

INGENIERÍA EN SISTEMAS COMPUTACIONALES

LENGUAJES Y AUTÓMATAS I

SEMESTRE VI

6J1

EQUIPO DE TRABAJO:

- PABLO GARCÍA LARA
- GUILLERMO ANTONIO GUZMÁN
VERGARA
- JOSE CARLOS XOLIO XOLIO

CATEDRÁTICO:

M.E OFELIA GUTIÉRREZ
GIRALDI

LENGUAJE MPL



Contenido

Introducción	3
Lenguaje MPL	4
Objetivos	4
Propósito	4
Logo	4
Marco teórico.....	5
Lenguaje	5
Compilador	5
Análisis léxico.....	5
Análisis sintáctico	6
Herramientas utilizadas en el desarrollo del lenguaje y aplicación MPL	8
JavaCC™	8
CMD	10
NetBeans	11
Definición del lenguaje MPL	12
Palabras reservadas	12
Condicionales.....	12
Ciclos	12
Operadores lógicos	13
Operadores aritméticos	13
Operadores de asignación	13
Comparación	13
Delimitadores	14
Tipos de datos.....	14
Dígitos	14
Variables	14
Símbolos especiales	14
Manual del usuario para la construcción de códigos basados en el lenguaje MPL	15
Desarrollo del lenguaje y aplicación MPL (Analizador léxico & sintáctico MPL)	15
Tabla de tokens del lenguaje MPL	15

Estructuras gramaticales del Lenguaje MPL.....	17
Estructura general	17
Declaración principal.....	17
Declaraciones de tipos de datos	18
Leer y guardar un dato o variable	18
Imprimir	19
Creación de ciclos (mientras, para, hacer) y condicionales (si, sino)	19
Operaciones básicas.....	21
Manual del usuario para manejo de la interfaz grafica.....	22
Descripción.....	22
Ventanas principales para análisis léxico y sintáctico	23
Componentes que integran la interfaz del usuario.....	23
1.- Barra de titulo	24
2.- Barra de menú.....	24
3.- Área de escritura-código.....	26
4.- Área de botones de acción rápida	26
5.- Área de resultados.....	26
Tareas que se pueden realizar haciendo uso de la interfaz:.....	27
Abrir una nueva ventana	27
Abrir un archivo existente.....	27
Crear un nuevo archivo (Guardar como)	28
Guardar cambios de un archivo	28
Ejecutar un análisis del código.....	28
Limpiar el área de escritura-código	28
Abrir manual de usuario	29
Imprimir	29
Salir.....	29
Conclusión	30
Referencias	31

Introducción

Un compilador es un programa que lee un código escrito en un lenguaje fuente para traducirlo en un programa equivalente en otro lenguaje, el cual se denomina lenguaje destino. A su vez, un compilador forma parte de la clasificación de los traductores, que son un programa que convierte un lenguaje de origen en un lenguaje destino.

El proceso de compilación opera con una secuencia de fases, actualmente bien definidas y sistematizadas, dentro de las cuales el proyecto anterior abarcó la primera fase: el análisis léxico o escaneo. Dicha fase suministra tokens para la siguiente fase del compilador, la cual corresponde a este proyecto.

La tarea que se realizó para construir el analizador sintáctico consistió en determinar la estructura sintáctica de un programa a partir de los tokens producidos por el analizador léxico (la entrega anterior realizada) y, ya sea de manera explícita o implícita, construir un árbol de análisis gramatical o árbol sintáctico que represente esta estructura. De esta forma, se puede ver el analizador sintáctico como una función que toma como su entrada la secuencia de tokens producidos por el analizador léxico y que produce como su salida el árbol sintáctico, aunque en el presente proyecto realizado, se muestra un mensaje indicando que el programa escrito tiene la sintaxis correcta, en su defecto se mostrará el error ocurrido.

También, como desarrollo de este proyecto se realizó la creación de un lenguaje de programación, el cual será reconocido y comprobado por el analizador sintáctico diseñado; en el presente documento se encuentran los detalles sobre este proyecto.

Lenguaje MPL

MPL (Mi Primer Lenguaje) es nombrado así debido a que era la primera vez que los desarrolladores crearon un lenguaje de programación. Fue desarrollado por los estudiantes Pablo García Lara, Guillermo Antonio Guzmán Vergara y José Carlos Xolio Xolio; además, desarrollaron la aplicación MPL.exe, que reconoce su lenguaje y realiza el análisis léxico o sintáctico del mismo.

Objetivos

General

El principal objetivo del lenguaje es lograr facilidad de uso sin comprometer su eficiencia, esto para el momento en el que el usuario maneje el lenguaje, haciendo uso del conjunto de palabras y símbolos que forman parte de la estructura del lenguaje.

Específicos

- Facilitar la sintaxis de las sentencias que se utilizarán para programar.
- Desarrollo de un entorno gráfico fácil de manipular para el usuario.
- Motivar el interés por este tipo de programación.

Propósito

Creación un lenguaje sencillo destinado a programadores inexpertos y personas con fundamentos básicos de programación interesados en este ámbito. Además de contar con una interfaz que, aunque parezca sencilla, permite trabajar en un ambiente amigable para los usuarios.

Logo



Marco teórico

Lenguaje

Un lenguaje de programación consiste en todos los símbolos, caracteres y reglas de uso que permiten a las personas "comunicarse" con las computadoras. Existen varios cientos de lenguajes y dialectos de programación diferentes. Algunos se crean para una aplicación especial, mientras que otros son herramientas de uso general más flexibles que son apropiadas para muchos tipos de aplicaciones. En todo caso, los lenguajes de programación deben tener instrucciones que pertenecen a las categorías ya familiares de entrada/salida, cálculo/manipulación de textos y lógica/comparación. Aunque todos los lenguajes de programación tienen un conjunto de instrucciones que permiten realizar dichas operaciones, existe una marcada diferencia en los símbolos, caracteres y sintaxis de los lenguajes de máquina, lenguajes ensambladores y lenguajes de alto nivel.

Compilador

Como se mencionó, el compilador es un traductor el cual su funcionamiento es traducir un lenguaje escrito a nivel de programador a un lenguaje a nivel máquina donde el hardware lo deberá comprender. Se compone de manera interna en fases o etapas (cinco o seis dependiendo del autor al que se haga referencia) las cuales hacen diferentes operaciones lógicas y se integran todas las partes para dar como resultado un traductor eficiente. Los compiladores tienen una función estelar dentro de la informática ya que sin estos no sería tan sencillo comunicarse con la máquina y no existiría el software, las aplicaciones o los lenguajes de programación como se conocen hoy en día. Un compilador cuenta con diferentes etapas, pero se pueden dividir en dos grandes bloques el primero se conoce como el análisis donde divide el programa fuente en componentes e impone una estructura gramatical sobre ellas, en este bloque se encuentran la parte del análisis léxico, el sintáctico y el semántico. El siguiente bloque es la síntesis aquí están la parte de la generación de código intermedio, la optimización de código fuente y la generación de código objeto.

Análisis léxico

La primera etapa conocida como análisis léxico o rastreo, es la que efectúa la lectura real del programa fuente que comúnmente se encuentra en la forma de un flujo de caracteres. El analizador lee el flujo de caracteres que componen al programa y se producen cadenas de caracteres en unidades llamadas tokens las cuales son palabras de un lenguaje natural, como el inglés o español; algunos ejemplos de tokens en los lenguajes de programación serían if, then, else, where, etc. Además, el analizador léxico puede hacer diferentes funciones, por ejemplo, este puede introducir identificadores en la tabla de símbolos y puede producir literales en la tabla de literales, por mencionar alguno puede ser el valor de pi. La función que hace el analizador léxico es la del reconocimiento de patrones y métodos de especificación (estos métodos son principalmente

los autómatas finitos y expresiones regulares). Cumplir esta tarea requiere una gran cantidad de tiempo, por ende, se necesitan tomar en cuenta los detalles prácticos de la estructura del analizador léxico.

