[[1]](#footnote-1)

Analizador Sintáctico y Tabla de Símbolos

Alex Morales, *Estudiante, EPN*, Wilson Ramos, *Estudiante, EPN*

*Resumen*—El informe a continuación detalla el planteamiento, diseño e implementación de un analizador sintáctico y su respectiva tabla de símbolos usando el lenguaje de programación C y Bison. Junto con el analizador léxico implementado previamente se completa el diseño del compilador.

*Abstract*—The report below details the approach, design and implementation of a syntactic analyzer and its symbol table using the C programming language and Bison. Together with the previously implemented lexical analyzer, the compiler design is completed.

*Index Terms*—código, compilador, expresiones regulares, instrucción, lenguaje, token, lexemas, símbolos.

# INTRODUCCIÓN

L

A traducción de un lenguaje a otro, pasando de un pseudocódigo a un lenguaje de alto nivel, requiere de un análisis de diferentes tipos para asegurar que las reglas definidas en el pseudocódigo se cumplan y se puedan transformar utilizando un compilador.

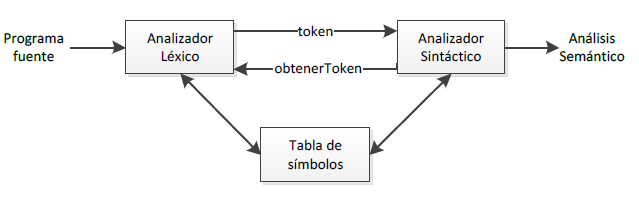
Un compilador es un traductor de alto nivel a un lenguaje de máquina o denominado lenguaje de bajo nivel. Cuando se pasa de un lenguaje a otro se debe tener en cuenta una serie de reglas que nos aseguren que lo descrito por nosotros será entendido y ejecutado por el computador en lenguaje de bajo nivel [1], es por eso que el compilador realiza un serie de pasos que permiten que este lenguaje sea comprendido e interpretado por el computador. Estos pasos analizan que la forma de escribir sea la correcta (Analizador Léxico), que este bien escrito (Analizador Sintáctico) y que este tenga sentido (Analizador Semántico).

# objetivos

El objetivo principal de este trabajo es aprender y comprender la implementación de un compilador.

El principal objetivo del analizador léxico es leer el flujo de caracteres de entrada y transformarlo en una secuencia de componentes léxicos, el analizador léxico se ocupa de eliminar los espacios en blanco o los comentarios, para simplificar los símbolos se agrupan en lo que se denomina categorías léxicas, donde tendremos que especificar qué elementos componen y que son las expresiones regulares.

Todos los lenguajes de programación tienen reglas precisas, las cuales definen una estructura sintáctica. El parser utiliza los primeros componentes de los tokens producidos por el analizador léxico para crear una representación en forma de árbol que describa la estructura gramatical del flujo de tokens.



*Fig. 1. Interacciones del analizador léxico y sintáctico*.

# Analizador sintáctico

El analizador sintáctico (parser) es donde se analiza la estructura gramatical del lenguaje fuente, estas reglas son representadas por las gramáticas libres de contexto y su escaneo de árboles sintácticos. Para la implementación se utilizar la herramienta *Bison,* usando el sistema operativo Ubuntu 16.04.

*II-A Bison*

Bison es un programa para generar analizadores sintácticos de propósito general perteneciente al proyecto GNU disponible para prácticamente todos los sistemas operativos, se usa normalmente acompañado de flex aunque los analizadores léxicos se pueden obtener de otras formas.

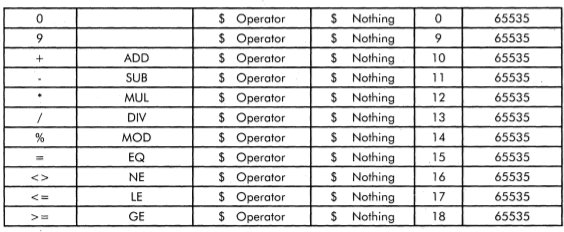
Bison convierte la descripción formal de un lenguaje, escrita como una gramática libre de contexto, en un programa en C, C++ o Java que realiza un análisis sintáctico. Es utilizado para crear analizadores para muchos lenguajes, desde simples calculadoras hasta lenguajes complejos. Para utilizar *Bison*, es necesaria una experiencia con la sintaxis usada para describir gramáticas. GNU bison tiene compatibilidad con Yacc: todas las grmaáticas bien escritas para Yacc.

Bison fue escrito en un principio por Robert Corbelt; Richard Stallaman lo hizo compatible con Yacc y Wilfred Hansen de la Carnegie Mellon University añadió soporte para literales multicaracter y otras características.

