Grado en Ingeniería Informática de Gestión y Sistemas de Información

Escuela de Ingeniería de Bilbao
UPV / EHU

Departamento de Lenguajes y Sistemas Informáticos

METODOLOGÍA DE LA PROGRAMACIÓN

Curso: 1º

Grupo: 01

Curso académico: 2022-2023

Tema 2
Especificación y documentación de programas

Tema 2
Especificación y documentación de programas

2.1. Introducción

2.1.1. Objetivo

A la hora de desarrollar un programa o un sistema es fundamental indicar o especificar de manera precisa qué debe hacer un programa.

Para indicar o especificar de manera precisa qué ha de realizar un programa, hay que utilizar un lenguaje de especificación formal.

Utilizando un lenguaje de especificación formal se consigue que no haya ambigüedades.

El primer objetivo de este tema es aprender a especificar programas utilizando el lenguaje de la **Lógica de Primer Orden** junto a la técnica conocida como **especificación pre-post**.

El segundo objetivo es utilizar el lenguaje de la Lógica de Primer Orden para documentar programas, indicando qué propiedades se cumplen en distintos puntos del programa.

2.1.2. Definición de Especificación

A continuación se recogen dos definiciones generales de "Especificación":

- Documento que delimita, de forma completa y precisa, las características, requerimientos o comportamiento de un sistema (o un componente de un sistema).
- Afirmación concisa de un conjunto de requerimientos que deben ser satisfechos por un producto, material o proceso.

2.1.3. Corrección de un programa

Para poder hablar de "corrección" de un algoritmo (o programa) es necesario describir con precisión (o especificar) la relación que existe entre los datos de entrada proporcionados y los resultados esperados.

La corrección es la coincidencia entre el comportamiento deseado (especificación) y el comportamiento real del programa.

2.2. Especificación pre-post

2.2.1. Precondición y Postcondición

Expresaremos el comportamiento esperado de los algoritmos mediante la especificación pre-post.

Una especificación pre-post consta de una precondición y de una postcondición.

La **precondición** describe las condiciones iniciales que deben cumplir los datos de entrada.

La **postcondición** describe la relación entre dichos datos y los resultados obtenidos.

El significado de una especificación pre-post es el siguiente: Suponiendo que al comienzo los datos de entrada cumplen la precondición, entonces al final los resultados han de cumplir la postcondición.

Ejemplo 1:

```
Pre ≡ {x ≥ 9}
        x : = x + 5;
Post ≡ {x ≥ 14}
```

Esa especificación indica que si antes de la asignación x tiene un valor mayor o igual que 9, después de la asignación tendrá un valor mayor o igual que 14.

Ejemplo 2:
La especificación pre-post correspondiente al programa que obtiene en la variable s la suma de los elementos de un vector no vacío A(1..n) de enteros es la siguiente:

```
Pre ≡ {n ≥ 1}
i : = 0; s : = 0;
while i < n loop
          i : = i + 1;
          s : = s + A(i);
end loop;
Post ≡ {s = Σ_{k=1}^{n} A(k) }
```

2.2.2. Diseño por contrato

Una especificación pre-post define un contrato entre una operación o subprograma (proveedor) y quien la llama (cliente):

- Precondición:
  - Expresa las restricciones bajo las que la operación funcionará correctamente.
  - Un sistema correcto nunca realizará una operación en un estado que no cumpla la precondición de dicha operación.
  - El cuerpo de la operación no se ocupará de los estados iniciales que no cumplan su precondición, ni siquiera para comprobar su cumplimiento o no.
- Postcondición:
  - Expresa cuál es el estado resultante de realizar la operación, supuesto que en su inicio se cumplía la precondición.

Si el cliente se compromete a llamar a la operación cumpliendo la precondición, la operación se compromete a cumplir la postcondición.

- Precondición: obligación del cliente y beneficio del proveedor.
- Postcondición: beneficio del cliente y obligación del proveedor.

	Obligaciones	Beneficios
Cliente (el que llama a un subprograma)	Cumplir Pre	Resultados que cumplen Post
Proveedor (subprograma llamado)	Realizar la operación correctamente	Trabajo más simple (por suponer Pre)

2.2.3. Más ejemplos

Habitualmente a la precondición le llamaremos φ (que se lee 'fi') y a la postcondición le llamaremos ψ (que se lee 'psi').

Ejemplo 3:
En este caso tenemos la especificación pre-post correspondiente al programa que, dados dos enteros x e y tal que x es mayor o igual que y, obtiene en z el valor absoluto de la resta y – x:

{φ} ≡ {x ≥ y}
z : = x – y;
{ψ} ≡ {z = |y – x|}

Ejemplo 4:
A continuación podemos observar la especificación pre-post correspondiente al programa que, dados dos enteros x e y, obtiene en z el valor absoluto de la resta y – x:

```
{φ} ≡ {True}
if x >= y then z : = x – y;
else z : = y – x; end if;
{ψ} ≡ {z = |y – x|}
```

Para indicar que los datos de entrada no han de cumplir ninguna restricción inicial se pone 'True' como precondición.

Ejemplo 5:
La siguiente especificación pre-post corresponde al programa que intercambia los valores de dos variables x e y:

```
{φ} ≡ {x = a ∧ y = b}
x : = x + y;
y : = x – y;
x : = x – y;
{ψ} ≡ {x = b ∧ y = a}
```

No sabemos qué valores concretos tienen x e y al principio pero para poder expresar que al final esos valores estarán intercambiados hay que darles un nombre. En este caso se les ha llamado ***a*** y **b**.

2.3. El Lenguaje de la Lógica de Primer Orden

Tal como se ha visto en los ejemplos de la sección anterior, utilizaremos el lenguaje de la Lógica de Primer Orden para escribir la precondición y la postcondición que conforman la especificación pre-post de un programa.

A continuación se repasan los elementos que conforman dicho lenguaje.

2.3.1. Alfabeto

El alfabeto nos indica los símbolos que podemos utilizar al escribir las aserciones:

- Símbolos de constante:
  - Números: ..., -3, -2, -1, 0, 1, 2, 3, ...
  - Letras: 'a', 'b', 'c', ..., 'z', ...
  - Otros valores constantes (o nombres de constantes): *a*, b, c, ...
  - Valores booleanos: True, False
- Símbolos de variable: x, y, z, s, num, suma, veces, ...
- Símbolos de función (función aritmética): +, -, –, *, mod, div, ..., ∑, ∏, N, ..., (**Las funciones aritméticas** son funciones que devuelven un valor numérico como resultado).
- Símbolos de predicado: =, ≠, <, ≤, >, ≥, par, impar, primo, ... (**Los predicados** son funciones que devuelven un booleano como resultado).
- Operadores lógicos: ¬, ∧, ∨, →, ↔
- Cuantificadores lógicos: ∃, ∀
- Paréntesis: (,)

2.3.2. Sintaxis

La sintaxis nos indica qué reglas se han de tener en cuenta para escribir las fórmulas, es decir, nos precisa las normas a seguir para construir fórmulas válidas o aceptables:

2.3.2.1. Términos

Reciben el nombre de **término**:

- las constantes: 4, 6, -6, 'a', 'b', ..., *a*, b, ...
- las variables: x, y, z, r, suma, veces, ...
- las funciones aplicadas a términos. Si f es una función n-aria y t_1, t_2, ..., t_n son términos, entonces $f(t_1, t_2, ..., t_n)$ es un término.

  Por ejemplo, si max es la función que devuelve el mayor de dos números, max(5, 2) es un término, max(y, z + 1) es un término, etc.

2.3.2.2. Fórmulas

Son **fórmulas**:

- los átomos o predicados aplicados a términos. Si p es un predicado n-ario y t_1, t_2, ..., t_n son términos, entonces $p(t_1, t_2, ..., t_n)$ es una fórmula.

  Por ejemplo, si par es el predicado que dado un entero devuelve True si ese entero es par y False en caso contrario, par(5) es una fórmula, par(y + 2) es una fórmula, etc.

- las expresiones formadas al juntar fórmulas mediante los conectivos lógicos:
  - $\neg\varphi$ (**no** fi)
  - $\varphi \wedge \psi$ (fi **y** psi)
  - $\varphi \vee \psi$ (fi **o** psi)
  - $\varphi \rightarrow \psi$ (**si** fi **entonces** psi)
  - $\varphi \leftrightarrow \psi$ (fi **si y solamente si** psi)

  donde φ y ψ son fórmulas.

- también son **fórmulas** las expresiones formadas al aplicar cuantificadores a las fórmulas:
  - $\exists x(\varphi)$ Cuantificador existencial
    Significado: Existe al menos un x que cumple φ
  - $\forall x(\varphi)$ Cuantificador universal
    Significado: Para todo x se cumple φ

  donde φ es una fórmula.

2.3.3. Semántica

La semántica nos indica cuál es el valor de cada fórmula, es decir, dada una fórmula nos dice si es cierta o falsa.

2.3.3.1. Semántica de los operadores lógicos $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

φ	$\neg\varphi$
False	True
True	False

φ	ψ	$\varphi \wedge \psi$
False	False	False
False	True	False
True	False	False
True	True	True

φ	ψ	$\varphi \vee \psi$
False	False	False
False	True	True
True	False	True
True	True	True

φ	ψ	$\varphi \rightarrow \psi$
False	False	True
False	True	True
True	False	False
True	True	True

φ	ψ	$\varphi \leftrightarrow \psi$
False	False	True
False	True	False
True	False	False
True	True	True

$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

Equivalencias importantes:

$\varphi \wedge True \equiv \varphi$	$\varphi \vee True \equiv True$	$\varphi \rightarrow True \equiv True$	$\varphi \leftrightarrow True \equiv \varphi$
$\varphi \wedge False \equiv False$	$\varphi \vee False \equiv \varphi$	$\varphi \rightarrow False \equiv \neg\varphi$	$\varphi \leftrightarrow False \equiv \neg\varphi$
		$False \rightarrow \varphi \equiv True$	
		$True \rightarrow \varphi \equiv \varphi$	

2.3.3.2. Semántica del cuantificador existencial ∃

∃x(φ) significa que existe algún elemento (llamémosle x) que cumple lo que se dice en φ para x. Precisando más, ∃x(φ) indica que existe por lo menos un elemento que cumple φ. Pero no se sabe cuántos son esos elementos.

Las fórmulas existenciales suelen tener habitualmente el siguiente formato:

$$\mathbf{\exists x(D(x) \wedge P(x))}$$

donde D(x) es un dominio o intervalo (generalmente dentro de los números enteros) y P(x) una propiedad. Su significado será que existe al menos un elemento x que es del dominio (o intervalo) D y que cumple la propiedad P.

Ejemplo 6:
La siguiente fórmula existencial expresa que el vector A(1..n) de enteros contiene al menos un número negativo:

$$\exists i(\underbrace{1 \le i \le n}_{D(i)} \wedge \underbrace{A(i) < 0}_{P(i)})$$

La fórmula $\exists i(1 \le i \le n \wedge A(i) < 0)$ significa lo siguiente:

$$A(1) < 0 \vee A(2) < 0 \vee ... \vee A(n) < 0$$

Ejemplo 7:
La siguiente fórmula existencial expresa que x es una potencia entera de 2:

$$\exists k(\underbrace{k \ge 0}_{D(k)} \wedge \underbrace{x = 2^k}_{P(k)})$$

La fórmula $\exists k(k \ge 0 \wedge x = 2^k)$ significa lo siguiente:

$$x = 2^0 \vee x = 2^1 \vee ... \vee x = 2^{50} \vee ...$$

Ejemplo 8:
Cuando el **dominio** de una fórmula existencial es **vacío**, el valor de la fórmula es **False**, porque no existe ningún elemento que sea del dominio D y cumpla la propiedad P.

$$\exists z\ (\underbrace{1 \le z \le 0}_{D(z)} \wedge \underbrace{x = 2^z}_{P(z)})$$

No existe ningún entero que sea mayor o igual que 1 y menor o igual que 0.

2.3.3.3. Semántica del cuantificador universal ∀

∀x(φ) significa que todos los elementos (llamémosles x) cumplen lo que se dice en φ para x.

Las fórmulas universales suelen tener habitualmente el siguiente formato:

$$\forall x(D(x) \rightarrow P(x))$$

donde D(x) es un dominio o intervalo (dentro de los números enteros) y P(x) una propiedad. Su significado será que todo elemento x que es del dominio (o intervalo) D cumple la propiedad P. Dicho de otra forma, si un elemento está en D entonces cumple P.

Ejemplo 9:
La siguiente fórmula universal expresa que el vector A(1..n) de enteros contiene solo números positivos:

$$\forall i(\underbrace{1 \leq i \leq n}_{D(i)} \rightarrow \underbrace{A(i) > 0}_{P(i)})$$

La fórmula ∀i(1 ≤ i ≤ n → A(i) > 0) significa lo siguiente:

$$A(1) > 0 \wedge A(2) > 0 \wedge ... \wedge A(n) > 0$$

Ejemplo 10:
La siguiente fórmula universal expresa que x no aparece en el vector A(1..n) de enteros:

$$\forall i(\underbrace{1 \leq i \leq n}_{D(i)} \rightarrow \underbrace{A(i) \neq x}_{P(i)})$$

La fórmula ∀i(1 ≤ i ≤ n → A(i) ≠ x) significa lo siguiente:

$$A(1) \neq x \wedge A(2) \neq x \wedge ... \wedge A(n) \neq x$$

Ejemplo 11:
La siguiente fórmula universal expresa que x no aparece en las posiciones pares del vector A(1..n) de enteros:

$$\forall i(\underbrace{(1 \le i \le n \wedge par(i))}_{D(i)} \rightarrow \underbrace{A(i) \neq x}_{P(i)})$$

En este caso par(i) es un predicado que devuelve True para los números pares y False para los impares. La fórmula se leería de la siguiente forma: para todo i, si i está entre 1 y n e i es par, entonces A(i) no contiene el valor x.

Ejemplo 12:
Cuando el **dominio** de una fórmula universal es **vacío**, el valor de la fórmula es **True**, porque es cierto que todo elemento que esté en el intervalo cumple la propiedad P. Dicho de otra forma, es cierto que si un elemento está en D entonces ese elemento cumple P.

$$\forall i(\underbrace{1 < i \le 1}_{D(i)} \rightarrow \underbrace{A(i) > 0}_{P(i)})$$

Para que fuese False habría que encontrar un elemento i tal que i fuese mayor que 1 y a la vez menor o igual que 1 y que no cumpliese A(i) > 0. Pero no es posible encontrar ningún elemento que sea mayor que 1 y a la vez menor o igual que 1. Puesto que en ese dominio no hay ningún elemento que no cumpla A(i) > 0, el valor de la fórmula es True.

2.3.3.4. Semántica de la función Σ

Vamos a ver tres posibles maneras de utilizar la función Σ:

- $\sum_{i=m}^{n} f(i)$ Es la suma de los f(i) de aquellos valores i que estén comprendidos entre m y n. Si n es menor que m, entonces el resultado es 0.
  Por tanto, $\sum_{i=m}^{n} f(i) = f(m) + f(m+1) + ... + f(n)$

  Ejemplo 13:
  El siguiente sumatorio es la suma de los elementos del vector A comprendidos entre las posiciones 2 y 5:

$$\sum_{i=2}^{5} A(i) = A(2) + A(3) + A(4) + A(5)$$

Ejemplo 14:
En este ejemplo vemos que el sumatorio es 0 cuando el límite superior es menor que el inferior:

$$\sum_{k=2}^{7} k = 2+3+4+5+6+7 = 27 \qquad \sum_{k=7}^{2} k = 0$$

- $\sum_{i \in C} \mathrm{f}(i)$ Es la suma de los f(i) de aquellos valores i que estén en el conjunto C. Si C es vacío, entonces el resultado es 0.

