



**Instituto Politécnico Nacional
Escuela Superior de Cómputo**

Compiladores

**Práctica #1
“HOC 1”**

3CM7

Alumno: Zepeda Flores Alejandro de Jesús

Profesor: Tecla Parra Roberto

INTRODUCCIÓN

HOC es un acrónimo para **High Order Calculator**, es un lenguaje de programación interpretado que fue usado en 1984 en el libro *“El Entorno de Programación de UNIX”* para demostrar como construir interpretes usando una herramienta llamada YACC y lenguaje C.

HOC fue desarrollado por Brian Kernighan y Rob Pike como una grandiosa calculadora interactiva. Su función básica es evaluar expresiones numéricas de puntos flotantes e.g. `“1+2*sin(0.7)”`. Después variables fueron agregadas, expresiones condicionales, ciclos, funciones definidas por el usuario, simple entrada/salida y más, todo esto usando una sintaxis parecida a lenguaje C.

Hasta ahora, las 6 etapas de HOC son:

- **HOC1: Calculadora Básica**
- HOC2: Calculadora con 26 variables
- HOC3: Calculadora Científica
- HOC4: Máquina Virtual de Pila
- HOC5: Ciclos / Decisiones
- HOC6: Funciones y Procedimientos

OBJETIVO

Usando el HOC1 proporcionado por el profesor, elegir un tipo de calculadora y modificarlo para que pueda hacer operaciones básicas como en HOC1 pero con el tipo de datos que se eligió.

Para esta práctica (y las que siguen) se eligió **números complejos**.

DESARROLLO

El profesor nos proporcionó con las funciones básicas para hacer operaciones con complejos, así de su definición. Así que sólo cambiamos el tipo de dato de la pila de YACC (además de agregar las bibliotecas de las funciones):

```
%union
{
    ComplejoAP val;
    double num;
}
```

Y también modificamos el tipo de Tokens y símbolos no terminales:

```
%token <num> NUMBER
%type <val> complejoNum complejo
%left '+' '-'
%left '*' '/'
%left '^'
%nonassoc '(' ')' '
```

Finalmente modificamos la sección de reglas:

```
%%


list:
    | list '\n'
    | list complejoNum '\n' {imprimirC($2);}
    ;

complejo:
    NUMBER '+' NUMBER 'i'          {;$2 = creaComplejo($1,$3);}
    | NUMBER '+' 'i'                {;$2 = creaComplejo($1,1);}
    | NUMBER '-' 'i'                {;$2 = creaComplejo($1,-1);}
    | NUMBER '-' NUMBER 'i'        {;$2 = creaComplejo($1,-$3);}
    | '-' NUMBER '+' NUMBER 'i'    {;$2 = creaComplejo(-$2,$4);}
    | '-' NUMBER '-' NUMBER 'i'    {;$2 = creaComplejo(-$2,-$4);}
    ;

complejoNum:
    complejo
    | complejoNum '+' complejoNum  {;$2 = Complejo_add($1,$3);}
    | complejoNum '-' complejoNum  {;$2 = Complejo_sub($1,$3);}
    | complejoNum '*' complejoNum  {;$2 = Complejo_mul($1,$3);}
    | complejoNum '/' complejoNum  {;$2 = Complejo_div($1,$3);}
    | complejoNum '^' NUMBER        {;$2 = Complejo_pot($1,$3);}
    | '(' complejoNum ')'           {;$2 = $2;}
    ;

%%
```

Un ejemplo de cómo funciona:



```
alejandro@alejandrozf: ~/Escritorio/Practicas/Practica1
Archivo Editar Ver Buscar Terminal Ayuda
alejandro@alejandrozf:~/Escritorio/Practicas/Practica1$ yacc -d complejo_cal.y
alejandro@alejandrozf:~/Escritorio/Practicas/Practica1$ gcc complejo_cal.c y.tab.c -o complejo_cal
alejandro@alejandrozf:~/Escritorio/Practicas/Practica1$ ./complejo_cal
1+i + 1-i
2.00
1+i - 1-i
0.00+2.00i
1+i^8
16.00
1+i^9
16.00+16.00i
```

CONCLUSIONES

Aunque al principio me costo un poco de trabajo de identificar un error que tuve, esta práctica me resulto muy interesante y me puso a pensar en las cosas que se podrían desarrollar con YACC. Mucho más porque es sencillo, y aunque YACC sólo esta para Linux, el código generado puedo ser usado en otros sistemas operativos, como en Windows.