# Proyecto final Construcci´on interpretador ObliQ

Yenifer Ronaldo Muñoz Valencia -2278665

Michael Steven Rodriguez Arana - 2266193

Juan Carlos Rojas Quintero - 2359358

Fundamentos De Interpretación y Compilación De Lenguajes De Programación

Carlos Andres Delgado

Universidad del Valle

Sede Tuluá

Facultad de Ingeniería

Tuluá - Valle

2024

# TABLA DE CONTENIDO

1. **Introducción**
   1.1 Descripción general del proyecto
   1.2 Objetivos del intérprete

2. **Descripción del Problema**
   2.1 Motivación
   2.2 Objetivos específicos del proyecto

3. **Descripción de la Solución**
   3.1 Enfoque general
   3.2 Características del intérprete

4. **Diseño de la Solución**
   4.1 Base en la gramática de Obliq
   4.2 Funciones principales del intérprete
   4.3 Gestión de estados y ambientes

5. **Implementación de la Solución**
   5.1 Descripción de los módulos implementados

6. **Pruebas Realizadas**
   6.1 Estrategias de prueba
   6.2 Casos de prueba
   6.3 Resultados obtenidos

7. **Conclusiones**
   7.1 Logros alcanzados
   7.2 Impacto en el aprendizaje

8. **Limitaciones**
   8.1 Funcionalidades no implementadas
   8.2 Aspectos a mejorar

9. **Posibles Mejoras**
   9.1 Implementación de manejo de excepciones
   9.2 Soporte para concurrencia
   9.3 Mejora del análisis y manejo de errores
   9.4 Sistema de tipos más robusto

# Introducción

El presente informe describe el desarrollo de un intérprete para el lenguaje de programación Obliq, como proyecto final del curso de Fundamentos de Interpretación y Compilación de Lenguajes de Programación. Obliq es un lenguaje orientado a objetos con un enfoque en la concurrencia y la distribución. Sin embargo, este intérprete se centra en las características básicas del lenguaje, sin abarcar la concurrencia ni la distribución.

## Descripción del Problema

El objetivo principal del proyecto es implementar un intérprete que permita ejecutar programas Obliq básicos, incluyendo la evaluación de expresiones, la gestión de objetos, el manejo de procedimientos y la ejecución de primitivas. Este proyecto nos permite profundizar en la comprensión de los conceptos fundamentales de la interpretación de lenguajes, implementar un intérprete funcional, y analizar y manejar estructuras sintácticas y semánticas de un lenguaje de programación.

## Descripción de la Solución

La solución propuesta consiste en un intérprete desarrollado en Racket que procesa y ejecuta programas escritos en Obliq. El intérprete maneja las construcciones básicas del

lenguaje, como expresiones primitivas, condicionales, iteración, definición y aplicación de procedimientos, y manejo básico de objetos y métodos.

**Características del Intérprete:**

- **Léxico y Sintáctico:** Implementa la especificación léxica y gramatical de Obliq.
- **Sistema de Tipos:** Define los datatypes para representar los elementos del lenguaje.
- **Ambientes:** Implementa un sistema de ambientes para el manejo de variables y procedimientos.
- **Evaluación:** Evalúa expresiones Obliq, incluyendo expresiones aritméticas, booleanas, de control de flujo y secuenciación de comandos.
- **Objetos:** Implementa el manejo básico de objetos, incluyendo la creación de objetos, acceso y actualización de campos.
- **Primitivas:** Define e implementa primitivas del lenguaje, como operaciones aritméticas, comparaciones, y operaciones sobre cadenas.
- **Procedimientos:** Maneja procedimientos y clausuras.
- **Recursión:** Implementa un ambiente extendido recursivo para el manejo de procedimientos recursivos.

## Diseño de la Solución

El diseño del intérprete se basa en la gramática de Obliq, construyendo un sistema que interpreta y ejecuta las instrucciones. Se crearon funciones para manejar las primitivas, implementar estructuras de control y gestionar estados a través de ambientes. El sistema maneja múltiples tipos de datos y estructuras de control, asegurando la corrección y robustez del intérprete.

