# Lenguajes de Programación
# Proyecto Final - 2025-1

Manuel Soto Romero  Demian Alejandro Monterrubio Acosta  Erik Rangél Limón
José Alejandro Pérez Márquez  Dicter Tadeo García Rosas

**Entrega (tentativa)**: 22 de noviembre de 2024
Facultad de Ciencias UNAM

## Objetivos

- Implementar recursión en MiniLisp mediante el uso de combinadores de punto fijo (como el combinador Y), permitiendo que las funciones puedan llamarse a sí mismas sin necesidad de un nombre explícito.
- Ampliar el sistema de tipos de MiniLisp para incluir soporte para listas y cadenas de caracteres, además de los tipos básicos existentes (number y bool).
- Diseñar e integrar un algoritmo de inferencia de tipos que no solo deduzca los tipos de expresiones y funciones en MiniLisp, sino que también soporte la inferencia en expresiones que involucren listas y cadenas.
- Asegurar la verificación de tipos en expresiones recursivas y en estructuras de datos complejas (listas y cadenas), gestionando los errores de tipo de manera clara y consistente.
- Definir y probar operaciones básicas para listas y cadenas (como concatenación, acceso a elementos y longitud) asegurando que el sistema deduzca correctamente los tipos de estas operaciones y verifique la compatibilidad de tipos.
- Documentar el diseño e implementación de las nuevas funcionalidades, explicando las decisiones clave y describiendo posibles extensiones o mejoras futuras para el sistema.

## Conformación de Equipos

- El Proyecto Final deberá resolverse en equipos de máximo 3 integrantes.
- Deberán registrar el nombre de los integrantes en el siguiente formulario: `https://forms.gle/FoafWajbYFRpjJT17`

## Instrucciones

Deberán entregar tres cosas:

1. Un reporte final en formato de artículo en LaTeX con la siguiente estructura:
   *a*) Marco Teórico
   *b*) Especificación Formal del Lenguaje
   *c*) Algoritmo de Inferencia de Tipos

   *d*) Desafíos Encontrados

   *e*) Trabajo a Futuro

   *f*) Referencias

2. El código con la implementación de su solución en Haskell. Separado en:

   *a*) Archivo de configuración de Happy para realizar el análisis léxico y sintáctico.

   *b*) Archivo de eliminación de azúcar sintáctica.

   *c*) Archivo con las reglas de evaluación del lenguaje.

   *d*) Archivo con el algoritmo de inferencia de tipos implementado.

   *e*) Archivo REPL que integre todos los componentes.

3. Un vídeo de exposición explicando el proyecto.

**Ejercicio 1. Marco Teórico.**

Realizar una investigación sobre los siguientes conceptos, pueden usar las notas de clase, pero no debe ser su única referencia:

- Especificación Formal de Lenguajes de Programación
- Recursión y Combinadores de Punto Fijo
- Combinador de Punto Fijo Y
- Semántica Estática
- Inferencia de Tipos

Para el reporte final deberán incluir un marco teóricos que abarque los conceptos anteriores.

**Ejercicio 2. Especificación Formal.**

Se deberá extender el lenguaje para que soporte cadenas y listas. Pueden tomar como referencia la documentación de Racket. Con lo cual deberán realizar lo siguiente:

1. Extender la Gramática Libre de Contexto de la Práctica 5 como sigue:

   *a*) Añadir constructores para representar cadenas.

   *b*) Añadir constructores para representar listas.

   *c*) Añadir el constructor para lograr recursión `letrec`. Sintácticamente es equivalente a `let`.

   *d*) Añadir constructores para: concatenar cadenas, obtener la cadena en un índice, convertir una lista a cadena.

*e*) Añadir constructores para: obtener la cabeza de una lista, obtener la cola de una lista, la longitud de una lista, reversa de una lista, concatenación de una lista, `map` y `filter`.

2. Dar las reglas de sintaxis abstracta correspondientes.

3. Dar las reglas de semántica de paso grande (natural) correspondientes. En este caso se deberá añadir la regla para lograr el comportamiento recursivo de `letrec` y añadir una explicación de su funcionamiento.

**Ejercicio 3. Algoritmo de Inferencia de Tipos**

Se deberá diseñar e implementar un algoritmo de inferencia de tipos para el lenguaje extendido. Para ello realizar los siguientes pasos:

1. Generar las restricciones que debe cumplir el lenguaje con respecto al sistema de tipos.

2. Generar un programa/función que dada una expresión identifique todas las subexpresiones encontradas.

3. Generar un programa/función que dadas todas las subexpresiones genere las restricciones asociadas (deberán definir un tipo o construcción para las restricciones).

4. Codificar el algoritmo de unificación visto en clase.

5. Codificar el algoritmo de inferencia de tipos completo revisado en clase.

## Rúbrica de Evaluación

| Criterio | Descripción | Puntos |
|---|---|---|
| **Reporte Final en LaTeX (50 pts.)** | | |
| **Marco Teórico** | Presenta una introducción clara a los conceptos de recursión, inferencia de tipos y el contexto de listas y cadenas en lenguajes funcionales. | 10 |
| **Especificación Formal del Lenguaje** | Descripción detallada y formal de la gramática, la sintaxis concretas, y las reglas de evaluación para las nuevas extensiones de recursión, listas y cadenas. | 10 |
| **Algoritmo de Inferencia de Tipos** | Explicación clara y detallada del diseño y lógica del algoritmo de inferencia de tipos, incluyendo la definición de las restricciones y la unificación. | 10 |
| **Desafíos Encontrados** | Documenta de manera precisa los problemas y desafíos técnicos enfrentados durante la implementación, y las soluciones aplicadas. | 8 |
| **Trabajo a Futuro** | Presenta propuestas relevantes y realistas para mejorar el sistema o implementar nuevas funcionalidades. | 6 |
| **Referencias** | Cita correctamente fuentes relevantes, incluyendo artículos académicos, libros y otros recursos de referencia. | 6 |
| **Implementación Final en Haskell (35 pts.)** | | |
| **Archivo de configuración de Happy** | Implementa un archivo funcional de \Haskell para el análisis léxico y sintáctico, sin errores y manejando los componentes del lenguaje. | 7 |

| | | |
|---|---|---|
| **Archivo de eliminación de azúcar sintáctico** | Implementa un archivo que elimina correctamente el azúcar sintáctico, manteniendo la claridad en las expresiones y sin introducir errores. | 7 |
| **Archivo con reglas de evaluación** | Define correctamente las reglas de evaluación para el lenguaje MiniLisp, incluyendo listas, cadenas, y recursión. | 7 |
| **Archivo del algoritmo de inferencia de tipos** | Código del algoritmo de inferencia de tipos implementado en Haskell, funcionando correctamente y manejando listas, cadenas y recursión. | 8 |
| **Archivo REPL** | Implementa un REPL que integra todos los componentes de forma funcional, permitiendo la ejecución e inferencia de tipos para expresiones de prueba. | 6 |
| **Exposición en Video (15 pts.)** | | |
| **Explicación del Algoritmo y Diseño** | Explica de manera clara el diseño y funcionamiento del algoritmo de inferencia de tipos y el sistema de recursión implementado. | 6 |
| **Ejecución y Resultados de Pruebas** | Demuestra la ejecución de MiniLisp con ejemplos de prueba que incluyen listas, cadenas y recursión, y explica los resultados. | 5 |
| **Claridad y Presentación del Video** | Presenta el proyecto de manera organizada, clara y comprensible, asegurando buena calidad de audio y video. | 4 |

**Total: 100 pts.**