Manual Técnico

Introducción:

El siguiente documento presenta la descripción del proyecto final realizado en el curso de Organización de lenguajes y compiladores 1. El proyecto consiste en crear un lenguaje de programación fuertemente tipado llamado **'TypeSecure'**, este último pareciéndose en sintaxis con el lenguaje TypeScript. El lenguaje cuenta con las mismas funcionalidades de lenguajes de alto nivel como java. Aunque cuenta con la flexibilidad de la inferencia de tipos que nos brindan lenguajes de programación como Javascript. TypeSecure cuenta con cuatro tipos de datos los cuales son: number, string, boolean y bigint. Siendo los primeros tres iguales a los presentes en los lenguajes de programación y el último contando con la particularidad de ser un valor entero de 64 bits con terminación 'n'.

Plataforma de desarrollo:

- Java LTS 17
- IntelliJ
- Cup
- JFlex

Requisitos:

- Java LTS 17 como mínimo
- 16 GB RAM recomendable

Estructura del análisis léxico del lenguaje TypeSecure.

```
END_LINE = \r|\n|\r\n
WHITE_SPACE = {END_LINE}|[ \t\f]
PLUS = "+"
INCREMENT = "++"
MINUS = "-"
DECREMENT = "--"
TIMES = "*"
DIVIDE = "/"
MOD = "%"
LPAREN = "("
RPAREN = ")"
COLON = ":"
SEMICOLON = ";"
ASSIGN = "="
EQUALS = "=="
NOT_EQUALS = "!="
GREATER = ">"
LESS = "<"
LESS_EQ = "<="
GREATER_EQ = ">="
AND = "&&"
OR = "||"
NOT = "!"
COMMA = ","
DOT = "."
```

```
LBRACE = "{"
RBRACE = "}"
TRUE = "true"
FALSE = "false"
NUMBER = "number"
BIGINT = "bigint"
STRING = "string"
BOOLEAN = "boolean"
VOID = "void"
UNDEFINED = "undefined"
CONST = "const"
LET = "let"
NUMBER_FUN = "Number"
BIGINT_FUN = "BigInt"
STRING_FUN = "String"
BOOLEAN_FUN = "Boolean"
LENGTH = "length"
CHAR_AT = "charAt"
LOWER = "toLowerCase"
UPPER = "toUpperCase"
CONCAT = "concat"
CONSOLE_LOG = "console.log"
SYMBOL_TABLE = "getSymbolTable"
PRINT_AST = "printAst"
IF = "if"
ELSE = "else"
FOR = "for"
WHILE = "while"
DO = "do"
BREAK = "break"
CONTINUE = "continue"
FUNCTION = "function"
RETURN = "return"
E = "Math.E"
PI = "Math.PI"
SQRT_TWO = "Math.SQRT2"
ABS = "Math.abs"
CEIL = "Math.ceil"
COS = "Math.cos"
SIN = "Math.sin"
TAN = "Math.tan"
EXP = "Math.exp"
FLOOR = "Math.floor"
POW = "Math.pow"
SQRT = "Math.sqrt"
RANDOM = "Math.random"
INTEGER = 0 | [1-9][0-9]*
NUMBER_VALUE = {INTEGER}|{INTEGER} \. [0-9]+
BIGINT_VALUE = {INTEGER} "n"
STRING_VALUE = (\"[^\"]*\") | (\'[^\']*\')
ID = [a-zA-Z_][a-zA-Z0-9_]*
```

```
SYM = [#$~%½¬@·\?°ª\[\]]+
COMMENT = "//"[^\r\n]* | "/*" [^*]* ("*"[^/][^*]*)* "*/"
```

