# IMPLEMENTACIÓN DE UNA CALCULADORA DE BOLSILLO USANDO LEX Y YACC.

**Coromoto León[1], Casiano Rodríguez[1], Francisco de Sande[1]**

*[1]U.L.L. (Universidad de La Laguna.)*
*e-mail: (cleon|crguezl|fsande)@ull.es*

**Resumen:** En este trabajo se presenta un ejemplo de la actividad desarrollada en la enseñanza de las asignaturas de compiladores en el Centro Superior de Informática de la Universidad de La Laguna. Se aborda la construcción de una calculadora de bolsillo utilizando las herramientas de ayuda a la construcción de compiladores LEX y YACC. En el laboratorio propuesto se introducen los conceptos de listas recursivas por la derecha y por la izquierda, los conflictos que surgen durante el análisis sintáctico ascendente por desplazamiento y reducción y la asignación de precedencia y asociatividad a operadores.

## 1.- INTRODUCCIÓN

En el Centro Superior de Informática de la Universidad de La Laguna se imparte la docencia para obtener el título de Ingeniero en Informática. Esta Universidad diversifica la materia troncal Procesadores de Lenguajes en dos asignaturas: Procesadores de Lenguajes I y Procesadores de Lenguajes II. El objetivo de las asignaturas es proporcionar algunas técnicas básicas de escritura de compiladores. Una propuesta de contenidos se desarrolla en [1]. En las prácticas de laboratorio de las asignaturas de Procesadores de Lenguajes [2],[3], se divide la implementación de un compilador en tareas más reducidas que se plantean como laboratorios prácticos interelacionados entre sí. Con estos ejercicios prácticos se quiere mostrar la complejidad en el diseño y la escritura de un compilador a mano. Por ello se potencia el uso de herramientas que faciliten el diseño y la implementación de compiladores como son las herramientas LEX y YACC.

El *Aprendizaje Activo* o *Aprendizaje por Descubrimiento* es una forma de denominar a un conjunto de aproximaciones a la educación que dan un

mayor protagonismo a los estudiantes en el proceso de aprendizaje. El elemento común en este tipo de aproximación es que el profesor suaviza su papel magistral de conductor y de presentador frente a la clase. En lugar de esto, se induce a los estudiantes a la autoenseñanza y el instructor se convierte más en un entrenador o un asistente en dicho proceso.
En las secciones siguientes se muestra un ejemplo concreto de aprendizaje activo. Corresponde al programa de Procesadores de Lenguajes I. El ejemplo ilustra una situación real de clase.

## 2.- ¿LISTAS RECURSIVAS POR LA DERECHA O POR LA IZQUIERDA?

A estas alturas del curso el alumno/a ya conoce el funcionamiento de la herramienta YACC (o *bison*), así como la herramienta LEX (o *Flex*). Comenzamos pidiéndole que consideren la gramática de la Figura 1, que reconoce el lenguaje de las expresiones aritméticas con dígitos y operadores de suma y producto.

$E \rightarrow E$ '-' $T \mid T$
$T \rightarrow T$ '*' $F \mid F$
$F \rightarrow$ **DIGITO**

Figura 1. Gramática de las expresiones

```
/* Expresiones con restas y productos */
%token DIGIT
%%
L  : ???
   ;
E  : ???
   ;
%%
main(){
  printf("A. Sintáctico>%s\n",
  ((yyparse()==0)?"correcto": "incorrecto"));
}
```

Figura 2 . Fichero EXPR.Y

Esta gramática no es ambigua y refleja la precedencia y asociatividad de los operadores admitidos. Se le propone al alumno que amplíe la gramática para que reconozca cero o más expresiones del estilo descrito separadas por retornos de carro ('\n') y que construya un programa YACC. Se le proporciona el analizador léxico y el esqueleto de programa de la Figura 2.
La segunda parte de este primer ejercicio consiste en añadir las acciones semánticas necesarias para obtener el resultado de evaluar dichas expresiones y mostrar por pantalla el valor obtenido en cada caso. Los alumnos suelen proponer soluciones a este problema en un plazo breve. Sin embargo, se plantea un debate acerca del lugar que debe ocupar el separador (en este caso el '\n') en la lista de expresiones así como la cuestión de si la lista debe ser recursiva por la derecha o por la izquierda. Muchos de ellos

también suelen olvidar la necesidad de finalizar las listas, dejándolas sin acabar ya sea con un único símbolo o considerando listas vacías.

Si la lista se hace recursiva por la derecha (Figura 3), el algoritmo de análisis sintáctico por desplazamiento y reducción LALR, provoca un resultado inesperado. Si se le proporciona la entrada **1-2-3 \n1-2*3 \n1*2-3\n^Z** nos muestra los resultados de la evaluación de las expresiones en orden inverso, esto es: **>-1 > 5 > -4**, cuando se esperaba que mostrara **>-4 > 5 > -1**. Esto se debe a que en las listas recursivas por la derecha se espera a haber desplazado todos los elementos hacia la pila antes de realizar la primera reducción, por lo tanto la última expresión de la lista será la primera en ejecutar la acción semántica asociada que es mostrar su valor por pantalla, luego lo hará la antepenúltima y así sucesivamente hasta llegar a la primera.

```
%%
L : E '\n' L {printf("> %d", $1);}
  | E '\n'   {printf("> %d", $1);}
  ;
```

Figura 3 . Lista recursiva por la derecha.

