# Programación 2

## Lección 6. Diseño de algoritmos recursivos que trabajan con ficheros

1. **Necesidad de un diseño recursivo por inmersión**
   - ***Algoritmos de recorrido de un texto***
2. **Diseño recursivo de algoritmos que trabajan con ficheros de texto**
   - ***Algoritmos de comparación de textos***
   - ***Algoritmos de presentación de un texto***
3. **Diseño recursivo de algoritmos que trabajan con ficheros binarios**
   - ***Algoritmos de recorrido de un fichero binario***
   - ***Algoritmos de creación de un fichero binario***
   - ***Algoritmos de búsqueda en un fichero binario***

# 1. Necesidad de un diseño recursivo por inmersión

En esta asignatura no vamos a especificar formalmente los algoritmos que trabajan con ficheros. No obstante, sus especificaciones van a ser rigurosas.

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el
 *        fichero <nombre>
 */
int contarLineas (const char nombre[]);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]);
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *  Post: Devuelve el número total de líneas que estaban pendientes de
 *        ser leídas del fichero asociado a <f> en el momento en que
 *        se invocó esta función
 */
int contarLineas1 (ifstream& f);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   int numLineas = contarLineas1(f);
   f.close();
   return numLineas;
}
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *  Post: Devuelve el número total de líneas que estaban pendientes de
 *        ser leídas del fichero asociado a <f> en el momento en que
 *        se invocó esta función
 */
int contarLineas1 (ifstream& f);
```

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un
 *       texto
 *  Post: Devuelve el número total de líneas que estaban
 *        pendientes de ser leídas del fichero asociado a <f>
 *        en el momento en que se invocó esta función
 */
int contarLineas1 (ifstream& f) {
   const int MAX = 128;
   char linea[MAX];
   f.getline(linea, MAX);      // intenta leer una nueva línea
   if (!f.eof()) {
       // Ha leído una nueva línea del fichero asociado a <f>
       return 1 + contarLineas1(f);
   }
   else {
       // No quedan líneas por leer del fichero asociado a <f>
       return 0;
   }
}
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   int numLineas = contarLineas1(f);
   f.close();
   return numLineas;
}

int contarLineas1 (ifstream& f) {
   const int MAX = 128;
   char linea[MAX];
   f.getline(linea, MAX);
   if (!f.eof()) {
      return 1 + contarLineas1(f);
   }
   else {  return 0;  }
}
```

**Ejemplo de un mal diseño a evitar a toda costa: aquí hay código, pero NO HAY diseño**

**¿Por qué el diseño de su código ha de depender del código de otra función? El diseño de cada función ha de ser independiente**

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   int numLineas = contarLineas1(f);
   f.close();
   return numLineas;
}

int contarLineas1 (ifstream& f);
```

**¿Puedes justificar esta invocación?**
**¿Qué devuelve?**
**¿Por qué?**

```
int contarLineas1 (ifstream& f) {

    . . .

}
```

**Sin contar con una especificación, ¿qué código escribirías? ¿cómo lo justificarías?**

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]);
```

**Diseño por inmersión reforzando la precondición**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *       y el valor de <leidas> es igual al número de líneas de ese
 *       fichero que ya han sido leídas
 *  Post: Devuelve el número total de líneas del fichero asociado a <f>
 */
int contarLineas2 (ifstream& f, const int leidas);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   int numLineas = contarLineas2(f, 0);
   f.close();
   return numLineas;
}
```

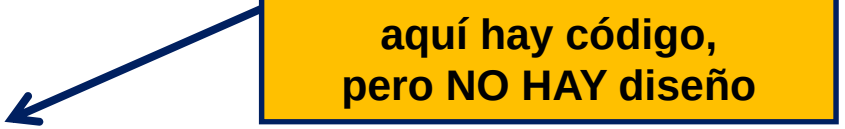
**Diseño por inmersión reforzando la precondición**

