Análisis Completo del Compilador

1. Análisis Léxico

LINE	COL	TOKEN	LEXEME
8	1	FUNCTION	function
8	10	ID	fibonacci
8	19	LPAREN	(
8	20	ID	n
8	21	RPAREN	)
8	23	LBRACE	{
9	5	IF	if
9	8	LPAREN	(
9	9	ID	n
9	11	LE	<=
9	14	NUM	1
9	15	RPAREN	)
9	17	LBRACE	{
10	9	RETURN	return
10	16	ID	n
10	17	SEMI	;
11	5	RBRACE	}
11	7	ELSE	else

LINE	COL	TOKEN	LEXEME
11	12	LBRACE	{
12	9	RETURN	return
12	16	ID	fibonacci
12	25	LPAREN	(
12	26	ID	n
12	28	MINUS	-
12	30	NUM	1
12	31	RPAREN	)
12	33	PLUS	+
12	35	ID	fibonacci
12	44	LPAREN	(
12	45	ID	n
12	47	MINUS	-
12	49	NUM	2
12	50	RPAREN	)
12	51	SEMI	;
13	5	RBRACE	}
14	1	RBRACE	}
17	1	FUNCTION	function
17	10	ID	esPar
17	15	LPAREN	(
17	16	ID	n
17	17	RPAREN	)

LINE	COL	TOKEN	LEXEME
17	19	LBRACE	{
18	5	IF	if
18	8	LPAREN	(
18	9	ID	n
18	11	EQEQ	==
18	14	NUM	0
18	15	RPAREN	)
18	17	LBRACE	{
19	9	RETURN	return
19	16	TRUE	true
19	20	SEMI	;
20	5	RBRACE	}
20	7	ELSE	else
20	12	LBRACE	{
21	9	RETURN	return
21	16	ID	esImpar
21	23	LPAREN	(
21	24	ID	n
21	26	MINUS	-
21	28	NUM	1
21	29	RPAREN	)
21	30	SEMI	;
22	5	RBRACE	}

LINE	COL	TOKEN	LEXEME
23	1	RBRACE	}
25	1	FUNCTION	function
25	10	ID	esImpar
25	17	LPAREN	(
25	18	ID	n
25	19	RPAREN	)
25	21	LBRACE	{
26	5	IF	if
26	8	LPAREN	(
26	9	ID	n
26	11	EQEQ	==
26	14	NUM	0
26	15	RPAREN	)
26	17	LBRACE	{
27	9	RETURN	return
27	16	FALSE	false
27	21	SEMI	;
28	5	RBRACE	}
28	7	ELSE	else
28	12	LBRACE	{
29	9	RETURN	return
29	16	ID	esPar
29	21	LPAREN	(

LINE	COL	TOKEN	LEXEME
29	22	ID	n
29	24	MINUS	-
29	26	NUM	1
29	27	RPAREN	)
29	28	SEMI	;
30	5	RBRACE	}
31	1	RBRACE	}
34	1	LET	let
34	5	ID	expresionCompleja
34	23	ASSIGN	=
34	25	LPAREN	(
34	26	LPAREN	(
34	27	LPAREN	(
34	28	LPAREN	(
34	29	NUM	10
34	32	PLUS	+
34	34	NUM	5
34	35	RPAREN	)
34	37	STAR	*
34	39	NUM	3
34	40	RPAREN	)
34	42	MINUS	-
34	44	LPAREN	(

LINE	COL	TOKEN	LEXEME
34	45	NUM	8
34	47	SLASH	/
34	49	NUM	2
34	50	RPAREN	)
34	51	RPAREN	)
34	53	PLUS	+
34	55	LPAREN	(
34	56	LPAREN	(
34	57	NUM	15
34	60	MINUS	-
34	62	NUM	3
34	63	RPAREN	)
34	65	STAR	*
34	67	LPAREN	(
34	68	NUM	9
34	70	PLUS	+
34	72	NUM	1
34	73	RPAREN	)
34	74	RPAREN	)
34	75	RPAREN	)
34	77	SLASH	/
34	79	LPAREN	(
34	80	LPAREN	(

LINE	COL	TOKEN	LEXEME
34	81	LPAREN	(
34	82	NUM	5
34	84	STAR	*
34	86	NUM	2
34	87	RPAREN	)
34	89	PLUS	+
34	91	NUM	10
34	93	RPAREN	)
34	95	MINUS	-
34	97	LPAREN	(
34	98	LPAREN	(
34	99	NUM	3
34	101	PLUS	+
34	103	NUM	7
34	104	RPAREN	)
34	106	SLASH	/
34	108	NUM	2
34	109	RPAREN	)
34	110	RPAREN	)
34	111	SEMI	;
37	1	LET	let
37	5	ID	condicionCompleja
37	23	ASSIGN	=

LINE	COL	TOKEN	LEXEME
37	25	LPAREN	(
37	26	LPAREN	(
37	27	LPAREN	(
37	28	ID	x
37	30	GT	>
37	32	NUM	5
37	33	RPAREN	)
37	35	AND	&&
37	38	LPAREN	(
37	39	ID	y
37	41	LT	<
37	43	NUM	10
37	45	RPAREN	)
37	46	RPAREN	)
37	48	OR	||
37	51	LPAREN	(
37	52	LPAREN	(
37	53	ID	z
37	55	EQEQ	==
37	58	NUM	15
37	60	RPAREN	)
37	62	AND	&&
37	65	LPAREN	(

LINE	COL	TOKEN	LEXEME
37	66	ID	w
37	68	NEQ	!=
37	71	NUM	20
37	73	RPAREN	)
37	74	RPAREN	)
37	75	RPAREN	)
37	77	AND	&&
37	80	LPAREN	(
37	81	BANG	!
37	82	LPAREN	(
37	83	LPAREN	(
37	84	ID	a
37	86	GE	>=
37	89	NUM	3
37	90	RPAREN	)
37	92	OR	||
37	95	LPAREN	(
37	96	ID	b
37	98	LE	<=
37	101	NUM	7
37	102	RPAREN	)
37	103	RPAREN	)
37	104	RPAREN	)

LINE	COL	TOKEN	LEXEME
37	105	SEMI	;
40	1	FUNCTION	function
40	10	ID	nivel1
40	16	LPAREN	(
40	17	RPAREN	)
40	19	LBRACE	{
41	5	LET	let
41	9	ID	a
41	11	ASSIGN	=
41	13	NUM	1
41	14	SEMI	;
43	5	FUNCTION	function
43	14	ID	nivel2
43	20	LPAREN	(
43	21	RPAREN	)
43	23	LBRACE	{
44	9	LET	let
44	13	ID	b
44	15	ASSIGN	=
44	17	ID	a
44	19	PLUS	+
44	21	NUM	2
44	22	SEMI	;

LINE	COL	TOKEN	LEXEME
46	9	FUNCTION	function
46	18	ID	nivel3
46	24	LPAREN	(
46	25	RPAREN	)
46	27	LBRACE	{
47	13	LET	let
47	17	ID	c
47	19	ASSIGN	=
47	21	ID	b
47	23	PLUS	+
47	25	NUM	3
47	26	SEMI	;
49	13	FUNCTION	function
49	22	ID	nivel4
49	28	LPAREN	(
49	29	RPAREN	)
49	31	LBRACE	{
50	17	LET	let
50	21	ID	d
50	23	ASSIGN	=
50	25	ID	c
50	27	PLUS	+
50	29	NUM	4

LINE	COL	TOKEN	LEXEME
50	30	SEMI	;
52	17	FUNCTION	function
52	26	ID	nivel5
52	32	LPAREN	(
52	33	RPAREN	)
52	35	LBRACE	{
53	21	LET	let
53	25	ID	e
53	27	ASSIGN	=
53	29	ID	d
53	31	PLUS	+
53	33	NUM	5
53	34	SEMI	;
54	21	RETURN	return
54	28	ID	a
54	30	PLUS	+
54	32	ID	b
54	34	PLUS	+
54	36	ID	c
54	38	PLUS	+
54	40	ID	d
54	42	PLUS	+
54	44	ID	e

LINE	COL	TOKEN	LEXEME
54	45	SEMI	;
55	17	RBRACE	}
57	17	RETURN	return
57	24	ID	nivel5
57	30	LPAREN	(
57	31	RPAREN	)
57	32	SEMI	;
58	13	RBRACE	}
60	13	RETURN	return
60	20	ID	nivel4
60	26	LPAREN	(
60	27	RPAREN	)
60	28	SEMI	;
61	9	RBRACE	}
63	9	RETURN	return
63	16	ID	nivel3
63	22	LPAREN	(
63	23	RPAREN	)
63	24	SEMI	;
64	5	RBRACE	}
66	5	RETURN	return
66	12	ID	nivel2
66	18	LPAREN	(

