INSTITUTO TECNOLÓGICO DE MÉRIDA

DEPARTAMENTO DE SISTEMAS COMPUTACIONALES

AUTOMATAS II

7SA

INTEGRANTES:

CANCHE CEN JUAN DE DIOS

MANZANERO MARTIN SHEILA SUGEY

NOMBRE DEL PROYECTO:

ANALIZADOR SEMÁNTICO

MAESTRA:

Dra. María Italia Jiménez Ochoa

Mérida, Yucatán, México a 2 de noviembre de 2017

Índice de contenido

INTRODUCCION

La fase del análisis léxico consiste en un programa que recibe caracteres de entrada y elaborar como salida una secuencia de componentes léxicos (tokens o símbolos), que después son utilizados por el analizador sintáctico para realizar el análisis.

La principal tarea en la fase del análisis sintáctico es comprobar que el orden en que el analizador léxico le va entregando la cadena de tokens es válido, para después comprobar si la cadena puede ser generada por la gramática del programa fuente.

La fase de análisis semántico es la fase posterior a la de análisis sintáctico y la última dentro del proceso de síntesis de un lenguaje de programación. La sintaxis de un lenguaje de programación es el conjunto de reglas formales que especifican la estructura de los programas pertenecientes a dicho lenguaje. La semántica de un lenguaje de programación es el conjunto de reglas que especifican el significado de cualquier sentencia sintácticamente válida. Finalmente, el análisis semántico de un procesador de lenguaje es la fase encargada de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico.

DESCRIPCION GENERAL

En este proyecto se aborda el problema del cálculo de información que no puede ser descrita por las gramáticas libres de contexto, por lo tanto es necesario realizar la comprobación en una fase posterior al análisis sintáctico, el cual tiene por nombre análisis semántico. La información que se calcula en esta fase está relacionada con el significado (la semántica) del programa y no con su estructura (la sintaxis).

En esta fase se asocia información a las construcciones del lenguaje de programación proporcionando atributos a los símbolos de la gramática (como el valor de una expresión, el tipo de una variable, su ámbito, el número de argumentos de una función, etc.).

En este proyecto el análisis semántico incluye la construcción de la tabla de símbolos para llevar un seguimiento del significado de los identificadores en el programa (variables, lexemas, tipos, parámetros, número de fila, etc.).

El análisis semántico realiza la comprobación e inferencia de tipos en expresiones y sentencias, por ejemplo:

Que ambos lados de una asignación tengan tipos adecuados, que no se declaren variables con el mismo nombre, que los parámetros de llamada a una función tengan tipos adecuados, número de parámetros correcto, etc. En caso que ocurra lo contario se manda a una tabla de errores y se arroja el error semántico en pantalla.

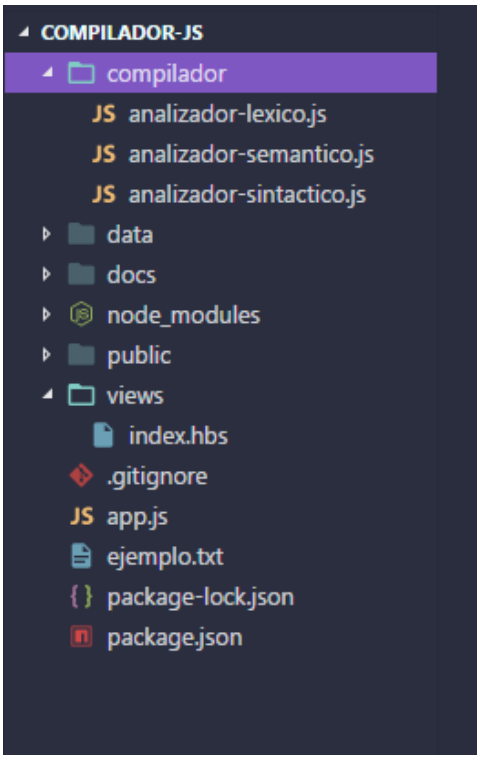
RESTRICCIONES DEL PROYECTO

1. En primera instancia el analizador semántico debe inicializarse, si no al momento de ser ejecutado marcará error.
2. El analizador semántico solo acepta identificadores que empiecen con los símbolos $ y _, o que comience con letras mayúsculas y minúsculas.
3. No puede tener identificadores que comiencen con números.
4. Solo acepta palabras reservadas como INT y FLOAT
5. Si tanto el analizador léxico o el sintáctico son incorrectos, el analizador semántico no muestra resultados.

