UNIVERSIDAD DE GUADALAJARA

Red Universitaria e Institución Benemérita de Jalisco

Alumno:

Romo Rodríguez José Alberto (216853747)

Materia:

I7028 | Seminario de Solucion de Problemas de Traductores de Lenguajes II

NRC:

103841

Sección:

D02

Maestro:

Lopez Franco Michel Emanuel

Horario:

Lunes y Miercoles | 13:00 - 14:55

Tarea:

Analizador Semantico

Fecha de Entrega:

25 de Mayo del 2023

Introduccion.
En el ámbito de la programación, los analizadores semánticos son componentes esenciales para verificar la corrección y la consistencia del significado de un programa.

Es importante destacar que, antes de abordar el análisis semántico, se debe haber desarrollado un analizador sintáctico. El analizador sintáctico, específicamente del tipo LR (parsing LR), se encarga de analizar la estructura gramatical del código fuente y construir un árbol de sintaxis abstracta que representa la estructura jerárquica del programa. Este árbol sintáctico será la base sobre la cual se realizarán las verificaciones semánticas.

Una vez que se ha completado la etapa de análisis sintáctico, el analizador semántico entra en juego. Su objetivo es examinar el árbol sintáctico y aplicar reglas y restricciones adicionales para garantizar la coherencia semántica del programa. Algunas de las tareas típicas que se realizan durante el análisis semántico incluyen:

1.- Comprobación de tipos: verificar que los tipos de datos utilizados sean compatibles en las operaciones y asignaciones correspondientes.

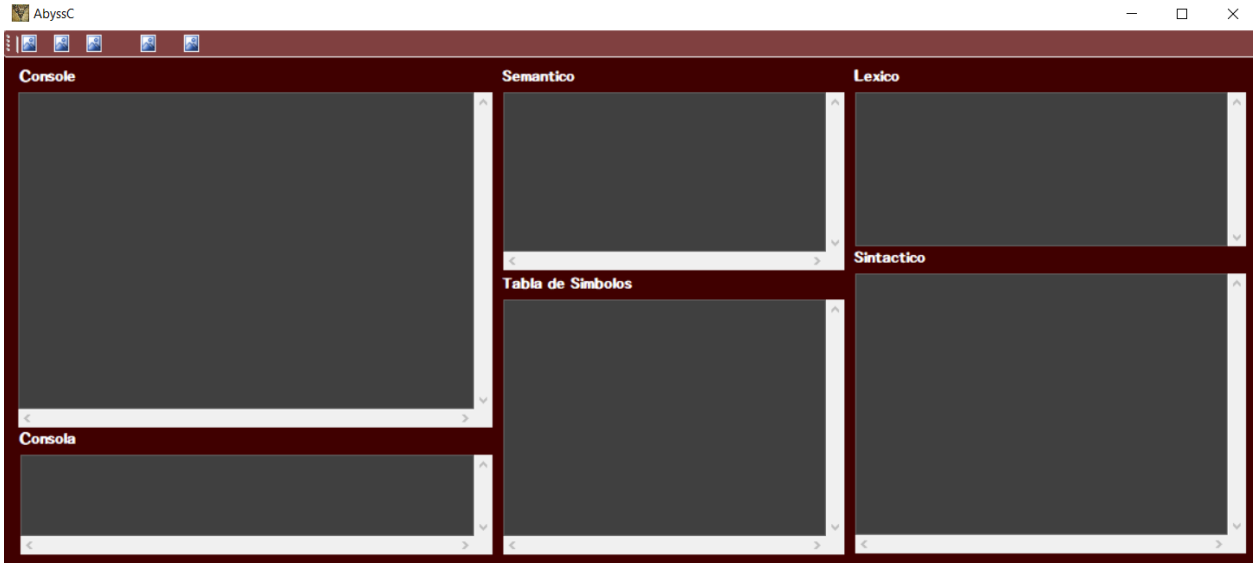
2.- Declaraciones y alcance de variables: asegurarse de que las variables estén correctamente declaradas y que su uso se encuentre dentro del alcance adecuado.

3.- Comprobación de reglas específicas del lenguaje: aplicar reglas semánticas propias del lenguaje de programación en uso, como la verificación de la presencia de métodos y propiedades, la validación de la herencia de clases, etc.

4.- Optimizaciones y sugerencias: realizar optimizaciones semánticas y brindar sugerencias para mejorar el rendimiento o la legibilidad del código.

Desarrollar un analizador semántico implica comprender las reglas y restricciones del lenguaje, así como aplicar algoritmos y estructuras de datos eficientes para realizar las comprobaciones semánticas de manera precisa y rápida. Además, es fundamental contar con un buen conocimiento de las técnicas de análisis semántico y de los enfoques específicos para el análisis de lenguajes de programación.

