Programación 2

Prueba formal de la corrección de un algoritmo con invocaciones a funciones

Problemas 13

Índice:

1. **Prueba de la corrección del diseño recursivo, sin inmersión, de la función `sumarDigitos(n)`**
2. **Prueba de la corrección del diseño recursivo, con inmersión, de la función `sumar(v,n)`**
3. **Prueba de la corrección del diseño recursivo, con inmersión, de la función `buscar(v,n,x)`**
4. **Prueba de la corrección del diseño de la función `menor(v,n)`, con invocaciones a una función auxiliar**

Problema 1. Demostrar formalmente la corrección del código de la función `sumarDigitos(n)`

```
/*
 * Pre: n ≥ 0
 * Post: sumarDigitos(n) = (Σα∈[1,∞]. digito(n,α))
 */
int sumarDigitos (const int n) {
    if (n < 10) {
        return n;
    }
    else {
        return n % 10 + sumarDigitos(n / 10);
    }
}
     // Donde: digito(n,i) = (n / 10^(i-1)) % 10
```

```
/*
 * Pre: n ≥ 0
 * Post: sumarDigitos(n) = (Σα∈[1,∞]. digito(n,α))
 */
int sumarDigitos (const int n) {
   // n ≥ 0    ¿ ⇒ ?   Dom(n < 10) ≡ cierto
   if (n < 10) {
      // n ≥ 0 ∧ n < 10  ¿ ⇒ ?  n = (Σα∈[1,∞]. digito(n,α))
      return n;
   }
   else {
      // n ≥ 0 ∧ n ≥ 10
      // ¿ ⇒ ?
      // n % 10 + sumarDigitos(n/10) = (Σα∈[1,∞].  digito(n,α))
      return n % 10 + sumarDigitos(n / 10);
   }
   // sumarDigitos(n) = (Σα∈[1,∞].digito(n,α))
}
```

1º

2º

3º

```
/*
 * Pre: n ≥ 0  */
 * Post: sumarDigitos(n) = (Σα∈[1,∞]. digito(n,α))
 */
int sumarDigitos (const int n) {
   // n ≥ 0  ⇒  Dom(n < 10) ≡ cierto
   if (n < 10) {
      // n ≥ 0 ∧ n < 10  ⇒  n = (Σα∈[1,∞]. digito(n,α))
      return n;
   }
   else {
      // n ≥ 10 ⇒ n / 10 ≥ 0    // ok PRE de sumarDigitos(n/10)
      // n ≥ 10  ∧ … ??? …
      // ¿ ⇒ ?
      // n % 10 + sumarDigitos(n/10) = (Σα∈[1,∞]. digito(n,α))
      return n % 10 + sumarDigitos(n / 10);
   }
   // sumarDigitos(n) = (Σα∈[1,∞].digito(n,α))
}
```

```
/*
 * Pre: n ≥ 0  */
 * Post: sumarDigitos(n) = (Σα∈[1,∞].digito(n,α))
 */
int sumarDigitos (const int n) {
   // n ≥ 0  ⇒  Dom(n<10) ≡ cierto
   if (n < 10) {
      // n ≥ 0 ∧ n < 10  ⇒  n = (Σα∈[1,∞]. digito(n,α))
      return n;
   }
   else {
      // n ≥ 0 ∧ n ≥ 10  ⇒  n / 10 ≥ 0
      // n ≥ 10 ∧ sumarDigitos(n/10) = (Σα∈[1,∞]. digito(n/10,α))
      // ⇒
      // n % 10 + sumarCifras(n/10) = (Σα∈[1,∞]. digito(n,α))
      return n % 10 + sumarDigitos(n / 10);
   }
}
```

```
// Pre: n ≥ 0
// Post: sumarDigitos(n) = (Σα∈[1,∞]. digito(n,α))
int sumarDigitos (const int n) {  // Termin.: f_cota = n
   // n ≥ 0   // f_cota(inicial) = n  ∧  f_cota(inicial) ≥ 0
   if (n < 10) {
      // n ≥ 0 ∧ n < 10
      return n;
   }
   else {
      // n ≥ 10
      return n % 10 + sumarDigitos(n / 10);
      // f_cota(invocación) = n / 10 ∧ f_cota(invocación) ≥ 0
   }
}
```

4º

5º

La terminación de la recurrencia queda demostrada ya que:

1. $f_{cota(inicial)} > f_{cota(invocación)}$ decrece **n > n / 10** ya que **n ≥ 10**
2. $f_{cota(inicial)} \geq 0$ y $f_{cota(innvocación)} \geq 0$, luego la función de cota está acotada inferiormente en $\mathbb{Z}$

Problema 2. Demostrar formalmente la corrección del diseño por inmersión de la función `sumar(v,n)`

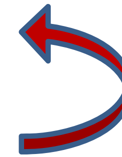
```
// Pre: i >= 0 ∧ i <= j ∧ j < #v
// Post: sumar(v,i,j) = (Σα∈[i,j]. v[α])
double sumar (const double v[], const int i, const int j) {
   if (i == j) {
      return v[i];
   }
   else {
      return sumar(v, i, (i+j)/2) + sumar(v, (i+j)/2 + 1, j);
   }
}
```

```
// Pre: n > 0 ∧ n <= #v
// Post: sumar(v,n) = (Σα∈[0,n-1]. v[α])
double sumar (const double v[], const int n) {
   return sumar(v, 0, n-1);
}
```