```
/*
 *  Pre: El flujo <f]> está asociado a un fichero que almacena un texto
 *       y el valor de <leidas> es igual al número de líneas de ese
 *       fichero que ya han sido leídas
 *  Post: Devuelve el número total de líneas del fichero asociado a <f>
 */
int contarLineas2 (ifstream& f, const int leidas);
```

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *       y el valor de <leidas> es igual al número de líneas de ese
 *       fichero que ya han sido leídas
 *  Post: Devuelve el número total de líneas del fichero asociado a <f>
 */
int contarLineas2 (ifstream& f, const int leidas) {
   const int MAX = 128;
   char linea[MAX];
   f.getline(linea, MAX);      // intenta leer una nueva línea
   if (!f.eof()) {
       // Ha leído una nueva línea del fichero asociado a <f>
       return contarLineas2(f, leidas + 1);
   }
   else {
       // No quedan líneas por leer del fichero asociado a <f>
       return leidas;
   }
}
```

```
/*
 *  Pre: <nombre> es un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   int numLineas = contarLineas2(f, 0);
   f.close();
   return numLineas;
}

int contarLineas2 (ifstream& f, const int leidas) {
   const int MAX = 128;
   char linea[MAX];
   f.getline(linea, MAX);
   if (!f.eof()) {  return contarLineas2(f, leidas + 1);  }
   else {  return leidas;  }
}
```



```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Devuelve el número de líneas almacenadas en el fichero
 *        <nombre>
 */
int contarLineas (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   int numLineas = contarLineas2(f, 0);
   f.close();
   return numLineas;
}

int contarLineas2 (ifstream& f, const int leidas);
```

**¿Puedes justificar esta invocación? ¿Por qué el segundo parámetro ha de ser 0? ¿Qué valor devuelve?**

```
int contarLineas2 (ifstream& f, const int leidas) {

   . . .

}
```

**Sin contar con una especificación, ¿qué código escribirías? ¿cómo lo justificarías?**

# 2. Diseño recursivo de algoritmos que trabajan con ficheros de texto

## 2.1. Comparación de dos ficheros de texto

```
/*
 *  Pre: <nombre1> y <nombre2> son los nombres de dos ficheros
 *       que almacenan textos
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados
 *        en los ficheros <nombre1> y <nombre2> son idénticos
 */
bool sonIguales (const char nombre1[], const char nombre2[]);
```

```
/*
 *  Pre: <nombre1> y <nombre2> son los nombres de dos ficheros que
 *       almacenan textos
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros <nombre1> y <nombre2> son idénticos
 */
bool sonIguales (const char nombre1[], const char nombre2[]);
```

**¿ Qué tipo de diseño se ha planteado?**

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto
 *  Post: Devuelve <cierto> si y sólo si eran idénticos los textos
 *        pendientes de ser leídos de los ficheros asociados a <f1>
 *        y <f2> en el momento en que se invocó a esta función
 */
bool sonIguales1 (ifstream& f1, ifstream& f2);
```

```
/*
 *  Pre: <nombre1> y <nombre2> son los nombres de dos ficheros que
 *       almacenan textos
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros <nombre1> y <nombre2> son idénticos
 */
bool sonIguales (const char nombre1[], const char nombre2[]);
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto
 *  Post: Devuelve <cierto> si y sólo si eran idénticos los textos
 *        pendientes de ser leídos de los ficheros asociados a <f1>
 *        y <f2> en el momento en que se invocó a esta función
 */
bool sonIguales1 (ifstream& f1, ifstream& f2);
```