# Tabla de símbolos

La tabla de símbolos “son estructuras de datos que utilizan los compiladores acerca de las construcciones de un programa fuente” [2]. La tabla de símbolos almacena de forma volátil los lexemas encontrados en el programa fuente y la información adicional necesaria para la generación de código, como su cadena de caracteres (lexema), su tipo, su posición en el espacio de almacenamiento, y cualquier otra información relevante. Por lo general la tabla de símbolos debe soportar varias declaraciones del mismo identificador dentro del programa.

Para que el analizador léxico sea capaz de asignar un lexema (o flujo de caracteres) a un token [3], es necesario especificar cada unidad léxica en un patrón determinado. La herramienta utilizada para realizar dicha especificación, es una expresión regular [4].

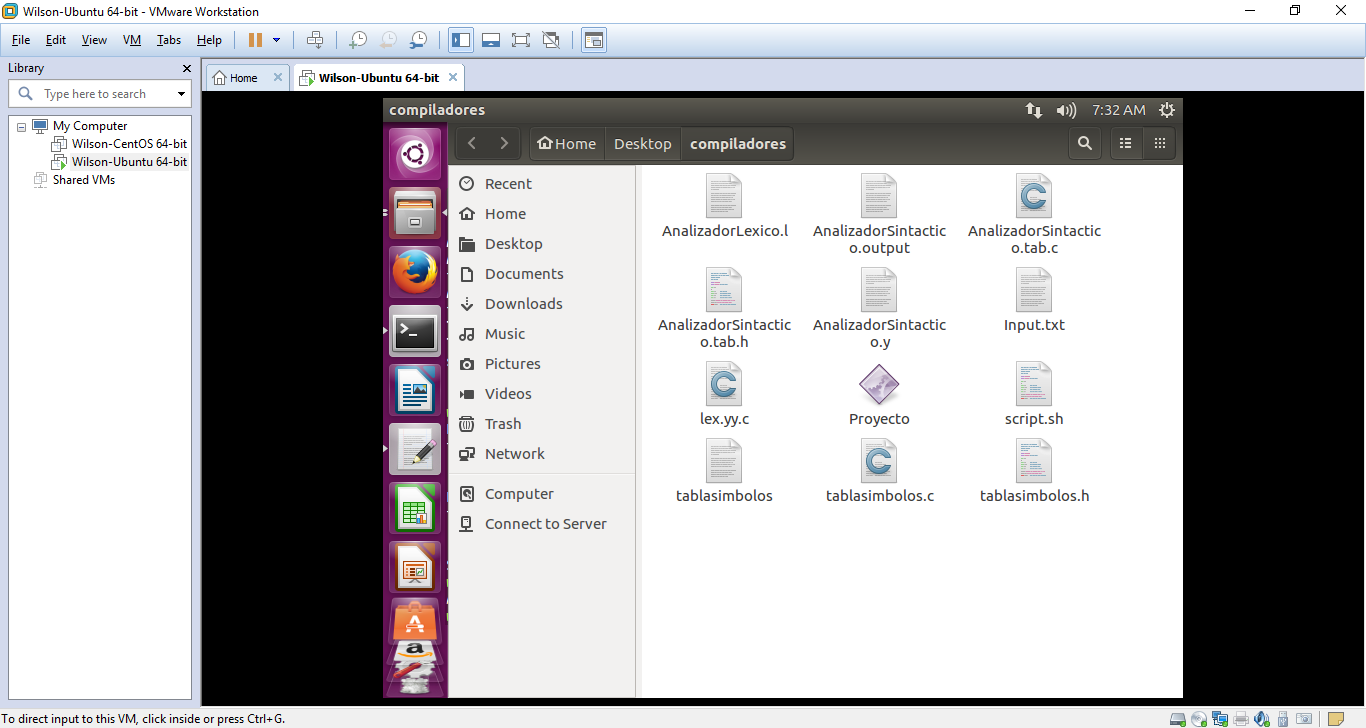


*Fig. 2: Tabla de Símbolos*

# Resultados y discusión

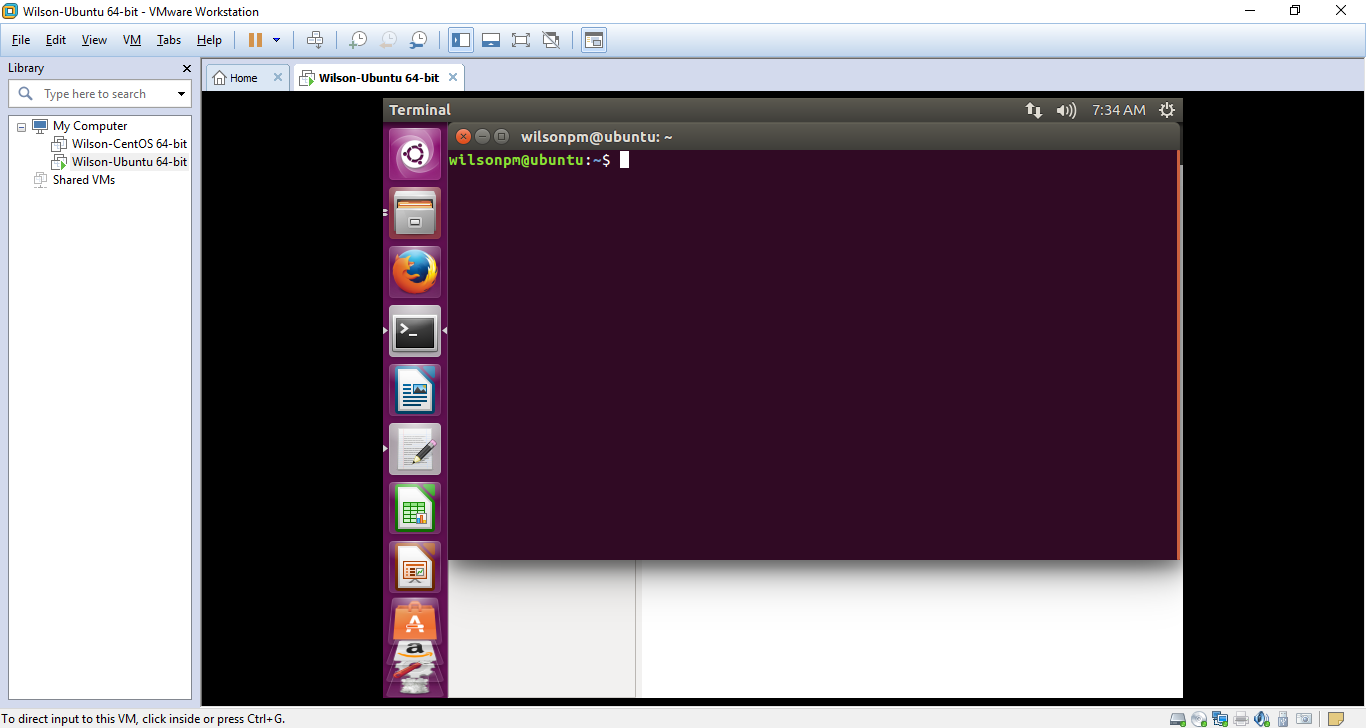
En los resultados se obtuvo la tabla de símbolos [5], la cual guarda todos los elementos que se han leído por el programa.

El programa leerá el txt, denominado como “Input”, encontrando errores en el cmd, de hallarlas y el de guardar los valores de variables en la tabla de símbolos.



