UTN – FACULTAD REGIONAL BUENOS AIRES

Materia: Sintáxis y Semántica de los Lenguajes – 2018

CURSO K2003

TRABAJO PRÁCTICO

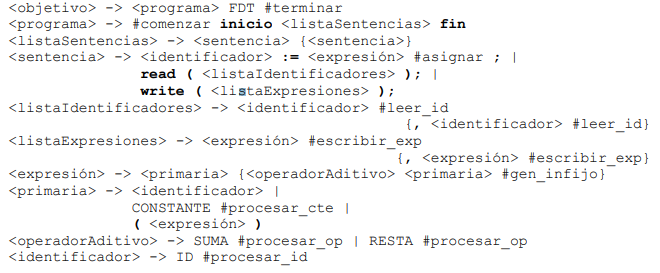
NÚMERO 12

TEMA: TABLA DE EVENTOS

|  |  |  |
| --- | --- | --- |
| **GRUPO 6** | | |
|  | | |
| **Feldman**, Maximiliano | 1632334 | maxifeldman@hotmail.com |
|  |  |  |
| **Barchiesi,** Franco | 1632619 | barchiesiefranco@gmail.com |
|  |  |  |
| **Mendez,** Martin  **Baldraco**, Carlos Guillermo | 1590947  1600837 | mendezmartin97@gmail.com  cgbaldraco@gmail.com |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FECHA DE PRESENTACIÓN: | 10/10/18 |  | FIRMA PROFESOR |  |
|  |  |  |  |  |
| FECHA DE DEVOLUCIÓN: |  |  | CALIFICACIÓN |  |

**GRAMÁTICA SINTÁCTICA UTILIZADA PARA EL PARSER**



**TABLA DE TRANSICIONES PARA EL AUTÓMATA FINITO QUE UTILIZARÁ EL SCANNER**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | L | D | + | - | ( | ) | , | ; | : | = | EOF | espacio | OTRO |
| 0- | 1 | 3 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 14 | 13 | 0 | 14 |
| 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2+ (ID) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 3 | 4 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 4+  (CTE) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 5+ (+) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 6+  (-) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 7+  (“(“) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 8+  (“)”) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 9+  (,) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 10+  (;) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 11+  (:) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 12+  (:=) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 13+  (FDT) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 14+  (ERROR) | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |

**ANALISIS SINTACTICO DESCENDIENTE DEL ARCHIVO prueba.m**

|  |  |  |  |
| --- | --- | --- | --- |
| Paso | Acción del Parser | Programa sin Procesar | Instruc. Generada para MV |
| 1 | Llamar a Objetivo() | inicio  leer (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 2 | Llamar a Programa() | inicio  leer (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 3 | Acción Semántica Comenzar() | inicio  leer (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 4 | Match(INICIO) | inicio  leer (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 5 | Llamar a ListaSentencias() | leer (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 6 | Llamar Sentencia() | leer (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 7 | Match(LEER) | leer (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 8 | Match(PARENIZQUIERDO) | (a,b);  c := a+b;  escribir (a, c);  fin |  |
| 9 | Llamar a ListaIdentificadores() | a,b);  c := a+b;  escribir (a, c);  fin |  |
| 10 | Llamar a Identificador() | a,b);  c := a+b;  escribir (a, c);  fin |  |
| 11 | Match(ID) | a,b);  c := a+b;  escribir (a, c);  fin |  |
| 12 | Acción semántica ProcesarId() | ,b);  c := a+b;  escribir (a, c);  fin | Declara a, Entera |
| 13 | Acción semántica leer() | ,b);  c := a+b;  escribir (a, c);  fin | Read a, Entera |
| 14 | Match(COMA) | ,b);  c := a+b;  escribir (a, c);  fin |  |
| 15 | Llamar a Identificador() | b);  c := a+b;  escribir (a, c);  fin |  |
| 16 | Match(ID) | b);  c := a+b;  escribir (a, c);  fin |  |
| 17 | Acción semántica ProcesarId() | );  c := a+b;  escribir (a, c);  fin | Declara b, Entera |
| 18 | Acción semántica leer() | );  c := a+b;  escribir (a, c);  fin | Read b, Entera |
| 19 | Match(PARENDERECHO) | );  c := a+b;  escribir (a, c);  fin |  |
| 20 | Match(PUNTOYCOMA) | ;  c := a+b;  escribir (a, c);  fin |  |
| 21 | Llamar a Sentencia() | c := a+b;  escribir (a, c);  fin |  |
| 22 | Llamar a Identificador() | c := a+b;  escribir (a, c);  fin |  |
| 23 | Match(ID) | c := a+b;  escribir (a, c);  fin |  |
| 24 | Acción semántica ProcesarId() | := a+b;  escribir (a, c);  fin | Declara c, entera |
| 25 | Match(ASIGNACION) | := a+b;  escribir (a, c);  fin |  |
| 26 | Llamar a Expresion() | a+b;  escribir (a, c);  fin |  |
| 27 | Llamar a Primaria() | a+b;  escribir (a, c);  fin |  |
| 28 | Llamar a Identificador() | a+b;  escribir (a, c);  fin |  |
| 29 | Match(ID) | a+b;  escribir (a, c);  fin |  |
| 30 | Acción semántica ProcesarId() | +b;  escribir (a, c);  fin |  |
| 31 | Llamar a OperadorAditivo() | +b;  escribir (a, c);  fin |  |
| 32 | Match(OPERADOR) | +b;  escribir (a, c);  fin |  |
| 33 | Acción semántica procesarOp() | b;  escribir (a, c);  fin |  |
| 34 | Llamar a Primaria() | b;  escribir (a, c);  fin |  |
| 35 | Llamar a Identificador() | b;  escribir (a, c);  fin |  |
| 36 | Match(ID) | b;  escribir (a, c);  fin |  |
| 37 | Acción semántica ProcesarId() | ;  escribir (a, c);  fin |  |
| 38 | Acción semántica GenInfijo() | ;  escribir (a, c);  fin | Declara Temp&1,Entera,  Sumar a,b,Temp&1 |
| 39 | Acción semántica Asignar() | ;  escribir (a, c);  fin | Almacena Temp&1,c |
| 40 | Match(PUNTOYCOMA) | ;  escribir (a, c);  fin |  |
| 41 | Llamar a Sentencia() | escribir (a, c);  fin |  |
| 42 | Match(ESCRIBIR) | escribir (a, c);  fin |  |
| 43 | Match(PARENIZQUIERDO) | (a, c);  fin |  |
| 44 | Llamar a ListaExpresiones() | a, c);  fin |  |
| 45 | Llamar a Expresion() | a, c);  fin |  |
| 46 | Llamar a Primaria() | a, c);  fin |  |
| 47 | Llamar a Identificador() | a, c);  fin |  |
| 48 | Match(ID) | a, c);  fin |  |
| 49 | Acción Semántica ProcesarId() | , c);  fin |  |
| 50 | Acción semántica Escribir() | , c);  fin | Write a, Entera |
| 51 | Match(COMA) | , c);  fin |  |
| 52 | Llamar a Expresion() | c);  fin |  |
| 53 | Llamar a Primaria() | c);  fin |  |
| 54 | Llamar a Identificador() | c);  fin |  |
| 55 | Match(ID) | c);  fin |  |
| 56 | Acción Semántica ProcesarId() | );  fin |  |
| 57 | Acción semántica Escribir() | );  fin | Write c, Entera |
| 58 | Match(PARENDERECHO) | );  fin |  |
| 59 | Match(PUNTOYCOMA) | ;  fin |  |
| 60 | Match(FIN) | Fin |  |
| 61 | Match(FDT) |  |  |
| 62 | Acción semántica Terminar() |  | Detiene |

**LEX Y YACC**

Lex genera analizadores léxicos a partir de una especificación de los componentes léxicos

en términos de expresiones regulares. Lex toma como un ingreso un archivo con extensión

.l y produce un fichero en c llamado lex.yy.c que contiene el analizador lexico.

El fichero de entrada tiene 3 partes

- Definiciones

- Expresiones regulares-acciones

- Codigo

El fichero lexx.yy.c contiene una función yylex que realiza la función del analizador lexico,

que devuelve un int con el numero que representa el token.

Yacc genera un fichero en C (y.tab.c) que contiene el analizador sintactico ascendente por

desplazamiento y reducción, y una función que ejecuta las acciones semanticas que

aparecen en el fichero de especificación.

El fichero tiene 3 partes

- Definiciones de tokens

- Reglas-Acciones

- Codigo

El fichero generado contiene una función yyparse que llama a yylex cada vez que necesita

un token.

Lex debe devolver los números de tokens que asigna Yacc. Se ejecuta Yacc con la opción -d

que hace que yacc genere un fichero y.tab.h que contenga solo los defines de los tokens.

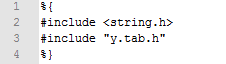
Ese fichero debe ser incluido con #Include en el fichero de lex, al principio, para que los

tokens sean conocidos dentro del código generado por lex.

