Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Curso: Laboratorio Organización de Lenguajes & Compiladores 1
Auxiliar: José Diego Pérez Toralla
Auxiliar: Maynor Octavio Piló Tuy

Manual Técnico

Juan F. Urbina S. **2019060651**
Sección: **C**

Guatemala, Mayo de 2023

Índice

Introducción

Dentro del segundo proyecto del laboratorio de Organización de Lenguajes & Compiladores 1, la aplicación realizada fue en base al uso de Jison para el lenguaje de programación de JavaScript, abordando la culminación de conceptos de gramáticas y expresiones regulares para crear un compilador y analizar por consiguiente los datos ingresados, la declaración de variables y cadenas, entre otras, mediante el archivo de prueba respectivo, la utilización de las listas para el almacenamiento de algunas variables dentro del archivo Jison y la utilización de memoria dinámica para almacenar las variables y tokens para la utilización posterior en lo que son las tablas de errores, tablas de símbolos y la generación de un árbol ast que contiene las concatenaciones de las respectivas variables y funciones que contienen valores e instrucciones aceptando mediante las expresiones regulares las variables y funciones correspondientes.

Se empleo el lenguaje de JavaScript como herramienta en la función de las estructuras para el análisis, almacenamiento de la información y la utilización del fronted y backend para el envío y recibimiento de información (en este caso no se utilizaron métodos de POST, UPDATE & GET) todo fue desarrollado para el uso dentro del fronted respecto a las funciones, para el análisis de archivos de información, envío del AST, obtención de tabla de errores, obtención de tabla de símbolos, etc.

Se pudo determinar que la implementación de las estructuras y el almacenamiento y/o arreglo de la información obtenida en cada una de ellas no mostro ningún problema durante su ejecución.

Requisitos del Sistema

- ***Sistema Operativo:*** Windows 7 o superior
- ***CPU:*** Intel Pentium D o AMD Athlon 64 (K8) 2.6GHz. (Requisitos Mínimo)
- ***RAM:*** 600MB
- ***Lenguaje Utilizado:*** JavaScript
- ***IDE:*** Visual Studio Code
- ***USO de Framework:*** React

Explicación del Código

Análisis Léxico

Utiliza las siguientes expresiones regulares para obtener los parámetros a la hora de leer el archivo, obteniendo los errores léxicos que tenga el archivo.

```
([a-zA-Z])([a-zA-Z0-9_])* return 'id'
[']\\\\[']|[']\\\"[']|[']\\\'[']|[']\\n[']|[']\\t[']|[']\\r[']|['].?['] return 'caracter'
[0-9]+("."[0-9]+)+\b    return 'doble'
[0-9]+                  return 'entero'

["]                     { cadena = ''; this.begin("string"); }
<string>[^"\\]+         { cadena += yytext; }
<string>"\\\""          { cadena += "\""; }
<string>"\\n"           { cadena += "\n"; }
<string>\s              { cadena += " ";  }
<string>"\\t"           { cadena += "\t"; }
<string>"\\\\"          { cadena += "\\"; }
<string>"\\\'"          { cadena += "\'"; }
<string>"\\r"           { cadena += "\r"; }
<string>["]             { yytext = cadena; this.popState(); return 'cadena'; }
```

```
"//".*                                  // comentario
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/] // comentario multiline

"clase"                 return 'prclase'
"double"                return 'prdouble'
"int"                   return 'printeger'
"boolean"               return 'prboolean'
"char"                  return 'prchar'
"string"                return 'prstring'
"list"                  return 'prlist'
"new"                   return 'prnew'
"add"                   return 'pradd'

"if"                    return 'prif'
"else"                  return 'prelse'
"switch"                return 'prswitch'
"case"                  return 'prcase'
"break"                 return 'prbreak'
"while"                 return 'prwhile'
"for"                   return 'prfor'
"do"                    return 'prdo'
"default"               return 'prdefault'
"continue"              return 'prcontinue'
"return"                return 'prreturn'
"void"                  return 'prvoid'
"++"                    return 'incremento'
"--"                    return 'decremento'

"print"                 return 'prprint'
"toLower"               return 'prtoLower'
"toUpper"               return 'prtoUpper'
"length"                return 'prlength'
"truncate"              return 'prtruncate'
"round"                 return 'prround'
"typeof"                return 'prtypeof'
"toString"              return 'prtoString'
"toCharArray"           return 'prtoCharArray'
"exec"                  return 'prexec'
"main"                  return 'premain'

"true"                  return 'true'
"false"                 return 'false'
```

```
"||"                    return 'or'
"&&"                    return 'and'
"!="                    return 'diferente'
"=="                    return 'igualigual'
"!"                     return 'not'
"="                     return 'igual'
"<="                    return 'menorigual'
">="                    return 'mayorigual'
">"                     return 'mayor'
"<"                     return 'menor'
","                     return 'coma'
";"                     return 'ptcoma'
"."                     return 'punto'
":"                     return 'dospuntos'
"{"                     return 'labre'
"}"                     return 'lcierra'
"*"                     return 'multi'
"/"                     return 'div'
"-"                     return 'menos'
"+"                     return 'suma'
"^"                     return 'exponente'
"%"                     return 'modulo'
"("                     return 'pabre'
")"                     return 'pcierra'
"?"                     return 'interrogacion'
"["                     return 'cabre'
"]"                     return 'ccierra'
```

Precedencia de Operadores

Sirve para obtener un orden a la hora de hacer las operaciones de valores, cuales tienen mayor “relevancia” o cuales operar antes de otros dentro de una función u orden.

```
%left 'interrogacion'
%left 'or'
%left 'and'
%right 'not'
%left 'igualigual' 'diferente' 'menor' 'menorigual' 'mayor' 'mayorigual'
%left 'suma' 'menos'
%left 'multi' 'div' 'modulo'
%left 'exponente'
%left 'incremento','decremento'
%left umenos
%left 'pabre'
```

Análisis Sintáctico

Se encarga de obtener los errores sintácticos y mediante una gramática regular cada en cada producción obtener los datos requeridos.