DESCRIPCIÓN DEL CÓDIGO

El código está segmentado en 3 partes:

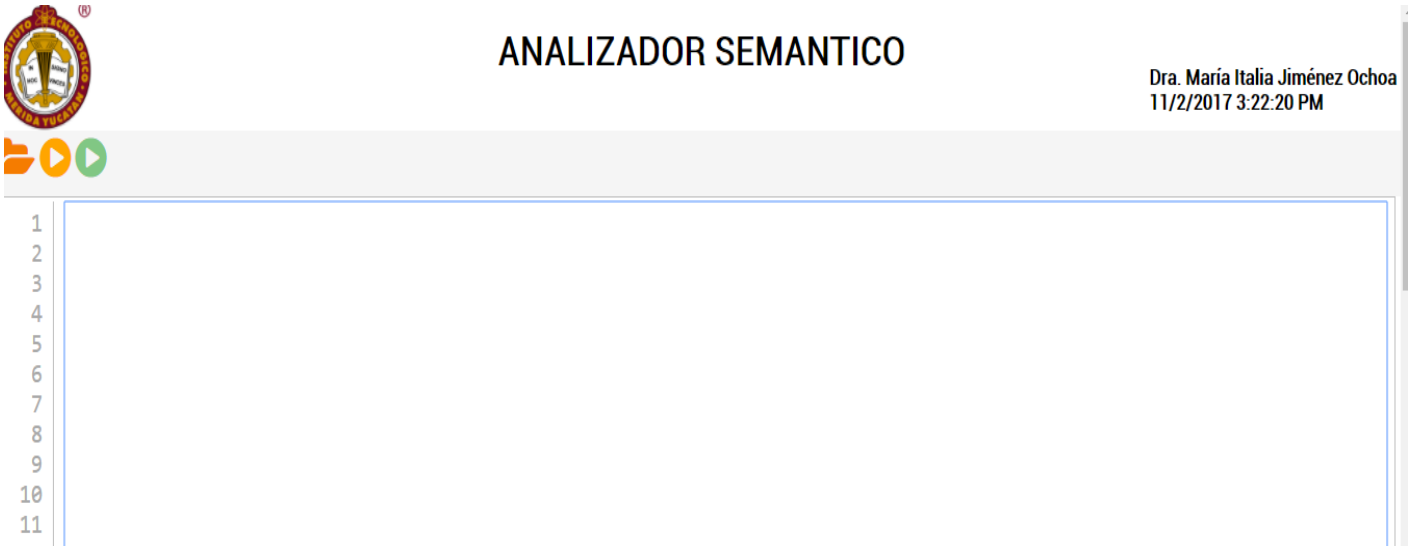
Analizador-lexico.js, analizador-sintatico.js y analizador-semantico.js:



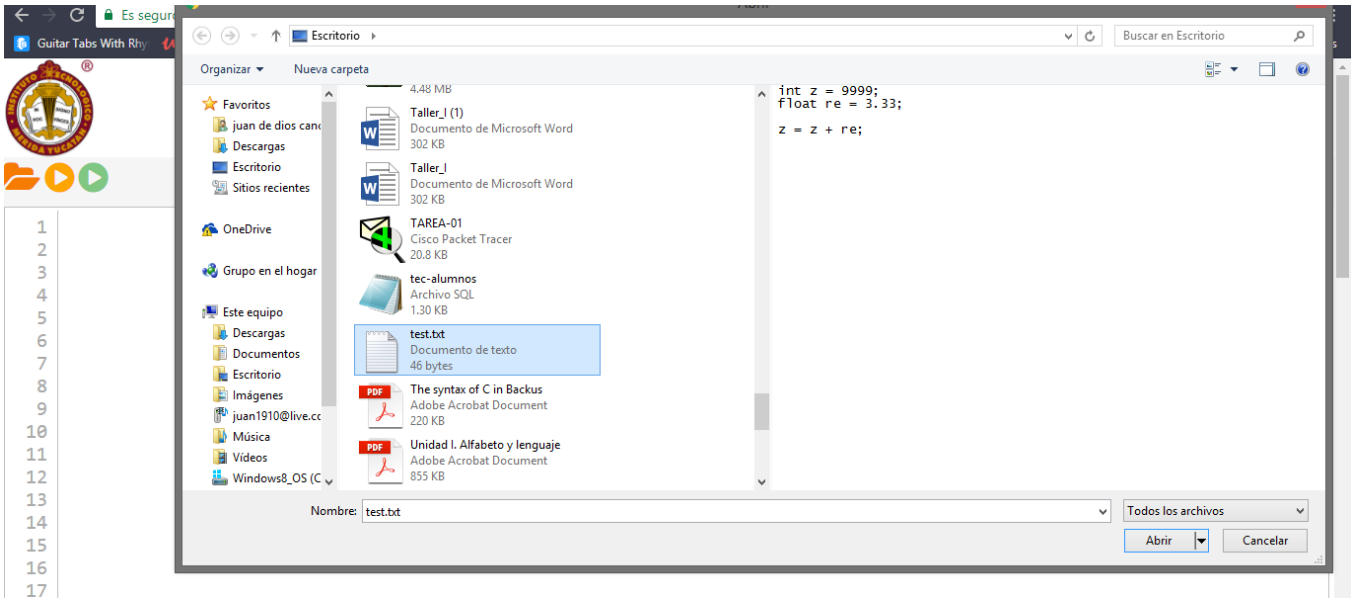
En la carpeta “compilador” se encuentra todo el código pertinente a los analizadores. En la carpeta “views” se encuentra la página web del programa, en el archivo app.js contiene el código para que el programa funcione como un servidor, la carpeta “public” para los archivos estáticos que permite el funcionamiento de la página web y la carpeta “node_modules” son las librerías que permiten el uso de la aplicación web.

En adición, se subió el sitio web a la página https://compiladorjsitm.herokuapp.com/

La página está diseñada con esta interfaz:



En la interfaz se puede observar un ícono de una carpeta, está sirve para cargar un archivo de texto txt, luego el botón naranja de play es para ejecutar el archivo recién cargado:



En la imagen de abajo se puede observar el archivo ya cargado.

```
int z = 9999;
float re = 3.33;

z = z + re;
```

Si se desea sólo escribir líneas de código, se utiliza el botón verde para ejecutar las líneas de código que se escribieron.

Debajo se encuentran todas las tablas requeridas, como la tabla de léxico, sintáctico y semántico.

ANALIZADOR LÉXICO

El analizador léxico ésta dividido entre varios métodos:

El primer método llamado ***findErrores*** busca algún error léxico de los valores de entrada, además de llenar la tabla de errores léxicos si encuentra alguno.

El segundo método ***llenarTablaLexico,*** llena la tabla con los símbolos correspondientes que se encontraron y la fila en la que se encontró.

El tercer método ***analizarTokens,*** analiza todos los valores de entrada por fila e invoca los otros métodos previamente mencionados para su debida ejecución:

```
let analizarTokens = ( entrada ) => {

    entrada.forEach(function(e, i)
    {
        findErrores(e,i);
        llenarTablaLexico(e.match(componentesLexicos),i);
        analizadorSintactico(match_replace, erroresLexicos, i);
    });
```

Gracias al método ***llenarTablaLexico***, se envía como parámetro un arreglo con los identificadores que se analizaron correctamente al método ***analizadorSintactico***, el cual se encarga de verificar la consistencia de la expresión.

