INSTITUTO POLITÉNICO NACIONAL

Escuela Superior de Cómputo

Práctica 1 “Calculadora de números complejos”.

Alumno: Rodríguez Martínez Heber Emanuel

5 DE OCTUBRE DE 2021

COMPILADORES 3CM15

Docente: Roberto Tecla Parra

Índice

Introducción

Los lenguajes de programación son notaciones que describen los cálculos a las personas y las máquinas. Nuestra percepción del mundo en que vivimos depende de los lenguajes de programación, ya que todo el software que se ejecuta en todas las computadoras se escribió en algún lenguaje de programación. Pero antes de poder ejecutar un programa, primero debe traducirse a un formato en el que una computadora pueda ejecutarlo. Por otro lado, este traductor puede ser conocido como compilador, y un compilador es un programa que puede leer un programa en un lenguaje (el lenguaje fuente) y traducirlo en un programa equivalente en otro lenguaje (el lenguaje destino).

Un compilador se compone de varios elementos, entre ellos, un analizador sintáctico, de donde podemos comenzar a hablar sobre YACC, el cual es un programa para generar analizadores sintácticos. Las siglas del nombre significan Yet Another Compiler-Compiler, es decir, "Otro generador de compiladores más". Genera un analizador sintáctico (la parte de un compilador que comprueba que la estructura del código fuente se ajusta a la especificación sintáctica del lenguaje) basado en una gramática analítica escrita en una notación similar a la BNF. Yacc genera el código para el analizador sintáctico en el Lenguaje de programación C.

La especificación de YACC funciona de la siguiente manera:

A partir de un fichero fuente en YACC, se genera un fichero fuente en C que contiene el analizador sintáctico. Sin embargo, un analizador sintáctico de YACC no puede funcionar por sí solo, sino que necesita un analizador léxico externo para funcionar. Dicho de otra manera, la fuente en C que genera YACC contiene llamadas a una función YYLEX() que debe estar definida y debe devolver el tipo de lexema encontrado. Además, es necesario incorporar también una función YYERROR(), que será invocada cuando el analizador sintáctico encuentre un símbolo que no encaja en la gramática.

Una especificación en YACC tiene tres secciones de gran importancia, estas son:

- Declaraciones: Esta sección contiene declaraciones previas para el uso del analizador (esta puede estar vacía).
- Reglas: Esta sección contiene una o más reglas gramaticales de la forma A:
  - Cuerpo {Acciones} donde, A representa un nombre no terminal, Cuerpo representa una secuencia de cero o más nombres y literales y, Acciones representa una sentencia de C donde se pueden producir entradas, salidas o llamadas a subprogramas.
- Programas: Dentro de esta sección se encuentran las funciones propias de la especificación.

Desarrollo

En el proceso de desarrollo de la práctica se hará uso de los siguientes códigos, el código que se tiene para el analizador léxico (complejo_cal.l) y una especificación que se tiene del lenguaje c (Calcu_compleja.h) donde podremos encontrar cada una de las funciones que serán usadas como acciones para cada regla que se diseñó para el análisis dentro del compilador. Como parte del inicio de la especificación, se definirá el token, con la respectiva asociatividad de los operadores, todo esto con la finalidad de poder determinar la precedencia de los operadores de cada nivel.

```
%token CNUMBER
%left '+' '-'
%left '*' '/'
```

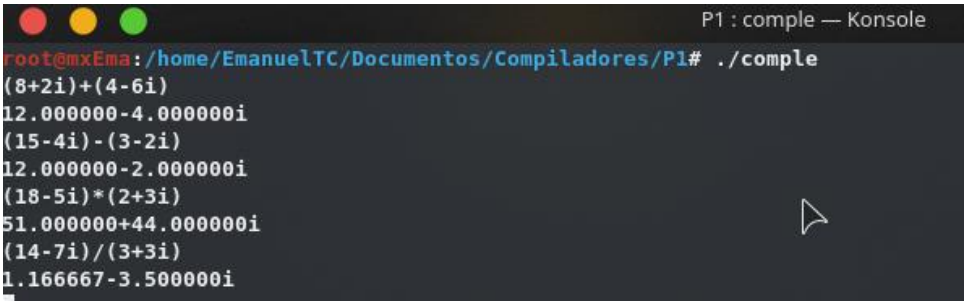
Como segundo paso, se definirán las reglas gramaticales que conforman a nuestro analizador, en el cual, se emplearán los archivos donde se encuentra la especificación donde se encuentran las acciones para cada tipo de cadena no terminal.

```
void warning(char *s, char *t);

void *(*add_)(void *dob1, void *dob2)=Complejo_add;
void *(*sub_)(void *dob1, void *dob2)=Complejo_sub;
void *(*mul_)(void *dob1, void *dob2)=Complejo_mul;
void *(*div_)(void *dob1, void *dob2)=Complejo_div;
void (*print_)(void *dob)=imprimirC;
```

```
list:
     | list '\n'
     | list exp '\n'   { print_($2); }
     ;
exp: CNUMBER           { $$ = $1;  }
     | exp '+' exp      { $$ = add_($1, $3);  }
     | exp '-' exp      { $$ = sub_($1, $3);  }
     | exp '*' exp      { $$ = mul_($1, $3);  }
     | exp '/' exp      { $$ = div_($1,$3);  }
     | '(' exp ')'      { $$ = $2;}
     ;
%%
```

Finalmente, se ejecutará el programa para mostrar los resultados de la misma.



Conclusiones

Durante el desarrollo de esta práctica pude reforzar mis conocimientos en la materia de compiladores, y específicamente el tema de YACC

Mediante el uso de un código externo de análisis léxico, se puede destacar que es una de las características más importantes, puesto que permite mantener un control sobre el código YACC, aunado a eso, facilita el uso de otras especificaciones externas para asignar acciones a las reglas gramaticales que componen a nuestro analizador.

Finalmente, me es indispensable comentar la relevancia de comprender la estructura y comportamiento principal de una especificación en YACC, por su gran importancia en el conocimiento sus oportunidades y limitantes en su servicio durante la creación de compiladores.