Se analizarán las cadenas y separarán los tokens para verificarlos posteriormente.

```
%% /*Inicio gramatica*/

ini
    : ENTRADA EOF   { retorno = { parse: $1, errores: errores }; errores = []; return retorno; }
    | error EOF     { retorno = { parse: null, errores: errores }; errores = []; return retorno; }
;

ENTRADA
    : ENTRADA ENTCERO                   { if($2!=="") $1.push($2); $$=$1; }
    | ENTCERO                           { if($1!=="") $$=[$1]; else $$=[]; }
;

ENTCERO
    : FUNCIONBODY       { $$=$1 }
    | METODOBODY        { $$=$1 }
    | EXECBODY          { $$=$1 }
    | DEC_VAR           { $$=$1 }
    | DEC_VECT          { $$=$1 }
    | DEC_LIST          { $$=$1 }
;

FUNCIONBODY
    : TIPO id pabre pcierra labre INSTRUCCION lcierra                       { $$ = INSTRUCCION.nuevaFuncion($2, null, $6, $1, this._$.first_line, thi
    | TIPO id pabre pcierra labre lcierra                                   { $$ = INSTRUCCION.nuevaFuncion($2, null, [], $1, this._$.first_line, thi
    | TIPO id pabre LISTAPARAMETROS pcierra labre INSTRUCCION lcierra       { $$ = INSTRUCCION.nuevaFuncion($2, $4, $7, $1, this._$.first_line, this.
    | TIPO id pabre LISTAPARAMETROS pcierra labre lcierra                   { $$ = INSTRUCCION.nuevaFuncion($2, $4, [], $1, this._$.first_line, this.
    | TIPO_VECT id pabre pcierra labre INSTRUCCION lcierra                  { $$ = INSTRUCCION.nuevaFuncion($2, null, $6, {vector: $1}, this._$.first
    | TIPO_VECT id pabre pcierra labre lcierra                              { $$ = INSTRUCCION.nuevaFuncion($2, null, [], {vector: $1}, this._$.first
    | TIPO_VECT id pabre LISTAPARAMETROS pcierra labre INSTRUCCION lcierra  { $$ = INSTRUCCION.nuevaFuncion($2, $4, $7, {vector: $1}, this._$.first_l
    | TIPO_VECT id pabre LISTAPARAMETROS pcierra labre lcierra              { $$ = INSTRUCCION.nuevaFuncion($2, $4, [], {vector: $1}, this._$.first_l
    | TIPO_LIST id pabre pcierra labre INSTRUCCION lcierra                  { $$ = INSTRUCCION.nuevaFuncion($2, null, $6, {lista: $1}, this._$.first_
    | TIPO_LIST id pabre pcierra labre lcierra                              { $$ = INSTRUCCION.nuevaFuncion($2, null, [], {lista: $1}, this._$.first_
    | TIPO_LIST id pabre LISTAPARAMETROS pcierra labre INSTRUCCION lcierra  { $$ = INSTRUCCION.nuevaFuncion($2, $4, $7, {lista: $1}, this._$.first_li
    | TIPO_LIST id pabre LISTAPARAMETROS pcierra labre lcierra              { $$ = INSTRUCCION.nuevaFuncion($2, $4, [], {lista: $1}, this._$.first_li
    | TIPO error lcierra                                                    { $$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de func
    | TIPO_VECT error lcierra                                               { $$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de func
    | TIPO_LIST error lcierra                                               { $$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de func
;

METODOBODY
    : prvoid id pabre pcierra labre INSTRUCCION lcierra                     { $$ = INSTRUCCION.nuevoMetodo($2, [], $6, this._$.first_line, this._$.fi
    | prvoid id pabre pcierra labre lcierra                                 { $$ = INSTRUCCION.nuevoMetodo($2, [], [], this._$.first_line, this._$.fi
    | prvoid id pabre LISTAPARAMETROS pcierra labre INSTRUCCION lcierra      { $$ = INSTRUCCION.nuevoMetodo($2, $4, $7, this._$.first_line, this._$.fi
    | prvoid id pabre LISTAPARAMETROS pcierra labre lcierra                 { $$ = INSTRUCCION.nuevoMetodo($2, $4, [], this._$.first_line, this._$.fi
    | prvoid error lcierra                                                  { $$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de méto
;
```

```
EXECBODY
    : premain id pabre pcierra ptcoma                          { $$ = INSTRUCCION.nuevoExec($2, null, this._$.first_line, this._$.first_
    | premain id pabre LISTAVALORES pcierra ptcoma             { $$ = INSTRUCCION.nuevoExec($2, $4, this._$.first_line, this._$.first_co
    | premain error ptcoma                                     { $$ = ""; errores.push({ tipo: "Sintáctico", error: "Llamada de main no
;

LISTAPARAMETROS
    : LISTAPARAMETROS coma PARAMETROS   {$1.push($3); $$=$1;}
    | PARAMETROS                        {$$=[$1];}
;

PARAMETROS
    : TIPO_VECT id      {$$ = INSTRUCCION.nuevoParametro($2, {vector: $1}, this._$.first_line, this._$.first_column+1)}
    | TIPO_LIST id      {$$ = INSTRUCCION.nuevoParametro($2, {lista: $1}, this._$.first_line, this._$.first_column+1)}
    | TIPO id           {$$ = INSTRUCCION.nuevoParametro($2, $1, this._$.first_line, this._$.first_column+1)}
;

INSTRUCCION
    : INSTRUCCION INSCERO       { if($2!=="") $1.push($2); $$=$1; }
    | INSCERO                   { if($1!=="") $$=[$1]; else $$=[]; }
;

INSCERO
    : DEC_VAR                   {$$=$1}
    | SENTENCIACONTROL          {$$=$1}
    | SENTENCIACICLO            {$$=$1}
    | DEC_VECT                  {$$=$1}
    | DEC_LIST                  {$$=$1}
    | SENTENCIATRANSFERENCIA    {$$=$1}
    | LLAMADA ptcoma            {$$=$1}
    | FPRINT                    {$$=$1}
    | error ptcoma              {$$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de instrucción no válida.", linea: this.
    | error lcierra             {$$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de instrucción no válida.", linea: this.
;

SENTENCIATRANSFERENCIA
    : prbreak ptcoma                { $$ = new INSTRUCCION.nuevoBreak(this._$.first_line, this._$.first_column+1) }
    | prreturn EXPRESION ptcoma     { $$ = new INSTRUCCION.nuevoReturn($2, this._$.first_line, this._$.first_column+1) }
    | prcontinue ptcoma             { $$ = new INSTRUCCION.nuevoContinue(this._$.first_line, this._$.first_column+1) }
    | prreturn ptcoma               { $$ = new INSTRUCCION.nuevoReturn(null, this._$.first_line, this._$.first_column+1) }
;

SENTENCIACICLO
    : WHILE         {$$=$1}
    | FOR           {$$=$1}
    | DOWHILE       {$$=$1}
;
```

```
WHILE
    : prwhile pabre EXPRESION pcierra labre INSTRUCCION lcierra     {$$ = new INSTRUCCION.nuevoWhile($3, $6, this._$.first_line,
    | prwhile pabre EXPRESION pcierra labre lcierra                {$$ = new INSTRUCCION.nuevoWhile($3, [], this._$.first_line,
    | prwhile error lcierra                                        {$$ = ""; errores.push({ tipo: "Sintáctico", error: "Declara
;

FOR
    : prfor pabre DEC_VAR EXPRESION ptcoma ACTUALIZACION pcierra labre INSTRUCCION lcierra      {$9.push($6); $$ = new INSTRUCCI
    | prfor pabre DEC_VAR EXPRESION ptcoma ACTUALIZACION pcierra labre lcierra                  {$$ = new INSTRUCCION.nuevoFor($
    | prfor error lcierra                                                                      {$$ = ""; errores.push({ tipo: "
;

ACTUALIZACION
    : id igual EXPRESION {$$ = INSTRUCCION.nuevaAsignacion($1, $3, this._$.first_line,this._$.first_column+1)}
    | id incremento {$$ = INSTRUCCION.nuevaAsignacion($1, { opIzq: { tipo: 'VAL_IDENTIFICADOR', valor: $1, linea: this._$.first_
    | id decremento {$$ = INSTRUCCION.nuevaAsignacion($1, { opIzq: { tipo: 'VAL_IDENTIFICADOR', valor: $1, linea: this._$.first_
;

DOWHILE
    : prdo labre INSTRUCCION lcierra prwhile pabre EXPRESION pcierra ptcoma     {$$ = new INSTRUCCION.nuevoDoWhile($7, $3, this._
    | prdo labre lcierra prwhile pabre EXPRESION pcierra ptcoma                 {$$ = new INSTRUCCION.nuevoDoWhile($7, [], this._
    | prdo error ptcoma                                                         {$$ = ""; errores.push({ tipo: "Sintáctico", err
;

SENTENCIACONTROL
    : CONTROLIF     {$$=$1}
    | SWITCH        {$$=$1}
;

CONTROLIF
    : IF                {$$=$1}
    | IFELSE            {$$=$1}
    | ELSEIF            {$$=$1}
    | prif error lcierra    {$$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de sentencia If no válida.", linea:
;

IF
    : prif pabre EXPRESION pcierra labre INSTRUCCION lcierra    { $$ = new INSTRUCCION.nuevoIf($3, $6, this._$.first_line,this._$
    | prif pabre EXPRESION pcierra labre lcierra                { $$ = new INSTRUCCION.nuevoIf($3, [], this._$.first_line,this._$
;
```

```
ELSEIF
    : prif pabre EXPRESION pcierra labre INSTRUCCION lcierra prelse CONTROLIF       { $$ = new INSTRUCCION.nuevoElseIf(
    | prif pabre EXPRESION pcierra labre lcierra prelse CONTROLIF                   { $$ = new INSTRUCCION.nuevoElseIf(
;

SWITCH
    : prswitch pabre EXPRESION pcierra labre CASESLIST DEFAULT lcierra      { $$ = new INSTRUCCION.nuevoSwitch($3, $6,
    | prswitch pabre EXPRESION pcierra labre CASESLIST lcierra              { $$ = new INSTRUCCION.nuevoSwitch($3, $6,
    | prswitch pabre EXPRESION pcierra labre DEFAULT lcierra                { $$ = new INSTRUCCION.nuevoSwitch($3, null
    | prswitch error lcierra                                                { $$ = ""; errores.push({ tipo: "Sintáctico
;

CASESLIST
    : CASESLIST prcase EXPRESION dospuntos INSTRUCCION      { $1.push(new INSTRUCCION.nuevoCaso($3, $5, this._$.first_l
    | CASESLIST prcase EXPRESION dospuntos                  { $1.push(new INSTRUCCION.nuevoCaso($3, [], this._$.first_l
    | prcase EXPRESION dospuntos INSTRUCCION                { $$ = [new INSTRUCCION.nuevoCaso($2, $4, this._$.first_lin
    | prcase EXPRESION dospuntos                            { $$ = [new INSTRUCCION.nuevoCaso($2, [], this._$.first_lin
    | prcase error dospuntos                                { $$ = ""; errores.push({ tipo: "Sintáctico", error: "Decla
;

DEFAULT
    : prdefault dospuntos INSTRUCCION           { $$ = new INSTRUCCION.nuevoCaso(null, $3, this._$.first_line, this._$.
    | prdefault dospuntos                       { $$ = new INSTRUCCION.nuevoCaso(null, [], this._$.first_line, this._$.
;

DEC_VAR
    : TIPO id igual EXPRESION ptcoma    {$$ = INSTRUCCION.nuevaDeclaracion($2, $4, $1, this._$.first_line,this._$.first
    | TIPO id ptcoma                    {$$ = INSTRUCCION.nuevaDeclaracion($2, null, $1, this._$.first_line,this._$.fir
    | id igual EXPRESION ptcoma         {$$ = INSTRUCCION.nuevaAsignacion($1, $3, this._$.first_line,this._$.first_colu
    | id incremento ptcoma              {$$ = INSTRUCCION.nuevaAsignacion($1, { opIzq: { tipo: 'VAL_IDENTIFICADOR', val
    | id decremento ptcoma              {$$ = INSTRUCCION.nuevaAsignacion($1, { opIzq: { tipo: 'VAL_IDENTIFICADOR', val
    | TIPO error ptcoma                 {$$ = ""; errores.push({ tipo: "Sintáctico", error: "Declaración de variable no
;

DEC_VECT
    : TIPO_VECT id igual prnew TIPO cabre EXPRESION ccierra ptcoma          { $$ = INSTRUCCION.nuevoVector($1, $5, $2,
    | TIPO_VECT id igual labre LISTAVALORES lcierra ptcoma                  { $$ = INSTRUCCION.nuevoVector($1, null, $2
    | id cabre EXPRESION ccierra igual EXPRESION ptcoma                     { $$ = INSTRUCCION.modificacionVector($1, $
    | TIPO_VECT id igual EXPRESION ptcoma                                   { $$ = INSTRUCCION.nuevoVector($1, null, $2
    | TIPO_VECT error ptcoma                                                { $$ = ""; errores.push({ tipo: "Sintáctico
;

DEC_LIST
    : TIPO_LIST id igual prnew prlist menor TIPO mayor ptcoma               { $$ = INSTRUCCION.nuevaLista($1, $7, $2, n
    | id punto pradd pabre EXPRESION pcierra ptcoma                         { $$ = INSTRUCCION.modificacionLista($1, nu
    | id cabre cabre EXPRESION ccierra ccierra igual EXPRESION ptcoma        { $$ = INSTRUCCION.modificacionLista($1, $4
    | TIPO_LIST id igual EXPRESION ptcoma                                   { $$ = INSTRUCCION.nuevaLista($1, null, $2,
    | TIPO_LIST error ptcoma                                                { $$ = ""; errores.push({ tipo: "Sintáctico
;
```

```
TIPO_VECT
    : TIPO cabre ccierra        {$$ = $1}
;

TIPO_LIST
    : prlist menor TIPO mayor   {$$ = $3}
;

TIPO
    : TIPODATO      {$$ = $1}
;

TIPODATO
    : prstring      {$$ = TIPO_DATO.CADENA}
    | printeger     {$$ = TIPO_DATO.ENTERO}
    | prdouble      {$$ = TIPO_DATO.DOBLE}
    | prchar        {$$ = TIPO_DATO.CARACTER}
    | prboolean     {$$ = TIPO_DATO.BOOLEANO}
;
```

