Ejercicio 1

(a)

El invariante de representación correcto es el segundo.
$Rep(e : estr) \equiv (|e.titulo| > 0) \land (|e.autor| > 0) \land (e.paginas\_totales > 0) \land (0 < e.pagina\_actual \leq e.paginas\_totales)$

(b)

Lo cumple:

- $e_1 \equiv$ ("Ana Karenina", León Tólstoi, 330 páginas, página actual 144)

No lo cumple:

- $e_2 \equiv$ ("", J.K. Rowling, 155 páginas, página actual 177)

(c)

Yo agregaría lo siguiente:
$0 \leq e.pagina\_actual \leq e.paginas\_totales$

Ejercicio 3

(a)

`Fecha`:

- Observadores: `dia(), mes(), anio()`;
  Constructores: `Fecha(dia, mes, anio)`;
  Modificadores: `avanzar_dia(), avanzar_n_dias(int n)`;
  Otras operaciones: `operator==(const Fecha & f) const`
- Cada observador devuelve justamente lo que indica su nombre.
- Constructores: como precondición se deben tener los tres parámetros positivos. `dia` debe estar entre 1 y 31 (asumiendo que todos los meses tienen 31 días), `mes` debe estar entre 1 y 12. Como postcondición se creará un objeto de tipo `Fecha`.
  Modificadores: no hay precondición para el modificador `avanzar_dia()`, y su postcondición será que la variable interna `_dia` se incrementará en uno. Para el modificador `avanzar_n_dias(int n)` no existe precondición y se tiene como postcondición

que `_dia` aumentará en `n`.
Otras operaciones: devolverá `True` sólo si son iguales las variables internas `_dia, _mes, _anio` con otra fecha.

- Se deben tener los tres parámetros positivos. `dia` debe estar entre 1 y 31 (asumiendo que todos los meses tienen 31 días), `mes` debe estar entre 1 y 12.

(b)

`Usuario`:

- Observadores: `nombre(), edad()`;
  Constructores: `Usuario(string nombre, int edad)`;
  Modificadores: `agregar_amigo(string nombre)`;
  Otras operaciones: `es_amigo(string nombre), es_popular()`

- Cada observador devuelve justamente lo que indica su nombre.

- Constructores: como postcondición creará un objeto de tipo `Usuario`.
  Modificadores: como postcondición se agrega un nuevo amigo a un Usuario.
  Otras operaciones: `es_amigo(string nombre)` será verdadero sólo si `nombre` está en el conjunto de amigos. `es_popular()` será verdadero si tiene más de 10 amigos.

- La variable interna `nombre` debe tener un tamaño mayor a cero y la variable interna `edad` también debe ser positiva.

(c)

`Multiconjunto`:

- Observadores: `cardinal(), contar_apariciones(int e), cantidad_elementos_distintos()`;
  Constructores: `Multiconjunto()`;
  Modificadores: `agregar(int e), quitar(int e)`

- Cada observador devuelve justamente lo que indica su nombre.

- Constructores: como postcondición se crea un multiconjunto vacío.
  Modificadores: se agrega o se quita un elemento ya presente en el multiconjunto. El tamaño va a variar según la operación que se haga.

- La variable interna `_cantidad_distintos` debe ser mayor o igual a cero.

Ejercicio 4

(a)

I) El carrito puede soportar desde 0kg hasta infinito. No tiene sentido que el peso sea negativo.
II) Los ítems presentes en el carrito no estarán repetidos.

Es una pelotudez esto, ¿por qué no podrías llevar más de un elemento? Podría ser que lleven dos paquetes de arroz. BOLUDOS!

III) Todos los productos que están en `pesos_items` estarán también en `precios_por_item`.
IV) Todos los productos que están en `items` estarán representados como claves de `pesos_items`.
V) De manera inversa, si hay un producto como clave en `pesos_items`, entonces estará también en `items`.
VI) Si un producto está en `pesos_items`, entonces su peso asociado será mayor a cero.
VII) Si un producto está en `precios_por_item`, tendrá un precio asociado mayor a cero.
VIII) El peso de todos los productos sumados debe ser menor o igual a `peso_maximo`.

(b)

I) Los pesos no siempre están representados en números enteros, pueden ser valores racionales también.
II) Está mal esto porque la lista de los productos puede ser cero, pero lógicamente no puede ser negativo.
III) *Eh? Cómo que no?*

(c)

Necesitan precondiciones los métodos:

- `peso_de_item(string nombre)`, ya que `nombre` debe estar presente en `items`.
- `borrar_item(string nombre)`, ya que `nombre` debe estar presente en `items`.
- Lo mismo aplica para `agregar_item(string nombre)`.

Ejercicio 5

(a)

- Pertenece a la clase `Usuario`.

- El tamaño de `nombre` debe ser mayor a cero, `edad` y `cantidad_amigos` debe ser mayor a cero. Además, `cantidad_amigos` es igual al cardinal de `amigos`.
- *Paja hacerlo. Otro día, maestro.*
- *Paja hacerlo. Otro día, maestro.*

(b)

- Pertenece a la clase `Fecha`.
- `dia_del_anio` debe estar contenido entre 1 y 364. `anio` debe ser un número mayor a cero.
- *Paja hacerlo. Otro día, maestro.*
- *Paja hacerlo. Otro día, maestro.*

(c)

- Pertenece a la clase `Carrito`.
- `peso_maximo` y `peso_total` deben ser mayores a cero. Los valores asociados en los maps también deben ser positivos.
- *Paja hacerlo. Otro día, maestro.*
- *Paja hacerlo. Otro día, maestro.*

(d)

- Pertenece a la clase `Multiconjunto`.
- Los valores asociados en el map debe ser positivo.
- *Paja hacerlo. Otro día, maestro.*
- *Paja hacerlo. Otro día, maestro.*

Ejercicio 6

(a)

No lo cumple.

(b)

Sí lo cumple.

(c)

No lo cumple. Hay que chequear si fue ingresado previamente al sistema o no.

(d)

Sí lo cumple. De todas formas, no hice un análisis muy exhaustivo.

Ejercicio 8

- No es necesario, ya que las dos variables de la interfaz privada son strings.
- No es necesario que los dos sean del mismo tamaño.
- Los observadores de `C` son necesarios de ser incluidos, porque se pueden llegar a necesitar precondiciones.
- No es necesario.
- No es necesario.

Ejercicio 9

(a)

Deben ser todas letras. No valen números ni caracteres vacíos.

(c)

Es $O(N^2)$ en peor caso: recorre todas las palabras, y además recorre todas las letras.

(d)

No. Debería modificar el private.

(e)

Debería haber una variable en la estructura que tenga la cantidad de vocales de cada palabra, y que, cada vez que se añada una palabra nueva, se registre.