= 1 AND n <= #v AND
 *      (PT alfa EN [0,n-1]. v[alfa] = Vo[alfa])
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1]. v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n);
```

```
/*
 * Pre: hasta >= 1 AND hasta < #v AND
 *      (PT alfa EN [0,hasta]. v[alfa] = V1[alfa])
 * Post: (PT alfa EN [1,hasta]. v[alfa] = V1[alfa-1])
 */
template <typename T>
void moverDcha (T v[], const int hasta);
```

```
/*
 * Pre: n >= 1 AND n <= #v AND
 *      (PT alfa EN [0,n-1]. v[alfa] = Vo[alfa])
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1]. v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n) {
   if (n > 1) {
      T ultimo = v[n-1];
      // (PT alfa EN [0,n-1]. v[alfa] = Vo[alfa]) AND
      // ultimo = Vo[n-1] AND n - 1 >= 1 AND n - 1 < #v
      moverDcha(v, n-1);
      // (PT alfa EN [1,n-1]. v[alfa] = Vo[alfa-1]) AND
      // v[0] = Vo[0] AND ultimo = Vo[n-1]
      v[0] = ultimo;
   }
}
```

```
/*
 * Pre: hasta >= 1 AND hasta < #v AND
 *      (PT alfa EN [0,hasta]. v[alfa] = V1[alfa])
 * Post: (PT alfa EN [1,hasta]. v[alfa] = V1[alfa-1])
 */
template <typename T>
void moverDcha (T v[], const int hasta) {
   v[hasta] = v[hasta-1];
   if (hasta >= 2) {
      // (PT alfa EN [0,hasta]. v[alfa] = V1[alfa])
      // AND v[hasta] = V1[hasta-1] AND 1 <= hasta - 1
      // AND hasta - 1 < #v
      moverDcha(v, hasta-1);
      // (PT alfa EN [1,hasta-1]. v[alfa] = V1[alfa-1]) AND
      // v[hasta] = V1[hasta-1]
   }
}
```

Se va a plantear un segundo diseño sin bucles de la función **`desplazar`**`(v,n)` programando una función auxiliar **`moverDcha`**`(v,desde,n)` mediante **refuerzo de la precondición** de la primera función.

```
/*
 * Pre: n >= 1 AND n <= #v AND
 *      (PT alfa EN [0,n-1]. v[alfa] = Vo[alfa])
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1].v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n);
```

```
/*
 * Pre: desde >= 1 AND desde <= n AND n <= #v AND
 *      (PT alfa EN [0,desde-1].v[alfa] = Vo[alfa]) AND
 *      (PT alfa EN [desde,n-1].v[alfa] = Vo[alfa-1])
 * Post: (PT alfa EN [1,n-1].v[alfa] = Vo[alfa-1])
 */
template <typename T>
void moverDcha (T v[], const int desde, const int n);
```

```
/*
 * Pre: n >= 1 AND n <= #v AND
 *      (PT alfa EN [0,n-1]. v[alfa] = Vo[alfa])
 * Post: v[0] = Vo[n-1] AND
 *       (PT alfa EN [1,n-1].v[alfa] = Vo[alfa-1])
 */
template <typename T>
void desplazar (T v[], const int n) {
    if (n > 1) {
        T ultimo = v[n-1];
        // (PT alfa EN [1,n-1].v[alfa] = Vo[alfa]) AND
        // (PT alfa EN [n,n-1].v[alfa] = Vo[alfa-1]) AND
        // n >= 1 AND n <= n AND n <= #v AND ultimo = Vo[n-1]
        moverDcha(v, n, n);
        // (PT alfa EN [1,n-1].v[alfa] = Vo[alfa-1]) AND
        // ultimo = Vo[n-1];
        v[0] = ultimo;
   }
}
```

```
/*
 * Pre: desde >= 1 AND desde <= n AND n <= #v AND
 *      (PT alfa EN [0,desde-1].v[alfa] = Vo[alfa]) AND
 *      (PT alfa EN [desde,n-1].v[alfa] = Vo[alfa-1])
 * Post: (PT alfa EN [1,n-1].v[alfa] = Vo[alfa-1])
 */
template <typename T>
void moverDcha (T v[], const int desde,  const int n) {
    if (desde > 1) {
        v[desde-1] = v[desde-2];
        // desde - 1 >= 1 AND desde - 1 <= n AND n <= #v AND
        // (PT alfa EN [1,desde-2].v[alfa] = Vo[alfa]) AND
        // (PT alfa EN [desde-1,n-1].v[alfa] = Vo[alfa-1])
        moverDcha(v, desde - 1, n);
        // (PT alfa EN [1,n-1].v[alfa] = Vo[alfa-1])
    }
}
```

En el siguiente problema se va a trabajar con ficheros binarios de datos de tipo T :

```
<fichero_binario_T> ::= { <dato> }
<dato> ::= T
```

Asumiremos definidas las siguientes operaciones binarias de relación entre un par de datos, a y b, de tipo T :

```
a == b
a != b
a < b
a <= b
a > b
a >= b
```

Problema 2. Diseñar sin bucles:

```
/*
 * Pre: <nombre1> y <nombre2> son los nombres de dos ficheros
 *      binarios que almacenan secuencias de datos de tipo T
 * Post: Devuelve <cierto> si y sólo si ambos ficheros almacenan
 *       el mismo número de datos
 */
template <typename T>
bool igualLongitud (const char nombre1[], const char nombre2[]);
```

Se propone intentar dos diseños diferentes:

1. Aplicando una inmersión mediante debilitamiento de la postcondición de **`igualLongitud`**`(nombre1,nombre2)`
2. Aplicando una inmersión mediante refuerzo de la precondición de **`igualLongitud`**`(nombre1,nombre2)`

Se va a plantear un primer diseño sin bucles de la función **`igualLongitud`**`(nombre1,nombre2)` programando una función auxiliar **`igualLongitud`**`(f1,f2)` mediante **debilitamiento de la postcondición** de la primera función.