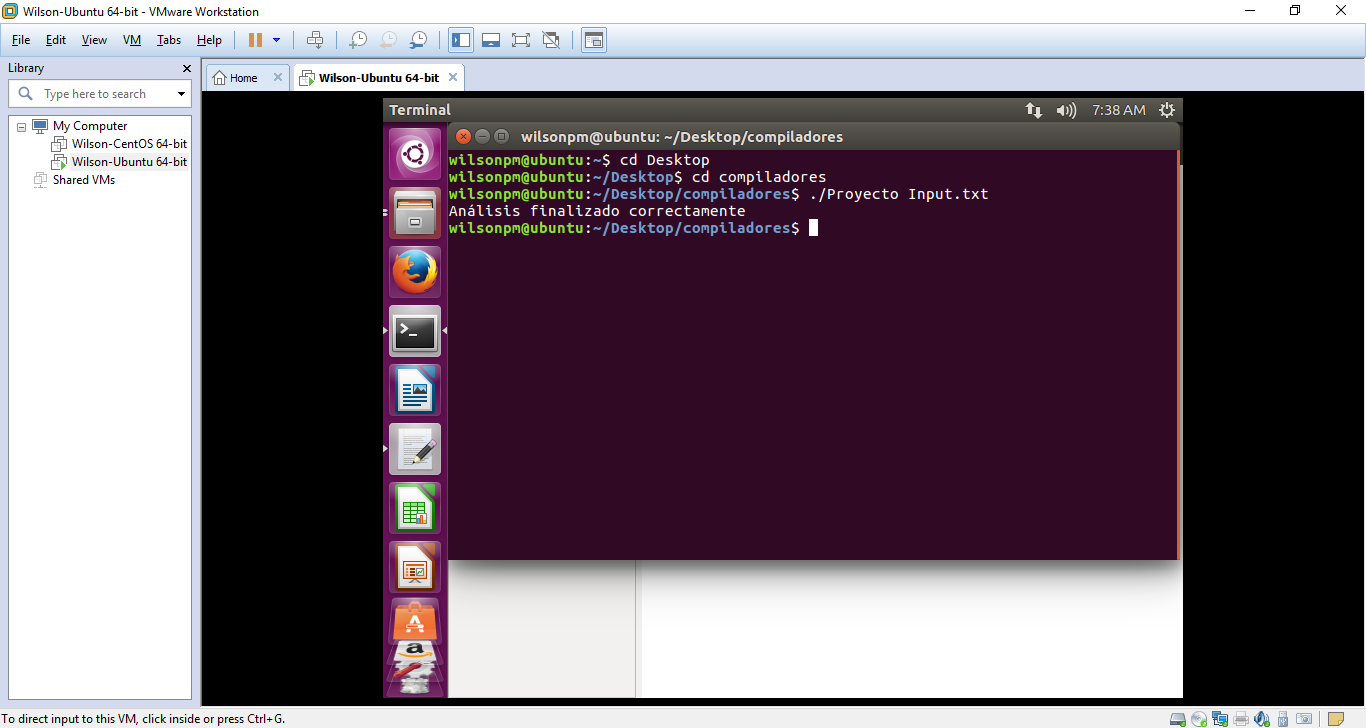
*Fig. 3: Input txt*

Procedemos a abrir un cmd, y lo direccionamos hacia la carpeta en donde tenemos el ejecutable, esto con el comando cd.



*Fig. 4: CMD en Linux*

Ahora escribimos el comando para ejecutar el programa, el cual tiene por nombre “Proyecto”. El comando es ./Proyecto Input.txt, con esto se analizará el documento en busca de errores y en caso de no haberlos devolverá un análisis correcto.

**

*Fig. 5: Ejecución sin errores*

Podemos ver entonces, que no existen errores en el txt, el cual consta de varias instrucciones.

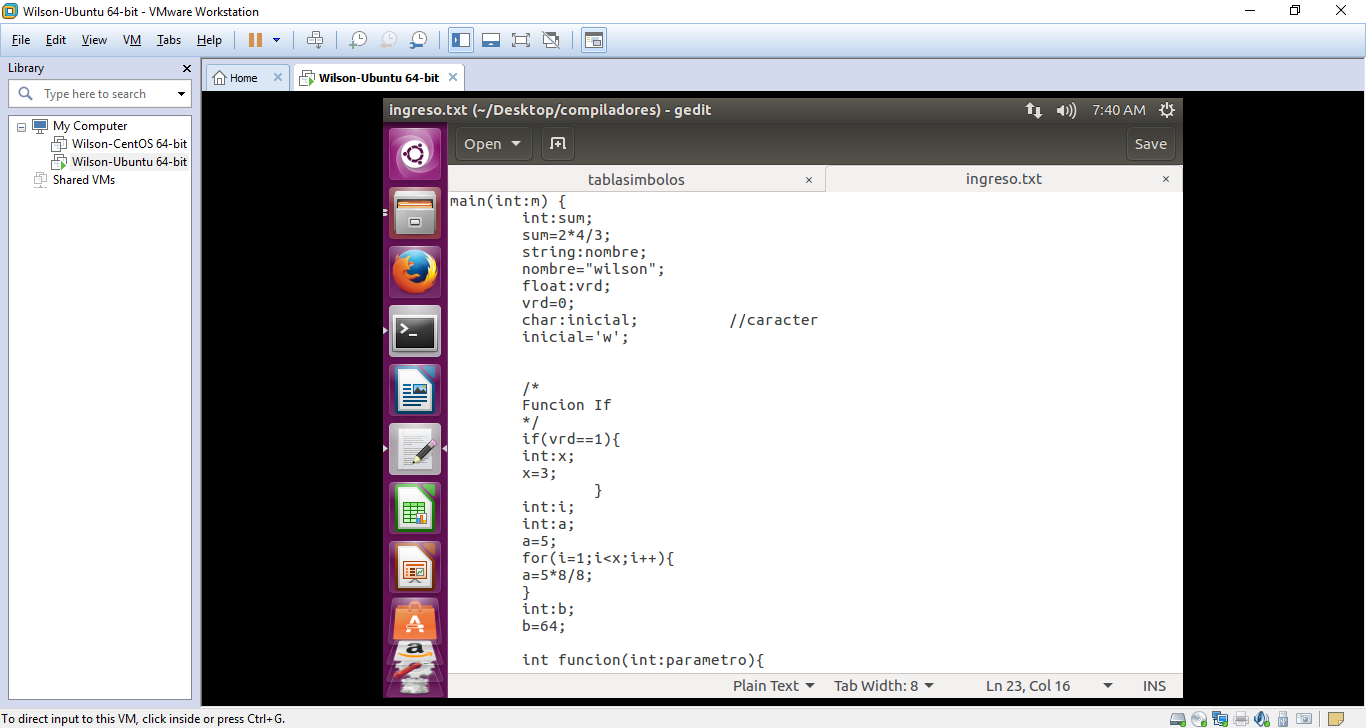


Fig. 6: Input txt sin errores

Ahora comprobamos la tabla de símbolos con los resultados guardados de la ejecución del programa.