LINE	COL	TOKEN	LEXEME
66	19	RPAREN	)
66	20	SEMI	;
67	1	RBRACE	}
70	1	LET	let
70	5	ID	x
70	7	ASSIGN	=
70	9	NUM	10
70	11	SEMI	;
72	1	FUNCTION	function
72	10	ID	testShadowing
72	23	LPAREN	(
72	24	RPAREN	)
72	26	LBRACE	{
73	5	LET	let
73	9	ID	x
73	11	ASSIGN	=
73	13	NUM	20
73	15	SEMI	;
75	5	IF	if
75	8	LPAREN	(
75	9	ID	x
75	11	GT	>
75	13	NUM	15

LINE	COL	TOKEN	LEXEME
75	15	RPAREN	)
75	17	LBRACE	{
76	9	LET	let
76	13	ID	x
76	15	ASSIGN	=
76	17	NUM	30
76	19	SEMI	;
78	9	WHILE	while
78	15	LPAREN	(
78	16	ID	x
78	18	GT	>
78	20	NUM	25
78	22	RPAREN	)
78	24	LBRACE	{
79	13	LET	let
79	17	ID	x
79	19	ASSIGN	=
79	21	NUM	40
79	23	SEMI	;
80	13	ID	print
80	18	LPAREN	(
80	19	ID	x
80	20	RPAREN	)

LINE	COL	TOKEN	LEXEME
80	21	SEMI	;
81	13	ID	x
81	15	ASSIGN	=
81	17	ID	x
81	19	MINUS	-
81	21	NUM	1
81	22	SEMI	;
82	9	RBRACE	}
84	9	ID	print
84	14	LPAREN	(
84	15	ID	x
84	16	RPAREN	)
84	17	SEMI	;
85	5	RBRACE	}
87	5	ID	print
87	10	LPAREN	(
87	11	ID	x
87	12	RPAREN	)
87	13	SEMI	;
88	1	RBRACE	}
91	1	FUNCTION	function
91	10	ID	operacionCompleja
91	27	LPAREN	(

LINE	COL	TOKEN	LEXEME
91	28	ID	a
91	29	COMMA	,
91	31	ID	b
91	32	COMMA	,
91	34	ID	c
91	35	COMMA	,
91	37	ID	d
91	38	COMMA	,
91	40	ID	e
91	41	RPAREN	)
91	43	LBRACE	{
92	5	RETURN	return
92	12	LPAREN	(
92	13	LPAREN	(
92	14	ID	a
92	16	PLUS	+
92	18	ID	b
92	19	RPAREN	)
92	21	STAR	*
92	23	LPAREN	(
92	24	ID	c
92	26	MINUS	-
92	28	ID	d

LINE	COL	TOKEN	LEXEME
92	29	RPAREN	)
92	30	RPAREN	)
92	32	SLASH	/
92	34	ID	e
92	35	SEMI	;
93	1	RBRACE	}
95	1	LET	let
95	5	ID	resultado1
95	16	ASSIGN	=
95	18	ID	operacionCompleja
95	35	LPAREN	(
96	5	ID	fibonacci
96	14	LPAREN	(
96	15	NUM	5
96	16	RPAREN	)
96	17	COMMA	,
97	5	ID	operacionCompleja
97	22	LPAREN	(
97	23	NUM	1
97	24	COMMA	,
97	26	NUM	2
97	27	COMMA	,
97	29	NUM	3

LINE	COL	TOKEN	LEXEME
97	30	COMMA	,
97	32	NUM	4
97	33	COMMA	,
97	35	NUM	5
97	36	RPAREN	)
97	37	COMMA	,
98	5	ID	esPar
98	10	LPAREN	(
98	11	NUM	10
98	13	RPAREN	)
98	14	COMMA	,
99	5	ID	esImpar
99	12	LPAREN	(
99	13	NUM	7
99	14	RPAREN	)
99	15	COMMA	,
100	5	ID	nivel1
100	11	LPAREN	(
100	12	RPAREN	)
101	1	RPAREN	)
101	2	SEMI	;
104	1	FUNCTION	function
104	10	ID	estructurasAnidadas

LINE	COL	TOKEN	LEXEME
104	29	LPAREN	(
104	30	ID	n
104	31	RPAREN	)
104	33	LBRACE	{
105	5	LET	let
105	9	ID	resultado
105	19	ASSIGN	=
105	21	NUM	0
105	22	SEMI	;
107	5	FOR	for
107	9	LPAREN	(
107	10	LET	let
107	14	ID	i
107	16	ASSIGN	=
107	18	NUM	0
107	19	SEMI	;
107	21	ID	i
107	23	LT	<
107	25	ID	n
107	26	SEMI	;
107	28	ID	i
107	30	ASSIGN	=
107	32	ID	i

LINE	COL	TOKEN	LEXEME
107	34	PLUS	+
107	36	NUM	1
107	37	RPAREN	)
107	39	LBRACE	{
108	9	IF	if
108	12	LPAREN	(
108	13	ID	i
108	15	PERCENT	%
108	17	NUM	2
108	19	EQEQ	==
108	22	NUM	0
108	23	RPAREN	)
108	25	LBRACE	{
109	13	WHILE	while
109	19	LPAREN	(
109	20	ID	resultado
109	30	LT	<
109	32	NUM	100
109	35	RPAREN	)
109	37	LBRACE	{
110	17	IF	if
110	20	LPAREN	(
110	21	ID	resultado

LINE	COL	TOKEN	LEXEME
110	31	GT	>
110	33	NUM	50
110	35	RPAREN	)
110	37	LBRACE	{
111	21	FOR	for
111	25	LPAREN	(
111	26	LET	let
111	30	ID	j
111	32	ASSIGN	=
111	34	NUM	0
111	35	SEMI	;
111	37	ID	j
111	39	LT	<
111	41	ID	i
111	42	SEMI	;
111	44	ID	j
111	46	ASSIGN	=
111	48	ID	j
111	50	PLUS	+
111	52	NUM	1
111	53	RPAREN	)
111	55	LBRACE	{
112	25	IF	if

LINE	COL	TOKEN	LEXEME
112	28	LPAREN	(
112	29	ID	j
112	31	PERCENT	%
112	33	NUM	3
112	35	EQEQ	==
112	38	NUM	0
112	39	RPAREN	)
112	41	LBRACE	{
113	29	ID	resultado
113	39	ASSIGN	=
113	41	ID	resultado
113	51	PLUS	+
113	53	ID	j
113	54	SEMI	;
114	25	RBRACE	}
114	27	ELSE	else
114	32	LBRACE	{
115	29	ID	resultado
115	39	ASSIGN	=
115	41	ID	resultado
115	51	MINUS	-
115	53	NUM	1
115	54	SEMI	;

LINE	COL	TOKEN	LEXEME
116	25	RBRACE	}
117	21	RBRACE	}
118	17	RBRACE	}
118	19	ELSE	else
118	24	LBRACE	{
119	21	ID	resultado
119	31	ASSIGN	=
119	33	ID	resultado
119	43	PLUS	+
119	45	ID	i
119	46	SEMI	;
120	17	RBRACE	}
122	17	IF	if
122	20	LPAREN	(
122	21	ID	resultado
122	31	GT	>
122	33	NUM	75
122	35	RPAREN	)
122	37	LBRACE	{
123	21	RETURN	return
123	28	ID	resultado
123	37	SEMI	;
124	17	RBRACE	}

LINE	COL	TOKEN	LEXEME
125	13	RBRACE	}
126	9	RBRACE	}
126	11	ELSE	else
126	16	LBRACE	{
127	13	IF	if
127	16	LPAREN	(
127	17	ID	i
127	19	GT	>
127	21	NUM	5
127	22	RPAREN	)
127	24	LBRACE	{
128	17	WHILE	while
128	23	LPAREN	(
128	24	ID	i
128	26	LT	<
128	28	NUM	20
128	30	RPAREN	)
128	32	LBRACE	{
129	21	ID	resultado
129	31	ASSIGN	=
129	33	ID	resultado
129	43	PLUS	+
129	45	NUM	1

