1. Determinar si la siguiente aserción de corrección parcial es cierta o no, mediante el sistema de prueba de la lógica de Hoare:

$$\left\{ i = j \right\}\ b[i] := 5\ \left\{ b[i] = b[j] \right\}$$

2. Consideremos el siguiente programa:

```
{ PRE: x >= 0 }
       q := 0;
       r := x;
       while r >= y do
         (r := r - y;
         q := q + 1)
       end
     end
   { POST: 0 <= r < y  and  x = q * y + r }
```

   Explica su semántica y comprueba que

```
{ INV: 0 <= r  and  x = q * y + r }
```

   es un invariante del bucle y que el programa es correcto respecto a PRE y POST.

3. Si x es una variable numérica es bien sabido que la aserción de corrección parcial $\{True\}\ x := 1\ \{x = 1\}$ es cierta (¿porqué?). Si b es ahora una variable de *array*, y s cualquier expresión de tipo entero ¿es cierta en general la aserción de corrección parcial: $\{True\}\ b[s] := 1\ \{b[s] = 1\}$? Razona la respuesta.

   Determinar si la siguiente aserción de corrección parcial es cierta o no, mediante el sistema de prueba de la lógica de Hoare:

$$\left\{ b[1] = 2 \wedge b[2] = 2 \right\}\ b\Big[b[2]\Big] := 1\ \left\{ b\Big[b[2]\Big] = 2 \right\}$$

   Respuesta 1 *En primer lugar* $\{True\}\ x := 1\ \{x = 1\}$ *es cierta porque* $\mathcal{WP}(\texttt{x:=1},\ x = 1) \equiv (1 = 1) \equiv True$.

   Pero

$$\{True\}\ b[s] := 1\ \{b[s] = 1\} \tag{1}$$

   no es necesariamente cierta dado que la expresión de tipo entero s *puede depender de* b *como ocurre precisamente en el ejemplo que viene a continuación donde* $s = b[2]$.

   $\mathcal{WP}\Big(\texttt{b[b[2]]:=1},\ b\Big[b[2]\Big] = 2\Big) = \Big(b'\Big[b'[2]\Big] = 2\Big)$ *donde* $b' = b[1/b[2]]$

   Así pues $\Big(b'\Big[b'[2]\Big] = 2\Big) \equiv (2 = b[2] \wedge b'[1] = 2) \vee \Big(2 \neq b[2] \wedge b'\Big[b[2]\Big] = 2\Big)$. *Si* $2 = b[2]$ *entonces* $1 \neq b[2]$ *por lo tanto* $b'[1] = b[1]$. *Por otro lado* $b'\Big[b[2]\Big] = 1$ *por definición de* b'. *Por lo tanto* $(2 = b[2] \wedge b'[1] = 2) \vee \Big(2 \neq b[2] \wedge b'[b[2]] = 2\Big) \equiv (2 = b[2] \wedge b[1] = 2) \vee (2 \neq b[2] \wedge 1 = 2) \equiv (2 = b[2] \wedge b[1] = 2) \vee (False) \equiv (2 = b[2] \wedge b[1] = 2)$.

   Por todo ello se obtiene la certeza de

$$\models \left\{ b[1] = 2 \wedge b[2] = 2 \right\}\ b\Big[b[2]\Big] := 1\ \left\{ b\Big[b[2]\Big] = 2 \right\} \tag{2}$$

Pero, si fuera cierta 1 se tendría:

$$\models \{True\}\ b\Big[b[2]\Big] := 1\ \left\{b\Big[b[2]\Big] = 1\right\}$$

y como cualquer predicado implica True, se tiene que $(b[1] = 2 \wedge b[2] = 2) \Rightarrow True$ *y, por reforzamiento de la precondición*

$$\models \left\{b[1] = 2 \wedge b[2] = 2\right\}\ b\Big[b[2]\Big] := 1\ \left\{b\Big[b[2]\Big] = 1\right\}$$

que contradeciría la certeza de la aserción de corrección parcial 2

4. Consideremos las siguiente aserción de corrección parcial:

```
{ True }
 x:=1; y:=1; c:=0;
 while y < 20 do     {Inv: x = p(c) /\ y = p(c+1)}
        y:= x+y;
        x:= y-x;
        c:= c+1;
{ x = p(c)}
```

¿Qué propiedades debe cumplir la función p para que esta aserción sea cierta?

Respuesta 2 *Ha de verificarse:*

$$True \Rightarrow \mathcal{WP}(\texttt{x:=1; y:=1; c:=0},\ x = p(c) \wedge y = p(c+1)) \tag{3}$$

de donde se obtiene que $p(0) = p(1) = 1$.

Si, para simplificar llamamos C *a* $(y < 20)$ *y* S *a* `y:= x+y;x:= y-x;c:= c+1`, *consideremos la regla*

$$\frac{\{Inv \wedge C\}\ S\ \{Inv\}}{\{Inv\}\ \texttt{while}\ C\ \texttt{do}\ S\ \{Inv \wedge \neg C\}}\ \textit{regla de Hoare del while}$$

Que se cumpla la parte superior significa que:

$$(Inv \wedge C) \Rightarrow \mathcal{WP}(S,\ Inv)$$

es decir

$$x = p(c) \wedge y = p(c+1) \wedge y < 20 \Rightarrow y = p(c+1) \wedge x + y = p(c+2)$$

para lo cual, bastará que

$$p(c+2) = p(c) + p(c+1).$$

Si esto sucede tenemos garantizado que, si se sale del bucle, en el momento de salir se verifica $x = p(c) \wedge y = p(c+1) \wedge y \geq 20$ *que evidentemente implica* $x = p(c)$.

5. Consideremos el siguiente esquema de aserción de corrección parcial con un programa con dos bucles anidados y cada uno con su invariante:

```
{P}
c1;
   while b do                {inv1}
       (c2;
       while b' do           {inv2}
            (c3);
       c4);
c5
{Q}
```

¿Cuales son las obligaciones de prueba?

Respuesta 3 *Las obligaciones de prueba son:*

a) *inv2 es un invariante:*
$inv2 \wedge b' \Rightarrow \mathcal{WP}(c3, inv2)$.
y como consecuencia, podemos poner las anotaciones P1 y P2.

```
{P}
c1;
   while b do                {inv1}
       (c2;
```

$P_1 : \{inv2\}$ while b' do inv2 (c3); $P_2 : \{inv2 \wedge \neg b'\}$	*y esta aserción queda probada*

```
       c4);
c5
{Q}
```

b) *inv1 es un invariante:*[1]

1) $P_2 \Rightarrow \mathcal{WP}(c4, inv1)$

2) $inv1 \wedge b \Rightarrow \mathcal{WP}(c2, P1)$

Y como consecuencia podemos poner la anotaciones P3 y P4

[1] Aquí se usa la regla de la concatenación para que se "peguen"bien los tres programas: c2, el bucle interior y c4 que constituyen el cuerpo del bucle exterior.

```
{P}
c1;
```

$P_3 : \{inv1\}$
`while b do`
`(c2;`
`while b' do`
`(c3);`
`c4);`
$P_4 : \{inv1 \wedge \neg b\}$
	y esta aserción queda probada

```
c5
{Q}
```

c) $P_4 \Rightarrow \mathcal{WP}(c5, Q)$

d) $P \Rightarrow \mathcal{WP}(c1, inv1)$

6. Aplicar el esquema anterior al siguiente ejemplo:

```
{true}
X:= n;                                          \
Y:= m;                                           > c1
S:=0;                                           /

while (X != 0)  do          Inv1:{S+X*Y=n*m}
       (while (X es par) do          inv2:{S+X*Y=n*m}
              ( Y = 2*Y;                        \
                                                 > c3
                X = X/2) ;                      /

       S = S+Y;                                 \
                                                 > c4
       X = X-1)                                 /

 {S=n*m}
```

