



UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS
FACULTAD DE INGENIERÍA

Docente

Canaval Sánchez, Luis Martin

Carrera: Ciencias de la Computación

Estudiante: Espíritu Cueva, Renzo Andree

Código: U202113340

Lima

Índice

Código de Ejemplo Utilizado (DOT language)	3
Implementación del Parser.....	4
Implementación de Lexer	5
Arbol Sintactico de Abstracción (AST).....	6
Github (Dot Compiler)	6

Código de Ejemplo Utilizado (DOT language)

```
digraph G {

    subgraph cluster_0 {
        style=filled;
        color=lightgrey;
        node [style=filled,color=white];
        a0 -> a1 -> a2 -> a3;
        label = "process #1";
    }

    subgraph cluster_1 {
        node [style=filled];
        b0 -> b1 -> b2 -> b3;
        label = "process #2";
        color=blue
    }

    start -> a0;
    start -> b0;
    a1 -> b3;
    b2 -> a3;
    a3 -> a0;
    a3 -> end;
    b3 -> end;

    start [shape=Mdiamond];
    end [shape=Msquare];
}
```

Implementación del Parser

```
parser grammar ExprParser;
options { tokenVocab=ExprLexer; }

program
    : graph
    ;

graph
    : gd=(GRAPH|DIGRAPH) id '{' stmt_list '}'      #Grafo
    ;

stmt_list
    : ( stmt ';' )*                                #Stc
    ;

stmt
    : node_stmt          #Nodo
    | edge_stmt          #Edge
    | attr_stmt          #Attr
    | id '=' id          #Assing
    ;

attr_stmt
    : gne=(GRAPH| NODE| EDGE) attr_list          #AList
    ;

attr_list
    : ('[' a_list '])+                                #Aarr
    ;

a_list
    : (id ('=' id)+ ',?')+                            #Ass
    ;

edge_stmt
    : (node_id|subgraph) edgeRHS? attr_list?      #Subgrafo
    ;

edgeRHS
    : EDGEOP (node_id|subgraph) edgeRHS?          #NodeEdge
    ;

node_stmt
    : node_id attr_list                            #NodeAttr
    ;

node_id
    : id port?                                     #NodeO
    ;
```

```

port
    : ':' id (':' id)?
    ;

subgraph
    : SUBGRAPH ID '{' stmt_list '}'          #Sub
    ;

id
    : ID                                     #Identify
    | STRING                                #String
    | NUMBER                                #Number
    ;

```

Implementación de Lexer

```
lexer grammar ExprLexer;
```

```

GRAPH : 'graph' ;
DIGRAPH : 'digraph';
SUBGRAPH : 'subgraph';
NODE : 'node';
EDGE : 'edge';

```

```

EQUAL : '=';
POINT : '.';
LCURLY : '{';
RCURLY : '}';
LBRACKET : '[';
RBRACKET : ']';

```

```

SEMICOLON : ';';
ENDC : ' ';

```

```
EDGEOP: '->' | '--';
```

```

ID: [a-zA-Z_][a-zA-Z_0-9]*;
STRING: '"' ('\\"|.)*? '"';
NUMBER: '-'? ('[0-9]+'|[0-9]+' ('[0-9]*')? );
WS: [ \t\n\r\f]+ -> skip ;

```

The figure displays two Abstract Syntax Trees (ASTs) for programs. The top tree is for 'program:1' and the bottom tree is for 'program:2'. Both trees show a hierarchical structure of statements, subgraphs, and nodes, with labels like 'id:1', 'edge_stmt:1', 'node_id:1', etc., indicating the abstract syntax tree structure.

<https://github.com/Raichi1/DotCompiler.git>