INDICE

Compilador

Este compilador es una herramienta que traduce el código fuente escrito en Lenguaje C. Este documento describe el funcionamiento del analizador léxico, sintáctico, semántico y la tabla de símbolos implementada en el compilador.

La función **yyparse()** es invocada desde la función **main()** para iniciar el análisis. Luego, se imprimen los resultados obtenidos, incluyendo los tokens reconocidos, tokens de sintaxis, mensajes de error y la tabla de símbolos.

Compilación de los archivos de análisis léxico y sintáctico:

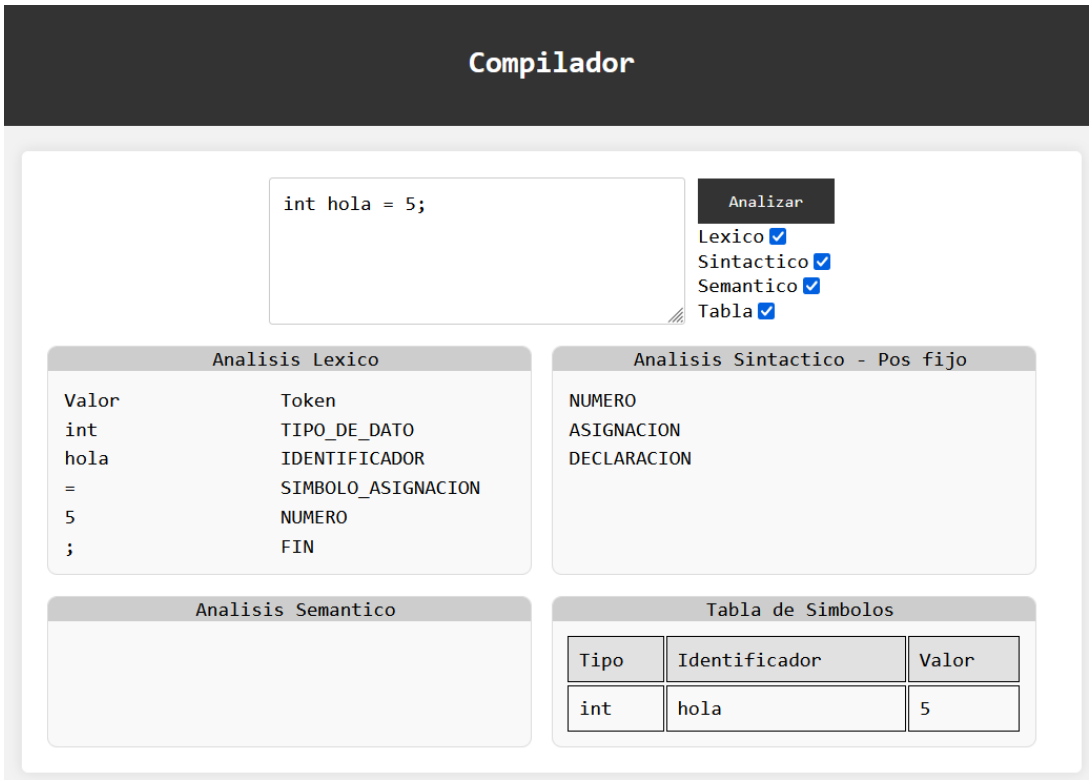
Al inicio del archivo index.php, se ejecutan tres comandos para compilar los archivos de análisis léxico (lexico.l) y sintáctico (sintactico.y).

- **flex lexico.l** genera el archivo lex.yy.c que contiene el analizador léxico.
- **bison -d sintactico.y** genera los archivos sintactico.tab.c y sintactico.tab.h que contienen el analizador sintáctico.
- **gcc -o analizador lex.yy.c sintactico.tab.c -lfl** compila estos archivos en un ejecutable llamado analizador.exe.

Interfaz de usuario

La interfaz de usuario está escrita en HTML y CSS. Consiste en un formulario donde los usuarios pueden ingresar el código a analizar y botones para iniciar el análisis.

Los resultados del análisis se muestran en tablas separadas para el análisis léxico, sintáctico, semántico y la tabla de símbolos.



Análisis del código

Cuando el usuario presiona el botón "Analizar", el código ingresado se envía al servidor PHP a través de una solicitud AJAX.

En el servidor, el código se pasa al ejecutable analizador.exe a través de la función shell_exec.

analizador.exe realiza el análisis léxico, sintáctico y semántico del código y devuelve los resultados vuelta al cliente en formato JSON.

Visualización de los resultados

Cuando se reciben los resultados del análisis, se muestran en las tablas correspondientes en la interfaz de usuario.