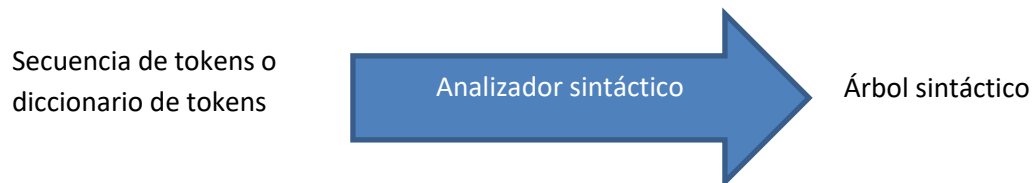
Como ya se mencionó, los tokens son entidades lógicas que por lo regular se definen como un tipo enumerado. Estos tienen diversas categorías entre las que entran las palabras reservadas y los símbolos especiales. Al ser entidades lógicas se deben distinguir de los caracteres que representan, para hacer clara la distinción, la cadena que de caracteres representada por un token se denomina en ocasiones su valor de cadena o su lexema. Algunos tokens pueden tener de un solo lexema como las palabras reservadas, pero también puede tener un número infinito de lexemas como los identificadores. Por lo tanto, un analizador léxico debe construir los valores de cadena de algunos tokens. Así que, al procesar el código con el analizador léxico, este genera los pares ordenados (Token, Lexema).

Cualquier valor asociado a un token se denomina como atributo del token. Por ejemplo, el token de símbolo “+” sirve tanto para la operación aritmética y para el valor de la cadena. Por lo tanto, el rastreador debe calcular al menor tantos atributos de un token como sean necesarios para permitir el siguiente procesamiento, para eso existen los registros o colecciones de token que sirven para recolectar todos los atributos en un solo tipo de datos estructurados.

Análisis sintáctico

El análisis gramatical es la tarea de determinar la sintaxis, o estructura, de un programa. Por esta razón también se le conoce como análisis sintáctico. Al hablar sobre esta tarea, es más útil pensar en que se va a construir un árbol de análisis sintáctico, aun cuando un compilador tal vez no lo construya en la práctica. No obstante, un analizador sintáctico debe ser capaz de construir el árbol en principio, o de lo contrario no se puede garantizar que la traducción sea correcta. La sintaxis de un lenguaje de programación por lo regular se determina mediante las reglas gramaticales de una gramática libre de contexto, de manera similar como se determina mediante expresiones regulares la estructura léxica de los tokens reconocida por el analizador léxico. En realidad, una gramática libre de contexto utiliza convenciones para nombrar y operaciones muy similares a las correspondientes en las expresiones regulares. Con la única diferencia de que las reglas de una gramática libre de contexto son recursivas. Por ejemplo. la estructura de una sentencia si debe permitir en general que otras sentencias si estén anidadas en ella, lo que no se permite en las expresiones regulares. Este cambio aparentemente elemental para el poder de la representación tiene enormes consecuencias. La clase de estructuras reconocible por las gramáticas libres de contexto se incrementa de manera importante en relación con las de las expresiones regulares. Los algoritmos empleados para reconocer estas estructuras también difieren mucho de los

algoritmos de análisis léxico, ya que deben utilizar llamadas recursivas. Las estructuras de datos utilizadas para representar la estructura sintáctica de un lenguaje ahora también deben ser recursivas en lugar de lineales (como lo son para lexemas y tokens). La estructura básica empleada es por lo regular alguna clase de árbol, que se conoce como árbol de análisis gramatical o árbol sintáctico.



Una gramática libre de contexto es una especificación para la estructura sintáctica de un lenguaje de programación. Una especificación así es muy similar a la especificación de la estructura léxica de un lenguaje utilizando expresiones regulares, excepto que una gramática libre de contexto involucra reglas de recursividad.

Las reglas gramaticales libres de contexto determinan el conjunto de cadenas sintácticamente legales de símbolos de token para las estructuras definidas por las reglas.

Herramientas utilizadas en el desarrollo del lenguaje y aplicación MPL.

JavaCC™

Java Compiler Compiler (JavaCC™) es el generador de rastreadores léxicos más popular, además que cuenta con la característica que es multiplataforma. Un generador de rastreadores es una herramienta que lee una gramática específica y la convierte en un programa de Java™ que puede reconocer parejas con entre la gramática escrita con las palabras específicas que fueron guardadas. Además del generador léxico, JavaCC™ ofrece otras capacidades relacionadas con la generación del rastreador como la construcción de árboles semánticos.

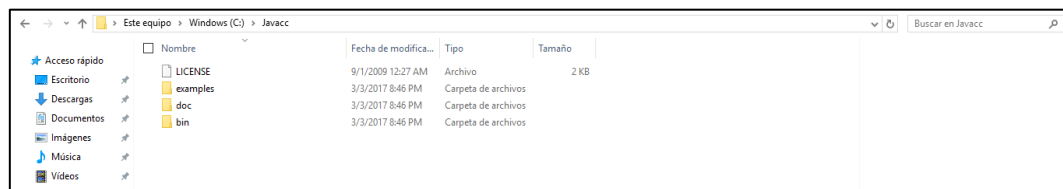
Uso de JavaCC™

1. Bajar el archivo de JavaCC en su versión de 5.0 ya que a la versión 6 le hacen falta librerías

[DESCARGAR](#)

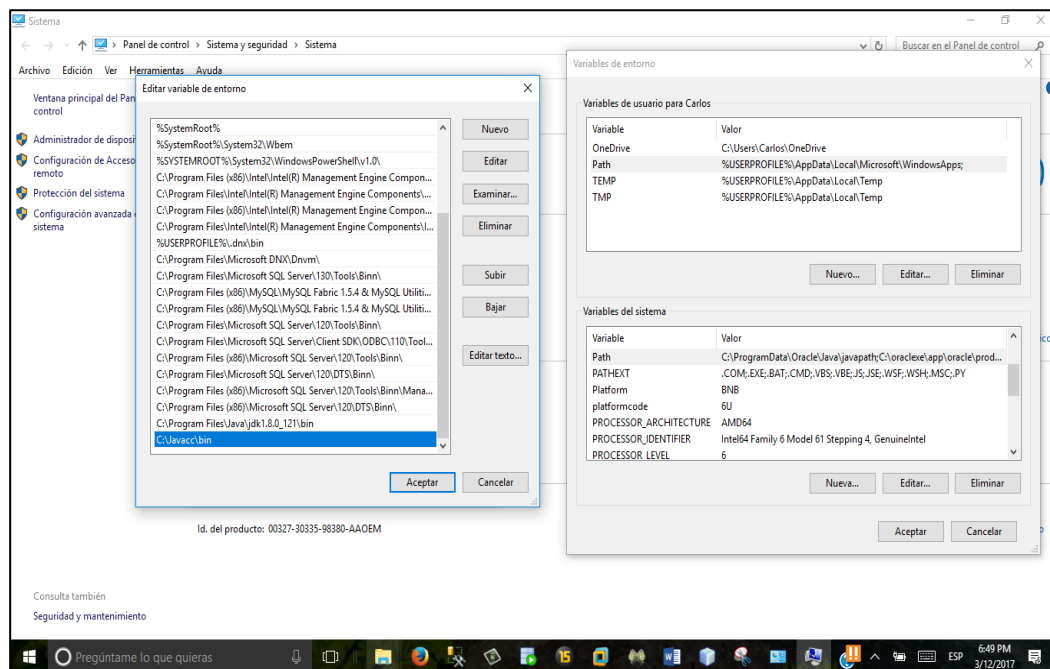
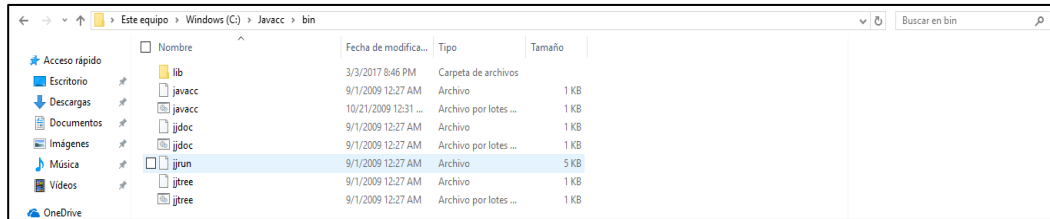


2. Extraer los archivos del archivo ZIP en el directorio llamado "C:\javacc"



3. JavaCC™ se ejecuta desde el símbolo de sistema (CMD), para ello se necesita conocer la ruta donde se encuentran los archivos.

