METODOLOGÍA DE LA PROGRAMACIÓN
EJERCICIOS DEL TEMA 3
VERIFICACIÓN DE PROGRAMAS

ÍNDICE

a) Asignación y composición secuencial[#]

1. Asignación (s : = s + A(i);)

Verificar si es correcto el siguiente programa:

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

$$s := s + A(i);$$

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

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

2. Asignación (k : = k div 2;)

Verificar si es correcto el siguiente programa:

$$\{\varphi\} \equiv \{par(k) \wedge y * z^{k} = c\}$$

$$k := k \text{ div } 2;$$

$$\{\psi\} \equiv \{y * z^{2 * k} = c\}$$

donde div es la división entera (Ejemplos: 30 div 2 = 15; 9 div 2 = 4).

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

3. Asignación (sin_ceros : = true;)

Verificar si es correcto el siguiente programa:

$$\{\varphi\} \equiv \{n \ge 1 \wedge i = 0\}$$

$$sin\_ceros := true;$$

$$\{\psi\} \equiv \{0 \le i \le n \wedge (sin\_ceros \leftrightarrow \forall k(1 \le k \le i \rightarrow A(k) \ne 0))\}$$

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

[#] Asumir que todas las variables de las que no se especifica tipo de dato (excepto las que son claramente booleanas) son de tipo *integer*, y los vectores (A, B, etc.) son de n elementos y contienen números enteros.

4. Asignación (i : = 1;)

Verificar si es correcto el siguiente programa:

$$\{\varphi\} \equiv \{n \geq 1 \wedge A(1) \neq 0 \wedge sin\_ceros\}$$

$$i := 1;$$

$$\{\psi\} \equiv \{1 \leq i \leq n \wedge (sin\_ceros \leftrightarrow \forall k(1 \leq k \leq i \rightarrow A(k) \neq 0))\}$$

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

5. Asignación (i : = i * j;)

Verificar si es correcto el siguiente programa:

$$\{\varphi\} \equiv \{i \geq j^k \wedge i < w\}$$

$$i := i * j;$$

$$\{\psi\} \equiv \{i = j^{k+1}\}$$

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

6. Asignación (z : = x;)

Verificar si es correcto el siguiente programa:

$$\{\varphi\} \equiv \{x \geq y\}$$

$$z := x;$$

$$\{\psi\} \equiv \{z = maximo(x, y)\}$$

La función **maximo(x, y)** devolverá como resultado x si se cumple $x \geq y$, en cambio, devolverá y si se cumple $x < y$.

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

7. Asignación (m : = A(i + 1);)

Verificar si es correcto el siguiente programa:

$$\{\varphi\} \equiv \{m = maximo(A(1..i)) \wedge (1 \leq i \leq n-1) \wedge A(i+1) > m\}$$

$$m := A(i+1);$$

$$\{\psi\} \equiv \{m = maximo(A(1..i+1)) \wedge (1 \leq i \leq n-1)\}$$

La función **maximo(Q(1..r))** devuelve el mayor elemento del vector Q(1..r). Por ejemplo, maximo((4, 0, 10, 6)) es 10.

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

8. Asignación (noceros : = neg + pos;)

Verificar si es correcto el siguiente programa:

$$\{\varphi\} \equiv \{neg = N\ i\ (1 \leq i \leq n \wedge A(i) < 0) \wedge pos = N\ i\ (1 \leq i \leq n \wedge A(i) > 0)\}$$

$$noceros := neg + pos;$$

$$\{\psi\} \equiv \{noceros = N\ i\ (1 \leq i \leq n \wedge A(i) \neq 0)\}$$

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

9. Composición secuencial (s : = s + A(i); i : = i + 1;)

Verificar si es correcto el siguiente programa:

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

$$s := s + A(i);$$
$$i := i + 1;$$

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

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

10. Composición secuencial (k : = k div 2; z : = z * z;)

Verificar si es correcto el siguiente programa:

$\{\varphi\} \equiv \{par(k) \wedge y * z^k = c\}$

k : = k div 2;
z : = z * z;

$\{\psi\} \equiv \{y * z^k = c\}$

donde div es la división entera (Ejemplos: 30 div 2 = 15; 9 div 2 = 4).