1. Primero, se va a probar la corrección de la función `sumar(v,n)`

```
// Pre: n > 0 ∧ n <= #v
// Post: sumar(v,n) = (Σα∈[0,n-1]. v[α])
double sumar (const double v[], int n) {
   // n > 0  ⇒  0 >= 0 ∧ 0 <= n - 1 ∧ n – 1 < #v
   // n > 0  ∧  … ??? …
   // ¿ ⇒ ?
   // sumar(v,0,n-1) = (Σα∈[0,n-1]. v[α])
   return sumar(v, 0, n-1);
   // sumar(v,n) = (Σα∈[0,n-1]. v[α])
}
```

1º



```
// Pre: i >= 0 ∧ i <= j ∧ j < #v
// Post: sumar(v,i,j) = (Σα∈[i,j]. v[α])
double sumar (const double v[], const int i, const int j) {
   …
}
```

```
// Pre: n > 0 ∧ n <= #v
// Post: sumar(v,n) = (Σα∈[0,n-1]. v[α])
double sumar (const double v[], const int n) {
   // n > 0 ∧ n <= #v  ⇒  0 >= 0 ∧ 0 <= n - 1 ∧ n – 1 < #v
   // n > 0 ∧ sumar(v,0,n-1) = (Σα∈[0,n-1]. v[α])
   // ⇒
   // sumar(v,0,n-1) = (Σα∈[0,n-1].v[α])
   return sumar(v, 0, n-1);
   // sumar(v,n) = (Σα∈[0,n-1].v[α])
}
```

```
// Pre: i >= 0 ∧ i <= j ∧ j < #v
// Post: sumar(v,i,j) = (Σα∈[i,j]. v[α])
double sumar (const double v[], const int i, const int j) {
   …
}
```

2. Y ahora, se va a probar la corrección de la función `sumar(v,i,j)`

```
// Pre: i >= 0 ∧ i <= j ∧ j < #v
// Post: sumar(v,i,j) = (Σα∈[i,j]. v[α])
double sumar (const double v[], const int i, const int j) {
   // i ≥ 0 ∧ i <= j ∧ j < #v ¿ ⇒ ?  Dom(i = j) ≡ cierto
   if (i == j) {
      // i ≥ 0 ∧ i = j ∧ j < #v  ¿ ⇒ ?  v[i] = (Σα∈[i,j]. v[α])
      return v[i];
   }
   else {
      // i ≥ 0 ∧ i < j ∧ j < #v ⇒
      // i >= 0 ∧ i ≤ (i+j)/2  ∧ (i+j)/2 < #v    // Pre de sumar(v,i,(i+j)/2)
      // i ≥ 0 ∧ i < j ∧ j < #v ⇒
      // (i+j)/2 >= 0 ∧ (i+j)/2 + 1 ≤ j ∧ j < #v // Pre de sumar(v,(i+j)/2+1,j)
      // i ≥ 0 ∧ i < j ∧ j < #v ∧ sumar(v,i,(i+j)/2) = …
      // ∧ sumar(v,(i+j)/2+1,j) = …
      // ¿ ⇒ ?
      // sumar(v,i,(i+j)/2) + sumar(v,(i+j)/2+1,j) = (Σα∈[i,j].v[α])
      return sumar(v, i, (i+j)/2) + sumar(v, (i+j)/2+1, j);
   }
   // sumar(v,i,j) = (Σα∈[i,j].v[α])
}
```

2º

3º

4º

```
// Pre: i >= 0 ∧ i <= j ∧ j < #v
// Post: sumar(v,i,j) = (Σα∈[i,j]. v[α])
double sumar (const double v[], const int i, const int j) {
   // i ≥ 0 ∧ i <= j ∧ j < #v ¿ ⇒ ?  Dom(i = j) ≡ cierto
   if (i == j) {
      // i ≥ 0 ∧ i = j ∧ j < #v  ¿ ⇒ ?  v[i] = (Σα∈[i,j]. v[α])
      return v[i];
   }
   else {
      // i ≥ 0 ∧ i < j ∧ j < #v ⇒
      // i >= 0 ∧ i ≤ (i+j)/2  ∧ (i+j)/2 < #v    // Pre de sumar(v,i,(i+j)/2)
      // i ≥ 0 ∧ i < j ∧ j < #v ⇒
      // (i+j)/2 >= 0 ∧ (i+j)/2 + 1 ≤ j ∧ j < #v // Pre de sumar(v,(i+j)/2+1,j)
      // i ≥ 0 ∧ i < j ∧ j < #v
      // ∧ sumar(v,i,(i+j)/2) = (Σα∈[i,(i+j)/2].v[α]) ∧
      // sumar(v,(i+j)/2+1,j) = (Σα∈[(i+j)/2+1,j].v[α])
      // ⇒
      // sumar(v,i,(i+j)/2) + sumar(v,(i+j)/2+1,j) = (Σα∈[i,j].v[α])
      return sumar(v, i, (i+j)/2) + sumar(v, (i+j)/2+1, j);
   }
   // sumar(v,i,j) = (Σα∈[i,j].v[α])
}
```

2º

3º

4º

```
// Pre: i >= 0 ∧ i <= j ∧ j < #v
// Post: sumar(v,i,j) = (Σα∈[i,j]. v[α])
double sumar (const double v[], const int i, const int j) { // fcota = j-i
   // i >= 0 ∧ i ≤ j  ∧ j < #v   ⇒   fcota(inicial) = j - i ≥ 0
   if (i == j) {
      // i = j
      return v[i];
   }
   else {
      // i >= 0 ∧ i < j ∧ j < #v
      return sumar(v, i, (i+j)/2) + sumar(v, (i+j)/2+1, j);
      // fcota_inv_1 = (i+j)/2 - i  ∧  j - i > (i+j)/2 - i
      // fcota_inv_2 = j - ((i+j)/2 + 1) ∧ j - i > j - ((i+j)/2 + 1)
   }
}
```

