Análisis Sintáctico.

Universidad de Guadalajara, Centro Universitario de Ciencias exactas e Ingenierías.

Autores:

Juan José Salazar Villegas. Código: 215661291

Paola Vanessa Del Rio Gómez. Código: 215480181

Hernández Martínez Mally Samira. Código: 220286113

Juan Emmanuel Fernández de Lara Hernández. Código:

26/02/2023

Carrera: Ingeniería En Computación (INCO)

Asignatura: Traductores De Lenguajes II

Maestro: Ramos Barajas Armando

Origen del informe: Guadalajara Jalisco México

Ciclo: 2023 A

Sección: D03

Actividad: 2

índice

Introducción

Hasta el momento tenemos nuestra base definida (Analizador Léxico) y lista para identificar cada uno de las entradas y caracteres con el fin de darle un sesgo perteneciente de qué tipo de tokens entran.

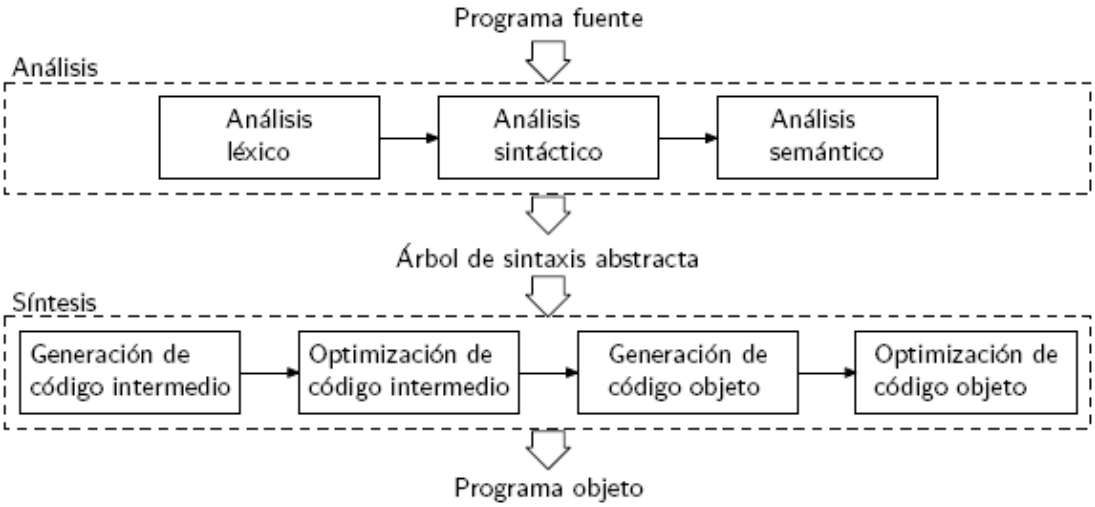
Ahora con esto podemos pasar a la verificación sintáctica dividiendo cada uno de los componentes gramaticales o tokens en unas reglas rígidas para nuestro lenguaje organizado por una estructura de árbol sintáctico que refleja la relación entre ellos. Al no cumplir las reglas gramaticales designadas nuestro analizador sintáctico mostrará un error mismo.

Este punto es de suma importancia para nuestro compilador ya que este pasará aquellos tokens de lenguaje natural a una división de identificador de errores sintácticos y no solo eso si no también de asegurar que dichas entradas sean válidas y coherentes para continuar procesando.

La lógica tras de esto parte de nuestro **árbol sintáctico** junto la información en la tabla de símbolos que comprueba la consistencia semántica del programa fuente con nuestra definición del lenguaje. Por otro lado, este recopila información sobre el tipo que pertenece y la guarda en el árbol sintáctico mismo o la tabla de símbolos esto para su uso posterior generando el código intermedio.

