Lenguajes de Programación, 2023-1
Práctica 2

Facultad de Ciencias, UNAM

Profesora: Karla Ramírez Pulido
Ayud. Lab.: Manuel Ignacio Castillo López

Fecha de inicio: Miércoles 31 de agosto de 2022
Fecha de entrega: Miércoles 14 de septiembre 2022

Descripción

La práctica consiste en implementar las funciones aquí descritas en un módulo de Racket (archivo con extensión *.rkt). Puede definir todas las funciones auxiliares que considere necesarias (salvo que se indique explícitamente lo contrario). Ponga atención a las restricciones y descripción de cada ejercicio.

Cada definición en Racket debe incluir un breve comentario que describa su propósito, en particular las precondiciones y postcondiciones para las que fue diseñada.

Puede usar el material de clase, laboratorio, manuales y guías en línea o apoyarse con el ayudante de laboratorio para resolver los ejercicios. Puede realizarla de manera individual o en equipos de 2 ó 3 integrantes; se les recomienda desarrollar la práctica en equipo. Pueden solicitarnos ayuda para encontrar algún equipo.

Su archivo con extensión *.rkt deberá incluir los nombres completos y números de cuenta de los integrantes del equipo que entregan la práctica.

Ejercicios

1. (1 pto). Defina un tipo de datos abstracto **`Figura`** que sea utilizado para trabajar con figuras geométricas, debe tener los siguientes constructores:

- Un constructor `(tirangulo a b c)` donde `a`, `b` y `c` son números reales que representan la longitud de los lados de un triángulo.
- Un constructor `(rectangulo a b)` donde `a` y `b` son números reales que representan la altura y base de un rectángulo.
- Un constructor `(rombo l D d)` donde `l`, `D` y `d` son números reales que representan el lado, diagonal mayor y diagonal menor de un rombo, respectivamente.
- Un constructor `(paralelogramo a b h)` donde `a`, `b` y `h` son números reales; donde `a` y `b` representan los lados de un paralelogramo y `h` su altura.

- Un constructor `(elipse a b)` donde `a` y `b` son números reales y representan el semieje mayor y el semieje menor de la elipse, respectivamente.

2. Utilizando el tipo de datos `Figura`, defina las siguientes funciones:

- (1.5 pts). La función `(perimetro f)` que calcula el perímetro de una `Figura` dada.
  `;; perimetro: Figura → number`
  - Precondiciones: una instancia de `Figura`.
  - Postcondiciones: el perímetro de la `Figura` dada. Recuerde que el perímetro es la suma de la longitud de los lados de una `Figura`.
    Ejemplos:

```
1 > (perimetro (triangulo 2 3 4))
2 9
3 > (perimetro (elipse 3 3))
4 18.84955592153876
```

- (1.5 pts). La función `(area f)` que calcula el área de una `Figura` dada.
  `;; area: Figura → number`
  - Precondiciones: una instancia de `Figura`.
  - Postcondiciones: el área de la `Figura` dada. Recuerde que el área es la extensión de la superficie contenida dentro de un espacio geométrico; en este caso, la `Figura`.
    Ejemplos:

```
1 > (area (rectangulo 2 3))
2 6
3 > (area (rombo 3 4 5))
4 10
```

3. Considere una clase de tren de pasajeros conformado por los siguientes tipos de vagones:

- Locomotora. Posee una potencia de arrastre máxima, que consideraremos un entero que indica el número de vagones no-locomotora que puede mover.
- Vagón de pasajeros. Posee una capacidad máxima de pasajeros que pueden abordarlo.
- Vagón restaurante. Posee un número de mesas y personal de servicio máximo.
- Vagón dormitorio. Posee un número fijo de camas.

(Descripción ✪): un tren puede tener una o varias locomotoras a los extremos; debe al menos haber una y deben ser consecutivas (salvo que estén en extremos opuestos del convoy). Entre locomotoras (o tras una única locomotora), pueden encontrarse otros tipos de vagones en cualquier acomodo y número.

Defina los siguientes tipos de datos:

- (1 pts). El tipo de datos `Vagon`, que debe incluir 4 constructores; uno por cada tipo de vagón descrito anteriormente:

    * Constructor `(locomotora p)`, donde `p` es un número entero positivo que representa el número de vagones que puede mover la locomotora.
    * Constructor `(pasajeros cap)`, donde `cap` es un número entero positivo que representa el número máximo de personas que pueden abordar el vagón.
    * Constructor `(restaurante mesas personal)`, donde `mesas` y `personal` son números enteros positivos que representan respectivamente el número de mesas y personal de servicio máximo.
    * Constructor `(dormitorio camas)`, donde `camas` es un número entero positivo que representa el número de camas en el vagón.

- (1 pto). El tipo de datos `Tren`, que modela trenes conforme a las condiciones descritas anteriormente (vea la descripción ✪).

