COMPILADORES
REPORTE #1
HOC1 -- NÚMEROS COMPLEJOS

1 de Noviembre del 2018

PROFESOR: TECLA PARRA ROBERTO

ALUMNO: SALDAÑA AGUILAR ANDRÉS

GRUPO: 3CM8

HOC1 es una calculadora básica, para el desarrollo de esta práctica, se hizo uso de YACC, FLEX y el lenguaje de programación C.

Yacc
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.

Flex
Lex es un programa para generar analizadores léxicos (en inglés scanners o lexers). Lex se utiliza comúnmente con el programa yacc que se utiliza para generar análisis sintáctico. Lex, escrito originalmente por Eric Schmidt y Mike Lesk, es el analizador léxico estándar en los sistemas Unix, y se incluye en el estándar de POSIX. Lex toma como entrada una especificación de analizador léxico y devuelve como salida el código fuente implementando el analizador léxico en C.

Desarrollo

En esta práctica se tiene como objetivo el diseño de una calculadora básica, con el uso de yacc, lex y un lenguaje de programación, que en este caso sera lenguaje C.

A continuación, un extracto de la lexica del programa, esta se encarga de buscar coincidencias entre la cadena tokenizar y las expresiones regulares que acepta el programa, haciendo el trabajo pertinente en cada caso y retornando finalmente el tipo de dato del cual se trata.

```
op [-+*/()]
ws [ \t]+
digits [0-9]
number (0|[1-9]+{digits}*)\.?{digits}*
im [i]
complexnum {ws}*[-]*{ws}*{number}{ws}*[+|-]{ws}*{number}{ws}*{im}{ws}*
%%
{complexnum} {
 double r, im;
 RmWs(yytext);
 sscanf(yytext,"%lf %lf",&r,&im);
 yylval=creaComplejo(r,im);
 return CNUMBER;}
{op} |
\n {return *yytext;}
{ws} { /* Do nothing */ }
. { /* Do nothing */ }
%%
```

Ahora, mostraremos a continuación la gramática para este programa, esta es la encargada de describir las reglas para poder producir las diferentes acciones que se pueden realizar con la calculadora basica, asi como que debe hacer el programa cuando éste encuentre una coincidencia con una de estas reglas de producción:

```
%token CNUMBER
%left '+' '-'
%left '*' '/'
%%
list:
        | list'\n'
   | list exp '\n'  { imprimirC($2); }
        ;
exp:    complex      { $$ = $1;  }
        | exp '+' exp     { $$ = Complejo_add($1,$3);  }
        | exp '-' exp     { $$ = Complejo_sub($1,$3);  }
        | exp '*' exp     { $$ = Complejo_mul($1,$3);  }
        | exp '/' exp     { $$ = Complejo_div($1,$3);  }
        | '(' exp ')'     { $$ = $2;}
        ;
complex: NUMBER '+' 'i' NUMBER { $$ = creaComplejo($1,$4); }
        |           NUMBER '-' 'i' NUMBER { $$ = creaComplejo($1,$4); }
        ;
%%
```

En las acciones gramaticales de “exp” tenemos la llamada los métodos que se encargan de las operaciones aritméticas de complejos como la suma,resta,multiplicacion y division.

En las acciones gramaticales de “complex”, mandamos a crear los números complejos dados dos números, uno para la parte imaginaria y otra para la parte compleja. Adicionalmente esta producción contiene el token number, que es el único token que tendremos en esta práctica.

Resultados

Una vez que se compila la gramática, léxica y lógica del programa podemos probar su funcionamiento:

```
Andress-MacBook-Pro-2:complejo andressaldana$ ./comp
(1+1i) - (1-1i)
0.000000+2.000000i
(1+1i) + (1+1i)
2.000000+2.000000i
(1+1i) * (1-1i)
2.000000
(1+1i) / (1-1i)
0.000000+1.000000i
```

Conclusión

El desarrollo de una calculadora básica no es tan difícil cuando entiendes el concepto de que es un analizador léxico y uno sintáctico, conceptos de teoría computacional como son una gramática y expresiones regulares, así el cómo se programan en YACC y FLEX.