1. En cada uno de los siguientes programas, anote los valores que las variables toman a medida que se ejecutan.

a)
Var x : Int;
[σ 0 : (x → 1)]
x := 5
[σ 1 : (x → 5)]

b)
Var x, y : Int;
[σ 0 : (x → 2, y → 5)]
x := x + y;
[σ 1 : (x → 7, y → 5)]
y := y + y
[σ 2 : (x → 7, y → 10)]

c)
Var x, y : Int;
[σ 0 : (x → 2, y → 5)]
y := y + y;
[σ 1 : (x → 2, y → 10)]
x := x + y
[σ 2 : (x → 12, y → 10)]

d)
Var x, y : Int;
[σ 0 : (x → 2, y → 5)]
y, x := y + y, x + y
[σ 1 : y → 10, x → 7]

e)
Var x, y : Int;
[σ 0 : (x → 3, y → 1)]
if x ≥ y →
[σ 1 : (x → 3, y → 1)]
x := 0
[σ 2 : (x → 0, y → 1)]
[ ] x ≤ y →
[σ ′ 1 :]
x := 2
[σ ′ 2 :]
fi
[σ 3 : (x → 0, y → 1)]

f)
Var x, y : Int;
[σ 0 : (x → −100, y → 1)]
if x ≥ y →
[σ1 :]
x := 0
[σ2 :]
x ≤ y →
[σ ′ 1 : (x → −100, y → 1)]
x := 2
[σ ′ 2 : (x → 2, y → 1)]
fi
[σ ′ 3 : (x → 2, y → 1)]

g)
Var x, y : Int;
[σ 0 : (x → 1, y → 1)]
if x ≥ y →
[σ 1 : (x → 1, y → 1)]
x := 0

[σ 2 : (x → 0, y → 1]

x ≤ y →

[σ ′ 1 :]
x := 2
[σ ′ 2 :]
fi

[σ 3 :(x → 0, y → 1) , σ′ 3 :]

h)
Var i : Int;

[σ 0 : (i → 4)]

do i/= 0 →

[σ 1 1 (i → 4) : , σ 2 1 : (i → 3) , σ 3 1 : (i → 2), σ 4 1 : (i → 1)]

i := i − 1

[σ 1 2 :(i → 3) , σ 2 2 : (i → 2) , σ 3 2 : (i → 1), σ 4 2 : (i → 0)]

od

[σ 4 : (i → 0)]

i)
Var i : Int;

[σ 0 : (i → 400)]

do i/= 0 →

[σ 1 1 : (i → 400) , σ 2 1 : , · · ·]

i := 0

[σ 1 2 : (i → 0) , σ 2 2 : , · · ·]

od

[σ 3 : (i → 0)]

j)
Var i : Int;

[σ 0 : (i → 4)]

do i < 0 →

[σ 1 1 : , σ2 1 : , · · ·]

i := i − 1

[σ 1 2 : , σ2 2 : , · · ·]
od

[σ 3 : (i → 4)]

k)
Var i : Int;

[σ 0 : (i → 0)]

do i ≤ 0 →

[σ 1 1 : (i → 0) , σ 2 1 : (i → -1) , σ 3 1 : (i → -2), ⋯. -inf] ⋆

i := i − 1

[σ 1 2 : (i → -1) , σ 2 2 :(i → -2) , σ 3 2 : (i → -3)…. -inf] ⋆′

od
[σ3 : i e (-inf,0]]

l)
Var r : Int;

[σ 0 : (r → 3)]

do r/= 0 →

[σ 1 1 : (r → 3), σ 2 1 : (r → 2), σ 3 1 : (r → 1)]

if r < 0 →

[ ]
r := r + 1
[ ]

[] r > 0 →

[σ 1 2 : (r → 3), σ 2 2 : (r → 2), σ 3 2 : (r → 1)]

r := r − 1

[σ 1 3 : (r → 2), σ 2 3 : (r → 1), σ 3 3 : (r → 0)]

fi

[σ 1 4 : (r → 2), σ 2 4 : (r → 1), σ 3 4 : (r → 0)]

od

[σ 4 : (r → 0)]

2. Responda las siguientes preguntas respecto al ejercicio anterior:

a) ¿Que se puede concluir de los ıtems 1b, 1c y 1d? ¿Que tienen en común? ¿Cuáles son las diferencias en la ejecución de cada uno de ellos?

Podemos concluir que los 3 ítems toman un estado inicial el cual es cambiado por sentencias produciendo un nuevo estado. Los tres ítems son similares, entre el ítem b y c cambia el orden de las asignaciones produciendo un estado diferente en cada ítem, luego, en ítem d solo contiene una sentencia la cual es una asignación doble en donde el estado final es diferente a los anteriores.

b) ¿Se puede escribir un programa equivalente al del ıtem 1c con solo una asignación múltiple? Justifique.

No podemos escribir un programa equivalente. Dado que en las asignaciones primero se revuelven las expresiones del lado derecho y luego se asigna, en estos casos los estados que se toma en la asignación múltiple son diferentes a los estados que toma el ítem b.

c) ¿Qué sucederá si en el ıtem 1g se utilizarán las guardas “x > y” y “x < y”?

El programa no ejecuta el condicional y el estado inicial no cambia.

d) Con respecto al programa del ıtem 1k: ¿Existen predicados que caracterizan exactamente todos los valores que puede tomar la variable i cuando el mismo se encuentra en ⋆ y en ⋆′ ? Si la respuesta es sı, escríbalo. Es más, si existen, ¿cuál es la ventaja con respecto a enumerar todos los estados? ¿La desventaja?

Si, existen predicados que caracterizan los valores que puede tomar i. El mismo podría ser { i ∈ (-inf, -1] } en ⋆′ y { i ∈ (-inf, 0] } en ⋆.

La ventaja con respecto a enumerar en este caso es que, como el programa nunca termina, la variable i toma infinitos valores imposibles de enumerar, la desventaja es que tenemos que contemplar todos los estados posibles en ese punto del programa para hacer un predicado correcto.