```
/*
 *  Pre: <nombre1> y <nombre2> son los nombres de dos ficheros que
 *       almacenan textos
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros <nombre1> y <nombre2> son idénticos
 */
bool sonIguales (const char nombre1[], const char nombre2[]) {
   ifstream f1, f2;
   f1.open(nombre1);  f2.open(nombre2);
   bool iguales = sonIguales1(f1, f2);
   f1.close();  f2.close();
   return iguales;
}
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto
 *  Post: Devuelve <cierto> si y sólo si eran idénticos los textos
 *        pendientes de ser leídos de los ficheros asociados a <f1>
 *        y <f2> en el momento en que se invocó a esta función
 */
bool sonIguales1 (ifstream& f1, ifstream& f2);
```

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto
 *  Post: Devuelve <cierto> si y sólo si eran idénticos los textos
 *        pendientes de ser leídos de los ficheros asociados a <f1>
 *        y <f2> en el momento en que se invocó a esta función
 */
bool sonIguales1 (ifstream& f1, ifstream& f2) {
   const int MAX = 128;
   char linea1[MAX], linea2[MAX];
   // Intenta leer una nueva línea de cada uno de los ficheros
   f1.getline(linea1, MAX); f2.getline(linea2, MAX);
   if (!f1.eof() && !f2.eof()) {
       // Ha leído una nueva línea de cada uno de los dos ficheros
       if (strcmp(linea1,linea2) == 0) { // Las dos líneas son idénticas
           return sonIguales1(f1, f2);
       }
       else { return false; }
   }
   else {
       return f1.eof() && f2.eof();
   }
}
```

```
/*
 *  Pre: <nombre1> y <nombre2> son los nombres de dos ficheros que
 *       almacenan textos
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros <nombre1> y <nombre2> son idénticos
 */
bool sonIguales (const char nombre1[], const char nombre2[]);
```

**¿ Qué tipo de diseño se ha planteado?**

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto.
 *       Las líneas leídas de ambos son idénticas
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros asociados a <f1> y <f2> son idénticos
 */
bool sonIguales2 (ifstream& f1, ifstream& f2);
```

```
/*
 *  Pre: <nombre1> y <nombre2> son los nombres de dos ficheros que
 *       almacenan textos
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros <nombre1> y <nombre2> son idénticos
 */
bool sonIguales (const char nombre1[], const char nombre2[]);
```

**Diseño por inmersión reforzando la precondición**

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto.
 *       Las líneas leídas de ambos son idénticas
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros asociados a <f1> y <f2> son idénticos
 */
bool sonIguales2 (ifstream& f1, ifstream& f2);
```

```
/*
 *  Pre: <nombre1> y <nombre2> son los nombres de dos ficheros que
 *       almacenan textos
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros <nombre1> y <nombre2> son idénticos
 */
bool sonIguales (const char nombre1[], const char nombre2[]) {
   ifstream f1, f2;
   f1.open(nombre1);  f2.open(nombre2);
   bool iguales = sonIguales2(f1, f2);
   f1.close();  f2.close();
   return iguales;
}
```

**Diseño por inmersión reforzando la precondición**

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto.
 *       Las líneas leídas de ambos son idénticas
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros asociados a <f1> y <f2> son idénticos
 */
bool sonIguales2 (ifstream& f1, ifstream& f2);
```

```
/*
 *  Pre: Los flujos <f1> y <f2> están asociados a dos ficheros de texto.
 *       Las líneas leídas de ambos son idénticas
 *  Post: Devuelve <cierto> si y sólo si los textos almacenados en los
 *        ficheros asociados a <f1> y <f2> son idénticos
 */
bool sonIguales2 (ifstream& f1, ifstream& f2) {
   const int MAX = 128;
   char linea1[MAX], linea2[MAX];
   // Intenta leer una nueva línea de cada uno de los ficheros
   f1.getline(linea1, MAX); f2.getline(linea2, MAX);
   if (!f1.eof() && !f2.eof()) {
       // Ha leído una nueva línea de cada uno de los dos ficheros
       if (strcmp(linea1,linea2) == 0) { // Las dos líneas son idénticas
           return sonIguales2(f1, f2);
       }
       else { return false; }
   }
   else {
       return f1.eof() && f2.eof();
   }
}
```

Conviene observar que los dos diseños anteriores son diferentes, aunque los códigos de las dos funciones que integran cada diseño coincidan.

# 2.2. Presentación por pantalla de un fichero de texto

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Ha presentado por pantalla el texto almacenado en el
 *        fichero <nombre>
 */
void mostrar (const char nombre[]);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        <nombre>
 */
void mostrar (const char nombre[]);
```

