Analizador Semántico

Alumno:

- Nombre: Amezcua García Juan Ángel David
- Código: 220286784

Carrera:

- Ingeniería de Computación (INCO)
- División de Tecnologías para la Integración Ciber-Humana
- Centro Universitario de Ciencias Exactas e Ingenierías

Fecha de entrega:

11 de noviembre de 2024

Asignatura:

- Seminario de Solución de Problemas de Traductores de Lenguajes II
- *Clave*: I7028
- *NRC*: 103841
- *Sección*: D02
- *Calendario*: 24-B

Profesor:

Michel Emanuel López Franco

Analizador Semántico

Introducción

Un compilador es una herramienta fundamental en la informática, diseñada para traducir el código fuente de un lenguaje de programación a un lenguaje de máquina que pueda ser ejecutado por un sistema. Este proyecto se centra en la construcción de un compilador que alcanza hasta la fase de análisis semántico, una etapa crucial para garantizar la corrección lógica y técnica de los programas procesados.

El proceso de compilación generalmente se divide en varias fases principales: análisis léxico, análisis sintáctico, análisis semántico, generación de código intermedio, optimización y generación de código máquina. Cada una de estas etapas cumple una función específica y contribuye a la transformación progresiva del código fuente. En este contexto, nuestro enfoque radica en las tres primeras etapas:

1. **Análisis léxico:**
    - Divide el código fuente en unidades mínimas significativas conocidas como tokens.
    - Identifica elementos como palabras clave, identificadores, operadores y símbolos.
2. **Análisis sintáctico:**
    - Verifica que los tokens obtenidos cumplan con las reglas gramaticales definidas para el lenguaje.
    - Genera un árbol de análisis sintáctico que representa la estructura jerárquica del programa.
3. **Análisis semántico:**
    - Valida que las construcciones sintácticamente correctas tengan sentido lógico y semántico.
    - Realiza tareas como la verificación de tipos, manejo de la tabla de símbolos y detección de errores semánticos.

El análisis semántico es una etapa crítica porque asegura que el programa no solo sea sintácticamente válido, sino también semánticamente coherente. Por ejemplo, verifica que las variables utilizadas hayan sido declaradas, que las asignaciones sean compatibles con los tipos de datos, y que las funciones reciban el número y tipo de argumentos adecuados.

Este reporte aborda la implementación de un compilador hasta la fase semántica, destacando los mecanismos, estructuras y reglas empleadas para garantizar la integridad semántica de los programas procesados. Además, se describen los resultados obtenidos en la validación de diferentes casos de prueba, tanto correctos como incorrectos.

---

2. Estructura General del Programa

El programa se organiza en varias fases fundamentales, reflejando el flujo típico de un compilador. Cada fase interactúa con las anteriores y proporciona resultados esenciales para la siguiente. Estas fases, implementadas hasta el análisis semántico, son las siguientes:

2.1. Análisis Léxico

La primera fase del compilador toma como entrada el código fuente y lo descompone en tokens. Estos tokens representan las unidades más pequeñas y significativas del lenguaje, como palabras clave, identificadores, operadores y símbolos.

- **Módulo principal:** `analizador_lexico.py`
- **Tareas principales:**
    - Leer el código fuente y dividirlo en secuencias manejables.
    - Clasificar cada token en una categoría específica (palabras clave, identificadores, literales, etc.).
    - Generar errores si encuentra secuencias inválidas.

El análisis léxico crea una lista de tokens que será utilizada por la fase de análisis sintáctico.

2.2. Análisis Sintáctico

El análisis sintáctico toma como entrada la lista de tokens generada por el analizador léxico y construye un árbol sintáctico que representa la estructura jerárquica del programa según la gramática del lenguaje.

- **Módulo principal:** `analizador_sintactico.py`

- **Tareas principales:**
    - Validar que la secuencia de tokens siga las reglas gramaticales del lenguaje.
    - Construir un árbol de análisis sintáctico (AST, por sus siglas en inglés) que representa las relaciones entre los tokens.
    - Generar errores si los tokens no forman una estructura válida.