3. En cada uno de los siguientes programas, anote los valores de las expresiones que se mencionan en la tabla respectiva, de acuerdo a cómo cambian los estados a medida que se ejecutan. Describa qué hace cada programa, en los términos más generales que encuentre

$a)$ Const $A : array[0,4)\ of Int;$
Var $i, s : Int;$
$[\![\sigma_0 : (i \mapsto -3, s \mapsto 5, A \mapsto \boxed{2 \mid 10 \mid 10 \mid -1})]\!]$
$i, s := 0, 0;$
$[\![\sigma'_0]\!]$
do $i < 4 \rightarrow$
$[\![\sigma_1^0, \cdots, \sigma_1^3]\!]$
$s, i := s + A.i, i + 1$
$[\![\sigma_2^0, \cdots, \sigma_2^3]\!]$
od
$[\![\sigma_3]\!]$

Estado	i	s
σ_0		
σ'_0		
σ_1^0		
σ_2^0		
σ_1^1		
σ_2^1		
σ_1^2		
σ_2^2		
σ_1^3		
σ_2^3		
σ_3		

El programa calcula la suma de los valores dentro de un Array y devuelve un estado con ese resultado en la variable s.

{ σ 0 : (i → -3, s → 5) }

{ σ ′ 0 : (i → 0, s → 0) }

{ σ 0 1 : (i → 0, s → 0) }

{ σ 0 2 : (i → 1, s → 2) }

{ σ 1 1 : (i → 1, s → 2) }

{ σ 1 2 : (i → 2, s → 12) }

{ σ 2 1 : (i → 2, s → 12) }

{ σ 2 2 : (i → 3, s → 22) }

{ σ 3 1 : (i → 3, s → 22) }

{ σ 3 2 : (i → 4, s → 21) }

{ σ 3 : (i → 4, s → 21) }

b) Const $A : array[0,4)\ of Int$;
Var $i, c : Int$;
$[\![\sigma_0 : (i \mapsto 3, c \mapsto 12, A \mapsto \boxed{12 \mid -9 \mid 10 \mid -1})]\!]$
$i, c := 0, 0$;
$[\![\sigma'_0]\!]$
do $i < 4 \rightarrow$
    $[\![\sigma_1^0, \cdots, \sigma_1^3]\!]$
    if $A.i > 0 \rightarrow$
        $c := c + 1$
    ▯ $A.i \leq 0 \rightarrow$
        skip
    fi
    $[\![\sigma_2^0, \cdots, \sigma_2^3]\!]$
    $i := i + 1$
    $[\![\sigma_3^0, \cdots, \sigma_3^3]\!]$
od
$[\![\sigma_4]\!]$

Estado	i	$A.i$	c
σ_0			
σ'_0			
σ_1^0			
σ_2^0			
σ_3^0			
σ_1^1			
σ_2^1			
σ_3^1			
σ_1^2			
σ_2^2			
σ_3^2			
σ_1^3			
σ_2^3			
σ_3^3			
σ_4			

El programa cuenta los valores positivos dentro de un Array.

{ σ 0 : (i → 3, c → 12, A → [12,-9,10,-1]) }

{ σ ′ 0 : (i → 0, c → 0, A → [12,-9,10,-1]) }

{ σ 0 1 : (i → 0, c → 0, A → [12,-9,10,-1]) }

{ σ 0 2 : (i → 0, c → 1, A → [12,-9,10,-1]) }

{ σ 0 3 : (i → 1, c → 1, A → [12,-9,10,-1]) }

{ σ 1 1 : (i → 1, c → 1, A → [12,-9,10,-1]) }

{ σ 1 2 : (i → 1, c → 1, A → [12,-9,10,-1]) }

{ σ 1 3 : (i → 2, c → 1, A → [12,-9,10,-1]) }

{ σ 2 1 : (i → 2, c → 1, A → [12,-9,10,-1]) }

{ σ 2 2 : (i → 2, c → 2, A → [12,-9,10,-1]) }

{ σ 2 3 : (i → 3, c → 2, A → [12,-9,10,-1]) }

{ σ 3 1 : (i → 3, c → 2, A → [12,-9,10,-1]) }

{ σ 3 2 : (i → 3, c → 2, A → [12,-9,10,-1]) }

{ σ 3 3 : (i → 4, c → 2, A → [12,-9,10,-1]) }

{ σ 4 : (i → 4, c → 2, A → [12,-9,10,-1]) }

4. Responda las siguientes preguntas sobre los programas del ejercicio anterior:
a) ¿Cómo modificaría el programa del ıtem 3a para que calcule el promedio de los valores en el arreglo A?
b) ¿Cómo modificaría el programa del ıtem 3b para que solo tenga en cuenta las posiciones pares del arreglo A?

a)

```
Const A : array[0, 4) of Int;
Var i, s : Int;
Var prom : Float;

[ σ 0 : (i → −3, s → 5, A → [2 ,10 ,10 , − 1]), prom → 7.2]
i, s := 0, 0;
prom := 0.0;

[σ ′ 0 ]
   do i < 4 →
[σ 0 1 , · · · , σ3 1 ]
      s, i := s + A.i, i + 1
[σ : 0 2 , · · · , σ3 2 ]
   od
prom := s / 4;
[σ3 : ]
```

b)

```
Const A : array[0, 4) of Int;
Var i, c : Int;

[ σ 0 : (i → 3, c → 12, A → [12, −9, 10, −1] )]
i, c := 0, 0;
[σ ′ 0 ]
   do i < 4 →
[σ 0 1 , · · · , σ3 1 ]
      if ( i mod 2 = 0) →
```

```
            c := c + 1
          [] ( i mod 2 != 0) →
            skip
          fi
      [σ 0 2 , · · · , σ3 2 ]
      i := i + 1
      [σ 0 3 , · · · , σ3 3 ]
         od
      [σ4 ]
```

5.

a) Escriba un programa que calcule N!, donde N ≥ 0 es una constante de tipo Int.

b) Escriba otro programa que calcule N! efectuando las multiplicaciones en un orden diferente.

c) Para N = 5, ejecute manualmente ambos programas y escriba las tablas de estados.

a)

```
Const N : Int;
Var factorial, i : Int;
factorial := 1;
i := 1;
{ N >= 0, factorial = 1, i = 1}
do i <= N →
   factorial := factorial * i;
   i := i + 1
od
{∏ i : 1 <= i <= N : i }
```

