**Fases del Proyecto: Desarrollo de un Analizador Semántico**

**Fase 1: Implementación del Analizador Léxico y Sintáctico (Pre-requisito)**

**Antes de entrar al análisis semántico, es necesario tener una base sólida del análisis léxico y sintáctico. En esta fase:**

* **Se debe haber implementado el análisis léxico que identifica los tokens del lenguaje.**
* **El análisis sintáctico debe generar un Árbol de Sintaxis Abstracta (AST) a partir de la entrada del programa.**

**Entregable: Un analizador léxico y sintáctico que produzca un AST del código fuente.**

**Fase 2: Tablas de Símbolos (Verificación de Declaraciones)**

**La primera tarea del analizador semántico es verificar que todas las variables, funciones, y otros identificadores estén correctamente declarados antes de ser usados. Para esto, los estudiantes deben:**

1. **Construir una Tabla de Símbolos, que almacene la información de las variables, funciones, y sus atributos (nombre, tipo, alcance, etc.).**
2. **Verificar las declaraciones de las variables antes de que se utilicen.**
3. **Gestionar el alcance de las variables (scope) para asegurar que las variables locales y globales se manejen correctamente.**

**Ejemplo de error a detectar:**

**c**

**Copiar código**

**x = 10; // Error: 'x' no está declarado**

**int y = x + 2; // Error: 'x' no está declarado**

**Entregable: Un analizador semántico que verifique que todas las variables están correctamente declaradas.**

**Fase 3: Verificación de Tipos (Type Checking)**

**En esta fase, los estudiantes deben implementar la verificación de compatibilidad de tipos. Esto incluye:**

1. **Asegurarse de que las operaciones aritméticas, lógicas, y de asignación sean realizadas entre tipos compatibles.**
2. **Verificar que los tipos de retorno de las funciones sean correctos.**
3. **Controlar la coerción de tipos (conversión implícita de tipos) cuando sea necesario, por ejemplo de int a float.**

**Ejemplo de error a detectar:**

**c**

**Copiar código**

**int a = 5;**

**float b = 3.2;**

**a = a + b; // Error: no se puede asignar float a una variable int sin conversión explícita**

**Entregable: Un analizador que verifique la compatibilidad de tipos entre operaciones y asignaciones.**

**Fase 4: Verificación de Alcance (Scope Management)**

**Implementar la gestión de alcances (scopes), es decir, controlar el acceso y la visibilidad de variables y funciones en diferentes bloques de código (funciones, ciclos, condicionales).**

1. **Asegurarse de que las variables locales no interfieran con las globales.**
2. **Controlar la creación de nuevos ámbitos dentro de funciones o bloques {}.**
3. **Manejar correctamente la anidación de bloques.**

**Ejemplo de error a detectar:**

**c**

**Copiar código**

**int x = 10;**

**{**

**int x = 5;**

**// Dentro de este bloque, 'x' debe referirse a la variable local (5), no a la global (10)**

**}**

**Entregable: Un analizador semántico que maneje correctamente los diferentes alcances de variables.**

**Fase 5: Verificación de Funciones y Procedimientos**

**En esta fase, el analizador debe:**

1. **Verificar que las funciones y procedimientos sean llamados con el número correcto de argumentos.**
2. **Asegurarse de que los tipos de los argumentos sean compatibles con los tipos esperados por la función.**
3. **Verificar que las funciones devuelvan el tipo correcto.**

**Ejemplo de error a detectar:**

**c**

**Copiar código**

**int suma(int a, int b) {**

**return a + b;**

**}**

**float res = suma(5, 2.5); // Error: el segundo argumento debe ser un int**

**Entregable: Verificación de la consistencia semántica en llamadas a funciones y retornos.**

**Fase 6: Manejo de Errores Semánticos (Error Handling)**

**En esta fase, se deben implementar mecanismos de detección y reporte de errores semánticos. El sistema debe:**

1. **Informar claramente dónde ocurre el error (número de línea, columna).**
2. **Mostrar mensajes de error comprensibles para el programador.**
3. **Continuar procesando el código para encontrar otros errores (no detenerse ante el primer error).**

**Ejemplo de reporte de errores:**

**plaintext**

**Copiar código**

**Error: variable 'x' no declarada. Línea 2, Columna 5**

**Error: tipo incompatible en asignación, se esperaba 'int', pero se encontró 'float'. Línea 5, Columna 7**

**Entregable: Un sistema robusto de manejo de errores semánticos.**

**Fase 7: Optimización Semántica (Opcional)**

**Como parte avanzada del proyecto, los estudiantes podrían implementar algunas optimizaciones semánticas:**

1. **Eliminación de código muerto (variables o funciones que nunca son usadas).**
2. **Simplificación de expresiones que sean semánticamente correctas pero innecesariamente complejas.**

**Ejemplo de optimización:**

**c**

**Copiar código**

**int a = 5;**

**int b = a \* 1; // Se puede optimizar a 'int b = a;'**

**Entregable: Un analizador que aplique optimizaciones básicas.**

**Herramientas Sugeridas para el Proyecto**

* **Python con herramientas como Ply o Lark para la implementación del compilador.**
* **Flex/Bison en C/C++ si se prefiere un enfoque más cercano a la industria.**
* **JFlex/CUP si están trabajando en Java.**

**Entregables Finales:**

1. **Analizador Semántico completo con tablas de símbolos, verificación de tipos y funciones, y manejo de errores.**
2. **Documentación técnica que describa el diseño y el proceso de implementación.**
3. **Casos de prueba que incluyan ejemplos de código correcto y código con errores semánticos para verificar que el analizador los detecte.**