4. Si ya se tiene instalado JDL se puede renombrar la ruta dando clic derecho en “mi PC” luego seleccionar propiedades y después ir a “avanzadas”, seguido de variables de entorno y finalmente editar las variables de uso. Se puede agregar la ruta de JavaCC™ al final de cada ruta variable o solo hacerlo temporalmente desde el símbolo de sistema escribiendo `set path=Directorio Para Buscar por ejemplo. set path=%path%; "C:\javacc\javacc-5.0\bin"`



5. Desde la consola se puede ejecutar JavaCC™ en el archivo de los tokens para generar el grupo de archivos a implementar en el analizador léxico o manejador de tokens. Hacer esto generará muchas más líneas de código de las que se esperan, sin embargo, esto mostrará los errores que se puedan presentar. Se debe escribir el siguiente comando desde el símbolo de sistema `javacc Ejemplo.ij`, esto especificando la ruta del archivo.

```
Símbolo del sistema
C:\Users\Carlos\Documents\NetBeansProjects\MPL\src\Javacc>javacc MPL.jj
Java Compiler Compiler Version 5.0 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file MPL.jj ...
Warning: Line 2, Column 3: Bad option name "Ignore_Casse". Option setting will be ignored.
File "TokenMgrError.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "SimpleCharStream.java" is being rebuilt.
Parser generated with 0 errors and 1 warnings.
C:\Users\Carlos\Documents\NetBeansProjects\MPL\src\Javacc>
```

6. Ahora para compilar los archivos que generó JavaCC™ se escribe lo siguiente: “javac *.java”

```
Símbolo del sistema
C:\Users\Carlos\Documents\NetBeansProjects\MPL\src\Javacc>javac *.java
Note: MPL.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
C:\Users\Carlos\Documents\NetBeansProjects\MPL\src\Javacc>
```

7. El analizador léxico está listo para usarse. Para ejecutarlo se escribe “java Ejemplo”
8. El rastreador solamente pedirá que se escriba una expresión y señalará los identificadores o tokens de la expresión.

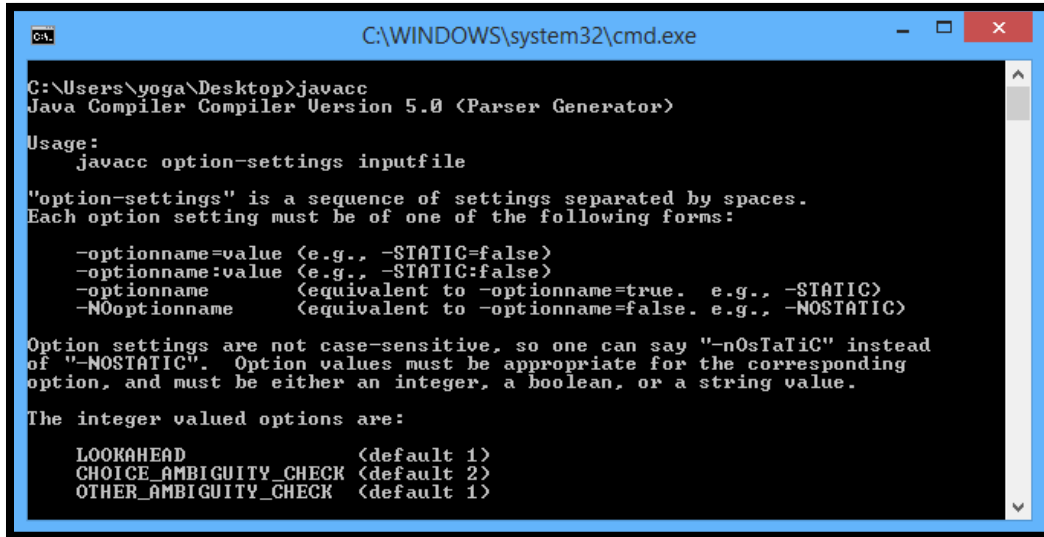
```
C:\Users\Carlos\Music\MPL>java MPL < prueba.txt
Inicializando...
Variables cadena
Cadenas de caracteres: "cadedad"
Dato entero: entero
Variables f
Constantes: 9
Operador multiplicar: *
Operador dividir: /
La entrada ha sido leída exitosamente
C:\Users\Carlos\Music\MPL>
```

CMD

El símbolo del sistema (en inglés, Command prompt, también conocido como cmd.exe o cmd) es el intérprete de comandos en OS/2 y sistemas basados en Windows NT (incluyendo Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows 7, Windows 8, Windows 8.1 y Windows 10). Es el equivalente de command.com en MS-DOS y sistemas de la familia Windows 9x.

Mediante la consola de CMD se pueden realizar tareas en el equipo sin usar la interfaz gráfica de Windows, ya que las instrucciones recibidas van directamente al núcleo del sistema. Con las órdenes que se introduzcan y se ejecuten en ella, se puede realizar prácticamente cualquier tipo de acción en Windows.

Se utilizó la consola CMD principalmente para hacer uso de las funciones de JavaCC™ a través de la ejecución de comandos.

A screenshot of a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.exe". The prompt shows the command "javacc" being executed, which displays the Java Compiler Compiler Version 5.0 (Parser Generator) usage information. The text includes the usage command "javacc option-settings inputfile", explains that "option-settings" is a sequence of settings separated by spaces, and lists various options like "-STATIC=false", "-STATIC", and "-NOSTATIC". It also mentions that option settings are not case-sensitive and lists integer-valued options: LOOKAHEAD (default 1), CHOICE_AMBIGUITY_CHECK (default 2), and OTHER_AMBIGUITY_CHECK (default 1).

```
C:\WINDOWS\system32\cmd.exe
C:\Users\yoga\Desktop>javacc
Java Compiler Compiler Version 5.0 (Parser Generator)

Usage:
  javacc option-settings inputfile

"option-settings" is a sequence of settings separated by spaces.
Each option setting must be of one of the following forms:

  -optionname=value <e.g., -STATIC=false>
  -optionname:value <e.g., -STATIC:false>
  -optionname       <equivalent to -optionname=true. e.g., -STATIC>
  -Noptionname      <equivalent to -optionname=false. e.g., -NOSTATIC>

Option settings are not case-sensitive, so one can say "-nOsTaTiC" instead
of "-NOSTATIC". Option values must be appropriate for the corresponding
option, and must be either an integer, a boolean, or a string value.

The integer valued options are:

  LOOKAHEAD           <default 1>
  CHOICE_AMBIGUITY_CHECK <default 2>
  OTHER_AMBIGUITY_CHECK <default 1>
```

La consola CMD mostrando los comandos disponibles para JavaCC™

NetBeans

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

Puede ser descargado a través del siguiente enlace: <https://netbeans.org/downloads/>

Allí se encuentran los distintos paquetes que cuentan con diferentes complementos.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos.

Se utilizó NetBeans para desarrollar el apartado gráfico del analizador léxico y el sintáctico, utilizando los archivos creados a partir de JavaCC™. Además, su entorno de desarrollo sirvió de apoyo para editar las sentencias de texto necesarias.

Definición del lenguaje MPL

El lenguaje contiene las siguientes tablas que a su vez están constituidas de: palabras reservadas, condicionales, ciclos, operadores (lógicos, aritméticos, asignación), comparación, delimitadores, tipos de datos, constantes y variables que se describen a continuación. Las cuales se deben considerar para implementar un código en este lenguaje de programación utilizando la aplicación MPL (Analizador Léxico & Sintáctico).