El árbol sintáctico generado es la entrada para el analizador semántico, que validará su coherencia lógica y semántica.

2.3. Análisis Semántico

El análisis semántico verifica que las construcciones del árbol sintáctico tengan sentido lógico y semántico en el contexto del programa. Esta fase utiliza una tabla de símbolos para llevar un registro de los identificadores y sus propiedades (tipos, ámbitos, etc.).

- **Módulo principal:** `analizador_semantico.py`
- **Tareas principales:**
    - Manejar la tabla de símbolos: Declarar y buscar identificadores, gestionar ámbitos.
    - Verificar tipos de datos en expresiones, asignaciones y retornos.
    - Validar reglas semánticas, como el número y tipo de parámetros en funciones.
    - Detectar errores como variables no declaradas, tipos incompatibles y retornos fuera de funciones.

2.4. Relación entre las Fases

Cada fase del programa proporciona información crucial para la siguiente:

- El analizador léxico entrega tokens al analizador sintáctico.
- El analizador sintáctico genera un árbol sintáctico para el analizador semántico.
- El analizador semántico utiliza la estructura del árbol sintáctico para validar las reglas semánticas.

Esta interdependencia asegura que los errores en fases anteriores se identifiquen y manejen antes de proceder, garantizando la robustez del compilador.

2.5. Modularidad y Escalabilidad

El diseño modular del programa permite:

- Modificar o extender una fase sin afectar significativamente a las demás.
- Escalar el compilador para incluir fases adicionales como generación de código intermedio o optimización.

---

3. Fase Semántica

El análisis semántico es la etapa en la que se verifica que las construcciones del árbol sintáctico generado tengan sentido lógico dentro del contexto del programa. Esta fase garantiza que el código no solo sea válido desde el punto de vista de la gramática, sino también en términos de significado.

3.1. Propósito y Objetivos

El principal objetivo del análisis semántico es detectar y reportar errores relacionados con la lógica del programa, incluyendo:

- Uso de variables no declaradas.
- Incompatibilidades de tipos en expresiones y asignaciones.
- Declaraciones duplicadas en un mismo ámbito.
- Verificación de la coherencia en las llamadas a funciones (número y tipo de argumentos).
- Asegurar que las funciones retornen valores adecuados según lo declarado.

En esta etapa, se construye y utiliza una **tabla de símbolos** para registrar información sobre los identificadores declarados en el programa.

3.2. Funcionalidades Principales

3.2.1. Manejo de la Tabla de Símbolos

La tabla de símbolos es una estructura jerárquica que permite gestionar los identificadores (variables, funciones, parámetros, etc.) y sus propiedades en diferentes niveles de ámbito.

- **Estructura:**
    - Implementada como una pila de diccionarios.
    - Cada nivel de la pila representa un ámbito, y los diccionarios almacenan información sobre los identificadores.
- **Operaciones clave:**
    - **Entrar a un ámbito:** Agrega un nuevo diccionario a la pila.
    - **Salir de un ámbito:** Elimina el diccionario superior de la pila.
    - **Declarar un identificador:** Añade una entrada al diccionario actual con información como tipo y clase.
    - **Buscar un identificador:** Recorre los diccionarios de la pila desde el nivel actual hasta el global.

3.2.2. Verificación de Tipos

El analizador semántico comprueba que las operaciones realizadas sean compatibles con los tipos de datos involucrados.

- Ejemplos:
    - En una asignación, el tipo de la expresión debe coincidir con el tipo de la variable.
    - Los operadores aritméticos solo se aplican a tipos numéricos compatibles.
    - Las condiciones en estructuras de control (`if`, `while`) deben ser de tipo `int` (booleano).

3.2.3. Validación de Reglas Semánticas

El programa implementa reglas semánticas específicas para diferentes construcciones del lenguaje:

- **Funciones:**
    - Verifica que el número y tipo de los argumentos en una llamada coincidan con los parámetros declarados.
    - Valida que las funciones retornen un valor del tipo declarado.