- Los resultados del análisis léxico muestran los tokens identificados y sus tipos.
- Los resultados del análisis sintáctico muestran la representación en postfijo de las expresiones.
- Los resultados del análisis semántico muestran los errores semánticos encontrados.
- La tabla de símbolos muestra los identificadores, sus tipos y sus valores.

Analizador Léxico

El analizador léxico es responsable de reconocer y dividir el código fuente en tokens, que son las unidades mínimas de significado en el código. Estos tokens representan diversas entidades como números, identificadores, palabras reservadas, símbolos, operadores, etc. Este análisis se realiza a través del archivo lexico.l.

Definición de Tokens

- **CONST_CADENA**: Reconoce cadenas constantes del tipo "[^\n"]*".
- **IDENTIFICADOR**: Reconoce identificadores que siguen el patrón [a-zA-Z][a-zA-Z0-9_]*.
- **TIPO_DE_DATO**: Identifica palabras clave que representan tipos de datos como int, char, float, bool, void.
- **PALABRA_RESERVADA**: Identifica palabras reservadas como if, else, for, while, do, printf, scanf, main.
- **IMPRIMIR_VAR**: Tokens que empiezan con & seguidos de un identificador.
- **NUMERO**: Reconoce números enteros o decimales.
- **SIMBOLO_ASIGNACION**: Representa el símbolo de asignación =.
- **COMPARACION**: Tokens que representan operadores de comparación como ==, <, >, <=, >=.

- **INCREMENTO**: Representa el operador de incremento ++.
- **FIN**: Token para el punto y coma ;.
- **COMA**: Token para la coma ,.
- **LLAVE**: Tokens para los corchetes { y }.
- **PAREN**: Tokens para los paréntesis (y).
- **OPERADOR**: Tokens para operadores aritméticos como +, -, *, /.
- **ERROR**: Representa cualquier carácter no reconocido.

Funciones Principales

- **add_token**: Agrega un nuevo token a la lista de tokens reconocidos.
- **print_tokens**: Imprime la lista de tokens reconocidos.

Análisis sintáctico:

El análisis sintáctico es la segunda fase de un compilador. Toma los tokens producidos durante el análisis léxico y verificar que los tokens formen una secuencia o estructura válida según las reglas gramaticales del lenguaje, las cuales se definen en la sección %% del código YACC en el archivo sintactico.y. Estas estructuras gramaticales se utilizan para generar el Árbol de Análisis Sintáctico (AST), que representa la estructura del programa.

Estructura y Funciones

El código comienza con las definiciones de estructuras y funciones auxiliares que se utilizan a lo largo del análisis sintáctico y léxico. Se definen estructuras para manejar errores, la tabla de símbolos, tokens y funciones para imprimir los resultados y realizar operaciones de manipulación de datos.

Funciones Principales

- **print_syntax_tokens**: Imprime la lista de tokens sintáctico en orden post-fijo.
- **add_syntax_token,**: Funciones para agregar tokens de sintaxis.

Tokens y No Terminales

- Se definen los tokens a través del uso de expresiones regulares como NUMERO, TIPO_DE_DATO, IDENTIFICADOR, CONST_CADENA, entre otros.
- Se definen precedencias de operadores y tipos para algunos no terminales (expresion, termino, factor).

Reglas de Gramática

Las reglas de gramática definen la sintaxis permitida del lenguaje de programación. Cada regla especifica una secuencia de símbolos terminales y no terminales que representan expresiones válidas en el lenguaje. Algunas reglas importantes son:

- **programa**: Define la estructura de un programa, especificando la secuencia de sentencias válidas.
- **sentencia**: Define las diferentes instrucciones válidas en el lenguaje, como expresiones, declaraciones, asignaciones, comparaciones, e instrucciones específicas.
- **funcion**: Reglas para definir funciones como if, else, while, do, for, y main.
- **instruccion**: Reglas para las instrucciones printf y scanf.
- **bloque**: Define la estructura de un bloque de código delimitado por llaves.
- **comparacion**: Define la estructura de una comparación entre expresiones.

Análisis Semántico

La parte semántica del compilador se encarga de verificar que las secuencias de tokens tengan un significado válido en el lenguaje. En este compilador, la verificación semántica se realiza en las acciones asociadas a las reglas gramaticales. Algunas de las verificaciones semánticas incluyen la verificación de tipos en las asignaciones, la verificación de la existencia de variables antes de su uso, y la verificación de la compatibilidad de tipos en las operaciones.