LINE	COL	TOKEN	LEXEME
129	46	SEMI	;
130	21	ID	i
130	23	ASSIGN	=
130	25	ID	i
130	27	PLUS	+
130	29	NUM	1
130	30	SEMI	;
131	17	RBRACE	}
132	13	RBRACE	}
133	9	RBRACE	}
134	5	RBRACE	}
136	5	RETURN	return
136	12	ID	resultado
136	21	SEMI	;
137	1	RBRACE	}
140	1	FUNCTION	function
140	10	ID	sumar
140	15	LPAREN	(
140	16	ID	a
140	17	COMMA	,
140	19	ID	b
140	20	RPAREN	)
140	22	LBRACE	{

LINE	COL	TOKEN	LEXEME
141	5	RETURN	return
141	12	ID	a
141	14	PLUS	+
141	16	ID	b
141	17	SEMI	;
142	1	RBRACE	}
144	1	FUNCTION	function
144	10	ID	multiplicar
144	21	LPAREN	(
144	22	ID	a
144	23	COMMA	,
144	25	ID	b
144	26	RPAREN	)
144	28	LBRACE	{
145	5	RETURN	return
145	12	ID	a
145	14	STAR	*
145	16	ID	b
145	17	SEMI	;
146	1	RBRACE	}
148	1	FUNCTION	function
148	10	ID	restar
148	16	LPAREN	(

LINE	COL	TOKEN	LEXEME
148	17	ID	a
148	18	COMMA	,
148	20	ID	b
148	21	RPAREN	)
148	23	LBRACE	{
149	5	RETURN	return
149	12	ID	a
149	14	MINUS	-
149	16	ID	b
149	17	SEMI	;
150	1	RBRACE	}
152	1	LET	let
152	5	ID	resultadoAnidado
152	22	ASSIGN	=
152	24	ID	sumar
152	29	LPAREN	(
153	5	ID	multiplicar
153	16	LPAREN	(
153	17	NUM	3
153	18	COMMA	,
153	20	NUM	4
153	21	RPAREN	)
153	22	COMMA	,

LINE	COL	TOKEN	LEXEME
154	5	ID	restar
154	11	LPAREN	(
155	9	ID	sumar
155	14	LPAREN	(
155	15	NUM	10
155	17	COMMA	,
155	19	NUM	5
155	20	RPAREN	)
155	21	COMMA	,
156	9	ID	multiplicar
156	20	LPAREN	(
156	21	NUM	2
156	22	COMMA	,
156	24	NUM	3
156	25	RPAREN	)
157	5	RPAREN	)
158	1	RPAREN	)
158	2	SEMI	;
161	1	LET	let
161	5	ID	mensaje1
161	14	ASSIGN	=
161	16	STRING	"Hola"
161	22	SEMI	;

LINE	COL	TOKEN	LEXEME
162	1	LET	let
162	5	ID	mensaje2
162	14	ASSIGN	=
162	16	STRING	"Mundo"
162	23	SEMI	;
163	1	LET	let
163	5	ID	numero
163	12	ASSIGN	=
163	14	NUM	42
163	16	SEMI	;
164	1	LET	let
164	5	ID	booleano
164	14	ASSIGN	=
164	16	TRUE	true
164	20	SEMI	;
166	1	LET	let
166	5	ID	concatenacion
166	19	ASSIGN	=
166	21	ID	mensaje1
166	30	PLUS	+
166	32	STRING	" "
166	36	PLUS	+
166	38	ID	mensaje2

LINE	COL	TOKEN	LEXEME
166	46	SEMI	;
170	1	LET	let
170	5	ID	variableNoUsada1
170	22	ASSIGN	=
170	24	NUM	100
170	27	SEMI	;
171	1	LET	let
171	5	ID	variableNoUsada2
171	22	ASSIGN	=
171	24	STRING	"nunca usada"
171	37	SEMI	;
172	1	CONST	const
172	7	ID	CONSTANTE_NO_USADA
172	26	ASSIGN	=
172	28	NUM	3.14159
172	35	SEMI	;
174	1	FUNCTION	function
174	10	ID	funcionNoLlamada
174	26	LPAREN	(
174	27	RPAREN	)
174	29	LBRACE	{
175	5	RETURN	return
175	12	NUM	42

LINE	COL	TOKEN	LEXEME
175	14	SEMI	;
176	1	RBRACE	}
184	1	LET	let
184	5	ID	verdadero
184	15	ASSIGN	=
184	17	TRUE	true
184	21	SEMI	;
185	1	LET	let
185	5	ID	falso
185	11	ASSIGN	=
185	13	FALSE	false
185	18	SEMI	;
187	1	LET	let
187	5	ID	logicaCompleja
187	20	ASSIGN	=
187	22	LPAREN	(
187	23	LPAREN	(
187	24	ID	verdadero
187	34	AND	&&
187	37	BANG	!
187	38	ID	falso
187	43	RPAREN	)
187	45	OR	||

LINE	COL	TOKEN	LEXEME
187	48	LPAREN	(
187	49	ID	falso
187	55	AND	&&
187	58	ID	verdadero
187	67	RPAREN	)
187	68	RPAREN	)
187	70	AND	&&
188	22	LPAREN	(
188	23	BANG	!
188	24	LPAREN	(
188	25	ID	verdadero
188	35	AND	&&
188	38	ID	falso
188	43	RPAREN	)
188	45	OR	||
188	48	LPAREN	(
188	49	BANG	!
188	50	ID	falso
188	56	OR	||
188	59	ID	verdadero
188	68	RPAREN	)
188	69	RPAREN	)
188	70	SEMI	;

LINE	COL	TOKEN	LEXEME
191	1	LET	let
191	5	ID	contador
191	14	ASSIGN	=
191	16	NUM	0
191	17	SEMI	;
192	1	WHILE	while
192	7	LPAREN	(
192	8	LPAREN	(
192	9	ID	contador
192	18	LT	<
192	20	NUM	100
192	23	RPAREN	)
192	25	AND	&&
192	28	LPAREN	(
192	29	LPAREN	(
192	30	ID	contador
192	39	PERCENT	%
192	41	NUM	2
192	43	EQEQ	==
192	46	NUM	0
192	47	RPAREN	)
192	49	OR	||
192	52	LPAREN	(

LINE	COL	TOKEN	LEXEME
192	53	ID	contador
192	62	PERCENT	%
192	64	NUM	3
192	66	EQEQ	==
192	69	NUM	0
192	70	RPAREN	)
192	71	RPAREN	)
192	72	RPAREN	)
192	74	LBRACE	{
193	5	IF	if
193	8	LPAREN	(
193	9	LPAREN	(
193	10	ID	contador
193	19	PERCENT	%
193	21	NUM	5
193	23	EQEQ	==
193	26	NUM	0
193	27	RPAREN	)
193	29	AND	&&
193	32	LPAREN	(
193	33	ID	contador
193	42	GT	>
193	44	NUM	20

LINE	COL	TOKEN	LEXEME
193	46	RPAREN	)
193	47	RPAREN	)
193	49	LBRACE	{
194	9	ID	contador
194	18	ASSIGN	=
194	20	ID	contador
194	29	PLUS	+
194	31	NUM	3
194	32	SEMI	;
195	5	RBRACE	}
195	7	ELSE	else
195	12	LBRACE	{
196	9	ID	contador
196	18	ASSIGN	=
196	20	ID	contador
196	29	PLUS	+
196	31	NUM	1
196	32	SEMI	;
197	5	RBRACE	}
198	1	RBRACE	}
201	1	FOR	for
201	5	LPAREN	(
201	6	LET	let

LINE	COL	TOKEN	LEXEME
201	10	ID	idx
201	14	ASSIGN	=
201	16	NUM	0
201	17	SEMI	;
201	19	ID	idx
201	23	LT	<
201	25	ID	fibonacci
201	34	LPAREN	(
201	35	NUM	8
201	36	RPAREN	)
201	37	SEMI	;
201	39	ID	idx
201	43	ASSIGN	=
201	45	ID	idx
201	49	PLUS	+
201	51	ID	esPar
201	56	LPAREN	(
201	57	ID	idx
201	60	RPAREN	)
201	61	RPAREN	)
201	63	LBRACE	{
202	5	LET	let
202	9	ID	temp

LINE	COL	TOKEN	LEXEME
202	14	ASSIGN	=
202	16	ID	idx
202	20	STAR	*
202	22	NUM	2
202	23	SEMI	;
204	5	IF	if
204	8	LPAREN	(
204	9	ID	temp
204	14	GT	>
204	16	NUM	50
204	18	RPAREN	)
204	20	LBRACE	{
205	9	FOR	for
205	13	LPAREN	(
205	14	LET	let
205	18	ID	inner
205	24	ASSIGN	=
205	26	ID	temp
205	30	SEMI	;
205	32	ID	inner
205	38	GT	>
205	40	NUM	0
205	41	SEMI	;

