# CC3032 - Construcción de Compiladores
# Proyecto 2: Generación de CI

Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencia de la Computación

Gabriel Brolo, Bidkar Pojoy

Ciclo 2, 2024

## Introducción

La generación de código intermedio (CI) es la siguiente fase del diseño de nuestro compilador. Luego de haber realizado el análisis semántico (análisis de tipos), utilizarán sus estructuas de datos (árboles sintácticos, tablas de símbolos) para generar una representación intermedia del código de alto nivel. Esta representación intermedia les será de utilidad al momento de la generación de código assembler (u objeto).

## Requerimientos

### Tareas Directas

1. Agregar acciones semánticas necesarias sobre el árbol sintáctico construido, con el objetivo de generar código intermedio. La sintáxis del código intermedio a utilizar es a discresión del diseñador (pueden utilizar la sintáxis de alguna bibliografía conocida o la vista en clase).

2. Complementar la información de la tabla de símbolos con datos necesarios para la generación de código assembler u objeto (direcciones de memoria, etiquetas temporales, etc.)

3. Implementar un algoritmo para asginación y reciclaje de variables temporales durante la transformacíon de expresiones aritméticas.

El diseño del lenguaje intermedio tiene que estar hecho de tal forma que los ayude a generar el código assembler u objeto en la última fase del proceso de compilación. Es por esto que durante su diseño deben de considerar la sintaxis y estructura del lenguaje assembler (u objeto) que utilizarán en la fase final.

### IDE Interactivo

1. Desarrollar un IDE, es decir, una interfaz de desarrollo interactiva que permita a los usuarios escribir su propio código y luego compilarlo, siguiendo todos los requerimientos anteriores.

2. Para este propósito, se permite utilizar cualquier framework de desarrollo web y crear una API para realizar llamadas a sus compiladores, o hacerlo de cualquier otra manera que se considere adecuada.

*a*) Es decir, puede crear una arquitectura de microservicios o bien un monolito.

*b*) Usted elija la arquitectura que considere más apropiada, lo importante es tener el IDE visual para la ejecución de sus códigos de CompiScript.

## Sugerencias de Frameworks y Arquitecturas

A continuación se ofrecen algunas sugerencias de frameworks y arquitecturas que pueden ser útiles para la implementación del compilador y el IDE:

- **Frameworks de Desarrollo Web:**
  - *React*: Para construir interfaces de usuario interactivas.
  - *Vue.js*: Otra opción popular para la construcción de interfaces de usuario.
  - *Angular*: Un framework robusto para aplicaciones web.
  - *Django*: Un framework de Python robusto para aplicaciones web.
  - *Flask*: Un framework de Python robusto para aplicaciones web.
  - *Vaadin*: Un framework de Java robusto para aplicaciones web.
- **Librerías, lenguajes y Herramientas:**
  - *ANTLR*: Para la generación del analizador sintáctico.
  - *Node.js*: Para la creación de la API del compilador.
  - *Express*: Un framework para Node.js que facilita la creación de APIs.
  - *Java, Go, C++*: Un lenguaje tipado ayuda bastante en la creación de un compilador porque da una buena estructura sobre la implementación. Si quieren integrarse después a herramientas como ANTLR no necesitan un SDK como tal, pueden usar el jar de ANTLR (Java) para generar sus clases necesarias de recorrido e implementar otros aspectos del diseño con otros lenguajes. También pueden usar las librerías de Python o Java de ANTLR directamente. O bien otra tool que estén usando.
- **Arquitecturas:**
  - Arquitectura basada en microservicios: Donde cada componente del compilador se implementa como un servicio independiente que puede ser desplegado y escalado de forma autónoma.
  - Arquitectura de cliente-servidor: Donde el IDE funciona como un cliente que envía el código fuente a un servidor para su compilación y luego recibe el resultado.
  - Monolito: Sería lo más similar a un compilador convencional en el aspecto que aumenta la portabilidad al no tener demasiados componentes intermedios. Ustedes mandan, lo importante es que tenga una arquitectura eficiente.

# Entrega

La entrega de este proyecto debe incluir:

1. El código fuente del generador de CI.

   *a*) Entrega a través de Canvas por medio de un documento PDF con el reporte de su proyecto y en este documento, un enlace a su repositorio **privado** de GitHub que deben compartir con su catedrático y auxiliares. Revisen la página de Canvas con los datos de los usuarios de GitHub.

2. Documentación que explique cómo compilar y ejecutar el proyecto, dentro del reporte mencionado anteriormente.

3. Documentación que explique a detalle del Lenguaje Intermedio diseñado, junto a ejemplos y supuestos considerados durante la traducción, Esto servirá a los calificadores para comprender sus decisiones de diseño y analizar la veracidad de su implementación.

4. Ejemplos de código en CompiScript y su correspondiente CI.

5. Un video no listado en YouTube con la ejecución de la compilación hasta esta fase de no más de 5 minutos de duración.

# Ponderación

Este laboratorio tiene como objetivo principal que ustedes comprendan y apliquen técnicas de análisis semántico en el contexto de la construcción de compiladores, desarrollando habilidades prácticas en la implementación de un lenguaje de programación y su entorno de desarrollo. La ponderación es como sigue:

| **Componente** | **Puntos** |
|---|---|
| Dieseño de Código Intermedio | 10 pts |
| Generador de Código Intermedio | 30 pts |
| Atributos adicionales en Tabla de Símbolos | 25 pts |
| Correcta traducción de CompiScript a Código Intermedio | 35pts |
| **Total** | **100 pts** |

Cuadro 1: Ponderación de los componentes del proyecto