5º

6º

7º

Prueba de la terminación:

- Decrece: $f_{cota(inicial)} > f_{cota_inv_1}$ y $f_{cota(inicial)} > f_{cota_inv_2}$
- No puede decrecer indefinidamente ya que $f_{cota} = j - i \geq 0$ está acotada inferiormente en Z

Problema 3. Demostrar formalmente la corrección del diseño por inmersión de la función `buscar(v,n,x)`

```
/*
 *   Pre: n > 0 ∧ n <= #v
 *   Post: ((∃α∈[0,n-1]. v[α] = x) → v[buscar(v,n,x)] = x)
 *         ∧ ((∀α∈[0,n-1]. v[α] != x) → buscar(v,n,x) < 0)
 */
int buscar (const int v[], const int n, const int x) {
   return buscar(v, x, 0, n-1);
}
```

```
/*
 * Pre: desde >= 0 ∧ desde <= hasta ∧ hasta < #v
 * Post: ((∃α∈[desde,hasta].v[α]=x) → v[buscar(v,x,desde,hasta)] = x)
 *        ∧ ((∀α∈[desde,hasta].v[α]!=x) → buscar(v,x,desde,hasta) < 0)
 */
int buscar (const int v[], const int x, const int desde,
            const int hasta) {
   if (desde == hasta) {
      if (v[desde] == x) { return desde; }
      else { return -1; }
   }
   else {
      int medio = (desde + hasta) / 2;
      int r = buscar(v, x, desde, medio);
      if (r >= 0) { return r; }
      else {
         return buscar(v, x, medio + 1, hasta);
      }
   }
}
```

```
/*
 *  Pre: n > 0 ∧ n <= #v
 *  Post: ((∃α∈[0,n-1].v[α]=x) → v[buscar(v,n,x)] = x) ∧
 *        ((∀α∈[0,n-1].v[α]!=x) → buscar(v,n,x) < 0)
 */
int buscar (const int v[], const int n, const int x) {
   // n > 0 ∧ n <= #v ⇒  0 ≥ 0 ∧ 0 ≤ n - 1 ∧ n - 1 <= #v
   // n > 0 ∧ n <= #v ∧ ((∃α∈[0,n-1].v[α]=x) → v[buscar(v,x,0,n-1)] = x)
   // ∧ ((∀α∈[0,n-1].v[α]!=x) → buscar(v,x,0,n-1) < 0)
   // ⇒
   // ((∃α∈[0,n-1].v[α]=x) → v[buscar(v,x,0,n-1)] = x) ∧
   // ((∀α∈[0,n-1].v[α]!=x) → buscar(v,x,0,n-1) < 0)
   return buscar(v, x, 0, n-1);
}
```

1º

```
/*
 * Pre: desde >= 0 ∧ desde <= hasta ∧ hasta < #v
 * Post: ((∃α∈[desde,hasta]. v[α] = x) → v[buscar(v,x,desde,hasta)] = x)
 *       ∧ ((∀α∈[desde,hasta]. v[α] != x) → buscar(v,x,desde,hasta) < 0)
 */
int buscar (const int v[], const int x, const int desde, const int hasta);
```

```
// Pre: desde >= 0 ∧ desde <= hasta ∧ hasta < #v
// Post: ((∃α∈[desde,hasta].v[α]=x) → v[buscar(v,x,desde,hasta)] = x) ∧
//       ((∀α∈[desde,hasta].v[α]!=x) → buscar(v,x,desde,hasta) < 0)
int buscar (const int v[], const int x, const int desde, const int hasta) {
   // desde >= 0 ∧ desde <= hasta ∧ hasta < #v
   // ⇒  Dom(desde = hasta) ≡ cierto
   if (desde == hasta)
      if (v[desde] == x) {
          // desde >= 0 ∧ desde <= hasta ∧ hasta < #v ∧ v[desde] = x
          //  ⇒
          // ((∃α∈[desde,hasta]. v[α] = x) → v[desde] = x) ∧
          // ((∀α∈[desde,hasta]. v[α] != x) → desde < 0)
          return desde;
      }
      else { return -1; }
   else {
      int medio = (desde + hasta) / 2;
      int r = buscar(v, x, desde, medio);
      if (r >= 0) { return r;}
      else { return buscar(v, x, medio + 1, hasta); }
   }
}
```

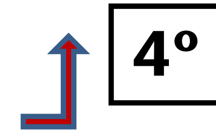
2º

```
// Pre: desde >= 0 ∧ desde <= hasta ∧ hasta < #v
// Post: ((∃α∈[desde,hasta].v[α]=x) → v[buscar(v,x,desde,hasta)] = x) ∧
//       ((∀α∈[desde,hasta].v[α]!=x) → buscar(v,x,desde,hasta) < 0)
int buscar (const int v[], const int x, const int desde,
            const int hasta) {
   // desde >= 0 ∧ desde <= hasta ∧ hasta < #v
   if (desde == hasta)
      if (v[desde] == x) { return desde; }
      else {
          // desde >= 0 ∧ desde = hasta ∧ hasta < #v ∧ v[desde] ≠ x
          // ⇒
          // ((∃α∈[desde,hasta]. v[α] = x) → v[-1] = x) ∧
          // ((∀α∈[desde,hasta]. v[α] != x) → -1 < 0)
          return -1;
      }
   else {
      int medio = (desde + hasta) / 2;
      int r = buscar(v, x, desde, medio);
      if (r >= 0) { return r; }
      else { return buscar(v, x, medio + 1, hasta); }
   }
}
```

