TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA

DEPARTAMENTO DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

ACTIVIDAD 4

LENGUAJES Y AUTOMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ

EQUIPO 1. INTEGRANTES:

HERNANDEZ MARTINEZ ALINA ADRIANA

AVILA CARDENAS RICARDO

CID SOTO PABLO

RICO GONZALEZ EDUARDO

CELAYA, GTO A 8 DE JUNIO DEL 2020

POR MEDIO DE LA PRESENTE MANIFESTAMOS QUE:

LOS INTEGRANTES DEL **EQUIPO NÚMERO 1**, ELABORAMOS LAS SIGUIENTES EVIDENCIAS ACADÉMICAS PARA CUMPLIR CON EL TEMARIO PROPUESTO EN LA **MATERIA LENGUAJES Y AUTOMATAS II** DE LA ESPECIALIDAD DE INGENIERIA EN SISTEMAS COMPUTACIONALES QUE CORRESPONDE A LA **EVALUACIÓN 4** DEL CICLO ESCOLAR ENERO-JUNIO 2020, EN EL TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA.

HERNANDEZ MARTINEZ ALINA ADRIANA

CID SOTO PABLO EMILIO

RICO GÓMEZ EDUARDO DANIEL

AVILA CARDENAS RICARDO

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: SOLICITUD DE ACTIVIDADES

Celaya, Guanajuato, 01/Junio/2020

LENGUAJES Y AUTÓMATAS II
DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ
SEMESTRE ENERO-JUNIO 2020

ACTIVIDAD 4 (VALOR 100 PUNTOS)

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTE ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN LOS DOCUMENTOS DE LA GUÍA TUTORIAL, Y LA RÚBRICA DE EVALUACIÓN,

EL LECTOR DEBE TOMAR MUY EN CUENTA QUE ESTA ACTIVIDAD ES UN EXAMEN, Y NO DE UNA TAREA SENCILLA, PUES DEMANDA TIEMPO PARA INVESTIGAR, LEER, ANALIZAR, REDACTAR, ILUSTRAR Y PROPONER DE MANERA PROFESIONAL LOS TEMAS PROPUESTOS EN LA ESTRUCTURA TEMÁTICA DE ESTA MATERIA.

1. PROYECTO FINAL.

   CONSIDERANDO QUE LA SEGUNDA ETAPA DEL ANALIZADOR SINTÁCTICO SE HAYA CONCLUIDO SATISFACTORIAMENTE Y CON ELLO EL EQUIPO ACREDITÓ LA EVALUACIÓN ANTERIOR, AHORA EL SIGUIENTE PASO SERÁ QUE EN EQUIPO INVESTIGUEN, DISEÑEN E IMPLEMENTEN LA TERCERA FASE O ETAPA DEL PROYECTO, ES DECIR UN ANALIZADOR SEMÁNTICO.

   MUY IMPORTANTE:

   SI LA EVALUACIÓN ANTERIOR NO FUE ACREDITADA, ESO SIGNIFICA QUE DEBERÁ PRIMERO IMPLEMENTAR DESDE EL INICIO TODOS LOS PLANTEAMIENTOS NECESARIOS PARA TENER UN ANÁLISIS LÉXICO, Y SINTÁCTICO LIBRE DE ERRORES. DE LO CONTRARIO, SI PERSISTEN LAS FALLAS DETECTADAS EN LA EVALUACIÓN ANTERIOR, Y NO SE CORRIGEN, ESTA ETAPA TAMBIÉN PRESENTARÁ FALLOS.

2. LA ACTIVIDAD COMO EQUIPO DEBERÁ COMENZAR CON LA INVESTIGACIÓN Y FUNDAMENTACIÓN DE LOS SIGUIENTES TEMAS.

   A. INVESTIGAR. ¿ QUÉ ES UN ANÁLISIS SEMÁNTICO APLICADO A LA VALORACIÓN DE UN LENGUAJE ?
   B. INVESTIGAR. ¿ EN QUÉ CONSISTE UN ANÁLISIS SEMÁNTICO Y QUÉ LO CARACTERIZA?
   C. IDENTIFICAR. ¿ QUÉ CASOS DE ESTUDIOS SON LOS IMPORTANTES A CONSIDERAR EN EL ANÁLISIS SEMÁNTICO ?
   D. INVESTIGAR Y PROPONER. ¿ QUÉ PROCESOS Y PROBLEMAS ATIENDE UN ANÁLISIS SEMÁNTICO ?
   E. INVESTIGAR. ¿ CÓMO IMPLEMENTAR UN ANÁLISIS SEMÁNTICO ?

   NOTA : ESTOS TEMAS DEBEN SER EL RESULTADO DE UNA INVESTIGACIÓN Y ANÁLISIS COMO EQUIPO, Y NO UNA TRANSCRIPCIÓN DE TEMAS AISLADOS, PUES EN TODO MOMENTO LAS FUENTES DE CONSULTA DEBEN SER LA BASE DEL DESARROLLO DE ESTE PUNTO Y SUS APARTADOS.

3. COMO EQUIPO Y DERIVADO DEL ANÁLISIS ANTERIOR SE DEBEN PROPONER LOS ALGORITMOS Y ESTRUCTURAS DE DATOS NECESARIAS PARA LA IMPLEMENTACIÓN DE UN PROTOTIPO DE ANALIZADOR SEMÁNTICO. ES DECIR, EL USO DE ESTRUCTURAS DE DATOS COMO LA TABLA DE SÍMBOLOS Y LOS ALGORITMOS PARA DETERMINAR EL CORRECTO SENTIDO DE LOS ELEMENTOS QUE SE ANALIZAN, ETC.

   LA TABLA DE SÍMBOLOS, SU CORRECTO DISEÑO Y SOBRE TODO SU APROPIADO LLENADO, SERÁ LA BASE DE ESTE SIGUIENTE EJERCICIO.

   CONCRETAMENTE EN ESTE PUNTO DEBERÁN COMO EQUIPO, REDACTAR Y DETALLAR TODOS LOS ELEMENTOS QUE SE USARÁN PARA LA IMPLEMENTACIÓN DEL ANALIZADOR SEMÁNTICO.

4. PARA EL INCISO C DEL PUNTO 2 ANTERIOR, SE DEBERÁN CREAR PROGRAMAS DE MÍNIMO 25 LÍNEAS (SIN CONSIDERAR LOS COMENTARIOS) CADA UNO, EN LOS CUALES SE CODIFIQUE EN EL LENGUAJE PROTOTIPO, INSTRUCCIONES CON LÓGICA QUE EJEMPLIFIQUEN LOS ERRORES QUE EN CADA CASO DE ESTUDIO SE PROPONGAN.

   TALES PROGRAMAS DEBEN ESTAR PERFECTAMENTE DOCUMENTADOS Y CO-RELACIONADOS CON LOS CASOS DE ESTUDIO CORRESPONDIENTES.

5. CARACTERÍSTICAS QUE LA ACTIVIDAD 4 DEBE POSEER PARA CONSIDERARSE COMPLETA.

   A. LOS ANÁLISIS LÉXICO Y SINTÁCTICO, LIBRES DE ERRORES Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.

   B. UNA TABLA DE SÍMBOLOS PERFECTAMENTE ESTRUCTURADA CON LA TOKENIZACIÓN APROPIADA Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.

C. PLANTEAMIENTO Y FUNDAMENTACIÓN DE LOS CASOS DE USO DEL ANALIZADOR SINTÁCTICO, ASÍ COMO LOS ERRORES EN QUE DERIVARÁ CADA UNO DE ELLOS.

D. PREPARACIÓN DE LOS PROGRAMAS SUFICIENTES, ESCRITOS EN EL LENGUAJE PROTOTIPO, QUE APOYEN LA COMPROBACIÓN DE CADA CASO DE ESTUDIO. TALES PROGRAMAS DEBERÁN ESTAR COMPLETAMENTE DOCUMENTADOS.

E. GENERACIÓN DE UN CATÁLOGO DE ERRORES, CORRELACIONADO A LOS CASOS DE USO PROPUESTOS.

F. DISEÑO DE UNA SOLUCIÓN MODELADA EN OBJETOS, APOYADA EN DIAGRAMAS DE CLASE QUE DESCRIBAN LA ARQUITECTURA PROPUESTA PARA EL PROTOTIPO DEL ANALIZADOR SEMÁNTICO.

G. MODELADO E IMPLEMENTACIÓN DE UNA INTERFACE GRÁFICA CON LAS CARACTERÍSTICAS INDICADAS EN CLASE Y EN LA EVALUACIÓN PRESENCIAL PASADA.

H. MODELADO E IMPLEMENTACIÓN DE UN PROCESO DE :

GENERACIÓN DE UNA INTERFACE GRÁFICA DE USUARIO =>
LECTURA DE LA TABLA DE SÍMBOLOS =>
GENERACIÓN DE ÁRBOLES DE DERIVACIÓN SINTÁCTICA =>
REVISIÓN DE LOS TOKENS, SEGÚN LA SINTAXIS PROPUESTA
EN EL LENGUAJE PROTOTIPO =>
VALIDACIÓN DE LOS CASOS DE ESTUDIOS IDENTIFICADOS Y PROPUESTOS.

I. GENERACIÓN DE UN CALENDARIO DE ACTIVIDADES PLANIFICADAS VS. ACTIVIDADES REALIZADAS.

GENERACIÓN DE UNA BITÁCORA DE INCIDENCIAS.

J. TODAS LAS EVIDENCIAS GENERADAS Y REUNIDAS DEBERÁN INTEGRARSE A UN ARCHIVO PDF, NOMBRADO COMO SE INDICA MÁS ADELANTE.

K. ESTAS EVIDENCIAS PODRÁN ELABORARSE CON HERRAMIENTAS ELECTRÓNICAS, COMO PROCESADOR DE TEXTO, DE IMÁGENES, HOJAS DE CÁLCULO, ETC.

L. EL NÚCLEO DE CADA ALGORITMO CODIFICADO TAMBIÉN DEBERÁ FORMAR PARTE DEL ARCHIVO DE EVIDENCIAS.

6. EL PROYECTO DEBERÁ CONTAR CON LA IMPLEMENTACIÓN COMPLETA, Y SIN FALLOS DE LA GUI (*GRAPHIC USER INTERFACE*), LA CUAL SE HA VENIDO DEFINIENDO EN SU ESTRUCTURA DESDE EL INICIO DEL PROYECTO.

   SI EXISTEN AÚN DUDAS ACERCA DE CÓMO SE DEBE IMPLEMENTAR ÉSTA, FAVOR DE CONTACTAR POR LAS VÍAS INSTITUCIONALES PARA CUALQUIER ACLARACIÓN.

LA DOCUMENTACIÓN FORMAL (DOCUMENTO PDF A ENTREGAR) DEL PROYECTO DEBERÁ CONTENER LOS SIGUIENTES ELEMENTOS.

- DESCRIPCIÓN DEL PROYECTO.
- PLANTEAMIENTO DE OBJETIVOS Y ALCANCES.
- SUSTENTO Y DESARROLLO DE LA INVESTIGACIÓN.
- GENERACIÓN DEL CRONOGRAMA DE PROYECTO.
- BITÁCORA DE INCIDENCIAS.
- COMPROBACIÓN DE RESULTADOS MEDIANTE EL DISEÑO DE PRUEBAS SOBRE LOS CASOS DE USO.
- CONCLUSIONES OBTENIDAS.
- UNA SERIE DE VIDEOS QUE MUESTREN DE FORMA CLARA Y CONCRETA LOS AVANCES EXPUESTOS POR LOS INTEGRANTES DEL EQUIPO. ESTOS VIDEOS SE DEBERÁN PRESENTAR DURANTE EL PERIODO DE DESARROLLO DEL PROYECTO, Y NO AL FINAL DEL MISMO.
- UN VIDEO FINAL DONDE SE DEMUESTRE LA OPERATIVIDAD LIBRE DE FALLOS DEL PROYECTO.

NOTAS :

- PARA LA CORRECTA ACREDITACIÓN DE ESTA ACTIVIDAD, EL PROYECTO DEBERÁ CONTAR CON UN REGISTRO DE AL MENOS 6 (SEIS) REVISIONES DURANTE EL DESARROLLO DEL MISMO, Y NO AL FINAL DE ÉSTE.

  PARA TAL EFECTO SE DEBERÁN VALER DE VIDEOS COMPARTIDOS EN LA PLATAFORMA DE YOUTUBE, HECHAS POR EL EQUIPO Y DEMOSTRANDO EN ÉSTOS LOS AVANCES ALCANZADOS.

- EL INCUMPLIMIENTO DEL PUNTO ANTERIOR, INCONTROVERTIBLEMENTE SERÁ MOTIVO DE LA NO ACREDITACIÓN DE LA EVALUACIÓN 4.

- EN LAS REVISIONES DE LOS VIDEOS COMPARTIDOS, DEBERÁN PARTICIPAR TODOS LOS INTEGRANTES DEL EQUIPO Y PLANTEAR EL ESTADO ACTUAL DEL DESARROLLO DEL PROYECTO DE INVESTIGACIÓN.

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

NOTA A CONSIDERAR.

CADA UNO DE LOS PUNTOS ANTERIORES DEBE SER DESARROLLADO CON LA PROFUNDIDAD ACORDE A UN NIVEL PROFESIONAL, Y APEGÁNDOSE COMPLETAMENTE A LAS DIRECTRICES DE LA GUÍA TUTORIAL.

NO CONCIBA ESTE TRABAJO, COMO UN SIMPLE RESUMEN O EJERCICIO DE TRANSCRIPCIÓN, PUES EL VALOR INDICADO AL INICIO DE ESTA ACTIVIDAD LE DARÁ A USTED UNA BUENA IDEA DE LO QUE SE ESPERA DE ELLA, EN CUANTO A CALIDAD Y EL APRENDIZAJE OBTENIDO, MISMO QUE SERÁ PUESTO A PRUEBA MEDIANTE UN EXAMEN ESCRITO O BIEN ORAL EN CLASE.

SI DECIDIÓ ELABORAR ESTA ACTIVIDAD EN EQUIPO, CADA INTEGRANTE DE ÉSTE DEBERÁ POSEER EL MISMO NIVEL DE CONOCIMIENTO, PUES TAN SOLO REPARTIR TEMAS ENTRE LOS INTEGRANTES DEL EQUIPO, SUPONDRÍA UN GRAVE ERROR DE INTERPRETACIÓN A LA INTENCIÓN DIDÁCTICA REAL DE ESTA ACTIVIDAD.

POR ÚLTIMO, ESTA ACTIVIDAD SOLO SE PODRÁ DESARROLLAR EN EQUIPO, SI SE REGISTRÓ EN UNO PREVIAMENTE, UTILIZANDO EL FORMATO ENTREGADO EN LA ACTIVIDAD INICIAL. DE LO CONTRARIO DEBERÁ ELABORAR Y ENTREGAR LA ACTIVIDAD DE FORMA INDIVIDUAL.

LA ENTREGA DE DICHO REGISTRO SE HARÁ VÍA CORREO ELECTRÓNICO ENVIANDO ÉSTE AL PROFESOR DESIGNADO, Y POSTERIORMENTE EN CLASE ENTREGANDO LA HOJA EN FÍSICO.

NOTA GENERAL DE LA ACTIVIDAD :

SE DEBE CONSIDERAR Y TOMAR EN CUENTA PARA EL CORRECTO CUMPLIMIENTO DE ESTA ACTIVIDAD, LO SOLICITADO EN LA GUÍA TUTORIAL, CONCRETAMENTE EN EL PUNTO 3 INCISO i (*Trabajo en equipo*).

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

FIRMAS EN LA ACTIVIDAD

DADA LAS ACTUALES RESTRICCIONES DE CONFINAMIENTO QUE VIVIMOS ACTUALMENTE, PARA EL CASO DE LAS FIRMAS DEL DOCUMENTO, SE SUGIERE REALIZAR LO SIGUIENTE.

EN LUGAR DE QUE SE FIRMEN TODAS LAS HOJAS DE LA EVIDENCIA, POR CADA UNO DE LOS INTEGRANTES DEL EQUIPO, SE HARÁ UNA HOJA QUE DEBERÁ PRECEDER A LA PORTADA DEL TRABAJO DONDE SE LEA LA SIGUIENTE LEYENDA.

"LOS INTEGRANTES DEL EQUIPO NÚMERO ___ , ELABORAMOS LAS SIGUIENTES EVIDENCIAS ACADÉMICAS PARA CUMPLIR CON EL TEMARIO PROPUESTO EN LA MATERIA _____
DE LA ESPECIALIDAD DE _____ QUE CORRESPONDE A LA EVALUACIÓN 3 DEL CICLO ESCOLAR ENERO-JUNIO 2020, EN EL TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA.

NOMBRES Y FIRMAS DE CADA UNO DE LOS INTEGRANTES DEL EQUIPO "

ESTA HOJA SE FIRMARÁ POR CADA UNO DE LOS INTEGRANTES Y SE ENVIARÁ VÍA ELECTRÓNICA ENTRE TODOS A FIN DE QUE EL JEFE DE EQUIPO PUEDA INTEGRAR TODAS LAS FIRMAS AL TRABAJO FINAL.

ADEMÁS DE ESTO, CADA INTEGRANTE DEL EQUIPO DEBERÁ FIRMAR LAS HOJAS DEL TRABAJO QUE APORTE PARA ESTA ACTIVIDAD 4.

OBSERVACIONES:

- ✓ LA REVISIÓN SERÁ EN DIVERSAS VERTIENTES. PUEDE SER AL MOMENTO DE SOLICITAR LA CARPETA DE EVIDENCIAS, O BIEN PUEDE SER AL SOLICITAR LA EXPOSICIÓN DE LA TAREA EN LA CLASE. TAMBIÉN PUEDE SER POR SOLICITUD EXPRESA DEL INTERESADO A PARTICIPAR EN CLASE EXPONIENDO BREVEMENTE SU ACTIVIDAD.
- ✓ AQUELLAS ACTIVIDADES EN FORMATO DIGITAL SE DEBERÁN TENER SIEMPRE, Y EN TODO MOMENTO A LA MANO EN UNA MEMORIA USB.
- ✓ ÉSTAS ACTIVIDADES PODRÁN SER SOLICITADAS EN LA CLASE, O BIEN PARA SU ENVÍO A UNA CUENTA DE CORREO.
- ✓ ESTAS ACTIVIDADES DEBEN ESTAR LISTAS E INTEGRADAS A LA CARPETA DE EVIDENCIAS (*FÍSICAMENTE*) A LA FECHA DE ENTREGA INDICADA AL FINAL DE ÉSTE DOCUMENTO.
- ✓ CADA HOJA QUE ENTREGUE DE SU ACTIVIDAD, DEBERÁ ESTAR FIRMADA AL MARGEN DERECHO, INCLUIDA LA PROPIA SOLICITUD DE LA ACTIVIDAD.
- ✓ UNA VEZ ELABORADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA NOMENCLATURA QUE SE INDICA MÁS ADELANTE EN ESTE DOCUMENTO.
- ✓ SI SUS EVIDENCIAS ENVIADAS POR CORREO, NO CUMPLEN CON LA NOMENCLATURA SOLICITADA, NO SERÁN CONSIDERADAS COMO EVIDENCIAS PARA SU EVALUACIÓN.
- ✓ CON ESTA ACTIVIDAD, USTED DEBERÁ IR INTEGRANDO SUS CARPETAS FÍSICA Y ELECTRÓNICA DE EVIDENCIAS, Y AL FINAL DEL SEMESTRE EN UN DISCO COMPACTO HARÁ ENTREGA DE SU CARPETA ELECTRÓNICA DE EVIDENCIAS.
- ✓ PARA TENER DERECHO A LA REVISIÓN Y EVALUACIÓN DE SUS ACTIVIDADES, DEBE REGISTRAR SU ASISTENCIA A CLASE, EL DÍA SEÑALADO PARA LA ENTREGA DE LA MISMA.
- ✓ MUY IMPORTANTE: SI LA ACTIVIDAD INCLUYE AL MENOS UN MATERIAL PLAGIADO, ES DECIR, TRANSCRITO Y PRESENTADO COMO REDACCIÓN PROPIA O DEL EQUIPO, EL VALOR DE ESTA ACTIVIDAD SERÁ NULO.
- ✓ FALTAR A CLASE EL DÍA DE LA ENTREGA, ANULA LA REVISIÓN DE SUS EVIDENCIAS.
- ✓ POR ÚLTIMO, POR FAVOR GESTIONE APROPIADAMENTE SU TIEMPO, Y SEA PUNTUAL EN SU ENTREGA Y ASÍ EVITAR PROBLEMAS DE NULIDAD POR EXTEMPORANEIDAD.
- ✓ AÚN PARA TRABAJOS EN EQUIPO APLICAN TODAS LAS MISMAS OBSERVACIONES ANTERIORES.

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

LA NOMENCLATURA SOLICITADA PARA ENVIAR SU TRABAJO ES LA SIGUIENTE :

```
AAAA-MM-DD_MATERIA_DOCUMENTO_EQUIPO_NOCTROL_APELLIDOS_NOMBRE_SEM.PDF
```

(NOTA : *** TODO EN MAYÚSCULA ***)

DONDE :

AAAA	:	AÑO
MM	:	MES
DD	:	DÍA
MATERIA	:	LAII, PG, LI, PE
DOCUMENTO	:	A4-ACTIVIDAD 4, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA, ETC. (CAMBIANDO EL NÚMERO CONSECUTIVO POR EL QUE CORRESPONDA)
EQUIPO	:	NÚMERO DEL EQUIPO QUE CORRESPONDA SEGÚN INDICACIÓN DEL PROFESOR.
NOCTROL	:	SU NÚMERO DE CONTROL
APELLIDOS	:	SUS APELLIDOS
NOMBRE	:	SU NOMBRE
SEM	:	EL PERIODO SEMESTRAL EN CURSO: ENE-JUN / AGO-DIC

EJEMPLO :

SI EL TRABAJO SE HACE EN EQUIPO.

```
2020-06-01_LAII_A4_EQUIPO_99_9999999_PEREZ_PEREZ_JUAN_ENE-JUN20.PDF
```

SI EL TRABAJO SE HACE INDIVIDUALMENTE.

```
2020-06-01_LAII_A4_9999999_PEREZ_PEREZ_JUAN_ENE-JUN20.PDF
```

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

FECHA DE ENTREGA:

VÍA CORREO ELECTRÓNICO, EL LUNES 8 DE JUNIO DEL 2020, CON HORA LÍMITE DE ENTREGA HASTA LAS 14:00 HORAS (2 DE LA TARDE).