- **Declaraciones:**
    - Detecta intentos de declarar una variable o función con un nombre ya existente en el mismo ámbito.
- **Control de Flujo:**
    - Asegura que las expresiones condicionales sean del tipo adecuado.
    - Valida que las estructuras de control (`if`, `while`) contengan bloques sintácticamente y semánticamente válidos.

3.3. Manejo de Errores

El analizador semántico identifica y reporta errores relacionados con:

- Uso de identificadores no declarados.
- Conflictos de tipo en expresiones, asignaciones y retornos.
- Llamadas a funciones con argumentos incorrectos.
- Declaraciones duplicadas.

Estos errores se almacenan en una lista, que posteriormente se imprime en un formato legible para el usuario.

3.4. Interacción con el Árbol Sintáctico

El analizador semántico utiliza el árbol generado en la fase sintáctica como entrada para recorrer las diferentes construcciones del programa. Cada nodo del árbol es procesado por un método específico que implementa las reglas semánticas correspondientes.

3.5. Resultados

El análisis semántico asegura que el programa procesado sea lógico y esté libre de errores semánticos. Si se detectan errores, estos se reportan antes de proceder con etapas posteriores del compilador.

---

4. Implementación Técnica

El análisis semántico se implementa a través de un diseño modular y extensible que incluye varias estructuras y funciones clave. En este punto, se describe cómo se aborda cada componente principal de la fase semántica.

4.1. Tabla de Símbolos

La tabla de símbolos es la estructura central para gestionar la información de los identificadores en diferentes niveles de ámbito. Su implementación incluye:

- **Estructura:**
    - Una pila de diccionarios (`self.ambitos`), donde:
        - Cada diccionario representa un nivel de ámbito.
        - Las claves son los nombres de los identificadores, y los valores son objetos que almacenan información relevante (tipo, clase, etc.).

- **Métodos principales:**
    - **`entrar_ambito`:**
        - Crea un nuevo ámbito al añadir un diccionario vacío a la pila.
        - Esto ocurre, por ejemplo, al entrar en el cuerpo de una función o bloque.
    - **`salir_ambito`:**
        - Elimina el ámbito actual al quitar el diccionario superior de la pila.
        - Esto se utiliza al finalizar el análisis de un bloque.
    - **`declarar`:**
        - Declara un nuevo identificador en el ámbito actual, verificando que no exista previamente en el mismo nivel.
        - Si ya existe, genera un error semántico.
    - **`buscar`:**
        - Busca un identificador recorriendo los diccionarios desde el ámbito actual hasta el global.
        - Si no se encuentra, genera un error semántico.

4.2. Verificación de Reglas Semánticas

El núcleo del analizador semántico son los métodos que implementan reglas específicas para garantizar la coherencia semántica del programa.

- **Verificación de tipos:**
  - Se comprueban las asignaciones, operaciones y expresiones.
  - Por ejemplo:
    - En `x = y + z`, los tipos de `y` y `z` deben ser compatibles, y el resultado debe coincidir con el tipo de `x`.
- **Reglas de funciones:**
  - Validación de que el número y tipo de argumentos coincidan con los parámetros de la función.
  - Verificación del tipo de retorno en las funciones.
- **Reglas de control de flujo:**
  - Asegurar que las condiciones (`if`, `while`) sean de tipo `int`.

4.3. Manejo de Errores Semánticos

Los errores semánticos se gestionan de forma centralizada mediante una lista (`self.errores`). Cada error identificado se añade a esta lista con un mensaje descriptivo.

- **Tipos de errores detectados:**

  - Uso de variables no declaradas.
  - Tipos incompatibles en asignaciones u operaciones.
  - Declaraciones duplicadas en el mismo ámbito.
  - Argumentos incorrectos en llamadas a funciones.
  - Retornos fuera de funciones o de tipos incorrectos.