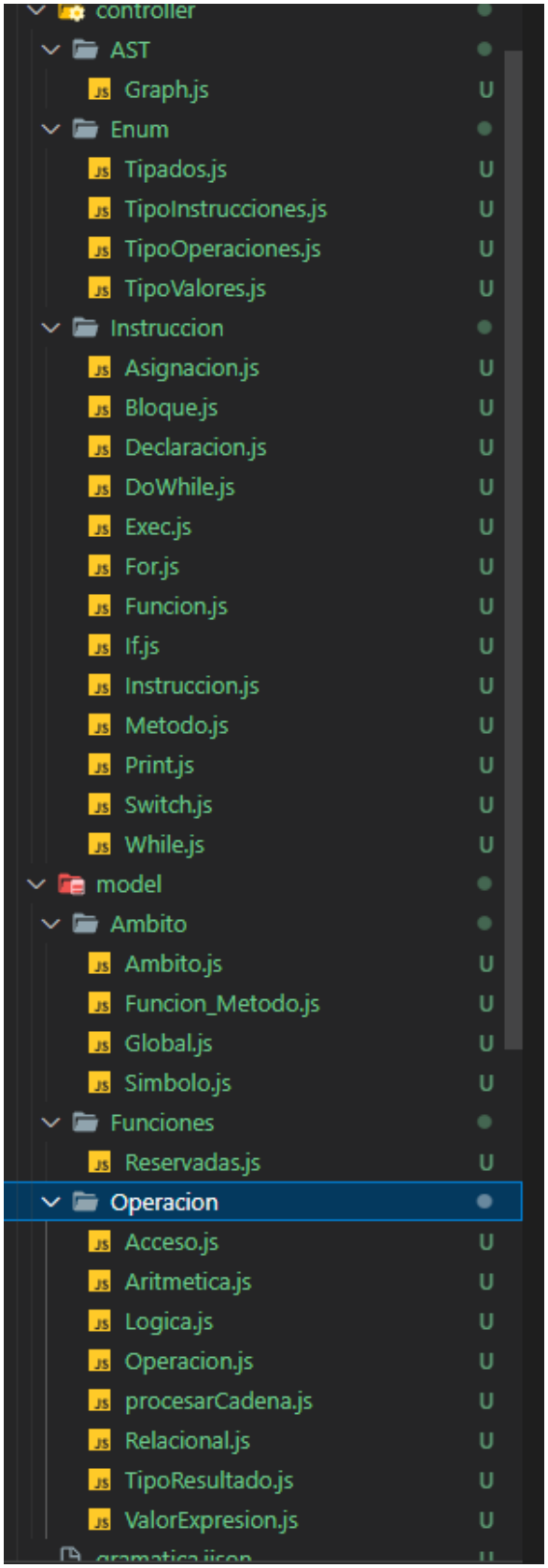
```
EXPRESION
    :   EXPRESION suma EXPRESION                    {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION menos EXPRESION                     {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION multi EXPRESION                     {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION div EXPRESION                       {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION exponente EXPRESION                 {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION modulo EXPRESION                    {$$= INSTRUCCION.nuevaOperacionBinaria($
    | menos EXPRESION %prec umenos                  {$$= INSTRUCCION.nuevaOperacionBinaria($
    | pabre EXPRESION pcierra                       {$$=$2}
    | EXPRESION igualigual EXPRESION                {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION diferente EXPRESION                 {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION menor EXPRESION                     {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION menorigual EXPRESION                {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION mayor EXPRESION                     {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION mayorigual EXPRESION                {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION or EXPRESION                        {$$= INSTRUCCION.nuevaOperacionBinaria($
    | EXPRESION and EXPRESION                       {$$= INSTRUCCION.nuevaOperacionBinaria($
    | not EXPRESION                                 {$$= INSTRUCCION.nuevaOperacionBinaria($
    | cadena                                        {$$ = INSTRUCCION.nuevoValor($1, TIPO_VA
    | caracter                                      {$$ = INSTRUCCION.nuevoValor($1.trim().s
    | true                                          {$$ = INSTRUCCION.nuevoValor($1.trim(),
    | false                                         {$$ = INSTRUCCION.nuevoValor($1.trim(),
    | entero                                        {$$ = INSTRUCCION.nuevoValor(Number($1.t
    | doble                                         {$$ = INSTRUCCION.nuevoValor(Number($1.t
    | id cabre cabre EXPRESION ccierra ccierra      {$$ = INSTRUCCION.accesoLista($1, $4, th
    | id cabre EXPRESION ccierra                    {$$ = INSTRUCCION.accesoVector($1, $3, t
    | id                                            {$$ = INSTRUCCION.nuevoValor($1.trim(),
    | CASTEO                                        {$$=$1}
    | TERNARIO                                      {$$=$1}
    | LLAMADA                                       {$$=$1}
    | FUNCIONESRESERVADAS                           {$$=$1}
;
```

```
CASTEO
    : pabre TIPO pcierra EXPRESION { $$ = new INSTRUCCION.nuevoCasteo($2, $
;

TERNARIO
    : EXPRESION interrogacion EXPRESION dospuntos EXPRESION { $$ = new INST
;

FUNCIONESRESERVADAS
    : FTOLOWER          { $$=$1 }
    | FTOUPPER          { $$=$1 }
    | FLENGTH           { $$=$1 }
    | FTRUNCATE         { $$=$1 }
    | FROUND            { $$=$1 }
    | FTYPEOF           { $$=$1 }
    | FTOSTRING         { $$=$1 }
    | FTOCHARARRAY      { $$=$1 }
;

FPRINT
    : prprint pabre EXPRESION pcierra ptcoma    {$$ = new INSTRUCCION.nuevo
    | prprint pabre pcierra ptcoma              {$$ = new INSTRUCCION.nuevo
    | prprint error ptcoma                      {$$ = ""; errores.push({ ti
;
```

```
FTOLOWER
    : prtoLower pabre EXPRESION pcierra         {$$ = new INSTRUCCION.toLower($3, this._$.first_line,this.
;

FTOUPPER
    : prtoUpper pabre EXPRESION pcierra         {$$ = new INSTRUCCION.toUpper($3, this._$.first_line,this.
;

FLENGTH
    : prlength pabre EXPRESION pcierra          {$$ = new INSTRUCCION.nuevoLength($3, this._$.first_line,t
;

FTRUNCATE
    : prtruncate pabre EXPRESION pcierra        {$$ = new INSTRUCCION.nuevoTruncate($3, this._$.first_line
;

FROUND
    : prround pabre EXPRESION pcierra           {$$ = new INSTRUCCION.nuevoRound($3, this._$.first_line,th
;

FTYPEOF
    : prtypeof pabre EXPRESION pcierra          {$$ = new INSTRUCCION.nuevoTypeOf($3, this._$.first_line,t
;

FTOSTRING
    : prtoString pabre EXPRESION pcierra        {$$ = new INSTRUCCION.nuevoToString($3, this._$.first_line
;

FTOCHARARRAY
    : prtoCharArray pabre EXPRESION pcierra     {$$ = new INSTRUCCION.nuevoToCharArray($3, this._$.first_l
;

LLAMADA
    : id pabre LISTAVALORES pcierra     { $$ = INSTRUCCION.nuevaLlamada($1, $3, this._$.first_line, this._
    | id pabre pcierra                  { $$ = INSTRUCCION.nuevaLlamada($1, [], this._$.first_line, this._
;

LISTAVALORES
    : LISTAVALORES coma VALORES { $1.push($3); $$=$1; }
    | VALORES                   { $$=[$1]; }
;

VALORES
    : EXPRESION { $$=$1 }
;
```