Palabras reservadas

Token	Palabra en el lenguaje	Descripción
mpl	Indica el inicio del código	Palabra reservada que se utiliza para indicar el inicio del código
imprimir	Muestra en pantalla	Palabra reservada para imprimir una serie de cualquier caracter en pantalla
lyg	Lee y guarda el dato	Palabra reservada para leer y guardar un dato

Condicionales

Token	Palabra en el lenguaje	Descripción
si	Condición “si”	Palabra reservada para indicar un ciclo “if”
sino	Condición “sino”	Palabra reservada para indicar un “else”

Ciclos

Token	Palabra en el lenguaje	Descripción
mientras	Ciclo “mientras”	Palabra reservada para indicar un ciclo “while”
hacer	Ciclo “hacer”	Palabra reservada para indicar un “do”
para	Ciclo “para”	Palabra reservada para indicar un ciclo “for”

Operadores lógicos

Token	Palabra en el lenguaje	Descripción
!	Operador lógico “diferente de”	Operador “diferente de”
verdad	Operador lógico “verdadero”	Operador “verdadero”
falso	Operador lógico “falso”	Operador “falso”

Operadores aritméticos

Token	Palabra en el lenguaje	Descripción
+	Operador “suma”	Operador para sumar dos números o caracteres
-	Operador “resta”	Operador para restar dos números o caracteres
*	Operador “multiplicación”	Operador para multiplicar dos números o caracteres
/	Operador “división”	Operador para dividir dos números o caracteres

Operadores de asignación

Token	Palabra en el lenguaje	Descripción
=	Operador “igual”	Operador para asignar un valor a un carácter o cadena
++	Operador “incremento”	Operador para incrementar de uno en uno
--	Operador “decremento”	Operador para decremento de uno en uno

Comparación

Token	Palabra en el lenguaje	Descripción
<	Comparar “mayor que”	Comparador “mayor que”
>	Comparar “menor que”	Comparador “menor que”
=>	Comparar “mayor o igual que”	Comparador “mayor o igual que”
<=	Comparar “menor o igual que”	Comparador “menor o igual que”
==	Comparar “igual que”	Comparador “igual que”

Delimitadores

Token	Palabra en el lenguaje	Descripción
(Paréntesis inicial	Paréntesis que abre
)	Paréntesis final	Paréntesis que cierra
{	Llave inicial	Llave que inicia
}	Llave final	Llave que cierra
;	Final de sentencia	Delimitador punto y coma
.	Valor decimal	Delimitador punto
,	Separar variables	Delimitador coma
“”	Inicio/fin de caracteres	Delimitador comillas

Tipos de datos

Token	Palabra en el lenguaje	Descripción
entero	Dato entero	Dato de tipo “int”
flotante	Dato flotante	Datos de tipo “float”
cadena	Cadena de caracteres y números	Cadenas de caracteres y números

Dígitos

Token	Palabra en el lenguaje	Descripción
0-9	Constante	Sistema numérico decimal: 0 – 9, todas sus posibles combinaciones.

Variables

Token	Palabra en el lenguaje	Descripción
“A-Z” o “a-z”, “0-9”	Variable	Desde a – z, A – Z del alfabeto y dígitos del 0-9.

Símbolos especiales

Token	Palabra en el lenguaje	Descripción
:	Símbolo especial	Símbolo especial
[Símbolo especial	Símbolo especial
#	Símbolo especial	Símbolo especial
\$	Símbolo especial	Símbolo especial
%	Símbolo especial	Símbolo especial
?	Símbolo especial	Símbolo especial
]	Símbolo especial	Símbolo especial
^	Símbolo especial	Símbolo especial
‘	Símbolo especial	Símbolo especial

Manual del usuario para la construcción de códigos basados en el lenguaje MPL

Desarrollo del lenguaje y aplicación MPL (Analizador léxico & sintáctico MPL)

Para desarrollar el **Analizador léxico & sintáctico MPL**, se utilizaron diversas herramientas (Javacc™, NetBeans 8.2, CMD, lenguaje de programación Java.) así como teorías de cómo debe estar formado y constituido un lenguaje de programación, para que a su vez se construyera un analizador capaz de entender y mostrar al usuario los resultados después de analizar de manera léxica y sintáctica, basado en un lenguaje de programación (Lenguaje MPL).

Los programadores de la aplicación primeramente se plantearon cuáles serían las estructuras de lenguaje, siempre basándose en las teorías que están respaldadas en los libros; para posteriormente crear la aplicación MPL.

La aplicación MPL (Analizador Léxico & Sintáctico MPL) fue implementado y creado en el Lenguaje de Java y JavaCC, utilizando un compilador (en este caso NeatBeans). Al ser la aplicación MPL un analizador léxico & sintáctico primero se tuvo que implementar y diseñar cuales serían los Tokens, para posteriormente crear la estructura sintáctica del lenguaje; cabe resaltar que el lenguaje MPL utiliza estructuras sencillas, ya que está enfocado a quienes apenas interactúan con un lenguaje de programación.

Tabla de tokens del lenguaje MPL

Los tokens del lenguaje MPL están basados en la sección anterior, en donde se describen la definición de lenguaje y las tablas de símbolos que se implementaron. A continuación, se ilustra cómo es que se implementaron:

Tabla de tokens	
Variables utilizadas al momento del desarrollo.	Palabras reservadas y símbolos dentro del lenguaje, también descritas en la Definición del Lenguaje.
<LENGUAJE>	mpl
<MOSTRAR>	imprimir
<LYG>	lyg
<MIENTRAS>	mientras
<HACER>	hacer
<PARA>	para
<SI>	si
<SINO>	sino
<ENTERO>	entero
<FLOTANTE>	flotante

<CADENA>	cadena
<OPRDIS>	!
<OPRTRU>	verdad
<OPRFALSE>	falso
<OPRMAS>	+
<OPRMIN>	-
<OPRMUL>	*
<OPRDIV>	/
<OPRIGUAL>	=
<OPRINC>	++
<OPRDEC>	--
<OPRMAXQ>	<
<OPRMINQ>	>
<OPRMAXIGUALQ>	=>
<OPRMINIGUALQ>	<=
<OPRIGUALQ>	==
<PARENI>	(
<PARENF>)
<LLAVEI>	{
<LLAVEF>	}
<FINSENTEN>	;
<PUNTO>	.
<COMA>	,
<COMILLAS>	“ ”
<E1>	:
<E2>	[
<E3>	#
<E4>	\$
<E5>	%
<E6>	?
<E7>]
<E8>	^
<E9>	‘

Estructuras gramaticales del Lenguaje MPL

En el apartado anterior se mostró cuáles son los tokens que constituyen el lenguaje MPL, sin embargo, puede surgir la pregunta de cuál es la estructura del lenguaje MPL, o cómo se pueden crear programas para conocer si están correctos sintáctica o léxicamente, recordando que la aplicación MPL.exe está diseñada para solo analizar de manera léxica y sintáctica; a continuación se detalla la estructura del lenguaje MPL, así como también las estructuras que se pueden implementar cuando se esté haciendo uso de la aplicación. Además, en el apartado de **manual del usuario para manejo de la interfaz gráfica**, se detalla cómo utilizar la aplicación. Es importante mencionar que se deben leer los apartados de la **Definición del lenguaje y tabla de tokens del lenguaje MPL**, para aprender y conocer a detalle este lenguaje.

Al término de la especificación de la estructura del lenguaje, se dejarán algunos ejemplos que también se encontrarán en el **Manual del usuario para manejo de la interfaz gráfica**.

Estructura general

Esta es la estructura general del programa, a partir de ahí se podrán crear diversas sentencias, por mencionar algunas: <si o sino, mientras, para, hacer> entre otras.