Estructura del análisis sintáctico del lenguaje TypeSecure.

```
main
::= instructions:list
   ;

instructions
::= instructions:list instruction:instruction
   |
   ;

parameters
::= parameters:list COMMA parameter:p
   | parameter:p
   ;

parameter
::= ID:id COLON type:variableType
   ;

type
::= NUMBER
   | BIGINT
   | STRING
   | BOOLEAN
   ;



instruction
::= declarationInstruction:i SEMICOLON
   | funInstruction:i
   | assignmentInstruction:i SEMICOLON
   | ifInstruction:i
   | forInstruction:i
   | whileInstruction:i
   | doWhileInstruction:i
   | inDecInstruction:i SEMICOLON
   | callFunction:i SEMICOLON
   | consoleInstruction:i SEMICOLON
   | continueInstruction:i SEMICOLON
   | breakInstruction:i SEMICOLON
   | returnInstruction:i SEMICOLON
   | error SEMICOLON
   ;

funInstruction
```

```
::= FUNCTION ID:id LPAREN parameters:paramList RPAREN returnType:type
LBRACE instructions:insList RBRACE
        |   FUNCTION   ID:id   LPAREN   RPAREN   returnType:type   LBRACE
instructions:insList RBRACE
   ;

returnType
::= COLON VOID:v
   | COLON NUMBER:n
   | COLON BIGINT:b
   | COLON STRING:s
   | COLON BOOLEAN:b
   |
   ;


ifInstruction
::= IF:ifI LPAREN w:val RPAREN LBRACE instructions:trueBlock RBRACE
    | IF:ifI LPAREN w:val RPAREN LBRACE instructions:trueBlock RBRACE
elseIfInstruction:falseBlock
   ;

elseIfInstruction
::= ELSE ifInstruction:ifI
   | ELSE:elI LBRACE instructions:insList RBRACE
   ;

forInstruction
::=  FOR:fr  LPAREN  declarationInstruction:d  SEMICOLON  w:o  SEMICOLON
assignmentInDec:i RPAREN LBRACE instructions:list RBRACE
   ;

assignmentInDec
::= inDecInstruction:i
   | assignmentInstruction:i
   ;

inDecInstruction
::= | ID:id INCREMENT:i
   | ID:id DECREMENT:d
   ;

whileInstruction
::= WHILE:wl LPAREN w:val RPAREN LBRACE instructions:list RBRACE
   ;

doWhileInstruction
::= DO:doI  LBRACE  instructions:list  RBRACE  WHILE  LPAREN  w:val  RPAREN
SEMICOLON
   ;
```

```
declarationInstruction
::= decType:decType declarationList:list
   ;

declarationList
::= declarationList:list COMMA ID:id idType:variableType ASSIGN w:val
   | declarationList:list COMMA ID:id idType:variableType
   | ID:id idType:variableType
   | ID:id idType:variableType ASSIGN w:val
   ;

idType
::= COLON type:type
   |
   ;

decType
::= LET:t
   | CONST:t
   ;

assignmentInstruction
::= ID:id ASSIGN w:val
   ;

callFunction
::= ID:id LPAREN ids:ids RPAREN
   | ID:id LPAREN RPAREN
   | PRINT_AST:print LPAREN w:val RPAREN
   |SYMBOL_TABLE:table LPAREN RPAREN
   | mathFun:math
   ;

ids
::= ids:list COMMA w:val
   | w:val
   | error COMMA
   ;

consoleInstruction
::= CONSOLE_LOG:console LPAREN ids:ids RPAREN
   | error RPAREN
   ;

continueInstruction
::= CONTINUE:cont
   ;

breakInstruction
::= BREAK:brk
   ;
```

```
returnInstruction
::= RETURN:rtn
   | RETURN:rtn w:val
   ;

w
::= w:lft OR:op x:rgt
   | x:val
   ;

x
::= x:lft AND:op y:rgt
   | y:val
   ;

y
::= NOT:op z:rgt
   | z:val
   ;

z
::= z:lft EQUALS:op a:rgt
   | z:lft NOT_EQUALS:op a:rgt
   | z:lft LESS:op a:rgt
   | z:lft GREATER:op a:rgt
   | z:lft LESS_EQ:op a:rgt
   | z:lft GREATER_EQ:op a:rgt
   | a:val
   ;

a
::= a:lft PLUS:ps b:rgt
   | a:lft MINUS:ms b:rgt
   | b:val
   ;

b
::= b:lft TIMES:tm c:rgt
   | b:lft DIVIDE:dv c:rgt
   | b:lft MOD:md c:rgt
   | c:val
   ;

c
::= d:val
   | d:val DOT stringFun:string
   ;

d
::= castFun:cast LPAREN w:val RPAREN
```

```
   | MINUS:m e:d
   | PLUS e:d
   | e:d
   ;

e
::= NUMBER_VALUE:number
   | BIGINT_VALUE:bigInt
   | TRUE:trueVal
   | FALSE:falseVal
   | STRING_VALUE:string
   | ID:id
   | callFunction:callFunction
   | mathConst:math
   | LPAREN w:val RPAREN
   ;

castFun
::= NUMBER_FUN:nbr
   | BIGINT_FUN:bg
   | STRING_FUN:st
   | BOOLEAN_FUN:bl
   ;

stringFun
::= LENGTH:lngt
   | CHAR_AT:ct LPAREN a:val RPAREN
   | LOWER:lw LPAREN RPAREN
   | UPPER:up LPAREN RPAREN
   | CONCAT:cct LPAREN a:val RPAREN
   | error RPAREN
   ;

mathFun
::= ABS:bs LPAREN ids:ids RPAREN
   | CEIL:cl LPAREN ids:ids RPAREN
   | COS:cs LPAREN ids:ids RPAREN
   | SIN:sn LPAREN ids:ids RPAREN
   | TAN:tn LPAREN ids:ids RPAREN
   | EXP:xp LPAREN ids:ids RPAREN
   | FLOOR:fr LPAREN ids:ids RPAREN
   | POW:pw LPAREN ids:ids RPAREN
   | SQRT:srt LPAREN ids:ids RPAREN
   | RANDOM:rm LPAREN RPAREN
   ;

mathConst
::= E:e
   | PI:pi
   | SQRT_TWO:sqrt
   ;
```