EN CASO DE QUE EL TRABAJO SE HAYA ELABORADO EN EQUIPO, EL JEFE DEL MISMO SERÁ EL ÚNICO RESPONSABLE DE ENVIAR LA ACTIVIDAD A LA SIGUIENTE CUENTA DE CORREO:

ricardo.gonzalez@itcelaya.edu.mx

MUY IMPORTANTE:

DESPUÉS DE LAS 14:00 HRS. EN PUNTO, LA ACTIVIDAD YA SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO CONTARÁ COMO EVIDENCIA PARA SU EVALUACIÓN.

SE LE SUGIERE ENVIAR CON ANTICIPACIÓN SU ACTIVIDAD A FIN DE EVITAR CONFLICTOS POR NO ENTREGAR ÉSTA A TIEMPO.

RECUERDE ANEXAR A SU ARCHIVO .PDF DE EVIDENCIAS, ESTA SOLICITUD DE ACTIVIDADES CON TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO.

POR ÚLTIMO, CONSIDERE EL SIGUIENTE CALENDARIO OFICIAL, PARA GESTIONAR ADECUADAMENTE EL TIEMPO QUE SE LE OTORGA PARA DESARROLLAR ESTA ACTIVIDAD.

CALENDARIO 2019-2020 - EVALUACIÓN 4

En el presente documento se presentará de primera instancia el marco teórico y toda la metodología de desarrollo correspondiente al análisis semántico como se solicita en la actividad número 4. En segunda instancia se presentará una pequeña unificación de los análisis anteriores (léxico y sintáctico) y en tercera instancia la culminación del proyecto libre de fallos y errores ejecutando los análisis anteriores, así como la propia documentación.

TABLA DE CONTENIDO

MARCO TEORÍCO

ANÁLISIS SEMÁNTICO

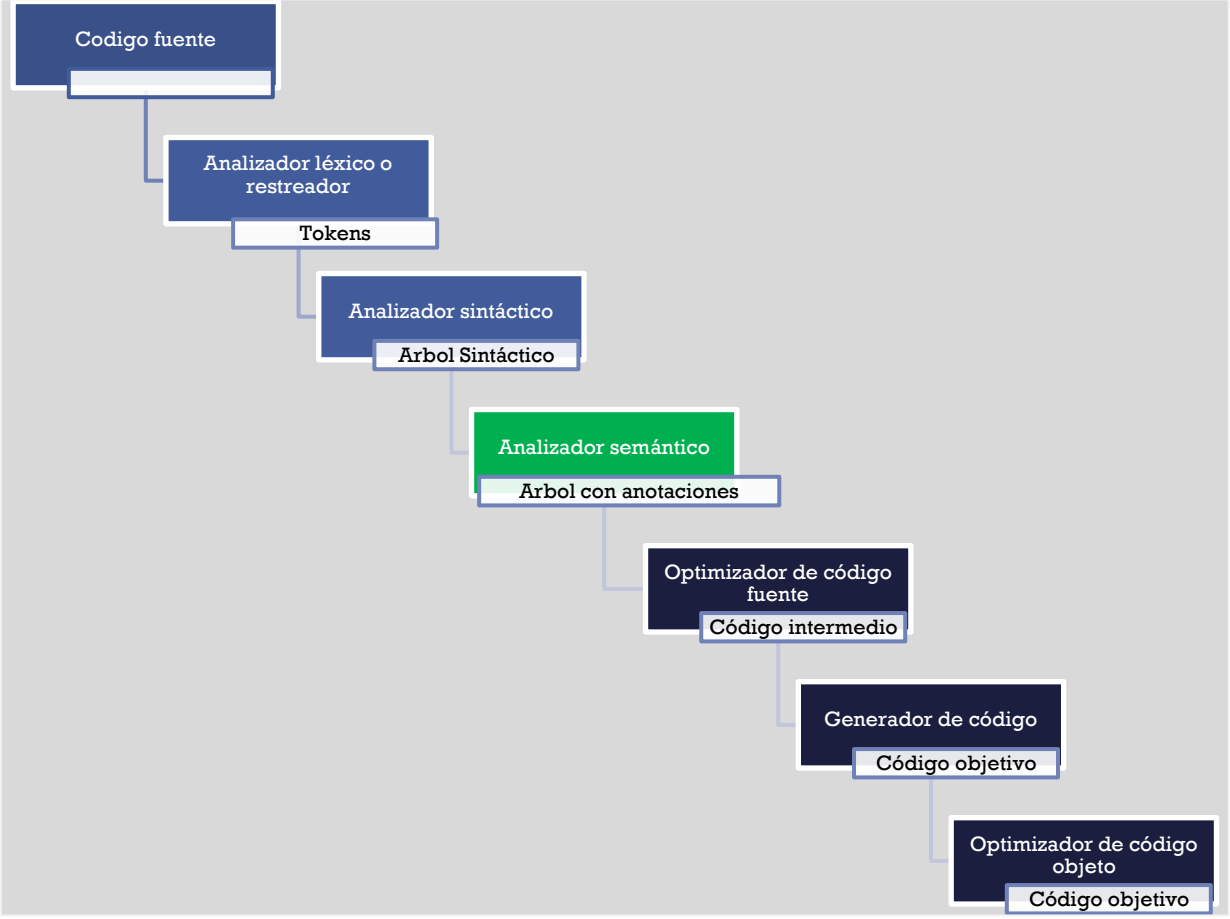


Figura 1. Fases de un compilador

En el presente trabajo analizaremos la fase de un compilador que calcula la información adicional necesaria para la compilación una vez que se conoce la estructura sintáctica de un programa, la fase semántica.

Es interesante como esta fase rebasa las capacidades de las gramáticas libres de contexto y los algoritmos de análisis sintáctico, por lo que no se considera como sintaxis.

La semántica de un programa está estrechamente relacionada con el SIGNIFICADO final del programa que se traduce. Como el análisis que realiza un

compilador tiene lugar antes de la ejecución (estático) dicho análisis semántico también se conoce como análisis semántico estático. En un lenguaje típico estáticamente tipificado como C. el análisis semántico involucra la construcción de una tabla de símbolos para mantenerse al tanto de los significados de nombres establecidos en declaraciones e inferir tipos y verificarlos en expresiones y sentencias con el fin de determinar su exactitud

dentro de las reglas de tipos del lenguaje.

El análisis semántico se puede dividir en dos categorías:

el análisis de un programa que requiere las reglas del lenguaje de programación para establecer su exactitud y garantizar una ejecución adecuada

• categoria 1

el análisis realizado por un compilador para mejorar la eficiencia de ejecución del programa traducido. Esta clase de análisis por lo regular se incluye en análisis de "optimización", o técnicas de mejoramiento de código.

• categoria 2

Es conveniente remarcar que las dos categorías no son mutuamente excluyentes, ya que los requerimientos de exactitud, tales como la verificación de tipos estáticos también permiten que un compilador genere código más eficiente que para un lenguaje sin estos requerimientos

El análisis semántico estático involucra tanto la descripción de los análisis a realizados como la implementación de los análisis utilizando algoritmos apropiados. En este sentido, es semejante al análisis léxico y sintáctico. En el análisis sintáctico, por ejemplo, utilizamos gramáticas libres de contexto en la Forma Backus-Naus (BNF, por sus siglas en inglés) para describir la sintaxis y diversos algoritmos de análisis sintáctico descendente y ascendente para implementar la sintaxis. En el análisis semántico la situación no es tan clara, en

parte porque no hay un método estándar (como el BNF) que permita especificar la semántica estática de un lenguaje, y en parte porque la cantidad y categoría del análisis semántico estático varía demasiado de un lenguaje a otro. Un método para describir el análisis semántico que los escritores de compiladores usan muy a menudo con buenos efectos es la **identificación de atributos**, o propiedades, de entidades del lenguaje que deben calcularse y escribir ecuaciones de atributos o reglas semánticas, que expresan cómo el cálculo de tales atributos está relacionado con las reglas gramaticales del lenguaje. **Un conjunto así de atributos y ecuaciones se denomina gramática con atributos. Las gramáticas con atributos son más útiles para los lenguajes que obedecen el principio de la semántica dirigida por sintaxis, la cual asegura que el contenido semántico de un programa se encuentra estrechamente relacionado con su sintaxis.**

En un lenguaje estáticamente tipificado como C , el tipo de datos de una variable o expresión es un importante atributo en tiempo de compilación. Un verificador de tipo es un analizador semántico que calcula el atributo de tipo de datos de todas las entidades del lenguaje para las cuales están definidos los tipos de datos y verifica que estos tipos cumplan con las reglas de tipo del lenguaje. En un lenguaje corno C un verificar de tipo es una parte importante del análisis semántico. Sin embargo, en un lenguaje como LISP, los tipos de datos son dinámicos, y un compilador LISP debe generar código para calcular tipos y realizar la verificación de tipos durante la ejecución del programa.

Como vimos a partir de estos ejemplos, los cálculos de atributos son muy variados. Cuando aparecen de manera explícita en un compilador pueden presentarse en cualquier momento durante la compilación: aun cuando asociarnos el cálculo de atributos más estrechamente con la fase de análisis semántico, tanto el analizador léxico como el analizador sintáctico pueden necesitar disponer de información de atributo, y puede ser necesario realizar algún análisis semántico al mismo tiempo que un análisis sintáctico

EJEMPLO DE SEMÁNTICA EN LENGUAJE C

la forma más práctica de entender la diferencia entre la semántica y la sintaxis en un lenguaje de programación (en tu caso quieres que sea el lenguaje C) es definiendo los errores semánticos y los errores sintácticos

Errores sintácticos: cuando existe código inválido que el compilador no entiende. Por ejemplo, intentas multiplicar una variable string (cadena) con un integer (entero) en C. El compilador lo detectará porque no puede compilarlo así. O cuando olvidas cerrar algún paréntesis o algún corchete, es un error sintáctico.

Errores semánticos: el código de programación es válido porque el compilador lo entiende, pero el programa resultante no hace lo que el programador quiere que haga. No hay forma de que el compilador detecte estos errores.

Los errores semánticos son "errores lógicos", la lógica detrás del código escrito no hace lo que el programador cree que hará. ´

Algunos errores semánticos en C

```
// Sumar uno 1 a la variable X

x -= 1;
```

En este caso estamos restándole 1 a la variable X.

```
// Sumar 1 a la variable X
```

Reglas Semanticas	Las reglas semánticas vienen dadas en función de los atributos de los demás símbolos que componen la regla.
	La evaluación de las reglas semánticas puede generar código, guardar información en una tabla de símbolos, emitir mensajes de error o realizar otras actividades.
	La traducción de la cadena de componentes léxicos es el resultado obtenido al evaluar las reglas semánticas.
	La evaluación de las reglas semánticas define los valores de los atributos en los nodos del árbol de análisis sintáctico para la cadena de entrada.

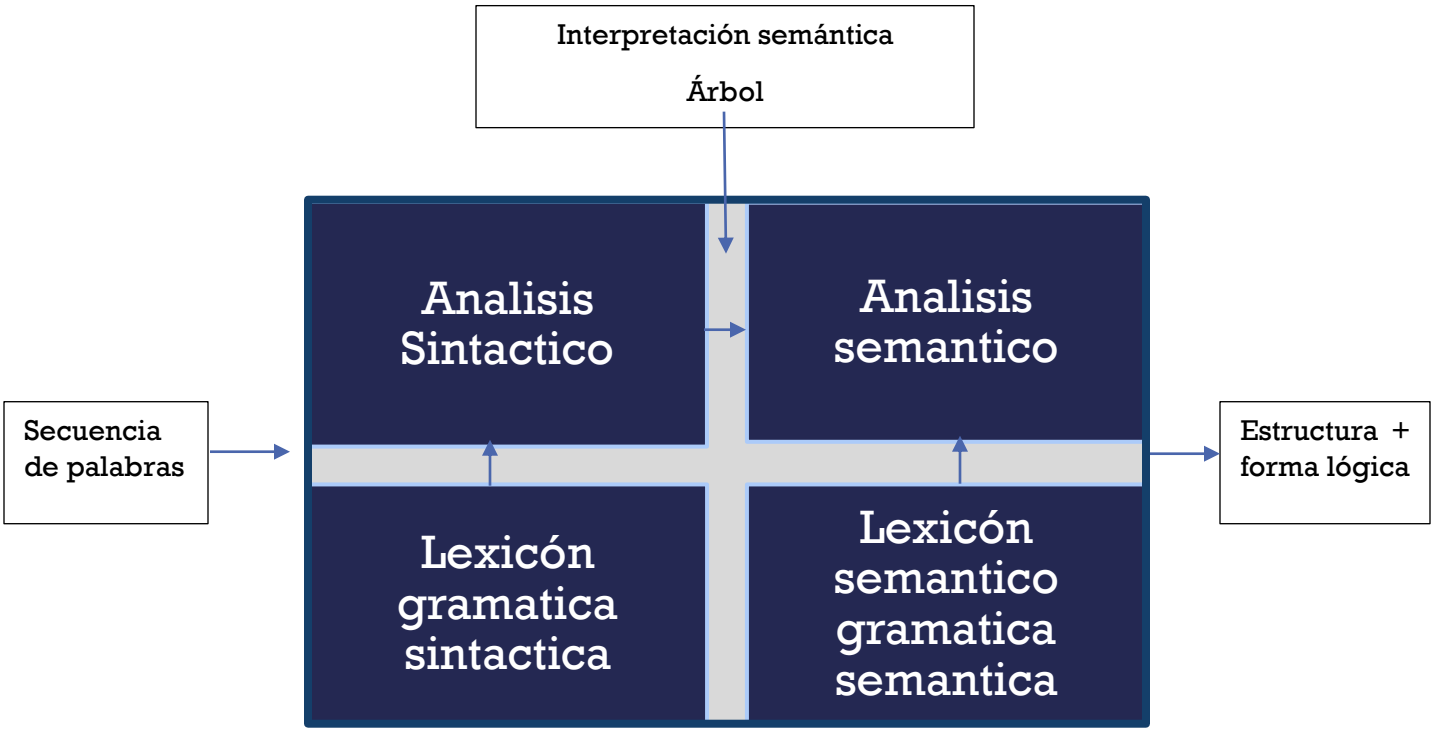
PROCESO DEL ANÁLISIS SEMÁNTICO

El análisis semántico utiliza como entrada el árbol sintáctico detectado por el análisis sintáctico para comprobar restricciones de tipo y otras limitaciones semánticas y preparar la generación de código. El chequeo semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles o que los parámetros reales de una función sean coherentes con los parámetros formales

FUNCIONES PRINCIPALES

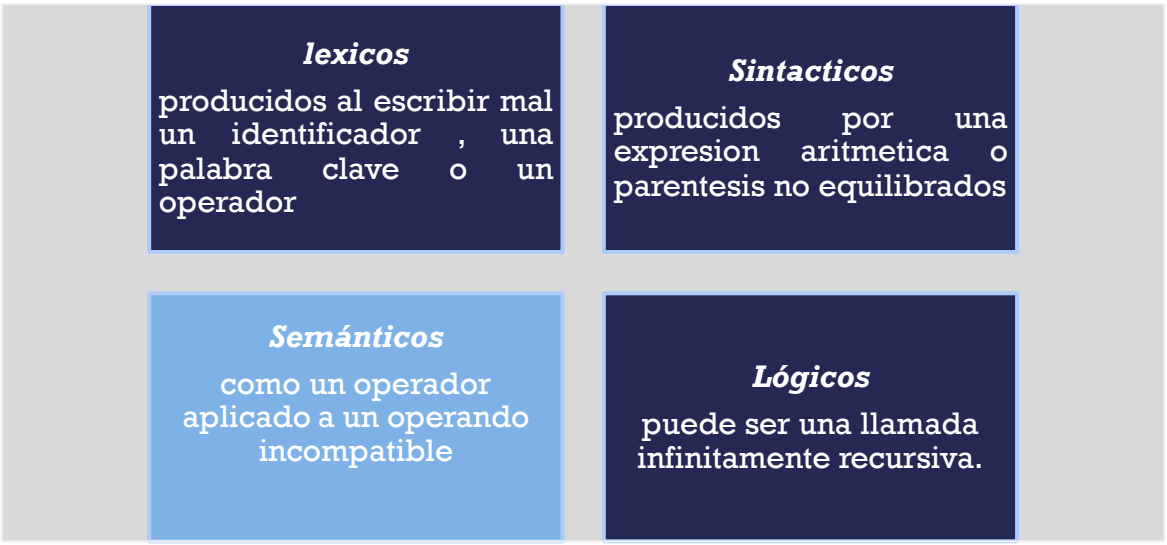
- Identificar cada tipo de instrucción y sus componentes
- Completar la Tabla de Símbolos
- Realizar distintas comprobaciones y validaciones:
- Comprobaciones de tipos.
- Comprobaciones del flujo de control.

A continuación, se muestra el proceso de interpretación semántica de una frase a partir del diagrama de bloques



Lexicón: diccionario de palabras, normalmente categorizado

MANEJO DE ERRORES SEMANTICOS



A continuación, se muestra una tabla con los errores que se consideraron en esta primera esta etapa del analizador semántico, los cuales cuentan con un número de error y un mensaje correspondiente a este.

ID	ORIGEN
300	No coinciden los tipos
301	Variable no declarada
302	Fuera de alcance

CASOS DE ERRORES SEMÁNTICOS

Descripción: Este tipo de errores cuando el tipo no coincide con lo que el leguaje solicita

Mensaje: No coinciden los tipos

Error 300

Entre los errores que se encuentran por la desigualdad de tipos están:

- Le asignan el valor de un tipo diferente del que se asignó a la variable.

```
ent $c;
$c='hola';
```

Esto marcaría un error pues la variable c tiene como tipo de dato un entero y se le esta asignando un tipo de dato cadena

- Se comparan diferentes tipos de datos.

```
cad $l="a";
If($c==$a)
```

Esto marcaría un error pues la variable c es de tipo entero y la variable a es de tipo String

- Se realizan operaciones aritméticas con diferentes tipos

```
rea $h;
$h=l+$c;
```

Esta operación no se puede realizar pues los tipos de datos no son iguales.

Descripción: Este tipo de errores se da cuando una variable que se va a utilizar en una instrucción no se encuentra declarada. En estos casos aplicaría cuando se realiza cualquier instrucción pues no encuentra la variable con la cual se va a trabajar.

Mensaje: Variable no declarada

Error 301

- Cuando intentas asignar un valor a una variable sin declarar

```
$a=10+5;
```

En este caso no se podría ejecutar dicha instrucción pues no existe la a.

- Cuando intentas comparar una variable que no está declarada

```
ent $a=4;
if($a==$b)
```

No se ejecuta el if pues la variable b no se encuentra declarada

Descripción: Este tipo de errores se da cuando en una subrutina se declara una variable y la quieres utilizar fuera de esta, en estos casos esa variable solo esta disponible en la subrutina y no fuera de esta.

Mensaje: fuera de alcance.

Error 302

Ejemplo :

```
Subrutina(){
   ent $c;
      }
   ent $l;
   $l=$c+5
```

PROGRAMAS PRUEBA PARA EL ANÁLISIS SEMÁNTICO

A continuación, se muestran algunos códigos de nuestro lenguaje prueba con el propósito de ejemplificar el correcto funcionamiento de nuestro análisis semántico dentro de nuestro proyecto.

