

DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: **Solicitud de Actividades**

Celaya, Gto., 2018-05-07

**LENGUAJES Y AUTÓMATAS II**  
DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ

**ACTIVIDAD 4 ( VALOR 70 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.

1. TOMANDO EN CUENTA QUE LA SEGUNDA ETAPA DEL ANÁLIZADOR SINTÁCTICO SE HA 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 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. INVERTIGAR. ¿ 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, CREACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS PARA DETERMINAR SI TIENEN EL SENTIDO CORRECTO LOS ELEMENTOS INTEGRADOS COMO CÓDIGO FUENTE.

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 PROPOGAN.

TALES PROGRAMAS DEBEN ESTAR PERFECTAMENTE DOCUMENTADOS Y CORRELACIONADOS CON LOS CASOS DE ESTUDIO CORRESPONDIENTES.

5. CARACTERÍSTICAS QUE LA ACTIVIDAD 4 DEBE POSEER PARA CONSIDERARSE COMPLETA.
- A. UN ANÁLISIS LÉXICO EFICIENTE, LIBRE DE ERRORES Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.
  - B. UN ANÁLISIS SEMÁNTICO EFICIENTE, LIBRE DE ERRORES Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.
  - C. UNA TABLA DE SÍMBOLOS PERFECTAMENTE ESTRUCTURADA CON LA TOKENIZACIÓN APROPIADA Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.
  - D. PLANTEAMIENTO Y FUNDAMENTACIÓN DE LOS CASOS DE USO DEL ANALIZADOR SINTÁCTICO, ASI COMO LOS ERRORES EN QUE DERIVARÁ CADA UNO DE ELLOS.
  - E. PLANTEAMIENTO Y FUNDAMENTACIÓN DE LOS CASOS DE USO DEL ANALIZADO SEMÁNTICO, ASI COMO LOS ERRORES EN QUE DERIVARÁ CADA UNO DE ELLOS.
  - F. 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.
  - G. GENERACIÓN DE UN CATÁLOGO DE ERRORES, CORRELACIONADO A LOS CASOS DE USO PROPUESTOS.
  - H. DISEÑO DE UNA SOLUCIÓN MODELADA EN OBJETOS, APOYADA EN DIAGRAMAS DE CLASE QUE DESCRIBAN LA ARQUITECTURA PROPUESTA PARA EL PROTOTIPO DEL ANALIZADOR SEMÁNTICO.



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

J. 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 SINTÁXIS PROPUESTA EN EL LENGUAJE PROTOTIPO => VALIDACIÓN SINTÁCTICA DE LOS CASOS DE ESTUDIOS IDENTIFICADOS Y PROPUESTOS => VALIDACIÓN SEMÁNTICA DE LOS CASOS DE ESTUDIOS IDENTIFICADOS Y PROPUESTOS.

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

L. GENERACIÓN DE UNA BITÁCORA DE INCIDENCIAS.

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

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

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

6. COMO EQUIPO GENERAR SU PORTAFOLIO DE EVIDENCIAS ELECTRÓNICAS ( PED ).

ESTO SE HARÁ USANDO UN CD O DVD EN EL CUAL EL EQUIPO CREARÁ UNA CARPETA PARA INTEGRAR TODAS SUS EVIDENCIAS DE TRABAJO SEMESTRAL, SEGÚN LO INDICA LA ESTRUCTURA PROPUESTA EN LA GUÍA TUTORIAL.

LA ESTRUCTURA RAÍZ DEL CD O DVD PUEDE TENER LA SIGUIENTE APARIENCIA:

```
\
|
+----LAII_EVALUACIÓN_1_SEM_ENE_JUN_2018
|
+----LAII_EVALUACIÓN_2_SEM_ENE_JUN_2018
|
+----LAII_EVALUACIÓN_3_SEM_ENE_JUN_2018
|
+----LAII_EVALUACIÓN_4_SEM_ENE_JUN_2018
```

POR ÚLTIMO, HACER ÉNFASIS EN QUE LA FECHA DE ENTREGA DEL PED, SERÁ EN LA SESIÓN DE EXAMEN PARA LA EVALUACIÓN 4.





**NOTAS GENERAL DE LA ACTIVIDAD:**

- LAS ACTIVIDADES INCOMPLETAS NO TENDRÁN VALOR ALGUNO, POR LO TANTO EN EQUIPO SE DEBE REVISAR QUE EL CONTENIDO DE LA ACTIVIDAD ESTÉ COMPLETO.
- 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* ).
- LAS PRÁCTICAS DEBERÁN CONTENER TODAS LAS SECCIONES SOLICITADAS EN LA GUÍA TUTORIAL, CONCRETAMENTE EL PUNTO 3 INCISO h ( *Prácticas* ).
- SE DEBE CONSIDERAR Y TOMAR EN CUENTA PARA EL CORRECTO CUMPLIMIENTO DE ESTA ACTIVIDAD LO SOLICITADO EN LA GUÍA TUTORIAL, CONCRETAMENTE EN EL PUNTO 6, ( *Evidencias tipo a* ).



**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.
- ✓ UNA VEZ ELABORADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA SIGUIENTE NOMENCLATURA.
- ✓ 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.
- ✓ 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.
- ✓ AÚN PARA TRABAJOS EN EQUIPO APLICAN TODAS LAS MISMAS OBSERVACIONES ANTERIORES.



Aniversario  
TecNM  
en Celaya  
1959-2019

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010  
Celaya, Gto. AP 57, Conmutador:(461)6117575, Correo electrónico: [lince@itcelaya.edu.mx](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)





**LA NOMENCLATURA SOLICITADA ES :**

AAAA-MM-DD\_MATERIA\_DOCUMENTO\_EQUIPO\_NOCTROL\_APELLIDOS\_NOMBRE\_SEM.PDF

( **NOTA : \*\*\* TODO EN MAYÚSCULA \*\*\*** )

**DONDE :**

AAAA : **AÑO**  
MM : **MES**  
DD : **DÍA**  
MATERIA : **SO, TSO, LAII, LI**  
DOCUMENTO : **A1-ACTIVIDAD 1, 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 :**

2018-05-21\_LAII\_A4\_EQUIPO\_99\_9999999\_PEREZ\_PEREZ\_JUAN\_ENE-JUN18.PDF

**FECHA DE ENTREGA:**

**VÍA CORREO ELECTRÓNICO, EL LUNES 21 DE MAYO DEL 2018, CON HORA LÍMITE DE ENTREGA HASTA LAS 14:00 HORAS ( 2 DE LA TARDE ).**

**DEPUÉS DE ESTA HORA, LA ACTIVIDAD SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO CONTARÁ COMO EVIDENCIA PARA SU EVALUACIÓN.**

**MUY IMPORTANTE:**

**POR FAVOR ANEXE A SU ARCHIVO .PDF DE EVIDENCIAS, ESTA SOLICITUD DE ACTIVIDADES CON TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO.**

**CALENDARIO 2017-2018 - EVALUACIÓN 4**

MAYO						
L	M	M	J	V	S	D
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUNIO						
L	M	M	J	V	S	D
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

EVALUACIÓN FORMATIVA DE OPORTUNIDAD (PARCIALES)





## INSTITUTO TECNOLÓGICO DE CELAYA

TERCER ETAPA DE UN COMPILADOR:  
ANÁLISIS SEMÁNTICO

EQUIPO:  
CASTRO GONZALEZ RICARDO ISAAC  
CERVANTES VALADEZ ARCADIO  
HERNANDEZ ALTAMIRA LUIS FELIPE  
PANTOJA VALLE LAURA JAZMIN

### CARRERA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

### MATERIA

LENGUAJES Y AUTOMATAS II

1

### INTRODUCCIÓN

El analizador semántico realiza una revisión del programa fuente para tratar de encontrar los errores de tipo semántico y reunir toda la información necesaria sobre los tipos de errores para la fase posterior de generación de código.

