SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES 2024

Grupo N°: 4

Curso: Z2054

Profesor: Roxana Leituz

Trabajo Práctico N°: 3

Título: Flex y Byson

Integrantes:

- **Alex Morales**
- **Gabriel Salazar**

Consigna

Hacer un programa utilizando flex y bison que realice analisis léxico, sintáctico y semántico de micro. Deben personalizar los errores e implementar al menos 2 rutinas semánticas.

Lineamientos de entrega:

- Carátula con todos los integrantes
- Archivos con el código fuente (*.l y *.y)
- Ejecutable
- Documento con un detalle de como se resolvió el problema, manual de usuario y pantallas que muestren que el programa funciona
- Archivos anexos: se deben incluir los archivos que se utilicen en las pruebas
- Grupo: el mismo de todo el año
- Caso de copia: desaprueban la materia y deberán recuperar en marzo perdiendo por completo la posibilidad de promoción.
- El trabajo para darse por cumplido debe entregarse funcionando con todos los ítems desarrollados.
- Se entrega por campus una copia por grupo.

Resolución

En el lenguaje micro se deben respectar las siguientes reglas:

- El único tipo de dato es entero.
- Todos los 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 (números enteros).
- 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 programa está delimitado por inicio y fin. – inicio, fin, leer y escribir son palabras reservadas y deben escribirse en minúscula.

Para resolver el análisis del lenguaje micro, usamos código hecho con el analizador léxico flex y analizador sintáctico bison, ambos generadores de código c para simplificar el análisis del lenguaje en este caso micro.

Primero debimos escribir un código para la herramienta flex, el cual se encarga de detectar los lexemas e imprimir de qué token se trata.

```
%{
    #include <stdio.h>
    #include <string.h>
    #include "y.tab.h"
%}

%option noyywrap

DIGITO [0-9]
LETRA [a-zA-Z]
IDENTIFICADOR {LETRA}({LETRA}|{DIGITO})*
CONSTANTE_ENTERA {DIGITO}({DIGITO})*

%%

"inicio" { return INICIO; }
"fin" { return FIN; }
"escribir" { return ESCRIBIR; }
"leer" { return LEER; }
":=" { return ASIGNACION; }
{CONSTANTE_ENTERA} { yylval.intValor = atoi(yytext); return CONSTANTE; }
{IDENTIFICADOR} { yylval.charValor = strdup(yytext); return ID; }
";" { return PUNTOYCOMA; }
"(" { return PARENTIZQ; }
")" { return PARENTDER; }
"+" { return SUMA; }
"-" { return RESTA; }
"," { return COMA; }
[ \t\n]+ { /* Ignora espacios y saltos de lineas */ }
. { printf("Error lexico: Token no reconocido: '%s'\n", yytext); }

%%
```

El código de flex tiene el formato:

Definiciones
%%
Reglas
%%
Código C

En la sección de reglas, cada palabra se agrupó según su token, todas las palabras disponibles en micro para ser detectadas y además agregamos los espacios en blanco, tabulación y nueva línea para no ser detectados como error léxico. El encabezado “y.tab.h” es generado por bison que es el encargado de implementar el código generado a través de ese analizador.

El código de bison tiene el formato:

Declaraciones bison
%%
Reglas gramática
%%
Código C

Luego creamos reglas gramaticales para el analizador sintáctico separadas por el símbolo |, para la correcta detección de la sintaxis de micro. Por ejemplo:

```
ID ASIGNACION expresion PUNTOYCOMA { }
```

El programa entonces detectará la secuencia de asignación **a:=b+c**; pudiendo ser a, b y c cualquier nombre de variable.

Luego en el código:

```
ID ASIGNACION expresion PUNTOYCOMA { asignarVariable($1, $3); }
```

se agregó una acción semántica donde el símbolo $ representa la variable que puede ser cualquiera, siendo igual a valores del tercer token desde la izquierda en este caso “ID”, como establece el símbolo $1.

Luego para cada gramática, de ser necesario se implemento codigo C para manejar el analisis semántico, sintáctico y el mismo funcionamiento del programa Micro.

Pruebas creando errores:

```
Alex@DESKTOP-KEI0UU7 MINGW64 /d/Facultad/DosMilVeinticuatro/SintaxisSemanticaLenguajes/ssl-tp3-flexByson (main)
$ ./micro.exe
Ingrese programa Micro:
inicio
a := 2
fin
Error sintactico: Token inesperado 'fin'.
```

Aquí el programa detecta un error sintáctico a `:= 2` debido a que le falta un punto y coma.

```
Alex@DESKTOP-KEI0UU7 MINGW64 /d/Facultad/DosMilVeinticuatro/SintaxisSemanticaLenguajes/ssl-tp3-flexByson (main)
$ ./micro.exe
Ingrese programa Micro:
fin a := 2; inicio
Error sintactico: Token inesperado 'fin'.
```

El programa detecta un incorrecto uso de “**inicio**” y “**fin**” y esto no es permitido en micro.

Manual de usuario

micro.exe

Ejecutar el archivo, luego se imprimirá por consola el mensaje “Ingrese programa Micro:” entonces se podrá ingresar mediante teclado el código fuente de micro para ver su análisis.

```
Alex@DESKTOP-KEI0UU7 MINGW64 /d/Facultad/DosMilVeinticuatro/SintaxisSemanticaLenguajes/ssl-tp3-flexByson (main)
$ ./micro.exe
Ingrese programa Micro:
inicio
a := 2;
fin
Programa Micro Finalizado.
```

El programa informa errores léxico, semánticos y sintácticos.

Error léxico:

```
Alex@DESKTOP-KEI0UU7 MINGW64 /d/Facultad/DosMilVeinticuatro/SintaxisSemanticaLenguajes/ssl-tp3-flexByson (main)
$ ./micro.exe
Ingrese programa Micro:
inicio
a := 2 * 5;
Error lexico: Token no reconocido: '*'
```

Error semántico:

```
Alex@DESKTOP-KEI0UU7 MINGW64 /d/Facultad/DosMilVeinticuatro/SintaxisSemanticaLenguajes/ssl-tp3-flexByson (main)
$ ./micro.exe
Ingrese programa Micro:
inicio
a := b;
Error semantico: 'b' no esta declarada.
```

Error sintáctico:

```
Alex@DESKTOP-KEI0UU7 MINGW64 /d/Facultad/DosMilVeinticuatro/SintaxisSemanticaLenguajes/ssl-tp3-flexByson (main)
$ ./micro.exe
Ingrese programa Micro:
inicio
a := 5
fin
Error sintactico: Token inesperado 'fin'.
```