PROGRAMA 1

```
programa $prueba {

    /*AQUI ES DONDE DEBEMOS DECLARAR TODAS LAS VARIABLES*/
    ent $uno,$dos;
    rea $tres,$cuatro;
    cad $cinco,$seis;


    subrutina $hola(){
        /*SE GENERA EL ERROR 303 YA QUE NUNCA SE ASIGNO ALGÚN VALOR
PREVIO A $dos*/
        $tres = $dos;

        /*SE GENERA EL ERROR 300 YA QUE NO COINCIDEN LOS TIPOS ENTRE $uno
y $cinco*/
        $uno = $cinco;

        $dos = 4241;
        $tres = 43.1;
    }


    subrutina $sub(){
        /*SE GENERA EL ERROR 301 YA QUE NO FUE DECLARADA LA VARIABLE
$siete*/
        si($dos <= $siete){
            salida(432423);
        }
        sino{
            entrada();
        }
    }


    principal() {
        $uno = entrada();
    }
}
```

PROGRAMA 2

```
programa $prueba {

    /*AQUI ES DONDE DEBEMOS DECLARAR TODAS LAS VARIABLES*/


    /*NO DECLARAMOS NINGUNA VARIABLE ASÍ QUE SE NOS GENERARAN MULTIPLES
ERRORES
    301 POR LA VARIABLE NO DECLARADA*/


    subrutina $hola(){
        $dos = 2;
        $tres = $dos + 1;

        $cinco = 5;
        $uno = $cinco;

        $dos = 4241;
        $tres = 43.1;
    }


    /*DE IGUAL MANERA COMO NO EXISTEN SE GENERAN ERRORES DE TIPOS NO
COMPATIBLES Y VARIABLES SIN VALOR, 300 Y 303 RESPECTIVAMENTE*/


    subrutina $sub(){
        si($dos <= 7){
            salida(432423);
        }
        sino{
            entrada();
        }
    }


    principal() {
        $uno = entrada();
    }
}
```

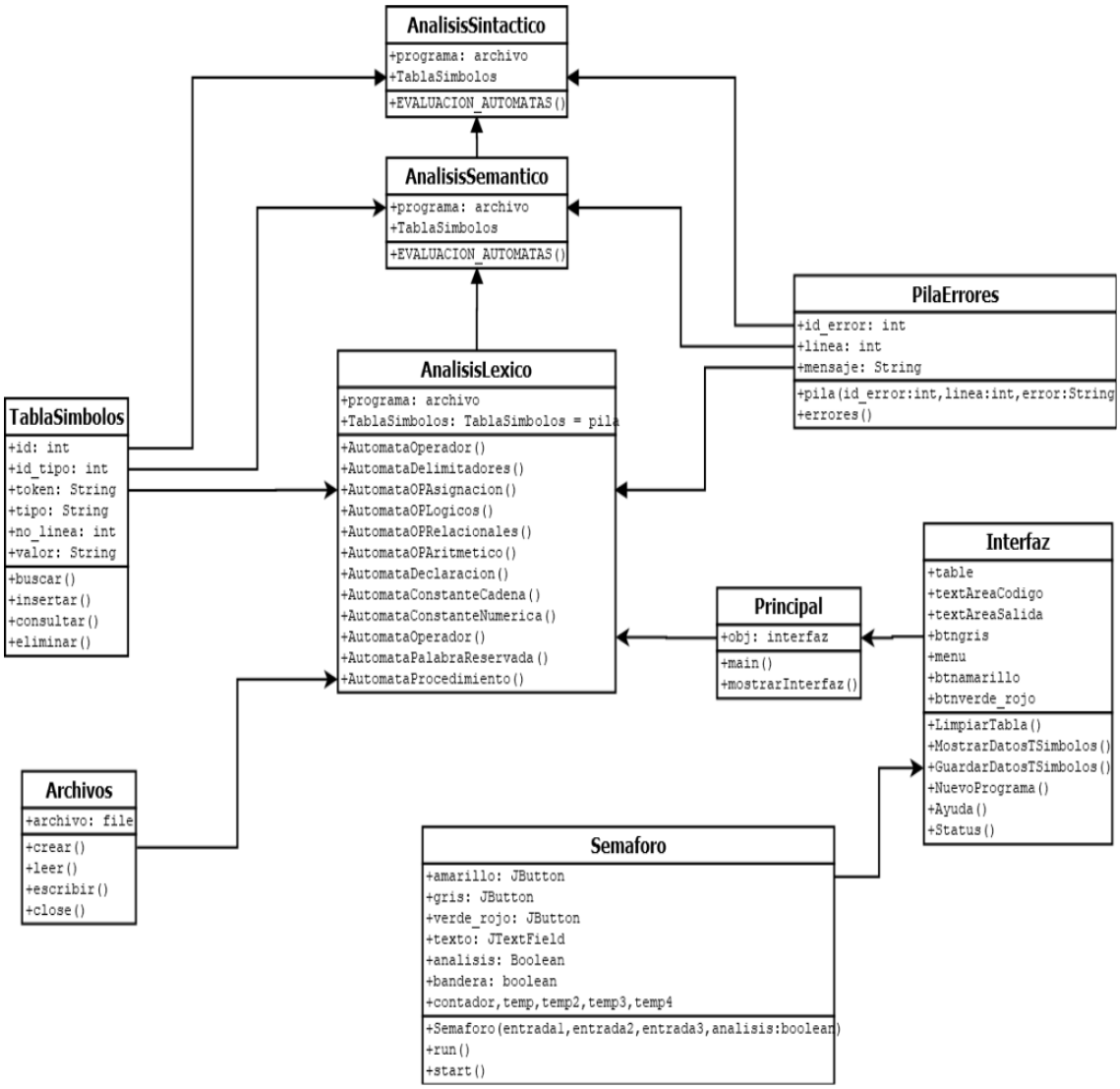
PROGRAMA 3

```
programa $prueba {

    /*AQUI ES DONDE DEBEMOS DECLARAR TODAS LAS VARIABLES*/
    ent $dos;
    ent $tres;
    rea $cinco;
    rea $siete;
```

```
    subrutina $hola(){
        $dos = 2;
        $tres = $dos + 1;

        $cinco = 5;

        /*NOS GENERA EL ERROR 301 YA QUE NO SE DECLARO LA VARIABLE $uno*/

        $uno = $cinco;

        /*NOS GENERA EL ERROR 300 YA QUE SE INTENTA ASIGNAR UN NÚMERO
REAL A UN ENTERO*/

        $dos = 4241.1;
        $tres = 43.1;
    }


    subrutina $sub(){
        si($dos <= 7){
            salida(432423);
        }
        sino{
            entrada();
        }
    }


    principal() {
        /*NOS GENERA EL ERROR 301 YA QUE NO SE DECLARO LA VARIABLE $uno*/
        $uno = entrada();
    }
}
```

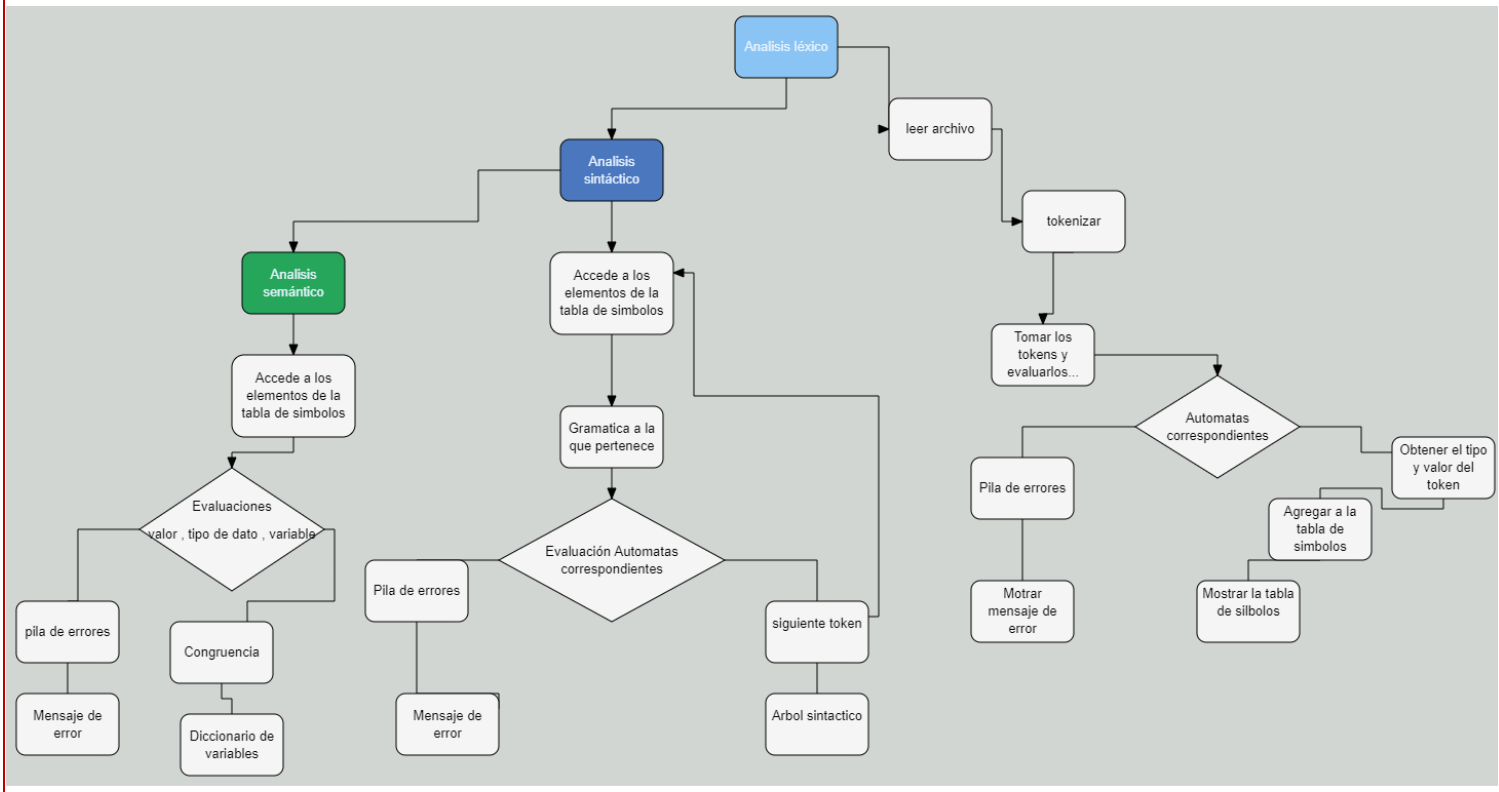
UNIFIED MODELING LANGUAGE

DIAGRAMA DE CLASES



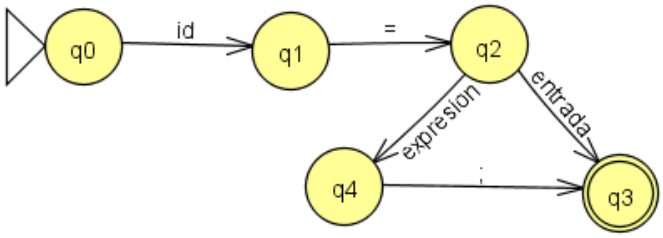
ESQUEMA DE LA ARQUITECTURA

El presente diagrama es un esquema de la arquitectura de los análisis léxico, sintáctico y semántico



AUTOMATAS (ANÁLISIS SEMÁNTICO)

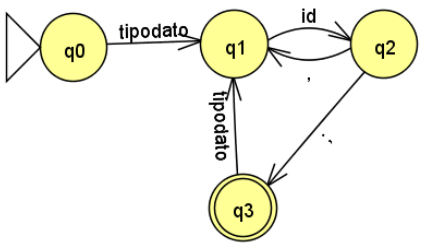
ASIGNACIÓN



	id	=	entrada	expresión	;
→Q0	Q1				
Q1		Q2			
Q2			Q3	Q4	

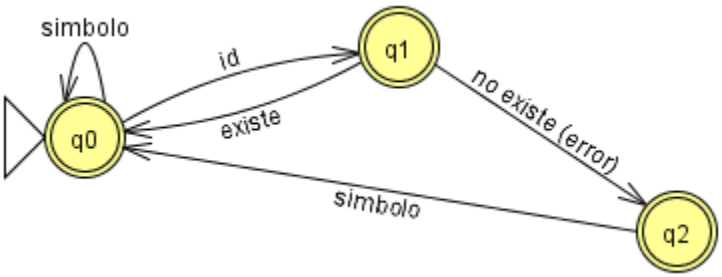
*Q3					
Q4					Q3

DECLARACIONES



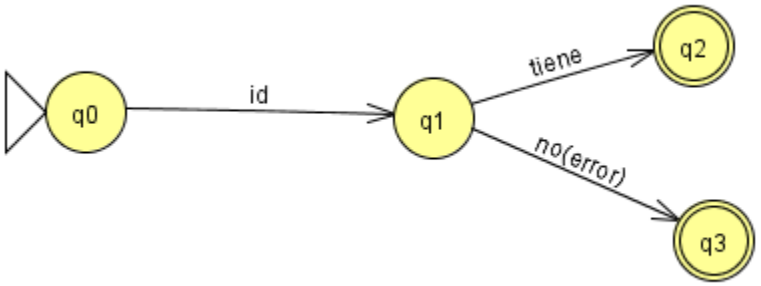
	tipodato	id	;	,
→Q0	Q1			
Q1		Q2		
Q2			Q3	Q1
*Q3	Q1			

VERIFICAR USO



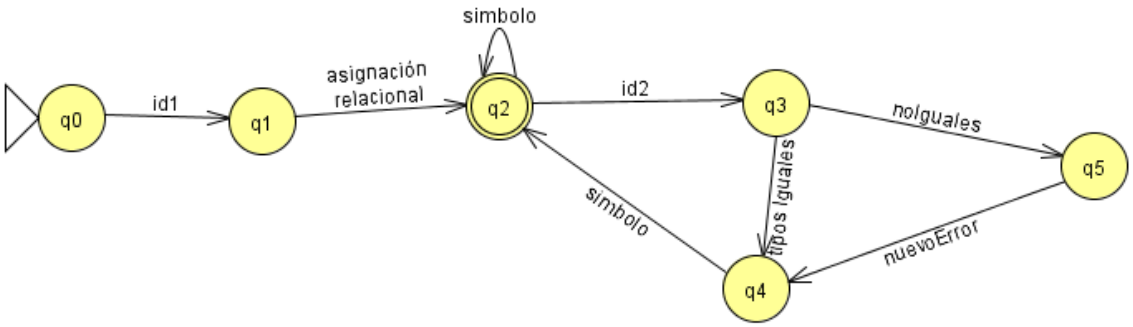
	simbolo	id	existe	no existe(error)
→Q0*	Q0	Q1		
Q1*			Q0	Q2
Q2*	Q0			

TIENE VALOR



	id	tiene	no(error)
→Q0	Q1		
Q1		Q2	Q3
Q2*			
Q3*			

MISMO TIPO



	id1	asignacion relacional	simbolo	id2	tipos iguales	no iguales	nuevoError
→Q0	Q1						
Q1		Q2					
Q2*			Q2	Q3			
Q3					Q4	Q5	
Q4			Q2				
Q5							Q4

DOCUMENTACIÓN ANÁLISIS SEMÁNTICO

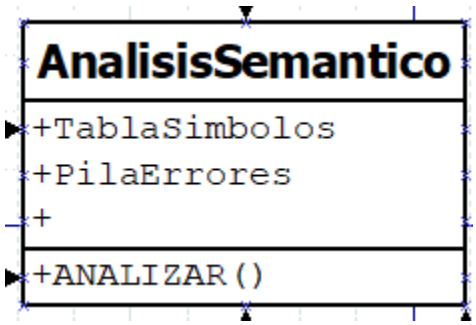
A continuación, se detallará como fue implementado el Analizador semántico dentro de nuestro Pseudocompilador que estamos desarrollando, se mostrara el diagrama de clase correspondiente a la parte mostrada, junto con una muestra del código en el cual se muestran los métodos que contiene la clase, los métodos se encuentran sin desplegar para mostrar la estructura en un espacio más compacto.

Se mencionarán las clases utilizadas e implementadas dentro del análisis semántico.

CLASE “ANALISISSEMANTICO”:

Primero vemos que nuestra clase de análisis semántico trabaja sobre el resultado del análisis sintáctico, el fin de esta clase es darle sentido a nuestro código generado, además de ello cuenta con la pila de errores donde se almacenaran y posteriormente se podrá acceder a los registros de los errores generados en la ejecución del analizador sintáctico.

Se tienen los métodos analizar, el cual de acuerdo a nuestra estructura llama a los autómatas designados para este análisis para que nos regrese si las instrucciones dadas tienen un sentido lógico o en caso de error de llame a nuevoError, donde se agrega un error a la pila de errores, finalizado ese análisis se pasa al siguiente token para ser analizado.



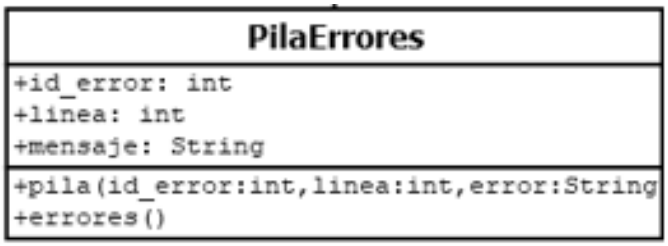
```
public class AnalisisSemantico {
    public TablaSimbolos tablaSimbolos = new TablaSimbolos();
    public PilaErrores pilaErrores = new PilaErrores();
    public int actual = 0;
    ArrayList<Simbolo> variables = new ArrayList<>();

    public AnalisisSemantico(TablaSimbolos tablaSimbolos, PilaErrores pilaErrores) {...}

    public AnalisisSemantico() {
    }

    public void analizar(){...}

    public void declaracion(){...}

    public void verificarUso(){...}

    public void verificarValores(){...}
}
```

Cada autómata se encuentra implementado dentro de la misma clase, ya que para esté análisis fueron necesarios menos que en los anteriores y más simplificados.

CLASE “PILAERRORES”

La pila de errores al mostrar lo contenido, muestra todo lo almacenado dentro de ella, imprimiendo en pantalla el id del error, la línea en la cual se generó y el correspondiente mensaje de error.

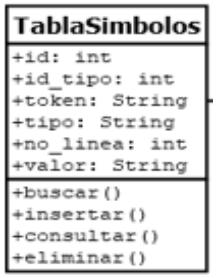
Cabe mencionar que la implementación dentro de la ejecución permite mostrar los errores específicos de cada analizador implementado, así como de todos los errores al mismo tiempo.



```
public class PilaErrores {
    public Stack<Error> pila = new Stack<~>();

    //MÉTODO PARA INGRESAR ALGÚN ERROR ENCONTRADO EN LA PILA DE ERRORES
    public void ingresa(int id,int linea) { pila.push(new Error(id,linea)); }

    //MÉTODO PARA MOSTRAR LO CONTENIDO EN LA PILA DE ERRORES
    public void mostrar(){...}

    //MÉTODO PARA IMPRIMIR EL CORRESPONDIENTE MENSAJE DE ERROR
    public String listaErrores(Error e){...}
}
```

CLASE "TABLASIMBOLOS"

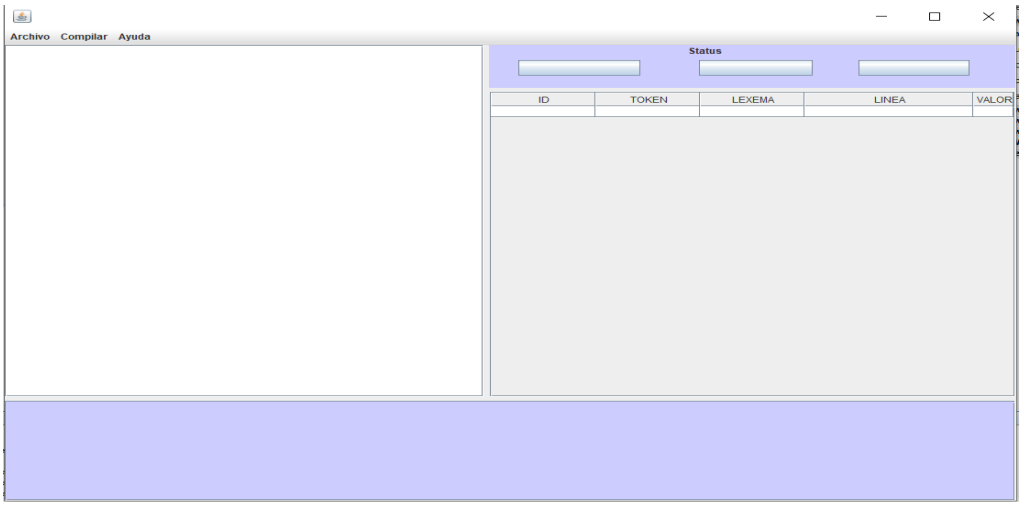
Accedemos a la tabla de símbolos para ir actualizándola conforme se utilizan las variables, además de esto verificar varias cuestiones que nos dicen si nuestro análisis de realizo exitosamente.



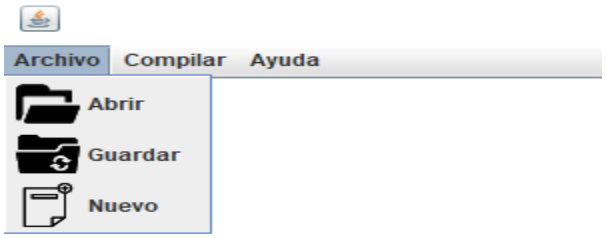

```
public class TablaSimbolos {
    private ArrayList<Simbolo> tablaSimbolos = new ArrayList<>();

    //MÉTODO QUE MUESTRA LO CONTENIDO EN LA TABLA DE SIMBOLOS
    public void consultar(){...}

    //MÉTODO QUE PERMITE INGRESAR UN REGISTRO EN LA TABLA DE SIMBOLOS
    public void ingresar(Simbolo s){...}

    //MÉTODO QUE VACIA LA TABLA DE SIMBOLOS
    public void eliminar() { tablaSimbolos.clear(); }

    //MÉTODO QUE PERMITE BUSCAR UN REGISTRO DENTRO DE LA TABLA DE SIMBOLOS
    public Simbolo buscar(int id){...}
}
```

ESPECIFICACIONES DE LA INTERFAZ DE USUARIO

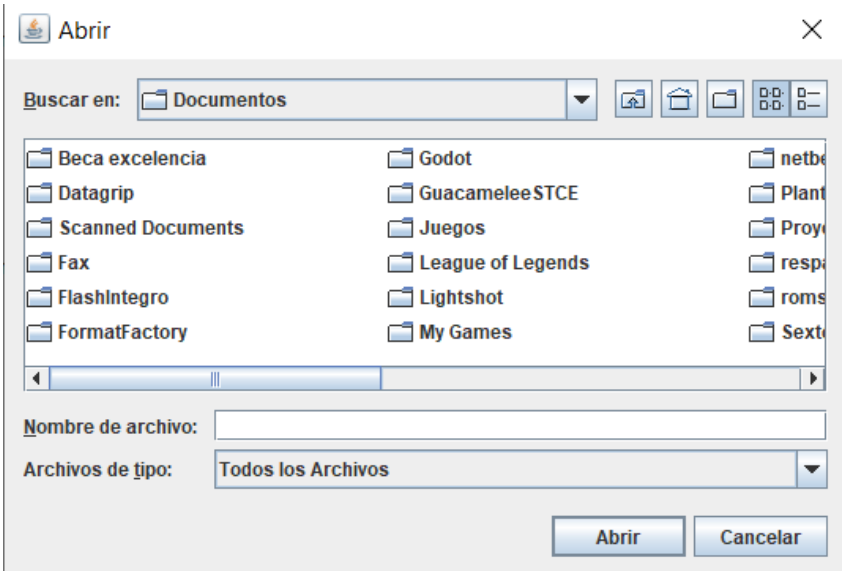
La vista principal de la interfaz de usuario es la siguiente:



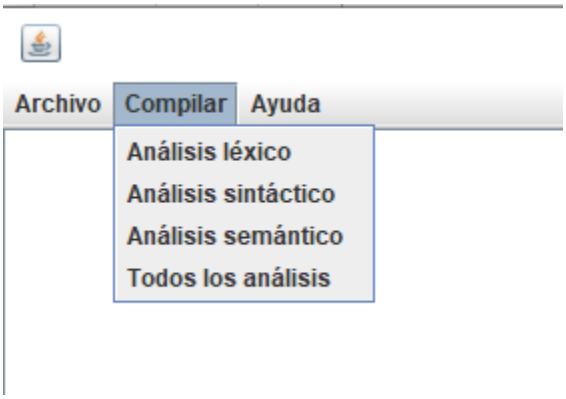
Para crear, guardar o abrir un archivo nos dirigimos a las siguientes opciones:



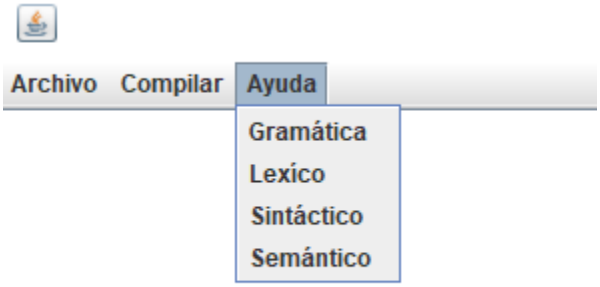
Se nos desplegaran ventanas como las siguientes al hacer uso de abrir y de guardar, al hacer uno nuevo se vacía nuestro espacio de código.