2

### OBJETIVO (COMPETENCIA)

Proponer, diseñar e implementar las herramientas necesarias para el desarrollo de un analizador semántico, es decir, diagramas, estructuras de datos y programas para la construcción de la tercera etapa de un compilador.

3

### FUNDAMENTO

**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?**

En la actividad anterior se generó el análisis sintáctico, que es el que procesa la secuencia de tokens generada con anterioridad (análisis léxico) y los caracteres se agrupan jerárquicamente en frases gramaticales que el compilador utiliza para sintetizar una salida es decir se comprueba si es sintácticamente correcto esta es una representación intermedia aun no es lenguaje maquina pero le permitirá al compilador realizar su labor con más facilidad en las fases sucesivas por lo cual las frases gramaticales del programa fuente son expresadas mediante un árbol de análisis sintáctico.

Ahora bien, para entender bien, la semántica de un lenguaje tiene como objetivo dar sentido a sus construcciones, como los tokens, estructuras y sintaxis. La semántica ayuda a interpretar los símbolos, sus tipos y sus relaciones con los demás tokens y además de determinar el tipo de resultado intermedios, comprobando que los argumentos que tiene un operador pertenecen al conjunto de los operadores posibles, y si son compatibles entre sí, etc. En definitiva, comprobando que el significado de lo que se va leyendo es válido.

Por ejemplo:

**int a = "value";**

No debe de emitir un error léxico, ya que cada uno de los tokens, pertenecen a los símbolos del lenguaje. En el sintáctico, tampoco, ya que las reglas para poder declarar variables y darle valor están correctas, pero en el análisis semántico, esta incorrecto y debe generar el error de **Tipo de asignación**.

Además, la semántica puede estudiarse desde diferentes puntos de vistas:

- ✓ **Semántica lingüística:** trata de la codificación y decodificación de los contenidos semánticos en las estructuras lingüísticas. Estudia la estructura de las formas léxicas, la estructura de las expresiones y su relación con sus referentes, así como los mecanismos mentales por los cuales los individuos atribuyen significados a las expresiones lingüísticas.
- ✓ **Semántica lógica:** desarrolla una serie de problemas lógicos de significación, estudia la relación entre el signo lingüístico y la realidad.
- ✓ **Semántica en ciencias cognitivas:** intenta explicar por qué nos comunicamos, y cuál es el mecanismo psíquico que se establece entre el hablante y el oyente durante este proceso.

Ahora bien, la fase de análisis semántico de un procesador de lenguaje es aquella que recopila la información adicional necesario para el procesamiento de un lenguaje, una vez que la estructura sintáctica de un programa haya sido obtenida.

El análisis semántico debe de realizar las siguientes tareas.

- Resolución de ámbito.
- Comprobación de tipos de datos.
- Matriz de control.

## **B. INVESTIGAR. ¿EN QUE CONSISTE UN ANALISIS SEMÁNTICO Y QUÉ LO CARACTERIZA?**

El análisis semántico consiste en detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico. El analizador semántico suele trabajar simultáneamente con el analizador sintáctico y en estrecha cooperación.

El analizador realiza una revisión del programa fuente para tratar de encontrar los errores de tipo semántico y reunir toda la información necesaria sobre los tipos de errores para la fase posterior de generación de código.

En esta etapa se utiliza una estructura jerárquica determinada por la fase del análisis sintáctico para identificar los operadores y operandos de expresiones y proposiciones.

El analizador semántico recibe la información del resultado del análisis sintáctico, en este caso, el



árbol de análisis sintáctico con la información relativa a la organización de los tokens para comprobar restricciones de tipo de otras limitaciones semánticas y así preparar la generación de código.

Se compone de un conjunto de rutinas independientes, llamadas por los analizadores morfológico y sintáctico.

Las rutinas semánticas deben realizar la evaluación de los atributos de las gramáticas siguiendo las reglas semánticas asociadas a cada producción de la gramática.

Ejemplo de una expresión:

$$(A+B)*(C+D)$$

El analizador semántico debe determinar qué acciones pueden realizar los operadores aritméticos (+,\*), sobre las variables A, D, C y D.

De esta manera cuando el analizador sintáctico reconoce un operador, como "+" y "\*", llama a una rutina semántica que especifica la acción que se puede llevar a cabo. Esta rutina puede comprobar que los dos operandos han sido declarados. También puede comprobar si a los operandos se les ha asignado previamente algún valor.

Trata de determinar el tipo de resultados intermedios, comprobar que los argumentos que tiene un operador pertenecen al conjunto de los operadores posibles. Es decir, comprobara que todo lo que vaya leyendo sea válido.

En los compiladores de un solo paso, las llamadas a las rutinas semánticas se realizan directamente desde el analizador sintáctico y son estas rutinas las que llaman al generador de código.

En los compiladores de dos o más pasos, el análisis semántico se realiza independientemente de la generación de código, pasándose información mediante un archivo que funciona de intermedio, por lo general contiene información sobre el árbol sintáctico en forma analizada (de esta manera se facilita el manejo y hace posible su almacenamiento en la memoria auxiliar).

En cualquier caso, las rutinas semánticas suelen hacer uso de una pila, en este caso, la pila semántica, esta contiene la información semántica asociada a los operandos (y en ocasiones a los operadores) en forma de registros semánticos.

Al decir pila semántica no se refiere a que hay varios tipos de pila, hace referencia a que se debe programar una pila en un solo lenguaje.

La salida teórica de la fase del análisis semántico sería un árbol semántico. Este consiste en un árbol sintáctico en el que cada una de las ramas ha adquirido el significado que debe tener.

### **Reglas semánticas.**

Son el conjunto de normas y especificaciones que definen al lenguaje de programación y están

dadas por la propia sintaxis del lenguaje, estas reglas asignan un significado lógico a ciertas expresiones definidas en la sintaxis del lenguaje.

Las evaluaciones de las reglas semánticas definen los valores de los atributos en los nodos del árbol del análisis sintáctico para la cadena de entrada.

### **Compatibilidad de tipos.**

Durante la fase de análisis semántico, el compilador debe verificar que los tipos y valores asociados a los objetos de un programa se utilizan de acuerdo con la especificación del lenguaje.

Además, cabe destacar que tiene que detectar conversiones implícitas de tipos para efectuarla o insertar el código apropiado para efectuarlas, así como almacenar información relativa a los tipos de los objetos y aplicar las reglas de verificación de tipos.