4.4. Métodos Auxiliares Clave

Algunos métodos auxiliares que soportan la implementación incluyen:

- **`visitar_Expresion`:**
  - Procesa expresiones, verificando tipos y operadores.
  - Maneja operaciones aritméticas, relacionales y lógicas.
- **`visitar_Sentencia`:**

    - Implementa reglas para asignaciones, llamadas a funciones, estructuras de control y retornos.
- **`visitar_DefFunc`:**
    - Gestiona la declaración de funciones, incluyendo la entrada/salida de ámbitos y la validación de parámetros.

4.5. Diseño Modular y Escalabilidad

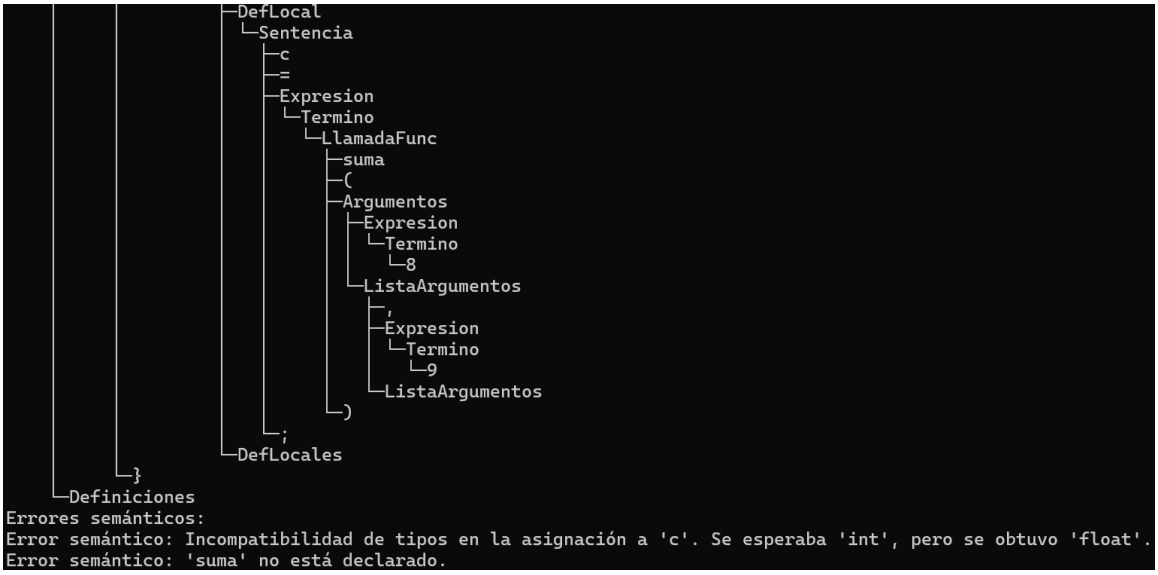
El diseño modular del programa permite:

- Agregar nuevas reglas semánticas sin modificar la arquitectura base.
- Ampliar el compilador con fases adicionales, como la generación de código intermedio.

