

TECNOLÓGICO NACIONAL DE MÉXICO

**INSTITUTO**

**TECNOLÓGICO NACIONAL DE VERACRUZ**

**Compilador “Master” en fases léxico y sintáctico**

Materia: Lenguajes y autómatas I

**Docente: Ing. Hernández Silva José**

**Integrantes:**

**-**

**-**

**Jorge Antonio Pedroza Rendon -----E17021588**

**2020**

**Introducción:**

En el siguiente documento se describirá el uso del compilador **“Master”,** cual es un compilador desarrollado en JavaCC del lenguaje básico de programación “**Cobra”**.

En este documento se engloban aspectos del manual del sistema, así como también del manual de usuario. Cabe decir que el compilador que se desarrolló, cuenta únicamente con las dos primeras fases de un compilador, la fase léxica y la fase sintáctica. Se detallan en primer lugar los aspectos generales del lenguaje **“Cobra**” como; palabras reservadas, tipos de datos, estructuras condicionales y ciclos, dicho lo anterior, para una compilación correcta en el compilador “**Master”** es necesario conocer la sintaxis del lenguaje **“Cobra”.**

Este documente se describe detalladamente el uso y las estructuras que definen la sintaxis del lenguaje como, las expresiones regulares y las gramáticas libres de contexto.

Definición del proyecto

Como parte de la materia de lenguajes y autómatas, se presenta el siguiente proyecto de la creación de un compilador en las primeras dos fases.

* **Nombre del compilador:** “Master”

Para este compilador fue necesario la creación de un lenguaje de programación básico basado en el lenguaje Python, con el fin de probar las primeras dos fases del compilador.

* **Nombre del lenguaje: “**Cobra**”**

**Descripción del lenguaje “cobra”**

Utilización del lenguaje ingles, en combinación con algunos símbolos que nos permiten crear los comandos necesarios para que el programa pueda reconocerlos y ejecutar las líneas de código para su fácil aprendizaje y rápida ejecución.

**Diferencia entre *Master* y *Cobra***

*Master*, es el nombre del compilador, el nombre del proyecto que se desarrolló para la materia, en cambio *Cobra* es el lenguaje que se implementa en el compilador; por lo tanto, el compilador *Master* implementa el lenguaje *Cobra* para poder crear su estructura.

Léxico del lenguaje

Para poder utilizar correctamente el compilador será necesario entender correctamente el uso del lenguaje, así como sus estructuras, con esto se ocuparán las gramáticas necesarias para darle sentido al lenguaje.

* **Definición del lenguaje Cobra**

Sentencias de inicio y fin.

|  |  |
| --- | --- |
| Estructura | Valor |
| <DEF><IDENTIFIER><COLON> | Inicio |
| <END> | fin |

Tipos de datos.

|  |  |
| --- | --- |
| Estructura | Valor |
| <CADENA\_STRING> | String |
| <NUMBER> | Numérico |

Concatenación.

|  |  |
| --- | --- |
| Estructura | Valor |
|  |  |
|  |  |

Estructuras condicionales

|  |  |
| --- | --- |
| Estructura | Valor |
| <IF> | Condición if |
| <ELSE> | Condición else |
| <ELSE\_IF> | Condición else if |
| <SWITCH> | Condición múltiple switch |
| <CASE> | Casos de switch |
| <DEFAULT\_CASE | Caso por defecto |
| <BREAK> | Sentencia break |

Ciclos

|  |  |
| --- | --- |
| Estructura | Valor |
| <WHILE> | Ciclo while |
| <FOR> | Ciclo for |

Entrada y salida

|  |  |
| --- | --- |
| Estructura | Valor |
| <PRINT> | Salida de datos |
| <INPUT> | Entrada de datos |

Símbolos

|  |  |
| --- | --- |
| Estructura | Valor |
| <LPAREN> | ( |
| <RPAREN> | ) |
| <LBRACKET> | [ |
| <RBRACKE> | ] |
| <COLON> | : |
| <lBRACE> | { |
| <RBRACE> | } |
| <COMMA> | , |
| <DOUBLECOMMA> | “ ” |

Operadores de comparación

|  |  |
| --- | --- |
| Estructura | Valor |
| <EQ> | == |
| <BIG> | > |
| <LESS> | < |
| <LESSEQ> | <= |
| <BIGEQ> | => |
| <NOTEQ> | != |

Operadores lógicos

|  |  |
| --- | --- |
| Estructura | Valor |
| <AND> | and |
| <OR> | or |
| <NOT> | not |

Operadores aritméticos

|  |  |
| --- | --- |
| Estructura | Valor |
| <PLUS> | + |
| <MINUS> | - |
| <TIMES> | \* |
| <DIVIDE> | / |
| <INCR> | ++ |
| <DECR> | -- |
| <ASIGNACION> | = |
| <PLUS\_EQ> | += |
| <MINUS\_EQ> | -= |
| <TIMES\_EQ> | \*= |
| <DIVIDE\_EQ> | /= |

**Identificadores**

Para crear un identificador se hace uso de la expresión regular:

<IDENTIFIER> ::= ["a"-"z","A"-"Z"](["a"-"z","A"-"Z","0"-"9","\_"])

|  |  |
| --- | --- |
| Estructura | Valor |
| <IDENTIFIER> | Ejemplo: Ped123fre |

**Números**

Para el uso de números en identificadores que manejen datos numéricos pueden ser declarados de la siguiente manera:

<NUMBER> ::= (["0"-"9"])[(["0"-"9"])"."(["0"-"9"])][(["0"-"9"])"."]["."(["0"-"9"])]

|  |  |
| --- | --- |
| Estructura | Valor |
| <NUMBER> | Ej. 100, 0.1, 4., .9 |

**Token <EOF>**

Existe un token especial que nos brinda JavaCC, este es el token de tipo <EOF> , el cuál cada vez que se lee, indica que ya no hay más valores dentro del código, por lo que lee espacios en blanco o vacíos, debido a que se consumieron todos los valores dentro del código pasado por el archivo de texto.

Sintaxis del lenguaje

* **Declaración y asignación de variables**

Una variable está compuesta por:

<Variables> ::= <IDENTIFIER><ASIGNACION>([([<IDENTIFIER>][<NUMBER>])][<DOUBLECOMMA>[<IDENTIFIER>][<NUMBER>]<DOUBLECOMMA>][[<IDENTIFIER>][<NUMBER>][<PLUS>][<MINUS>][<TIMES>][<DIVIDE>]{[<IDENTIFIER>][<NUMBER>]}]

Ej. Cadena = “ Esta es una cadena “, suma = 3 + 1, operación = a + b / c, Aux = operacion

# Estructura de un programa

La estructura básica de un programa está definida por:

<Programa> ::= <DEF><IDENTIFIER><COLON>Code( )<END><EOF>

Ejemplo:

def function :

op = "Nombre"

contador = 3 + 1

end

# Sentencias

## Sentencia if

<IF> ::= <IF>Comparacion()<COLON>Code()<END>

### Comparación

Una comparación puede ser usada en una sentencia if y está dada por:

<Comparacion> ::= {[<IDENTIFIER>][<IDENTIFIER>[<LESS>][<LESSEQ>][<BIG>][<BIGEQ>][<NOT>]<IDENTIFIER>][<IDENTIFIER>[<LESS>][<LESSEQ>][<BIG>][<BIGEQ>][<NOT>]<IDENTIFIER>[<OR>][AND][<NOT>]]}