UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE CIENCIAS DE LA COMPUTACIÓN

CC0E3 Compiladores

Laboratorio N°03 A

TRABAJO PARA EL CURSO DEL COMPILADORES

Alumno	Código
-Armijo Ramos Carlos Armijo	20142588B

Profesor: Jaime Osorio Ubaldo

Índice

1. Problemática

La Programación Orientada a Objetos(en adelante POO) es uno de los paradigmas de programación más usado en la actualidad. Aunque esto se debe más que todo a la explosión de popularidad del lenguaje de programación Java(que es un lenguaje con una fuerte orientación a Objetos) durante la última década, sería ingenuo ignorar los fundamentos de POO, que en su forma más básica preceden a cualquier lenguaje que lo implemente.

Expuesto la importancia de este paradigma en la actualidad, nos interesa una forma fácil de entender y acceder a sus principales principios. Pues aunque básicos y fundamentales, pueden ser difíciles de comprender en su totalidad. Este es la problemática que abordaremos en este trabajo.

2. Objetivo general

El objetivo general consiste en desarrollar un compilador simple que permita ilustrar los principios fundamentales de la POO. En vista que nuestro público objetivo es mayormente de habla hispana, consideramos adecuado usar palabras clave en español.

3. Objetivos específicos

El lenguaje debe ser capaz se mostrar los cuatro principios fundamentales del POO. La abstracción, encapsulamiento, herencia y polimorfismo. Eso implica ser capaz de definir clases, subclases e instanciar objetos. Las palabras claves a usar serán tomandas en su mayoría del lenguaje Java, pero traducidas al español.

4. Resultados esperados

Cuando el trabajo esté terminado, deberíamos ser capaces de escribir código como el siguiente:

```
Clase Vehículo{
    privado variable1
    privado variable2

    publico Vehiculo()
    publico esFuncional(){versión del método 1}

}

Clase Automovil extends Vehiculo{
    publico Automovil(
```

```
    publico esFuncional(){version del metodo 2}
}

Vehiculo vehiculo1 = nuevo Vehiculo()
Automovil auto1 = nuevo Automovil()

vehiculo1.variable1 = valor
auto1.variable = valor2
```

5. Herramientas,métodos y procedimientos

Para el analizador léxico planeamos usar lex en su implementación más conocida, la cual es flex.
Para el analizador sintáctico planeamos usar bison, el cual es un generador de analizadores sintácticos compatible con yacc.
Ambos programas(flex y bison) ayudan a generar un analizador léxico y un analizador sintáctico en C++. El output despues de estos dos procesos es un AST(Abstract Syntax Tree/Árbol de sintaxis de abstracto) que luego se debe traducir posiblemente a lenguaje máquina o assembler para luego proceder a producir un ejecutable.

6. Delimitación del proyecto

Nuestro proyecto no apunta a crear un lenguage de programación çompleto.[en] el sentido formal de la palabra. Pero queremos ser capaces de observar comportamientos propios de POO como la herencia y el polimorfismo, incluso si solo se es capaz de definir métodos simples para las clases.

7. Justificación y viabilidad

La creación de un lenguaje orientado a objetos al menos en una versión temprano no es revolucionaria ni nueva, pero el proceso de crearlo puede proveer un punto de inicio didáctico y muy útil para alguien interesado en aprender los fundamentos de como construir un compilador.

8. Nombre del compilador

Para el proyecto designado hemos decidido nombrar a nuestro pequeño lenguaje como ggEsp (good game Esp)

9. Estado del arte

Hemos considerado dos trabajos como referencia.

El primer trabajo que consideramos se titula **“El Diseño e Implementación de un compilador para un Lenguaje de Definición de Datos Orientado a Objetos´´**[2]. Como sabemos, los modelos de datos clásicos: El relacional y el jerárquico, no permiten relaciones tipo POO como la herencia y el polimorfismo. Para implementar estas características en un modelo de datos, desarrollan su propio modelo de datos O-ODM(Modelo de Datos Orientado a Objetos) y un compilador para su lenguaje(Lenguaje de Definición de Datos orientado a Objetos o abreviado O-ODDL). Este compilador se encarga de traducir el O-ODDL a un lenguaje que sea procesable por el MDBS, que es un Sistema de Base de Datos Multilingüe producido por la Escuela Naval de Posgrado.

El segundo trabajo es titulado **“Polyglot: Un Sistema de Compilación Extendible para Java´´**[1] es la construcción de Polyglot, del cual rescatamos su flexibilidad y facibilidad de usar, al permitir implementar DSLs(Lenguajes específicos de dominio) en Java de manera eficiente. En este trabajo podemos observar diversos principios usados en un constructor de lenguajes, aunque no es un çompilador"propiamente dicho, pues se crea ya sobre la estructura de Java.