Código Java

Obtendrá los tokens ingresados, formará los árboles y tablas y analizará o mostrará si las expresiones y funciones son aceptadas (para posteriormente correr el código).



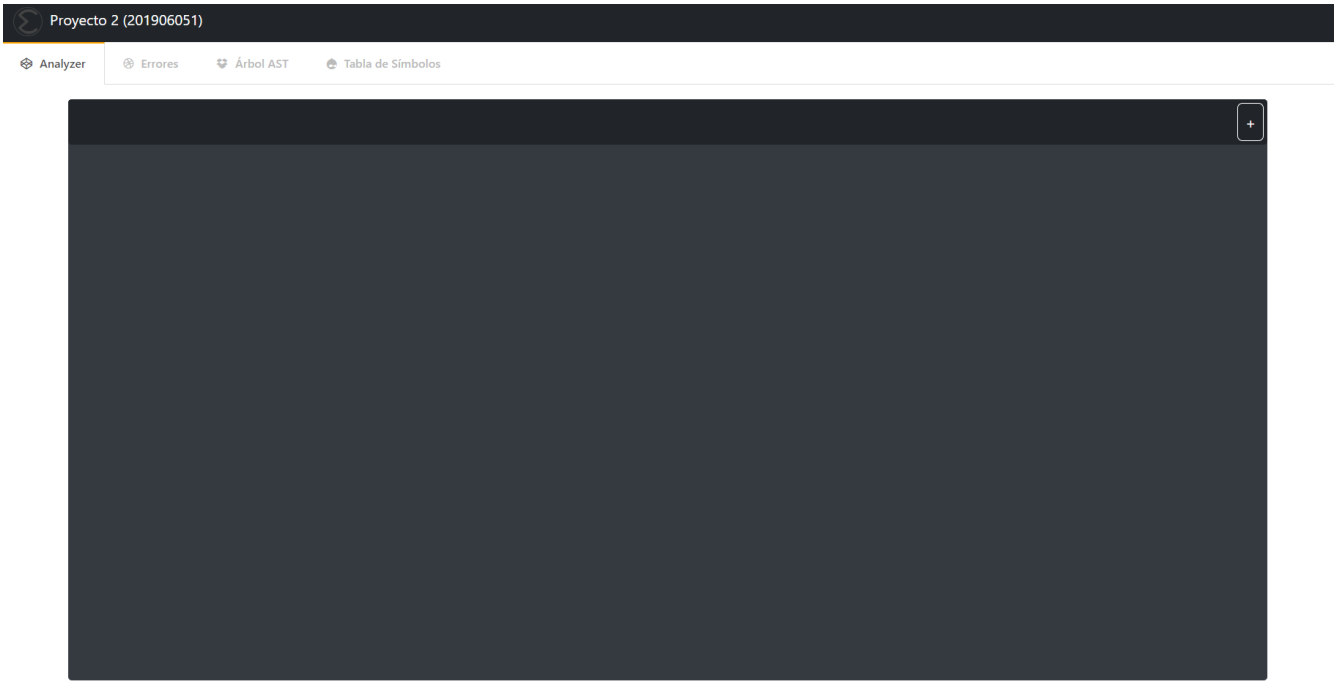
Se exportarán clases las cuales servirán para su posterior uso:

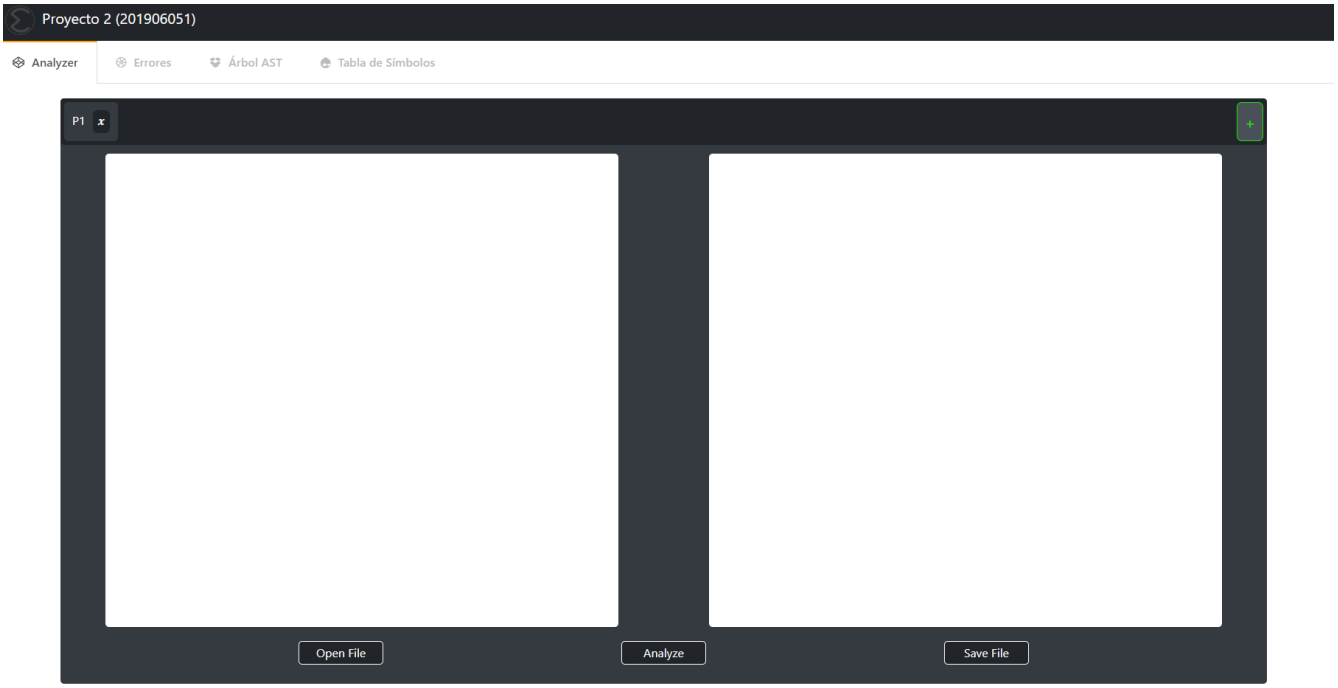
- Las clase en la que se pasan los parámetros (en general) son la TipoOperaciones, TipoValores, Tipados e Instrucciones dentro del archivo jison.
- La clase index.js es la que corre la aplicación.
- Al abrir la carpeta del proyecto usar el comando “npm install” o “npm i” para instalar todas las librerías y módulos que requiere para usar el programa.
- Con “npm start” corre el programa abriendo en el “localhost:3000” corra el proyecto.

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject",
  "jison": "jison ./src/Controllers/gramatica.jison && mv gramatica.js ./src/Controllers/"
},
```

```
PS C:\Users\Pacos\Desktop\Proyectos\React + JScript\proyecto2> npm start

> proyecto2@0.1.0 start
> react-scripts start
```

Fronted del proyecto





Función al Abrir el programa

- Al correr el servidor nos abre en el servidor local el puerto en donde uno va a usar la aplicación.

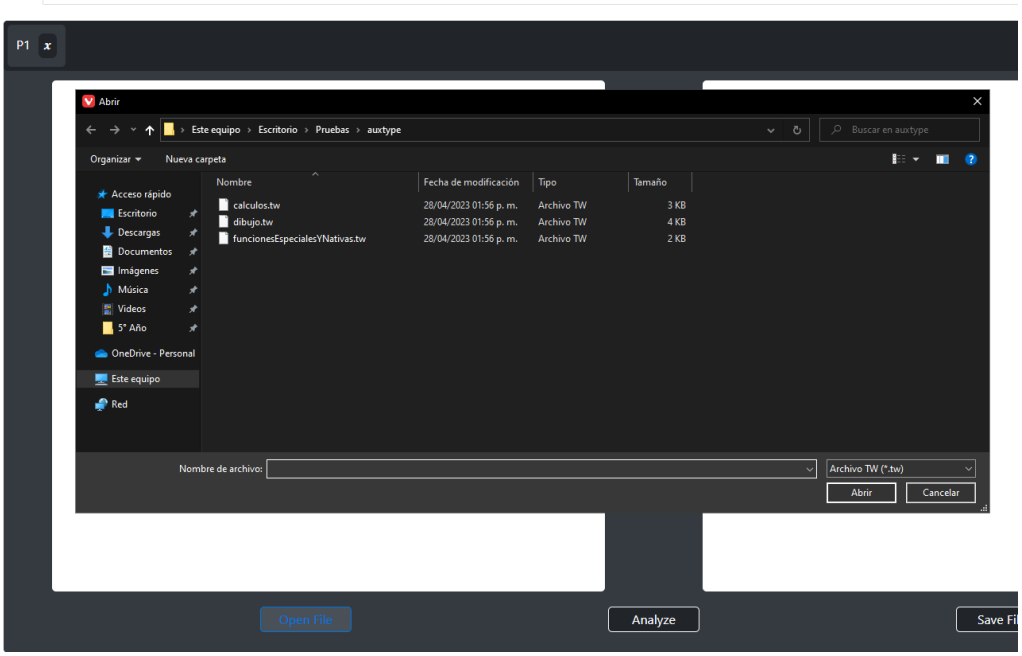
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('sb-nav-fixed'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