Al realizar la compilación, lo podemos hacer por partes, si decidimos hacer uso de un único análisis, si el código cuenta con un error, únicamente se mostrarán los errores referentes al análisis realizado, si se realizan todos los análisis se muestra todos los errores que surjan.

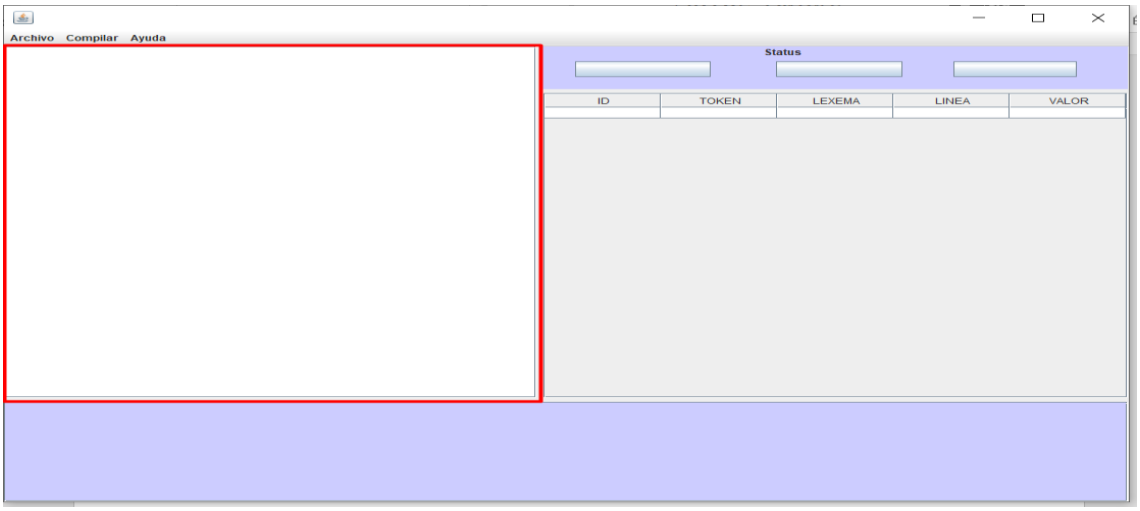


Mediante la opción ayuda se desplegarán los PDF referentes a la actividad realizada en cuestión, siendo gramática la actividad número 1, léxico la actividad número 2 y de esta manera sucesivamente.

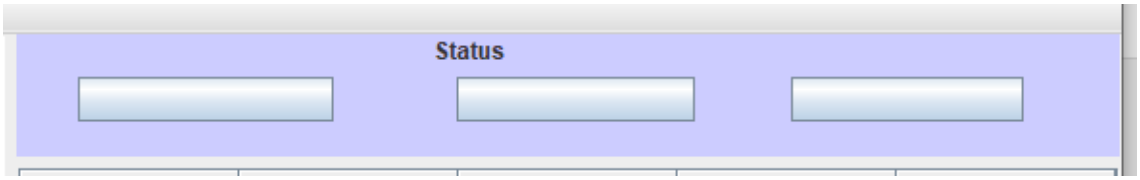


El PDF se desplegará en nuestro programa visualizador de documentos PDF preferido o en dado caso nos preguntará donde visualizarlo.

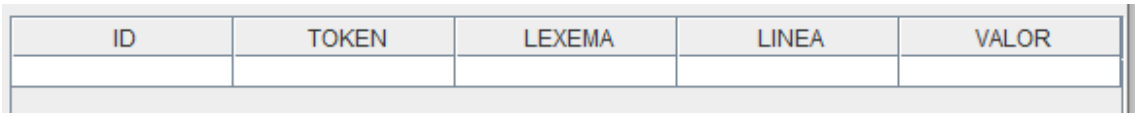
En el espacio marcado en color rojo es el espacio dedicado para escribir nuestro código de nuestros programas



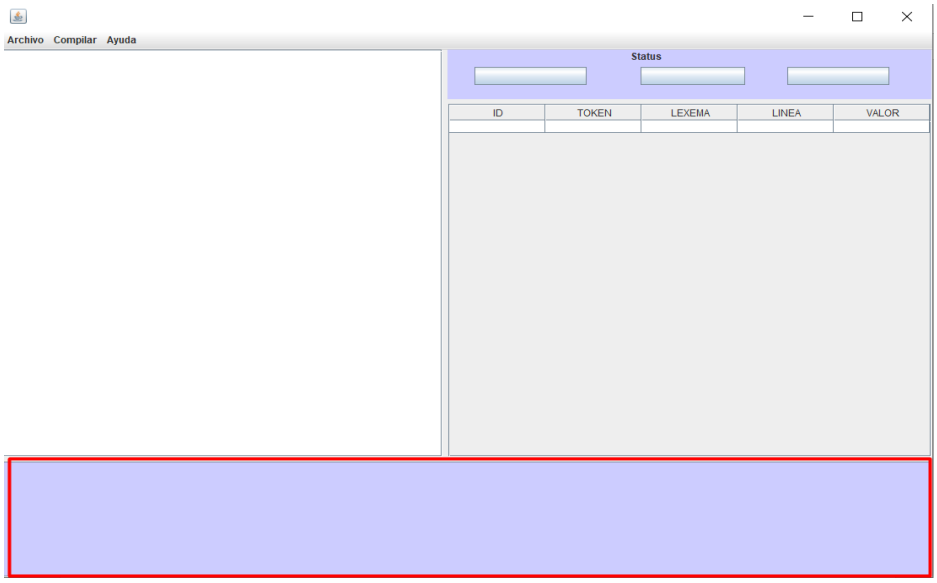
El espacio designado con la palabra "status", es donde nuestro semáforo marca si fue compilado exitosamente mostrando un color verde al final o si genero algún error, mostrando el color rojo.



Donde vemos una tabla únicamente con las cabeceras se mostrará nuestra tabla de símbolos una vez se realice algún análisis.

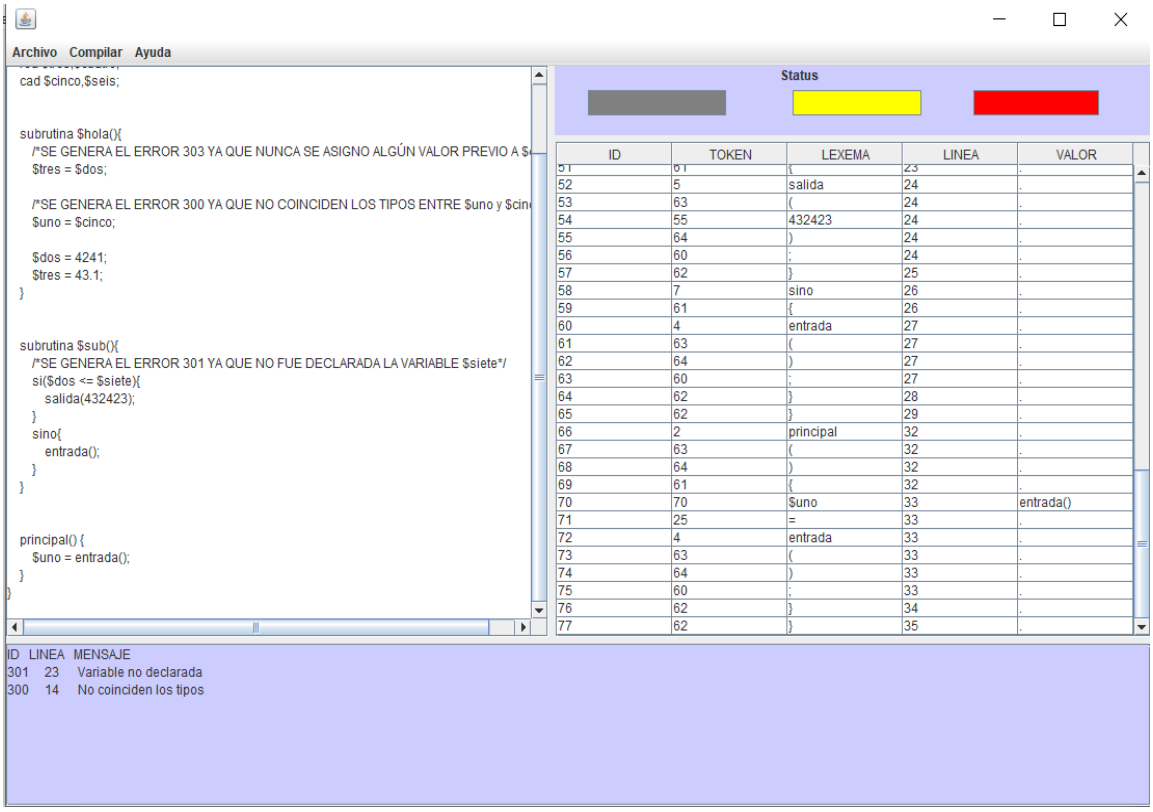


Por último, en el parte inferior marcado de rojo, es nuestra "consola" donde se muestran los mensajes que deban ser desplegados al usuario o en dado caso los errores que se generaron.

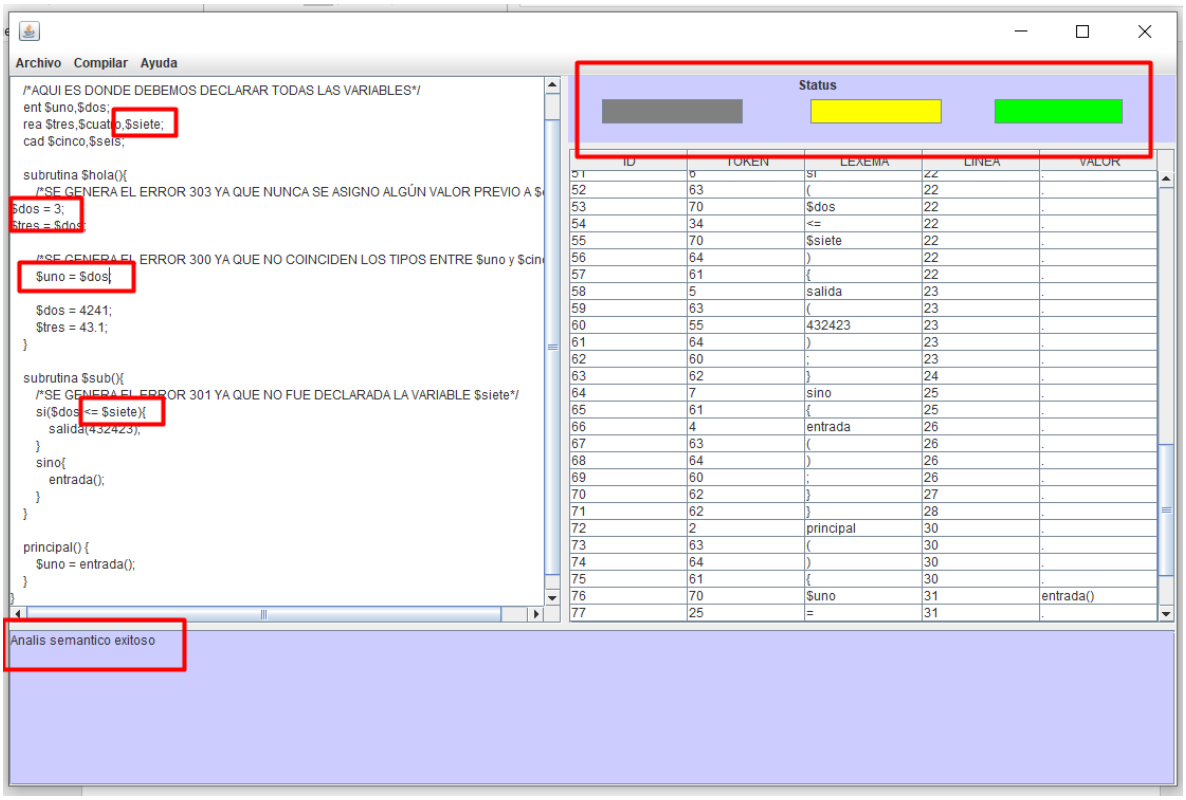


EJECUCIÓN DE LA FASE SEMÁNTICA

Para ejemplificar el comportamiento de nuestro Pseudocompilador en la implementación del análisis semántico veamos un ejemplo a continuación haciendo uso del programa prueba número 1 del presente documento.



Al corregir los errores nuestro compilador funciona correctamente.



BITACORA DE INCIDENCIAS DURANTE EL ANÁLISIS SEMÁNTICO

BITACORA DE INCIDENCIAS ANÁLISIS SEMANTICO			
Fecha	**Hora**	**Descripción de la incidencia**	**Solución**
Lunes 18 de Mayo del 2020	12:00 PM	Reunión para establecer el desarrollo del Análisis Semántico	El equipo se dio a la tarea de reunirse para analizar el desarrollo del análisis semántico respecto a sus necesidades técnicas para su elaboración
Jueves 21 de Mayo del 2020	10:00 AM	Elaboración de los autómatas para el análisis semántico	Los miembros del equipo se reunieron para elaborar los autómatas a utilizar en el analizador semántico.
Lunes 25 de Mayo del 2020	8:00 AM	Reunión para la elaboración del documento final	El equipo se reunió para ver los requerimientos del documento a entregar y repartir los tópicos requeridos para su elaboración.
Lunes 1 de Junio del 2020	8:00 AM	Codificación de los autómatas a utilizar en el analizador semántico.	El equipo se dio a la tarea de reunirse para realizar la codificación de los autómatas para el analizador semántico, así como pruebas en consola de su funcionamiento.
Jueves 4 de Junio del 2020	12:00 PM	Unificación del analizador semántico con la GUI	El equipo realizó la unificación del analizador semántico con la GUI para finalizar la creación del pseudocompilador.

PSEUDOCOMPILADOR

DESCRIPCIÓN DEL PROYECTO.

Una vez concluidas de manera satisfactoria la primera (análisis léxico) y segunda (análisis sintáctico) etapa del proyecto lo que compete a continuación será el desarrollo, investigación diseño e implementación de la tercera etapa un analizador semántico. Comenzaremos con el desarrollo del marco teórico y todos los conceptos fundamentales como en las actividades anteriores, cabe mencionar que para llegar hasta este punto son imprescindibles las fases léxica y sintáctica que se desarrollaron a lo largo del curso ya que cada uno de los análisis desencadenan una serie de eventos que dependen uno del otro para su correcto funcionamiento, de no ser así se presentarían fallos en el sistema.

OBJETIVO GENERAL DEL CURSO:

Desarrollar software de base: traductor, intérprete o compilador que le permita al alumno comprender la base de la construcción de un lenguaje de propósito específico o general

OBJETIVOS ESPECÍFICOS:

Conocer las bases para el diseño de un lenguaje de programación, para la construcción de un léxico y analizador sintáctico, a partir de un lenguaje de programación o un generador de analizadores, así como el diseño de un analizador semántico para un meta-compilador

GRAMATICA TIPO 2 LIBRE DE CONTEXTO DEL LENGUAJE PROTOTIPO

1. <programa>:: =programa <identificador> ´{´<declaraciones> * <procedimientos> * principal´ ()´ ´{´<instrucciones>*´}´ ´}'

2. <identificador>::= '$'<letra>[(<letra> | <digito>)*]

3. <declaraciones>:: =<tipoDato> { <identificador> [´,´<identificador>] } ';'

4. <tipoDato>:== entero | real | cadena

5. <letra> ::= a|b|c|...|x|y|z |A|B|C|...|X|Y|Z|

6. <digito> ::= 0|1|2|3|4|5|6|7|8|9

7. <procedimientos>::= subrutina <identificador> ´()´ ´{´[<instrucciones>*]´}´

8. <instrucciones>::={ <estructuraDecision> | <estructuraCiclo> | <entrada> | <salida> | <asignacion>}

9. <asignación> ::= <identificador> '=' (<entrada> | <expresión>)

10. <entrada> ::= entrada();

11. <salida>::=salida(<concatenación>);

12. <concatenación>::= { (<expresion> | <cadena>)'__'{<concatenación>} }

13. <expresion>::=(<identificador>| {<digito>}) [<OprAritmetico> <expresion>] | ´(´<expresión>´)´

14. <estructuraDecision>::= si '(' <condición> ´)´´{´ <instrucciones> ´}´ [sino ´{´<instrucciones>´}´]

15. <condición>::=<expresión> [(<OpRelacional>|<OpLogico>)<condicion>]

16. <estructuraCiclo>::= mientras´(´ <condición> ´)´ ´{´ <instrucciones> ´}´

TABLA DE SÍMBOLOS

Para distinguir unas palabras (o lexemas) de otras se utilizan patrones de reconocimiento. En muchos casos los patrones se describen mediante expresiones regulares. Los identificadores pueden también sustituirse por referencias a la tabla de símbolos, para una utilización más eficiente. La tabla de símbolos asocia a cada identificador un número, así como una serie de atributos (tipo de datos, etc.).

Una tabla de símbolos también se la llama tabla de nombres o tabla de identificadores, es una estructura de datos que usa el proceso de traducción de un lenguaje de programación, por un compilador o un intérprete, donde cada símbolo en el código fuente de un programa está asociado con información tal como la ubicación, el tipo de datos y el ámbito de cada variable, constante o procedimiento.

Permanece sólo en tiempo de compilación, no de ejecución, excepto en aquellos casos en que se compila con opciones de depuración. La tabla almacena la información que en cada momento se necesita sobre las variables del programa, información tal como: Nombre, tipo, dirección de localización, tamaño, etc. La gestión de la tabla de símbolos es muy importante, ya que consume gran parte del tiempo de compilación.

De ahí que su eficiencia sea crítica. Aunque también sirve para guardar información referente a los tipos creados por el usuario, tipos enumerados y, en general, a cualquier identificador creado por el usuario, nos vamos a centrar principalmente en las variables de usuario

Almacenamiento del nombre. Se puede hacer con o sin límite. Si lo hacemos con límite, emplearemos una longitud fija para cada variable, lo cual aumenta la velocidad de creación, pero limita la longitud en unos casos, y desperdicia espacio en la mayoría.

El tipo también se almacena en la tabla, como veremos en un apartado dedicado a ello.

Dirección de memoria en que se guardará. Esta dirección es necesaria, porque las instrucciones que referencian a una variable deben saber dónde encontrar el valor de esa variable en tiempo de ejecución, también cuando se trata de variables globales. En lenguajes que no permiten recursividad, las direcciones se van asignando secuencialmente a medida que se hacen las declaraciones. En lenguajes con estructuras de bloques, la dirección se da con respecto al comienzo del bloque de datos de ese bloque, (función o procedimiento) en concreto.

También podemos guardar información de los números de línea en los que se ha usado un identificador, y de la línea en que se declaró.

IDENTIFICADOR	DIRECCIÓN	TIPO	DIMENSIÓN	OTROS ATRIBUTOS
compañía	STATIC+0	C	10	...
x3	STATIC+10	I	0	...
forma1	STATIC+12	B	0	...
b	STATIC+13	F	3	...

La interfaz de la tabla de símbolos debe quedar clara desde el principio de manera que cualquier modificación en la implementación de la tabla de símbolos no tenga repercusión en las fases del compilador ya desarrolladas.

Las operaciones básicas que debe poseer son:

Crear ()	Insertar(símbolo)	Buscar (nombre)	Imprimir()
crea una tabla vacía.	añade a la tabla el símbolo dado	devuelve el símbolo cuyo nombre coincide con el parámetro. Si el símbolo no existe devuelve null.	a efectos informativos, visualiza por la salida estándar la lista de variables almacenadas en la tabla de símbolos junto con sus

			valores asociados.

La tabla de símbolos propuesta será la siguiente:

ID	TOKEN	LEXEMA	LINEA	VALOR
1	29	$hola	1	0
2	24	{	2	-
3	29	$suma	3	0
4	15	=	4	-
6	28	6	5	-
8	13	+	6	-
7	28	5	7	-
5	25	}	8	-

ID: Este es un id que se asigna a cada token que se ingresa en la tabla, este debe ser único para cada uno.

TOKEN: Este TOKEN que se asigna al lexema corresponde al número de dependiendo de su clasificación, es decir si es una palabra reservada todos los tokens que correspondan a una palabra reservada tendrán el mismo TOKEN.

A continuación, se listan el TOKEN que corresponde a cada token según sea su clasificación.

LEXEMA: Esta columna corresponde al nombre de todos los tokens que se encuentran en la tabla, ya sea una palabra reservada, un identificador, delimitador etc.

LINEA: es el numero de la línea correspondiente al programa

TABLA DE TOKENS

TOKEN	LEXEMA
1	programa
2	principal
3	subrutina
4	entrada
5	salida
6	si
7	sino
8	mientras
20	+
21	-
22	*
23	/
25	=
30	<
31	>
32	==
33	>=
34	<=
35	!=
40	&
41	|
50	ent
51	rea
52	cad

55	Entero
56	Real
57	cadena
60	;
61	{
62	}
63	(
64	)
65	/*
66	*/
67	,
68	_
70	Identificador ($xy)

CATALOGO DE ERRORES

ANÁLISIS LÉXICO

TABLA DE ERRORES

Para un mayor control de errores en el análisis léxico se propone una tabla de errores en la cual se clasificarán dependiendo el análisis que se esté ejecutando.

