INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

GARZÓN DOMÍNGUEZ GERARDO ISMAEL

3CV18

COMPILADORES

PRÁCTICA 5:
BISON

INTRODUCCIÓN

A continuación se presentan conceptos básicos para el desarrollo de esta práctica.

Intuitivamente, una gramática es un conjunto de reglas que generan un subconjunto L de $\Sigma*$,para algún un alfabeto Σ, esto es: $L \subseteq \Sigma *$. Estas reglas permiten reemplazar símbolos o cadenas de símbolos hasta que finalmente se puede probar que alguna cadena $w \in L$, a este proceso se le conoce como derivación. Bajo esta descripción, los símbolos que son reemplazados por otros símbolos son llamados no terminales, mientras que los símbolos que no pueden ser reemplazados por otros símbolos son referidos como terminales, por último, las reglas bajo las cuales está permitida la sustitución de símbolos son llamadas producciones. **[1]**

Un analizador sintáctico recibe la salida del analizador léxico, esto es, una secuencia de tokens, que en términos prácticos significa una secuencia de lexemas con significado y un valor asociado si es que es aplicable, alguna bibliografía describiría esto como una tabla de tokens o similar y depende de los implementadores del analizador que estructura de datos usa para representar esta información.

Bison es un generador de analizadores sintácticos (o parsers en inglés) de propósito general que convierte una gramática libre de contexto en un analizador determinista LR o generalizado LR. **[2]** .

Para obtener resultados con este programa debemos de darle como entrada a Bison una gramática libre de contexto, comunmente esto se realiza mediante una descripción de la misma en una forma que se conoce como “BNF” o *Backus-Naur Form*. **[2]** .

Hay varias subclases importantes de gramáticas libres de contexto. Aunque Bison puede manejar casi todas las gramáticas libres de contexto, Bison está optimizado para las gramáticas LR (1). En resumen, estas gramáticas son posibles de analizar revisando que reglas de producción de la gramática que son aplicables según el símbolo con que comienza un token una posición después del que se está analizando actualmente **[2]** .

Los analizadores de gramáticas LR (1) son deterministas, esto significa que la siguiente regla gramatical que se aplicará en cualquier punto de la entrada está determinada de forma única por la entrada anterior y una parte fija y finita (la cual llamamos *lookahead* y se denota como el número en paréntesis de la clase de la gramática, en este caso 1 símbolo,) de la entrada restante. Una gramática libre de contexto puede ser ambigua, lo que significa que hay varias formas de aplicar las reglas gramaticales para obtener las mismas entradas. **[2]** .

DESARROLLO

En esta sección se muestra el proceso para generar un parser sencillo para una gramática de operaciones matemáticas con orden de precedencia describiendola mediante BNF y procesandola mediante Bison. Además, se utilizó el programa Flex para generar la parte del analizador léxico.

Este parser será capaz de interpretar expresiones matemáticas de números de punto flotante con orden de precedencia, esto es, la gramática aceptará cadenas que representen números que puedan ser comprobados mediante la expresión regular: *[0-9]+"\."{1}[0-9]+* y que estén operados entre sí con algún operador binario, como +,-.*,/, y en el caso de la exponenciación se representará mediante la letra mayúscula 'E', que separa a la base y al exponente.

A continuación se muestra parte del archivo que define la gramática y el cual se le da a bison para que genere el analizador.

```
%{
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
%}

%union
{
    double floating;
}

%token <floating> FPNUM
%type <floating> fexpr

%left '+'
%left '-'
%left '*'
%left '/'
%left 'E'

%%

input: /*  */ | input line;

line: '\n' | fexpr '\n' {printf("bison, resultado %lf\n", $1);}
;

fexpr: FPNUM {$$ = $1;} | fexpr '+' fexpr {$$ = $1+$3;} | fexpr '*' fexpr {$$ = $1*$3;}
     | fexpr '/' fexpr {$$ = $1/$3;} | fexpr 'E' fexpr {$$ = pow($1, $3);}
;

%%
```

Figura 1. Definición de la gramática para una calculadora de números de punto flotante sencilla.

Vale la pena describir de manera breve que está sucediendo en este archivo. Las primeras líneas son directivas para incluir las declaraciones necesarias para poder usar ciertas funciones como *printf, pow,* y *strtod* en el archivo para flex.

