Ejercicio 1. Consideremos el problema de sumar los elementos de un arreglo y la siguiente implementación en SmallLang, con el invariante del ciclo.

Especificación

proc sumar (in s: $array < \mathbb{Z} >$) : $\mathbb{Z}$
  requiere $\{True\}$
  asegura $\{res = \sum_{j=0}^{|s|-1} s[j]\}$

Implementación en SmallLang

```
res := 0;
i := 0;
while (i < s.size()) do
  res := res + s[i];
  i := i + 1
endwhile
```

Invariante de Ciclo

$$I \equiv 0 \leq i \leq |s| \wedge_L res = \sum_{j=0}^{i-1} s[j]$$

a) Escribir la precondición y la postcondición del ciclo.

b) ¿Qué punto falla en la demostración de corrección si el primer término del invariante se reemplaza por $0 \leq i < |s|$?

c) ¿Qué punto falla en la demostración de corrección si el límite superior de la sumatoria $(i-1)$ se reemplaza por i?

d) ¿Qué punto falla en la demostración de corrección si se invierte el orden de las dos instrucciones del cuerpo del ciclo?

e) Mostrar la corrección parcial del ciclo, usando los primeros puntos del teorema del invariante.

f) Proponer una función variante y mostrar la terminación del ciclo, utilizando la función variante.

a) La poscondicion del ciclo sera la misma que el *asegura* de la especificacion, $res = \sum_{j=0}^{i-1} s[j]$
La precondicion por otro lado no estoy seguro, sera que res=0 y i=0(las primeras dos lineas de S)?
Could be
b) de reemplazarse el primer termino del invariante este fallaria a la hora de mantenerse cierto al salir del ciclo
Esto pues el while aumenta i en uno |s| veces, donde la |s|-ava vez no entra al ciclo y el programa termina.
c) Deja de ser cierto durante toda la ejecucion del programa, basicamente te esta pidiendo que el termino que el programa esta por sumar ya este sumado a res
d) Al llegar a la ultima iteracion se intentaria acceder a un elemento del arreglo que no existe?¿? creo
f)

- $Pc \implies I$

  $res = 0 \wedge i = 0 \implies 0 \leq i \leq |s| \wedge_L res = \sum_{j=0}^{i-1} s[j]$

  $res = 0 \wedge i = 0 \implies 0 \leq 0 \leq |s| \wedge_L 0 = \sum_{j=0}^{1-1}$

  $res = 0 \wedge i = 0 \implies true$

- $\{I \wedge B\} S \{I\}$

  $\{0 \leq i \leq |s| \wedge_L res = \sum_{j=0}^{i-1} s[j] \wedge i < |s|\} \equiv \{0 \leq i < |s| \wedge_L res = \sum_{j=0}^{i-1} s[j]\}$

  `res:= res + s[i]`

  `i:=i+1`

  $\{0 \leq i \leq |s| \wedge_L res = \sum_{j=0}^{i-1} s[j]\}$

  Esto se traduce a comparar la primera parte de la tripla de Hoare con wp(res:=res+s[i]; i:=i+1, I)

  $\{0 \leq i < |s| \wedge_L 0 \leq i+1 \leq |s| \wedge_L res + s[i] = \sum_{j=0}^{i} s[j]\}$

  $\equiv \{0 \leq i < |s| \wedge_L -1 \leq i \leq |s| - 1 \wedge_L res = \sum_{j=0}^{i-1} s[j]\}$

  $\equiv \{0 \leq i < |s| \wedge_L res = \sum_{j=0}^{i-1} s[j]\}$

  Y podemos ver que ambos $I \wedge B$ y la wp calculada son iguales, por lo tanto la tripla de Hoare es correcta

- $I \wedge \neg B \implies Q_c$

  $0 \leq i \leq |s| \wedge_L res = \sum_{j=0}^{i-1} s[j] \wedge \neg(i < |s|)$

  $\equiv 0 \leq i \leq |s| \wedge_L res = \sum_{j=0}^{i-1} s[j] \wedge i \geq |s|$

  $\equiv i = |s| \wedge_L res = \sum_{j=0}^{i-1} s[j]$

  $\equiv res = \sum_{j=0}^{|s|-1} s[j]$

Que es efectivamente nuestra postcondicion de la especificacion.