```
/*
 * Pre: <nombre1> y <nombre2> son los nombres de dos ficheros
 *      binarios que almacenan secuencias de datos de tipo T
 * Post: Devuelve <cierto> si y sólo si ambos ficheros almacenan
 *       el mismo número de datos
 */
template <typename T>
bool igualLongitud (const char nombre1[], const char nombre2[]);
```

```
/*
 * Pre: <f1> y <f2> están asociados a dos ficheros binarios que
 *      almacenan secuencias de datos de tipo T
 * Post: Devuelve <cierto> si y sólo si los ficheros asociados a <f1>
 *       y <f2> almacenan, en el momento de ser invocada esta función,
 *       un mismo número de datos no leídos
 */
template <typename T>
bool igualLongitud (ifstream& f1, ifstream& f2);
```

```
/*
 * Pre: <nombre1> y <nombre2> son los nombres de dos ficheros
 *      binarios que almacenan secuencias de datos de tipo T
 * Post: Devuelve <cierto> si y sólo si ambos ficheros almacenan
 *       el mismo número de datos
 */
template <typename T>
bool igualLongitud (const char nombre1[], const char nombre2[]) {
   // Asocia los ficheros <nombre1> y <nombre2> a los flujos <f1> y <f2>
   ifstream f1, f2;
   f1.open(nombre1, ios::binary);
   f2.open(nombre2, ios::binary);
   //  Determina si ambos ficheros almacenan un número igual de datos
   bool igual = igualLongitud <T> (f1, f2);
   // Libera los ficheros asociados a los flujos <f1> y <f2>
   f1.close();
   f2.close();
   // Devuelve el resultado de la comparación
   return igual;
}
```

```
/*
 * Pre: <f1> y <f2> están asociados a dos ficheros binarios que
 *      almacenan secuencias de datos de tipo T
 * Post: Devuelve <cierto> si y sólo si los ficheros asociados a <f1>
 *       y <f2> almacenan, en el momento de ser invocada esta función,
 *       un mismo número de datos no leídos
 */
template <typename T>
bool igualLongitud (ifstream& f1, ifstream& f2) {
   T nuevo;
   f1.read(reinterpret_cast<char *>(&nuevo), sizeof(T));
   f2.read(reinterpret_cast<char *>(&nuevo), sizeof(T));
   if (!f1.eof() && !f2.eof()) {
        return igualLongitud <T> (f1, f2);    // recurrencia
   }
   else {
        return f1.eof() && f2.eof();          // base
   }
}
```

Se va a plantear un primer diseño sin bucles de la función **`igualLongitud`**`(nombre1,nombre2)` programando una función auxiliar **`igualLongitud`**`(f1,f2)` mediante **refuerzo de la precondición** de la primera función.

```
/*
 * Pre: <nombre1> y <nombre2> son los nombres de dos ficheros
 *      binarios que almacenan secuencias de datos de tipo T
 * Post: Devuelve <cierto> si y sólo si ambos ficheros almacenan
 *       el mismo número de datos
 */
template <typename T>
bool igualLongitud (const char nombre1[], const char nombre2[]);
```

```
/*
 * Pre: <f1> y <f2> están asociados a dos ficheros binarios que
 *      almacenan secuencias de datos de tipo T y ya han sido
 *      leídos de cada uno de ellos un número igual de datos
 * Post: Devuelve cierto si y sólo si los ficheros asociados a <f1>
 *       y <f2> almacenan el mismo número de datos
 */
template <typename T>
bool igualLongitud (ifstream& f1, ifstream& f2);
```

```
/*
 * Pre: <nombre1> y <nombre2> son los nombres de dos ficheros
 *      binarios que almacenan secuencias de datos de tipo T
 * Post: Devuelve <cierto> si y sólo si ambos ficheros almacenan
 *       el mismo número de datos
 */
template <typename T>
bool igualLongitud (const char nombre1[], const char nombre2[]) {
   // Asocia los ficheros <nombre1> y <nombre2> a los flujos <f1> y <f2>
   ifstream f1, f2;
   f1.open(nombre1, ios::binary); f2.open(nombre2, ios::binary);
   // De los ficheros asociados a f1 y f2 no se ha leído ningún dato.
   // Determina si ambos ficheros almacenan un número igual de datos
   bool igual = igualLongitud <T> (f1, f2);
   // Libera los ficheros asociados a los flujos <f1> y <f2>
   f1.close(); f2.close();
   // Devuelve el resultado de la comparación
   return igual;
}
```

```
/*
 * Pre: <f1> y <f2> están asociados a dos ficheros binarios que
 *      almacenan secuencias de datos de tipo T y ya han sido
 *      leídos de cada uno de ellos un número igual de datos
 * Post: Devuelve <cierto> si y sólo si los ficheros asociados a <f1>
 *       y <f2> almacenan el mismo número de datos
 */
template <typename T>
bool igualLongitud (ifstream& f1, ifstream& f2) {
   T nuevo;
   f1.read(reinterpret_cast<char *>(&nuevo), sizeof(T));
   f2.read(reinterpret_cast<char *>(&nuevo), sizeof(T));
   if (!f1.eof() && !f2.eof()) {
        return igualLongitud <T> (f1, f2);    // recurrencia
   }
   else {
        return f1.eof() && f2.eof();          // base
   }
}
```