10. Definición de los tokens usando bison

CLASE	clase	ENTRE	/
IGUAL	=	PCOMA	;
VAR	var	COMA	,
MAS	+	PI	)
MENOS	-	PD	(
POR	*	CI	{
HEREDA	her	CD	}
RETURN	return		

11. Definición de los patrones de los tokens

La gramática de la definición de las clases es la siguiente:

```
programa: listaclases;
listaclases: clase listaclases | clase;

clase: CLASE ID CI listaatributos constructor listametodos CD
        | CLASE ID HEREDA ID CI
        listaatributos constructor listametodos CD;
listaatributos: atributo PCOMA listaatributos
        | atributo PCOMA;
atributo: type ID;
```

```
constructor: ID PI listavar PD CI listainicializacion CD;

listavar: type ID COMA listavar
        | type ID
        | ;

listainicializacion: inicializacion PCOMA listainicializacion
        | inicializacion PCOMA;
inicializacion: ID IGUAL ID;

listametodos: metodo listametodos
        | metodo;
metodo: type ID PI listavar PD CI asignaciones RETURN ID PCOMA CD;
asignaciones: asignacion PCOMA asignaciones
        | asignacion PCOMA
        | ;
asignacion: ID IGUAL expresion;
expresion: expresion MAS termino
        | expresion MENOS termino
        | termino;
termino: termino POR factor
        | termino ENTRE factor
        | factor;
factor: NUM
        | ID
        | PI expresion PD;
```

12. Características y ventajas de su compilador respecto a los existentes

Las ventajas de mi compilador respecto a otros son dos principalmente:

1. Las palabras clave son abreviaciones de palabras en español, lo cual lo hace más accesible a personas sin conocimiento de inglés.

2. La implementación nativa de los principios de la Programación Orientada a Objetos(parcialmente logrado).

13. Implementación de la gramática de operaciones aritméticas

```
        %%
prog: expr '\n'   prog { printf("VALOR %g\n",$1);  };
prog:  ;
expr: expr '+'  term    { $$ = $1 + $3; }
    | expr '-' term     { $$ = $1 - $3; }
    | term {$$=$1;}
term: term '*' factor   { $$ = $1 * $3; }
    | term '/' factor   { $$ = $1 / $3  }
    | factor {$$=$1;};
factor: NUMBER     { $$ = $1; }
    |  '(' expr ')' {$$ = $2;};

%%
```

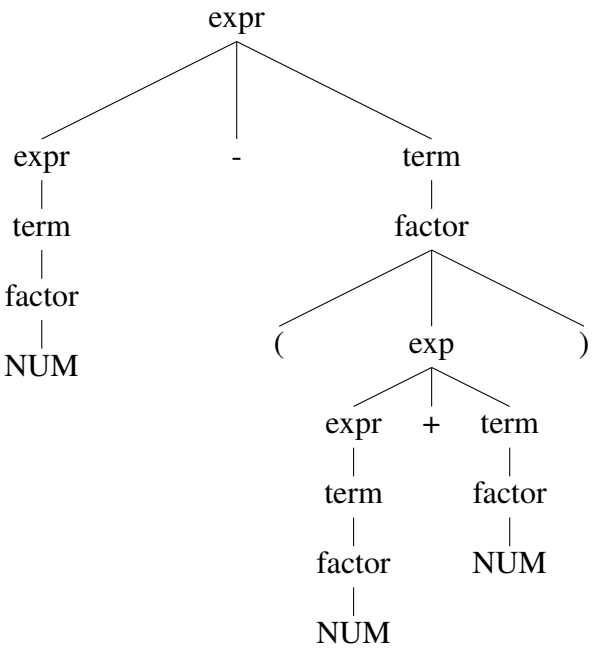
14. Algunos cálculos de op. aritméticas en nuestro compilador

```
C:\Users\Alumno\Desktop\parsertest>calcC2.tab
4*20 - (12*4) - 18/6
VALOR 29

C:\Users\Alumno\Desktop\parsertest>calcC2.tab
3 + (23 - 3) - 18/2
VALOR 14
```