**¿Qué tipo de diseño se ha planteado?**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *  Post: Ha mostrado por pantalla las líneas que estaban pendientes de
 *        ser leídas del fichero asociado a <f> en el momento en que se
 *        invocó esta función
 */
void mostrar1 (ifstream& f);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        <nombre>
 */
void mostrar (const char nombre[]);
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *  Post: Ha mostrado por pantalla las líneas que estaban pendientes de
 *        ser leídas del fichero asociado a <f> en el momento en que se
 *        invocó esta función
 */
void mostrar1 (ifstream& f);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        <nombre>
 */
void mostrar (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   mostrar1(f);
   f.close();
}
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *  Post: Ha mostrado por pantalla las líneas que estaban pendientes de
 *        ser leídas del fichero asociado a <f> en el momento en que se
 *        invocó esta función
 */
void mostrar1 (ifstream& f);
```

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto
 *  Post: Ha mostrado por pantalla las líneas que estaban pendientes de
 *        ser leídas del fichero asociado a <f> en el momento en que se
 *        invocó esta función
 */
void mostrar1 (ifstream& f) {
   const int MAX = 128;
   char linea[MAX];
   // Intenta leer una nueva línea del fichero asociado a <f>
   f.getline(linea, MAX);
   if (!f.eof()) {
       // Presenta la nueva línea del fichero asociado a <f>
       cout << linea << endl;
       // Presenta las siguientes líneas del fichero asociado a <f>
       mostrar1(f);
   }
}
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        <nombre>
 */
void mostrar (const char nombre[]);
```

**¿Qué tipo de diseño se ha planteado?**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto.
 *       Las líneas ya leídas de dicho texto han sido presentadas por
 *       pantalla
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        asociado a <f>
 */
void mostrar2 (ifstream& f);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        <nombre>
 */
void mostrar (const char nombre[]);
```

**Diseño por inmersión reforzando la precondición**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto.
 *       Las líneas ya leídas de dicho texto han sido presentadas por
 *       pantalla
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        asociado a <f>
 */
void mostrar2 (ifstream& f);
```

```
/*
 *  Pre: <nombre> es el nombre de un fichero que almacena un texto
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        <nombre>
 */
void mostrar (const char nombre[]) {
   ifstream f;
   f.open(nombre);
   mostrar2(f);
   f.close();
}
```

**Diseño por inmersión reforzando la precondición**

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto.
 *       Las líneas ya leídas de dicho texto han sido presentadas por
 *       pantalla
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        asociado a <f>
 */
void mostrar2 (ifstream& f);
```

```
/*
 *  Pre: El flujo <f> está asociado a un fichero que almacena un texto.
 *       Las líneas ya leídas de dicho texto han sido presentadas por
 *       pantalla
 *  Post: Ha presentado por pantalla el texto almacenado en el fichero
 *        asociado a <f>
 */
void mostrar2 (ifstream& f) {
   const int MAX = 128;
   char linea[MAX];
   // Intenta leer una nueva línea del fichero asociado a <f>
   f.getline(linea, MAX);
   if (!f.eof()) {
       // Presenta la nueva línea del fichero asociado a <f>
       cout << linea << endl;
       // Presenta las siguientes líneas del fichero asociado a <f>
       mostrar2(f);
   }
}
```

# 3. Diseño recursivo de algoritmos que trabajan con ficheros binarios

## 3.1. Recorrido de un fichero binario

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve el número de datos almacenados en el fichero <nombre>
 *       que estén comprendidos en el intervalo [min,max]
 */
int contar (const char nombre[], const int min, const int max);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve el número de datos almacenados en el fichero <nombre>
 *       que estén comprendidos en el intervalo [min,max]
 */
int contar (const char nombre[], const int min, const int max);
```