LINE	COL	TOKEN	LEXEME
205	43	ID	inner
205	49	ASSIGN	=
205	51	ID	inner
205	57	MINUS	-
205	59	NUM	5
205	60	RPAREN	)
205	62	LBRACE	{
206	13	LET	let
206	17	ID	squared
206	25	ASSIGN	=
206	27	ID	inner
206	33	STAR	*
206	35	ID	inner
206	40	SEMI	;
207	13	ID	print
207	18	LPAREN	(
207	19	ID	squared
207	26	RPAREN	)
207	27	SEMI	;
208	9	RBRACE	}
209	5	RBRACE	}
210	1	RBRACE	}
213	1	LET	let

LINE	COL	TOKEN	LEXEME
213	5	ID	negativo
213	14	ASSIGN	=
213	16	MINUS	-
213	17	NUM	10
213	19	SEMI	;
214	1	LET	let
214	5	ID	dobleNegacion
214	19	ASSIGN	=
214	21	MINUS	-
214	22	MINUS	-
214	23	ID	negativo
214	31	SEMI	;
215	1	LET	let
215	5	ID	negacionLogica
215	20	ASSIGN	=
215	22	BANG	!
215	23	ID	verdadero
215	32	SEMI	;
216	1	LET	let
216	5	ID	combinado
216	15	ASSIGN	=
216	17	BANG	!
216	18	MINUS	-

LINE	COL	TOKEN	LEXEME
216	19	LPAREN	(
216	20	MINUS	-
216	21	NUM	5
216	22	RPAREN	)
216	23	SEMI	;
219	1	FUNCTION	function
219	10	ID	conMultiplesReturns
219	29	LPAREN	(
219	30	ID	n
219	31	RPAREN	)
219	33	LBRACE	{
220	5	IF	if
220	8	LPAREN	(
220	9	ID	n
220	11	LT	<
220	13	NUM	0
220	14	RPAREN	)
220	16	LBRACE	{
221	9	RETURN	return
221	16	MINUS	-
221	17	NUM	1
221	18	SEMI	;
222	5	RBRACE	}

LINE	COL	TOKEN	LEXEME
224	5	IF	if
224	8	LPAREN	(
224	9	ID	n
224	11	EQEQ	==
224	14	NUM	0
224	15	RPAREN	)
224	17	LBRACE	{
225	9	RETURN	return
225	16	NUM	0
225	17	SEMI	;
226	5	RBRACE	}
228	5	FOR	for
228	9	LPAREN	(
228	10	LET	let
228	14	ID	i
228	16	ASSIGN	=
228	18	NUM	0
228	19	SEMI	;
228	21	ID	i
228	23	LT	<
228	25	ID	n
228	26	SEMI	;
228	28	ID	i

LINE	COL	TOKEN	LEXEME
228	30	ASSIGN	=
228	32	ID	i
228	34	PLUS	+
228	36	NUM	1
228	37	RPAREN	)
228	39	LBRACE	{
229	9	IF	if
229	12	LPAREN	(
229	13	ID	i
229	15	GT	>
229	17	NUM	10
229	19	RPAREN	)
229	21	LBRACE	{
230	13	RETURN	return
230	20	ID	i
230	21	SEMI	;
231	9	RBRACE	}
232	5	RBRACE	}
234	5	RETURN	return
234	12	ID	n
234	14	STAR	*
234	16	NUM	2
234	17	SEMI	;

LINE	COL	TOKEN	LEXEME
235	1	RBRACE	}
238	1	FUNCTION	function
238	10	ID	factorial
238	19	LPAREN	(
238	20	ID	n
238	21	RPAREN	)
238	23	LBRACE	{
239	5	IF	if
239	8	LPAREN	(
239	9	ID	n
239	11	LE	<=
239	14	NUM	1
239	15	RPAREN	)
239	17	LBRACE	{
240	9	RETURN	return
240	16	NUM	1
240	17	SEMI	;
241	5	RBRACE	}
242	5	RETURN	return
242	12	ID	n
242	14	STAR	*
242	16	ID	factorial
242	25	LPAREN	(

LINE	COL	TOKEN	LEXEME
242	26	ID	n
242	28	MINUS	-
242	30	NUM	1
242	31	RPAREN	)
242	32	SEMI	;
243	1	RBRACE	}
245	1	LET	let
245	5	ID	fact5
245	11	ASSIGN	=
245	13	ID	factorial
245	22	LPAREN	(
245	23	NUM	5
245	24	RPAREN	)
245	25	SEMI	;
246	1	LET	let
246	5	ID	fact10
246	12	ASSIGN	=
246	14	ID	factorial
246	23	LPAREN	(
246	24	ID	factorial
246	33	LPAREN	(
246	34	NUM	3
246	35	RPAREN	)

LINE	COL	TOKEN	LEXEME
246	36	RPAREN	)
246	37	SEMI	;
249	1	LET	let
249	5	ID	a
249	7	ASSIGN	=
249	9	NUM	5
249	10	SEMI	;
250	1	LET	let
250	5	ID	b
250	7	ASSIGN	=
250	9	NUM	10
250	11	SEMI	;
251	1	LET	let
251	5	ID	c
251	7	ASSIGN	=
251	9	NUM	15
251	11	SEMI	;
252	1	LET	let
252	5	ID	d
252	7	ASSIGN	=
252	9	NUM	20
252	11	SEMI	;
254	1	LET	let

LINE	COL	TOKEN	LEXEME
254	5	ID	comparacion
254	17	ASSIGN	=
254	19	LPAREN	(
254	20	LPAREN	(
254	21	ID	a
254	23	LT	<
254	25	ID	b
254	26	RPAREN	)
254	28	AND	&&
254	31	LPAREN	(
254	32	ID	b
254	34	LT	<
254	36	ID	c
254	37	RPAREN	)
254	38	RPAREN	)
254	40	AND	&&
254	43	LPAREN	(
254	44	LPAREN	(
254	45	ID	c
254	47	LT	<
254	49	ID	d
254	50	RPAREN	)
254	52	OR	||

LINE	COL	TOKEN	LEXEME
254	55	LPAREN	(
254	56	ID	a
254	58	EQEQ	==
254	61	ID	b
254	62	RPAREN	)
254	63	RPAREN	)
254	64	SEMI	;
257	1	IF	if
257	4	LPAREN	(
257	5	ID	verdadero
257	14	RPAREN	)
257	16	LBRACE	{
259	5	LET	let
259	9	ID	x
259	11	ASSIGN	=
259	13	NUM	1
259	14	SEMI	;
260	1	RBRACE	}
262	1	WHILE	while
262	7	LPAREN	(
262	8	ID	falso
262	13	RPAREN	)
262	15	LBRACE	{

LINE	COL	TOKEN	LEXEME
264	5	LET	let
264	9	ID	imposible
264	19	ASSIGN	=
264	21	NUM	999
264	24	SEMI	;
265	1	RBRACE	}
268	1	LET	let
268	5	ID	v1
268	8	ASSIGN	=
268	10	NUM	10
268	12	SEMI	;
269	1	LET	let
269	5	ID	v2
269	8	ASSIGN	=
269	10	ID	v1
269	12	SEMI	;
270	1	LET	let
270	5	ID	v3
270	8	ASSIGN	=
270	10	ID	v2
270	12	SEMI	;
271	1	LET	let
271	5	ID	v4

LINE	COL	TOKEN	LEXEME
271	8	ASSIGN	=
271	10	ID	v3
271	12	SEMI	;
272	1	ID	v1
272	4	ASSIGN	=
272	6	ID	v2
272	9	ASSIGN	=
272	11	ID	v3
272	14	ASSIGN	=
272	16	ID	v4
272	18	SEMI	;
275	1	ID	print
275	6	LPAREN	(
275	7	ID	fibonacci
275	16	LPAREN	(
275	17	NUM	5
275	18	RPAREN	)
275	20	PLUS	+
275	22	ID	factorial
275	31	LPAREN	(
275	32	NUM	4
275	33	RPAREN	)
275	34	RPAREN	)

LINE	COL	TOKEN	LEXEME
275	35	SEMI	;
276	1	ID	print
276	6	LPAREN	(
276	7	ID	sumar
276	12	LPAREN	(
276	13	ID	multiplicar
276	24	LPAREN	(
276	25	NUM	2
276	26	COMMA	,
276	28	NUM	3
276	29	RPAREN	)
276	30	COMMA	,
276	32	ID	restar
276	38	LPAREN	(
276	39	NUM	10
276	41	COMMA	,
276	43	NUM	5
276	44	RPAREN	)
276	45	RPAREN	)
276	46	RPAREN	)
276	47	SEMI	;
278	1	LET	let
278	5	ID	entrada