Estructura del Proyecto:
Todo lo realizado en el proyecto se encuentra dentro de las carpetas: com.mio.typeSecure Hay que mencionar que en este paquete se encuentra la clase principal **App** encargada de ejecutar el programa. Ahora se intentará dar un breve resumen de lo que ocurre en cada uno de los paquetes importantes:

Paquete compiler:
Contiene la clase Token utilizada tanto en la clase generada por JFlex y Cup. Dentro de este paquete existen dos paquetes más llamados **lexer** y **parser**, estos paquetes contienen las clases generadas por jflex y cup respectivamente.

Paquete controllers:
Contiene clases controladoras que sirven de apoyo a otras clases.

- **MainController:** controlador que sirve para comunicar el backend con la vista MainFrame.
- **CompilerController:** controlador que sirve como puente entre el CompilerPanel con TSParserController.
- **TSParserController:** controlador que sirve como apoyo para ejecutar el parser generado por cup.

Paquete models:
Paquete que cuenta con los modelos usados a lo largo del programa. Cuenta con una clase TSError (colocada ahí por falta de creatividad con los paquetes).

- **helpers:** Paquete que contiene todas las clases que sirven de apoyo para los visitor, La clase ReportHelper es la encargada de exportar toda la información a un archivo HTML. La clase OperationHelper es la encargada de realizar las operaciones binarias del visitor.
- **instructions:** Paquete que contiene todas las clases utilizadas para elaborar el AST.
- **symbolTable:** Paquete que contiene las clases relacionadas a la tabla de simbolos.
- **visitor:** Paquete que contiene las clases del patrón de diseño visitor. La clase Debugger es la encargada de realizar la primera pasada del compilador y encontrar la mayor parte de errores posibles. La clase Runner es la encargada de ejecutar el programa como tal.

Paquete utils:
Paquete utilizado para las utilidades del programa, cuenta con una sola clase que es la encargada de la lógica para colocar el número de línea en el textArea de la vista.

Paquete views:
Paquete que contiene las vistas usadas en el programa.