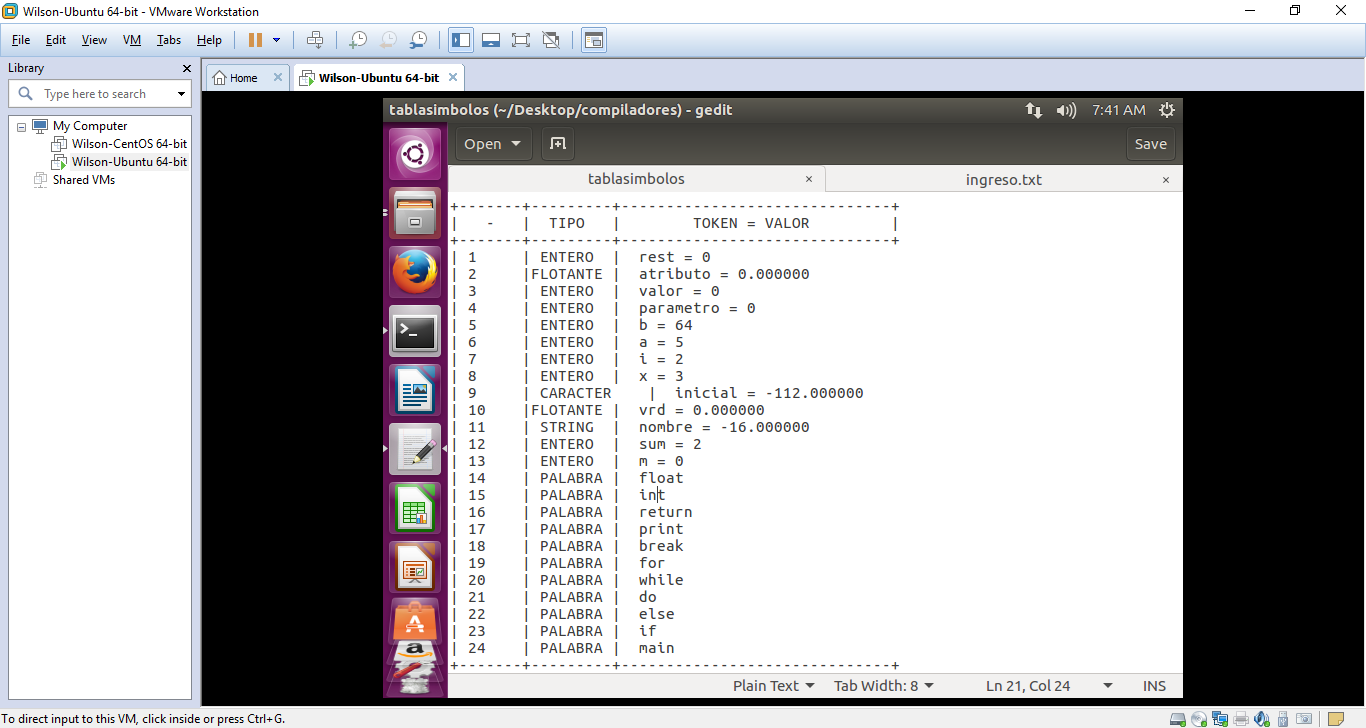


Fig. 7. Tabla de símbolos Input.txt

Estos son los resultados en un ingreso sin errores, se puede notar que las variables son guardadas de acuerdo a su declaración y asignación en el txt [6].

Se puede mencionar entonces que el analizador funciona de una manera correcta, al generar la tabla a partir del input, esto se realiza también cuando el análisis tiene errores, sin embargo, aquí se perderán ciertas variables.

Ahora se mostrará los resultados con errores de análisis, el error será no definir una variable [7].