LINE	COL	TOKEN	LEXEME
278	13	ASSIGN	=
278	15	ID	input
278	20	LPAREN	(
278	21	RPAREN	)
278	22	SEMI	;
279	1	LET	let
279	5	ID	numeroEntrada
279	19	ASSIGN	=
279	21	ID	parseInt
279	29	LPAREN	(
279	30	ID	entrada
279	37	RPAREN	)
279	38	SEMI	;
282	1	LET	let
282	5	ID	variable
282	14	ASSIGN	=
282	16	NUM	10
282	18	SEMI	;
283	1	LET	let
283	5	ID	Variable
283	14	ASSIGN	=
283	16	NUM	20
283	18	SEMI	;

LINE	COL	TOKEN	LEXEME
284	1	LET	let
284	5	ID	variabLe
284	14	ASSIGN	=
284	16	NUM	30
284	18	SEMI	;
285	1	LET	let
285	5	ID	variabl3
285	14	ASSIGN	=
285	16	NUM	40
285	18	SEMI	;
288	1	FUNCTION	function
288	10	ID	programaPrincipal
288	27	LPAREN	(
288	28	RPAREN	)
288	30	LBRACE	{
289	5	LET	let
289	9	ID	resultado
289	19	ASSIGN	=
289	21	NUM	0
289	22	SEMI	;
292	5	ID	resultado
292	15	ASSIGN	=
292	17	ID	resultado

LINE	COL	TOKEN	LEXEME
292	27	PLUS	+
292	29	ID	fibonacci
292	38	LPAREN	(
292	39	NUM	10
292	41	RPAREN	)
292	42	SEMI	;
293	5	ID	resultado
293	15	ASSIGN	=
293	17	ID	resultado
293	27	PLUS	+
293	29	ID	nivel1
293	35	LPAREN	(
293	36	RPAREN	)
293	37	SEMI	;
294	5	ID	resultado
294	15	ASSIGN	=
294	17	ID	resultado
294	27	PLUS	+
294	29	ID	estructurasAnidadas
294	48	LPAREN	(
294	49	NUM	5
294	50	RPAREN	)
294	51	SEMI	;

LINE	COL	TOKEN	LEXEME
295	5	ID	resultado
295	15	ASSIGN	=
295	17	ID	resultado
295	27	PLUS	+
295	29	ID	factorial
295	38	LPAREN	(
295	39	NUM	5
295	40	RPAREN	)
295	41	SEMI	;
298	5	IF	if
298	8	LPAREN	(
298	9	LPAREN	(
298	10	ID	resultado
298	20	GT	>
298	22	NUM	100
298	25	RPAREN	)
298	27	AND	&&
298	30	LPAREN	(
298	31	ID	resultado
298	41	LT	<
298	43	NUM	1000
298	47	RPAREN	)
298	48	RPAREN	)

LINE	COL	TOKEN	LEXEME
298	50	LBRACE	{
299	9	FOR	for
299	13	LPAREN	(
299	14	LET	let
299	18	ID	i
299	20	ASSIGN	=
299	22	NUM	0
299	23	SEMI	;
299	25	ID	i
299	27	LT	<
299	29	NUM	10
299	31	SEMI	;
299	33	ID	i
299	35	ASSIGN	=
299	37	ID	i
299	39	PLUS	+
299	41	NUM	1
299	42	RPAREN	)
299	44	LBRACE	{
300	13	ID	resultado
300	23	ASSIGN	=
300	25	ID	resultado
300	35	PLUS	+

LINE	COL	TOKEN	LEXEME
300	37	ID	conMultiplesReturns
300	56	LPAREN	(
300	57	ID	i
300	58	RPAREN	)
300	59	SEMI	;
301	9	RBRACE	}
302	5	RBRACE	}
305	5	IF	if
305	8	LPAREN	(
305	9	ID	esPar
305	14	LPAREN	(
305	15	ID	resultado
305	24	RPAREN	)
305	25	RPAREN	)
305	27	LBRACE	{
306	9	ID	print
306	14	LPAREN	(
306	15	STRING	"El resultado es par"
306	36	RPAREN	)
306	37	SEMI	;
307	5	RBRACE	}
307	7	ELSE	else
307	12	LBRACE	{

LINE	COL	TOKEN	LEXEME
308	9	ID	print
308	14	LPAREN	(
308	15	STRING	"El resultado es impar"
308	38	RPAREN	)
308	39	SEMI	;
309	5	RBRACE	}
311	5	RETURN	return
311	12	ID	resultado
311	21	SEMI	;
312	1	RBRACE	}
315	1	LET	let
315	5	ID	resultadoFinal
315	20	ASSIGN	=
315	22	ID	programaPrincipal
315	39	LPAREN	(
315	40	RPAREN	)
315	41	SEMI	;
316	1	ID	print
316	6	LPAREN	(
316	7	STRING	"Resultado final: "
316	26	RPAREN	)
316	27	SEMI	;
317	1	ID	print

LINE	COL	TOKEN	LEXEME
317	6	LPAREN	(
317	7	ID	resultadoFinal
317	21	RPAREN	)
317	22	SEMI	;
328	1	EOF	EOF