**¿Qué tipo de diseño se ha planteado?**

```
/*
 *  Pre: <f> es un flujo asociado a un fichero binario que almacena una
 *       secuencia de datos de tipo <int>
 *  Post: Devuelve el número de datos aún no leídos del fichero asociado
 *        a <f> en el momento en que se invocó esta función cuyos valores
 *        estén comprendidos en el intervalo [min,max]
 */
int contar1 (ifstream& f, const int min, const int max);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve el número de datos almacenados en el fichero <nombre>
 *       que estén comprendidos en el intervalo [min,max]
 */
int contar (const char nombre[], const int min, const int max);
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: <f> es un flujo asociado a un fichero binario que almacena una
 *       secuencia de datos de tipo <int>
 *  Post: Devuelve el número de datos aún no leídos del fichero asociado
 *        a <f> en el momento en que se invocó esta función cuyos valores
 *        estén comprendidos en el intervalo [min,max]
 */
int contar1 (ifstream& f, const int min, const int max);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve el número de datos almacenados en el fichero <nombre>
 *       que estén comprendidos en el intervalo [min,max]
 */
int contar (const char nombre[], const int min, const int max) {
   ifstream f;
   f.open(nombre, ios::binary);
   int cuenta = contar1(f, min, max);
   f.close();
   return cuenta;
}
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 *  Pre: <f> es un flujo asociado a un fichero binario que almacena una
 *       secuencia de datos de tipo <int>
 *  Post: Devuelve el número de datos aún no leídos del fichero asociado
 *        a <f> en el momento en que se invocó esta función cuyos valores
 *        estén comprendidos en el intervalo [min,max]
 */
int contar1 (ifstream& f, const int min, const int max);
```

```
/*
 *  Pre: <f> es un flujo asociado a un fichero binario que almacena una
 *       secuencia de datos de tipo <int>
 *  Post: Devuelve el número de datos aún no leídos del fichero asociado
 *        a <f> en el momento en que se invocó esta función cuyos valores
 *        estén comprendidos en el intervalo [min,max]
 */
int contar1 (ifstream& f, const int min, const int max) {
   int nuevoDato;
   f.read(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
   if (!f.eof()) {
      if (nuevoDato >= min && nuevoDato <= max) {
          return 1 + contar1(f, min, max);
      }
      else {
          return contar1(f, min, max);
      }
   }
   else {
      return 0;
   }
}
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve el número de datos almacenados en el fichero <nombre>
 *       que estén comprendidos en el intervalo [min,max]
 */
int contar (const char nombre[], const int min, const int max) {
   ifstream f;
   f.open(nombre, ios::binary);
   int cuenta = contar2(f, min, max, 0);
   f.close();  return cuenta;
}
```

**Diseño por inmersión reforzando la precondición**

```
/* Pre: <f> es un flujo asociado a un fichero binario que almacena una
 *      secuencia de datos de tipo <int> y el valor de <contados> es
 *      igual al número de datos leídos del fichero asociado a <f>
 *      comprendidos en el intervalo [min,max]
 *  Post: Devuelve el número de datos almacenados en el fichero asociado
 *        a <f> que estén comprendidos en el intervalo [min,max]   */
int contar2 (ifstream& f, const int min, const int max,
             const int contados);
```

```
/*
 * Pre: <f> es un flujo asociado a un fichero binario que almacena una
 *      secuencia de datos de tipo <int> y el valor de <contados> es
 *      igual al número de datos leídos del fichero asociado a <f>
 *      comprendidos en el intervalo [min,max]
 *  Post: Devuelve el número de datos almacenados en el fichero asociado
 *        a <f> que estén comprendidos en el intervalo [min,max]
 */
int contar2 (ifstream& f, const int min, const int max,
             const int contados) {
   int nuevoDato;
   f.read(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
   if (!f.eof()) {
      if (nuevoDato >= min && nuevoDato <= max) {
          return contar2(f ,min, max, contados + 1);
      }
      else {
          return contar2(f, min, max, contados);
      }
   }
   else { return contados; }
}
```