[σ 0 : (i → 1, factorial → 1, N → 5)]

[σ ' 0 : (i → 1, factorial → 1, N → 5)]

[σ 0 1 : (i → 1, factorial → 1, N → 5)]

[σ 0 2 : (i → 2, factorial → 1, N → 5)]

[σ 1 1 : (i → 2, factorial → 1, N → 5)]

[σ 1 2 : (i → 3, factorial → 2, N → 5)]

[σ 2 1 : (i → 3, factorial → 2, N → 5)]

[σ 2 2 : (i → 4, factorial → 6, N → 5)]

[σ 3 1 : (i → 4, factorial → 6, N → 5)]

[σ 3 2 : (i → 5, factorial → 24, N → 5)]

[σ 4 1 : (i → 5, factorial → 24, N → 5)]

[σ 4 2 : (i → 6, factorial → 120, N → 5)]   [σ 5 : (i → 6, factorial → 120, N → 5)]

b)

```
Const N : Int;
Var factorial, i : Int;
factorial, i := 1, N;
{ N >= 0, factorial = 1, i = N}
do (i <= N) ^ (i > 0) →
   factorial := factorial * i;
   i := i - 1
od
{∏ i : 1 <= i <= N : i }
```

[σ 0 : (i → 5, factorial → 1, N → 5)]

[σ ' 0 : (i → 5, factorial → 1, N → 5)]

[σ 0 1 : (i → 5, factorial → 1, N → 5)]

[σ 0 2 : (i → 4, factorial → 5, N → 5)]

[σ 1 1 : (i → 4, factorial → 5, N → 5)]

[σ 1 2 : (i → 3, factorial → 20, N → 5)]

[σ 2 1 : (i → 3, factorial → 20, N → 5)]

[σ 2 2 : (i → 2, factorial → 60, N → 5)]

[σ 3 1 : (i → 2, factorial → 60, N → 5)]

[σ 3 2 : (i → 1, factorial → 120, N → 5)]

[σ 4 1 : (i → 1, factorial → 120, N → 5)]

[σ 4 2 : (i → 0, factorial → 120, N → 5)]   [σ 5 : (i → 0, factorial → 120, N → 5)]

6.

a) Escriba un programa que sume, por un lado, los valores positivos y, por otro, los valores múltiplos de 3 de un arreglo A de tamaño N.

b) Para N = 5, A = 12 −9 10 0 −1 , ejecute manualmente el programa y escriba la tabla de estados

a)

```
Const N : Int;
Const A : array[0, N) of Int
Var pos, mult3, i : Int;
pos, mult3, i := 0,0,0;
inicial
do (i < N) →
   if (A.i mod 3 = 0) →
      mult3 := mult3 + A.i;
   [ ] (A.i mod 3 != 0) →
      skip
   fi

   if  (A.i >= 0) →
      pos := pos + A.i;
   [ ] (A.i < 0) →
      skip
   fi
   i := i + 1
od
```

b) Para N = 5, A = [12 , − 9 ,10 ,0 , − 1] , ejecute manualmente el programa y escriba la tabla de estados.

{ σ 0 : (N → 5, i → 0, pos → 0, mult3 → 0, A → [12,-9,10,0,-1]) }

{ σ 0' : (N → 5, i → 0, pos → 0, mult3 → 0, A → [12,-9,10,0,-1]) }

{ σ 0' 1 : (N → 5, i → 0, pos → 0, mult3 → 12, A → [12,-9,10,0,-1]) }

{ σ 0' 2 : (N → 5, i → 0, pos → 12, mult3 → 12, A → [12,-9,10,0,-1]) }

{ σ 0' 3 : (N → 5, i → 1, pos → 12, mult3 → 12, A → [12,-9,10,0,-1]) }

{ σ 1 1 : (N → 5, i → 1, pos → 12, mult3 → 12, A → [12,-9,10,0,-1]) }

{ σ 1 2 : (N → 5, i → 1, pos → 12, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 1 3 : (N → 5, i → 1, pos → 12, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 1 4 : (N → 5, i → 2, pos → 12, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 2 1 : (N → 5, i → 2, pos → 12, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 2 2 : (N → 5, i → 2, pos → 12, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 2 3 : (N → 5, i → 2, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 2 4 : (N → 5, i → 3, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 3 1 : (N → 5, i → 3, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 3 2 : (N → 5, i → 3, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 3 3 : (N → 5, i → 3, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 3 4 : (N → 5, i → 4, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 4 1 : (N → 5, i → 4, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 4 2 : (N → 5, i → 4, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 4 3 : (N → 5, i → 4, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

{ σ 4 4 : (N → 5, i → 5, pos → 22, mult3 → 3, A → [12,-9,10,0,-1]) }

7. a) Escriba un programa que, dados dos arreglos A y B de tamaño N y M respectivamente, calcule cuántas veces coinciden dos elementos entre ellos.

Const N, M : Int;
Const A : array[0,N) of Int;
Const B : array[0,M) of Int;
var result, i, j : Int;

{N>0 ^ M>0}

```
result, i, j := 0, 0, 0;
—----------------------  i
do(i<N) →
—-------------------  A
   do(j<M) →
+++++++++++++++++ AA
      if(A.i = B.j) →
         result := result + 1;
      [] else(A.i != B.j) →
         skip
      fi
+++++++++++++++ B 1
      j := j + 1;
++++++++++++++++++ C 2
   od
—----------------- B
   i =: i + 1;
   j := 0;
—---------------- C
od
```

{result = ⟨N i : o <= i < N : ⟨E j : 0 <= j < M : A.i = B.j ⟩ ⟩ }

b) Para A = [8, 0, 8] , B = [0, 8] , ejecute manualmente el programa y escriba la tabla de estados.

{ σ 0 : (N → 3, M → 2, i → 0, j → 0, result → 0, A → [8, 0, 8], B → [0, 8]) }

{ σ 0' : (N → 3, M → 2, i → 0, j → 0, result → 0, A → [8, 0, 8], B → [0, 8]) }

{ σ 0'' : (N → 3, M → 2, i → 0, j → 0, result → 0, A → [8, 0, 8], B → [0, 8]) }

{ σ 0'' 1 : (N → 3, M → 2, i → 0, j → 0, result → 0, A → [8, 0, 8], B → [0, 8]) }

{ σ 0'' 2 : (N → 3, M → 2, i → 0, j → 1, result → 0, A → [8, 0, 8], B → [0, 8]) }

{ σ 1'' : (N → 3, M → 2, i → 0, j → 1, result → 0, A → [8, 0, 8], B → [0, 8]) }

{ σ 1'' 1 : (N → 3, M → 2, i → 0, j → 1, result → 1, A → [8, 0, 8], B → [0, 8]) }

{ σ 1'' 2 : (N → 3, M → 2, i → 0, j → 2, result → 1, A → [8, 0, 8], B → [0, 8]) }

{ σ 0' 1 : (N → 3, M → 2, i → 0, j → 2, result → 1, A → [8, 0, 8], B → [0, 8]) }

{ σ 0' 2 : (N → 3, M → 2, i → 1, j → 0, result → 1, A → [8, 0, 8], B → [0, 8]) }

{ σ 1' : (N → 3, M → 2, i → 1, j → 0, result → 1, A → [8, 0, 8], B → [0, 8]) }

{ σ 1'' : (N → 3, M → 2, i → 1, j → 0, result → 1, A → [8, 0, 8], B → [0, 8]) }

{ σ 1'' 1 : (N → 3, M → 2, i → 1, j → 0, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 1'' 2 : (N → 3, M → 2, i → 1, j → 1, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 2'' : (N → 3, M → 2, i → 1, j → 1, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 2'' 1 : (N → 3, M → 2, i → 1, j → 1, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 2'' 2 : (N → 3, M → 2, i → 1, j → 2, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 1' 1 : (N → 3, M → 2, i → 1, j → 2, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 1' 1 : (N → 3, M → 2, i → 2, j → 0, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 2' : (N → 3, M → 2, i → 2, j → 0, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 2'' : (N → 3, M → 2, i → 2, j → 0, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 2'' 1 : (N → 3, M → 2, i → 2, j → 0, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 2'' 2 : (N → 3, M → 2, i → 2, j → 1, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 3'' : (N → 3, M → 2, i → 2, j → 1, result → 2, A → [8, 0, 8], B → [0, 8]) }

{ σ 3'' 1 : (N → 3, M → 2, i → 2, j → 1, result → 3, A → [8, 0, 8], B → [0, 8]) }

{ σ 3'' 2 : (N → 3, M → 2, i → 2, j → 2, result → 3, A → [8, 0, 8], B → [0, 8]) }

{ σ 2' 1 : (N → 3, M → 2, i → 2, j → 2, result → 3, A → [8, 0, 8], B → [0, 8]) }

{ $\sigma 2' 2$: (N → 3, M → 2, i →3, j → 0, result → 3, A → [8, 0, 8], B → [0, 8]) }

Estado final
{ $\sigma 2' 2$: **(N → 3, M → 2, i →3, j → 0, result → 3, A →** [8, 0, 8], **B → [0, 8]) }**

Ternas de Hoare y Weakest Precondition

8. Decidir si los siguientes programas anotados como ternas de Hoare son correctos (equivalentes a True) o incorrectos (equivalentes a False).

a)
Var x : Int;
{x > 0}
x := x ∗ x
{True}

es equivalente a True ya que S siempre termina

b)
Var x : Int;
{x ⊭ 100}
x := x ∗ x
{x ≥ 0}

asumimos P = {x ⊭ 100}

wp.(x := x ∗ x).(x ≥ 0)
≡ { def wp.asignación
(x ≥ 0).(x := x ∗ x)
≡ { sustituimos
x * x ≥ 0
≡ { lógica
true

c)
Var x : Int;
{x > 0 ∧ x < 100}
x := x ∗ x
{x ≥ 0}

asumimos P = x > 0 ∧ x < 100

wp.(x := x ∗ x).(x ≥ 0)
≡ { def de wp asignación
(x ≥ 0).(x := x ∗ x)
≡ { sustitución
x * x ≥ 0
≡ { lógica
true

d)
Var x : Int;

{x > 0}
x := x ∗ x
{x < 0}

asumimos P = x > 0

wp.(x := x ∗ x).(x < 0)

≡{ def de wp asignación

(x < 0).(x := x ∗ x)

≡{ sustituimos

(x * x < 0)

≡{ divido por x y tenemos ¬P entonces es false, o lógica.

false

contraejemplo:

estado inicial : x → 3 (cumple P)

estado final: x → 9 (no cumple Q)

e)
Var x : Int;
{x < 0}
x := x ∗ x
{x < 0}

asumimos P = x < 0

wp.(x := x ∗ x).(x < 0)

≡{ def de wp asignación

(x < 0).(x := x * x)

≡{ sustituimos

x * x < 0

≡{ dividido por x, da vuelta la desigualdad x < 0

x > 0

≡{ ¬P

false

contraejemplo:

estado inicial : x → -3 (cumple P)

estado final: x → 9 (no cumple Q)

f)
Var x : Int;
{True}
x := x ∗ x
{**x** ≥ **0**}

equivalente a true, siempre y cuando cualquier estado inicial luego de ejecutar S valga Q

asumimos P = True

wp.(x := x * x).(x ≥ 0)

≡{ def wp

x * x ≥ 0

≡{lógica

true

9. Responda las siguientes preguntas en función del ejercicio anterior:

a) ¿Es útil la anotación del programa (8a)?¿Por qué? Es útil solo cuando no nos interesa un estado final Q, ya que el programa se ejecutará siempre y cuando S termine y no tenga error sin importar el estado inicial.

b) Descartando los programas incorrectos, ¿Cual programa tiene la precondición más débil? ¿Cuál tiene la precondición más fuerte? Explíquelo con sus propias palabras.

El programa 8)f) es aquel con la precondición más débil P= true ≡ wp.S.Q. Si tenemos P' tq (P' → P) estamos diciendo que P' es más fuerte que P y como {P} S {Q} vale, por definición P' → wp.S.Q ≡ {P'} S {Q}

El programa 8)c) es aquel con la precondición más fuerte P= {x > 0 ∧ x < 100} ya que a diferencia de los demás programas este solo admite estados iniciales finitos, mientras que los otros admiten infinitos estados. Por ende, una precondición es más fuerte si exige más cosas, es decir menos posibilidades.

c) Con respecto a la última pregunta, ¿se puede definir alguna otra precondición tal que esta sea aún más débil?

No, la precondición P = {True} es la más débil posible, y la precondición P = {False} es la más fuerte.

d) En general, ¿que implica tener un "{T rue}" al principio de un programa? ¿Al final?

Implica que cualquier estado inicial satisface True, pero debe de satisfacer Q luego de ejecutar S.
Tener un {True} al final nos dice que cualquier estado final lo satisface siempre y cuando S haya terminado y no de error.

10. Para cada uno de los siguientes programas, calcule la precondición más débil y las anotaciones intermedias. Para ello utilice el transformador de predicados wp.

a)
Var x, y : Num;
{ }
x := x + y
{x = 6 ∧ y = 5}

wp.(x := x + y).(x = 6 ^ y = 5)

≡{ def de wp asignación

(x=6^y=5).(x:= x + y)

≡{ sustituimos

x + y = 6 ^ y = 5

≡{ lógica

x + 5 = 6 ^ y = 5

≡{ aritmética

x = 1 ^ y = 5

finalmente
Var x, y : Num;

{x = 1 ^ y = 5 }
x := x + y
{**x** = **6** ∧ **y** = **5**}

b)
Var x : Num;
{ }
x := 8
{x = 8}

wp.(x:=8).(x=8)
≡{ def de wp asignación
(x=8).(x:=8)
≡{ sustituimos
8=8
≡{ reflexividad
true

finalmente
Var x : Num;
{true}
x := 8
{x = 8}

c) Var x : Num;
{ }
x := 8
{x = 7}

wp.(x := 8).(x = 7)
≡{ def de wp asignación
(x = 7).(x := 8)
≡{ sustitución
8 = 7
≡{
false

finalmente
{ false }
x := 8
{x = 7}

d) Var x, y : Num;
{ }
x, y := y, x
{**x** = **B** ∧ **y** = **A**}

wp.(x, y := y, x).(x = B ^ y = A)
≡{ def de wp asignación
(x = B ^ y = A).(x, y := y, x)
≡{ sustituimos
(y = B ^ x = A)

finalmente
Var x, y : Num;
{y = B ^ x = A}
x, y := y, x
{**x** = **B** ∧ **y** = **A**}

e) Var x, y, a : Num;
{ }
a, x := x, y;
{ }
y := a
{**x** = **B** ∧ **y** = **A**}

wp.(y:=a).(x=B ^ y=A)
≡{ def wp asignación
(x=B ^ y=A).(y:=a)
≡{ sustituimos
(x=B ^ a=A)

Var x, y, a : Num;
{ }
a, x := x, y;
{x = B ^ a = A }
y := a
{**x** = **B** ∧ **y** = **A**}

wp.(a, x := x, y).(x=B ^ a=A)
≡{ def wp asignación
(x=B ^ a=A).(a, x := x, y)
≡{ sustituimos
(y=B ^ x=A)

finalmente
Var x, y, a : Num;
{y=B ^ x=A}
a, x := x, y;
{x = B ^ a = A }
y := a
{**x** = **B** ∧ **y** = **A**}

f)
Var x, y : Num;
{ }
if x ≥ y →
  { }
  x := 0
  { }
x ≤ y →
  { }
  x := 2
  { }
fi

{(x = 0 ∨ x = 2) ∧ y = 1}

B0 = x ≥ y S0 x := 0 debemos demostrar que B0 → wp.S0.Q y esto es

asumimos B0 = x ≥ y

wp.(x := 0).((x = 0 ∨ x = 2) ∧ y = 1)

≡{ def de wp asignación

((x = 0 ∨ x = 2) ∧ y = 1).(x := 0)

≡{ sustituimos

((0 = 0 ∨ 0 = 2) ∧ y = 1)

≡{ logica

((True ∨ False) ∧ y = 1)

≡{ absorb. disy, neutro conjunción

y = 1

B1 = x ≤ y S1 = x := 2 debemos demostrar que B1 → wp.S1.Q esto es

asumimos B1 = x ≤ y

wp.(x := 2).((x = 0 ∨ x = 2) ∧ y = 1)

≡{ def de wp asignación

((x = 0 ∨ x = 2) ∧ y = 1).(x := 2)

≡{ sustituimos

((2 = 0 ∨ 2 = 2) ∧ y = 1)

≡{ lógica

((False ∨ True) ∧ y = 1)

≡{ abs. disy, neutro conjunción

y = 1

finalmente la precondición es (y = 1)^(y = 1)

Var x, y : Num;
{y = 1}
if x ≥ y →
{ x = E, y = 1 }
x := 0
{ x = 0, y = 1 }
x ≤ y →
{ x = E, y = 1 }
x := 2
{ x = 2, y = 1 }
fi
{(x = 0 ∨ x = 2) ∧ y = 1}

11. Demuestre que las siguientes ternas de Hoare son correctas. En todos los casos las variables x, y son de tipo Int, y a, b de tipo Bool.

a)
{T rue}
if x ≥ 1 →
x := x + 1
x ≤ 1 →
x := x − 1
fi
{x /= 1}

(i) [**P** ⇒ (**B0** ∨ **B1** ∨ **. . .** ∨ **Bn**)]