Ademas de eso para esta nueva revision del codigo se implementan un par de cosas adicionales a la interfaz:



Desarrollo.

El desarrollo de un analizador semántico implica varias etapas y consideraciones importantes.

Definir las reglas semánticas:
Como se definio al comienzo se usaran las mismas reglas del lenguaje de programacion ’c’.

Obtener el árbol sintáctico:
Este paso es sencillo pues gracias al correcto desarrollo del analizador sintactico logramos hacerlo de una forma organizada:

```
Results.Text = "";
Results_Console.Text = "";
Results_Sintactico.Text = "";
Results_Semantico.Text = "";
Lexico = new List<string>();
semantico = new Semantico();
//Fase 1: Analisis Lexico
Lexico
//Fase 2: Analisis Sintactico
#region Sintactico
pila = new Pila();
AnalizadorSintactico(programa);
#endregion
//Fase 3. Analisis Semantico
#region Semantico
if (!error)
{
    AnalizadorSemantico(Lexico);
}
#endregion
//Fase 4. Codigo Objeto
```

*Nota: Recordando que «Lexico» es nuestra pila.

Diseñar las estructuras de datos:
Para llevar a cabo las comprobaciones semánticas, es necesario diseñar las estructuras de datos adecuadas para almacenar información relevante sobre el código fuente. Esto puede incluir tablas de símbolos para variables y funciones, tablas de tipos, pilas de alcance, entre otros. Estas estructuras ayudarán a realizar verificaciones coherentes y eficientes.

```
//int definido
if (element == "int:tipo:4")
{
    foreach (char caracter in programa[i + 1])
    {
        if (caracter == ':')
        {
            break;
        }
        else
        {
            variable += caracter;
        }
    }
    if (programa[i + 2] == "(:parentesisOpen:14")
    {
        valor = "<Funcion>";
        if (semantico.FindFloat(variable) > -1 || semantico.FindInt(variable) > -1 ||
        semantico.FindVoidFunction(variable) > -1 || semantico.FindIntFunction(variable) > -1 || semantico.Fi
        {
            Results_Console.Text += "'" + variable + "' -> was defined before" + "\r\n";
        }
        else
```

Y se hace de la siguiente manera, considerando que en este caso se inclue el tipo int definido y ademas se colocan las validaciones pertinentes asi como el desarrollo de las reglas

en el programa.

```
4 referencias
public class Semantico
{
    List<string> intDefined = new List<string>();
    List<string> intValue = new List<string>();

    List<string> floatDefined = new List<string>();
    List<string> floatValue = new List<string>();

    List<string> voidFuncion = new List<string>();
    List<List<string>> voidParametros = new List<List<string>>();

    List<string> intFuncion = new List<string>();
    List<List<string>> intParametros = new List<List<string>>();

    List<string> floatFuncion = new List<string>();
    List<List<string>> floatParametros = new List<List<string>>();
```

De igual manera se crean tipos para cada uno de nuestros datos, esto aprovechando que c# nos da facilidades para manipular los punteros.

Implementar las comprobaciones semánticas:
En esta etapa, se implementan las reglas semánticas definidas anteriormente. Se recorre el árbol sintáctico y se aplican las comprobaciones correspondientes a cada nodo. Por ejemplo, se pueden verificar las asignaciones de tipos, la declaración y el uso correcto de variables, la resolución de nombres, la herencia adecuada, etc.
Manejar errores semánticos:
Durante el análisis semántico, es posible encontrar errores en el código fuente que violen las reglas semánticas. En esta etapa, se deben manejar estos errores de manera apropiada, generando mensajes de error descriptivos para ayudar al programador a corregir los problemas detectados.