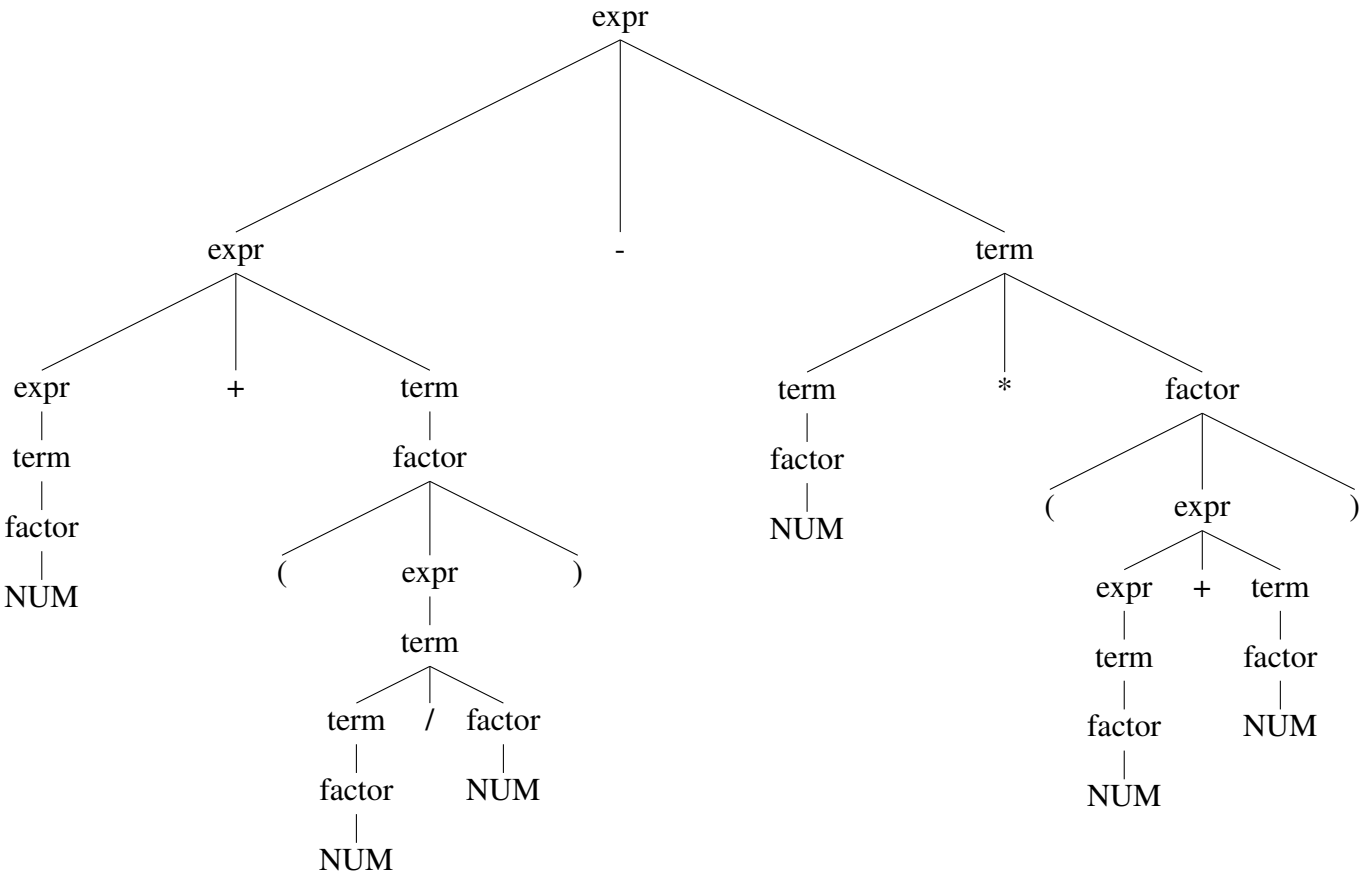
```
C:\Users\Alumno\Desktop\parsertest>calcC2.tab
0.4*40 + (20+7) + 2/3
VALOR 43.6667

C:\Users\Alumno\Desktop\parsertest>calcC2.tab
0.2*10 + (23-3+10*4) - 2/4*8
VALOR 58
^C
```

15. Árboles de sintaxis

Expresión: 25 - (12 + 26)
En tokens es: NUM - (NUM + NUM)



Expresión: 11 + (3 / 2) - 3 * (4 + 8)
En tokens: NUM + (NUM / NUM) - NUM * (NUM + NUM)



16. Tabla de Símbolos y Tabla de Código

La **Tabla de Símbolos** es una tabla que guarda los valores de todos los tokens que hemos definido en nuestro código.

Asimimos **la Tabla de Código** es una tabla que guarda todas las operaciones definidas en tu código, tal como las **lasignaciones, sumas, restas, divisiones**, etc.