### **C. IDENTIFICAR. ¿QUÉ CASOS DE ESTUDIO SON LOS MAS IMPORTANTES QUE CONSIDERAR EN EL ANÁLISIS SEMÁTICO?**

Una de las misiones más importantes de un compilador y una de las más complicadas es el manejo de errores a veces un error provoca una avalancha de muchos errores que se solucionan con el primero. Es conveniente un buen manejo de errores, y que el compilador detecte todos los errores que tiene el programa y no se detenga al primero que se encuentre.

Por lo que se pretende que el compilador:

- Sea capaz de detectar errores en la entrada
- El compilador debe de recuperarse de los errores sin perder demasiada información.
- Debe producir un mensaje de error que permita al programador encontrar y corregir fácilmente los elementos incorrectos del programa.

Un error semántico se genera cuando la sintaxis del código es correcta, pero la semántica o el significado no es el que se pretende. La construcción obedece las reglas del lenguaje y por esto, el compilador no detecta los errores semánticos. Un error semántico puede hacer que el programa termine de forma anormal, con e incluso sin mensajes de error.



## Casos de uso

### ➤ Conversiones de tipos no permitidas.

```
int $z;  
String $j;  
$z= 1+2*$j+4;
```

→ No es posible multiplicar un número por una cadena de caracteres.

```
Int $x;  
$x=5.4;
```

→ La variable “\$x” está declarada como entero, pero se le asigna un dato de tipo flotante.

### ➤ Variables usadas y no definidas.

```
String $saludo;  
$saludo=” hola”;  
$despedida=” adiós”;
```

→ La variable “\$despedida” no ha sido declarada.

### ➤ Los identificadores solo se declaran en una ocasión.

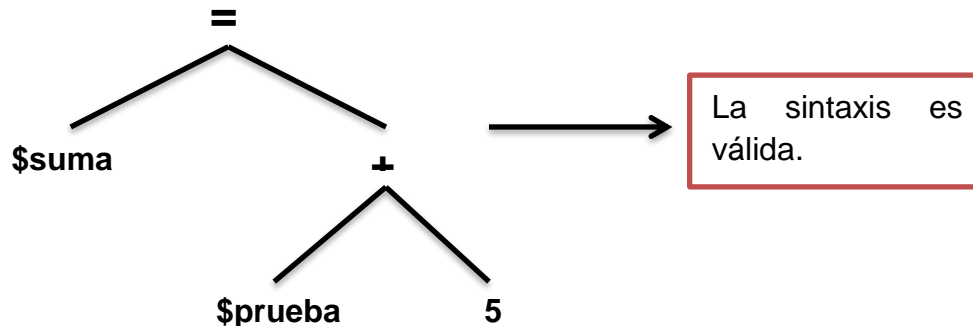
```
int $a =1;  
int $b=2;  
int $a;
```

→ La variable “\$a” se ha declarado en más de una ocasión.

### ➤ Operandos de tipos no compatibles.

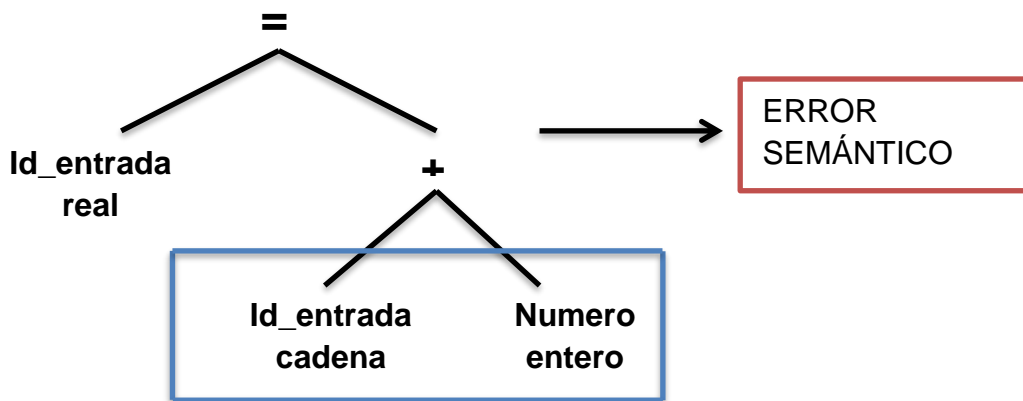
```
int $suma;  
String $prueba;  
$suma = $prueba + 5;
```

Para este caso se realiza el árbol sintáctico y comprobamos su validez.



Análisis Semántico.

Hasta este punto la entrada es léxica y sintácticamente correcta, ahora se analiza desde el punto de vista semántico.



Se comprueba que los operandos son de tipo no compatibles, por lo cual existe un error semántico.

## CATALOGO DE ERRORES

IDENTIFICADOR.	ERROR.
300	Ya había sido declarado
301	No ha sido declarado
302	No se pudo asignar un valor tipo STRING a un identificador tipo INT
303	No se pudo asignar un valor tipo INT a un identificador tipo STRING
304	No se le ha asignado algún a valor a:
305	No se puede hacer una comparación con un identificador tipo STRING:
306	Solo se aceptan valores tipo INT
307	Solo se aceptan valores tipo STRING



## TRATAMIENTO DE ERRORES SEMÁNTICOS

### Comprobación de tipos

#### 1.-Aspectos Generales

Un lenguaje con comprobación fuerte de tipos es capaz de garantizar que los programas se pueden ejecutar sin errores de tipo, por lo que los errores de tipo se detectarían siempre en tiempo de compilación.

Como mínimo, ante un error, en un comprobador de tipos debe informar de la naturaleza y posición del error y recuperarse para continuar con la comprobación del resto del programa a analizar.

Veamos algunas de las operaciones a tener en cuenta en una comprobación de tipos:

- ✓ **Conversión de tipos:** A veces es necesario transformar el tipo de una expresión para utilizar correctamente un operador o para pasar de forma adecuada un parámetro a una función.
- ✓ **Coerciones:** Una conversión de tipos que realiza de forma implícita el propio compilador. Si es el programador el que realiza la conversión se trataría entonces de una conversión explícita.
- ✓ **Sobrecarga de operadores:** La sobrecarga se resuelve determinando el tipo de cada una de las expresiones intervinientes en la sobrecarga.
- ✓ **Funciones polimórficas:** Son aquellas que trabajan con argumentos cuyo tipo puede cambiar en llamadas a la función.

#### 2. Especificación de un comprobador de tipos básico

**Básicamente se deberán realizar dos tareas:**

- a) **Asignación de tipos:** en las declaraciones.
- b) **Evolución y comprobación de tipos:** En las expresiones y en las funciones, así como en las sentencias.

### D. **INVESTIGAR Y PROPONER.** ¿QUÉ PROCESOS Y PROBLEMAS ATIENDE UNA ANÁLISIS SEMÁNTICO?

El análisis semántico le da un significado coherente a todo el proceso que se realizó en el análisis sintáctico. El chequeo semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles.