Así púes la verificación de tipos, donde nuestro compilador verifica que cada operador tenga operando que coincida, esto puede ser visto en aquellas definiciones de lenguajes que requieran un índice de arreglo entero y al no tener como resultado esperado este entero si no un flotante el compilador reporta un error a la hora de indexar dicho arreglo, por ello, así como este ejemplo podemos definir aún más que en a este punto podemos definir y estructurar a nuestro antojo..

En esta fase se verá cómo convertir el texto que tenemos por entrada a una estructura más firme con dicho árbol sintáctico, organizando así cada elemento y dando un sentido de traducción, donde se tendrá una tabla de símbolos correspondientes y un manejador de errores semánticos mismos.



Objetivo General:

Como se planteó en la introducción se busca el demostrar un análisis complementario a nuestra primera fase, creando un analizador semántico en C, esto base a los Tokens establecidos como palabras reservadas y continuando con el sesgo de los diversos tipos de datos y caracteres de entrada.

Por otro lado, se busca el obtener y recabar dicha información para comprender el funcionamiento del compilador, el cual está dividido en diferentes etapas que en conjunto nos proporcionan las funciones que conocemos así como la traducción del lenguaje de alto nivel a un lenguaje máquina para el procesamiento directo con la computadora así se maneja un mismo lenguaje y se establece lo que se realizará.

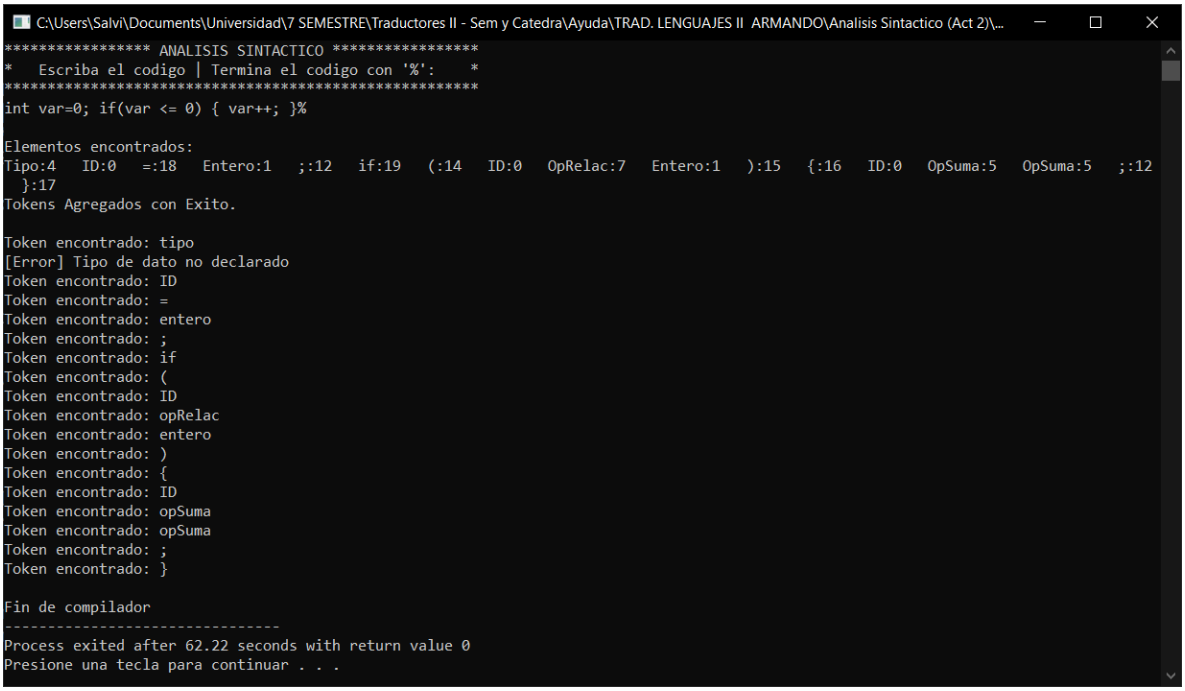
Objetivo Particular:

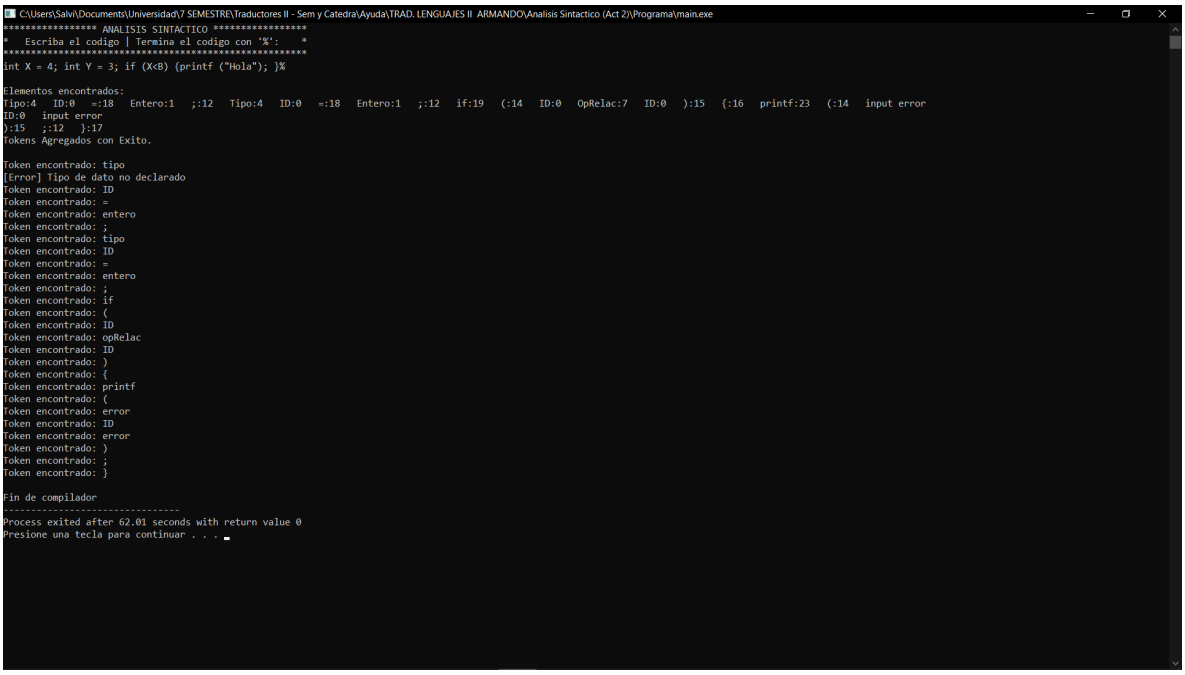
Se busca realizar complementar la primera fase realizada sumándole el análisis semántico de las palabras reservadas establecidas esto mismo por los tokens definidos anteriormente así se dará un análisis con metodologías típicas de ingeniería de software. Además de realizar diagramas que permitan la representación gráfica del sistema para un mejor entendimiento.

Desarrollo (Pantallazos)

Desarrollo del programa:

Dentro de los objetivos planteados y complementando los puntos anteriores ahora se establece el análisis sintáctico esto retomando los tokens seccionando los elementos encontrados y posteriormente mostrarlos esta lectura es gracias a un limitante siendo este “%”.





Conclusiones

Juan José Salazar Villegas

Dentro de esta práctica pude comprender como se van ensamblando cada vez más las piezas de nuestro compilador el cual ahora no solo valida si no también da un análisis profundo a una gramática establecida que no conocía y hacen todos los compiladores. Con este análisis posibilita no solo el entendimiento en relación constante con el árbol sintáctico si no también una validación de errores por incompatibilidad por algún por una incorrecta asignación. Asi pues en conclusión puedo observar como poco a poco nuestro lenguaje de programación es definido y sobre todo toma una forma funcional como los lenguajes cotidianos.