ANALIZADOR SINTÁCTICO

Al igual que el léxico, está dividido en varios métodos:

```
let errores_sintacticos = [];
let exito_sintacticos = [];

const tabla_sintactica =
    [       /*  =     |-      |+      |*      |/      |id     |cte    |(      |)  |wr     |$ */
      /*Z*/["",      "",     "",     "",     "",     "S",    "",     "",     "", "RE",    ""  ],
      /*E*/["",      "",     "",     "",     "",     "Ae",   "Ae",   "Ae",   "", "",      ""  ],
      /*e*/["=Ae",   "",     "",     "",     "",     "",     "",     "",     "ε", "",     "ε" ],
      /*A*/["",      "",     "",     "",     "",     "Ba",   "Ba",   "Ba",   "", "",      ""  ],
      /*a*/["ε",     "-Ba",  "",     "",     "",     "",     "",     "",     "ε", "",     "ε" ],
      /*B*/["",      "",     "",     "",     "",     "Cb",   "Cb",   "Cb",   "", "",      ""  ],
      /*b*/["ε",     "ε",    "+Cb",  "",     "",     "",     "",     "",     "ε", "",     "ε" ],
      /*C*/["",      "",     "",     "",     "",     "Dc",   "Dc",   "Dc",   "", "",      ""  ],
      /*c*/["ε",     "ε",    "ε",    "*Dc",  "",     "",     "",     "",     "ε", "",     "ε" ],
      /*D*/["",      "",     "",     "",     "",     "Fd",   "Fd",   "Fd",   "", "",      ""  ],
      /*d*/["ε",     "ε",    "ε",    "ε",    "/Fd",  "",     "",     "",     "ε", "",     "ε" ],
      /*F*/["",      "",     "",     "",     "",     "id",   "cte",  "(E)",  "", "",      ""  ],
      /*S*/["",      "",     "",     "",     "",     "ide",  "",     "",     "", "",      ""  ],
      /*R*/["",      "",     "",     "",     "",     "",     "",     "",     "", "wr",    ""  ],
    ];

    let analizadorSintactico = ( lex, errores,fila ) =>
    {···
    };      Juan de Dios Canche Cen, 4 days ago • mensajes de error sintactico terminados, falta

    let cleanTablaSintac = () => {···
    }
```

Se utiliza una tabla sintáctica para determinar si la sintaxis de los lexemas es la correcta línea por línea.

El método ***analizadorSintactico*** se encarga de verificar si el camino de la expresión que es dada por el analizador léxico es correcto. Si existe algún error en el léxico el analizador sintáctico no se ejecuta hasta que el léxico sea correcto. El resultado se guarda en el arreglo ***exito_sintacticos*** y si se encontró un error en el sintáctico se guarda en el arreglo ***errores_sintacticos***.

ANALIZADOR SEMÁNTICO

El analizador semántico está dividido en varias fases, dependiendo de los errores que pueden ocurrir y la asignación del valor de los identificadores. Sólo funciona esta fase si la fase del sintáctico es correcta.

```
let errores_semanticos = [];
let exito_semantico = [];

let analizadorSemantico = ( array_lex, err_sintac  ) =>
{
    if (err_sintac.length > 0 || (array_lex[0].identificador == "IDENTIFICADOR" && array_lex[1].identificador ==
    "PUNTOYCOMA") || array_lex[0].identificador !== "PALABRA_RESERVADA")
    {
        console.log("Repare la sintaxis por favor");
        console.log(err_sintac);

    }else
    {

        for( var x = 0; x < array_lex.length; x++ )
        {
            if( array_lex[x].identificador == "CONSTANTE" ){
                let dato = parseFloat(array_lex[x].token);
                if( dato % 1 !== 0 ){
                    array_lex[x].tipo = "float";
                }else{
                    array_lex[x].tipo = "int";
                }
            }
        }
```

La primera condición limita la forma en que la expresión debe estar formada. Si por alguna razón empieza con un identificador y le sigue una coma, marca un error en el sintáctico y si la primera oración no empieza con palabra reservada, de igual manera marca error, es por ello que siempre se debe inicializar la variable de esta manera:

```
1  int x = 0;
2
```

En el primer ciclo for se le da el valor a la constante dependiendo del token que identificó (en el analizador léxico devuelve un objeto por cada lexema, por lo que el valor del identificador recae en una condición que verifica si es float o entero).

```
for( var x = 0; x < array_lex.length; x++ )
{
    if( array_lex[x].identificador == "CONSTANTE" ){
        let dato = parseFloat(array_lex[x].token);
        if( dato % 1 !== 0 ){
            array_lex[x].tipo = "float";
        }else{
            array_lex[x].tipo = "int";
        }
    }
}
```

El siguiente ciclo for determina si por alguna razón la sintaxis ***PALABRA_RESERVADA IDENTIFICADOR*** está constituida por ***PALABRA_RESERVADA CONSTANTE,*** marque un error:

```
for (var x = 0; x < array_lex.length; x++)      Juan de Dios Canche Cen, 18 hours ago • terminado backend se
{
    if (array_lex[x].identificador == "PALABRA_RESERVADA" && array_lex[x+1].identificador == "CONSTANTE")
    {
        errores_semanticos.push({
            codigo_err: "ID_ERROR",
            msg: "EL IDENTIFICADOR NO PUEDE SER CONSTANTE",
            fila_err: array_lex[x + 1].fila,
            lexema: array_lex[x + 1].token,
            tipo: array_lex[x + 1].tipo
        });
    }
}
```