```

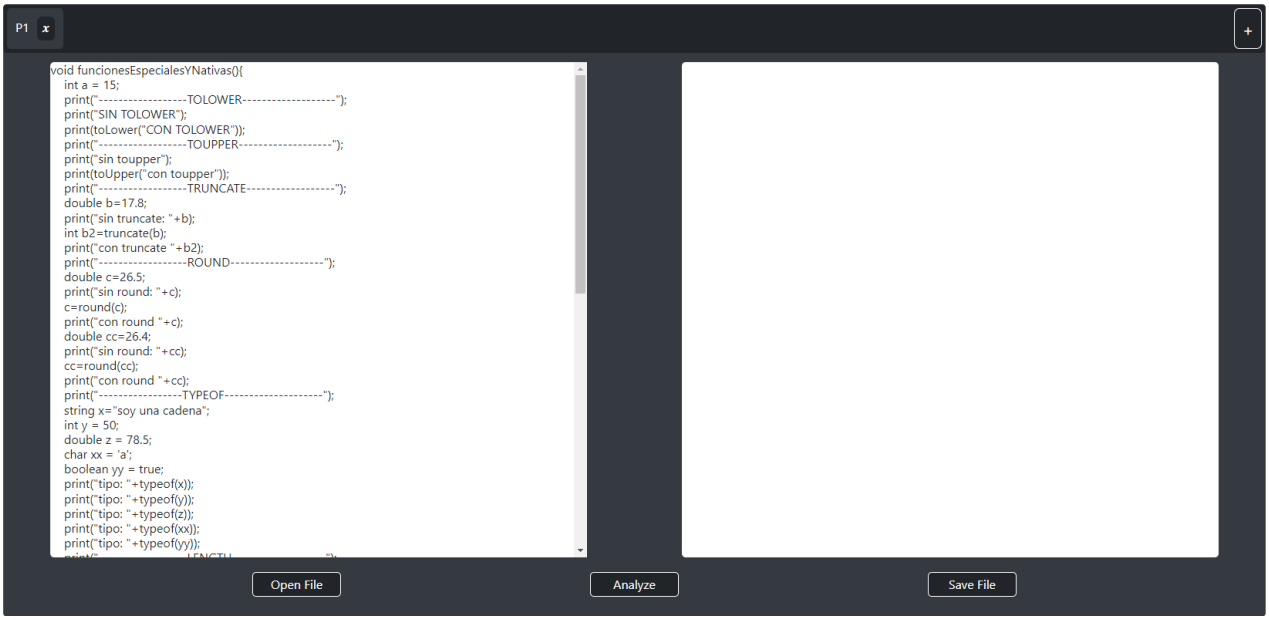
Funciones del Programa

Consta de 3 botones, básicos para el análisis, guardado y abrir archivos.

Open File

- Abre el explorador de medios para que el usuario busque archivos de extensión .tw
- Al seleccionar el archivo, en el primer cuadro de texto muestra el contenido del archivo, este usa una función llamada handleOpenFile el cual se encarga de obtener la información del archivo y enviar todo el contenido del mismo a el cuadro de la pestaña que este abierta (si esta el usuario en la pestaña 1, en esa se mostrará el contenido, en caso de ser otra pestaña, solo esa tendrá el contenido del archivo)





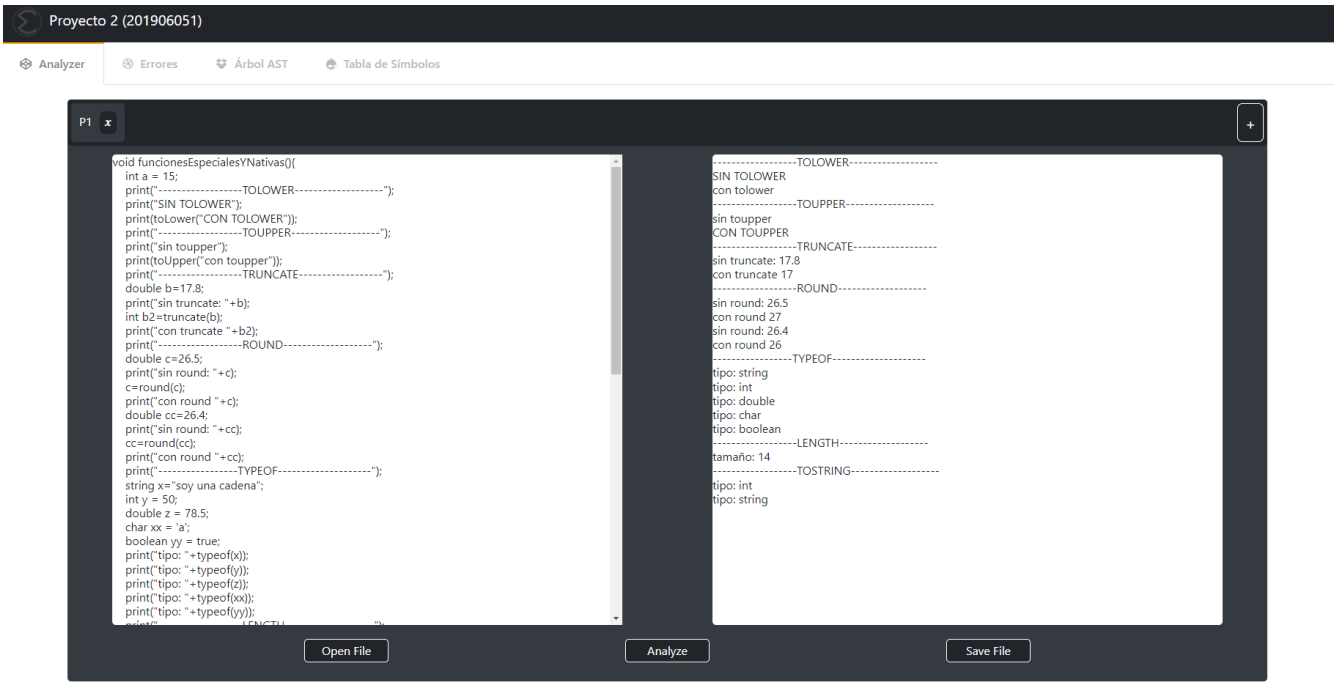
```
//-------------------------------------
const handleOpenFile = async () => {
  const fileInput = document.createElement("input");
  fileInput.type = "file";
  fileInput.accept = ".tw";
  fileInput.addEventListener("change", async (event) => {
    const target = event.target;
    const file = target.files?.[0];
    if (file) {
      const fileText = await file.text();
      const newTabs = [...tabs];
      const tabIndex = newTabs.findIndex((tab) => tab.id === activeTab);
      newTabs[tabIndex] = { ...newTabs[tabIndex], value1: fileText };
      setTabs(newTabs);
    }
  });
  fileInput.click();
};
```

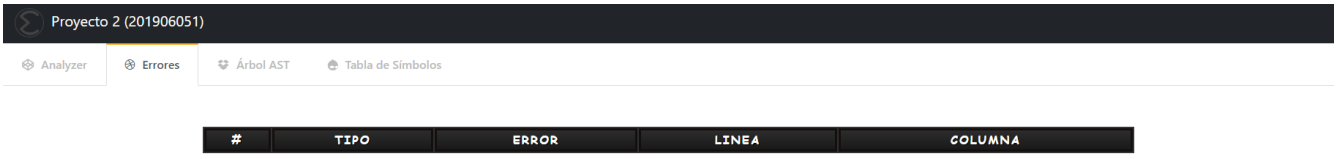
```
//-----------------------------------------
const handleTabClick = (id) => {
  setActiveTab(id);
};

const handleAddTab = () => {
  const newId = tabs.length === 0 ? 1 : tabs[tabs.length - 1].id + 1;
  const newTab = { id: newId, value1: "", value2: "" };
  setTabs([...tabs, newTab]);
  setActiveTab(newId);
};

const handleDeleteTab = (id) => {
  const newTabs = tabs.filter((tab) => tab.id !== id);
  setTabs(newTabs);
  setActiveTab(newTabs.length === 0 ? null : newTabs[0].id);
};
```

Este código hace que cuando el usuario abre una pestaña (que está para abrir nuevas pestañas en la parte superior derecha de la ventana) esta misma tenga el un valor tanto los TextArea 1 y 2, evitando compartir información con los otros de las otras pestañas, pudiendo así el usuario tener abiertas varias pestañas independientes.

