MANUAL TÉCNICO PROYECTO 2, COMPILADORES I 2023

Erick Daniel Morales Xicara
Carné: 201930699

Manual Técnico

El software puede ser utilizado en cualquier sistema operativo ya que es un proyecto de escritorio, hecho en java, también se puede ejecutar en la mayoría de computadores que tengan java 13 o superior. Se utilizó el JDK 17 de java, y el IDE IntelliJ.

El software consiste en el manejo de un lenguaje muy parecido a TypeScript, tomando en cuenta la mayoría de funciones y sintaxis de este lenguaje, siendo un intérprete, en el cual se pueden ejecutar los ciclos for, while, do while de cualquier lenguaje , a su vez, los if, else if, else y declaraciones de funciones, y variables. El software a su vez contiene varias clases, modelos y manejadores que permiten iniciar, ejecutar y realizar las instrucciones.

Lenguaje Type Secure

Paquete Models: este paquete contiene todos los modelos utilizados para poder manejar el lenguaje Type Secure

Las clases:

1. Assignment
2. Ast
3. Break
4. Call
5. Cast
6. Console Log
7. Continue
8. Declare
9. Do While
10. Else State
11. ForState
12. Function
13. GetTable
14. IfState
15. Instruccion
16. MethodMath
17. MethodString
18. OnlyAssingment
19. OperacionBinaria
20. OperationType
21. Parámetro
22. Return
23. Tabla Simbolos
24. Value
25. Variable

26. While

Todas las clases tienen un método accept que recibe un visitor y ejecuta el visitor que son clases que tienen el método visit dependiendo que tipo de clase sea.

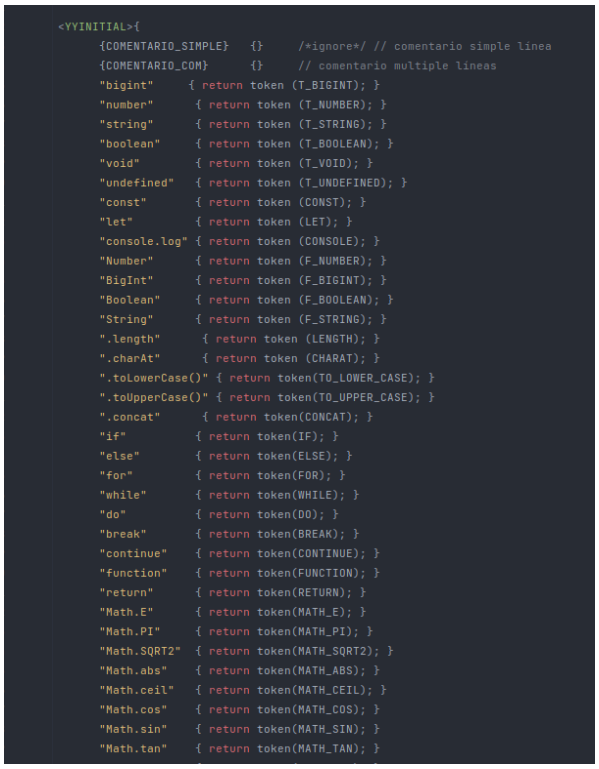
En el paquete cup contiene el jcup con las producciones y jflex contiene el jflex del lenguaje léxico definido :

El siguiente método es el que nos ayuda a poder recopilar los errores léxicos.

```
{SYM}     {return token(SYM);}
```

```
SYM =    [@¨~$¡?·]+
```