A continuación, le listan los id de error junto con el mensaje de descripción que corresponde a cada uno partiendo del numero100.

CASOS	*ORIGEN*
100	"No puede definirse como un identificador, debe de iniciar con un signo de $"
101	"No deben de existir espacios en blanco entre los identificadores"
102	"El token no se puede establecer como identificador ni palabra reservada, no pertenece al lenguaje"
103	"Las constantes numéricas no pueden mezclarse con letras o caracteres extraños"

104	"No se puede establecer como identificador"
105	"Los números decimales no son válidos"
106	"los caracteres con acentos no son válidos"
107	"Token invalido no se pueden utilizar operadores en la nomenclatura de las variables"
108	"se deben de separar los tokens"

CASOS DE ERRORES LEXÍCOS:

Descripción: se declara un identificador sin el carácter $, este caso ocurre cuando el programador intenta declarar una variable que no contenga el carácter $ al inicio de este.

- Error: 100
- Mensaje: "no puede definirse como un identificador, debe de iniciar con un signo de $"
- Ejemplo:

```
Entero numero=12;
 Cadena palabra;
```

Descripción: existen espacios en blanco en la nomenclatura de las variables, este caso ocurre cuando el programador intenta declarar una variable y existen espacio en blanco entre ellas.

- Error: 101
- Mensaje:"no deben de existir espacios en blanco entre los identificadores"
- Ejemplo:

```
$es un numero=12;
Cadena $la  palabra;
```

Descripción: las palabras reservadas y declaraciones se escriben de manera incorrecta o contienen símbolos extraños

- Error: 102
- Mensaje: "el token no se puede establecer como identificador ni palabra reservada, no pertenece al lenguaje"
- Ejemplo:

```
S1 , S| , S?
S1N0 , S|N0 , SIN°
MIENTR4Z , $°>ídem
```

Descripción: no se puede escribir una constante numérica con caracteres diferentes a números, si el programador ingresa alguna constante numérica, pero le coloca letras o algún otro carácter no establecido

- Error: 103
- Mensaje: "las constantes numéricas no pueden mezclarse con letras o caracteres extraños"
- Ejemplo:

```
Entero $numero = 123ABC;
123|.
123?)
123xc__23
```

Descripción: no puede existir un $ intermediario en los identificadores

- Error: 104
- Mensaje: "no se puede establecer como identificador"
- Ejemplo:

```
Entero $enter$o;
Real $soy$real$;
Cadena $hola$c;
```

Descripción: números inválidos, el error se presenta cuando el programador ingresa un nuero que contenga varios puntos o punto y coma.

- Error: 105
- Mensaje: "los números decimales no son válidos"
- Ejemplo:

```
REAL $numero = 12.3.4,2 ;
```

Descripción: acentos en los caracteres del token, si el programador ingresa alguna cadena que contenga algún acento en alguno de sus caracteres será incorrecto.

- Error: 106
- Mensaje: "los caracteres con acentos no son válidos"
- Ejemplo:

```
$programación , $camión , $estación , $jugaría
```

Descripción: no se pueden utilizar operadores en la nomenclatura de las palabras, si el programador ingresa alguna variable la cual contenga un operador aritmético, lógico u relacional en medio de esta

- Error: 107
- Mensaje: "token invalido no se pueden utilizar operadores en la nomenclatura de las variables"
- Ejemplo:

```
$progr*amación, $ca>mión, $es&&tación , $jug>=aría
```

Descripción: se introduce una serie de tokens validos sin espacio entre ellos, si el programador ingresa alguna cadena la cual está formada por varios tokens validos en el lenguaje, pero estas no contienen un espacio entre ellas se genera el error

- Error: 108
- Mensaje: "se deben de separar los tokens"
- Ejemplo:

```
Entero$numero=32;
Cadena$unacadena=" hola soy cadena";
```

ANÁLISIS SINTÁCTICO

TABLA DE ERRORES

Para un mayor control de errores en el análisis léxico se propone una tabla de errores en la cual se clasificarán dependiendo el análisis que se esté ejecutando.

A continuación, le listan los id de error junto con el mensaje de descripción que corresponde a cada uno.

ID	ORIGEN
210	No se inició el programa
211	No se asignó un identificador
212	Se esperaba que se abriera una llave {
213	Se esperaba el cierre de llaves }
214	No se incluyó el método principal
215	Se esperaba abrir un paréntesis (
216	Se esperaba el cierre de paréntesis)

217	Se esperaba la asignación a la variable
218	No se asignó nada a la variable
219	Falto punto y coma
220	No hay operador relacional
221	Expresión no reconocida

CASOS DE ERRORES SINTÁCTICOS

Descripción: se inicia erróneamente el programa. Este caso ocurre cuando el programador intenta iniciar el programa sin la palabra reservada

- Error: 210
- Mensaje: "no se inició el programa"
- Ejemplo:

```
$programa {
/*CONTENIDO*/
}
```

Descripción: cualquier subrutina debe de tener su identificador asignado. Este caso ocurre cuando el programador intenta declarar una subrutina sin su identificador

- Error: 211
- Mensaje: "no se asignó un identificador"
- Ejemplo:

```
Subrutina(){
/*CONTENIDO*/
}
```

Descripción: llaves de apertura mal equilibrados .Este caso ocurre cuando el programador olvida agregar la llave de apertura

- Error: 212
- Mensaje: "se esperaba que se abriera una llave {"
- Ejemplo:

```
Subrutina $suma()
/*CONTENIDO*/
}
```

Descripción: llaves de cierre mal equilibrados .Este caso ocurre cuando el programador olvida agregar la llave de cierre

- Error: 213
- Mensaje: "se esperaba que se cerrara una llave }"
- Ejemplo:

```
Subrutina $suma() {
   /*CONTENIDO*/
```

Descripción: no se incluyó el procedimiento principal. Este caso ocurre cuando el programador olvida incluir el procedimiento principal

- Error: 214
- mensaje: "no se incluyó el método principal"
- ejemplo:

```
    ( ) {
entrada( );
    }
```

Descripción: paréntesis de apertura mal equilibrados . Este caso ocurre cuando el programador olvida agregar los paréntesis de apertura

- Error: 215
- Mensaje: "se esperaba que se abriera una paréntesis ("
- Ejemplo:

```
Subrutina $suma ) {
   /*CONTENIDO*/
    }
```

Descripción: paréntesis de cierre mal equilibrados . Este caso ocurre cuando el programador olvida agregar el paréntesis de cierre

- Error: 216
- Mensaje: "se esperaba que se cerrara una paréntesis)"
- Ejemplo:

```
Subrutina $suma(  {
   /*CONTENIDO*/
    }
```

Descripción: falta un punto y coma. Este caso ocurre cuando el programador olvida agregar el punto y coma al final de la expresión

- Error: 219
- mensaje: "falto punto y coma".
- ejemplo:

```
Entrada()
```

Descripción: falto operador relacional dentro de la expresión. Este caso ocurre cuando el programador olvida ingresar el operador relacional

- Error: 220
- Mensaje: "no hay operador relacional".
- Ejemplo:

```
si($cuatro  $cinco){
```

```
salida(432423);
}
```

Descripción: existe una expresión no reconocida por el lenguaje .Este caso ocurre cuando el programador ingresa alguna expresión que no es perteneciente al lenguaje definido

- Error: 221
- mensaje: "expresión no reconocida".
- ejemplo:

```
Entrada()
    $id = (;##
```

PROGRAMAS PRUEBA

ANÁLISIS LÉXICO

PROGRAMA 1

```
//Iniciamos con el cuerpo base del programa
programa $prueba {
    //En la siguiente línea nos generara el error 100, ya que
    //un identificador no puede estar sin el signo $
    entero numero;
    real $x,$y;
    //En la siguiente línea genera el error 108 ya que se ingresan
    //tokens validos sin espacio entre ellos
    cadena$hola;

    principal () {
        $x = 1;
        //La siguiente línea genera el error 105, ya que
        //un numero no puede tener dos puntos
        $y = 2..43;

        si($x >= $y){
            salida($x);
        }
        sino {
            salida($y);
        }
    }
}
```

PROGRAMA 2

```
//Iniciamos con el cuerpo base del programa
programa $prueba {

    //La siguiente línea genera el error 107 ya que se ingresan
    //operadores en la nomenclatura de las palabras
    entero $pru*ba;

    real $x,$y,$res;

    //La siguiente línea genera el error 106 ya que se ingresan
    //acentos en el identificador
    cadena $númeroReal;

    principal () {
        $x = 7654764;

        //La siguiente línea genera el error 103 ya que se ingresan
        //caracteres no validos en un número
        $y = 5849FJDSAJFKAJF;

        si($x >= $y) {
            $res = $x * $y;
            salida($res);
        }
        sino{
            $res = 2 * $x;
            salida($res);
        }
    }
}
```

PROGRAMA 3

```
//Iniciamos con el cuerpo base del programa
programa $prueba {

    //La siguiente línea genera el error 101, ya que
    //no debe existir espacio en medio del identificador
    entero $prue ba;

    real $x,$y,$res;

    //La siguiente línea genera el error 104, ya que
    //se tiene el signo $ dos veces
    cadena $numero$Real;

    principal () {
        $x = entrada ();
        $y = entrada ();

        si ($x >= $y) {
            $res = $x * $y;
            salida($res);
        }
        //La siguiente línea genera el error 102, ya que
        //se escribe mal la palabra reservada sino
        sin0{
            $res = 2 * $x;
            salida($res);
        }
    }
}
```

ANÁLISIS SINTÁCTICO

En la ejecución de los programas prueba aparecerán más errores de los mencionados, ya que un programa con errores sintácticos puede desencadenar más errores en líneas posteriores, aunque tengamos una cantidad significativa de errores en cascada, comúnmente se solucionan con solucionar el primer error.

PROGRAMA 1

```
/*En la siguiente linea nos genera un error 210 ya que nos hace falta la
palabra reservada programa*/
 $fdsafsa {
    ent $gfdsg,$fdaf;
    rea $fdsa,$gfsd;
    /*En la siguiente linea nos marcara el error 211 ya que no se asigno
un identificador a la subrutina*/
    subrutina (){
        entrada();
        mientras($tres>$cuatro){
            /*En la siguiente linea nos marcara un error 215 ya que no se
agregaron los parentesis*/
            $gfds = entrada;
        }
    }
    subrutina $gfdgfds(){
        si($cuatro <= $cinco){
            salida(432423);
        }
        sino{
            /*En la siguiente linea nos marcara un error 216 ya que no se
cerraron los parentesis*/
            entrada(;
        }
    }
    principal() {
        entrada();
    }
}
```

PROGRAMA 2

```
/*En la siguiente linea se mostrara un error 212 ya que no se abrieron
las llaves*/
programa $fdsafsa
    ent $gfdsg,$fdaf;
    rea $fdsa,$gfsd;

    subrutina $hola(){
        entrada();
        mientras($tres>$cuatro){
            $gfds = entrada();
        }
```

```
    }

    subrutina $gfdgfds(){
        si($cuatro <= $cinco){
            salida(432423);
        }
        sino{
            entrada();
        }
    /*En la siguiente linea se mostrara un error 213 ya que no se
abrieron las llaves*/


    principal() {
        /*En la siguiente linea marcara el error 219 ya que hace falta un
punto y coma*/
        entrada()

        /*En la siguiente linea marcara el error 221 ya que se quiere
asignar algo no reconocido por el lenguaje*/
        $id = (;
    }
}
```

PROGRAMA 3

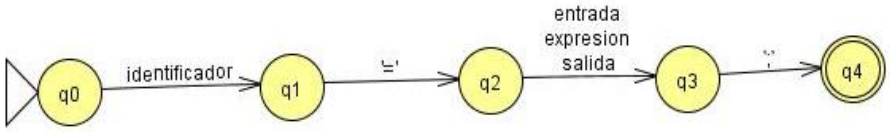
```
programa $fdsafsa {
    ent $gfdsg,$fdaf;
    rea $fdsa,$gfsd;

    subrutina $hola(){
        entrada();
        mientras($tres>$cuatro){
            $gfds = entrada();
        }
    }

    subrutina $gfdgfds(){
        /*La siguiente linea dara el error 221 ya que no se puso un
condición en si*/
        si(){
            salida(432423);
        }
        sino{
            entrada();
        }
    }

    /*En la siguiente linea marcara el error 214 ya que no se incluyo el
procedimiento principal*/
    () {
        entrada();
        /*En la siguiente linea marcara el error 218 ya que se esperaba
realizar algo con la variable*/
        $fdafdsa
```

```
        }
}
```

AUTOMATAS

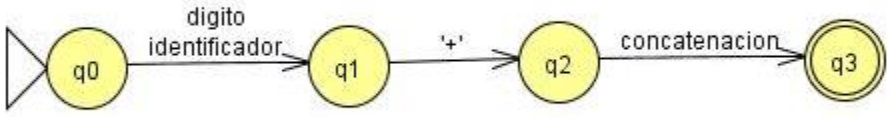
ANÁLISIS LÉXICO

○ ASIGNACIÓN



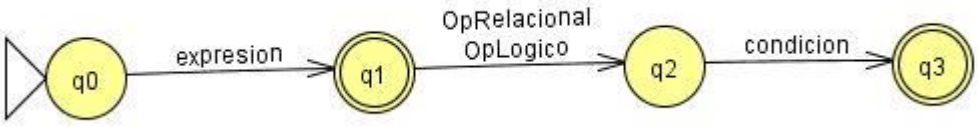
	identificador	=	entrada	expresión	salida	;
→Q0	Q1					
Q1		Q2				
Q2			Q3	Q3	Q3	
Q3						Q4
*Q4						

○ CONCATENACIÓN



	digito	identificador	+	concatenación
→Q0	Q1	Q1		
Q1			Q2	
Q2				Q3
*Q3				

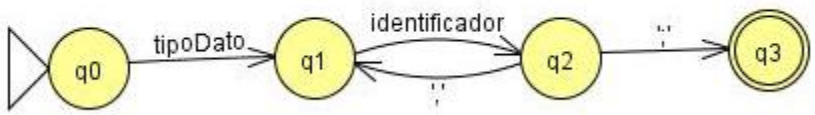
○ EXPRESIÓN



	expresión	Operacional	OpLogico	Condición
→Q0	Q1			

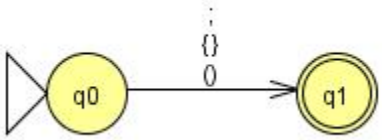
*Q1		Q2	Q2	
Q2				Q3
*Q3				

- TIPO DE DATO



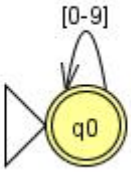
	tipoDato	identificador	;
→Q0	Q1		
Q1		Q2	
Q2			Q1,Q3
*Q3			

- DELIMITADORES



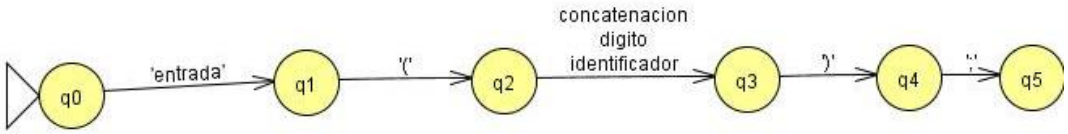
	{}	()
→Q0	Q1	Q1
*Q1		

- DÍGITOS



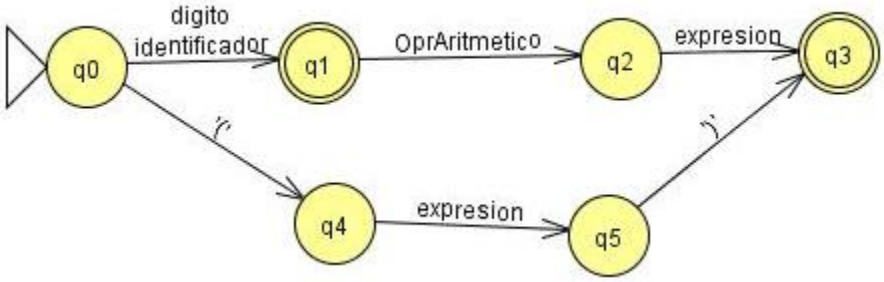
	0-9
→*Q0	Q0

- ENTRADAS



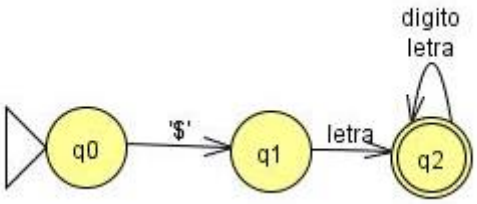
	entrada	(	concatenacion	digito	identificador	)	;
→Q0	Q1						
Q1		Q2					
Q2			Q3	Q3	Q3		
Q3						Q4	
Q4							Q5
*Q5							

- EXPRESIÓN



	digito	identificador	OprAritmetico	expresion	(	)
→Q0	Q1	Q1			Q4	
*Q1			Q2			
Q2				Q3		
*Q3						
Q4				Q5		
Q5						Q3

- IDENTIFICADOR

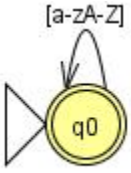


	$	letra	Digito
→Q0	Q1		
Q1		Q2	
*Q2		Q2	Q2

- INSTRUCCIÓN

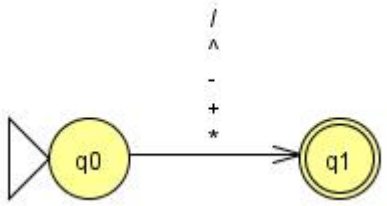
	salida	estructuraDecision	entrada	estructuraCiclo	asignacion	expresion
→Q0	Q1	Q1	Q1	Q1	Q1	Q1
*Q1						

- LETRAS



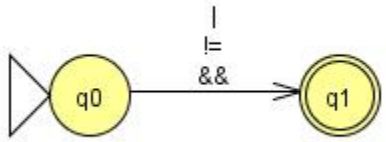
	a-zA-Z
→*Q0	Q0

- OPERADORES ARITMÉTICOS



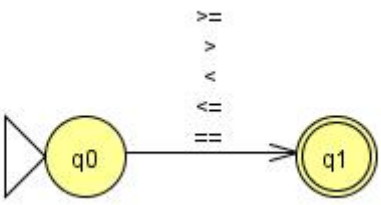
	/	^	-	+	*
*Q0	Q1	Q1	Q1	Q1	Q1
→Q1					

- OPERADORES LÓGICOS



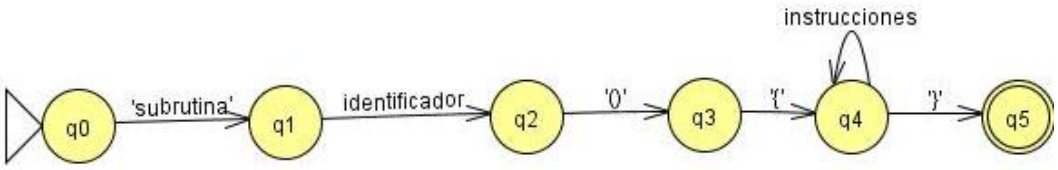
	|	¡=	&&
→Q0	Q1	Q1	Q1
*Q1			

- OPERADORES RELACIONALES



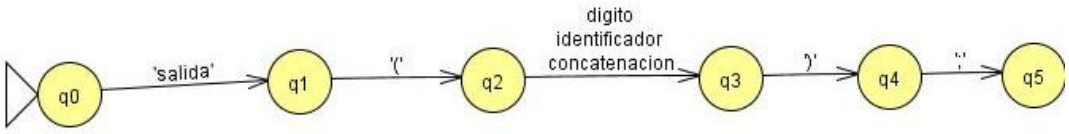
	>=	>	<	<=	==
→Q0	Q1	Q1	Q1	Q1	Q1
*Q1					

- SUBRUTINA



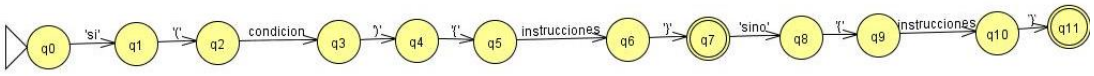
	subrutina	identificador	()	{	instruccion	}
→Q0	Q1					
Q1		Q2				
Q2			Q3			
Q3				Q4		
Q4					Q4	Q5
*Q5						

- SALIDA



	salida	(	digito	identificador	concatenacion	)	;
→Q0	Q1						
Q1		Q2					
Q2			Q3	Q3	Q3		
Q3						Q4	
Q4							Q5
*Q5							

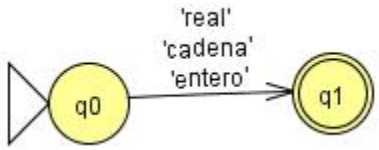
- ESTRUCTURA DE DECISIÓN



	si	(	condicion	)	{	instruccion	}	sino
→Q0	Q1							
Q1		Q2						
Q2			Q2					
Q3				Q4				
Q4					Q5			
Q5						Q6		
Q6							Q7	

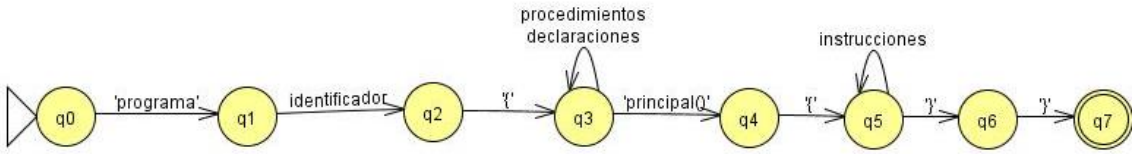
*Q7								Q8
Q8					Q9			
Q9						Q10		
Q10							Q11	
*Q11								