Juan Emmanuel Fernández de Lara Hernández

El análisis sintáctico es una parte fundamental del proceso de compilación de un programa, ya que es el encargado de verificar que la estructura del código esté bien formada y cumpla con la gramática del lenguaje de programación. Desarrollar una práctica sobre este tema puede ser una tarea compleja, ya que requiere un conocimiento profundo de la teoría de lenguajes formales, habilidades en programación y la implementación de algoritmos complejos.

Mally Samira Hernandez Martinez

En mi opinión esta práctica fue algo complicada ya que tuvimos ciertos problemas a la hora de codificar y saber con exactitud que debíamos de entrar, sin embargo pudimos completarla ya que cada uno de los integrantes hizo su parte correspondida. También, pude comprender que el análisis semántico es una parte fundamental del proceso de desarrollo de software, ya que garantiza que el código fuente cumpla con las reglas semánticas del lenguaje de programación utilizado y se pueda ejecutar de manera efectiva.

Paola Vanessa Del Rio Gómez

A mí parecer está práctica fue algo más compleja para hacerla, ya que se nos dificulta un poco al momento de estar codificando, en cuanto a la lógica y a las funciones que se necesitan, pero al final al ser equipo se puede resolver todo más fácilmente, de igual forma pude aprender sobre más de cómo funciona un compilador y también a cómo programarlo.

Bibliografía

- Alfred V. Aho Monica S. Lam Ravi Sethi Jeffrey D. Ullman. (s. f.). Compiladores principios, técnicas y herramientas (2.a ed.). Pearson.
- Ejemplos de Análisis Semántico. (2014, 2 abril). Gramáticas. Recuperado 25 de octubre de 2021, de https://www.gramaticas.net/2012/05/ejemplos-de- analisis-semantico.html
- Reinhard Wilhelm; Helmut Seidl; Sebastian Hack (13 May 2013). Compiler Design: Syntactic and Semantic Analysis. Springer Science & Business Media.

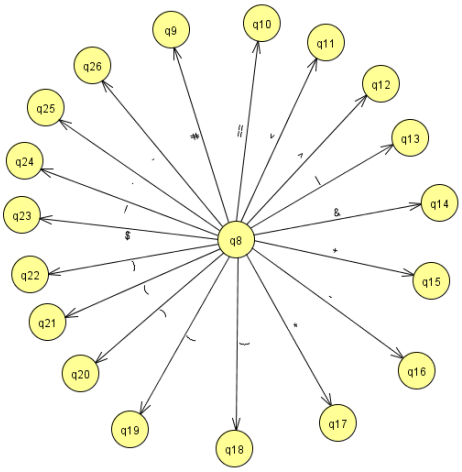
Apéndices

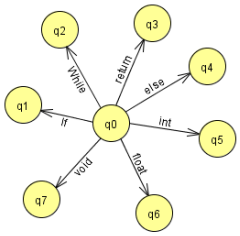
No se cuenta con apéndices para este reporte.

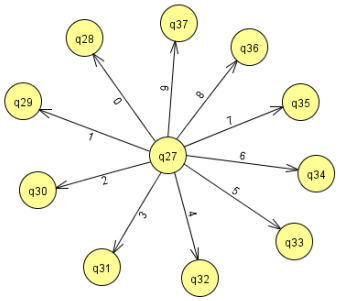
Acrónimos

Diagramas

Grafos:







Caso de uso:

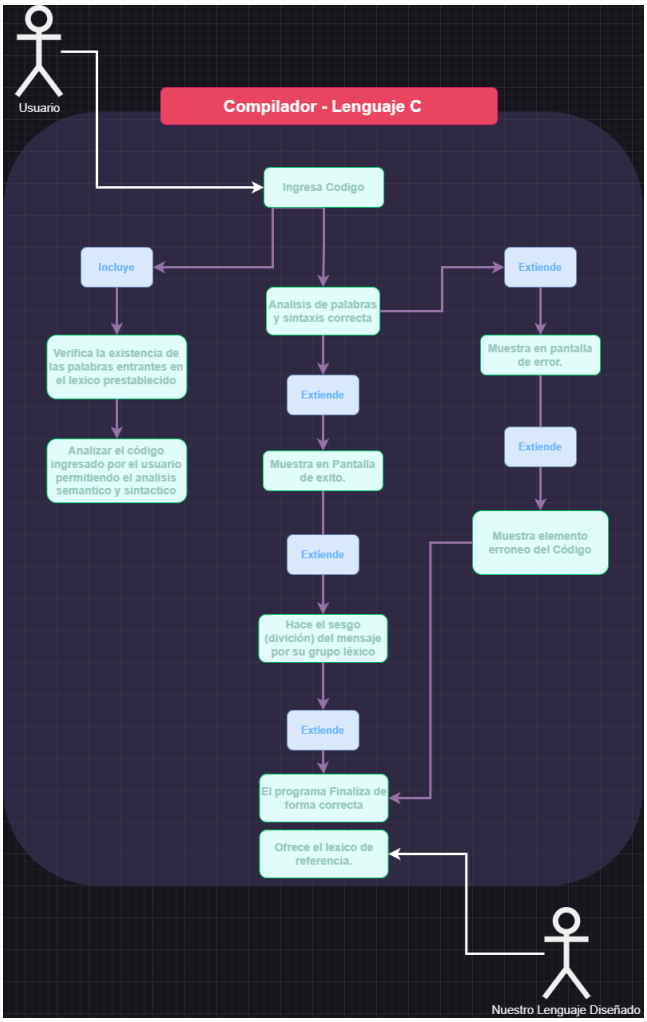


Tabla de transiciones

EDO	if	while	return	else	int	float	void
q0	{q1}	{q2}	{q3}	{q4}	{q5}	{q6}	{q7}
q1	0	0	0	0	0	0	0
q2	0	0	0	0	0	0	0
q3	0	0	0	0	0	0	0
q4	0	0	0	0	0	0	0
q5	0	0	0	0	0	0	0
q6	0	0	0	0	0	0	0
q7	0	0	0	0	0	0	0

EDO	0	1	2	3	4	5	6	7	8	9
q27	{q28}	{q29}	{q30}	{q31}	{q32}	{q33}	{q34}	{q35}	{q36}	{q37}
q28	0	0	0	0	0	0	0	0	0	0
q29	0	0	0	0	0	0	0	0	0	0
q30	0	0	0	0	0	0	0	0	0	0
q31	0	0	0	0	0	0	0	0	0	0
q32	0	0	0	0	0	0	0	0	0	0
q33	0	0	0	0	0	0	0	0	0	0
q34	0	0	0	0	0	0	0	0	0	0
q35	0	0	0	0	0	0	0	0	0	0
q36	0	0	0	0	0	0	0	0	0	0
q37	0	0	0	0	0	0	0	0	0	0

EDO	#	==	<	>	|	&	+	-	*	{	}	(	)	[	$	!	.	,
q8	{q9}	{q10}	{q11}	{q12}	{q13}	{q14}	{q15}	{q16}	{q17}	{q18}	{q19}	{q20}	{q21}	{q22}	{q23}	{q24}	{q25}	{q26}
q9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Requisitos Funcionales:

Número de requisito: **RF01**

Nombre de requisito: Actualización de tokens

Tipo: Requisito

Fuente del requisito: Función básica de compilador

Prioridad del requisito: Alto/Esencial

Descripción:

Al iniciar el programa este actualizara los tokens establecidos junto las funciones que clasifican los tipos de datos.

Número de requisito: **RF02**

Nombre de requisito: Procesamiento de entradas.

Tipo: Requisito

Fuente del requisito: Función básica de compilador

Prioridad del requisito: Alto/Esencial

Descripción:

El usuario ingresa algún código, cadena o palabra la cual se procesará y detectará en que categoría pertenece y asignarla para posteriormente mostrar en pantalla y dar el tipo de dato relacionado a su entrada. De no ser así se mostrará un mensaje de error.