Programa

```
<LENGUAJE> <VARIABLE> <LLAVEI> declaracionprin() <LLAVEF>
```

Declaración principal

En esta parte se pueden declarar variables con sus correspondientes tipos de dato, en el caso del lenguaje MPL se cuenta con tres tipos: entero, flotante y cadena. Después de declarar las variables se tendrán que abrir y cerrar llaves, dentro de las cuales se podrán utilizar otras opciones tales como: volver a declarar variables, leer y guardar variables, imprimir en pantalla, ciclos, condicionales y operaciones.

declaracionprin

```
entero() | flotante() | cadena() <LLAVEI> areacodigo() <LLAVEF>
```

areacodigo

```
mostrar() | leerguardar() | ciclos() | condicional() | operacion() | declaracionsec()
```

Declaraciones de tipos de datos

Estas declaraciones están basadas en cómo se pueden declarar las variables que se requieran utilizar al momento de programar, con sus correspondientes tipos de datos; se podrá elegir la manera más adecuada según los requerimientos propios, en este caso puede ser que se declare solo la variable, o bien se le asigne un valor, aquí se muestra cuál es la estructura que se puede utilizar.

declaraciones

entero() | flotante() | cadena()

cadena

<CADENA> nombrev() <FINSENTEN> |

<CADENA> nombrev() <OPRIGUAL> <COMILLAS><VARIABLE><COMILLAS><FINSENTEN>

entero

<ENTERO>nombrev() <FINSENTEN> |

<ENTERO>nombrev()<OPRIGUAL><DIGITOS><FINSENTEN>

flotante

<FLOTANTE>nombrev()<FINSENTEN> |

<FLOTANTE> nombrev()<OPRIGUAL> <DIGITOS><PUNTO><DIGITOS><FINSENTEN>

nombrev

<VARIABLES>

Leer y guardar un dato o variable

Con las siguientes estructuras usted podrá implementar como guardar una variable, solo tiene que seguir la estructura que se indica.

leerguardar

<LYG> <PARENI> leer() <PARENF> <FINSENTEN>

leer

(<ENTERO> | <FLOTANTE> | <CADENA>) <COMA> nombrev()

Imprimir

Con las siguientes estructuras se podrá implementar un mensaje de mostrar en pantalla, siguiendo la estructura que se indica.

mostrar

<MOSTRAR> monitor() <FINSENTEN>

monitor

<PARENI> <COMILLAS> llenar() <COMILLAS> <PARENF> |

<PARENI> <COMILLAS> llenar() <COMILLAS> <OPRMAS> nombreviar() <PARENF>

llenar

<OPRMAS> | <OPRMIN> | <OPRMUL> | <OPRDIV> | <OPRDIS> | <OPRIGUAL> |

<OPRMAXQ> | <OPRMINQ> | <OPRMAXIGUALQ> | <OPRMINIGUALQ> | <PARENI> |

<PARENF> | <LLAVEI> | <LLAVEF> | <FINSENTEN> | <PUNTO> | <COMA> | <DIGITOS> |

<VARIABLE> | <E1> | <E2> | <E3> | <E4> | <E5> | <E6> | <E7> | <E8> | <E9>) *

Creación de ciclos (mientras, para, hacer) y condicionales (si, sino)

El lenguaje MPL cuenta con ciclos y condicionales, teniendo cada uno su estructura correspondiente. Para aprender a utilizarlos, se deben seguir las estructuras que se describen a continuación:

ciclos

cpara() | cmientras() | chacer()

condicional

csi() | csino()

ciclo mientras (cmientras)

<MIENTRAS> <PARENI> condicion() <PARENF> <LLAVEI> areacodigo()

<PARENI> conteo() <PARENF> <FINSENTEN> <LLAVEF>

ciclo hacer (chacer)

<HACER> <LLAVEI> areacodigo() <PARENI> conteo() <PARENF> <FINSENTEN>

<LLAVEF> <MIENTRAS> <PARENI> condicion() <PARENF> <FINSENTEN>

ciclo para (cpara)

<PARA> <PARENI> asignar() <FINSENTEN>

condicionpara()<FINSENTEN> conteo() <PARENF>

<LLAVEI> areacodigo() <LLAVEF>

condición si (csi)

<SI> <PARENI> condicion() <PARENF> <LLAVEI> areacodigo() <LLAVEF>

condición sino (csino)

<SINO> <LLAVEI> areacodigo() <LLAVEF>

asignar

(<ENTERO> | <FLOTANTE>) nombrev() <OPRIGUAL> <DIGITOS> |

<VARIABLE> <OPRIGUAL> <DIGITOS>

condicion

<VARIABLE> oprcompara() <DIGITOS> |

<VARIABLE> oprcompara() (<OPRTRUE> | <OPRFALSE>)

comparaciónpara

<VARIABLE>oprcompara()<DIGITOS>

oprcompara

<OPRMAXQ> | <OPRMINQ> | <OPRMAXIGUALQ> | <OPRMINIGUALQ> | <OPRIGUALQ> |
<OPRDIS>

conteo

<VARIABLE> <OPRINC> | <VARIABLE> <OPRDEC>

Operaciones básicas

En este lenguaje solo se pueden realizar las operaciones básicas, es decir, suma, resta, multiplicación y división, a continuación, se presenta la estructura correspondiente a cada una.

operacion

<VARIABLE> <OPRIGUAL> (<PARENI> opr () <PARENF> | opr()) <FINSENTEN>

operaciones (opr)

<VARIABLE> | <DIGITOS> operador () (<VARIABLE>|<DIGITOS>)

operador

<OPRMAS> | <OPRMIN> | <OPRMUL> | <OPRDIV>

Programa “Hola a todos” (Ejemplo1)

```
mpl Hola a todos {  
  
{  
  
  imprimir (“Hola a todos”);  
  
  imprimir (“Este es el Lenguaje MPL”);  
  
  imprimir (“Mucho éxito”);  
  
}  
  
}
```

Programa “Sumar a+b” (Ejemplo2)

```
mpl Suma {  
  
  entero a=2;  
  
  entero b=8;  
  
  entero resp;  
  
  {  
  
    resp = a+b;  
  
    imprimir(“El resultado de sumar a+b es:”+ resp);  
  
  }  
}
```



Manual del usuario para manejo de la interfaz grafica

Descripción

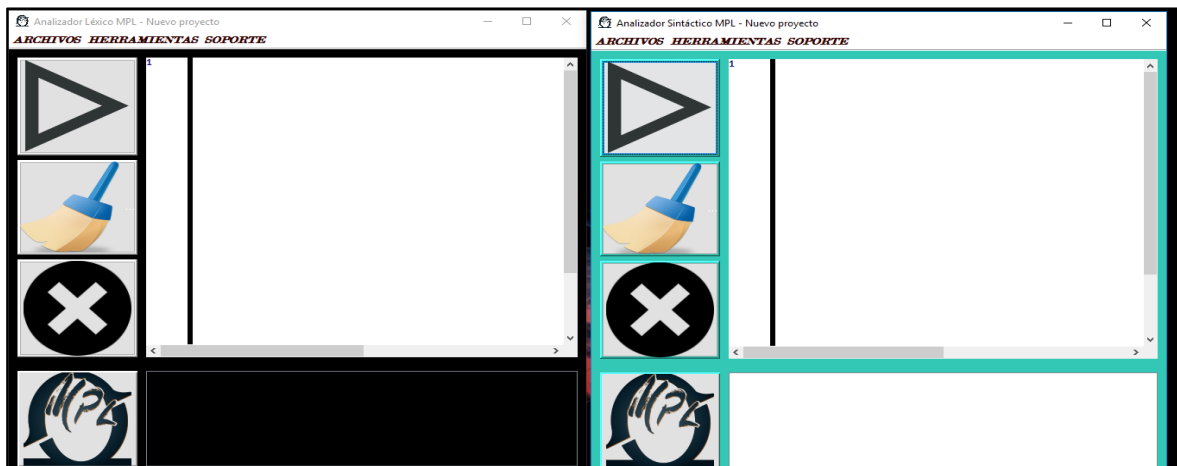
MPL es una aplicación que permite análisis léxico y sintáctico de un lenguaje, en este caso realiza el análisis bajo el lenguaje MPL (Mi Primer Lenguaje), fue desarrollado por los estudiantes Pablo García Lara, Guillermo Antonio Guzmán Vergara y José Carlos Xolio Xolio. La aplicación cuenta con la siguiente Interfaz y las distintas funciones que a continuación se detallan para que se pueda hacer uso completo de dicha aplicación de manera satisfactoria.