Ejemplo 15:
Consideremos el conjunto C = {4, 8, 9}:

$$\sum_{i \in C} x^i = x^4 + x^8 + x^9$$

Ejemplo 16:
En este ejemplo vemos que el sumatorio es 0 cuando el conjunto considerado es vacío (∅):

$$\sum_{i \in \varnothing} x^i = 0$$

- $\sum_{P(i)} \mathrm{f}(i)$ Es la suma de los f(i) de aquellos valores i que cumplan la propiedad P. Si no hay ningún valor que cumpla la propiedad P, entonces el resultado es 0.

Ejemplo 17:
Consideremos la siguiente propiedad: **P(i) ≡ 1 ≤ i ≤ 10 ∧ par(i)**

$$\sum_{\mathrm{P(i)}} x^i = x^2 + x^4 + x^6 + x^8 + x^{10}$$

Los elementos que cumplen la propiedad P son 2, 4, 6, 8 y 10.

Ejemplo 18:
En este ejemplo vemos que cuando no hay ningún valor que cumpla la propiedad P, entonces el resultado es 0. Consideremos la siguiente propiedad: **P(i) ≡ i ≥ 10 ∧ (8 mod i = 0)**. Por tanto, estamos considerando aquellos números mayores o iguales a 10 de los cuales 8 es múltiplo pero no hay ningún número que cumpla esa propiedad

$$\sum_{\mathrm{P(i)}} x^i = 0$$

2.3.3.5. Semántica de la función Π

Vamos a ver tres posibles maneras de utilizar la función Π:

- $\prod_{i=m}^{n} \mathrm{f}(i)$ Es el producto de los f(i) de aquellos valores i que estén comprendidos entre m y n. Si n es menor que m, entonces el resultado es 1.

  Por tanto, $\prod_{i=m}^{n} \mathrm{f}(i) = \mathrm{f}(m) * \mathrm{f}(m+1) * ... * \mathrm{f}(n)$

  Ejemplo 19:
  La siguiente expresión es el producto de los elementos del vector A comprendidos entre las posiciones 2 y 5:

  $$\prod_{i=2}^{5} \mathrm{A}(i) = \mathrm{A}(2) * \mathrm{A}(3) * \mathrm{A}(4) * \mathrm{A}(5)$$

  Ejemplo 20:
  En este ejemplo vemos que el resultado es 1 cuando el límite superior es menor que el inferior:

  $$\prod_{k=2}^{7} k = 2 * 3 * 4 * 5 * 6 * 7 = 5040 \qquad \prod_{k=7}^{2} k = 1$$

- $\prod_{i \in C} \mathrm{f}(i)$ Es el producto de los f(i) de aquellos valores i que estén en el conjunto C. Si C es vacío, entonces el resultado es 1.

  Ejemplo 21:
  Consideremos el conjunto C = {4, 8, 9}:

  $$\prod_{i \in C} x^i = x^4 * x^8 * x^9$$

  Ejemplo 22:
  En este ejemplo vemos que el producto es 1 cuando el conjunto considerado es vacío (∅):

  $$\prod_{i \in \varnothing} x^i = 1$$

- $\prod_{P(i)} \mathrm{f}(i)$ Es el producto de los f(i) de aquellos valores i que cumplan la propiedad P. Si no hay ningún valor que cumpla la propiedad P, entonces el resultado es 1.

Ejemplo 23:
Consideremos la siguiente propiedad: **P(i) ≡ 1 ≤ i ≤ 10 ∧ par(i)**

$$\prod_{P(i)} x^i = x^2 * x^4 * x^6 * x^8 * x^{10}$$

Los elementos que cumplen la propiedad P son 2, 4, 6, 8 y 10.

Ejemplo 24:
En este ejemplo vemos que cuando no hay ningún valor que cumpla la propiedad P, entonces el resultado es 1. Consideremos la siguiente propiedad: **P(i) ≡ i ≥ 10 ∧ par(i) ∧ primo(i)**, donde primo es el predicado que, dado un entero, devuelve True si el número es primo y devuelve False en caso contrario. Por tanto, estamos considerando aquellos números mayores o iguales a 10 que son pares y primos pero no hay ningún número que cumpla esa propiedad

$$\prod_{P(i)} x^i = 1$$

2.3.3.6. Semántica de la función N

La función N se define de la siguiente forma:

$$Nx(D(x) \wedge P(x)) = card\{x \mid D(x) \wedge P(x)\} = \sum_{D(x) \wedge P(x)} 1$$

Por tanto, Nx(D(x) ∧ P(x)) nos dice cuántos elementos x hay que pertenecen al dominio D y cumplen la propiedad P. El dominio puede venir dado por un intervalo o un conjunto.

Ejemplo 25:
Ni(4 ≤ i ≤ 9 ∧ par(i)) = 3, porque entre 4 y 9 hay 3 elementos pares (4, 6 y 8).

Ejemplo 26:
Nx(x ∈ {3, 6, 14} ∧ par(x)) = 2, porque en el conjunto {3, 6, 14} hay dos pares (6 y 14).

Ejemplo 27:
Nx(4 ≤ x ≤ 0 ∧ par(x)) = 0, porque no hay ningún número que sea mayor o igual que 4 y menor o igual que 0.

Ejemplo 28:
Consideremos el siguiente vector A(1..8):

3	15	6	7	20	14	11	33
1	2	3	4	5	6	7	8

Nj(1 ≤ j ≤ 8 ∧ par(A(j))) = 3 porque hay tres elementos que son pares.

2.3.4. Variables libres y ligadas

En una fórmula las variables asociadas a los cuantificadores ∃ y ∀ y a las funciones Σ, ∏ y N son **variables ligadas** y el resto son **variables libres**.

Al escribir una fórmula que corresponde a la especificación de un programa o una fórmula que represente lo que expresa una sentencia

- conviene que las variables del programa o de la sentencia aparezcan siempre como libres
- y es necesario que las variables que se inventen para poder escribir la fórmula aparezcan como ligadas.

Dicho de otra forma, conviene que las variables ligadas sean nuevas, es decir, que no aparezcan en el programa o sentencia y es necesario que las variables libres sean variables que aparecen en el programa o sentencia.

Ejemplo 29:
Vamos a escribir la fórmula correspondiente a la sentencia o afirmación "Ningún número comprendido entre i y j (ambos inclusive) es múltiplo de x"

$$\forall k(i \leq k \leq j \rightarrow k \bmod x \neq 0)$$

En esa fórmula las variables i, j y x son libres, son justo las variables que aparecen en la sentencia. En cambio la variable k está ligada al cuantificador universal ∀ y no aparece en la sentencia, es una variable que se ha inventado para poder escribir la fórmula.

Ejemplo 30:
Ahora vamos a escribir la fórmula correspondiente a la sentencia o afirmación "En el vector de enteros A(1..n) el valor x aparece exactamente una vez":

$$Ni(1 \leq i \leq n \wedge A(i) = x) = 1$$

En esa fórmula las variables A, n y x son libres, son justo las variables que aparecen en la sentencia. En cambio la variable i está ligada a la función N y no aparece en la sentencia, es una variable que se ha inventado para poder escribir la fórmula.

La siguiente fórmula expresa lo mismo:

$$\exists i(1 \leq i \leq n \wedge x = A(i) \wedge \forall j((1 \leq j \leq n \wedge i \neq j) \rightarrow A(j) \neq x))$$

En este caso i y j serían variables ligadas. La variable i está ligada a ∃ y la variable j está ligada a ∀.

2.3.5. Definición de predicados nuevos

2.3.5.1. Definición de predicados mediante fórmulas

Los predicados se definen mediante fórmulas. Los argumentos o parámetros de los predicados serán las variables libres de la fórmula. Un predicado es como una función que devuelve un True o un False. A continuación se dan varios ejemplos en los que se definen predicados y se muestra qué valores (True o False) devolverían esos predicados para valores concretos.

Ejemplo 31:
Vamos a definir un predicado **todospositivos(A(1..n))** que exprese que todos los elementos de A(1..n) son mayores o iguales a 1:

$$\text{todospositivos}(A(1..n)) \equiv \forall i(1 \leq i \leq n \rightarrow A(i) \geq 1)$$

Casos concretos:
todospositivos((4, 10, 5, 8)) devolvería como resultado True.
todospositivos((4, -10, 5, -8, 9)) devolvería como resultado False.

Ejemplo 32:
Vamos a definir un predicado **todosmayores(A(1..n), x)** que exprese que todos los elementos de A(1..n) son mayores que x:

$$\text{todosmayores}(A(1..n), x) \equiv \forall i(1 \leq i \leq n \rightarrow A(i) > x)$$

Como se puede observar, los argumentos del predicado aparecen como variables libres en la fórmula.

Casos concretos:
todosmayores((4, 10, 5, 8), 2) devolvería como resultado True.
todosmayores((4, 10, 5, 8), 6) devolvería como resultado False.

Ejemplo 33:
Vamos a definir un predicado **algunomayor(A(1..n), x)** que exprese que en A(1..n) hay al menos un elemento que es mayor que x:

$$\text{algunomayor}(A(1..n), x) \equiv \exists i(1 \leq i \leq n \wedge A(i) > x)$$

Casos concretos:
algunomayor((4, 10, 5, 8), 6) devolvería como resultado True.
algunomayor((4, 10, 5, 8), 12) devolvería como resultado False.

Ejemplo 34:
A continuación se define un predicado **suma(A(1..n), s)** que expresa que la suma de los elementos de A(1..n) es s:

$$\text{suma}(A(1..n), s) \equiv \sum_{i=1}^{n} A(i) = s$$

Los argumentos del predicado aparecen como variables libres en la fórmula.

Casos concretos:
suma((4, 12, 5, 8), 29) devolvería como resultado True.
suma((4, 12, 5, 8), 20) devolvería como resultado False.

Ejemplo 35:
En este ejemplo se define un predicado **suma_mult(A(1..n), s, x)** que expresa que la suma de los elementos de A(1..n) que son múltiplos de x es s:

$$\text{suma\_mult}(A(1..n), s, x) \equiv \sum_{1 \le i \le n \wedge A(i) \bmod x = 0} A(i) = s$$

Casos concretos:
sumamult((4, 16, 5, 8), 28, 2) devolvería como resultado True.
sumamult((4, 16, 5, 8), 20, 2) devolvería como resultado False.
sumamult((4, 16, 5, 8), 0, 3) devolvería como resultado True.

Ejemplo 36:
Vamos a definir un predicado **doble(A(1..n), (c_1, …, c_n))** que exprese que cada elemento del vector A(1..n) es el doble del correspondiente elemento en el vector (c_1, …, c_n).

$$\text{doble}(A(1..n), (c_1, \ldots, c_n)) \equiv \forall i(1 \le i \le n \rightarrow A(i) = 2 * c_i)$$

En este ejemplo se puede apreciar que un vector puede ser representado de dos formas, dando su nombre como en el caso de A(1..n) o dando la lista de los elementos como en el caso del vector (c_1, …, c_n).

Casos concretos:
doble((8, 32, 10, 16), (4, 16, 5, 8)) devolvería como resultado True.
doble((8, 25, 10, 10), (4, 16, 5, 8)) devolvería como resultado False.
doble((4, 16, 5, 8) , (8, 32, 10, 16)) devolvería como resultado False porque los dobles han de estar en el primer vector y no en el segundo.

2.3.5.2. Predicado asociado a una fórmula

Dada una fórmula, el predicado correspondiente o asociado se obtiene inventando un nombre y poniendo como parámetros las variables libres de la fórmula.

Ejemplo 37:
Fórmula: $\exists k(k \geq 0 \wedge x = 2^k)$
Predicado asociado: **potdos(x)**

La fórmula expresa que x es una potencia entera de 2. Es importante recordar que solo manejamos números enteros y que, por tanto, los valores k considerados son enteros.

Casos concretos:
potdos(8) devolvería como resultado True porque $8 = 2^3$.
potdos(7) devolvería como resultado False porque 7 no es una potencia entera de 2.

Ejemplo 38:
Fórmula: $Ni(1 \leq i \leq n \wedge A(i) = x) = 1$
Predicado asociado: **unavez(A(1..n), x)**

La fórmula expresa que x aparece una única vez en A(1..n).

Casos concretos:
unavez((4, 10, 8, 6), 8) devolvería como resultado True.
unavez((4, 10, 8, 6), 7) devolvería como resultado False.
unavez((4, 10, 8, 6, 8), 8) devolvería como resultado False porque 8 aparece más de una vez.

Ejemplo 39:
Fórmula: $(Ni(1 \leq i \leq n \wedge A(i) = x) = 1) \wedge (Ni(1 \leq i \leq n \wedge A(i) = w) = 2)$
Predicado asociado: **una_dos(A(1..n), x, w)**

La fórmula expresa que x aparece una única vez en A(1..n) y que w aparece 2 veces.

Casos concretos:
una_dos((4, 10, 8, 6, 8), 6, 8) devolvería como resultado True porque el 6 aparece una vez y el 8 dos veces.
una_dos((8, 10, 8, 6, 8), 6, 8) devolvería como resultado False porque aunque el 6 aparezca una vez, el 8 no aparece justo dos veces.

2.3.5.3. Definición de nuevos predicados utilizando otros predicados

También es posible definir predicados nuevos utilizando predicados definidos previamente.

Ejemplo 40:
Definir un predicado posdif(A(1..n)) que exprese que todos los elementos de A(1..n) son positivos y diferentes. Utilizar los predicados **todospositivos** y **unavez**:

$$posdif(A(1..n)) \equiv todospositivos(A(1..n)) \wedge \forall j(1 \leq j \leq n \rightarrow unavez(A(1..n), A(j)))$$

Casos concretos:

Consideremos los vectores B(1..4) = (4, 16, 5, 8) y C(1..5) = (4, 16, 8, 5, 8).

posdif(B(1..4)) devolvería True.
posdif(C(1..5)) devolvería False porque el 8 se repite.
posdif((9, 5, -2, 7)) devolvería False porque hay un negativo.

2.4. Ejemplos de especificación de programas

2.4.1. Programa que calcula en s la suma de los elementos del vector A(1..n)

Se va a dar la especificación pre-post correspondiente al siguiente programa que dado un vector A(1..n) con n mayor o igual que 1, calcula en s la suma de los elementos de A(1..n).

```
{φ} ≡ ???
i : = 1;
s : = 0;
while i ≠ n + 1 loop
        s : = s + A(i);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Especificación pre-post:

{φ} ≡ {n ≥ 1}

{ψ} ≡ $\{ s = \sum_{k=1}^{n} A(k) \}$

Es importante observar que en la especificación pre-post no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post.

2.4.2. Programa que genera una copia del vector A(1..n) en B(1..n) (Versión 1)

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, obtiene en B(1..n) una copia de A(1..n).

```
{φ} ≡ ???
i : = 0;
while i < n loop
        i : = i + 1;
        B(i) : = A(i);
end loop;
{ψ} ≡ ???
```

Especificación pre-post:

{φ} ≡ {n ≥ 1}

{ψ} ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k))}

Aquí también es importante observar que en la especificación pre-post no aparece la variable índice i.

2.4.3. Programa que genera una copia del vector A(1..n) en B(1..n) (Versión 2)

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, obtiene en B(1..n) una copia de A(1..n).

```
{φ} ≡ ???
i : = n + 1;
while i > 1 loop
        i : = i – 1;
        B(i) : = A(i);
end loop;
{ψ} ≡ ???
```

Especificación pre-post:
{φ} ≡ {n ≥ 1}
{ψ} ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k))}

Aunque el programa sea distinto al del ejemplo anterior, como los dos programas parten de la misma situación y calculan el mismo resultado, la especificación pre-post coincide.

2.4.4. Programa que genera una copia del vector A(1..n) en B(1..n) (Versión 3)

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, obtiene en B(1..n) una copia de A(1..n).