Las funciones principales que realiza el analizador semántico son:

- **Identificar cada tipo de instrucción y sus componentes**
- **Completar la Tabla de Símbolos**
- **Realizar distintas comprobaciones y validaciones:**

- Comprobaciones de tipos
- Comprobaciones de flujo de control
- Comprobaciones de unicidad

Hay situaciones en que se debe definir un objeto una vez exactamente. Por ejemplo, en Pascal, un identificador debe declararse de forma única, la etiqueta en una proposición case deben ser diferentes y no se pueden repetir los elementos en un tipo escalar.

- Comprobaciones de emparejamiento

#### ➤ **Manejo de errores semánticos**

Es una de las misiones más importantes de un compilador, aunque, al mismo tiempo, es lo que más dificulta su realización. Existen dos criterios a seguir a la hora de manejar errores:

- Pararse al detectar el primer error.
- Detectar todos los errores de una pasada.

El análisis semántico es posterior al sintáctico y mucho más difícil de formalizar que éste. Se trata de determinar el tipo de los resultados intermedios, comprobar que los argumentos que tiene un operador pertenecen al conjunto de los operadores posibles, y si son compatibles entre sí, etc.

En definitiva, comprobará que el significado de lo que se va leyendo es válido. La salida "teórica" de la fase de análisis semántico sería un árbol semántico. Consiste en un árbol sintáctico en el que cada una de sus ramas ha adquirido el significado que debe tener.

Con el analizador semántico finaliza la fase de Análisis del compilador y comienza la fase de Síntesis en la cual se comienza a generar el código objeto.

#### **Reglas semánticas**

Son el conjunto de normas y especificaciones que definen al lenguaje de programación y están dadas por la sintaxis del lenguaje, las reglas semánticas asignan un significado lógico a ciertas expresiones definidas en la sintaxis del lenguaje. La evaluación de las reglas semánticas puede generar código, guardar información en una tabla de símbolos o emitir mensajes de error además la evaluación de estas reglas define los valores de los atributos en los nodos del árbol de análisis sintáctico para la cadena de entrada.

Un compilador debe comprobar si el programa fuente sigue tanto las convenciones sintácticas como las semánticas del lenguaje entre las comprobaciones que realiza están las siguientes:

- **Compatibilidad de tipos:**

Durante la fase de análisis semántico, el compilador debe verificar que los tipos y valores asociados a los objetos de un programa se utilizan de acuerdo con la especificación del lenguaje. Como la comprobación de tipos tiene la capacidad de descubrir errores en los



programas, es importante que un comprobador de tipos haga algo razonable cuando se descubre un error. Como mínimo, el compilador debe informar de la naturaleza y la posición del error.

- **Comprobaciones del flujo de control:**

Las proposiciones que hacen que el flujo del control abandone una construcción deben tener algún lugar a dónde transferir el flujo de control. Por ejemplo, una proposición break en C hace que el control abandone la proposición que la engloba, while, for o switch más cercana; si dicha proposición que engloba no existe, ocurre un error.

- **Identificación de variables:**

Comprobar si un identificador ha sido declarado antes de utilizarlo. En ocasiones, el mismo nombre debe aparecer dos o más veces en un mismo bloque de instrucciones, el compilador debe comprobar que se utilice el mismo nombre en ambos sitios.

- **Constantes:**

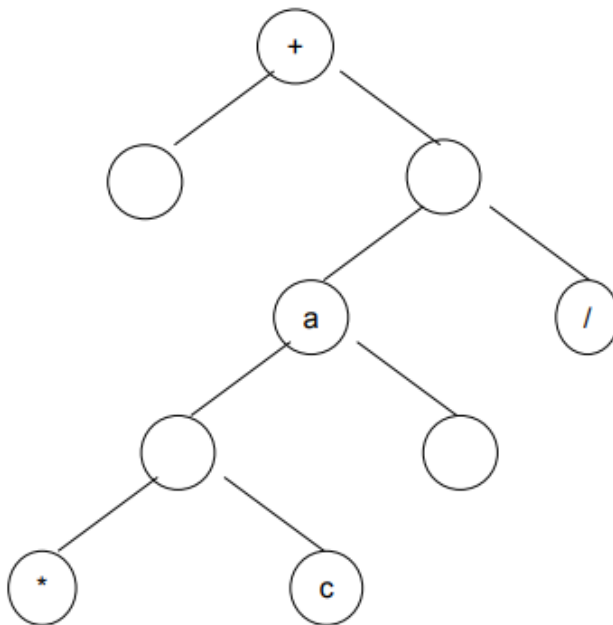
Comprobar que no se utilicen en la parte izquierda de una asignación.

- **Sobrecarga de operadores y funciones**

Detectar y solventar.

## Árbol semántico o Árbol de expresiones

Los árboles de expresiones representan el código de nivel del lenguaje en forma de datos. Los datos se almacenan en una estructura con forma de árbol. Cada nodo del árbol de expresión representa una expresión, por ejemplo, una llamada al método o una operación binaria, como  $x < y$ . Un árbol de expresión sirve para evaluar expresiones del tipo:  $(a + b) * c / d$



Para que un árbol represente una expresión se deben tomar en cuenta 2 características muy importantes:

- Cualquier hoja está etiquetada sólo con un operando.
- Cualquier nodo interior n está etiquetada por un operador

Al introducir la expresión debemos de tomar en cuenta las siguientes características:

- La raíz siempre debe ser un operador
- Las hojas siempre deben ser operandos
- Los nodos deben estar etiquetados por operadores
- Si un operador tiene mayor prioridad que la raíz se coloca como hijo.
- Si un operador tiene igual o menor prioridad que un nodo se coloca como padre.
- Un nodo puede contener como hijo otro subárbol que contiene una pequeña expresión.

### **E. INVESTIGAR. ¿CÓMO IMPLEMENTAR UN ANÁLISIS SEMÁNTICO?**

El análisis semántico se compone de un conjunto de rutinas independientes, llamadas por los analizadores morfológico y sintáctico.

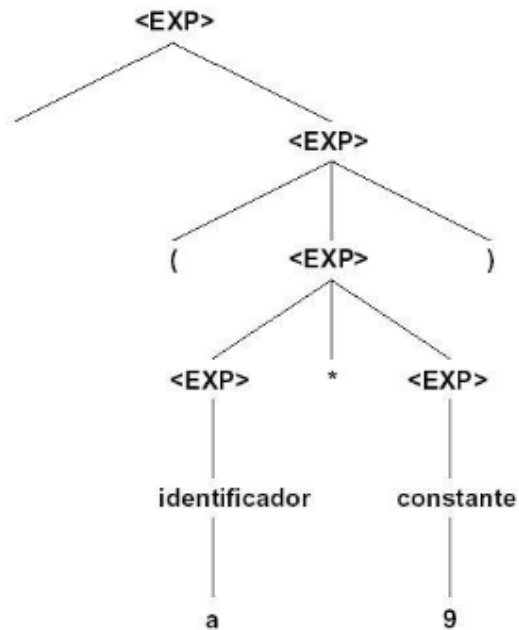
El análisis semántico utiliza como entrada el árbol sintáctico detectado por el análisis sintáctico para comprobar restricciones de tipo y otras limitaciones semánticas y preparar la generación de código. Las rutinas semánticas suelen hacer uso de una pila (la pila semántica) que contiene la información semántica asociada a los operandos (y a veces a los operadores) en forma de registros semánticos.