3º

```
// Pre: desde >= 0 ∧ desde <= hasta ∧ hasta < #v
// Post: ((∃α∈[desde,hasta].v[α] = x) → v[buscar(v,x,desde,hasta)] = x) ∧
//       ((∀α∈[desde,hasta].v[α] != x) → buscar(v,x,desde,hasta) < 0)
int buscar (const int v[], const int x, const int desde, const int hasta) {
   // desde >= 0 ∧ desde <= hasta ∧ hasta < #v
   if (desde == hasta)
      if (v[desde] == x) { return desde; }
      else  { return -1; }
   else { // desde >= 0 ∧ desde < hasta ∧ hasta < #v
      int medio = (desde + hasta) / 2;
      // desde >= 0 ∧ desde < hasta ∧ hasta < #v ∧ medio = (desde + hasta)/2
      int r = buscar(v, x, desde, medio);
      if (r >= 0) { return r; }
      else { return buscar(v, x, medio + 1, hasta); }
   }
   // ((∃α∈[desde,hasta].v[α]=x) → v[buscar(v,x,desde,hasta)] = x) ∧
   // ((∀α∈[desde,hasta].v[α]!=x) →  buscar(v,x,desde,hasta) < 0)
}
```

```
else { // desde >= 0 ∧ desde < hasta ∧ hasta < #v
       // ⇒  desde >= 0 ∧ desde < hasta ∧ hasta < #v
              ∧ (desde + hasta) / 2 = (desde + hasta) / 2
   int medio = (desde + hasta)/2;
   // desde >= 0 ∧ desde < hasta ∧ hasta < #v ∧ medio = (desde + hasta)/2
   //  ⇒  desde >= 0 ∧ desde ≤ medio
   // desde >= 0 ∧ desde < hasta ∧ hasta < #v ∧ medio = (desde + hasta)/2
   // ∧ ((∃α∈[desde,medio].v[α] = x) → v[buscar(v,x,desde,medio)] = x)
   // ∧ ((∀α∈[desde,medio].v[α] != x) → buscar(v,x,desde,medio) < 0)
   int r = buscar(v, x, desde, medio);
   // desde >= 0 ∧ desde < hasta ∧ hasta < #v ∧ medio = (desde + hasta)/2
   // ∧ ((∃α∈[desde,medio].v[α] = x) → v[r] = x)
   // ∧ ((∀α∈[desde,medio].v[α]! = x) → r < 0)
   if (r >= 0) {
      return r;
   }
   else {
      return buscar(v, x, medio+1, hasta);
   }
   // ((∃α∈[desde,hasta].v[α] = x) → v[buscar(v,x,desde,hasta)] = x) ∧
   // ((∀α∈[desde,hasta].v[α] != x) →  buscar(v,x,desde,hasta) < 0)
}
```



```
else {   // desde >= 0 ∧ desde < hasta
   . . .
   // desde >= 0 ∧ desde < hasta ∧ hasta < #v ∧ medio = (desde + hasta)/2
   // ∧ ((∃α∈[desde,medio]. v[α] = x) → v[r] = x)
   // ∧ ((∀α∈[desde,medio]. v[α] != x) → r < 0)
   if (r >= 0) {
      // desde >= 0 ∧ desde < hasta ∧ hasta < #v
      // ∧ medio = (desde + hasta)/2 ∧ r ≥ 0
      // ∧ ((∃α∈[desde,medio]. v[α] = x) → v[r] = x)
      // ∧ ((∀α∈[desde,medio]. v[α] != x) → r < 0)
      return r;
   }
   else {
      // desde >= 0 ∧ desde < hasta ∧ hasta < #v
      // ∧ medio = (desde + hasta) / 2 ∧ r < 0
      // ∧ ((∃α∈[desde,medio]. v[α] = x) → v[r] = x)
      // ∧ ((∀α∈[desde,medio]. v[α] != x) → r < 0)
      return buscar(v, x, medio + 1, hasta);
   }
   // ((∃α∈[desde,hasta].v[α] = x) → v[buscar(v,x,desde,hasta)] = x) ∧
   // ((∀α∈[desde,hasta].v[α] !=x ) → buscar(v,x,desde,hasta) < 0)
}
```

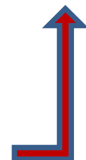
```
. . .
if (r >= 0) {
   // desde >= 0 ∧ desde < hasta ∧ hasta < #v
   // ∧ medio = (desde + hasta) / 2 ∧ r ≥ 0
   // ∧ ((∃α∊[desde,medio]. v[α] = x) → v[r] = x)
   // ∧ ((∀α∊[desde,medio]. v[α] !=x ) → r < 0)
   // ⇒
   // ((∃α∊[desde,hasta]. v[α] = x) → v[r] = x) ∧
   // ((∀α∊[desde,hasta]. v[α] != x) → r < 0)
   return r;
}
else {
   // desde >= 0 ∧ desde < hasta ∧ hasta < #v
   // ∧ medio = (desde + hasta) / 2 ∧ r < 0
   // ∧ ((∃α∊[desde,medio]. v[α] = x) → v[r] = x)
   // ∧ ((∀α∊[desde,medio]. v[α] != x) → r < 0)
   //   ⇒
   // medio + 1 >= 0 ∧ medio + 1 ≤ hasta ∧ hasta < #v
   return buscar(v, x, medio+1, hasta);
}
// ((∃α∊[desde,hasta].v[α] = x) → v[buscar(v,x,desde,hasta)] = x) ∧
// ((∀α∊[desde,hasta].v[α] != x) → buscar(v,x,desde,hasta) < 0)
```