```
{φ} ≡ ???
i : = 1;
while i ≤ n loop
        B(i) : = A(i);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Especificación pre-post:
{φ} ≡ {n ≥ 1}
{ψ} ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k))}

Mediante este ejemplo y los dos anteriores, donde tenemos programas que copian los elementos del vector A(1..n) en B(1..n), se puede ver que la especificación pre-post es independiente del **cómo** se hacen los cálculos ya que la especificación indica **qué** resultado obtiene el programa y no cómo lo obtiene.

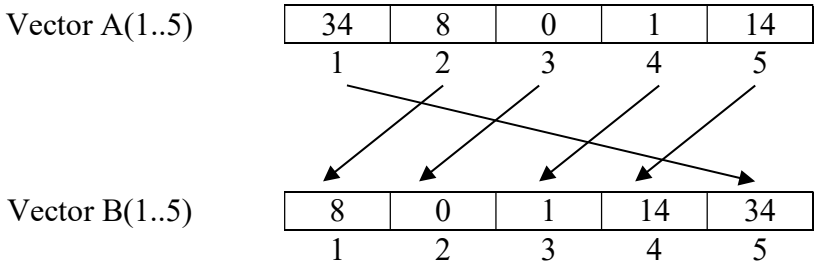
2.4.5. Programa que guarda en B(1..n) los elementos de A(1..n) pero rotándolos hacia la izquierda

En este caso se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, guarda los elementos de A(1..n) en B(1..n) pero rotándolos hacia la izquierda.

```
{φ} ≡ ???
B(n) := A(1);
i : = 1;
while i < n loop
        B(i) : = A(i + 1);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Ejemplo

En este ejemplo se muestra gráficamente qué hace el programa en un caso concreto:



Especificación pre-post:

$\{\varphi\} \equiv \{n \geq 1\}$

$\{\psi\} \equiv \{(B(n) = A(1)) \wedge \forall k(1 \leq k \leq n - 1 \rightarrow B(k) = A(k + 1))\}$

2.4.6. Programa que rota los elementos de A(1..n) hacia la izquierda en el propio vector A(1..n)

En este caso se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, rota los elementos de A(1..n) hacia la izquierda en el propio vector.

```
{φ} ≡ ???
aux := A(1);
i : = 1;
while i < n loop
        A(i) : = A(i + 1);
        i : = i + 1;
end loop;
A(n) : = aux;
{ψ} ≡ ???
```

Primero se va a dar una **especificación incorrecta**:

{φ} ≡ {n ≥ 1}

{ψ} ≡ {(A(n) = A(1)) ∧ ∀k(1 ≤ k ≤ n – 1 → A(k) = A(k + 1))}

El siguiente caso concreto muestra que la especificación no es correcta:

Vector A(1..5) antes de ejecutar el programa	34	8	0	1	14
	1	2	3	4	5
Vector A(1..5) después de ejecutar el programa	8	0	1	14	34
	1	2	3	4	5

No es verdad que A(5) = A(1) porque al final A(1) es 8 y A(5) es 34 y tampoco es cierto que para cada k del intervalo [1, n – 1] se cumpla A(k) = A(k + 1). Por ejemplo A(3) es 1 y A(4) es 14.

En la especificación se pretende decir que en A(n) se tendrá al final lo que al principio se tenía en A(1), y que en el resto de posiciones k del intervalo [1, n – 1] se tendrá lo que al principio se tenía en la posición k + 1. Pero como el vector A(1..n) es modificado durante el proceso, para poder referirnos a los valores iniciales de A(1..n) es necesario darles un nombre en la precondición. Llamaremos a_k al elemento que ocupa la posición k antes de ejecutar el programa:

Especificación correcta:

{φ} ≡ {n ≥ 1 ∧ ∀k(1 ≤ k ≤ n → A(k) = a_k)}

{ψ} ≡ {(A(n) = a_1) ∧ ∀k(1 ≤ k ≤ n – 1 → A(k) = a_{k+1})}

Es importante observar que en la especificación pre-post no aparecen la variable aux y la variable índice i, ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables intermedias no se mencionan porque no son ni datos de entrada ni resultados.

2.4.7. Programa que cuenta en c el número de ceros de A(1..n) (Versión 1)

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, obtiene en c el número de ceros de A(1..n).

```
{φ} ≡ ???
i : = 1;
c : = 0;
while i <= n loop
        if A(i) = 0 then c : = c + 1; end if;
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Especificación pre-post:
{φ} ≡ {n ≥ 1}
{ψ} ≡ {c = Nk(1 ≤ k ≤ n ∧ A(k) = 0)}

2.4.8. Programa que cuenta en c el número de ceros de A(1..n) (Versión 2)

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, obtiene en c el número de ceros de A(1..n).

```
{φ} ≡ ???
i : = 0;
c : = 0;
while i < n loop
        i : = i + 1;
        if A(i) = 0 then c : = c + 1; end if;
end loop;
{ψ} ≡ ???
```

Especificación pre-post:
{φ} ≡ {n ≥ 1}
{ψ} ≡ {c = Nk(1 ≤ k ≤ n ∧ A(k) = 0)}

Otra vez vemos con este ejemplo y el anterior que aunque los programas sean distintos, si hacen lo mismo la especificación pre-post también coincidirá.

2.4.9. Programa que decide en b si A(1..n) contiene algún 0

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, decide en la variable booleana b si A(1..n) contiene o no algún 0:

```
{φ} ≡ ???
i : = 1;
b : = False;
while i <= n and not b loop
        b : = (A(i) = 0);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Especificación pre-post:

$\{\varphi\} \equiv \{n \geq 1\}$

$\{\psi\} \equiv \{b \leftrightarrow \exists k(1 \leq k \leq n \wedge A(k) = 0)\}$

2.4.10. Programa que decide en b si A(1..n) contiene más pares que impares

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, decide en la variable booleana b si A(1..n) contiene más pares que impares:

```
{φ} ≡ ???
i : = 1;
p: = 0; imp := 0;
while i <= n loop
        if A(i) mod 2 = 0 then p : = p + 1;
        else imp : = imp + 1; end if;
        i : = i + 1;
end loop;
b : = (p > imp);
{ψ} ≡ ???
```

Especificación pre-post:

$\{\varphi\} \equiv \{n \geq 1\}$

$\{\psi\} \equiv \{b \leftrightarrow Nk(1 \leq k \leq n \wedge par(A(k))) > Nk(1 \leq k \leq n \wedge \neg par(A(k)))\}$

Es importante observar que en la especificación pre-post no aparecen las variables del programa que han sido utilizados para obtener resultados parciales pero que no son resultados finales. Tal es el caso de los contadores p e imp y la variable índice i.

El dato de entrada es A(1..n) y el resultado es b. Por tanto, las otras tres variables no aparecen en la especificación pre-post ya que la especificación pre-post sirve para describir los datos de entrada y el resultado.

2.4.11. Programa que decide en pos la posición de la primera aparición de x en A(1..n) sabiendo que x aparece en A(1..n)

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1 y que contiene al menos una aparición de x, decide en la variable pos la posición de la primera aparición de x en A(1..n):

```
{φ} ≡ ???
i : = 1;
seguir: = (A(1) ≠ x);
while seguir loop
        seguir: = (A(i + 1) ≠ x);
        i : = i + 1;
end loop;
pos : = i;
{ψ} ≡ ???
```

Especificación pre-post:

$\{\varphi\} \equiv \{n \geq 1 \wedge \exists k(1 \leq k \leq n \wedge A(k) = x)\}$

$\{\psi\} \equiv \{1 \leq pos \leq n \wedge A(pos) = x \wedge \forall k(1 \leq k \leq pos - 1 \rightarrow A(k) \neq x)\}$

Mediante la precondición φ se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y en el que el valor x aparece por lo menos una vez. Mediante la postcondición ψ se ha indicado que al final la variable pos contiene la posición de la primera aparición de x.

2.4.12. Programa que si x aparece en A(1..n) decide en pos la posición de la primera aparición de x y si x no aparece devuelve el valor n + 1 en pos

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, decide en la variable pos la posición de la primera aparición de x en A(1..n). Si x no está en A(1..n), pos terminará con valor n + 1:

```
{φ} ≡ ???
i : = 2;
seguir: = (A(1) ≠ x);
while i ≤ n ∧ seguir loop
        seguir: = (A(i) ≠ x);
        i : = i + 1;
end loop;
if seguir then pos : = i;
else pos : = i – 1;
end if;
{ψ} ≡ ???
```

Se utilizará el predicado esta(z, C(1..r)) definido de la siguiente forma:

$$\exists k(1 \leq k \leq r \wedge C(k) = z)$$

Especificación pre-post:

{φ} ≡ {n ≥ 1}

{ψ} ≡ {1 ≤ pos ≤ n + 1 ∧ ¬esta(x, A(1..pos – 1)) ∧
esta(x, A(1..n)) → (1 ≤ pos ≤ n ∧ A(pos) = x) ∧
¬esta(x, A(1..n)) → pos = n + 1}

Mediante la precondición φ se ha expresado que el programa parte de un vector que contiene por lo menos un elemento. Mediante la postcondición ψ se ha indicado que al final si x aparece en A(1..n), la variable pos contiene la posición de la primera aparición de x y si x no aparece en A(1..n), pos valdrá n + 1.

2.5. Documentación de programas: idea a seguir

En los ejercicios de documentación de programas se nos dará un programa y habrá que dar las aserciones que se indican.

- Primero habrá que formular la precondición y la postcondición.

- En cuanto a las aserciones intermedias que aparecen en programas no iterativos la idea es partir de la precondición e ir realizando aquellas modificaciones que se originan debido a la ejecución de instrucciones y evaluación de condiciones.

- En los programas iterativos, después de formular la precondición y la postcondición, habrá que dar el invariante. En el caso de las aserciones intermedias que aparecen dentro de las instrucciones iterativas, habrá que partir del invariante y habrá que ir realizando aquellas modificaciones que se originan debido a la ejecución de instrucciones y evaluación de condiciones. Por tanto, hay que pensar cómo afecta la ejecución de las instrucciones al invariante en los puntos señalados.

- En los programas iterativos se pedirá también la expresión cota que será una expresión que indique en cada momento cuántas vueltas de while quedan por dar como máximo. La expresión cota a veces indica el número exacto de vueltas que quedan por dar y otras veces indica un tope máximo (como mucho quedan tantas vueltas de while). La expresión cota tiene sentido en el punto donde se cumple el invariante, justo antes de evaluar la condición del while.

Normalmente primero se pedirá definir algunos predicados que luego habrá que utilizar al formular las aserciones indicadas en el programa. El uso de predicados hace que las fórmulas sean más cortas y más entendibles (algo así como las funciones y los procedimientos en programación).

A continuación se añaden varios ejemplos en los que se refleja cómo documentar programas.

2.6. Ejemplos de documentación de programas

2.6.1. Programa que guarda en B(1..n) los valores del vector A(1..n) multiplicados por x (Versión 1)

Se va a documentar, con las aserciones que se indican, el siguiente programa que, dado un vector no vacío A(1..n) de enteros, guarda en B(1..n) los valores del vector A(1..n) multiplicados por x, recorriendo los vectores de izquierda a derecha.

```
(1) {Precondición}
i : = 1;
(2) {Aserción intermedia}
while (3) {Invariante} i ≠ n + 1 loop
      (4) {Aserción intermedia}
      B(i) : =A(i) * x;
      (5) {Aserción intermedia}
      i : = i + 1;
      (6) {Aserción intermedia}
end loop;
(7) {Postcondición}
(8) {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4), (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E:

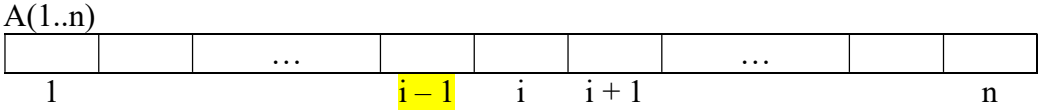
- (1) ≡ {n ≥ 1}
  (7) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k) * x)}

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final en cada posición de B se tendrá lo mismo que en A pero multiplicado por x.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 1:

  (2) ≡ {n ≥ 1 ∧ i = 1}

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 (la última vez, justo antes de terminar el while por no cumplirse la condición $i \neq n + 1$). Por otra parte, como una vez entrado en el while lo primero que se hace es guardar en la posición i de B el valor correspondiente, en el punto (3) se tiene que todavía no se ha guardado el elemento de la posición i del vector A(1..n) multiplicado por x en la posición i del vector B(1..n).

$$(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$$

- A continuación, a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y, por tanto, sabemos que i no es igual a n + 1.

  $$(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que en B(i) se ha guardado A(i) * x, pero el valor de i no se ha incrementado. Por tanto i no cambia pero sí se avanza en el número de posiciones calculadas, que ahora llega hasta i.

  $$(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que ahora ya se ha incrementado el valor de la variable i. Las posiciones calculadas vuelven a ser desde la 1 hasta la i – 1. El valor de i es por lo menos 2 y puede que sea n + 1.

  $$(6) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n + 1 – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.2. Programa que guarda en B(1..n) los valores del vector A(1..n) multiplicados por x (Versión 2)

Se va a documentar, con las aserciones que se indican, el siguiente programa que, dado un vector no vacío A(1..n) de enteros, guarda en B(1..n) los valores de A(1..n) multiplicados por x, recorriendo los vectores de izquierda a derecha. El programa es ligeramente distinto al presentado en el apartado 2.6.1 y se pretende ver qué cambios se producen en la documentación al tener un programa que, aunque obtenga el mismo resultado, calcula ese resultado de manera ligeramente distinta.

```
(1) {Precondición}
i : = 1;
(2) {Aserción intermedia}
while (3) {Invariante} i ≠ n + 1 loop
      (4) {Aserción intermedia}
      i : = i + 1;
      (5) {Aserción intermedia}
      B(i – 1) : =A(i – 1) * x;
      (6) {Aserción intermedia}
end loop;
(7) {Postcondición}
(8) {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4), (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k) * x)\}$

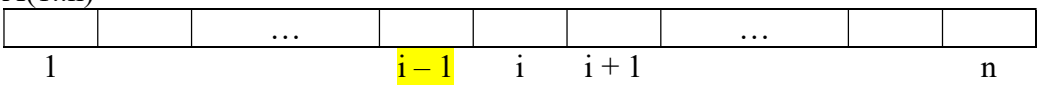
  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final en cada posición de B se tendrá lo mismo que en A pero multiplicado por x. Estas dos fórmulas son las mismas fórmulas dadas en los puntos (1) y (7) del apartado 2.6.1 porque los dos programas parten de la misma situación y devuelven el mismo resultado aunque calculen el resultado de manera distinta.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 1:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.

  A(1..n)



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 (la última vez, justo antes de terminar el while por no cumplirse la condición $i \neq n + 1$). Por otra parte, como una vez entrado en el while se incrementa el valor de i y se hace el cálculo en la posición anterior, en el punto (3) se tiene que todavía no se ha guardado el elemento de la posición i del vector A(1..n) multiplicado por x en la posición i del vector B(1..n).

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- A continuación a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i no es igual a n + 1.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de i se ha incrementado en 1 pero todavía no se han hecho los cálculos ni para la nueva posición i ni para la anterior. Por tanto, al crecer i podemos asegurar que ya no es 1 y que podría ser n + 1 y que las posiciones calculadas son las que van desde la posición 1 hasta la i – 2.

  $(5) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 2 \rightarrow B(k) = A(k) * x)\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que en B(i – 1) se ha guardado A(i – 1) * x. Por tanto, las posiciones calculadas son las que van desde la posición 1 hasta la i – 1.

  $(6) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n + 1 – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i

sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.3. Programa que guarda en B(1..n) los valores del vector A(1..n) multiplicados por x (Versión 3)

Se va a documentar, con las aserciones que se indican, el siguiente programa que, dado un vector no vacío A(1..n) de enteros, guarda en B(1..n) los valores de A(1..n) multiplicados por x, recorriendo los vectores de izquierda a derecha. El programa es ligeramente distinto a los presentados en los apartados 2.6.1 y 2.6.2 y se pretende ver qué cambios se producen en la documentación al tener un programa que, aunque obtenga el mismo resultado, calcula ese resultado de manera ligeramente distinta.