Total de tokens: 1334

2. Análisis Sintáctico

```
Program:
FunctionDecl(fibonacci(n))
IfStmt:
Condition:
BinaryOp(<=)
Identifier(n)
Literal(number: 1)
Then:
Block:
ReturnStmt:
Identifier(n)
Else:
Block:
ReturnStmt:
BinaryOp(+)
CallExpr:
Callee:
Identifier(fibonacci)
Args:
BinaryOp(-)
Identifier(n)
Literal(number: 1)
CallExpr:
Callee:
Identifier(fibonacci)
Args:
BinaryOp(-)
Identifier(n)
Literal(number: 2)
FunctionDecl(esPar(n))
IfStmt:
Condition:
BinaryOp(==)
Identifier(n)
Literal(number: 0)
Then:
Block:
ReturnStmt:
Literal(boolean: True)
Else:
Block:
ReturnStmt:
CallExpr:
```

```
Callee:
Identifier(esImpar)
Args:
BinaryOp(-)
Identifier(n)
Literal(number: 1)
FunctionDecl(esImpar(n))
IfStmt:
Condition:
BinaryOp(==)
Identifier(n)
Literal(number: 0)
Then:
Block:
ReturnStmt:
Literal(boolean: False)
Else:
Block:
ReturnStmt:
CallExpr:
Callee:
Identifier(esPar)
Args:
BinaryOp(-)
Identifier(n)
Literal(number: 1)
VarDecl(let expresionCompleja)
BinaryOp(/)
BinaryOp(+)
BinaryOp(-)
BinaryOp(*)
BinaryOp(+)
Literal(number: 10)
Literal(number: 5)
Literal(number: 3)
BinaryOp(/)
Literal(number: 8)
Literal(number: 2)
BinaryOp(*)
BinaryOp(-)
Literal(number: 15)
Literal(number: 3)
BinaryOp(+)
Literal(number: 9)
Literal(number: 1)
BinaryOp(-)
BinaryOp(+)
BinaryOp(*)
Literal(number: 5)
Literal(number: 2)
Literal(number: 10)
BinaryOp(/)
BinaryOp(+)
Literal(number: 3)
Literal(number: 7)
Literal(number: 2)
VarDecl(let condicionCompleja)
BinaryOp(&&)
BinaryOp(||)
BinaryOp(&&)
BinaryOp(>)
Identifier(x)
Literal(number: 5)
BinaryOp(<)
Identifier(y)
Literal(number: 10)
BinaryOp(&&)
BinaryOp(==)
Identifier(z)
Literal(number: 15)
BinaryOp(!=)
Identifier(w)
Literal(number: 20)
UnaryOp(!)
BinaryOp(||)
BinaryOp(>=)
```

```
Identifier(a)
Literal(number: 3)
BinaryOp(<=)
Identifier(b)
Literal(number: 7)
FunctionDecl(nivel1())
VarDecl(let a)
Literal(number: 1)
FunctionDecl(nivel2())
VarDecl(let b)
BinaryOp(+)
Identifier(a)
Literal(number: 2)
FunctionDecl(nivel3())
VarDecl(let c)
BinaryOp(+)
Identifier(b)
Literal(number: 3)
FunctionDecl(nivel4())
VarDecl(let d)
BinaryOp(+)
Identifier(c)
Literal(number: 4)
FunctionDecl(nivel5())
VarDecl(let e)
BinaryOp(+)
Identifier(d)
Literal(number: 5)
ReturnStmt:
BinaryOp(+)
BinaryOp(+)
BinaryOp(+)
BinaryOp(+)
Identifier(a)
Identifier(b)
Identifier(c)
Identifier(d)
Identifier(e)
ReturnStmt:
CallExpr:
Callee:
Identifier(nivel5)
ReturnStmt:
CallExpr:
Callee:
Identifier(nivel4)
ReturnStmt:
CallExpr:
Callee:
Identifier(nivel3)
ReturnStmt:
CallExpr:
Callee:
Identifier(nivel2)
VarDecl(let x)
Literal(number: 10)
FunctionDecl(testShadowing())
VarDecl(let x)
Literal(number: 20)
IfStmt:
Condition:
BinaryOp(>)
Identifier(x)
Literal(number: 15)
Then:
Block:
VarDecl(let x)
Literal(number: 30)
WhileStmt:
Condition:
BinaryOp(>)
Identifier(x)
Literal(number: 25)
Body:
Block:
VarDecl(let x)
```

```
Literal(number: 40)
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
Identifier(x)
ExprStmt:
Assignment(x)
BinaryOp(-)
Identifier(x)
Literal(number: 1)
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
Identifier(x)
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
Identifier(x)
FunctionDecl(operacionCompleja(a, b, c, d, e))
ReturnStmt:
BinaryOp(/)
BinaryOp(*)
BinaryOp(+)
Identifier(a)
Identifier(b)
BinaryOp(-)
Identifier(c)
Identifier(d)
Identifier(e)
VarDecl(let resultado1)
CallExpr:
Callee:
Identifier(operacionCompleja)
Args:
CallExpr:
Callee:
Identifier(fibonacci)
Args:
Literal(number: 5)
CallExpr:
Callee:
Identifier(operacionCompleja)
Args:
Literal(number: 1)
Literal(number: 2)
Literal(number: 3)
Literal(number: 4)
Literal(number: 5)
CallExpr:
Callee:
Identifier(esPar)
Args:
Literal(number: 10)
CallExpr:
Callee:
Identifier(esImpar)
Args:
Literal(number: 7)
CallExpr:
Callee:
Identifier(nivel1)
FunctionDecl(estructurasAnidadas(n))
VarDecl(let resultado)
Literal(number: 0)
ForStmt:
Init:
VarDecl(let i)
Literal(number: 0)
Condition:
BinaryOp(<)
```

```
Identifier(i)
Identifier(n)
Update:
Assignment(i)
BinaryOp(+)
Identifier(i)
Literal(number: 1)
Body:
Block:
IfStmt:
Condition:
BinaryOp(==)
BinaryOp(%)
Identifier(i)
Literal(number: 2)
Literal(number: 0)
Then:
Block:
WhileStmt:
Condition:
BinaryOp(<)
Identifier(resultado)
Literal(number: 100)
Body:
Block:
IfStmt:
Condition:
BinaryOp(>)
Identifier(resultado)
Literal(number: 50)
Then:
Block:
ForStmt:
Init:
VarDecl(let j)
Literal(number: 0)
Condition:
BinaryOp(<)
Identifier(j)
Identifier(i)
Update:
Assignment(j)
BinaryOp(+)
Identifier(j)
Literal(number: 1)
Body:
Block:
IfStmt:
Condition:
BinaryOp(==)
BinaryOp(%)
Identifier(j)
Literal(number: 3)
Literal(number: 0)
Then:
Block:
ExprStmt:
Assignment(resultado)
BinaryOp(+)
Identifier(resultado)
Identifier(j)
Else:
Block:
ExprStmt:
Assignment(resultado)
BinaryOp(-)
Identifier(resultado)
Literal(number: 1)
Else:
Block:
ExprStmt:
Assignment(resultado)
BinaryOp(+)
Identifier(resultado)
Identifier(i)
IfStmt:
```

```
Condition:
BinaryOp(>)
Identifier(resultado)
Literal(number: 75)
Then:
Block:
ReturnStmt:
Identifier(resultado)
Else:
Block:
IfStmt:
Condition:
BinaryOp(>)
Identifier(i)
Literal(number: 5)
Then:
Block:
WhileStmt:
Condition:
BinaryOp(<)
Identifier(i)
Literal(number: 20)
Body:
Block:
ExprStmt:
Assignment(resultado)
BinaryOp(+)
Identifier(resultado)
Literal(number: 1)
ExprStmt:
Assignment(i)
BinaryOp(+)
Identifier(i)
Literal(number: 1)
ReturnStmt:
Identifier(resultado)
FunctionDecl(sumar(a, b))
ReturnStmt:
BinaryOp(+)
Identifier(a)
Identifier(b)
FunctionDecl(multiplicar(a, b))
ReturnStmt:
BinaryOp(*)
Identifier(a)
Identifier(b)
FunctionDecl(restar(a, b))
ReturnStmt:
BinaryOp(-)
Identifier(a)
Identifier(b)
VarDecl(let resultadoAnidado)
CallExpr:
Callee:
Identifier(sumar)
Args:
CallExpr:
Callee:
Identifier(multiplicar)
Args:
Literal(number: 3)
Literal(number: 4)
CallExpr:
Callee:
Identifier(restar)
Args:
CallExpr:
Callee:
Identifier(sumar)
Args:
Literal(number: 10)
Literal(number: 5)
CallExpr:
Callee:
Identifier(multiplicar)
Args:
```

```
Literal(number: 2)
Literal(number: 3)
VarDecl(let mensaje1)
Literal(string: Hola)
VarDecl(let mensaje2)
Literal(string: Mundo)
VarDecl(let numero)
Literal(number: 42)
VarDecl(let booleano)
Literal(boolean: True)
VarDecl(let concatenacion)
BinaryOp(+)
BinaryOp(+)
Identifier(mensaje1)
Literal(string: )
Identifier(mensaje2)
VarDecl(let variableNoUsada1)
Literal(number: 100)
VarDecl(let variableNoUsada2)
Literal(string: nunca usada)
VarDecl(const CONSTANTE_NO_USADA)
Literal(number: 3.14159)
FunctionDecl(funcionNoLlamada())