Para obtener el resultado deseado, se ha de implementar una lista recursiva por la izquierda como la de la Figura 4. En este caso, se reduce la primera vez que se encuentra una expresión y se ejecuta la acción semántica de mostrar el resultado por pantalla para la expresión que se especifica en primer lugar, luego para la segunda, la tercera y así sucesivamente.

```
%%
L : L E '\n' {printf("> %d", $2);}
  | E '\n'   {printf("> %d", $1);}
  ;
```

Figura 4 . Lista recursiva por la izquierda.

## 3.- CONFLICTOS

A continuación se le propone a los estudiantes que consideren la siguiente gramática, equivalente a la de la Figura 1 pero ambigua:

$$E \rightarrow E\ '-'\ E \mid E\ '*'\ E \mid \textbf{DIGITO}$$

Nótese que al compilar con YACC este nuevo programa, nos indica que se han producido 4 conflictos Desplazamiento/Reducción. Cuando las gramáticas que se dan como especificación a YACC son ambiguas el algoritmo LALR que implementa generará conflictos en las acciones del analizador sintáctico. YACC informará del número de conflictos en las acciones del análisis sintáctico que se generen. A menos que se indique lo contrario, YACC resolverá todos los conflictos en las acciones del análisis sintáctico utilizando las dos reglas siguientes:

- Un conflicto Reducción/Reducción se resuelve eligiendo la producción en conflicto que aparezca primero en la especificación YACC.
- Un conflicto Desplazamiento/Reducción se resuelve en favor del desplazamiento (esta regla resuelve correctamente el conflicto Desplazamiento/Reducción que se deriva del ***else*** ambiguo).

Como éstas son las reglas por defecto, no siempre reflejan lo que el diseñador del compilador persigue. YACC proporciona un método general para resolver los conflictos Desplazamiento/Reducción asignando precedencias y asociatividades tanto a las unidades léxicas (tokens) como a las variables sintácticas implicadas en el conflicto. Si se debe elegir entre hacer un desplazamiento con símbolo de entrada '*a*' o hacer una reducción por la producción $A \rightarrow a$ , YACC ejecuta:

| | |
|---|---|
| Reducción | si la precedencia de la producción $A \rightarrow a$ es mayor que la de '*a*', |
| | o bien, si la precedencia es la misma y la producción es asociativa por la izquierda. |
| Desplazamiento | en cualquier otro caso. |

Después de explicado el funcionamiento de la resolución de conflictos, y a la vista de que el nuevo traductor no funciona de forma correcta, es necesario realizar las modificaciones apropiadas al fichero de entrada para YACC. El mal funcionamiento de la nueva aproximación, es debida a la forma en la que se solucionan por defecto los conflictos. A continuación se le plantea al alumno/a el problema de construir un programa YACC en el que no aparezcan conflictos.

## 4.- BUSCANDO LA SOLUCIÓN USANDO ASOCIATIVIDAD

Cuando se añade a la especificación YACC, las declaraciones necesarias para que ambos operadores, el menos '-' y el producto '*', sean "Asociativos por la Izquierda" (Figura 5) desaparecen los cuatro conflictos Desplamiento / Reducción que se producían. Supongamos que el símbolo a analizar es "-" y la regla por la que hay que reducir es $E \rightarrow E$ '-' $E$. La precedencia del "-" es la que se le haya asignado en la sección de declaraciones. La precedencia de la regla de producción es la del último token que aparece en la misma, en este caso la precedencia de la regla es la del símbolo "-". Así

```
%left '-' '*'
%%
E : E '-' E   {$$ = $1 - $3;}
  | E '*' E   {$$ = $1 * $3;}
  | DIGITO
  ;
```

Figura 5 . Especificación de asociatividad.

pues, el símbolo y la regla tienen la misma precedencia. La operación de resta es asociativa por la izquierda, por lo tanto, la regla de producción es asociativa por la izquierda y el conflicto Desplazamiento/Reducción que se presenta se resuelve mediante una reducción por la regla de producción $E \rightarrow E$ '-' $E$. Sin embargo, si se realizan las misma pruebas que hasta el momento se evidencia que el traductor sigue sin funcionar en algunos casos.