Y luego tenemos todos los tokens reconocidos



```
<YYINITIAL>{
    {COMENTARIO_SIMPLE}   {}     /*ignore*/ // comentario simple linea
    {COMENTARIO_COM}      {}     // comentario multiple lineas
    "bigint"     { return token (T_BIGINT); }
    "number"     { return token (T_NUMBER); }
    "string"     { return token (T_STRING); }
    "boolean"    { return token (T_BOOLEAN); }
    "void"       { return token (T_VOID); }
    "undefined"  { return token (T_UNDEFINED); }
    "const"      { return token (CONST); }
    "let"        { return token (LET); }
    "console.log" { return token (CONSOLE); }
    "Number"     { return token (F_NUMBER); }
    "BigInt"     { return token (F_BIGINT); }
    "Boolean"    { return token (F_BOOLEAN); }
    "String"     { return token (F_STRING); }
    ".length"     { return token (LENGTH); }
    ".charAt"     { return token (CHARAT); }
    ".toLowerCase()" { return token(TO_LOWER_CASE); }
    ".toUpperCase()" { return token(TO_UPPER_CASE); }
    ".concat"     { return token(CONCAT); }
    "if"         { return token(IF); }
    "else"       { return token(ELSE); }
    "for"        { return token(FOR); }
    "while"      { return token(WHILE); }
    "do"         { return token(DO); }
    "break"      { return token(BREAK); }
    "continue"   { return token(CONTINUE); }
    "function"   { return token(FUNCTION); }
    "return"     { return token(RETURN); }
    "Math.E"     { return token(MATH_E); }
    "Math.PI"    { return token(MATH_PI); }
    "Math.SQRT2" { return token(MATH_SQRT2); }
    "Math.abs"   { return token(MATH_ABS); }
    "Math.ceil"  { return token(MATH_CEIL); }
    "Math.cos"   { return token(MATH_COS); }
    "Math.sin"   { return token(MATH_SIN); }
    "Math.tan"   { return token(MATH_TAN); }
```

```
"Math.floor"  { return token(MATH_FLOOR); }
"Math.pow"    { return token(MATH_POW); }
"Math.sqrt"   { return token(MATH_SQRT); }
"Math.random()" { return token(MATH_RANDOM); }
"printAst"    { return token(PRINT_AST); }
"getSymbolTable" { return token(GET_SYMBOL_TABLE); }
{TRUE}          { return token(TRUE); }
{FALSE}         { return token(FALSE); }
":"       { return token(DOS_PUNTOS);}
"=="      { return token (DOBLE_IGUAL);}
"<="      { return token(MENOR_IGUAL_QUE); }
">="      { return token(MAYOR_IGUAL_QUE); }
"<"       { return token(MENOR_QUE); }
">"       { return token(MAYOR_QUE); }
"!="      { return token(DISTINTO_QUE); }
"!"       { return token(NOT);}
"||"    { return token(OR); }
"&&"    {return token(AND);}
"("       {return token(L_PARENT);}
")"       {return token(R_PARENT);}
"{"       {return token(L_LLAVE);}
"}"       {return token(R_LLAVE);}
"++"      {return token(MAS_MAS);}
"--"    {return token(MENOS_MENOS);}
"-"       {return token(MENOS);}
"+"       {return token(MAS);}
"%"       {return token(MOD);}
"/"       {return token(DIVIDE);}
"*"       {return token(MULTIPLY);}
"="       {return token(IGUAL);}
";"       {return token(PUNTO_COMA);}
","       {return token(COMA);}
(\"[^\"]*\")  { return token(CADENA, yytext().substring(1,yylength()-1)); }
(\'[^\']*\')       { return token(CADENA, yytext().substring(1,yylength()-1)); }
([a-zA-Z_][a-zA-Z0-9_]*) { return token(LITERAL); }
{ENTERO}              {return token(ENTERO);}
{NUM_DECIMAL}   {return token(NUM_DECIMAL);}
{BIG_INT}   {return token(BIG_INT);}
{SYM}    {return token(SYM);}
{whiteSpace}  {/*ignore*/}
[^] {return token(ERROR);}
```

Este método es para controlar los errores en el parser

```
parser code {:
        public ParserSecure(SecureLex lexer){
            super(lexer);
        }
       public Symbol scan() throws Exception {
                Symbol symbol = this.getScanner().next_token();
                if (symbol == null) {
                    return this.getSymbolFactory().newSymbol("END_OF_FILE", this.EOF_sym());
                }

                while(symbol != null && symbol.sym == ParserSecureSym.SYM) {
                    this.report_expected_token_ids();
                    System.out.println("Ignorando: " + symbol.value.toString());
                    Token token = (Token) symbol.value;
                   errorForClient.add(new ObjectErr(token.getLexeme(),token.getLine(),token.getColumn(), "LEXICO","No existe esta cadena en el lenguaje"));
                    symbol = this.getScanner().next_token();
                }

                if (symbol == null) {
                    return this.getSymbolFactory().newSymbol("END_OF_FILE", this.EOF_sym());
                }

                return symbol;
        }

        public void report_error(String message, Object info) {
            System.out.println("public void report_error");
        }

        public void report_fatal_error(String message, Object info) {
            System.out.println("public void report_fatal_error");
        }

        public void syntax_error(Symbol cur_token) {
            Token token = (Token) cur_token.value;
```