5º

```
else {  // desde >= 0 ∧ desde < hasta ∧ hasta < #v
   . . .
   if (r >= 0) { return r;  }
   else {
      // desde >= 0 ∧ desde < hasta ∧ hasta < #v ∧ medio = (desde + hasta) / 2
      // ∧ r < 0 ∧ ((∃α∈[desde,medio]. v[α] = x) → v[r] = x)
      // ∧ ((∀α∈[desde,medio]. v[α] != x) → r < 0)
      // ⇒
      // medio + 1 >= 0 ∧ medio + 1 ≤ hasta ∧ hasta < #v

      // desde >= 0 ∧ desde < hasta ∧ hasta < #v ∧ medio = (desde + hasta )/ 2
      // ∧ r < 0 ∧ ((∃α∈[desde,medio]. v[α] = x) → v[r] = x)
      // ∧ ((∀α∈[desde,medio]. v[α] != x) → r < 0)
      // ∧ ((∃α∈[medio+1,hasta].v[α] = x) → v[buscar(v,x,medio+1,hasta)] = x)
      // ∧ ((∀α∈[medio+1,hasta].v[α] != x) → buscar(v,x,medio+1,hasta) < 0)
      //  ⇒
      // ((∃α∈[desde,hasta].v[α]=x) → v[buscar(v,x,medio+1,hasta)] = x) ∧
      // ((∀α∈[desde,hasta].v[α]!=x) → buscar(v,x,medio+1,hasta) < 0)
      return buscar(v, x, medio+1, hasta);
   }
   // ((∃α∈[desde,hasta].v[α] = x) → v[buscar(v,x,desde,hasta)] = x) ∧
   // ((∀α∈[desde,hasta].v[α] != x) → buscar(v,x,desde,hasta) < 0)
}
```

6º



```
/*
 * Pre: desde >= 0 ∧ desde <= hasta ∧ hasta < #v
 * Post: ((∃α∈[desde,hasta].v[α] = x) → v[buscar(v,x,desde,hasta)] = x) ∧
 *       ((∀α∈[desde,hasta].v[α] != x) → buscar(v,x,desde,hasta) < 0)
 */
int buscar (const int v[], const int x, const int desde, const int hasta) {
   // Terminación: f_cota = hasta - desde
   // desde >= 0 ∧ desde <= hasta ∧ hasta < #v
   //  → f_cota(inicial) = hasta - desde ≥ 0
   if (desde == hasta) {               // Acotada inferiormente en ℕ
      if (v[desde] == x) { return desde; } else { return -1; }
   }
   else { // desde >= 0 ∧ desde < hasta ∧ hasta < #v
      int medio = (desde + hasta) / 2;
      int r = buscar(v, x, desde, medio);
      // f_cota_inv_1 = (desde + hasta) / 2 - desde ∧ desde < hasta →
      // hasta - desde > (desde + hasta)/2 - desde  // decrece en ℤ
      if (r >= 0) { return r; }
      else { return buscar(v, x, medio+1, hasta);
      // f_cota_inv_2 = hasta – (desde + hasta) / 2 - 1 ∧ desde < hasta →
      // hasta - desde > hasta – (desde + hasta)/2 - 1  // decrece en ℤ
   }
}
```

7º

8º

9º

Problema 4. Demostrar formalmente la corrección de la función `menor(v,n)`, asumiendo que la función auxiliar `menor(v,i,j)` es correcta.

```
/*
 * Pre: n > 0 AND n <= #v
 * Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
 */
double menor (const double v[], const int n) {
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
   if (m1 <= m2) {  return m1;  }
   else {  return m2;  }
}
```

```
// Pre: i >= 0 AND i <= j AND j < #v
// Post: menor(v,i,j) = (MIN alfa EN [i,j]. v[alfa])
double menor (const double v[], const int i, const int j);
```

Estrategia:

1. Escribir el predicado más fuerte **P** que se satisfaga tras ejecutar las dos primeras instrucciones.
2. Demostrar formalmente que las dos primeras instrucciones son correctas, para su postcondición **P**.
3. Demostrar formalmente que la instrucción condicional es también correcta, para su precondición **P**.