A la gramática se le agrega ciertas sentencias al final de cada token, y al costado de cada producción que sea una operación. Tal como se muestra continuación:

```
%%
asignacion: ID {$$=localizaSimb(lexema,ID);} IGUAL expresion PCOMA | ;
expresion:    expresion MAS termino
                {int i=GenVarTemp();generaCodigo(SUMAR,i,$1,$3);$$=i;}
            | expresion MENOS termino
                {int i=GenVarTemp();generaCodigo(RESTAR,i,$1,$3);$$=i;}
            | termino ;
termino:      termino POR factor
```

```
                {int i=GenVarTemp();generaCodigo(MULTIPLICAR,i,$1,$3);$$=i
            | termino ENTRE factor
                {int i=GenVarTemp();generaCodigo(DIVIDIR,i,$1,$3);$$=i;}
            | factor;
factor: NUM {$$=localizaSimb(lexema,NUM);}
      | ID {$$=localizaSimb(lexema,ID);} | PI expresion PD;

type: VAR;
%%
```

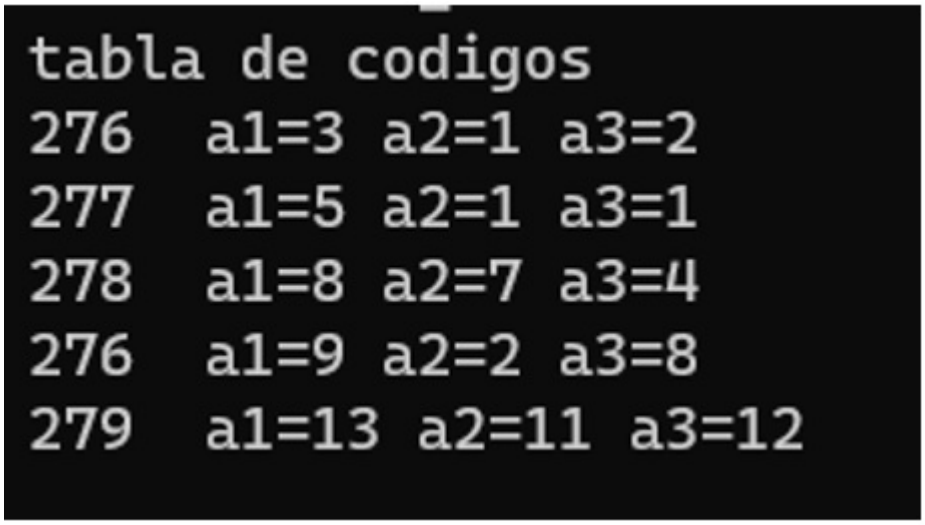
Para ilustrar el funcionamiento, vamos a mostrar un ejemplo. La **entrada** que vamos a pasar a nuestro parser simplemente será un archivo **.txt** sin extensión llamado **input**, el cual mostramos a continuación:

```
clase Animal {
    var vel;
    var peso;
    Animal(var vel, var peso){
        vel = vel;
        peso = peso;
    }
    var calcvel(var vel){
        vel = 23 + 32;
        return vel;
    }
    var calcpeso(var peso, var vel){
        densidad = 23 - 23;
        peso = 32 + 12*densidad;
        return peso;
    }
}

clase Vehiculo {
    var peso;
    var tamanyo;
    Vehiculo(var peso, var tamanyo){
        peso = peso;
        tamanyo = tamanyo;
    }
    var getpeso(){
        return peso;
    }
    var getTamanyo(){
    return tamanyo;
```

```
    }
}

clase Auto her Vehiculo {
    var distRec;
    var velMax;
    Auto(var dist, var vel){
        distRec = dist;
        velMax = vel;
    }
    var tiempoMin(var dist){
        tMin = dist/velMax;
        return tMin;
    }
}
```

Y a continuación mostramos la **tabla de símbolos** y la **tabla de código** que se muestra al pasar el input a nuestro parser en la línea de comandos.

```
tabla de simbolos
0  nombre=vel tok=259 valor=0.000000
1  nombre=23 tok=262 valor=23.000000
2  nombre=32 tok=262 valor=32.000000
3  nombre=_T0 tok=259 valor=0.000000
4  nombre=densidad tok=259 valor=0.000000
5  nombre=_T1 tok=259 valor=0.000000
6  nombre=peso tok=259 valor=0.000000
7  nombre=12 tok=262 valor=12.000000
8  nombre=_T2 tok=259 valor=0.000000
9  nombre=_T3 tok=259 valor=0.000000
10  nombre=tMin tok=259 valor=0.000000
11  nombre=dist tok=259 valor=0.000000
12  nombre=velMax tok=259 valor=0.000000
13  nombre=_T4 tok=259 valor=0.000000
```



17. Bibliografía

Referencias

[1] Nathaniel Nystrom. Polyglot: An Extensible Compiler Framework for Java. URL: https://www.cs.cornell.edu/andru/papers/polyglot.pdf.

[2] Luis M. Ramirez. The design and implementation of a compiler for the object-oriented data definition lan URL: https://core.ac.uk/download/pdf/36727786.pdf.