Como ya se mencionó anteriormente el análisis semántico incluye el hacer uso de la tabla de símbolos esto para hacer un seguimiento del significado de los identificadores (variables, funciones, tipos y método de paso de parámetros en funciones, etc.). Un paso fundamental en el analizador semántico es realizar la comprobación e inferencia de tipos en expresiones y sentencias (por ejemplo, que ambos lados de una asignación tengan tipos adecuados, que no se declaren variables con el mismo nombre, que los parámetros de llamada a una función tengan tipos adecuados, numero de parámetros correcto).

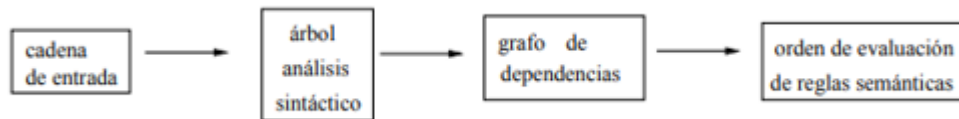
Apartir de la construcción del árbol de análisis sintáctico, este se recorre en un determinado orden y se calcula en cada nodo la información semántica necesaria (el valor de una expresión, el tipo de variable, su ámbito de declaración, el número de argumentos en una función, etc).

1.  $\langle \text{EXP} \rangle ::= \langle \text{EXP} \rangle + \langle \text{EXP} \rangle$
2.  $\langle \text{EXP} \rangle ::= \langle \text{EXP} \rangle * \langle \text{EXP} \rangle$
3.  $\langle \text{EXP} \rangle ::= \langle \text{EXP} \rangle - \langle \text{EXP} \rangle$
4.  $\langle \text{EXP} \rangle ::= \langle \text{EXP} \rangle / \langle \text{EXP} \rangle$
5.  $\langle \text{EXP} \rangle ::= \langle \text{EXP} \rangle ^ \langle \text{EXP} \rangle$
6.  $\langle \text{EXP} \rangle ::= - \langle \text{EXP} \rangle$
7.  $\langle \text{EXP} \rangle ::= ( \langle \text{EXP} \rangle )$
8.  $\langle \text{EXP} \rangle ::= \text{identificador}$
9.  $\langle \text{EXP} \rangle ::= \text{constante}$

Árbol de análisis sintáctico de  $-(a*9)$



Conceptualmente, se analiza sintácticamente la cadena de componentes léxicos de entrada, se construye el árbol de análisis sintáctico y después se recorre el árbol, en un determinado orden para tener en cuenta las dependencias,, para evaluar las reglas semánticas en sus nodos.



Hay casos especiales que se pueden implantar de una sola pasada evaluando las reglas semánticas durante el proceso de análisis sintáctico, sin construir explícitamente un árbol de análisis sintáctico. Mayor eficiencia en cuanto a tiempo de compilación.

Existen método para la evaluación de los tokens, que son dos:

### **Método basado en grafos de dependencias**

En el momento de la compilación estos métodos obtienen un orden de evaluación a partir de un grafo de dependencias sobre el árbol de análisis sintáctico para la entrada dada (programa fuente). Poco eficientes (en espacio y tiempo) porque necesitan construir todo el árbol de análisis sintáctico y sobre él, el grafo de dependencias para cada entrada, además de que existe la posibilidad de que se crean ciclos.

### **Método basado en reglas.**

Para cada producción queda predeterminado por el diseñador del compilador el orden de evaluación de los atributos de esa construcción lingüística, y a su la forma de recorrer el árbol para calcular ese atributo.

El implementador del compilador analiza la gramática y establece un orden de evaluación de los atributos en el momento de la construcción del compilador. Implica un recorrido prefijado del árbol de análisis sintáctico.

### **Recorrido del árbol con atributos sintetizados.**

El valor de los atributos en una gramática de atributos sintetizados puede ser calculado mediante



un recorrido ascendente del árbol (bottom-up) o en modo postorder (el padre el ultimo).

Representación en pseudocódigo:

procedimiento EvaluaPostOrder(T:nodo)

**inicio**

**para\_cada** hijo h de T **hacer**

    EvaluaPostOrder(h) // visitamos los hijos

**Fin\_para**

    // procesamos el nod padre

    Calcular los atributos sintetizados para T

**Fin**

### **Recorrido del árbol con atributos heredados.**

Cuando los atributos son heredados la información pasa del padre a los hijos y/o de hermanos a hermanos. El valor de los atributos puede ser calculado mediante un recorrido del árbol en modo preorder (procesamos el padre el primero).

A veces es necesario combinar recorrido prefijo/infijo.

Representación en pseudocódigo:

procedimiento EvaluaPreOrder(T:nodo)

**inicio**

**para\_cada** hijo h de T (en orden adecuado) **hacer**

    Evaluar los atributos heredados para h

    EvaluaPreOrder(h) //visitamos los hijos

**Fin\_para**

**fin**

**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, CREACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS PARA DETERMINAR SI TIENEN EL SENTIDO CORRECTO LOS ELEMENTOS INTEGRADOS COMO CÓDIGO FUENTE.**

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

Las estructuras de datos utilizados fueron:

#### **1. Pilas.**

Una pila (stack en inglés) es una lista ordenada o estructura de datos que permite almacenar y recuperar datos, el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In, First Out, «último en entrar, primero en salir).

Esta estructura fue utilizada para guardar los errores generados por el analizador semántico, teniendo como atributos:

- ✓ Identificador del error.
- ✓ Identificador.

- ✓ Línea del error.

## 2. Listas.

La lista enlazada es un TDA que nos permite almacenar datos de una forma organizada, al igual que los vectores, pero, a diferencia de estos, esta estructura es dinámica, por lo que no tenemos que saber "a priori" los elementos que puede contener.

En una lista enlazada, cada elemento apunta al siguiente excepto el último que no tiene sucesor y el valor del enlace es null. Por ello los elementos son registros que contienen el dato a almacenar y un enlace al siguiente elemento. Los elementos de una lista, suelen recibir también el nombre de nodos de la lista.

Se utilizo para guardar todos aquellos tokens que fueron declarados al iniciar el programa, ya que estos deberían de ser usados después para poder identificar que identificadores fueron o no fueron declarados. Y contiene los siguientes atributos:

- ✓ Tipo de dato.
- ✓ Identificador.

## 3. Arreglos.

Un arreglo puede definirse como un grupo o una colección finita, homogénea y ordenada de elementos. Los arreglos pueden ser de los siguientes tipos:

- ✓ De una dimensión.
- ✓ De dos dimensiones.
- ✓ De tres o más dimensiones.
- ✓ Tipos de arreglos
- ✓ Arreglos unidimensionales.
- ✓ Arreglos multidimensionales.
- ✓ Arreglo con múltiples subíndices.

### Arreglos unidimensionales

Es un tipo de datos estructurado que está formado de una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos iguales. Están formados por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre, y se diferencian por la posición que tiene cada elemento dentro del arreglo de datos. Al declarar un arreglo, se debe inicializar sus elementos antes de utilizarlos. Para declarar un arreglo tiene que indicar su tipo, un nombre único y la cantidad de elementos que va a contener.

