|  |
| --- |
| **Sintaxis y semántica de los lenguajes** |



|  |
| --- |
| **Curso: K2054** |

|  |
| --- |
| **PROFESOR: Ing. Pablo Mendez** |
| **ASISTE LOS DÍAS: Jueves** |
| **EN EL TURNO: Noche** |
| **TRABAJO PRÁCTICO N°: 2** |
| **TÍTULO: Intérprete de lenguaje micro** |

|  |  |  |
| --- | --- | --- |
| **INTEGRANTES** | | |
| Dorr Pedro | 152.845-2 | dorrpei@gmail.com |
| Gioria Gonzalo Luis | 149.850-2 | gonzalolgioria@gmail.com |
| Gola Pablo Andres | 119.878-6 | pabloandresgola@gmail.com |
| Villavicencio Sergio | 157.157-6 | sergionicolasvillavicencio@gmail.com |

|  |  |  |
| --- | --- | --- |
|  | **FECHAS** | **FIRMA Y ACLARACIÓN DEL DOCENTE** |
| **Fecha de entrega N°1** | 08/11/18 |  |
| **Fecha de entrega N°2** | 22/11/18 |  |

|  |
| --- |
| **INDICACIONES PARA LAS CORRECCIONES:** |

# Objetivo

Realizar un intérprete del lenguaje MICRO.

Para ello se debe utilizar FLEX y BISON (cualquier producto que lo implemente) y lenguaje C. La entrada de las sentencias debe ser por entrada estándar o desde un archivo que el usuario pueda seleccionar. El programa debe compilar, es decir deben reducirse las expresiones. Además, se deben leer y sacar por pantalla los valores correspondientes cuando las sentencias lo requieran.

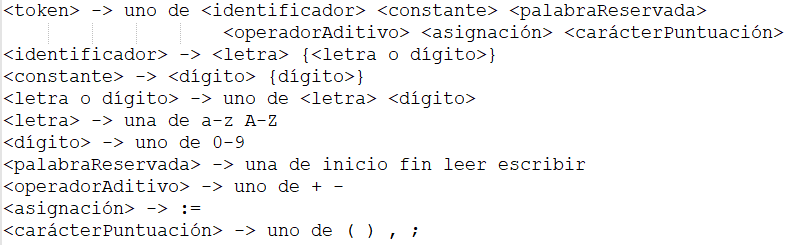
# Lenguaje Micro

Es un lenguaje teórico desarrollado por Charles Fischer. Es un lenguaje muy simple diseñado específicamente para poseer un LP concreto en el cual se pueda analizar la construcción de un compilador básico. Este lenguaje cuenta con:

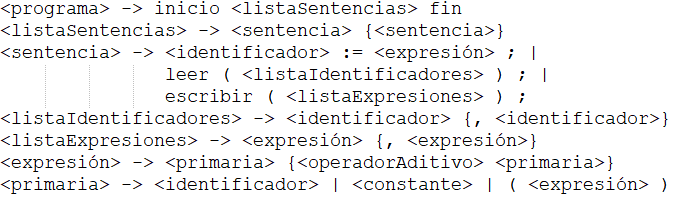
* Un único tipo de dato entero.
* Todos sus identificadores son declarados implícitamente y con una longitud máxima de 32 caracteres.
* Los identificadores deben comenzar con una letra y están compuestos de letras y dígitos.
* Las constantes son secuencias de dígitos.
* Las sentencias pueden ser:
  + Asignación:
    - **ID:=Expresión;**
    - Expresión es infija y se construye con identificadores, constantes y los operadores + y -; los paréntesis están permitidos.
  + Entrada/Salida:
    - **leer (Lista de IDs);**
    - **escribir(lista de Expresiones);**
* Cada sentencia termina con un “punto y coma” (;). El cuerpo de un programada está delimitado por inicio y fin.
* **inicio**, **fin**, **leer** y **escribir** son palabras reservadas y deben escribirse en minúscula.

# Sintaxis de micro

## Gramática léxica:



## Gramática sintáctica:



# Estructura del compilador

## Análisis léxico

El análisis léxico lo realizaremos con un módulo Sanner (Por recomendación del profesor utilizamos win\_flex).

El analizador lee, uno a uno, los caracteres que forman un programa fuente, y produce una secuencia de representaciones de tokens. El scanner es una rutina que produce y retorna la representación del correspondiente token, una por vez, en la medida que es invocada por el parser.

### Formas de implementar un scanner

* A través de la utilización de un programa auxiliar tipo lex, en el que los datos son tokens representados mediante expresiones regulares.
* Mediante la construcción de una rutina basada en el diseño de un apropiado AFD.

Para este trabajo, utilizaremos la segunda forma.

## Análisis sintáctico

El análisis sintáctico lo realizaremos con un módulo Parser(Por recomendación del profesor utilizamos win\_bison).

Este analizador procesa los tokens que le entrega el scanner hasta que reconoce una construcción sintáctica que requiere un procesamiento semántico. Entonces, invoca directamente a la rutina semántica que corresponde.

### Formas de realizar un análisis sintáctico

* Descendente (Top-Down) que permite ser construido por un programador.
* Ascendente (Bottom-Up) que requiere auxilio de un programa especializado tipo yacc.

Para este trabajo, utilizaremos la forma descendente.

## Análisis semántico

El análisis semántico consiste en la realización de tareas de verificación y de conversión que no se pueden realizar en el análisis sintáctico.

# Micro.l

En este archivo se describen un conjunto de expresiones regulares. Con esto flex busca concordancias en un fichero de entrada y ejecuta acciones asociadas a estas expresiones.

Los ficheros de entrada de flex tienen el siguiente formato:

%%

patrón1 {acción1}

patrón2 {acción2}

Donde un patrón es una expresión regular y la acción es un código en c que permita ejecutar la acción necesaria.

Flex se encarga de recorrer la entrada hasta encontrar una concordancia y ahí ejecuta el código asociado.

El fichero de entrada de Flex está compuesto de tres secciones, separadas por una línea donde aparece únicamente un ‘%%’ en esta:

definiciones

%%

reglas

%%

código de usuario

## Código fuente.l



# Micro.y

Este es el archivo fuente de bison que describe la gramática. El ejecutable que se genera indica si un fichero de entrada dado pertenece o no al lenguaje generado por esta gramática.

La forma general de una gramática de bison es la siguiente:

%{

declaraciones en C

%}

Declaraciones de Bison

%%

Reglas gramaticales

%%

Código C adicional

## Código fuente.y