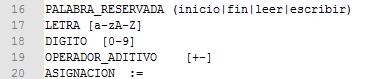
**Aplicación de Lex y Yacc**

Lex:

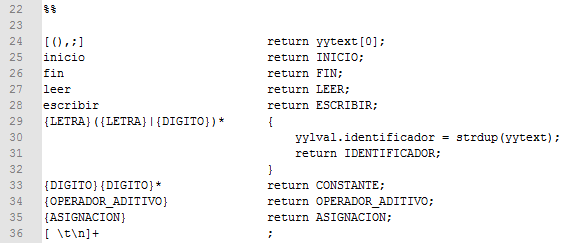
En la primer parte de lex se incluyen los header correspondientes para poder usar ciertas funciones y reconocer los TOKEN, además del fichero que recolecta la información necesaria para luego ser utilizada en YACC. Para esto, se encierra entre “%{“y “%}” el código de C o C++ que se quiera agregar.



También en la primer parte se especifican las declaraciones léxicas del lenguaje utilizado, en este caso, Micro.

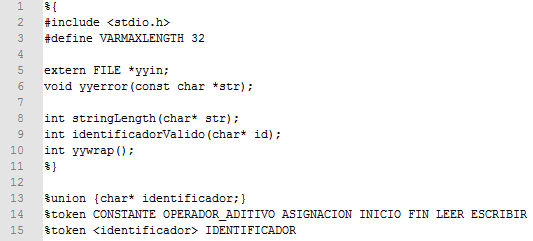


En la segunda parte, separada por “%%”, se encuentras las expresiones regulares las cuales representan los componentes léxicos que del lenguaje que se quieran reconocer. A cada una se le asigna una acción, escrita en código C o C++. Para aquellos en los que no se presenta un return, el analizador debe aceptarlos, pero ignorarlos (tales como el enter y espacio en blanco).En el caso específico de los identificadores, almacenamos el nombre del identificador reconocido para luego poder acceder al mismo desde YACC. Por otro lado, si no se utiliza strdup() el puntero de yytext será copiado en yylval.identificador. Cabe aclarar el posicionamiento de las palabras reservadas sobre los identificadores debido a que las primeras pueden ser consideradas identificadores, pero lex las considera como la primera expresión por la que lo reconozca

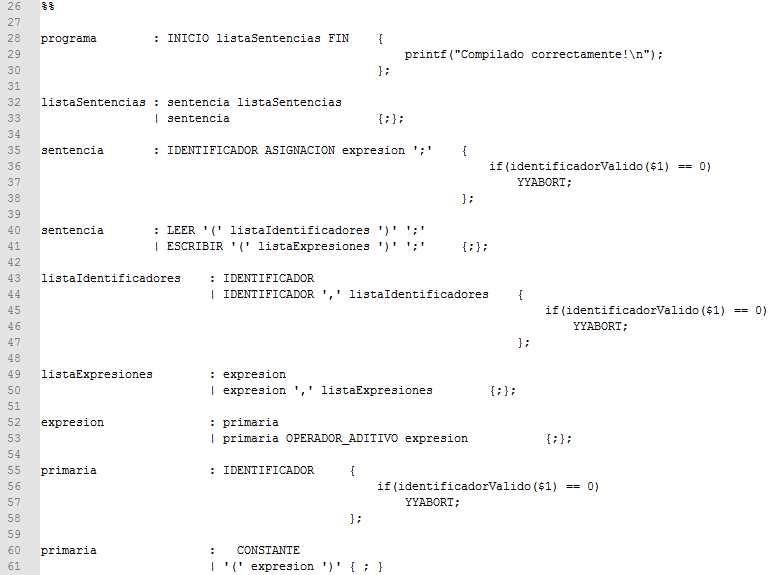


Yacc

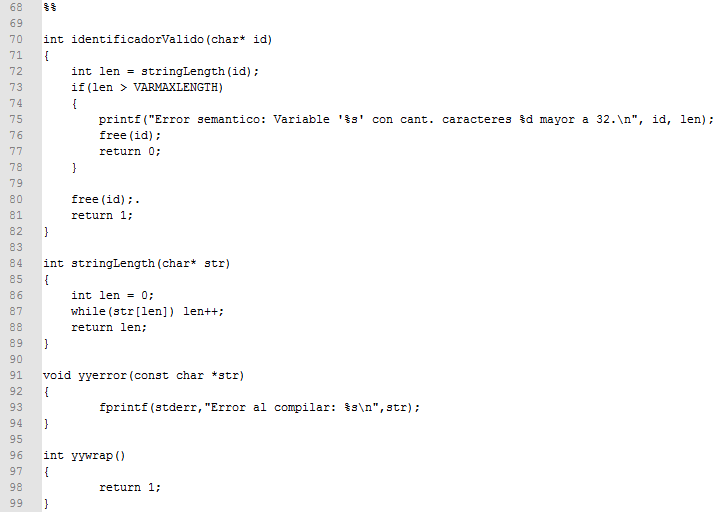
En la primera parte de Yacc se definen los token. Además se coloca código en C, como por ejemplo los elementos necesarios para relacionarse con lex, y los prototipos de funciones auxiliares que serán expandidas más adelante. El elemento yyin debe declararse como extern pues el mismo está declarado inicialmente en el programa YACC y de lo contrario se obtendría un error porque se estaría redefiniendo el elemento.