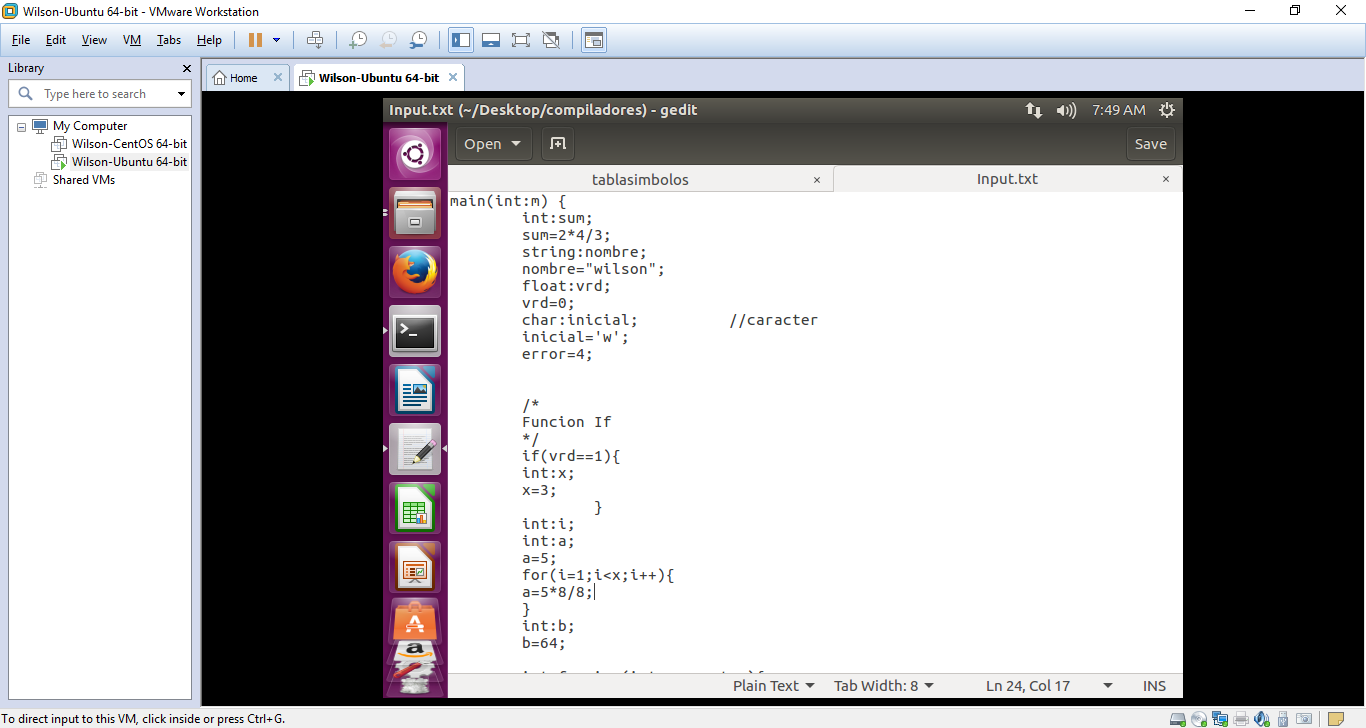


Fig. 8. Input.txt con un error.

El error está definido por su propio nombre, y se le asigna un valor cualquiera, sin haberlo declarado con ningún tipo y por ende generando así un error.