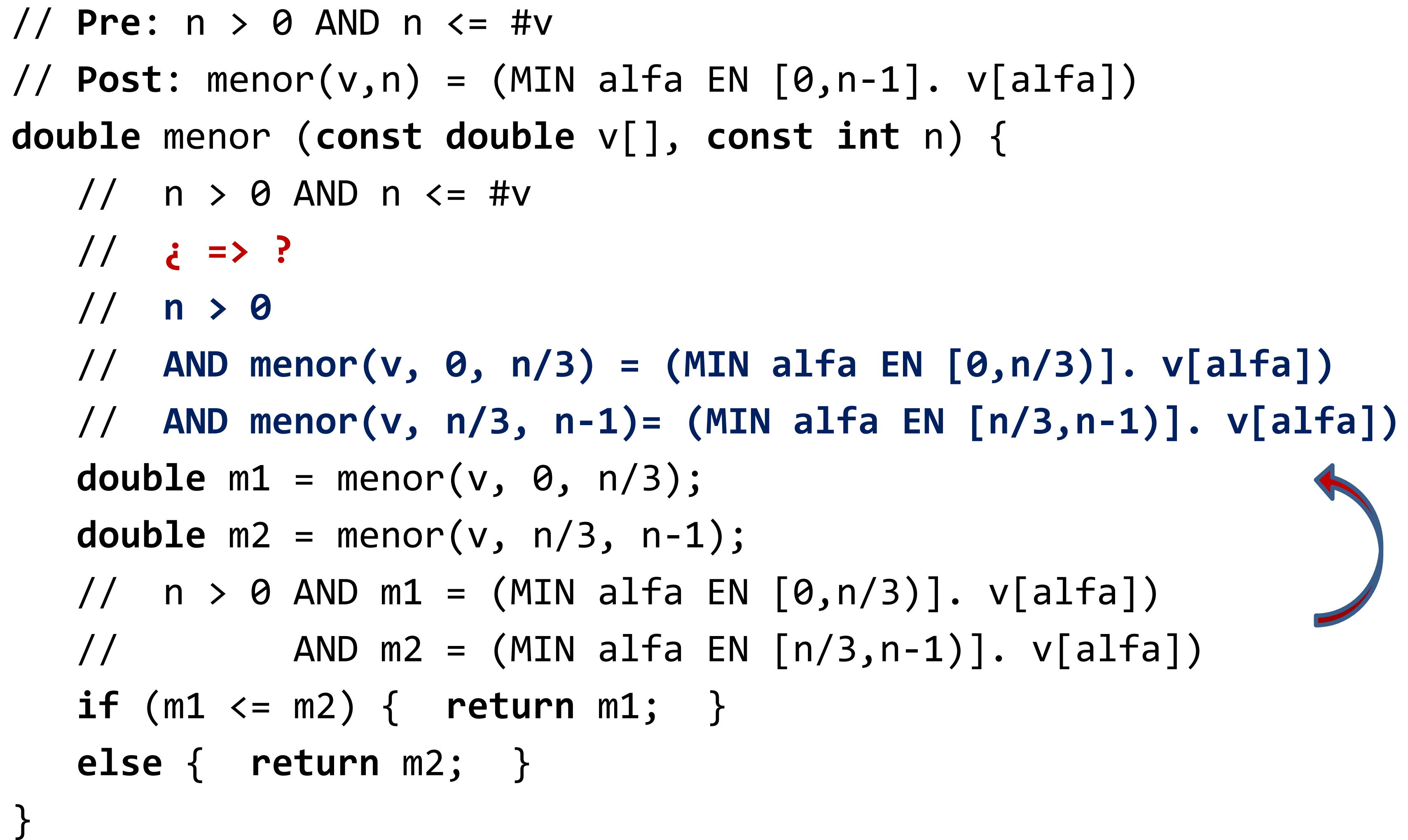
```
/*
 * Pre: n > 0 AND n <= #v
 * Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
 */
double menor (const double v[], const int n) {
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
   // P
   if (m1 <= m2) {  return m1;  }
   else {  return m2;  }
}
```

1. Escribir el predicado más fuerte **P** que se satisfaga tras ejecutar las dos primeras instrucciones.

```
/*
 * Pre: n > 0 AND n <= #v
 * Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
 */
double menor (const double v[], const int n) {
   //  n > 0 AND n <= #v
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
   //  n > 0
   //  AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
   //  AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
   if (m1 <= m2) {  return m1;  }
   else {  return m2;  }
   //  menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
}
```

2. Demostrar formalmente que las dos primeras instrucciones son correctas, para su postcondición **P**.

```
/*
 * Pre: n > 0 AND n <= #v
 * Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
 */
double menor (const double v[], const int n) {
   //  n > 0 AND n <= #v
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
   //  n > 0 AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
   //        AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
   if (m1 <= m2) {  return m1;  }
   else {  return m2;  }
}
```


// Pre: n > 0 AND n <= #v
// Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
double menor (const double v[], const int n) {
// n > 0 AND n <= #v
// ¿ => ?
// n > 0
// AND menor(v, 0, n/3) = (MIN alfa EN [0,n/3)]. v[alfa])
// AND menor(v, n/3, n-1)= (MIN alfa EN [n/3,n-1)]. v[alfa])
double m1 = menor(v, 0, n/3);
double m2 = menor(v, n/3, n-1);
// n > 0 AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
// AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
if (m1 <= m2) { return m1; }
else { return m2; }
}

```
// Pre: n > 0 AND n <= #v
// Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
double menor (const double v[], const int n) {
   //  n > 0 AND n <= #v  =>  0 >= 0 AND 0 <= n/3 AND n/3 < #v
   //  n > 0 AND n <= #v  =>  n/3 >= 0 AND n/3 <= n-1 AND n-1 < #v

   //  n > 0  AND …
   //  ¿ => ?
   //  n > 0
   //  AND menor(v, 0, n/3) = (MIN alfa EN [0,n/3)]. v[alfa])
   //  AND menor(v, n/3, n-1)= (MIN alfa EN [n/3,n-1)]. v[alfa])
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
   if (m1 <= m2) {  return m1;  }
   else {  return m2;  }
}
```

```
// Pre: i >= 0 AND i <= j AND j < #v
// Post: menor(v,i,j) = (MIN alfa EN [i,j]. v[alfa])
double menor (const double v[], const int i, const int j);
```

```
// Pre: n > 0 AND n <= #v
// Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
double menor (const double v[], const int n) {
   //  n > 0 AND n <= #v  =>  0 >= 0 AND 0 <= n/3 AND n/3 < #v
   //  n > 0 AND n <= #v  =>  n/3 >= 0 AND n/3 <= n-1 AND n-1 < #v

   //  n > 0 AND n <= #v
   //  AND menor(v, 0, n/3) = (MIN alfa EN [0,n/3)]. v[alfa])
   //  AND menor(v, n/3, n-1)= (MIN alfa EN [n/3,n-1)]. v[alfa])
   //  ¿ => ?
   //  n > 0
   //  AND menor(v, 0, n/3) = (MIN alfa EN [0,n/3)]. v[alfa])
   //  AND menor(v, n/3, n-1)= (MIN alfa EN [n/3,n-1)]. v[alfa])
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
    ...
}
```