ReturnStmt:
Literal(number: 42)
VarDecl(let verdadero)
Literal(boolean: True)
VarDecl(let falso)
Literal(boolean: False)
VarDecl(let logicaCompleja)
BinaryOp(&&)
BinaryOp(||)
BinaryOp(&&)
Identifier(verdadero)
UnaryOp(!)
Identifier(falso)
BinaryOp(&&)
Identifier(falso)
Identifier(verdadero)
BinaryOp(||)
UnaryOp(!)
BinaryOp(&&)
Identifier(verdadero)
Identifier(falso)
BinaryOp(||)
UnaryOp(!)
Identifier(falso)
Identifier(verdadero)
VarDecl(let contador)
Literal(number: 0)
WhileStmt:
Condition:
BinaryOp(&&)
BinaryOp(<)
Identifier(contador)
Literal(number: 100)
BinaryOp(||)
BinaryOp(==)
BinaryOp(%)
Identifier(contador)
Literal(number: 2)
Literal(number: 0)
BinaryOp(==)
BinaryOp(%)
Identifier(contador)
Literal(number: 3)
Literal(number: 0)
Body:
Block:
IfStmt:
Condition:
BinaryOp(&&)
BinaryOp(==)
BinaryOp(%)
Identifier(contador)
Literal(number: 5)
Literal(number: 0)
BinaryOp(>)
Identifier(contador)
Literal(number: 20)
Then:
Block:
ExprStmt:
Assignment(contador)
BinaryOp(+)
Identifier(contador)
Literal(number: 3)
Else:
Block:
ExprStmt:
Assignment(contador)
BinaryOp(+)
Identifier(contador)
Literal(number: 1)
ForStmt:
Init:
VarDecl(let idx)
Literal(number: 0)
Condition:
BinaryOp(<)
Identifier(idx)
CallExpr:
Callee:
Identifier(fibonacci)
Args:
Literal(number: 8)
Update:
Assignment(idx)
BinaryOp(+)
Identifier(idx)
CallExpr:
Callee:
Identifier(esPar)
Args:
Identifier(idx)
Body:
Block:
VarDecl(let temp)
BinaryOp(*)
Identifier(idx)
Literal(number: 2)
IfStmt:
Condition:
BinaryOp(>)
Identifier(temp)
Literal(number: 50)
Then:
Block:
ForStmt:
Init:
VarDecl(let inner)
Identifier(temp)
Condition:
BinaryOp(>)
Identifier(inner)
Literal(number: 0)
Update:
Assignment(inner)
BinaryOp(-)
Identifier(inner)
Literal(number: 5)
Body:
Block:
VarDecl(let squared)
BinaryOp(*)
Identifier(inner)
Identifier(inner)
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
```

```
Identifier(squared)
VarDecl(let negativo)
UnaryOp(-)
Literal(number: 10)
VarDecl(let dobleNegacion)
UnaryOp(-)
UnaryOp(-)
Identifier(negativo)
VarDecl(let negacionLogica)
UnaryOp(!)
Identifier(verdadero)
VarDecl(let combinado)
UnaryOp(!)
UnaryOp(-)
UnaryOp(-)
Literal(number: 5)
FunctionDecl(conMultiplesReturns(n))
IfStmt:
Condition:
BinaryOp(<)
Identifier(n)
Literal(number: 0)
Then:
Block:
ReturnStmt:
UnaryOp(-)
Literal(number: 1)
IfStmt:
Condition:
BinaryOp(==)
Identifier(n)
Literal(number: 0)
Then:
Block:
ReturnStmt:
Literal(number: 0)
ForStmt:
Init:
VarDecl(let i)
Literal(number: 0)
Condition:
BinaryOp(<)
Identifier(i)
Identifier(n)
Update:
Assignment(i)
BinaryOp(+)
Identifier(i)
Literal(number: 1)
Body:
Block:
IfStmt:
Condition:
BinaryOp(>)
Identifier(i)
Literal(number: 10)
Then:
Block:
ReturnStmt:
Identifier(i)
ReturnStmt:
BinaryOp(*)
Identifier(n)
Literal(number: 2)
FunctionDecl(factorial(n))
IfStmt:
Condition:
BinaryOp(<=)
Identifier(n)
Literal(number: 1)
Then:
Block:
ReturnStmt:
Literal(number: 1)
ReturnStmt:
BinaryOp(*)
```

```
Identifier(n)
CallExpr:
Callee:
Identifier(factorial)
Args:
BinaryOp(-)
Identifier(n)
Literal(number: 1)
VarDecl(let fact5)
CallExpr:
Callee:
Identifier(factorial)
Args:
Literal(number: 5)
VarDecl(let fact10)
CallExpr:
Callee:
Identifier(factorial)
Args:
CallExpr:
Callee:
Identifier(factorial)
Args:
Literal(number: 3)
VarDecl(let a)
Literal(number: 5)
VarDecl(let b)
Literal(number: 10)
VarDecl(let c)
Literal(number: 15)
VarDecl(let d)
Literal(number: 20)
VarDecl(let comparacion)
BinaryOp(&&)
BinaryOp(&&)
BinaryOp(<)
Identifier(a)
Identifier(b)
BinaryOp(<)
Identifier(b)
Identifier(c)
BinaryOp(||)
BinaryOp(<)
Identifier(c)
Identifier(d)
BinaryOp(==)
Identifier(a)
Identifier(b)
IfStmt:
Condition:
Identifier(verdadero)
Then:
Block:
VarDecl(let x)
Literal(number: 1)
WhileStmt:
Condition:
Identifier(falso)
Body:
Block:
VarDecl(let imposible)
Literal(number: 999)
VarDecl(let v1)
Literal(number: 10)
VarDecl(let v2)
Identifier(v1)
VarDecl(let v3)
Identifier(v2)
VarDecl(let v4)
Identifier(v3)
ExprStmt:
Assignment(v1)
Assignment(v2)
Assignment(v3)
Identifier(v4)
ExprStmt:
```

```
CallExpr:
Callee:
Identifier(print)
Args:
BinaryOp(+)
CallExpr:
Callee:
Identifier(fibonacci)
Args:
Literal(number: 5)
CallExpr:
Callee:
Identifier(factorial)
Args:
Literal(number: 4)
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
CallExpr:
Callee:
Identifier(sumar)
Args:
CallExpr:
Callee:
Identifier(multiplicar)
Args:
Literal(number: 2)
Literal(number: 3)
CallExpr:
Callee:
Identifier(restar)
Args:
Literal(number: 10)
Literal(number: 5)
VarDecl(let entrada)
CallExpr:
Callee:
Identifier(input)
VarDecl(let numeroEntrada)
CallExpr:
Callee:
Identifier(parseInt)
Args:
Identifier(entrada)
VarDecl(let variable)
Literal(number: 10)
VarDecl(let Variable)
Literal(number: 20)
VarDecl(let variabLe)
Literal(number: 30)
VarDecl(let variabl3)
Literal(number: 40)
FunctionDecl(programaPrincipal())
VarDecl(let resultado)
Literal(number: 0)
ExprStmt:
Assignment(resultado)
BinaryOp(+)
Identifier(resultado)
CallExpr:
Callee:
Identifier(fibonacci)
Args:
Literal(number: 10)
ExprStmt:
Assignment(resultado)
BinaryOp(+)
Identifier(resultado)
CallExpr:
Callee:
Identifier(nivel1)
ExprStmt:
Assignment(resultado)
BinaryOp(+)
```

```
Identifier(resultado)
CallExpr:
Callee:
Identifier(estructurasAnidadas)
Args:
Literal(number: 5)
ExprStmt:
Assignment(resultado)
BinaryOp(+)
Identifier(resultado)
CallExpr:
Callee:
Identifier(factorial)
Args:
Literal(number: 5)
IfStmt:
Condition:
BinaryOp(&&)
BinaryOp(>)
Identifier(resultado)
Literal(number: 100)
BinaryOp(<)
Identifier(resultado)
Literal(number: 1000)
Then:
Block:
ForStmt:
Init:
VarDecl(let i)
Literal(number: 0)
Condition:
BinaryOp(<)
Identifier(i)
Literal(number: 10)
Update:
Assignment(i)
BinaryOp(+)
Identifier(i)
Literal(number: 1)
Body:
Block:
ExprStmt:
Assignment(resultado)
BinaryOp(+)
Identifier(resultado)
CallExpr:
Callee:
Identifier(conMultiplesReturns)
Args:
Identifier(i)
IfStmt:
Condition:
CallExpr:
Callee:
Identifier(esPar)
Args:
Identifier(resultado)
Then:
Block:
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
Literal(string: El resultado es par)
Else:
Block:
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
Literal(string: El resultado es impar)
ReturnStmt:
Identifier(resultado)
VarDecl(let resultadoFinal)
```

```
CallExpr:
Callee:
Identifier(programaPrincipal)
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
Literal(string: Resultado final: )
ExprStmt:
CallExpr:
Callee:
Identifier(print)
Args:
Identifier(resultadoFinal)
```