### Arreglos multidimensionales

Es un tipo de dato estructurado, que está compuesto por dimensiones. Para hacer referencia a cada componente del arreglo es necesario utilizar n índices, uno para cada dimensión. El término dimensión representa el número de índices utilizados para referirse a un elemento particular en el arreglo. Los arreglos de más de una dimensión se llaman arreglos multidimensionales.

### Arreglos con múltiples subíndices

Es la representación de tablas de valores, consistiendo de información arreglada en renglones y columnas. Para identificar un elemento particular de la tabla, deberemos de especificar dos subíndices; el primero identifica el renglón del elemento y el segundo identifica la columna del elemento. A los arreglos que requieren dos subíndices para identificar un elemento en particular se conocen como arreglo de doble subíndice. Note que los arreglos de múltiples subíndices pueden tener más de dos subíndices. El estándar ANSI indica que un sistema ANSI C debe soportar por lo menos 12 subíndices de arreglo.

Se utilizó para guardar todos aquellos tokens que fueron guardados a la tabla de símbolos, se utilizó con un medio de enlace para poder acceder a los tokens guardados en la tabla de símbolos. Contando con los siguientes atributos:

- ✓ Etiqueta.
- ✓ Identificador.
- ✓ Numero de línea.
- ✓ Lexema.

**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 CODIFIQUEN 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 CORRELACIONADOS A LOS CASOS DE ESTUDIO CORRESPONDIENTES.**

## **PROGRAMA 1**

*#Programa en que las declaraciones o asignaciones no son válidas y falta declarar una variable \$m*

```
init()
{

    String $x = "Hola mundo";
    String $val;
    $m="";
    int $i =0 ;
    int $j = 10 + 4 - $x;
    while($x<=10)
    {
        if($i == 10)
        {
            print($i+$x);
```

```

    }
    $j = 0;
    while($j < 10)
    {
        $j = $j + 1;
        print("Jeje");
    }
    $i = $i-1;
    print("Hola mundo");
}
}

```

## PROGRAMA 2

#Programa donde no se declara la variable \$g y esta mal asignado un valor int en un string en #la linea 6

```

init()
{
    String $g;
    String $g=0;
    String $valor = "Hola mundo";
    String $val=4;
    int $i;
    #Le falta valor antes del punto y coma
    int $j = 10 + 4 - 89;
    while( $i < 5 )
    {
        #Falta valor después del =
        if($i == $g)
        {
            print("Holi");
        }
        $j = 0;
        while($j < 10)
        {
            $j = $j + 1;
            print("Jeje");
        }
        $i = $i-1;
        print("Hola mundo");
    }
}
}

```



## PROGRAMA 3

#Caso de éxito en el programa cumpliendo con las tres fases del compilador

```
init(){
    int $lado;
    int $areacu;
    int $base;
    int $arearec;
    int $altura;
    int $areatri;
    print("Calcula el area de diversas figuras");
    $areacu = ($lado) ($lado);
    print("area del cuadrado" + $areacu);
    $arearec = ($base) ($altura);
    print("area del rectangulo" + $arearec);
    $areatri = (($base) ($altura))/2;
    print("area del triangulo" + $areatri);
    if($areatri == 0){
        print("error");
        print("ingrese los datos de nuevo");
    }#if
    if($areacu <= 10){
        print("correcto");
    }#if
    while($lado < (($areacu) ($areatri))){
        print("Prueba");
    }#while
}#init
```

## PROGRAMA 4

#Caso de éxito en el programa cumpliendo con las tres fases del compilador

```
init()
{
    int $suma;
    int $resultado;
    if($suma > $resultado){
        $resultado = $suma + 10;
        print("correcto" + $resultado);
    }
```

```

}#if
print("Prueba");
if($suma < $resultado){
    $resultado = $suma - 5;
    print("resultado" + $resultado);
}#if
if($suma == $resultado){
    $resultado = $suma / 2;
    print("resultado" + $resultado);
}#if
print("Esto es una prueba");
while(0){
    print("prueba correcta");
    $resultado = $suma + 10;
    print("Total" + $resultado);
}#while
}#init

```

## PROGRAMA 5

#Caso de fallo, no se puede asignar un valor tipo String a un identificador tipo int, variables no declaradas

```

init()
{
    int $suma;
    int $resultado;
    String $pru = "prueba";
    if($suma > $resultado){
        $resultado = $suma + $pru;
        print("correcto" + $resultado);
    }#if
    print("Prueba");
    if($suma < $resultado){
        $resultado = $suma - 5;
        print("resultado" + $resultado);
    }#if
    if($suma == $resultado){
        $resultado = $suma / 2;
        print("resultado" + $resultado);
    }#if
    print("Esto es una prueba");
    while($area <= (($suma)(12))){
        print("prueba correcta");
        $resultado = $suma + 10;
    }
}

```

```
        print("Total" + $resultado);  
    }#while  
    if($suma >= 70){  
        $pru = $suma + 120;  
        print("Total" + $pru);  
    }#if  
}#init
```



Se realizaron las correcciones previas para el manejo adecuado del analizador semántico.



mayo de 2018						
do.	lu.	ma.	mi.	ju.	vi.	sá.
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Del día 4 de mayo al 7 de mayo se realizó la corrección del análisis sintáctico, ya que se observaron problemas muy graves en nuestro desarrollo, por lo cual no podríamos seguir avanzando sin antes haber realizado las correcciones necesarias.

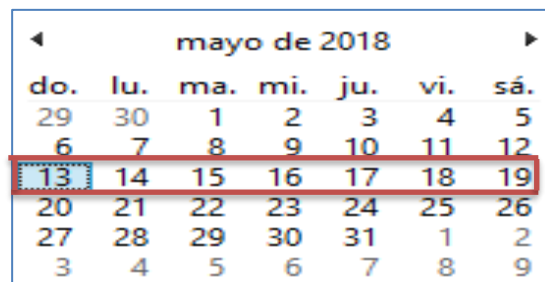
### **Etapas de Investigación acerca del Análisis Semántico.**

Se realizó la investigación necesaria sobre los temas sugeridos en la actividad 4 sobre la tercer etapa del compilador; análisis semántico.

mayo de 2018						
do.	lu.	ma.	mi.	ju.	vi.	sá.
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Fecha	Actividad Planeada	Actividad Realizada	Tiempo Planeado	Tiempo Real
<b>08-Mayo-18</b>	Investigación del punto 2 A), B), C)	Investigación del punto 2 A) y B), C)	3 hras	4 hras
<b>09- Mayo-18</b>	Investigación del punto 2 D), E) y punto 3	Investigación del punto 2 D), E) y punto 3	3 hras	3 hras
<b>10- Mayo-18</b>	Generar programas de mínimo 25 líneas	Generar programas de mínimo 25 líneas	1 hras	1 hras
<b>11- Mayo-18</b>	Generación de un catálogo de errores y diseño del diagrama de clases	Generación de un catálogo de errores y diseño del diagrama de clases	2 hras	2 hras

## Etapa de programación del analizador semántico.



mayo de 2018						
do.	lu.	ma.	mi.	ju.	vi.	sá.
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Del 13 al 18 de mayo se realizó la programación de nuestro analizador semántico.