Y los métodos para poder solicitar algunos enum dependiendo del string

```
    public Variable.TypeV returnTypeV(String data){
          if(data.equalsIgnoreCase("CONST")){
              return Variable.TypeV.CONST;
          }else if(data.equalsIgnoreCase("LET")){
              return Variable.TypeV.LET;
          }
          return null;
    }
    public Cast.CastType returnTypeCast(String data){
          if(data.equalsIgnoreCase("NUMBER")){
              return Cast.CastType.NUMBER;
          }else if(data.equalsIgnoreCase("BIGINT")){
              return Cast.CastType.BIGINT;
          }else if(data.equalsIgnoreCase("BOOLEAN")){
              return Cast.CastType.BOOLEAN;
          }else if(data.equalsIgnoreCase("STRING")){
              return Cast.CastType.STRING;
          }
          return null;
    }
    public MethodString.MethodType returnTypeMethodString(String data){
          if(data.equalsIgnoreCase(".TOLOWERCASE()")){
              return MethodString.MethodType.LOWERCASE;
          }else if(data.equalsIgnoreCase(".TOUPPERCASE()")){
              return MethodString.MethodType.UPPERCASE;
          }else if(data.equalsIgnoreCase(".CONCAT")){
              return MethodString.MethodType.CONCAT;
          }else if(data.equalsIgnoreCase(".LENGTH")){
              return MethodString.MethodType.LENGTH;
          }else if(data.equalsIgnoreCase(".CHARAT")){
              return MethodString.MethodType.CHARAT;
          }
          return null;
    }
```