Lo siguiente es verificar si existe una fila con la sintaxis ***PALABRA_RESERVADA IDENTIFICADOR IGUALACION CONSTANTE PUNTOYCOMA***, se asigne el tipo de dato al identificador, de lo contrario marque un mensaje de error:

```
for (var x = 0; x < array_lex.length; x++)
{
    if (array_lex[x].identificador == "PALABRA_RESERVADA")
    {
        if (array_lex[x+1].identificador == "IDENTIFICADOR")
        {
            if (array_lex[x + 2].identificador == "IGUALACION" && array_lex[x + 3].identificador == "CONSTANTE" && array_lex[x + 4].identificador == "PUNTOYCOMA")
            {
                if (array_lex[x].token !== array_lex[x + 3].tipo)
                {
                    //console.log("Los tipos de variable son incompatibles " + array_lex[x + 1].token + " " + array_lex[x + 1].tipo);
                    errores_semanticos.push({
                        codigo_err: "DATA_TYPE_ERROR",
                        msg: "LOS TIPOS DE DATOS SON INCOMPATIBLES",
                        fila_err: array_lex[x + 1].fila,
                        lexema: array_lex[x + 1].token,
                        tipo: array_lex[x + 1].tipo
                    });
                }else{
                    array_lex[x + 1].tipo = array_lex[x].token;
                    //console.log("Esta inicializado: " + array_lex[x + 1].token + " " + array_lex[x + 1].tipo);
                    exito_semantico.push({
                        codigo_exito: "INIT->SUCCESSFULL",
                        msg: "LA VARIABLE SE INICIALIZO SATISFACTORIAMENTE",
                        filla_exito: array_lex[x + 1].fila,
                        lexema: array_lex[x + 1].token,
                        tipo: array_lex[x + 1].tipo
                    });
                }
            }
        }
    }
}
```

```
1  int x = 0;
2
```

Después debe verificar si en alguna expresión el identificador ya ha sido inicializado como el caso de:

```
1  int x = 0;
2  x = 23;
3
```

Mediante el método ***buscarSimbolo***, el cual tiene de argumentos el identificador que no tiene un tipo de dato establecido y la tabla de símbolos, busca si existe ese lexema en la tabla de símbolos, si es así le otorga el valor a la variable que está inicializada. Si no está en la tabla de símbolos, significa que no estaba inicializada por lo que marca el error correspondiente:

```
for (var x = 0; x < array_lex.length; x++)
{
    if (array_lex[x].identificador == "IDENTIFICADOR") {
        if (array_lex[x].tipo.length == 0) {
            if(buscarSimbolo(array_lex[x], array_lex))
            {
                //console.log("se cambió el valor:" + array_lex[x
            }else
            {
                //console.log("no esta inicializado!!!: " + array
                errores_semanticos.push({
                    codigo_err: "INIT->ERROR",
                    msg: "LA VARIABLE NO ESTA INICIALIZADA",
                    fila_err: array_lex[x].fila,
                    lexema: array_le (local var) x: number
                    tipo: array_lex[x].tipo
                });
            }
        }
    }
}
```

```
let buscarSimbolo = ( simbolo, lex ) =>
{
    var si_existe = false;
    for (var s = 0; s < lex.length; s++  ){
        if( lex[s].token ==  simbolo.token && lex[s].tipo.length > 0){
            simbolo.tipo = lex[s].tipo;
            si_existe = true;
        }
    }

    return si_existe;
};
```

Lo siguiente es verificar que cuando se tenga como sintaxis una fila ***IDENTIFICADOR IGUALACION CONSTANTE PUNTOYCOMA*** el valor que tenga el identificador sea el mismo que al que se le está igualando. Es decir:

```
1  int x = 333;
2  x = 23;
3  /* que int concuerde con la constante */
4
```

Si no concuerda el valor marca un error correspondiente:

```
for (var x = 0; x < array_lex.length; x++)
{

    if (array_lex[x].identificador == "IDENTIFICADOR" && array_lex[x + 1].identificador == "IGUALACION" && array_lex[x + 2]
    .identificador == "CONSTANTE" && array_lex[x + 3].identificador == "PUNTOYCOMA" && array_lex[x - 1].identificador !=
    "PALABRA_RESERVADA")
    {
        if (array_lex[x].tipo !== array_lex[x + 2].tipo ){
            array_lex[x].tipo = "ERROR_TYPE";
            errores_semanticos.push({
                codigo_err: "CHANGE_VALUE->ERROR",
                msg: "EL CAMBIO DE VALOR DE LA VARIABLE ES INCOMPATIBLE",
                fila_err: array_lex[x].fila,
                lexema: array_lex[x].token,
                tipo: array_lex[x].tipo
            });
        }
    }
}
```