## 5.- UNA SOLUCIÓN SIN CONFLICTOS USANDO PRECEDENCIA

Por lo tanto es necesario añadir a la especificación YACC, las declaraciones necesarias para que el operador producto, "*", tenga mayor precedencia que el operador menos "-" (Figura 6). Consideremos ahora que el símbolo a analizar es "*" y la regla por la que hay que reducir es $E \rightarrow E$ '-' $E$. La precedencia que se ha asignado al producto "*" en la sección de declaraciones es mayor que la de la resta "-" y la precedencia de la regla en conflicto es la del símbolo "-". Por lo tanto, el conflicto Desplazamiento/Reducción que se presenta en este caso se resuelve mediante un desplazamiento. Al realizar las pruebas se comprueba que el traductor obtenido esta vez funciona de forma correcta.

```
%left '-'
%left '*'
%%
E : E '-' E   {$$ = $1 - $3;}
  | E '*' E   {$$ = $1 * $3;}
  | DIGITO
  ;
```

Figura 6 . Especificación de Precedencia.

## 6.- AÑADIENDO NUEVOS OPERADORES

Consideremos el problema de escribir una calculadora que admita operandos en doble precisión. La entrada para YACC, ha de reconocer las operaciones habituales de suma, resta, producto, división y exponenciación y también de otras funciones de interés matemático tales como el seno, coseno, el logaritmo, etc. Para reconocer las expresiones binarias habituales se ha de utilizar la siguiente gramática de las expresiones que amplía a la anterior: E → E '+' E | E '-' E | E '*' E | E '/' E | E '^' E | '-' E | '(E)' | DIGITO

Se cuenta con una tabla de precedencias y asociatividades que resuelven de forma satisfactoria los conflictos Desplazamiento/Reducción tal y como se hizo en los dos apartados anteriores. Sin embargo, se presenta un problema nuevo al tratar el operador "-" que pasa a tener dos significados, uno para restar y otro para indicar un número negativo. Por lo tanto, es necesario establecer la precedencia y asociatividad adecuadas para el operador menos unario.

Consideremos ahora que el símbolo a analizar es "*" y la regla por la que hay que reducir es $E \rightarrow$ '-' $E$. La precedencia que se ha asignado al producto "*" en la sección de declaraciones es mayor que la de la resta "-" y la precedencia de la regla en conflicto es la del símbolo "-" (que coincide con la de la regla $E \rightarrow E$ '-' $E$). Por lo tanto, el conflicto Desplazamiento/Reducción que se presenta en este caso se resuelve mediante un desplazamiento, lo cual sería incorrecto. Utilizando la directiva ***%prec*** que proporciona YACC podemos asignarle a la regla de producción $E \rightarrow$ '-' $E$ una precedencia y una asociatividad distinta de la del símbolo "-". En el caso anterior, si consideramos la asignación de precedencia de la Figura , la precedencia de la regla de producción $E \rightarrow$ '-' $E$ es la del MENOSUNARIO, por lo tanto es mayor que la del símbolo "*" por lo tanto en el conflicto se optará por la reducción. La implemetación de la operación de potenciación se puede realizar usando la función "pow" de la librería "math" de C. Asociado a la unidad léxica (token) DIGITO se ha de tener un atributo en el que se almacena el valor del número reconocido en formato de doble precisión.

```
%left '-'
%left '*'
%left MENOSUNARIO
%%
E : E '-' E   {$$ = $1 - $3;}
  | E '*' E   {$$ = $1 * $3;}
  ...
  | '-' E %prec MENOSUNARIO {$$ = $2;}
  | DIGITO
  ;
```

Figura 7 . Uso de la directiva *%prec.*

Se pide a continuación al alumno, que amplíe la calculadora desarrollada para que admita funciones matemáticas. La calculadora también tiene que permitir al usuario la definición y uso de sus propias variables. Para conseguir esto, ha de utilizar las directivas %union y %type. Se han de permitir asignaciones múltiples y llamadas a funciones anidadas. La figura 8 muestra un ejemplo de una sesión con la calculadora multifunción (calcmf).

```
%calcmf
pi = 3.141592653589
3.1415926536
sin(pi)
0.0000000000
alpha = betha1 = 2.3
2.3000000000
alpha
2.3000000000
ln(alpha)
0.8329091229
%
```

Figura 8 . Sesión con la calculadora multifunción.

## 7.- CONCLUSIONES

El principal inconveniente de la metodología del aprendizaje activo reside en la cantidad de tiempo que requiere. Además, puede producir desequilibrios entre los diferentes grupos de la clase, debido a diferencias de formación y motivación. Por esta razón creemos que es conveniente aplicarla sólo en cursos pequeños y la hemos adoptado sólo en las clases de laboratorios prácticos, combinándola con clases magistrales complementadas con ejercicios en las horas de teoría y problemas.

## 8.- REFERENCIAS

[1] León C., Sande F., *Las asignaturas de Procesadores de Lenguajes en la Universidad de La Laguna*. JENUI'97. pp.435-442.

[2] León C., *Prácticas de Compiladores en C y Pascal*. Gobierno de Canarias, 1997.

[3] León C., Rodríguez C., Sande F., *Laboratorios de Compiladores*. JENUI'98. pp.468-480.