Y la gramática utilizada:

```
inic ::=
       instrucciones:
;

instrucciones ::=
               instrucciones instruccion
            |  instruccion
;

instruccion ::=
                   declare_pro PUNTO_COMA
            |   console_pro
            |   settear
            |   if_pro
            |   for_pro
            |   do_while_pro
            |   while_pro
            |   sumadores_pro PUNTO_COMA
            |   funtion_pro
            |   call_funtion PUNTO_COMA
            |   assig_pro PUNTO_COMA
            |   return_pro PUNTO_COMA
            |   continue_pro PUNTO_COMA
            |   break_pro  PUNTO_COMA
            |   get_sym
            |   error PUNTO_COMA
;

declare_pro ::=
           type_f declare
;
declare ::= declare COMA assignacion_pro
      |  assignacion_pro
;

assignacion_pro ::=
               LITERAL DOS_PUNTOS type IGUAL a
         |  LITERAL DOS_PUNTOS type
         |   LITERAL IGUAL a
;
```

```
array_assig_pro::=
       array_assig_pro COMA assig_pro
       | assig_pro
;
assig_pro ::=
         |   LITERAL IGUAL a
;

type ::=  T_NUMBER
       | T_BIGINT
       | T_STRING
      | T_BOOLEAN
      | T_VOID
      | T_UNDEFINED
      |

;

type_f ::=  CONST
         | LET
;

console_pro ::=
           CONSOLE L_PARENT expr R_PARENT PUNTO_COMA
;
expr ::=
       expr COMA a
       | a
;
if_pro ::= IF:pro_if L_PARENT a:pi2 R_PARENT L_LLAVE instrucciones:pi4 R_LLAVE else_pro
         |  IF:pro_if L_PARENT a:pi2 R_PARENT L_LLAVE R_LLAVE else_pro
;
else_pro ::=
              ELSE:pro_else IF L_PARENT a:pro2 R_PARENT L_LLAVE instrucciones:pro4 R_LLAVE
else_pro:
         | ELSE:pro_else L_LLAVE instrucciones:pro2 R_LLAVE

         | ELSE:pro_else L_LLAVE  R_LLAVE
         |
;

for_pro ::=
                    FOR:id  L_PARENT  declare_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
sumadores_pro:incre  R_PARENT L_LLAVE instrucciones:instr R_LLAVE

                  |  FOR:id  L_PARENT  declare_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
assig_pro:incre  R_PARENT L_LLAVE instrucciones:instr R_LLAVE
:}
```

```
                  |  FOR:id  L_PARENT  declare_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
sumadores_pro:incre  R_PARENT L_LLAVE R_LLAVE

                  |  FOR:id  L_PARENT  declare_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
assig_pro:incre  R_PARENT L_LLAVE  R_LLAVE

              |   FOR:id  L_PARENT  array_assig_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
sumadores_pro:incre  R_PARENT L_LLAVE instrucciones:instr R_LLAVE

              |   FOR:id  L_PARENT  array_assig_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
assig_pro:incre  R_PARENT L_LLAVE instrucciones:instr R_LLAVE

              |   FOR:id  L_PARENT  array_assig_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
sumadores_pro:incre  R_PARENT L_LLAVE  R_LLAVE

              |   FOR:id  L_PARENT  array_assig_pro:decla  PUNTO_COMA  a:compare  PUNTO_COMA
assig_pro:incre  R_PARENT L_LLAVE R_LLAVE

;

while_pro ::=
          WHILE:prod L_PARENT a:prod3 R_PARENT L_LLAVE instrucciones:prod6 R_LLAVE

         | WHILE:prod L_PARENT a:prod3 R_PARENT L_LLAVE  R_LLAVE



;

do_while_pro::=
             DO:prod L_LLAVE instrucciones:prod6  R_LLAVE WHILE L_PARENT  a:prod3 R_PARENT
PUNTO_COMA

         | DO:prod L_LLAVE  R_LLAVE WHILE L_PARENT  a:prod3 R_PARENT PUNTO_COMA
;
funtion_pro ::=
                   FUNCTION:prod LITERAL:id L_PARENT parametros:param R_PARENT DOS_PUNTOS
type:pro_tipo L_LLAVE instrucciones:prod1 R_LLAVE

                  |   FUNCTION:prod LITERAL:id L_PARENT parametros:param R_PARENT  L_LLAVE
instrucciones:prod1 R_LLAVE

                  |   FUNCTION:prod LITERAL:id L_PARENT parametros:param R_PARENT  L_LLAVE
R_LLAVE

;
parametros ::=
            parametros:prod COMA parame:prod1
```

```
            | parame:prod
;
parame ::=
           LITERAL:id DOS_PUNTOS type:pro           |
;
call_funtion::=
              LITERAL:id L_PARENT call_f:asig R_PARENT

             |  LITERAL:id L_PARENT R_PARENT


;
call_f::=
        call_f:c COMA a:b
      | a:b

;
sumadores_pro::=
          LITERAL:id MAS_MAS
         | LITERAL:id MENOS_MENOS
;

continue_pro::=
   CONTINUE :conti
;
break_pro::=
   BREAK
;
return_pro::=
       RETURN:retu
;
get_sym::=
       GET_SYMBOL_TABLE:pro L_PARENT R_PARENT PUNTO_COMA



;

a ::= a:pro1 OR b:pro2
   | b:pro1
;

b ::= b:pro1 AND c:pro2
   | c:pro1
;

c ::= NOT c:pro1
   | d:pro1
;
```

```
d ::= d:pro1 DISTINTO_QUE e:pro2
   | d:pro1 MENOR_QUE e:pro2
   | d:pro1 MENOR_IGUAL_QUE e:pro2
   | d:pro1 MAYOR_QUE e:pro2
   | d:pro1 MAYOR_IGUAL_QUE e:pro2
   | d:pro1 DOBLE_IGUAL e:pro2
   | e:pro1 {:RESULT= pro1;:}
;
e ::=   e:pro1 MAS f:pro2
    | e:pro1 MENOS f:pro2
    | f:pro1
;
f ::=   f:pro1 MULTIPLY g:pro2
    | f:pro1 DIVIDE g:pro2
    |  f:pro1 MOD g:pro2
    | g:pro1 {:RESULT = pro1;:}
;
g::= F_NUMBER:pro1 L_PARENT a:pro2 R_PARENT
      | F_BIGINT:pro1 L_PARENT a:pro2 R_PARENT
      | F_BOOLEAN:pro1 L_PARENT a:pro2 R_PARENT
      | F_STRING:pro1  L_PARENT a:pro2 R_PARENT
      | method_math:pro1
;
method_math ::=
              MATH_E:prod
            | MATH_PI:prod
            | MATH_SQRT2:prod
            | MATH_ABS:prod L_PARENT h:prod2 R_PARENT
            | MATH_CEIL:prod L_PARENT h:prod2 R_PARENT
            | MATH_COS:prod L_PARENT h:prod2 R_PARENT
            | MATH_SIN:prod L_PARENT h:prod2 R_PARENT
            | MATH_TAN:prod L_PARENT h:prod2 R_PARENT
            | MATH_EXP:prod L_PARENT h:prod2 R_PARENT
            | MATH_FLOOR:prod L_PARENT h:prod2 R_PARENT
            | MATH_POW:prod L_PARENT a:prod2 COMA a:prod3 R_PARENT
            | MATH_SQRT:prod L_PARENT a:prod2 R_PARENT
            | MATH_RANDOM:prod
            | method_string:prod1
;
method_string ::=
               h:id type_method_string:prod2
               | h:id CHARAT L_PARENT a:prod R_PARENT   ,
               | h:id CONCAT L_PARENT a:prod R_PARENT
               |h:prod1 {:RESULT = prod1;:}
;
type_method_string ::=
                   TO_LOWER_CASE:prod {:RESULT=((Token)prod);:}
                |  TO_UPPER_CASE:prod {:RESULT=((Token)prod);:}
```

```
                |  LENGTH :prod          {:RESULT=((Token)prod);:}
;

h ::= MENOS i:pro1
 | MAS i:pro1
 | call_funtion:pro {:RESULT = pro;:}
 | i:pro1   {:RESULT=pro1;:}
;
i ::=  BIG_INT:id
   |ENTERO:id
   | NUM_DECIMAL:id
   | CADENA:id
   | FALSE:id
   | TRUE:id
   | LITERAL:id
/*    | call_funtion:pro {:RESULT=pro;:}*/
   | L_PARENT a:id R_PARENT {:RESULT= id; :}
/*    | call_funtion:pp {:RESULT= pp;:}*/
;
```