(ii) {**P** ∧ **Bi**} **Si** {**Q**} **o, equivalentemente,** [**P** ∧ **Bi** ⇒ **wp.Si .Q**]

ii)
asumimos x ≥ 1
True ^ wp.(x := x + 1).(x /= 1)
≡ { def de wp
True ^ (x /= 1).(x := x + 1)
≡ { sustitución
True ^ x + 1 /= 1
≡ { neutro conjunción
x + 1 /= 1
≡ { aritmética
x /= 0
≡ { asumiendo x ≥ 1
true

asumimos x ≤ 1
True ^ wp.(x := x - 1).(x /= 1)
≡ { neutro ^, def de wp
(x /= 1).(x := x - 1)
≡ { sustitución
x - 1 /= 1
≡ { aritmética
x /= 2
≡ { asumiendo x ≤ 1
true

i)
x ≥ 1 v x ≤ 1
≡ { lógica
x > 1 v x = 1 v x < 1
≡ { tricotomía true

b)
{x /= y}
if x > y →

skip

x < y →

x, y := y, x

fi

{x > y}

(i) [P ⇒ (B0 ∨ B1 ∨ . . . ∨ Bn)]

(ii) {P ∧ Bi} Si {Q} o, equivalentemente, [P ∧ Bi ⇒ wp.Si .Q]

(i)
asumimos (x /= y)
(x > y) v (x < y)
≡ { como x /= y se cumple
true

(ii)
asumimos B0 = (x > y)
(x /= y) ^ wp.skip.(x > y)
≡ { def de wp skip
(x /= y) ^ (x > y)
≡ { asumiendo B0 también (x /= y)
true

asumimos B1 = (x < y)
(x /= y) ^ wp.(x, y := y, x).(x > y)
≡ { def de wp
(x /= y) ^ (x > y).(x, y := y, x)
≡ { sustituimos
(x /= y) ^ (y > x)
≡ { como asumimos B1 también sucede que (x /= y)
true

c)

{T rue}

x, y := y ∗ y, x ∗ x;

{R}

if x ≥ y →

x := x − y

x ≤ y →

y := y − x

fi

{x ≥ 0 ∧ y ≥ 0}

{P} S; T {Q} ≡ existe R tal que {P} S {R} ∧ {R} T {Q}

(i) [P ⇒ (B0 ∨ B1 ∨ . . . ∨ Bn)]

(ii) {P ∧ Bi} Si {Q} o, equivalentemente, [P ∧ Bi ⇒ wp.Si .Q]

Veamos quien es {R} tq:

{R}

if x ≥ y →

x := x − y

x ≤ y →

y := y − x

fi

{x ≥ 0 ∧ y ≥ 0}

Por definición tenemos que R = wp.(if…fi).(Q)

wp.if.Q ≡ [(B0 ∨ B1 ∨ . . . ∨ Bn) ∧ (B0 ⇒ wp.S0.Q) ∧ . . . ∧ (Bn ⇒ wp.Sn.Q)]

Luego

B0 = (x ≥ y) S0 = (x := x − y) Q = (x ≥ 0 ∧ y ≥ 0)

B1 = (x ≤ y) S1 = (y := y − x)

R ≡ [((x ≥ y) v (x ≤ y)) ^ (x ≥ y) → wp.(x := x − y).(x ≥ 0 ∧ y ≥ 0) ^ (x ≤ y) → wp.(y := y − x).(x ≥ 0 ∧ y ≥ 0)]

≡ { lógica

[(x > y) v (x = y) v (x < y) ^ (x ≥ y) → wp.(x := x − y).(x ≥ 0 ∧ y ≥ 0) ^ (x ≤ y) → wp.(y := y − x).(x ≥ 0 ∧ y ≥ 0)]

≡ { tricotomía, neutro ^

[(x ≥ y) → wp.(x := x − y).(x ≥ 0 ∧ y ≥ 0) ^ (x ≤ y) → wp.(y := y − x).(x ≥ 0 ∧ y ≥ 0)]

≡ { def de wp

[(x ≥ y) → (x ≥ 0 ∧ y ≥ 0).(x := x − y) ^ (x ≤ y) → (x ≥ 0 ∧ y ≥ 0).(y := y − x)]

≡ { sustitución

[(x ≥ y) → (x - y ≥ 0 ∧ y ≥ 0) ^ (x ≤ y) → (x ≥ 0 ∧ y - x ≥ 0)]

≡ { aritmética

[(x ≥ y) → (x ≥ y ∧ y ≥ 0) ^ (x ≤ y) → (x ≥ 0 ∧ y ≥ x)]

≡ { debilitamiento a derecha → ^

(x ≥ y) → (x ≥ y) ^ (x ≥ y) → (y ≥ 0) ^ (x ≤ y) → (x ≥ 0) ^ (x ≤ y) → (y ≥ x)

≡ { logica, neutro ^

(x ≥ y) → (y ≥ 0) ^ (x ≤ y) → (x ≥ 0)

≡ { reordenamiento

R = (y ≤ x) → (0 ≤ y) ^ (x ≤ y) → (0 ≤ x)

tenemos que

{T rue} **-P1**

x, y := y ∗ y, x ∗ x; **-S0**

$\{(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x)\}$ -P2

if $x \ge y \rightarrow$ -B1

$x := x - y$ -S1 → S3

$x \le y \rightarrow$ -B2

$y := y - x$ -S2

fi

$\{x \ge 0 \wedge y \ge 0\}$ -Q

queremos ver si se cumplen las ternas {P1} S0 {P2} ^ {P2} S3 {Q}

asumimos P1 = True

$wp.(x, y := y * y, x * x).(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x)$

$\equiv$ { def wp

$(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x).(x, y := y * y, x * x)$

$\equiv$ { sustituimos

$(x * x \le y * y) \rightarrow \underline{(0 \le x * x)} \wedge (y * y \le x * x) \rightarrow \underline{(0 \le y * y)}$

$\equiv$ { lógica

$(x * x \le y * y) \rightarrow true \wedge (y * y \le x * x) \rightarrow true$

$\equiv$ { lógica $\rightarrow$, neutro ^

true

(i) $[P \Rightarrow (B0 \vee B1 \vee ... \vee Bn)]$

(ii) $\{P \wedge Bi\}$ Si $\{Q\}$ o, equivalentemente, $[P \wedge Bi \Rightarrow wp.Si .Q]$

(i) $((y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x)) \rightarrow ((x \ge y) \vee (x \le y))$