f) Propuesta 1: $fv = |s| - i + 1$

$\{I \wedge B \wedge v_0 = fv\} res := res + s[i]; i := i + 1 \{fv < v_0\} \equiv Pc \implies wp(s, fv < v_0)$

$wp(S, fv < v_0) \equiv wp(res := res + s[i], wp(i := i + 1, |s| - i + 1 < v_0))$
cosas...
$\equiv |s| - (i + 1) + 1 < v_0 \equiv |s| - i < v_0$

retomando el principio:
$I \wedge B \wedge v_0 = fv \implies |s| - i < v_0$?
Si, pues $v_o = fv \implies |s| - i < |s| - i + 1$ que es trivialmente cierto.
Ahora, mi propuesta cumple con el segundo statement?
$I \wedge fv \leq 0 \implies \neg B$
$\alpha: \ |s| - i + 1 \leq 0 \implies |s| + 1 \leq i$
$\beta: \ \neg B \equiv i \geq |s| \equiv |s| \leq i$ $\longleftarrow$A lo que quiero llegar
$I \equiv 0 \leq i \leq |s| \wedge_L res = \sum_{j=0}^{i-1} s[j]$
Tengo informacion de sobra, $\alpha \implies |s| \leq i \equiv \neg B$
Seems a bit too easy pero es el primer ejercicio de la guia so...

Ejercicio 2. Dadas la especificación y la implementación del problema `sumarParesHastaN`, escribir la precondición y la postcondición del ciclo, y mostrar su corrección a través del teorema del invariante.

Especificación

proc sumarParesHastaN (in n: $\mathbb{Z}$) : $\mathbb{Z}$
  requiere $\{n \geq 0\}$
  asegura $\{res = \sum_{j=0}^{n-1}(\text{if } j \ mod\ 2 = 0 \text{ then } j \text{ else } 0 \text{ fi})\}$

Implementación en SmallLang

```
res := 0;
i := 0;
while (i < n) do
  res := res + i;
  i := i + 2
endwhile
```

Invariante de ciclo

$$I \equiv 0 \leq i \leq n+1 \wedge i \ mod\ 2 \ = \ 0 \wedge res = \sum_{j=0}^{i-1}(\text{if } j \ mod\ 2 = 0 \text{ then } j \text{ else } 0 \text{ fi})$$

La precondicion del ciclo seria .....
La poscondicion es $res = \sum_{j=0}^{n-1} if\ \ j\ \ mod 2 = 0\ \ then\ \ j\ \ else\ \ 0$
$P \implies I$ es trivialmente cierto

$2^{da}) \{I \wedge B\} res := res + i; i := i + 2 \{I\}$

$I \equiv 0 \leq i \leq n+1 \wedge i\,\%2 = 0 \wedge res = \sum_{j=0}^{i-1} if\ \ j\,\%2 = 0\ \ then\ \ j\ \ else\ \ 0$ ←Por las dudas: Uso % en lugar de mod por cuestiones de spacing
$B \equiv i < n$
$wp(S, I) \equiv 0 \leq i \leq n+1 \wedge i\,\%2 = 0 \wedge res + i = \sum_{j=0}^{i+1} if\ \ j\,\%2 = 0\ \ then\ \ j\ \ else\ \ 0$
Analizando un poco nos damos cuenta de que i siempre sera par, por lo que i+1 es impar, entonces el termino i+1 de la sumatoria siempre sumara 0 (por el condicional), asi que podemos obviarlo. A su vez, podemos restar a ambos lados el termino i-esimo, que este sera par y por lo tanto si contara en la sumatoria, simplificando la expresion.
$wp(S, I) \equiv 0 \leq i \leq n+1 \wedge i\,\%2 = 0 \wedge res = \sum_{j=0}^{i-1} if\ \ j\,\%2 = 0\ \ then\ \ j\ \ else\ \ 0$
Entonces, $I \wedge B \equiv 0 \leq i < n \wedge i\,\%2 = 0 \wedge res = \sum_{j=0}^{i-1} if\ \ j\,\%2 = 0\ \ then\ \ j\ \ else\ \ 0$
Que efectivamente implica mi wp, pues el segundo termino es identico y en el primero,
$0 \leq i < n \implies 0 \leq i \leq n+1$

$3^{ra})\ \ I \wedge \neg B \implies Q_c$
$I \equiv 0 \leq i \leq n+1 \wedge i\,\%2 = 0 \wedge res = \sum_{j=0}^{i-1} if\ \ j\,\%2 = 0\ \ then\ \ j\ \ else\ \ 0$
$\neg B \equiv i \geq n$
$n \leq i \wedge i \leq n+1 \equiv n \leq i \leq n+1$
Pero i tiene que ser par, por lo que: $n \leq i \leq n+1 \equiv n = i = n \equiv i = n$
$i = n \wedge i\,\%2 = 0 \wedge res = \sum_{j=0}^{i-1} if\ \ j\,\%2 = 0\ \ then\ \ j\ \ else\ \ 0 \equiv res = \sum_{j=0}^{n-1} if\ \ j\,\%2 = 0\ \ then\ \ j\ \ else\ \ 0 \equiv Q_c$
Posible invariante: $n - i + 1$
$\{I \wedge B \wedge v_0 = fv\} S \{fv < v_0\}$
$wp(i := i + 2, n - i + 1 < v_0) \equiv n - i - 1 < v_0$
entonces $v_0 = fv \implies n - i - 1 < n - i + 1$
$5^{ta})\ \ I \wedge fv \leq 0 \implies \neg B$
Quiero entonces llegar a $\neg B \equiv i \geq n$
$fv \leq 0 \implies n + 1 \leq i$
$I \implies 0 \leq i \leq n+1$
Estas dos cosas implican que $i = n+1$ que a su vez implica $i \geq n$
De forma un poco confusa y poco organizada, pero queda entonces demostrada la correctitud del ciclo.

Ejercicio 3. Considere el problema `sumaDivisores`, dado por la siguiente especificación:

`proc sumaDivisores` (in n: $\mathbb{Z}$) : $\mathbb{Z}$
`requiere` $\{n \geq 1\}$
`asegura` $\{res = \sum_{j=1}^{n}(\text{if } n \ mod \ j = 0 \text{ then } j \text{ else } 0 \text{ fi})\}$

a) Escribir un programa en SmallLang que satisfaga la especificación del problema y que contenga exactamente un ciclo.

---

b) Escribir la pre y post condición del ciclo y su invariante.

c) Considere el siguiente invariante para este problema

$$I \equiv 1 \leq i \leq n/2 \wedge res = \sum_{j=1}^{i}(\text{if } n \ mod \ j = 0 \text{ then } j \text{ else } 0 \text{ fi})$$

Si no coincide con el propuesto en el inciso anterior, ¿qué cambios se le deben hacer al programa para que lo represente este invariante? ¿Deben cambiar la pre y post condición?

```
while i<=n do
if n mod i == 0 then
res:=res+i
i:=i+1
```

$P_c \equiv i = 0 \wedge res = 0$
$Q_c \equiv Q \equiv res = \sum_{j=1}^{n} \ if \ n\%j = 0 \ then \ j \ else \ 0$
Propuesta de invariante:
$I \equiv 0 \leq i \leq n+1 \wedge res = \sum_{j=1}^{i} \ if \ n\%j = 0 \ then \ j \ else \ 0$
c) habria que cambiar la guarda del while, la pre y la post condicion no cambian.

Ejercicio 4. Considere la siguiente especificación e implementación del problema `copiarSecuencia`, y la pre y post condiciones del ciclo

Especificación

`proc copiarSecuencia` (in s: $array < \mathbb{Z} >$, inout r: $array < \mathbb{Z} >$)
`requiere` $\{|s| = |r| \wedge r = r_0\}$
`asegura` $\{|s| = |r| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] = r[j])\}$

Implementación en SmallLang

```
i := 0;
while (i < s.size()) do
  r[i]:=s[i];
  i:=i+1
endwhile
```

$$P_c \equiv |s| = |r| \wedge i = 0$$

$$Q_c \equiv (\forall j : \mathbb{Z})(0 \leq j < |r| \rightarrow_L s[j] = r[j])$$

a) ¿Qué variables del programa deben aparecer en el invariante?

b) Proponer un invariante e indicar qué clausula del mismo es necesario para cada paso de la demostración.

c) Proponer una función variante que permita demostrar que el ciclo termina.

d) Comparar la solución propuesta con la que ofrecemos al final de la guía.

a) Todas?¿?¿ i, s y r
b)
$I \equiv 0 \leq i \leq |s| \wedge |s| = |r| \wedge (\forall j : \mathbb{Z})\ (0 \leq j < i \wedge r[j] = s[j])$
Lo rojo y azul es necesario para $P_c \implies I$
Azul y negro para $\{B \wedge I\}S\{I\}$
Denuevo, azul y negro para $I \wedge \neg B \implies Q_c$
c) fv=|s|-i
d)

Soluciones

Ejercicio 4

- $I \equiv 0 \leq i \leq |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = r[j])$
- $f_v = |s| - i$

Bueno no hacia falta lo rojo parece, fair enough

Ejercicio 5. Sea el siguiente ciclo con su correspondiente precondición y postcondición:

```
while (i >= s.size() / 2) do
        suma := suma + s[s.size()-1-i];
  i := i - 1
endwhile
```

$$P_c : \{|s| \ mod \ 2 = 0 \land i = |s| - 1 \land suma = 0\}$$

$$Q_c : \{|s| \ mod \ 2 = 0 \land i = |s|/2 - 1 \ \land_L \ suma = \sum_{j=0}^{|s|/2-1} s[j]\}$$

a) Proponer un invariante e indicar qué clausula del mismo es necesario para cada paso de la demostración.

b) Proponer una función variante que permita demostrar que el ciclo termina.

c) Comparar la solución propuesta con la que ofrecemos al final de la guía.

Idealmente en papel haria un par de iteraciones del programa para identificar correctamente que es lo que hace, honestamente acá podria hacer una tabla y bla bla pero me da paja. Y ademas me parece que simplemente lo que hace es, de forma enrevesada, solo sumar la primera mitad de un arreglo de longitud par.

Entonces mi propuesta de invariante es:
$I \equiv \frac{|s|}{2} - 1 \leq i \leq |s| - 1 \land suma = \sum_{j=0}^{|s|-2-i} s[j]$
Y mi propuesta de funcion variante es $f_v = |s|/2 - 1 - i$

Ejercicio 5

a) $I \equiv |s|/2 - 1 \leq i \leq |s| - 1 \land_L suma = \sum_{j=0}^{|s|-2-i} s[j]$

b) $f_v = i - (|s|/2 - 1)$

Ahora viendo la solucion me doy cuenta de que claro, como i esta "decreciendo"tengo que invertir los signos a como lo hago normalmente, porque asi como lo hice no se cumple que al finalizar el ciclo mi fv sea menor o igual a cero

El seis me da paja, asi que va el siete.

Ejercicio 7. Considerando el siguiente Invariante:

$$I \equiv \{0 \leq i \leq |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L ((j \ mod \ 2 = 0 \wedge s[j] = 2 \times j) \vee (j \ mod \ 2 \neq 0 \wedge s[j] = 2 \times j + 1)))\}$$

- Escribir un programa en SmallLang que se corresponda al invariante dado.
- Defina las P_c, B y Q_c que correspondan a su programa.
- Dar una función variante para que se pueda completar la demostración.

```
i:=0
while i < s.len() do
      if i%2 = 0 then
            s[j]:=i*2
      else
            s[j]:=i*2+1
```

$P_c \equiv i = 0$
$B \equiv i < s.len()$
$Q_c \equiv (\forall i : \mathbb{Z}) \ \ (0 \leq i < |s| \wedge_L (i\,\%2 = 0 \wedge s[i] = 2 * i) \vee (i\,\%2 \neq 0 \wedge s[i] = i * 2 + 1))$
$fv = |s| - i$

raro que no pida demostrar nada, quizas si me queda tiempo antes del recu intento hacerlo por los loles

Ejercicio 8. Considerando el siguiente Invariante:

$$I \equiv \{0 \leq i \leq |s|/2 \wedge (\forall j : \mathbb{Z})(0 \leq j < i) \rightarrow_L (s[j] = 0 \wedge s[|s| - j - 1] = 0)\}$$

- Escribir un programa en SmallLang que se corresponda al invariante dado.
- Defina las P_c, B y Q_c que correspondan a su programa.
- Dar una función variante para que se pueda completar la demostración.