A simple vista podemos ver que primero se deben declarar las variables seguidas del otro tipo de instrucciones reconocidas, en la declaración de variables:

- Se acepta LET variable : string = “hola” es decir se declara una variable y se le puede asignar algo.
- Se acepta CONST variable, variable2 as string es decir se declara dos o más variables y se le asigna un tipo de variable
- Se acepta LET variable, variable2 : string = “hola” es decir se declara más de una variable y se le quiere asignar algo.

A su vez la gramática para los statements es bastante práctico, ya que podemos recibir instrucciones como:

- CONSOLE.LOG()
- IF
- ELSE
- ELSE IF
- FOR

- WHILE
- DO WHILE
- FUNCTION

que son como las principales,

- El IF, se controla por medio del if_stmt , el cual se modificó para tener el formato de un if común, donde el else_statement es el que indica que puede venir else if * , else o ninguno de estos.
- El WHILE, DO WHILE, FOR funcionan como en cualquier lenguaje de programación.

VISITOR

Por último tenemos el visitor, el cual es el encargado de toda la lógica para verificar el código:

DEBUG VISITOR:

Este es el encargado de leer todas las instrucciones y no verifica las comparaciones, simplemente ejecuta todas las instrucciones, y en for luego de verificarlas las añade a la lista de variables que utilizará el RUNNER.

RUNNER VISITOR: Este luego de haber leído todo el código ya contiene la lógica de los if y las demás instrucciones para ejecutarlas según sea el caso.