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

11. Composición secuencial (k : = k + 1; i : = i * j;)

Verificar si es correcto el siguiente programa:

$\{\varphi\} \equiv \{i = j^k\}$

k : = k + 1;
i : = i * j;

$\{\psi\} \equiv \{i = j^k\}$

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

b) Iteraciones[#]

1. Programa que calcula en z el producto de x e y, siendo x e y dos enteros no negativos

Verificar si el siguiente programa es correcto con respecto a la especificación pre-post y al invariante dados. Según la especificación pre-post, el programa debería calcular en z el producto de x e y, siendo x e y dos enteros no negativos:

{φ} ≡ {x = *a* ≥ 0 ∧ y ≥ 0}
z : = 0;
while {INV} x ≠ 0 loop
    z : = z + y;
    x : = x – 1;
end loop;
{ψ} ≡ {z = *a* * y}
	{INV}≡ {0 ≤ x ≤ *a* ∧ z = (*a* – x) * y }
	
	E = x

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

2. Programa que decide en la variable booleana prod si los elementos de B(1..n) son los valores de A(1..n) multiplicados por x

Verificar si el siguiente programa es correcto con respecto a la especificación pre-post y al invariante dados. Según la especificación pre-post, el programa debería decidir en la variable booleana prod si los elementos de B(1..n) son los valores de A(1..n) multiplicados por x:

```
{φ} ≡ {n ≥ 1 ∧ prod}
i : = 1;
while {INV} i ≤ n and prod loop
      prod : =(A(i) * x = B(i));
      i : = i + 1;
end loop;
{ψ} ≡ {prod ↔ ∀k(1 ≤ k ≤ n → B(k) = A(k) * x)}
```

{INV}≡ {(1 ≤ i ≤ n + 1) ∧ (prod ↔ ∀k(1 ≤ k ≤ i – 1 → B(k) = A(k) * x))}
E = n + 1 – i

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

[#] Asumir que todas las variables de las que no se especifica tipo de dato (excepto las que son claramente booleanas) son de tipo *integer*, y los vectores (A, B, etc.) son de n elementos y contienen números enteros.

3. Programa que decide en la variable booleana sum si los elementos de C(1..n) son la suma de los de A(1..n) y B(1..n)

Verificar si el siguiente programa es correcto con respecto a la especificación pre-post y al invariante dados. Según la especificación pre-post, el programa debería decidir en la variable booleana sum si los elementos de C(1..n) son la suma de los de A(1..n) y B(1..n):

{φ} ≡ {n ≥ 1 ∧ i = 1}
sum : = true;
<u>while</u> **{INV}** i ≤ n <u>and</u> sum <u>loop</u>
    sum : = (C(i) = A(i) + B(i));
    i : = i + 1;
<u>end</u> <u>loop</u>;
{ψ} ≡ {sum ↔ ∀k(1 ≤ k ≤ n → C(k) = A(k) + B(k))}

{INV}≡ {(1 ≤ i ≤ n + 1) ∧ (sum ↔ ∀k(1 ≤ k ≤ i – 1 → C(k) = A(k) + B(k)))}
E = n + 1 – i

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

4. Programa que decide en la variable booleana menores si todos los elementos de A(1..n) son menores que los que ocupan la misma posición en B(1..n)

Verificar si el siguiente programa es correcto con respecto a la especificación pre-post y al invariante dados. Según la especificación pre-post, el programa debería decidir en la variable booleana menores si todos los elementos de A(1..n) son menores que los que ocupan la misma posición en B(1..n):

{φ} ≡ {n ≥ 1 ∧ i = 1}
menores : = true;
<u>while</u> **{INV}** i ≤ n <u>and</u> menores <u>loop</u>
    i : = i + 1;
    menores : = (A(i) < B(i));
<u>end</u> <u>loop</u>;
{ψ} ≡ {menores ↔ ∀k(1 ≤ k ≤ n → A(k) < B(k))}

{INV}≡ {(1 ≤ i ≤ n + 1) ∧ (menores ↔ ∀k(1 ≤ k ≤ i – 1 → A(k) < B(k)))}

E = n + 1 – i

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

5. Programa que calcula en x el valor del término s_n de la sucesión de Fibonacci

Verificar si el siguiente programa es correcto con respecto a la especificación pre-post y al invariante dados. Según la especificación pre-post, dado $n \geq 0$ y teniendo las variables x, z y j inicializadas respectivamente a 0, 1 y 1, el programa debería calcular en x el valor del término s_n de la sucesión de Fibonacci ($s_0 = 0$, $s_1 = 1$ y $s_k = s_{k-1} + s_{k-2}$ para $k \geq 2$):

```
{φ} ≡ {n ≥ 1 ∧ x = 0 ∧ z = 1 ∧ j = 1}
while {INV} j ≤ n loop
      z : = z + x;
      x : = z – x;
      j : = j + 1;
end loop;
{ψ} ≡ {x = sn}
```