## Implementación de la Solución

La implementación se realizó en Racket. Se desarrollaron módulos para manejar aspectos específicos de Obliq:

- Primitivas numéricas y de cadenas.
- Evaluación de expresiones booleanas y condicionales.
- Gestión de secuencias y procedimientos.
- Creación y manipulación básica de objetos.
- Manejo de variables y asignaciones.
- Implementación de ciclos.

Se implementó un manejo de ambientes que permite la extensibilidad para soportar el alcance de variables y procedimientos, y el manejo de variables actualizables.

# Pruebas Realizadas

**Pruebas para primitivas:**

```
;; Pruebas para primitivas
(define exp1 (scan&parse "+(2,3)"))
(define expected-exp1 5)
(check-equal? (evaluar-programa exp1) expected-exp1)

(define exp2 (scan&parse "-(5,2)"))
(define expected-exp2 3)
(check-equal? (evaluar-programa exp2) expected-exp2)

(define exp3 (scan&parse "*(2,3)"))
(define expected-exp3 6)
(check-equal? (evaluar-programa exp3) expected-exp3)

(define exp4 (scan&parse "/(10,2)"))
(define expected-exp4 5)
(check-equal? (evaluar-programa exp4) expected-exp4)

(define exp5 (scan&parse "%(10,3)"))
(define expected-exp5 1)
(check-equal? (evaluar-programa exp5) expected-exp5)

(define exp6 (scan&parse "&(\"Hola\",\" mundo\")"))
(define expected-exp6 "Hola mundo")
(check-equal? (evaluar-programa exp6) expected-exp6)
```

**Pruebas para bool-expr**

```
;; Pruebas para expresiones booleanas
(define exp7 (scan&parse ">(5,2)"))
(define expected-exp7 #t)
(check-equal? (evaluar-programa exp7) expected-exp7)

(define exp8 (scan&parse "<(2,5)"))
(define expected-exp8 #t)
(check-equal? (evaluar-programa exp8) expected-exp8)

(define exp9 (scan&parse ">=(5,5)"))
(define expected-exp9 #t)
(check-equal? (evaluar-programa exp9) expected-exp9)

(define exp10 (scan&parse "<=(5,5)"))
(define expected-exp10 #t)
(check-equal? (evaluar-programa exp10) expected-exp10)

(define exp11 (scan&parse "is(5,5)"))
(define expected-exp11 #t)
(check-equal? (evaluar-programa exp11) expected-exp11)

(define exp12 (scan&parse "and(true,false)"))
(define expected-exp12 #f)
(check-equal? (evaluar-programa exp12) expected-exp12)

(define exp13 (scan&parse "or(true,false)"))
(define expected-exp13 #t)
(check-equal? (evaluar-programa exp13) expected-exp13)

(define exp14 (scan&parse "not(true)"))
(define expected-exp14 #f)
(check-equal? (evaluar-programa exp14) expected-exp14)
```

### Prueba de los condicionales

```
;; Pruebas para control de flujo (if)
(define exp15 (scan&parse "if(>(5,2),then 10,else 20,end)"))
(define expected-exp15 10)
(check-equal? (evaluar-programa exp15) expected-exp15)

(define exp16 (scan&parse "if(<(2,5),then 10,elseif(>(2,5),then 20,else 30,end),else 30,end)"))
(define expected-exp16 10)
(check-equal? (evaluar-programa exp16) expected-exp16)

(define exp17 (scan&parse "if(<(2,5),then 10,elseif(<(2,5),then 20,else 30,end),else 30,end)"))
(define expected-exp17 10)
(check-equal? (evaluar-programa exp17) expected-exp17)

(define exp18 (scan&parse "if(false,then 10,elseif(false,then 20,else 30,end),else 30,end)"))
(define expected-exp18 30)
(check-equal? (evaluar-programa exp18) expected-exp18)
```

### Pruebas var, let y letrec

```
;; Pruebas para var, let y letrec
(define exp19 (scan&parse "var x = 10, y = 20 in +(x,y) end"))
(define expected-exp19 30)
(check-equal? (evaluar-programa exp19) expected-exp19)

(define exp20 (scan&parse "let x = 10, y = 20 in +(x,y) end"))
(define expected-exp20 30)
(check-equal? (evaluar-programa exp20) expected-exp20)

(define exp21 (scan&parse "letrec f(x) = if(is(x,0),then 1,else *(x,(f(-(x,1)))),end) in (f 5) end"))
(define expected-exp21 120)
(check-equal? (evaluar-programa exp21) expected-exp21)
```