Analyze

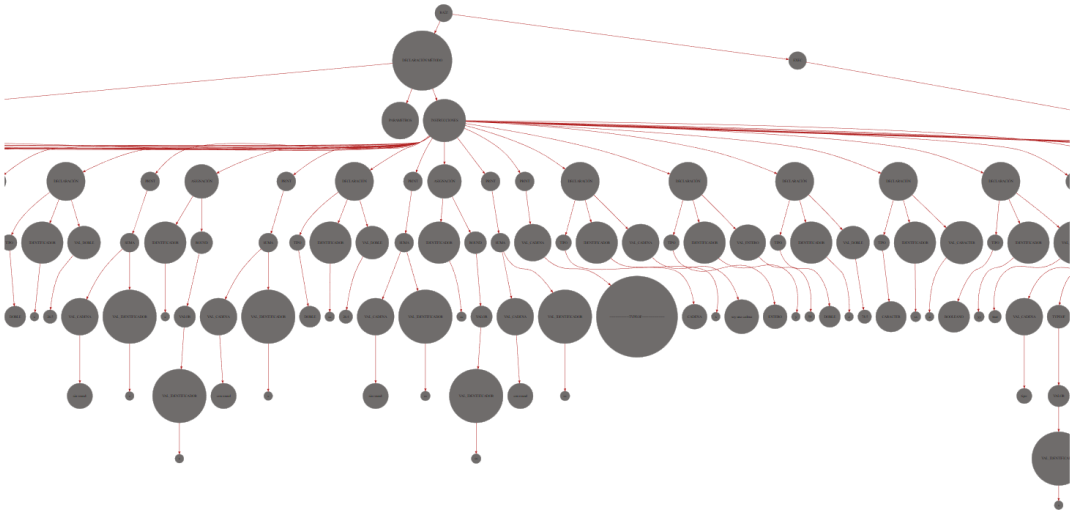
- Es de las más ¿complejas?, se encarga de obtener toda la información del cuadro de texto abierto en la pestaña y envía el contenido de la misma a una función que pasa posterior por 2, para parsear el contenido y para generar el árbol AST, la tabla de errores y la tabla de símbolos, sin mencionar la salida que también es generada por la misma.





Proyecto 2 (201906051)

Analyzer | Errores | Árbol AST | Tabla de Símbolos



Proyecto 2 (201906051)

Analyzer | Errores | Árbol AST | Tabla de Símbolos

#	NOMBRE	SIMBOLO	TIPO DATO	AMBITO	LINEA	COLUMNA
1	funcionesEspecialesYNativas	Método	VOID	global	1	1
2	a	Variable	ENTERO	funcionesEspecialesYNativas	2	5
3	b	Variable	DOBLE	funcionesEspecialesYNativas	10	5
4	b2	Variable	ENTERO	funcionesEspecialesYNativas	12	5
5	c	Variable	DOBLE	funcionesEspecialesYNativas	15	5
6	cc	Variable	DOBLE	funcionesEspecialesYNativas	19	5
7	x	Variable	CADENA	funcionesEspecialesYNativas	24	5
8	y	Variable	ENTERO	funcionesEspecialesYNativas	25	5
9	z	Variable	DOBLE	funcionesEspecialesYNativas	26	5
10	xx	Variable	CARACTER	funcionesEspecialesYNativas	27	5
11	yy	Variable	BOOLEANO	funcionesEspecialesYNativas	28	5
12	cadena	Variable	CADENA	funcionesEspecialesYNativas	35	5
13	numero	Variable	ENTERO	funcionesEspecialesYNativas	38	5

```
const handleCopyValue = (id) => {
  const newTabs = [...tabs];
  const tabIndex = newTabs.findIndex((tab) => tab.id === id);
  const value1 = newTabs[tabIndex].value1;
  //para parsear
  //const result = parser.parse(value1);

  /*
  if (errores.length > 0) {
    console.log('Error para generar el AST');
    return;
  }else{
    console.log(parse);
  */
  /*
    console.log('errores');
    console.log(errores);
    console.log('errores');
    */
  //-----------------
  /**/
```

```
  try {
    var input = value1;
    var ast = parser.parse(input);
    //console.log(respuesta);
    //var parse = ast.parse;
    //var errores = ast.errores;
    if (ast.parse === null) {
      var output = {
        arreglo_simbolos: [],
        arreglo_errores: ast.errores,
        output: "Error, no se ha podido obtener el valor",
      };
      return;
    }
    var parse = ast.parse;
    var errores = ast.errores;
    const global = new Ambito(null, "global");
    var cadena = Global(parse, global);
    var simbolos = global.getArraySimbols();
    for (let i = 0; i < cadena.errores; i++) {
      const err = cadena.errores[i];
      errores.push(err);
    }
    var output = {
        arreglo_simbolos: simbolos,
        arreglo_errores: errores,
        output: cadena.cadena
    }
```

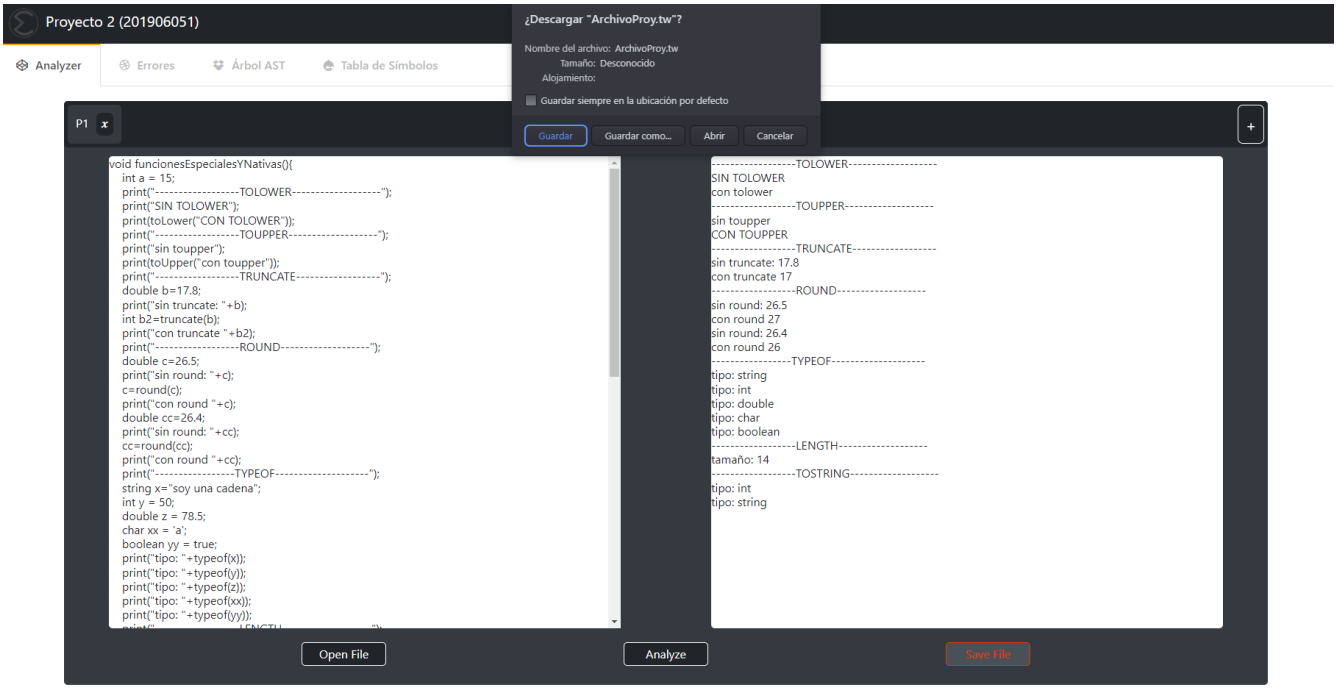
```
    ErrorTable(output);
    SimbolTable(simbolos);
    //console.log(errores);
    newTabs[tabIndex] = { ...newTabs[tabIndex], value2: cadena.cadena };
    setTabs(newTabs);
    var graph = new Graph(parse);
    var grafico_dot = graph.getDot();
    renderGraph(grafico_dot);
  } catch (error) {
    console.log(error);

    //console.log(output);

    newTabs[tabIndex] = { ...newTabs[tabIndex], value2: String(error) };
    setTabs(newTabs);
    renderGraph(dot);
  }

  /*}*/
};
```

```
//Tabla de Simbolos
const SimbolTable = (sim_data) => {
  //console.log(sim_data[0].valor)
  //err_data[n].<tipo_valor>
  var str_data_table = '';
  str_data_table += '<table><thead><tr><th scope="col">#</th><th scope="col">NOMBRE</th><th scope="col">SIMBOLO</th><th scope="col">TIPO DATO</th><th scope=
  let i = 0;
  //console.log(sim_data.length)
  while(i < sim_data.length){
    //str_data_table += '<table><tr><th>ID</th><th>Objeto</th><th>Tipo</th><th>Entorno</th><th>Linea</th><th>Columna</th></tr>';
    //str_data_table += '<tr><td>' + sim_data[i].id + '</td><td>' + sim_data[i].objeto + '</td><td>' + sim_data[i].tipo + '</td><td>' + sim_data[i].entorno
    str_data_table += '<tr><td scope="row">' + (i + 1) +'</td><td>' + sim_data[i].id + '</td><td>' + sim_data[i].objeto + '</td><td>' + sim_data[i].tipo + '
    i++;
  }
  str_data_table += '</tbody></table>';
  setText(str_data_table);
}
//--------------------------------------------
```