3. Análisis Semántico

```
============================================================
ANÁLISIS SEMÁNTICO
============================================================
ERRORES (13):
------------------------------------------------------------
■ Línea 21, Col 16: Función 'esImpar' no declarada
■ Línea 37, Col 28: Variable 'x' no declarada
■ Línea 37, Col 39: Variable 'y' no declarada
■ Línea 37, Col 53: Variable 'z' no declarada
■ Línea 37, Col 66: Variable 'w' no declarada
■ Línea 37, Col 84: Variable 'a' no declarada
■ Línea 37, Col 96: Variable 'b' no declarada
■ Línea 57, Col 17: 'return' solo puede usarse dentro de una función
■ Línea 60, Col 13: 'return' solo puede usarse dentro de una función
■ Línea 63, Col 9: 'return' solo puede usarse dentro de una función
■ Línea 66, Col 5: 'return' solo puede usarse dentro de una función
■ Línea 166, Col 21: Operador '+' requiere operandos numéricos, se encontró string y string
■ Línea 166, Col 21: Operador '+' requiere operandos numéricos, se encontró error y string
ADVERTENCIAS (26):
------------------------------------------------------------
■■ Warning línea 216, col 17: Operador '!' requiere operando booleano
■■ Warning línea 34, col 1: variable 'expresionCompleja' declarada pero no usada
■■ Warning línea 37, col 1: variable 'condicionCompleja' declarada pero no usada
■■ Warning línea 70, col 1: variable 'x' declarada pero no usada
■■ Warning línea 95, col 1: variable 'resultado1' declarada pero no usada
■■ Warning línea 152, col 1: variable 'resultadoAnidado' declarada pero no usada
■■ Warning línea 163, col 1: variable 'numero' declarada pero no usada
■■ Warning línea 164, col 1: variable 'booleano' declarada pero no usada
■■ Warning línea 166, col 1: variable 'concatenacion' declarada pero no usada
■■ Warning línea 170, col 1: variable 'variableNoUsada1' declarada pero no usada
■■ Warning línea 171, col 1: variable 'variableNoUsada2' declarada pero no usada
■■ Warning línea 172, col 1: constant 'CONSTANTE_NO_USADA' declarada pero no usada
■■ Warning línea 187, col 1: variable 'logicaCompleja' declarada pero no usada
■■ Warning línea 214, col 1: variable 'dobleNegacion' declarada pero no usada
■■ Warning línea 215, col 1: variable 'negacionLogica' declarada pero no usada
■■ Warning línea 216, col 1: variable 'combinado' declarada pero no usada
■■ Warning línea 245, col 1: variable 'fact5' declarada pero no usada
■■ Warning línea 246, col 1: variable 'fact10' declarada pero no usada
■■ Warning línea 254, col 1: variable 'comparacion' declarada pero no usada
■■ Warning línea 279, col 1: variable 'numeroEntrada' declarada pero no usada
■■ Warning línea 282, col 1: variable 'variable' declarada pero no usada
■■ Warning línea 283, col 1: variable 'Variable' declarada pero no usada
■■ Warning línea 284, col 1: variable 'variabLe' declarada pero no usada
■■ Warning línea 285, col 1: variable 'variabl3' declarada pero no usada
■■ Warning línea 259, col 5: variable 'x' declarada pero no usada
■■ Warning línea 264, col 5: variable 'imposible' declarada pero no usada
TABLA DE SÍMBOLOS:
------------------------------------------------------------
global (level 0):
print [function:function] used:✓ init:✓
input [function:function] used:✓ init:✓
```

```
parseInt [function:function] used:✓ init:✓
parseFloat [function:function] used:✗ init:✓
fibonacci [function:function] used:✓ init:✓
esPar [function:function] used:✓ init:✓
esImpar [function:function] used:✓ init:✓
expresionCompleja [variable:number] used:✗ init:✓
condicionCompleja [variable:boolean] used:✗ init:✓
nivel1 [function:function] used:✓ init:✓
x [variable:number] used:✗ init:✓
testShadowing [function:function] used:✗ init:✓
operacionCompleja [function:function] used:✓ init:✓
resultado1 [variable:unknown] used:✗ init:✓
estructurasAnidadas [function:function] used:✓ init:✓
sumar [function:function] used:✓ init:✓
multiplicar [function:function] used:✓ init:✓
restar [function:function] used:✓ init:✓
resultadoAnidado [variable:unknown] used:✗ init:✓
mensaje1 [variable:string] used:✓ init:✓
mensaje2 [variable:string] used:✓ init:✓
numero [variable:number] used:✗ init:✓
booleano [variable:boolean] used:✗ init:✓
concatenacion [variable:error] used:✗ init:✓
variableNoUsada1 [variable:number] used:✗ init:✓
variableNoUsada2 [variable:string] used:✗ init:✓
CONSTANTE_NO_USADA [constant:number] used:✗ init:✓
funcionNoLlamada [function:function] used:✗ init:✓
verdadero [variable:boolean] used:✓ init:✓
falso [variable:boolean] used:✓ init:✓
logicaCompleja [variable:boolean] used:✗ init:✓
contador [variable:number] used:✓ init:✓
negativo [variable:number] used:✓ init:✓
dobleNegacion [variable:number] used:✗ init:✓
negacionLogica [variable:boolean] used:✗ init:✓
combinado [variable:boolean] used:✗ init:✓
conMultiplesReturns [function:function] used:✓ init:✓
factorial [function:function] used:✓ init:✓
fact5 [variable:unknown] used:✗ init:✓
fact10 [variable:unknown] used:✗ init:✓
a [variable:number] used:✓ init:✓
b [variable:number] used:✓ init:✓
c [variable:number] used:✓ init:✓
d [variable:number] used:✓ init:✓
comparacion [variable:boolean] used:✗ init:✓
v1 [variable:number] used:✓ init:✓
v2 [variable:number] used:✓ init:✓
v3 [variable:number] used:✓ init:✓
v4 [variable:number] used:✓ init:✓
entrada [variable:string] used:✓ init:✓
numeroEntrada [variable:number] used:✗ init:✓
variable [variable:number] used:✗ init:✓
Variable [variable:number] used:✗ init:✓
variabLe [variable:number] used:✗ init:✓
variabl3 [variable:number] used:✗ init:✓
programaPrincipal [function:function] used:✓ init:✓
resultadoFinal [variable:unknown] used:✓ init:✓
function_fibonacci (level 1):
n [parameter:unknown] used:✓ init:✓
if_then (level 2):
block (level 3):
if_else (level 2):
block (level 3):
function_esPar (level 1):
n [parameter:unknown] used:✓ init:✓
if_then (level 2):
block (level 3):
if_else (level 2):
block (level 3):
function_esImpar (level 1):
n [parameter:unknown] used:✓ init:✓
if_then (level 2):
block (level 3):
if_else (level 2):
block (level 3):
function_nivel1 (level 1):
a [variable:number] used:✓ init:✓
nivel2 [function:function] used:✓ init:✓
```

```
function_nivel2 (level 2):
b [variable:number] used:✓ init:✓
nivel3 [function:function] used:✓ init:✓
function_nivel3 (level 3):
c [variable:number] used:✓ init:✓
nivel4 [function:function] used:✓ init:✓
function_nivel4 (level 4):
d [variable:number] used:✓ init:✓
nivel5 [function:function] used:✓ init:✓
function_nivel5 (level 5):
e [variable:number] used:✓ init:✓
function_testShadowing (level 1):
x [variable:number] used:✓ init:✓
if_then (level 2):
block (level 3):
x [variable:number] used:✓ init:✓
while (level 4):
block (level 5):
x [variable:number] used:✓ init:✓
function_operacionCompleja (level 1):
a [parameter:unknown] used:✓ init:✓
b [parameter:unknown] used:✓ init:✓
c [parameter:unknown] used:✓ init:✓
d [parameter:unknown] used:✓ init:✓
e [parameter:unknown] used:✓ init:✓
function_estructurasAnidadas (level 1):
n [parameter:unknown] used:✓ init:✓
resultado [variable:number] used:✓ init:✓
for (level 2):
i [variable:number] used:✓ init:✓
block (level 3):
if_then (level 4):
block (level 5):
while (level 6):
block (level 7):
if_then (level 8):
block (level 9):
for (level 10):
j [variable:number] used:✓ init:✓
block (level 11):
if_then (level 12):
block (level 13):
if_else (level 12):
block (level 13):
if_else (level 8):
block (level 9):
if_then (level 8):
block (level 9):
if_else (level 4):
block (level 5):
if_then (level 6):
block (level 7):
while (level 8):
block (level 9):
function_sumar (level 1):
a [parameter:unknown] used:✓ init:✓
b [parameter:unknown] used:✓ init:✓
function_multiplicar (level 1):
a [parameter:unknown] used:✓ init:✓
b [parameter:unknown] used:✓ init:✓
function_restar (level 1):
a [parameter:unknown] used:✓ init:✓
b [parameter:unknown] used:✓ init:✓
function_funcionNoLlamada (level 1):
while (level 1):
block (level 2):
if_then (level 3):
block (level 4):
if_else (level 3):
block (level 4):
for (level 1):
idx [variable:number] used:✓ init:✓
block (level 2):
temp [variable:number] used:✓ init:✓
if_then (level 3):
block (level 4):
```

```
for (level 5):
inner [variable:number] used:✓ init:✓
block (level 6):
squared [variable:number] used:✓ init:✓
function_conMultiplesReturns (level 1):
n [parameter:unknown] used:✓ init:✓
if_then (level 2):
block (level 3):
if_then (level 2):
block (level 3):
for (level 2):
i [variable:number] used:✓ init:✓
block (level 3):
if_then (level 4):
block (level 5):
function_factorial (level 1):
n [parameter:unknown] used:✓ init:✓
if_then (level 2):
block (level 3):
if_then (level 1):
block (level 2):
x [variable:number] used:✗ init:✓
while (level 1):
block (level 2):
imposible [variable:number] used:✗ init:✓
function_programaPrincipal (level 1):
resultado [variable:number] used:✓ init:✓
if_then (level 2):
block (level 3):
for (level 4):
i [variable:number] used:✓ init:✓
block (level 5):
if_then (level 2):
block (level 3):
if_else (level 2):
block (level 3):
```