Número de requisito: **RF03**

Nombre de requisito: Reacción a errores.

Tipo: Requisito

Fuente del requisito: Función básica de compilador

Prioridad del requisito: Alto/Esencial

Descripción:

Cuando se presenta algún tipo de error desde la perspectiva del usuario no generará conflictos ya que no interrumpirá el curso del programa ni la validación de códigos consecuentes.

Requisitos No Funcionales:

Número de requisito: **RNF01**

Nombre de requisito: GUI / Interfaz Visual

Tipo: Requisito

Fuente del requisito: Interfaz de usuario.

Prioridad del requisito: Media/Deseado

Descripción:

Al pensar en Interfaz hablamos de un menú simple donde se pueda desplazar fácilmente y tenga un orden claro y directa esto para que el usuario pueda ingresar datos fácilmente sin tener que comprender un complejo sistema de interfaz

Número de requisito: **RNF02**

Nombre de requisito: Eficacia y calidad del software.

Tipo: Requisito

Fuente del requisito: Funcionalidad optima del programa brindando una buena experiencia.

Prioridad del requisito: Alta/Esencial

Descripción:

El programa debe ser capaz de analizar y catalogar correctamente la mayoría de los caracteres de entrada por el usuario, dando una experiencia optima.

Número de requisito: **RNF03**

Nombre de requisito: Accesibilidad y funcionalidad optima

Tipo: Requisito

Fuente del requisito: Facilidad de uso.

Prioridad del requisito: Alta/Esencial

Descripción:

El acceso y desempeño de las funciones del software deberán ser rápidos y fácil de usar siendo intuitivo y limitando trabas de uso. De ser necesario se buscará optimizar ante el número de errores y el poco desempeño.