4.6. Casos de Prueba en la Implementación

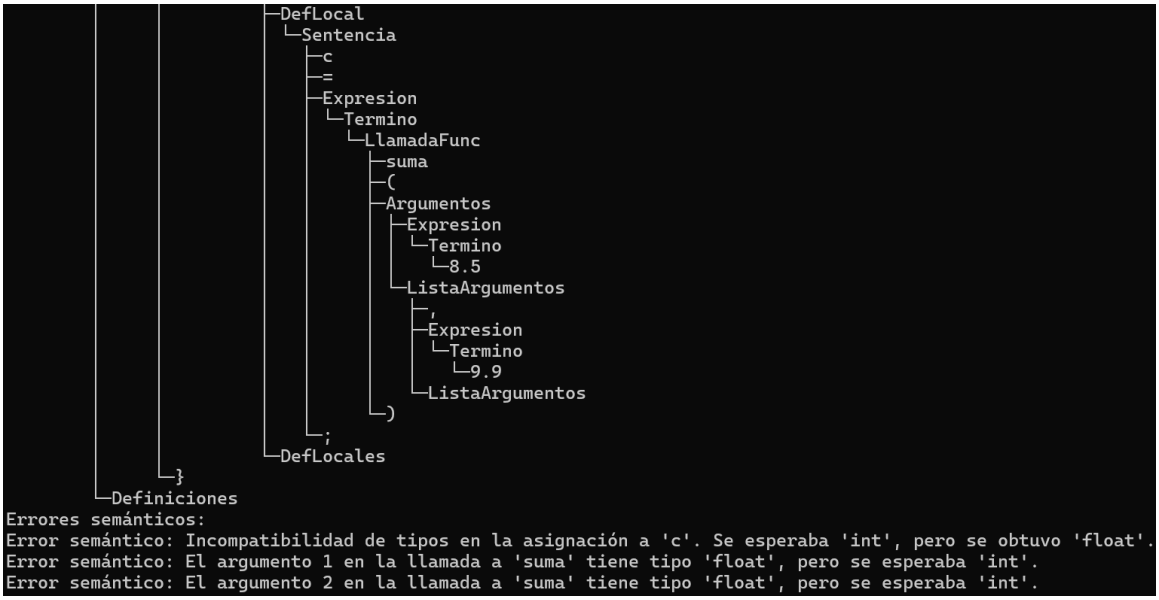
Cadena 1 (incorrecta semánticamente)

```
int main() {
   float a;
   int b;
   int c;
   c = a + b;
   c = suma(8, 9);
}
```



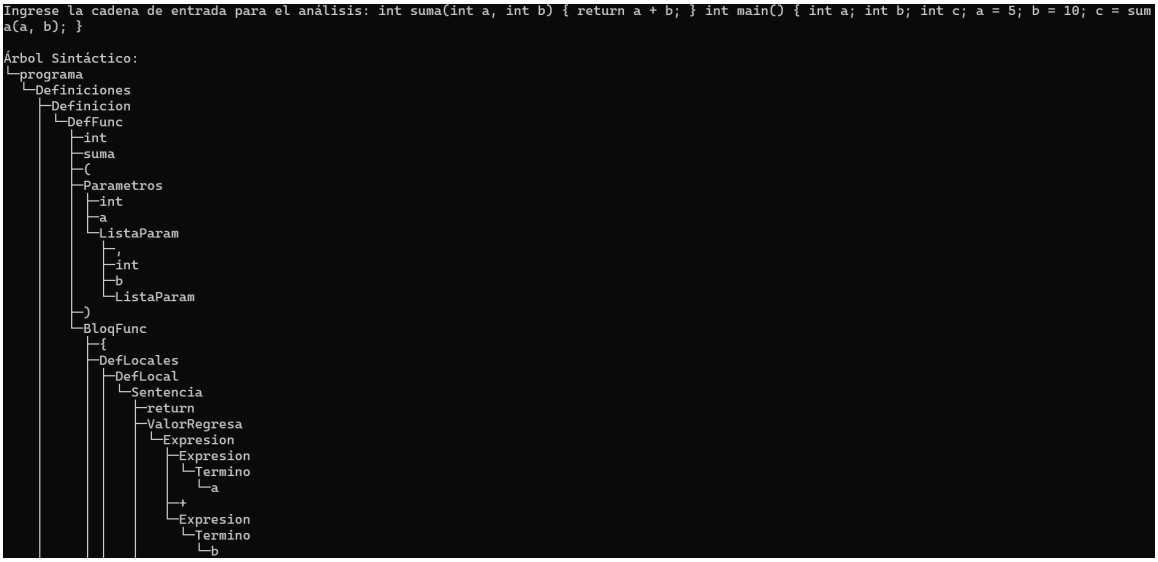
Cadena 2 (incorrecta semánticamente)