$\equiv$ { lógica, tricotomía, lógica implicación

true

ahora asumimos P2 = $(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x)$

B1 = $x \ge y$, S1 = $x := x - y$, Q = $\{x \ge 0 \wedge y \ge 0\}$

B2 = $x \le y$, S2 = $y := y - x$

(ii)

asumiendo $b1 = (x \ge y)$

$(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x) \wedge \underline{wp.(x := x - y).(x \ge 0 \wedge y \ge 0)}$

$\equiv$ { def de wp

$(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x) \wedge \underline{(x \ge 0 \wedge y \ge 0).(x := x - y)}$

$\equiv$ { sustitución

$(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x) \wedge (\underline{x - y \ge 0} \wedge y \ge 0)$

$\equiv$ { aritmética

$(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x) \wedge (x \ge y) \wedge (y \ge 0)$

$\equiv$ { asumimos b1

$[(y \le x) \rightarrow (0 \le y) \wedge (x \le y) \rightarrow (0 \le x)] \wedge [(x \ge y) \rightarrow (x \ge y) \wedge (y \ge 0)]$

---

d)
{True}
if ¬a ∨ b →
a := ¬a
a ∨ ¬b →
b := ¬b
fi
{**a ∨ b**}

(i) [**P ⇒ (B0 ∨ B1 ∨ . . . ∨ Bn**)]
(ii) {**P ∧ Bi**} **Si** {**Q**} **o, equivalentemente,** [**P ∧ Bi ⇒ wp.Si .Q**]

(i)
True → (¬a v b v a v ¬ b)
≡{ conmutatividad v
True → (¬a v a v b v ¬ b)
≡{ 3ro ex, absorvente v
True → True
≡{ lógica
True

(ii)
True ^ ¬a ∨ b → wp.(a := ¬a).(a ∨ b)
≡{ def de wp
True ^ ¬a ∨ b → (a ∨ b).(a := ¬a)
≡{ sustitución
True ^ ¬a ∨ b → (¬a ∨ b)
≡{ p → p, neutro conjunción
true
^
true ^ a ∨ ¬b → wp.(b := ¬b).(a ∨ b)
≡{ def wp, sustitución
true ^ a v ¬b → a v ¬b
≡{ p → p y neutro conjunción
true
≡{ neutro conjunción
true

e)
{**N ≥ 0**}
x := 0;
{ }
do x ≠ N →
x := x + 1
od
{x = N}

tenemos {P} S;T {Q} y debemos encontrar un R tal que {P} S {R} ^ {R} T {Q}
para eso veamos la wp.T.Q la cual es equivalente a R. Pero como no sabemos como calcular wp.do⋯od.Q proponemos R = {x = 0 ^ N ≥ 0} y demostremos la terna {P} S {R}

N ≥ 0 → wp.(x := 0).(x = 0 ^ N ≥ 0)

≡{ def de wp. sustituimos

N ≥ 0 → (0 = 0 ^ N ≥ 0)

≡{ reflexividad, y p → p

true

ahora, para demostrar la terna {R} T {Q} proponemos la invariante I = 0 ≤ x ≤ N

a) **I vale inicialmente?**
   verifiquemos {N ≥ 0} x := 0; {0 ≤ x ≤ N}

   N ≥ 0 → wp.(x := 0).(0 ≤ x ≤ N)

   ≡{ def de wp, sustituimos

   N ≥ 0 → (0 ≤ 0 ≤ N)

   ≡{ lógica

   N ≥ 0 → 0 ≤ N

   ≡{ lógica p → p

   true

b) **el invariante es invariante? {I ^ B} S {I}**
   {0 ≤ x ≤ N ^ x /= N} x := x + 1 {0 ≤ x ≤ N}

   asumimos 0 ≤ x ≤ N ^ x /= N

   wp.(x := x + 1).(0 ≤ x ≤ N)

   ≡{ def de wp y sust.

   (0 ≤ x + 1 ≤ N)

   ≡{ como 0 ≤ x → 0 < x + 1 y x ≤ N → x < N + 1 es decir 0 < x + 1 < N + 1, por lógica 0 < x + 1 < N + 1 ^ N = x finalmente resto 1 y tenemos -1 < x < N ^ N = x por lógica⋯ 0 ≤ x ≤ N

   true

c) **el ciclo termina? [I ^ ¬B → Q]**

   0 ≤ x ≤ N ^ x = N → x = N

   ≡{ conmutatividad ^

   x = N ^ 0 ≤ x ≤ N → x = N

   ≡{ debilitamiento p ^ q → p

   true

proponemos t = N - x
veamos que se cumpla [P ^ B → t >= 0] ^ {P ^ B ^ t = X} S {T < X}

[**P ^ B → t >= 0**]

suponemos P ^ B es decir, 0 ≤ x ≤ N ^ x =/ N (0 ≤ x < N) entonces

N - x >= 0

≡{ aritmética

N >= x

≡{ lógica

N > x v x =/ N

≡{ suponer

True v x =/ N

≡{ abs disyunción

true

{P ^ B ^ t = X} S {T < X}

{0 ≤ x ≤ N ^ x /= N ^ N - x = X} x := x + 1 {N - x < X}

≡{ asumo 0 ≤ x ≤ N ^ x /= N ^ N - x = X

wp.(x := x + 1).(N - x < X)

≡{ def wp

(N - (x + 1) < X)

≡{ aritmética

(N - x - 1 < X)

≡{ de suponer N - x = X

X - 1 < X

≡{ true

f)

```
{True}
r := N;
   do r/= 0 →
      if r < 0 →
         r := r + 1
      r > 0 →
         r := r − 1
      fi
   od
{r = 0}
```

#Tenemos una composición, entonces existe un {R} tq {P} S {R} ^ {R} T {Q} proponiendo R = (r = N) vemos que vale ya que luego de ejecutar (r:=N) cumple con R

#Debemos encontrar una invariante I y demostrar que

proponemos {r >= 0 v r <= 0} ≡ {true}

i) {R} S {I} (I se cumple inicialmente)

ii) [I ^ ¬B → Q] (la invariante termina)

iii) {I ^ B} S {I} (la invariante es invariante)

Luego proponer una función cota tq se cumpla
[I ^ B → t >= 0] ^ {I ^ B ^ t = X} S {t < X}

