

Guías Resueltas de Algoritmos y Estructura de Datos I

Resueltas por Ezequiel Companeetz y Galileo Federico Capella Lewi

Segundo Cuatrimestre Virtual, 2020

Aclaraciones

Introducción

Esta materia la cursamos virtualmente con Matías López y Rosenfeld, Pablo Turjanski y Javier Marenco como profesores en las clases teóricas; con Cyntia Bonomi y Gabriela Di Piazza como JTPS ; Sebastián Vita, Javier Godoy, Leandro Lera Romero , Bruno Bianchi como ayudantes de primera y Alejandro Pontalti María Belén Ticona Oquendo como ayudantes de segunda. Esta guía esta hecha en parte gracias a los docentes que nos respondieron todas las consultas habidas y por haber copado una increíble paciencia. No esta de más aclarar que aunque la revisamos y comprobamos la misma  **debe tener errores** o formas de resolver ejercicios que no son lo que se busca en la materia. Aún así creemos que es una buena herramienta para comparar con lo que uno hizo o inspirarse para algún ejercicio muy complicado. Además les dejamos un repo con el [código](#) de los ejercicios de algunas prácticas y labos

Uso

Sepan que al estar revisando las consignas de los ejercicios pueden hacerle click a la consigna y los va a llevar directamente a la resolución. Además acá les dejamos un indice para ir directamente a las guías que necesiten (además les aparecerá un indice si usan un lector de Pdf).

1. Práctica 1
2. Práctica 2
3. Práctica 3
4. Práctica 4
5. Práctica 5
6. Práctica 6
7. Práctica 7
8. Práctica 8
9. Práctica 9

Bonus Tracks

Esto es lo más inqueable de las prácticas pero a la vez lo que mas nos divirtió hacer. Al final de algunas guías van a tener un dato, ejercicio o aplicación que nos resultó interesante/importante acerca de los contenidos tratados en esa práctica. Esperamos que les sean útiles y les sirvan para entender un poco más cada tema.



1. Lógica binaria (Verdadero o Falso)

Ejercicio 1. ★ Sean p y q variables proposicionales. ¿Cuáles de las siguientes expresiones son *fórmulas bien formadas*?

- | | | |
|---|-------------------------------------|---------------------|
| a) $(p \neg q)$ | d) $\neg(p)$ | g) $(\neg p)$ |
| b) $p \vee q \wedge True$ | e) $(p \vee \neg p \wedge q)$ | h) $(p \vee False)$ |
| c) $(p \rightarrow \neg p \rightarrow q)$ | f) $(True \wedge True \wedge True)$ | i) $(p = q)$ |

Ejercicio 2. ★ Sean $x : \mathbb{Z}$, $y : \mathbb{Z}$ y $z : \text{Bool}$ tres variables. Indique cuáles de las siguientes expresiones están bien definidas, teniendo en cuenta que estén bien tipadas las subexpresiones que correspondan.

- | | |
|---------------------------|--|
| a) $(1 = 0) \vee (x = y)$ | d) $z = \text{true} \leftrightarrow (y = x)$ |
| b) $(x + 10) = y$ | e) $(z = 0) \vee (z = 1)$ |
| c) $(x \vee y)$ | f) $y + (y < 0)$ |

Ejercicio 3. La fórmula $(3 + 7 = \pi - 8) \wedge True$ es una fórmula bien formada. ¿Por qué? Justifique informal, pero detalladamente, su respuesta.

Ejercicio 4. ★ Determinar el valor de verdad de las siguientes proposiciones

- | | |
|--|---|
| a) $(\neg a \vee b)$ | e) $((c \vee y) \wedge (x \vee b))$ |
| b) $(c \vee (y \wedge x) \vee b)$ | f) $((((c \vee y) \wedge (x \vee b)) \leftrightarrow (c \vee (y \wedge x) \vee b))$ |
| c) $\neg(c \vee y)$ | g) $(\neg c \wedge \neg y)$ |
| d) $(\neg(c \vee y) \leftrightarrow (\neg c \wedge \neg y))$ | |

cuando el valor de verdad de a , b y c es *verdadero*, mientras que el de x e y es *falso*.

Ejercicio 5. Determinar, utilizando tablas de verdad, si las siguientes fórmulas son tautologías, contradicciones o contingencias.

- | | |
|--|--|
| a) $(p \vee \neg p)$ | f) $(p \rightarrow p)$ |
| b) $(p \wedge \neg p)$ | g) $((p \wedge q) \rightarrow p)$ |
| c) $((\neg p \vee q) \leftrightarrow (p \rightarrow q))$ | h) $((p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r)))$ |
| d) $((p \vee q) \rightarrow p)$ | i) $((p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r)))$ |
| e) $(\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q))$ | |

Ejercicio 6. ★ Dadas las proposiciones lógicas α y β , se dice que α es más fuerte que β si y sólo si $\alpha \rightarrow \beta$ es una tautología. En este caso, también decimos que β es más débil que α . Determinar la relación de fuerza de los siguientes pares de fórmulas:

- | | |
|-------------------------------|---------------------------|
| a) $True, False$ | e) $False, False$ |
| b) $(p \wedge q), (p \vee q)$ | f) $p, (p \vee q)$ |
| c) $True, True$ | g) p, q |
| d) $p, (p \wedge q)$ | h) $p, (p \rightarrow q)$ |

¿Cuál es la proposición más fuerte y cuál la más débil de las que aparecen en este ejercicio?

Ejercicio 7. ★ Usando reglas de equivalencia (comutatividad, asociatividad, De Morgan, etc) determinar si los siguientes pares de fórmulas son equivalencias. Indicar en cada paso qué regla se utilizó.

- a)
 - $((\neg p \vee \neg q) \vee (p \wedge q)) \rightarrow (p \wedge q)$
 - $(p \wedge q)$
- b)
 - $(p \vee q) \wedge (p \vee r)$
 - $\neg p \rightarrow (q \wedge r)$
- c)
 - $\neg(\neg p) \rightarrow (\neg(\neg p \wedge \neg q))$
 - q
- d)
 - $((True \wedge p) \wedge (\neg p \vee False)) \rightarrow \neg(\neg p \vee q)$
 - $p \wedge \neg q$
- e)
 - $(p \vee (\neg p \wedge q))$
 - $\neg p \rightarrow q$
- f)
 - $\neg(p \wedge (q \wedge s))$
 - $s \rightarrow (\neg p \vee \neg q)$
- g)
 - $p \rightarrow (q \wedge \neg(q \rightarrow r))$
 - $(\neg p \vee q) \wedge (\neg p \vee (q \wedge \neg r))$

Ejercicio 8. Decimos que un conectivo es *expresable* mediante otros si es posible escribir una fórmula utilizando exclusivamente estos últimos y que tenga la misma tabla de verdad que el primero (es decir, son equivalentes). Por ejemplo, la disyunción es expresable mediante la conjunción más la negación, ya que $(p \vee q)$ tiene la misma tabla de verdad que $\neg(\neg p \wedge \neg q)$.

Mostrar que cualquier fórmula de la lógica proposicional que utilice los conectivos \neg (negación), \wedge (conjunción), \vee (disyunción), \rightarrow (implicación), \leftrightarrow (equivalencia) puede reescribirse utilizando sólo los conectivos \neg y \vee .

Ejercicio 9. ★ Sean las variables proposicionales f , e y m con los siguientes significados:

$$f \equiv \text{"es fin de semana"} \quad e \equiv \text{"Juan estudia"} \quad m \equiv \text{"Juan escucha música"}$$

a) Escribir usando lógica proposicional las siguientes oraciones:

- “Si es fin de semana, Juan estudia o escucha música, pero no ambas cosas”
- “Si no es fin de semana entonces Juan no estudia”
- “Cuando Juan estudia los fines de semana, lo hace escuchando música”

b) Asumiendo que valen las tres proposiciones anteriores ¿se puede deducir que Juan no estudia? Justificar usando argumentos de la lógica proposicional.

Ejercicio 10. En la salita verde de un jardín se sabe que las siguientes circunstancias son ciertas:

- a) Si todos conocen a Juan entonces todos conocen a Camila (podemos pensar que esto se debe a que siempre caminan juntos).
- b) Si todos conocen a Juan, entonces que todos conozcan a Camila implica que todos conocen a Gonzalo.

La pregunta entonces es: ¿Es cierto que si todos conocen a Juan entonces todos conocen a Gonzalo? Justificar.

Ejercicio 11. Siempre que Haroldo se pelea con sus compañeritos, vuelve a casa con un ojo morado. Si un día lo viéramos llegar con el ojo destrozado, podríamos sentirnos inclinados a concluir que se ha tomado a golpes de puño y cabezazos con los otros niñitos. ¿Puede identificar el error en el razonamiento anterior? *Pista:* Es conocido como *falacia de afirmar el consecuente*.

2. Lógica ternaria (Verdadero, Falso o Indefinido)

Ejercicio 12. ★ Asignar un *valor de verdad* (*verdadero*, *falso* o *indefinido*) a cada una de las siguientes *expresiones aritméticas* en los reales.

- | | | |
|------------------------------|--------------------------------|--|
| a) $5 > 0$ | d) $0 \geq 5$ | g) $0 \cdot \sqrt{-1} = 0$ |
| b) $1 \leq 1$ | e) $\frac{1}{0} = \frac{1}{0}$ | h) $\sqrt{-1} \cdot 0 = 0$ |
| c) $(5 + 3 - 8)^{-1} \neq 2$ | f) $0 > \log_2(2^{2^0-1} - 1)$ | i) $\tan(\frac{\pi}{2}) = \tan(\pi) - \tan(2)$ |

Ejercicio 13. ★ ¿Cuál es la diferencia entre el operador \wedge y el operador \wedge_L ? Describir la tabla de verdad de ambos operadores

Ejercicio 14. ★ ¿Cuál es la diferencia entre el operador \vee y el operador \vee_L ? Describir la tabla de verdad de ambos operadores.

Ejercicio 15. ★ ¿Cuál es la diferencia entre el operador \rightarrow y el operador \rightarrow_L ? Describir la tabla de verdad de ambos operadores.

Ejercicio 16. ★ Determinar los valores de verdad de las siguientes proposiciones cuando el valor de verdad de b y c es *verdadero*, el de a es *falso* y el de x e y es *indefinido*

- | | |
|--|---|
| a) $(\neg x \vee_L b)$ | e) $((c \vee_L y) \wedge_L (a \vee_L b))$ |
| b) $((c \vee_L (y \wedge a)) \vee b)$ | f) $((((c \vee_L y) \wedge_L (a \vee_L b)) \leftrightarrow (c \vee_L (y \wedge_L a) \vee_L b))$ |
| c) $\neg(c \vee_L y)$ | g) $(\neg c \wedge_L \neg y)$ |
| d) $(\neg(c \vee_L y) \leftrightarrow (\neg c \wedge_L \neg y))$ | |

Ejercicio 17. Sean p , q y r tres variables de las que se sabe que:

- p y q nunca están indefinidas,
- r se define si q es *verdadera*

Proponer una fórmula que nunca se indefina, utilizando siempre las tres variables y que sea verdadera si y solo si se cumple que:

- | | |
|---|--|
| a) Al menos una es verdadera | d) Sólo p y q son verdaderas |
| b) Ninguna es verdadera | e) No todas al mismo tiempo son verdaderas |
| c) Exactamente una de las tres es verdadera | f) r es verdadera |

3. Cuantificadores

Ejercicio 18.

a) ★ Determinar para cada aparición de variables, si dicha aparición se encuentra libre o ligada. En caso de estar ligada, aclarar a qué cuantificador lo está.

- I) $(\forall x : \mathbb{Z})(0 \leq x < n \rightarrow x + y = z)$
- II) $(\forall x : \mathbb{Z})((\forall y : \mathbb{Z})((0 \leq x < n \wedge 0 \leq y < m) \rightarrow x + y = z))$
- III) $(\forall j : \mathbb{Z})(0 \leq j < 10 \rightarrow j < 0)$
- IV) $s \wedge a < b - 1 \wedge ((\forall j : \mathbb{Z})(a \leq j < b \rightarrow_L 2 * j < b \vee s))$
- V) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow (\forall j : \mathbb{Z})(j > 0 \rightarrow j \neq 0))$
- VI) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow P(j))$
- VII) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow P(j)) \wedge P(j)$

b) ★ En los casos en que sea posible, proponer valores para las variables libres del ítem anterior de modo tal que las expresiones sean verdaderas.

Ejercicio 19. Sean $P(x : \mathbb{Z})$ y $Q(x : \mathbb{Z})$ dos predicados cualesquiera que nunca se indefinen. Considerar los siguientes enunciados y su predicado asociado. Determinar, en cada caso, por qué el predicado no refleja correctamente el enunciado. Corregir los errores.

a) “Todos los naturales menores a 10 que cumplen P , cumplen Q ”:

$$\text{pred } a() \{ (\forall x : \mathbb{Z}) ((0 \leq x < 10) \rightarrow_L (P(x) \wedge Q(x))) \}$$

b) “No hay ningún natural menor a 10 que cumpla P y Q ”:

$$\text{pred } c() \{ \neg((\exists x : \mathbb{Z}) (0 \leq x < 10 \wedge P(x) \wedge \neg((\exists x : \mathbb{Z}) (0 \leq x < 10 \wedge Q(x))))) \}$$

4. Funciones auxiliares

Ejercicio 20. ★ Escriba los siguientes predicados y funciones en el lenguaje de especificación:

a) $\text{aux } \text{suc } (x : \mathbb{Z}) : \mathbb{Z}$, que corresponde al sucesor de x .

b) $\text{aux } \text{suma } (x, y : \mathbb{R}) : \mathbb{R}$, que corresponda a la suma entre x e y .

c) $\text{aux } \text{producto } (x, y : \mathbb{R}) : \mathbb{R}$, que corresponde al producto entre x e y .

d) $\text{pred } \text{esCuadrado } (x : \mathbb{Z})$ que sea verdadero si y solo si x es un numero cuadrado.

e) $\text{pred } \text{esPrimo } (x : \mathbb{Z})$ que sea verdadero si x es primo.

f) $\text{pred } \text{sonCoprimos } (x, y : \mathbb{Z})$ que sea verdadero si y solo si x e y son coprimos.

g) $\text{pred } \text{divisoresGrandes } (x, y : \mathbb{Z})$ que sea verdadero cuando todos los divisores de x , sin contar el uno, son mayores que y .

h) $\text{pred } \text{mayorPrimoQueDivide } (x : \mathbb{Z}, y : \mathbb{Z})$ que sea verdadero si y es el mayor primo que divide a x .

i) $\text{pred } \text{sonPrimosHermanos } (x : \mathbb{Z}, y : \mathbb{Z})$ que sea verdadero cuando x es primo, y es primo, y son primos consecutivos

1. Práctica 1 : Lógica

1.1. Lógica binaria

1.
 - a) Incorrecta
 - b) Incorrecta
 - c) Incorrecta
 - d) Correcta, pero parentesis innecesarios
 - e) Incorrecta
 - f) Correcta
 - g) Correcta, pero parentesis innecesarios
 - h) Correcta
 - i) Correcta, aunque podría usar \leftrightarrow
2.
 - a) Correcta
 - b) Correcta
 - c) Incorrecta, no existe un \forall para \mathbb{Z}
 - d) Correcta
 - e) Incorrecta, no se puede comparar un Bool (z) con un Int (0)
 - f) Incorrecta, no se puede usar un $<$ con Bool
3. Es correcta ya que $3+7$ es un Float, luego $\pi -8$ es un Float también, por lo tanto la comparación $3+7 = \pi - 8$ es igual a False, luego queda $\text{False} \wedge \text{True}$. Ergo todo correcto
4.
 - a) True
 - b) True
 - c) False
 - d) True
 - e) True
 - f) True
 - g) False
5. Si quieren ver cómo se resuelven las tablas tiene este [link](#) para ver su resolución.
 - a) Tautología
 - b) Contradicción
 - c) Tautología
 - d) Contingencia
 - e) Tautología
 - f) Tautología
 - g) Tautología
 - h) Tautología
 - i) Tautología
6.
 - a) $\text{False} \rightarrow \text{True}$
 - b) $(p \wedge q) \rightarrow (p \vee q)$
 - c) $\text{True} \leftrightarrow \text{True}$

- d) $(p \wedge q) \rightarrow q$
e) False \leftrightarrow False
f) $p \rightarrow (p \vee q)$
g) No hay relación de fuerza
h) No hay relación de fuerza
7. a) $((\neg p \vee \neg q) \vee (p \wedge q)) \rightarrow (p \wedge q) \equiv (\neg(p \vee q) \vee (p \wedge q)) \rightarrow (p \wedge q) \equiv True \rightarrow (p \wedge q) \equiv (p \wedge q)$
 \uparrow
por Morgan $(\neg p \vee \neg q) \equiv \neg(p \wedge q)$
- b) $(p \vee q) \wedge (p \vee r) \stackrel{Distributiva}{\equiv} p \vee (q \wedge r) \equiv \neg p \rightarrow (q \wedge r)$
 \uparrow
Pues $\neg a \vee b \equiv a \rightarrow b$
- c) $\neg(\neg p) \rightarrow (\neg(\neg p \wedge \neg q)) \stackrel{Morgan}{\equiv} p \rightarrow (p \wedge q) \equiv \neg p \vee (p \vee q) \equiv True \vee q \equiv True$ Lo cuál
 \uparrow
Pues $\neg a \vee b \equiv a \rightarrow b$
es distinto de q, por lo tanto no son equivalentes
- d) $((True \wedge p) \wedge (\neg p \vee False)) \rightarrow \neg(\neg p \vee q) \stackrel{Distributiva}{\equiv} (True \wedge p \wedge False) \vee (\neg p \wedge p \wedge True) \equiv$
False False $\neq (p \vee \neg q)$, por lo tanto no son equivalentes.
- e) $(p \vee (\neg p \wedge q)) \stackrel{Distributiva}{\equiv} (p \vee \neg p) \wedge (p \vee q) \equiv True \wedge (p \vee q) \equiv p \vee q \equiv \neg p \rightarrow q$
 \uparrow
Pues $\neg a \vee b \equiv a \rightarrow b$
Son equivalentes
- f) $\neg(p \wedge (q \wedge s)) \stackrel{Morgan *2}{\equiv} (\neg p \vee \neg q) \vee \neg s \equiv s \rightarrow (\neg p \vee \neg q)$ Son equivalentes
 \uparrow
Pues $\neg a \vee b \equiv a \rightarrow b$
- g) $p \rightarrow (q \wedge \neg(q \rightarrow r)) \equiv \neg p \vee (q \wedge (q \wedge \neg r)) \stackrel{Distributiva}{\equiv} (\neg p \vee q) \wedge (\neg p \vee (q \wedge \neg r))$
 \uparrow
Pues $\neg a \vee b \equiv a \rightarrow b +$ Morgan
Son equivalentes
8. a) $\neg a$
b) $a \wedge b \stackrel{Morgan}{\equiv} \neg(\neg a \wedge \neg b)$
c) $a \vee b$
d) $a \rightarrow b \equiv \neg a \vee b$
e) $a \leftrightarrow b \stackrel{Morgan}{\equiv} \neg(a \vee b) \vee \neg(\neg a \vee \neg b)$
Cómo cualquier conectivo se puede escribir a partir de \vee , \neg entonces cualquier derivada de los mismos se puede escribir meramente con estos dos simbolos.
9. f = "es fin de semana", e = "Juan estudia", m = "Juan escucha música"
- a) 1) $f \rightarrow (e \vee m) \wedge \neg(e \wedge m)$
2) $\neg f \rightarrow \neg e$
3) $(f \wedge e) \rightarrow m$
- b) Haciendo la tabla de verdad de
 $((f \rightarrow (e \vee m) \wedge \neg(e \wedge m)) \wedge (\neg f \rightarrow \neg e) \wedge ((f \wedge e) \rightarrow m)) \leftrightarrow \neg e$ podemos probar que es cierto lo que pregunta el enunciado.
10. j = "conocer a Juan", c = "Conocer a Camila", g = "Conocer a Gonzalo". Además estas afirmaciones son ciertas
- a) $j \rightarrow c$
b) $j \rightarrow (c \rightarrow g)$
- Para responder la pregunta alcanza con hacer la tabla de verdad de
 $((j \rightarrow c) \wedge (j \rightarrow (c \rightarrow g))) \rightarrow (j \rightarrow g)$ y ver si es verdadera.

11. $p = \text{pelea}$, $o = \text{ojo morado}$. Si $o = \text{True} \rightarrow P = \text{True}$ o False , no necesariamente True . Si yo digo por ejemplo "Si estudias entonces vas a aprobar", esto solo significa que si estudio entonces debo aprobar. Pero si aprobé puedo haber estudiado cómo no, pues si no estudio también puedo aprobar.

1.2. Lógica ternaria

12. a) Verdadero
 b) Verdadero
 c) Indefinido, $0^{-1} = \infty$
 d) Falso
 e) Indefinido, abierto a debate
 f) Indefinido, $\log(0) = \infty$
 g) Como el enunciado dice que trabajamos con los reales $\sqrt{-1}$ esta indefinido, por lo tanto toda la expresión se indefine.
 h) Como el enunciado dice que trabajamos con los reales $\sqrt{-1}$ esta indefinido, por lo tanto toda la expresión se indefine.
 i) $\tan(\pi/2) = \infty$, Indefinido
13. $+14 +15 I = \text{Indefinido}$, los conectivos con luego sirven para hacer un "cortocircuito", a diferencia de sus contrapartes binarias que se indefinen **siempre** que aparece un Indefinido. Ergo ahora si importa el orden en el cuál aparecen los valores de verdad en el \wedge y \vee .

P	Q	\vee_L	\wedge_L	\rightarrow_L
V	V	V	V	V
V	F	V	F	F
V	I	V	V	I
F	F	F	F	V
F	V	V	F	V
F	I	I	F	V
I	V	I	I	I
I	F	I	I	I
I	I	I	I	I

16. $b=c = \text{True}$, $a = \text{False}$, $x=y=\text{Indefinido}$

- a) Indefinido
- b) True
- c) False
- d) True
- e) True
- f) True
- g) False

17. $p, q \neq \text{Indefinido} \wedge (q = \text{True}) \rightarrow (r = \text{Indefinido})$

- a) $((p \vee q) \vee_L r)$
- b) $\neg((p \vee q) \vee_L r)$
- c) $(q \wedge \neg p) \vee (p \wedge_L \neg r) \vee_l r$

- d) $(p \wedge q) \wedge_L ((p \wedge q) \vee_L r)$
 e) $((\neg q \wedge_L \neg r) \rightarrow_L r) \wedge (p \leftrightarrow p) \wedge \neg q$
18. a + b)
- a) Ligada: x , Libres : n=1, y=1, z = 1
 - b) Ligadas : x,y , Libres : n=1, m=1
 - c) Ligada: j , Siempre Falsa
 - d) Ligada: j , Libres : s = True, a =-1 , b=1
 - e) Ligada: j , Siempre Verdadera
 - f) Ligada: j , Su valor de verdad depende de P(j)
 - g) Ligada : j, Su valor de verdad depende de P(j)
- 19.
- a) "Todos los naturales menores a 10", en vez de $0 \leq x < 10$ iria $0 \leq x \leq 10$. Tomando a los \mathbb{N} y no a \mathbb{N}_0 . En esta expresión $P(x) \wedge Q(x)$ debería ir un entonces $P(x) \rightarrow Q(x)$
 - b) Hay un problema de parentesis , y el mismo .eror"de tomar \mathbb{N}_0 y no a \mathbb{N} . Versión corregida:
 $\neg((\exists x : \mathbb{Z})(0 < x < 10 \wedge (P(x) \wedge Q(x))))$
- 20.
- a) aux suc (x : \mathbb{Z}) : $\mathbb{Z} = x + 1$;
 - b) aux suma (x,y : \mathbb{R}) : $\mathbb{R} = x + y$;
 - c) aux producto (x,y : \mathbb{R}) : $\mathbb{R} = x * y$;
 - d) pred esCuadrado (x : \mathbb{Z}) $\{(x \geq 0) \wedge (\exists y : \mathbb{Z})(y * y = x)\}$
 - e) pred esPrimo (x: \mathbb{Z}) $\{(x > 1) \wedge (\forall y : \mathbb{Z})(1 < y < x \rightarrow_L x \text{ mód } y \neq 0)\}$
 - f) pred sonCoprimos (x,y : \mathbb{Z}) $\{(\forall z : \mathbb{Z})((z \neq 1 \wedge z \neq -1) \rightarrow_L \neg((x \text{ mód } z == 0) \wedge (y \text{ mód } z == 0)))\}$
 - g) pred divisoresGrandes (x, y : \mathbb{Z}) $\{(\forall z : \mathbb{Z})((z \neq 1) \wedge ((x \text{ mód } z == 0) \rightarrow_L (z > y)))\}$
 - h) pred mayorPrimoQueDivide (x,y : \mathbb{Z}) $\{\neg(\exists z : \mathbb{Z}) ((z > y \wedge esPrimo(z)) \wedge (x \text{ mód } z == 0)) \wedge (esPrimo(y) \wedge z \text{ mód } y == 0)\}$
 - i) pred primosHermanos (x,y : \mathbb{Z}) $\{(esPrimo(x) \wedge esPrimo(y) \wedge (x - y = 2 \vee y - x = 2))\}$

1.3. Bonus Track: Diferencia entre Booleanos y Valores de Verdad

Algo muy fácil de confundir son los booleanos y los valores de verdad. Los booleanos representan un **tipo de dato** binario. Cómo son *True* \vee *False* podrían ser *Hamburguesa* \vee *Pancho* u otro conjunto de dos datos. En cambio los valores de verdad representan *Verdadero* \vee *Falso* , ninguna otra cosa. Por ejemplo una variable *Bool* *p* tiene un valor *True*, no *Verdadero* . De igual manera si *p* = *True*, *q* = *False* entonces la expresión *p* \wedge *q* tiene un valor *Falso* , no *False*. Para ir cerrando esta diferencia también la podemos ver en los *pred* y *aux*, ya que por un lado los *pred* devuelven valores de verdad, y las *aux* pueden devolver un *Bool*.



1. Secuencias

Ejercicio 1. ★ Evaluar las siguientes expresiones:

- a) $|\langle 4, 3, 1 \rangle|$
- b) $\text{addFirst}(\pi, \langle 2, 3, 5, 7, 11 \rangle)$
- c) $\langle 0, 1, 2, 3 \rangle[3]$
- d) $\text{concat}(\langle 2, 3 \rangle, \langle 5, 7, 11 \rangle)$
- e) $\text{head}(\text{tail}(\langle 5, 6, 7, 8 \rangle))$
- f) $\text{subseq}(\langle 2, 3, 5, 7, 11 \rangle, 0, 3)$
- g) $\pi \in \langle 2, 3, 5, 7, 11 \rangle$
- h) $\text{subseq}(\langle 2, 3, 5, 7, 11 \rangle, 3, 2)$
- i) $1 \in \langle 1, 2, 3, 4, 5 \rangle$
- j) $\text{subseq}(\langle 2, 3, 5, 7, 11 \rangle, 0, 65536)$

Ejercicio 2. ★ Sea x de tipo $\text{seq}(\mathbb{Z})$. ¿Cuáles de las siguientes igualdades sobre secuencias son válidas?

- a) $|x| = |\text{tail}(x)| + 1$
- b) $x = \text{subseq}(x, 0, |x| - 1)$
- c) $x = \text{subseq}(x, 0, |x|)$
- d) $\text{concat}(\text{addFirst}(3, x), y) = \text{addFirst}(3, \text{concat}(x, y))$
- e) $x = \text{addFirst}(\text{head}(x), \text{tail}(x))$
- f) $x[0] = \text{head}(x)$
- g) $i \in x = \text{head}(\text{subseq}(x, i, i + 1))$
- h) $\text{tail}(x) = \text{subseq}(x, 1, |x|)$

En los casos incorrectos, ¿puede dar condiciones sobre las listas en cuestión para que lo sean?

Ejercicio 3. ★ Sea s_0, s_1 secuencias de tipo T y e un elemento de tipo T . Indicar para cada una de las siguientes afirmaciones si son verdaderas o falsas. En caso de ser falsa, mostrar un contraejemplo.

- a) $|\text{addFirst}(e, s_0)| = 1 + |s_0|$
- b) $|\text{addFirst}(e, s_0)| = |\text{tail}(s_0)|$
- c) $|\text{concat}(s_0, s_1)| = |s_0| + |s_1|$
- d) $s_0 = \text{tail}(\text{addFirst}(e, s_0))$
- e) $\text{head}(\text{addFirst}(e, s_0)) = e$
- f) $\text{addFirst}(e, s_0) = \text{tail}(s_0)$
- g) $\text{head}(\text{addFirst}(e, \text{tail}(s_0))) = \text{head}(\text{tail}(\text{addFirst}(e, s_0)))$
- h) $\text{addFirst}(e, s_0)[0] = e$
- i) $\text{addFirst}(e, s_0)[0] = \text{head}(\text{addFirst}(e, s_0))$

Ejercicio 4. ★ Escriba las siguientes predicados auxiliares sobre secuencias de enteros, aclarando los tipos de los parámetros que recibe:

- a) *estáAcotada*, que determina si todos los elementos de una secuencia están dentro del rango $[1, 100]$.
- b) *capicúa*, que es verdadera si una secuencia es capicúa. (Por ejemplo, $\langle 0, 2, 1, 2, 0 \rangle$ es capicúa y $\langle 0, 2, 1, 4, 0 \rangle$ no).
- c) *esPrefijo*, que es verdadera si una secuencia es prefijo de otra.

- d) *estáOrdenada*, que es verdadera si la secuencia está ordenada de menor a mayor.
- e) *todosPrimos*, que es verdadera si todos los elementos de la secuencia son números primos.
- f) *primosEnPosicionesPares*, que es verdadero si todos los elementos primos de una secuencia están en una posición par.
- g) *todosIguales*, que es verdadera si todos los elementos de la secuencia son iguales.
- h) *hayUnoParQueDivideAlResto*, que determina si hay un elemento par en la secuencia que divide a todos los otros elementos de la secuencia.
- i) *hayUnoEnPosiciónParQueDivideAlResto*, que determina si hay un elemento en una posición par de la secuencia que divide a todos los otros elementos contenidos en la secuencia.
- j) *sinRepetidos*, que determina si la secuencia no tiene repetidos.
- k) *otroMayorADerecha*, que determina si todo elemento de la secuencia, salvo el último, tiene otro mayor a su derecha.
- l) *todoEsMúltiplo*, que determina si todo elemento de la secuencia es múltiplo de algún otro.
- m) *enTresPartes*, que determina si en la secuencia aparecen (de izquierda a derecha) primero 0s, después 1s y por último 2s. Por ejemplo $\langle 0, 0, 1, 1, 1, 2 \rangle$ cumple con *enTresPartes*, pero $\langle 0, 1, 3, 0 \rangle$ o $\langle 0, 0, 0, 1, 1 \rangle$ no. ¿Cómo modificaría la expresión para que se admitan cero apariciones de 0s, 1s y 2s (es decir, para que por ejemplo $\langle 0, 0, 0, 1, 1 \rangle$ o $\langle \rangle$ sí cumplan *enTresPartes*)?
- n) *esPermutaciónOrdenada*, que dadas dos secuencias s y t sea verdadero si s es permutación de t y está ordenada.

Ejercicio 5. Especificar las siguientes funciones y predicados auxiliares. En caso de no ser posible, explicar las razones.

- a) *aux intercambiarPrimeroPorUltimo(s : seq (\mathbb{Z})) : seq (\mathbb{Z})* . Que intercambia el último valor por el primero en una secuencia.
- b) *pred esReverso(s : seq (\mathbb{Z}) , t : seq (\mathbb{Z}))*. Que indica si la secuencia s es el reverso de la secuencia t .
- c) *aux reverso(s : seq (\mathbb{Z})) : seq (\mathbb{Z})* . Que indica el reverso de una secuencia.
- d) *aux agregarTresCeros(s : seq (\mathbb{Z})) : seq (\mathbb{Z})* . Que agrega 3 ceros al final de la secuencia s .
- e) *aux agregarNCeros(s : seq (\mathbb{Z}) , n : \mathbb{Z}) : seq (\mathbb{Z})* . Que agrega n ceros al final de la secuencia s .
- f) *aux sumarUno(s : seq (\mathbb{Z})) : seq (\mathbb{Z})* . Que suma 1 a cada uno de los elementos de la secuencia s .
- g) *aux ordenar(s : seq (\mathbb{Z})) : seq (\mathbb{Z})* . Que ordena la lista de menor a mayor.

Ejercicio 6. ★ Sean $P(x : \mathbb{Z})$ y $Q(x : \mathbb{Z})$ dos predicados cualesquiera que nunca se indefinen y sea s una secuencia de enteros. Escribir el predicado asociado a cada uno de los siguientes enunciados:

- a) “Si un entero en s cumple P , también cumple Q ”
- b) “Todos los enteros de s que cumplen P , no cumplen Q ”
- c) “Todos los enteros de s que están en posiciones pares y cumplen P , no cumplen Q ”
- d) “Todos los enteros de s que cumplen P y están en posiciones que cumplen Q , son pares”
- e) “Si hay un entero en s que no cumple P entonces ninguno en s cumple Q ”
- f) “Si hay un entero en s que no cumple P entonces ninguno en s cumple Q ; y si todos los enteros de s cumplen P entonces hay al menos dos elementos de s que cumplen Q ”

Ejercicio 7. Sea $P(x : \mathbb{Z})$ un predicado cualquiera y s una secuencia de enteros. Explicar cuál es el error de traducción a fórmulas de los siguientes enunciados. Dar un ejemplo en el cuál sucede el problema y luego corregirlo.

- a) “Todo elemento en una posición válida de la secuencia cumple P ”: $(\forall i : \mathbb{Z})((0 \leq i < |s|) \wedge_L P(s[i]))$
- b) “Algún elemento en una posición válida de la secuencia cumple P ”: $(\exists i : \mathbb{Z})((0 \leq i < |s|) \rightarrow_L P(s[i]))$

Ejercicio 8. ★

Sean $P(x : \mathbb{Z})$ y $Q(x : \mathbb{Z})$ dos predicados cualesquiera que nunca se indefinen, sea s una secuencia de enteros y sean a, b y k enteros. Decidir en cada caso la relación de fuerza entre las dos fórmulas:

- a) $P(3)$ y $(\forall k : \mathbb{Z})((0 \leq k < 10) \rightarrow P(k))$
- b) $P(3)$ y $k > 5 \wedge (\forall i : \mathbb{Z})((0 \leq i < k) \rightarrow P(i))$
- c) $(\forall n : \mathbb{Z})((n \in s \wedge P(n)) \rightarrow Q(n))$ y $(\forall n : \mathbb{Z})((n \in s) \rightarrow Q(n))$
- d) $(\exists n : \mathbb{Z})(n \in s \wedge P(n) \wedge Q(n))$ y $(\forall n : \mathbb{Z})((n \in s) \rightarrow Q(n))$
- e) $(\exists n : \mathbb{Z})(n \in s \wedge P(n) \wedge Q(n))$ y $|s| > 0 \wedge ((\forall n : \mathbb{Z})((n \in s) \rightarrow Q(n)))$
- f) $(\exists n : \mathbb{Z})(n \in s \wedge P(n) \wedge Q(n))$ y $(\forall n : \mathbb{Z})(n \in s \rightarrow (P(n) \wedge Q(n)))$

Ejercicio 9. Sea s una secuencia de enteros. Determinar si los siguientes pares de expresiones son equivalentes. En caso de que no lo sean, ilustrar con ejemplos.

- a) ■ $(\forall i : \mathbb{Z})((0 \leq i < |s|) \rightarrow_L ((\forall j : \mathbb{Z})(0 \leq j < |s|) \wedge i < j) \rightarrow_L s[i] < s[j])$ y
■ $(\forall j : \mathbb{Z})((0 \leq j < |s|) \rightarrow_L ((\forall i : \mathbb{Z})(0 \leq i < |s|) \wedge i < j) \rightarrow_L s[i] < s[j])$
- b) ■ $(\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L ((\exists j : \mathbb{Z})((0 \leq j < |s|) \wedge i < j - 1) \wedge_L TodosIguales(\text{subseq}(s, i, j))))$ y
■ $(\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L ((\exists i : \mathbb{Z})((0 \leq i < |s|) \wedge i < j - 1) \wedge_L TodosIguales(\text{subseq}(s, i, j))))$.
donde *todosIguales* es el definido en el ítem e) del ejercicio 4.
- c) ■ $(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L ((\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L s[i] = s[j]))$ y
■ $(\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |s|) \rightarrow_L s[i] = s[j]))$

2. Sumatorias y Productorias

Ejercicio 10. ★ Evaluar las siguientes expresiones:

- | | |
|---|--|
| a) $\sum_{i=0}^2 \langle 4, 3, 1 \rangle[i]$ | f) $\sum_{i=15}^2 \langle 2, 3, 5, 7, 11 \rangle[i]$ |
| b) $\sum_{i=0}^0 \langle \pi, 2, 3, 5, 7, 11 \rangle[i]$ | g) $\sum_{i=2}^{15} \langle 2, 3, 5, 7, 11 \rangle[i]$ |
| c) $\sum_{i=0}^{-1} \langle 1, 2, 3, 4, 5 \rangle[i]$ | h) $\sum_{i=1}^3 \langle 2, 3, 5, 7, 11 \rangle[i]$ |
| d) $\sum_{i=0}^5 \frac{1}{i}$ | i) $\sum_{i=0}^4 \langle 1, 1, 1, 1, 1 \rangle[i]$ |
| e) $\sum_{i=0}^{\sqrt{-1}} \langle 2, 3, 5, 7, 11 \rangle[i]$ | j) $\sum_{i=0}^4 \langle 0, 0, 0, 0, 0 \rangle[i]$ |

Ejercicio 11. ★ Escribir un predicado que usando sumatorias indique si un número entero es primo .

Ejercicio 12. Sea s una secuencia de elementos de tipo \mathbb{Z} . Escribir una expresión tal que:

- a) Cuente la cantidad de veces que aparece el elemento e de tipo \mathbb{Z} en la secuencia s .
- b) Sume los elementos en las posiciones impares de la secuencia s .
- c) Sume los elementos mayores a 0 contenidos en la secuencia s .
- d) Sume los inversos multiplicativos ($\frac{1}{x}$) de los elementos contenidos en la secuencia s distintos a 0.
- e) Cuente la cantidad de elementos primos no repetidos en la secuencia s .

Ejercicio 13. Escribir un predicado que indique si una secuencia es permutación de otra secuencia. Una secuencia es permutación de otra secuencia si ambas secuencias poseen los mismos elementos y la misma cantidad de apariciones por elemento. Ejemplos:

- $\langle 1, 2, 3 \rangle$ es permutación de $\langle 3, 2, 1 \rangle$
- $\langle 1, 2, 3 \rangle$ es permutación de $\langle 1, 2, 3 \rangle$
- $\langle 1, 1, 2, 3 \rangle$ es permutación de $\langle 3, 2, 1, 1 \rangle$
- $\langle 1, 2, 3 \rangle$ no es permutación de $\langle 1, 1, 3 \rangle$
- $\langle 1, 1, 2, 3 \rangle$ es permutación de $\langle 1, 3, 2, 1 \rangle$

Ejercicio 14. ★ Sea m una secuencia de secuencias de tipo \mathbb{Z} , escribir una expresión tal que:

- a) Sume los elementos contenidos en todas las secuencias.
- b) Cuente la cantidad de secuencias vacías
- c) Sume el valor del último elemento de cada secuencia no vacía
- d) Retorne True si todas las secuencias poseen el mismo tamaño.
- e) Retorne la suma de todas las posiciones impares de cada secuencia.

Ejercicio 15. Sea s un *String*, escribir una expresión que cuente la cantidad de apariciones del carácter vacío (' ').

Ejercicio 16. ★ Sea s un *String*, escribir una expresión que cuente la cantidad de apariciones de un dígito (caracteres '0' al '9').

