



**Universidad Nacional Autónoma de México**

**Facultad de Ingeniería**



## **Tercer Programa: Analizador Semántico.**

Alumnos:

- García López Erik
- Serapio Hernández Alexis Arturo

Materia: Compiladores

Profesora: M.C. Laura Sandoval Montaña

Semestre 2024-1

Fecha de Entrega: 30/11/2023

- **Objetivo:**

Construir en un mismo programa, los analizadores Léxico, Sintáctico, Semántico que revisen programas fuente escritos con el lenguaje elaborado en clase.

- **Descripción del Programa:**

**Procesos Semánticos**

- Adecuar la gramática definida en clase que incluya símbolos de acción y atributos para que realice la actualización del campo tipo en la tabla de símbolos.
- Revisar que un mismo identificador no se declare dos o más veces.
- Completar el analizador léxico-sintáctico para que incluya el análisis semántico con las funciones mencionadas en los puntos anteriores.

**Resultados**

- Mostrar la situación de la tabla de símbolos una vez que se haya hecho todo el análisis semántico.

**Entregar en archivo .rar o .zip:**

Un documento con la siguiente estructura:

- Descripción del problema (no del programa).
- Gramática de traducción con atributos que realiza la revisión semántica.
- Indicaciones de cómo correr el programa.
- Conclusiones de cada participante.

Sólo el programa fuente definitivo

- Se podrá elaborar en equipo de 2 personas.
- Un archivo de prueba (opcional)

- **Descripción del Problema:**

Un analizador semántico es una parte crucial en el proceso de análisis de un lenguaje de programación. Su principal función es comprender el significado y la estructura de las expresiones o sentencias en un código o en un texto.

Un analizador semántico examina el código fuente para asegurarse de que cumple con las reglas semánticas del lenguaje de programación. Esto implica verificar la coherencia y corrección de las estructuras gramaticales y la interpretación adecuada de las instrucciones.

Por ejemplo, verifica si las variables están siendo utilizadas correctamente, si los tipos de datos son compatibles, entre otras

comprobaciones que no son estrictamente de sintaxis, sino de significado en el contexto del programa.

En nuestro caso, el problema consistió en la realización de un analizador semántico que cumpliera con las características:

- Adecuar la gramática definida en clase que incluya símbolos de acción y atributos para que realice la actualización del campo tipo en la tabla de símbolos.
- Revisar que un mismo identificador no se declare dos o más veces.

Para resolver estos puntos, primero adecuamos la gramática incluyendo símbolos de acción en ella.

Los símbolos de acción en un analizador semántico representan las operaciones que se realizan cuando se valida y comprende la estructura y el significado de una expresión o sentencia en un lenguaje de programación o en el análisis de un texto.

Estos símbolos de acción suelen estar asociados con reglas gramaticales o producciones en un analizador sintáctico que se activan cuando se verifica la sintaxis de una construcción y se necesita ejecutar operaciones más allá de la estructura puramente sintáctica.

Para la revisión de la gramática, se adecuó la función `AsignaTipo()`, en donde al momento de generarse posiciones adicionales a las que realmente existen, (esto dado por identificadores adicionales), nos arroja un mensaje de error indicándonos que existen dichos identificadores adicionales.

```
655 // Función que consulta la tabla de símbolos y actualiza el tipo
656 // Si ya se ha declarado la variable, muestra una advertencia
657 void asignaTipo(int tipo, int posicion) {
658     struct Iden *actual = Simbolos.raiz;
659
660     while (actual != NULL) {
661         if (actual->posicion == posicion) {
662             actual->tipo = tipo;
663             return;
664         }
665         actual = actual->next;
666     }
667     // En este punto, la posición no se encontró en la tabla de símbolos.
668     //printf("ERROR: No se encontró la posición %d en la tabla de símbolos\n", posicion);
669     banderita++;
670 }
671 }
```

```

674 int VAT(int tipo, int posicion){
675     int actual = obtenerTipoActual(posicion);
676
677     if (actual == -1){
678         asignaTipo(tipo, posicion);
679         return 0;
680     } else {
681         printf("La variable ya fue declarada anteriormen-teeeee");
682         //printf("ADVERTENCIA: Variable '%s' ya declarada anteriormente\n", obtenerTipoActual(posicion));
683         return -1;
684     }
685 }

```

- Ejemplos:

ASemantico\_GarciaSerapio.l

cadena.txt

fuelle.txt X

fuelle.txt

```

1  assinado _fun1 (inteiro _string1 , flutuador _string2)
2  {
3  inteiro _val=2, _num1, _num2;
4  inteiro _hola=5;
5  flutuador _num3=4;
6  }