Note que después se expresa que un no terminal de tipo *FPNUM* puede tener asociado un valor de tipo *double*, una linea después, se expresa que los no terminales de tipo fexpr pueden tener asociados un valor de tipo *double*.

Note que los no terminales llamados *fexpr* están básicamente definidos de manera recursiva de tal manera que una expresión en su caso base puede ser una constante flotante *FPNUM* y dada su definición de recursiva, esta puede expandirse a n constantes flotantes o expresiones separadas por un operador binario. Note que se hace uso de las pseudovariables de bison para que cuando se cumpla una regla o producción de la gramática, se realice la operación correspondiente.

A continuación se muestran algunos ejemplos del funcionamiento correcto del analizador generado por Bison, cabe mencionar que los operandos de la gramática solo pueden ser números representados con punto decimal independientemente si el punto decimal se use o no.

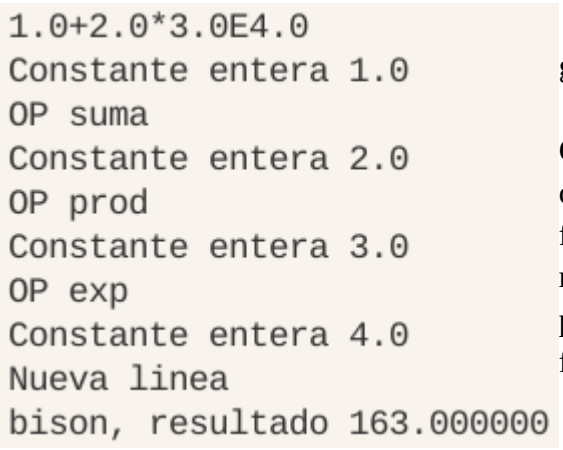

Figura 2. Ejemplo 1 de funcionamiento del parser generado por Bison.

Como se puede notar el orden de operaciones se realizo como es de esperarse, esto se debe a que Bison ofrece una forma de desambiguar este tipo de gramáticas, esto es, mediante la directiva *%left* se puede definir un orden de precedencia de los operadores, tal como se hizo en la figura 1.

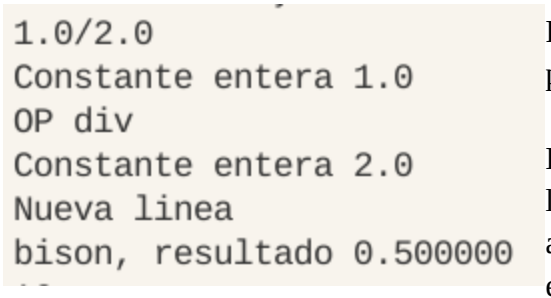

Figura 3. Ejemplo 2 de funcionamiento del parser generado por Bison.

En este caso se muestra una operación bastante sencilla, en la última figura se muestra una cadena que llevará al analizador a fallar debido a que no se reconoce el tipo de entrada que se le da.

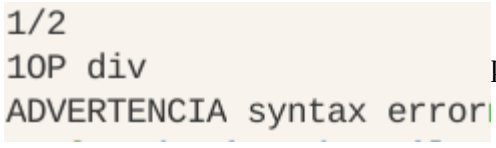

Figura 4. Ejemplo 3 del funcionamiento del parser generado por Bison, en este caso el parser falla.

Como se puede notar, esta vez el parser falla debido a que las cadenas numéricas no cumplen con la expresión regular de números con punto decimal.

CONCLUSIÓN

En esta práctica se notó la importancia de los analizadores sintácticos ya que a mi entendimiento, además de ser usados exclusivamente por compiladores de lenguajes de programación, tienen usos en muchas más áreas, puedo imaginar que tienen uso en situaciones donde se requiera análisis de gramáticas para otros propósitos, incluso puede que uno de los usos más comunes que tienen es en las calculadoras, ya que estas deben de “parsear” la entrada para saber cuales son las operaciones que deben de realizar y con que operandos.
Puedo decir que veo el potencial y la gran variedad de usos que pueden tener los analizadores sintáctico.

REFERENCIAS

[1] J. Anderson, Automata theory with modern applications, 1ra ed. Cambridge, New York, Melbourne: Cambridge University Press, 2006, p. 114.

[2] Free Software Foundation, Inc. Bison Manual 3.8.1, 10 Sep 2021. Accedido: 13 Dic, 2021. [Online]. Disponible en: https://www.gnu.org/software/bison/manual/bison.html