4. Utilizando el tipo de datos `Tren`, defina las siguientes funciones:

- (1 pto). La función `(num-pasajeros tren)` que calcula el número de pasajeros máximo que pueden abordar el `tren`.
  `;; num-pasajeros: Tren → positive-integer`
    * Precondiciones: un `tren` que satisface las condiciones establecidas en el ejercicio anterior.
    * Postcondiciones: la suma de las capacidades máximas de sus `vagones de pasajeros`.
      Ejemplos:

```
> (num-pasajeros (tren-f (tren-f (tren-f (tren-loc (locomotora 1))
                                         (pasajeros 10)) (restaurante 5 2))
                         (dormitorio 10)))
10
> (num-pasajeros (tren-f (tren-f (tren-loc (locomotora 1))
                               (pasajeros 10)) (pasajeros 20)))
30
```

- (1 pto). La función `(arrastre-usado tren)` que calcula el porcentaje de la potencia de arrastre utilizada del `tren`. Debe regresar valores mayores a 100 si la capacidad de arrastre de las `locomotoras` no es suficiente para mover el resto de los vagones (y este resultado debe ser la proporción de arrastre que excede la capacidad de las locomotoras).
  `;; arrastre-usado: Tren → number`
    * Precondiciones: un `tren` que satisface las condiciones establecidas en el ejercicio anterior.
    * Postcondiciones: la proporción del total de la capacidad de arrastre de todas las locomotoras respecto al número de `vagones no-locomotoras` como porcentaje numérico.
      Ejemplos:

```
> (arrastre-usado (tren-f (tren-f (tren-f (tren-loc (locomotora 1))
                                          (pasajeros 10)) (restaurante 5 2))
                          (dormitorio 10)))
300
> (arrastre-usado (tren (tren-f (tren-loc (locomotora 1))
                              (pasajeros 10)) (pasajeros 20)
                        (tren-loc (locomotora 1))))
100
```

- (1 pto). La función `(sin-cama tren)` que calcula el número de pasajeros que quedarían sin cama durante; de acuerdo al total de pasajeros y camas en el `tren`. Considere que las camas son individuales.
  `;; sin-cama: Tren → nonnegative-integer`

  * Precondiciones: un `tren` que satisface las condiciones establecidas en el ejercicio anterior.
  * Postcondiciones: el número máximo de pasajeros que excede la capacidad del total de los `vagones dormitorio`.
    Ejemplos:

```
> (sin-cama (tren-f (tren-f (tren-f (tren-loc (locomotora 1))
                                    (pasajeros 10)) (restaurante 5 2))
                    (dormitorio 10)))
0
> (sin-cama (tren-f (tren-f (tren-loc (locomotora 1)) (pasajeros 10))
                    (pasajeros 20)))
30
```

- (1 pto). La función `(max-comensales tren)` que determina el número máximo de pasajeros que pueden ser atendidos al mismo tiempo en los `vagones restaurante` del `tren`. Considere que una mesa puede albergar 4 pasajeros y un personal de servicio puede atender a 8 pasajeros. Tome en cuenta que la capacidad de las mesas y del personal se limitan entre sí: la capacidad del personal puede exceder la capacidad de las mesas, pero no es posible servir a un pasajero sin mesa; como puede haber mesas suficientes para todos los pasajeros pero puede que no se cuente con personal suficiente para atenderlos a todos.
  `;; max-comensales: Tren → nonnegative-integer`

  * Precondiciones: un `tren` que satisface las condiciones establecidas en el ejercicio anterior.
  * Postcondiciones: el máximo de pasajeros que pueden ser atendidos en un `vagón restaurante`; que está limitado por su número de mesas o de personal de servicio.
    Ejemplos:

```
> (max-comensales (tren-f (tren-f (tren-f (tren-loc (locomotora 1))
                                          (pasajeros 10)) (restaurante 5 2))
                          (dormitorio 10)))
16
> (max-comensales (tren-f (tren-f (tren-loc (locomotora 5)) (pasajeros 20))
                          (restaurante 10 5)))
40
```

5. (1.5 pts. Extra) Defina funciones que realicen una prueba unitaria de cada uno de los ejercicios en la práctica; es decir, para obtener los puntos extra deberá definir 14 funciones de prueba:

- 10 funciones de prueba para el tipo de datos `Figura`; uno para la función `perimetro` y otro para `area`; una prueba de cada función por cada constructor de `Figura`.
- 4 funciones de prueba para los tipos de datos `Vagon` y `Tren`; probando las funciones `num-pasajeros`, `arrastre-usado`, `sin-cama` y `max-comensales`. Cada prueba debe usar instancias no-equivalentes del tipo `Tren`.

Utilice la función incluida en el lenguaje `plai test` para implementar las pruebas.

Ejemplos:

```
> (prueba-area)
good (area (elipse 3 3)) at line 146
     expected: 18.84955592153876
     given: 18.84955592153876
>
> (prueba-arrastre-usado)
good (arrastre-usado (tren-f (tren-f (tren-f (tren-loc (locomotora 1))
                                             (pasajeros 10)) (restaurante 5 2))
                             (dormitorio 10))) at line 152
     expected: 300
     given: 300
```

Requerimientos

Deberá entregar su práctica a través de la plataforma **Google Classroom** antes que termine el día 14 de septiembre; tal y como lo indican los lineamientos de entrega. Deberá adjuntar su código en un archivo llamado **practica02.rkt**. Revise bien mayúsculas, minúsculas y espacios para el nombre de archivo que se pide. El enlace de Classroom es el siguiente: `https://classroom.google.com/c/NTI1OTc5ODU1NzI5?cjc=2nbnbrh`

Deberá indicar en los comentarios solicitados para sus funciones, el número del ejercicio que resuelven y su descripción debe resumir lo que solicita el ejercicio. Las funciones auxiliares deben esclarecer en sus comentarios que son funciones auxiliares y en qué ejercicios se utilizan.

Recuerden que estamos atentos a cualquier duda, ¡suerte!