# 3.2. Creación de un fichero binario

```
/*
 * Pre: <nombre1> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Crea un fichero binario de nombre <nombre2> y almacena en él
 *       aquellos datos del fichero <nombre1> cuyo valor está comprendido
 *       en el intervalo [min,max]
 */
void seleccionar (const char nombre1[], const char nombre2[],
                  const int min, const int max);
```

```
/*
 * Pre: <nombre1> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Crea un fichero binario de nombre <nombre2> y almacena en él
 *       aquellos datos del fichero <nombre1> cuyo valor está comprendido
 *       en el intervalo [min,max]
 */
void seleccionar (const char nombre1[], const char nombre2[],
                  const int min, const int max);
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 * Pre: <f1> es un flujo de entrada asociado a un fichero binario que
 *      almacena una secuencia de datos de tipo <int> y <f2> es un flujo
 *      de salida asociado a un fichero binario que almacena datos de
 *      tipo <int>
 * Post: Ha añadido al fichero asociado a <f2> la secuencia de datos no
 *       leídos en el momento de invocar esta función del fichero asociado
 *       a <f1> cuyo valor está comprendido en el intervalo [min,max]
 */
void seleccionar1 (ifstream& f1, ofstream& f2,
                   const int min, const int max);
```

```
/*
 * Pre: <nombre1> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Crea un fichero binario de nombre <nombre2> y almacena en él
 *       aquellos datos del fichero <nombre1> cuyo valor está comprendido
 *       en el intervalo [min,max]
 */
void seleccionar (const char nombre1[], const char nombre2[],
                  const int min, const int max) {
   ifstream f1;
   f1.open(nombre1, ios::binary);
   ofstream f2;
   f2.open(nombre2, ios::binary);
   seleccionar1(f1, f2, min, max);
   f1.close();
   f2.close();
}
```

```
/*
 * Pre: <f1> es un flujo de entrada asociado a un fichero binario que
 *      almacena una secuencia de datos de tipo <int> y <f2> es un flujo
 *      de salida asociado a un fichero binario que almacena datos de
 *      tipo <int>
 * Post: Ha añadido al fichero asociado a <f2> la secuencia de datos no
 *       leídos en el momento de invocar esta función del fichero asociado
 *       a <f1> cuyo valor está comprendido en el intervalo [min,max]
 */
void seleccionar1 (ifstream& f1, ofstream& f2,
                   const int min, const int max) {
   int nuevoDato;
   f1.read(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
   if (!f1.eof()) {
      if (nuevoDato >= min && nuevoDato <= max) {
         f2.write(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
      }
      seleccionar1(f1, f2, min, max);
   }
}
```

```
/*
 * Pre: <nombre1> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Crea un fichero binario de nombre <nombre2> y almacena en él
 *       aquellos datos del fichero <nombre1> cuyo valor está comprendido
 *       en el intervalo [min,max]
 */
void seleccionar (const char nombre1[], const char nombre2[],
                  const int min, const int max);
```

**¿Tipo de diseño planteado?**

```
/* Pre: <f1> es flujo de entrada asociado a un fichero binario que almacena
 *      una secuencia de datos de tipo <int> y <f2> es flujo de salida
 *      asociado a un fichero binario que almacena datos de tipo <int>
 *      en el que se han escrito la secuencia de datos leídos del fichero
 *      asociado a <f1> cuyos valores está comprendido en [min,max]
 * Post: El fichero asociado a <f2> almacena los datos del fichero
 *       asociado al flujo <f1> cuyos valores está comprendido en el
 *       intervalo [min,max]                                          */
void seleccionar2 (ifstream& f1, ofstream& f2,
                   const int min, const int max);
```

```
/*
 * Pre: <nombre1> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Crea un fichero binario de nombre <nombre2> y almacena en él
 *       aquellos datos del fichero <nombre1> cuyo valor está comprendido
 *       en el intervalo [min,max]
 */
void seleccionar (const char nombre1[], const char nombre2[],
                  const int min, const int max);
```

**Diseño por inmersión reforzando la precondición**

```
/* Pre: <f1> es flujo de entrada asociado a un fichero binario que almacena
 *      una secuencia de datos de tipo <int> y <f2> es flujo de salida
 *      asociado a un fichero binario que almacena datos de tipo <int>
 *      en el que se han escrito la secuencia de datos leídos del fichero
 *      asociado a <f1> cuyos valores está comprendido en [min,max]
 * Post: El fichero asociado a <f2> almacena los datos del fichero
 *       asociado al flujo <f1> cuyos valores está comprendido en el
 *       intervalo [min,max]                                          */
void seleccionar2 (ifstream& f1, ofstream& f2,
                   const int min, const int max);
```

```
/*
 * Pre: <nombre1> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Crea un fichero binario de nombre <nombre2> y almacena en él
 *       aquellos datos del fichero <nombre1> cuyo valor está comprendido
 *       en el intervalo [min,max]
 */
void seleccionar (const char nombre1[], const char nombre2[],
                  const int min, const int max) {
   ifstream f1;
   f1.open(nombre1, ios::binary);
   ofstream f2;
   f2.open(nombre2, ios::binary);
   seleccionar2(f1, f2, min, max);
   f1.close();
   f2.close();
}
```

```
/*
 * Pre: <f1> es flujo de entrada asociado a un fichero binario que almacena
 *      una secuencia de datos de tipo <int> y <f2> es flujo de salida
 *      asociado a un fichero binario que almacena datos de tipo <int>
 *      en el que se han escrito la secuencia de datos leídos del fichero
 *      asociado a <f1> cuyos valores está comprendido en [min,max]
 * Post: El fichero asociado a <f2> almacena los datos del fichero
 *       asociado al flujo <f1> cuyos valores está comprendido en el
 *       intervalo [min,max]
 */
void seleccionar2 (ifstream& f1, ofstream& f2,
                   const int min, const int max) {
   int nuevoDato;
   f1.read(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
   if (!f1.eof()) {
      if (nuevoDato >= min && nuevoDato <= max) {
         f2.write(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
      }
      seleccionar2(f1, f2, min, max);
   }
}
```

# 3.3. Búsqueda en un fichero binario

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve un valor <cierto> si y solo si el valor de algún dato
 *       del fichero <nombre> es igual a <dato> y, en tal caso, el valor
 *       final de <posicion> es igual a la posición que ocupa en el
 *       fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta (const char nombre[], const int dato, int& posicion);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve un valor <cierto> si y solo si el valor de algún dato
 *       del fichero <nombre> es igual a <dato> y, en tal caso, el valor
 *       final de <posicion> es igual a la posición que ocupa en el
 *       fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta (const char nombre[], const int dato, int& posicion);
```

**¿Tipo de diseño planteado?**

```
/*
 * Pre: <f> es un flujo asociado a fichero binario que almacena una
 *      secuencia de datos de tipo <int> y el valor de <leidos> es igual
 *      al número de datos leídos del fichero asociado a <f>
 * Post: Devuelve un valor <cierto> si y solo si el valor de alguno de los
 *       datos que aún no han sido leídos del fichero asociado a <f> en el
 *       momento en que se invocó a esta función es igual a <dato> y, en
 *       tal caso, el valor final de <posicion> es igual a la posición que
 *       ocupa en el fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta1 (ifstream& f, const int dato, int& posicion, const int leidos);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve un valor <cierto> si y solo si el valor de algún dato
 *       del fichero <nombre> es igual a <dato> y, en tal caso, el valor
 *       final de <posicion> es igual a la posición que ocupa en el
 *       fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta (const char nombre[], const int dato, int& posicion);
```

**Diseño por inmersión debilitando la postcondición**

```
/*
 * Pre: <f> es un flujo asociado a fichero binario que almacena una
 *      secuencia de datos de tipo <int> y el valor de <leidos> es igual
 *      al número de datos leídos del fichero asociado a <f>
 * Post: Devuelve un valor <cierto> si y solo si el valor de alguno de los
 *       datos que aún no han sido leídos del fichero asociado a <f> en el
 *       momento en que se invocó a esta función es igual a <dato> y, en
 *       tal caso, el valor final de <posicion> es igual a la posición que
 *       ocupa en el fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta1 (ifstream& f, const int dato, int& posicion, const int leidos);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve un valor <cierto> si y solo si el valor de algún dato
 *       del fichero <nombre> es igual a <dato> y, en tal caso, el valor
 *       final de <posicion> es igual a la posición que ocupa en el
 *       fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta (const char nombre[], const int dato, int& posicion) {
   ifstream f;
   f.open(nombre, ios::binary);
   bool encontrado = esta1(f, dato, posicion, 0);
   f.close();
   return encontrado;
}
```

```
/*
 * Pre: <f> es un flujo asociado a fichero binario que almacena una
 *      secuencia de datos de tipo <int> y el valor de <leidos> es igual
 *      al número de datos leídos del fichero asociado a <f>
 * Post: Devuelve un valor <cierto> si y solo si el valor de alguno de los
 *       datos que aún no han sido leídos del fichero asociado a <f> en el
 *       momento en que se invocó a esta función es igual a <dato> y, en
 *       tal caso, el valor final de <posicion> es igual a la posición que
 *       ocupa en el fichero <nombre> un dato cuyo valor sea igual a <dato>
   */
bool esta1 (ifstream& f, const int dato, int& posicion, const int leidos) {
   int nuevoDato;
   f.read(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
   if (!f.eof()) {
      if (nuevoDato == dato) {
         posicion = leidos + 1;
         return true;
      }
      else { return esta1(f, dato, posicion, leidos + 1); }
   }
   else { return false; }
}
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve un valor <cierto> si y solo si el valor de algún dato
 *       del fichero <nombre> es igual a <dato> y, en tal caso, el valor
 *       final de <posicion> es igual a la posición que ocupa en el
 *       fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta (const char nombre[], const int dato, int& posicion);
```

**¿Tipo de diseño planteado?**

```
/*
 * Pre: <f> es un flujo asociado a fichero binario que almacena una
 *      secuencia de datos de tipo <int>, el valor de <leidos> es igual
 *      al número de datos leídos del fichero asociado a <f> y, entre
 *      los datos leídos no consta ninguno igual a <dato>
 * Post: Devuelve un valor <cierto> si y solo si el valor de alguno de los
 *       datos del fichero asociado a <f> es igual a <dato> y, en tal caso,
 *       el valor final de <posicion> es igual a la posición que ocupa
 *       en el fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta2 (ifstream& f, const int dato, int& posicion, const int leidos);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve un valor <cierto> si y solo si el valor de algún dato
 *       del fichero <nombre> es igual a <dato> y, en tal caso, el valor
 *       final de <posicion> es igual a la posición que ocupa en el
 *       fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta (const char nombre[], const int dato, int& posicion);
```

**Diseño por inmersión reforzando la precondición**

```
/*
 * Pre: <f> es un flujo asociado a fichero binario que almacena una
 *      secuencia de datos de tipo <int>, el valor de <leidos> es igual
 *      al número de datos leídos del fichero asociado a <f> y, entre
 *      los datos leídos no consta ninguno igual a <dato>
 * Post: Devuelve un valor <cierto> si y solo si el valor de alguno de los
 *       datos del fichero asociado a <f> es igual a <dato> y, en tal caso,
 *       el valor final de <posicion> es igual a la posición que ocupa
 *       en el fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta2 (ifstream& f, const int dato, int& posicion, const int leidos);
```

```
/*
 * Pre: <nombre> es el nombre de un fichero binario que almacena una
 *      secuencia de datos de tipo <int>
 * Post: Devuelve un valor <cierto> si y solo si el valor de algún dato
 *       del fichero <nombre> es igual a <dato> y, en tal caso, el valor
 *       final de <posicion> es igual a la posición que ocupa en el
 *       fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta (const char nombre[], const int dato, int& posicion) {
   ifstream f;
   f.open(nombre, ios::binary);
   bool encontrado = esta2(f, dato, posicion, 0);
   f.close();
   return encontrado;
}
```

```
/*
 * Pre: <f> es un flujo asociado a fichero binario que almacena una
 *      secuencia de datos de tipo <int>, el valor de <leidos> es igual
 *      al número de datos leídos del fichero asociado a <f> y, entre
 *      los datos leídos no consta ninguno igual a <dato>
 * Post: Devuelve un valor <cierto> si y solo si el valor de alguno de los
 *       datos del fichero asociado a <f> es igual a <dato> y, en tal caso,
 *       el valor final de <posicion> es igual a la posición que ocupa
 *       en el fichero <nombre> un dato cuyo valor sea igual a <dato>
 */
bool esta2 (ifstream& f, const int dato, int& posicion, const int leidos) {
   int nuevoDato;
   f.read(reinterpret_cast<char *>(&nuevoDato), sizeof(int));
   if (!f.eof()) {
      if (nuevoDato == dato) {
         posicion = leidos + 1;
         return true;
      }
      else { return esta2(f, dato, posicion, leidos + 1); }
   }
   else { return false; }
}
```