Universidad Carlos III
Grado en ingeniería informática
Curso Procesadores del Lenguaje 2024-25
Laboratorio 4
Curso 2024-25

Fecha: **12/02/2025** - ENTREGA: **4**

GRUPO: **403**

Alumnos: **Mario Ramos Salsón (100495849)**

Miguel Yubero Espinosa (100495984)

CUESTIONES ABIERTAS:

4.1 Prueba las siguientes expresiones para comprobar el funcionamiento de la precedencia. ¿a qué conclusión llegas? 2*3+1 2+3*1 1+3*2 1*3+2.

2*3+1
Expresion=8.000000
2+3*1
Expresion=5.000000
1+3*2
Expresion=7.000000
1*3+2
Expresion=5.000000

En base a los resultados obtenidos concluimos que sucede como en prácticas anteriores, se hace primero la operación de la derecha del todo por eso mismo hay resultados que no coinciden con la realidad.

4.2 Reforma la gramática sustituyendo el no terminal operando por el de expresion en aquellas producciones que contengan una operación binaria. Recuerda que habíamos introducido operando para evitar la ambigüedad debida a la doble recursivdad (expresionexpresion+expresion). Prueba de nuevo las expresiones anteriores.

La gramática reformada es la siguiente:

```
expresion:
        expresion '+' expresion { $$ = $1 + $3; }
        | expresion '-' expresion { $$ = $1 - $3; }
        | expresion '*' expresion { $$ = $1 * $3; }
        | expresion '/' expresion { $$ = $1 / $3; }
        | NUMERO { $$ = $1; }
        | '+' expresion %prec SIGNO_UNARIO { $$ = $2; }
        | '-' expresion %prec SIGNO_UNARIO { $$ = -$2; }
        | '(' expresion ')' { $$ = $2; } ;
```

Al volver a realizar las operaciones del apartado anterior, obtenemos los siguientes resultados:

2*3+1
Expresion=7.000000
2+3*1

Expresion=5.000000
1+3*2
Expresion=7.000000
1*3+2
Expresion=5.000000

4.3 Si nos plantearan usar variables como operandos dentro del lenguaje, ¿qué cambios plantearías en la gramática? Partimos de la idea de que las variables estarán identificadas por una secuencia alfanumérica, siendo el primer carácter una letra. No se contempla la definición previa de variables.

En primer lugar agregaría un nuevo token para variables. En la sección de bison introduciremos:

```
%token IDENTIFICADOR
```

Modificaremos la gramática para aceptar variables como operandos. Sería igual que en el apartado anterior, pero añadiendo IDENTIFICADOR.

```
expresion:
    expresion '+' expresion { $$ = $1 + $3; }
    | expresion '-' expresion { $$ = $1 - $3; }
    | expresion '*' expresion { $$ = $1 * $3; }
    | expresion '/' expresion { $$ = $1 / $3; }
    | NUMERO { $$ = $1; }
    | IDENTIFICADOR /* Representa una variable */
    | '+' expresion %prec SIGNO_UNARIO { $$ = $2; }
    | '-' expresion %prec SIGNO_UNARIO { $$ = -$2; }
    | '(' expresion ')' { $$ = $2; } ;
```

Por último modificaremos el analizador léxico para reconocer identificadores.En flex, añadimos una regla para reconocer identificadores (una letra seguida de letras o números):

```
[a-zA-Z][a-zA-Z0-9]* { yylval = strdup(yytext); return IDENTIFICADOR; }
```

4.4 Incluye la operación de asignación de una expresión a una variable en la gramática.

Para incluir la operación de asignación de una expresión a una variable en la gramatica, añadimos el siguiente código:

En primer lugar añadiremos una nueva producción para la asignación en bison, que nos permitirá la asignación de una expresión a una variable:

```
instruccion:
IDENTIFICADOR '=' expresion { printf("Asignando a %s el valor %lf\n", $1, $3); }
| expresion ;
```

Luego modificaremos el axioma para aceptar instrucciones:

```
axioma: instruccion '\n' { printf ("Resultado=%lf\n", $1) ; } r_axioma ;
```

Por último también realizaremos un pequeño cambio en el analizador léxico para asegurarnos que "=" sea reconocido como un literal:

```
[a-zA-Z][a-zA-Z0-9]* { yylval = strdup(yytext); return IDENTIFICADOR; }
"="          { return '='; }
```

4.5 Intenta adaptar el analizador lexicográfico correspondiente.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "calc4.tab.h" // Archivo generado por Bison
%}

%% [ \t]+ /* Ignorar espacios y tabulaciones */
\n { return '\n'; }
[a-zA-Z][a-zA-Z0-9]* { yylval.str = strdup(yytext); return IDENTIFICADOR; }
"="                                                 { return '='; }
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMERO; }
.                   { return yytext[0]; } /* Retornar otros caracteres como literales */
%%

 int yywrap() { return 1; }
```

4.6 ¿Qué problema ves a la hora de implementar un analizador sintáctico que sea totalmente operativo (a nivel semántico) con las últimas cuestiones propuestas?

El principal problema es la gestión de la tabla de símbolos y la asignación de valores a las variables. Además, se debe manejar la semántica de las expresiones que involucran variables, lo que puede complicar la implementación del analizador sintáctico. También es necesario asegurarse de que las variables estén correctamente definidas antes de su uso, lo que puede requerir un análisis adicional.