Para abrir dicha aplicación se debe dar clic en



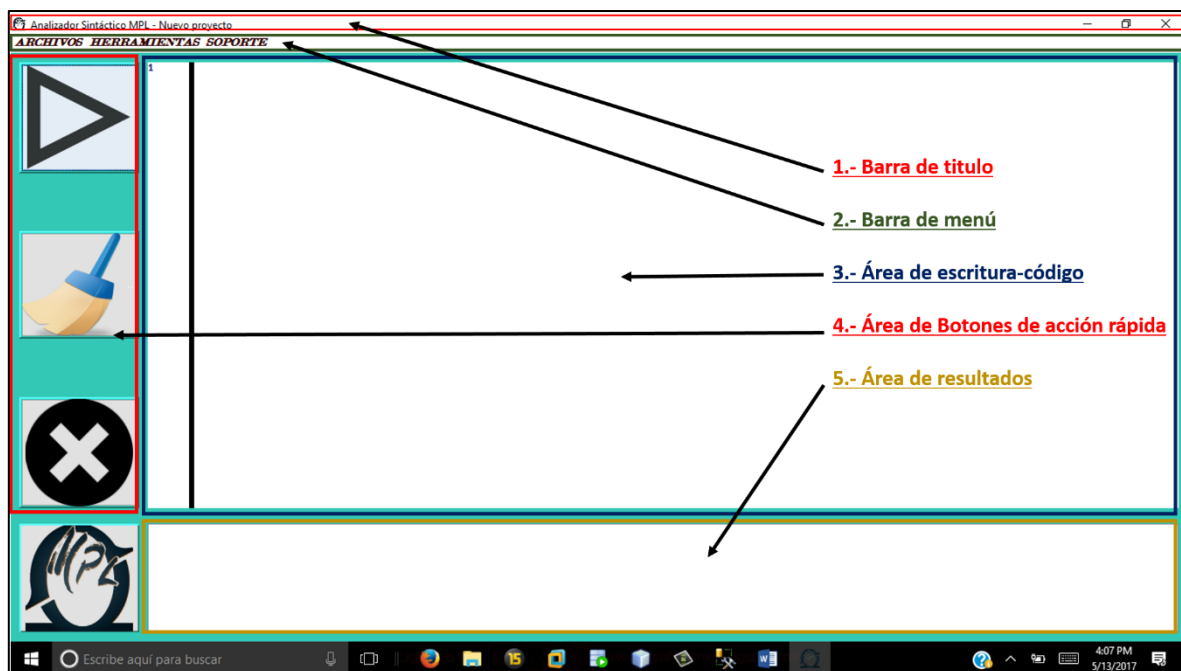
Menú Principal	
 Una ventana de carga con un fondo negro y el logo 'MPL' en verde. En la parte inferior izquierda, se muestra el texto 'Cargando MPL...' con una barra de progreso verde.	 El menú principal de la aplicación. A la izquierda, el logo 'MPL' en verde sobre fondo negro. A la derecha, tres botones: 'ANÁLISIS LÉXICO' (verde), un botón de cerrar con una 'X' blanca sobre fondo negro, y 'ANÁLISIS SINTÁCTICO' (verde).
Lo primero que aparece es una ventana de carga, indica que se están cargando todos los componentes de la aplicación.	Después de que se cargaron correctamente todos los componentes de la aplicación, se muestra el menú principal , este permite elegir que análisis desea realizar, se podrá elegir cual es el que desea ejecutar, así como también la opción de cerrar el programa.

Ventanas principales para análisis léxico y sintáctico



MPL cuenta con una interfaz general como se muestra en la parte de arriba; es decir tanto para el análisis léxico como para el sintáctico. Por ello a continuación se describirá cada una de las opciones y herramientas que se pueden utilizar al momento de realizar el análisis, además de las instrucciones que se deben seguir para lograr analizar un código de lenguaje MPL.

Componentes que integran la interfaz del usuario



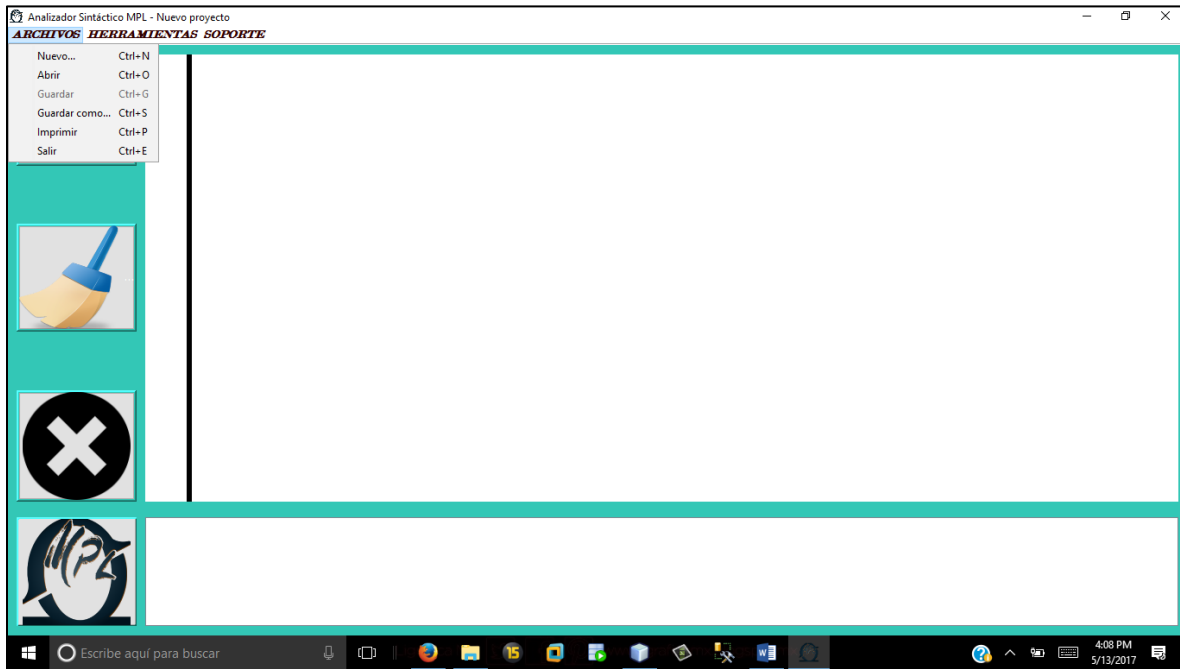
1.- Barra de título

En ella se muestra el nombre de archivo en el que se está trabajando, además de las opciones de minimizar, maximizar y cerrar.

2.- Barra de menú

La barra de menú cuenta con tres opciones que a su vez proporcionan más submenús, estas tres opciones son:

- **ARCHIVOS**



Como se puede observar esta opción proporciona las siguientes opciones:

1.- Nuevo: permite crear un nuevo espacio de trabajo, es decir, abrir otra ventana que ofrece el menú principal.

2.- Abrir: permite abrir un archivo creado, la carpeta por default es Documentos, allí se selecciona el archivo y en automático, su archivo se refleja en el área de escritura-código.

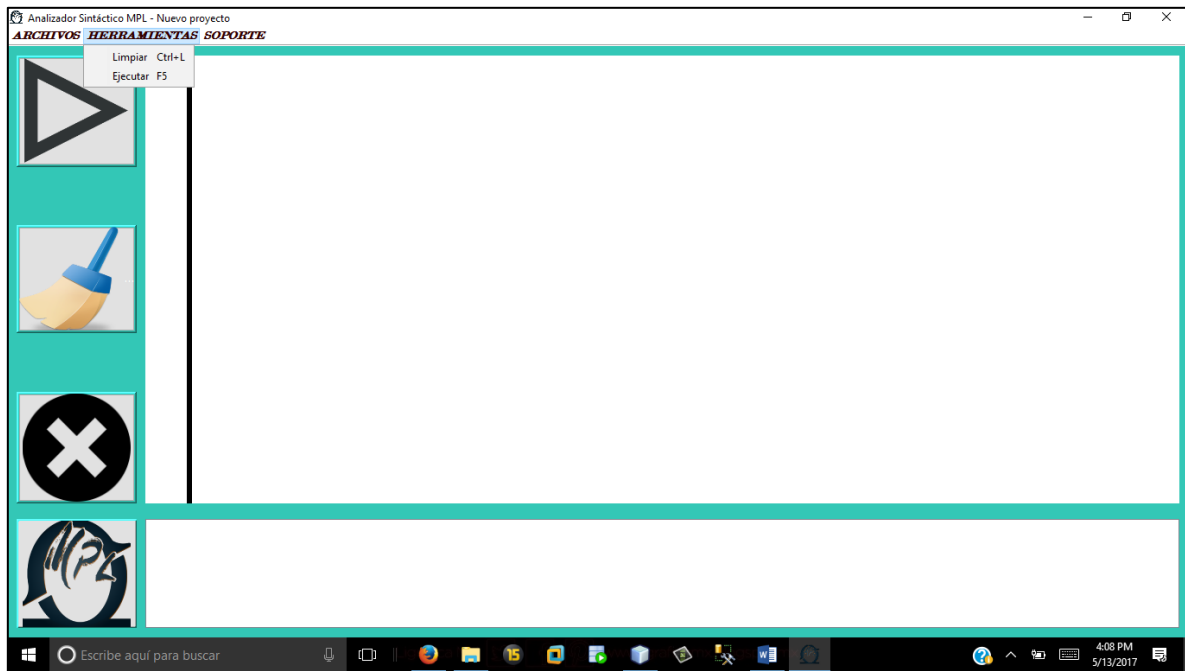
3.- Guardar: permite guardar los cambios que se realicen al archivo o si es la primera vez permite guardar su archivo en la ruta y con el nombre que se desee.

4.- Guardar como: permite guardar el archivo en la dirección que se desee.

5.- Imprimir: permite imprimir el archivo, siempre y cuando el equipo esté configurado con una impresora.

6.- Salir: permite cerrar la ventana en la que se esté trabajando, preguntando si se desea guardar cambios según lo realizado en dicha ventana.

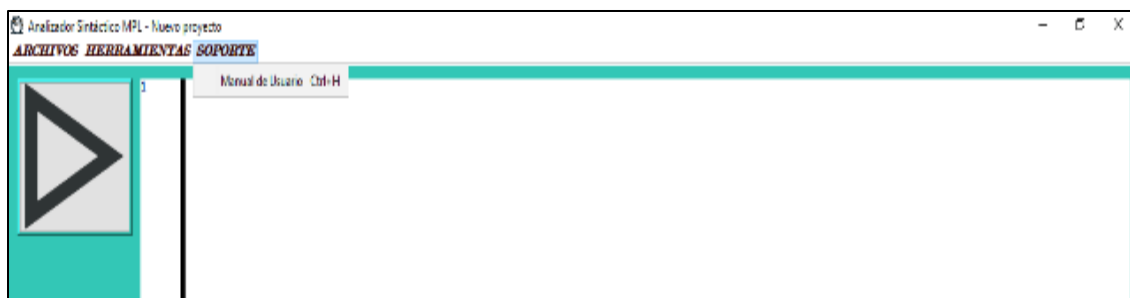
- **HERRAMIENTAS**



Como se puede observar esta opción proporciona las siguientes opciones:

- 1.- **Ejecutar:** su función principal es analizar el código que se encuentre en el área de escritura-código, y mostrar los resultados, ya sea que se guarde el archivo si es uno nuevo, o bien, solo se analice sin guardar, para ello se muestra un mensaje de diálogo.
- 2.- **Limpiar:** su función es limpiar de manera rápida toda el área escritura-código, con ello no se tendrá que seleccionar todo y después borrarse con el teclado, esta opción es viable si en realidad se desea borrar todo lo que se encuentre en el área de escritura-código, antes de realizar esa acción se pregunta si en realidad se quiere borrar.

- **SOPORTE**



Como se puede observar esta opción proporciona la siguiente opción:

1.- Manual de usuario: permite visualizar el manual de usuario en dado caso de no saber cómo realizar alguna opción dentro de la aplicación.

3.- Área de escritura-código

En el área de escritura- código se podrá crear y modificar el código realizado en el lenguaje MPL, haciendo uso de las herramientas de barra de menú y del área de botones de acción rápida.

4.- Área de botones de acción rápida

En esta área se encuentran tres opciones que usted puede hacer uso:



1.- El botón de ejecutar, su función principal es analizar el código que se encuentre en el área de escritura-código, y mostrar los resultados, ya sea que se guarde el archivo si es uno nuevo, o bien, solo se analice sin guardar, para ello se muestra un mensaje de diálogo.



2.- El Botón de limpiar, su función es limpiar de manera rápida toda el área escritura-código, con ello no se tendrá que seleccionar todo y después borrarse con el teclado, esta opción es viable si en realidad se desea borrar todo lo que se encuentre en el área de escritura-código, antes de realizar esa acción se pregunta si en realidad se quiere borrar.



3.- El botón de cerrar, su función es cerrar la ventana en la cual se esté trabajando, preguntando siempre antes de salir si se desean guardar los cambios al archivo o si desea guarda el código.

5.- Área de resultados

En esta área se muestran los resultados del análisis que se realizó, es decir, en ella se muestran en el caso del análisis léxico, el par ordenado, y en el caso del análisis sintáctico, los errores de sintaxis que haya detectado el programa.

Tareas que se pueden realizar haciendo uso de la interfaz:

En la aplicación MPL se puede realizar un análisis léxico o sintáctico, a partir de allí se cuenta con las siguientes opciones:

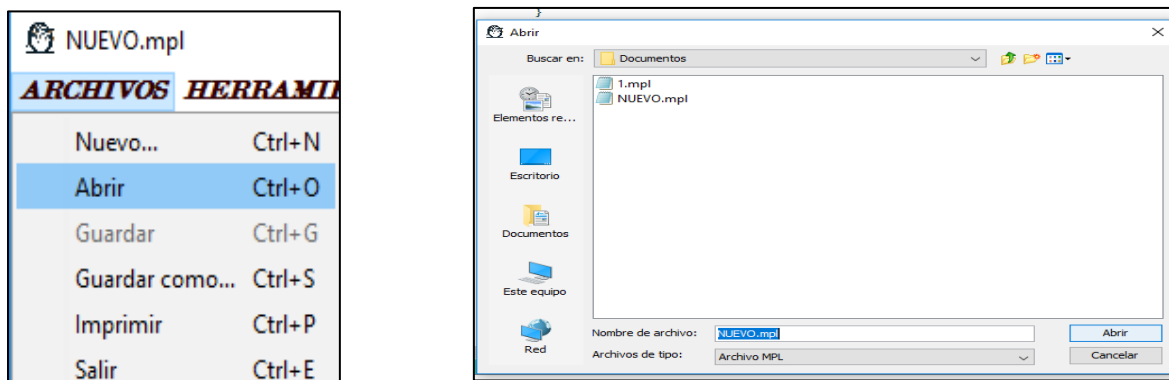
Abrir una nueva ventana

Para abrir una nueva ventana ir a la barra de menú, clic en ARCHIVOS, clic en Nuevo, saldrá el menú principal en el cual se podrá elegir cual es la opción que se desea; o bien con el teclado presionar Ctrl+N y se ejecutará la misma acción.



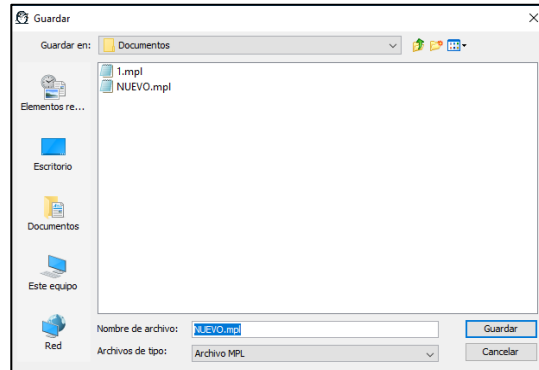
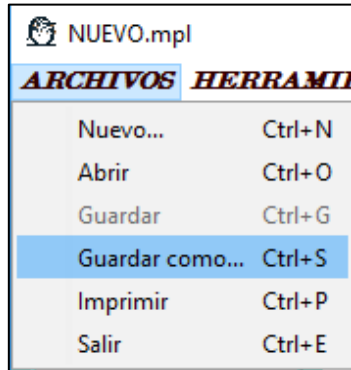
Abrir un archivo existente

Para abrir un archivo creado ir a la barra de menú, clic en ARCHIVOS, clic en Abrir, saldrá una ventana de selección de archivos, la ruta por default es Documentos; en el cual se podrá elegir cual es el archivo que se quiere abrir, dar clic en abrir y en automático el archivo se abrirá en el espacio de trabajo; o bien, con el teclado presionar Ctrl+O y después realizar los pasos anteriormente descritos.