- TIPO DE DATO



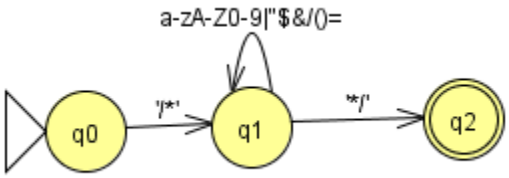
	real	cadena	entero
→Q0	Q1	Q1	Q1
*Q1			

- PROGRAMA



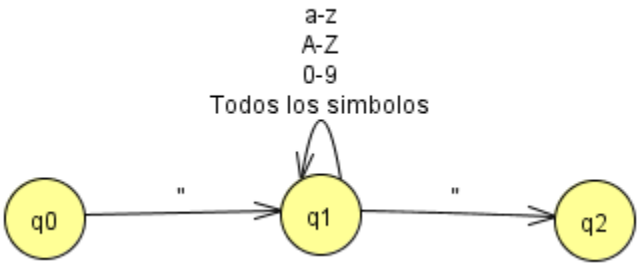
	programa	identificador	{	procedimientos	declaraciones	principal()	}	instrucciones
→Q0	Q1							
Q1		Q2						
Q2			Q3					
Q3				Q3	Q3	Q4		
Q4							Q5	
Q5							Q6	Q5
Q6							Q7	
*Q8								

- COMENTARIO



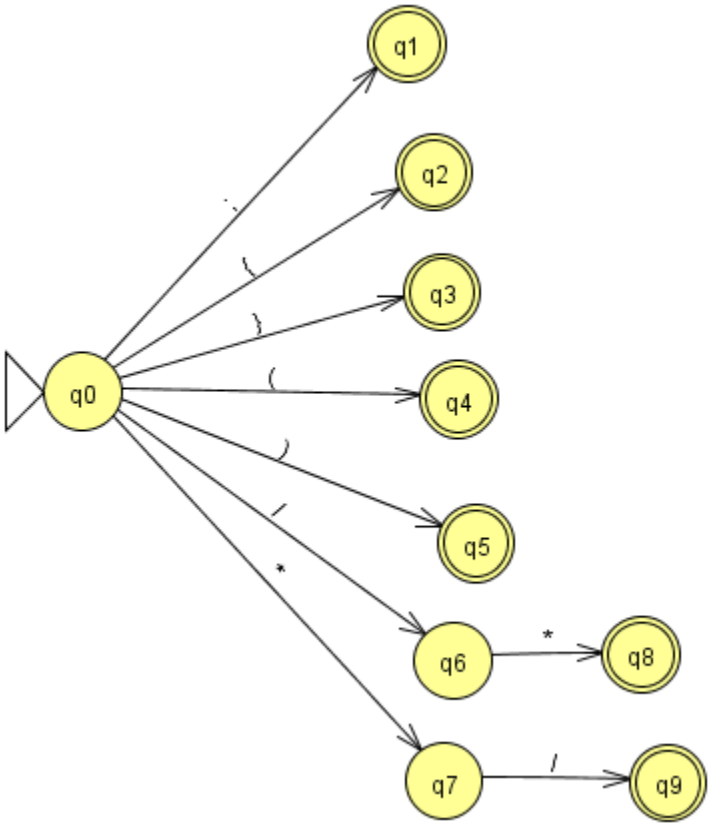
	/*	a-zA-Z0-9|"$&/()=	*/
→Q0	Q1		
Q1		Q1	Q2
*Q2			

- CADENA



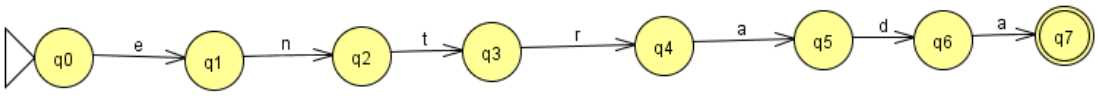
	“	a-zA-Z0-9|"$&/()=	“
→Q0	Q1		
Q1		Q1	Q2
*Q2			

- DELIMITADORES



	;	{	}	(	)	/	*
→Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
*Q1							
*Q2							
*Q3							
*Q4							
*Q5							
Q6							Q8
Q7						Q9	
*Q8							
*Q9							

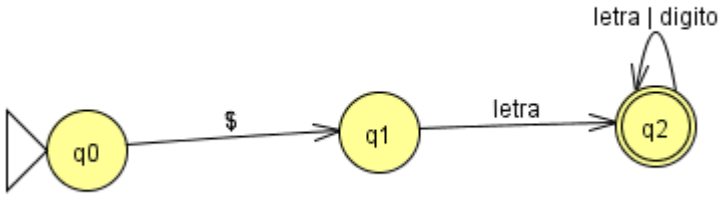
- ENTRADA



	e	n	t	r	a	d
→Q0	Q1					

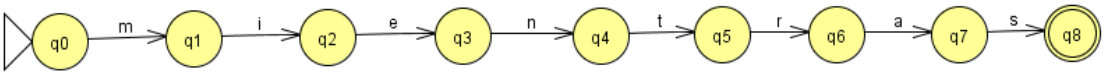
Q1		Q2				
Q2			Q3			
Q3				Q4		
Q4					Q5	
Q5						Q6
Q6					Q7	
*Q7						

- ID



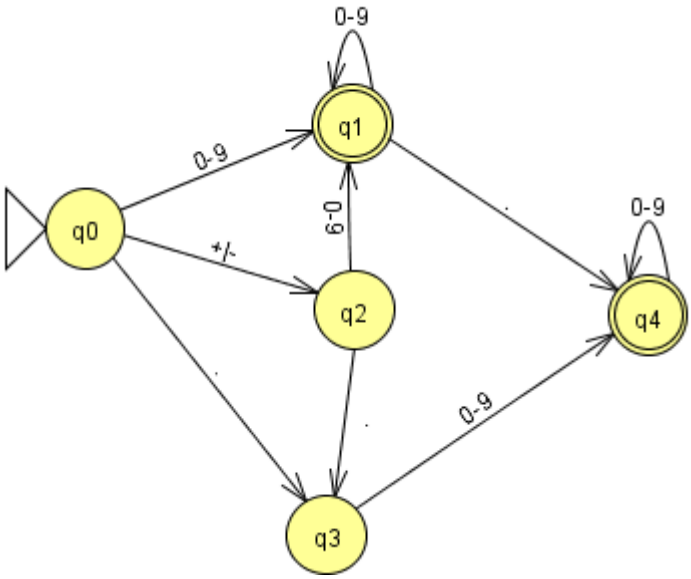
	$	letra	digito
→Q0	Q1		
Q1		Q2	
*Q2		Q2	Q2

- MIENTRAS



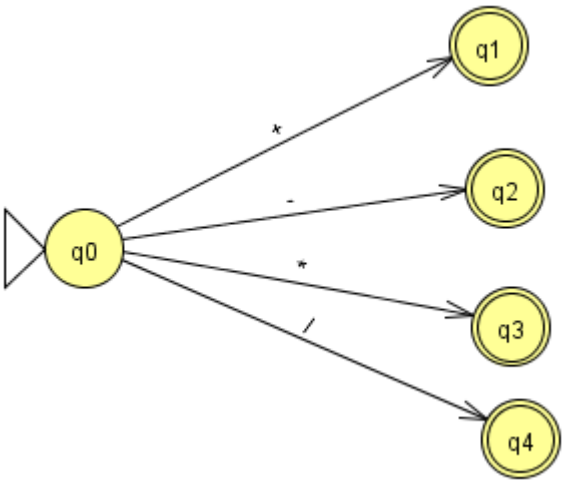
	m	i	e	n	t	r	a	s
→Q0	Q1							
Q1		Q2						
Q2			Q3					
Q3				Q4				
Q4					Q5			
Q5						Q6		
Q6							Q7	
Q7								Q8
*Q8								

- NÚMEROS



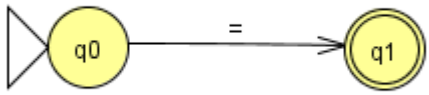
	0-9	+	-	.
→Q0	Q1	Q2	Q2	Q3
*Q1	Q1			Q4
Q2	Q1			Q3
Q3	Q4			
*Q4	Q4			

- OPERACIONES ARITMÉTICAS



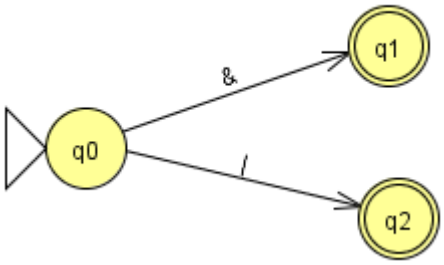
	+	-	*	/
→Q0	Q1	Q2	Q3	Q4
*Q1				
*Q2				
*Q3				
*Q4				

- OPERACIÓN ASIGNACIÓN



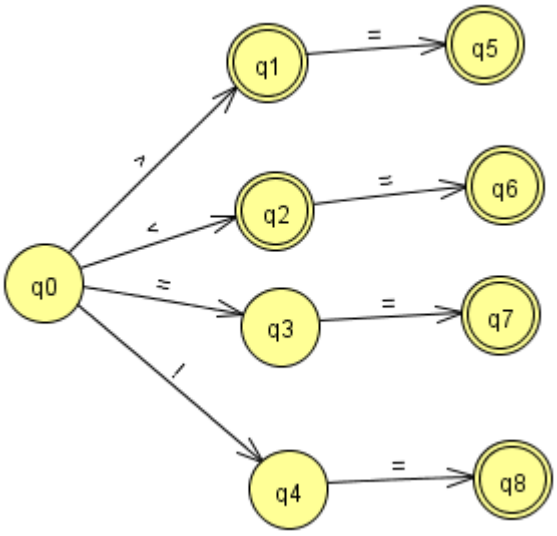
	=
→Q0	Q1
*Q1	

- OPERADORES LÓGICOS



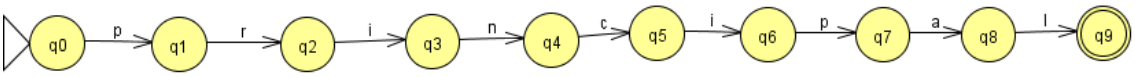
	&	|
→Q0	Q1	Q2
*Q1		
*Q2		

○ OPERADORES RELACIONALES



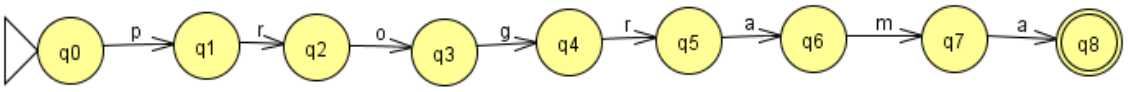
	>	<	=	¡
→Q0	Q1	Q2	Q3	Q4
*Q1			Q5	
*Q2			Q6	
Q3			Q7	
Q4			Q8	
*Q5				
*Q6				
*Q7				
*Q8				

○ PRINCIPAL



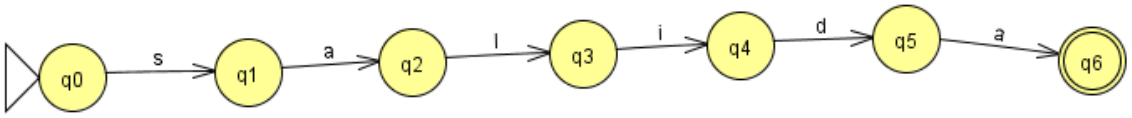
	p	r	i	n	c	a	l
→Q0	Q1						
Q1		Q2					
Q2			Q3				
Q3				Q4			
Q4					Q5		
Q5			Q6				
Q6	Q7						
Q7						Q8	
Q8							Q9
*Q9							

- PROGRAMA



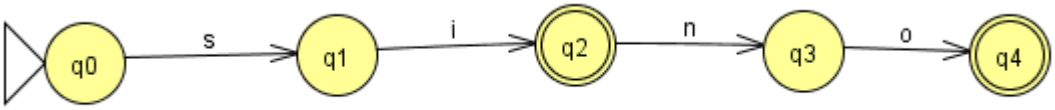
	p	r	o	g	a	m
→Q0	Q1					
Q1		Q2				
Q2			Q3			
Q3				Q4		
Q4		Q5				
Q5					Q6	
Q6						Q7
Q7					Q8	
Q8						

- SALIDA



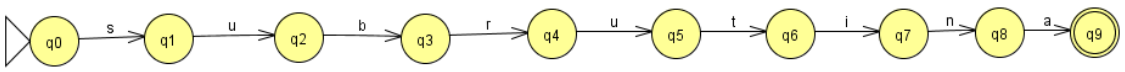
	s	a	l	i	d
→Q0	Q1				
Q1		Q2			
Q2			Q3		
Q3				Q4	
Q4					Q5
Q5		Q6			
*Q6					

- SI-SINO



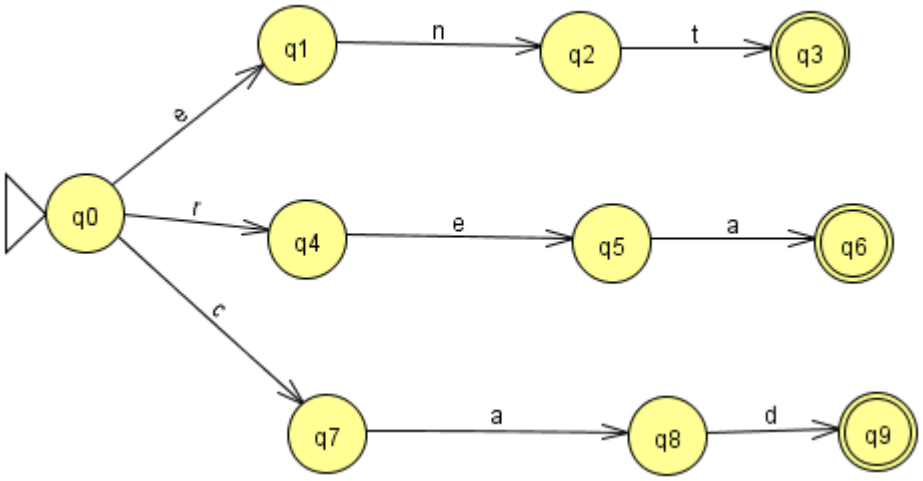
	s	i	n	o
→Q0	Q1			
Q1		Q2		
*Q2			Q3	
Q3				Q4
*Q4				

- SUBRUTINA



	s	u	b	r	t	i	n	a
→Q0	Q1							
Q1		Q2						
Q2			Q3					
Q3				Q4				
Q4		Q5						
Q5					Q6			
Q6						Q7		
Q7							Q8	
Q8								Q9
*Q9								

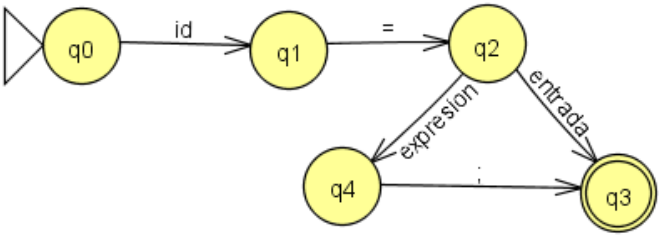
- TIPO DE DATO



	e	n	t	r	a	c	d
→Q0	Q1			Q4		Q7	
Q1		Q2					
Q2			Q3				
*Q3							
Q4	Q5						
Q5					Q6		
*Q6							
Q7					Q8		
Q8							Q9
*Q9							

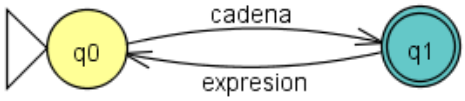
ANÁLISIS SINTÁCTICO

○ ASIGNACIÓN



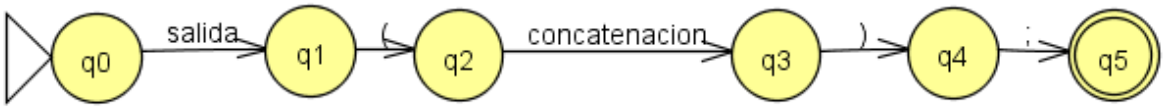
	id	=	entrada	expresión	;
→Q0	Q1				
Q1		Q2			
Q2			Q3	Q4	
*Q3					
Q4					Q3

○ CONCATENACIÓN



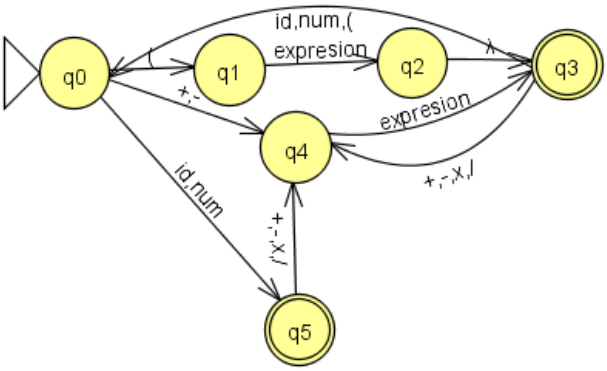
	cadena	expresion
→Q0	Q1	
*Q1		Q0

○ SALIDA



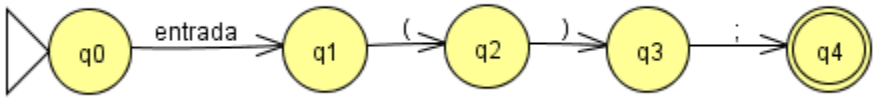
	salida	(	concatenacion	)	;
→Q0	Q1				
Q1		Q2			
Q2			Q3		
Q3				Q4	
Q4					Q5
*Q5					

- EXPRESIÓN



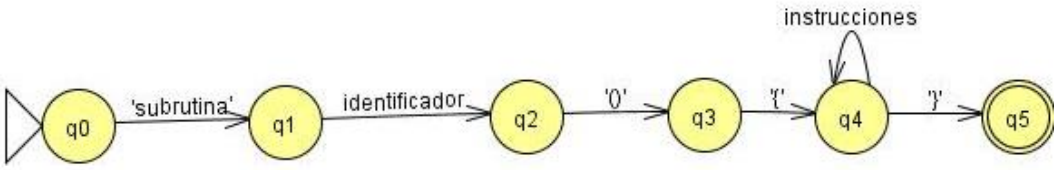
	expresión	(	+,-	*,/	id	num	
→Q0		Q1	Q4		Q5	Q5	
Q1	Q2	Q2	Q2				
Q2					Q3		Q3
*Q3		Q0	Q4	Q4	Q0	Q0	
Q4	Q3						
*Q5			Q4	Q4			

- ENTRADAS



	entrada	(	)	;
→Q0	Q1			
Q1		Q2		
Q2			Q3	
Q3				Q4
*Q4				

- SUBRUTINA

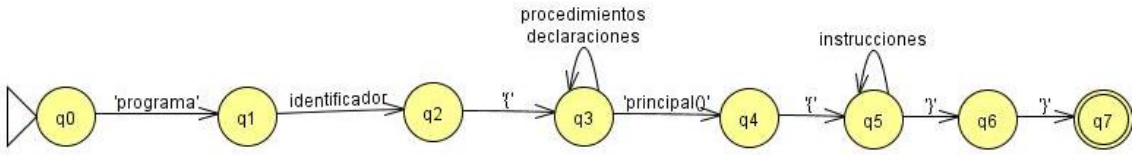


	subrutina	identificador	()	{	instruccion	}
→Q0	Q1					
Q1		Q2				
Q2			Q3			
Q3				Q4		
Q4					Q4	Q5
*Q5						

- ESTRUCTURA DE DECISIÓN

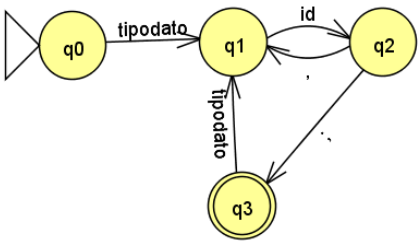
	si	(	condicion	)	{	instruccion	}	sino
→Q0	Q1							
Q1		Q2						
Q2			Q2					
Q3				Q4				
Q4					Q5			
Q5						Q6		
Q6							Q7	
*Q7								Q8
Q8					Q9			
Q9						Q10		
Q10							Q11	
*Q11								

- PROGRAMA



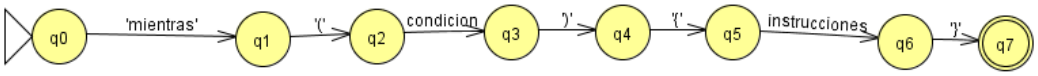
	programa	identificador	{	procedimientos	declaraciones	principal()	}	instrucciones
→Q0	Q1							
Q1		Q2						
Q2			Q3					
Q3				Q3	Q3	Q4		
Q4							Q5	
Q5							Q6	Q5
Q6							Q7	
*Q8								

- DECLARACIONES



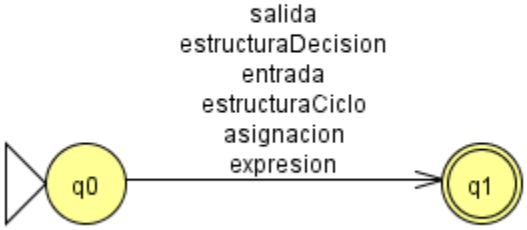
	tipodato	id	;	,
→Q0	Q1			
Q1		Q2		
Q2			Q3	Q1
*Q3	Q1			

○ ESTRUCTURA DE CICLO



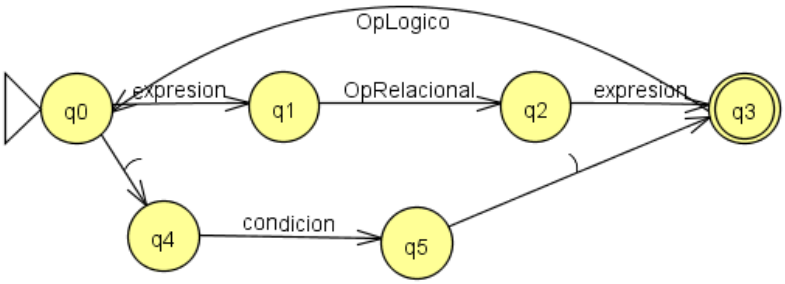
	mientras	(	condicion	)	{	instrucciones	}
→Q0	Q1						
Q1		Q2					
Q2			Q2				
Q3				Q4			
Q4					Q5		
Q5						Q6	
Q6							Q7
*Q7							