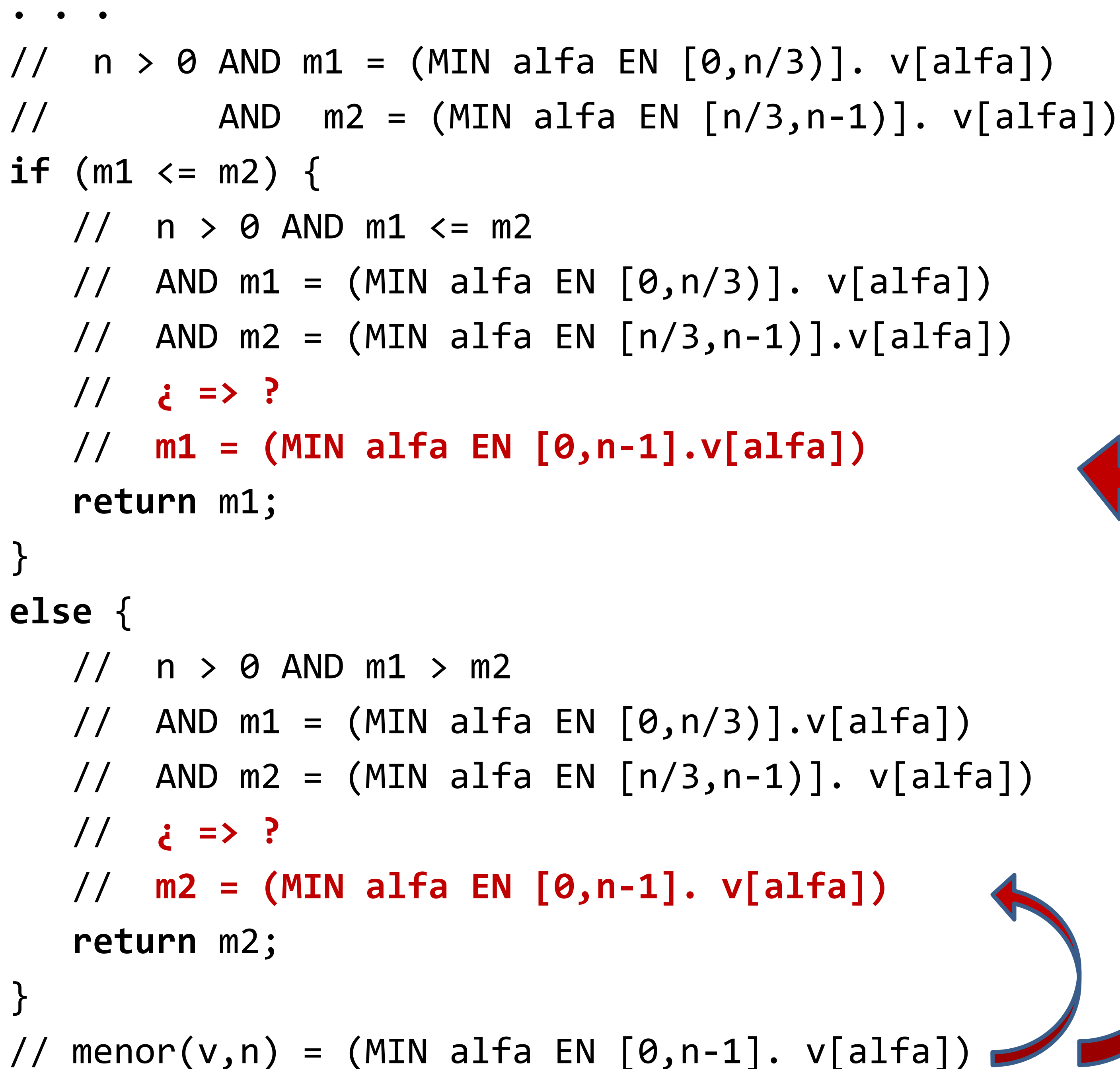
```
// Pre: i >= 0 AND i <= j AND j < #v
// Post: menor(v,i,j) = (MIN alfa EN [i,j]. v[alfa])
double menor (const double v[], const int i, const int j);
```