```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```

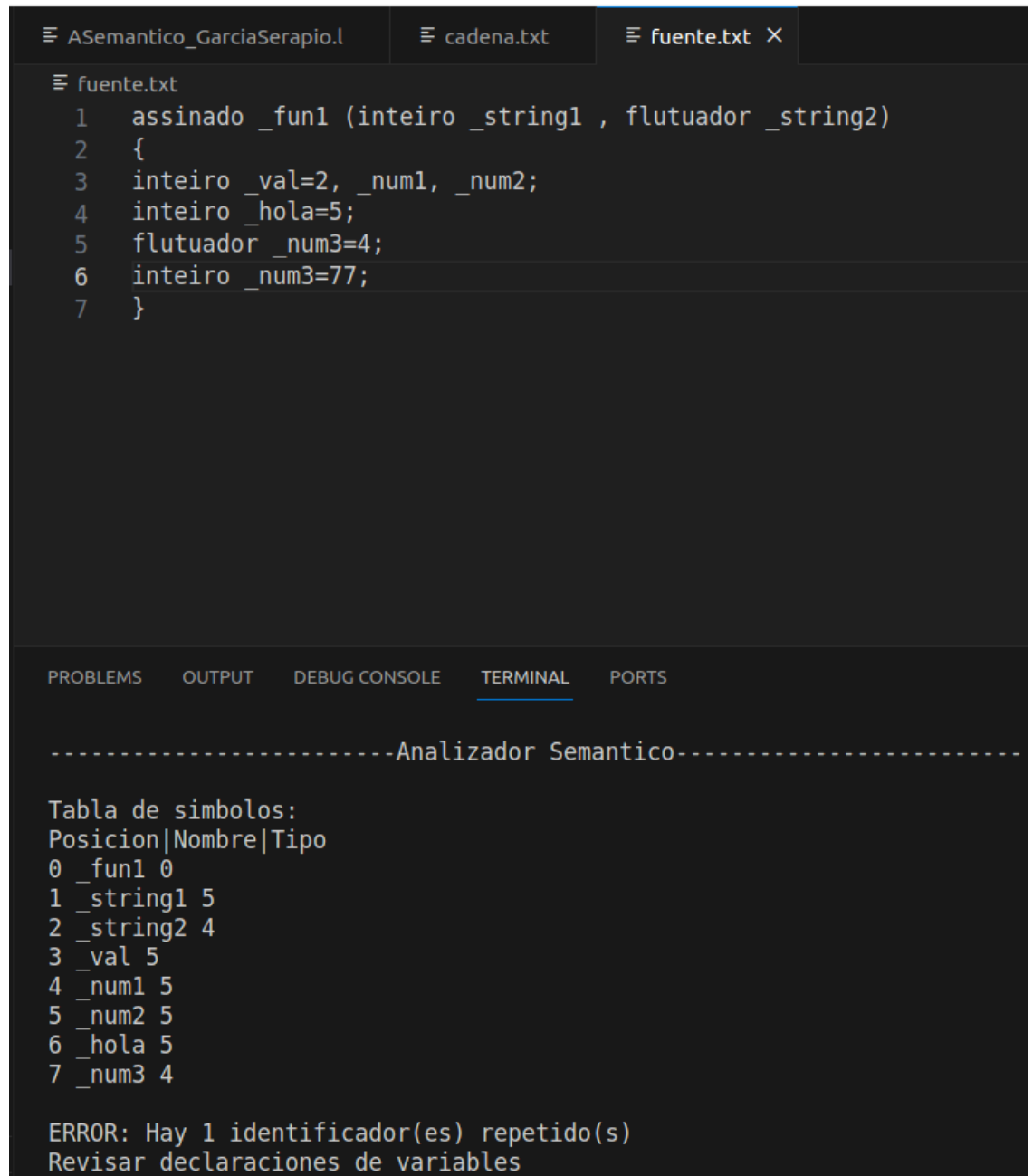
-----Analizador Semantico-----

Tabla de simbolos:
Posicion|Nombre|Tipo
0 _fun1 0
1 _string1 5
2 _string2 4
3 _val 5
4 _num1 5
5 _num2 5
6 _hola 5
7 _num3 4

Programa correcto semanticamente, se han asignado los tipos de dato.

```

En esta primera captura, observamos una correcta declaración de variables sin identificadores repetidos, al momento de ejecutar el programa, no nos sale ningún mensaje de error y se nos indica que el programa es **correcto semánticamente**.



The screenshot shows a code editor with three tabs: 'ASemantico\_GarciaSerapio.l', 'cadena.txt', and 'fuente.txt'. The 'fuente.txt' tab is active, displaying a C program with the following code:

```
1 assinado _fun1 (inteiro _string1 , flutuador _string2)
2 {
3     inteiro _val=2, _num1, _num2;
4     inteiro _hola=5;
5     flutuador _num3=4;
6     inteiro _num3=77;
7 }
```

Below the code editor, the 'TERMINAL' tab is active, showing the output of the semantic analyzer. The output is as follows:

```
-----Analizador Semantico-----

Tabla de simbolos:
Posicion|Nombre|Tipo
0 _fun1 0
1 _string1 5
2 _string2 4
3 _val 5
4 _num1 5
5 _num2 5
6 _hola 5
7 _num3 4

ERROR: Hay 1 identificador(es) repetido(s)
Revisar declaraciones de variables
```

En la segunda captura, observamos una repetición al momento de declarar una variable extra. Al momento de ejecutar podemos observar que el programa nos indica que existe un error y nos proporciona el

número de identificadores repetidos que encontró. **No nos proporciona un mensaje de salida correcto.**

- **Gramática de traducción con atributos que realiza la revisión semántica**

Para realizar la gramática de traducción se agregó el símbolo de acción {AT} en las producciones donde había un identificador que en este caso corresponde al átomo 'a'. Estas producciones fueron las siguientes:

4: <Func> → <TipoF>a{AT}(<listArg>){<Cuerpo >}  
7: <listArg> → <Tipo>a{AT}<otroArg>  
9: <otroArg> → ,<Tipo>a{AT}<otroArg>  
14: <Decl> → <Tipo>a{AT}<valorIni><listaVar>;  
19: <listaVar> → ,a{AT}<valorIni><listaVar>  
23: <Asig> → a{AT}<opArit>E;  
40: F → a{AT}  
77: <expCad> → a{AT}<opCad>  
82: <Para> → pa{AT} [n,n]#<listaS>#  
83: <Trocar> → b(a{AT}):#c(n){<listaS>}<casos>#

1:	$\langle \text{Program} \rangle \rightarrow \langle \text{Func} \rangle \langle \text{otraFunc} \rangle$
2:	$\langle \text{otraFunc} \rangle \rightarrow \langle \text{Func} \rangle \langle \text{otraFunc} \rangle$
3:	$\langle \text{otraFunc} \rangle \rightarrow \xi$
4:	$\langle \text{Func} \rangle \rightarrow \langle \text{TipoF} \rangle a\{\text{AT}\}(\langle \text{listArg} \rangle)\{\langle \text{Cuerpo} \rangle\}$
5:	$\langle \text{TipoF} \rangle \rightarrow \langle \text{Tipo} \rangle$
6:	$\langle \text{TipoF} \rangle \rightarrow g$
7:	$\langle \text{listArg} \rangle \rightarrow \langle \text{Tipo} \rangle a\{\text{AT}\} \langle \text{otroArg} \rangle$
8:	$\langle \text{listArg} \rangle \rightarrow \xi$
9:	$\langle \text{otroArg} \rangle \rightarrow \langle \text{Tipo} \rangle a\{\text{AT}\} \langle \text{otroArg} \rangle$
10:	$\langle \text{otroArg} \rangle \rightarrow \xi$
11:	$\langle \text{Cuerpo} \rangle \rightarrow \langle \text{listDecl} \rangle \langle \text{listaS} \rangle$
12:	$\langle \text{listDecl} \rangle \rightarrow \xi$
13:	$\langle \text{listDecl} \rangle \rightarrow \langle \text{Decl} \rangle \langle \text{listDecl} \rangle$
14:	$\langle \text{Decl} \rangle \rightarrow \langle \text{Tipo} \rangle a\{\text{AT}\} \langle \text{valorIni} \rangle \langle \text{listaVar} \rangle;$
15:	$\langle \text{Tipo} \rangle \rightarrow f$
16:	$\langle \text{Tipo} \rangle \rightarrow i$
17:	$\langle \text{valorIni} \rangle \rightarrow \langle \text{tipoVal} \rangle$
18:	$\langle \text{valorIni} \rangle \rightarrow \xi$
19:	$\langle \text{listaVar} \rangle \rightarrow a\{\text{AT}\} \langle \text{valorIni} \rangle \langle \text{listaVar} \rangle$
20:	$\langle \text{listaVar} \rangle \rightarrow \xi$
21:	$\langle \text{tipoVal} \rangle \rightarrow n$
22:	$\langle \text{tipoVal} \rangle \rightarrow v$
23:	$\langle \text{Asig} \rangle \rightarrow a\{\text{AT}\} \langle \text{opArit} \rangle E;$
24:	$\langle \text{opArit} \rangle \rightarrow =$
25:	$\langle \text{opArit} \rangle \rightarrow x$
26:	$\langle \text{opArit} \rangle \rightarrow y$
27:	$\langle \text{opArit} \rangle \rightarrow k$
28:	$\langle \text{opArit} \rangle \rightarrow r$
29:	$\langle \text{opArit} \rangle \rightarrow o$
30:	$E \rightarrow TE'$
31:	$E' \rightarrow +TE'$