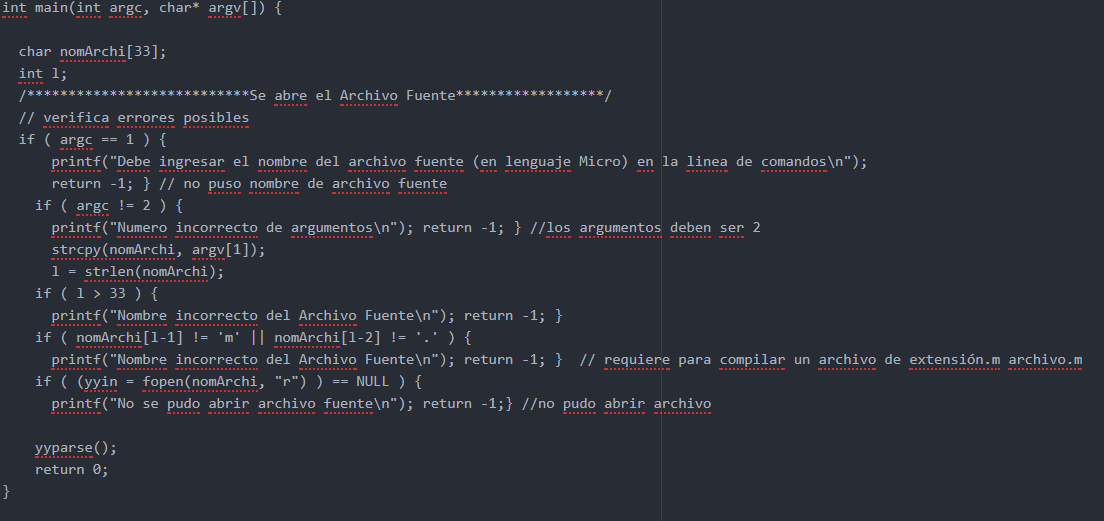
En la segunda parte se especifican las reglas gramaticales y acciones semánticas. Se utiliza el macro YYABORT para detener la ejecución en caso de identificadores no validos



En la tercer parte se coloca el código C de las funciones que fueron utilizadas en la segunda parte. La función yywrap() identifica el EOF, el cual no es posible de identificar mediante reglas gramaticales, por lo cual la expresión “<objetivo>: programa EOF” queda implícitamente descripta por la función yywrap, que confirma la compilación del programa



Por último, el main hace los controles necesarios para la ejecucion

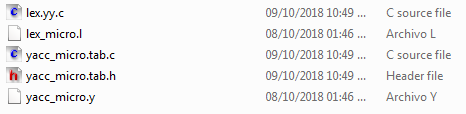


Ejemplos de implementación

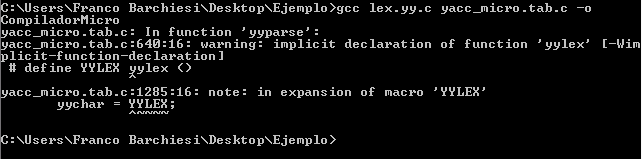
En primer lugar, se deberá contar con los archivos lex (.l) y Yacc(.y) a partir de los cuales se analizará. A partir de ahí se ejecutan con flex y bison, creándose los archivos de C correspondientes.

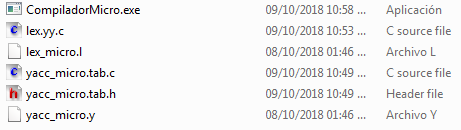






Una vez creados los archivos en c, se creará el Compilador.exe

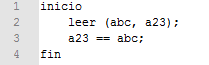




Con esto ya estaría listo el compilador para ser puesto en marcha

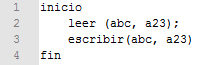
Casos de prueba

Comenzaremos probando el caso de un error léxico. En el ejemplo, la línea 3 es errónea y debería ser considerado un error léxico





En este caso de error sintáctico, en la línea 3 para ejecutar correctamente la función escribir, haber un “;” luego del cierre de paréntesis.





Caso satisfactorio:

