

Seminario de Solución de problemas de Traductores de Lenguajes II

Centro Universitario de Ciencias Exactas en
ingenierías

Universidad de Guadalajara



Maestro: **LUIS FELIPE MUNOZ MENDOZA**

Juan Antonio Pérez Juárez
Diego Andrés Hernández Rodríguez
Juan Fernando Prieto Gómez

Práctica 3 - Analizador Semántico

Introducción:

Objetivo: Desarrollar un analizador semántico que valide la corrección de las estructuras sintácticas procesadas en la fase anterior, asegurando la correcta gestión de tipos de datos, ámbito de variables y evaluación de expresiones aritméticas.

Desarrollo:

Primero debemos definir lo que es un Analizador semántico.

El análisis semántico, expresado así, es el proceso de extraer el significado de un texto. El análisis gramatical y el reconocimiento de vínculos entre palabras específicas en un contexto determinado permiten a los ordenadores comprender e interpretar frases, párrafos o incluso manuscritos enteros.

Es un componente crucial del Procesamiento del Lenguaje Natural (PLN) y la inspiración de aplicaciones como los chatbots, los motores de búsqueda y el análisis de textos mediante aprendizaje automático.

Las herramientas basadas en el análisis semántico pueden ayudar a las empresas a extraer automáticamente información útil de datos no estructurados, como correos electrónicos, solicitudes de asistencia y comentarios de los consumidores. A continuación repasamos su funcionamiento.

El análisis semántico, también denominado relación semántica, es el uso de ontologías (en el sentido informático de este término, no en el filosófico) para analizar el contenido de textos almacenados en un soporte informático, como Internet. Este conjunto de procedimientos informáticos combina minería de textos y tecnologías de Web Semántica como Marco de Descripción de Recursos (RDF por sus siglas en inglés). El análisis semántico mide la relación de diferentes conceptos ontológicos.

Varios grupos de investigación académica tienen proyectos activos en esta área. Uno de ellos es el Centro Kno.e.sis de la Universidad Estatal de Wright.

Es la fase del compilador en la que se conectan definiciones de variables con sus usos, verificar que cada expresión tenga un tipo correcto y traducir la sintaxis abstracta en una representación más simple y adecuada para la generación de código de máquina. Dicho de otra manera, la tarea del análisis semántico es la de determinar propiedades y verificar las condiciones que son relevantes para la buena formación de los programas de acuerdo con las reglas del lenguaje de

programación, pero va más allá de lo que pueden describir las gramáticas libres de contexto.

Para explicar el proceso que sucede en la fase de análisis semántico es importante explicar ciertos subtemas como los tipos de datos, semántica estática y dinámica, los errores en tiempo ejecución, lenguajes seguros y no seguros así como los tipados y no tipados.

Código:

```
Python
import ply.lex as lex
import ply.yacc as yacc
import sys

# Tabla de símbolos para almacenar variables y sus tipos
tabla_simbolos = {}

# Definición del analizador léxico
tokens = (
    'NUMERO',
    'ID',
    'SUMA',
    'RESTA',
    'MULT',
    'DIV',
    'LPAREN',
    'RPAREN',
    'IGUAL',
    'PUNTOCOMA',
    'INT',
    'FLOAT'
)

# Reglas para tokens simples
t_SUMA = r'\+'
t_RESTA = r'\-'
t_MULT = r'\*'
t_DIV = r'\/'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_IGUAL = r'\='
t_PUNTOCOMA = r'\;'

# Palabras reservadas
def t_INT(t):
    r'int'
    return t
```

```

def t_FLOAT(t):
    r'float'
    return t

# Regla para identificadores
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    return t

# Regla para números (enteros y flotantes)
def t_NUMERO(t):
    r'\d+(\.\d+)?'
    if '.' in t.value:
        t.value = float(t.value)
    else:
        t.value = int(t.value)
    return t

# Ignorar espacios y tabulaciones
t_ignore = ' \t'

# Nueva línea
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

# Error handling
def t_error(t):
    print(f"Carácter ilegal '{t.value[0]}' en la línea {t.lexer.lineno}")
    t.lexer.skip(1)

# Construcción del lexer
lexer = lex.lex()

# Definición de la gramática
def p_programa(p):
    '''programa : declaracion
                | asignacion
                | expresion'''
    p[0] = p[1]

def p_declaracion(p):
    '''declaracion : INT ID PUNTOCOMA
                  | FLOAT ID PUNTOCOMA'''
    if p[1] == 'int':
        tipo = 'int'
    else:

```

```

        tipo = 'float'

        # Verificar si la variable ya está declarada
        if p[2] in tabla_simbolos:
            print(f"Error semántico: Variable '{p[2]}' ya declarada
previamente.")
        else:
            tabla_simbolos[p[2]] = {'tipo': tipo, 'valor': None}
            print(f"Variable '{p[2]}' declarada como {tipo}")

        p[0] = None

def p_asignacion(p):
    '''asignacion : ID IGUAL expresion PUNTOCOMA'''
    # Verificar si la variable está declarada
    if p[1] not in tabla_simbolos:
        print(f"Error semántico: Variable '{p[1]}' no declarada.")
        p[0] = None
        return

    # Asignar el valor y verificar compatibilidad de tipos
    valor = p[3]
    tipo_var = tabla_simbolos[p[1]]['tipo']

    if tipo_var == 'int' and isinstance(valor, float):
        print(f"Advertencia: Asignando valor flotante a variable entera
'{p[1]}'. Se truncará el valor.")
        valor = int(valor)

    tabla_simbolos[p[1]]['valor'] = valor
    print(f"Asignación: {p[1]} = {valor}")
    p[0] = valor

def p_expression_binaria(p):
    '''expression : expresion SUMA expresion
                    | expresion RESTA expresion
                    | expresion MULT expresion
                    | expresion DIV expresion'''
    if p[2] == '+':
        p[0] = p[1] + p[3]
    elif p[2] == '-':
        p[0] = p[1] - p[3]
    elif p[2] == '*':
        p[0] = p[1] * p[3]
    elif p[2] == '/':
        # Verificar división por cero
        if p[3] == 0:
            print("Error: División por cero")

```

```

        p[0] = None
    else:
        p[0] = p[1] / p[3]

def p_expression_parenthesis(p):
    'expression : LPAREN expression RPAREN'
    p[0] = p[2]

def p_expression_numero(p):
    'expression : NUMERO'
    p[0] = p[1]

def p_expression_id(p):
    'expression : ID'
    # Verificar si la variable existe en la tabla de símbolos
    if p[1] not in tabla_simbolos:
        print(f"Error semántico: Variable '{p[1]}' no declarada.")
        p[0] = 0 # Valor por defecto para continuar la evaluación
        return

    # Verificar si la variable tiene valor asignado
    if tabla_simbolos[p[1]]['valor'] is None:
        print(f"Error semántico: Variable '{p[1]}' no inicializada.")
        p[0] = 0 # Valor por defecto para continuar la evaluación
    else:
        p[0] = tabla_simbolos[p[1]]['valor']

def p_error(p):
    if p:
        print(f"Error de sintaxis en '{p.value}', línea {p.lineno}")
    else:
        print("Error de sintaxis al final de la entrada")

# Construir el parser
parser = yacc.yacc()

def mostrar_tabla_simbolos():
    print("\n=== TABLA DE SÍMBOLOS ===")
    print("Variable\tTipo\tValor")
    print("-" * 30)
    for var, info in tabla_simbolos.items():
        print(f"{var}\t\t\t{info['tipo']}\t\t{info['valor']}")
    print("=" * 30)

def evaluar_ast(expresion):
    """Evalúa una expresión y muestra los resultados"""
    try:
        result = parser.parse(expresion)

```

```

        if result is not None:
            print(f"Resultado de evaluación: {result}")
            print("\nValidación semántica completada exitosamente.")
            mostrar_tabla_simbolos()
            return result
    except Exception as e:
        print(f"Error durante el análisis: {e}")
        return None

# Función principal para probar el analizador
def main():
    print("Analizador Semántico - Evaluador de Expresiones")
    print("Ingrese 'salir' para terminar")

    while True:
        try:
            expresion = input(">> ")
            if expresion.lower() == 'salir':
                break
            evaluar_ast(expresion)
        except EOFError:
            break

if __name__ == "__main__":
    main()