○ INSTRUCCIONES



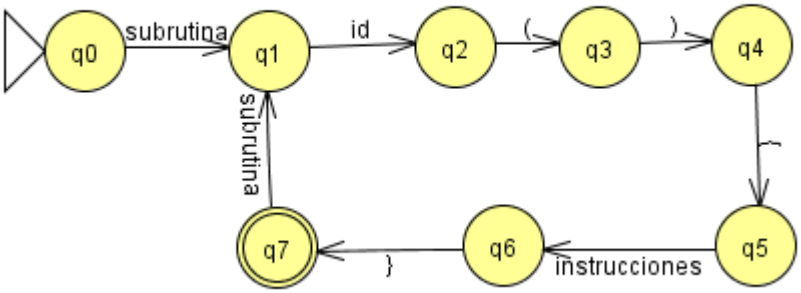
	Salida	estructuraDecision	Entrada	estructuraCiclo	Asignación	Expresión
→Q0	Q1	Q1	Q1	Q1	Q1	Q1
*Q1						

- CONDICIÓN



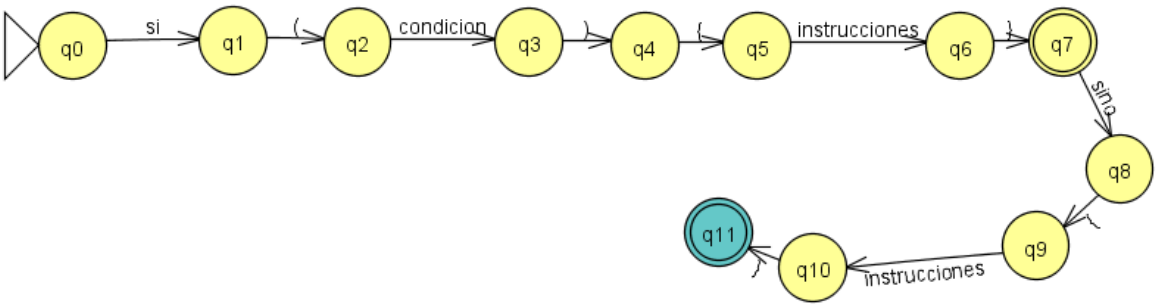
	expresion	OpRelacional	OpLogico	(	condicion	)
→Q0	Q1			Q4		
Q1		Q2				
Q2	Q3				Q3	
*Q3			Q0			
Q4					Q5	
Q5						Q3

- PROCEDIMIENTOS



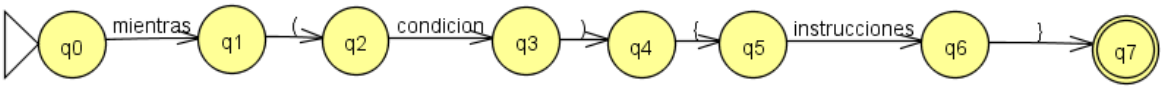
	subrutina	id	(	)	{	}	instrucciones
→Q0	Q1						
Q1		Q2					
Q2			Q3				
Q3				Q4			
Q4					Q5		
Q5							Q6
Q6						Q7	
*Q7	Q1						

- SI



	si	condicion	(	)	{	}	instrucciones	sino
→Q0	Q1							
Q1			Q2					
Q2		Q3						
Q3				Q4				
Q4					Q5			
Q5							Q6	
Q6						Q7		
*Q7								Q8
Q8					Q9			
Q9							Q10	
Q10						Q11		
*Q11								

- MIENTRAS



	Mientras	condicion	(	)	{	}	instrucciones	sino
→Q0	Q1							
Q1			Q2					
Q2		Q3						
Q3				Q4				
Q4					Q5			
Q5							Q6	
Q6						Q7		
*Q7								

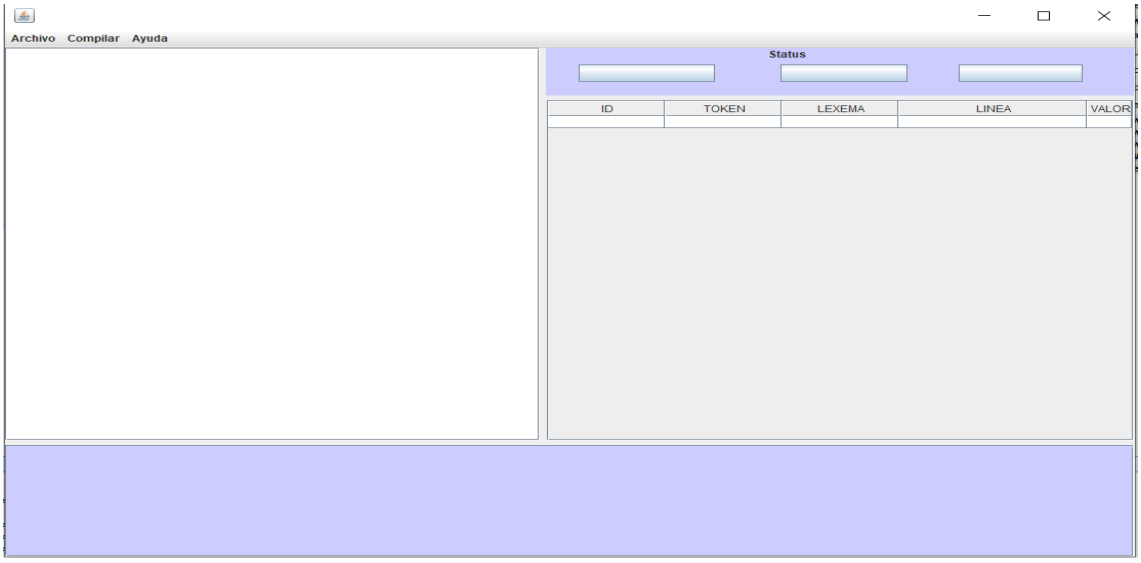
EJECUCIÓN FINAL DEL PROYECTO

COMPROBACIÓN DE RESULTADOS A PARTIR DE PRUEBAS DE LOS CASOS DE USO

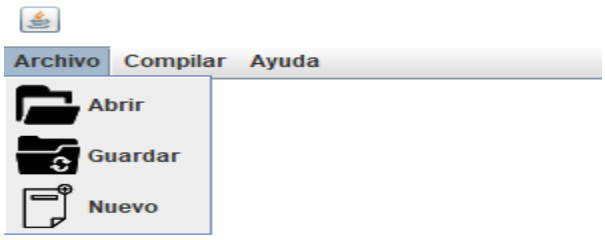
Para ejemplificar el comportamiento de nuestro Pseudocompilador en la implementación final veremos varios ejemplos referentes a los casos de errores en los diferentes análisis implementados, los cuales son:

- Análisis léxico
- Análisis sintáctico
- Análisis semántico

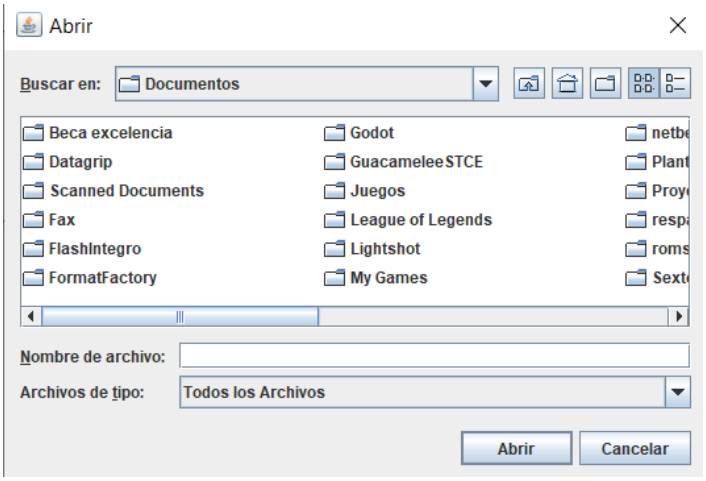
Primeramente, explicaremos como funciona nuestra interfaz de usuario, la vista principal es la siguiente:



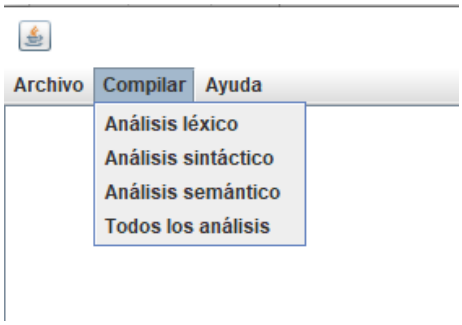
Para crear, guardar o abrir un archivo nos dirigimos a las siguientes opciones:



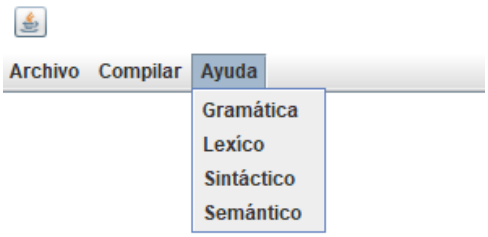
Se nos desplegaran ventanas como las siguientes al hacer uso de abrir y de guardar, al hacer uno nuevo se vacía nuestro espacio de código.



Al realizar la compilación, lo podemos hacer por partes, si decidimos hacer uso de un único análisis, si el código cuenta con un error, únicamente se mostrarán los errores referentes al análisis realizado, si se realizan todos los análisis se muestra todos los errores que surjan.

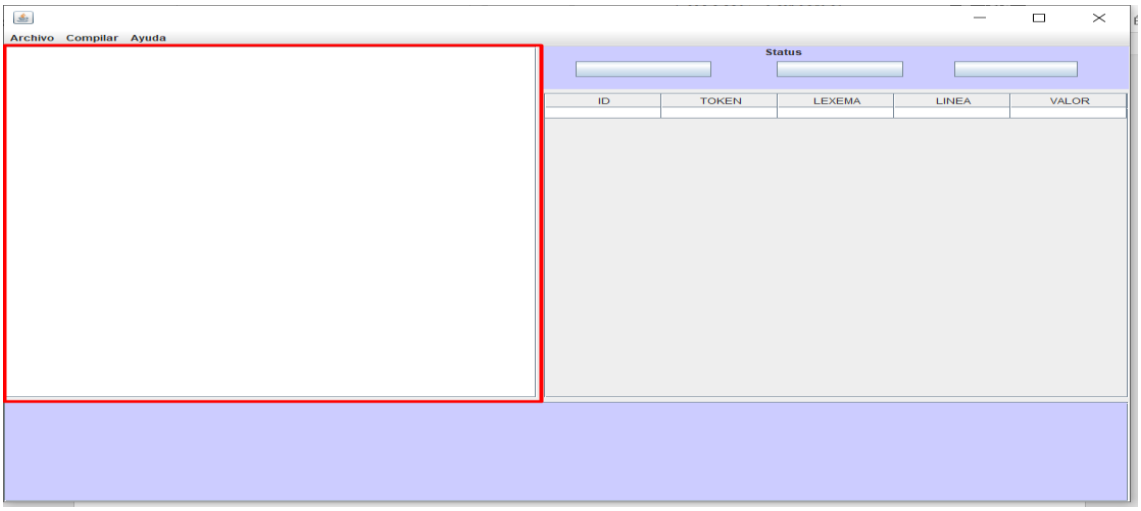


Mediante la opción ayuda se desplegarán los PDF referentes a la actividad realizada en cuestión, siendo gramática la actividad número 1, léxico la actividad número dos y de esta manera sucesivamente.



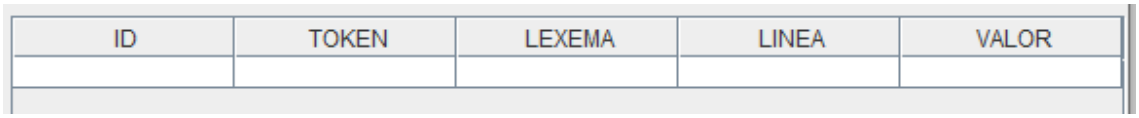
El PDF se desplegará en nuestro programa visualizador de documentos PDF preferido o en dado caso nos preguntará donde visualizarlo.

En el espacio marcado en color rojo es el espacio dedicado para escribir nuestro código de nuestros programas

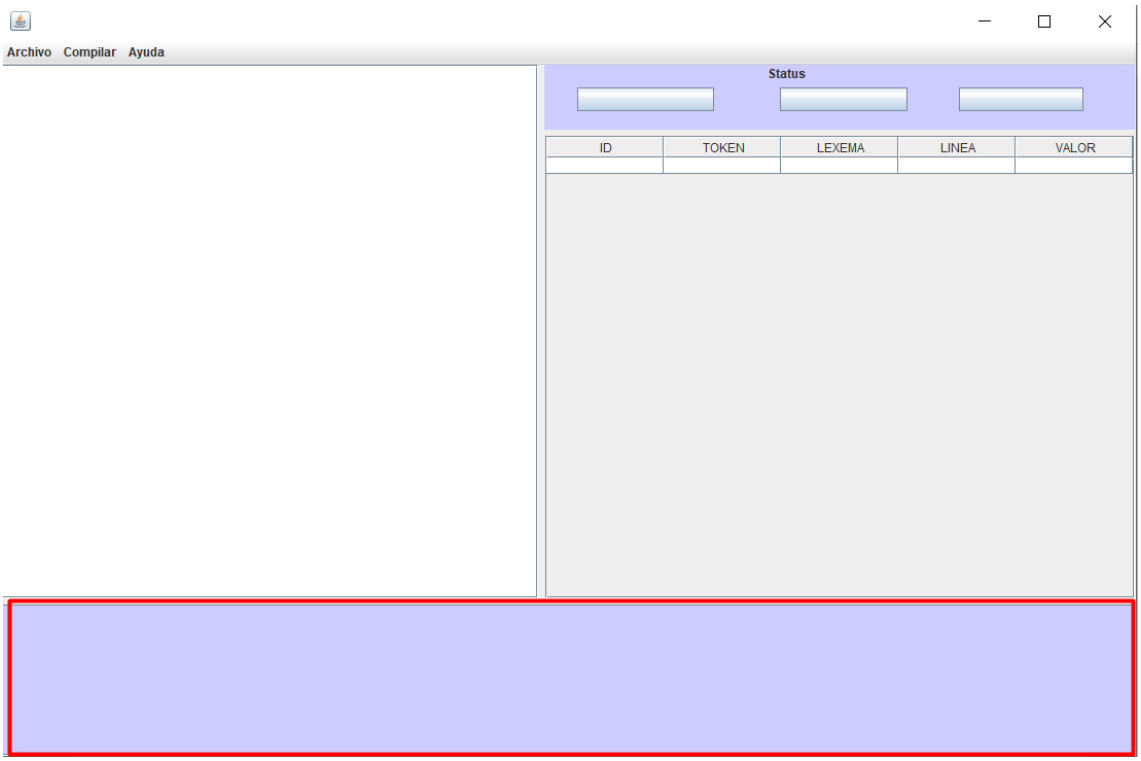


El espacio designado con la palabra "status", es donde nuestro semáforo marca si fue compilado exitosamente mostrando un color verde al final o si genero algún error, mostrando el color rojo.

Donde vemos una tabla únicamente con las cabeceras se mostrará nuestra tabla de símbolos una vez se realice algún análisis.



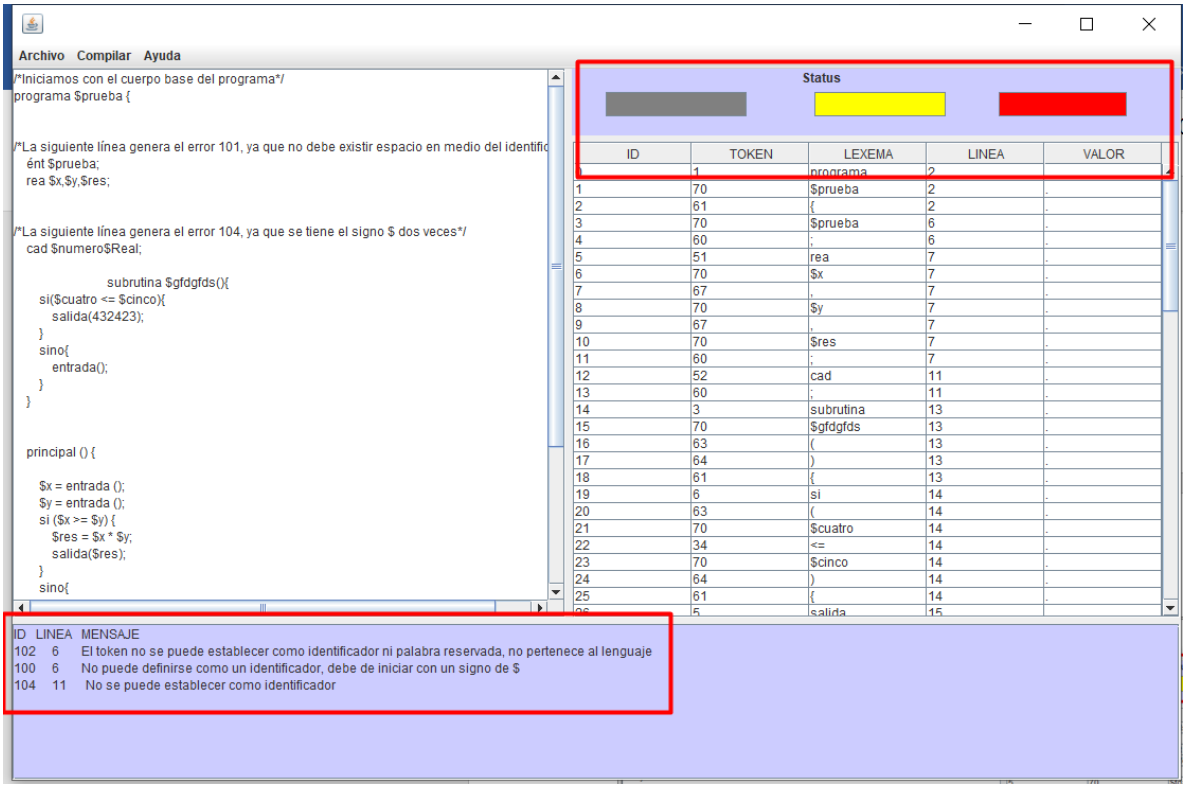
Por último, en el parte inferior marcado de rojo, es nuestra "consola" donde se muestran los mensajes que deban ser desplegados al usuario o en dado caso los errores que se generaron.



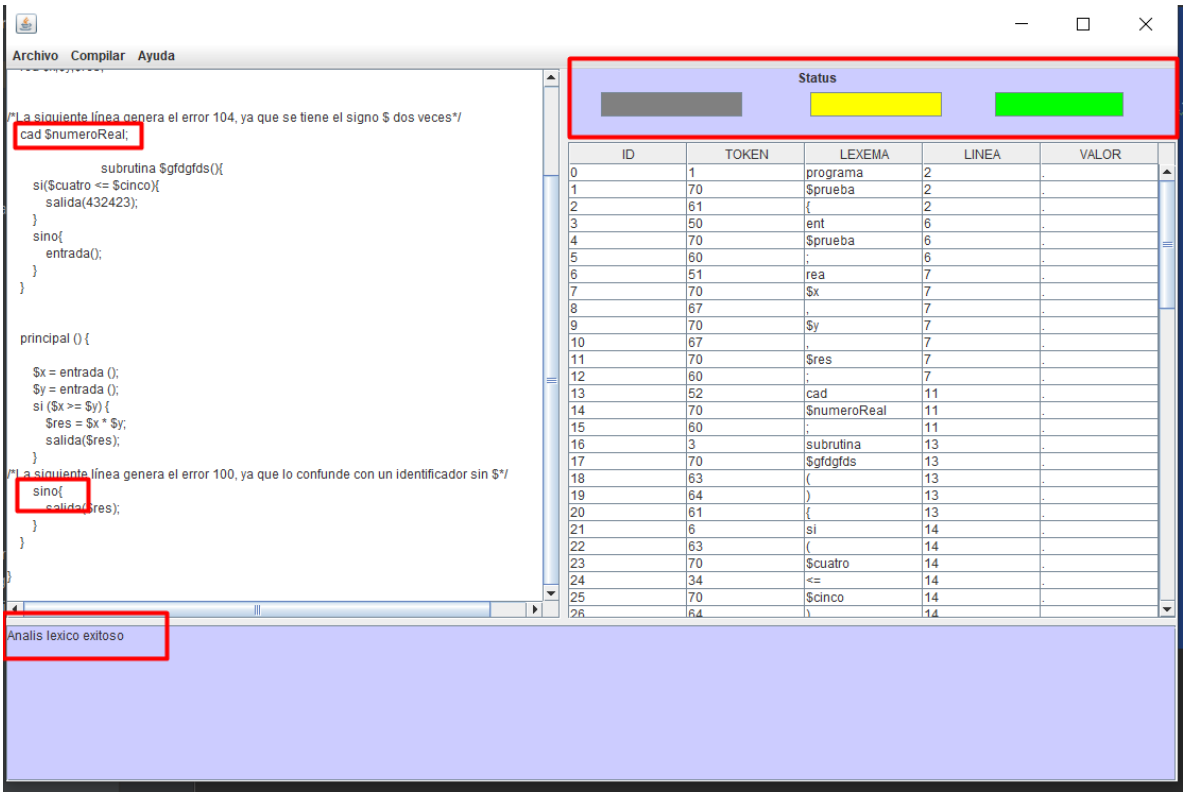
EJECUCIÓN DEL ANÁLISIS LÉXICO

El código ejecutado es el siguiente:

```
/*Iniciamos con el cuerpo base del programa*/
programa $prueba {


/*La siguiente línea genera el error 101, ya que no debe existir espacio
en medio del identificador*/
    ént $prueba;
    rea $x,$y,$res;


/*La siguiente línea genera el error 104, ya que se tiene el signo $ dos
veces*/
    cad $numero$Real;

        subrutina $gfdgfds(){
        si($cuatro <= $cinco){
            salida(432423);
        }
        sino{
            entrada();
        }
    }


    principal () {

        $x = entrada ();
        $y = entrada ();
        si ($x >= $y) {
            $res = $x * $y;
            salida($res);
        }
        sino{
            salida($res);
        }

/*La siguiente línea genera el error 100, ya que lo confunde con un
identificador sin $*/




    }

}
```



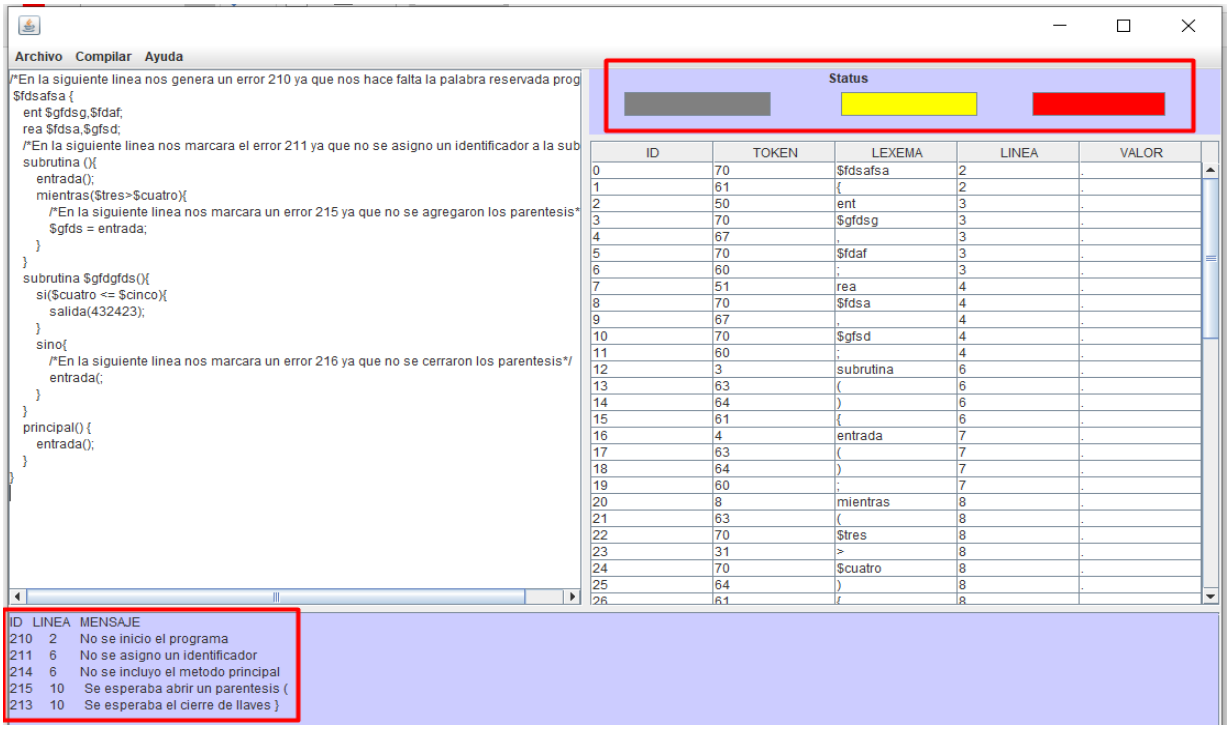
Al corregir los errores nuestro compilador funciona correctamente.