1. Práctica 2 : Secuencias

1.1. Secuencias

1.
 - a) 3
 - b) $\langle \pi, 2, 3, 5, 7, 11 \rangle$
 - c) 3
 - d) $\langle 2, 3, 5, 7, 11 \rangle$
 - e) 6
 - f) $\langle 2, 3, 5, \rangle$
 - g) False
 - h) Indefinido
 - i) True
 - j) Indefinido
2.
 - a) Válida
 - b) Inválida, es imposible que se cumpla
 - c) Válida
 - d) Válida
 - e) Válida
 - f) Válida
 - g) Inválida, aun si x fuera una $seq(Bool)$ no funcionaria ya que \in no está definida para datos de distintos tipos. (puesto que i es de tipo \mathbb{Z})
 - h) Válida
3. $s_0, s_1 : seq(T)$ $e : T$
 - a) Verdadera
 - b) Falsa, $|addFirst(1, \langle 2 \rangle)| = |tail(\langle 2 \rangle)| \leftrightarrow 2 = 0 \leftrightarrow False$
 - c) Verdadera
 - d) Verdadera
 - e) Verdadera
 - f) Falsa, $addFirst(1, \langle 2 \rangle) = tail(\langle 2 \rangle) \leftrightarrow \langle 1, 2 \rangle = \langle \rangle \leftrightarrow False$
 - g) Falso, $head(addFirst(1, tail(\langle 2 \rangle))) = head(tail(addFirst(1, \langle 2 \rangle))) \leftrightarrow head(\langle 2 \rangle) = head(\langle \rangle) \leftrightarrow 2 = 1 \leftrightarrow False$
 - h) Verdadera
 - i) Verdadera
4. $\text{pred rango } (s : seq(\mathbb{Z}), i : \mathbb{Z}) \{0 \leq i < |s|\}$
 - a) $\text{pred estaAcotada } (s : seq(\mathbb{Z})) \{ (\forall i : \mathbb{Z})(rango(s, i) \rightarrow_L 1 \leq s[i] \leq 100) \}$
 - b) $\text{pred capicua } (s : seq(\mathbb{Z})) \{ (\forall i : \mathbb{Z})(rango(s, i) \rightarrow_L s[i] == s[|s| - 1 - i]) \}$

- c) `pred esPrefijo (s,t : seq<Z>) {
 $(\exists i : Z)(0 \leq i \leq |s| \wedge_L subseq(s, 0, i) == t)$
 }`
- d) `pred estaOrdenada (s : seq<Z>) {
 $(\forall i : Z)(0 \leq i < |s| - 1 \rightarrow_L s[i] \leq s[i + 1])$
 }`
- e) `pred todosPrimos (s : seq<Z>) {
 $(\forall i : Z) (rango(s, i) \rightarrow_L esPrimo(s[i]))$
 }`
- f) `pred primosEnPosicionesPares (s : seq<Z>) {
 $(\forall i : Z) (rango(s, i) \rightarrow_L ((i \text{ m\'od } 2 == 0 \wedge esPrimo(s[i])) \vee ((i \text{ m\'od } 2 == 1 \wedge \neg esPrimo(s[i])))$
)`
- g) `pred todasIguales (s : seq<Z>) {
 $(\forall i : Z)(0 \leq i < |s| - 1 \rightarrow_L s[i] = s[i + 1])$
 }`
- h) `pred hayUnoParQueDivideAlResto (s : seq<Z>) {
 $((\exists j : Z)(\forall i : Z) (rango(s, i) \rightarrow_L (rango(s, j) \wedge_L s[j] \text{ m\'od } 2 == 0 \wedge_L s[i] \text{ m\'od } s[j] == 0))$
)`
- i) `hayUnoEnPosicionParQueDivideAlResto (s : seq<Z>) {
 $\exists j : Z ((\exists j : Z) (rango(s, i) \rightarrow_L (rango(s, j) \wedge_L j \text{ m\'od } 2 == 0 \wedge_L s[i] \text{ m\'od } s[j] == 0)))$
 }`
- j) `pred sinRepetidos (s : seq<Z>) {
 $(\forall i, j : Z)((rango(s, i) \wedge rango(s, j) \wedge i \neq j) \rightarrow_L s[i] \neq s[j])$
 }`
- k) `pred otroMayorADerecha (s : seq<Z>) {
 $(\forall i : Z) ((\exists j : Z)(rango(s, i) \rightarrow_L (rango(s, j) \wedge_L j > i \wedge_L s[i] < s[j])))$
)`
- l) `pred todoEsMultiplo (s : seq<Z>) {
 $(\forall i : Z) ((\exists j : Z)(rango(s, i) \rightarrow_L (rango(s, j) \wedge_L j \neq i \wedge_L s[i] \text{ m\'od } s[j] ==)))$
)`
- m) `pred enTresPartes (s : seq<Z>) {estaOrdenada(s)
 $\wedge (\forall i : Z)(0 \leq i \leq 2 \leftrightarrow i \in s)$
 }`
- n) Este ejercicio es sumamente complicado y por lo tanto necesite m\'ultiples predicados para poder resolverlo, si se pudiera usar una sumatoria saldr\'ia mucho m\'as sencillo, pero al no poderse tuve que recurrir a esto.
- ```
pred esPermutacionOrdenada (s, t : seq<Z>) {

 esPermutacion(s, t) \wedge estaOrdenada(s)
}
```
- Esto surge de la idea de hacer traspo(  $\langle 9, -5, 23 \rangle$  = secuencia a modificar ,  $\langle 1, 0, 2 \rangle$  = nuevo indice ) =  $\langle -5, 9, 23 \rangle$
- ```
pred esPermutacion (s, t : seq<Z>) { $|s| = |t| \wedge_L$   

 $(\exists indi : seq<Z>)(|indi| = |s| \wedge indicePermutacion(indi) \wedge (\forall i : Z)(0 \leq i < s \rightarrow_L s[i] = t[indi[i]]))$   

    }
```
- Estos van a ser los indices para una nueva trasposicion. pred indicePermutacion (newIndex : seq<Z>) {

$(\forall i : \mathbb{Z})(rango(newIndex, i) \rightarrow_L 0 \leq newIndex[i] < |newIndex|) \wedge sinRepetidos(newIndex)$
 }

Si es que se pudiera usar la sumatoria entonces haría de esta forma el predicado.

```
pred esPermutacion (s,t : seq<\mathbb{Z}>) {
    |s| = |t| \wedge (\forall e : \mathbb{Z})((e \in s \leftrightarrow e \in t) \wedge apariciones(s,e) = apariciones(t,e))
}
```

5. a) aux interacmbiosPrimersPorUltimo (s : seq<\mathbb{Z}>) : seq<\mathbb{Z}> =
 $setAt(setAt(s, 0, s[|s| - 1]), |s| - 1, s[0])$
 ;
 b) pred esReversa (s,t : seq<\mathbb{Z}>) { $|s| = |t| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L s[|s| - 1 - i] == t[i])$ }
 c) aux reverso (s : seq<\mathbb{Z}>) : seq<\mathbb{Z}> = No se puede ;
 Si pudiera usar recursión entonces sería posible, pero al no poder y tampoco poder hacer una solución iterativa al no saber el largo de s es imposible hacer una auxiliar que cumpla lo pedido.
 d) aux agregarTresCeros (s : seq<\mathbb{Z}>) : seq<\mathbb{Z}> = $s + + \langle 0, 0, 0 \rangle$;
 e) aux agregarNCeros (s : seq<\mathbb{Z}>, n \mathbb{Z}) : seq<\mathbb{Z}> = Nosepuede ;
 Mismo razonamiento que el mismo c)
 f) + g) Mismo razonamiento que el c)

6. P(x) y Q(x) nunca se indefinen.

- a) $(\forall x : \mathbb{Z})(x \in s \wedge (P(x) \rightarrow Q(x)))$
 b) $(\forall x : \mathbb{Z})(x \in s \wedge (P(x) \rightarrow \neg Q(x)))$
 c) $(\forall i : \mathbb{Z})((rango(s, i) \wedge i \text{ m\'od } 2 == 0) \wedge_L P(s[i]) \rightarrow \neg Q(s[i]))$
 d) $(\forall i : \mathbb{Z})((rango(s, i) \wedge_L P(s[i]) \wedge Q(i)) \rightarrow_L s[i] \text{ m\'od } 2 == 0)$
 e) $(\exists j : \mathbb{Z})(j \in s \wedge \neg P(j))$
 f) $(\exists j : \mathbb{Z})(j \in s \wedge \neg P(j) \rightarrow (\forall x : \mathbb{Z})(x \in s \rightarrow \neg Q(x))) \wedge (\forall m : \mathbb{Z})((m \in s \wedge P(s)) \rightarrow (\exists r, t : \mathbb{Z})(r \in s \wedge t \in s \wedge r \neq t \wedge Q(r) \wedge Q(t)))$

7. a) Si $i > 0$ entonces la formula devuelve falso, ergo siempre es falsa. Versión corregida :
 $(\forall i : \mathbb{Z})(rango(s, i) \rightarrow_L P(s[i]))$
 b) Si $i < 0$ entonces la formula devuelve verdadero, por lo tanto siempre es verdadera.
 $(\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L P(s[i]))$

8. Decir A \rightarrow B es equivalente a "A es más fuerte que B".

- a) $(\forall k : \mathbb{Z})(0 \leq k < 10 \rightarrow P(k)) \rightarrow P(3)$
 b) $(k > 5 \wedge (\forall i : \mathbb{Z})(0 \leq i < 1k \rightarrow P(i))) \rightarrow P(3)$
 c) $(\forall n : \mathbb{Z})((n \in s \wedge P(n)) \rightarrow Q(n))$
 d) NO hay relacion de fuerza
 e) No hay relaci\'on de fuerza
 f) $(\forall n : \mathbb{Z})(n \in s \rightarrow (Q(n) \wedge P(n))) \rightarrow (\exists n : \mathbb{Z})(n \in s \wedge P(n) \wedge Q(n))$

9. a) Son equivalentes, meramente cambian el orden en el que aparecen los cuantificadores, pero al ser los dos cuantificadores iguales el orden es indistintos.
 b) Son equivalentes, mismo razonamiento que el a
 c) i) Dice que para todo elemento de s existe existe un elemento de s que es igual a el (por ej: si mismo) ii) Dice que existe un elemento de s al que todos los elementos de s son iguales, osea todos los elementos son iguales. **Ejemplo:** s= $\langle 1, 2 \rangle$ i) Verdadero, ii) Falso

1.2. Sumatorias y Productorias

10. a) 8
 b) π
 c) 0
 d) Indefinido
 e) Indefinido
 f) 0
 g) Indefinido
 h) 15
 i) 5
 j) 0
11. pred esPrimo (n : \mathbb{Z}) { $n > 1 \wedge (\sum_{i=2}^{n-1} \text{if } n \bmod i == 0 \text{ then } 1 \text{ else } 0 \text{ fi}) == 0$ }
12. a) aux apariciones (s : $\text{seq}(\mathbb{Z})$, e : \mathbb{Z}) : $\mathbb{Z} = \sum_{i=0}^{|s|-1} \text{if } s[i] == e \text{ then } 1 \text{ else } 0 \text{ fi} ;$
 b) aux sumaImpar (s : $\text{seq}(\mathbb{Z})$) : $\mathbb{Z} = \sum_{i=0}^{|s|-1} \text{if } i \bmod 2 == 1 \text{ then } s[i] \text{ else } 0 \text{ fi} ;$
 c) aux sumaPositivos (s : $\text{seq}(\mathbb{Z})$) : $\mathbb{Z} = \sum_{i=0}^{|s|-1} \text{if } s[i] > 0 \text{ then } s[i] \text{ else } 0 \text{ fi} ;$
 d) aux sumaInversosMult (s : $\text{seq}(\mathbb{Z})$) : $\mathbb{Z} = \sum_{i=0}^{|s|-1} \text{if } s[i] \neq 0 \text{ then } 1/s[i] \text{ else } 0 \text{ fi} ;$
 e) aux sumaInversosMult (s : $\text{seq}(\mathbb{Z})$) : $\mathbb{Z} = \sum_{i=0}^{|s|-1} \text{if } \text{apariciones}(s, s[i]) == 1 \wedge \text{esPrimo}(s[i]) \text{ then } 1 \text{ else } 0 \text{ fi} ;$
13. pred esPermutacion (s,t : $\text{seq}(\mathbb{Z})$) {
 $|s| = |t| \wedge (\forall e : \mathbb{Z}) ((e \in s \leftrightarrow e \in t) \wedge \text{apariciones}(s, e) = \text{apariciones}(t, e))$
 }
14. m : $\text{seq}(\text{seq}(\mathbb{Z}))$
 a) aux sumaTodosElementos (m : $\text{seq}(\text{seq}(\mathbb{Z}))$) : $= \sum_{i=0}^{|m|-1} (\sum_{j=0}^{|m[i]|-1} m[i][j]) ;$
 b) aux secuenciasVacias (m : $\text{seq}(\text{seq}(\mathbb{Z}))$) : $\mathbb{Z} = \sum_{i=0}^{|m|-1} \text{if } |m[i]| == 0 \text{ then } 1 \text{ else } 0 \text{ fi} ;$
 c) aux sumaUltimoElemento (m : $\text{seq}(\text{seq}(\mathbb{Z}))$) : $\mathbb{Z} = \sum_{i=0}^{|m|-1} \text{if } |m[i]| != 0 \text{ then } m[i][|m[i]| - 1] \text{ else } 0 \text{ fi} ;$
 d) aux subSecusMismoTamaño (m : $\text{seq}(\text{seq}(\mathbb{Z}))$) : Bool = $(\sum_{i=0}^{|m|-1} \text{if } |m[i]| == |m[0]| \text{ then } 0 \text{ else } 1 \text{ fi}) == 0 ;$
 e) aux sumaPosisImpares (m : $\text{seq}(\text{seq}(\mathbb{Z}))$) : $\mathbb{Z} = \sum_{i=0}^{|m|-1} (\sum_{j=0}^{|m[i]|-1} \text{if } j \bmod 2 == 1 \text{ then } m[i][j] \text{ else } 0 \text{ fi}) ;$
15. aux aparaisCaracterVacio (s : String) : $\mathbb{Z} = \sum_{i=0}^{|s|-1} \text{if } s[i] == ' ' \text{ then } 1 \text{ else } 0 \text{ fi} ;$
16. Dejo dos formas de resolverlo
La primera :
 $\text{digitos} = \langle '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' \rangle$
 $\sum_{i=0}^{|s|-1} \text{if } s[i] \in \text{digitos} \text{ then } 1 \text{ else } 0 \text{ fi}$
La segunda : $\sum_{i=0}^{|s|-1} \text{if } \text{ord}('0') \leq \text{ord}(s[i]) \leq \text{ord}('9') \text{ then } 1 \text{ else } 0 \text{ fi}$



Ejercicio 1. ★ Las siguientes especificaciones no son correctas. Indicar por qué, y corregirlas para que describan correctamente el problema.

- a) **buscar:** Dada una secuencia y un elemento de ésta, devuelve en *result* alguna posición de la secuencia en la cual se encuentre el elemento.

```
proc buscar (in l: seq<R>, in elem: R, out result: Z) {
    Pre {elem ∈ l}
    Post {l[result] = elem}
}
```

- b) **progresionGeometricaFactor2:** Indica si la secuencia *l* representa una progresión geométrica factor 2. Es decir, si cada elemento de la secuencia es el doble del elemento anterior.

```
proc progresionGeometricaFactor2 (in l: seq<Z>, out result: Bool) {
    Pre {True}
    Post {result = True ↔ ((∀i : Z)(0 ≤ i < |l| →L l[i] = 2 * l[i - 1]))}
}
```

- c) **minimo:** Devuelve en *result* el menor elemento de *l*.

```
proc minimo (in l: seq<Z>, out result: Z) {
    Pre {True}
    Post {((∀y : Z)((y ∈ l ∧ y ≠ x) → y > result))}
}
```

Ejercicio 2. La siguiente no es una especificación válida, ya que para ciertos valores de entrada que cumplen la precondición, no existe una salida que cumpla con la postcondición.

```
proc elementosQueSumen (in l: seq<Z>, in suma: Z, out result: seq<Z>) {
    Pre {True}
    Post {
        /* La secuencia result está incluída en la secuencia l*/
        (∀x : Z)(x ∈ result → #apariciones(x, result) ≤ #apariciones(x, l))
        /* La suma de la result coincide con el valor suma */
        ∧ suma = ∑i=0|result|-1 result[i]
    }
}
```

- a) Mostrar valores para *l* y *suma* que hagan verdadera la precondición, pero tales que no exista *result* que cumpla la postcondición.

- b) Supongamos que agregamos a la especificación la siguiente cláusula:

```
Pre : min_suma(l) ≤ suma ≤ max_suma(l)
fun min_suma(l) : Z = ∑i=0|l|-1 if l[i] < 0 then l[i] else 0 fi
fun max_suma(l) : Z = ∑i=0|l|-1 if l[i] > 0 then l[i] else 0 fi
```

¿Ahora es una especificación válida? Si no lo es, justificarlo con un ejemplo como en el punto anterior.

c) Dar una precondición que haga correcta la especificación.

Ejercicio 3. ★ Para los siguientes problemas, dar todas las soluciones posibles a las entradas dadas.

a) proc raizCuadrada (in $x:\mathbb{R}$, out $result:\mathbb{R}$) {

Pre $\{x \geq 0\}$

Post $\{result^2 = x\}$

}

I) $x = 0$

II) $x = 1$

III) $x = 27$

b) ★

proc indiceDelMaximo (in $l: seq(\mathbb{R})$, out $result:\mathbb{Z}$) {

Pre $\{|l| > 0\}$

Post {

$0 \leq result < |l|$

$\wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] \leq l[result]))$

}

}

I) $l = \langle 1, 2, 3, 4 \rangle$

II) $l = \langle 15.5, -18, 4.215, 15.5, -1 \rangle$

III) $l = \langle 0, 0, 0, 0, 0, 0 \rangle$

c) ★

proc indiceDelPrimerMaximo ($l:seq(\mathbb{R})$, $result:\mathbb{Z}$) {

Pre $\{|l| > 0\}$

Post {

$0 \leq result < |l|$

$\wedge ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L (l[i] < l[result] \vee (l[i] = l[result] \wedge i \geq result))))$

}

}

I) $l = \langle 1, 2, 3, 4 \rangle$

II) $l = \langle 15.5, -18, 4.215, 15.5, -1 \rangle$

III) $l = \langle 0, 0, 0, 0, 0, 0 \rangle$

d) ¿Para qué valores de entrada `indiceDelPrimerMaximo` y `indiceDelMaximo` tienen necesariamente la misma salida?

Ejercicio 4. ★ Sea $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ definida como:

$$f(a, b) = \begin{cases} 2b & \text{si } a < 0 \\ b - 1 & \text{en otro caso} \end{cases}$$

¿Cuáles de las siguientes especificaciones son correctas para el problema de calcular $f(x, y)$?

Para las que no lo son, indicar por qué.

a) proc f (in a, b: \mathbb{R} , out result: \mathbb{R}) {

```
  Pre {True}
  Post {
    ( $a < 0 \wedge result = 2 * b$ )
    ^
    ( $a \geq 0 \wedge result = b - 1$ )
  }
}
```

b) proc f (in a, b: \mathbb{R} , out result: \mathbb{R}) {

```
  Pre {True}
  Post { $(a < 0 \wedge result = 2 * b) \vee (a > 0 \wedge result = b - 1)$ }
}
```

c) proc f (in a, b: \mathbb{R} , out result: \mathbb{R}) {

```
  Pre {True}
  Post { $(a < 0 \wedge result = 2 * b) \vee (a \geq 0 \wedge result = b - 1)$ }
}
```

d) proc f (in a, b: \mathbb{R} , out result: \mathbb{R}) {

```
  Pre {True}
  Post {
     $a < 0 \rightarrow result = 2 * b$ 
    ^
     $a \geq 0 \rightarrow result = b - 1$ 
  }
}
```

e) proc f (in a, b: \mathbb{R} , out result: \mathbb{R}) {

```
  Pre {True}
  Post { $(a < 0 \rightarrow result = 2 * b) \vee (a \geq 0 \rightarrow result = b - 1)$ }
}
```

f) proc f (in a, b: \mathbb{R} , out result: \mathbb{R}) {

```
  Pre {True}
  Post {result = (if  $a < 0$  then  $2 * b$  else  $b - 1$  fi)}
}
```

Ejercicio 5. ★ Considerar la siguiente especificación, junto con un algoritmo que dado x devuelve x^2 .

proc unoMasGrande (in x: \mathbb{R} , out result: \mathbb{R}) {

```
  Pre {True}
  Post {result > x}
}
```

a) ¿Qué devuelve el algoritmo si recibe $x = 3$? ¿El resultado hace verdadera la postcondición de unoMasGrande?

b) ¿Qué sucede para las entradas $x = 0.5$, $x = 1$, $x = -0.2$ y $x = -7$?

c) Teniendo en cuenta lo respondido en los puntos anteriores, escribir una precondición para unoMasGrande, de manera tal que el algoritmo sea una implementación correcta.

Ejercicio 6. ★ Sean x y r variables de tipo \mathbb{R} . Considerar los siguientes predicados:

P1: $\{x \leq 0\}$
 P2: $\{x \leq 10\}$
 P3: $\{x \leq -10\}$

Q1: $\{r \geq x^2\}$
 Q2: $\{r \geq 0\}$
 Q3: $\{r = x^2\}$

- a) Indicar la relación de fuerza entre P1, P2 y P3.
- b) Indicar la relación de fuerza entre Q1, Q2 y Q3.
- c) Sea E1 la siguiente especificación. Escribir 2 programas que cumplan con E1.

```
proc hagoAlgo (in x: ℝ, out r:ℝ) {
    Pre { $x \leq 0$ }
    Post { $r \geq x^2$ }
}
```

- d) Sea A un algoritmo que cumple con E1. Decidir si necesariamente cumple las siguientes especificaciones:
 - a) Pre: $\{x \leq -10\}$, Post: $\{r \geq x^2\}$
 - b) Pre: $\{x \leq 10\}$, Post: $\{r \geq x^2\}$
 - c) Pre: $\{x \leq 0\}$, Post: $\{r \geq 0\}$
 - d) Pre: $\{x \leq 0\}$, Post: $\{r = x^2\}$
 - e) Pre: $\{x \leq -10\}$, Post: $\{r \geq 0\}$
 - f) Pre: $\{x \leq 10\}$, Post: $\{r \geq 0\}$
 - g) Pre: $\{x \leq -10\}$, Post: $\{r = x^2\}$
 - h) Pre: $\{x \leq 10\}$, Post: $\{r = x^2\}$
- e) ¿Qué conclusión pueden sacar? ¿Qué debe cumplirse con respecto a las precondiciones y postcondiciones para que sea seguro reemplazar la especificación?

Ejercicio 7. ★ Considerar las siguientes dos especificaciones, junto con un algoritmo a que satisface la especificación de p2.

```
proc p1 (in x: ℝ, in n: ℤ, out result: ℤ) {
    Pre { $x \neq 0$ }
    Post { $x^n - 1 < result \leq x^n$ }
}

proc p2 (in x: ℝ, in n: ℤ, out result: ℤ) {
    Pre { $n \leq 0 \rightarrow x \neq 0$ }
    Post { $result = \lfloor x^n \rfloor$ }
}
```

- a) Dados valores de x y n que hacen verdadera la precondición de p1, demostrar que hacen también verdadera la precondición de p2.
- b) Ahora, dados estos valores de x y n , supongamos que se ejecuta a : llegamos a un valor de res que hace verdadera la postcondición de p2. ¿Será también verdadera la postcondición de p1?
- c) ¿Podemos concluir que a satisface la especificación de p1?

Ejercicio 8. Considerar las siguientes especificaciones:

```
proc n-esimo1 (in l: seq(ℤ), in n: ℤ, out result: ℤ) {
    Pre {
        /*Los elementos están ordenados*/
        ( $\forall i : \mathbb{Z})(0 \leq i < |l| - 1 \rightarrow_L l[i] < l[i + 1])$ 
         $\wedge 0 \leq n < |l|$ 
    }
}
```

```

    Post {result = l[n]}
}

proc n-esimo2 (in l: seq<Z>, in n: Z, out result:Z) {
    Pre {
        /*Los elementos son distintos entre sí*/
        ( $\forall i : Z$ ) ( $0 \leq i < |l| \rightarrow_L ((\forall j : Z) (0 \leq j < |l| \wedge i \neq j) \rightarrow_L l[i] \neq l[j])$ )
        ^
         $0 \leq n < |l|$ 
    }
    Post {
        result  $\in l$ 
        ^
         $n = \sum_{i=0}^{|l|-1}$  (if  $l[i] < result$  then 1 else 0 fi)
    }
}

```

¿Es cierto que todo algoritmo que cumple con n-esimo1 cumple también con n-esimo2? ¿Y al revés?
Sugerencia: Razonar de manera análoga a la del ejercicio anterior.

Ejercicio 9. ★ Especificar los siguientes problemas:

- Dado un número entero, decidir si es par.
- Dado un entero n y uno m , decidir si n es un múltiplo de m .
- Dado un número real, devolver su inverso multiplicativo.
- Dada una secuencia de caracteres, obtener de ella sólo los que son numéricos (con todas sus apariciones sin importar el orden de aparición).
- Dada una secuencia de reales, devolver la secuencia que resulta de duplicar sus valores en las posiciones impares.
- Dado un número entero, listar todos sus divisores positivos (sin duplicados).

Ejercicio 10. Considerar el problema de decidir, dados n y m enteros, si n es múltiplo de m , y la siguiente especificación.

```

proc esMultiplo (in n, m: Z, out result:Bool) {
    Pre { $m \neq 0$ }
    Post {result = ( $n \bmod m = 0$ )}
}

```

- Según la definición matemática de múltiplo, ¿tiene sentido preguntarse si 4 es múltiplo de 0? ¿Cuál es la respuesta?
- ¿Debería ser $n = 4$, $m = 0$ una entrada válida para el problema? ¿Lo es en esta especificación?
- Corregir la especificación de manera tal que $n = 4$, $m = 0$ satisfaga la precondición (¡cuidado con las indefiniciones!).
- ¿Qué relación de fuerza hay entre la precondición nueva y la original?

Ejercicio 11. Considerar el problema de, dada una secuencia de números reales, devolver la que resulta de duplicar sus valores en las posiciones impares.

- Para la secuencia $\langle 1, 2, 3, 4 \rangle$, ¿es $\langle 0, 4, 0, 8 \rangle$ un resultado correcto?
- Sea la siguiente especificación:

```

proc duplicarEnImpares (in l: seq<R>, out result: seq<R>) {
    Pre {True}
    Post {|result| = |l|  $\wedge$  ( $\forall i : Z$ ) ( $(0 \leq i < |l| \wedge i \bmod 2 = 1) \rightarrow_L result[i] = 2 * l[i]$ )}
}

```

Si $l = \langle 1, 2, 3, 4 \rangle$, ¿ $result = \langle 0, 4, 0, 8 \rangle$ satisface la postcondición?

- c) Si es necesario, corregir la especificación para que describa correctamente el resultado esperado.
- d) ¿Qué relación de fuerza hay entre la nueva postcondición y la original?

Ejercicio 12. ★ Especificar el problema de dado un entero positivo retornar una secuencia de 0s y 1s que represente ese número en base 2 (es decir, en binario).

Ejercicio 13. Con lo visto en los ejercicios 9 a 12, ¿Encuentra casos de sub y sobreespecificación en las especificaciones del ejercicio 8?

Ejercicio 14. Especificar los siguientes problemas:

- a) ★ Dado un número entero positivo, obtener la suma de sus factores primos.
- b) Dado un número entero positivo, decidir si es perfecto. Se dice que un número es perfecto cuando es igual a la suma de sus divisores (excluyéndose a sí mismo).
- c) Dado un número entero positivo n , obtener el menor entero positivo $m > 1$ tal que m sea coprimo con n .
- d) ★ Dado un entero positivo, obtener su descomposición en factores primos. Devolver una secuencia de tuplas (p, e) , donde p es un factor primo y e es su exponente, ordenada en forma creciente con respecto a p .
- e) Dada una secuencia de números reales, obtener la diferencia máxima entre dos de sus elementos.
- f) ★ Dada una secuencia de números enteros, devolver aquel que divide a más elementos de dicha secuencia. El elemento tiene que pertenecer a la secuencia original. Si existe más de un elemento que cumple esta propiedad, devolver alguno de ellos.

Ejercicio 15. Especificar los siguientes problemas sobre secuencias:

- a) proc nEsimaAparicion(in $l : seq(\mathbb{R})$, in $e : \mathbb{R}$, in $n : \mathbb{Z}$, out $result : \mathbb{Z}$), que devuelve el índice de la n -ésima aparición de e en l .
- b) Dadas dos secuencias s y t , decidir si s es una subcadena de t .
- c) ★ Dadas dos secuencias s y t , decidir si s está *incluida* en t , es decir, si todos los elementos de s aparecen en t en igual o mayor cantidad.
- d) proc mezclarOrdenado(in $s, t : seq(\mathbb{Z})$, out $result : seq(\mathbb{Z})$), que recibe dos secuencias ordenadas y devuelve el resultado de intercalar sus elementos de manera ordenada.
- e) Dadas dos secuencias s y t especificar el procedimiento *intersecciónSinRepetidos* que retorna una secuencia que contiene únicamente los elementos que aparecen en ambas secuencias.
- f) ★ Dadas dos secuencias s y t , devolver su *intersección*, es decir, una secuencia con todos los elementos que aparecen en ambas. Si un mismo elemento tiene repetidos, la secuencia retornada debe contener la cantidad mínima de apariciones en s y de t .

Ejercicio 16. Especificar los siguientes problemas:

- a) proc cantApariciones(in $l : String$, out $result : seq(Char \times \mathbb{Z})$) que devuelve la secuencia con todos los elementos de l , sin duplicados, con su cantidad de apariciones(en un orden cualquiera). Ejemplos:
 - $cantApariciones('a') = \langle ('a', 1) \rangle$
 - $cantApariciones('a', 'b', 'c') = \langle ('a', 1), ('c', 1), ('b', 1) \rangle$
 - $cantApariciones('a', 'b', 'c', 'b', 'd', 'b') = \langle ('a', 1), ('b', 3), ('d', 1), ('c', 1) \rangle$
 - $cantApariciones(\langle \rangle) = \langle \rangle$
- b) Dada una secuencia, devolver una secuencia de secuencias que contenga todos sus prefijos, en orden creciente de longitud.
- c) ★ Dada una secuencia de secuencias de enteros l , devolver una secuencia de l que contenga el máximo valor. Por ejemplo, si $l = \langle \langle 2, 3, 5 \rangle, \langle 8, 1 \rangle, \langle 2, 8, 4, 3 \rangle \rangle$, devolver $\langle 8, 1 \rangle$ o $\langle 2, 8, 4, 3 \rangle$.
- d) proc interseccionMultiple(in $ls : seq(seq(\mathbb{R}))$, out $l : seq(\mathbb{R})$) que devuelve en l el resultado de la intersección de todas las secuencias de ls .
- e) ★ Dada una secuencia l con todos sus elementos distintos, devolver la secuencia de *partes*, es decir, la secuencia de todas las secuencias incluidas en l , cada una con sus elementos en el mismo orden en que aparecen en l .

Especificación de problemas usando inout

Ejercicio 17. ★ Dados dos enteros a y b , se necesita calcular su suma, y retornarla en un entero c . ¿Cuáles de las siguientes especificaciones son correctas para este problema? Para las que no lo son, indicar por qué.

a) proc **sumar** (inout a, b, c: \mathbb{Z}) {

Pre {True}

Post { $a + b = c$ }

}

b) proc **sumar** (in a, b: \mathbb{Z} , in c: \mathbb{Z}) {

Pre {True}

Post { $c = a + b$ }

}

c) proc **sumar** (in a, b: \mathbb{Z} , out c: \mathbb{Z}) {

Pre {True}

Post { $c = a + b$ }

}

d) proc **sumar** (inout a, b: \mathbb{Z} , out c: \mathbb{Z}) {

Pre { $a = A_0 \wedge b = B_0$ }

Post { $a = A_0 \wedge b = B_0 \wedge c = a + b$ }

}

Ejercicio 18. ★ Dada una secuencia l , se desea sacar su primer elemento y devolverlo. Decidir cuáles de estas especificaciones son correctas. Para las que no lo son, indicar por qué y justificar con ejemplos.

a) proc **tomarPrimero** (inout l: $seq\langle\mathbb{R}\rangle$, out result: \mathbb{R}) {

Pre { $|l| > 0$ }

Post { $result = \text{head}(l)$ }

}

b) proc **tomarPrimero** (inout l: $seq\langle\mathbb{R}\rangle$, out result: \mathbb{R}) {

Pre { $|l| > 0 \wedge l = L_0$ }

Post { $result = \text{head}(L_0)$ }

}

c) proc **tomarPrimero** (inout l: $seq\langle\mathbb{R}\rangle$, out result: \mathbb{R}) {

Pre { $|l| > 0$ }

Post { $result = \text{head}(L_0) \wedge |l| = |L_0| - 1$ }

}

d) proc **tomarPrimero** (inout l: $seq\langle\mathbb{R}\rangle$, out result: \mathbb{R}) {

Pre { $|l| > 0 \wedge l = L_0$ }

Post { $result = \text{head}(L_0) \wedge l = \text{tail}(L_0)$ }

}

```

e) proc tomarPrimero (inout l: seq<R>, out result:R) {
    Pre { $|l| > 0 \wedge l = L_0$ }
    Post {
        result = head( $L_0$ )
         $\wedge |l| = |L_0| - 1$ 
         $\wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] = L_0[i + 1]))$ 
    }
}

```

Ejercicio 19. Considerar la siguiente especificación:

```

proc intercambiar (inout l: seq<R>, in i, j: Z) {
    Pre { $0 \leq i < |l| \wedge 0 \leq j < |l| \wedge l = L_0$ }
    Post {
        /*Las secuencias tienen la misma longitud*/
         $|l| = |L_0|$ 
        ^
        /*Intercambia i*/
         $l[i] = L_0[j]$ 
        ^
        /*Intercambia j*/
         $l[j] = L_0[i]$ 
    }
}

```

- a) ¿Esta especificación es válida? Si lo es, ¿qué problema describe?
- b) Mostrar con un ejemplo que la postcondición está sub-especificada (es decir, que hay valores que la hacen verdadera aunque no son deseables como solución).
- c) Corregir la especificación agregando a la postcondición una o más cláusulas Post : .

Ejercicio 20. Explicar coloquialmente la siguiente especificación:

```

proc copiarPrimero (inout l: seq<Z>, inout i: Z) {
    Pre {
        /*Valores iniciales*/
         $l = L_0 \wedge i = I_0$ 
        ^
        /*Secuencia no vacía*/
         $|l| > 0$ 
        ^
        /*Indice en rango*/
         $0 \leq i < |l|$ 
    }
    Post {
         $l[I_0] = L_0[0]$ 
        ^
         $i = L_0[I_0]$ 
        ^
         $((\forall j : \mathbb{Z})((0 \leq j < |l| \wedge j \neq I_0) \rightarrow_L l[j] = L_0[j]))$ 
    }
}

```

Ejercicio 21. Dada una secuencia de enteros, se requiere multiplicar por 2 aquellos valores que se encuentran en posiciones pares. Indicar por qué son incorrectas las siguientes especificaciones, y proponer una alternativa correcta.

a) proc duplicarPares (inout l: $seq\langle \mathbb{Z} \rangle$) {
Pre { $l = L_0$ }
Post {
 $|l| = |L_0|$
 \wedge
 $(\forall i : \mathbb{Z})(0 \leq i < |l| \wedge i \text{ m\'od } 2 = 0) \rightarrow_L l[i] = 2 * L_0[i]$
}
}

b) proc duplicarPares (inout l: $seq\langle \mathbb{Z} \rangle$) {
Pre { $l = L_0$ }
Post { $(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \text{ m\'od } 2 \neq 0) \rightarrow_L l[i] = L_0[i])$
 \wedge
 $(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \text{ m\'od } 2 = 0) \rightarrow_L l[i] = 2 * L_0[i])$
}
}

c) proc duplicarPares (inout l: $seq\langle \mathbb{Z} \rangle$, out result: $seq\langle \mathbb{Z} \rangle$) {
Pre { $True$ }
Post { $|l| = |result|$
 \wedge
 $(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \text{ m\'od } 2 \neq 0) \rightarrow_L result[i] = l[i])$
 \wedge
 $(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \text{ m\'od } 2 = 0) \rightarrow_L result[i] = 2 * l[i])$
}
}

Ejercicio 22. Especificar los siguientes problemas de modificación de secuencias:

- a) ★ proc primosHermanos(inout l : $seq\langle \mathbb{Z} \rangle$), que dada una secuencia de enteros mayores a dos, reemplaza dichos valores por el n\'umero primo menor m\'as cercano. Por ejemplo, si $l = \langle 6, 5, 9, 14 \rangle$, luego de aplicar $primosHermanos(l)$, $l = \langle 5, 3, 7, 13 \rangle$
- b) ★ proc reemplazar(inout l : String, in a, b : Char), que reemplaza todas las apariciones de a en l por b.
- c) proc recortar(inout l : $seq\langle \mathbb{Z} \rangle$, in a : \mathbb{Z}), que saca de l todas las apariciones de a consecutivas que aparezcan al principio. Por ejemplo $recortar(\langle 2, 2, 3, 2, 4 \rangle, 2) = \langle 3, 2, 4 \rangle$, mientras que $recortar(\langle 2, 2, 3, 2, 4 \rangle, 3) = \langle 2, 2, 3, 2, 4 \rangle$.
- d) proc intercambiarParesConImpares(inout l : String), que toma una secuencia de longitud par y la modifica de modo tal que todas las posiciones de la forma $2k$ quedan intercambiadas con las posiciones $2k + 1$. Por ejemplo, $intercambiarParesConImpares("adinkle")$ modifica de la siguiente manera: "daniel".
- e) ★proc limpiarDuplicados(inout l : $seq\langle \text{Char} \rangle$, out dup : $seq\langle \text{Char} \rangle$), que elimina los elementos duplicados de l dejando s\'olo su primera aparici\'on (en el orden original). Devuelve adem\'as, dup una secuencia con todas las apariciones eliminadas (en cualquier orden).

1. Práctica 3 : Especificación de Problemas

1.
 - a) proc buscar (in l: $seq(\mathbb{R})$, elem: \mathbb{R} , out res: \mathbb{Z}) {
 Pre $\{elem \in l\}$
 Post $\{0 \leq result < |l| \wedge_L l[result] = elem\}$
}
 - b) proc progresionGeometricaFactor2 (in l: $seq(\mathbb{Z})$, out res: Bool) {
 Pre $\{True\}$
 Post $\{result = True \leftrightarrow ((\forall i : \mathbb{Z})(1 \leq i < |l| \longrightarrow_L l[i] = 2 \cdot l[i - 1]))\}$
}
 - c) proc minimo (in l: $seq(\mathbb{Z})$, out res: \mathbb{Z}) {
 Pre $\{True\}$
 Post $\{(\forall y : \mathbb{Z})((y \in l \wedge y \neq result) \rightarrow y > result) \wedge result \in l\}$
}
2.
 - a) $l = \langle \rangle$, suma = 1
 - b) No, sigue sin ser válida. Se rompe con: $l = \langle 1, 3 \rangle$, suma = 2
 - c) Agregaría este predicado:
$$\text{pred esSumaValida (suma : } \mathbb{Z}, l : seq(\mathbb{Z})) \{$$
$$(\exists s : seq(Bool)) (|s| = |l| \wedge_L ((\sum_{i=0}^{|l|-1} \text{if } s[i] == True \text{ then } l[i] \text{ else } 0) == suma))$$
3.
 - a)
 - 1) 0
 - 2) 1,-1
 - 3) $\sqrt{27}, -\sqrt{27}$
 - b)
 - 1) 3
 - 2) 0,3
 - 3) 0,1,2,3,4,5
 - c)
 - 1) 3
 - 2) 0
 - 3) 0
 - d) Para el I)
4.
 - a) Incorrecta, pues es imposible que ambas condiciones son verdaderas
 - b) Incorrecta, pues es imposible cumplir la post si $a=0$
 - c) Correcta
 - d) Incorrecta, faltan parentesis
 - e) Incorrecta, siempre va a ser verdadera la post
 - f) Correcta
5. Algoritmo : pot(x) = x^2
 - a) $pot(3)= 9$, se hace verdadera la postcondición
 - b) $x= 0.5$, 1 Incorrecta y $x = -0.2$, -7 Correcta
 - c) Pre $\{x < 0 \vee x > 1\}$
6.
 - a) $P3 > P1 > P2$

- b) $Q3 > Q1 > Q2$
- c) $r := (x - 1)^2$, $r := x^2$
- d)
- 1) Cumple
 - 2) No cumple, problema en la pre
 - 3) Cumple
 - 4) No cumple, problema en la post
 - 5) Cumple
 - 6) No cumple, problema en la pre
 - 7) No cumple, problema en la post
 - 8) No cumple, ni la pre o post.
- e) Se debe cumplir que las nuevas postcondiciones sean más débiles que las anteriores y las precondiciones más fuertes que los anteriores
- 7.
- a) Si $x \neq 0$ cumplen $Pre_{p1} \rightarrow x \neq 0$, por lo tanto como $Pre_{p2} \equiv n \leq 0 \rightarrow x \leq 0 \xrightarrow{x \neq 0 = True} n \leq 0 \rightarrow True \equiv True$ por lo tanto siempre se cumple Pre_{p2} si se cumple Pre_{p1}
 - b) Si se cumple $Post_{p2}$ entonces $\text{res} = \lfloor x^n \rfloor$ y además $x^n - 1 < \lfloor x^n \rfloor \wedge \lfloor x^n \rfloor \leq x^n$ podemos saber que si res cumple $Post_{p2}$ entonces cumple $Post_{p1}$
 - c) No, porque $Post_{p2} \rightarrow Post_{p1}$ y $Pre_{p1} \rightarrow Pre_{p2}$, no viceversa. Por lo tanto si $n > 0$ a no cumple necesariamente Pre_{p1}
8. Si un algoritmo cumple la pre de p1, entonces cumple la de p2. Ya que una lista ordenada tiene todos elementos distintos, y ambos aseguran que $0 \leq n < |l|$. Pero si cumple la pre de p2 no necesariamente cumple la de p1 (ej.: $\langle 1, 3, 2 \rangle$). Y la post de p1 también cumple p2. Porque la sumatoria de p2 está preguntando cuántos números menores a result hay, que es lo mismo que la posición de la secuencia ordenada.
9. aux **apariciones** ($n: \mathbb{Z}$, $l: seq(\mathbb{Z})$) : $\mathbb{Z} = \sum_{i=0}^{|l|-1} \text{if } l[i] == n \text{ then } 1 \text{ else } 0 \text{ fi ;}$
- a) proc **esPar** (**in** $n: \mathbb{Z}$, **out** $\text{res}: \text{Bool}$) {

 Pre { $True$ }

 Post { $\text{res} == True \leftrightarrow (n \bmod 2 = 0)$ }

}
 - b) proc **esMultiplo** (**in** $n, m: \mathbb{Z}$, **out** $\text{res}: \text{Bool}$) {

 Pre { $True$ }

 Post { $\text{res} == True \leftrightarrow (\exists k : \mathbb{Z})(n = m * k)$ }

}
 - c) proc **inverso** (**in** $x: \mathbb{R}$, **out** $\text{res}: \mathbb{R}$) {

 Pre { $x \neq 0$ }

 Post { $\text{res} == x^{-1}$ }

}
 - d) proc **obtenerNumericos** (**in** $l: seq(\text{Char})$, **out** $lis: seq(\text{Char})$) {

 Pre { $True$ }

 Post { $(\forall c : \text{Char}) ((ord('0') \leq ord(c) \leq ord('9')) \rightarrow apariciones(c, l) == apariciones(c, lis)) \wedge (c \in lis \rightarrow ord('0') \leq ord(c) \leq ord('9'))$ }

}

- e) `pred esPar (In n: Z) {
 n mod 2 = 0
 }
 pred posicionesParesIguales (l: seq(R), res: seq(R)) {
 ($\forall i : \mathbb{Z}$) $((0 \leq i < |l| \wedge esPar(i)) \rightarrow_L res[i] == l[i])$
 }
 pred posicionesImparesDuplicadas (l: seq(R), res: seq(R)) {
 ($\forall i : \mathbb{Z}$) $((0 \leq i < |l| \wedge \neg esPar(i)) \rightarrow_L res[i] = 2 \cdot l[i])$
 }
 proc duplicarPosicionesImpares (in l: seq(R), out res: seq(R)) {
 Pre {True}
 Post {
 $|l| = |res| \wedge_L$
 posicionesParesIguales(l, res) \wedge
 posicionesImparesDuplicadas(l, res)
 }
 }`
- f) Uso el Pred *sinRepetidos* Ej 4J Práctica 2
`proc divisoresPositivos (in n: Z, out res: seq(Z)) {
 Pre {True}
 Post {sinRepetidos(res) \wedge ($\forall i : \mathbb{Z}$) $((i > 0 \wedge_L n \text{ m\'od } i == 0) \leftrightarrow i \in res)$ }
 }`
10. a) Sí tiene sentido preguntarlo. Y la respuesta es que no es m\'ultiplo, porque $\nexists n \in \mathbb{Z} / 0 \cdot n = 4$
b) Sí debería ser una entrada válida, y debería devolver *False*. En esta especificación no lo es por la precondición.
- c) `proc esMultiplo_corregido (in n, m: Z, out res: Bool) {
 Pre {True}
 Post {res == True \leftrightarrow (m $\neq 0 \wedge_L n \text{ mod } m == 0$)}
 }`
- d) La precondición original es más fuerte que la nueva
11. a) No, porque las posiciones pares deberían seguir iguales.
b) Sí, la satisface
c) `proc duplicarEnImpares_corregido (in l: seq(R), out res: seq(R)) {
 Pre {True}
 Post {|res| = |l| \wedge ($\forall i : \mathbb{Z}$) $(0 \leq i < |res| \rightarrow_L$
 $(i \text{ mod } 2 == 0 \rightarrow res[i] = l[i]) \wedge (i \text{ mod } 2 == 1 \rightarrow res[i] == \text{[redacted]})$)
 }`
- d) La nueva postcondición es más fuerte que la vieja
12. `pred solo0Y1 (res: seq(Z)) {
 ($\forall i : \mathbb{Z}$) $(0 \leq i < |res| \rightarrow_L res[i] == 0 \vee res[i] = 1)$
 }
 aux binADec (res: seq(Z)) : Z = $\sum_{i=0}^{|res|-1} 2^{|res|-1-i} \cdot res[i]$;`

```

proc esBinario (in n:  $\mathbb{Z}$ , out res: seq( $\mathbb{Z}$ )) {
    Pre { $0 \leq n$ }
    Post {solo0Y1(res)  $\wedge$   $n == binADec(res)$ }
}

```

13. En el caso de **n-esimo 1** hay un caso de subespecificación, ya que se está dando una precondición más *restringiva* de la necesaria. Aún así esta precondición es requerida para que la postcondición de **n-esimo 1** describa el problema. Pero como vemos en **n-esimo 2** no hace falta pedir que la secuencia este ordenada para poder especificar el problema, por lo tanto **n-esimo 2** termina abarcando más casos.