```
(1) {Precondición}
i : = 0;
(2) {Aserción intermedia}
while (3) {Invariante} i ≠ n loop
      (4) {Aserción intermedia}
      i : = i + 1;
      (5) {Aserción intermedia}
      B(i) : =A(i) * x;
      (6) {Aserción intermedia}
end loop;
(7) {Postcondición}
(8) {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4), (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- (1) ≡ {n ≥ 1}
  (7) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k) * x)}

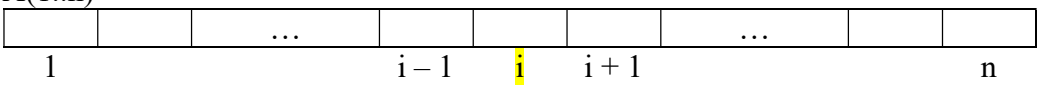
  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final en cada posición de B se tendrá lo mismo que en A pero multiplicado por x. Estas dos fórmulas son las mismas fórmulas dadas en los puntos (1) y (7) de los apartado 2.6.1 y 2.6.2 porque los tres programas parten de la misma situación y devuelven el mismo resultado aunque calculen el resultado de manera distinta.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 0. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 0:

  (2) ≡ {n ≥ 1 ∧ i = 0}

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.

  A(1..n)



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 0 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición $i \neq n$). Por otra parte, como una vez entrado en el while lo primero que se hace es incrementar el valor de i, en el punto (3) se tiene que ya se ha guardado el elemento de la posición i del vector A(1..n) multiplicado por x en la posición i del vector B(1..n).

  $(3) \equiv \{(0 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- A continuación a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i no es igual a n.

  $(4) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de i se ha incrementado en 1 pero todavía no se ha guardado x multiplicado por el elemento de la nueva posición i de A(1..n) en B(1..n). Por tanto, al crecer i podemos asegurar que ya no es 0 y que podría ser n y que las posiciones calculadas son las que van desde la posición 1 hasta la i – 1.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que en B(i) se ha guardado A(i) * x. Por tanto, las posiciones calculadas son las que van desde la posición 1 hasta la i.

  $(6) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en

el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.4. Programa que guarda en B(1..n) los valores del vector A(1..n) multiplicados por x (Versión 4)

Se va a documentar, con las aserciones que se indican, el siguiente programa que, dado un vector no vacío A(1..n) de enteros, guarda en B(1..n) los valores de A(1..n) multiplicados por x, recorriendo los vectores de izquierda a derecha. El programa es ligeramente distinto a los presentados en los apartados 2.6.1, 2.6.2 y 2.6.3 y se pretende ver qué cambios se producen en la documentación al tener un programa que, aunque obtenga el mismo resultado, calcula ese resultado de manera ligeramente distinta

```
(1) {Precondición}
i : = 0;
(2) {Aserción intermedia}
while (3) {Invariante} i ≠ n loop
      (4) {Aserción intermedia}
      B(i + 1) : =A(i + 1) * x;
      (5) {Aserción intermedia}
      i : = i + 1;
      (6) {Aserción intermedia}
end loop;
(7) {Postcondición}
(8) Expresión cota E.
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4) y (5) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- (1) ≡ {n ≥ 1}
  (7) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k) * x)}

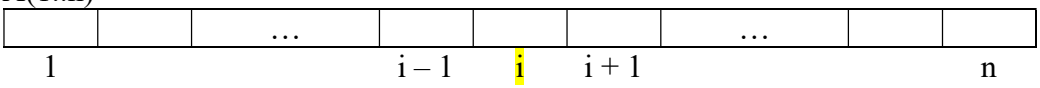
  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final en cada posición de B se tendrá lo mismo que en A pero multiplicado por x. Estas dos fórmulas son las mismas fórmulas dadas en los puntos (1) y (7) de los apartado 2.6.1, 2.6.2 y 2.6.3 porque los cuatro programas parten de la misma situación y devuelven el mismo resultado aunque calculen el resultado de manera distinta.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 0. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 0:

  (2) ≡ {n ≥ 1 ∧ i = 0}

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.

  A(1..n)



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 0 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición $i \neq n$). Por otra parte, como una vez entrado en el while lo primero que se hace es guardar en la posición i + 1 de B el valor correspondiente, en el punto (3) se tiene que ya se ha guardado el elemento de la posición i del vector A(1..n) multiplicado por x en la posición i del vector B(1..n).

  $(3) \equiv \{(0 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- A continuación a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y, por tanto, sabemos que i no es igual a n.

  $(4) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que en B(i + 1) se ha guardado A(i + 1) * x, pero el valor de i no se ha incrementado. Por tanto, i no cambia pero sí se avanza en el número de posiciones calculadas, que ahora llega hasta i + 1.

  $(5) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i + 1 \rightarrow B(k) = A(k) * x)\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que ahora ya se ha incrementado el valor de la variable i. Las posiciones calculadas van desde la 1 hasta la i. Ahora la variable i tendrá un valor comprendido entre 1 y n.

  $(6) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en

el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.5. Programa que multiplica los valores del vector A(1..n) por x modificando A(1..n)

Se va a documentar, con las aserciones que se indican, el siguiente programa que, dado un vector no vacío A(1..n) de enteros, modifica los elementos de A(1..n) multiplicándolos por x, recorriendo el vector de izquierda a derecha.

```
(1) {Precondición}
i : = 0;
(2) {Aserción intermedia}
while (3) {Invariante} i ≠ n loop
      (4) {Aserción intermedia}
      i : = i + 1;
      (5) {Aserción intermedia}
      A(i) : =A(i) * x;
      (6) {Aserción intermedia}
end loop;
(7) {Postcondición}
(8) Expresión cota E.
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4), (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- Como los valores iniciales de A(1..n) van a ser modificados y los nuevos valores van a depender de los iniciales, en la precondición hay que dar un nombre a los valores iniciales (habitualmente esto se hace utilizando minúsculas)
  $(1) \equiv \{n \geq 1 \wedge \forall k(1 \leq k \leq n \rightarrow A(k) = a_k)\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow A(k) = a_k * x)\}$

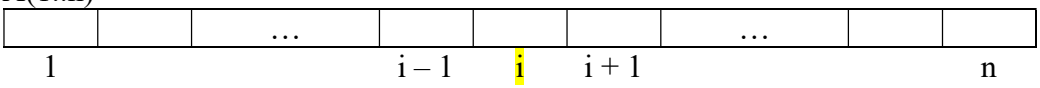
  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y que en cada posición k de A tenemos el valor inicial a_k. Mediante la postcondición (7) se ha indicado que al final en cada posición k de A se tendrá el valor inicial a_k multiplicado por x.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 0. En el punto (2) se cumple todo lo del punto (1) y además que i vale 0:

  $(2) \equiv \{n \geq 1 \wedge \forall k(1 \leq k \leq n \rightarrow A(k) = a_k) \wedge i = 0\}$

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.

  A(1..n)



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 0 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición $i \neq n$). Por otra parte, como una vez entrado en el while lo primero que se hace es incrementar el valor de i, en el punto (3) se tiene que ya se ha guardado el elemento de la posición i del vector A(1..n) multiplicado por x.

  $(3) \equiv \{(0 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow A(k) = a_k * x)\}$

- A continuación a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y, por tanto, sabemos que i no es igual a n.

  $(4) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i \rightarrow A(k) = a_k * x)\}$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de i se ha incrementado en 1 pero todavía no se ha guardado x multiplicado por el elemento de la nueva posición i de A(1..n). Por tanto, al crecer i podemos asegurar que ya no es 0 y que podría ser n y que las posiciones calculadas son las que van desde la posición 1 hasta la i – 1.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = a_k * x)\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que en A(i) se ha guardado $a_i * x$. Por tanto, las posiciones calculadas son las que van desde la posición 1 hasta la i.

  $(6) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow A(k) = a_k * x)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en

el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.6. Programa que calcula en s la suma de los elementos del vector A(1..n) (Versión 1)

Se va a documentar, con las aserciones que se indican, el siguiente programa que dado un vector A(1..n) con n mayor o igual que 1, calcula en s la suma de los elementos de A(1..n).

```
(1) ≡ {Precondición}
i : = 0;
(2) ≡ {Aserción intermedia}
s : = 0;
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i < n loop
        (5) ≡ {Aserción intermedia}
        i : = i + 1;
        (6) ≡ {Aserción intermedia}
        s : = s + A(i);
        (7) ≡ {Aserción intermedia}
end loop;
(8) ≡ {Postcondición}
(9) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá las aserciones intermedias (2) y (3) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- (1) ≡ {n ≥ 1}
  (8) ≡ $\{ s = \sum_{k=1}^{n} A(k) \}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (6) se ha indicado que al final se tiene en s la suma de los elementos de A(1..n).

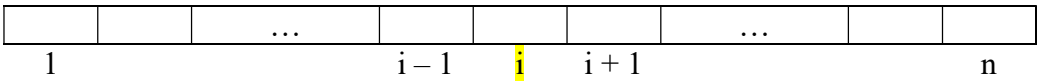
- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 0. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 0:

  (2) ≡ {n ≥ 1 ∧ i = 0}

- A partir de la aserción (2) se obtiene la aserción intermedia (3) teniendo en cuenta que a s se le ha asignado el valor 0. En el punto (3) sabemos que n es mayor o igual que 1, que i vale 0 y que s vale 0:

  (3) ≡ {n ≥ 1 ∧ i = 0 ∧ s = 0}

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 0 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición i < n). Por otra parte, como una vez entrado en el while lo primero que se hace es incrementar el valor de i, en el punto (4) se tiene que ya se ha sumado a s el elemento de la posición i del vector A(1..n).

$$(4) \equiv \{(0 \leq i \leq n) \wedge s = \sum_{k=1}^{i} A(k)\}$$

- A continuación a partir del invariante se obtendrá la aserción intermedia (5). En el punto (5) lo que ha cambiado con respecto al invariante, es que se ha entrado en el while, es decir, se ha cumplido la condición del while y, por tanto, sabemos que i es menor que n.

$$(5) \equiv \{(0 \leq i \leq n-1) \wedge s = \sum_{k=1}^{i} A(k)\}$$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de i se ha incrementado en 1 pero todavía no se ha sumado el elemento de la nueva posición i. Por tanto, al crecer i podemos asegurar que ya no es 0 y que podría ser n y que los elementos sumados son los que van desde la posición 1 hasta la i – 1.

$$(6) \equiv \{(1 \leq i \leq n) \wedge s = \sum_{k=1}^{i-1} A(k)\}$$

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que se ha sumado el elemento de la nueva posición i a s. Por tanto, tenemos que los elementos sumados son los que van desde la posición 1 hasta la i.

$$(7) \equiv \{(1 \leq i \leq n) \wedge s = \sum_{k=1}^{i} A(k)\}$$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.7. Programa que calcula en s la suma de los elementos del vector A(1..n) (Versión 2)

Se va a documentar, con las aserciones que se indican, el siguiente programa que dado un vector A(1..n) con n mayor o igual que 1, calcula en s la suma de los elementos de A(1..n).

```
(1) ≡ {Precondición}
i : = 1;
(2) ≡ {Aserción intermedia}
s : = 0;
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i ≤ n loop
        (5) ≡ {Aserción intermedia}
        s : = s + A(i);
        (6) ≡ {Aserción intermedia}
        i : = i + 1;
        (7) ≡ {Aserción intermedia}
end loop;
(8) ≡ {Postcondición}
(9) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá las aserciones intermedias (2) y (3) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E. La precondición y la postcondición coinciden con las del ejemplo anterior porque ambos programas parten de la misma situación y calculan el mismo resultado pero el invariante y el resto de las aserciones cambian ligeramente porque los dos programas calculan el resultado de manera diferente.

- $(1) \equiv \{n \geq 1\}$

  $(8) \equiv \{ s = \sum_{k=1}^{n} A(k) \}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (6) se ha indicado que al final se tiene en s la suma de los elementos de A(1..n).

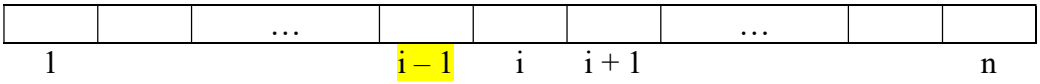
- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 1:

  (2) ≡ {n ≥ 1 ∧ i = 1}

- A partir de la aserción (2) se obtiene la aserción intermedia (3) teniendo en cuenta que a s se le ha asignado el valor 0. En el punto (3) sabemos que n es mayor o igual que 1, que i vale 1 y que s vale 0:

  (3) ≡ {n ≥ 1 ∧ i = 1 ∧ s = 0}

- A continuación, a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 (la última vez, justo antes de terminar el while por no cumplirse la condición i ≤ n). Por otra parte como una vez entrado en el while lo primero que se hace es sumar a s el valor contenido en la posición i del vector A(1..n), en el punto (4) se tiene que ya se han sumado en s los elementos que van desde la posición 1 hasta la posición i – 1 del vector A(1..n).

  $$(4) \equiv \{(1 \leq i \leq n+1) \wedge s = \sum_{k=1}^{i-1} A(k)\}$$

- A continuación a partir del invariante se obtendrá la aserción intermedia (5). En el punto (5) lo que ha cambiado con respecto al invariante, es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor que n + 1.

  $$(5) \equiv \{(1 \leq i \leq n) \wedge s = \sum_{k=1}^{i-1} A(k)\}$$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el elemento de la posición i se ha sumado a s pero todavía no se ha incrementado el valor de i. Por tanto los elementos sumados son los que van desde la posición 1 hasta la i.

  $$(6) \equiv \{(1 \leq i \leq n) \wedge s = \sum_{k=1}^{i} A(k)\}$$

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que se ha incrementado el valor de i. Por tanto, tenemos que los elementos sumados son los que van desde la posición 1 hasta la i – 1 y también tenemos que los límites del intervalo de i crecen en uno.

$$(7) \equiv \{(2 \leq i \leq n+1) \wedge s = \sum_{k=1}^{i-1} A(k)\}$$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n + 1 – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.8. Programa que genera una copia del vector A(1..n) en B(1..n) recorriendo el vector de izquierda a derecha (Versión 1)

Se va a documentar, con las aserciones que se indican, el siguiente programa que dado un vector A(1..n) con n mayor o igual que 1, obtiene en B(1..n) una copia de A(1..n) recorriendo los vectores de izquierda a derecha.

```
(1) ≡ {Precondición}
i : = 0;
(2) ≡ {Aserción intermedia}
while (3) ≡ {Invariante} i < n loop
        (4) ≡ {Aserción intermedia}
        i : = i + 1;
        (5) ≡ {Aserción intermedia}
        B(i) : = A(i);
        (6) ≡ {Aserción intermedia}
end loop;
(7) ≡ {Postcondición}
(8) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4) y (5) se formularán partiendo del invariante. Por último se dará la expresión-cota E:

- (1) ≡ {n ≥ 1}

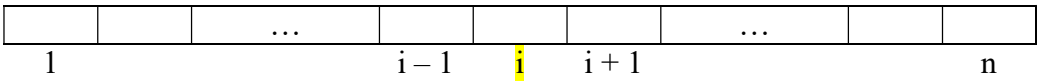
(7) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k))}

Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final los vectores A(1..n) y B(1..n) contendrán los mismos elementos.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 0. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 0:

  (2) ≡ {n ≥ 1 ∧ i = 0}

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 0 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición i < n). Por otra parte como una vez entrado en el while lo primero que se hace es incrementar el valor de i, en el punto (3) se tiene que ya se han copiado el elemento de la posición i del vector A(1..n) a la posición i del vector B(1..n).

  (3) ≡ {(0 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i → B(k) = A(k))}

- A continuación a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor que n.

  (4) ≡ {(0 ≤ i ≤ n – 1) ∧ ∀k(1 ≤ k ≤ i → B(k) = A(k))}

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de i se ha incrementado en 1 pero todavía no se ha copiado el elemento de la nueva posición i de A(1..n) a B(1..n). Por tanto al crecer i podemos asegurar que ya no es 0 y que podría ser n y que los elementos copiados son los que van desde la posición 1 hasta la i – 1.

  (5) ≡ {(1 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i – 1 → B(k) = A(k))}

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que se ha copiado el elemento de la nueva posición i de A(1..n) a B(1..n). Por tanto, los elementos copiados son los que van desde la posición 1 hasta la i.

$$(6) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k))\}$$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.9. Programa que genera una copia del vector A(1..n) en B(1..n) recorriendo el vector de izquierda a derecha (Versión 2)

En este ejemplo se va a documentar, con las aserciones que se indican, el siguiente programa que dado un vector A(1..n) con n mayor o igual que 1, obtiene en B(1..n) una copia de A(1..n) recorriendo los vectores de izquierda a derecha. El programa es ligeramente distinto al presentado en el apartado 2.6.8 y se pretende ver qué cambios se producen en la documentación al tener un programa que, aunque obtenga el mismo resultado, calcula ese resultado de manera ligeramente distinta.

```
(1) ≡ {Precondición}
i : = 1;
(2) ≡ {Aserción intermedia}
while (3) ≡ {Invariante} i ≤ n loop
        (4) ≡ {Aserción intermedia}
        B(i) : = A(i);
        (5) ≡ {Aserción intermedia}
        i : = i + 1;
        (6) ≡ {Aserción intermedia}
end loop;
(7) ≡ {Postcondición}
(8) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4) y (5) se formularán partiendo del invariante. Por último se dará la expresión-cota E:

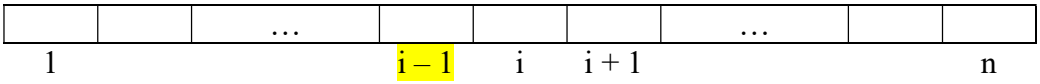
- (1) ≡ {n ≥ 1}
  (7) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k))}

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final los vectores A(1..n) y B(1..n) contendrán los mismos elementos.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 1:

  (2) ≡ {n ≥ 1 ∧ i = 1}

- A continuación, a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 (la última vez, justo antes de terminar el while por no cumplirse la condición i ≤ n). Por otra parte como una vez entrado en el while lo primero que se hace es copiar el elemento de la posición i del vector A(1..n) a la posición i del vector B(1..n), en el punto (3) se tiene que ya se han copiado los elementos de A(1..i – 1) a B(1..i – 1) pero el elemento de la posición i está todavía sin ser copiado.

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k))\}$

- A continuación, a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor o igual que n.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k))\}$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de la posición i del vector A(1..n) ha sido copiado en la posición i del vector B(1..n). Por tanto, los elementos copiados son los que van desde la posición 1 hasta la i.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k))\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de la variable i se ha incrementado en uno. Por tanto los elementos copiados son los que van desde la posición 1 hasta la i – 1 y los límites de i crecen en uno.

  $(5) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k))\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n + 1 – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en

el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.10. Programa que genera una copia del vector A(1..n) en B(1..n) recorriendo el vector de derecha a izquierda (Versión 1)

En este ejemplo se va a documentar, con las aserciones que se indican, el siguiente programa que dado un vector A(1..n) con n mayor o igual que 1, obtiene en B(1..n) una copia de A(1..n) recorriendo los vectores de derecha a izquierda. Este programa produce el mismo resultado final que los programas presentados en los apartados 2.6.8 y 2.6.9 pero el resultado se obtiene de manera distinta. Es importante observar las diferencias que hay en la documentación con respecto a esos dos ejemplos previos.

```
(1) ≡ {Precondición}
i : = n + 1;
(2) ≡ {Aserción intermedia}
while (3) ≡ {Invariante} i > 1 loop
        (4) ≡ {Aserción intermedia}
        i : = i – 1;
        (5) ≡ {Aserción intermedia}
        B(i) : = A(i);
        (6) ≡ {Aserción intermedia}
end loop;
(7) ≡ {Postcondición}
(8) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4) y (5) se formularán partiendo del invariante. Por último se dará la expresión cota E:

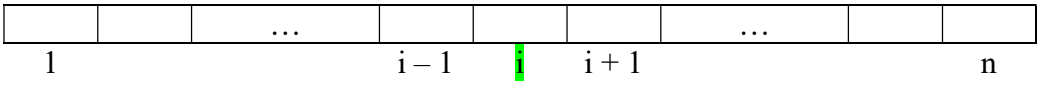
- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k))\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final los vectores A(1..n) y B(1..n) contendrán los mismos elementos. La precondición y la postcondición coinciden con los dados en los ejemplos de los apartados 2.6.8 y 2.6.9.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor n + 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale n + 1:

  $(2) \equiv \{n \geq 1 \wedge i = n + 1\}$

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre n + 1 (la primera vez que pase por ahí) y 1 (la última vez, justo antes de terminar el while por no cumplirse la condición i > 1). Por otra parte, como una vez entrado en el while lo primero que se hace es decrementar el valor de i, en el punto (3) se tiene que ya se han copiado los elementos que van desde la posición i hasta la n.

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(i \leq k \leq n \rightarrow B(k) = A(k))\}$

- A continuación a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y, por tanto, sabemos que i es mayor que 1.

  $(4) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(i \leq k \leq n \rightarrow B(k) = A(k))\}$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de i se ha decrementado en 1 pero todavía no se ha copiado el elemento de la nueva posición i de A(1..n) a B(1..n). Por tanto, al decrecer i podemos asegurar que ya no es n + 1 y que podría ser 1 y que los elementos copiados son los que van desde la posición i + 1 hasta la n.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(i + 1 \leq k \leq n \rightarrow B(k) = A(k))\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que se ha copiado el elemento de la nueva posición i de A(1..n) a B(1..n). Por tanto, los elementos copiados son los que van desde la posición i hasta la n.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(i \leq k \leq n \rightarrow B(k) = A(k))\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar como máximo. Cuando se está recorriendo un vector de derecha a izquierda la expresión cota suele ser "i menos el último valor que tomará i ". Así que en este caso E = i – 1.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en

el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.11. Programa que genera una copia del vector A(1..n) en B(1..n) recorriendo el vector de derecha a izquierda (Versión 2)

En este ejemplo se va a documentar, con las aserciones que se indican, el siguiente programa que dado un vector A(1..n) con n mayor o igual que 1, obtiene en B(1..n) una copia de A(1..n) recorriendo los vectores de derecha a izquierda. El programa obtiene el mismo resultado final que los programas presentados en los apartados 2.6.8, 2.6.9 y 2.6.10 pero es ligeramente distinto y eso conlleva que la documentación sea también ligeramente distinta. Conviene fijarse en los detalles que cambian.

```
(1) ≡ {Precondición}
i : = n;
(2) ≡ {Aserción intermedia}
while (3) ≡ {Invariante} i > 0 loop
        (4) ≡ {Aserción intermedia}
        B(i) : = A(i);
        (5) ≡ {Aserción intermedia}
        i : = i – 1;
        (6) ≡ {Aserción intermedia}
end loop;
(7) ≡ {Postcondición}
(8) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (4) y (5) se formularán partiendo del invariante. Por último se dará la expresión cota E:

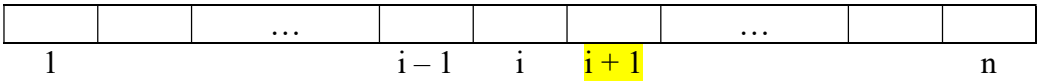
- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k))\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (6) se ha indicado que al final los vectores A(1..n) y B(1..n) contendrán los mismos elementos. La precondición y la postcondición coinciden con los dados en los tres ejemplos previos ya que los cuatro programas parten de la misma situación y obtienen el mismo resultado aunque lo hagan de manera distinta.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor n. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale n:

  $(2) \equiv \{n \geq 1 \wedge i = n\}$

- A continuación, a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre n (la primera vez que pase por ahí) y 0 (la última vez, justo antes de terminar el while por no cumplirse la condición i > 0). Por otra parte, como una vez entrado en el while lo primero que se hace es copiar el elemento de la posición i del vector A(1..n) a la posición i del vector B(1..n), en el punto (3) se tiene que ya se han copiado los elementos desde la posición i + 1 hasta la posición n.

  $(3) \equiv \{(0 \leq i \leq n) \wedge \forall k(i + 1 \leq k \leq n \rightarrow B(k) = A(k))\}$

- A continuación, a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante, en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es mayor que 0.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(i + 1 \leq k \leq n \rightarrow B(k) = A(k))\}$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de la posición i del vector A(1..n) se ha copiado a la posición i del vector B(1..n) pero la variable i no ha sido actualizada todavía.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(i \leq k \leq n \rightarrow B(k) = A(k))\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de i se ha decrementado en uno. Por tanto, los elementos copiados son los que van desde la posición i + 1 hasta la n. por su parte, los nuevos límites de i son 0 y n – 1.

  $(6) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(i + 1 \leq k \leq n \rightarrow B(k) = A(k))\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de derecha a izquierda la expresión cota suele ser "i menos el último valor que tomará i ". Así que en este caso E = i – 0, es decir, E = i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.12. Programa que guarda en B(1..n) los elementos de A(1..n) pero rotándolos hacia la izquierda

En este ejemplo se va a documentar, con las aserciones que se indican, el siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, guarda los elementos de A(1..n) en B(1..n) pero rotándolos hacia la izquierda.

```
(1) ≡ {Precondición}
B(n) := A(1);
(2) ≡ {Aserción intermedia}
i : = 1;
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i < n loop
        (5) ≡ {Aserción intermedia}
        B(i) : = A(i + 1);
        (6) ≡ {Aserción intermedia}
        i : = i + 1;
        (7) ≡ {Aserción intermedia}
end loop;
(8) ≡ {Postcondición}
(9) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2), a partir de la aserción intermedia (2) se obtendrá la (3) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- $(1) \equiv \{n \geq 1\}$
  $(8) \equiv \{B(n) = A(1) \wedge \forall k(1 \leq k \leq n - 1 \rightarrow B(k) = A(k + 1))\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (8) se ha indicado que al final el vector B(1..n) contendrá los elementos de A(1..n) pero rotados una posición hacia la izquierda.

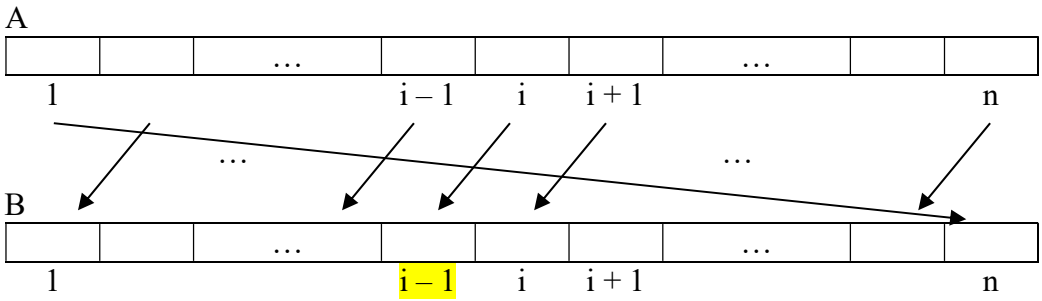
- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a B(n) se le ha asignado el valor A(1). En el punto (2) sabemos que n es mayor o igual que 1 y que B(n) es igual a A(1):

  $(2) \equiv \{n \geq 1 \wedge B(n) = A(1)\}$

- Ahora a partir de la precondición (2) se obtiene la aserción intermedia (3) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (3) sabemos que n es mayor o igual que 1, que B(n) es igual a A(1) y que i vale 1:

  $(3) \equiv \{n \geq 1 \wedge B(n) = A(1) \wedge i = 1\}$

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición i < n). Por otra parte, como una vez entrado en el while lo primero que se hace es copiar el elemento A(i + 1) en la posición i del vector B, se tiene que en las primeras i – 1 posiciones de B(1..n) ya se han copiado los correspondientes elementos de A(1..n) (los que van de la posición 2 hasta la i).

$$(4) \equiv \{B(n) = A(1) \land (1 \leq i \leq n) \land \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k + 1))\}$$

- A continuación, a partir del invariante, se obtendrá la aserción intermedia (5). Con respecto al invariante, lo que ha cambiado en el punto (5) es que se ha entrado en el while, es decir, se ha cumplido la condición del while y, por tanto, sabemos que i es menor que n.

$$(5) \equiv \{B(n) = A(1) \land (1 \leq i \leq n - 1) \land \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k + 1))\}$$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el elemento de la posición i + 1 del vector A se ha copiado a la posición i del vector B.

$$(6) \equiv \{B(n) = A(1) \land (1 \leq i \leq n - 1) \land \forall k(1 \leq k \leq i \rightarrow B(k) = A(k + 1))\}$$

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que el valor de i se ha incrementado en uno. Ahora, con el nuevo valor de i, los elementos copiados van desde la posición 1 hasta la i – 1. Los límites de i también crecen en uno.

$$(6) \equiv \{B(n) = A(1) \land (2 \leq i \leq n) \land \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k + 1))\}$$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser " el último valor que tomará i menos i". Así que en este caso E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.13. Programa que rota los elementos de A(1..n) hacia la izquierda en el propio vector A(1..n)

En este caso se va a documentar el siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, rota los elementos de A(1..n) hacia la izquierda.

```
(1) ≡ {Precondición}
aux := A(1);
(2) ≡ {Aserción intermedia}
i : = 1;
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i < n loop
        (5) ≡ {Aserción intermedia}
        A(i) : = A(i + 1);
        (6) ≡ {Aserción intermedia}
        i : = i + 1;
        (7) ≡ {Aserción intermedia}
end loop;
(8) ≡ {Aserción intermedia}
A(n) := aux;
(9) ≡ {Postcondición}
(10) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición y la aserción intermedia (8) que es la verdadera "postcondición" del while. A partir de la precondición se obtendrá la aserción intermedia (2), a partir de la aserción intermedia (2) se obtendrá la (3) y a partir de la aserción intermedia (8) se obtendrá el invariante. Las aserciones intermedias (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión-cota E:

- $(1) \equiv \{n \geq 1 \wedge \forall k(1 \leq k \leq n \rightarrow A(k) = a_k)\}$
  $(9) \equiv \{A(n) = a_1 \wedge \forall k(1 \leq k \leq n - 1 \rightarrow A(k) = a_{k+1})\}$
  $(8) \equiv \{aux = a_1 \wedge \forall k(1 \leq k \leq n - 1 \rightarrow A(k) = a_{k+1})\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y que los valores iniciales son a_1, …, a_n. Mediante la postcondición (9) se ha indicado que al final el vector A(1..n) contendrá los valores iniciales a_1, …, a_n pero rotados una posición hacia la izquierda. La aserción intermedia (8) indica que tras terminar el while los elementos a_2, …, a_n ocupan ahora las posiciones que van desde la 1 hasta la n – 1 y que el valor a_1 está en la variable auxiliar aux.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a la variable auxiliar aux se le ha asignado el valor A(1). En el punto (2) sabemos que n es mayor o igual que 1 y que aux es igual a A(1):

  (2) ≡ {n ≥ 1 ∧ aux = A(1) = a_1}

- Ahora a partir de la precondición (2) se obtiene la aserción intermedia (3) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (3) sabemos que n es mayor o igual que 1, que aux es igual a a_1 y que i vale 1:

  (3) ≡ {n ≥ 1 ∧ aux = A(1) = a_1 ∧ i = 1}

- A continuación a partir de la aserción intermedia (7) se obtendrá el invariante. La diferencia entre la aserción (7) y el invariante es que en la aserción (7) se indica cuál es la situación una vez recorrido todo el vector y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.