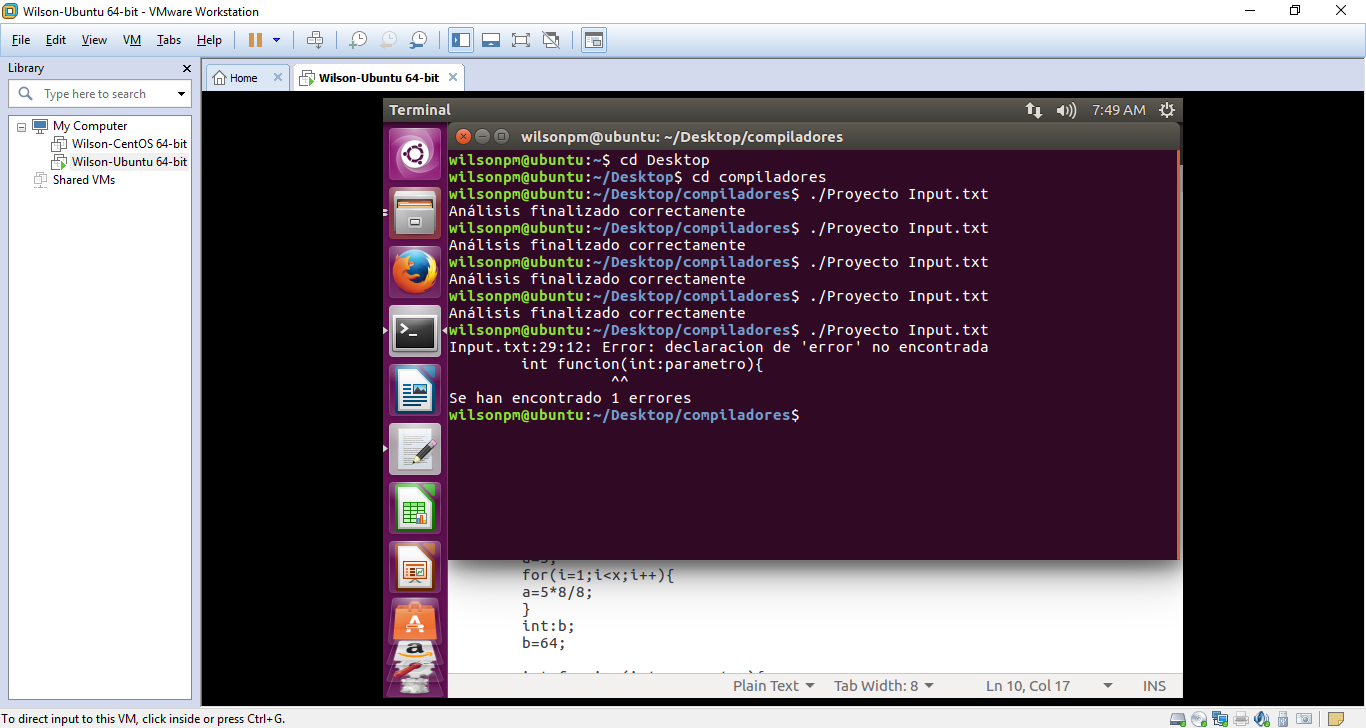


Fig. 8. Error por definición

Entonces se da el error en la línea 29, en la cual se finaliza la función main que contenía al error, por ende, al no encontrarse asignación de “error”, se muestra en el cmd dicha cuestión.

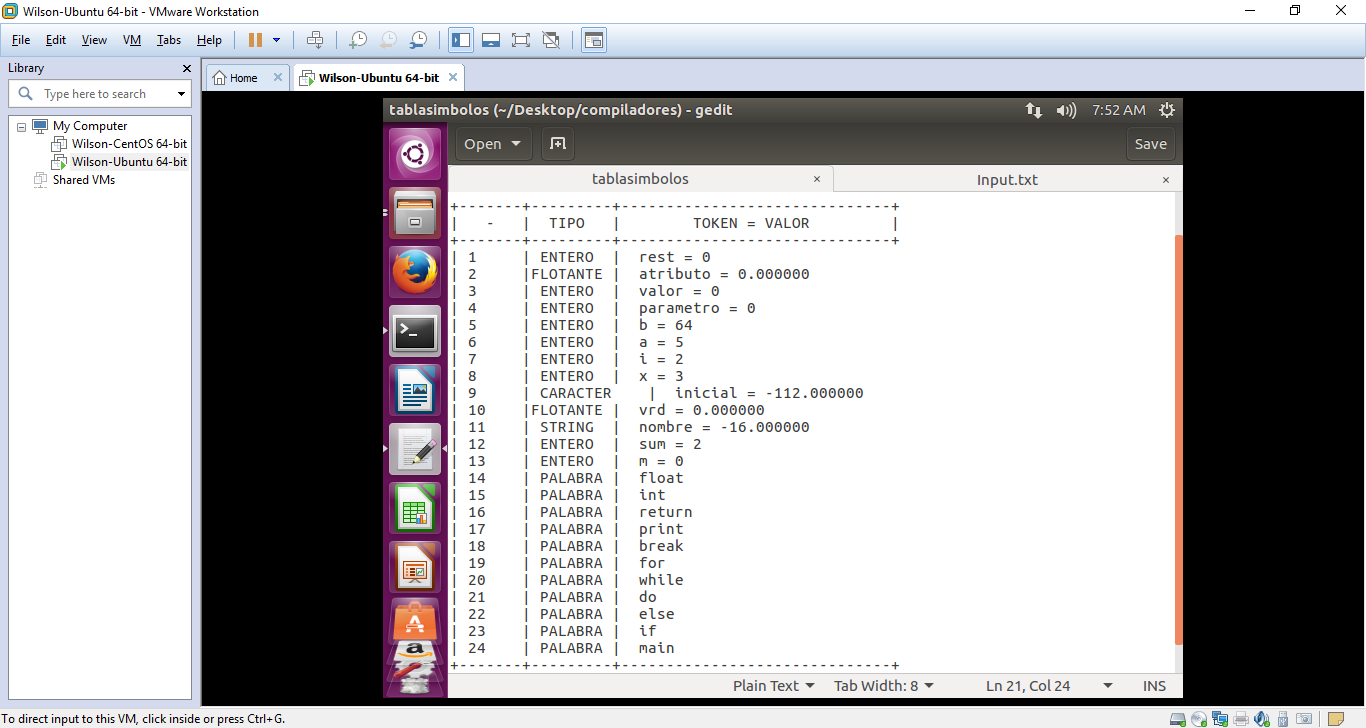


Fig. 10. Tabla de símbolos con errores

Así se muestra entonces la tabla de símbolos cuando se dio un error de análisis. Se observa que las variables sin problema, son almacenadas, sin embargo, las variables que incurrían en error, no son tomadas y se excluyen.