Tabla de Símbolos

La tabla de símbolos (tabla_simbolos) almacena información sobre las variables declaradas, sus tipos y valores. Durante la declaración (declaracion) de variables, se verifica si una variable ya ha sido declarada previamente y se agregan las nuevas variables a esta tabla.

Verificación de Tipos y Valores

La función *es_tipo_compatible* se utiliza para verificar si un valor es compatible con un tipo de dato específico (int, float, char, bool). Esto se aplica en la asignación de valores a variables para asegurar la coherencia entre los tipos.

Las reglas *asignacion* y *declaracion* verifican y asignan valores a las variables usando la funcion *agregar_simbolo*, asegurándose de que el tipo de dato coincida con la variable declarada.

Funciones Principales

- **es_tipo_compatible(char* tipo_dato, char* valor):** Verifica si un valor es compatible con un tipo de dato.
- **agregar_valor(char *identificador, char* valor):** Agrega un valor a un símbolo en la tabla de símbolos.
- **agregar_simbolo(char* tipo, char* identificador, char* valor):** Agrega un nuevo símbolo a la tabla de símbolos

- **buscar_indice_simbolo(char* identificador):** Busca el índice de un símbolo en la tabla de símbolos.
- **buscar_valor(char* identificador):** Busca el valor de un símbolo en la tabla de símbolos.
- **print_tabla_simbolos():** Imprime la tabla de símbolos, mostrando los tipos, identificadores y valores de las variables.

Pruebas

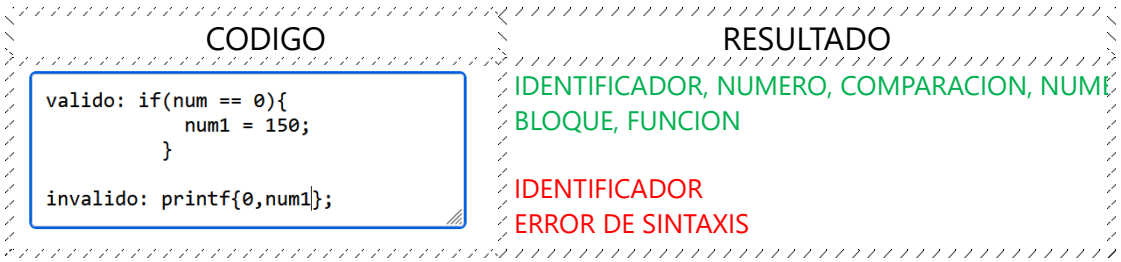
Esta parte de la documentación nos ayuda a validar que todo el compilador este funcionando correctamente al someterlo a distintas pruebas y ver su funcionamiento ante cada caso.

Pruebas Léxicas

- **Entradas Válidas** Proporciona programas pequeños con expresiones, declaraciones y asignaciones básicas que cumplan con la sintaxis del lenguaje.
- **Entradas Inválidas**: Introduce errores léxicos como tokens incorrectos, caracteres no reconocidos y ver cómo el compilador maneja estos errores

Pruebas Sintácticas

- **Programas Válidos**: Proporciona programas válidos con estructuras como bucles, condicionales, funciones y operaciones matemáticas. Verifica que se generen los tokens sintácticos correctamente.
- **Programas con Errores Sintácticos**: Introduce errores sintácticos como declaraciones incorrectas, estructuras mal formadas y verifica que se capturen los errores adecuadamente.



Pruebas Semánticas

- **Asignación y Declaración validas**: Verifica que las variables se declaren correctamente y que las asignaciones se realicen adecuadamente según los tipos de datos.
- **Asignación y Declaración invalidas**: Crea declaraciones a repetidas variables y asignaciones a variables que no se hay declarado y verifica que se capturen los errores adecuadamente.
- **Expresiones y Evaluación**: Crea expresiones matemáticas y verifica que el compilador realice las operaciones esperadas y genere los tokens semánticos correspondientes.

CODIGO

```
valido: int num;
        num = 5*8+(4/2);

invalido: nombre = "jose";
          float cantidad;
          int cantidad;
```

RESULTADO

Tipo	Identificador	Valor
int	num	42

ERROR: La variable 'nombre' no ha sido declarada
ERROR: variable 'cantidad' ya declarada

Pruebas de Salida

Impresión de Errores: Introduce errores a propósito y verifica que los mensajes de error se impriman correctamente.

Impresión de Tokens y Tabla de Símbolos: Comprueba que la visualización de los tokens léxicos, tokens sintácticos y la tabla de símbolos sea coherente con la entrada proporcionada

CODIGO

```
int num 5;
num + "hola"
```

RESULTADO

ERROR DE SINTAXIS
ERROR: Tipo de dato incorrecto para la variable 'num'