



# Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

## Ingeniería en sistemas de información

*Sintaxis y Semántica de los Lenguajes [K2055]*

### TP N°3 – Compilador: Flex y Bison

#### Alumnos:

- Luciano Schirripa
- Santiago García
- Tomás Zignego

Docente: Roxana Leituz

.....

Fecha: 19/11/2021 – 26/11/2021

.....

## SCANNER:

Para realizar el análisis léxico correspondiente al lenguaje micro, lo primero a realizar fue la revisión de los tokens y palabras reservadas que este lenguaje posee. Una vez estudiado, se procedió a ubicar primeramente las declaraciones en el archivo ".l", las cuales para facilitar las posteriores ER, declaramos la composición del DIGITO y la LETRA.

```
1  %{
2      #include <string.h>
3      #include<stdio.h>
4      #include<stdlib.h>
5      #include "sin.tab.h"
6      extern void yyerror(char*);
7  %}
8
9      DIGITO [0-9]
10     LETRA [a-zA-Z]
11
12     %%
```

Posterior, en la sección de reglas, se han definido las palabras reservadas de micro, y los tokens a retornar, junto a su correspondiente acción.

```
%%
[ \t\n]+ ;
"inicio" {printf("se ha detectado una palabra reservada \n");return INICIO;};
"fin" {printf("se ha detectado una palabra reservada \n");return FIN;};
"escribir" {printf("se ha detectado una palabra reservada \n");return ESCRIBIR;};
"leer" {printf("se ha detectado una palabra reservada \n");return LEER;};

{LETRA}{LETRA}{DIGITO}* {{if(yylen>32) {yyerror("El identificador posee mas de 32 caracteres");} else {printf("identificado
{DIGITO}+
"!=" return CONSTANTE;
"+" return SUMA;
"- " return RESTA;
"(" return PARENTESIS_A;
")" return PARENTESIS_C;
";" return PUNTOYCOMA;
"," return COMA;
"/*".* ;
. {printf("Error lexico: caracter invalido.");};
You, a week ago • add files
%%
```

Cabe destacar, que primero se realizó solamente la devolución de los tokens para el parser, y después al comprobar su correcto funcionamiento, se han agregado mas acciones como la validación de caracteres.

## PARSER:

Para realizar el parser, en el archivo “.y”, primeramente en la sección de declaraciones, se declararon algunas variables externas próximas a utilizar, y bibliotecas necesarias para el funcionamiento del código.

Además, se definieron los tokens retornados por el scanner según el parser lo necesite, la unión, y por que No Terminal el parser debería comenzar.

```
You, seconds ago | 1 author (You)
1  %{
2      #include <string.h>
3      #include<stdio.h>
4      #include<stdlib.h>
5      extern int yylex(void);
6      extern void yyerror(char*);
7      extern int yyleng;
8  %}
9
10
11  %union{
12      char* cadena;
13      int num;
14  }
15
16  %token INICIO FIN SUMA RESTA ASIGNACION PARENTESIS_A PARENTESIS_C PUNTOYCOMA LEER ESCRIBIR COMA
17  %token <cadena> IDENTIFICADOR
18  %token <num> CONSTANTE
19
20  %start startProgram
21
22  %%      You, a week ago • fix flex.l
```

Con respecto a la gramática sintáctica utilizada, se declaró en la sección de reglas, utilizando los tokens retornados por el scanner. Como se puede visualizar, comienza evaluando el inicio de la sentencia solo con la palabra “inicio” y “fin” al terminar. Esta gramática especifica el uso de sentencias, expresiones, y además, retorna el aviso de una compilación exitosa, en el caso de respetar la gramática.

```
%%      You, a week ago • fix flex.l
startProgram:      INICIO sentencias FIN {printf("compilacion exitosa");};
sentencias:        sentencias sentencia | sentencia;
sentencia:         LEER PARENTESIS_A listaDeIds PARENTESIS_C PUNTOYCOMA | ESCRIBIR PARENTESIS_A listaDeExpresiones PARENTESIS_C PUNTOYCOMA | IDENTIFI
listaDeExpresiones: expresion COMA listaDeExpresiones | expresion;
listaDeIds:        IDENTIFICADOR COMA listaDeIds | IDENTIFICADOR;
expresion:         primaria | expresion operador primaria;
primaria:          IDENTIFICADOR | CONSTANTE | PARENTESIS_A expresion PARENTESIS_C;
operador:          SUMA | RESTA;
%%
```

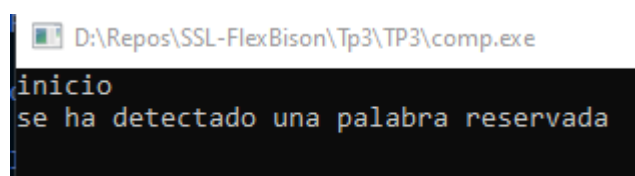
Por último, en la sección código de usuario, se declaro para el correcto funcionamiento la función main, el yywrap, y el yyerror, el cual establece como un prefijo “error en la compilación: (string retornado del yyerror)“.

```
39
40 %%
41 int main() {
42     yyparse();
43 }
44
45 int yywrap() {
46     return 1;
47 }
48
49 void yyerror (char *s){
50     printf ("Error en la compilacion: %s\n",s);
51 }
```

## **RESULTADO:**

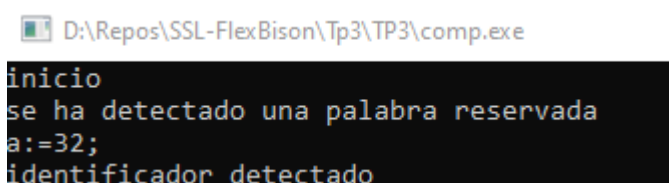
Como resultado de compilar ambos archivos, el ejecutable es el siguiente:

- 1) Obligatoriamente se debe registrar la palabra reservada inicio, y el compilador da aviso de esta detección.



```
D:\Repos\SSL-FlexBison\Tp3\TP3\comp.exe
inicio
se ha detectado una palabra reservada
```

- 2) Al declarar también una constante mediante el identificador “a”, el compilador también lo notifica.



```
D:\Repos\SSL-FlexBison\Tp3\TP3\comp.exe
inicio
se ha detectado una palabra reservada
a:=32;
identificador detectado
```

- 3) A su vez, si se intenta declarar un identificador con mas de 32 caracteres, el compilador arrojará un yyerror:

```
D:\Repos\SSL-FlexBison\Tp3\TP3\comp.exe
inicio
se ha detectado una palabra reservada
a:=32;
identificador detectado
esteIdentificadorPoseeMuchosCaracteresPorLoTantoEsInvalido:=1;
Error en la compilacion: El identificador posee mas de 32 caracteres
```

- 4) También, si en cualquier momento se utiliza un lexema no reconocido por el scanner, el compilador dará aviso de este error:

```
%
Error lexico: caracter invalido.^
Error lexico: caracter invalido.
&
Error lexico: caracter invalido.
*
Error lexico: caracter invalido.
!
Error lexico: caracter invalido._
```

- 5) Por último, un ejemplo de una compilación correcta, devuelve un aviso de “compilación exitosa”.

```
D:\Repos\SSL-FlexBison\Tp3\TP3\comp.exe
inicio
se ha detectado una palabra reservada
a:=32;
identificador detectado
b:=33;
identificador detectado
leer(a,b);
se ha detectado una palabra reservada
identificador detectado
identificador detectado
fin
se ha detectado una palabra reservada
compilacion exitosa_
```

CODIGO FUENTE: <https://github.com/utn-frba-ssl/21-055-05>