Aparentemente pone todo en cero desde el principio y el final no?
Me parece raro que no tenga como requiere que la longitud de s sea par

---

```
i:=0
while i < |s|/2
      s[i]:=0
      s[|s|-i-1]:=0
      i:=i+1
```

$P_c \equiv i = 0 \wedge |s| \,\%2 = 0$?
$B \equiv i < |s|/2$
$Q_c \equiv (\forall i : \mathbb{Z})\ (0 \leq i < |s| \wedge_L s[i] = 0 \wedge s[|s| - i - 1] = 0)$
Puede ser que no haga falta lo de que sea par ya que de ultima si i y |s|-i-1 apuntan al mismo slot del array igualmente es valido porque ambos solo requieren que este sea igual a cero
$fv = |s|/2 - i$

Ejercicio 9. Indique si el siguiente enunciado es verdadero o falso; fundamente:

Si dados B y I para un ciclo S existe una función f_v que cumple lo siguiente:

- $\{I \wedge B \wedge f_v = V_0\} S \{f_v > V_0\}$
- $\exists (k : \mathbb{Z})(I \wedge f_v \geq k \rightarrow \neg B)$

entonces el ciclo siempre termina.

Estos dos predicados pareciera que hacen referencia a un i creciente, la duda que me surge es como determinar k (¿hace falta determinarlo?).
Comparandolo con el predicado que usamos nosotros, que usa el signo invertido y k igual a cero, parece innecesariamente mas complejo, aunque imagino que es valido igualmente.
Por ejemplo cuando recorres los indices de una lista, k seria |s| no? Es realmente tanto mas complejo entonces?
Se me hace que k siempre seria esa primera parte que le pongo al invariante, a la que basicamente le resto i de alguna forma mas o menos compleja.
Esto haria entonces que fv sea simplemente i (y sus cosas extra), y de hecho esto tiene sentido porque
$fv = i \wedge k = |s| \implies fv \geq k \equiv i \geq |s| \equiv i - |s| \geq 0 \equiv |s| - i < 0$
Not quite there :/
tl;dr: ni idea, creo que si pero no se como fundamentar mi respuesta, parece una generalizacion del predicado que usamos.

Ejercicio 10. Considere la siguiente especificación y su implementación

Especificación

proc existeElemento (in s: $array < \mathbb{Z} >$, in e: $\mathbb{Z}$) : Bool
requiere $\{True\}$
asegura $\{res = \text{true} \leftrightarrow ((\exists k : \mathbb{Z})(0 \leq k < |s|) \wedge_L s[k] = e)\}$

Implementación en SmallLang

```
i := 0;
j := -1;
while (i < s.size()) do
  if (s[i] = e) then
    j := i
  else
    skip
  endif;
  i := i + 1
endwhile;
if (j != -1)
  res := true
else
  res := false
endif
```

Escribir los pasos necesarios para demostrar la correctitud de la implementación respecto a la especificación usando WP y el teorema del invariante

Habria que comenzar viendo cual es la Q principal, osea la de la especificacion, a la que llamare Q_e
$Q_e \equiv res = true \leftrightarrow (\exists k : \mathbb{Z})\ \ (0 \leq k < |s| \wedge_L s[k] = e)$
Ahora de forma secuencial puedo comenzar a calcular las wp por linea/instruccion

$$\begin{aligned} wp(\texttt{if } j! = 1 \texttt{ then } res := true \texttt{ else } res := false, Q_e) \equiv & def(b) \wedge_L (j \neq -1 \wedge Q^{res}_{true}) \vee (j = -1 \wedge Q^{res}_{false}) \\ \equiv & (j \neq -1 \wedge true = true \leftrightarrow (\exists k : \mathbb{Z})\ \ (0 \leq k < |s| \wedge_L s[k] = e)) \\ & \vee (j = -1 \wedge false = true \leftrightarrow (\exists k : \mathbb{Z})\ \ (0 \leq k < |s| \wedge_L s[k] = e)) \\ \#1 \equiv & (j \neq -1 \wedge (\exists k : \mathbb{Z})\ \ (0 \leq k < |s| \wedge_L s[k] = e)) \\ & \vee (j = -1 \wedge (\forall k : \mathbb{Z})\ \ (0 \leq k < |s| \wedge_L s[k] \neq e)) \end{aligned}$$