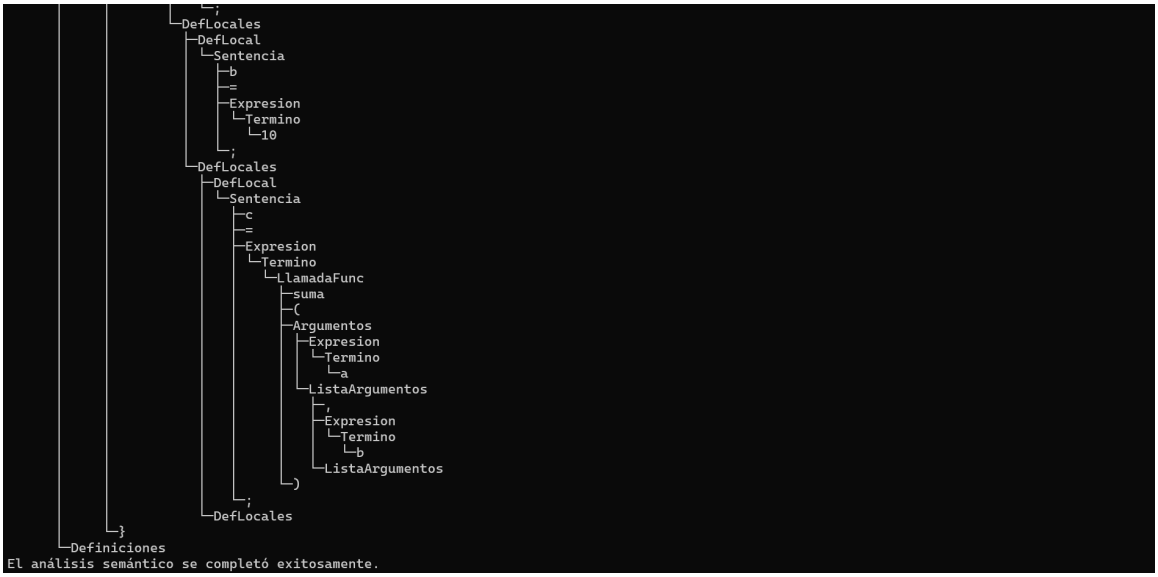
```
int a;

int suma(int a, int b) {

   return a + b;

}

int main() {

   float a;

   int b;

   int c;

   c = a + b;

   c = suma(8.5, 9.9);

}
```



Cadena 3 (correcta semánticamente)

```
int suma(int a, int b) {
   return a + b;
}
int main() {
   int a;
   int b;
   int c;
   a = 5;
   b = 10;
   c = suma(a, b);
}
```





Conclusión

Este proyecto demuestra el diseño e implementación de un compilador funcional hasta la fase de análisis semántico, cumpliendo con los objetivos iniciales de validar tanto la estructura sintáctica como la lógica de los programas procesados. Las principales conclusiones del desarrollo incluyen:

Logros del Proyecto

1. **Estructura Modular y Extensible:**

    - La implementación divide claramente las fases del compilador, permitiendo su desarrollo y prueba de forma independiente.
    - Cada módulo está diseñado para ser escalable y adaptable a nuevos requisitos.
2. **Éxito del Análisis Semántico:**

    - Se implementaron las principales reglas semánticas, como la verificación de tipos, gestión de ámbitos y control de flujos.
    - Los errores comunes (variables no declaradas, incompatibilidades de tipo, etc.) se detectan con mensajes claros y precisos.
3. **Casos de Prueba Representativos:**

    - El compilador se probó con entradas variadas, confirmando su capacidad para procesar tanto casos válidos como inválidos.
    - Los resultados del análisis son coherentes y reflejan el diseño esperado.

Impacto de la Fase Semántica

El análisis semántico es un componente crítico en cualquier compilador, ya que asegura la coherencia lógica y previene errores difíciles de identificar en etapas posteriores. En este proyecto:

- La fase semántica garantiza que el programa no solo sea válido en términos de gramática, sino también en cuanto a su significado.
- La integración con las fases léxica y sintáctica refuerza la robustez del compilador.

Reflexión General

Este proyecto ofrece una sólida introducción al desarrollo de compiladores, destacando la importancia de cada fase en el procesamiento del código fuente. La experiencia adquirida establece una base para proyectos futuros más complejos, al mismo tiempo que proporciona un sistema útil para validar programas y detectar errores.