32:	$E' \rightarrow -TE'$
33:	$E' \rightarrow \xi$
34:	$T \rightarrow FT'$
35:	$T' \rightarrow *FT'$
36:	$T' \rightarrow /FT'$
37:	$T' \rightarrow \%FT'$
38:	$T' \rightarrow \xi$
39:	$F \rightarrow (E)$
40:	$F \rightarrow a\{\text{AT}\}$
41:	$F \rightarrow n$

42:	$F \rightarrow \langle \text{Llama} \rangle$
43:	$R \rightarrow E \langle \text{opRel} \rangle E$
44:	$\langle \text{opRel} \rangle \rightarrow >$
45:	$\langle \text{opRel} \rangle \rightarrow <$
46:	$\langle \text{opRel} \rangle \rightarrow e$
47:	$\langle \text{opRel} \rangle \rightarrow d$
48:	$\langle \text{opRel} \rangle \rightarrow m$
49:	$\langle \text{opRel} \rangle \rightarrow w$
50:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Asig} \rangle$
51:	$\langle \text{Sent} \rangle \rightarrow Q$
52:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Ret} \rangle$
53:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Trocar} \rangle$
54:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Enq} \rangle$
55:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Faz} \rangle$
56:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Se} \rangle$
57:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Para} \rangle$
58:	$\langle \text{Sent} \rangle \rightarrow \langle \text{Llama} \rangle$
59:	$\langle \text{listaS} \rangle \rightarrow \langle \text{Sent} \rangle \langle \text{listaS} \rangle$
60:	$\langle \text{listaS} \rangle \rightarrow \xi$
61:	$\langle \text{expLogica} \rangle \rightarrow ! \langle \text{expRel} \rangle$
62:	$\langle \text{expLogica} \rangle \rightarrow R \langle \text{expLog} \rangle$
63:	$\langle \text{expLog} \rangle \rightarrow \langle \text{opLog} \rangle R$
64:	$\langle \text{expLog} \rangle \rightarrow \xi$
65:	$\langle \text{expRel} \rangle \rightarrow \{R\}$
66:	$\langle \text{expRel} \rangle \rightarrow E$
67:	$\langle \text{opLog} \rangle \rightarrow h$
68:	$\langle \text{opLog} \rangle \rightarrow j$
69:	$\langle \text{Enq} \rangle \rightarrow q(\langle \text{expLogica} \rangle) \# \langle \text{listaS} \rangle \#$
70:	$\langle \text{Se} \rangle \rightarrow s(\langle \text{expLogica} \rangle) \# \langle \text{listaS} \rangle \#$
71:	$Q \rightarrow u;$
72:	$\langle \text{Faz} \rangle \rightarrow z \# \langle \text{listas} \rangle \# q(\langle \text{expLogica} \rangle);$
73:	$\langle \text{Ret} \rangle \rightarrow t \langle \text{valRet} \rangle;$
74:	$\langle \text{valRet} \rangle \rightarrow E$
75:	$\langle \text{valRet} \rangle \rightarrow \{ \langle \text{expCad} \rangle \}$
76:	$\langle \text{valRet} \rangle \rightarrow \xi$
77:	$\langle \text{expCad} \rangle \rightarrow a\{\text{AT}\} \langle \text{opCad} \rangle$
78:	$\langle \text{expCad} \rangle \rightarrow v \langle \text{opCad} \rangle$
79:	$\langle \text{opCad} \rangle \rightarrow l \langle \text{expCad} \rangle$
80:	$\langle \text{opCad} \rangle \rightarrow h \langle \text{expCad} \rangle$
81:	$\langle \text{opCad} \rangle \rightarrow \xi$
82:	$\langle \text{Para} \rangle \rightarrow pa\{\text{AT}\} [n,n] \# \langle \text{listaS} \rangle \#$

	<Trocar> →
83:	b(a{AT}):#c(n){<listaS><casos>#
84:	<casos> → ξ
85:	<casos> → c(n){<listaS><casos>
86:	<casos> → (){<listaS><casos>
87:	<Llama> → [a{<listP>}]

88:	<listP> → ξ
89:	<listP> → E<Param>
90:	<listP> → {<expCad><Param>
91:	<Param> → ,<listP>
92:	<Param> → ξ

### • Indicaciones de como correr el programa

Dentro de una terminal de linux debemos estar en la ruta donde se encuentra nuestro archivo. Una vez hecho lo anterior escribimos los siguientes comandos:

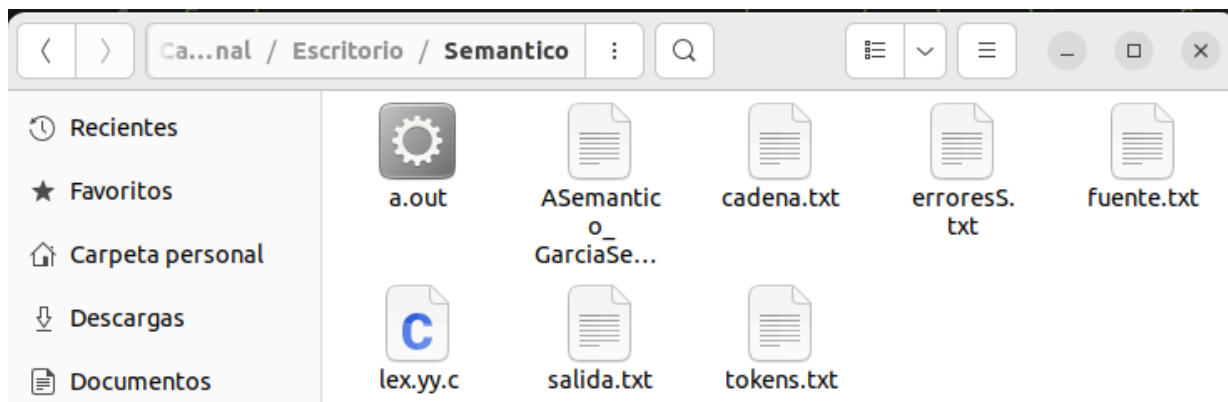
```
flex ASemantico_GarciaSerapio.l
gcc lex.yy.c -lfl -o salida.out
./a.out <entrada.txt
```

entrada.txt es el nombre del archivo que va a analizar, puede llevar cualquier nombre siempre y cuando tenga la extensión “.txt”.

Ejemplo:

```
• erik@erik-VirtualBox:~/Escritorio/Semantico$ flex ASemantico_GarciaSerapio.l
• erik@erik-VirtualBox:~/Escritorio/Semantico$ gcc lex.yy.c -lfl
• erik@erik-VirtualBox:~/Escritorio/Semantico$ ./a.out fuente.txt
```

Despues apareceran los archivos en la carpeta:





- **Conclusiones**

- **García López Erik:** Realizar este analizador me ayudó a comprender de mejor forma como funciona un analizador semántico con todo lo que implica su realización. Nosotros solo hicimos la parte para asignar el tipo de dato y no fue rápido porque hay que analizar la gramática para ver donde se ponen los símbolos de acción e implementar la función. Pero en un analizador semántico más complejo si se tendría que revisar más cosas como que se usen bien los apuntadores, que tengan coherencia las operaciones que se hacen y ahora veo que sería compleja su implementación. Finalmente puedo decir que tras haber realizado el analizador léxico, sintáctico y semántico me doy cuenta de la complejidad de estos pero a su vez logre entender su funcionamiento de mejor manera y veo el porque se presentan algunos errores cuando uno está programando.
- **Serapio Hernández Alexis Arturo:** Gracias a este proyecto pude comprender en su totalidad el funcionamiento de un compilador. Pues en el presente se desarrollaron tanto el Analizador Lexico, Sintactico y Semantico, los 3 ejecutados de manera correcta.  
En el caso del analizador semántico, la realización de este mismo me ayudó a entender que es lo que sucede en un compilador al momento de querer realizar acciones que pueden provocar errores de lógica dentro de nuestros programas. Uno de los mayores retos fue el uso de las direcciones de memoria que almacena el programa, pues pueden surgir muchos errores de acceso por el uso de los mismos. Como mejora a nuestro compilador se podría realizar una interpretación a otro lenguaje de programación logrando así una comunicación directa con el hardware del equipo.

Finalmente puedo concluir que me llevo todo el conocimiento posible sintetizado y bien desarrollado gracias al desarrollo de este proyecto a lo largo del curso de compiladores.