Respuesta 4 a) $par(X) \wedge S + X * Y = n * m \Rightarrow S + X/2 * 2 * Y = n * m$ *porque si* X *es par entonces* $X/2 * 2 = X$. *(Nótese que* $X/2$ *representa la división entera:* $3/2 = 1$ *y* $4/2 = 2$ *....)*

b) $(S + X * Y = n * m \wedge impar(X) \Rightarrow S + Y + (X - 1) * Y = n * m$ *porque esta segunda proposición se simplifica a* $S + X * Y = n * m$

c) $S + X * Y = n * m \wedge \neg(X = 0) \Rightarrow S + X * Y = n * m$ *evidentemente. Nótese que, respecto al esquema,* c2 *es aquí* `skip`.

d) $S + X * Y = n * m \wedge X = 0 \Rightarrow S = n * m$ *pues al sustituir* X *por* 0 *en la primera cláusula queda justo la segunda proposición.*

e) $true \Rightarrow 0 + n * m = n * m$ *evidentemente.*

El siguiente código java implementa este programa. Pueden compilarlo y comprobar experimentalmente que efectivamente cumple las especificaciones dadas. Naturalmente que estos experimentos no añaden ninguna certidumbre a la verificación formal que hemos hecho pues ésta garantiza que este programa es correcto.

Generalmente el problema es cómo dado un programa concreto como el que se pone a continuación, se prueba que PARA CUALESQUIERA números n y m el programa, si termina, lo hace colocando en S el valor de n*m. El método que hemos seguido nos da esa seguridad.

```
class prod{
       public static void main(String[] args){
       int X = 0;
       int Y = 0;
       int S = 0;

       if (args.length > 0)
               {X = Integer.parseInt(args[0]);
                Y = Integer.parseInt(args[1]);
               }

               while (X != 0) {

                   System.out.println("Invariante primero "
                                      + "------------>"
                                      + (S+X*Y) );

                       while (X % 2 == 0) {

                         System.out.println("Invariante segundo "
                                      + (S+X*Y) );

                         Y = 2*Y;
                         X = X/2;
                                 }
                       S = S+Y;
                       X = X-1;
               }
        System.out.println(S);
       }
}
```

Una vez compilado úsese como "./java prod n m"

7. Los métodos de la lógica de Hoare dados este curso permiten también razonar sobre la corrección de la transformación de programas. Como un ejemplo consideremos la transformación de programas *tail*–recursivos en secuenciales sin recursión.

   Dadas funciones g, r y un predicado P, una función f se dice que es *tail*–recursiva si es de la forma:

$$f(x) = \begin{cases} g(x) & \text{si } P(x) \\ f(r(x)) & \text{en otro caso} \end{cases}$$

   El siguiente código anotado debe implementar la función f.

```
{ x = M }
  while not P(x) do    Inv: { f(x) = f(M) }
     x := r(x);
  x:= g(x)
{ x = f(M) }
```

   Comprobar, usando las condiciones de verificación de las reglas de Hoare, que esta especificación es correcta.

8. Determinar y simplificar la siguiente *weakest precondition*, donde b es un *array* declarado como $b[0:n-1]$ y suponiendo que todos los índices están en rango:

$$wp(\text{"}b[i] := b[i-1] + b[i]\text{"}, b[i] = \Big(\sum j :\ 1 \leq j < i : b[j]\Big))$$

Respuesta 5 *Si llamamos b' al* array *definido por $b' = b[b[i-1] + b[i]/i]$, entonces $wp(\text{"}b[i] := b[i-1] + b[i]\text{"}, b[i] = (\sum j :\ 1 \leq j < i : b[j])) = (b'[i] = (\sum j :\ 1 \leq j < i : b'[j])) = (b[i-1] + b[i] = (\sum j :\ 1 \leq j < i : b[j])) = (b[i] = (\sum j :\ 1 \leq j < i-1 : b[j]))$. pues, siendo la j de la suma menor que i, resulta $b'[j] = b[j]$ y al aparecer en ambos miembros de la suma el sumando $b[i-1]$, éste se cancela.*