#1) Acá es confuso, dado que *q sii p* es una expresion que es verdadera si q y p son falsos o verdaderos simultaneamente, creo que puedo reducirlo a lo escrito dado que

$$\begin{gathered} q \leftrightarrow p \equiv q \implies p \wedge p \implies q \\ \texttt{si } q = true \\ true \implies p \wedge p \implies true \\ p \wedge p \\ p \\ \texttt{si } q = false \\ false \implies p \wedge p \implies false \\ true \wedge \neg p \\ \neg p \end{gathered}$$

Voy a llamar a esta postcondicion Q_i por ser la wp de mi Q_e y el if exterior.
$Q_i \equiv (j \neq -1 \wedge (\exists k : \mathbb{Z})\ \ (0 \leq k < |s| \wedge_L s[k] = e)) \vee (j = -1 \wedge (\forall k : \mathbb{Z})\ \ (0 \leq k < |s| \wedge_L s[k] \neq e))$
Justo tambien es la postcondicion de mi ciclo, asi que ahora tengo que demostrar la correctitud del ciclo
$P_c \equiv i = 0 \wedge j = -1$
$B_c \equiv i < |s|$
$S \equiv$ `if s[i]=e then j:=i else skip`$; i := i + 1$
Propuesta de invariante:
$I \equiv 0 \leq i \leq |s| \wedge -1 \leq j < |s|$
¿$P_c \implies I$?
$i = 0 \wedge j = -1 \implies I \equiv 0 \leq i \leq |s| \wedge -1 \leq j < |s|$ ■

¿$\{I \wedge B\}S\{I\}$?

Esto equivale a ver si $I \wedge B \implies wp(S, I)$, asi que comienzo calculando dicha wp

$wp(i := i + 1, I) \equiv 0 \leq i + 1 \leq |s| \wedge -1 \leq j < |s|$

$wp(i := i + 1, I) \equiv -1 \leq i \leq |s| - 1 \wedge -1 \leq j < |s|$

Ahora con el if:

$wp(\texttt{if s[i]=e then j:=i else skip}, wp(i := i + 1, I)) \equiv def(B) \wedge ((B \wedge Q_i^j) \vee (\neg B \wedge true?))$

$0 \leq i < |s| \wedge_L ((s[i] = e \wedge -1 \leq i \leq |s| - 1 \wedge -1 \leq i < |s|) \vee s[i] \neq e)$

$0 \leq i < |s| \wedge_L (s[i] = e \vee s[i] \neq e)$

$0 \leq i < |s|$

Se siente raro

$I \wedge B \implies 0 \leq i < |s|$ ■?

¿$I \wedge \neg B \implies Q_i$?

Esto es medio raro de demostrar, no se si se puede de hecho. Siento que I y no B no me dan suficiente informacion sobre la postcondicion, o es que estoy en una situacion de algo pseudo true implies whatever? estoy en una situacion de true implies p o no p? parece no? Asumo entonces que estoy en lo correcto y que la desmotracion es valida (?)

Una posible funcion variante es: $fv = |s| - i$

$\{I \wedge B \wedge fv = v_0\}Sfv < v_0$

Trivialmente cierto pues i crece en cada iteracion.

¿$I \wedge fv \leq 0 \implies \neg B$?

$I \wedge fv \leq 0 \equiv 0 \leq i \leq |s| \wedge |s| - i \leq 0$

$0 \leq i \leq |s| \wedge |s| - 1 \leq i \equiv |s| \leq i \leq |s| \implies i \geq |s| \equiv \neg B$ ■

Muy dudoso todo pero tengo sueño y hambre y es tarde y no quiero saber mas nada de esto. Mañana sera otro dia.