Del 18 al 20 de mayo se realizaron pruebas al proyecto en general para comprobar que funcione de la manera correcta.

A continuación, se muestra la bitácora de incidencias, es decir aquellos problemas que surgieron durante y la solución que se generó para concluir esta etapa.

### Bitácora de Incidencias.

FECHA DE ERROR	FECHA DE SOLUCIÓN	PROBLEMA	SOLUCIÓN
10/05/2018	11/05/2018	Problemas en las estructuras de datos utilizadas.	Se volvió a rediseñar el algoritmo de inserción de identificadores con su respectivo tipo de dato, para la declaración de variables.
11/05/2018	12/05/2018	Sobre carga de memoria en las estructuras de datos.	Se investigo como eliminar la memoria ocupada durante una ejecución de un programa del lenguaje prototipo, para volver a soltar memoria y volverla a utilizar en la siguiente ejecución del programa en el lenguaje prototipo.
14/05/2018	14/05/2018	Guardar cambios del programa al realizar cambios.	Guardar en segundo plano los cambios e iniciando desde el análisis léxico.

6	DESARROLLO DEL ANALISIS SEMANTICO

## DESARROLLO DEL ANALISIS SEMANTICO

Registro.java X

Source History

```
1 package semantico;
2
3
4 import utils.models.Token;
5
6 import java.util.ArrayList;
7
8 public class Registro {
9     Token identificador;
10    Token tipoDato;
11    ArrayList<Token> valores;
12    Token estructura;
13    public Token getIdentificador() {
14        return identificador;
15    }
16
17    public void setIdentificador(Token identificador) {
18        this.identificador = identificador;
19    }
20
21    public Token getTipoDato() {
22        return tipoDato;
23    }
24
25    public void setTipoDato(Token tipoDato) {
26        this.tipoDato = tipoDato;
27    }
28
29    public ArrayList<Token> getValores() {
30        return valores;
31    }
```

Registro.java X

Source History

```
25    public void setTipoDato(Token tipoDato) {
26        this.tipoDato = tipoDato;
27    }
28
29    public ArrayList<Token> getValores() {
30        return valores;
31    }
32
33    public void setValores(ArrayList<Token> valores) {
34        this.valores = valores;
35    }
36
37    public Token getEstructura() {
38        return estructura;
39    }
40
41    public void setEstructura(Token estructura) {
42        this.estructura = estructura;
43    }
44 }
45
```



Semantico.java

Source History

```
1 package semantico;
2
3
4 import com.sun.corba.se.impl.interceptors.PICurrent;
5 import lexico.Lexico;
6 import sintactico.Sintactico;
7 import utils.PilaErrores;
8 import utils.models.*;
9
10 import javax.jws.soap.SOAPBinding;
11 import java.io.File;
12 import java.io.RandomAccessFile;
13 import java.util.*;
14
15 public class Semantico {
16
17     Registro []registros;
18     ArrayList<Registro> declarados = new ArrayList<>();
19     Map<Token,Token> declaradoComo = new HashMap<>();
20     public static void main(String arg[])
21     {
22         try
23         {
24             new Lexico(new RandomAccessFile(new File("palabras"), "r"));
25             if(PilaErrores.vacia())
26             {
27                 new Sintactico();
28                 if(PilaErrores.vacia())
29                 {
30                     System.out.println("Hare el semantico");
31                     System.out.println("_____");
```

```
Semantico.java X
Source History
System.out.println(
new Semantico();
if(!PilaErrores.vacia())
    System.out.println(PilaErrores.getErrors());
}else System.out.println(PilaErrores.getErrors());

}else System.out.println(PilaErrores.getErrors());

}catch (Exception ex){
    System.out.println(ex);
}
}

public Semantico()
{
    System.out.println("_____");
    registros = TablaSemantica.toArray();
    analizar();
}

public void analizar()
{
    Registro registro;
    for(int i = 0 ; i<registros.length; i++)
    {
        registro = registros[i];
        if(registro.getTipoDato() != null)
        {
            if(!isDeclarado(registro.getIdentificador()))
            {
                declarado(registro.getIdentificador(),registro.getTipoDato());
                Token tipoDato;
                declarado(registro.getIdentificador(),registro.getTipoDato());
                Token tipoDato;
                if(registro.getValores() != null && registro.getValores().size() > 0)
                {
                    tipoDato = validarAsignacion(registro.getValores());
                    if(registro.getTipoDato().getEtiqueta() == Etiquetas.STRING && tipoDato.getEtiqueta() != Etiquetas.VALOR_STRING)
                    {
                        PilaErrores.pushErrorSemantico(302,registro.getIdentificador(),registro.getIdentificador().getLinea());
                        continue;
                    }
                    if(registro.getTipoDato().getEtiqueta() == Etiquetas.INT && tipoDato.getEtiqueta() != Etiquetas.VALOR_INT)
                    {
                        PilaErrores.pushErrorSemantico(303,registro.getIdentificador(),registro.getIdentificador().getLinea());
                        continue;
                    }
                }
            }
            else
                PilaErrores.pushErrorSemantico(300,registro.getIdentificador(),registro.getIdentificador().getLinea());
        }else if(registro.getIdentificador() !=null && !isDeclarado(registro.getIdentificador()))
            PilaErrores.pushErrorSemantico(301,registro.getIdentificador(),registro.getIdentificador().getLinea());
        else if(registro.getIdentificador() !=null && isDeclarado(registro.getIdentificador())){
            Token tipoDato;
            Token declarado = declaradoComo(registro.getIdentificador());
            if(registro.getValores() != null && registro.getValores().size() > 0)
            {
                tipoDato = validarAsignacion(registro.getValores());
                if(declarado.getEtiqueta() == Etiquetas.VALOR_STRING && tipoDato.getEtiqueta() != Etiquetas.VALOR_STRING)
                {
                    PilaErrores.pushErrorSemantico(302,registro.getIdentificador(),registro.getIdentificador().getLinea());
                }
            }
        }
    }
}
```

```

Semantico.java x
Source History
90 PilaErrores.pushErrorSemantico(302, registro.getIdentificador(), registro.getIdentificador().getLinea());
91 continue;
92 }
93 if(declarado.getEtiqueta() == Etiquetas.VALOR_INT && tipoDato.getEtiqueta() != Etiquetas.VALOR_INT)
94 {
95     PilaErrores.pushErrorSemantico(303, registro.getIdentificador(), registro.getIdentificador().getLinea());
96     continue;
97 }
98 }
99 }else if(registro.getEstructura() != null){
100     if(registro.getValores().size()>0) {
101         Token tipoDato = validarAsignacion(registro.getValores());
102         if (tipoDato.getEtiqueta() != Etiquetas.VALOR_STRING)
103         {
104             PilaErrores.pushErrorSemantico(307, registro.getEstructura(), registro.getEstructura().getLinea());
105         }
106     }
107 }
108 }else
109 {
110     if(registro.getValores().size()>0)
111         validarExpresionBooleana(registro.getValores());
112 }
113 }
114 }
115
116 Token validarAsignacion(ArrayList<Token> valores)
117 {
118     Stack<Token> pila = new Stack<>();
119     Token valor;
120     for(int i = 0 ; i<valores.size(); i++)