9. Sea `int a[n];` la declaración de un *array* de enteros y n una constante natural. Una **sección** de a es una pieza contigua $a[i], \ldots a[j]$, donde $1 \leq i \leq j \leq n$. Una **sección de suma minimal** es una sección $a[i], \ldots a[j]$ tal que la suma $a[i] + a[i+1] + \ldots + a[j]$ es minimal entre todas las secciones de a. (es decir, no hay otra sección que sume menos). Por ejemplo, si a es [-1,3,15,-6,4,-5], [3,15,-6] y [-6] son secciones de a pero [3,-6,4] no lo es pues falta el 15. Una sección de suma minimal es [-6,4,-5] que suma -7; de hecho esta es la única en este caso. En general, las secciones de suma minimal no tienen porqué ser únicas: el *array* [4,-8,3,-4,8,-6,-3,5] tiene dos secciones de suma minimal, [-8,3,-4] y [-6,-3] ambas sumando -9.

   Denotamos $S_{i,j}$ la suma del *array* entre i y j, es decir $a[i] + a[i+1] + \ldots + a[j]$

   Demuestra la siguiente propiedad:

   Propiedad 1 *Sean k, s y t números cualesquiera, y $S_{i,j}$ la suma definida antes. Entonces:*

   a) *Si $\forall i(1 \leq i < k \Rightarrow t \leq S_{i,k-1})$, entonces $\forall i(1 \leq i < k+1 \Rightarrow min(t + a[k], a[k]) \leq S_{i,k}$*

   b) *Si $\forall i(1 \leq i < k \Rightarrow t \leq S_{i,k-1}) \wedge \forall i, j(1 \leq i \leq j < k \Rightarrow s \leq S_{i,j})$, entonces $\forall i, j(1 \leq i \leq j < k+1 \Rightarrow min(s, t + a[k], a[k]) \leq S_{i,j})$.*

   donde min es una función que computa el mínimo.

   (Indicación: Toma un i tal que $1 \leq i < k+1$ y analiza los casos $i < k$ y $i = k$, para el primero; y un par i, j con $(1 \leq i \leq j < k+1$ y analiza los casos $i \leq j < k$ y $i \leq j = k)$ para el segundo.

   Respuesta 6 a) *Sea i tal que $1 \leq i < k+1$; Demostremos que $min(t + a[k], a[k]) \leq S_{i,k}$.*

   Si $i < k$, entonces $S_{i,k} = S_{i,k-1} + a[k]$, por lo tanto tenemos que probar que $min(t + a[k], a[k]) \leq S_{i,k-1} + a[k]$; pero sabemos que $t \leq S_{i,k-1}$, por lo tanto el resultado se deduce añadiendo $a[k]$ a cada lado.

   Si $i = k$, se tiene que $S_{i,k} = a[k]$ y el resultado es evidente.

   b) *Tomemos cualesquiera i y j tales que $1 \leq i \leq j < k+1$; y probemos que $min(s, t + a[k], a[k]) \leq S_{i,j}$.*

   Si $i \leq j < k$, entonces el resultado es inmediato.

   Si, $i \leq j = k$ el resultado se sigue del caso anterior.

   El programa `MinSum`:

```
k = 2;
t = a[1];
s = a[1];
while (k != n+1) {
   t = min (t + a[k], a[k]);
   s = min (s,t);
   k = k + 1;
}
```

para cada valor de k almacena, en t la suma minimal de las secciones que terminan en $a[k]$, y en s la suma minimal obtenida hasta ese momento.

a) Para los *arrays* [-3,1,-2,3,-8] y [1,45,-1,23,-1] simula el código y comprueba que da una suma minimal.