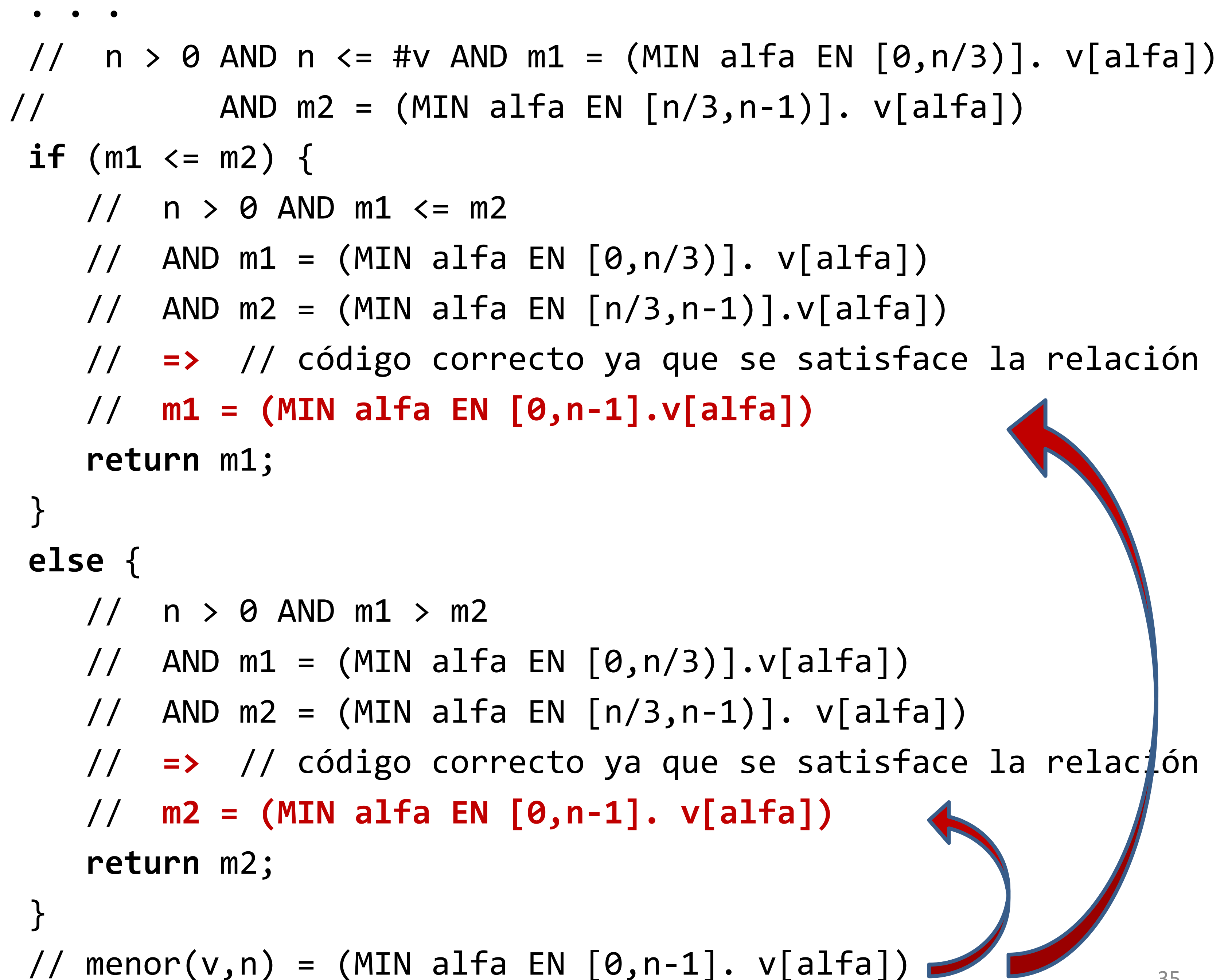
```
// Pre: n > 0 AND n <= #v
// Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
double menor (const double v[], const int n) {
   //  n > 0 AND n <= #v  =>  0 >= 0 AND 0 <= n/3 AND n/3 < #v
   //  n > 0 AND n <= #v  =>  n/3 >= 0 AND n/3 <= n-1 AND n-1 < #v

   //  n > 0 AND n <= #v
   //  AND menor(v, 0, n/3) = (MIN alfa EN [0,n/3)]. v[alfa])
   //  AND menor(v, n/3, n-1)= (MIN alfa EN [n/3,n-1)]. v[alfa])
   //  =>  // código correcto ya que se satisface la relación
   //  n > 0 AND
   //  menor(v, 0, n/3) = (MIN alfa EN [0,n/3)]. v[alfa]) AND
   //  menor(v, n/3, n-1)= (MIN alfa EN [n/3,n-1)]. v[alfa])
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
   if (m1 <= m2) {  return m1;  }
   else {  return m2;  }
}
```

3. Demostrar formalmente que la instrucción condicional es también correcta, para su precondición P.

```
// Pre: n > 0 AND n <= #v
// Post: menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])
double menor (const double v[], const int n) {
   double m1 = menor(v, 0, n/3);
   double m2 = menor(v, n/3, n-1);
   //  P: n > 0
   //      AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
   //      AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
   if (m1 <= m2) {
      return m1;
   }
   else {
      return m2;
   }
   // menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa]) */
}
```


. . .
// n > 0 AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
// AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
if (m1 <= m2) {
// n > 0 AND m1 <= m2
// AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
// AND m2 = (MIN alfa EN [n/3,n-1)].v[alfa])
// ¿ => ?
// m1 = (MIN alfa EN [0,n-1].v[alfa])
return m1;
}
else {
// n > 0 AND m1 > m2
// AND m1 = (MIN alfa EN [0,n/3)].v[alfa])
// AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
// ¿ => ?
// m2 = (MIN alfa EN [0,n-1]. v[alfa])
return m2;
}
// menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])


. . .
// n > 0 AND n <= #v AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
// AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
if (m1 <= m2) {
// n > 0 AND m1 <= m2
// AND m1 = (MIN alfa EN [0,n/3)]. v[alfa])
// AND m2 = (MIN alfa EN [n/3,n-1)].v[alfa])
// => // código correcto ya que se satisface la relación
// m1 = (MIN alfa EN [0,n-1].v[alfa])
return m1;
}
else {
// n > 0 AND m1 > m2
// AND m1 = (MIN alfa EN [0,n/3)].v[alfa])
// AND m2 = (MIN alfa EN [n/3,n-1)]. v[alfa])
// => // código correcto ya que se satisface la relación
// m2 = (MIN alfa EN [0,n-1]. v[alfa])
return m2;
}
// menor(v,n) = (MIN alfa EN [0,n-1]. v[alfa])