# Proyecto final - Fundamentos de Lenguajes Programación

Robinson Duque, Ph.D
robinson.duque@correounivalle.edu.co

Enero de 2022

## 1. Introducción

El presente proyecto tiene por objeto enfrentar a los estudiantes del curso:

- a la comprensión de todos los conceptos vistos en clase
- a la implementación de tres versiones de interpretadores de un lenguaje de programación
- al análisis de estructuras sintácticas, de datos y de control de un lenguaje de programación para la implementación de los interpretadores

**Importante:** Tenga en cuenta que las descripciones presentadas en este documento son bastante generales y solo pretenden dar un contexto que le permita a cada estudiante entender el proyecto y las características del lenguaje que debe desarrollar. Así mismo, se proponen algunas construcciones sintácticas, es posible que se requiera modificar o adaptar la sintaxis para cumplir con el requerimiento.

## 2. (50pts) Lenguaje UVProg

UVProg es un lenguaje de programación (no tipado) con ciertas características de lenguajes de programación declarativos e imperativos () . Se propone para este proyecto que usted implemente un lenguaje de programación interpretado para lo cual usted es libre de proponer una sintaxis inspirandose en uno o varios lenguajes de programación existentes (e.g., C, C++, Java, Python, R, Scala, Octave, etc). La semantica del lenguaje estará determinada por las especificaciones en este proyecto.

### 2.1. Sintaxis Gramatical

- El desarrollo del proyecto y la participación de los integrantes debe ser rastreable y verificable en todo momento (no podrán aparecer entregas de la nada o cambios completos del interpretador sin evidenciar un desarrollo contínuo). Para esto, cada grupo deberá crear un **repositorio privado** en **Github** con el nombre (UVProg-#grupo e invitar al usuario **robinsonduque**). Recuerde incluir la información de los integrantes en el archivo Readme al igual que información relevante del proyecto).

  La gramática **debe ser socializada con el profesor** durante las primeras dos semanas después de la asignación de este proyecto (bastará con un correo para revisar el repositorio). Dentro de los siguientes 2 a 4 días hábiles después del envío recibirá una respuesta con observaciones.

- **La gramática debe estar documentada con comentarios indicando el lenguaje en el cual se han inspirado para especificar cada regla de producción**.
- La gramática deberá contener ejemplos de cada producción utilizando llamados a `scan&parse`. Es decir, deberá contener ejemplos de cómo se crean las variables, procedimientos, invocación a procedimientos, etc.
- Todo interpretador cuya gramática esté sin aprobar por el profesor (o que no sea rastreable a través del tiempo), tendrá una nota de sustentación de 0.

Usted debe implementar un interpretador para darle la semántica que se describe en los numerales subsecuentes.

### 2.2. Valores:

- **Valores denotados:** enteros, flotantes, numeros en base 32, hexadecimales, octales, cadenas de caracteres, booleanos (true, false), procedimientos, listas.
- **Valores expresados:** enteros, flotantes, números en base 32, hexadecimales, octales, cadenas de caracteres, booleanos (true, false), procedimientos, listas.

**Aclaración:** Los números en una base distinta de 10, deberán representarse así: [x32 0 23 12], [x16 4 1 0 7 14], [x8 2 1 4 0 7], teniendo en cuenta que el primer elemento de la lista indica la base del número y el resto de la lista puede utilizar la representación BIGNUM vista en clase.

**Sugerencia:** trabaje los valores enteros, flotantes desde la especificación léxica. Implemente los números en base 32, hexadecimales, octales, cadenas de caracteres, booleanos (true, false) y procedimientos desde la especificación gramatical.

### 2.3. Características del Lenguaje UV-Prog

El lenguaje debe permitir utilizar:

- **Identificadores:** Son secuencias de caracteres alfanuméricos que comienzan con una letra o un simbolo específico dependiendo de las características que desee implementar en su lenguaje
- **Definiciones:** Este lenguaje permitirá crear distintas definiciones:
  - **Constantes:** introduce una colección de identificadores no actualizables y sus valores iniciales. Una constante es de asignación unica y debe ser declarada con un valor inicial, por ejemplo: `Constante y = 9;`. Para este caso `y` no podrá ser modificada durante la ejecución de un programa.
  - **Variables de asignación unica:** introduce una colección de identificadores actualizables una única vez. Una variable de asignación unica puede ser declarada así: `Val z = 9, x = @UVProgVAL;`. Para este caso, `z` no podrá ser modificada durante la ejecución de un programa puesto que ya ha sido asignada. Sin embargo, `x` podrá ser asignada una única vez a algún valor del lenguaje `UVProg`. Por ejemplo: `x ->29;`. En caso que se utilice la variable `x` sin estar ligada, el interpretador deberá lanzar un error.
  - **Variables de múltiple asignación (mutables)**: introduce una colección de variables actualizables y sus valores iniciales. El lenguaje deberá distinguir entre constantes, variables mutables y variables asignación única. Una variable mutable puede ser modificada cuantas veces se desee. Una variable mutable puede ser declarada así: `Var a = 5, b = @UVProgVAL;`. En ambos casos, ambas variables podrán ser modificadas durante la ejecución de un programa. Por ejemplo: `a ->9;` o `b->true;`
- **Condicionales:** Son estructuras para controlar el flujo de un programa
- **Secuenciación:** el lenguaje deberá permitir expresiones para la creación de bloques de instrucciones
- **Expresiones:** las estructuras sintácticas son una expresión. Algunas expresiones producen un valor, otras producen un efecto (ejemplo, las expresiones relacionadas con asignación)
- **Primitivas booleanas:** $<, >, <=, >=, ==, != , ==, and, or, not$. Estas primitivas son binarias (exceptuando el *not*, que es unaria) y permiten evaluar expresiones para generar un valor booleano
- **Primitivas aritméticas para enteros:** $+, -, *, \%, /, add1, sub1$
- **Primitivas aritméticas para hexadecimales, octales y base 32:** $+, -, *, add1, sub1$
- **Primitivas sobre cadenas:** longitud, concatenar
- **Primitivas sobre listas:** se deben crear primitivas que simulen el comportamiento de: `empty?`, `empty`, `cons`, `list?`, `car`, `cdr`, `append`
- **Definición/invocación de procedimientos:** el lenguaje debe permitir la creación/invocación de procedimientos que retornan un valor al ser invocados. El paso de parámetros será por valor y por referencia, el lenguaje deberá permitir identificar de alguna manera la forma como se enviarán los argumentos.
- **Definición/invocación de procedimientos recursivos:** el lenguaje debe permitir la creación/invocación de procedimientos que pueden invocarse recursivamente. . El paso de parámetros será **por valor y por referencia**, el lenguaje deberá permitir identificar de alguna manera la forma como se enviarán los argumentos.
- ¿Cómo se evalúa? El lenguaje creado debe ser lo suficientemente expresivo para poder implementar las funciones del Taller 1 del curso de FLP.

## 3. (25pts) UVProg+

Extienda el lenguaje UVProg y añada:

- **Iteración:** el lenguaje debe permitir la defición de una estrutura de repetición tipo `for`. Por ejemplo: `for x =` $a_1$ `to` $a_2$ `do` $a_3$ `end`. Se sugiere agregar funcionalidad al lenguaje para que permita "imprimir" resultados por salida estándar tipo `print`.
- **Primitivas sobre vectores:** se debe extender el lenguaje y agregar manejo de vectores. Se deben crear primitivas que simulen el comportamiento de: `vector?`, `make-vector`, `vector-ref`, `vector-set!`.
- **Tipos de datos**: el lenguaje debe permitir manejo de tipos de datos. El lenguaje debe inferencia o chequeo de tipos. Se deben definir reglas para:

- todas las primitivas (aritméticas para enteros, flotantes, octales, hexadecimales, cadenas, listas, vectores)
- condicionales (if, cond)
- invocación de procedimientos
- iteración

Los tipos de datos a incluir son:

- Int
- Float
- Bool
- Oct
- Hex
- B32
- String
- List
- Vector
- Function

De esta manera, deberá extender su lenguaje para escribir programas tipados por el estilo de:

```
val(
  Int x = 5,
  Float y = 3.4,
  Oct w = [x8 1,3,0],
  List p = [3;7;1;9]
)in
var (
 Function f = proc(Int a, Int b)return (a*b),
 Int y = 3
) in
    sequence
      x -> (y+5);
      f(x, (y+3));
    end
```

# 4. (25pts) UVProg++

Extienda el lenguaje UVProg (sin manejo de chequeo de tipos), y añada:

- **Declaración de clases y métodos**
- **Creación de objetos:** simples o planos, cualquier implementación es válida
- **Invocación de métodos y selección de campos:** el lenguaje debe permitir invocar métodos asociados a objetos y obtener los valores asociados a los campos
- **Actualización de campos:** el lenguaje debe permitir actualizar los campos asociados a un objeto
- El lenguaje debe incluir conceptos como: herencia, llamados a métodos de la superclase y polimorfismo.

# 5. Evaluación UVProg, UVProg+, UVProg++

El proyecto podrá ser realizado en grupos de hasta 5 personas utilizando la librería SLLGEN de Dr Racket. Este debe ser sustentado y cada persona del grupo obtendrá una nota entre 0 y 1 (por sustentación), la cual se multiplicará por la nota obtenida en el proyecto.

Sólo la especificación léxica y gramatical del lenguaje base UVProg, deberá ser aprobado por el profesor. Los interpretadores UVProg+ y UVProg++ deben ser variaciones simples del lenguaje base. Dado el caso que un grupo se presente con lenguajes UVProg+ y UVProg++ ajenos a la gramática definida para el interpretador UVProg, obtendrán una nota máxima de 0.2 en la sustentación asociada con dichos interpretadores.