### Pruebas set y begin

```
;; Pruebas para set y begin
(define exp22 (scan&parse "var x = 10 in begin set x := 20, x end end"))
(define expected-exp22 20)
(check-equal? (evaluar-programa exp22) expected-exp22)

(define exp23 (scan&parse "begin 1,2,3 end"))
(define expected-exp23 3)
(check-equal? (evaluar-programa exp23) expected-exp23)
```

### Pruebas procedimientos

```
;; Pruebas para procedimientos (proc y apply)
(define exp24 (scan&parse "proc(x,y) +(x,y) end"))
(define expected-exp24 (closure '(x y) '(+ x y) (ambiente-vacio)))
(check-equal? (evaluar-programa exp24) expected-exp24)

(define exp25 (scan&parse "apply(proc(x,y) +(x,y) end,(1,2))"))
(define expected-exp25 3)
(check-equal? (evaluar-programa exp25) expected-exp25)

;; Pruebas para métodos (meth)
(define exp26 (scan&parse "meth(self,x) +(self.a,x) end"))
(define expected-exp26 (closure '(self x) '(+ self.a x) (ambiente-vacio)))
(check-equal? (evaluar-programa exp26) expected-exp26)
```

### Pruebas para métodos

```
;; Pruebas para métodos (meth)
(define exp26 (scan&parse "meth(self,x) +(self.a,x) end"))
(define expected-exp26 (closure '(self x) '(+ self.a x) (ambiente-vacio)))
(check-equal? (evaluar-programa exp26) expected-exp26)
```

### Pruebas Objetos

```
;; Pruebas para objetos (object, get, send, update)
(define exp27 (scan&parse "object {}"))
(define expected-exp27 (object-empty))
(check-equal? (evaluar-programa exp27) expected-exp27)

(define exp28 (scan&parse "object {a => 10, b => 20}"))
(define expected-exp28 (object-atributes 'a 10 (object-atributes 'b 20 (object-empty))))
(check-equal? (evaluar-programa exp28) expected-exp28)

(define exp29 (scan&parse "var obj = (object {a => 10}) in (get obj.a) end"))
(define expected-exp29 10)
(check-equal? (evaluar-programa exp29) expected-exp29)

(define exp30 (scan&parse "var obj = (object {a => 10, m => (meth(self,x) +(self.a,x) end)}) in (send obj.m,(5)) end"))
(define expected-exp30 15)
(check-equal? (evaluar-programa exp30) expected-exp30)

(define exp31 (scan&parse "var obj = (object {a => 10}) in begin update obj.a := 20, (get obj.a) end end"))
(define expected-exp31 20)
(check-equal? (evaluar-programa exp31) expected-exp31)
```

### Pruebas clone

```
;; Pruebas para clone
(define exp32 (scan&parse "var obj1 = (object {a => 10}) in var obj2 = (clone (obj1)) in (get obj2.a) end end"))
(define expected-exp32 10)
(check-equal? (evaluar-programa exp32) expected-exp32)
```

**Prueba Ciclo For**

```
;; Pruebas para ciclos (for)
(define exp33 (scan&parse "for i = 1 to 5 do +(i,i) end"))
(define expected-exp33 10)
(check-equal? (evaluar-programa exp33) expected-exp33)
```

## Conclusiones

La implementación del intérprete de Obliq permitió un entendimiento profundo de las estructuras de programación y de los mecanismos necesarios para interpretar un lenguaje. La experiencia en la manipulación de ambientes y en la interpretación de estructuras como objetos y procedimientos es invaluable para el desarrollo de habilidades en programación. El proyecto sienta las bases para el desarrollo de sistemas más complejos en el futuro.

## Limitaciones

- El manejo de errores puede ser mejorado.

## Posibles Mejoras

- Implementar manejo de excepciones.
- Mejorar el análisis de errores.