Entonces, con los ejemplos de ejecución errónea y libre de errores, se puede notar el correcto almacenamiento por la tabla de símbolos. Por lo tanto, el analizador construido en Linux, es funcional y puede ayudar a reconocer códigos en un nivel básico [8].

# Conclusiones

* Para que un analizador léxico cumpla su función principal, entregar un flujo de tokens, para su infraestructura debe contar con diagramas que describan los lexemas sin ninguna ambigüedad.
* En un lenguaje de programación todas las palabras reservadas son identificadores válidos, en este caso para solventar este problema se colocó en primer lugar las expresiones regulares que identifican dichas palabras para luego colocar los identificadores.
* Una vez finalizado el trabajo, se puede apreciar que siguió las normas establecidas para la creación de un analizador sintáctico.
* Este, en primer lugar, hace uso del analizador léxico, desarrollado con anterioridad y del analizador sintáctico, usando funcionalidades de flex y bison.
* Se cumple el objetivo principal del analizador, el cual era formar la tabla de símbolos y compilar y detectar ciertos errores de programación, esto en base a reglas bien definidas.
* Cabe mencionar que el trabajo en Linux, como sistema operativo, facilitó la creación e implementación de este analizador.
* Es así, que se recomienda utilizarlo en proyectos que involucren la creación de compiladores.

Referencias

1. A. Garrido-Alenda and M. L. Forcada, “Morphtrans: un lenguaje y un compilador para especificar y generar módulos de transferencia morfológica para sistemas de traducción automática.” Procesamiento del Lenguaje Natural, vol. 27, pp. 157–164, 2001.
2. AHO, Compiladores: Principios, Técnicas y Herramientas, 2/ed. Pearson Education, 2008.
3. F. J. Sanchis Llorca and C. G. Pascual, “Compiladores: teoría y construcción,” Paraninfo, 1986.
4. F. J. C. Limón, «Galeon.com,» 2015. [En línea]. Available: <http://10380054.galeon.com/u5.htm>
5. V. Paxson, «TLDP,» Abril 1995. [En línea]. Available: <http://es.tldp.org/Manuales-LuCAS/FLEX/flex-es-2.5.html#SEC2>.
6. J. Ezpeleta, «Universidad de Saragoza,» 2013. [En línea]. Available: <http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:compi:2bis.introflex.pdf>.
7. D. J. B. Torralvo, «Universidad Politécnica de Madrid,» Junio 2014. [En línea]. Available: <http://oa.upm.es/32288/1/PFC_JOSE_BARBERA_TORRALVO.pdf>.
8. S. Autor, «Tutos.com,» 2013. [En línea]. Available: <http://www.unse-prog2.comxa.com/downloads/Tutorial%20de%20Instalacion%20y%20utilizacion%20de%20Flex-bison.pdf>.

VI. BIOGRAFÍAS

**Wilson Gabriel Ramos Bravo**, nació el 28 de noviembre de 1996 en Quito - Ecuador, hijo de Wilson Ramos y Lupe Bravo, es el tercero de sus hermanos. Comenzó sus estudios en la Escuela Fiscal Mixta “Gonzalo Zaldumbide” en la cual obtuvo el honor de llegar a ser abanderado del pabellón Nacional. Estuvo en el “Instituto Nacional Mejía” institución en la cual alcanzo el mérito de ser segundo escolta del pabellón Nacional del Ecuador. En la universidad ingreso a la Escuela Politécnica Nacional para seguir su sueño que está en la ingeniería en sistemas informáticos y de computación. Actualmente cursa el cuarto semestre de dicha carrera. Como pasatiempo le gusta toca la guitarra, jugar videojuegos, salir con amigos y aprender un poquito más del medio que le rodea. Su visión está planteada en utilizar sus conocimientos en el ámbito astronómico, en el cual Ecuador tiene falencias o poca preocupación.

Alex Javier Morales Cóndor, nació el 4 de septiembre de 1996 en Quito – Ecuador, hijo de Victor Morales e Isabel Cóndor, es el tercero de cuatro hijos. Comenzó sus estudios en la Escuela Fiscal Mixta “Joaquín Gallegos Lara”, continuando sus estudios en la escuela particular #Niño Jesús de Praga”. Estuvo en el Colegio Nacional Mixto “Eloy Alfaro” donde obtuvo el título de bachiller Físico Matemático. Comenzó sus estudios en la Escuela Politécnica Nacional en el 2010 para conseguir el título de Ingeniero en Electrónica y Redes de la Información, pero cambio su área de estudio en el 2015 a Sistemas Informáticos y de Computación. Como pasatiempo disfruta de leer y escuchar música, practicar deportes como el ciclismo y correr.

1. [↑](#footnote-ref-1)