Demostración:
i)
{r = N} skip {True} (esto siempre vale)
≡{ lógica
true

ii)
true ^ ¬(r /= 0) → (r = 0)
≡{ lógica
(r = 0) → (r = 0)
≡{ lógica
true

iii)
{true ^ r /= 0}
if r < 0 → r := r + 1
else r > 0 → r := r - 1
fi
{true}
≡{ lógica, def wp del if
(r /= 0) → ((r /= 0 → r < 0 v r > 0) ^ {r /= 0 ^ r < 0} r := r + 1 {true} ^ {r /= 0 ^ r > 0} r := r - 1 {true})
≡{ lógica
(r /= 0) → ((r /= 0 → r /= 0) ^ {r /= 0 ^ r < 0} r := r + 1 {true} ^ {r /= 0 ^ r > 0} r := r - 1 {true})
≡{ lógica
(r /= 0) → (true ^ (r /= 0 ^ r < 0) → wp.(r := r + 1).(true) ^ (r /= 0 ^ r > 0) → wp.(r := r - 1).(true))
≡{ def de wp, neutro ^
(r /= 0) → ((r /= 0 ^ r < 0) →(true).(r := r + 1) ^ (r /= 0 ^ r > 0) → (true).(r := r - 1))
≡{ sustitución
(r /= 0) → ((r /= 0 ^ r < 0) → true ^ (r /= 0 ^ r > 0) → true)
≡{ logica p → true
(r /= 0) → (true ^ true)
≡{ neutro conjunción, p → true
true

ahora demostremos [I ^ B → t >= 0] ^ {I ^ B ^ t = X} S {t < X} con lo cual proponemos t = | r |

[true ^ r /=0 → | r | >= 0] ^ {true ^ r /= 0 ^ | r | = X} if r < 0 → r := r + 1 { | r | < X}
else r > 0 → r := r - 1
≡{ lógica , propiedad | r |
[r /=0 → true] ^ {true ^ r /= 0 ^ | r | = X} if r < 0 → r := r + 1 { | r | < X}
else r > 0 → r := r - 1
≡{ p → true, demostración del if

(true ^ r /= 0 ^ | r | = X) → (r < 0) ^ (r > 0) ^ (((true ^ r /= 0 ^ | r | = X ^ r < 0) → wp.(r := r + 1).(| r | < X)) ^ (true ^ r /= 0 ^ | r | = X ^ r > 0) → wp.(r := r - 1).(| r | < X)))

≡{ asumiendo r /= 0 se cumple

(((true ^ r /= 0 ^ | r | = X ^ r < 0) → wp.(r := r + 1).(| r | < X)) ^ (true ^ r /= 0 ^ | r | = X ^ r > 0) → wp.(r := r - 1).(| r | < X)))

≡{ asumimos (true ^ r /= 0 ^ | r | = X ^ r < 0) ≡ **(| r | = X ^ r < 0)**

wp.(r := r + 1).(| r | < X)) ^ (true ^ r /= 0 ^ | r | = X ^ r > 0) → wp.(r := r - 1).(| r | < X))

≡{ def wp

| r + 1 | < X ^ (true ^ r /= 0 ^ | r | = X ^ r > 0) → wp.(r := r - 1).(| r | < X))

≡{ asumimos true ^ r /= 0 ^ | r | = X ^ r > 0 ≡ **(| r | = X ^ r > 0)**

| r + 1 | < X ^ | r - 1 | < X

≡{ lógica

| r + 1 | /= X v | r | =< X ^ | r - 1 | /= X v | r | =< X

≡{ de asumir

| r + 1 | /= X v X =< X ^ | r - 1 | /= X v X =< X

≡{ lógica

| r + 1 | /= X v true ^ | r - 1 | /= X v true

≡{ absorbente disyunción

true ^ true

≡{ neutro conjunción

true

12. Analice los siguientes programas anotados. En cada caso, describa en lenguaje natural la postcondición, y decida si el programa efectivamente válida las anotaciones. No es necesario hacer las demostraciones.

a) Const $N : Int,\ A : array[0, N)\ of Num;$
Var $s : Num,\ i : Int;$
$\{N \geq 0\}$
$i, s := 0, 0\ ;$
do $i \neq N \rightarrow$
    $s := s + A.i$
od
$\{s = \langle \sum k\ : 0 \leq k < N :\ A.k \rangle\}$

s es igual a la suma de los elementos de un array. El programa no funciona ya que el bucle nunca termina.

b) Const $N : Int,\ A : array[0, N)\ of Num;$
Var $s : Num,\ i : Int;$
$\{N \geq 0\}$
$i, s := 0, 0\ ;$
do $i \neq N \rightarrow$
    $i := i + 1\ ;$
    $s := s + A.i$
od
$\{s = \langle \sum k\ : 0 \leq k < N :\ A.k \rangle\}$

s es igual a la suma de los elementos de un array. El programa no funciona ya que la variable i que usamos para recorrer el array, se saltea la posición 0.

$c)$ Const $N : Int,\ A : array[0, N)\ of Num;$
Var $s : Num,\ i : Int;$
$\{N \geq 0\}$
$i, s := -1, 0$ **;**
do $i \neq N \rightarrow$
$\quad i := i + 1$ **;**
$\quad s := s + A.i$
od
$\{s = \langle \sum k\ : 0 \leq k < N :\ A.k\, \rangle\}$

s es igual a la suma de los elementos de un array. El programa si es válido.

$d)$ Const $E : Num,\ N : Int,\ A : array[0, N)\ of Num;$
Var $i : Int,\ r : Bool;$
$\{N \geq 0\}$
$i, r := 0, False$ **;**
do $i \neq N \wedge \neg r \rightarrow$
$\quad$ **if** $A.i = E \rightarrow r := True$
$\quad$ $\square\ A.i \neq E \rightarrow$ **skip**
$\quad$ **fi ;**
$\quad i := i + 1$
od
$\{\langle \exists k\ : 0 \leq k < N :\ A.k = E\, \rangle \Rightarrow A.i = E\}$

La postcondición nos dice que si existe a lo largo del array, un elemento tal que A.k = E, entonces A.i = E. Este programa es válido si y sólo si la A.i = A.k, es decir i = k al momento de cerrar el bucle.