14. a) Uso el pred *esPrimo* del Ej11 Práctica 2

```

proc sumaFactores (in n:  $\mathbb{Z}$ , out res:  $\mathbb{Z}$ ) {
    Pre { $1 \leq n$ }
    Post { $res = \sum_{i=1}^n$  if esPrimo( $i$ )  $\wedge$   $n \ mod \ i == 0$  then  $i$  else 0 fi}
}

```

b) proc *esPerfecto* (in n: \mathbb{Z} , out res: Bool) {
 Pre { $0 \leq n$ }
 Post { $res = True \leftrightarrow (n = \sum_{i=1}^{n-1}$ if $n \ mod \ i == 0$ then i else 0 fi)}

c) Uso el pred *sonCoprimos* del Ej20)f) Práctica 1 aux *min* (n: \mathbb{Z} , m: \mathbb{Z}) : $\mathbb{Z} =$ if $n \leq m$ then n else m fi ;

```

proc minimoCoprimo (in n:  $\mathbb{Z}$ , out res:  $\mathbb{Z}$ ) {
    Pre { $n > 0$ }
    Post { $1 > res \wedge sonCoprimos(n, res) \wedge (\nexists m : \mathbb{Z})(2 \leq m < res \wedge sonCoprimos(n, m))$ }
}

```

d) Uso el pred *esPrimo* del Ej11 Práctica 2

aux *desFactorizar* (res: seq($\mathbb{Z} \times \mathbb{Z}$)) : $\mathbb{Z} = \prod_{i=0}^{|res|-1} (res[i]_0)^{res[i]_1}$;

pred *tuplasOrdenadas* (res: seq($\mathbb{Z} \times \mathbb{Z}$)) {
 $(\forall i : \mathbb{Z})(1 \leq i < |res| \longrightarrow_L res[i-1]_0 < res[i]_0)$
}

pred *primosUnicos* (res: seq($\mathbb{Z} \times \mathbb{Z}$)) {
 $(\forall i : \mathbb{Z})(0 \leq i < |res| \longrightarrow_L esPrimo(res[i]_0) \wedge$
 $(\forall j : \mathbb{Z})(0 \leq j < |res| \wedge j \neq i \longrightarrow_L res[j]_0 \neq res[i]_0))$
}

```

proc factores (in n:  $\mathbb{Z}$ , out res: seq( $\mathbb{Z} \times \mathbb{Z}$ )) {
    Pre { $2 \leq n$ }
    Post { $((primosUnicos(res) \wedge tuplasOrdenadas(res)) \wedge \exists l \text{ s.t. } l == desFactorizar(res))$ }
}

```

e) aux *abs* ($x : \mathbb{R}$) : $\mathbb{R} =$ if $x > 0$ then x else $-x$ fi ;

```

proc maximaDistancia (in l: seq( $\mathbb{R}$ ), out res:  $\mathbb{R}$ ) {
    Pre { $2 \leq |l|$ }
    Post { $((\forall i, j : \mathbb{Z})((rango(l, i) \wedge rango(l, j)) \rightarrow_L abs(l[i] - l[j]) \leq res) \wedge$   

 $(\exists i, j : \mathbb{Z})((rango(l, i) \wedge rango(l, j)) \wedge_L abs(l[i] - l[j]) == res)$ }
}

```

- f) aux `elementosQueDivide` (`l: seq<Z>`, `divi: Z`) : $Z = \sum_{i=0}^{|l|-1} (\text{if } l[i] \bmod n == 0 \text{ then } 1 \text{ else } 0)$ fi ;
`proc divisorPopular` (`in l: seq<Z>`) {
 `Pre` { $1 \leq |l|$ }
 `Post` { $\exists res \in l \wedge \neg(\exists n : Z)(n \in l \wedge \text{elementosQueDivide}(l, res) < \text{elementosQueDivide}(l, n))$ }
15. a) `proc nEsimaAparicion` (`in l: seq<R>`, `in e: R`, `in n: Z`, `out res: Z`) {
 `Pre` { $0 < n \leq \text{apariciones}(e, l)$ }
 `Post` { $0 \leq res < |l| \wedge_L (l[res] == e \wedge \text{apariciones}(e, \text{subseq}(l, 0, res)) == n - 1)$ }
}
- b) `proc esSubcadena` (`in s, t: seq<T>`, `out res: Bool`) {
 `Pre` {*True*}
 `Post` { $|s| \leq |t| \wedge_L res == \text{True} \leftrightarrow ((\exists i : Z)(0 \leq i < |t| \wedge (\exists j : Z)(0 \leq j < i \wedge \text{subseq}(t, j, i) == s))$ }
}
- c) `proc incluida` (`in s, t: seq<T>`, `out res: Bool`) {
 `Pre` {*True*}
 `Post` { $res == \text{True} \leftrightarrow (\forall n : T)(n \in s \rightarrow \text{cantidadEn}(s, t) \leq \text{cantidadEn}(n, t))$ }
}
- d) Uso el Pred *estaOrdenada* Ej 4D Práctica 2
`proc mezclarOrdenado` (`in s, t: seq<Z>`, `out res: seq<Z>`) {
 `Pre` {*esOrdenada*(`s`) \wedge *esOrdenada*(`t`)}
 `Post` { $|res| = |s| + |t| \wedge \text{estaOrdenada}(res) \wedge (\forall e : Z)(\text{apariciones}(e, s + +t) == \text{apariciones}(e, res))$ }
}
- e) Uso el Pred *sinRepetidos* Ej 4J Práctica 2
`proc interseccionSinRepetidos` (`in s, t: seq<Z>`, `out res: seq<Z>`) {
 `Pre` {*True*}
 `Post` {*sinRepetidos*(`res`) \wedge $(\forall e : Z)((e \in s \wedge e \in t) \leftrightarrow e \in res)$ }
}
- f) aux `min` (`n: Z, m: Z`) : $Z = \text{if } n \leq m \text{ then } n \text{ else } m$ fi ;
`proc interseccion` (`in s, t: seq<T>`, `out res: seq<T>`) {
 `Pre` {*True*}
 `Post` { $(\forall ele : T)(\min(\text{apariciones}(ele, s), \text{apariciones}(ele, t)) == \text{apariciones}(ele, inter))$ }
}
16. a) aux `cantidadEn` (`e: Char, l: String`) : $Z = \sum_{i=0}^{|l|-1} \text{if } l[i] == e \text{ then } 1 \text{ else } 0$ fi ;
`pred incluyeTodos` (`l: String, res: seq<Char \times Z>`) {
 $(\forall c : Char)(c \in l \leftrightarrow (\exists t : Char \times Z)(t \in res \wedge t_0 == c))$ }
aux `cantidadEnTuplas` (`t: Char, res: seq<Char \times Z>`) : $Z = \sum_{i=0}^{|res|-1} \text{if } res[i]_0 == t \text{ then } 1 \text{ else } 0$ fi ;

```

pred sinDuplicados (res: seq<Char × Z>) {
    ( $\forall t : \text{Char} \times \mathbb{Z}$ ) $(t \in res \rightarrow cantidadEnTuplas(t_0, res) == 1)$ 
}

proc cantApariciones (in l: String, out res: seq<Char × Z>) {
    Pre {True}
    Post {incluyeTodos(l, res)  $\wedge$  sinDuplicados(res)
         $\wedge$  ( $\forall e : \text{Char} \times \mathbb{Z}$ ) $(e \in res \rightarrow cantidadEn(e_0, l) == e_1)$ }
}

```

- b) Uso el Pred *esPrefijo* Ej 4)c) Práctica 2

```

proc prefijosOrdenados (in l: seq<seq<T>>, out res: seq<seq<T>>) {
    Pre {True}
    Post {|l| + 1 = |res|  $\wedge_L (\forall i : \mathbb{Z})(rango(res, i) \rightarrow_L (|s[i]| = i \wedge esPrefijo(s[i], l)))$ }
}

c) pred estaEnL (l: seq<seq<Z>>, max: Z) {
    ( $\exists subL : seq<\mathbb{Z}>$ ) $(subL \in l \wedge max \in subL)$ 
}

pred maximoElementoDeL (l : seq<seq<Z>>, max: Z) {
    estaEnL(l, max)  $\wedge$ 
    ( $\forall subL : seq<\mathbb{Z}>$ ) $(subL \in l \rightarrow (\forall n : \mathbb{Z})(n \in subL \rightarrow n \leq max))$ 
}

proc secuenciaConMax (in l: seq<seq<Z>>, out res: seq<Z>) {
    Pre {True}
    Post {
        ( $\exists max : \mathbb{Z}$ ) $(maximoElementoDeL(l, max) \wedge max \in res) \wedge res \in l$ 
    }
}

d) Las apariciones de e en l son menores o iguales a las apariciones en todas las otras
sublistas
pred menosApariciones (l: seq<R>, ls: seq<seq<R>>) {
    ( $\forall e : R$ ) $(e \in l \leftrightarrow (\forall sublis : seq<R>)(sublis \in ls \rightarrow apariciones(e, sublis) \leq apariciones(e, sublis)))$ 
}
Las apariciones de e en l son iguales a las apariciones en alguna de las sublistas.
pred aparicionesIguals (l: seq<R>, ls: seq<seq<R>>) {
    ( $\exists sublis : seq<R>$ ) $(sublis \in ls \wedge apariciones(e, l) == apariciones(e, sublis))$ 
}

proc interseccionMultiple (in ls: seq<seq<R>>, out res: seq<R>) {
    Pre {True}
    Post {menosApariciones(res, ls)  $\wedge$  aparicionesIguals(res, ls)}
}

e) Uso el Pred sinRepetidos Ej 4)j) Práctica 2
pred sonSecusDistintas (ls : seq<seq<Z>>) {
    ( $\forall i, j : \mathbb{Z}$ ) $((rango(ls, i) \wedge rango(ls, j) \wedge i \neq j) \rightarrow_L l[i] \neq l[j])$ 
}

pred estaIncluida (s,t : seq<Z>) {
    ( $\forall n : T$ ) $(n \in s \rightarrow apariciones(s, n) \leq apariciones(n, t))$ 
}

aux devuelveIndice (e : Z, l seq<Z>) : Z =
 $\sum_{i=0}^{|l|-1} \text{if } l[i] == e \text{ then } i \text{ else } 0$  fi;

Para ver si se mantiene el orden de las apariciones de los elementos de partes en l
pred mismoOrden (partes : seq<Z>, l : seq<Z>) {
    ( $\forall i : \mathbb{Z}$ ) $(0 \leq i < |partes| - 1 \rightarrow_L devuelveIndice(partes[i], l) \leq devuelveIndice(partes[i+1], l))$ 
}

```

```

proc conjuntoPartes ( in l : seq<Z>, out ls : seq<seq<Z>>) {
    Pre {sinRepetidos(l)}
    Post {sonSecusDistintas(ls) ∧ |ls| = 2^|l| ∧ (∀sublis : seq<Z>)(sublis ∈ ls ↔
        (estaIncluido(sublis, l) ∧ sinRepetidos(sublis) ∧ mismoOrden(sublis, l)))}
}

```

17. a) No es necesario que ningun parametro sea inout
 b) C debería ser un parametro out
 c) Funciona
 d) Funciona, pero no es necesario que a y b sean inout.
18. El problema dice ” se desea sacar su primer elemento y devolverlo”. Por lo tanto se debe quitar el primer elemento de l y además devolverlo. Solo si se realizan **ambas** operaciones la especificación es correcta.
- a) No se esta sacando el primer elemento de l, meramente se lo extrae
 b) No se esta sacando el primer elemento de l, meramente se lo extrae
 c) No se guarda el valor inicial de L, por lo tanto no se sabe que sería L_0
 d) Es correcta
 e) Es incorrecta, se indefine cuándo $i = -l-1$, pues intenta buscar $L_0[-l-]$.
19. a) Es una especificación válida, describe el problema de intercambiar dos posiciones en una lista
 b) Te puede devolver una lista con los valores cambiados, poer no dice nada de las otras posiciones.
 Ejemplo: intercambiar (⟨1, 3, 5, 4⟩, 0,2) = ⟨5, π * 3, 1, e/4⟩
 c) Se le puede agregar a la post :
 $(\forall k : \mathbb{Z})((0 \leq k < |l| \wedge k \neq i \wedge k \neq j) \rightarrow_L l[k] = L_0[k])$
20. Lo que hace es agarrar una lista y un indice **i** válido de la misma, para luego devolver la misma lista con el valor de la posición $l[i]$ en todas las posiciones excepto en la **i** que esta el primer valor de la lista original. Ej; copiarPrimero(⟨1, 3, 7, -9, 15, 17⟩, 2) = ⟨7, 7, 1, 7, 7, 7⟩
21. a) Es incorrecta ya que no hace ninguna aclaración acerca de las posiciones impares. Puede ser agregado a la post:
 $(\forall j : \mathbb{Z})((0 \leq j < |l| \wedge j \text{ mód } 2 == 1) \rightarrow_L l[j] == L_0[j])$
 b) Es incorrecta, no aclara nada acerca de la longitud de L_0 o l por lo que se puede definir.
 c) Es incorrecta, l deberíai ser una variable in, no inout.
22. a) Uso el Pred *esPrimo* Ej 11 Práctica 2
- ```

pred primoDeabajo (a: Z, b: Z) {
 b < a ∧ esPrimo(b) ∧ (∀n : Z)(b < n < a → ¬esPrimo(n))
}

proc primosHermanos (inout l: seq<Z>) {
 Pre {l = l₀ ∧ (∀i : Z)(0 ≤ i < |l| →_L 2 < l₀[i])}
 Post {|l| = |l₀| ∧ (∀i : Z)(0 ≤ i < |l| →_L primoDeabajo(l₀[i], l[i]))}
}

```

b) proc **recortar** (inout l:  $seq\langle \mathbb{Z} \rangle$ , in a:  $\mathbb{Z}$ ) {  
**Pre**  $\{l = l_0\}$   
**Post**  $\{|l| = |l_0| \wedge_L (asReemplazadasPorBs(l, l_0, a, b) \wedge restoIguales(l, l_0, a, b))\}$   
}  
pred **asReemplazadasPorBs** (l,  $l_0 : seq\langle Char \rangle$ , a,b : Char)  $\{(\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L (l_0[i] == a \rightarrow l[i] == b))\}$   
pred **restoIguales** (l,  $l_0 : seq\langle Char \rangle$ , a,b : Char)  $\{(\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L (l_0[i]! = a \rightarrow l[i] == l_0[i]))\}$   
c) pred **principioRecortado** (l,  $l_0 : seq\langle \mathbb{Z} \rangle$ , i :  $\mathbb{Z}$ )  $\{(0 \leq i \leq |l_0| \rightarrow_L subseq(l_0, i, |l_0|) == l)\}$   
pred **asQuitadas** (l,  $l_0 : seq\langle \mathbb{Z} \rangle$ , i :  $\mathbb{Z}$ , a:  $\mathbb{Z}$ )  $\{apariciones(a, l) == apariciones(a, l_0) - i\}$   
proc **recortar** (inout l :  $seq\langle \mathbb{Z} \rangle$ , a :  $\mathbb{Z}$ ) {  
**Pre**  $\{l = l_0\}$   
**Post**  $\{(\exists i : \mathbb{Z})(principioRecortado(l, l_0, i) \wedge asQuitadas(l, l_0, i, a))\}$   
}  
d) proc **intercambiarParesConImpares** (inout l : String) {  
**Pre**  $\{|l| mod 2 == 0 \wedge l = l_0\}$   
**Post**  $\{|l| = |l_0| \wedge_L (\forall i : \mathbb{Z})((rango(l, i) \wedge i \bmod 2 == 1) \rightarrow_L (l[i] == l_0[i - 1] \wedge l[i - 1] == l_0[i]))\}$   
}  
e) Uso el Pred *MismoOrden* Ej 16)e) Práctica 3  
pred **mismosElementos** ( $l_1, l_2, : seq\langle Char \rangle$ )  $\{(\forall c : Char)(c \in l_2 \leftrightarrow c \in l_1)\}$   
pred **duplicados** (dup,  $l_0 : seq\langle Char \rangle$ ) {  
 $(\forall i : \mathbb{Z})( (rango(l_0, i) \rightarrow_L apariciones(l_0[i], l) - apariciones(l_0[i], dup) == 1) \wedge (rango(dup, i) \rightarrow_L apariciones(dup[i], l) - apariciones(dup[i], dup) == 1) )$   
}  
proc **limpiarDuplicados** (inout l :  $seq\langle Char \rangle$ , out dup :  $seq\langle Char \rangle$ ) {  
**Pre**  $\{l = l_0\}$   
**Post**  $\{|l_0| = |l| + |dup| \wedge_L (sinRepetidos(l) \wedge mantieneOrden(l, l_0) \wedge mismosElementos(l, l_0) \wedge duplicados(dup, l_0))\}$   
}

## 2. Bonus Track : Diferencia entre especificar y programar

### 1. Problema: Sudoku

El sudoku es un juego matemático muy conocido que consiste en llenar un tablero, en este caso de 9X9, de tal forma que no haya número repetidos en ninguna fila, columna o cuadraditos de 3X3.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 4 | 5 | 1 | 7 | 3 | 6 | 8 | 2 | 9 |
| 9 | 6 | 2 | 8 | 5 | 1 | 4 | 3 | 7 |
| 8 | 7 | 3 | 4 | 9 | 2 | 5 | 6 | 1 |
| 6 | 1 | 4 | 3 | 8 | 5 | 7 | 9 | 2 |
| 2 | 9 | 7 | 6 | 1 | 4 | 3 | 8 | 5 |
| 5 | 3 | 8 | 9 | 2 | 7 | 6 | 1 | 4 |
| 1 | 4 | 9 | 5 | 6 | 3 | 2 | 7 | 8 |
| 7 | 2 | 6 | 1 | 4 | 8 | 9 | 5 | 3 |
| 3 | 8 | 5 | 2 | 7 | 9 | 1 | 4 | 6 |

Figura 1: Tablero de Sudoku Resuelto

Intentar hacer un programa que resuelva un sudoku no es trivial. Para hacer algo moderadamente eficiente uno necesitaría usar la estrategia de **backtracking**. Además de pensar en cómo acceder, modificar y trabajar con el tablero. Pero en cambio especificar este problema sale usando 5 predicados relativamente sencillos.

```

predesMatriz (ls : seq<seq<Z>>){
 ($\forall i : \mathbb{Z}$) ($0 \leq i < |ls| \rightarrow (m[i] \geq 0 \wedge |m[i]| == |m[0]|)$)
}

pred menoresA10 (tablero : seq<seq<Z>>) {
 ($\forall i, j : \mathbb{Z}$) ($(0 \leq i < |tablero| \wedge 0 \leq j < |tablero|) \rightarrow_L (0 < tablero[i][j] < 10)$)
}

pred esSudoku (tablero : seq<seq<Z>>) {
 ($\forall f : \mathbb{Z}$) ($0 \leq f < |tablero| \wedge_L (\forall i, j : \mathbb{Z}) ((0 \leq i < |tablero| \wedge 0 \leq j < |tablero| \wedge i \neq j) \rightarrow_L ((tablero[f][i] \neq tablero[f][j]) \wedge (tablero[i][f] \neq tablero[j][f])))$)
}

pred distintosEnCuadraditos (tablero : seq<seq<Z>>) {
 ($\forall i, j : \mathbb{Z}$) ($((0 \leq i < |tablero| \wedge 0 \leq j < |tablero| \wedge i - 1 \text{ mód } 3 == 0 \wedge j - 1 \text{ mód } 3 == 0) \rightarrow_L sonDistintosEnCuadradito(tablero, i, j))$)
}

pred sonDistintosEnCuadradito (tablero : seq<seq<Z>>, colu : Z, fila : Z) {
 ($\exists i_1, j_1, i_2, j_2 \in \mathbb{Z}$) ($((fila - 1 \leq i_1, i_2 \leq fila + 1 \wedge colu - 1 \leq j_1, j_2 \leq colu + 1 \wedge (i_1 \neq i_2 \vee j_1 \neq j_2)) \rightarrow_L tablero[i_1][j_1] \neq tablero[i_2][j_2])$)
}

proc Sudoku (inout tablero : seq<seq<Z>>) {
 Pre {esMatriz(tablero) \wedge_L (|tablero[0]| == 9 \wedge |tablero| == 9)}
 Post {esMatriz(tablero) \wedge_L (|tablero[0]| == 9 \wedge |tablero| == 9 \wedge esSudoku(tablero) \wedge menoresA10(tablero) \wedge
 distintosEnCuadraditos(tablero))}
}

```

Con ese procedimiento ya fue especificado el problema. La ventaja de especificar es que uno no debe pensar cómo resolvería el problema, sino describir con lógica de primer orden cuál es el contrato del programa.



**Ejercicio 1.** ★ Calcular las siguientes expresiones, donde  $a, b$  son variables reales,  $i$  una variable entera y  $A$  es una secuencia de reales.

- a)  $\text{def}(a + 1)$ .
- b)  $\text{def}(a/b)$ .
- c)  $\text{def}(\sqrt{a/b})$ .
- d)  $\text{def}(A[i] + 1)$ .
- e)  $\text{def}(A[i + 2])$ .
- f)  $\text{def}(0 \leq i \leq |A|)$ .
- g)  $\text{def}(0 \leq i \leq |A| \wedge_L A[i] \geq 0)$ .

**Ejercicio 2.** Calcular las siguientes precondiciones más débiles, donde  $a, b$  son variables reales,  $i$  una variable entera y  $A$  es una secuencia de reales.

- a)  $\text{wp}(\mathbf{a} := \mathbf{a+1}, a \geq 0)$ .
- b)  $\text{wp}(\mathbf{a} := \mathbf{a/b}, a \geq 0)$ .
- c)  $\text{wp}(\mathbf{a} := \mathbf{A[i]}, a \geq 0)$ .
- d)  $\text{wp}(\mathbf{a} := \mathbf{b*b}, a \geq 0)$ .
- e)  $\text{wp}(\mathbf{b} := \mathbf{b+1}, a \geq 0)$ .

**Ejercicio 3.** ★ Calcular las siguientes precondiciones más débiles, donde  $a, b$  son variables reales,  $i$  una variable entera y  $A$  es una secuencia de reales.

- a)  $\text{wp}(\mathbf{a} := \mathbf{a+1}; \mathbf{b} := \mathbf{a/2}, b \geq 0)$ .
- b)  $\text{wp}(\mathbf{a} := \mathbf{A[i]} + 1; \mathbf{b} := \mathbf{a*a}, b \neq 2)$ .
- c)  $\text{wp}(\mathbf{a} := \mathbf{A[i]} + 1; \mathbf{a} := \mathbf{b*b}, a \geq 0)$ .
- d)  $\text{wp}(\mathbf{a} := \mathbf{a-b}; \mathbf{b} := \mathbf{a+b}, a \geq 0 \wedge b \geq 0)$ .

**Ejercicio 4.** ★ Sea  $Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \geq 0)$ . Calcular las siguientes precondiciones más débiles, donde  $i$  es una variable entera y  $A$  es una secuencia de reales.

- a)  $\text{wp}(\mathbf{A[i]} := \mathbf{0}, Q)$ .
- b)  $\text{wp}(\mathbf{A[i+2]} := \mathbf{0}, Q)$ .
- c)  $\text{wp}(\mathbf{A[i+2]} := \mathbf{-1}, Q)$ .
- d)  $\text{wp}(\mathbf{A[i]} := \mathbf{2 * A[i]}, Q)$ .
- e)  $\text{wp}(\mathbf{A[i]} := \mathbf{A[i-1]}, Q)$ .

**Ejercicio 5.** Calcular  $wp(S, Q)$ , para los siguientes pares de programas  $S$  y postcondiciones  $Q$ .

- a)  $S \equiv i := i + 1$   
 $Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \neq 0)$
- b)  $S \equiv A[0] := 4$   
 $Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \neq 0)$
- c)  $S \equiv A[2] := 4$   
 $Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \neq 0)$
- d)  $S \equiv A[i] := A[i+1] - 1$   
 $Q \equiv (\forall j : \mathbb{Z})(0 < j < |A| \rightarrow_L A[j] \geq A[j - 1])$
- e)  $S \equiv A[i] := A[i+1] - 1$   
 $Q \equiv (\forall j : \mathbb{Z})(0 < j < |A| \rightarrow_L A[j] \leq A[j - 1])$

**Ejercicio 6.** Escribir programas para los siguientes problemas y demostrar formalmente su corrección usando la precondición más débil.

- a) 

```
proc problema1 (inout a: Z) {
 Pre {a = a0 ∧ a ≥ 0}
 Post {a = a0 + 2}
}
```
- b) 

```
proc problema2 (in a: Z, out b: Z) {
 Pre {a ≠ 0}
 Post {b = a + 3}
}
```
- c) 

```
proc problema3 (in a: Z, in b: Z, out c: Z) {
 Pre {true}
 Post {c = a + b}
}
```
- d) 

```
proc problema4 (in a: seq<Z>, in i: Z, out result: Z) {
 Pre {0 ≤ i < |a|}
 Post {result = 2 * a[i]}
}
```
- e) 

```
proc problema5 (in a: seq<Z>, in i: Z, out result: Z) {
 Pre {0 ≤ i ∧ i + 1 < |a|}
 Post {result = a[i] + a[i + 1]}
}
```

**Ejercicio 7.** ★ Calcular  $wp(S, Q)$ , para los siguientes pares de programas  $S$  y postcondiciones  $Q$ .

a)  $S \equiv$

```
if(a < 0)
 b := a
else
 b := -a
endif
```

$$Q \equiv (b = -|a|)$$

b)  $S \equiv$

```
if(a < 0)
 b := a
else
 b := -a
endif
```

$$Q \equiv (b = |a|)$$

c)  $S \equiv$

```
if(i > 0)
 s[i] := 0
else
 s[0] := 0
endif
```

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

d)  $S \equiv$

```
if(i > 1)
 s[i] := s[i - 1]
else
 s[i] := 0
endif
```

$$Q \equiv (\forall j : \mathbb{Z})(1 \leq j < |s| \rightarrow_L s[j] = s[j - 1])$$

e)  $S \equiv$

```
if(s[i] < 0)
 s[i] := -s[i]
else
 skip
endif
```

$$Q \equiv 0 \leq i < |s| \wedge_L s[i] \geq 0$$

f)  $S \equiv$

```
if(s[i] > 0)
 s[i] := -s[i]
else
 skip
endif
```

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

**Ejercicio 8.** ★ Escribir programas para los siguientes problemas y demostrar formalmente su corrección usando la precondición más débil.

a) proc problema1 (in s:  $seq\langle \mathbb{Z} \rangle$ , in i:  $\mathbb{Z}$ , inout a:  $\mathbb{Z}$ ) {

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

Post  $\{a = \sum_{j=0}^i s[j]\}$

}

b) proc problema2 (in s:  $seq\langle \mathbb{Z} \rangle$ , in i:  $\mathbb{Z}$ , inout a:  $\mathbb{Z}$ ) {

Pre  $\{0 \leq i < |s| \wedge_L a = \sum_{j=0}^i s[j]\}$

Post  $\{a = \sum_{j=1}^i s[j]\}$

}

c) proc problema3 (in s:  $seq\langle \mathbb{Z} \rangle$ , in i:  $\mathbb{Z}$ , out res: Bool) {

Pre  $\{0 \leq i < |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \geq 0)\}$

Post  $\{res = true \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0)\}$

}

d) proc problema4 (in s:  $seq\langle \mathbb{Z} \rangle$ , in i:  $\mathbb{Z}$ , inout a:  $\mathbb{Z}$ ) {

Pre  $\{0 \leq i < |s| \wedge_L a = \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

Post  $\{a = \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

}

e) proc problema5 (in s:  $seq\langle \mathbb{Z} \rangle$ , in i:  $\mathbb{Z}$ , inout a:  $\mathbb{Z}$ ) {

Pre  $\{0 < i \leq |s| \wedge_L a = \sum_{j=1}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

Post  $\{a = \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

}

## 1. Axiomas a utilizar

$$1. \text{ Asignación : } Wp(x := E, Q) \stackrel{\text{Axioma 1}}{\equiv} def(E) \wedge_L Q_E^x$$

Asignación con Secuencias:  $Wp(A[i] := k, Q) \equiv Wp(A := setAt(A, i, k), Q) \equiv def(setAt(A, i, k)) \wedge Q_A^{setAt(A,i,k)}$

$$SetAt(A, i, 0)[x] = \begin{cases} A[x] & x \neq i \\ 0 & x == i \end{cases}$$

$$2. \text{ Skip : } Wp(Skip, Q) \stackrel{\text{Axioma 2}}{\equiv} Q$$

$$3. S1;S2 : Wp(S1; S2, Q) \stackrel{\text{Axioma 3}}{\equiv} Wp(S1, Wp(S2, Q))$$

$$4. \text{ Alternativa: } Wp(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}, Q)$$

$$\stackrel{\text{Axioma 4}}{\equiv} def(B) \wedge_L ((B) \wedge Wp(S1, Q)) \vee (\neg(B) \wedge Wp(S2, Q))$$

## 2. Práctica 4 : Precondición más débil en Small Lang

1. Siempre asumo que las variables están definidas

- a) True
- b)  $b \neq 0$
- c)  $b \neq 0 \wedge_L a/b \geq 0$
- d)  $0 \leq i < |A|$
- e)  $-2 \leq u < |A| - 2$
- f) True
- g)  $i \neq |A|$

$$2. \text{ Para resolver este ejercicio uso el Axioma 1 } Wp(x := E, Q) \stackrel{\text{Axioma 1}}{\equiv} def(E) \wedge_L Q_E^x$$

- a)  $Wp(a := a + 1, a \geq 0) \equiv def(a) \wedge_L a + 1 \geq 0 \equiv a \geq -1$
- b)  $Wp(a := a/b, a \geq 0) \equiv def(a/b) \wedge_L a/b \geq 0 \equiv b \neq 0 \wedge_L a/b \geq 0$
- c)  $Wp(a := A[i], a \geq 0) \equiv def(A[i]) \wedge_L A[i] \geq 0 \equiv 0 \leq i < |A| \wedge_L A[i] \geq 0$
- d)  $Wp(a := b * b, a \geq 0) \equiv def(b^2) \wedge_L b^2 \geq 0 \equiv True$
- e)  $Wp(b := b + 1, a \geq 0) \equiv def(b + 1) \wedge a \geq 0 \equiv a \geq 0$

$$3. a, b : \mathbb{R}; i : \mathbb{Z}; A : seq\langle \mathbb{R} \rangle$$

- a)  $Wp(a := a + 1; b := a/2, b \geq 0) \stackrel{\text{Axioma 3}}{\equiv} Wp(a := a + 1, Wp(b := a/2, b \geq 0)) \equiv Wp(a := a + 1, a \geq 0) \equiv a \geq -1$
- b)  $Wp(a := A[i]; b := a * a, b \neq 2) \stackrel{\text{Axioma 3}}{\equiv} Wp(a := A[i], Wp(b := a * a, b \neq 2)) \equiv Wp(a := A[i] + 1, |a| \neq \sqrt{2}) \equiv 0 \leq i < |A| \wedge_L |A[i] + 1| \neq \sqrt{2}$
- c)  $Wp(a := A[i] + 1; a := b * b, a \geq 0) \stackrel{\text{Axioma 3}}{\equiv} Wp(a := A[i] + 1, Wp(a := b * b, a \geq 0)) \equiv Wp(a := A[i] + 1, True) \equiv 0 \leq i < |A|$
- d)  $Wp(a := a - b; b := a + b, a \geq 0 \wedge b \geq 0) \stackrel{\text{Axioma 3}}{\equiv} Wp(a := a - b, Wp(b := a + b, a \geq 0 \wedge b \geq 0)) \equiv Wp(a := a - b, a \geq 0 \wedge a + b \geq 0) \equiv a - b \geq 0 \wedge (a - b) + b \geq 0 \equiv a \geq b \wedge a \geq 0$

$$4. Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \geq 0)$$

Además uso la versión "setAt" del axioma 1:  $Wp(A[i] := 0, Q) \equiv Wp(A := setAt(A, i, 0), Q)$

$$\text{Con setAt siendo } SetAt(A, i, 0)[x] = \begin{cases} A[x] & x \neq i \\ 0 & x == i \end{cases}$$

- a)  $Wp(A[i] := 0, Q) \equiv Wp(setAt(A, i, 0), Q) \xrightarrow{Axioma1} 0 \leq i < |A| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |setAt(A, i, 0)| \rightarrow_L setAt(A, i, 0)[j] \geq 0) \equiv$
- $$SetAt(A, i, 0)[x] = \begin{cases} A[x] & x \neq i \\ 0 & x == i \end{cases}$$
- $$0 \leq i < |A| \wedge_L (\forall j : \mathbb{Z})((0 \leq j < |A| \wedge i \neq j) \rightarrow_L A[j] \geq 0)$$
- b)  $Wp(A[i + 2] := 0, Q) \equiv Wp(setAt(A, i + 2, 0), Q) \xrightarrow{Axioma1} 0 \leq i + 2 < |A| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |setAt(A, i + 2, 0)| \rightarrow_L setAt(A, i + 2, 0)[j] \geq 0) \equiv$
- $$SetAt(A, i + 2, 0)[x] = \begin{cases} A[x] & x \neq i + 2 \\ 0 & x == i + 2 \end{cases}$$
- $$0 \leq i + 2 < |A| \wedge_L (\forall j : \mathbb{Z})((0 \leq j < |A| \wedge i + 2 \neq j) \rightarrow_L A[j] \geq 0)$$
- c)  $Wp(A[i + 2] := -1, Q) \equiv Wp(setAt(A, i + 2, -1), Q) \xrightarrow{Axioma1} 0 \leq i + 2 < |A| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |setAt(A, i + 2, -1)| \rightarrow_L setAt(A, i + 2, -1)[j] \geq 0) \equiv 0 \leq i + 2 < |A| \wedge_L False$
- $$A[i + 2] = -1 \geq 0 \equiv False$$
- d)  $Wp(A[i] := A[i] * 2, Q) \equiv Wp(setAt(A, i, A[i] * 2), Q) \xrightarrow{Axioma1} 0 \leq i < |A| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |setAt(A, i, A[i] * 2)| \rightarrow_L setAt(A, i, A[i] * 2)[j] \geq 0) \equiv 0 \leq i < |A| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \geq A[i] * 2 \geq 0 \equiv A[i] \geq 0)$
- e)  $Wp(A[i] := A[i - 1], Q) \equiv Wp(setAt(A, i, A[i - 1]), Q) \xrightarrow{Axioma1} 1 \leq i < |A| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |setAt(A, i, A[i - 1])| \rightarrow_L setAt(A, i, A[i - 1])[j] \geq 0) \equiv 1 \leq i < |A| \wedge_L$
- $$A[i - 1] \geq 0 \equiv A[i] \geq 0$$
- $$(\forall j : \mathbb{Z})((0 \leq j < |A| \wedge j \neq i) \rightarrow_L A[j] \geq 0)$$
5. a)  $Wp(S, Q) \equiv Q$
- b)  $Wp(S, Q) \equiv |A| > 0 \wedge_L (\forall j : \mathbb{Z})(0 < j < |A| \rightarrow_L A[j] \neq 0)$
- $$SetAt(A, 0, 4)[x] = \begin{cases} A[x] & x \neq 0 \\ 4 & x == 0 \end{cases} \wedge A[0] = 4 \neq 0 \equiv True$$
- c)  $Wp(S, Q) \equiv |A| > 2 \wedge_L (\forall j : \mathbb{Z})((0 \leq j < |A| \wedge j \neq 2) \rightarrow_L A[j] \neq 0)$
- $$SetAt(A, 2, 4)[x] = \begin{cases} A[x] & x \neq 2 \\ 4 & x == 2 \end{cases} \wedge A[2] = 4 \neq 0 \equiv True$$
- d)  $Wp(S, Q) \equiv 0 \leq i < |A| - 1 \wedge_L (0 < j < |A| \wedge j \neq i + 1) \rightarrow_L A[j] \geq A[j - 1]$
- $$SetAt(A, i, A[i + 1] - 1)[x] = \begin{cases} A[x] & x \neq i \\ A[i + 1] - 1 & x == i \end{cases} \wedge A[i] = A[i + 1] - 1 \leq A[i + 1] \equiv True$$
- e)  $Wp(S, Q) \equiv 0 \leq i < |A| - 1 \wedge False$
- $$SetAt(A, i, A[i + 1] - 1)[x] = \begin{cases} A[x] & x \neq i \\ A[i + 1] - 1 & x == i \end{cases} \wedge A[i] = A[i + 1] - 1 \geq A[i + 1] \equiv False$$

6. Lo que vamos a buscar es ver que la Pre  $\rightarrow Wp(S, Q)$

- a) Programa:  $a := a + 2$   
 $Wp(a := a + 2, a == a_0 + 2) \equiv a + 2 == a_0 + 2 \equiv a = a_0$   
 $(a == a_0 \wedge a \geq 0) \rightarrow a = a_0$
- b) Programa :  $b := a + 3$   
 $Wp(b := a + 3, b == a + 3) \equiv a + 3 == a + 3 \equiv True$   
 $a \neq 0 \rightarrow True$
- c) Programa :  $c := a + b$   
 $Wp(c := a + b, c == a + b) \equiv a + b == a + b \equiv True$   
 $True \rightarrow True$
- d) Programa :  $result := 2 * a[i]$   
 $Wp(result := 2 * a[i], result == 2 * a[i]) \rightarrow def(a[i]) \wedge_L True \equiv 0 \leq i < |s|$   
 $0 \leq i < |s| \rightarrow 0 \leq i < |s|$

e) Programa: result:=a[i] + a[i+1]

$$\begin{aligned} Wp(result := a[i] + a[i+1], result = a[i] + a[i+1]) &\equiv (def(a[i]) \wedge def(a[i+1])) \wedge a[i] + a[i+1] == \\ a[i] + a[i+1] &\equiv 0 \leq i < |A| - 1 \\ 0 \leq i &< |A| - 1 \rightarrow 0 \leq i < |A| - 1 \end{aligned}$$

7. Para resolver estos problemas se necesita usar este axioma

$$\begin{aligned} Wp(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}, Q) \\ \stackrel{\text{Axioma 4}}{\equiv} def(B) \wedge_L ((B) \wedge Wp(S_1, Q)) \vee (\neg(B) \wedge Wp(S_2, Q)) \end{aligned}$$

a)  $Wp(\text{if } a < 0 \text{ then } b := a \text{ else } b := -a \text{ fi}, b = -|a|)$

$$\stackrel{\text{Axioma 4}}{\equiv} def(a < 0) \wedge_L ((a < 0) \wedge Wp(b := a, b = -|a|)) \vee (\neg(a < 0) \wedge Wp(b := -a, b = -|a|)) \equiv$$

$$True \wedge_L ((a < 0 \wedge a = -|a|) \vee (a \geq 0 \wedge a = |a|)) \equiv (True \vee True) \equiv True$$

b)  $Wp(\text{if } a < 0 \text{ then } b := a \text{ else } b := -a \text{ fi}, b = |a|)$

$$\stackrel{\text{Axioma 4}}{\equiv} def(a < 0) \wedge_L ((a < 0) \wedge Wp(b := a, b = |a|)) \vee (\neg(a < 0) \wedge Wp(b := -a, b = |a|)) \equiv$$

$$True \wedge_L ((a < 0 \wedge a = |a|) \vee (a \geq 0 \wedge a = -|a|))$$

$$\stackrel{\uparrow}{\equiv} False \vee a == 0 \equiv a == 0 \\ (a \geq 0 \wedge a == -|a|) \equiv a == 0$$

c)  $Wp(\text{if } i > 0 \text{ then } s[i] := 0 \text{ else } s[0] := 0 \text{ fi}, (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))$

$$\stackrel{\text{Axioma 4}}{\equiv} def(i > 0) \wedge_L ((i > 0) \wedge Wp(s[i] := 0, (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))) \vee (\neg(i >$$

$$0) \wedge Wp(s[0] := 0, (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))) \equiv$$

$$True \wedge_L ((0 < i < |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < |setAt(s, i, 0)| \rightarrow_L setAt(s, i, 0)[j] \geq 0)) \vee$$

$$(i \leq 0 \wedge |s| > 0 \wedge (\forall j : \mathbb{Z})(0 \leq j < |setAt(s, 0, 0)| \rightarrow_L setAt(s, 0, 0)[j] \geq 0))) \equiv$$

$$((0 < i < |s| \wedge (\forall j : \mathbb{Z})((0 \leq j < |s| \wedge i \neq j) \rightarrow_L s[j] \geq 0)) \vee$$

$$(i \leq 0 \wedge |s| > 0 \wedge (\forall j : \mathbb{Z})(0 < j < |s| \rightarrow_L s[j] \geq 0)))$$

d)  $Wp(\text{if } i > 1 \text{ then } s[i] := s[i - 1] \text{ else } s[i] := 0 \text{ fi}, (\forall j : \mathbb{Z})(1 \leq j < |s| \rightarrow_L s[j] == s[j - 1]))$

$$\begin{aligned} &\stackrel{\text{Axioma 4}}{\equiv} def(i > 1) \wedge_L ((i > 1) \wedge Wp(s[i] := s[i - 1], (\forall j : \mathbb{Z})(1 \leq j < |s| \rightarrow_L s[j] == s[j - 1]))) \vee \\ &(\neg(i > 1) \wedge Wp(s[i] := 0, (\forall j : \mathbb{Z})(1 \leq j < |s| \rightarrow_L s[j] == s[j - 1]))) \equiv \\ &True \wedge_L ((1 \leq i < |s| \wedge (\forall j : \mathbb{Z}) 1 \leq j < |setAt(s, i, s[i - 1])| \rightarrow_L setAt(s, i, s[i - 1])[j] == \\ &setAt(s, i, s[i - 1])[j - 1])) \vee (\|s\| \geq 2 \wedge 0 \leq i \leq 1 \wedge (\forall j : \mathbb{Z})(1 \leq j < |setAt(s, i, 0)| \rightarrow_L setAt(s, i, 0)[j] == \\ &setAt(s, i, 0)[j - 1])) \equiv \\ &(((1 \leq i < |s| \wedge (\forall j : \mathbb{Z}) ((1 \leq j < |s| \wedge j \neq i) \rightarrow_L s[j] == s[j - 1])) \vee \\ &\quad \uparrow \\ &\quad SetAt(s, i, s[i - 1])[x] = \begin{cases} s[x] & x \neq i \\ s[i - 1] & x == i \end{cases} \quad \text{por lo tanto } s[i] == s[i - 1] \equiv True \\ &(|s| \geq 2 \wedge 0 \leq i \leq 1 \wedge (\forall j : \mathbb{Z}) ((0 \leq j < |s| \wedge i \neq j) \rightarrow_L s[j] == 0)) \uparrow \\ &\quad \uparrow \\ &\quad SetAt(s, i, 0)[x] = \begin{cases} s[x] & x \neq i \\ 0 & x == i \end{cases} \quad \text{por lo tanto } s[i] == 0, \text{ así que } \\ &\quad 0 \leq i < |s| \wedge_L True \equiv 0 \leq i < |s| \end{aligned}$$

e)  $Wp(\text{if } s[i] < 0 \text{ then } s[i] := -s[i] \text{ else } Skip \text{ fi}, 0 \leq i < |s| \wedge_L s[i] \geq 0)$

$$\begin{aligned} &\stackrel{\text{Axioma 4}}{\equiv} def(s[i] < 0) \wedge_L ((s[i] < 0) \wedge Wp(s[i] := -s[i], 0 \leq i < |s| \wedge_L s[i] \geq 0)) \vee (\neg(s[i] < 0) \wedge Wp(Skip, 0 \leq i < |s| \wedge_L s[i] \geq 0)) \equiv \\ &0 \leq i < |s| \wedge_L ((s[i] < 0 \wedge -s[i] \geq 0) \vee (s[i] \geq 0 \wedge s[i] \geq 0)) \equiv 0 \leq i < |s| \wedge_L (s[i] < 0 \vee s[i] \geq 0) \equiv \\ &0 \leq i < |s| \wedge_L True \equiv 0 \leq i < |s| \end{aligned}$$

f)  $Wp(\text{if } s[i] > 0 \text{ then } s[i] := -s[i] \text{ else } Skip \text{ fi}, (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))$

$$\begin{aligned} &\stackrel{\text{Axioma 4}}{\equiv} def(s[i] > 0) \wedge_L ((s[i] > 0) \wedge Wp(s[i] := -s[i], (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))) \vee (\neg(s[i] > 0) \wedge Wp(Skip, (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))) \equiv \\ &0 \leq i < |s| \wedge_L ((s[i] > 0 \wedge (\forall j : \mathbb{Z})(0 \leq j < |setAt(s, i, -s[i])| \rightarrow_L setAt(s, i, -s[i])[j] \geq 0)) \vee \\ &(s[i] \leq 0 \wedge (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))) \equiv \\ &0 \leq i < |s| \wedge_L ((s[i] > 0 \wedge (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L ((i = j \wedge -s[j] \geq 0) \vee (i \neq j \wedge s[j] \geq 0)))) \vee \\ &(s[i] == 0 \wedge (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))) \equiv \\ &0 \leq i < |s| \wedge_L (False \vee (s[i] == 0 \wedge (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0))) \\ &\quad \uparrow \\ &\quad s[i] > 0 \wedge -s[i] \geq 0 \equiv False \end{aligned}$$

8. Para este ejercicio tenemos que probar que  $\text{Pre} \rightarrow Wp(S, Post)$

a) Programa :  $a := a + s[i]$

$$\begin{aligned} &Wp(a := a + s[i], a == \sum_{j=0}^i s[j]) \stackrel{\text{Axioma 1}}{\equiv} def(a + s[i]) \wedge_L a + s[i] == \sum_{j=0}^i s[j] \equiv 0 \leq i < |s| \wedge_L a == \sum_{j=0}^{i-1} s[j] \\ &PreProc \rightarrow 0 \leq i < |s| \wedge_L a == \sum_{j=0}^{i-1} s[j] \end{aligned}$$

b) Programa :  $a := a - s[0]$

$$\begin{aligned} Wp(a := a - s[0], a == \sum_{j=1}^i s[j]) &\stackrel{\text{Axioma 1}}{\equiv} \text{def}(\sum_{j=1}^i s[j]) \wedge_L a - s[0] == \sum_{j=1}^i s[j] \equiv 0 \leq i < |s| \wedge_L a == \sum_{j=0}^i s[j] \\ Pre_{Proc} \rightarrow 0 \leq i < |s| \wedge_L a == \sum_{j=1}^i s[j] \end{aligned}$$

c) Programa: if  $res = True$  then  $res = False$  else  $s[i] \geq 0$  fi

$$\begin{aligned} Wp(res = True, res == True \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0)) &\stackrel{\text{Axioma 1}}{\equiv} \\ True == True \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0) &\equiv (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0) \\ Wp(res = False, res == True \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0)) &\stackrel{\text{Axioma 1}}{\equiv} \\ False == True \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0) &\equiv \neg(\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0) \end{aligned}$$

$$\begin{aligned} Wp(\text{if } s[i] \geq 0 \text{ then } res = True \text{ else } res = False \text{ fi}, res == True \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0)) &\stackrel{\text{Axioma 4}}{\equiv} \text{def}(s[i] \geq 0) \wedge_L ((s[i] \geq 0) \wedge Wp(res = True, res == True \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0))) \vee (\neg(s[i] \geq 0) \wedge Wp(res = False, res == True \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0))) \\ 0 \leq i < |s| \wedge_L ((s[i] \geq 0 \wedge (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0)) \vee (s[i] < 0 \wedge \neg(\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0))) &\equiv \\ 0 \leq i < |s| \wedge_L ((\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0) \vee (s[i] < 0)) &\stackrel{\substack{s[i] \geq 0 \\ s[i] < 0 \equiv \text{True}}}{} \rightarrow \neg(\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0) \end{aligned}$$

$$Pre_{proc} \rightarrow_L (0 \leq i < |s| \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \geq 0) \vee (s[i] < 0)))$$

d) Programa :if  $a := a + 1$  then  $Skip$  else  $s[i] \neq 0$  fi

$$\begin{aligned} Wp(a := a + 1, a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) &\equiv \\ a + 1 == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) &\equiv \\ a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1 &\equiv \\ Wp(Skip, a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) &\stackrel{\text{Axioma 2}}{\equiv} a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \end{aligned}$$

$$\begin{aligned} Wp(\text{if } s[i] \neq 0 \text{ then } a := a + 1 \text{ else } Skip \text{ fi}, a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) &\stackrel{\text{Axioma 4}}{\equiv} \text{def}(s[i] \neq 0) \wedge_L ((s[i] \neq 0) \wedge Wp(a := a + 1, a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}))) \vee \\ (\neg(s[i] \neq 0) \wedge Wp(Skip, a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}))) &\equiv \\ 0 \leq i < |s| \wedge_L ((s[i] \neq 0 \wedge a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1) \vee (s[i] == 0 \wedge a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}))) &\equiv \end{aligned}$$

Ahora para demostrar:

$$((0 \leq i < |s| \wedge_L a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}))) \rightarrow (0 \leq i < |s| \wedge_L ((s[i] \neq 0 \wedge a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1) \vee (s[i] == 0 \wedge a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}))))$$

Lo vamos a separar en dos casos, por un lado vamos a ver si se cumple cuándo  $s[i] \neq 0$  es verdadero, y luego veremos si se cumple cuándo  $s[i] == 0$  es verdadero. Cómo esas dos opciones abarcan todos los casos si logramos probar que es verdadero para ambos entonces habrá sido demostrada correctamente la implicación.

**Caso 1**  $s[i] \neq 0$  verdadero :

$$a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1 \equiv a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) + 1 \quad \begin{matrix} \nearrow 1 \\ \uparrow \\ s[i] \neq 0 \end{matrix}$$

**Caso 2**  $s[i] == 0$  verdadero :

$$a == \sum_{j=0}^i (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \equiv a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) + 0 \quad \begin{matrix} \uparrow \\ s[i] \neq 0 \end{matrix} \text{ Luego tenemos que :}$$

$$0 \leq i < |s| \rightarrow 0 \leq i < |s|$$

Y además cómo el caso 1 y caso 2 son iguales basta con decir que :

$$a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \rightarrow a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})$$

Por lo tanto queda demostrada esta implicación  $((0 \leq i < |s| \wedge_L a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) \rightarrow (0 \leq i < |s| \wedge_L ((s[i] \neq 0 \wedge a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) \vee (s[i] == 0 \wedge a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})))) \equiv True$

e) Programa : if  $s[0] \neq 0$  then  $a := a + 1$  else  $Skip$  fi

$$\begin{aligned} Wp(a := a + 1, a == \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) &\equiv \\ 1 \leq i < |s| \wedge_L a == \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1 & \\ Wp(Skip, a == \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) &\stackrel{\text{Axioma 2}}{\equiv} a == \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \end{aligned}$$

$$\begin{aligned} Wp(\text{if } s[0] \neq 0 \text{ then } a := a + 1 \text{ else } Skip \text{ fi}, a == \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) &\equiv \\ \text{Axioma 4} \quad def(s[0] \neq 0) \wedge_L ((s[0] \neq 0) \wedge Wp(a := a + 1, a == \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) \vee & \\ (\neg(s[0] \neq 0) \wedge Wp(Skip, a == \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}))) &\equiv \\ |s| < 0 \wedge_L ((s[0] \neq 0 \wedge 1 \leq i < |s| \wedge_L \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1) \vee & \\ (s[0] == 0 \wedge 0 \leq i < |s| \wedge_L \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) & \end{aligned}$$

Ahora para demostrar:

$$\begin{aligned} (1 \leq i < |s| \wedge_L \sum_{j=1}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) \rightarrow & \\ |s| < 0 \wedge_L ((s[0] \neq 0 \wedge 1 \leq i < |s| \wedge_L \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1) \vee & \\ (s[0] == 0 \wedge 0 \leq i < |s| \wedge_L \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) & \end{aligned}$$

Lo vamos a separar en dos casos, por un lado vamos a ver si se cumple cuándo  $s[0] \neq 0$  es verdadero, y luego veremos si se cumple cuándo  $s[0] == 0$  es verdadero. Cómo esas dos opciones abarcan todos los casos si logramos probar que es verdadero para ambos entonces habrá sido demostrada correctamente la implicación.

**Caso 1**  $s[0] \neq 0$  verdadero :

$$a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1 \equiv a == \sum_{j=1}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \cancel{\uparrow} \cancel{1-1} \\ s[0] \neq 0$$

**Caso 2**  $s[i] == 0$  verdadero :

$$a == \sum_{j=0}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \equiv a == \sum_{j=1}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \cancel{\uparrow} \cancel{0} \\ s[0] == 0$$

Luego tenemos que :

$$1 \leq i < |s| \rightarrow (1 \leq i < |s| \wedge |s| > 0)$$

Y además cómo el caso 1 y caso 2 son iguales basta con decir que :

$$a == \sum_{j=1}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) \rightarrow a == \sum_{j=1}^{i-1} (\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})$$

Por lo tanto queda demostrada esta implicación  $(1 \leq i < |s| \wedge_L \sum_{j=1}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})) \rightarrow |s| < 0 \wedge_L ((s[0] \neq 0 \wedge 1 \leq i < |s| \wedge_L \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi}) - 1) \vee (s[0] == 0 \wedge 0 \leq i < |s| \wedge_L \sum_{j=0}^{i-1} (\text{if } s[0] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})))$

### 3. Bonus Track : Probar la corrección de un programa con alternativas

Cómo ya vimos en esta práctica, probar la corrección de un programa con alternativas es un proceso largo y complejo. Para probar la corrección del mismo debemos demostrar que

$$Pre_{proc} \rightarrow Wp(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}, Q) \stackrel{\text{Axioma 4}}{\equiv} def(B) \wedge_L ((B) \wedge Wp(S_1, Q)) \vee (\neg(B) \wedge Wp(S_2, Q))$$

Pero no desesperes!! Pues hay una equivalencia lógica la cuál puede ayudarte. Si renombramos esta expresión llamando :

$$\begin{cases} a := Pre_{Proc} \\ b := B \\ wp1 := WP(S_1, Q) \\ wp2 := WP(S_2, Q) \end{cases}$$

Nos queda la expresión lógica :

$$a \rightarrow (def(b) \wedge_L ((b \wedge wp1) \vee (\neg b \wedge wp2)))$$

la cuál luego de manipular aplicando algunas propiedades es equivalente a :

$$((a \wedge b \wedge def(b)) \rightarrow wp1) \wedge (a \wedge \neg b \wedge def(b)) \rightarrow wp2)$$

lo que sería un "separar en casos" lógico. Ya que debemos probar que cuándo a y b son verdaderos, wp1 también lo es ( $(a \wedge b \wedge def(b)) \rightarrow wp1$ ) y que cuándo a y  $\neg b$  son verdaderos, wp2 también lo es ( $(a \wedge \neg b \wedge def(b)) \rightarrow wp2$ ). Sólo tomamos el caso en el que a y b (o a y  $\neg b$ ) son verdaderos puesto que si ese predicado es falso entonces la implicación es verdadera. Ya que  $False \rightarrow \dots \equiv True$ . Este es un "tip" para poder probar esta implicación.



## Teorema del invariante: corrección de ciclos

**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: seq<Z>, out result: Z) {
 Pre {true}
 Post {result = $\sum_{j=0}^{|s|-1} s[j]$ }
}
```

### Implementación en SmallLang

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

### Invariante de Ciclo

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

- a) Escribir la precondición y la postcondición del ciclo.
- b) ¿Qué punto falla en la demostración de corrección si el primer término de la sumatoria 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) Demostrar formalmente la corrección parcial del ciclo, usando los primeros puntos del teorema del invariante.
- f) Proponer una función variante y demostrar formalmente la terminación del ciclo, utilizando la función variante.

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

### Especificación

```
proc sumarParesHastaN (in n: Z, out result: Z) {
 Pre {n ≥ 0}
 Post {result = $\sum_{j=0}^{n-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0)$ }
}
```

### Implementación en SmallLang

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

### Invariante de ciclo

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

**Ejercicio 3.** Supongamos que se desea implementar la función `exponenciacion`, cuya especificación es la siguiente:

```
proc exponenciacion (in m: Z, in n: Z, out result: Z) {
 Pre { $n \geq 0 \wedge \neg(m = 0 \wedge n = 0)$ }
 Post { $result = m^n$ }
}
```

Consideremos además el siguiente invariante:  $I \equiv 0 \leq i \leq n \wedge result = m^i$

- a) Escribir un programa en SmallLang que resuelva este problema, y que incluya un ciclo que tenga a  $I$  como invariante. Demostrar formalmente la corrección de este ciclo.

- b) La siguiente implementación en SmallLang es trivialmente errónea. ¿Qué punto del teorema del invariante falla en este caso?<sup>1</sup>

```
i := 0;
result := 0;
```

```
while(i < m) do
 result := result * n;
 i := i + 1
endwhile
```

- c) ¿Qué puede decir de la siguiente implementación en SmallLang? En caso de que sea correcta, proporcione una demostración. En caso de que sea incorrecta, explique qué punto del teorema del invariante falla.

```
i := 0;
result := 1;
while(i < n) do
 i := i + 1;
 result := result * m
endwhile
```

- d) ¿Qué puede decir de la siguiente implementación? En caso de que sea incorrecta, ¿se puede reforzar la precondition del problema para que esta especificación pase a ser correcta?

```
i := 2;
result := m*m;

while(i < n) do
 result := result * m;
 i := i + 1
endwhile
```

**Ejercicio 4.** ★ Considere el problema `sumaDivisores`, dado por la siguiente especificación:

```
proc sumaDivisores (in n: Z, out result: Z) {
 Pre { $n \geq 1$ }
 Post { $result = \sum_{j=1}^n (\text{if } n \bmod j = 0 \text{ then } j \text{ else } 0 \text{ fi})$ }
}
```

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

- b) El ciclo del programa propuesto, ¿puede ser demostrado mediante el siguiente invariante?

$$I \equiv 1 \leq i \leq n \wedge result = \sum_{j=1}^i (\text{if } n \bmod j = 0 \text{ then } j \text{ else } 0 \text{ fi})$$

Si no puede, ¿qué cambios se le deben hacer al invariante para que se corresponda con el ciclo propuesto?

---

<sup>1</sup>Recordar que para mostrar que una implicación  $A \rightarrow B$  no es cierta alcanza con dar valores de las variables libres que hagan que  $A$  sea verdadero y que  $B$  sea falso. Para mostrar que una tripla de Hoare  $\{P\}S\{Q\}$  no es válida, alcanza con dar valores de las variables que satisfacen  $P$ , y tales que luego de ejecutar  $S$ , el estado final no satisface  $Q$ .

**Ejercicio 5.** Considere la siguiente especificación de la función `sumarPosicionesImpares`.

```
proc sumarPosicionesImpares (in s: seq<Z>, out result: Z) {
 Pre {true}
 Post {result = $\sum_{i=0}^{|s|-1}$ (if $i \bmod 2 = 1$ then $s[i]$ else 0 fi)}
}
```

- a) Implementar un programa en SmallLang para resolver este problema, que incluya exactamente un ciclo con el siguiente invariante:

$$I \equiv 0 \leq j \leq |s| \wedge_L result = \sum_{i=0}^{j-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})$$

- b) Demostrar formalmente la corrección del ciclo propuesto.

**Ejercicio 6.** Considere la siguiente especificación e implementación del problema `maximo`.

#### Especificación

```
proc maximo (in s: seq<Z>, out i: Z) {
 Pre { $|s| \geq 1$ }
 Post { $0 \leq i < |s| \wedge_L (\forall j : Z)(0 \leq j < |s| \rightarrow_L s[j] \leq s[i])$ }
}
```

#### Implementación en SmallLang

```
i := 0;
j := 1;
while (j < s.size()) do
 if (s[j] > s[i])
 i := j
 else
 skip
 endif;
 j := j + 1
endwhile
```

- a) Escribir la precondición y la postcondición del ciclo.

- b) Demostrar que el ciclo es parcialmente correcto, utilizando el siguiente invariante:

$$I \equiv (0 \leq i < |s| \wedge 1 \leq j \leq |s|) \wedge_L (\forall k : Z)(0 \leq k < j \rightarrow_L s[k] \leq s[i])$$

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

**Ejercicio 7.** ★ Considere la siguiente especificación e implementación del problema `copiarSecuencia`.

#### Especificación

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

#### Implementación en SmallLang

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

- a) Escribir la precondición y la postcondición del ciclo.

- b) Proponer un invariante y demostrar que el ciclo es parcialmente correcto.

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

**Ejercicio 8.** Considere la siguiente especificación e implementación del problema `llenarSecuencia`.

#### Especificación

```
proc llenarSecuencia (inout s: seq<Z>, in d: Z, in e: Z) {
 Pre { $d \geq 0 \wedge d < |s| \wedge s = S_0$ }
 Post { $|s| = |S_0| \wedge_L (\forall j : Z)(0 \leq j < d \rightarrow_L s[j] = S_0[j]) \wedge (\forall j : Z)(d \leq j < |s| \rightarrow_L s[j] = e)$ }
}
```

#### Implementación en SmallLang

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

- a) Escribir la precondición y la postcondición del ciclo.

- b) Proponer un invariante y demostrar que el ciclo es parcialmente correcto.
- c) Proponer una función variante que permita demostrar que el ciclo termina.

**Ejercicio 9.** ★ Sea el siguiente ciclo con su correspondiente precondición y postcondición:

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

$$P_c : \{|s| \bmod 2 = 0 \wedge i = |s| - 1 \wedge suma = 0\}$$

$$Q_c : \{|s| \bmod 2 = 0 \wedge i = |s|/2 - 1 \wedge_L suma = \sum_{j=0}^{|s|/2-1} s[j]\}$$

- a) Especificar un invariante de ciclo que permita demostrar que el ciclo cumple la postcondición.
- b) Especificar una función variante que permita demostrar que el ciclo termina.
- c) Demostrar formalmente la corrección y terminación del ciclo usando el Teorema del invariante.

**Ejercicio 10.** Considere la siguiente especificación del problema **reemplazarTodos**.

```
proc reemplazarTodos (inout s: seq<Z>, in a: Z, in b: Z) {
 Pre {s = S₀}
 Post {|s| = |S₀| \wedge_L
 (\forall j : Z)((0 \leq j < |s| \wedge_L S₀[j] = a) \rightarrow_L s[j] = b) \wedge
 (\forall j : Z)(d \leq j < |s| \wedge_L S₀[j] \neq a) \rightarrow_L s[j] = S₀[j])}
}
```

- a) Dar un programa en SmallLang que implemente la especificación dada.
- b) Escribir la precondición y la postcondición del ciclo.
- c) Proponer un invariante y demostrar que el ciclo es parcialmente correcto.
- d) Proponer una función variante que permita demostrar que el ciclo termina.

## Demostración de correctitud: programas completos

**Ejercicio 11.** ★ Demostrar que el siguiente programa es **correcto** respecto a la especificación dada.

### Especificación

```
proc indice (in s: seq<Z>, in e: Z, out r: Z) {
 Pre {True}
 Post {r = -1 \rightarrow
 (\forall j : Z)(0 \leq j < |s| \rightarrow_L s[j] \neq e)
 \wedge
 r \neq -1 \rightarrow
 (0 \leq r < |s| \wedge_L s[r] = e)}
}
```

### Implementación en SmallLang

```
i := s.size() - 1;
j := -1;
while (i >= 0) do
 if (s[i] = e) then
 j := i
 else
 skip
 endif;
 i := i - 1
endwhile;
r := j;
```

**Ejercicio 12.** ★ Demostrar que el siguiente programa es correcto respecto a la especificación dada.

#### Especificación

```
proc existeElemento (in s: seq<Z>, in e: Z, out r: Bool) {
 Pre {True}
 Post {r = 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)
 r := true
else
 r := false
endif
```

**Ejercicio 13.** Demostrar que el siguiente programa es correcto respecto a la especificación dada.

#### Especificación

```
proc esSimetrico (in s: seq<Z>, out r:Bool) {
 Pre {True}
 Post {r = True \leftrightarrow ($\forall i : \mathbb{Z}$)($0 \leq i < |s| \rightarrow_L s[i] = s[|s| - (i + 1)]$)}
}
```

#### Implementación en SmallLang

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

**Ejercicio 14.** ★ Demostrar que el siguiente programa es correcto respecto a la especificación dada.

#### Especificación

```
proc concatenarSecuencias (in a: seq<Z>,
in b: seq<Z>,
inout r:seq<Z>) {
 Pre { $|r| = |a| + |b| \wedge r = R_0$ }
 Post { $|r| = |R_0| \wedge (\forall j : \mathbb{Z})(0 \leq j < |a| \rightarrow_L r[j] = a[j]) \wedge (\forall j : \mathbb{Z})(0 \leq j < |b| \rightarrow_L r[j + |a|] = b[j])$ }
}
```

#### Implementación en SmallLang

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

**Ejercicio 15.** Dar dos programas en SmallLang que satisfagan la siguiente especificación, y demostrar que ambos son correctos.

```
proc buscarPosicionUltimoMaximo (in s: seq<Z>, out r:Z) {
 Pre { $|s| > 0$ }
 Post { $0 \leq r < |s| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j])$ }
}
```

## 1. Axiomas a utilizar

$$1. \text{ Asignación : } Wp(x := E, Q) \stackrel{\text{Axioma 1}}{\equiv} def(E) \wedge_L Q_E^x$$

Asignación con Secuencias:  $Wp(A[i] := k, Q) \equiv Wp(A := setAt(A, i, k), Q) \equiv$

$$def(setAt(A, i, k)) \wedge Q_A^{setAt(A, i, k)}$$

$$SetAt(A, i, 0)[x] = \begin{cases} A[x] & x \neq i \\ 0 & x = i \end{cases}$$

$$2. \text{ Skip : } Wp(Skip, Q) \stackrel{\text{Axioma 2}}{\equiv} Q$$

$$3. S1;S2 : Wp(S1; S2, Q) \stackrel{\text{Axioma 3}}{\equiv} Wp(S1, Wp(S2, Q))$$

$$4. \text{ Alternativa: } Wp(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}, Q)$$

$$\stackrel{\text{Axioma 4}}{\equiv} def(B) \wedge_L ((B) \wedge Wp(S_1, Q)) \vee (\neg(B) \wedge Wp(S_2, Q))$$

## 2. Practica 5: Demostracion de corrección de ciclos en SmallLang

### Aclaracion

Si queremos demostrar que una implicación del estilo  $A \rightarrow B$  es una tautología (ergo es verdadera siempre) solo nos vamos a preocupar del caso en el que  $A$  sea verdadero. Puesto que si  $A$  es Falso entonces queda Falso  $\rightarrow B$  que siempre es verdadero. Por lo tanto al demostrar la correctitud de estas siempre buscaremos probar que "cuando  $A$  es verdadero entonces  $B$  es verdadero también.

PD: No siempre estan bien puestos los parentesis en las implicaciones, por ejemplo si  $A = c \wedge d$  y  $B = e \wedge f$ , entonces  $A \rightarrow B$  puede estar escrito como  $c \wedge d \rightarrow e \wedge f$  en vez de  $(c \wedge d) \rightarrow (e \wedge f)$ . Aún así se entiende cómo deben estar separados los predicados.

$$1. a) P_c \equiv result = 0 \wedge i = 0 \text{ y } Q_c \equiv result = \sum_{j=0}^{|s|-1} s[j] \wedge i = |s|$$

b) Falla cuando  $i = |s|$  al salir del ciclo, porque no se vale el Invariante

c) Falle cuando  $i = |s|$  al salir del ciclo, porque  $s[|s|]$  no está definido

d) Falla en dos momentos: Se saltaría el primer valor de  $s$  en la suma; Y en la ultima iteración  $i = |s|$  y  $s[|s|]$  está indefinido

e)

$$P_c \rightarrow I : i = 0 \rightarrow 0 \leq i \rightarrow 0 \leq i \leq |s|$$

$$result = 0 \wedge i = 0 \rightarrow result = \sum_{j=0}^{-1} s[j] = \sum_{j=0}^{i-1} s[j] \square$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$S_0 \equiv result := result + s[i]; \quad S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; \quad S_1;$$

$$\Rightarrow Wp(S, I) \equiv Wp(S_0; S_1, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(S_0, Wp(S_1, I))$$

$$Wp(S_1, I) \stackrel{\text{Axioma 1}}{\equiv} 0 \leq i + 1 \leq |s| \wedge_L result = \sum_{j=0}^i s[j]$$

$$\Rightarrow Wp(S_0, Wp(S_1, I)) \equiv Wp(result := result + s[i], 0 \leq i + 1 \leq |s| \wedge_L result = \sum_{j=0}^i s[j])$$

$$\stackrel{Axioma\ 1}{\equiv} 0 \leq i + 1 \leq |s| \wedge_L result + s[i] = \sum_{j=0}^i s[j]$$

$$0 \leq i \leq |s| \wedge i < |s| \rightarrow 0 \leq i + 1 \leq |s|$$

$$i \leq |s| \wedge_L result = \sum_{j=0}^{i-1} s[j] \rightarrow result + s[i] = \sum_{j=0}^{i-1} (s[j]) + s[i] = \sum_{j=0}^i s[j] \square$$

$$(I \wedge \neg B) \rightarrow Qc : \neg (i < |s|) \equiv i \geq |s|$$

$$0 \leq i \leq |s| \wedge i \geq |s| \equiv i = |s|$$

$$i = |s| \wedge_L result = \sum_{j=0}^{i-1} s[j] \rightarrow result = \sum_{j=0}^{|s|-1} s[j] \square$$

f)

$$4to\ paso:\ f_v(i) = |s| - i$$

$$\{I \wedge B \wedge v_0 = f_v\} S \{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$S_0 \equiv result := result + s[i];\ S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0;\ S_1;$$

$$\Rightarrow Wp(S, f_v < v_0) \equiv Wp(S_0; S_1, f_v < v_0) \stackrel{Axioma 3}{\equiv} Wp(S_0, Wp(S_1, f_v < v_0))$$

$$Wp(S_1, f_v < v_0) \equiv Wp(i := i + 1, |s| - i < v_0) \stackrel{Axioma\ 1}{\equiv} |s| - (i + 1) = |s| - i - 1 < v_0$$

$$Wp(S_0, |s| - i - 1 < v_0) \equiv Wp(result := result + s[i], |s| - i - 1 < v_0) \stackrel{Axioma\ 1}{\equiv} 0 \leq i < |s| \wedge |s| - i - 1 < v_0$$

$$0 \leq i \leq |s| \wedge i < |s| \rightarrow 0 \leq i < |s|$$

$$v_0 = f_v = |s| - i \rightarrow |s| - i - 1 < |s| - i \equiv -1 < 0 \equiv True \square$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |s| \wedge |s| - i \leq 0 \rightarrow i = |s|$$

$$i = |s| \rightarrow i \geq |s| \equiv \neg B \square$$

2.  $Pc : i = 0 \wedge result = 0 \wedge n \geq 0$

$$Qc : result = \sum_{j=0}^{n-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0)$$

$$Pc \rightarrow I : i = 0 \rightarrow 0 \leq i \wedge i \bmod 2 = 0 \rightarrow 0 \leq i \leq n + 1 \wedge i \bmod 2 = 0$$

$$i = 0 \wedge result = 0 \rightarrow result = 0 = \sum_{j=0}^{n-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \quad \square$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$\begin{aligned} S_0 &\equiv result := result + i; \quad S_1 \equiv i := i + 2; \Rightarrow S \equiv S_0; \quad S_1; \\ \Rightarrow Wp(S, I) &\equiv Wp(S_0; S_1, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(S_0, Wp(S_1, I)) \end{aligned}$$

$$\begin{aligned} Wp(S_1, I) &\stackrel{\text{Axioma 1}}{\equiv} 0 \leq i + 2 \leq n + 1 \wedge i + 2 \bmod 2 = 0 \wedge \sum_{j=0}^{i+2-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \equiv \\ &0 \leq i + 2 \leq n + 1 \wedge i \bmod 2 = 0 \wedge \sum_{j=0}^{i+1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \\ \Rightarrow Wp(S_0, Wp(S_1, I)) &\equiv Wp(S_0, 0 \leq i + 2 \leq n + 1 \wedge i \bmod 2 = 0 \wedge result = \sum_{j=0}^{i+1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \equiv \\ &0 \text{ then } j \text{ else } 0) \stackrel{\text{Axioma 1}}{\equiv} 0 \leq i + 2 \leq n + 1 \wedge i \bmod 2 = 0 \wedge result + i = \sum_{j=0}^{i+1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \\ &0 \text{ then } j \text{ else } 0) \end{aligned}$$

$$0 \leq i \leq n + 1 \wedge i < n \rightarrow 0 \leq i + 2 \leq n + 1$$

$$\begin{aligned} i \bmod 2 = 0 \wedge result &= \sum_{j=0}^{i-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \rightarrow \\ result + i &= \sum_{j=0}^{i+1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) = \sum_{\substack{j=0 \\ i \bmod 2 = 0}}^{i-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) + i \equiv \\ result &= \sum_{j=0}^{i-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \quad \square \end{aligned}$$

$$(I \wedge \neg B) \rightarrow Qc : \neg i < n \equiv i \geq n$$

$$0 \leq i \leq n + 1 \wedge i \geq n \equiv i = n \vee i = n + 1$$

$$\begin{aligned} (i = n \vee i = n + 1 \wedge i \bmod 2 = 0) \wedge_L result &= \sum_{j=0}^{i-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \rightarrow \\ result &= \sum_{j=0}^{n-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0) \quad \square \end{aligned}$$

4to paso:  $f_v(i) = n - i$

$$\begin{aligned} \{I \wedge B \wedge v_0 = f_v\}S\{f_v < v_0\} &\equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0) \\ S_0 &\equiv result := result + i; \quad S_1 \equiv i := i + 2; \Rightarrow S \equiv S_0; \quad S_1; \\ \Rightarrow Wp(S, f_v < v_0) &\equiv Wp(S_0; S_1, f_v < v_0) \stackrel{\text{Axioma 3}}{\equiv} Wp(S_0, Wp(S_1, f_v < v_0)) \end{aligned}$$

$$\begin{aligned} Wp(S_1, f_v < v_0) &\stackrel{\text{Axioma 1}}{\equiv} n - i - 2 < v_0 \\ Wp(S_0, Wp(S_1, f_v < v_0)) &\stackrel{\text{Axioma 1}}{\equiv} n - i - 2 < v_0 \\ \Rightarrow Wp(S, f_v < v_0) &\equiv n - i - 2 < v_0 \end{aligned}$$

$$v_0 = n - i \rightarrow n - i - 2 < v_0 \equiv -2 < 0 \quad \square$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq n + 1 \wedge n - i \leq 0 \rightarrow i = n \vee i = n + 1$$

$$i = n \vee i = n + 1 \rightarrow i \geq n \equiv \neg B \quad \square$$

3. a)  $S \equiv$

---

```

1 If m=0 then result:=0;i=n else result:=1;i=0 fi
2 while (i < n) do
3 i := i+1;
4 result := result*m;
5 endwhile

```

---

$$Qc \equiv result = m^n$$

$$Pc \equiv n \geq 0 \wedge ((m = 0 \wedge i = n \wedge res = 0) \vee (m \neq 0 \wedge i = 0 \wedge res = 1))$$

$Pc \rightarrow I$  : Caso  $m \neq 0$ :

$$\begin{aligned} i = 0 \wedge n \geq 0 &\rightarrow 0 \leq i \leq n \\ i = 0 \wedge result = 1 &\rightarrow result = m^i \square \end{aligned}$$

Caso  $m = 0$ : Trivial

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$\begin{aligned} S_0 &\equiv i := i + 1; \quad S_1 \equiv result := result * m; \Rightarrow S \equiv S_0; \quad S_1; \\ \Rightarrow Wp(S, I) &\equiv Wp(S_0; \quad S_1, I) \stackrel{\text{Axioma } 3}{=} Wp(S_0, Wp(S_1, I)) \\ Wp(S_1, I) &\stackrel{\text{Axioma } 1}{=} 0 \leq i \leq n \wedge result * m = m^i \\ \Rightarrow Wp(S_0, Wp(S_1, I)) &\equiv Wp(S_0, 0 \leq i \leq n \wedge result * m = m^i) \stackrel{\text{Axioma } 1}{=} 0 \leq i + 1 \leq n \wedge result * m = \\ &m^{i+1} \end{aligned}$$

$$0 \leq i \leq n \wedge i < n \rightarrow 0 \leq i + 1 \leq n$$

$$result = m^i \rightarrow result * m = m^i * m = m^{i+1} \square$$

$$(I \wedge \neg B) \rightarrow Qc : \neg i < n \equiv i \geq n$$

$$\begin{aligned} 0 \leq i \leq n \wedge i \geq n &\equiv i = n \\ i = n \wedge result = m^i &\rightarrow result = m^n \square \end{aligned}$$

$$4\text{to paso } f(i) = n - i$$

$$\begin{aligned} \{I \wedge B \wedge v_0 = f_v\}S\{f_v < v_0\} &\equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0) \\ S_0 &\equiv i := i + 1; \quad S_1 \equiv result := result * m; \quad S \equiv S_0; \quad S_1; \\ \Rightarrow Wp(S, f_v < v_0) &\equiv Wp(S_0; \quad S_1, f_v < v_0) \stackrel{\text{Axioma } 3}{=} Wp(S_0, Wp(S_1, f_v < v_0)) \\ Wp(S_1, f_v < v_0) &\equiv Wp(result := result * m, n - i < v_0) \stackrel{\text{Axioma } 1}{=} n - i < v_0 \\ Wp(S_0, Wp(S_1, f_v < v_0)) &\equiv Wp(i := i + 1, n - i < v_0) \stackrel{\text{Axioma } 1}{=} n - i - 1 < v_0 \end{aligned}$$

$$v_0 = f_v = n - i \rightarrow n - i - 1 < v_0 \equiv -1 < 0 \equiv True \square$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq n \wedge n - i \leq 0 \rightarrow i = n$$

$$i = n \rightarrow i \geq n \equiv \neg B \square$$

- b) Falla en el primer paso porque  $result = 0 \rightarrow result \neq m^i$  para cualquier  $m$  e  $i$
- c) Falla si se introduce un  $m=0$ , puesto que es imposible que  $res=0$ .
- d) Es incorrecta, ya que si  $n=0,1 \rightarrow 0 \leq i \leq n$  no se va a cumplir .

4. a)

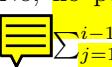
---

```

1 result := 0;
2 i := 1;
3 while (i <= n) do
4 if (n mod i = 0) then
5 result := result+i
6 else
7 skip
8 endif
9 i := i +1
10 endwhile

```

---

- b) No,  no puede ser demostrado con ese Invariante. Se debe modificar  $1 \leq i \leq |n| + 1$  y además  $\sum_{j=1}^{i-1} \dots$

5.  $S \equiv$

---

```

1 result := 0;
2 j := 0;
3 while (j < s.size()) do
4 if (j mod 2 = 1) then
5 result := result + s[j];
6 else
7 skip;
8 endif
9 j := j+1;
10 endwhile

```

---

$$Qc \equiv result = \sum_{i=0}^{|s|-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})$$

$$Pc \equiv j=0 \wedge res=0$$

$$f_v(j) = |s| - j$$

$$Pc \rightarrow I : j = 0 \rightarrow 0 \leq j \leq |s|$$

$$j = 0 \wedge result = 0 \rightarrow result = \sum_{i=0}^{j-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi}) = 0 \square$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$\begin{aligned} C &\equiv j \bmod 2 = 1; S_0 \equiv result := result + s[j]; S_1 \equiv skip; S_2 \equiv j := j + 1; S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}; S_2 \\ Wp(S, I) &\stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, I)) \\ &\stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, I)))) \\ &\stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_2, I))) \end{aligned}$$

$$def(C) \equiv def(j \bmod 2 = 1) \equiv True$$

$$\neg C \equiv j \bmod 2 = 0$$

$$\begin{aligned} Wp(S_2, I) &\stackrel{\text{Axioma 1}}{\equiv} 0 \leq j + 1 \leq |s| \wedge_L result = \sum_{i=0}^j (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi}) \\ Wp(S_0, Wp(S_1, I)) &\stackrel{\text{Axioma 1}}{\equiv} 0 \leq j + 1 \leq |s| \wedge_L result + s[j] = \sum_{i=0}^j (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi}) \end{aligned}$$

$$\begin{aligned} \Rightarrow Wp(S, I) &\equiv (j \bmod 2 = 1 \wedge 0 \leq j + 1 \leq |s| \wedge_L result + s[j] = \sum_{i=0}^j (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})) \vee \\ &(j \bmod 2 = 0 \wedge 0 \leq j + 1 \leq |s| \wedge_L result = \sum_{i=0}^j (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})) \\ &\equiv 0 \leq j + 1 \leq |s| \wedge_L (j \bmod 2 = 1 \wedge result + s[j] = \sum_{i=0}^j (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})) \vee (j \bmod 2 = 0 \wedge result = \sum_{i=0}^j (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})) \equiv 0 \leq j + 1 \leq |s| \wedge_L \sum_{i=0}^{j-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi}) \end{aligned}$$

$$0 \leq j \leq |s| \wedge j < |s| \rightarrow 0 \leq j + 1 \leq |s|$$

$$\begin{aligned} j < |s| \wedge result &= \sum_{i=0}^{j-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi}) \rightarrow \\ (0 \leq j + 1 \leq |s| \wedge_L \sum_{i=0}^{j-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})) &\square \end{aligned}$$

$$(I \wedge \neg B) \rightarrow Qc : \neg j < |s| \equiv j \geq |s|$$

$$0 \leq j \leq |s| \wedge j \geq |s| \equiv j = |s|$$

$$\begin{aligned} j = |s| \wedge_L result &= \sum_{i=0}^{j-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi}) \rightarrow \\ result &= \sum_{i=0}^{|s|-1} (\text{if } i \bmod 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi}) \square \end{aligned}$$

4to paso  $f(j) = |s| - j$

$$\{I \wedge B \wedge v_0 = f_v\}S\{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$\begin{aligned} C &\equiv j \bmod 2 = 1; S_0 \equiv result := result + s[j]; S_1 \equiv skip; S_2 \equiv j := j + 1; S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}; S_2 \\ Wp(S, f_v < v_0) &\stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, f_v < v_0)) \\ &\stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, f_v < v_0)))) \\ &\stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_2, f_v < v_0))) \end{aligned}$$

$$def(C) \equiv def(j \bmod 2 = 1) \equiv True$$

$$\neg C \equiv j \bmod 2 = 0$$

$$Wp(S_2, f_v < v_0) \equiv Wp(j := j + 1, |s| - j < v_0) \stackrel{\text{Axioma 1}}{\equiv} |s| - j - 1 < v_0$$

$$Wp(S_0, Wp(S_2, f_v < v_0)) \xrightarrow{Axioma\ 1} |s| - j - 1 < v_0$$

$$\begin{aligned} Wp(S, f_v < v_0) &\equiv (j \bmod 2 = 1 \wedge |s| - j - 1 < v_0) \vee (j \bmod 2 = 0 \wedge |s| - j - 1 < v_0) \\ &\equiv |s| - j - 1 \wedge (j \bmod 2 = 1 \vee j \bmod 2 = 0) \equiv |s| - j - 1 \end{aligned}$$

$$v_0 = |s| - j \rightarrow |s| - j - 1 < v_0 \equiv -1 < 0_{\square}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq j \leq |s| \wedge |s| - j \leq 0 \rightarrow j = |s|$$

$$j = |s| \rightarrow j \geq |s| \equiv \neg B_{\square}$$

6. a)

$$Pc \equiv i = 0 \wedge j = 1 \wedge |s| \geq 1$$

$$Qc \equiv 0 \leq i < |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < |s| \longrightarrow_L s[j] \leq s[i])$$

b)

$$Pc \rightarrow I : i = 0 \rightarrow 0 \leq i < |s|$$

$$j = 1 \wedge |s| \geq 1 \rightarrow 1 \leq j \leq |s|$$

$$i = 0 \wedge j = 1 \wedge |s| \geq 1 \rightarrow (\forall k : \mathbb{Z})(0 \leq k < j \longrightarrow_L s[k] \leq s[i]) \equiv s[0] \leq s[0] \equiv True_{\square}$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$C \equiv s[j] > s[i]; S_0 \equiv i := j; S_1 \equiv skip; S_2 \equiv j := j + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi } S_2$$

$$Wp(S, I) \xrightarrow{Axioma\ 3} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, I))$$

$$\xrightarrow{Axioma\ 4} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, I))))$$

$$\xrightarrow{Axioma\ 2} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_2, I)))$$

$$def(C) \equiv def(s[j] > s[i]) \equiv 0 \leq i < |s| \wedge 0 \leq j < |s|$$

$$\neg C \equiv s[j] \leq s[i]$$

$$Wp(S_2, I) \xrightarrow{Axioma\ 1} (0 \leq i < |s| \wedge 1 \leq j + 1 \leq |s|) \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[i])$$

$$Wp(S_0, Wp(S_1, I)) \xrightarrow{Axioma\ 1} (0 \leq j < |s| \wedge 1 \leq j + 1 \leq |s|) \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[j]) \equiv 1 \leq j + 1 \leq |s| \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[j])$$

$$\Rightarrow Wp(S, I) \equiv (0 \leq i < |s| \wedge 0 \leq j < |s|) \wedge_L ((s[j] > s[i]) \wedge 1 \leq j + 1 \leq |s| \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[j])) \vee (s[j] \leq s[i] \wedge (0 \leq i < |s| \wedge 1 \leq j + 1 \leq |s|) \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[i])) \equiv (0 \leq i < |s| \wedge 1 \leq j + 1 \leq |s|) \wedge_L ((s[j] > s[i]) \wedge (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[i])) \vee (s[j] \leq s[i] \wedge (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[i]))$$

$$0 \leq j \leq |s| \wedge j < |s| \rightarrow 0 \leq j < |s|$$

**Caso 1:**

$$s[j] > s[i] \wedge (\forall k : \mathbb{Z})(0 \leq k < j \longrightarrow_L s[k] \leq s[i]) \rightarrow (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[j])$$

**Caso 2:**

$$s[j] \leq s[i] \wedge (\forall k : \mathbb{Z})(0 \leq k < j \longrightarrow_L s[k] \leq s[i]) \rightarrow \wedge (\forall k : \mathbb{Z})(0 \leq k < j + 1 \longrightarrow_L s[k] \leq s[i]))_{\square}$$

**Explicación escrita:** Si se cumple el I entonces  $s[i]$  es el máximo actual, y si además  $s[j] > s[i]$  entonces por transitividad de la desigualdad  $S[j]$  es el nuevo máximo, por lo tanto se cumple este punto del invariante.

$$(I \wedge \neg B) \rightarrow Qc : \neg j < |s| \equiv j \geq |s|$$

$$1 \leq j \leq |s| \wedge j \geq |s| \equiv j = |s|$$

$$0 \leq i < |s| \wedge j = |s| \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j \longrightarrow_L s[k] \leq s[i]) \rightarrow (\forall k : \mathbb{Z})(0 \leq k < |s| \longrightarrow_L s[k] \leq s[i])_{\square}$$

$$c) f_v(j) = |s| - j$$

$$4\text{to paso: } \{I \wedge B \wedge v_0 = f_v\} S\{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$S_1 \equiv \text{skip}; S_2 \equiv j := j + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi} S_2$$

$$Wp(S, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, f_v < v_0))$$

$$\stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, f_v < v_0))))$$

$$\stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_2, f_v < v_0)))$$

$$def(C) \equiv def(s[j] > s[i]) \equiv 0 \leq i < |s| \wedge 0 \leq j < |s|$$

$$\neg C \equiv s[j] \leq s[i]$$

$$Wp(S_2, f_v < v_0) \stackrel{\text{Axioma 1}}{\equiv} |s| - j - 1 < v_0$$

$$Wp(S_0, Wp(S_2, f_v < v_0)) \stackrel{\text{Axioma 1}}{\equiv} |s| - j - 1 < v_0$$

$$\Rightarrow Wp(S, f_v < v_0) \equiv (0 \leq i < |s| \wedge 0 \leq j < |s|) \wedge_L ((s[j] > s[i] \wedge |s| - j - 1 < v_0) \vee (s[j] \leq s[i] \wedge |s| - j - 1 < v_0)) \equiv 0 \leq i < |s| \wedge 0 \leq j < |s| \wedge |s| - j - 1 < v_0$$

$$1 \leq j \leq |s| \wedge j < |s| \equiv 0 \leq j < |s|$$

$$v_0 = |s| - j \rightarrow |s| - j - 1 < v_0 \equiv -1 < 0 \equiv \text{True}_{\square}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 1 \leq j \leq |s| \wedge |s| - j \leq 0 \rightarrow j = |s|$$

$$j = |s| \rightarrow j \geq |s| \equiv \neg B_{\square}$$

7. a)

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

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

b)

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

$$Pc \rightarrow I : i = 0 \rightarrow 0 \leq i \leq |s|$$

$$i = 0 \rightarrow (\forall j : \mathbb{Z})(0 \leq j < 0 \rightarrow_L s[j] = r[j]) = s = \text{True}_\square$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$S_0 \equiv r[i] := s[i]; S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; S_1;$$

$$Wp(S, I) \stackrel{\text{Axioma 3}}{=} Wp(S_0, Wp(S_1, I))$$

$$Wp(S_1, I) \stackrel{\text{Axioma 1}}{=} (|s| = |r| \wedge 0 \leq i + 1 \leq |s|) \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i + 1 \rightarrow_L s[j] = r[j])$$

$$Wp(S_0, Wp(S_1, I)) \stackrel{\text{Axioma 1}}{=} (|s| = |SetAt(r, i, s[i])| \wedge 0 \leq i + 1 \leq |s|) \wedge_L (\forall j : \mathbb{Z})(0 \leq j <$$

$$i + 1 \rightarrow_L s[j] = SetAt(r, i, s[i])[j])$$

$$\equiv (|s| = |r| \wedge 0 \leq i + 1 \leq |s|) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < i + 1 \rightarrow_L ((i = j \wedge s[j] = s[i]) \vee \underset{\substack{\uparrow \\ \text{SetAt}(r, i, s[i])[i]=s[i]}}{(j = i \wedge s[j] = r[j]})))$$

$$\equiv (|s| = |r| \wedge 0 \leq i + 1 \leq |s|) \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = r[j])$$

$$0 \leq i \leq |s| \wedge i < |s| \rightarrow 0 \leq i < |s| \rightarrow 0 \leq i + 1 \leq |s|$$

$$(\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = r[j]) \rightarrow (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = r[j])_\square$$

$$(I \wedge \neg B) \rightarrow Qc : \neg i < |s| \equiv i \geq |s|$$

$$0 \leq i \leq |s| \wedge i \geq |s| \equiv i = |s|$$

$$|s| = |r| \wedge i = |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = r[j]) \rightarrow (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] = r[j])_\square$$

c)

$$f_v(i) = |s| - i$$

$$4\text{to paso: } \{I \wedge B \wedge v_0 = f_v\}S\{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$S_0 \equiv r[i] := s[i]; S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; S_1;$$

$$Wp(S, I) \stackrel{\text{Axioma 3}}{=} Wp(S_0, Wp(S_1, f_v < v_0))$$

$$Wp(S_1, f_v < v_0) \stackrel{\text{Axioma 1}}{=} |s| - i - 1 < v_0$$

$$Wp(S_0, Wp(S_1, f_v < v_0)) \stackrel{\text{Axioma 1}}{=} 0 \leq i < |s| \wedge 0 \leq i < |r| \wedge |s| - i - 1 < v_0$$

$$(|s| = |r| \wedge 0 \leq i \leq |s|) \rightarrow \mathbf{0 \leq i < |r| \wedge 0 \leq i < |s|}$$

$$v_0 = |s| - i \rightarrow \mathbf{|s| - i - 1 < v_0 \equiv -1 < 0} \square$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |s| \wedge |s| - i \leq 0 \rightarrow i = |s|$$

$$i = |s| \rightarrow \textcolor{blue}{i \geq |s| \equiv \neg B} \square$$

8. a)

$$Pc \equiv s = S_0 \wedge 0 \leq d < |S_0| \wedge i = d$$

$$Qc \equiv |s| = |S_0| \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = S_0[j]) \wedge (\forall j : \mathbb{Z})(d \leq j < |s| \rightarrow_L s[j] = e))$$

b)

$$\begin{aligned} I \equiv & (|s| = |S_0| \wedge 0 \leq d \leq i \leq |S_0|) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = S_0[j])) \\ & \wedge (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[j] = e) \end{aligned}$$

$$Pc \rightarrow I : 0 \leq d < |S_0| \wedge i = d \rightarrow 0 \leq d \leq i \leq |S_0|$$

$$\begin{aligned} & s = S_0 \wedge i = d \rightarrow ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = \underset{s=S_0}{\overset{\uparrow}{s}}[j])) \\ & \wedge (\forall j : \mathbb{Z})(d \leq j < \underset{i=d}{\overset{\uparrow}{i}} \rightarrow_L s[j] = e) = \text{True} \wedge \text{True}_\square \end{aligned}$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$\begin{aligned} & S_0 \equiv s[i] := e; S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; S_1; \\ & Wp(S, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(S_0, Wp(S_1, I)) \end{aligned}$$

$$\begin{aligned} & Wp(S_1, I) \stackrel{\text{Axioma 1}}{\equiv} (|s| = |S_0| \wedge 0 \leq d \leq i + 1 \leq |S_0|) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = S_0[j]) \wedge (\forall j : \mathbb{Z})(d \leq j < i + 1 \rightarrow_L s[j] = e)) \\ & Wp(S_0, Wp(S_1, I)) \stackrel{\text{Axioma 1}}{\equiv} (|SetAt(s, i, e)| = |S_0| \wedge 0 \leq d \leq i + 1 \leq |S_0|) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L \underset{\text{SetAt}(s, i, e)[i]=e}{\text{At}}(s, i, e)[j] = S_0[j]) \wedge (\forall j : \mathbb{Z})(d \leq j < i + 1 \rightarrow_L SetAt(s, i, e)[j] = e)) \\ & \equiv (|s| = |S_0| \wedge 0 \leq d \leq i + 1 \leq |S_0|) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = S_0[j]) \wedge ((\forall j : \mathbb{Z})(d \leq j < i + 1 \rightarrow_L ((i = j \wedge e = e) \vee (i \neq j \wedge SetAt(s, i, e)[j] = e))) \\ & \quad \underset{\text{SetAt}(s, i, e)[i]=e}{\text{SetAt}(s, i, e)[i]=e}) \\ & \equiv (|s| = |S_0| \wedge 0 \leq d \leq i + 1 \leq |S_0|) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = S_0[j]) \wedge (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[j] = e)) \end{aligned}$$

$$0 \leq d \leq i \leq |S_0| \wedge i < |s| \rightarrow 0 \leq d \leq i + 1 \leq |S_0|$$

$$((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = S_0[j]) \wedge (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[j] = e)) \rightarrow ((\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = S_0[j]) \wedge (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[j] = e)) \square$$

$$(I \wedge \neg B) \rightarrow Qc : \neg i < |s| \equiv i \geq |s|$$

$$0 \leq d \leq i \leq |S_0| \wedge i \geq |s| \wedge |s| = |S_0| \rightarrow i = |S_0|$$

$$0 \leq d \leq i \wedge i = |S_0| \rightarrow 0 \leq d \leq i \leq |S_0|$$

$$(\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[j] = e) \wedge i = |s| \rightarrow (\forall j : \mathbb{Z})(d \leq j < \underset{i=|s|}{\overset{\uparrow}{|s|}} \rightarrow_L s[j] = e) \square$$

c)

$$f_v(i) = |s| - i$$

$$4\text{to}\text{paso: } \{I \wedge B \wedge v_0 = f_v\}S\{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$S_0 \equiv s[i] := e; S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; S_1;$$

$$Wp(S, I) \xrightarrow{\text{Axioma 3}} Wp(S_0, Wp(S_1, f_v < v_0))$$

$$\begin{aligned} Wp(S_1, f_v < v_0) &\xrightarrow{\text{Axioma 1}} |s| - i - 1 < v_0 \\ Wp(S_0, Wp(S_1, f_v < v_0)) &\xrightarrow{\text{Axioma 1}} 0 \leq i < |s| \wedge |s| - i - 1 < v_0 \end{aligned}$$

$$\begin{aligned} 0 \leq i \leq |s| \wedge i < |s| &\equiv 0 \leq i < |s| \\ v_0 = |s| - i \rightarrow |s| - i - 1 < v_0 &\equiv -1 < 0 \quad \square \end{aligned}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |s| \wedge |s| - i \leq 0 \rightarrow i = |s|$$

$$i = |s| \rightarrow \textcolor{blue}{i \geq |s|} \equiv \neg B \quad \square$$

$$9. \quad a) \quad I \equiv |s| \bmod 2 = 0 \wedge |s|/2 - 1 \leq i \leq |s| - 1 \wedge \text{suma} = \sum_{j=0}^{|s|-2-i} s[j]$$

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

c)

$$Pc \rightarrow I : i = |s| - 1 \rightarrow |s|/2 - 1 \leq i \leq |s| - 1 \wedge |s| - 2 - i = -1 \rightarrow \sum_{j=0}^{-1} s[j] = 0 = \text{suma}_{\square}$$

$\uparrow$   
 $i = |s| - 1 \text{ y } (|s| - 2) - i$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$S_0 \equiv \text{suma} := \text{suma} + s[|s| - 1 - i]; \quad S_1 \equiv i := i - 1; \Rightarrow S \equiv S_0; \quad S_1;$$

$$Wp(S, I) \xrightarrow{\text{Axioma 3}} Wp(S_0, Wp(S_1, I))$$

$$\begin{aligned} Wp(S_1, I) &\xrightarrow{\text{Axioma 1}} |s| \bmod 2 = 0 \wedge |s|/2 - 1 \leq i - 1 \leq |s| - 1 \wedge \text{suma} = \sum_{j=0}^{|s|-1-i} s[j] \\ \Rightarrow Wp(S_0, Wp(S_1, I)) &\xrightarrow{\text{Axioma 1}} |s| \bmod 2 = 0 \wedge |s|/2 - 1 \leq i - 1 \leq |s| - 1 \wedge_L \text{suma} + s[|s| - 1 - i] = \\ &\sum_{j=0}^{|s|-1-i} s[j] \end{aligned}$$

$$|s|/2 - 1 \leq i \leq |s| - 1 \wedge i \geq |s|/2 \rightarrow |s|/2 \leq i \leq |s| - 1 \rightarrow |s|/2 - 1 \leq i - 1 \leq |s| - 1$$

$$\text{suma} = \sum_{j=0}^{|s|-2-i} s[j] \rightarrow \text{suma} + s[|s| - 1 - i] = \sum_{j=0}^{(|s|-2-i)+1} s[j] = \sum_{j=0}^{|s|-2-i} s[j] + s[|s| - 1 - i] \quad \square$$

$$(I \wedge \neg B) \rightarrow Qc : \neg(i \geq |s|/2) \equiv i < |s|/2$$

$$|s|/2 - 1 \leq i \leq |s| - 1 \wedge i < |s|/2 \equiv i = |s|/2 - 1$$

$$|s| \bmod 2 = 0 \wedge i = |s|/2 - 1 \wedge_L \text{suma} = \sum_{j=0}^{|s|-2-i} s[j] \rightarrow \text{suma} = \sum_{j=0}^{|s|/2-1} s[j] \wedge |s| \bmod 2 = 0 \quad \square$$

$\uparrow$   
 $i = |s|/2 - 1 \text{ y } |s| - 2 - i - (|s|/2 - 1) = |s|/2 - 1$

$$4\text{to paso: } \{I \wedge B \wedge v_0 = f_v\}S\{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$S_0 \equiv \text{suma} := \text{suma} + s[|s| - 1 - i]; \quad S_1 \equiv i := i - 1; \Rightarrow S \equiv S_0; \quad S_1;$$

$$Wp(S, f_v < v_0) \xrightarrow{\text{Axioma 3}} Wp(S_0, Wp(S_1, f_v < v_0))$$

$$\begin{aligned} Wp(S_1, f_v < v_0) &\xrightarrow{\text{Axioma 1}} |s| - i - 1 < v_0 \\ \Rightarrow Wp(S_0, Wp(S_1, f_v < v_0)) &\xrightarrow{\text{Axioma 1}} |s| - i - 1 < v_0 \end{aligned}$$

$$|s| - i = v_0 \rightarrow |s| - i - 1 < v_0 \equiv |s| - i - 1 < |s| - i \equiv -1 < 0 \quad \square$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : |s|/2 - 1 \leq i \leq |s| \wedge i - (|s|/2 - 1) \leq 0 \rightarrow i = |s|/2 - 1$$

$$i = |s|/2 - 1 \rightarrow \textcolor{blue}{i < |s|/2} \equiv \neg B \square$$

10. Cambiamos s por l, para que no haya conflicto con la demostración

a)

---

```

1 i := 0;
2 while i < |l| do
3 if l[i] = a then
4 l[i] = b;
5 else
6 skip;
7 endif
8 i := i+1;
9 endwhile

```

---

b)

$$Pc : l = L_0 \wedge i = 0$$

$$Qc : |l| = |L_0| \wedge_L$$

$$(\forall j : \mathbb{Z})(0 \leq j < |l| \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge \\ (\forall j : \mathbb{Z})(0 \leq j < |l| \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j])$$

c)

$$I \equiv (|l| = |L_0| \wedge 0 \leq i \leq |l|) \wedge_L$$

$$(\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge$$

$$(\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j]) \wedge$$

$$(\forall j : \mathbb{Z})(i \leq j < |l| \rightarrow_L l[j] = L_0[j]) \text{(este predicado luego no aparece por un tema de espacio pero la demostración sale igual agregandolo)}$$

$$Pc \rightarrow I : l = L_0 \rightarrow |l| = |L_0|$$

$$i = 0 \rightarrow 0 \leq i \leq |l| \wedge_L$$

$$(\forall j : \mathbb{Z})(0 \leq j < 0 \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge$$

$$(\forall j : \mathbb{Z})(0 \leq j < 0 \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j]) = \text{True} \wedge \text{True}_\square$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$C \equiv l[i] = a; S_0 \equiv l[i] = b; S_1 \equiv \text{skip}; S_2 \equiv i := i + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2;$$

$$Wp(S, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, I))$$

$$\stackrel{\text{Axioma 4}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, I))))$$

$$\stackrel{\text{Axioma 2}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_2, I)))$$

$$def(C) \equiv def(l[i] = a) \equiv 0 \leq i < |l|$$

$$\neg C \equiv l[i] \neq a$$

$$Wp(S_2, I) \stackrel{Axioma\ 1}{\equiv} (|l| = |L_0| \wedge 0 \leq i + 1 \leq |l|) \wedge_L$$

$$(\forall j : \mathbb{Z})(0 \leq j < i + 1 \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge$$

$$(\forall j : \mathbb{Z})(0 \leq j < i + 1 \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j])$$

$$Wp(S_0, Wp(S_2, I)) \stackrel{Axioma\ 1}{\equiv} (|setAt(l, i, b)| = |L_0| \wedge 0 \leq i + 1 \leq |setAt(l, i, b)|) \wedge_L$$

$$(\forall j : \mathbb{Z})(0 \leq j < i + 1 \wedge_L L_0[j] = a \rightarrow_L setAt(l, i, b)[j] = b) \wedge$$

$$(\forall j : \mathbb{Z})(0 \leq j < i + 1 \wedge_L L_0[j] \neq a \rightarrow_L setAt(l, i, b)[j] = L_0[j])$$

$$\equiv (|l| = |L_0| \wedge 0 \leq i + 1 \leq |l|) \wedge_L$$

$$(\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge (L_0[i] = a \rightarrow_l l[i] = b) \wedge$$

$$(\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j]) \wedge (L_0[i] \neq a \rightarrow_l l[i] = L_0[i])$$

$$\Rightarrow Wp(S, I) \equiv 0 \leq i < |L_0| \wedge_L$$

$$((l[i] = a \wedge (|l| = |L_0| \wedge 0 \leq i + 1 \leq |l|)) \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge (L_0[i] = a \rightarrow_l l[i] = b) \wedge (\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j]) \wedge (L_0[i] \neq a \rightarrow_l l[i] = L_0[i])) \vee$$

$$(l[i] \neq a \wedge (|l| = |L_0| \wedge 0 \leq i + 1 \leq |l|)) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < i + 1 \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge (\forall j : \mathbb{Z})(0 \leq j < i + 1 \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j]))$$

$$\equiv 0 \leq i < |L_0| \wedge |l| = |L_0| \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge (\forall j : \mathbb{Z})(0 \leq j <$$

$$0 \leq i \leq |l| \wedge i < |l| \rightarrow 0 \leq i < |l|_{\square}$$

$$(I \wedge \neg B) \rightarrow Qc : \neg B \equiv i \geq |l|$$

$$0 \leq i \leq |l| \wedge i \geq |l| \equiv i = |l|$$

$$|l| = |L_0| \wedge i = |l| \wedge_L$$

$$(\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge$$

$$(\forall j : \mathbb{Z})(0 \leq j < i \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j]) \rightarrow$$

$$(\forall j : \mathbb{Z})(0 \leq j < |l| \wedge_L L_0[j] = a \rightarrow_L l[j] = b) \wedge$$

$$(\forall j : \mathbb{Z})(0 \leq j < |l| \wedge_L L_0[j] \neq a \rightarrow_L l[j] = L_0[j])$$

d)

$$f_v(i) = |l| - i$$

$$4\text{to paso } \{I \wedge B \wedge v_0 = f_v\}S\{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$C \equiv L_0[i] = a; S_0 \equiv l[i] = b; S_1 \equiv skip; S_2 \equiv i := i + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2;$$

$$Wp(S, f_v < v_0) \stackrel{Axioma\ 3}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, f_v < v_0))$$

$$\stackrel{Axioma\ 4}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, f_v < v_0))))$$

$$\stackrel{Axioma\ 2}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_2, f_v < v_0))))$$

$$def(C) \equiv def(L_0[i] = a) \equiv 0 \leq i < |L_0|$$

$$\begin{aligned}
Wp(S_2, f_v < v_0) &\stackrel{\text{Axioma 1}}{\equiv} |l| - i - 1 < v_0 \\
Wp(S_0, Wp(S_2, f_v < v_0)) &\stackrel{\text{Axioma 1}}{\equiv} |l| - i - 1 < v_0 \\
Wp(S, f_v < v_0) &\equiv 0 \leq i < |l| \wedge |l| - i - 1 < v_0
\end{aligned}$$

$$0 \leq i \leq |l| \wedge |l| - i = v_0 \rightarrow 0 \leq i < |l| \wedge |l| - i - 1 < |l| - i_{\square}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |l| \wedge |l| - i \leq 0 \rightarrow i = |s|$$

$$i = |s| \rightarrow \textcolor{blue}{i} \geq |s| \equiv \neg B_{\square}$$

11.

$$Pc \equiv i = |s| - 1 \wedge j = -1$$

$$Qc \equiv (j = -1 \rightarrow (\forall k : \mathbb{Z})(0 \leq k < |s| \rightarrow_L s[k] \neq e)) \wedge (j \neq -1 \rightarrow (0 \leq k < |s| \wedge_L s[k] = e))$$

$$I \equiv -1 \leq i < |s| \wedge_L (j = -1 \rightarrow (\forall k : \mathbb{Z})(i < k < |s| \rightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i < j < |s| \wedge_L s[j] = e))$$

$$Pre \rightarrow Wp(i := |s| - 1; j := 1, Pc) \stackrel{\text{Axioma 3}}{\equiv} Wp(i := |s| - 1, Wp(j := 1, Pc)) \stackrel{\text{Axioma 1}}{\equiv} Wp(i := |s| - 1, (i = |s| - 1 \wedge -1 = -1)) \stackrel{\text{Axioma 1}}{\equiv} |s| - 1 = |s| - 1 \wedge -1 = -1 \equiv True_{\square}$$

$$Qc \rightarrow Wp(r := j, Post) \stackrel{\text{Axioma 1}}{\equiv} (j = -1 \rightarrow (\forall k : \mathbb{Z})(0 \leq k < |s| \rightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (0 \leq j < |s| \wedge_L s[j] = e))$$

$$\Rightarrow (Qc \rightarrow Wp(r := j, Post)) \equiv Qc \rightarrow Qc \equiv True_{\square}$$

$$Pc \rightarrow I : i = |s| - 1 \wedge j = -1 \rightarrow -1 \leq i < |s| \wedge_L (j = -1 \rightarrow (\forall k : \mathbb{Z})(i < k < |s| \rightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i < j < |s| \wedge_L s[j] = e))_{\square}$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$\begin{aligned}
C &\equiv s[i] = e; S_0 \equiv j := i; S_1 \equiv \text{skip}; S_2 \equiv i := i - 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2; \\
Wp(S, I) &\stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, I)) \\
&\stackrel{\text{Axioma 4}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, I)))) \\
&\stackrel{\text{Axioma 2}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_2, I)))
\end{aligned}$$

$$\text{def}(C) \equiv \text{def}(s[i] = e) \equiv 0 \leq i < |s|$$

$$\neg C \equiv s[i] \neq e$$

$$Wp(S_2, I) \stackrel{\text{Axioma 1}}{\equiv} -1 \leq i - 1 < |s| \wedge_L (j = -1 \rightarrow (\forall k : \mathbb{Z})(i - 1 < k < |s| \rightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i - 1 < j < |s| \wedge_L s[j] = e))$$

$$Wp(S_1, Wp(S_2, I)) \stackrel{\text{Axioma 1}}{\equiv} -1 \leq i - 1 < |s| \wedge_L (i = -1 \rightarrow (\forall k : \mathbb{Z})(i - 1 < k < |s| \rightarrow_L s[j] \neq e)) \wedge (i \neq -1 \rightarrow (i - 1 < i < |s| \wedge_L s[i] = e)) \equiv -1 \leq i - 1 < |s| \wedge s[i] = e$$

$$\begin{aligned}
\Rightarrow Wp(S, I) &\equiv 0 \leq i < |s| \wedge_L ((s[i] = e \wedge -1 \leq i - 1 < |s| \wedge s[i] = e) \vee (s[i] \neq e \wedge -1 \leq i - 1 < |s| \wedge (j = -1 \rightarrow (\forall k : \mathbb{Z})(i - 1 < k < |s| \rightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i - 1 < j < |s| \wedge_L s[j] = e))) \equiv 0 \leq i < |s| \wedge_L (s[i] = e \vee (j = -1 \rightarrow (\forall k : \mathbb{Z})(i \leq k < |s| \wedge s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i \leq j < |s| \wedge_L s[j] = e))) \\
&\equiv 0 \leq i < |s| \wedge_L (s[i] = e \vee (j = -1 \rightarrow (\forall k : \mathbb{Z})(i < k < |s| \rightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i < j < |s| \wedge_L s[j] = e)))
\end{aligned}$$

$$\begin{aligned}
& -1 \leq i < |s| \wedge i \geq 0 \equiv 0 \leq i < |s| \\
& (j = -1 \rightarrow (\forall k : \mathbb{Z})(i < k < |s| \longrightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i < j < |s| \wedge_L s[j] = e)) \rightarrow s[i] \vee (j = -1 \rightarrow (\forall k : \mathbb{Z})(i < k < |s| \longrightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i < j < |s| \wedge_L s[j] = e)) \square
\end{aligned}$$

$$(I \wedge \neg B) \rightarrow Qc : \neg i \geq 0 \equiv i < 0$$

$$\begin{aligned}
& -1 \leq i < |s| \wedge i < 0 \equiv i = -1 \\
& i = -1 \wedge_L (j = -1 \rightarrow (\forall k : \mathbb{Z})(i < k < |s| \longrightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (i < j < |s| \wedge_L s[j] = e)) \rightarrow \\
& (j = -1 \rightarrow (\forall k : \mathbb{Z})(0 \leq k < |s| \longrightarrow_L s[j] \neq e)) \wedge (j \neq -1 \rightarrow (0 \leq j < |s| \wedge_L s[j] = e)) \square
\end{aligned}$$

4to paso  $f_v(i) = i + 1$

$$\begin{aligned}
C &\equiv s[i] = e; S_0 \equiv j := i; S_1 \equiv \text{skip}; S_2 \equiv i := i - 1; S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2; \\
Wp(S, f_v < v_0) &\stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, f_v < v_0)) \\
&\stackrel{\text{Axioma 4}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, f_v < v_0)))) \\
&\stackrel{\text{Axioma 2}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_2, f_v < v_0)))
\end{aligned}$$

$$\text{def}(C) \equiv \text{def}(s[i] = e) \equiv 0 \leq i < |s|$$

$$\neg C \equiv s[i] \neq e$$

$$\begin{aligned}
Wp(S_2, f_v < v_0) &\stackrel{\text{Axioma 1}}{\equiv} i - 2 < v_0 \\
Wp(S_0, Wp(S_2, f_v < v_0)) &\stackrel{\text{Axioma 1}}{\equiv} i - 2 < v_0
\end{aligned}$$

$$\begin{aligned}
& \Rightarrow Wp(S, f_v < v_0) \equiv 0 \leq i < |s| \wedge_L ((s[i] = e \wedge i - 2 < v_0) \vee (s[i] \neq e \wedge i - 2 < v_0)) \equiv 0 \leq i < |s| \wedge i - 2 < v_0 \\
& -1 \leq i < |s| \wedge i \geq 0 \equiv 0 \leq i < |s| \\
& i - 1 = v_0 \rightarrow i - 2 < i - 1 \equiv -1 < 0 \equiv \text{True} \square
\end{aligned}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : -1 \leq i < |s| \wedge i + 1 \leq 0 \rightarrow i = -1$$

$$i = -1 \rightarrow \textcolor{blue}{i < 0 \equiv \neg B} \square$$

12.

$$Pc \equiv i = 0 \wedge j = -1$$

$$\begin{aligned}
Qc &\equiv j \neq -1 \leftrightarrow (\exists k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = e) \\
&\quad \boxed{\exists k : \mathbb{Z}} 0 \leq i \leq |s| \wedge (j = -1 \leftrightarrow (\exists k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] = e))
\end{aligned}$$

$$\begin{aligned}
Pre \rightarrow Wp(i := 0; j := -1, Pc) &\stackrel{\text{Axioma 3}}{\equiv} Wp(i := 0, Wp(j := -1, Pc)) \stackrel{\text{Axioma 1}}{\equiv} Wp(i := 0, i = 0 \wedge -1 = \\
&-1) \stackrel{\text{Axioma 1}}{\equiv} 0 = 0 \wedge -1 = -1 \equiv \text{True} \square
\end{aligned}$$

$$Qc \rightarrow Wp(\text{if } j \neq -1 \text{ then } r := \text{True} \text{ else } r := \text{False} \text{ fi, Post})$$

$$\stackrel{\text{Axioma 4}}{\equiv} \text{def}(j \neq -1) \wedge_L ((j \neq -1 \wedge Wp(r := \text{True}, \text{Post})) \vee (j = -1 \wedge Wp(r := \text{False}, \text{Post})))$$

$$\stackrel{\text{Axioma 1}}{\equiv} (j \neq -1 \wedge (\text{True} = \text{True} \leftrightarrow (\exists k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = e))) \vee (j = -1 \wedge (\text{False} =$$

$$\text{True} \leftrightarrow (\exists k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = e)))$$

$$\equiv (j \neq -1 \wedge (\exists k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = e)) \vee \boxed{j = -1}$$

$$\equiv j \neq -1 \leftrightarrow (\exists k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = e)$$

$$Pc \rightarrow I : i = 0 \rightarrow 0 \leq i \leq |s|$$

$$i = 0 \wedge j = -1 \rightarrow (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = e))$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$C \equiv s[i] = e; S_0 \equiv j := i; S_1 \equiv skip; S_2 \equiv i := i + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2;$$

$$Wp(S, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, I))$$

$$\stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, I))))$$

$$\stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_2, I)))$$

$$def(C) \equiv def(s[i] = e) \equiv 0 \leq i < |s|$$

$$\neg C \equiv s[i] \neq e$$

$$Wp(S_2, I) \stackrel{\text{Axioma 1}}{\equiv} 0 \leq i + 1 \leq |s| \wedge (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] = e))$$

$$Wp(S_0, Wp(S_2, I)) \stackrel{\text{Axioma 1}}{\equiv} 0 \leq i + 1 \leq |s| \wedge (i = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] = e))$$

$$\equiv 0 \leq i + 1 \leq |s| \wedge_L (\exists k : \mathbb{Z})(0 \leq k < i + 1 \wedge s[k] = e)$$

$$\Rightarrow Wp(S, I) \equiv 0 \leq i < |s| \wedge_L ((s[i] = e \wedge 0 \leq i + 1 \leq |s| \wedge_L (\exists k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] = e)) \vee (s[i] \neq e \wedge 0 \leq i + 1 \leq |s| \wedge_L (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] = e))))$$

$$\equiv 0 \leq i < |s| \wedge_L ((s[i] = e \wedge (\exists k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] = e)) \vee (s[i] \neq e \wedge (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] = e))))$$

$$\equiv 0 \leq i < |s| \wedge_L (s[i] = e \vee (s[i] \neq e \wedge (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] = e))))$$

$$\equiv 0 \leq i < |s| \wedge_L (s[i] = e \vee (s[i] \neq e \wedge ((j \neq -1 \wedge s[i] = e) \vee (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] = e)))))$$

$$\equiv 0 \leq i < |s| \wedge_L (s[i] = e \vee (s[i] \neq e \wedge (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] = e))))$$

$$\equiv 0 \leq i < |s| \wedge_L (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] = e))$$

$$0 \leq i \leq |s| \wedge i < |s| \equiv 0 \leq i < |s|_{\square}$$

$$(I \wedge \neg B) \rightarrow Qc : \neg i < |s| \equiv i \geq |s|$$

$$0 \leq i \leq |s| \wedge i \geq |s| \equiv i = |s|$$

$$i = |s| \wedge (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] = e)) \rightarrow \wedge (j = -1 \leftrightarrow (\#k : \mathbb{Z})(0 \leq k < |s| \wedge_L s[k] = e))_{\square}$$

4to paso:  $f_v(i) = |s| - i$

$$C \equiv s[i] = e; S_0 \equiv j := i; S_1 \equiv skip; S_2 \equiv i := i + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2;$$

$$Wp(S, f_v < v_0) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, f_v < v_0))$$

$$\stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, f_v < v_0))))$$

$$\stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_2, f_v < v_0)))$$

$$def(C) \equiv 0 \leq i < |s|$$

$$\neg C \equiv s[i] \neq e$$

$$Wp(S_2, f_v < v_0) \stackrel{\text{Axioma 1}}{\equiv} |s| - i - 1 < v_0$$

$$Wp(S_0, Wp(S_2, f_v < v_0)) \stackrel{\text{Axioma 1}}{\equiv} |s| - i - 1 < v_0$$

$$\Rightarrow Wp(S, f_v < v_0) \equiv 0 \leq i < |s| \wedge_L ((s[i] = e \wedge |s| - i - 1 < v_0) \vee (s[i] \neq e \wedge |s| - i - 1 < v_0)) \\ \equiv 0 \leq i < |s| \wedge |s| - i - 1 < v_0$$

$$0 \leq i \leq |s| \wedge i < |s| \rightarrow 0 \leq i < |s| \\ v_0 = |s| - i \rightarrow |s| - i - 1 < |s| - i \equiv -1 < 0 \equiv True_{\square}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |s| \wedge |s| - i \leq 0 \rightarrow i = |s|$$

$$i = |s| \rightarrow \textcolor{blue}{i \geq |s|} \equiv \neg B_{\square}$$

13.

$$Pc \equiv i = 0 \wedge j = |s| - 1 \wedge r = True$$

$$Qc \equiv r = True \leftrightarrow (\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L s[i] = s[|s| - i - 1])$$

$$I \equiv (0 \leq i \leq |s| \wedge -1 \leq j < |s| \wedge j = |s| - i - 1) \wedge_L (r = True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1]))$$

$$Pre \rightarrow Wp(i := 0; j := |s| - 1; r := True, Pc) \stackrel{\text{Axioma 3}}{\equiv} Wp(i := 0, Wp(j := |s| - 1, Wp(r := True, Pc))) \equiv \\ True_{\square}$$

$$Qc \rightarrow Post_{\square}$$

$$Pc \rightarrow I : i = 0 \wedge r = True \rightarrow 0 \leq i \leq |s| \wedge_L (r = True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1]))$$

$$j = |s| - 1 \rightarrow -1 \leq j < |s|_{\square}$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$C \equiv s[i] \neq s[j]; S_0 \equiv r := False; S_1 \equiv skip; S_2 \equiv i := i + 1; S_3 \equiv j := j - 1; \Rightarrow S \equiv \\ \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2; S_3; \\ Wp(S, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, Wp(S_3, I))) \\ \stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, Wp(S_3, I)))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, Wp(S_3, I))))) \\ \stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, Wp(S_3, I)))) \vee (\neg C \wedge Wp(S_2, Wp(S_3, I))))$$

$$def(C) \equiv def(s[i] \neq s[j]) \equiv 0 \leq i < |s| \wedge 0 \leq j < |s|$$

$$\neg C \equiv s[i] = s[j]$$

$$Wp(S_3, I) \stackrel{\text{Axioma 1}}{\equiv} (0 \leq i \leq |s| \wedge -1 \leq j - 1 < |s|) \wedge_L (r = True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1]))$$

$$Wp(S_2, Wp(S_3, I)) \stackrel{\text{Axioma 1}}{\equiv} (0 \leq i + 1 \leq |s| \wedge -1 \leq j - 1 < |s| \wedge j - 1 = |s| - i - 2) \wedge_L (r = True \leftrightarrow \\ (\forall k : \mathbb{Z})(0 \leq k < i + 1 \longrightarrow_L s[k] = s[|s| - k - 1]))$$

$$\equiv (0 \leq i + 1 \leq |s| \wedge -1 \leq j - 1 < |s| \wedge j = |s| - i - 1) \wedge_L (r = True \leftrightarrow ((\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L \\ s[k] = s[|s| - k - 1]) \wedge s[i] = s[|s| - i - 1]))$$

$$Wp(S_0, Wp(S_2, Wp(S_3, I))) \stackrel{\text{Axioma 1}}{\equiv} (0 \leq i + 1 \leq |s| \wedge -1 \leq j - 1 < |s| \wedge j = |s| - i - 1) \wedge_L (False = \\ True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < i + 1 \longrightarrow_L s[k] = s[|s| - k - 1]))$$

$$\equiv (0 \leq i + 1 \leq |s| \wedge -1 \leq j - 1 < |s| \wedge j = |s| - i - 1) \wedge_L (\exists k : \mathbb{Z})(0 \leq k < i + 1 \wedge_L s[k] \neq s[|s| - k - 1]) \\ \equiv (0 \leq i + 1 \leq |s| \wedge -1 \leq j - 1 < |s| \wedge j = |s| - i - 1) \wedge_L ((\exists k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] \neq$$

$$s[|s| - k - 1]) \vee s[i] \neq s[|s| - i - 1])$$

$$\begin{aligned}
& \Rightarrow Wp(S, I) \equiv (0 \leq i < |s| \wedge 0 \leq j < |s|) \wedge_L \\
& ((s[i] \neq s[j] \wedge ((0 \leq i + 1 \leq |s| \wedge -1 \leq j - 1 < |s| \wedge j = |s| - i - 1) \wedge_L ((\exists k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] \neq s[|s| - k - 1]) \vee s[i] \neq s[|s| - i - 1]))) \vee \\
& (s[i] = s[j] \wedge ((0 \leq i + 1 \leq |s| \wedge -1 \leq j - 1 < |s| \wedge j = |s| - i - 1) \wedge_L (r = True \leftrightarrow ((\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1]) \wedge s[i] = s[|s| - i - 1])))) \\
& \equiv (0 \leq i < |s| \wedge 0 \leq j < |s| \wedge j = |s| - i - 1) \wedge_L \\
& ((s[i] \neq s[j] \wedge ((\exists k : \mathbb{Z})(0 \leq k < i \wedge_L s[k] \neq s[|s| - k - 1]) \vee s[i] \neq s[|s| - i - 1])) \vee \\
& (s[i] = s[j] \wedge (r = True \leftrightarrow ((\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1]) \wedge s[i] = s[|s| - i - 1]))) \\
& \equiv (0 \leq i < |s| \wedge 0 \leq j < |s| \wedge j = |s| - i - 1) \wedge_L \\
& (s[i] \neq s[j] \vee \\
& (s[i] = s[j] \wedge (r = True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1])))) \\
& \equiv (0 \leq i < |s| \wedge 0 \leq j < |s| \wedge j = |s| - i - 1) \wedge_L (r = True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1]))
\end{aligned}$$

$$\begin{aligned}
& 0 \leq i \leq |s| \wedge i < |s| \equiv 0 \leq i < |s| \\
& j = |s| - i - 1 \equiv i = |s| - j - 1 \\
& 0 \leq i < |s| \wedge i = |s| - j - 1 \rightarrow 0 \leq |s| - j - 1 < |s| \equiv 1 - |s| \leq -j < -1 \equiv -1 < j \leq |s| - 1 \equiv 0 \leq j < |s|_{\square}
\end{aligned}$$

$$(I \wedge \neg B) \rightarrow Qc : \neg i < |s| \equiv i \geq |s|$$

$$\begin{aligned}
& 0 \leq i \leq |s| \wedge i \geq |s| \equiv i = |s| \\
& i = |s| \wedge_L (r = True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = s[|s| - k - 1])) \rightarrow \\
& (r = True \leftrightarrow (\forall k : \mathbb{Z})(0 \leq k < |s| \longrightarrow_L s[k] = s[|s| - k - 1]))
\end{aligned}$$

$$4\text{to paso } f_v(i) = |s| - i$$

$$\begin{aligned}
C & \equiv s[i] \neq s[j]; S_0 \equiv r := False; S_1 \equiv skip; S_2 \equiv i := i + 1; S_3 \equiv j := j - 1; \Rightarrow S \equiv \\
& \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2; S_3; \\
Wp(S, f_v < v_0) & \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, Wp(S_3, f_v < v_0))) \\
& \stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, Wp(S_3, f_v < v_0)))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, Wp(S_3, f_v < v_0))))) \\
& \stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, Wp(S_3, f_v < v_0)))) \vee (\neg C \wedge Wp(S_2, Wp(S_3, f_v < v_0))))
\end{aligned}$$

$$def(C) \equiv def(s[i] \neq s[j]) \equiv 0 \leq i < |s| \wedge 0 \leq j < |s|$$

$$\neg C \equiv s[i] = s[j]$$

$$Wp(S_3, f_v < v_0) \stackrel{\text{Axioma 1}}{\equiv} f_v < v_0$$

$$Wp(S_2, Wp(S_3, f_v < v_0)) \stackrel{\text{Axioma 1}}{\equiv} |s| - i - 1 < v_0$$

$$Wp(S_0, Wp(S_2, Wp(S_3, f_v < v_0))) \stackrel{\text{Axioma 1}}{\equiv} |s| - i - 1 < v_0$$

$$\Rightarrow Wp(S, f_v < v_0) \equiv (0 \leq i < |s| \wedge 0 \leq j < |s|) \wedge_L ((s[i] \neq s[j] \wedge |s| - i - 1 < v_0) \vee (s[i] =$$

$$\begin{aligned} & s[j] \wedge |s| - i - 1 < v_0) \\ & \equiv 0 \leq i < |s| \wedge 0 \leq j < |s| \wedge |s| - i - 1 < v_0 \end{aligned}$$

$$\begin{aligned} & 0 \leq i \leq |s| \wedge i < |s| \equiv 0 \leq i < |s| \\ & j = |s| - i - 1 \equiv i = |s| - j - 1 \\ & 0 \leq i < |s| \wedge i = |s| - j - 1 \rightarrow 0 \leq |s| - j - 1 < |s| \equiv 1 - |s| \leq -j < -1 \equiv -1 < j \leq |s| - 1 \equiv 0 \leq j < |s| \\ & v_0 = f_v = |s| - i \rightarrow |s| - i - 1 < v_0 \equiv -1 < 0 \equiv \text{True}_{\square} \end{aligned}$$

$$\begin{aligned} & (I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |s| \wedge |s| - i \leq 0 \rightarrow i = |s| \\ & i = |s| \rightarrow \textcolor{blue}{i \geq |s|} \equiv \neg B_{\square} \end{aligned}$$

14. Este problema lo voy a separar en dos, una parte para el primer ciclo, y otra para el segundo:

a) Primer ciclo:

$$\begin{aligned} & P_c \equiv |r| = |a| + |b| \wedge i = 0 \\ & Q_c \equiv |r| = |a| + |b| \wedge (\forall j : \mathbb{Z})(0 \leq j < |a| \rightarrow r[j] = a[j]) \\ & I \equiv (|r| = |a| + |b| \wedge 0 \leq i \leq |a|) \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow r[j] = a[j]) \end{aligned}$$

$$Pre \rightarrow P_{c_{\square}}$$

$Q_c \rightarrow Post$ : La Post es el  $P_c$  del segundo ciclo, y como son iguales es trivial.

$$P_c \rightarrow I : i = 0 \rightarrow \textcolor{blue}{0 \leq i \leq |a|} \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow r[j] = a[j])$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$\begin{aligned} & S_0 \equiv r := \text{setAt}(r, i, a[i]); S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; S_1; \\ & \Rightarrow Wp(S, I) \equiv Wp(S_0, Wp(S_1, I)) \end{aligned}$$

$$\begin{aligned} & Wp(S_1, I) \stackrel{\text{Axioma 1}}{\equiv} (|r| = |a| + |b| \wedge 0 \leq i + 1 \leq |a|) \wedge (\forall j : \mathbb{Z})(0 \leq j < i + 1 \rightarrow r[j] = a[j]) \\ & \Rightarrow Wp(S, I) \equiv Wp(S_0, Wp(S_1, I)) \stackrel{\text{Axioma 1}}{\equiv} \\ & (|\text{setAt}(r, i, a[i])| = |a| + |b| \wedge 0 \leq i + 1 \leq |a|) \wedge (\forall j : \mathbb{Z})(0 \leq j < i + 1 \rightarrow \text{setAt}(r, i, a[i])[j] = a[j]) \\ & \equiv (|r| = |a| + |b| \wedge 0 \leq i + 1 \leq |a|) \wedge ((\forall j : \mathbb{Z})(0 \leq j < i \rightarrow r[j] = a[j]) \wedge \text{setAt}(r, i, a[i])[i] = a[i]) \\ & \equiv (|r| = |a| + |b| \wedge 0 \leq i + 1 \leq |a|) \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow r[j] = a[j]) \end{aligned}$$

$$0 \leq i \leq |a| \wedge i < |a| \equiv 0 \leq i < a \rightarrow \textcolor{blue}{0 \leq i + 1 \leq a_{\square}}$$

$$(I \wedge \neg B) \rightarrow Q_c : \neg B \equiv i \geq |a|$$

$$0 \leq i \leq |a| \wedge i \geq |a| \equiv i = |a|$$

$$i = |a| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow r[j] = a[j]) \rightarrow (\forall j : \mathbb{Z})(0 \leq j < |a| \rightarrow r[j] = a[j])$$

$$f_v(i) = |a| - i$$

$$\text{4to paso } \{I \wedge B \wedge v_0 = f_v\} S\{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$\begin{aligned} S_0 &\equiv r := \text{setAt}(r, i, a[i]); \quad S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; \quad S_1; \\ \Rightarrow Wp(S, f_v < v_0) &\equiv Wp(S_0, Wp(S_1, f_v < v_0)) \end{aligned}$$

$$\begin{aligned} Wp(S_1, f_v < v_0) &\stackrel{\text{Axioma 1}}{\equiv} |a| - i - 1 < v_0 \\ \Rightarrow Wp(S, f_v < v_0) &\equiv Wp(S_0, Wp(S_1, f_v < v_0)) \stackrel{\text{Axioma 1}}{\equiv} 0 \leq i < |a| \wedge |a| - i - 1 < v_0 \end{aligned}$$

$$0 \leq i \leq |a| \wedge i < |a| \equiv \textcolor{blue}{0 \leq i < a}$$

$$v_0 = |a| - i \rightarrow \textcolor{blue}{|a| - i - 1 < v_0 \equiv -1 < 0} \square$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |a| \wedge |a| - i \leq 0 \rightarrow i = |a|$$

$$i = |a| \rightarrow \textcolor{blue}{i \geq |a| \equiv \neg B} \square$$

b) Segundo ciclo:

$$Pc \equiv i = 0 \wedge |r| = |a| + |b| \wedge (\forall j : \mathbb{Z})(0 \leq j < |a| \longrightarrow_L r[j] = a[j])$$

$$Qc \equiv |r| = |a| + |b| \wedge (\forall j : \mathbb{Z})(0 \leq j < |a| \longrightarrow_L r[j] = a[j]) \wedge (\forall j : \mathbb{Z})(0 \leq j < |b| \longrightarrow_L r[j + |a|] = b[j])$$

$$I \equiv (|r| = |a| + |b| \wedge 0 \leq i \leq |b|) \wedge_L$$

$$((\forall j : \mathbb{Z})(0 \leq j < |a| \longrightarrow_L r[j] = a[j]) \wedge (\forall j : \mathbb{Z})(0 \leq j < i \longrightarrow_L r[j + |a|] = b[j]))$$

*Pre*  $\rightarrow$  *Pc* : La *Pre* es el *Qc* del primer ciclo, y son iguales, por lo que es trivial

$$Qc \rightarrow Post_{\square}$$

$$Pc \rightarrow I : i = 0 \rightarrow \textcolor{blue}{0 \leq i \leq |b| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \longrightarrow_L r[j + |a|] = b[j])}$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$\begin{aligned} S_0 &\equiv r := \text{setAt}(r, i + |a|, b[i]); \quad S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; \quad S_1; \\ \Rightarrow Wp(S, I) &\equiv Wp(S_0, Wp(S_1, I)) \end{aligned}$$

$$\begin{aligned} Wp(S_1, I) &\stackrel{\text{Axioma 1}}{\equiv} (|r| = |a| + |b| \wedge 0 \leq i + 1 \leq |b|) \wedge_L \\ &((\forall j : \mathbb{Z})(0 \leq j < |a| \longrightarrow_L r[j] = a[j]) \wedge (\forall j : \mathbb{Z})(0 \leq j < i + 1 \longrightarrow_L r[j + |a|] = b[j])) \\ \Rightarrow Wp(S, I) &\equiv Wp(S_0, Wp(S_1, I)) \stackrel{\text{Axioma 1}}{\equiv} (|\text{setAt}(r, i + |a|, b[i])| = |a| + |b| \wedge 0 \leq i + 1 \leq |b|) \wedge_L \\ &((\forall j : \mathbb{Z})(0 \leq j < |a| \longrightarrow_L \text{setAt}(r, i + |a|, b[i])[j] = a[j]) \wedge (\forall j : \mathbb{Z})(0 \leq j < i + 1 \longrightarrow_L \\ &\quad \text{setAt}(r, i + |a|, b[i])[j + |a|] = b[j])) \\ &\equiv (|r| = |a| + |b| \wedge 0 \leq i + 1 \leq |b|) \wedge_L \\ &((\forall j : \mathbb{Z})(0 \leq j < |a| \longrightarrow_L r[j] = a[j]) \wedge (\forall j : \mathbb{Z})(0 \leq j < i \longrightarrow_L r[j + |a|] = b[j]) \wedge \\ &\quad \text{setAt}(r, i + |a|, b[i])[i + |a|] = b[i]) \\ &\equiv (|r| = |a| + |b| \wedge 0 \leq i + 1 \leq |b|) \wedge_L \\ &((\forall j : \mathbb{Z})(0 \leq j < |a| \longrightarrow_L r[j] = a[j]) \wedge (\forall j : \mathbb{Z})(0 \leq j < i \longrightarrow_L r[j + |a|] = b[j])) \end{aligned}$$

$$0 \leq i \leq |b| \wedge i < |b| \rightarrow 0 \leq i < |b| \rightarrow \textcolor{blue}{0 \leq i + 1 \leq |b|} \square$$

$$\begin{aligned}
(I \wedge \neg B) \rightarrow Qc : \neg B \equiv i \geq |b| \\
0 \leq i \leq |b| \wedge i \geq |b| \equiv i = |b| \\
i = |b| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \longrightarrow_L r[j + |a|] = b[j]) \rightarrow (\forall j : \mathbb{Z})(0 \leq j < |b| \longrightarrow_L r[j + |a|] = b[j])_{\square}
\end{aligned}$$

$$f_v(i) = |b| - i$$

$$\text{4to paso: } \{I \wedge B \wedge v_0 = f_v\} S \{f_v < v_0\} \equiv (I \wedge B \wedge v_0 = f_v) \rightarrow Wp(S, f_v < v_0)$$

$$\begin{aligned}
S_0 &\equiv setAt(r, i + |a|, b[i]); \quad S_1 \equiv i := i + 1; \Rightarrow S \equiv S_0; \quad S_1; \\
&\Rightarrow Wp(S, f_v < v_0) \equiv Wp(S_0, Wp(S_1, f_v < v_0))
\end{aligned}$$

$$\begin{aligned}
Wp(S_1, f_v < v_0) &\stackrel{\text{Axioma 1}}{\equiv} |b| - i - 1 < v_0 \\
\Rightarrow Wp(S, f_v < v_0) &\equiv Wp(S_0, Wp(S_1, f_v < v_0)) \stackrel{\text{Axioma 1}}{\equiv} 0 \leq i < |b| \wedge |b| - i - 1 < v_0
\end{aligned}$$

$$\begin{aligned}
0 \leq i \leq |b| \wedge i < |b| &\equiv 0 \leq i < |b| \\
v_0 = |b| - i \rightarrow |b| - i - 1 < v_0 &\equiv -1 < 0_{\square}
\end{aligned}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |b| \wedge |b| - i \leq 0 \rightarrow i = |b|$$

$$i = |b| \rightarrow \textcolor{blue}{i \geq |b|} \equiv \neg B_{\square}$$

15. a) Código:

---

```

1 i := 1;
2 r := 0;
3 while (i < |s|) do
4 if (s[r] <= s[i]) then
5 r := i;
6 else
7 skip
8 endif;
9 i := i+1;
10 endwhile

```

---

$$Pc \equiv |s| \geq 1 \wedge i = 1 \wedge r = 0$$

$$Qc \equiv 0 \leq r < |s| \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r \leq j < |s| \rightarrow_L s[r] > s[j]))$$

$$I \equiv (|s| \geq 1 \wedge 0 \leq i \leq |s| \wedge 0 \leq r < i) \wedge_L$$

$$((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < i \rightarrow_L s[r] > s[j]))$$

$$Pre \rightarrow Wp(i := 1, Wp(r := 0, Pc)) \stackrel{\text{Axioma 1}}{\equiv} |s| \geq 1 \wedge 1 = 1 \wedge 0 = 0 \equiv |s| \geq 1_{\square}$$

$$Qc \rightarrow Post_{\square}$$

$$Pc \rightarrow I : |s| \geq 1 \wedge i = 0 \rightarrow 0 \leq i \leq |s|$$

$$|s| \geq 1 \wedge i = 1 \wedge r = 0 \rightarrow 0 \leq r < i$$

$$|s| \geq 1 \wedge i = 1 \wedge 0 \leq r < i \rightarrow ((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < i \rightarrow_L s[r] > s[j])) \equiv s[r] \geq s[r] \equiv True_{\square}$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$C \equiv s[r] \leq s[i]; S_0 \equiv r := i; S_1 \equiv skip; S_2 \equiv i := i + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2;$$

$$Wp(S, I) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, I))$$

$$\stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, I))))$$

$$\stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_2, I)))$$

$$def(C) \equiv def(s[r] \leq s[i]) \equiv 0 \leq i < |s| \wedge 0 \leq r < |s|$$

$$\neg C \equiv s[r] > s[i]$$

$$Wp(S_2, I) \stackrel{\text{Axioma 1}}{\equiv} (|s| \geq 1 \wedge 0 \leq i + 1 \leq |s| \wedge 0 \leq r < i + 1) \wedge_L$$

$$((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < i + 1 \rightarrow_L s[r] > s[j]))$$

$$Wp(S_0, Wp(S_2, I)) \stackrel{\text{Axioma 1}}{\equiv} (|s| \geq 1 \wedge 0 \leq i + 1 \leq |s| \wedge 0 \leq i < i + 1) \wedge_L$$

$$((\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[i] \geq s[j]) \wedge (\forall j : \mathbb{Z})(i < j < i + 1 \rightarrow_L s[i] > s[j]))$$

$$\equiv (|s| \geq 1 \wedge 0 \leq i + 1 \leq |s|) \wedge (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[i] \geq s[j])$$

$$\begin{aligned}
&\Rightarrow Wp(S, I) \equiv (0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L \\
&((s[r] \leq s[i] \wedge ((|s| \geq 1 \wedge 0 \leq i + 1 \leq |s|) \wedge (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[i] \geq s[j]))) \vee (s[r] > s[i] \wedge \\
&(|s| \geq 1 \wedge 0 \leq i + 1 \leq |s| \wedge 0 \leq r < i + 1) \wedge_L ((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < \\
&j < i + 1 \rightarrow_L s[r] > s[j]))) \\
&\equiv (|s| \geq 1 \wedge 0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L ((s[r] \leq s[i] \wedge (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[i] \geq s[j])) \vee \\
&(s[r] > s[i] \wedge ((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge ((\forall j : \mathbb{Z})(r < j < i \rightarrow_L s[r] > s[j]) \wedge s[r] > \\
&s[i]))) \\
&\equiv (|s| \geq 1 \wedge 0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L ((s[r] \leq s[i] \wedge (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[i] \geq s[j])) \vee \\
&(s[r] > s[i] \wedge ((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < i \rightarrow_L s[r] > s[j])))
\end{aligned}$$

$$\begin{aligned}
&0 \leq i \leq |s| \wedge i < |s| \equiv 0 \leq i < |s| \\
&((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < i \rightarrow_L s[r] > s[j])) \rightarrow \\
&((s[r] \leq s[i] \wedge (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[i] \geq s[j])) \vee (s[r] > s[i] \wedge ((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L \\
&s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < i \rightarrow_L s[r] > s[j]))))_{\square} \\
(I \wedge \neg B) \rightarrow Qc : \neg i < |s| \equiv i \geq |s| \\
&0 \leq i \leq |s| \wedge i \geq |s| \equiv i = |s| \\
&i = |s| \wedge 0 \leq r < i \rightarrow 0 \leq r < |s| \\
&(|s| \geq 1 \wedge i = |s| \wedge 0 \leq r < |s|) \wedge_L \\
&((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < i \rightarrow_L s[r] > s[j])) \rightarrow \\
&((\forall j : \mathbb{Z})(0 \leq j \leq r \rightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j]))
\end{aligned}$$

4to paso  $f_v(i) = |s| - i$

$$\begin{aligned}
C \equiv s[r] \leq s[i]; S_0 \equiv r := i; S_1 \equiv \text{skip}; S_2 \equiv i := i + 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2; \\
Wp(S, f_v < v_0) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, f_v < v_0)) \\
\stackrel{\text{Axioma 4}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, f_v < v_0)))) \\
\stackrel{\text{Axioma 2}}{\equiv} \text{def}(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_2, f_v < v_0)))
\end{aligned}$$

$$\begin{aligned}
\text{def}(C) \equiv \text{def}(s[r] \leq s[i]) \equiv 0 \leq i < |s| \wedge 0 \leq r < |s| \\
\neg C \equiv s[r] > s[i] \\
Wp(S_2, f_v < v_0) \stackrel{\text{Axioma 1}}{\equiv} |s| - i - 1 < v_0 \\
Wp(S_0, Wp(S_2, f_v < v_0)) \stackrel{\text{Axioma 1}}{\equiv} |s| - i - 1 < v_0 \\
\Rightarrow Wp(S, f_v < v_0) \equiv (0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L ((s[r] \leq s[i] \wedge |s| - i - 1 < v_0) \vee (s[r] > \\
[i] \wedge |s| - i - 1 < v_0)) \\
\equiv 0 \leq i < |s| \wedge 0 \leq r < |s| \wedge |s| - i - 1 < v_0
\end{aligned}$$

$$\begin{aligned}
0 \leq i \leq |s| \wedge i < |s| \equiv 0 \leq i < |s| \\
v_0 = |s| - i \rightarrow |s| - i - 1 < v_0 \equiv -1 < 0 \equiv \text{True}_{\square}
\end{aligned}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : 0 \leq i \leq |s| \wedge |s| - i \leq 0 \rightarrow i = |s|$$

$$i = |s| \rightarrow \textcolor{blue}{i} \geq |s| \equiv \neg B_{\square}$$

b) Código:

---

```

1 i := |s|-2;
2 r := |s|-1;
3 while (0 <= i) do
4 if (s[r] < s[i]) then
5 r := i;
6 else
7 skip
8 endif;
9 i := i-1;
10 endwhile

```

---

$$Pc \equiv |s| \geq 1 \wedge i = |s| - 2 \wedge r = |s| - 1$$

$$Qc \equiv 0 \leq r < |s| \wedge_L ((\forall j : \mathbb{Z})(0 \leq j < r \longrightarrow_L s[r] \geq s[j]) \wedge (\forall j : \mathbb{Z})(r \leq j < |s| \longrightarrow_L s[r] > s[j]))$$

$$I \equiv (|s| \geq 1 \wedge -1 \leq i < |s| - 1 \wedge i < r < |s|) \wedge_L$$

$$((\forall j : \mathbb{Z})(r < j < |s| \longrightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i < j \leq r \longrightarrow_L s[r] \geq s[j]))$$

$$Pre \rightarrow Wp(i := |s| - 2, Wp(r := |s| - 1, Pc)) \stackrel{\text{Axioma } 1}{=} |s| \geq 1 \wedge |s| - 2 = |s| - 2 \wedge |s| - 1 = |s| - 1 \square$$

$$Qc \rightarrow Post_{\square}$$

$$Pc \rightarrow I : |s| \geq 1 \wedge i = |s| - 2 \rightarrow -1 \leq i < |s| - 1$$

$$|s| \geq 1 \wedge i = |s| - 2 \wedge r = |s| - 1 \rightarrow i < r < |s|$$

$$|s| \geq 1 \wedge i = |s| - 2 \wedge i < r < |s| \rightarrow ((\forall j : \mathbb{Z})(r < j < |s| \longrightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i < j \leq r \longrightarrow_L s[r] \geq s[j])) \equiv s[r] \geq s[r] \equiv True_{\square}$$

$$\{B \wedge I\}S\{I\} : \equiv (B \wedge I) \rightarrow Wp(S, I)$$

$$C \equiv s[r] < s[i]; S_0 \equiv r := i; S_1 \equiv skip; S_2 \equiv i := i - 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2;$$

$$Wp(S, I) \stackrel{\text{Axioma } 3}{=} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, I))$$

$$\stackrel{\text{Axioma } 4}{=} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, I))))$$

$$\stackrel{\text{Axioma } 2}{=} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, I))) \vee (\neg C \wedge Wp(S_2, I)))$$

$$def(C) \equiv def(s[r] < s[i]) \equiv 0 \leq i < |s| \wedge 0 \leq r < |s|$$

$$\neg C \equiv s[r] \geq s[i]$$

$$Wp(S_2, I) \stackrel{\text{Axioma } 1}{=} (|s| \geq 1 \wedge -1 \leq i - 1 < |s| - 1 \wedge i - 1 < r < |s|) \wedge_L$$

$$((\forall j : \mathbb{Z})(r < j < |s| \longrightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i - 1 < j \leq r \longrightarrow_L s[r] \geq s[j]))$$

$$Wp(S_0, Wp(S_2, I)) \stackrel{\text{Axioma } 1}{=} (|s| \geq 1 \wedge -1 \leq i - 1 < |s| - 1 \wedge i - 1 < i < |s|) \wedge_L$$

$$((\forall j : \mathbb{Z})(i < j < |s| \longrightarrow_L s[i] > s[j]) \wedge (\forall j : \mathbb{Z})(i - 1 < j \leq i \longrightarrow_L s[i] \geq s[j]))$$

$$\equiv (|s| \geq 1 \wedge -1 \leq i - 1 < |s| - 1) \wedge_L ((\forall j : \mathbb{Z})(i < j < |s| \longrightarrow_L s[i] > s[j]) \wedge s[i] \geq s[i])$$

$$\begin{aligned}
& \equiv |s| \geq 1 \wedge -1 \leq i - 1 < |s| - 1 \wedge_L ((\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[i] > s[j])) \\
& \Rightarrow Wp(S, I) \equiv (0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L \\
& ((s[r] < s[i] \wedge (|s| \geq 1 \wedge -1 \leq i - 1 < |s| - 1) \wedge_L (\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[i] > s[j])) \vee \\
& (s[r] \geq s[i] \wedge (|s| \geq 1 \wedge -1 \leq i - 1 < |s| - 1 \wedge i - 1 < r < |s|) \wedge_L ((\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i - 1 < j \leq r \rightarrow_L s[r] \geq s[j])))) \\
& \equiv (0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L ((s[r] < s[i] \wedge (\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[i] > s[j])) \vee \\
& (s[r] \geq s[i] \wedge ((\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i - 1 < j \leq r \rightarrow_L s[r] \geq s[j])))) \\
& \equiv (0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L ((s[r] < s[i] \wedge (\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[i] > s[j])) \vee \\
& (s[r] \geq s[i] \wedge ((\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j]) \wedge ((\forall j : \mathbb{Z})(i < j \leq r \rightarrow_L s[r] \geq s[j]) s[r] \geq s[i])))) \\
& \equiv (0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L ((s[r] < s[i] \wedge (\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[i] > s[j])) \vee \\
& (s[r] \geq s[i] \wedge ((\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i < j \leq r \rightarrow_L s[r] \geq s[j]))))
\end{aligned}$$

$$\begin{aligned}
& -1 \leq i < |s| \wedge 0 \leq i \equiv 0 \leq i < |s| \\
& ((\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i < j \leq r \rightarrow_L s[r] \geq s[j])) \rightarrow ((s[r] < s[i] \wedge (\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[i] > s[j])) \vee \\
& (s[r] \geq s[i] \wedge ((\forall j : \mathbb{Z})(r < j < |s| \rightarrow_L s[r] > s[j]) \wedge (\forall j : \mathbb{Z})(i < j \leq r \rightarrow_L s[r] \geq s[j]))))_{\square}
\end{aligned}$$

$$(I \wedge \neg B) \rightarrow Qc : \neg 0 \leq i \equiv 0 > i$$

$$-1 \leq i < |s| \wedge 0 > i \equiv 0 \leq i < |s| \rightarrow -1 \leq i < |s|_{\square}$$

$$4\text{to paso } f_v(i) = i + 1$$

$$C \equiv s[r] < s[i]; S_0 \equiv r := i; S_1 \equiv \text{skip}; S_2 \equiv i := i - 1; \Rightarrow S \equiv \text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi; } S_2;$$

$$Wp(S, f_v < v_0) \stackrel{\text{Axioma 3}}{\equiv} Wp(\text{if } C \text{ then } S_0 \text{ else } S_1 \text{ fi}, Wp(S_2, f_v < v_0))$$

$$\stackrel{\text{Axioma 4}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_1, Wp(S_2, f_v < v_0))))$$

$$\stackrel{\text{Axioma 2}}{\equiv} def(C) \wedge_L ((C \wedge Wp(S_0, Wp(S_2, f_v < v_0))) \vee (\neg C \wedge Wp(S_2, f_v < v_0)))$$

$$def(C) \equiv def(s[r] < s[i]) \equiv 0 \leq i < |s| \wedge 0 \leq r < |s|$$

$$\neg C \equiv s[r] \geq s[i]$$

$$Wp(S_2, f_v < v_0) \stackrel{\text{Axioma 1}}{\equiv} i < v_0$$

$$Wp(S_0, Wp(S_2, f_v < v_0)) \stackrel{\text{Axioma 1}}{\equiv} i < v_0$$

$$\Rightarrow Wp(S, I) \equiv (0 \leq i < |s| \wedge 0 \leq r < |s|) \wedge_L ((s[r] < s[i] \wedge i < v_0) \vee (s[r] \geq s[i] \wedge i < v_0))$$

$$\equiv 0 \leq i < |s| \wedge 0 \leq r < |s| \wedge i < v_0$$

$$-1 \leq i < |s| \wedge 0 \leq i \equiv 0 \leq i < |s|$$

$$v_0 = i + 1 \rightarrow i < v_0 \equiv 0 < 1_{\square}$$

$$(I \wedge f_v \leq 0) \rightarrow \neg B : -1 \leq i < |s| - 1 \wedge i + 1 \leq 0 \rightarrow i = -1$$

$$i = -1 \rightarrow i < 0 \equiv \neg B_{\square}$$

# Algoritmos y Estructuras de Datos I

Segundo Cuatrimestre 2020

## Guía Práctica 6 Testing



Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

**Ejercicio 1.** ★ Sea el siguiente programa:

```
int max(int x, int y) {
L1: int result = 0;
L2: if (x<y) {
L3: result = y;
} else {
L4: result = x;
}
L5: return result;
}
```

Y los siguientes casos de test:

■ test1:

- Entrada x = 0, y=0
- Resultado esperado result=0

■ test2:

- Entrada x = 0, y=1
- Resultado esperado result=1

1. Describir el diagrama de control de flujo (control-flow graph) del programa `max`.
2. Detallar qué líneas del programa cubre cada test

| Test  | L1 | L2 | L3 | L4 | L5 |
|-------|----|----|----|----|----|
| test1 |    |    |    |    |    |
| test2 |    |    |    |    |    |

3. Detallar qué decisiones (branches) del programa cubre cada test

| Test  | L2-True | L2-False |
|-------|---------|----------|
| test1 |         |          |
| test2 |         |          |

4. Decidir si la siguiente afirmación es verdadera o falsa: “El test suite compuesto por `test1` y `test2` cubre el 100 % de las líneas del programa y el 100 % de las decisiones (branches) del programa”

**Ejercicio 2.** ★ Sea la siguiente especificación del problema de retornar el mínimo elemento entre dos números enteros:

```
proc min (in x: Z, in y: Z, out result: Z) {
 Pre {True}
 Post {(x < y → result = x) ∧ (x ≥ y → result = y)}
}
```

Un programador ha escrito el siguiente programa para implementar la especificación descripta:

```

int min(int x, int y) {
L1: int result = 0;
L2: if (x < y) {
L3: result = x;
} else {
L4: result = x;
}
L5: return result;
}

```

Y el siguiente conjunto de casos de test (test suite):

■ minA:

- Entrada x=1,y=0
- Salida esperada 0

■ minB:

- Entrada x=0,y=1
- Salida esperada 0

1. Describir el diagrama de control de flujo (control-flow graph) del programa `min`.
2. ¿La ejecución del test suite resulta en la ejecución de todas las líneas del programa `min`?
3. ¿La ejecución del test suite resulta en la ejecución de todas las decisiones (branches) del programa?
4. ¿Es el test suite capaz de detectar el defecto de la implementación del problema de encontrar el mínimo?
5. Agregar nuevos casos de tests y/o modificar casos de tests existentes para que el test suite detecte el defecto.

**Ejercicio 3.** ★ Sea la siguiente especificación del problema de sumar y una posible implementación en lenguaje imperativo.

```

proc sumar (in x: Z, in y: Z, out result: Z) {
 Pre {True}
 Post {result = x + y}
}

```

```

int sumar(int x, int y) {
L1: int result = 0;
L2: result = result + x;
L3: result = result + y;
L4: return result;
}

```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `sumar`.
2. Escribir un conjunto de casos de test (o “test suite”) que ejecute todas las líneas del programa `sumar`.

**Ejercicio 4.** Sea la siguiente especificación del problema de restar y una posible implementación en lenguaje imperativo:

```

proc restar (in x: Z, in y: Z, out result: Z) {
 Pre {True}
 Post {result = x - y}
}

```

```

int restar(int x, int y) {
L1: int result = 0;
L2: result = result + x;
L3: result = result + y;
L4: return result;
}

```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `restar`.
2. Escribir un conjunto de casos de test (o “test suite”) que ejecute todas las líneas del programa `restar`.
3. La línea L3 del programa `restar` tiene un defecto, ¿es el test suite descripto en el punto anterior capaz de detectarlo? En caso contrario, modificar o agregar nuevos casos de test hasta lograr detectarlo.

**Ejercicio 5.** Sea la siguiente especificación del problema de `signo` y una posible implementación en lenguaje imperativo:

```
proc signo (in x: ℝ, out result: ℤ) {
 Pre {True}
 Post {(result = 0 ∧ x = 0) ∨ (result = -1 ∧ x < 0) ∨ (result = 1 ∧ x > 0)}
}

int signo (float x) {
L1: int result = 0;
L2: if (x < 0) {
L3: result = -1;
L4: } else if (x > 0){
L5: result = 1;
}
L6: return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `signo`.
2. Escribir un test suite que ejecute todas las líneas del programa `signo`.
3. ¿El test suite del punto anterior ejecuta todas las posibles decisiones (“branches”) del programa?

**Ejercicio 6.** Sea la siguiente especificación del problema de `signo` y una posible implementación en lenguaje imperativo:

```
proc signo (in x: ℝ, out result: ℤ) {
 Pre {True}
 Post {(r = 0 ∧ x = 0) ∨ (r = -1 ∧ x < 0) ∨ (r = 1 ∧ x > 0)}
}

int signo (float x) {
L1: int result = 0;
L2: if (x < 0) {
L3: result = -1;
}
L4: return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `signo`.
2. Escribir un test suite que ejecute todas las líneas del programa `signo`.
3. Escribir un test suite que ejecute todas las posibles decisiones (“branches”) del programa.
4. Escribir un test suite que ejecute todas las líneas del programa pero no ejecute todos las decisiones del programa.

**Ejercicio 7. ★** Sea la siguiente especificación:

```
proc fabs (in x: ℤ, out result: ℤ) {
 Pre {True}
 Post {result = |x|}
}
```

Y la siguiente implementación:

```

int fabs(int x) {
L1: if (x<0) {
L2: return -x;
} else {
L3: return +x;
}
}

```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `fabs`.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

**Ejercicio 8.** ★ Sea la siguiente especificación del problema de `mult10` y una posible implementación en lenguaje imperativo:

```

proc mult10 (in x: Z, out result: Z) {
 Pre {True}
 Post {result = x * 10}
}

int mult10(int x) {
L1: int result = 0;
L2: int count = 0;
L3: while (count < 10) {
L4: result = result + x;
L5: count = count + 1;
}
L6: return result;
}

```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `mult10`.
2. Escribir un test suite que ejecute todas las líneas del programa `mult10`.
3. ¿El test suite anterior ejecuta todas las posibles decisiones (“branches”) del programa?

**Ejercicio 9.** Sea la siguiente especificación del problema de `sumar` y una posible implementación en lenguaje imperativo:

```

proc sumar (in x: Z, in y: Z, out result: Z) {
 Pre {True}
 Post {result = x + y}
}

int sumar(int x, int y) {
L1: int sumando = 0;
L2: int abs_y = 0;
L3: if (y<0) {
L4: sumando = -1;
L5: abs_y = -y;
} else {
L7: sumando = 1;
L8: abs_y = y;
}
L9: int result = x;
L10: int count = 0;
L11: while (count < abs_y) {
L12: result = result + sumando;
L13: count = count + 1;
}
L14: return result;
}

```

1. Describir el diagrama de control de flujo (control-flow graph) del programa **sumar**.
2. Escribir un test suite que ejecute todas las líneas del programa **sumar**.
3. Escribir un test suite que ejecute todas las posibles decisiones (“branches”) del programa.

**Ejercicio 10.** Sea el siguiente programa que computa el máximo común divisor entre dos enteros.

```
int mcd(int x, int y) {
L1: assert(x >= 0 && y >= 0) // Pre: x e y tienen que ser no negativos
L2: int tmp = 0;
L3: while(y != 0) {
L4: tmp = x % y;
L5: x = y;
L6: y = tmp;
}
L7: return x; // gcd
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa **mcd**.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

**Ejercicio 11.** ★ Sea el siguiente programa que retorna diferentes valores dependiendo si  $a$ ,  $b$  y  $c$ , definen lados de un triángulo inválido, equilátero, isósceles o escaleno.

```
int triangle(int a, int b, int c) {
L1: if (a <= 0 || b <= 0 || c <= 0) {
L2: return 4; // invalido
}
L3: if (!(a + b > c && a + c > b && b + c > a)) {
L4: return 4; // invalido
}
L5: if (a == b && b == c) {
L6: return 1; // equilatero
}
L7: if (a == b || b == c || a == c) {
L8: return 2; // isosceles
}
L9: return 3; // escaleno
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa **triangle**.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

**Ejercicio 12.** ★ Sea la siguiente especificación del problema de **multByAbs** y una posible implementación en lenguaje imperativo:

```
proc multByAbs (in x: Z, in y:Z, out result: Z) {
 Pre {True}
 Post {result = x * |y|}
}
```

```
int multByAbs(int x, int y) {
L1: int abs_y = fabs(y); // ejercicio anterior
L2: if (abs_y < 0) {
L3: return -1;
} else {
L4: int result = 0;
```

```

L5: int i = 0;
L6: while (i<abs_y) {
L7: result = result + x;
L8: i = i + 1;
}
L9: return result;
}
}

```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `multByAbs`.
2. Detallar qué líneas y branches del programa no pueden ser cubiertos por ningún caso de test. ¿A qué se debe?
3. Escribir el test suite que cubra todas las líneas y branches que puedan ser cubiertos.

**Ejercicio 13.** Sea la siguiente especificación del problema de `vaciarSecuencia` y una posible implementación en lenguaje imperativo:

```

proc vaciarSecuencia (inout s: seq<Z>) {
 Pre {S0 = s}
 Post {|s| = |S0| ∧
 (∀j : Z)(0 ≤ j < |s| →L s[j] = 0)}
}

void vaciarSecuencia (vector<int> &s) {
L1,L2,L3: for (int i=0; i<s.size(); i++) {
L4: s[i]=0;
}
}

```

1. Escribir el diagrama de control de flujo (control-flow graph) del programa `vaciarSecuencia`.
2. Escribir un test suite que cubra todos las líneas de programa (observar que un `for` contiene 3 líneas distintas)
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

**Ejercicio 14.** Sea la siguiente especificación del problema de `existeElemento` y una posible implementación en lenguaje imperativo:

```

proc existeElemento (in s: seq<Z>, in e:Z, out result: Bool) {
 Pre {True}
 Post {result = True ↔ (∃j : Z)(0 ≤ j < |s| ∧L s[j] = e)}
}

bool existeElemento (vector<int> s, int e) {
L1: bool result = false;
L2,L3,L4: for (int i=0; i<s.size(); i++) {
L5: if (s[i]==e) {
L6: result = true;
L7: break;
}
}
L8: return result;
}

```

1. Escribir el diagrama de control de flujo (control-flow graph) del programa `existeElemento`.
2. Escribir un test suite que cubra todos las líneas de programa (observar que un `for` contiene 3 líneas distintas)

3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

**Ejercicio 15.** Sea la siguiente especificación del problema de `cantidadDePrimos` y una posible implementación en lenguaje imperativo:

```

proc cantidadDePrimos (in n: Z, out result: Z) {
 Pre {n ≥ 0}
 Post {result = ∑i=2n (if ((∀j : Z)((1 < j < i) → i mod j ≠ 0)) then 1 else 0 fi)}
}

int cantidadDePrimos(int n) {
L1: int result = 0;
L2,L3,L4: for (int i=2; i<=n; i++) {
L5: bool inc = esPrimo(i);
L6: if (inc==true) {
L7: result++;
 }
 }
L8: return result;
}
/* procedimiento auxiliar */
bool esPrimo(int x) {
L9: int result = true;
L10,L11,L12: for (int i=2 ; i <x; i++) {
L13: if (x % i == 0) {
L14: result = false;
L15: break;
 }
 }
L16: return result;
}

```

1. Escribir los diagramas de control de flujo (control-flow graph) para `cantidadDePrimos` y la función auxiliar `esPrimo`.
2. Escribir un test suite que cubra todos las líneas de programa del programa `cantidadDePrimos` y `esPrimo`.
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

**Ejercicio 16.** Sea la siguiente especificación del problema de `esSubsecuencia` y una posible implementación en lenguaje imperativo:

```

proc esSubsecuencia (in s: seq<Z>, in r: seq<Z>, out result: Bool) {
 Pre {True}
 Post {result = True ↔ |r| ≤ |s|
 ∧L (∃i : Z)((0 ≤ i < |s| ∧ i + |r| < |s|) ∧L (∀j : Z)(0 ≤ j < |r| →L s[i + j] = r[j]))}
}

```

```

1 bool esSubsecuencia(vector<int> s, vector<int> r) {
2 bool result = false;
3 int ultimoIndice = s.size() - r.size();
4 for (int i = 0; i < ultimoIndice; i++) {
5
6 /* obtener una subsecuencia de s */
7 vector<int> subseq = subsecuencia(s, i, r.size());
8
9 /* chequear si la subsecuencia es igual a r */
10 bool sonIguales = iguales(subseq, r);

```

