Lenguajes de Programación

Práctica 2 - Tipos de datos abstractos

Semestre 2017-1

Fecha de inicio: 25 de agosto de 2016

Fecha de término: 14 de septiembre de 2016

Instrucciones

- Resolver los siguientes ejercicios de manera clara y ordenada en un archivo `practica2.rkt`.
- Todas las funciones deben ir acompañadas de al menos 2 pruebas unitarias en un archivo por separado `test.rkt`.
- Para ejecutar las pruebas unitarias agregar al inicio la línea `(require ‘‘practica2.rkt’’)`.
- La entrega es por equipos de máximo 3 integrantes. Seguir los lineamientos especificados en: `http://sites.ciencias.unam.mx/ldp171/formato`.
- Para cada ejercicio, se recomienda usar la técnica de cazamiento de patrones.
- Los puntos extra son individuales y sólo se puede escoger uno. El ensayo debe de realizarse **a mano y entregarse a más tardar el 14 de septiembre en la hora de laboratorio**. Los puntos extra de programación se envían por correo en un archivo `<no.cuenta>_P02.rkt` con asunto `[LDP-EXP2]`. Incluir el nombre completo del autor en el cuerpo del correo.

Ejercicios

1. **Figuras geométricas**

   Definir un tipo de dato abstracto `Figura` que sea utilizado para trabajar con figuras geométricas.

   El tipo de dato abstracto debe contener:

   - Un constructor `(triangulo a b c)` donde `a`, `b` y `c` son números reales y representan los lados del triángulo.
   - Un constructor `(cuadrado a)` donde `a` es un número real y representa el lado del cuadrado.
   - Un constructor `(rectangulo a b)` donde `a` y `b` son números reales y representan la altura y base del rectángulo.
   - Un constructor `(rombo a D d)` donde `a`, `D` y `d` son números reales y representan el lado, diagonal mayor y diagonal menor del rombo respectivamente.
   - Un constructor `(paralelogramo a b h)` donde `a`, `b` y `h` son números reales y representan los lados y altura del paralelogramo respectivamente.
   - Un constructor `(circulo D)` donde D es un número real y representa el diámetro del círculo.

- 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.

Una vez definido el tipo de dato, se deben de definir las siguientes funciones:

a) Una función `(perimetro fig)` que dada una figura regrese el perímetro de ésta.

b) Una función `(area fig)` que dada una figura calcule el área de ésta.

Algunos ejemplos de uso del tipo de dato abstracto:

```
> (define figura (triangulo 17 29 20))
> figura
(triangulo 17 29 20)
> (perimetro figura)
66
> (area figura)
165.698
```

2. **Funciones simples**

Definir un tipo de dato abstracto `Funcion`[1]para trabajar con funciones simples.

El tipo de dato abstracto debe contener:

- Un constructor `(x)` que representa una variable independiente x. Nótese que no recibe parámetros.
- Un constructor `(cte n)` para representar constantes como 2 o 1729, `n` es un número entero.
- Un constructor `(sum f g)` que representa una suma de funciones como $x + 2$, `dónde f` y `g` son funciones.
- Un constructor `(mul f g)` para representar el producto de funciones como $3x$, talque `f` y `g` son funciones.
- Un constructor `(pot b n)` para representar una función elevada a una potencia n como $3x^2$, donde `b` es una función y `n` representa un entero.

Una vez definido el tipo de dato, definir las siguientes funciones:

a) Una función `(Funcion->string fun)` que regresa una cadena de caracteres representando a la función que recibe como parámetro.

b) Una función `(evalua fun v)` que devuelve la la función evaluada en v, es decir $f(v)$.

c) Una función `(deriva fun)` que regresa la derivada de una función.

Algunos ejemplos de uso del tipo de dato abstracto:

[1]Se hace referencia al concepto de función en el sentido matemático del Cálculo Diferencial y no debe confundirse con el concepto de función de Lenguajes de Programación.

```
> (define f (mul (cte 2) (x)))
> f
(mul (cte 2) (x))
> (Funcion->string f)
``(2*x)''
> (evalua f 1729)
(mul (cte 2) (cte 1729))
> (deriva f)
(sum (mul (cte 2) (cte 1)) (mul (x) (cte 0)))
```

3. **Pilas y Colas**

Dada la siguiente definición de nodos simples:

```
(define-type Nodo
   [vacio]
   [nodo (elemento any?) (siguiente Nodo?)])
```

Define los siguientes tipos de datos abstractos para trabajar con pilas y colas:

El tipo `Pila` debe contener:

- Un constructor `(pila n)` para representar una `Pila`, dónde `n` es una sucesión de nodos.
- Un constructor `(mete-p e p)` el cual representa la operación que agrega un elemento `e` en la `Pila p`. El elemento se agrega al final de la sucesión de nodos.
- Un constructor `(saca-p p)` para representar la operación de eliminar un elemento de la `Pila p`. El elemento a eliminar es el último en la sucesión de nodos.
- Un constructor `(mira-p p)` el cual representa la operación que muestra el elemento encima de la `Pila p`. El elemento a mostrar es el último en la sucesión de nodos.