  A

a_2	a_3	…	a_i	a_i	a_{i+1}	…		a_n
1			i – 1	i	i + 1			n

  En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición i < n). Por otra parte como una vez entrado en el while lo primero que se hace es copiar el elemento A(i + 1) en la posición i del vector A, se tiene que se han desplazado hacia la izquierda los elementos que van desde la posición 2 hasta la i y ahora ocupan las posiciones que van desde la 1 hasta la i – 1.

  (4) ≡ {aux = a_1 ∧ (1 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i – 1 → A(k) = a_{k+1})}

  Por otra parte ya no se puede decir que aux sea igual a A(1) porque puede que el valor de A(1) haya cambiado y por eso tenemos que decir solamente que aux es igual al valor inicial de A(1), es decir, a_1.

- A continuación a partir del invariante se obtendrá la aserción intermedia (5). Con respecto al invariante lo que ha cambiado en el punto (5) es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor que n.

  (5) ≡ {aux = a_1 ∧ (1 ≤ i ≤ n – 1) ∧ ∀k(1 ≤ k ≤ i – 1 → A(k) = a_{k+1})}

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el elemento de la posición i + 1 del vector A se ha copiado a la posición i del vector A.

  (6) ≡ {aux = a_1 ∧ (1 ≤ i ≤ n – 1) ∧ ∀k(1 ≤ k ≤ i → A(k) = a_{k+1})}

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que el valor de i se ha incrementado en uno. Por tanto, los límites de i crecen en uno y las posiciones que tienen su valor definitivo son las que van desde 1 hasta la i – 1.

  $(7) \equiv \{aux = a_1 \wedge (2 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = a_{k+1})\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser " el último valor que tomará i menos i". Así que en este caso E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ni la variable auxiliar aux ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (las variables i y aux en este caso) no se mencionan. En el enunciado tampoco aparecen la variables i y aux y eso ya nos indica que tampoco hay que incluirlas en la especificación pre-post. Pero las variables i y aux sí aparecen en el invariante y en las aserciones intermedias porque durante el proceso de cálculo del resultado sí tienen sentido y son importantes.

2.6.14. Programa que cuenta en c el número de ceros de A(1..n) (Versión 1)

En este apartado se va a documentar el siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, obtiene en c el número de ceros de A(1..n).

```
(1) ≡ {Precondición}
i : = 1;
(2) ≡ {Aserción intermedia}
c : = 0;
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i <= n loop
        (5) ≡ {Aserción intermedia}
        if A(i) = 0 then (6) ≡ {Aserción intermedia}
                        c : = c + 1;
                        (7) ≡ {Aserción intermedia}
        end if;
        (8) ≡ {Aserción intermedia}
        i : = i + 1;
        (9) ≡ {Aserción intermedia}
end loop;
(10) ≡ {Postcondición}
(11) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2), a partir de la aserción (2) se obtendrá la (3) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (5), (6),

(7), (8) y (9) se formularán partiendo del invariante. Por último, se dará la expresión cota E:

- (1) ≡ {n ≥ 1}
  (10) ≡ {c = Nk(1 ≤ k ≤ n ∧ A(k) = 0)}

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postconción (10) se ha indicado que al final en c se tendrá el número de ceros que hay en A(1..n).

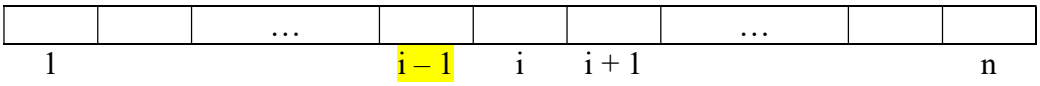
- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 1:

  (2) ≡ {n ≥ 1 ∧ i = 1}

- La aserción intermedia (3) se obtiene a partir de la aserción (2) teniendo en cuenta que c ha sido inicializada con el valor 0:

  (3) ≡ {n ≥ 1 ∧ i = 1 ∧ c = 0}

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 (la última vez, justo antes de terminar el while por no cumplirse la condición i ≤ n). Por otra parte como una vez entrado en el while lo primero que se hace es comparar el elemento A(i) con 0, en el punto (4) se tiene que ya se han comparado los elementos que van desde la posición 1 a la i – 1.

  (4) ≡ {(1 ≤ i ≤ n + 1) ∧ c = Nk(1 ≤ k ≤ i – 1 ∧ A(k) = 0)}

- A continuación, a partir del invariante se obtendrá la aserción intermedia (5). Con respecto al invariante en el punto (5) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor o igual que n.

  (5) ≡ {(1 ≤ i ≤ n) ∧ c = Nk(1 ≤ k ≤ i – 1 ∧ A(k) = 0)}

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de A(i) es 0 pero que todavía no se ha actualizado c. Por tanto, los

elementos comparados son los que van desde la posición 1 hasta la i y además se sabe que en la posición i hay un cero.

$(6) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i - 1 \wedge A(k) = 0) \wedge A(i) = 0\}$

Como A(k) es 0 y como en c hemos contabilizado hasta ahora los ceros que hay desde la posición 1 hasta la i – 1, podemos decir que c + 1 es el número de ceros que hay hasta la posición i incluida. Por tanto la aserción del punto (6) también podría ser la siguiente:

$(6) \equiv \{(1 \leq i \leq n) \wedge c + 1 = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que el valor de A(i) es 0 y que se ha actualizado c. Por tanto, los elementos comparados son los que van desde la posición 1 hasta la i y además se sabe que en la posición i hay un cero y ese cero ya se ha contabilizado en c.

  $(7) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i \wedge A(k) = 0) \wedge A(i) = 0\}$

- La aserción intermedia (8) se calculará a partir de la (4). En la (8) se ha de describir la situación de ese punto pero no sabemos si el elemento de la posición i es un cero o no pero sabemos que si es cero la variable c ha sido incrementada, es decir, sabemos que la variable c está actualizada. Por tanto, los elementos comparados son los que van desde la posición 1 hasta la i y, además, la variable c ha sido actualizada.

  $(8) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- La aserción intermedia (9) se calculará a partir de la (8). En la (9) se ha de describir la situación de ese punto en el que se acaba de actualizar la variable i. Por tanto, los elementos comparados son los que van desde la posición 1 hasta la i – 1, la variable c ha sido actualizada y, además, los límites de i son 2 y n + 1.

  $(9) \equiv \{(2 \leq i \leq n + 2) \wedge c = Nk(1 \leq k \leq i - 1 \wedge A(k) = 0)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso: E = n + 1 – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.15. Programa que cuenta en c el número de ceros de A(1..n) (Versión 2)

Ahora se va a documentar el siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, obtiene en c el número de ceros de A(1..n). El programa obtiene el mismo resultado que el programa presentado en el apartado 4.2.9 pero obtiene el resultado de manera distinta. Conviene fijarse en las diferencias que surgen al documentar este programa y el del apartado 4.2.9.

```
(1) ≡ {Precondición}
i : = 0;
(2) ≡ {Aserción intermedia}
c : = 0;
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i < n loop
        (5) ≡ {Aserción intermedia}
        i : = i + 1;
        (6) ≡ {Aserción intermedia}
        if A(i) = 0 then (7) ≡ {Aserción intermedia}
                        c : = c + 1;
                        (8) ≡ {Aserción intermedia}
        end if;
        (9) ≡ {Aserción intermedia}
end loop;
(10) ≡ {Postcondición}
(11) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2); a partir de la aserción (2) se obtendrá la (3) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (5), (6), (7), (8) y (9) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- $(1) \equiv \{n \geq 1\}$
  $(10) \equiv \{c = Nk(1 \leq k \leq n \wedge A(k) = 0)\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (10) se ha indicado que al final en c se tendrá el número de ceros que hay en A(1..n). La precondición y la postcondición coinciden con los del programa del apartado anterior porque los dos parten de la misma situación y calculan el mismo resultado.

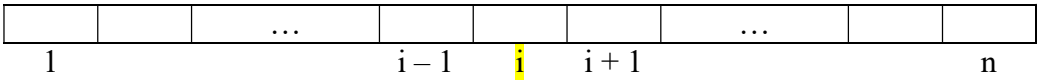
- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 0. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 0:

  $(2) \equiv \{n \geq 1 \wedge i = 0\}$

- La aserción intermedia (3) se obtiene a partir de la aserción (2) teniendo en cuenta que c ha sido inicializada con el valor 0:

  (3) ≡ {n ≥ 1 ∧ i = 0 ∧ c = 0}

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 0 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición i < n). Por otra parte, como una vez entrado en el while lo primero que se hace es incrementar el valor de i en 1, en el punto (4) se tiene que ya se han comparado los elementos que van de la posición 1 hasta la i.

  (4) ≡ {(0 ≤ i ≤ n) ∧ c = Nk(1 ≤ k ≤ i ∧ A(k) = 0)}

- A continuación a partir del invariante se obtendrá la aserción intermedia (5). Con respecto al invariante en el punto (5) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor que n.

  (5) ≡ {(0 ≤ i ≤ n – 1) ∧ c = Nk(1 ≤ k ≤ i ∧ A(k) = 0)}

- La aserción intermedia (6) se calculará a partir de la (5). En el punto (5) se ha incrementado el valor de i, pero la nueva posición i del vector A(1..n) no ha sido analizada. Al haberse incrementado el valor de i, sabemos que estará entre 1 y n y c contiene el número de ceros hasta la posición i – 1:

  (6) ≡ {(1 ≤ i ≤ n) ∧ c = Nk(1 ≤ k ≤ i – 1 ∧ A(k) = 0)}

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que el valor de A(i) es 0 pero que todavía no se ha actualizado c. Por tanto los elementos comparados son los que van desde la posición 1 hasta la i y además se sabe que en la posición i hay un cero.

  (7) ≡ {(1 ≤ i ≤ n) ∧ c = Nk(1 ≤ k ≤ i – 1 ∧ A(k) = 0) ∧ A(i) = 0}

  Como A(k) es 0 y como en c hemos contabilizado hasta ahora los ceros que hay desde la posición 1 hasta la i – 1, podemos decir que c + 1 es el número de ceros que hay hasta la posición i incluida. Por tanto la aserción del punto (7) también podría ser la siguiente:

$(7) \equiv \{(1 \leq i \leq n) \wedge c + 1 = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- La aserción intermedia (8) se calculará a partir de la (7), teniendo en cuenta que el valor de A(i) es 0 y que se ha actualizado c. Por tanto, los elementos comparados son los que van desde la posición 1 hasta la i y además se sabe que en la posición i hay un cero y que la variable c está actualizada.

  $(8) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i \wedge A(k) = 0) \wedge A(i) = 0\}$

- La aserción intermedia (9) se calculará a partir de la (6). En la aserción (9) se ha de describir la situación de ese punto pero no sabemos si el elemento de la posición i es un cero o no pero sabemos que si es cero la variable c ha sido incrementada, es decir, sabemos que la variable c está actualizada. Por tanto, los elementos comparados son los que van desde la posición 1 hasta la i y, además, la variable c ha sido actualizada.

  $(9) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso: E = n – i.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i si aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i si tiene sentido y es importante.

2.6.16. Programa que decide en b si A(1..n) contiene algún 0 (Versión 1)

Se va a documentar el siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, decide en la variable booleana b si A(1..n) contiene o no algún 0:

```
(1) ≡ {Precondición}
i : = 1;
(2) ≡ {Aserción intermedia}
b : = False;
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i <= n and not b loop
        (5) ≡ {Aserción intermedia}
        b : = (A(i) = 0);
        (6) ≡ {Aserción intermedia}
        i : = i + 1;
        (7) ≡ {Aserción intermedia}
end loop;
(8) ≡ {Postcondición}
(9) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2); a partir de la aserción (2) se obtendrá la (3) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (5), (6) y (7) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{b \leftrightarrow \exists k(1 \leq k \leq n \wedge A(k) = 0)\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final la variable booleana b denota si en A(1..n) hay algún cero o no, es decir, b valdrá true si hay algún cero (por lo menos uno) y valdrá false si no hay ningún cero.

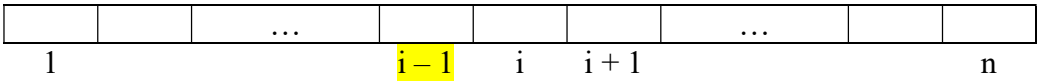
- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 1:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- Ahora a partir de la aserción intermedia (2) se obtendrá la (3) teniendo en cuenta que a b se le ha asignado el valor False. En el punto (3) sabemos que n es mayor o igual que 1, que i vale 1 y que b vale False:

  $(3) \equiv \{n \geq 1 \wedge i = 1 \wedge \neg b\}$

- A continuación a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 (la última vez, justo antes de terminar el while por no cumplirse la condición i ≤ n). En este caso i no tiene porqué llegar siempre a n + 1 ya que si se encuentra un cero el while termina. Por otra parte como una vez entrado en el while lo primero que se hace es comparar el elemento A(i) con 0, en el punto (4) se tiene que ya se han comparado los elementos hasta la posición i – 1 incluida pero el elemento de la posición i no se ha comparado todavía.

  $(4) \equiv \{(1 \leq i \leq n + 1) \wedge b \leftrightarrow \exists k(1 \leq k \leq i - 1 \wedge A(k) = 0)\}$

- A continuación a partir del invariante se obtendrá la aserción intermedia (5). Con respecto al invariante en el punto (5) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor o igual que n.

  $(5) \equiv \{(1 \leq i \leq n) \wedge b \leftrightarrow \exists k(1 \leq k \leq i - 1 \wedge A(k) = 0)\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de b ha sido actualizado tras comprobar si en la posición i se tiene un 0 o no.

  $(6) \equiv \{(1 \leq i \leq n) \wedge b \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = 0)\}$

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que el valor de i se ha incrementado en 1. Por tanto, b indica si en la posición i – 1 se tiene un 0 o no y los límites de i han crecido en uno.

  $(7) \equiv \{(2 \leq i \leq n + 1) \wedge b \leftrightarrow \exists k(1 \leq k \leq i - 1 \wedge A(k) = 0)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n + 1 – i. En este caso E indica cuántas vueltas se darán como máximo, ya que es posible encontrar un 0 antes de recorrer todo el vector, en cuyo caso el while se termina sin que acabe de recorrer todo el vector.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que

tampoco hay que incluirla en la especificación pre-post. Pero la variable i si aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i si tiene sentido y es importante.

2.6.17. Programa que decide en b si A(1..n) contiene algún 0 (Versión 2)

Se va a documentar el siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, decide en la variable booleana b si A(1..n) contiene o no algún 0. La especificación pre-post es la misma que en el ejemplo de la sección 4.2.9 porque ambos programas parten de la misma situación y calculan el mismo resultado pero como lo hacen de manera distinta, el invariante y las aserciones intermedias serán distintas:

```
(1) ≡ {Precondición}
i : = 1;
(2) ≡ {Aserción intermedia}
b : = (A(1) = 0);
(3) ≡ {Aserción intermedia}
while (4) ≡ {Invariante} i ≠ n and not b loop
        (5) ≡ {Aserción intermedia}
        b : = (A(i + 1) = 0);
        (6) ≡ {Aserción intermedia}
        i : = i + 1;
        (7) ≡ {Aserción intermedia}
end loop;
(8) ≡ {Postcondición}
(9) ≡ {Expresión cota E}
```

Primero hay que dar la precondición y la postcondición. A partir de la precondición se obtendrá la aserción intermedia (2); a partir de la aserción (2) se obtendrá la (3) y a partir de la postcondición se obtendrá el invariante. Las aserciones intermedias (5), (6) y (7) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{b \leftrightarrow \exists k(1 \leq k \leq n \wedge A(k) = 0)\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (7) se ha indicado que al final la variable booleana b denota si en A(1..n) hay algún cero o no, es decir, b valdrá true si hay algún cero (por lo menos uno) y valdrá false si no hay ningún cero.

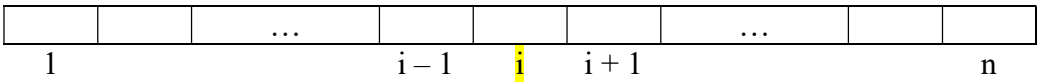
- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1. En el punto (2) sabemos que n es mayor o igual que 1 y que i vale 1:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- Ahora a partir de la aserción intermedia (2) se obtendrá la (3) teniendo en cuenta que a b se le ha asignado el resultado de comparar A(1) con 0. En el punto (3) sabemos que n es mayor o igual que 1, que i vale 1 y que b contiene el resultado de comparar A(1) con 0:

  $(3) \equiv \{n \geq 1 \wedge i = 1 \wedge b \leftrightarrow (A(1) = 0)\}$

- A continuación, a partir de la postcondición se obtendrá el invariante. La diferencia entre la postcondición y el invariante es que en la postcondición se indica cuál es la situación una vez terminado el programa y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (4) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n (la última vez, justo antes de terminar el while por no cumplirse la condición $i \neq n$). En este caso i no tiene porqué llegar siempre a n ya que si se encuentra un cero el while termina. Por otra parte como una vez entrado en el while lo primero que se hace es comparar el elemento A(i + 1) con 0, en el punto (4) se tiene que ya se han comparado los elementos hasta la posición i incluida.

  $(4) \equiv \{(1 \leq i \leq n) \wedge b \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = 0)\}$

- A continuación, a partir del invariante se obtendrá la aserción intermedia (5). Con respecto al invariante en el punto (5) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i no es igual a n.

  $(5) \equiv \{(1 \leq i \leq n - 1) \wedge b \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = 0)\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de b ha sido actualizado tras comprobar si en la posición i + 1 se tiene un 0 o no.

  $(6) \equiv \{(1 \leq i \leq n - 1) \wedge b \leftrightarrow \exists k(1 \leq k \leq i + 1 \wedge A(k) = 0)\}$

- La aserción intermedia (7) se calculará a partir de la (6), teniendo en cuenta que el valor de i se ha incrementado en uno y que, consecuentemente, b indica si en la posición i se tiene un 0 o no. Además, los límites de i han crecido en uno.

  $(7) \equiv \{(2 \leq i \leq n) \wedge b \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = 0)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso E = n – i. En este caso E indica cuántas vueltas se darán como

máximo, ya que es posible encontrar un 0 antes de recorrer todo el vector, en cuyo caso el while se termina sin que acabe de recorrer todo el vector.

Es importante observar que en la precondición y en la postcondición no aparece la variable índice i ya que la especificación pre-post sirve para describir los datos de entrada y el resultado. Las variables utilizadas como auxiliares en el proceso (la i en este caso) no se mencionan. En el enunciado tampoco aparece la i y eso ya nos indica que tampoco hay que incluirla en la especificación pre-post. Pero la variable i sí aparece en el invariante y en las aserciones intermedias porque durante el proceso de cálculo la variable i sí tiene sentido y es importante.

2.6.18. Programa que decide en b si A(1..n) contiene más pares que impares

Se va a dar la especificación pre-post correspondiente al siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1, decide en la variable booleana b si A(1..n) contiene más pares que impares:

```
(1) ≡ {Precondición}
i : = 1;
p: = 0;
imp := 0;
(2) ≡ {Aserción intermedia}
while (3) ≡ {Invariante} i <= n loop
        (4) ≡ {Aserción intermedia}
        if A(i) mod 2 = 0 then (5) ≡ {Aserción intermedia}
                                p : = p + 1;
                                (6) ≡ {Aserción intermedia}
        else (7) ≡ {Aserción intermedia}
                imp : = imp + 1;
             (8) ≡ {Aserción intermedia}
        end if;
        (9) ≡ {Aserción intermedia}
        i : = i + 1;
        (10) ≡ {Aserción intermedia}
end loop;
(11) ≡ {Aserción intermedia}
b : = (p > imp);
(12) ≡ {Postcondición}
(13) ≡ {Expresión cota E}
```

Primero hay que dar la precondición, la postcondición y la aserción (11) que es la "postcondición " del while. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la aserción (11) se obtendrá el invariante. Las aserciones intermedias (4), (5), (6), (7), (8), (9) y (10) se formularán partiendo del invariante. Por último se dará la expresión cota E:

- (1) ≡ {n ≥ 1}

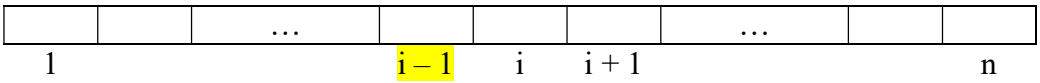
(9) ≡ {b ↔ Nk(1 ≤ k ≤ n ∧ par(A(k))) > Nk(1 ≤ k ≤ n ∧ ¬par(A(k)))}
(8) ≡ {p = Nk(1 ≤ k ≤ n ∧ par(A(k))) ∧ imp = Nk(1 ≤ k ≤ n ∧ ¬par(A(k)))}

Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y mediante la postcondición (9) se ha indicado que al final la variable booleana b denota si en A(1..n) hay más pares que impares o no. La aserción (8) indica que al finalizar el while se tendrá en p el número de elementos pares y en imp el número de elementos impares.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 1 y a p e imp el valor 0:

  (2) ≡ {n ≥ 1 ∧ i = 1 ∧ p = 0 ∧ imp = 0}

- A continuación, a partir de la aserción (8) se obtendrá el invariante. La diferencia entre la aserción (8) y el invariante es que en la aserción (8) se indica cuál es la situación una vez recorrido todo el vector y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



  En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 (la última vez, justo antes de terminar el while por no cumplirse la condición i ≤ n). Por otra parte como una vez entrado en el while lo primero que se hace es comprobar si el elemento A(i) es par o impar, en el punto (3) se tiene que ya se han comparado los elementos desede la posición 1 hasta la posición i – 1 incluida, pero el elemento de la posición i no se ha comparado todavía.

  (3) ≡ {(1 ≤ i ≤ n + 1) ∧ p = Nk(1 ≤ k ≤ i – 1 ∧ par(A(k))) ∧ imp = Nk(1 ≤ k ≤ i – 1 ∧ ¬par(A(k)))}

- A continuación a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y por tanto sabemos que i es menor o igual que n.

  (4) ≡ {(1 ≤ i ≤ n) ∧ p = Nk(1 ≤ k ≤ i – 1 ∧ par(A(k))) ∧ imp = Nk(1 ≤ k ≤ i – 1 ∧ ¬par(A(k)))}

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de A(i) es par pero que las variables p e imp no han sido actualizadas todavía.

  (5) ≡ {(1 ≤ i ≤ n) ∧ par(A(i)) ∧ p = Nk(1 ≤ k ≤ i – 1 ∧ par(A(k))) ∧ imp = Nk(1 ≤ k ≤ i – 1 ∧ ¬par(A(k))) }

Pero sabiendo que en p tenemos los pares hasta la posición i – 1 y en imp los impares hasta la posición i – 1 y que A(i) es par, podemos decir que p + 1 es el número de pares hasta la posición i y que imp es el número de impares hasta la posición i. por tanto la siguiente fórmula es también adecuada para el punto (5):

$(5) \equiv \{(1 \leq i \leq n) \wedge p + 1 = Nk(1 \leq k \leq i \wedge par(A(k))) \wedge imp = Nk(1 \leq k \leq i \wedge \neg par(A(k)))\}$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de A(i) es par y que las variables p e imp están actualizadas.

$(6) \equiv \{(1 \leq i \leq n) \wedge par(A(i)) \wedge p = Nk(1 \leq k \leq i \wedge par(A(k))) \wedge imp = Nk(1 \leq k \leq i \wedge \neg par(A(k)))\}$

- La aserción intermedia (7) se calculará a partir de la (4), teniendo en cuenta que el valor de A(i) es impar pero que la variable imp no ha sido actualizada todavía.

$(7) \equiv \{(1 \leq i \leq n) \wedge \neg par(A(i)) \wedge p = Nk(1 \leq k \leq i - 1 \wedge par(A(k))) \wedge imp = Nk(1 \leq k \leq i - 1 \wedge \neg par(A(k)))\}$

Pero sabiendo que en p tenemos los pares hasta la posición i – 1 y en imp los impares hasta la posición i – 1 y que A(i) es impar, podemos decir que p es el número de pares hasta la posición i y que imp + 1 es el número de impares hasta la posición i. Por tanto la siguiente fórmula es también adecuada para el punto (6):

$(6) \equiv \{(1 \leq i \leq n) \wedge p = Nk(1 \leq k \leq i \wedge par(A(k))) \wedge imp + 1 = Nk(1 \leq k \leq i \wedge \neg par(A(k)))\}$

- La aserción intermedia (8) se calculará a partir de la (7), teniendo en cuenta que el valor de A(i) es impar y que la variable imp ha sido actualizada.

$(8) \equiv \{(1 \leq i \leq n) \wedge \neg par(A(i)) \wedge p = Nk(1 \leq k \leq i \wedge par(A(k))) \wedge imp = Nk(1 \leq k \leq i \wedge \neg par(A(k)))\}$

- La aserción intermedia (9) se calculará a partir de la (4). En la (9) se ha de describir la situación de ese punto pero no sabemos si el elemento de la posición i es par o impar pero sabemos que las variables p e imp han sido actualizadas debidamente. Por tanto los elementos comparados son los que van desde la posición 1 hasta la i y además las variables p e imp están actualizadas.

$(9) \equiv \{(1 \leq i \leq n) \wedge p = Nk(1 \leq k \leq i \wedge par(A(k))) \wedge imp = Nk(1 \leq k \leq i \wedge \neg par(A(k)))\}$

- La aserción intermedia (10) se calculará a partir de la (9). En la (10) se ha de tener en cuenta que se acaba de incrementar en uno el valor de i. Consecuentemente, crecen en uno los límites de i y los elementos comparados son los que van desde la posición 1 hasta la i – 1.

$$(10) \equiv \{(2 \leq i \leq n + 1) \wedge p = Nk(1 \leq k \leq i - 1 \wedge par(A(k))) \wedge$$
$$imp = Nk(1 \leq k \leq i - 1 \wedge \neg par(A(k)))\}$$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso: E = n + 1 – i. En este caso E indica cuántas vueltas quedan por dar exactamente porque siempre se recorrerá todo el vector.

Es importante observar que en la especificación pre-post no aparecen las variables del programa que han sido utilizados para obtener resultados parciales pero que no son resultados finales. Tal es el caso de los contadores p e imp y la variable índice i.

El dato de entrada es A(1..n) y el resultado es b. Por tanto las otras tres variables no aparecen en la especificación pre-post ya que la especificación pre-post sirve para describir los datos de entrada y el resultado.

Por otra parte la variable b no aparece hasta la postcondición porque no ha sido utilizada en el while.

2.6.19. Programa que decide en pos la posición de la primera aparición de x en A(1..n) sabiendo que x aparece en A(1..n)

Se va a documentar el siguiente programa que, dado un vector A(1..n) con n mayor o igual que 1 y que contiene al menos una aparición de x, decide en la variable pos la posición de la primera aparición de x en A(1..n):

