Proyecto: Parser para Fangless Python

Descripción del Proyecto

El proyecto avanza a la tercera y última fase del proceso de compilación: la generación de código. El objetivo es diseñar e implementar un transpilador que utilice el Árbol de Sintaxis Abstracta (AST) generado por el Parser para traducir código fuente de "Fangless Python" a un código semánticamente equivalente en C++.

Este componente no solo realizará la traducción de las estructuras sintácticas, sino que también deberá diseñar e implementar una solución para emular el sistema de tipado dinámico de Python dentro del entorno de tipado estático de C++. Finalmente, el proyecto culmina con un análisis de rendimiento comparativo entre el código original en Python y el código generado en C++, evaluando la eficiencia de la transpilación.

Aspectos Administrativos y Técnicos

- Herramienta de Desarrollo: La herramienta designada para el desarrollo es Python, con el uso obligatorio de la librería PLY (Python Lex-Yacc) para las fases de Lexer y Parser. Se requerirá un compilador de C++ moderno (como G++) para compilar y ejecutar el código generado.
- Idioma: El código fuente, los comentarios internos y toda la documentación técnica deberán redactarse en inglés.
- Control de Versiones: Se requiere el uso del sistema de control de versiones Git en conjunto con la plataforma GitHub. La integración de cambios al repositorio principal deberá realizarse exclusivamente a través de Pull Requests.
- Nomenclatura de Ramas: Las ramas de desarrollo deben adherirse estrictamente al formato TASK_#_BriefDescription.

- Equipos de Trabajo: El proyecto será desarrollado en equipos de hasta tres estudiantes.
- Fecha de Entrega: La fecha límite para la entrega del proyecto es el jueves 6 de Noviembre de 2025, a las 23:59 hrs.
- Buenas Prácticas de Código: Se espera que el código producido sea limpio, eficiente, modular y mantenible, facilitando así su futura extensión y revisión.
- El valor de este proyecto es de un 60% de la totalidad de la rúbrica de proyectos.

Requerimientos del Lexer (Analizador Léxico)

Entrada y Salida

- **Entrada:** Un archivo de código fuente en "Fangless Python" (.py). El sistema deberá procesarlo con el Lexer y Parser previamente construidos para generar el AST.
- **Salida:** Un archivo de código fuente en C++ (.cpp) que sea la traducción funcional del programa de entrada. El código C++ generado debe ser autocontenido y compilable y el dicho archivo compilado.

Mapeo de Características Sintácticas

El transpilador debe recorrer el AST y generar código C++ equivalente para las siguientes construcciones:

Programa Principal y Funciones:

- El código en el ámbito global del archivo Python se encapsulará dentro de la función main() en C++.
- Las definiciones de funciones (def) se traducirán a funciones de C++. Estas funciones deben ser capaces de aceptar parámetros y devolver valores de diferentes tipos para replicar el comportamiento de Python.

Manejo de Tipado Dinámico:

- Para emular el comportamiento de Python, donde una variable puede cambiar de tipo (ej. x = 10, luego x = "hola"), el código C++ generado deberá utilizar una estructura de datos existe o creada por el equipo.

Estructuras de Control:

- Condicionales: Las sentencias if/elif/else se traducirán a if/else if/else. La condición deberá ser evaluada extrayendo y convirtiendo el valor booleano desde la estructura de tipado dinámico.
- Bucles while: Se mapearán directamente a bucles while en C++.
- Bucles for: El bucle for NAME in expression: se traducirá a un bucle for de C++, asumiendo que la expresión es un iterable simple.

Expresiones:

- Operadores Aritméticos, Relacionales y Lógicos: Se traducirán a sus equivalentes en C++ (+, -, *, /, %, ==, !=, <, >, <=, >=, &&, ||, !), pero encapsulados en una lógica de verificación de tipos en tiempo de ejecución que opere sobre la estructura de datos creada.

- Llamadas a Funciones: Se traducirán directamente, pasando los argumentos envueltos en la estructura de tipado dinámico.

Análisis de Rendimiento

Una parte fundamental del proyecto es medir y analizar el rendimiento del código C++ generado en comparación con la ejecución del script original de Python y otro programa escrito a mano en C++. Se deben realizar y documentar los resultados de las siguientes 3 pruebas:

- Fibonacci Recursivo: Medir el tiempo de cálculo para los valores del 1 al 50.
- Fibonacci Iterativo: Medir el tiempo de cálculo para los valores del 1 al 50.
- Prueba Propuesta 1: Un algoritmo de su elección con tamaño de entrada variable (ej. ordenamiento de un arreglo). Se deben ejecutar y medir al menos 10 tamaños de entrada diferentes.

Los resultados deben presentarse en tablas y gráficos, acompañados de un análisis que explique las diferencias de rendimiento observadas.

Manejo de Errores

El analizador sintáctico debe gestionar los errores de forma robusta, sin interrumpir su ejecución abruptamente e incluyendo los anteriormente programados en el analizador léxico.

- **Caracteres Desconocidos:** Cualquier carácter que no corresponda a un token definido debe generar un error léxico.
- **Secuencias de Escape Inválidas:** El uso de secuencias de escape no reconocidas dentro de literales de cadena debe ser reportado como un error.
- **Indentación Incorrecta:** Deben manejarse y reportarse los errores en la estructura de indentación del código.
- **Gramática Inválida:** Un error sintáctico fundamental, como una sentencia sin la estructura correcta (ej., if sin dos puntos).

- **Expresiones Mal Formadas:** Errores de precedencia o asociatividad, como el uso de un operador binario sin un segundo operando.

Rúbrica de Evaluación

Importante: Los puntos serán evaluados en su totalidad por la defensa del proyecto. La entrega de este es un criterio mínimo de evaluación.

Criterio	Puntos	Descripción
Correctitud del Transpilador	40	Traducción correcta y funcional de todas las estructuras a C++, incluyendo la implementación de un mecanismo robusto para el tipado dinámico. El código C++ generado debe ser compilable y producir resultados correctos.
Calidad del Código	20	El código debe ser bien estructurado, modular y estar debidamente documentado, demostrando una implementación eficiente.
Análisis de Rendimiento		Ejecución completa y rigurosa de las 5 pruebas de rendimiento. La documentación debe incluir tablas, gráficos y un análisis claro de los resultados comparativos.
Manejo de Errores	15	Capacidad del sistema para detectar y reportar errores léxicos y sintácticos con mensajes claros y precisos que faciliten la depuración.
Documentación	10	Calidad y claridad de la documentación técnica (comentarios) y de usuario (guía de uso), explicando los componentes y su funcionamiento.