El tipo `Cola` debe contener:

- Un constructor `(cola n)` para representar una `Cola`, dónde `n` es una sucesión de nodos.
- Un constructor `(mete-c e c)` para representar la operación de agregar un elemento `e` en la `Cola c`. El elemento se agrega al final de la sucesión de nodos.
- Un constructor `(saca-c c)` el cual representa la operación de eliminación de un elemento de la `Cola c`. El elemento a eliminar es el primero en la sucesión de nodos.
- Un constructor `(mira-c c)` para representar la operación que muestra el primer elemento de la `Cola c`. El elemento a mostrar es el primero en la sucesión de nodos.

Una vez definidos los tipos de datos, se debe de definir una función `(calc-p pila)` que evalúe expresiones del tipo `Pila` y una función `(calc-c cola)` que evalúe expresiones del tipo `Cola`.

Algunos ejemplos de uso del tipo de dato abstracto:

```
> (define p (pila (nodo 1 (nodo 2 (nodo 3 (vacio))))))
> (calc-p p)
(pila (nodo 1 (nodo 2 (nodo 3 (vacio)))))
> (calc-p (mete-p 4 p))
(pila (nodo 1 (nodo 2 (nodo 3 (nodo 4 (vacio))))))
> (calc-p (saca-p p))
(pila (nodo 1 (nodo 2 (vacio))))
> (calc-p (mira-p p))
4

> (define c (cola (nodo 1 (nodo 2 (nodo 3 (vacio))))))
> (calc-c c)
(cola (nodo 1 (nodo 2 (nodo 3 (vacio)))))
> (calc-c (mete-c 4 c))
(cola (nodo 1 (nodo 2 (nodo 3 (nodo 4 (vacio))))))
> (calc-c (saca-c c))
(cola (nodo 2 (nodo 3 (vacio))))
> (calc-c (mira-c c))
1
```

4. **Conjuntos**

   Definir un tipo de dato abstracto `Conjunto` para trabajar con conjuntos.

   Su tipo de dato abstracto debe contener:

   - Un constructor `(conjunto l)` para representar un `Conjunto`, donde `l` es una lista.
   - Un constructor `(esvacio? c)` el cual representa el predicado que indica si el `Conjunto` `c` está vacío.
   - Un constructor `(contiene? c e)` para representar el predicado que indica si el `Conjunto` `c` contiene al elemento `e`.
   - Un constructor `(agrega c e)` el cual representa la operación para agregar el elemento `e` en el `Conjunto` `c`.
   - Un constructor `(union c1 c2)` para representar la operación de unión entre dos conjuntos `c1` y `c2`.
   - Un constructor `(interseccion c1 c2)` el cual representa la operación de intersección entre dos conjuntos `c1` y `c2`.
   - Un constructor `(diferencia c1 c2)` para representar la operación de diferencia entre los conjuntos `c1` y `c2`.

   Una vez definido el tipo de dato, deben definir una función `(calc-cjto cjto)` que evalúe expresiones del tipo `Conjunto`. Se recomienda seguir la técnica de cazamiento de patrones.

   Algunos ejemplos de uso del tipo de dato abstracto:

```
> (define a (conjunto '(1 7 2 9)))
> (define b (conjunto '(1 2 3 4))
> (calc-cjto (conjunto '(1 1 7 2 9))
(conjunto '(1 7 2 9))
> (calc-cjto (esvacio? a))
#f
> (calc-cjto (contiene? a 1))
#t
> (calc-cjto (agrega a 9))
(conjunto '(1 7 2 9))
> (calc-cjto (union a b))
(conjunto '(1 7 2 9 3 4))
> (calc-cjto (interseccion a b))
(conjunto '(1 2))
```

Puntos extra

- (2 pts.) Investiga quién es Bárbara Liskov e indica sus contribuciones al diseño de lenguajes de programación, principalmente relacionados con la abstracción de datos. Escribe un ensayo de mínimo una cuartilla donde describas quién es Barbara Liskov, qué es la abstracción, qué utilidad tiene la abstracción en las ciencias de la computación, qué utilidad tiene la abstracción en los lenguajes de programación y ejemplifica con aplicaciones. Tu ensayo debe contener introducción, desarrollo, conclusiones, bibliografía y debe estar debidamente citado.

- (1 pto.) Con base en el ejercicio 2 de esta práctica, escribe una función `(interpreta f v)` que interprete la evaluación de una función. Por ejemplo:

```
> (define f (sum (cte 4) (x)))
> (evalua f 1729)
(sum (cte 4) (cte 1729))
> (interpreta (evalua f 1729))
1733
```

- (1 pto.) Modifica el ejercicio 3 de esta práctica para que muestre los resultados de acuerdo al siguiente formato: `[a b c d]`, donde `a`, `b`, `c` y `d` son los elementos de los nodos. No debes modificar las definiciones de tipos sino las funciones `calc-p` y `calc-c`. Por ejemplo:

```
> (define p (pila (nodo 1 (nodo 2 (nodo 3 (vacio))))))
> (calc-p p)
[1 2 3]
> (calc-p (mete-p 4 p))
[1 2 3 4]
> (calc-p (saca-p p))
[1 2]
> (calc-p (mira-p p))
3
```