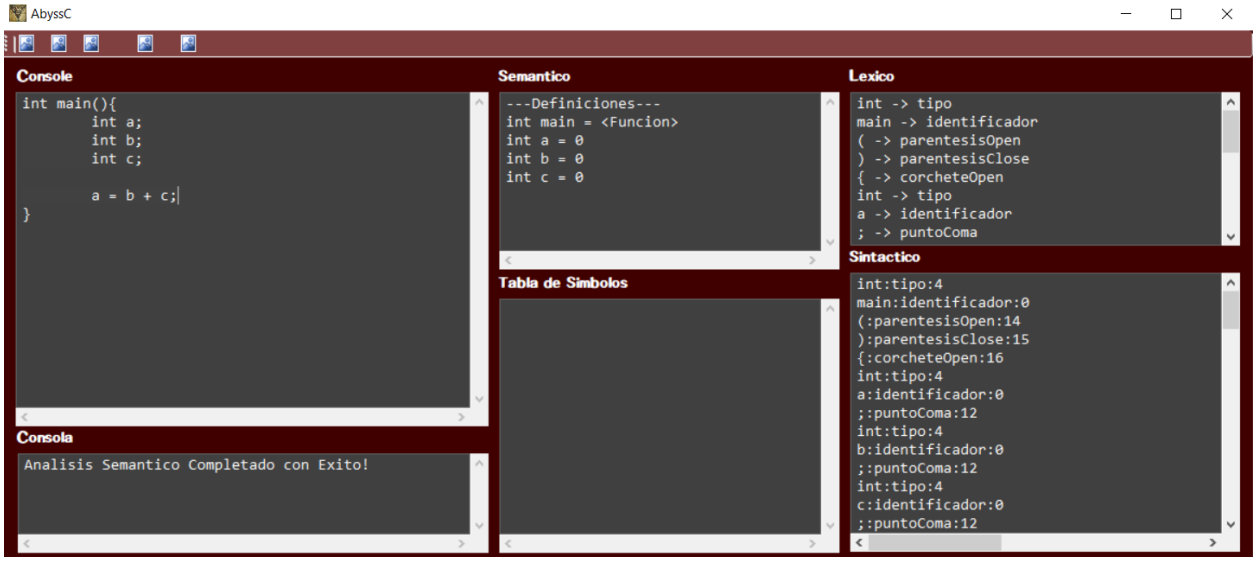
```
//Funciones
if (semantico.FindVoidFunction(variable2) > -1)
{
    Results_Console.Text += "'" + variable2 + "'" + " -> Incoherencia de tipo" + "\r\n";
    error = true;
}
else if (semantico.FindIntFunction(variable2) > -1)
{
    Results_Console.Text += "'" + variable2 + "'" + " -> Incoherencia de tipo" + "\r\n";
    error = true;
}
else if (semantico.FindFloatFunction(variable2) > -1)
{
}
else if (float.TryParse(variable2,out flotante))
{

}
else if (semantico.FindFloat(variable2) == -1)
{
    Results_Console.Text += "'" + variable2 + "'" + " -> Incoherencia de tipo" + "\r\n";
    error = true;
```

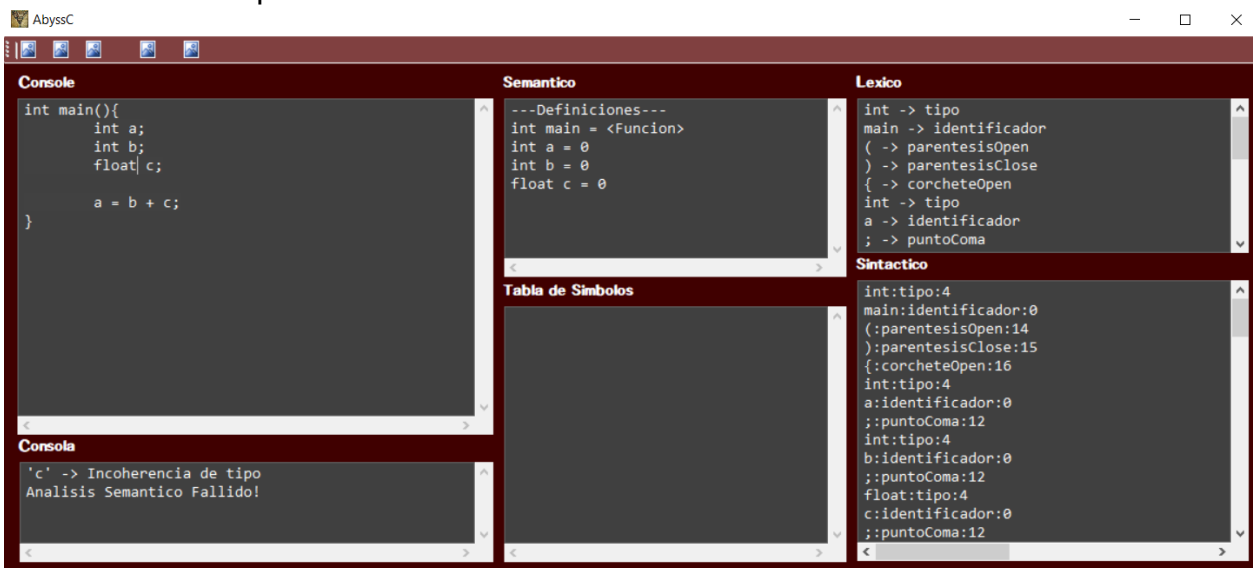
Para ello dependiendo la fase de revision que este sucediendo en nuestro analizador semantico, sabremos cual es el error segun la parte del arbol que falle.

