🛠️División de Tareas para el Proyecto: Análisis Semántico de Compiscript

Este documento describe las tareas que se deben realizar para completar la fase de compilación enfocada en el análisis semántico del lenguaje **Compiscript**. Cada tarea puede ser asignada a un integrante distinto del grupo.

🧰Tarea 1: Preparación de la Gramática

- **Descripción:** Revisar y validar la gramática `Compiscript.g4`, asegurándose de que esté correctamente estructurada y libre de ambigüedades antes de su compilación con ANTLR.
- **Qué hace:** Prepara y depura la definición formal del lenguaje.
- **Recibe:** Archivo `Compiscript.g4`.
- **Salida:** Versión lista para compilación.
- **Origen de la información:** Diseño del lenguaje y archivo de gramática original.
- **Destino de la información:** Proceso de compilación con ANTLR.
- **Ejemplo:** Detectar ambigüedad en reglas como `expr → expr + expr | expr - expr | ...`

🧰Tarea 2: Compilación con ANTLR y Generación de Parser

- **Descripción:** Usar ANTLR para compilar el archivo `Compiscript.g4` ya validado, y generar los archivos del scanner (lexer) y parser.
- **Qué hace:** Traduce la gramática a código fuente usable en el compilador.
- **Recibe:** El archivo `.g4` de gramática.
- **Salida:** Archivos `.py` (o `.java`) del lexer y parser.
- **Origen de la información:** Gramática lista.
- **Destino de la información:** Clases generadas por ANTLR.
- **Ejemplo:** Ejecutar `antlr -Dlanguage=Python3 Compiscript.g4`

🧰Tarea 3: Construcción del Árbol Sintáctico y Visualización

- **Descripción:** Construir el árbol sintáctico (AST) usando Visitors de ANTLR y generar una visualización del mismo.
- **Qué hace:** Permite representar la estructura jerárquica del código fuente.
- **Recibe:** Código fuente `.cps` válido.
- **Salida:** Objeto AST y visualización gráfica.
- **Origen:** Lexer y parser generados.
- **Destino:** Herramienta de visualización (como Graphviz o lib de Python).
- **Ejemplo:** Visualizar el árbol de `let x = 5 + 3;`

🧰Tarea 4: Implementación del Sistema de Tipos

- **Descripción:** Verificar tipos de datos en operaciones aritméticas, lógicas, asignaciones, listas, estructuras, etc.
- **Qué hace:** Garantiza que los tipos utilizados en el programa sean compatibles.
- **Recibe:** Árbol sintáctico con nodos etiquetados.
- **Salida:** Validación o errores semánticos de tipo.
- **Origen:** AST recorrido con Visitor.
- **Destino:** Consola de errores o IDE.
- **Ejemplo:** `let x: integer = true;` debe marcar error.

🧰Tarea 5: Manejo de Ámbitos y Tabla de Símbolos

- **Descripción:** Diseñar e implementar una tabla de símbolos con jerarquía de entornos para manejar variables, funciones y clases.
- **Qué hace:** Verifica uso correcto de identificadores y sus contextos.
- **Recibe:** Información de nodos del AST.
- **Salida:** Validación de declaraciones, errores de ámbito o redefiniciones.
- **Origen:** Visitor al recorrer nodos de declaración y uso.
- **Destino:** Estructura de tabla de símbolos.
- **Ejemplo:** Detectar uso de variable no declarada `print(y);`

🧰Tarea 6: Análisis Semántico Completo usando Visitor

- **Descripción:** Implementar el Visitor principal que recorra el AST y valide todas las reglas semánticas.
- **Qué hace:** Integra las validaciones del sistema de tipos, ámbitos, funciones y estructuras.
- **Recibe:** AST completo.
- **Salida:** Lista de errores o éxito de compilación.
- **Origen:** Lexer + Parser → AST.
- **Destino:** Consola/IDE.
- **Ejemplo:** Rechazar `const PI;` sin inicializar.

🧰Tarea 7: Creación de Pruebas Unitarias

- **Descripción:** Desarrollar una batería de casos de prueba que incluyan entradas válidas e inválidas para todas las reglas semánticas.
- **Qué hace:** Verifica automáticamente que el compilador detecta errores y acepta código correcto.
- **Recibe:** Casos de prueba `.cps`
- **Salida:** Resultados de validación (éxito o error).
- **Origen:** Archivos de prueba del grupo y profesor.
- **Destino:** Consola o sistema de testing.
- **Ejemplo:** Validar error de `return` fuera de función.

🧰Tarea 8: Desarrollo del IDE Interactivo

- **Descripción:** Crear una interfaz gráfica (simple o completa) para escribir código en Compiscript, compilarlo y ver errores en línea.
- **Qué hace:** Mejora la experiencia del usuario.
- **Recibe:** Código fuente `.cps`
- **Salida:** Visualización del código + errores resaltados + consola.
- **Origen:** Entrada del usuario.
- **Destino:** Módulo de compilación y consola gráfica.
- **Ejemplo:** Editor con área de texto, botón "compilar" y panel de errores.

🧰Tarea 9: Documentación Técnica y README(s)

- **Descripción:** Redactar los archivos `README.md` por fase. Cada uno debe explicar qué es, qué hace, qué entradas/salidas tiene, y ejemplos.
- **Qué hace:** Facilita el entendimiento del proyecto y su mantenimiento.
- **Recibe:** Código y contexto de cada fase.
- **Salida:** Archivos `README.md`
- **Origen:** Todo el proyecto.
- **Destino:** Documentación en GitHub.

🧰Tarea 10: Administración del Repositorio y Evidencias

- **Descripción:** Configurar y mantener el repositorio en GitHub, validando commits por integrante.
- **Qué hace:** Asegura trazabilidad y trabajo colaborativo real.
- **Recibe:** Archivos de código y documentación.
- **Salida:** Repositorio organizado con ramas, commits y documentación.
- **Origen/Destino:** GitHub.
- **Ejemplo:** Pull requests por tarea, con revisión entre integrantes.

👉**Recomendación:** Usar Python para el visitor + pruebas, y JavaScript o Electron para el IDE si se desea hacerlo visual. Para consola, se puede usar `Tkinter` o `PyQt`.