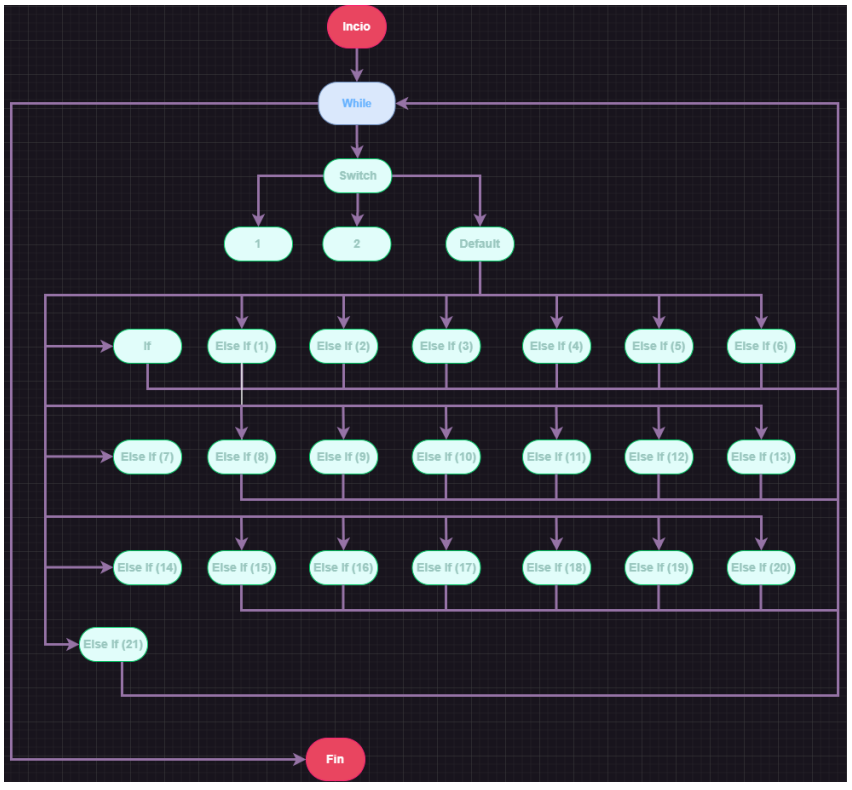
Complejidad Ciclomática:

```
34      } while( ch!='%' );
35      cout<<endl<<"Elementos encontrados: "<<endl;
36      p=0;
37      do {
38          scaner();
39          switch( fg ){
40              case 8:
41                  printf("ID:0    ");
42                  fputs("ID|", fichero);
43                  break;
44              case -1:printf("input error\n");
45                      fputs("error|", fichero);
46              break;
47              default:
48
49                  if (fg == 9){
50                      printf("Entero:1    ");
51                      fputs("entero|", fichero);
52                  }
53                  else if (fg == 6 || fg == 7 || fg == 8){
54                      printf("Tipo:4    ");
55                      fputs("tipo|", fichero);
56                  }
57                  else if (fg == 35 || fg == 10){
58                      printf("OpSuma:5    ");
59                      fputs("opSuma|", fichero);
60                  }
61                  else if (fg == 11 || fg == 12){
62                      printf("OpMul:6    ");
63                      fputs("opMul|", fichero);
64                  }
65                  else if (fg == 17 || fg == 16 || fg == 38 || fg == 39){
66                      printf("OpRelac:7    ");
67                      fputs("opRelac|", fichero);
68                  }
69                  else if (fg == 40){
70                      printf("OpOr:8    ");
71                      fputs("opOr|", fichero);
72                  }
73                  else if (fg == 42){
74                      printf("OpAnd:9    ");
75                      fputs("opAnd|", fichero);
76                  }
77                  else if (fg == 31){
78                      printf("OpNot:10    ");
79                      fputs("opNot|", fichero);
80                  }
81                  else if (fg == 30 || fg == 32){
82                      printf("OpIgualdad:11    ");
83                      fputs("opIgualdad|", fichero);
84                  }
85                  else if (fg == 19){
86                      printf(";:12    ");
87                      fputs(";|", fichero);
88                  }
89                  else if (fg == 44){
90                      printf(",:13    ");
91                      fputs(",|", fichero);
92                  }
93                  else if (fg == 20){
94                      printf("(:14    ");
95                      fputs("(|", fichero);
```

```
97                  else if (fg == 21){
98                      printf("):15    ");
99                      fputs(")|", fichero);
100                 }
101                 else if (fg == 45){
102                     printf("{:16    ");
103                     fputs("{|", fichero);
104                 }
105                 else if (fg == 46){
106                     printf("}:17    ");
107                     fputs("}|", fichero);
108                 }
109                 else if (fg == 33){
110                     printf("=:18    ");
111                     fputs("=|", fichero);
112                 }
113                 else if (fg == 1){
114                     printf("if:19    ");
115                     fputs("if|", fichero);
116                 }
117                 else if (fg == 2){
118                     printf("while:20    ");
119                     fputs("while|", fichero);
120                 }
121                 else if (fg == 3){
122                     printf("return:21    ");
123                     fputs("return|", fichero);
124                 }
125                 else if (fg == 4){
126                     printf("else:22    ");
127                     fputs("else|", fichero);
128                 }
129                 else if (fg == 5){
130                     printf("printf:23    ");
131                     fputs("printf|", fichero);
132                 }
133                 else if (fg == 47){
134                     printf("$:24    ");
135                     fputs("$|", fichero);
136                 }
137                 else if (fg == 34){
138                     printf(".:24    ");
139                     fputs(".|", fichero);
140                 }
141         }
142     } while( fg != 99 );
143     //cerrar fichero
144     fclose(fichero);
145     cout<<endl<<"Tokens Agregados con Exito."<<endl<<endl;
```

Fórmula Complejidad Ciclomática:

C(V) = Aristas – Nodos + 2
C(V) = 50 – 29 + 2
C(V) = 21



COCOMO

Tipo orgánico:

Tamaño	466
Tipo	Organic

$$PM_{nominal} = A_{PM} \cdot (KSLOC)^{B_{PM}}$$

PM	Organic	Semidetached	Embedded
A_{PM}	2.40	3.00	3.60
B_{PM}	1.05	1.12	1.20

$$TDEV = A_{TDEV}(PM)^{B_{TDEV}}$$

TDEV	Organic	Semidetached	Embedded
A_{TDEV}	2.50	2.50	2.50
B_{TDEV}	0.38	0.35	0.32

$$PM = PM_{nominal} \cdot \Pi_{i=1}^{15} EM_i$$

Parametros elegidos			
A_{PM}	2.40	A_{TDEV}	2.50
B_{PM}	1.05	B_{TDEV}	0.38

Esfuerzo	1.08	MM
Duracion	2.57	Meses
Team	0.42	Por persona

	Esfuerzo	Schedule
Planes y requisitos	0.1	0.3
Diseño	0.1	0.2
Desarrollo	0.7	1.7
Integracion y pruebas	0.3	0.7
Total	**1.1**	**2.9**

	Esfuerzo	Schedule
Planes y requisitos	6%	12%
Diseño	7%	8%
Desarrollo	62%	65%
Integracion y pruebas	31%	27%
Total	**100%**	**100%**

Tipo semi-acoplado:

Tamaño	466
Tipo	Semidetached

$$PM_{nominal} = A_{PM} \cdot (KSLOC)^{B_{PM}}$$

PM	Organic	Semidetached	Embedded
A_{PM}	2.40	3.00	3.60
B_{PM}	1.05	1.12	1.20

$$TDEV = A_{TDEV}(PM)^{B_{TDEV}}$$

TDEV	Organic	Semidetached	Embedded
A_{TDEV}	2.50	2.50	2.50
B_{TDEV}	0.38	0.35	0.32

$$PM = PM_{nominal} \cdot \Pi_{i=1}^{15} EM_i$$

Parametros elegidos			
A_{PM}	3.00	A_{TDEV}	2.50
B_{PM}	1.12	B_{TDEV}	0.35

Esfuerzo	1.28	MM
Duracion	2.72	Meses
Team	0.47	Por persona

	Esfuerzo	Schedule
Planes y requisitos	0.1	0.3
Diseño	0.1	0.2
Desarrollo	0.8	1.8
Integracion y pruebas	0.4	0.7
Total	1.4	3.0

	Esfuerzo	Schedule
Planes y requisitos	6%	12%
Diseño	7%	8%
Desarrollo	62%	65%
Integracion y pruebas	31%	27%
Total	100%	100%

Tipo empotrado:

Tamaño	466
Tipo	Embedded

$$PM_{nominal} = A_{PM} \cdot (KSLOC)^{B_{PM}}$$

PM	Organic	Semidetached	Embedded
A_{PM}	2.40	3.00	3.60
B_{PM}	1.05	1.12	1.20

$$TDEV = A_{TDEV}(PM)^{B_{TDEV}}$$

TDEV	Organic	Semidetached	Embedded
A_{TDEV}	2.50	2.50	2.50
B_{TDEV}	0.38	0.35	0.32

$$PM = PM_{nominal} \cdot \Pi_{i=i}^{15} EM_i$$

Parametros elegidos			
A_{PM}	**3.60**	A_{TDEV}	**2.50**
B_{PM}	**1.20**	B_{TDEV}	**0.32**

Esfuerzo	1.44	MM
Duracion	2.81	Meses
Team	0.51	Por persona

	Esfuerzo	Schedule
Planes y requisitos	0.1	0.3
Diseño	0.1	0.2
Desarrollo	0.9	1.8
Integracion y pruebas	0.4	0.8
Total	**1.5**	**3.1**

	Esfuerzo	Schedule
Planes y requisitos	6%	12%
Diseño	7%	8%
Desarrollo	62%	65%
Integracion y pruebas	31%	27%
Total	**100%**	**100%**

Pruebas Caja Negra y Caja Blanca