Respuesta 7 *Emulando la ejecución del programa:*

- *Con [-3,1,-2,3,-8]:*

```
k: t: s
2:-3:-3
2:-2:-3
3:-4:-4
4:-1:-4
5:-9:-9
```

- *Con [1,45,-1,23,-1]:*

```
k: t: s
2: 1: 1
2:45: 1
3:-1:-1
4:22:-1
5:-1:-1
```

junto con las comprobaciones correspondientes.

b) Justifica que el predicado $\forall i, j(1 \leq i \leq j \leq n \Rightarrow s \leq S_{i,j})$ describe la postcondición $\mathbf{Q}$ que deseamos obtener.

Respuesta 8 *Efectivamente, si s debe almacenar una suma minimal del vector, entonces para cualquier otra suma $S_{i,j}$ de un segmento comprendido entre entre i y j, ha de cumplirse que s nunca es mayor que dicha suma.*

c) Demuestra que $\forall i, j(i < k \Rightarrow t \leq S_{i,k-1}) \wedge (i \leq j < k \Rightarrow s \leq S_{i,j})$ es un invariante del bucle. (Indicación: puedes usar la propiedad 1 aunque no la hayas demostrado.)

Respuesta 9 *Para mayor claridad se pone encima de cada línea de programa el resultado de "subir" la condición a través de ella:*

`while (k != n+1) {`

$(i < k \Rightarrow t \leq S_{i,k-1}) \wedge (i \leq j < k \Rightarrow s \leq S_{i,j}) \wedge k \neq n+1)$

$\Downarrow$ *debido a la propiedad 1.*

$(i < k+1 \Rightarrow \mathit{min}(t + a[k], a[k]) \leq S_{i,k}) \wedge (i \leq j < k+1 \Rightarrow \mathit{min}(s, \mathit{min}(t + a[k], a[k])) \leq S_{i,j})$

`t = min (t + a[k], a[k]);`

$(i < k+1 \Rightarrow t \leq S_{i,k}) \wedge (i \leq j < k+1 \Rightarrow \mathit{min}(s, t) \leq S_{i,j})$

`s = min (s,t);`

$(i < k+1 \Rightarrow t \leq S_{i,k}) \wedge (i \leq j < k+1 \Rightarrow s \leq S_{i,j})$

`k = k + 1;`

$(i < k \Rightarrow t \leq S_{i,k-1}) \wedge (i \leq j < k \Rightarrow s \leq S_{i,j})$

d) Demuestra la especificación de corrección parcial: $\mathbf{True}$ $\{\texttt{MinSum}\}$ $\mathbf{Q}$.

Respuesta 10 *Efectivamente:*

$\mathbf{True}$

$\Downarrow$

$(a[1] \leq S_{1,1}) \wedge (a[1] \leq S_{1,1})$

$\Downarrow$

$(i < 2 \Rightarrow a[1] \leq S_{i,1}) \wedge (i \leq j < 2 \Rightarrow a[1] \leq S_{i,j})$

```
k = 2;
```

$(i < k \Rightarrow a[1] \leq S_{i,k-1}) \wedge (i \leq j < k \Rightarrow a[1] \leq S_{i,j})$

```
t = a[1];
```

$(i < k \Rightarrow t \leq S_{i,k-1}) \wedge (i \leq j < k \Rightarrow a[1] \leq S_{i,j})$

```
s = a[1];
```

$(i < k \Rightarrow t \leq S_{i,k-1}) \wedge (i \leq j < k \Rightarrow s \leq S_{i,j})$

```
while (k != n+1) {
   t = min (t + a[k], a[k]);
      s = min (s,t);
         k = k + 1;
         }
```

$(i < k \Rightarrow t \leq S_{i,k-1}) \wedge (i \leq j < k \Rightarrow s \leq S_{i,j}) \wedge k = n + 1)$

$\Downarrow$

$(i \leq j \leq n \Rightarrow s \leq S_{i,j})$