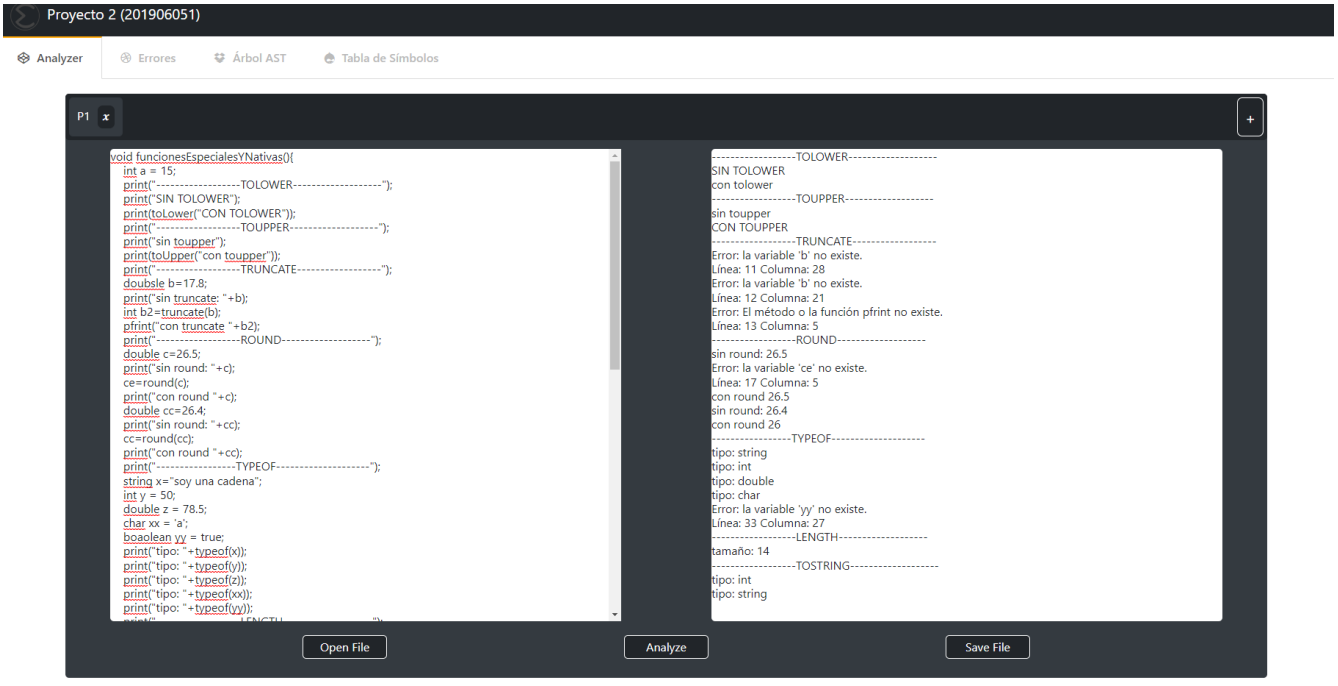
```
//TAblaErrores
const ErrorTable = (err_data) => {
  var str_data_err = '';
  //console.log(err_data.arreglo_errores)
  var err_fs = err_data.arreglo_errores;


  str_data_err += '<table><thead><tr><th scope="col">#</th><th scope="col">TIPO</th><th scope="col">ERROR</th><th scope="col">LINEA</th><th scope="col">COLUM
  let i = 0;
  //console.log(err_fs)
  while(i < err_fs.length){
    str_data_err += '<tr><td scope="row">' + (i + 1) + '</td><td>' + err_fs[i].tipo + '</td><td>' + err_fs[i].error + '</td><td>' + err_fs[i].linea + '</td><
    i++;
  }
  str_data_err += '</tbody></table>';
  setText_err(str_data_err);
}
//--------------------------------------------
```

Save File

- Guarda el archivo como uno nuevo de extensión .tw



En caso de que el archivo tenga errores, lo mostrará en consola y en la tabla de errores.



Proyecto 2 (201906051)

Analyzer | Errores | Árbol AST | Tabla de Símbolos

#	TIPO	ERROR	LINEA	COLUMNA
1	Sintáctico	Declaración de instrucción no válida.	10	19
2	Sintáctico	Declaración de instrucción no válida.	28	23

Proyecto 2 (201906051)

Analyzer | Errores | Árbol AST | Tabla de Símbolos

P1 x +

```
void funciosnesEspecialesYNativas(){
    int a = 15;
    print("------------------TOLOWER--------------------");
    print("SIN TOLOWER");
    print(toLower("CON TOLOWER"));
    print("------------------TOUPPER--------------------");
    print("sin toupper");
    print(toUpper("con toupper"));
    print("------------------TRUNCATE--------------------");
    doubsle b=17.8;
    print("sin truncate: "+b);
    int b2=truncate(b);
    pfrint("con truncate "+b2);
    print("------------------ROUND--------------------");
    double c=26.5;
    print("sin round: "+c);
    ce=round(c);
    print("con round "+c);
    double cc=26.4;
    print("sin round: "+cc);
    cc=round(cc);
    print("con round "+cc);
    print("------------------TYPEOF--------------------");
    string x="soy una cadena";
    int y = 50;
    double z = 78.5;
    char xx = 'a';
    boaolean yy = true;
    print("tipo: "+typeof(x));
    print("tipo: "+typeof(y));
    print("tipo: "+typeof(z));
    print("tipo: "+typeof(xx));
    print("tipo: "+typeof(yy));
```

```
Error: El método o la función funcionesEspecialesYNativas no existe.
Línea: 43 Columna: 1
```

Open File | Analyze | Save File

Conclusiones

- La utilización de expresiones regulares ayuda en la simplificación y obtención de mejor forma de expresiones que uno quiere obtener.
- El análisis sintáctico sirve para obtener reglas del funcionamiento del léxico que uno creó, para el análisis de un archivo.