Por último, se tiene que si existe una expresión de la forma ***IDENTIFICADOR IGUALACION*** se verifique el lexema **IDENTIFICADOR** con los demás CONSTANTES e **IDENTIFICADORES** que le siguen mediante el método ***realizarOperacion:***

```
let realizarOperacion = ( simbolo, fila, lex ) =>
{
    var contErrTipo = 0;
    for (var s = 0; s < lex.length; s++)
    {
        if ((lex[s].identificador == "IDENTIFICADOR" || lex[s].identificador == "CONSTANTE") && lex[s].fila == fila && lex[s].tipo ==
        simbolo.tipo)
        {
            //console.log(lex[s].token);
        } else if ((lex[s].identificador == "IDENTIFICADOR" || lex[s].identificador == "CONSTANTE" ) && lex[s].fila == fila && lex[s].tipo
        !== simbolo.tipo )
        {
            errores_semanticos.push({
                codigo_err: "OP_TYPE->ERROR",
                msg: "EL TIPO DE DATO ES INCOMPATIBLE CON LA OPERACION",
                fila_err: lex[s].fila,
                lexema: lex[s].token,
                tipo: lex[s].tipo
            });
            contErrTipo++;
        }
    }

    return contErrTipo;

}
```

Si los ***IDENTIFICADORES*** o ***CONSTANTES*** no concuerdan con el tipo de dato del ***IDENTIFICADOR***, simplemente el método aumenta el contador de errores y retorna el contador, si es mayor a 0 significa que tiene algún error si no despliega un mensaje de exito:

```
for (var x = 0; x < array_lex.length; x++)
{
    if (array_lex[x].identificador == "IDENTIFICADOR" && array_lex[x - 1].identificador != "PALABRA_RESERVADA") {
        if (array_lex[x + 1].identificador == "IGUALACION") {
            var contErr = realizarOperacion(array_lex[x], array_lex[x].fila, array_lex);
            if( contErr == 0 )
            {
                exito_semantico.push({
                    codigo_exito: "OP_TYPE->SUCCESSFULL",
                    msg: "LAS VARIABLES TIENEN EL MISMO TIPO QUE EL OPERANDO",
                    filla_exito: array_lex[x].fila,
                    lexema: array_lex[x].token,
                    tipo: array_lex[x].tipo
                });
            }
        }
    }
}
```

Para finalizar se devuelve al script ***app.js*** los siguientes valores:

```
array_lex = array_lex.filter((e) => {
    return e.identificador != "OPERADOR" && e.identificador != "PUNTOYCOMA" && e.identificador != "IGUALACION";
});

return {
    errores_semanticos,
    exito_semantico,
    array_lex
}
```

Los cuales son: la tabla de errores semánticos, los éxitos semánticos y la tabla de símbolos.

PROGRAMA PRINCIPAL:

APP.JS

En este método es donde se concreta todo el proceso de los analizadores:

1. Se obtiene los valores por medio de una petición http tipo POST hacia el programa app.js, después se divide por filas los valores de entrada y son enviados hacia el método *analizarTokens* el cuál es el encargado del análisis léxico, retorna tanto un array de los lexemas identificados, una tabla de errores y la tabla de símbolos sin tipo de dato.
2. Estos valores se envían al método *analizadorSintactico* el cual se encarga de analizar la sintaxis de la expresión y dar el visto bueno para su siguiente fase.
3. Por último, el método *analizadorSintactico* devuelve tanto una tabla de éxitos sintácticos como una de errores, los cuales el método *analizadorSemantico* los utiliza de argumentos para realizar su operación.
4. El resultado de estos tres procesos son enviados hacia unas rutas, específicamente https://compiladorjsitm.herokuapp.com/analizar y https://compiladorjsitm.herokuapp.com/upload el cual es recogida por la página web y los resultados mostrados en ella.

5. El código y cómo usarlo están alojados en: https://github.com/Kami-Juan/compilador-js