```

11 if (sonIguales==true) {
12 result = true;
13 break;
14 }
15 }
16 return result;
17 }
18 /* procedimiento auxiliar subsecuencia*/
19 vector<int> subsecuencia(vector<int> s, int desde, int longitud) {
20 vector<int> rv;
21 int hasta = desde+longitud;
22 for (int i=desde;i<hasta ;i++) {
23 int elem = s[i];
24 rv.push_back(elem);
25 }
26 return rv;
27 }
28 /* procedimiento auxiliar iguales*/
29 bool iguales(vector<int> a, vector<int> b) {
30 bool result = true;
31 if (a.size()==b.size()) {
32 for (int i=0;i<a.size() ;i++) {
33 if (a[i]!=b[i]) {
34 result = false;
35 break;
36 }
37 }
38 } else {
39 result = false;
40 }
41 return result;
42 }

```

1. Escribir los diagramas de control de flujo (control-flow graph) para `esSubsecuencia` y las funciones auxiliares `subsecuencia` e `iguales`.
2. Escribir un test suite que cubra todos las líneas de programa *ejecutables* de todos los procedimientos. Observar que un `for` contiene 3 líneas distintas.
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

# 1. Práctica 6: Testing

Esta es la página con la cuál hicimos los diagramas de flujo

1. a) Diagrama

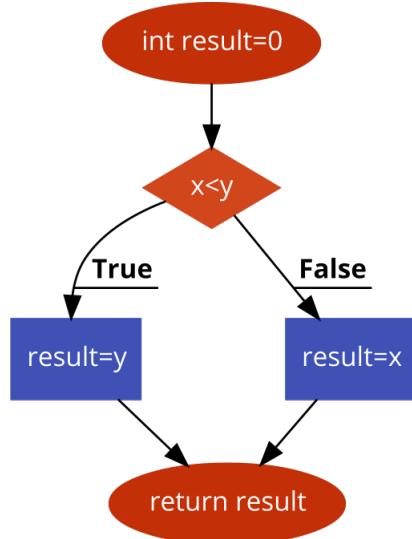


Figura 1: Control Flow Graph de **Máx**

- b) Cubrimiento de líneas de cada test

| Test  | L1 | L2 | L3 | L4 | L5 |
|-------|----|----|----|----|----|
| Test1 | X  | X  |    | X  | X  |
| Test2 | X  | X  | X  |    | X  |

- c) Branches del programa que cubre cada test

| Test  | L2-True | L2-False |
|-------|---------|----------|
| Test1 |         | X        |
| Test2 | X       |          |

- d) Es verdadera

2. a) Diagrama

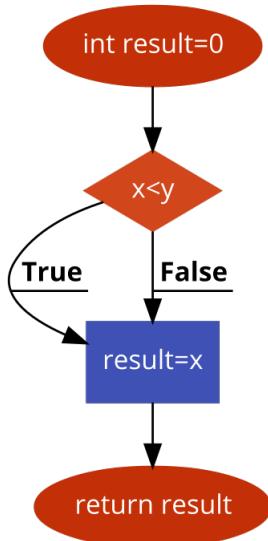


Figura 2: Control Flow Graph de **Min**

- b) Si
  - c) Si
  - d) Si, el primer test case lo encuentra
  - e) No es necesario, ya que el test suite original detecta el error
3. a) Diagrama

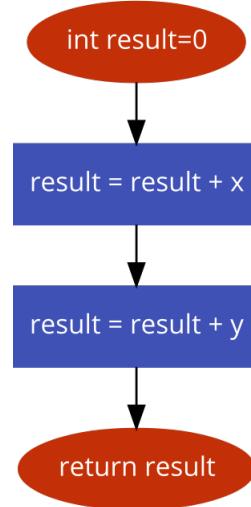


Figura 3: Control Flow Graph de **Sumar**

- b) **Test Case:** = { Entrada = [  $x = 2 \ y = 3$  ] Salida esperada= [result==5] }
4. a) Diagrama

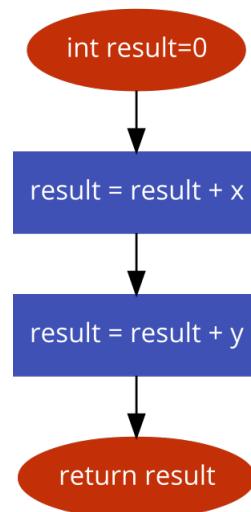


Figura 4: Control Flow Graph de **Restar**

- b) **Test Case:** = { Entrada = [  $x = 1 \ y = 1$  ] Salida esperada= [result==0] }
- c) Si, el test case detecta el defecto

5. a) Diagrama

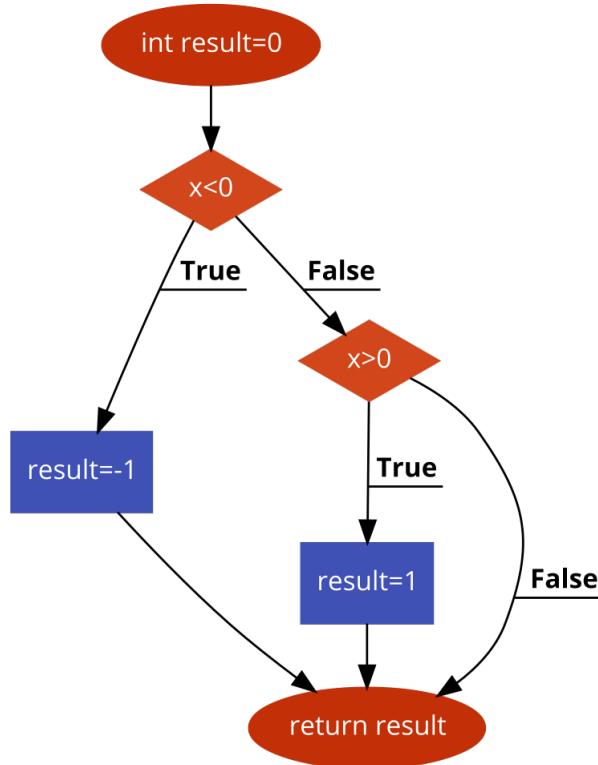


Figura 5: Control Flow Graph de **Signo**

b) Test Suite

| Tests Cases | Entrada | Salida Esperada |
|-------------|---------|-----------------|
| Test1       | 0       | 0               |
| Test2       | 2       | 1               |
| Test3       | -5      | -1              |

c) Si, ejecuta todas las decisiones posibles

6. a) Diagrama

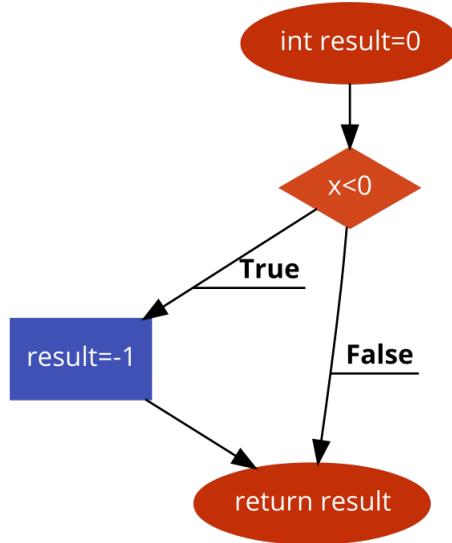


Figura 6: Control Flow Graph de **Signo2**

b) Test Suite

| Tests Cases | Entrada | Salida Esperada |
|-------------|---------|-----------------|
| Test1       | 0       | 0               |
| Test2       | 2       | 1               |
| Test3       | -5      | -1              |

c) Puedo usar el test suite anterior

d) Test Suite

| Tests Cases | Entrada | Salida Esperada |
|-------------|---------|-----------------|
| Test1       | -3      | -1              |

7. a) Diagrama

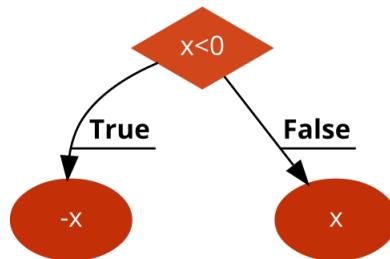


Figura 7: Control Flow Graph de **Fabs**

b) Test Suite

| Tests Cases | Entrada | Salida Esperada |
|-------------|---------|-----------------|
| Test1       | 2       | 2               |
| Test2       | -3      | 3               |

8. a) Diagrama

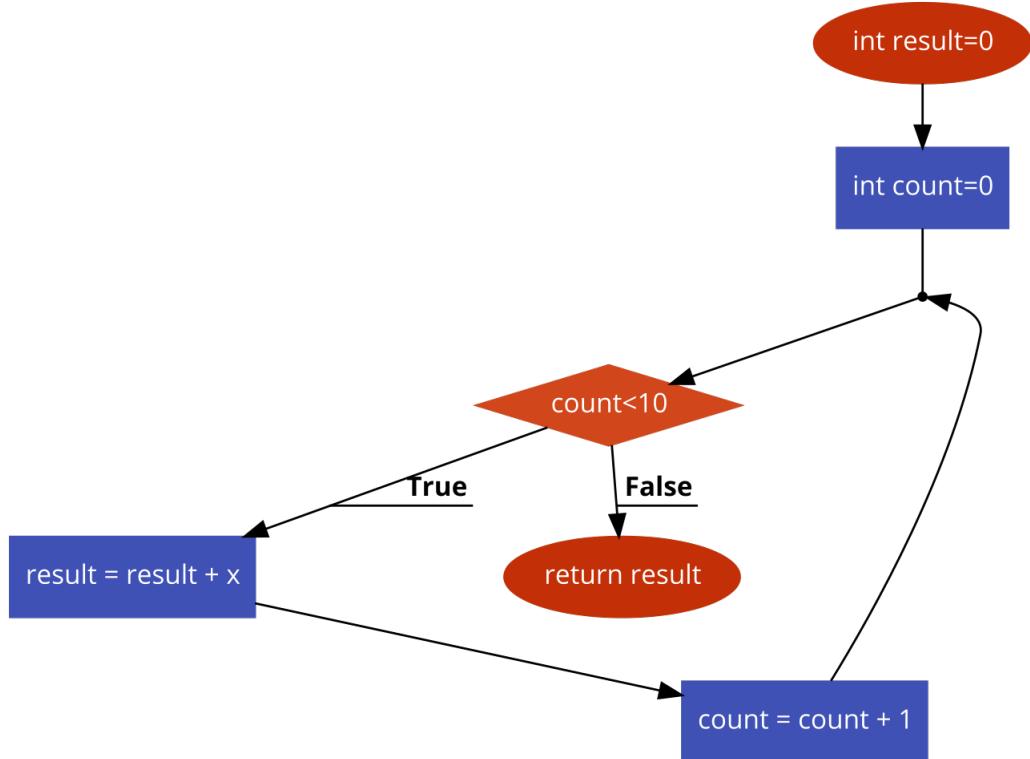


Figura 8: Control Flow Graph de **mult10**

- b) **Test Case:** = { Entrada = 2 Salida esperada= [result==20] }
- c) Si, ejecuta todas las decisiones

9. a) Diagrama

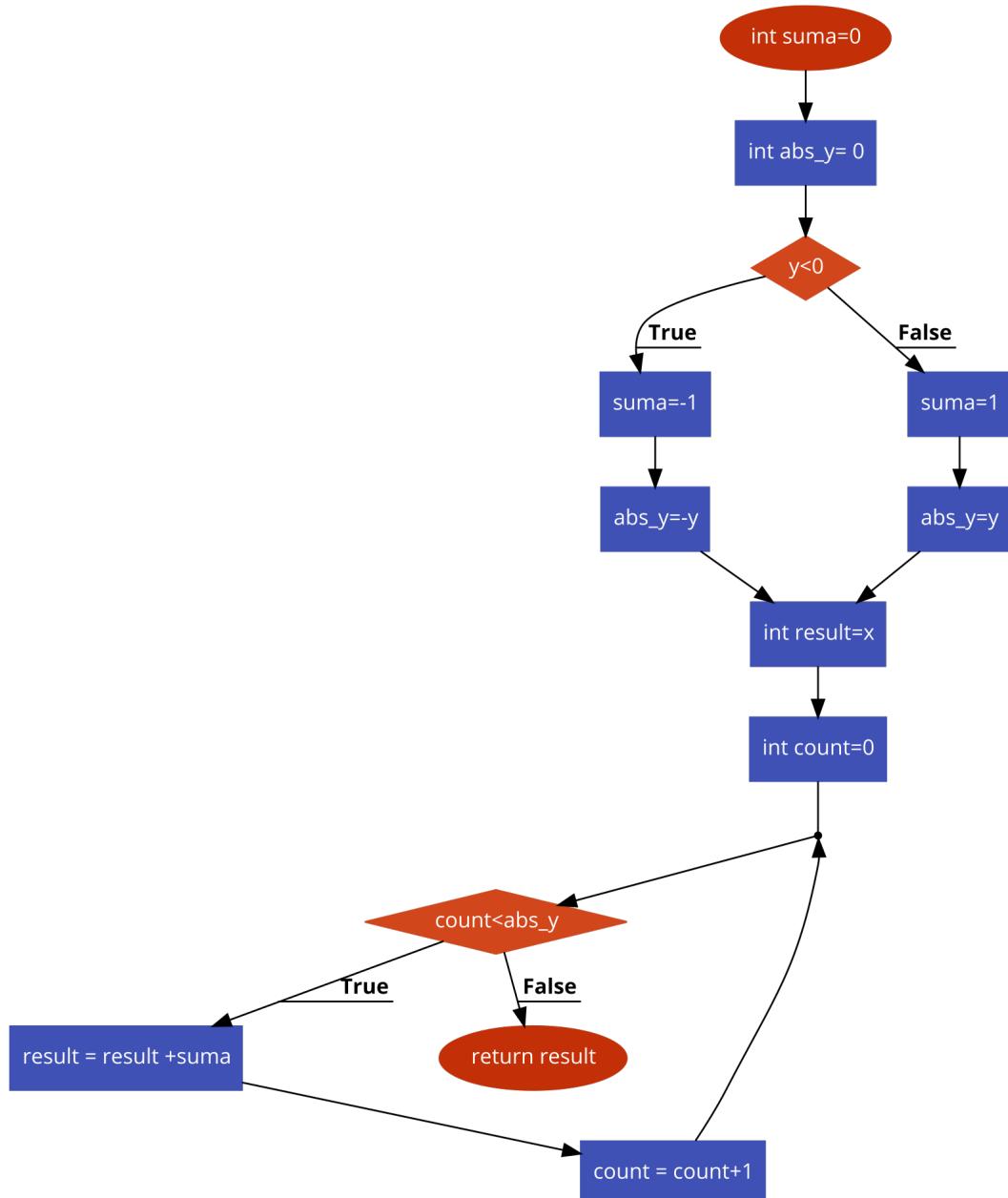


Figura 9: Control Flow Graph de **Sumar**

b) +c) Test Suite

| Tests Cases | Entrada   | Salida Esperada |
|-------------|-----------|-----------------|
| Test1       | x=2 , y=5 | x=2 , y=-5      |
| Test2       | 7         | -3              |

10. a) Diagrama

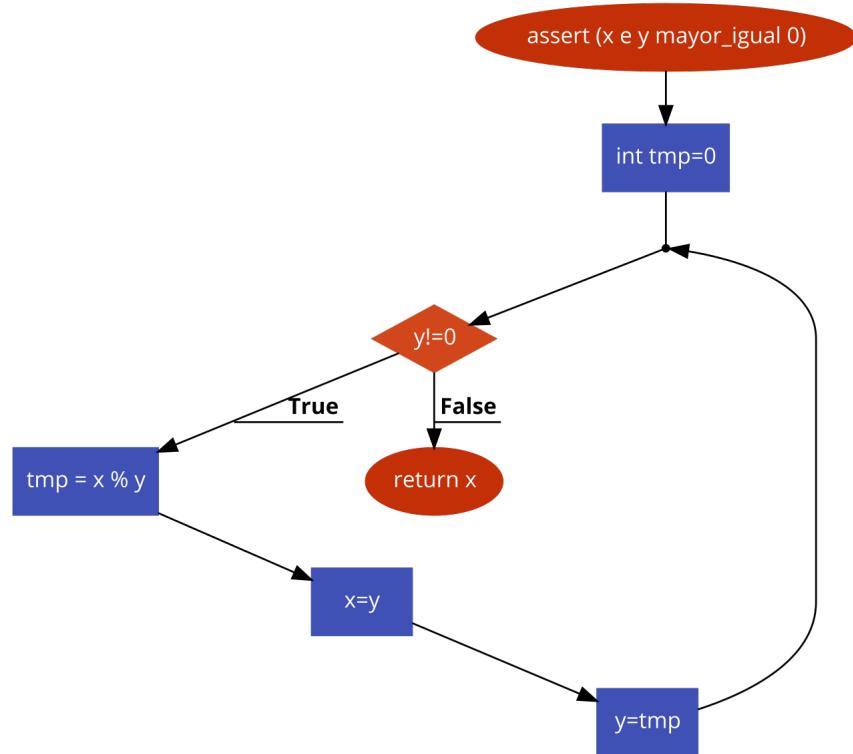


Figura 10: Control Flow Graph de Mcd

b) **Test Case:** = { Entrada = [x=4, y=2] Salida esperada= [result==2] }

11. a) Diagrama

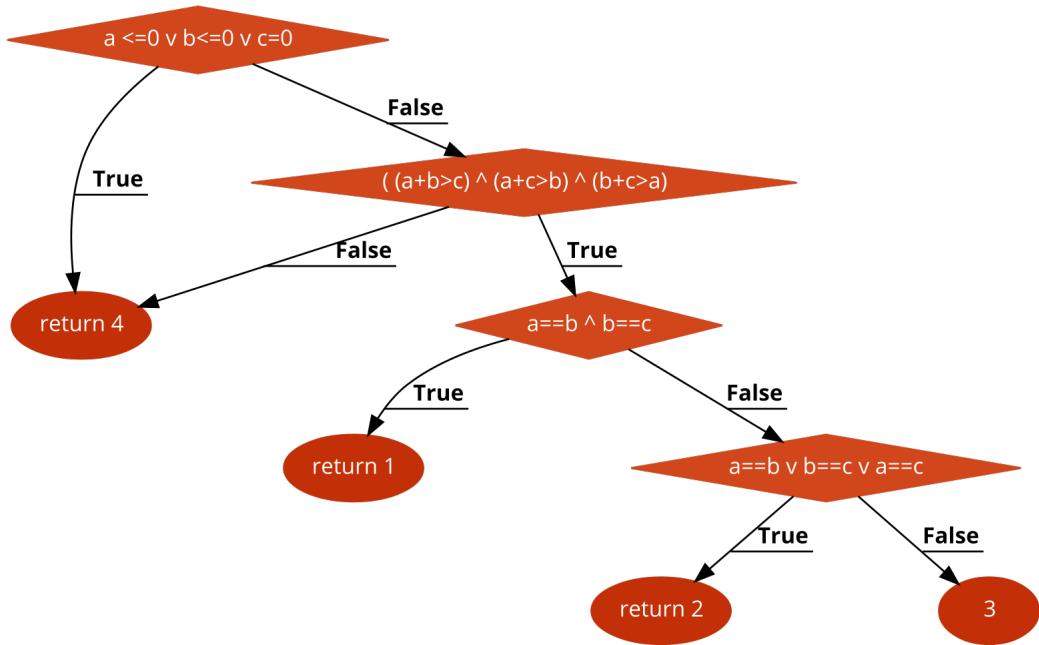


Figura 11: Control Flow Graph de **Triangle**

b) Test Suite

| Tests Cases | a | b | c | Salida Esperada |
|-------------|---|---|---|-----------------|
| Test1       | 0 | 0 | 0 | 4               |
| Test2       | 1 | 1 | 3 | 4               |
| Test3       | 3 | 3 | 3 | 1               |
| Test4       | 3 | 3 | 2 | 2               |
| Test4       | 2 | 3 | 4 | 3               |

12. a) Diagrama

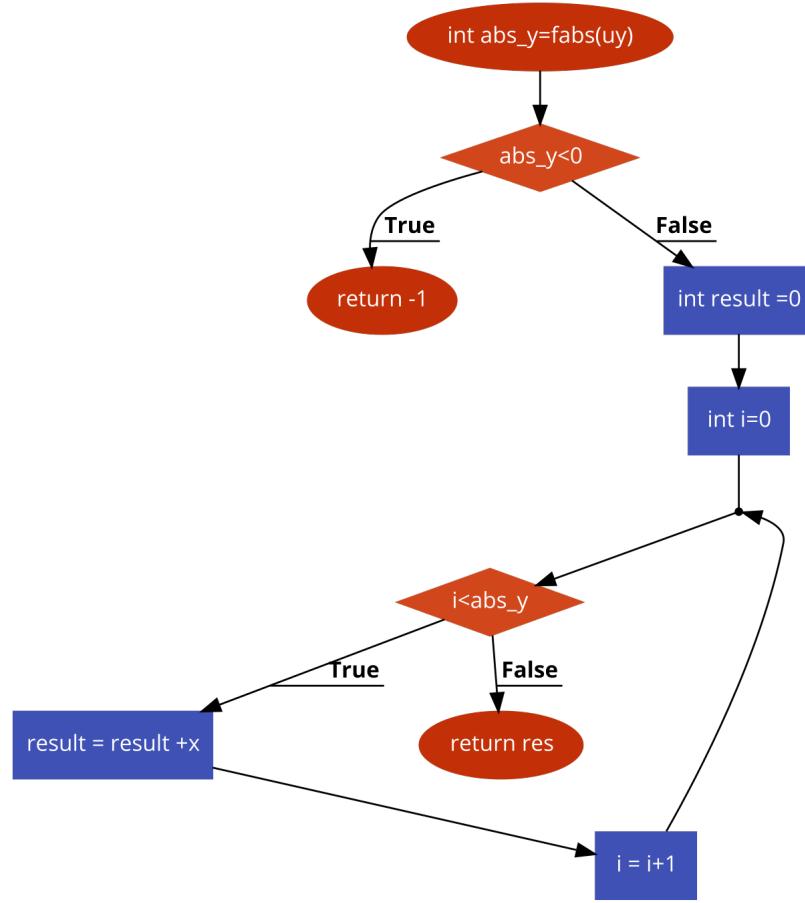


Figura 12: Control Flow Graph de **MultByAbs**

- b) Nunca puede llegar a  $return -1$  ya que  $fabs(x) \geq 0$ , por lo tanto nunca se cumple la guarda ( $absy < 0$ )
- c) **Test Case:** = { Entrada = [x=3, y=2] Salida esperada= [result==6] }

13. a) Diagrama

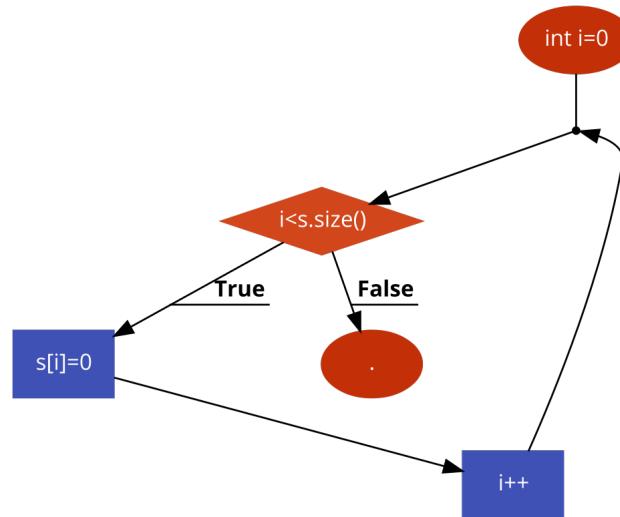


Figura 13: Control Flow Graph de **VaciarSecuencia**

b) **Test Case:** = { Entrada =  $s=\langle 1, 2, 3 \rangle$  Salida esperada=  $s=\langle 0, 0, 0 \rangle$  }

c)

14. a) programa

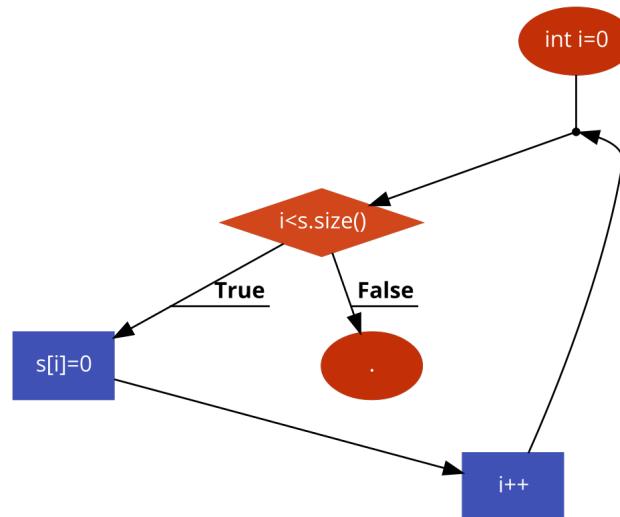


Figura 14: Control Flow Graph de **existeElemento**

b) **Test Case:** = { Entrada = [  $s=\langle 1, 2, 3, 4 \rangle$  ,  $e=3$  ] Salida esperada= result==True }

15. a) Diagramas

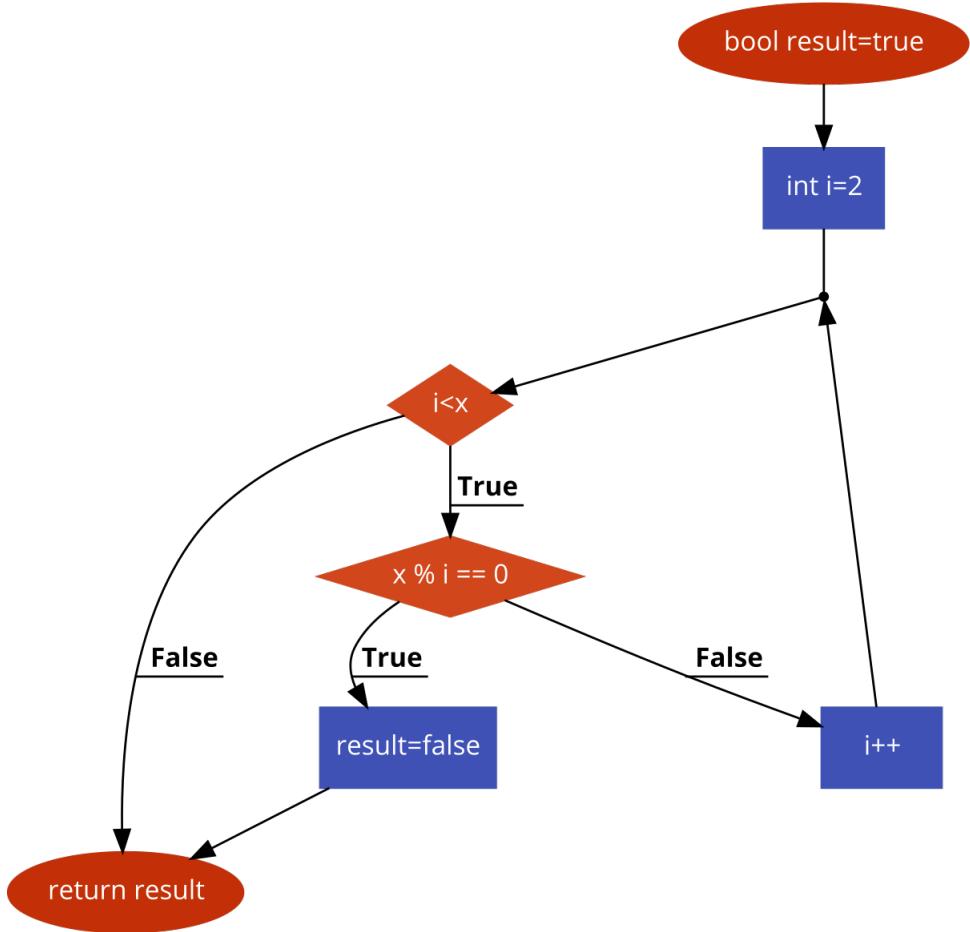


Figura 15: Control Flow Graph de `esPrimo`

b) +c) **Test Case:** = { Entrada =  $x=7$  Salida esperada=  $\text{result}==3$  }

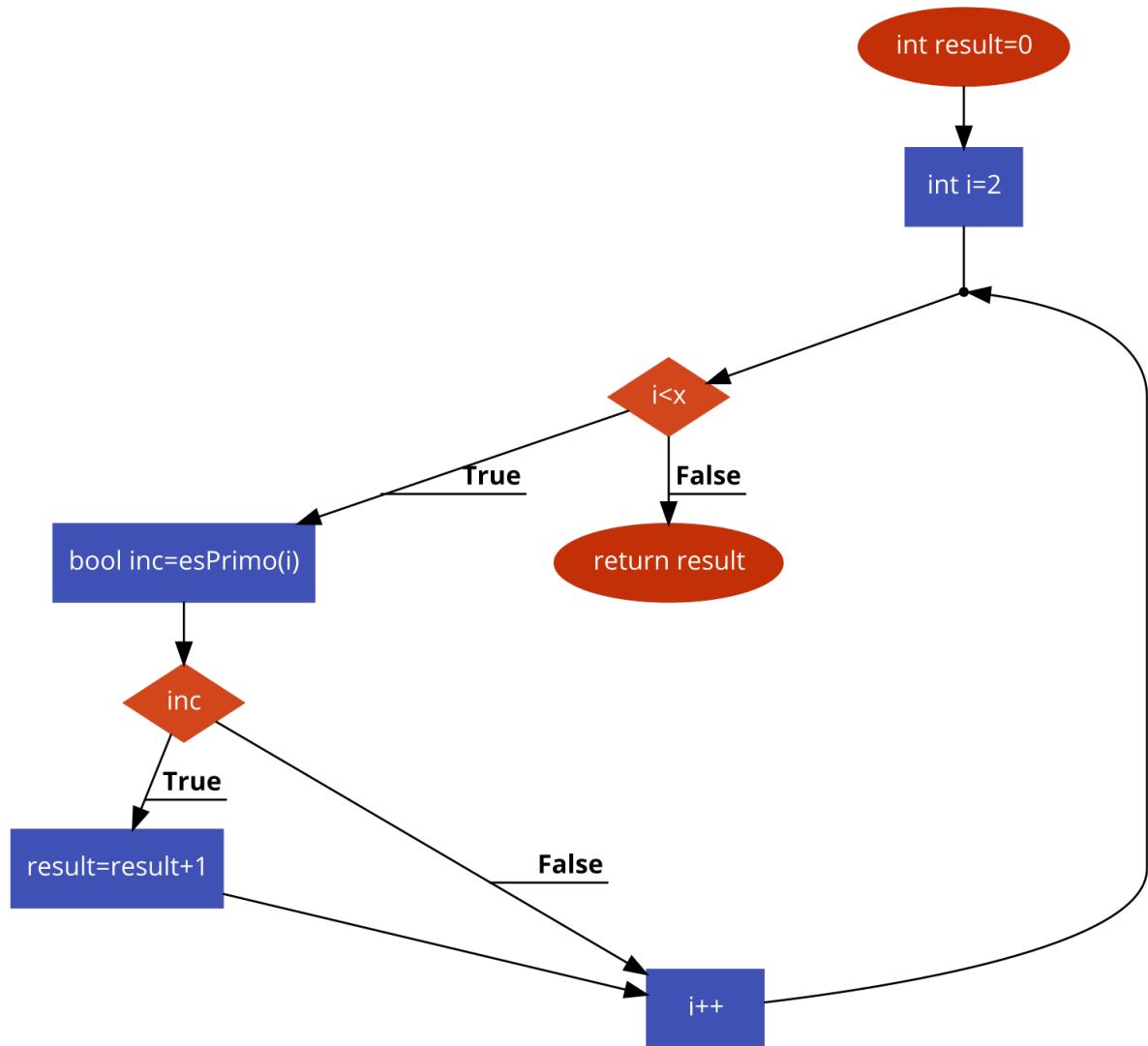


Figura 16: Control Flow Graph de **cantidadDePrimos**

16. a) Diagramas
- b) **Test Case:** = { Entrada = [ r= $\langle 1, 2, 3, 4 \rangle$  , s= $\langle -3, -2, 0, 1, 2, 3, 4, 5, 6 \rangle$  ] , Salida esperada= result==True }
 

Nunca va a pasar que dos secuencias tengan distinta longitud, pues subseq siempre tiene longitud —r—.

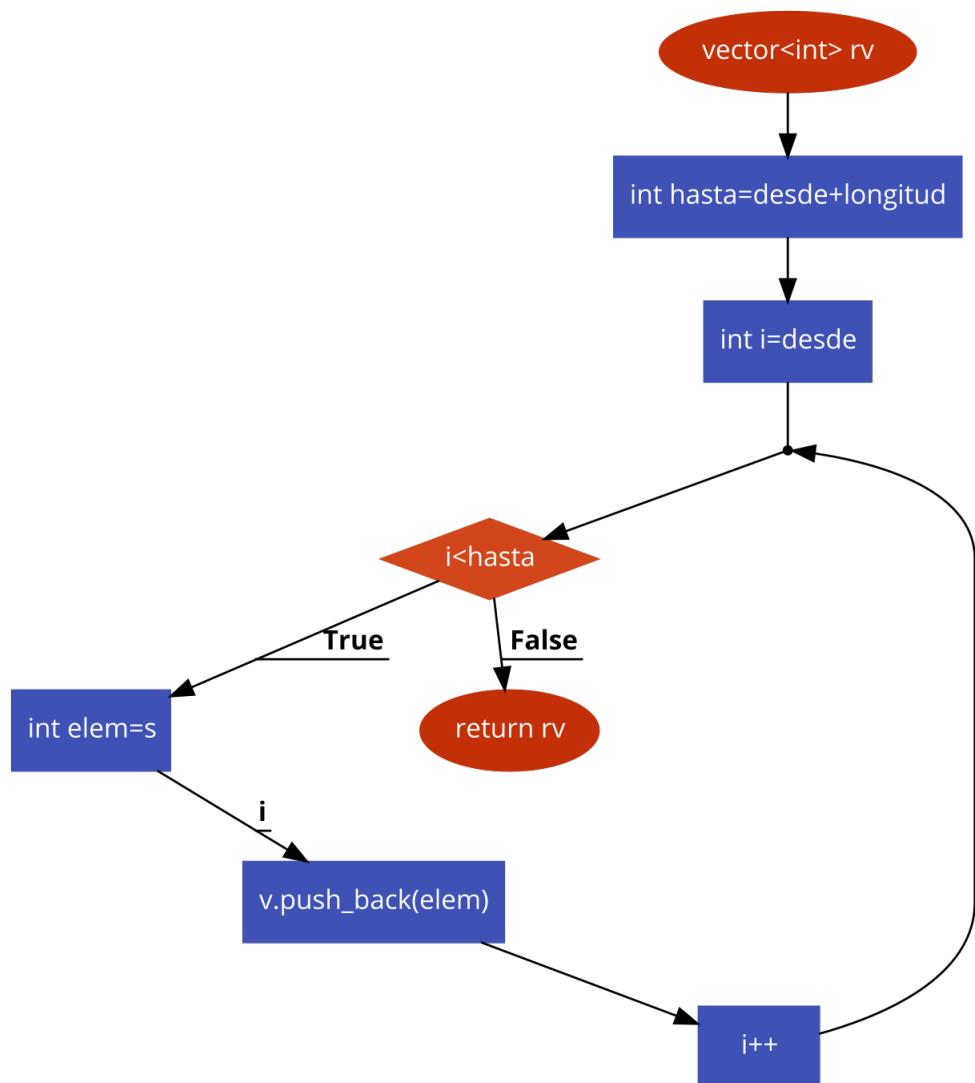


Figura 17: Control Flow Graph de **Subsecuencia**

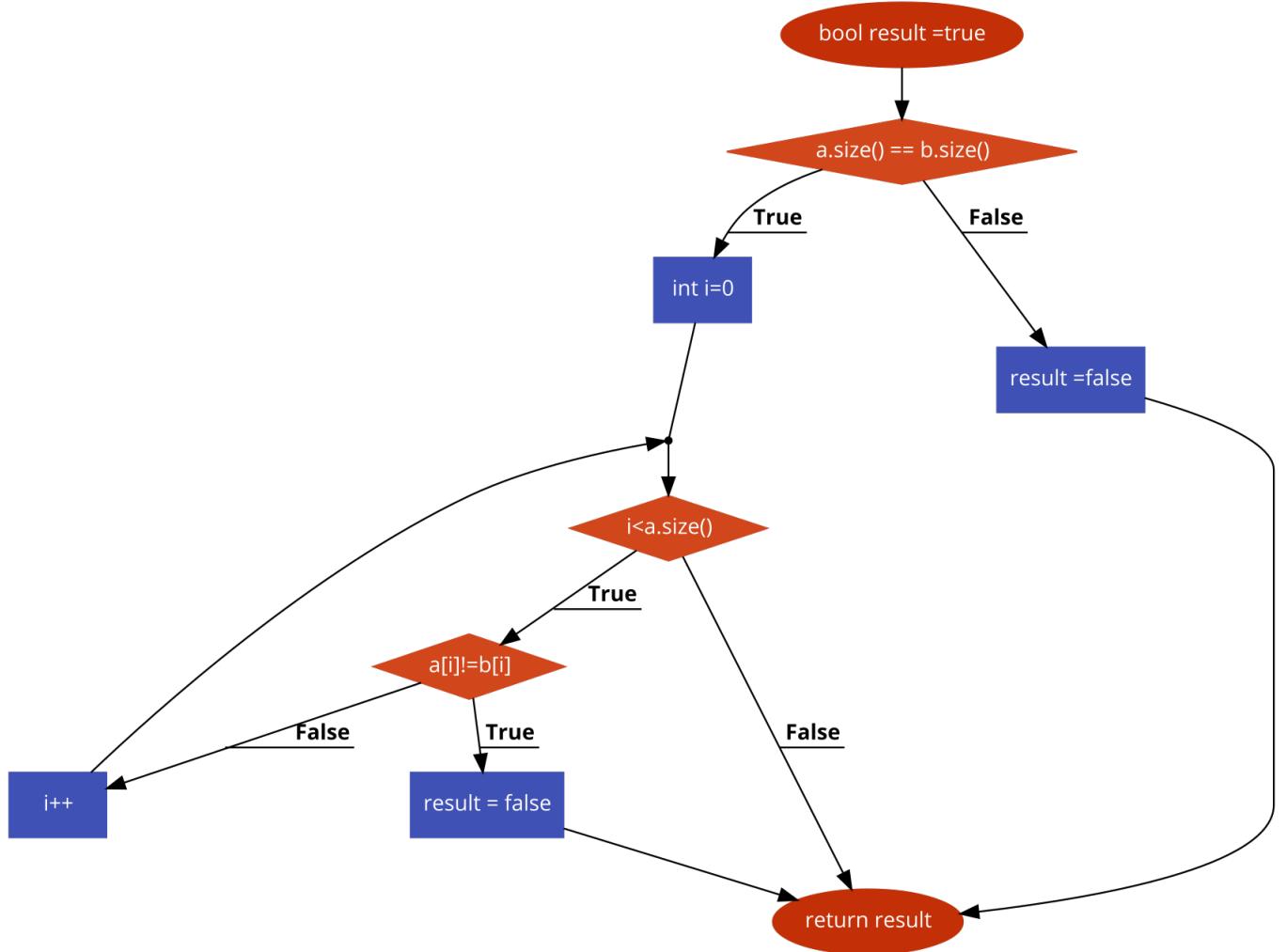


Figura 18: Control Flow Graph de **Iguals**

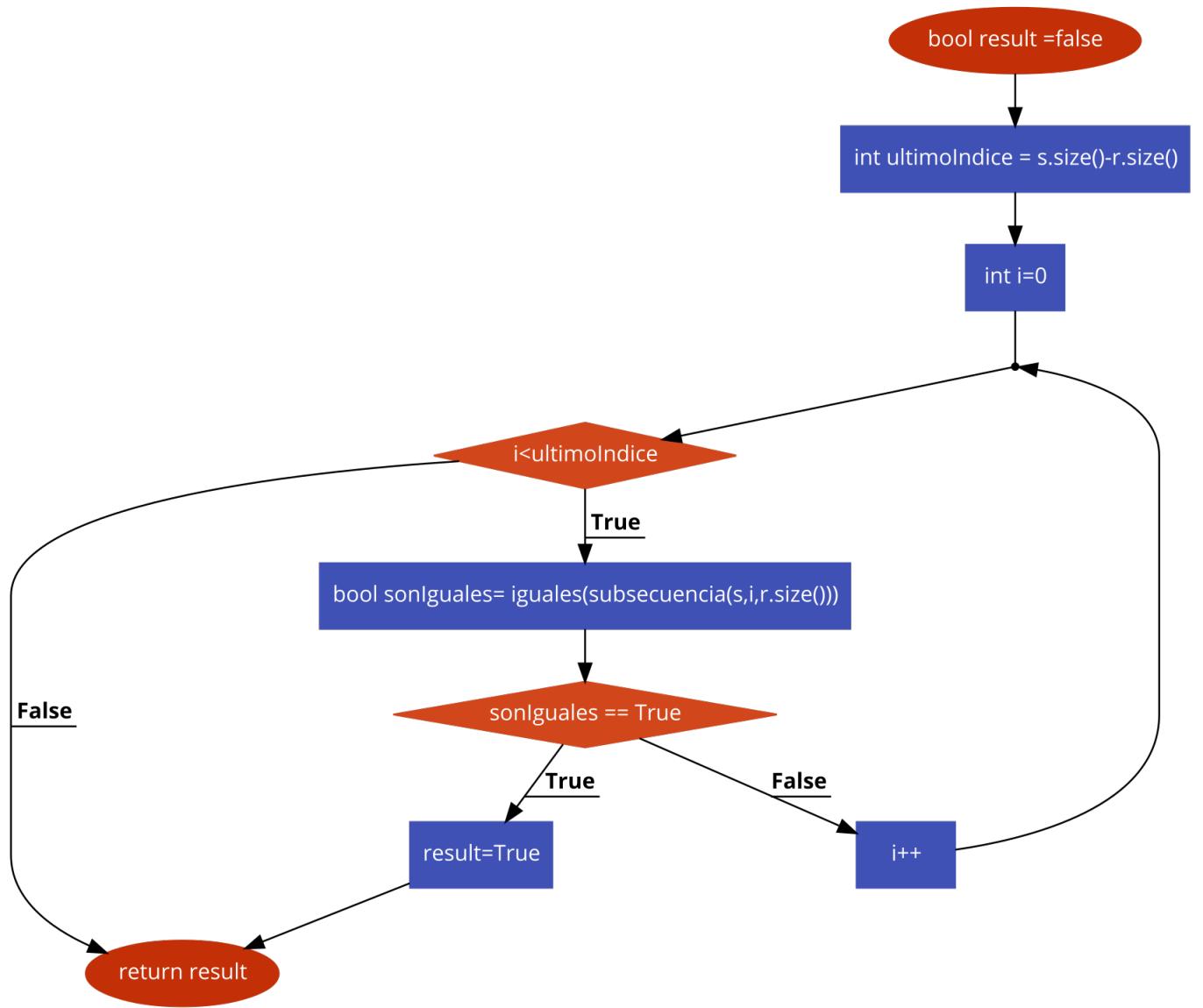


Figura 19: Control Flow Graph de `esSubsecuencia`

## 2. Bonus track : Moraleja de Buenas Prácticas de la Programación

*Clean Code : Robert C Martin*

”Conocía una compañía la cuál en los años 80 creo una app increíble. Era muy popular y muchos profesionales la compraban y usaban. Pero luego los tiempos entre actualizaciones comenzaron a alargarse. Los bugs no eran reparados de una actualización a la otra. Los tiempos de carga crecían y los crashes aumentaban. Recuerdo ese día en el que cerre la aplicación frustrado y nunca la volví a usar. La compañía cerro poco tiempo después.

Dos decadas más tarde conocí a uno de los primeros empleados de esta empresa y le pregunte que había pasado. Su respuesta confirmo mis temores. Habían apresurado la salida del producto y terminaron con una maraña de código. Mientras iban agregando más y más funcionalidades el código se ponía cada vez peor, hasta que llegar a un punto en el que era inmanejable. Fue el mál código quién llevo a la compañía a su ruina.”

Aunque el testing pueda ser algo tedioso o repetitivo es una actividad clave para generar código de calidad y confiable. Es mejor escribir un código lentamente e ir testeandolo de a poco. Que escribirlo todo en un día y debuggear por una semana.

# Algoritmos y Estructuras de Datos I

Segundo Cuatrimestre 2020

Guía Práctica 7  
Ciclos a partir de invariantes



Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

## Ejercicio 1.

Sea la siguiente especificación del problema de copiar una secuencia de enteros:

```
proc copiarSecuencia (in s: seq<Z>, out result: seq<Z>) {
 Pre {True}
 Post {s = result}
}
```

Sea la siguiente implementación incompleta de la función copiarSecuencia:

```
vector<int> copiarSecuencia(vector<int> s) {
 vector<int> r;
 int i = 0;
 while(i < s.size()) {
 ...
 }
 return r;
}
```

Completar el programa (i.e. escribir el cuerpo del `while`) de forma que cumpla el siguiente invariante de ciclo:

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

## Ejercicio 2.

Sea la siguiente especificación:

```
proc incSecuencia (inout s: seq<Z>) {
 Pre {s = S_0}
 Post {|s| = |S_0| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L s[i] = S_0[i] + 1)}
}
```

Sea la siguiente implementación incompleta de la función incSecuencia:

```
void incSecuencia(vector<int> &a) {
 int i = 0;
 while(...) {
 ...
 }
}
```

Completar el programa (i.e. escribir el cuerpo del `while` y su guarda) de forma que cumpla el siguiente invariante de ciclo:

$$I = 0 \leq i \leq |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = S_0[j] + 1)$$

## Ejercicio 3.

Sea la siguiente especificación del problema de retornar la cantidad de apariciones de un elemento en una secuencia de enteros:

```
proc cantApariciones (in s: seq<Z>, in e: Z, out result: Z) {
 Pre {True}
 Post {result = #apariciones(s, e)}
}
```

Sea la siguiente implementación incompleta de la función cantApariciones:

```
int cantApariciones(vector<int> s, int e) {
 int r = 0;
 for(int i=0; ... ; ...) {
 ...
 }
 return r;
}
```

Completar el programa (i.e. escribir el cuerpo y la declaración del **for**) de forma que cumpla el siguiente invariante de ciclo:

$$I = 0 \leq i \leq |s| \wedge_L r = \#\text{apariciones}(\text{subseq}(s, 0, i), e)$$

#### Ejercicio 4.

Sea la siguiente especificación de un ciclo:

- $P_c : i = -1 \wedge s = S_0$
- $Q_c : |s| = |S_0| \wedge_L (\forall z : \mathbb{Z})(0 \leq z < |s| \rightarrow_L s[z] = S_0[z]^2)$

Dar dos implementaciones distintas que satisfagan la especificación del ciclo con el siguiente invariante:

$$I = (|s| = |S_0| \wedge -1 \leq i \leq |s| - 1) \wedge_L (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] = S_0[j]^2) \wedge (\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[j] = S_0[j])$$

#### Ejercicio 5.

Escribir un programa para el siguiente problema que respete la especificación y el invariante dado

```
proc duplicarElementos (inout s : seq<\mathbb{Z}>) {
 Pre {s = s0 \wedge |s| mod 2 = 0}
 Post {|s| = |s0| \wedge_L
 (\forall i : \mathbb{Z})(enRango(i, s) \longrightarrow_L s[i] = 2 * s0[i])}
```

$$\begin{aligned} I \equiv & (|s| = |s_0| \wedge (0 \leq i \leq |s|/2)) \wedge_L \\ & \text{subseq}(s, 0, |s| - 2 * i) = \text{subseq}(s_0, 0, |s_0| - 2 * i) \wedge \\ & (\forall k : \mathbb{Z})(|s| - 2 * i \leq k < |s| \longrightarrow_L s[k] = 2 * s_0[k])) \end{aligned}$$

#### Ejercicio 6.

Escribir un programa para el siguiente problema que respete la especificación y contenga un ciclo el invariante dado

```
proc dividirPorPromedio (inout s : seq<\mathbf{R}>) {
 Pre {s = s0 \wedge |s| mod 2 = 0 \wedge |s| > 0}
 Post {|s| = |s0| \wedge_L
 (\forall i : \mathbb{Z})(enRango(i, s) \longrightarrow_L s[i] = \frac{s_0[i]}{\text{promedio}(s_0)})}
 aux promedio (s : seq<\mathbf{R}>) : \mathbf{R} = \frac{\sum_{i=0}^{|s|-1} s[i]}{|s|};
```

$$\begin{aligned} I \equiv & (|s| = |s_0| \wedge 0 \leq i \leq \frac{|s|}{2}) \wedge_L \\ & \text{subseq}(s, i, |s| - i) = \text{subseq}(s_0, i, |s_0| - i) \wedge \\ & (\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = \frac{s_0[k]}{\text{promedio}(s_0)}) \wedge \\ & (\forall k : \mathbb{Z})(|s| - i - 1 < k < |s| \longrightarrow_L s[k] = \frac{s_0[k]}{\text{promedio}(s_0)}) \end{aligned}$$

#### Ejercicio 7.

Dar un programa que satisfaga la especificación y tenga un ciclo con el invariante:

```
proc armarPiramide (in v: \mathbb{Z}, inout l: seq<\mathbb{Z}>) {
 Pre {l = L0}
 Post {|L0| = |l| \wedge \text{esPiramide}(l, v)}
 pred esPiramide (l: seq<\mathbb{Z}>, v: \mathbb{Z}) {
```

$$\begin{aligned} & (\forall j : \mathbb{Z})(0 \leq j < |l|/2 \Rightarrow_L l[j] = v + j) \wedge \\ & (\forall j : \mathbb{Z})(|l|/2 \leq j < |l| \Rightarrow_L l[j] = v + |l| - j - 1) \end{aligned}$$

```
}
```

a. **Invariante:**  $|l| = |L_0| \wedge |l|/2 \leq i \leq |l| \wedge_L ((i = |l|/2 \wedge l = L_0) \vee_L (\exists p : \text{seq}(\mathbb{Z}))(\text{esPiramide}(p, v) \wedge |p| = |l| \wedge_L \text{subseq}(p, |l| - i, i) = \text{subseq}(l, |l| - i, i)))$

b. **Invariante:**  $|l| = |L_0| \wedge 0 \leq i \leq |l| \wedge_L \text{piramideHastaI}(l, i, v)$

```
pred piramideHastaI (l: seq<\mathbb{Z}>, i: \mathbb{Z}, v: \mathbb{Z}) {
```

$$(\exists p : \text{seq}(\mathbb{Z}))\text{esPiramide}(p, v) \wedge |p| = |l| \wedge_L \text{subseq}(p, 0, i) = \text{subseq}(l, 0, i)$$

```
}
```

#### Ejercicio 8.

Escribir un programa para el siguiente problema que respete la especificación y el invariante dado

```
proc multiplicar (inout s : seq<\mathbf{R}>) {
 Pre {s = s0 \wedge |s| mod 4 = 0}
 Post {|s| = |s0| \wedge_L
 (\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L s[i] = 10 * s0[i])}
```

$$\begin{aligned} I \equiv & (|s| = |s_0| \wedge \frac{|s|}{2} \leq i \leq |s|) \wedge_L \text{esPar}(i) \wedge \\ & \text{subseq}(s, i, |s|) = \text{subseq}(s_0, i, |s_0|) \wedge \text{subseq}(s, 0, |s| - i) = \text{subseq}(s_0, 0, |s_0| - i) \wedge \\ & (\forall k : \mathbb{Z})(\frac{|s|}{2} \leq k < i \longrightarrow_L s[k] = 10 * s_0[k]) \wedge (\forall k : \mathbb{Z})(|s| - i \leq k < \frac{|s|}{2} \longrightarrow_L s[k] = 10 * s_0[k]) \end{aligned}$$

**Ejercicio 9.**

Escribir un programa para el siguiente problema que respete la especificación y el invariante dado

```
proc cerearYsumar (inout s : seq<Z>, inout suma : Z) {
 Pre {s = s0 ∧ |s| mod 8 = 0}
 Post {|s| = |s0| ∧L ((∀i : Z)(enRango(i, s) →L s[i] = 0) ∧ suma = ∑i=0^|s|-1 s0[i])}
}

I ≡ (|s| = |s0| ∧ 0 ≤ i ≤ |s|/4) ∧L
(subseq(s, 2 * i, |s| - 2 * i) = subseq(s0, 2 * i, |s0| - 2 * i) ∧ (∀k : Z)(0 ≤ k < 2 * i →L s[k] = 0) ∧
(∀k : Z)(|s| - 2 * i ≤ k < |s| →L s[k] = 0) ∧ suma = ∑j=0^2*i-1 s0[j] + ∑j=|s|-2*i^|s|-1 s0[j])
```

## 1. Práctica 7 : Ciclos a partir de Invariantes

### 1. Ejercicio 1

---

```
1 vector<int> copiarSecuencia (vector<int> s) {
2 vector<int> r;
3 int i = 0;
4 while (i < s.size()) {
5 r.push_back(s[i]);
6 i++;
7 }
8 return r;
9 }
```

---

### 2. Ejercicio 2

---

```
1 void incSecuencia(vector<int>& a){
2 int i=0;
3 while (i < a.size()) {
4 a[i]++;
5 i++;
6 }
7 }
```

---

### 3. Ejercicio 3

---

```
1 int cantApariciones(vector<int> s,int e) {
2 int r = 0;
3 for (int i=0;i < s.size();i++){
4 if (s[i] == e) {
5 r++;
6 }
7 }
8 return r;
9 }
```

---

4. Ejercicio 4

a) Primera Implementación

---

```
1 void elevar_cuadrado(vector<int>& s) {
2 int i=-1;
3 while (i+1<s.size()) {
4 i++;
5 s[i] = s[i]*s[i];
6 }
```

---

b) Segunda Implementación

---

```
1 void elevar_cuadrado(vector<int>& s) {
2 for (int i=-1;i+1<s.size();i++) {
3 s[i + 1] *= s[i + 1];
4 }
5 }
```

---

5. Ejercicio 5

---

```
1 void duplicar_elementos(vector<int>& s) {
2 cout << s.size()/2 << endl;
3 for (int i=0;i<(s.size()/2);i++) {
4 s[s.size() -1 - 2*i] *=2;
5 s[s.size() -2 - 2*i] *=2;
6 }
7 }
```

---

6. Ejercicio 6

---

```
1 void dividirPorPromedio(vector<float>& s) {
2 float sumaTotal = accumulate(s.begin(),s.end(),0); //Sumo todos
3 // los elementos del vector
4 float promedio = sumaTotal/s.size();
5 for (int i=0;i<s.size()/2;i++){
6 s[i] /= promedio;
7 s[s.size() -1 - i] /= promedio;
8 }
9 }
```

---

7. Ejercicio 7

a) Invariante a)

---

```
1 void armarPiramide_a(int v, vector<int>& l){
2 for (int i=l.size()/2;i <l.size();i++) {
3 l[i] = v+ l.size() -i -1;
4 l[l.size()-1 - i] = v+ (l.size()-1 - i);
5 }
6 }
```

---

b) Invariante b)

---

```
1 void armarPiramide_b(int v, vector<int>& l) {
2 for (int i=0; i< l.size()/2;i++){
3 l[i] = v+i;
4 }
5 for (int i= l.size()/2; i<l.size();i++){
6 l[i] = v + l.size() -i -1;
7 }
8 }
```

---

8. Ejercicio 8

---

```
1 void multiplicar(vector<int>& s) {
2 for (int i=s.size()/2; i<s.size();i+=2) {
3 s[i] *= 10;
4 s[i+1] *= 10;
5 s[s.size()-1 -i] *= 10;
6 s[s.size()-2-i] *= 10;
7 }
8 }
```

---

9. Ejercicio 9

---

```
1 void cerearYSumar(vector<int>& s, int& suma){
2 suma = 0;
3 for(int i=0; i < s.size()/4;i++) {
4 suma += s[2*i] + s[2*i+1] + s[s.size() -1 -2*i] + s[s.size() -2 -2*i];
5 s[2*i] =0;
6 s[2*i+1] =0;
7 s[s.size() -1 -2*i] =0;
8 s[s.size() -2 -2*i] =0;
9 }
10 }
```

---

# Algoritmos y Estructuras de Datos I

Primer Cuatrimestre 2019

Guía Práctica 8

Tiempo de Ejecución de Peor Caso de un Programa



Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

## Ejercicio 1.

Contar la cantidad de operaciones elementales que realizan los siguientes programas.

```
int ultimo1(vector<int> v) {
 return v[v.size() - 1];
}

int ultimo2(vector<int> v) {
 int i = v.size();
 return v[i - 1];
}

int ultimo3(vector<int> v) {
 int i = 0;
 while (i < v.size()) {
 i++;
 }
 return v[i - 1];
}
```

## Ejercicio 2.

Calcular el tiempo de ejecución de peor caso (en notación  $O$  grande) de los siguientes programas con respecto al tamaño de las secuencias de entrada. Recordar que tanto la lectura como la escritura de un elemento en un vector tiene tiempo de ejecución perteneciente a  $O(1)$ .

```
void f1(vector<int> &vec){
 i = vec.size() / 2;
 while (i >= 0){
 vec[vec.size() / 2 - i] = i;
 vec[vec.size() / 2 + i] = i;
 i--;
 }
}

// pre: |vec| > 20000
void f2(vector<int> &vec){
 i = 0;
 while (i < 10000){
 vec[vec.size() / 2 - i] = i;
 vec[vec.size() / 2 + i] = i;
 i++;
 }
}

// pre: e pertenece a v1
int f3(vector<int> &v1, int e){
 int i = 0;
 while (v1[i] != e){
 i++;
 }
 return i ;
}

void f4(vector<int> &vec){
 int rec = 0;
 int max_iter = 1000;
 if max_iter > vec.size(){
 max_iter = vec.size();
 }

 for(int i=0; i < max_iter; i++){
 for(int j=0; j < max_iter; j++){
 res += vec[i] * vec[j];
 }
 }
}

void f5(vector<int> &v1, vector<int> &v2){
 vector<int> res();
 for(int i=0; i < v1.size(); i++){
 res.push_back(v1[i]); // O(1)
 }
 for(int i=0; i < v2.size(); i++){
 res.push_back(v2[i]); // O(1)
 }
 return res;
}
```

## Ejercicio 3.

Verdadero o falso (justificar).

- a)  $A$  y  $B$  son dos programas que resuelven el mismo problema con un vector  $v$  como única entrada. Para un vector de tamaño 100  $A$  demora 2 minutos en ejecutarse y bajo las mismas condiciones (misma entrada, computadora, recursos, etc)  $B$  demora 20 segundos. Por lo tanto, el programa  $B$  es más eficiente.
- b) Si el tiempo de ejecución de peor caso de dos programas pertenece a  $O(n)$ , entonces el tiempo de cómputo es equivalente (variando un poco según el estado de la computadora en el momento de ejecutar).
- c) Se tienen dos programas cuyo tiempo de ejecución de peor caso perteneciente a  $O(n)$  y a  $O(\log(n))$  respectivamente. Para cualquier entrada con tamaño mayor a 100, el segundo programa demorará menos tiempo en ejecutar.
- d) Se tienen dos programas con tiempo de ejecución de peor caso perteneciente a  $O(n)$  y a  $O(\log(n))$  respectivamente. Para cualquier entrada con tamaño mayor a un cierto número, el segundo programa demorará menos tiempo en ejecutar.
- e) Si se hace un llamado a una función cuyo tiempo de ejecución de peor caso pertenece a  $O(n^2)$  dentro de un ciclo, el tiempo de ejecución de peor del ciclo resultante pertenecerá a  $O(n^3)$ .

#### Ejercicio 4.

```
int mesetaMasLarga(vector<int> &v) {
 int i = 0;
 int maxMeseta = 0;
 int meseta;
 while (i < v.size()) {
 int j = i + 1;
 while (j < v.size() && v[i] == v[j]) {
 j++;
 }
 meseta = j - i;
 i = j;

 if (meseta > maxMeseta) {
 maxMeseta = meseta;
 }
 }
 return maxMeseta;
}
```

- a) ¿Qué hace este programa?
- b) Calcular el tiempo de ejecución de peor caso de este programa en función del tamaño del vector.
- c) ¿Es posible escribir otro programa que resuelva el problema utilizando solo un ciclo?

#### Ejercicio 5.

```
vector<int> hacerAlgo(vector<int> &v) {
 vector<int> res;
 for (int i = 0; i < 100; i++) {
 res.push_back(contarApariciones(v, i+1));
 }

 return res;
}

int contarApariciones(vector<int> &v, int elem) {
 int cantAp = 0;
 for (int i = 0; i < v.size(); i++) {
 if (v[i] == elem) {
 cantAp++;
 }
 }
}
```

Calcular el tiempo de ejecución de peor caso del programa `hacerAlgo` con respecto a  $|v|$ .

```

Ejercicio 6. int sumarPotenciaHasta(int n) {
 int res = 0;
 int i = 1;
 while(i < n) {
 res = res + i;
 i = i * 2;
 }
 return res;
}

```

- a) ¿Qué tiempo de ejecución de peor caso tiene el programa en función del valor  $n$ ?
- b) ¿Es posible calcular la suma de potencias hasta  $n$  con un tiempo de ejecución de peor caso asintóticamente mejor que el programa del punto anterior si la pre condición es True?
- c) ¿Es posible calcular la suma de potencias hasta  $n$  con un tiempo de ejecución de peor caso asintóticamente mejor que el item a si la pre condición es  $(\exists x : \mathbb{Z}) n = 2^x + 1$  ?

### Ejercicio 7.

Una matriz cuadrada se dice *triangular* si todos los elementos por debajo de la diagonal son iguales a cero.

- a) Escribir un programa que calcule el determinante de una matriz triangular. Recordar que el determinante de una matriz triangular es el producto de los elementos de su diagonal.
- b) Escribir un programa que determine si una matriz de  $N \times N$  es o no triangular.
- c) Calcular el tiempo de ejecución de peor caso de los programas:
  - a) en función de la cantidad de elementos de la matriz.
  - b) en función de la cantidad de filas de la matriz.

### Ejercicio 8.

Dadas dos matrices  $A$  y  $B$  se desea multiplicarlas siguiendo esta especificación:

```

proc multiplicar (in m1: seq<seq<\mathbb{Z}\>>, in m2: seq<seq<\mathbb{Z}\>>, out res: seq<seq<\mathbb{Z}\>>) {
 Pre {completar}
 Post {|res| = |m1| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |m1| \rightarrow_L (|res[i]| = |m2[0]| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |m2[0]| \rightarrow_L res[i][j] = \sum_{k=0}^{|m2|-1} m1[i][k] * m2[k][j])))}
}

```

- a) Completar la precondición del problema.
- b) Escribir un programa que retorne  $AB$
- c) Determinar el tiempo de ejecución de peor caso de este programa en función de:
  - a) La cantidad de filas y columnas de cada una de las matrices.
  - b) Suponiendo que  $N = \text{filas}_{m1} = \text{filas}_{m2} = \text{columnas}_{m1} = \text{columnas}_{m2}$

### Ejercicio 9.

Sea  $\text{mat}$  una matriz de  $N \times N$  y la siguiente función:

```

void sumasAcumuladas(vector<vector<int>> &mat, int N){
 for(int i = N - 1; i >= 0, i--) {
 for(int j = N - 1, j >= 0; j--) {
 mat[i][j] = sumHasta(mat, i, j, N);
 }
 }
}

int sumHasta(vector<vector<int>> &mat, int I, int J, int N) {
 int res = 0;

```

```

int lim;
for (int i = 0; i <= I; i++) {
 if (i == I) {
 lim = J
 } else {
 lim = N - 1;
 }
 for (int j <= lim; j++) {
 res += mat[i][j]
 }
}

```

1. Calcular el tiempo de ejecución de peor caso del programa `sumasAcumuladas` en función de la cantidad de elementos de la matriz.
2. Dar un programa que resuelva `sumasAcumuladas` cuyo tiempo de ejecución en peor caso sea  $O(n^2)$

**Ejercicio 10.** Dada una secuencia de  $n$  números enteros, dar un programa que encuentre la máxima cantidad de elementos impares consecutivos cuya tiempo de ejecución de peor caso pertenezca a  $O(n)$ .

**Ejercicio 11.** Escribir un programa que sea correcto respecto de la siguiente especificación, y cuyo tiempo de ejecución de peor caso pertenezca a  $O(|s|)$ .