Realizar pruebas y depuración:
Una vez implementado el analizador semántico, es esencial realizar pruebas exhaustivas para verificar su correcto funcionamiento.

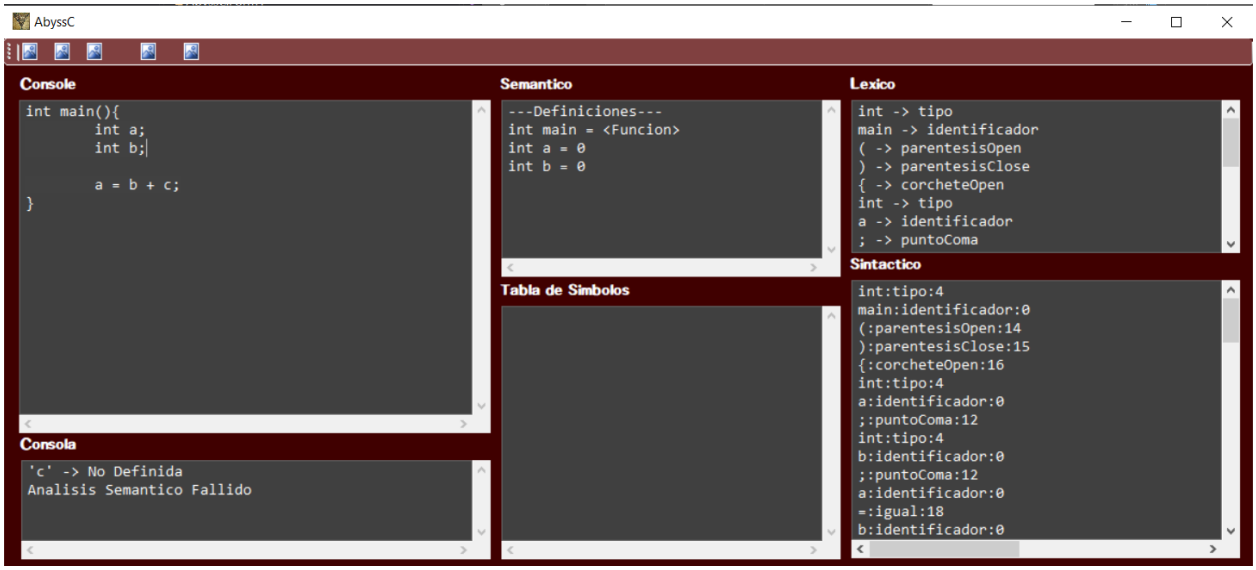
En seguida vemos un procesamiento que finaliza satisfactoriamente:



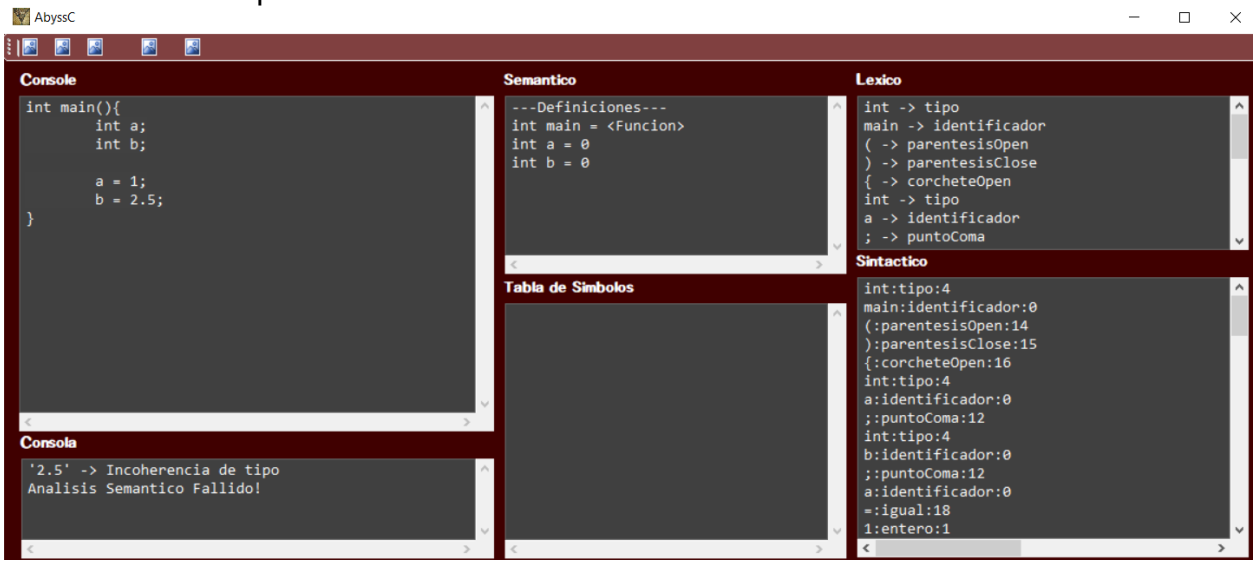
Ademas de la manera satisfactoria que es bien conocida, se muestran unos casos de errores donde tenemos:
Incoherencia de tipo:



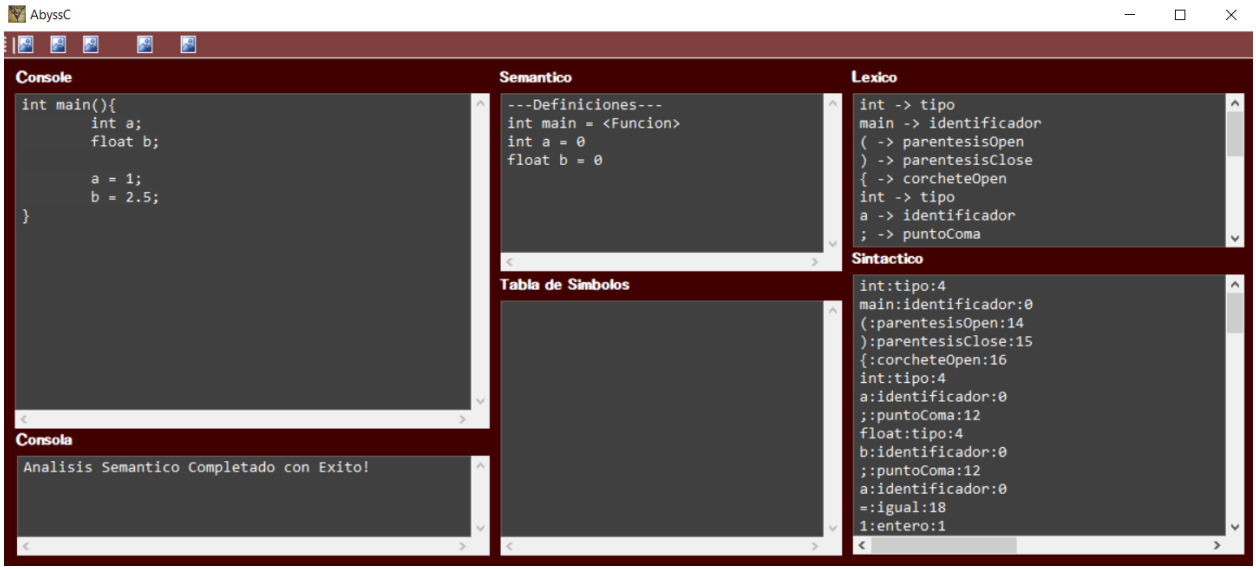
Variable no definida:



Incoherencia de tipo con valores directos:



En caso de colocar el tipo correctamente el error dejara de suceder:



Conclusiones:

El desarrollo de un analizador semántico en el contexto de la programación es un proceso fundamental para garantizar la corrección y la coherencia del código fuente.

A través de la implementación de reglas semánticas adecuadas y la realización de comprobaciones precisas, podemos identificar y prevenir errores en tiempo de compilación, mejorando así la calidad y la robustez de los programas.

Durante el desarrollo de un analizador semántico, es importante tener claras las reglas semánticas del lenguaje de programación en uso.

Esto nos permitirá establecer las restricciones y las verificaciones necesarias para garantizar la corrección semántica del código.

Además, es esencial comprender el funcionamiento del analizador sintáctico previo, ya que el análisis semántico se basa en la estructura sintáctica generada.

El árbol sintáctico es la base sobre la cual aplicaremos las comprobaciones semánticas correspondientes.

La implementación del analizador semántico implica diseñar y utilizar estructuras de datos eficientes para almacenar la información relevante del código fuente.

Estas estructuras nos ayudarán a realizar las comprobaciones semánticas de manera coherente y eficiente.

Asimismo, es necesario manejar adecuadamente los errores semánticos detectados durante el análisis.

Generar mensajes de error descriptivos permitirá al programador identificar y corregir los problemas encontrados.

Es importante destacar que el desarrollo de un analizador semántico requiere de pruebas exhaustivas y depuración para verificar su correcto funcionamiento.