$\{INV\} \equiv \{(1 \leq j \leq n + 1) \wedge x = s_{j-1} \wedge z = s_j\}$
$E = n + 1 - j$

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

6. Programa que calcula en f el factorial de x

Verificar si el siguiente programa es correcto con respecto a la especificación pre-post y al invariante dados. Según la especificación pre-post, dado $x \geq 0$ y teniendo la variable f inicializada con 1, el programa debería calcular en f el factorial de x:

```
{φ} ≡ {x ≥ 0 ∧ f = 1}
t : = x;
while {INV} t ≥ 1 loop
      t : = t – 1;
      f : = f ∗ t;
end loop;
```
$\{\psi\} \equiv \{f = \prod_{i=1}^{x} i\}$

$\{INV\} \equiv \{(0 \leq t \leq x) \wedge f = \prod_{i=t+1}^{x} i\}$
$E = t$

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

7. Programa que decide en la variable booleana ord si los elementos del vector A(1..n) están en orden creciente

Verificar, utilizando el Cálculo de Hoare, la corrección total del siguiente programa que según la especificación pre-post debería decidir en la variable booleana ord si los elementos del vector A(1..n) están en orden creciente (es decir, si cada elemento es menor o igual que el siguiente):

```
{φ} ≡ {n ≥ 1 ∧ i = 1}
ord : = true;
while {INV} i ≠ n and ord loop
    ord : = (A(i) ≤ A(i + 1));
    i : = i + 1;
end loop;
{ψ} ≡ {ord ↔ creciente(A(1..n))}

{INV} ≡ {(1 ≤ i ≤ n) ∧ (ord ↔ creciente(A(1..i)))}
E = n – i
creciente(C(1..p)) ≡ {∀k(2 ≤ k ≤ p → C(k – 1) ≤ C(k))}
```

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

8. Programa que decide en la variable booleana res si se cumple que para cada valor par del vector A(1..n) en la correspondiente posición de R(1..n) se tiene un 0 y para cada valor impar del vector A(1..n) en la correspondiente posición de R(1..n) se tiene un 1.

Verificar, utilizando el Cálculo de Hoare, la corrección total del siguiente programa que según la especificación pre-post debería decidir en la variable booleana *res* si se cumple que para cada valor par del vector A(1..n) en la correspondiente posición de R(1..n) se tiene un 0 y para cada valor impar del vector A(1..n) en la correspondiente posición de R(1..n) se tiene un 1 (es decir, el programa decide en *res* si en R(1..n) se tienen los restos de dividir los elementos de A(1..n) por 2):

```
{φ} ≡ {n ≥ 1 ∧ res}
i : = 0;
while {INV} i ≠ n and res loop
    res : = (A(i + 1) mod 2 = R(i + 1));
    i : = i + 1;
end loop;
{ψ} ≡ {res ↔ concuerda(A(1..n), R(1..n))}

{INV} ≡ {(0 ≤ i ≤ n) ∧ (res ↔ concuerda(A(1..i), R(1..i)))}
E = n – i
concuerda(D(1..p), F(1..p)) ≡ {∀k(1 ≤ k ≤ p → D(k) mod 2 = F(k))}
mod es el resto de la división entera
(Ejemplos: 10 mod 5 = 0; 10 mod 4 = 2; 10 mod 3 = 1)
```

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

9. **(Junio 2009) Programa que decide en la variable booleana sim si A(1..n) es simétrico.**
Verificar, utilizando el **Cálculo de Hoare**, la corrección total del siguiente programa que según la especificación pre-post, dado un vector A(1..n), debería decidir en la variable booleana *sim* si A(1..n) es simétrico. Un vector A(1..n) es simétrico si ocurre que al poner los elementos de A(1..n) en orden inverso, el nuevo vector es igual al vector original. Por ejemplo, el vector (1, 8, 5, 8, 1) es simétrico y también (7, 2, 2, 7) es simétrico:

{φ} ≡ {n ≥ 1 ∧ i = 0}
sim : = true;
while {INV} i ≠ (n div 2) **and** sim **loop**
    i : = i + 1;
    sim : = (A(i) = A(n – i + 1));
end loop;
{ψ} ≡ {sim ↔ ∀k(1 ≤ k ≤ n div 2 → A(k) = A(n – k + 1))}

{INV} ≡ {(0 ≤ i ≤ n div 2) ∧ (sim ↔ simetrico(A(1..n), i))} E = n div 2 – i simetrico(A(1..n), pos) ≡ ∀k(1 ≤ k ≤ pos → A(k) = A(n – k + 1)) **div** es la división entera (Ejemplos: 30 div 2 = 15; 9 div 2 = 4)

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

10. **(Septiembre 2009) Programa que decide en la variable booleana mult si A(1..n) es B(1..n) multiplicado por x.**
Verificar, utilizando el **Cálculo de Hoare**, la corrección total del siguiente programa que según la especificación pre-post, dados dos vectores A(1..n) y B(1..n) y un número x, debería decidir en la variable booleana *mult* si A(1..n) es B(1..n) multiplicado por x. Por ejemplo, el vector (3, 12, 9, 15, 15) es (1, 4, 3, 5, 5) multiplicado por 3:

{φ} ≡ {n ≥ 1 ∧ i = 1}
mult : = true;
while {INV} i ≠ (n + 1) **and** mult **loop**
    mult : = (A(i) = B(i) * x);
    i : = i + 1;
end loop;
{ψ} ≡ {mult ↔ ∀k(1 ≤ k ≤ n → A(k) = B(k) * x)}

{INV} ≡ {(1 ≤ i ≤ n + 1) ∧ (mult ↔ ∀k(1 ≤ k ≤ i – 1 → A(k) = B(k) * x))} E = n + 1 – i

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

11. (Junio 2010) Programa que decide en la variable booleana multpos si cada elemento de A(1..n) es múltiplo de la posición que ocupa.

Verificar, utilizando el **Cálculo de Hoare**, la corrección total del siguiente programa que según la especificación pre-post, dado un vector A(1..n), debería decidir en la variable booleana *multpos* si cada elemento de A(1..n) es múltiplo de la posición que ocupa. Por ejemplo, en el vector (1, 8, 15, 8, 20) cada elemento es múltiplo de la posición que ocupa (las posiciones son 1, 2, 3, 4 y 5). En cambio, en el vector (1, 8, 7, 8, 20) el tercer elemento no es múltiplo de la posición que ocupa:

{φ} ≡ {n ≥ 1 ∧ i = 0}
multpos : = true;
while {INV} i ≠ n **and** multpos **loop**
    i : = i + 1;
    multpos : = ((A(i) **mod** i) = 0);
end loop;
{ψ} ≡ {multpos ↔ ∀k(1 ≤ k ≤ n → (A(k) mod k) = 0)}

{INV} ≡ {(0 ≤ i ≤ n) ∧ (multpos ↔ ∀k(1 ≤ k ≤ i → (A(k) mod k) = 0))} E = n – i **mod** es el resto de la división entera (Ejemplos: 10 mod 5 = 0; 10 mod 4 = 2; 10 mod 3 = 1)

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.

12. (Septiembre 2010) Programa que decide en la variable booleana mult si algún elemento de A(1..n) es múltiplo de x.

Verificar, utilizando el **Cálculo de Hoare**, la corrección total del siguiente programa que según la especificación pre-post, dado un vector A(1..n), debería decidir en la variable booleana *mult* si algún elemento de A(1..n) es múltiplo de x:

{φ} ≡ {n ≥ 1 ∧ i = 1 ∧ x ≠ 0}
mult : = false;
while {INV} i ≠ n + 1 **and not** mult **loop**
    i : = i + 1;
    mult : = (A(i – 1) **mod** x = 0);
end loop;
{ψ} ≡ {mult ↔ ∃k(1 ≤ k ≤ n ∧ A(k) mod x = 0)}

{INV} ≡ {x ≠ 0 ∧ (1 ≤ i ≤ n + 1) ∧ (mult ↔ ∃k(1 ≤ k ≤ i – 1 ∧ A(k) mod x = 0))} E = n + 1 – i **mod** es el resto de la división entera (Ejemplos: 10 mod 5 = 0; 10 mod 4 = 2; 10 mod 3 = 1)

En caso de que sea correcto hay que **dar la demostración formal** de la corrección.