Proyecto:

Simulador de Lenguaje Racket

Descripción General:

Este proyecto fue desarrollado pensando en aprender a duplicar el comportamiento de lenguajes de la rama de Lisp, más especifico Racket. Imitando un aproximado de la sintaxis de varias de sus funciones más básicas, almacenamiento de variables, almacenamiento de procedimientos, entre otros.

Las tecnologías principales empleadas para el desarrollo fueron Java, como lenguaje en el que corren los procesos necesarios para imitar el comportamiento; DrRacket, como interprete oficial para aprender la sintaxis y comportamiento original de los procedimientos y datos del lenguaje Racket; y Chatgpt como herramienta para agilizar la creación de las pruebas para asegurar los comportamientos esperados.

Retos del Desarrollo:

Para la comprensión del proyecto, tuve que tomar múltiples sesiones de investigación acerca de Racket, el mayor problema fue organizar las estructuras necesarias para las Expresiones, pues pese a que Racket solo se divide en “QExpression” y “SExpression”, hay que entender las mismas distinciones que tienen dentro de las QExpression, dado que una QExpression numérica no es igual a una QExpression de pares; además de la dificultad en poder hacer distinciones entre símbolos que evocan procedimientos y símbolos de QExpression, este reto fue solucionado con clases Expression Builder que toman arreglos de texto y los procesan en distintos niveles para realizar los reemplazos necesarios.

Descripción de las funciones simuladas:

Símbolo/Proc	Datos Aceptados	Datos Retornados	Ejemplo
AND	#t, #f	#t, #f	(AND #t #f) • #f (AND #t #t) • #t
OR	#t, #f	#t, #f	(OR #t #f) • #t (OR #f #f) • #f
NOT	#t, #f	#t, #f	(NOT #f) • #t
#t		#t	(#t) • #t
#f		#f	(#f) • #f
*	Numéricos	Numéricos	(* 1 2 3) • 6
/	Numéricos	Numéricos	(/ 10 5) • 2
%	Numéricos	Numéricos	(% 10 5) • 0
+	Numéricos	Numéricos	(+ 3 4 5) •
-	Numéricos	Numéricos	(- 10 4 8) • -2
=	Numéricos	#t, #f	(= 1 2) • #f (= 1 1) • #t
>	Numéricos	#t, #f	(> 2 1) • #t (> 1 1) • #f
>=	Numéricos	#t, #f	(>= 1 1) • #t (>= 1 2) • #f
<	Numéricos	#t, #f	(< 1 2) • #t (< 1 1) • #f

<=	Numéricos	#t, #f	(<= 1 2) • #t (<= 2 1) • #f
quote	Numéricos, símbolos	Numéricos, QExpression	(quote a) • ‘a (quote 2) • 2 (quote (+ 1 2)) • ‘(+ 1 2) ‘a • ‘a
list	Numéricos, símbolos	QExpression pares	(list 1 2 3) • ‘(1 2 3) (list (list 1 2) 3) • ‘(‘(1 2) 3)
car	QExpression pares	QExpression, QExpression pares	(car (list 1 2 3)) • 1 (car (list (list 1) 2)) • ‘(1)
cdr	QExpression pares	QExpression, QExpression pares, Empty	(cdr (list 1 2 3)) • ‘(2 3) (cdr (list (list 1) 2)) • 2 (cdr (list 1)) • ‘()
apply	Proc + QExpression pares	Numéricos, QExpression, QExpression pares, Empty	(apply + (list 1 2 3)) • 6 (apply list (list 1 2)) • ‘(‘(1 2))
eval	QExpression	Numéricos, QExpression, QExpression pares, Empty	(eval ‘(+ 1 2)) • 3 (eval (quote (NOT #t))) • #f
cond	#t, #f	Numéricos, QExpression, QExpression pares, Empty, #t, #f	(cond ((> 1 2) (quote Mayor)) (else (quote Menor))) • ‘Menor

define	Proc + SExpression, QExpression	Proc	(define A 1000) (A) • 1000 (define fun (+ A B)) (fun 1) • 1001

Consideraciones:

- AND, OR y NOT en Racket tienen un comportamiento que considero inesperado al utilizar QExpression numéricas, por ende, solo es permitido expresiones que evalúen a #t y #f.
- Para los define, cualquier función creada que tenga variables, las variables deben ser preferentemente nombradas distintas a procedimientos ya existentes.
- Aunque ya considero el proyecto terminado, estoy interesado en añadir más funciones de Racket como la lambda que es de hecho un eje importante en dicho lenguaje, a la vez que incluir una interfaz gráfica que permita cargar documentos con codigo interpretable para el simulador.

Agradecimiento especial al profesor Diego Munguía Molina.