```
(1) ≡ {Precondición}
i : = 2;
seguir: = (A(1) ≠ x);
(2) ≡ {Aserción intermedia}
while (3) ≡ {Invariante} seguir loop
        (4) ≡ {Aserción intermedia}
        if A(i) = x then (5) ≡ {Aserción intermedia}
                                seguir : = False;
                        (6) ≡ {Aserción intermedia}
        end if;
        (7) ≡ {Aserción intermedia}
        i : = i + 1;
        (8) ≡ {Aserción intermedia}
end loop;
(9) ≡ {Aserción intermedia}
pos : = i – 1;
(10) ≡ {Postcondición}
(11) ≡ {Expresión cota E}
```

Primero hay que dar la precondición, la postcondición y la aserción (7) que es la verdadera "postcondición" del while. A partir de la precondición se obtendrá la aserción intermedia (2) y a partir de la aserción (7) se obtendrá el invariante. Las aserciones intermedias (4), (5) y (6) se formularán partiendo del invariante. Por último se dará la expresión cota E:

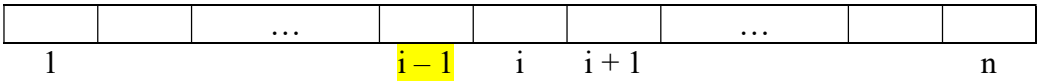
- $(1) \equiv \{n \geq 1 \wedge \exists k(1 \leq k \leq n \wedge A(k) = x)\}$
  $(10) \equiv \{1 \leq pos \leq n \wedge A(pos) = x \wedge \forall k(1 \leq k \leq pos - 1 \rightarrow A(k) \neq x)\}$
  $(9) \equiv \{2 \leq i \leq n + 1 \wedge A(i - 1) = x \wedge \forall k(1 \leq k \leq i - 2 \rightarrow A(k) \neq x)\}$

  Mediante la precondición (1) se ha expresado que el programa parte de un vector que contiene por lo menos un elemento y en el que el valor x aparece por lo menos una vez. Mediante la postcondición (10) se ha indicado que al final la variable pos contiene la posición de la primera aparición de x. La aserción (9) indica que tras finalizar el while se sabe que la primera aparición de x está en la posición i – 1.

- Ahora a partir de la precondición (1) se obtiene la aserción intermedia (2) teniendo en cuenta que a i se le ha asignado el valor 2 y a *seguir* se le ha asignado la comparación $(A(1) \neq x)$:

  $(2) \equiv \{n \geq 1 \wedge i = 2 \wedge (seguir \leftrightarrow (A(1) \neq x))\}$

- A continuación, a partir de la aserción (9) se obtendrá el invariante. La diferencia entre la aserción (9) y el invariante es que en la aserción (9) se indica cuál es la situación una vez terminada la búsqueda de x en A(1..n) y en el invariante se indica cuál es la situación en una vuelta intermedia cualquiera. Nos podemos imaginar que i va por la mitad del vector y describiremos la situación.



En el punto (3) del programa sabemos que i tendrá un valor comprendido entre 1 (la primera vez que pase por ahí) y n + 1 porque estamos seguros de que x aparecerá y por tanto i nunca estará fuera del rango [1..n + 1]. Por otra parte se entrará en el while dependiendo de que *seguir* valga True o False. Si *seguir* vale True, todavía no se ha encontrado la primera aparición de x y como una vez entrado en el while lo primero que se hace es comprobar si el elemento A(i) es igual a x, en el punto (3) se tiene que ya se han comparado los elementos que van desde la posición 1 hasta la posición i – 1 incluida, pero el elemento de la posición i no se ha comparado todavía. Si *seguir* vale False, ya se ha encontrado la primera aparición de x que está en la posición i – 1. Por tanto, sabemos que hasta i – 2 no ha aparecido y si *seguir* vale True, entonces en la posición i – 1 tampoco está x pero si *seguir* vale False entonces x está en la posición i – 1:

$$(3) \equiv \{(2 \leq i \leq n+1) \wedge \forall k(1 \leq k \leq i-2 \rightarrow A(k) \neq x) \wedge \\ \wedge (seguir \leftrightarrow A(i-1) \neq x)\}$$

Si *seguir* vale True no se sabe si en A(i) está x o no porque no se ha mirado todavía.

- A continuación, a partir del invariante se obtendrá la aserción intermedia (4). Con respecto al invariante, en el punto (4) lo que ha cambiado es que se ha entrado en el while, es decir, se ha cumplido la condición del while y, por tanto, sabemos que *seguir* es True. Consecuentemente, puesto que x ha de aparecer, podemos deducir que i es menor que n + 1 y que en la posición i – 1 no está x.

$$(4) \equiv \{(2 \leq i \leq n) \wedge \forall k(1 \leq k \leq i-1 \rightarrow A(k) \neq x) \wedge (seguir \leftrightarrow A(i-1) \neq x) \wedge seguir\}$$

- La aserción intermedia (5) se calculará a partir de la (4), teniendo en cuenta que el valor de A(i) es x pero que la variable *seguir* no ha sido actualizada todavía.

$$(5) \equiv \{(2 \leq i \leq n) \wedge \forall k(1 \leq k \leq i-1 \rightarrow A(k) \neq x) \wedge (seguir \leftrightarrow A(i-1) \neq x) \wedge seguir \wedge A(i) = x\}$$

- La aserción intermedia (6) se calculará a partir de la (5), teniendo en cuenta que el valor de A(i) es x y que la variable *seguir* ha sido actualizada.

$$(6) \equiv \{(2 \leq i \leq n) \wedge \forall k(1 \leq k \leq i-1 \rightarrow A(k) \neq x) \wedge (seguir \leftrightarrow A(i) \neq x) \wedge \neg seguir \wedge A(i) = x\}$$

- La aserción intermedia (7) se calculará a partir de la (4) ya que en la (7) se ha de describir la situación sin saber si en la posición i está o no está x pero sabiendo que la variable *seguir* está actualiza pero que la variable i no ha sido actualizada todavía. Por tanto, los elementos comparados son los que van desde la posición 1 hasta la i y además la variable *seguir* está actualizada.

  $(7) \equiv \{(2 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow A(k) \neq x) \wedge (seguir \leftrightarrow A(i) \neq x)\}$

- La aserción intermedia (8) se calculará a partir de la (7) teniendo en cuenta que la variable i ha sido actualizada. Por tanto, los elementos comparados son los que van desde la posición 1 hasta la i – 1 y además la variable *seguir* indica si el elemento de la posición i – 1 es x. Los límites de i también han crecido.

  $(8) \equiv \{(3 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 2 \rightarrow A(k) \neq x) \wedge (seguir \leftrightarrow A(i - 1) \neq x)\}$

- Por último, hay que dar la expresión cota E que indicará cuántas vueltas quedan por dar cómo máximo. Cuando se está recorriendo un vector de izquierda a derecha la expresión cota suele ser "El último valor que tomará i menos i". Así que en este caso $E = n + 1 - i$. En este caso E indica cuántas vueltas quedan por dar como máximo porque no siempre se recorrerá todo el vector.

Es importante observar que en la especificación pre-post no aparecen las variables del programa que han sido utilizados para obtener resultados parciales pero que no son resultados finales. Tal es el caso de las variables *seguir* e i. Por otra parte la variable pos aparece solo en la postcondición porque no se utiliza en el while.

2.6.20. (Parcial mayo 2006 #1)

a) **Definir** un predicado **positivos(C(1..r))** que exprese que todos los elementos del vector C(1..r) son estrictamente mayores que 0 (> 0).

b) **Definir** un predicado **anulado(z, C(1..r), D(1..r))** que exprese que el vector D(1..r) contiene un 0 en las posiciones en las que en C(1..r) se tiene el elemento z y coincide con el vector C(1..r) en el resto de las posiciones.

Ejemplo 1
Considerando que el valor z es 8, para los vectores C(1..6) y D(1..6) de este ejemplo el predicado sería cierto:

C(1..6)	2	8	1	15	20	8
	1	2	3	4	5	6
D(1..6)	2	0	1	15	20	0
	1	2	3	4	5	6

Ejemplo 2
Considerando que el valor z es 15, para los vectores C(1..6) y D(1..6) de este ejemplo el predicado no sería cierto:

C(1..6)	2	8	1	15	20	8
	1	2	3	4	5	6
D(1..6)	2	8	6	15	20	0
	1	2	3	4	5	6

c) **Definir** un predicado **veces(C(1..r), z, v)** que expresa que el elemento z aparece v veces en el vector C(1..r).

d) **Documentar**, en los puntos indicados y utilizando los predicados definidos en los apartados anteriores, el siguiente programa que, dados un entero x y un vector A(1..n) donde todos los elementos son mayores que 0, calcula en la variable p el número de apariciones de x en A(1..n) y genera en B(1..n) una copia de A(1..n) pero sustituyendo las apariciones de x por 0.

```
(1) {Precondición}
i : = 1;
p : = 0;
while (2) {Invariante} i ≠ n + 1 loop
    (3) {Aserción intermedia}
    if A(i) = x then (4) {Aserción intermedia}
                     B(i) : = 0;
                     (5) {Aserción intermedia}
                     p : = p + 1;
    else (6) {Aserción intermedia}
         B(i) : = A(i);
    end if;
    (7) {Aserción intermedia}
    i : = i + 1;
end loop;
(8) {Postcondición}
(9) {Expresión cota}
```

Solución:

a) **positivos(C(1..r))** ≡ $\{\forall i\ (1 \le i \le r \rightarrow C(i) > 0)\}$

b) **anulado(z, C(1..r), D(1..r))** ≡ $\{\forall i\ ((1 \le i \le r \wedge C(i) = z) \rightarrow D(i) = 0) \wedge$
$\wedge\ \forall i\ ((1 \le i \le r \wedge C(i) \ne z) \rightarrow D(i) = C(i))\}$

c) **veces(C(1..r), z, v)** ≡ $\{v = Ni\ (1 \le i \le r \wedge C(i) = z)\}$

d) Primero se darán la precondición, la postcondición y el invariante:

(1) $\{n \ge 1 \wedge$ positivos(A(1..n))$\}$
(8) {veces(A(1..n), x, p) ∧ anulado(x, A(1..n), B(1..n))}
(2) $\{1 \le i \le n + 1 \wedge$ veces(A(1..i – 1), x, p) ∧
anulado(x, A(1..i – 1), B(1..i – 1))}

A continuación se darán las aserciones que se encuentran dentro del while. Aquí la idea es ir indicando qué cambia en cada punto con respecto al invariante o con respecto a la aserción anterior:

(3) $\{1 \le i \le$ **n** $\wedge$ veces(A(1..i – 1), x, p) ∧ anulado(x, A(1.. i – 1), B(1..i – 1))}

En la aserción (3) sabemos que se ha entrado en el while y por tanto se ha tenido que cumplir la condición del while. Ello significa que ahora sabemos que i ≤ n, y eso es lo único que cambia en (3) con respecto a (2).

(4) $\{1 \le i \le n \wedge$ veces(A(1..i – 1), x, p) ∧ anulado(x, A(1..i – 1), B(1..i – 1)) ∧
A(i) = x$\}$

En la aserción (4) sabemos que se ha entrado en la rama then del if y por tanto se ha tenido que cumplir la condición del if. Eso significa que ahora sabemos que A(i) = x, y eso es lo único que cambia en (4) con respecto a (3). En este caso hay otra manera de expresar eso mismo:

$\{1 \le i \le n \wedge$ veces(A(1..**i**), x, **p + 1**) ∧ anulado(x, A(1..i – 1), B(1..i – 1))$\}$

Esta segunda versión es mejor.

(5) $\{1 \le i \le n \wedge$ veces(A(1..i), x, p + 1) ∧ anulado(x, A(1..**i**), B(1..**i**))$\}$

En la aserción (5) sabemos que se ha entrado en la rama then del if y que además se ha actualizado la posición i de la tabla B. La aserción (5) se ha obtenido modificando la segunda versión de la aserción (4).

(6) $\{1 \leq i \leq n \wedge veces(A(1..i), x, p) \wedge anulado(x, A(1..i-1), B(1..i-1))\}$

En la aserción (6) sabemos que se ha entrado en la rama else del if pero de momento no se ha actualizado la posición i de la tabla B. La aserción (6) se ha obtenido modificando la aserción (3) porque la (3) es su aserción anterior. Como x no está en A(i), no hay que actualizar p o mejor dicho, ya está actualizada e indica cuántas veces aparece x hasta la posición i incluida, y eso es lo que ha cambiado con respecto a (3).

(7) $\{1 \leq i \leq n \wedge veces(A(1..i), x, p) \wedge anulado(x, A(1..i), B(1..i))\}$

En la aserción (7) sabemos que, independientemente de la rama del if que se haya recorrido, B(i) y p han sido actualizados y la tabla A ha sido analizada hasta la posición i incluida.

Por último, se dará la expresión cota E:

(9) $\{E = n + 1 - i\}$
Siempre que estemos en el punto donde se cumple el invariante, la expresión E nos indica cuántas vueltas quedan por dar exactamente.

2.6.21. (Parcial mayo 2006 #2)

a) **Definir** el predicado **par(x)** que expresa que x es un número par.

b) **Definir** el predicado **particion(C(1..r), x, z)** que expresa que todos los elementos de C(1..r) que están entre las posiciones 1 y x – 1 (ambas incluidas) son pares y que todos los elementos que están entre las posiciones z y r (ambas incluidas) son impares. Hay que utilizar el predicado del apartado anterior.

c) **Definir** un predicado **perm(C(1..r), (c_1, c_2, …, c_r))** que exprese que el vector C(1..r) es una permutación de (c_1, c_2, …, c_r). Que C(1..r) sea una permutación de (c_1, c_2, …, c_r) quiere decir que cada elemento de C aparece el mismo número de veces en C que en (c_1, c_2, …, c_r).

Ejemplo 1
En este caso C(1..6) es una permutación de (2, 8, 1, 15, 20, 8):

C(1..6)	2	20	1	8	8	15
	1	2	3	4	5	6

Ejemplo 2
En este caso C(1..6) no es una permutación de (2, 8, 1, 15, 20, 8):

C(1..6)	15	1	1	8	2	20
	1	2	3	4	5	6

d) **Documentar**, en los puntos indicados y utilizando los predicados definidos en los apartados anteriores, el siguiente programa que, dado un vector de enteros A(1..n), recoloca los elementos de A(1..n) dejando todos los pares en la parte izquierda y todos los impares en la parte derecha y decide en la variable booleana p si al menos la mitad de los elementos son pares.

```
{Precondición} ≡ {n ≥ 1 ∧ ∀j(1 ≤ j ≤ n →A(j) = aj)}
i : = 1;
k : = n + 1;
while (1) {Invariante} i ≠ k loop
    (2) {Aserción intermedia}
    if A(i) mod 2 ≠ 0 then (3) {Aserción intermedia}
                           aux : = A(i);
                           (4) {Aserción intermedia}
                           A(i) : = A(k – 1);
                           A(k – 1) : = aux;
                            (5) {Aserción intermedia}
                           k : = k – 1;
    else (6) {Aserción intermedia}
        i : = i + 1;
    end if;
end loop;
(7) {Aserción intermedia}
p : = ((i – 1) > n / 2);
(8) {Postcondición}
(9) {Expresión cota}
```

Solución:

a) **par(x)** ≡ {x mod 2 = 0}

b) **particion(C(1..r), x, z)** ≡ {$\forall$i (1 ≤ i ≤ x – 1 → par(C(i))) ∧
∧ $\forall$i (z ≤ i ≤ r → ¬par(C(i)))}

c) **perm(C(1..r), (c_1, c_2, …, c_r))** ≡
≡ {$\forall$i (1 ≤ i ≤ r → Nj (1 ≤ j ≤ r ∧ C(i) = C(j)) = Nk (1 ≤ k ≤ r ∧ C(i) = c_k))

d) Primero habría que dar la precondición, la postcondición del programa (8), la postcondición del while (7) y el invariante (1). En este caso la precondición viene dada en el enunciado y por tanto tenemos que dar los otros tres:

(8) {perm(A(1..n), (a_1, a_2, …, a_n)) ∧ $\exists$j (1 ≤ j ≤ n + 1 → particion(A(1..n), j, j)) ∧
p ↔ (Nℓ (1 ≤ ℓ ≤ n ∧ par(A(ℓ))) > (n / 2))}

(7) {(1 ≤ i ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ particion(A(1..n), i, i)}

(1) {(1 ≤ i ≤ k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ particion(A(1..n), i, k)}

A continuación se darán las aserciones que se encuentran dentro del while. Aquí la idea es ir indicando qué cambia en cada punto con respecto al invariante o con respecto a la aserción anterior:

(2) {(1 ≤ **i < k** ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ particion(A(1..n), i, k)}

En la aserción (2) sabemos que se ha entrado en el while y por tanto se ha tenido que cumplir la condición del while. Ello significa que ahora sabemos que i ≠ k, y eso es lo único que cambia en (2) con respecto a (1).

(3) {(1 ≤ i < k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ particion(A(1..n), i, k)
∧ **¬par(A(i))**}

En la aserción (3) sabemos que se ha entrado en la rama then del if y por tanto se ha tenido que cumplir la condición del if. Eso significa que ahora sabemos A(i) no es par, y eso es lo único que cambia en (3) con respecto a (2).

(4) {(1 ≤ i < k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ particion(A(1..n), i, k)
∧ ¬par(A(i)) ∧ **aux = A(i)**}

En la aserción (4) sabemos que se ha entrado en la rama then del if y que además se ha asignado A(i) a aux. La aserción (4) se ha obtenido añadiendo esa información a la aserción (3).

(5) $\{(1 \leq i < k \leq n + 1) \wedge perm(A(1..n), (a_1, a_2, \ldots, a_n)) \wedge$
$particion(A(1..n), i, \mathbf{k - 1})\}$

En la aserción (5) sabemos que se ha entrado en la rama else del if y se ha hecho el intercambio de valores de las posiciones i y k – 1 pero de momento no se han actualizado los valores de i y k. La aserción (5) se ha obtenido modificando la aserción (4) para expresar que ahora los impares empiezan desde la posición k – 1 incluida.

(6) $\{(1 \leq i < k \leq n + 1) \wedge perm(A(1..n), (a_1, a_2, \ldots, a_n)) \wedge$
$particion(A(1..n), \mathbf{i + 1}, k)\}$

En la aserción (6) sabemos que el elemento de la posición i es par y que por tanto no habrá que moverlo de sitio, solo habrá que actualizar la variable i pero como de momento esa variable no ha sido actualizada, se indicará que los pares llegan hasta la i incluida. Esta aserción se ha obtenido modificando la aserción (2), que es su anterior aserción.

Por último, se dará la expresión cota E:

(9) $\{E = k - i\}$
También en este ejemplo E nos indicará (cuando estemos en el punto donde se cumple el invariante) cuántas vueltas quedan por dar exactamente.