```

proc restarAcumulado (in s: seq<Z>, in x: Z, out res: seq<Z>) {
 Pre {True}
 Post {|res| = |s| ∧_L (∀i : Z)(0 ≤ i < |s| →_L res[i] = x - ∑_{j=0}^i s[j])}
}

```

**Ejercicio 12.** Sea  $m$  una matriz de  $N \times N$  donde cada posición contiene el costo de recorrer dicha posición (dicho costo es positivo para todas las posiciones). Asumiendo que la única forma de recorrer la matriz es avanzando hacia abajo y hacia la derecha, escribir un programa que calcule el mínimo costo para llegar desde la posición  $(1, 1)$  hasta la posición  $(N, N)$  y cuyo tiempo de ejecución de peor caso pertenezca a  $O(N^2)$ .

**Ejercicio 13.**

Sea  $A$  una matriz cuadrada y  $n$  un número natural.

- a) Escribir un programa que calcule  $\prod_{i=1}^n A$  (es decir  $A^n$ ) (reutilizar el programa del ejercicio 8) ¿Cuál es el tiempo de ejecución de peor caso de esta función?
- b) Resolver el punto anterior suponiendo que  $n = 2^m$  ( $n$  potencia de 2) ¿Se pueden reutilizar cuentas ya realizadas? ¿Cuál es el tiempo de ejecución de peor caso para cada programa?

**Ejercicio 14.**

Dada una matriz de booleanos de  $n$  filas y  $m$  columnas con  $n$  impar. Se sabe que hay exactamente una fila que no está repetida, y el resto se encuentra exactamente dos veces en la matriz.

- a) Escribir un programa que devuelva un vector con los valores de la fila que no se repite. ¿Cuál es su tiempo de ejecución de peor caso descripto ?
- b) ¿Es posible un programa que recorra cada casillero de la matriz sólo una vez? En caso de no haberlo resuelto con esta restricción, modificar el programa para que la cumpla. ¿Cuál es su tiempo de ejecución de peor caso ?
- c) La solución al punto anterior, ¿utiliza vectores auxiliares?, en caso que lo haga, escribir un programa que no los necesite.

## 1. Aclaración importante

La cantidad de operaciones elementales puede variar según los criterios que uno tome. Cómo por ejemplo si tenemos la operación:

$$i = i + 1 \equiv i + +$$

por un lado uno puede tomar que son **2** operaciones, la suma " $i + 1$ " y la asignación " $i = ..$ ". Pero también alguien podría decir que son **3**, si cuenta el acceso a la variable " $i$ " como una operación. O en un caso más extremo tomarlo como una sola operación " $i + +$ ". Por lo tanto no te preocupes si en estas resoluciones aparece  $9 * n$  y vos tenes  $13 * n$  operaciones.

Lo más importante a fin de cuentas es que la complejidad sea la misma ya que ambas  $9 * n$  y  $13 * n$  son  $O(n)$ . Mientras la notación O grande sea correcta entonces no te preocupes por cuánto varían las constantes.

## 2. Práctica 8 : Tiempo de Ejecución de Peor Caso de un Programa

1.
  - a) ultimo1 = 3 operaciones
  - b) ultimo2 = 4 operaciones
  - c) ultimo3 =  $4n+3$  operaciones  
 $\uparrow$   
 $n=s.size()$

2.  $n=s.size()$

- a)  $f_1(n) = 2 + 13n$
  - b)  $f_2(n) = 10000 * 13 + 1$
  - c)  $f_3(n) = 1 + 4n$
  - d)  $f_4(n) = \begin{cases} n * (3 + 8n) + 6 & n < 1000 \\ 1000 * (3 + 8 * 1000) + 4 & n \geq 1000 \end{cases}$
  - e)  $f_5(n1 = v1.size(), n2 = v2.size()) = 1 + 6 * (n1 + n2)$

3. a) Falso, por ejemplo si tenemos :

$f_a(n) = 118 + \log_{10}(n)$  y  $F_b(n) = \sqrt{n} + 10$ , podemos ver que  $f_a(100) > f_b(100)$  pero luego para  $n_0 \geq 1000$   $f_a(n_0) < f_b(n_0)$ . Por lo tanto no es verdadero que B es más eficiente que A.

- b) Falso, no necesariamente pues si tenemos  $f_a(n) = 2^{50000}n$ ,  $f_b(n) = 2n$  entonces  $A, B \in O(n)$  pero A se va a tardar más que el programa B.
  - c) Falso, por ejemplo si tenemos :  
 $f_a(n) = 118 + \log_{10}(n)$  y  $F_b(n) = \sqrt{n} + 10$  no se cumple el enunciado para  $n=101$ .
  - d) Verdadero, puesto que  $O(\log(n)) \in O(n)$ . Esto se puede probar por transitividad usando la definición de  $f \in O(\log(n)) \wedge g \in O(n)$
  - e) Segundo el ciclo. Si el ciclo tiene n iteraciones entonces la afirmación es verdadera. Si las operaciones totales del ciclo pertenecen a  $O(1)$  entonces es falso.

4. a) Revisa un vector busca la longitud de la meseta más larga. Siendo una meseta una subsecuencia de v, de valores consecutivos e iguales.  
 $f(n) = 3 + 16n$
- b) El peor caso es cuando no hay dos elementos iguales consecutivos.

---



```
int mesetaMasLarga (vector <int> & v) {
 2 int maxMeseta =0; //O(1)
 3 int comienzo=0; //O(1)
 4 for (int i=0;i<v.size();i++) { //O(n)
 5 if (v[i] != v[comienzo]) { //O(1)
 6 if (maxMeseta < i-comienzo) { //O(1)
 7 maxMeseta = i-comienzo; //O(1)
 8 }
 9 comienzo = i+1; //O(1)
```

```

10 }
11 }
12 return maxMeseta ;
13 }
```

---

5.  $f(n) = 1 + 100 * [4 + (2 + 7n)]$

6. a)  $f(n) = 2 + 5 * (\log_2(n) + 1)$

b) Falso

c) Partimos de  $n = 2^x + 1$  y  $\sum_{i=0}^{r-1} 2^i \stackrel{\text{sumatoria geométrica}}{\equiv} 2^{r+1-1} - 1$ . Siendo r la cantidad de repeticiones.

Luego si uno prueba para n=2,3,5,9,17, etc te podes fijar que la sumatoria te devuelve  $2n - 3$

Lo cuál sale de saber que  $x = \log_2(n-1)$  Y si x son las repeticiones entonces te queda, por lo visto previamente  $2^{\log_2(n-1)+1} - 1 = 2(n - 1) - 1 = 2 * n - 3$  Por lo tanto basta con hacer un programa así :

```

1 int sumarPotenciasHasta(int n){
2 return 2*n-3; //O(1)
3 }
```

---

## 7. Ejercicio 7

```

a) int determinanteMatrizTriangular(vector<vector<int>> mat){
 int deter= 1; //O(1)
 for (int i=0; i<mat.size();i++) { //O(n)
 deter *= mat[i][i]; //O(1)
 }
```

---

```

b) bool esTriangular(vector<vector<int>> mat) {
 for (int i = 0; i < mat.size(); i++) { //O(n)
 for (int j = 0; j < i; j++) { //O(i)
 if (mat[i][j] != 0) { //O(1)
 return false;
 }
 }
 }
 return true;
}
```

---

c)  $n^2$  =elementos de la matriz      n=cantidad de filas de la matriz

Lo resuelvo con n como parametro, para hacerlo con  $n^2$  habría que reemplazar en la función ese valor.

$$f_a(n) = 8 * n + 2$$

$$f_b(n) = 1 + 4n + \sum_{i=0}^n 7 * i \stackrel{\text{Gauss}}{\equiv} 1 + 4n + 7 * n * (n - 1)/2$$

↑  
Puesto que la cantidad de iteraciones del ciclo varía

8. pred esMatriz (ls : seq<seq< $\mathbb{Z}$ >>) {  

$$(\forall i : \mathbb{Z})( 0 \leq i < |ls| \rightarrow (m[i] \geq 0 \wedge |m[i]| == |m[0]|))$$
  

$$\}$$

- a) Pre {  $(esMatriz(m1) \wedge esMatriz(m2)) \wedge_L |m1[0]| == |m2|$  }  
b) Un **recordatorio** sobre cómo multiplicar matrices  
f1 = Filas de m1, c1=f2= Columnas de m1=Filas de m2, c2=Columnas de m2
- 

```

1 /*Se que si A x B = C -> C_ij (la posición (i,j) de C) =
2 *fila i de A * columna j de B, debido a la especificación del ejercicio.
3 */
4 mat = vector<vector<int>>
5 mat multiplicar(mat m1, mat m2){
6 mat res(m1.filas(), m2.columnas()); //O(f1*c2)
7 for(int i = 0; i < res.filas(); i++){ //O(f1)
8 for(int j = 0; j < res.columnas(); j++){ // O(c2)
9 for(int k = 0; k < m1.columnas(); k++){ //O(c1)
10 res[i][j] = m1[k][i] * m2[j][k]; //O(1)
11 }
12 }
13 }
14 }
15 return res;
16 }
```

---

- c) a)  $Mult(f1, c1 = f2, c2) = f1 * c2 + 1 + f1 * [5 + c2 * (5 + c1 * 10)]$   
f1=c1=f2=c2= n  
b)  $Mult(n) = n^2 + 1 + n * [5 + n * (5 + n * 10)] \in O(n^3)$
9. a)  $SumasAcumuladas(|Mat|, N) = 2 + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \sum_{m=0}^i$  if  $m == i$  then  $j$  else  $N - 1$  fi esto es un estimativo, no es exacto. Lo importante es que  $SumasAcumuladas(|Mat|, N) \in O(N^4)$
- 

b) *//La idea es ir acumulando la suma a medida que voy recorriendo las posiciones*

```

2
3 void sumasAcumuladas(vector<vector<int>>& mat, int N){
4 int suma_acum=0;
5 for (int i=0;i<=N-1;i++){ //O(N)
6 for (int j=0;j<=N-1;j++){ //O(N)
7 suma_acum += mat[i][j];
8 mat[i][j] = suma_acum;
9 }
10 }
11 }
```

---

---

10. Ejercicio 10

```
1 int elementosImparesConsecu(vector<int> v){
2 int contador=0, maximo=0;
3 for (int i=0;i<v.size();i++) { //O(n)
4 if (v[i] % 2 == 1) {
5 contador++;
6 }
7 else {
8 if (contador > maximo) {
9 maximo= contador;
10 }
11 contador =0;
12 }
13 }
14 return maximo;
15 }
```

---

11. Ejercicio 11

```
1 void restarAcumulado(vector<int>& v) {
2 int suma_acumu =0;
3 for (int i=0;i<v.size();i++){ //O(|s|)
4 suma_acumu += v[i];
5 v[i] -= suma_acumu;
6 }
7 }
```

---

12. Ejercicio 12 : [Idea explicada](#)

```
1 int DpcostoMinimo(const vector<vector<int>>& costos) {
2 int N = costos.size();
3 vector<vector<int>> dp(N, vector<int>(N, 0));
4
5 dp[0][0] = costos[0][0];
6 for (int f = 1; f < N; f++) dp[f][0] = dp[f - 1][0] + costos[f][0]; //O(N)
7 for (int c = 1; c < N; c++) dp[0][c] = dp[0][c - 1] + costos[0][c]; //O(N)
8
9 for (int f = 1; f < N; f++) { //O(N)
10 for (int c = 1; c < N; c++) { //O(N)
11 dp[f][c] = min(dp[f - 1][c], dp[f][c - 1]) + costos[f][c];
12 }
13 }
14
15 return dp[N - 1][N - 1];
16 }
```

---

### 13. Ejercicio 13

a) Parte a)

---

```
1 mat potenciaMatriz(mat a, int n) { // O(n * N^3)
2 mat result = a;
3 for(int i = 1; i < n; i++) {
4 result = multiplicar(a, result);
5 }
6 return result;
7 }
```

---

b)  $n=2^m$

---

```
1 mat potenciaMatrizPotDe2(mat a, int n) { // n is POT - O(log(n)*N^3)
2 mat result = a;
3 for(int i = 1; i <= log2(n); i++) {
4 result = multiplicar(result, result);
5 }
6 return result;
7 }
```

---

c) Se cuentan la cantidad de trues en una columna, si el número es impar significa que en esa columna el vector de booleanos distinto tiene un elemento true, si la cantidad es par entonces el elemento es false

---

```
1 vector<bool> boolsDistintosCheto(vector<vector<bool>> mat){
2 vector<bool> distinto = {};
3 for (int i=0; i <mat[0].size();i++){ //O(n)
4 int cant_trues =0;
5 for (int j =0; j<mat.size();j++) {
6 cant_trues += mat[i][j];
7 }
8 distinto.push_back(cant_trues%2);
9 }
10 return distinto;
11 }
```

---

### 3. Bonus Track: N Reinas

**Consigna :** El problema de las N reinas consiste en colocar N reinas en un tablero de N x N sin que se amenacen entre ellas. Las reinas en el ajedrez se pueden atacar horizontalmente, verticalmente y en diagonal.

**Solución**

#### 1. Condiciones a cumplir de las reinas:

Antes que nada, lo único que nos va a importar son las posiciones de las reinas. Las posiciones de cada reina van a estar denotadas como una coordenada  $(i, j)$ . Siendo  $i$ =fila y  $j$ =columna. Notando a la reina número  $n$  en la coordenada  $(i, j)$   $R_{i,j}^n$ . Primero pensemos en que condiciones tienen que cumplir las distintas reinas ubicadas en el tablero para que no se amenacen entre si.

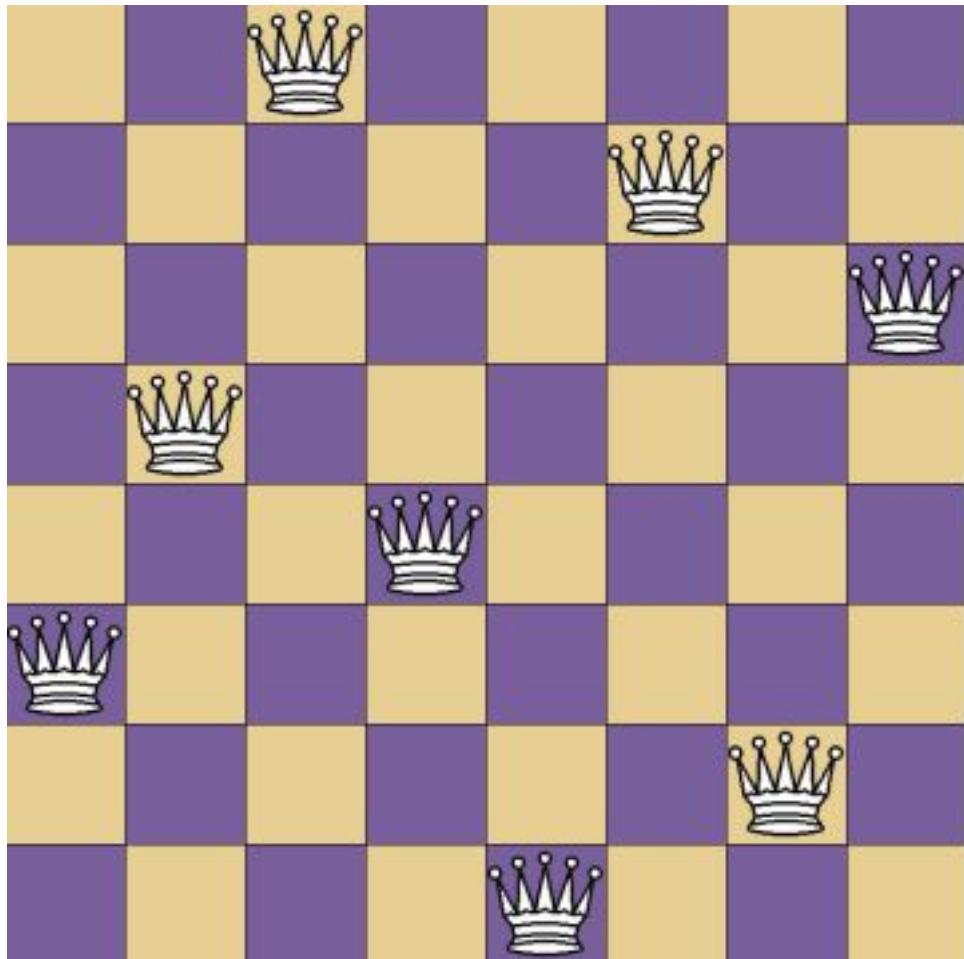


Figura 1: Posible solución del problema

Cómo se puede ver en la imagen hay 3 condiciones que tienen que cumplir las reinas. Por un lado las reinas no tienen que estar ni en la misma fila, ni en la misma columna. Por lo tanto se tiene que cumplir que

$$\forall(m \neq k)(R_{(i_m, j_m)}^m \wedge R_{(i_k, j_k)}^k \wedge (i_k \neq i_m) \wedge (j_k \neq j_m))$$

Y además no pueden estar en la misma diagonal, por lo tanto no puede pasar que la **pendiente** de una recta entre dos reinas sea igual a 1 (diagonal creciente) o -1 (diagonal decreciente). Recordemos que la fórmula de la pendiente es  $m = \frac{\Delta Y}{\Delta X}$ , por lo tanto si queremos  $|m| \neq 1$  es equivalente pedir  $\equiv |\Delta Y| \neq |\Delta X|$ . Lo podemos escribir en el lenguaje de especificación cómo

$$\forall(m \neq k)(R_{(i_m, j_m)}^m \wedge R_{(i_k, j_k)}^k \wedge (|i_k - i_m| \neq |j_k - j_m|))$$

Por lo tanto siempre que ingresemos una reina deberemos comparar que cumpla estas condiciones con todas las reinas anteriores. Para eso tenemos esta función:

---

```
1 bool nuevaReinaValida(vector<pair<int,int>> posiciones_reinas,int colu,int fila) {
2 bool funca = true;
3 for (int j = 0; j < posiciones_reinas.size() - 1; j++) { //Reviso que
4 //sea una reina válida
5 int x = posiciones_reinas[j].first;
6 int y = posiciones_reinas[j].second;
7 if (x == colu || y == fila || abs(x - colu) == abs(y - fila)) {
8 funca = false;
9 break;
10 }
11 }
12 return funca;
13 }
```

---

Esta función se fija si la última reina ingresada a un vector de posiciones de reinas cumple con lo pedido.

## 2. Estrategia para resolverlo:

Una idea sería probar todas las combinaciones posibles, el tema es que el número de las mismas enorme. Puesto que hay  $\frac{!(n^2)}{!(n^2-n)}$  combinaciones posibles, ergo muchísimas.

Por lo tanto la estrategia va a ser otra. La idea sería ir colocando las reinas *una por una*, empezando desde la columna izquierda y viendo en qué fila la podemos insertar. Si colocamos una reina que amenaza a alguna de las anteriores entonces pasamos a agregar a la siguiente reina. Así hasta tener nuestra secuencia completa.

Lo interesante de esta idea es que si llegamos a una configuración de 4 reinas ya colocadas por ejemplo y vemos que no podemos colocar la quinta reina en ninguna columna significa que ese camino no era válido. Por lo tanto no chequea el resto de combinaciones y **backtrackea** a una configuración previa de 3 reinas.

Ahora sí, este es el programa que resuelve el problema.

---

```
1 bool ubicarReinaEnIesimaColumna(vector<pair<int,int>> posis_reinas, int colu, int n)
2 if (colu ==n) { //Si ya coloque todas las reinas devuelvo true
3 cout << "[";
4 for (pair<int,int> t : posis_reinas) { //Imprime la solución en la terminal
5 cout << "(" << t.first << ", " << t.second << ") ";
6 }
7 cout << "] " << endl;
8 //return true; //Aca meramente devuelvo la primera que encuentre
9 return false; //Acá devuelvo todas las soluciones
10 }
11 vector<pair<int,int>> nueva_posi_reinas;
12 bool esValida=false;
13 for (int fila=0;fila<n;fila++) { //Compruebo si hay alguna fila en la
14 //cuál puedo insertar una nueva reina
15 nueva_posi_reinas = posis_reinas;
16 nueva_posi_reinas.push_back(make_pair(colu,fila)); //Agrego la nueva reina
17 esValida = nuevaReinaValida(nueva_posi_reinas,colu,fila);
18
19 if (esValida) { //Si es una posición válida entonces
20 //busco en la siguiente columna
21 if (ubicarReinaEnIesimaColumna(nueva_posi_reinas,colu+1,n)){
22 return true;
23 }
24 }
25 }
26 return false; //Si no hay ninguna configuración válida entonces devuelvo false
27 }
```

---

## 3. Comentarios finales :

- Para usar la función hay que llamarla usando

```
posi_reinas = {}, colu=0, n=Dimensión del Tablero
```

- Gracias Lombi por explicar este bello ejercicio :)

**Ejercicio 1.**

Escribir un programa que dada una matriz cuadrada de enteros y un entero  $x$ , retorne la fila y columna que contiene al elemento  $x$  (o  $\langle -1, -1 \rangle$  si no existe ese elemento). Calcular el tiempo de ejecución de peor caso.

**Ejercicio 2.**

Se tiene una secuencia de  $N - 1$  elementos enteros (con  $N \geq 2$ ) en la cual se encuentran los valores 0 a  $N - 1$  ordenados de manera creciente y sin repetidos, a excepción de un único elemento faltante.

Escribir un programa que tome como parámetro una secuencia como la descripta y encuentre cuál es el elemento faltante. El tiempo de ejecución de peor caso del programa propuesto debe pertenecer a  $O(\log n)$ . Ejemplos: para  $\langle 0, 1, 2, 4, 5, 6, 7 \rangle$  debe devolver 3, mientras que para  $\langle 0 \rangle$  debe devolver 1.

**Ejercicio 3.**

Escribir algoritmos que resuelvan cada uno de los siguientes problemas:

- Dada una secuencia cuyos elementos son todos cero o uno, calcular la suma de los elementos de la secuencia.
- Resolver el mismo problema que en el inciso anterior en tiempo logarítmico si se sabe que la secuencia está ordenada.
- Resolver el mismo problema en tiempo logarítmico si se sabe que la secuencia está ordenada, y que en lugar de cero y uno, los posibles elementos de la secuencia son 15 y 22.

**Ejercicio 4.**

Dada una matriz de números enteros  $mat \in \mathbb{Z}^{N \times M}$  con los elementos de cada fila ordenados de manera creciente y los elementos de cada columna ordenados de manera creciente. Escribir un programa que cuente la cantidad de veces que aparece un elemento dado en la matriz:

- Suponiendo que no hay elementos repetidos en la matriz. El tiempo de ejecución debe pertenecer a  $O(N + M)$
- Suponiendo que puede haber repetidos en las columnas (pero no en las filas). El tiempo de ejecución debe pertenecer a  $O(N + M)$
- Suponiendo que tanto las filas como las columnas pueden tener elementos repetidos. Calcular el tiempo de ejecución de peor caso.

**Ejercicio 5.**

Escribir un programa para resolver los siguientes problemas que toman como entrada una secuencia de enteros  $v$ , en cada caso calcular el tiempo de ejecución de peor caso.

- Contar la cantidad de veces que aparece un número  $e$  en  $v$ .
- Encontrar la diferencia entre  $\max(v)$  y  $\min(v)$
- Encontrar el número que más veces aparece en la secuencia.
- Resolver el item anterior en  $O(|v|)$ , bajo la suposición de que los valores de  $v$  se encuentran acotados en el rango  $[r_1, r_2]$ . ¿Podrían utilizar el algoritmo anterior modificado para  $v$  no acotado? ¿Cuál sería el tiempo de ejecución de peor caso?
- Para cada uno de los incisos anteriores, ¿Cómo podemos aumentar la eficiencia del algoritmo si suponemos  $v$  ordenado como entrada?
- Resolver los tres primeros incisos para la secuencia  $\langle 3, 1, -2, 0, , 2, -2, -2, -2, 3, 10, 0, 4 \rangle$ . Si tenemos que resolver los tres ejercicios de manera consecutiva, ¿Es conveniente ordenar primero la secuencia?

**Ejercicio 6.**

Escribir un programa que devuelva la versión ordenada de un string  $s$  con costo de ejecución perteneciente a  $O(|s|)$ . Por ejemplo, si  $s = \text{"hola Homero!"}$ , el resultado deberá ser  $'!Haehlmooor'$ . Suponer que se cuenta con la función  $ord(c)$  que dado un carácter indica su código en el standard Unicode<sup>1</sup>.

<sup>1</sup>El estandard Unicode 7.0 contiene 112956 caracteres

### Ejercicio 7.

Escribir un programa que, dado una secuencia de enteros  $v$  y dos enteros  $e$  y  $k$  (con  $k \leq |v|$ ), devuelva los  $k$  números más cercanos a  $e$ . En caso de empates, considerar el de menor valor. Calcular el tiempo de ejecución de peor caso.

### Ejercicio 8.

Consideremos el siguiente programa de ordenamiento, llamado ordenamiento por burbujeo (*bubble sort*):

```
int i = 0;
int j;
while(i < a.size()-1) {
 j = 0;
 while (j < a.size()-1) {
 if(a[j] > a[j+1]) {
 swap(a, j, j+1);
 }
 j++;
 }
 i++;
}
```

- a) Describir con palabras qué hace este programa.
- b) Proponer un invariante para el ciclo principal (el más externo).
- c) Proponer un invariante para el ciclo interno.
- d) Calcular el tiempo de ejecución de peor caso.

### Ejercicio 9.

El *bubble sort* puede ser modificado de manera que el burbujeo se realice en ambas direcciones (*cocktail sort*). Una primera pasada para adelante en la lista, y una segunda pasada de regreso. Este patrón alternado se repite hasta que no sea necesario continuar.

- a) Implementar un programa para el algoritmo descripto.
- b) Calcular el tiempo de ejecución de peor caso.
- c) Mostrar paso a paso como el algoritmo ordenaría la siguiente secuencia:  $\langle 4, 2, 1, 5, 0 \rangle$

### Ejercicio 10.

Ordenar las siguientes secuencias utilizando los algoritmos de *ordenamiento por inserción*, *ordenamiento por selección* y *ordenamiento por burbujeo*. Para cada uno, indicar cuál (o cuáles) de los algoritmos utiliza una menor cantidad de operaciones<sup>2</sup> que los demás:

- |                                    |                                              |
|------------------------------------|----------------------------------------------|
| a) $\langle 1, 2, 3, 4, 5 \rangle$ | d) $\langle 1, 1, 1, 2, 2, 2 \rangle$        |
| b) $\langle 5, 4, 3, 2, 1 \rangle$ | e) $\langle 1, 2, 1, 2, 1, 2, 1, 2 \rangle$  |
| c) $\langle 1, 3, 5, 2, 4 \rangle$ | f) $\langle 1, 10, 50, 30, 25, 4, 6 \rangle$ |

**Ejercicio 11. Análisis de algoritmos.** Decidir qué algoritmo de ordenamiento de los vistos en la materia es conveniente utilizar bajo las siguientes suposiciones. ¿Sería útil usar una versión modificada de dichos algoritmos?. Justificar:

- a) Ordenar una secuencia que está ordenada.
- b) Insertar  $k$  elementos en su posición en una secuencia  $v$  ordenada, con  $k$  significativamente más chico que  $|v|$ .
- c) Encontrar los  $k$  elementos más chicos de la secuencia  $v$ , con  $k$  significativamente más chico que  $|v|$ .
- d) Dadas dos secuencias ordenadas, devolver una secuencia que contenga sus elementos ordenados.
- e) Ordenar una secuencia que está ordenada de forma decreciente.

<sup>2</sup>Cuando decimos cantidad de operaciones, nos referimos a cantidad de comparaciones y asignaciones que realiza el algoritmo. En caso de ser una cantidad grande, buscamos una respuesta aproximada con respecto a la longitud de la secuencia.

- f) Encontrar los  $k$  elementos más grandes de una secuencia  $v$ , con  $k$  significativamente más chico que  $|v|$ .
- g) Ordenar una secuencia  $v$  en el que sus elementos están desordenados en a lo sumo  $k$  posiciones, con  $k$  significativamente más chico que  $|v|$ .

**Ejercicio 12.**

Dar un algoritmo que resuelva este problema:

```
proc dosMitades (inout a: seq<Z>) {
 Pre {|a| ≥ 2 ∧ (∃i : Z)(0 ≤ i < |a| ∧ ordenado(subseq(a, 0, i)) ∧ ordenado(subseq(a, i, |a|)))}
 Post {ordenado(a)}
}
```

**Ejercicio 13.**

Escribir un programa que implemente este problema:

```
proc reconstruye (in a: seq<Z>, out b: seq<Z>) {
 Pre {|a| = ∑_{i=0}^{|a|-1} a[i]}
 Post {|a| = |b| ∧ ordenado(b) ∧ (∀i : Z)(0 ≤ i < |b| →_L 0 ≤ b[i] < |a|) ∧ (∀j : Z)(0 ≤ j < |a| →_L a[j] = #apariciones(b, j))}
```

## 1. Búsqueda y Ordenamiento

1. `n=mat.size()`

---

```
1 pair<int,int> posicionElemento(int x,const vector<vector<int>>& mat){
2 pair<int,int> posi = make_pair(-1,-1);
3 for (int i =0;i<mat.size();i++) { //O(n)
4 for (int j=0;j<mat[0].size();j++){ //O(n)
5 if (mat[i][j] == x) posi = make_pair(i,j);
6 }
7 }
8 return posi;
9 }
```

---

$$f(n) = 2 + n * (8 + 8n)$$

2. Ejercicio 2

---

```
1 int elementoFaltante(vector<int> s){
2 int low=0, high=s.size()-1;
3 while (low+1 < high) { //O(log(n))
4 int mid = (low +high)/2;
5 if (s[mid] > mid) high = mid;
6 else low = mid;
7 }
8 return high;
9 }
```

---

3. Ejercicio 3

---

a) `int sumaElementos(vector<int> s){  
 int suma =0;  
 for (int i=0;i<s.size();i++){  
 suma += s[i];  
 }  
 return suma;  
}`

---

- b) Voy revisando el vector para encontrar la posición en la cuál está el primer 1, por lo tanto se que la cantidad de 1's va a ser la longitud de la lista menos la primer posición del 1.
- 

```
1 int sumaElementosOrdenados(vector<int> s){
2 int low=0, high=s.size()-1;
3 while (low+1 < high) { //O(log(n))
4 int mid = (low +high)/2;
5 if (s[mid] == 0) low = mid;
6 else high = mid;
7 }
8 return s.size()- high;
9 }
```

---

- c) Mismo proceso reemplzando 0 y 1, por 15 y 22.
- 

```
1 int sumaElementosOrdenados2(vector<int> s){
2 int low=0, high=s.size()-1;
3 while (low+1 < high) { //O(log(n))
4 int mid = (low +high)/2;
5 if (s[mid] == 15) low = mid;
6 else high = mid;
7 }
8 return (s.size()- high)*22 + high*15;
9 }
```

---

#### 4. Ejercicio 4

- a) Comienza desde la posición (0,N-1) y va bajando / desplazandose hacia la derecha según la posición en la que se encuentre.
- 

```
1 int elementosSinRepetir(vector<vector<int>> m, int e) {
2 int i = 0, j = m[0].size() - 1;
3 while (j >= 0 && i < m.size() && m[i][j] != e) { // O(N + M)
4 if (m[i][j] > e) {
5 j--; // descarto la columna
6 }
7 else {
8 i++; // descarto la fila
9 }
10 }
11 return j >= 0 && i < m.size(); // si terminó en rango, entonces (m[i][j] == e)
```

---

- b) Sigo el mismo razonamiento pero cada vez que encuentro una aparición sigo y la sumo al contador en vez de eliminarla.
- 

```
1 int aparisEColuRepe(vector<vector<int>> m, int e) { // O((N + M)
2 int i = 0, j = m[0].size() - 1;
3 int aparis=0;
4 while (j >= 0 && i < m.size()) { //O(N+M)
5 if(m[i][j] == e) {
6 aparis++; //sumo una aparición
7 i++; // descarto la fila
8 }
9 else if (m[i][j] > e) {
10 j--; // descarto la columna
11 }
12 else {
13 i++; // descarto la fila
14 }
15 }
16 return aparis;
17 }
```

---

- c) Ya que pueden ser todos iguales entonces la complejidad del algoritmo en el peor caso va a ser  $O(N * M)$
- 

```
1 int aparicionesE(vector<vector<int>> m, int e){
2 int aparis =0;
3 for (int i=0;i<m.size();i++){ //O(N)
4 for (int j = 0; j < m[0].size(); ++j) { //O(M)
5 if (m[i][j]) aparis++;
6 }
7 }
8 return aparis;
9 }
```

---

## 5. Ejercicio 5

---

a) `int aparicionesListaE(int e, vector<int> v){  
 int apari=0;  
 for (int i =0;i<v.size();i++){  
 if (v[i] == e) apari++;  
 }  
 return apari;  
}`

---

b) Se va guardando el valor del máximo y mínimo al recorrer el vector

---

```
1 int rangoV(vector<int> v){
2 int maxi=v[0], min = v[0];
3 for (int i=0;i<v.size();i++){
4 if (v[i] < min) min = v[i];
5 if (v[i] > maxi) maxi = v[i];
6 }
7 return maxi-min;
8 }
```

---

c) Hay un vector en el cuál guarda pairs  $\langle \text{elemento}, \text{apariciones - elemento} \rangle$ , a través de las cuales va manteniendo un conteo de cada elemento. Si el elemento no tiene un pair asociado entonces crea un nuevo, si ya estaba en la lista entonces le suma 1 a su cantidad de apariciones. Al final se fija cuál es el mayor.

---

```
1 int elementoMasApariciones(vector<int> s){ //O(n^2)
2 vector<pair<int,int>> aparis = {};
3 for (int i=0;i<s.size();i++){ //O(n)
4 agregarRepetis(aparis, s[i]); //O(|aparis|)
5 }
6 int indi_maxi=0 ;
7 for (int i =0;i<aparis.size();i++){ //O(N)
8 if (aparis[indi_maxi].second < aparis[i].second){
9 indi_maxi =i;
10 }
11 }
12 return aparis[indi_maxi].first;
13 }
14
15 //Agrega la aparición del elemento en cuestión
16 void agregarRepetis(vector<pair<int,int>>& aparis, int ele){
17 bool agregar= true;
18 for (pair<int,int>& apari : aparis){ // //O(|aparis|)
19 if (apari.first == ele){
20 apari = make_pair(apari.first, apari.second+1);
21 agregar = false;
22 break;
23 }
24 }
25 if (agregar){
```

```

26 aparis.push_back(make_pair(ele,1));
27 }
28 }
```

---

- d) Misma idea que en el ejercicio anterior solo que hay un indice para todos los números que esten en el rango de v, por lo tanto su indice representa que elemento es y no hay que tomarse el trabajo de buscarlo. Al final busca la posicion de aparis con más apariciones y devuelve su indice. Es un **counting sort**.
- 

```

1 int elementoMasRepetidoAcotado(vector<int> s, int cota_inf, int cota_sup) {
2 vector<int> aparis(cota_sup-cota_inf, 0);
3 for (int i=0;i<s.size();i++) { //O(n)
4 aparis[s[i]-cota_inf]++;
5 }
6 int indi_maxi=0;
7 for (int i=0;i<aparis.size();i++){ //O(cota_sup-cota_inf)
8 if (aparis[indi_maxi] < aparis[i]){
9 indi_maxi = i;
10 }
11 }
12 return indi_maxi-cota_inf;
13 }
```

---

- e) 1) Para encontrar la cantidad de apariciones de e podemos encontrarlo con búsqueda binaria y luego desde esa aparición contar cuantas hay.  
 2) Agarro el primer y último elemento, luego los resto.  
 3) Recorro la lista una vez y voy contando las apariciones consecutivas de cada elemento, similar al *MesetaMasLarga* de la P8

f) Es conveniente ordenar primero la secuencia ya que nos ahorra operaciones a la larga.

6. La idea es similar al 5)d) ya que el rango de los chars es finito

---

```

1 string ordenarString(string s){
2 vector<int> aparis_char(112956,0);
3 for (char carac : s){ //Cuento las apariciones de cada caracter
4 aparis_char[int(carac)]++;
5 }
6 string string_ordenado = "";
7 for (int i=0;i<aparis_char.size();i++){ //Agrego las apariciones de
8 //cada caracter al nuevo string ordenado
9 for (int j=0;j<aparis_char[i];j++) {
10 string_ordenado += char(i);
11 }
12 }
13 return string_ordenado;
14 }
```

---

7. El programa va guardando los k valores más cercanos en un vector y calcula la distancia de los nuevos valores que agrega al vector con los que ya estaban. El mismo está ordenados de forma creciente en base a su distancia de e, de esta forma uno los va sustituyendo por los nuevos elementos más cercanos a e.

```

1 vector<int> kNumerosMasCercanos(int k, int e, vector<int> v){
2 vector<int> mas_cerca = {};
3 for (int i=0;i < v.size();i++) { //O(n)
4 insertarCercano(k,e,v[i], mas_cerca); //O(k)
5 }
6 return mas_cerca;
7 }
8
9 void insertarCercano(int k, int e, int elem, vector<int>& cercanos){
10 bool agregar = true;
11 if (cercanos.size() < k) {
12 cercanos.push_back(elem);
13 agregar = false;
14 }
15 for (int i=0;i<cercanos.size() && agregar;i++) { //O(k)
16 if (abs(elem -e) < abs(cercanos[i]-e) || (abs(elem -e) == ...
17 abs(cercanos[i]-e) && elem<e)) {
18 //Si es más cercano a "e" que algún elemento
19 cercanos.insert(cercanos.begin()+i, elem);
20 agregar = false;
21 }
22 }
23 if (cercanos.size() > k) { //Se elimina el elemento menos cercano a e
24 cercanos.pop_back();
25 }
26 }
```

8. a) Comienza desde la iésima posición y lo que hace es ir swappeando un elemento con el siguiente si es que es mayor. Si no lo es sigue avanzando y va haciendo estas modificaciones. De esta forma van ordenando los elementos mientras lleva el máximo a la última posición no ordenada.

- b)  $\prod_{i=0}^{|s|-1} \exists s_i \in s \wedge \text{mismos}(s, S_0) \wedge \text{ordenado}(\text{subseq}(s, |s| - 1 - i, |s|))$
- c)  $\prod_{j=0}^{|s|-1} \exists s_j \in s \wedge \text{mismos}(s, S_0) \wedge \text{subseq}(s, j, |s|) == \text{subseq}(S_0, j, |s|) \wedge (\forall k : \mathbb{Z})(0 \leq k < j \rightarrow s[k] \leq s[j])$
- d) El peor caso sería que este ordenado de forma decreciente.  
 $f(n) = 1 + n * (6 + 11n) \in O(n^2)$

9. *Cocktail Sort*

---

```
a) void cocktailSort(vector<int>& s){
 2 bool ordenada = false;
 3 for (int i=0;i<s.size() && !ordenada;i++) {
 4 ordenada =true;
 5 for (int j=i;j<s.size()-1;j++) {
 6 if (s[j] > s[j+1]) {
 7 swap(s[j], s[j+1]);
 8 ordenada = false;
 9 }
 10 }
 11 for (int k=s.size()-2-i; k>0 && !ordenada;k--) {
 12 if (s[k-1] > s[k]){
 13 swap(s[k], s[k-1]);
 14 ordenada=false;
 15 }
 16 }
 17 }
 18}
```

---

10. Vector-operaciones = $\langle$ operaciones – insertion, operaciones – selection, operaciones – bubble $\rangle$   
El código con el que se contabilizaron estas operaciones está en el repo general.

- a)  $\langle 1, 2, 3, 4, 5 \rangle \rightarrow \langle 20, 145, 181 \rangle$
  - b)  $\langle 5, 4, 3, 2, 1 \rangle \rightarrow \langle 150, 145, 221 \rangle$
  - c)  $\langle 1, 3, 5, 2, 4 \rangle \rightarrow \langle 59, 145, 193 \rangle$
  - d)  $\langle 1, 1, 1, 2, 2, 2 \rangle \rightarrow \langle 24, 195, 271 \rangle$
  - e)  $\langle 1, 2, 1, 2, 1, 2, 1, 2 \rangle \rightarrow \langle 110, 316, 529 \rangle$
  - f)  $\langle 1, 10, 50, 30, 25, 4, 6 \rangle \rightarrow \langle 171, 252, 423 \rangle$
11.
  - a) El insertion sort, ya que hace una comparación por cada elemento
  - b) De vuelta la elección sería insertion sort ya que insertaría los elementos donde van.
  - c) Aquí se puede usar una variación de selection sort para agarrar los mínimos.
  - d) Con un sorted merge sale de una, ergo apareando las dos secuencias.
  - e) Con un cocktail sort, uno va "llevando"los elementos a su posición deseada.
  - f) Primero recorrer la lista e ir agarrando los elementos que estan desordenados en un vector. Luego ordenarlo con cualquiera de los algoritmos vistos. Y volver a insertar los mismos elementos pero ahora ordenados.
  - g) Aquí se puede usar una variación de selection sort para agarrar los máximos.

---

12. Ejercicio 12

---

```
1 void dosMitades(vector<int>& s) {
2 int i=0;
3 while (s[i] < s[i+1] && i<s.size()) i++;
4 int sec_b = i+1,sec_a =0;
5 vector<int> fusion_ordenada = {};
6 while (sec_b < s.size() || sec_a < sec_b){
7 if (sec_b == s.size() || s[sec_b] > s[sec_a]) {
8 fusion_ordenada.push_back(s[sec_a]);
9 sec_a++;
10 }
11 else if(sec_a == sec_b || s[sec_b] < s[sec_a]){
12 fusion_ordenada.push_back(s[sec_b]);
13 sec_b++;
14 }
15 }
16 s = fusion_ordenada;
17 }
```

---

13. Ejercicio 13

Le falta agregar a la Pre  $(\forall i : \mathbb{Z})(0 \leq i < |a| \rightarrow a[i] \geq 0)$  sino es imposible de cumplir la post.

---

```
1 vector<int> reconstruye(vector<int> a) {
2 vector<int> aparis(0, a.size());
3 for (int i = 0; i < a.size(); i++) {
4 aparis[a[i]] += i;
5 }
6 vector<int> b = {};
7 for (int i = 0; i < aparis.size(); i++) {
8 for (int j = 0; j < aparis[i]; j++) {
9 b.push_back(i);
10 }
11 }
12 return b;
13 }
14 }
```

---

## 2. Bonus Track: ¿Porque no usamos siempre counting sort?

Cuándo nos introducen los distintos de ordenamientos podemos ver que unos cuántos tienen una función de complejidad  $\in O(n^2)$ , y más adelante se ven algoritmos de complejidad  $O(n*\log(n))$  inclusive. Pero hay un algoritmo que tiene complejidad  $O(n)$  , **counting sort** , la pregunta entonces es. ¿Porque no usamos siempre counting sort?

Pensemos en un ejemplo, tenemos el código del ejercicio 6 por ejemplo :

---

```
1 string ordenarString(string s){
2 vector<int> aparis_char(112956,0);
3 for (char carac : s){ //O(N)
4 aparis_char[int(carac)]++;
5 }
6 string string_ordenado = "";
7 for (int i=0;i<aparis_char.size();i++){ //O(112956)
8 for (int j=0;j<aparis_char[i];j++) {
9 string_ordenado += char(i);
10 }
11 }
12 return string_ordenado;
13 }
```

---

$c = \text{constante}$

Si tenemos que ordenar 5 strings de 1.000.000 caracteres entonces este algoritmo es muy útil ya que debemos revisar meramente una vez cada string. Ahora pensemos. ¿Que sucede si tenemos que revisar 1.000.000 strings de 5 caracteres?

Por cada uno de esos strings de 5 caracteres va a entrar en el segundo ciclo de la función y hacer **112956 \* c** operaciones. Por lo tanto es más eficiente hacer un algoritmo de  $O(n^2)$  ya que  $n=5$ . Podemos ver que depende el caso puede ser mejor usar **counting sort** u otro algoritmo.

Por lo tanto para los casos en los que  $n < \text{cota}$  podemos decir que hace  $\text{cota} * \text{operaciones}$ .

O si tenemos un vector  $\langle 2^{32}, 5, 7, -2, 34, 5, -2^{32}, 8 \rangle$  y quiero ordenarlo con **counting sort** la cantidad de operaciones sería de  $8 * c_1 + 2 * 2^{32} * c_2$  que es enorme a comparación de otros sorts que podrían hacerlo en  $8^2 * c$  operaciones. Cómo conclusión, no siempre es mejor usar counting sort.