Crear un nuevo archivo (Guardar como)

Para crear un nuevo archivo ir a la barra de menú, clic en ARCHIVOS, clic en Guardar como, saldrá una ventana de guardar archivos, la ruta por default es Documentos; en el cual se podrá guardar el archivo con el nombre que se desee, ya sea que solo se coloque el nombre o bien se introduzca la extensión .mpl. Dar clic en guardar y en automático el archivo se guardará en la ruta especificada; o bien, con el teclado presionar Ctrl+S y después realizar los pasos anteriormente descritos.




Guardar cambios de un archivo

Para guardar cambios de un archivo ir a la barra de menú, clic en ARCHIVOS, clic en Guardar; o bien, con el teclado presionar Ctrl+G. La aplicación MPL realiza un autoguardado cada cierta cantidad de caracteres escritos, en caso de que exista un cierre inesperado.

Ejecutar un análisis del código

Para ejecutar el análisis del archivo existen dos maneras; cabe mencionar que se podrá analizar sin guardar un archivo o bien guardarlo si así se lo desea:


1.- Ir a la barra de menú, clic en HERRAMIENTAS, clic en Ejecutar; o bien con el teclado presionar F5.

2.- Ir al área de botones de acción rápida y presionar el botón 

Limpiar el área de escritura-código

Para limpiar toda el área de escritura-código existen dos maneras:

1.- Ir a la barra de menú, clic en HERRAMIENTAS, clic en Limpiar; o bien con el teclado presionar Ctrl+L.

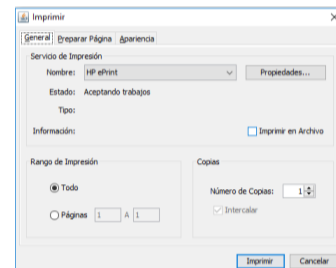
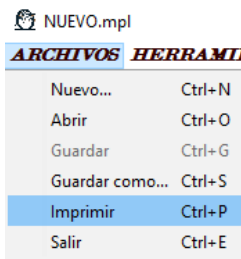
2.- Ir al área de botones de acción rápida y presionar el botón 

Abrir manual de usuario


Ir a la barra de menú, clic en SOPORTE, clic en Manual de Usuario; o bien con el teclado presionar Ctrl+M.

Imprimir

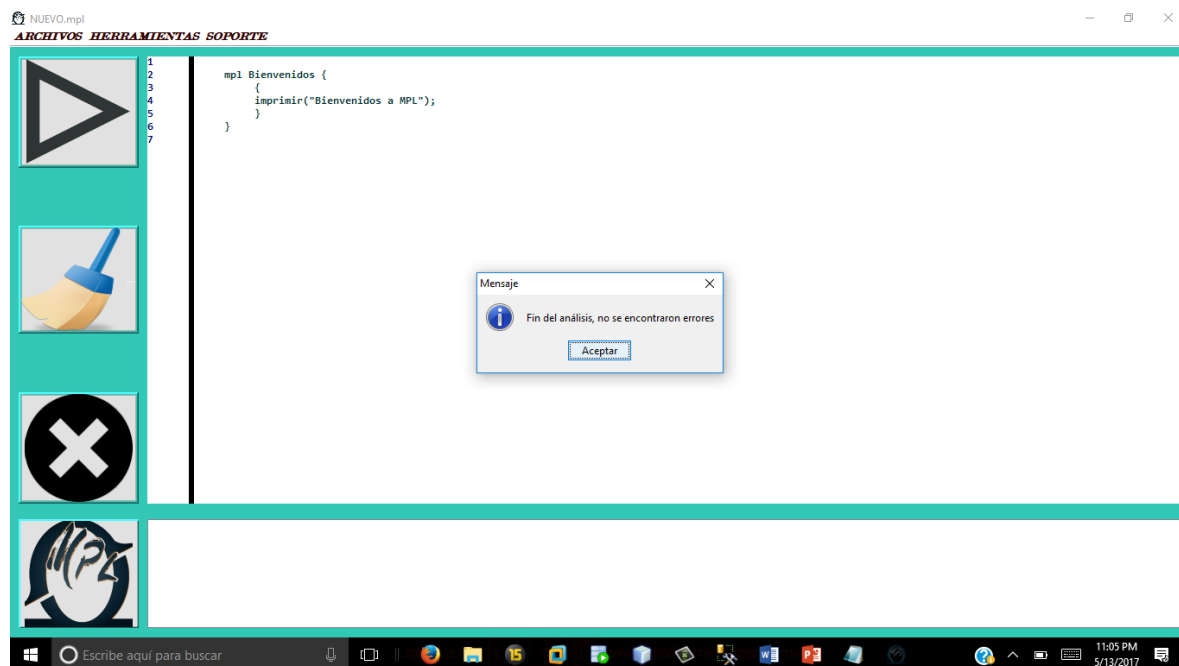
Para imprimir un archivo ir a la barra de menú, clic en ARCHIVOS, clic en Imprimir, saldrá el asistente de impresión, siempre y cuando el equipo esté conectado con una impresora se podrá imprimir el archivo; o bien con el teclado presionar Ctrl+P y realizar los mismos pasos.



Salir

Si se desea salir de una ventana existen varias maneras; ir a la barra de menú, clic en ARCHIVOS, clic en Salir, o bien con el teclado presionar, Ctrl+S, así mismo clic en el botón 

Programa “Bienvenidos” Análisis sintáctico



Conclusión

Los compiladores tienen un amplio uso dentro de la computación actualmente y han sido herramientas muy importantes para el desarrollo de este campo.

Durante este proyecto se pudo comprender más acerca de cómo comienza el proceso de desarrollo de un compilador, a través del análisis y diseño de su primera fase, sobre la cual se puede destacar que se encarga de reconocer si el código que se ingresa pertenece o no al lenguaje establecido.

Además, se describió un lenguaje de programación que pudiera distinguirse de los que ya existen.

Es importante mencionar la importancia del desarrollo de un analizador léxico, ya que el lenguaje especificado no podría ser identificado en alguna otra herramienta, sino en una que tuviera los requerimientos establecidos para hacerlo.

El análisis léxico funge como primera fase del proceso de compilación, y es notable que es un punto de partida muy importante, sin embargo, también puede presentar errores y complementarse.

El siguiente paso fue la elaboración del analizador sintáctico, donde se aplicaron los conocimientos adquiridos en clase para la creación de las gramáticas de las que hace uso el analizador sintáctico, tomando para su operación el diccionario de tokens que fue presentado en la revisión anterior. El proyecto de esta unidad consistió, en algunos casos, en prueba y error para determinar una gramática sólida para tener un lenguaje bien estructurado, sin errores al probar códigos.

Referencias

Libros:

1. Ceballos Sierra, F. (2011). *Java 2*. 1st ed. México: Alfaomega.
2. Hopcroft, J., Ullman, J., Motwani, R. and Vuelapluma, S. (2010). *Introducción a la teoría de autómatas, lenguajes y computación*. 1st ed. Madrid: Pearson Educación.

Páginas web:

1. Javacc.org. (2017). *JavaCC - The Java Parser Generator*. [online] Available at: <https://javacc.org/> [Accessed 13 Mar. 2017].
2. Netbeans.org. (2017). *Welcome to NetBeans*. [online] Available at: <https://netbeans.org/> [Accessed 13 Mar. 2017].