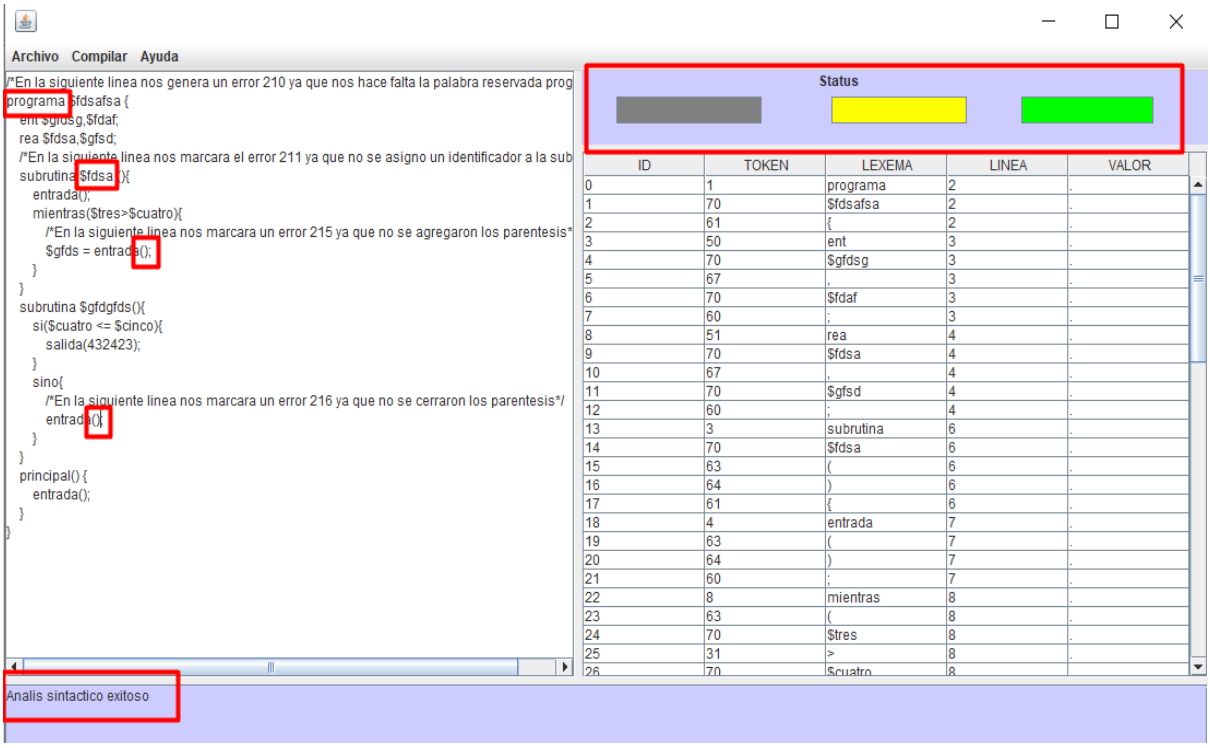
EJECUCIÓN DEL ANÁLISIS SINTÁCTICO

El código ejecutado es el siguiente:

```
/*En la siguiente linea nos genera un error 210 ya que nos hace falta la
palabra reservada programa*/
 $fdsafsa {
    ent $gfdsg,$fdaf;
    rea $fdsa,$gfsd;



    /*En la siguiente linea nos marcara el error 211 ya que no se asigno
un identificador a la subrutina*/
    subrutina (){
        entrada();
        mientras($tres>$cuatro){
            /*En la siguiente linea nos marcara un error 215 ya que no se
agregaron los parentesis*/
            $gfds = entrada;
        }
    }



    subrutina $gfdgfds(){
        si($cuatro <= $cinco){
            salida(432423);
        }
        sino{
            /*En la siguiente linea nos marcara un error 216 ya que no se
cerraron los parentesis*/
            entrada(;
        }
    }


    principal() {
        entrada();
    }
}
```



Al corregir los errores nuestro compilador funciona correctamente.

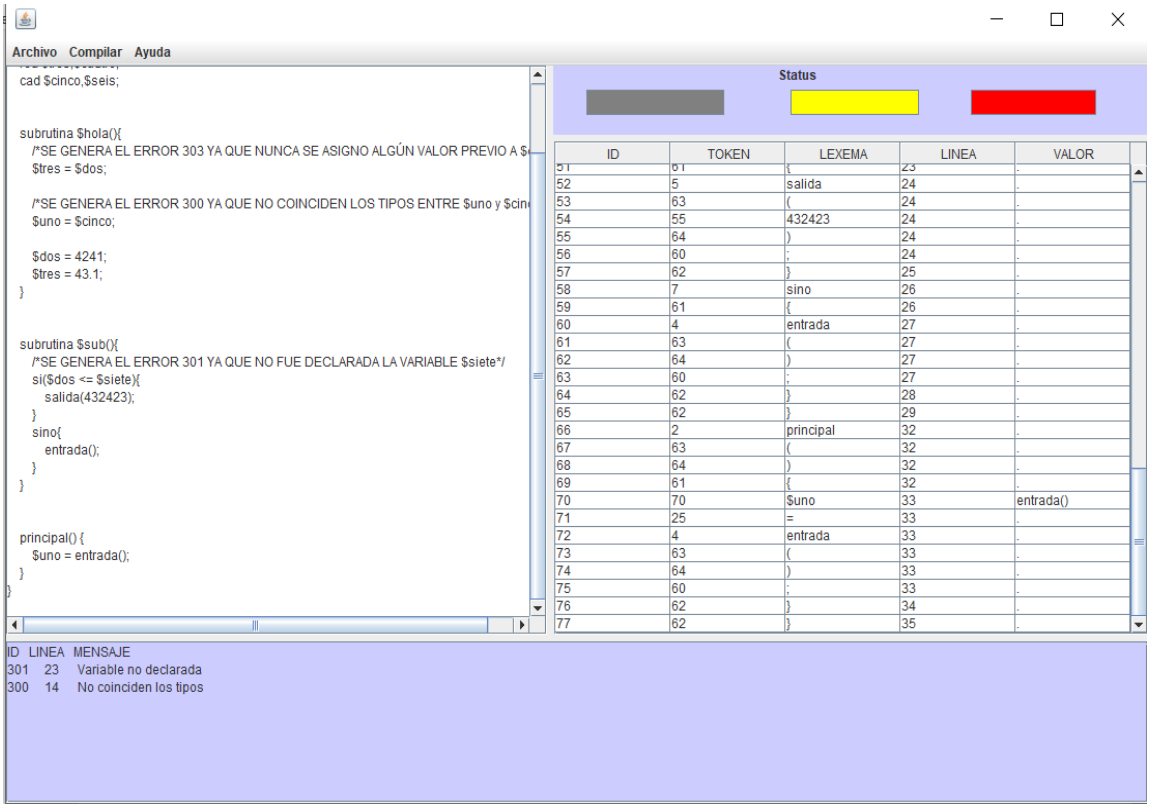


En el proceso de corrección nos encontramos con nuevos errores, pero si solucionamos el principal lo normal es que todos los demás se solucionen correctamente.

EJECUCIÓN DEL ANÁLISIS SEMÁNTICO

El código ejecutado es el siguiente:

```
programa $prueba {

    /*AQUI ES DONDE DEBEMOS DECLARAR TODAS LAS VARIABLES*/
    ent $uno,$dos;
    rea $tres,$cuatro;
    cad $cinco,$seis;


    subrutina $hola(){
        /*SE GENERA EL ERROR 303 YA QUE NUNCA SE ASIGNO ALGÚN VALOR
PREVIO A $dos*/
        $tres = $dos;

        /*SE GENERA EL ERROR 300 YA QUE NO COINCIDEN LOS TIPOS ENTRE $uno
y $cinco*/
        $uno = $cinco;

        $dos = 4241;
        $tres = 43.1;
    }


    subrutina $sub(){
        /*SE GENERA EL ERROR 301 YA QUE NO FUE DECLARADA LA VARIABLE
$siete*/
        si($dos <= $siete){
            salida(432423);
        }
        sino{
            entrada();
        }
    }


    principal() {
        $uno = entrada();
    }
}
```



Al corregir los errores nuestro compilador funciona correctamente.

BITACORA DE INCIDENCIAS ANÁLISIS LEXÍCO/SINTÁCTICO			
Fecha	**Hora**	**Descripción de la incidencia**	**Solución**
Martes 25 de Febrero del 2020	7:00 AM	falta establecer algunos puntos de la notación formal del lenguaje , el profesor realizo las observaciones pertinentes en clase al momento de exponer el trabajo	El equipo se dio a la tarea de reunirse para integrar mediante una notación formal (BNF) los elementos de su lenguaje estos ya habían sido establecidos en la actividad número 1 . Pero faltaba ser mas explícitos en el caso de las expresiones así que se delegaron tareas a cada integrante
Jueves 27 de Febrero del 2020	8:00 AM	se solicitó elaborar la tabla de símbolos del lenguaje prototipo y la pila de errores	Los miembros del equipo se reunieron para discutir de manera pertinente los elementos que contendrían dichas estructuras, cada uno aporto conocimiento mediante una lluvia de ideas y una vez que se estableció esto se delegaron las tareas para formalizar lo que se había comentado en la reunión de trabajo.
Martes 3 de Marzo del 2020	8:00 AM	falta elaborar los autómatas correspondientes a las expresiones regulares y el maestro de igual manera solicito implementar las matrices de estado correspondientes a cada autómata . se realizaron unas pequeñas observaciones en la tabla de símbolos respecto a los campos	se descargo el software JFLAP para elaborar en este nuestros autómatas ,cada miembro realizo la instalación ya que se repartieron de manera equitativa dichos autómatas , esto fue una tarea sencilla ya que como se tenia bien formalizada la gramática y las expresiones regulares simplemente se diseñaron los autómatas correspondientes y se evaluaron en el mismo software , las pequeñas observaciones respecto a la tabla de símbolos se realizaron al momento en la misma reunión que se tuvo entre los miembros del equipo después de la revisión del profesor.

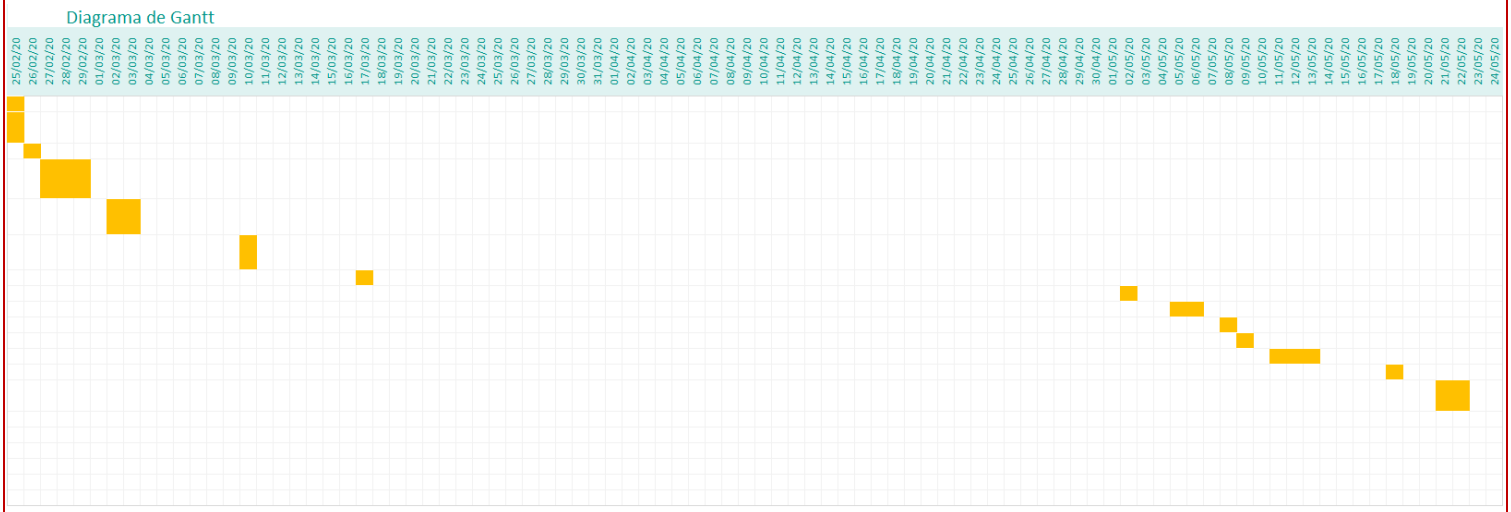
Martes 10 de Marzo del 2020	8:00 AM	se solicitó la elaboración del modelado uml del analizador léxico , y plasmar la manera en la que interactúan los elementos de este análisis en nuestros diagramas previos a la programación , también diseñar algunos programas muestra y plasmar los casos de uso en cuanto a los errores léxicos	Para la elaboración del modelado se fijó una reunión y prácticamente lo que se realizo fue bajo una metodología SCRUM
Martes 17 de Marzo del 2020	12:00 AM	se emitió un comunicado por parte de la escuela en donde informaba a cerca de la suspensión de labores a partir del 20 de marzo debido a la pandemia por el covid-19 , justo después de que el profesor publicara la solicitud de la actividad 2	El equipo el día martes asistió a la institución para tratar temas relacionados con la actividad ante la situación y se tomaron cartas en el asunto , Se decidió seguir trabajando bajo el mismo enfoque de esta metodología ágil de desarrollo , establecer un tablero con sprints para la planificación de las actividades semanales , delegar tareas priorizarlas y delegar roles para el desarrollo del software.
Viernes 2 de mayo del 2020	7:00 PM	La forma de trabajo que se iba utilizar para realizar la actividad a distancia debido al confinamiento.	El equipo decidió organizarse por mensaje y ciertos días se haría reuniones a través de videollamadas para desarrollar correctamente esta y cumplir todos sus aspectos

Martes 5 de mayo del 2020	7:00 PM	Se solicito realizar algunos cambios a la GUI con base a un esquema enviado por el profesor.	El equipo se reunió para decidir qué contendría la GUI para ser más amigable al usuario y mejor organizada para observar todos los elementos importantes que debía mostrarse al usuario.
Viernes 8 de mayo del 2020	4:00 PM	Revisando la gramática que se tenía se pude concluir que algunos autómatas no eran correctos.	Se realizo una revisión de la gramática y de cada uno de los autómatas enfocados en el análisis sintáctico y se realizó la modificación de estos para llevar a cabo una correcta revisión
Sábado 9 de mayo del 2020	8:00A M	Se tenían dudas acerca de como sería la implementación del programa	Se realizo una reunión donde se quedo de acuerdo a realizar las funcionalidades por un lado y la interfaz por otro para finalizando juntar las partes.
Lunes 11 de mayo de 2020	12:00 PM	Se tuvo problema el unir la GUI y el programa	Se reviso y se procedió a realizar la unión de la GUI y el programa, realizando un pequeño cambio de IDE seleccionando NetBeans por su experiencia en su uso.

GENERACIÓN DEL CRONOGRAMA DE PROYECTO.

A continuación, se muestran las tareas y la duración de las mismas, así como las respectivas fechas de inicio y finalización de cada tarea en ejecución para el desarrollo del proyecto desde su primera hasta última etapa.

Descripción de la etapa	Duración de la etapa (días)	Tarea dependiente	Tipo de Dependencia	Días de dependencia	Comienzo	Fin	Responsable	Estatus	Fecha de finalización	Días que efectivamente llevó la etapa
Reunión Inicial	1	No Aplica	No Aplica	+0	25/02/20	25/02/20	Alina	Completado	25/02/20	1
Creación de el lenguaje formal en BNF	1	1	FC	+0	25/02/20	25/02/20	all	Completado	25/02/20	1
Evidencia numero 1	1	2	FC	+1	26/02/20	26/02/20	Alina	Completado	26/02/20	1
Elaboración de la tabla de símbolos y pila de errores	3	3	FC	+1	27/02/20	29/02/20	all	Completado	29/02/20	3
Elaboración de automates de las expresiones regulares	2	4	FC	+2	02/03/20	03/03/20	all	Completado	03/03/20	2
Modelado UML del analisador Léxico	1	5	FC	+7	10/03/20	10/03/20	all	Completado	10/03/20	1
Renion Emergencia Covid-19	1	6	FC	+7	17/03/20	17/03/20	all	Completado	17/03/20	1
Reunion en línea del equipo	1	7	FC	+46	02/05/20	02/05/20	all	Completado	02/05/20	2
Elabaoración GUI	2	8	FC	+3	05/05/20	06/05/20	all	Completado	06/05/20	2
Reelaboracion de automatas errone	1	9	FC	+2	08/05/20	08/05/20	all	Completado	08/05/20	2
Reunión implementacion	1	10	FC	+1	09/05/20	09/05/20	all	Completado	09/05/20	1
Unificacion GUI y programa	3	11	FC	+2	11/05/20	13/05/20	all	Completado	13/05/20	3
Reunion analisis semántico	1	12	FC	+5	18/05/20	18/05/20	all	Completado	18/05/20	1
Elaboración automátas para semantico	2	13	FC	+3	21/05/20	22/05/20	all	Completado	22/05/20	2
Reunion para el documento final	1	14	FC	+3	25/05/20	25/05/20	all	Completado	23/05/20	-1
codificacion de los automatas	3	15	FC	+7	01/06/20	03/06/20	all	Completado	03/06/20	3
Finalización del proyecto	2	16	FC	+1	04/06/20	05/06/20	all	Completado	06/06/20	3



La brecha de vemos con aparentemente inactividad entre las fechas del 18 de marzo al 30 de abril en parte se debe a diversos factores externos fuera de nuestro alcance que afectaron al flujo de trabajo (pandemia y vacaciones de semana santa) , asi como mencionaba anteriormente "aparentemente inactividad" porque al contrario si hubo flujo de trabajo solo que a un ritmo menos latente en comparación con las primeras y últimas semanas.

CONCLUSIONES

Durante la realización del compilador nos pudimos dar cuenta la importancia de llevar a cabo las fases anteriores correctamente pues si una de ellas fallaba o se realizaba de la manera incorrecta la siguiente fase no tendría éxito o efecto, es importante tener bien definido lo anterior para tener un compilador estable y funcional.

El análisis semántico es muy importante pues le da sentido a cada palaba e instrucción que se usa en un programa, nos ayuda a tener una relación con los datos que se tiene y no sea ambiguo.

Con este analizador nos damos cuenta de la importancia en la coherencia que debe tener nuestro lenguaje pues puede estar bien sintácticamente pero semánticamente puede tener errores claves que a la hora de ejecutar o tratar de entender no va a tener sentido y va a perder todo éxito de comunicación hablando de temas de lenguaje y no se va a ejecutar correctamente en términos de programación.

Durante la realización del proyecto se tuvieron complicaciones en la elección de errores que podría tener nuestro compilador, pero con la investigación y el entendimiento sobre el análisis semántico pudimos llegar a los errores que atendía esta etapa.

El trabajar como equipo nos fue de mucha utilidad para retroalimentarnos sobre temas que algún miembro no entendía y sobre todo con la dificultad de comunicación por la pandemia, pero supimos hacer frente a este problema como equipo y llevar a cabo el proyecto por buen camino.