```

```

Semantico.java x
Source History
116 Token validarAsignacion(ArrayList<Token> valores)
117 {
118     Stack<Token> pila = new Stack<>();
119     Token valor;
120     for(int i = 0 ; i<valores.size(); i++)
121     {
122         valor = valores.get(i);
123         if(valor.getEtiqueta() == Etiquetas.IDENTIFICADOR || valor.getEtiqueta() == Etiquetas.VALOR_INT || valor.getEtiqueta()
124             == Etiquetas.VALOR_STRING)
125         {
126             if(valor.getEtiqueta() == Etiquetas.IDENTIFICADOR)
127             {
128                 Token declarado = declaradoComo(valor);
129                 if(declarado == null){
130                     PilaErrores.pushErrorSemantico(304, valor, valor.getLinea());
131                 }
132                 pila.push(declarado);
133                 continue;
134             }
135             pila.push(valor);
136         }
137     }
138     else {
139         if(valor.getEtiqueta() == Etiquetas.ABRE_PARENTESIS || valor.getEtiqueta() ==Etiquetas.CIERRA_PARENTESIS)
140             continue;
141         valor = valores.get(i+1);
142         if(valor.getEtiqueta() ==Etiquetas.IDENTIFICADOR)
143         {
144             if(!isDeclarado(valor))
145                 PilaErrores.pushErrorSemantico(301, valor, valor.getLinea());
146             else

```

```

Semantico.java x
145         if(valor.getEtiqueta() == Etiquetas.IDENTIFICADOR || valor.getEtiqueta() == Etiquetas.VALOR_INT || valor.getEtiqueta()
146             else
147                 valor = declaradoComo(valor);
148             }
149             Token temp = pila.pop();
150             i++;
151             if((temp.getEtiqueta() == Etiquetas.VALOR_STRING && valor.getEtiqueta() == Etiquetas.VALOR_INT)
152                 || (temp.getEtiqueta() == Etiquetas.VALOR_INT && valor.getEtiqueta() == Etiquetas.VALOR_STRING)
153                 || (temp.getEtiqueta() == Etiquetas.VALOR_STRING && valor.getEtiqueta() == Etiquetas.VALOR_STRING) )
154                 pila.push(new TString(Etiquetas.VALOR_STRING,temp.getLinea(),0,""));
155             else if(valor.getEtiqueta() == Etiquetas.VALOR_INT && temp.getEtiqueta() == Etiquetas.VALOR_INT)
156                 pila.push(new TInt(Etiquetas.VALOR_INT,valor.getLinea(),0,0));
157             else pila.push(temp);
158         }
159     }
160     return pila.pop();
161 }
162
163 void validarExpresionBooleana(ArrayList<Token> valores)
164 {
165     Stack<Token> pila = new Stack<>();
166     System.out.println(valores.size());
167     for(int i = 0 ; i<valores.size(); i++)
168     {
169         Token valor = valores.get(i);
170         if(valor.getEtiqueta() == Etiquetas.VALOR_INT || valor.getEtiqueta() == Etiquetas.IDENTIFICADOR)
171             pila.push(valor);
172     }
173     while (!pila.empty())
174     {

```

```


Semantico.java x
Source History
173 while (!pila.empty())
174 {
175     Token valor = pila.pop();
176     if(valor.getEtiqueta() == Etiquetas.IDENTIFICADOR)
177     {
178         if (!isDeclarado(valor))
179             PilaErrores.pushErrorSemantico(301,valor,valor.getLinea());
180         else if(declaradoComo(valor).getEtiqueta() != Etiquetas.VALOR_INT)
181             PilaErrores.pushErrorSemantico(305,valor,valor.getLinea());
182     }else if(valor.getEtiqueta() != Etiquetas.VALOR_INT)
183         PilaErrores.pushErrorSemantico(306,valor,valor.getLinea());
184     }
185 }
186
187
188 Token declaradoComo(Token identificador)
189 {
190     for(Registro r:declarados)
191         if(r.getIdentificador().getId() == identificador.getId() )
192         {
193             if(r.getTipoDato().getEtiqueta() == Etiquetas.STRING)
194                 return new TString(Etiquetas.VALOR_STRING,0,0,"");
195             else
196                 return new TInt(Etiquetas.VALOR_INT,0,0,0);
197         }
198     return null;
199 }
200

```



Semantico.java X

Source History



200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

```
public void declarado(Token identificador, Token tipoDato)
{
    Registro r = new Registro();
    r.setTipoDato(tipoDato);
    r.setIdentificador(identificador);
    declarados.add(r);
}

/**
 * @param t
 * @return true if token is declared, false if not
 */
public boolean isDeclarado(Token t)
{
    for(Registro r:declarados)
        if(r.getIdentificador().getId() == t.getId() )
            return true;
    return false;
}
```

```

1 package semantico;
2
3
4
5 import utils.models.Token;
6 import java.util.ArrayList;
7
8 public class TablaSemantica {
9
10     public static ArrayList<Registro> valores = new ArrayList<>();
11     public static void limpiar()
12     {
13         valores = new ArrayList<>();
14     }
15
16     public static void insertar(Registro registro)
17     {
18         valores.add(registro);
19     }
20     public static Registro []toArray()
21     {
22         Registro registros[] = new Registro[valores.size()];
23         for(int i = 0; i < valores.size(); i++)
24             registros[i] = valores.get(i);
25
26         return registros;
27     }
28 }
29

```

1, C. and perfil, V. (2018) UNIDAD 5: ANÁLISIS SEMÁNTICO [http://bloggcompiladores7mo1.blogspot.mx/2010/11/unidad-5\\_20.html](http://bloggcompiladores7mo1.blogspot.mx/2010/11/unidad-5_20.html)

Arantxa.ii.uam.es. (2018). *Capítulo 5. Análisis semántico*

<http://arantxa.ii.uam.es/~alfonsec/docs/compila5.html>

Cidecame.uaeh.edu.mx. (2018). *4.4 Tipos de Datos y Verificación de Tipos.*

[http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/autocontenido/autocon/44\\_\\_tipos\\_de\\_datos\\_y\\_verificacin\\_de\\_tipos.html](http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/autocontenido/autocon/44__tipos_de_datos_y_verificacin_de_tipos.html)

Itpn.mx. (2018). <http://itpn.mx/recursosisc/7semestre/leguajesyautomatas2/Unidad%20I.pdf>

Semantico, A. (2018). *analisis semantico*. Analisissemantico.blogspot.mx. Available at: <http://analisissemantico.blogspot.mx/>

Biblioteca.uns.edu.pe. (2018).

[http://biblioteca.uns.edu.pe/saladocentes/archivoz/publicacionez/Sesion\\_III\\_3U\\_\\_\\_Analisis\\_Semantico.pdf](http://biblioteca.uns.edu.pe/saladocentes/archivoz/publicacionez/Sesion_III_3U___Analisis_Semantico.pdf)

Informatica.uv.es. (2018). <http://informatica.uv.es/docencia/iiguia/asignatu/2000/PL/2007/tema6.pdf>