SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

CURSO K2006

GRUPO 9

TP N°3 GRUPAL

|  |  |
| --- | --- |
| INTEGRANTES | LEGAJO |
| Roberto Rodas | 2081301 |
| Alex Fiorenza | 2089865 |
| Oscar Mercado | 2083012 |
| Joaquín Fatur | 2079847 |

Link del repositorio: <https://github.com/OscarMercadoRuizDiaz/SSL/tree/master/03-FlexYBison>

Resolución del problema

El **análisis sintáctico** lo realiza Bison. En el archivo bison.y están definidos los tokens del lenguaje micro. Cada carácter, palabra clave y operador está en su propio token y también hay uno para el error léxico. En el archivo además está la gramática sintáctica de micro, el cual es la siguiente:

<programa> -> INICIO <listaSentencias> FIN

<listaSentencias> -> <sentencia> |

<listaSentencias> <sentencia>

<sentencia> -> ID ASIGNACION <expresion> PYCOMA |

LEER PARENIZQUIERDO <listaIdentificadores> PARENDERECHO PYCOMA |

ESCRIBIR PARENIZQUIERDO <listaExpresiones> PARENDERECHO PYCOMA

<listaIdentificadores> -> ID |

<listaIdentificadores> COMA ID

<listaExpresiones> -> <expresion> |

<listaExpresiones> COMA <expresion>

<expresion> -> <primaria> |

<expresion> SUMA <primaria> |

<expresion> RESTA <primaria>

<primaria> -> ID |

CONSTANTE |

PARENIZQUIERDO <expresion> PARENDERECHO

Los no terminales están escritos en mayúscula mientras que los terminales en minúscula. Se modificó ligeramente la GIC respecto del libro de Muchnik para aprovechar la recursividad a izquierda y también para facilitar la implementación de las operaciones aditivas.

El **análisis léxico** lo realiza Flex. En el archivo flex.l definimos los patrones que retornan el token correspondiente. Además, pusimos dos patrones adicionales: el espacio en blanco que no realiza ninguna acción y un patrón al final para que matchee con el lexema que no coincidió con ningún patrón anterior, el cual lanza un error léxico.

El **análisis semántico** está escrito en el archivo bison.y. Las rutinas semánticas están implementadas como acciones de las reglas gramaticales de Bison, las cuales algunas ejecutan funciones definidas en TS.c, como asignar y chequear. Estas funciones van completando la tabla de símbolos (TS). La TS que implementamos no es más que un vector de mil posiciones de tipo RegTS, el cual es un struct que tiene un campo para el nombre del identificador y otro campo para el valor de este. En la TS siempre va a haber una posición en la que su campo nombre tiene “$”, que sirve como centinela y se va moviendo según se requiera.

Las palabras reservadas no están en la TS porque no es necesario ya que cuando Flex lee una palabra reservada no lo reconoce como identificador, sino como el token de la palabra correspondiente, y cuando Bison recibe un token de una palabra reservada en un lugar no válido tira un error sintáctico.

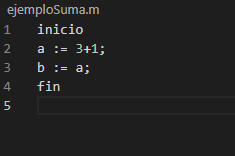
El código de la TS estaba originalmente en bison.y pero al compilar no reconocía RegTS como tipo de dato, incluso cuando estaba en la sección de declaraciones en C, por lo que lo pusimos en su archivo propio y lo incluimos en Bison como header.

Manual de usuario

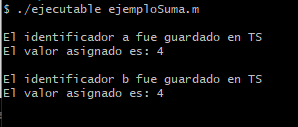
El programa “ejecutable.exe” es un compilador del lenguaje micro que realiza la parte de análisis del proceso de compilación. Para usarlo debe ejecutarlo por líneas de comando escribiendo: *ejecutable.exe nombreDelArchivo.m*. No acepta otro tipo o cantidad de argumentos.

Una vez que el programa comienza, se realizará el análisis léxico, sintáctico y semántico del código escrito en el archivo pasado al compilador. Si el programa termina y no muestra ningún error, significa que no se ha detectado ningún error.

El programa puede realizar sumas y restas y las asigna al identificador, por lo que podrá hacer esto:



Salida:

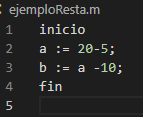


También se puede apreciar que el programa muestra cuando un identificador se guarda en la tabla de símbolos y el valor que le fue asignado.

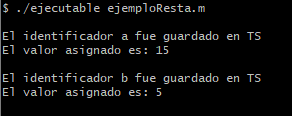
Funcionamiento del programa

Como se ha mencionado, el programa no avisa cuando no encuentra errores, pero sí avisa cuando los encuentra. A continuación unos ejemplos del funcionamiento del compilador:

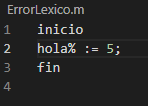
Ejemplo de restas:



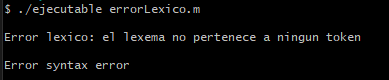
Salida:



Ejemplo de error léxico:

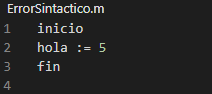


salida:



El lexema al que se refiere es % y, como el error lo dice, no pertenece a ningún token, por lo que se produce un error léxico y también se produce un error sintáctico ya que el parser esperaba un :=.

Ejemplo de error sintáctico:

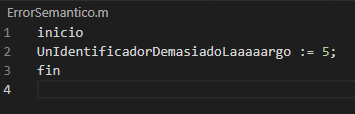


Salida:

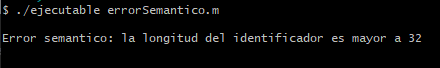


El error es bastante sutil, solamente falta el punto y coma, pero como esto no forma parte de la gramática sintáctica de micro, tira un error sintáctico.

Ejemplo de error semántico:

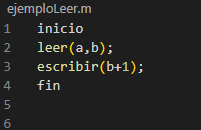


salida:

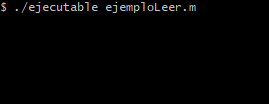


Esto es debido a que micro no acepta indicadores de más de 32 caracteres, por lo que se produce un error semántico.

Ejemplo de leer y escribir:



salida:



El compilador no escribe nada, por lo que no encontró ningún error, esto es correcto. Las rutinas de lectura y escritura no están implementadas por lo que ejemploLeer.m no realiza nada.