```

Lógica:

Arquitectura General

El analizador se compone de tres componentes principales interconectados:

1. **Analizador Léxico**: Convierte el texto de entrada en tokens
2. **Analizador Sintáctico**: Organiza los tokens en estructuras gramaticales
3. **Analizador Semántico**: Verifica la coherencia y evalúa las expresiones

Flujo de Procesamiento

Unset

Texto de entrada → Analizador Léxico → Tokens → Analizador Sintáctico → Árbol Sintáctico → Analizador Semántico → Resultado/Errores

Análisis Detallado del Código

1. Inicialización y Estructuras de Datos

Python

```
import ply.lex as lex
import ply.yacc as yacc
import sys
```

Tabla de símbolos para almacenar variables y sus tipos

Unset

```
tabla_simbolos = {}
```

- **Importaciones**: Se importan los módulos necesarios de PLY para análisis léxico y sintáctico.
- **Tabla de símbolos**: Diccionario que almacena las variables declaradas con su tipo y valor. Estructura clave para el análisis semántico.

2. Analizador Léxico (Lexer)

Python

```
# Definición del analizador léxico
```

```
tokens = (
    'NUMERO',
    'ID',
    'SUMA',
    'RESTA',
    'MULT',
    'DIV',
    'LPAREN',
    'RPAREN',
    'IGUAL',
    'PUNTOCOMA',
    'INT',
    'FLOAT'
)
```

- **Tokens**: Define todos los elementos atómicos del lenguaje que el analizador léxico debe identificar.

Python

```
# Reglas para tokens simples
```



```

t_SUMA = r'\+'
t_RESTA = r'\-'
t_MULT = r'\*'
t_DIV = r'\/'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_IGUAL = r'\='
t_PUNTOCOMA = r';'

```

- ****Definición de tokens simples****

Cada token se define mediante una expresión regular. Los tokens como operadores y símbolos utilizan este formato simple.

```

Python
# Palabras reservadas
def t_INT(t):
    r'int'
    return t

def t_FLOAT(t):
    r'float'
    return t

```

- ****Palabras reservadas****: Se definen como funciones para facilitar su procesamiento específico si es necesario.

```

Python
# Regla para identificadores
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    return t

```

- ****Identificadores****Reconoce nombres de variables que comienzan con una letra o guión bajo, seguidos por letras, números o guiones bajos.

Python

Regla para números (enteros y flotantes)

```
def t_NUMERO(t):
    r'\d+(\.\d+)?'
    if '.' in t.value:
        t.value = float(t.value)
    else:
        t.value = int(t.value)
    return t
```

- ****Números****: Reconoce enteros y flotantes. Además de identificar el token, realiza la conversión al tipo de dato correspondiente.

Python

Ignorar espacios y tabulaciones

t_ignore = ' \t'

Nueva línea

```
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)
```

Error handling

```
def t_error(t):
    print(f"Carácter ilegal '{t.value[0]}' en la línea {t.lexer.lineno}")
    t.lexer.skip(1)
```

Construcción del lexer

lexer = lex.lex()

- ****Caracteres ignorados****: Espacios y tabulaciones no generan tokens.
- ****Control de líneas****: Incrementa el contador de líneas para reportar errores con la ubicación correcta.
- ****Manejo de errores léxicos****: Identifica caracteres no reconocidos en el flujo de entrada.
- ****Iniciación del lexer****: Crea el analizador léxico con las reglas definidas.

3. Analizador Sintáctico (Parser)

Python

Definición de la gramática

```
def p_programa(p):
    '''programa : declaracion
```

```

        | asignacion
        | expresion'''
p[0] = p[1]

```

- ****Punto de entrada****: Define la estructura de alto nivel del programa como declaraciones, asignaciones o expresiones.
- ****Asignación $p[0] = p[1]$ ****: Transfiere el valor semántico de la producción derecha a la izquierda.

```

Python
def p_declaracion(p):
    '''declaracion : INT ID PUNTOCOMA
                   | FLOAT ID PUNTOCOMA'''
    if p[1] == 'int':
        tipo = 'int'
    else:
        tipo = 'float'

    # Verificar si la variable ya está declarada
    if p[2] in tabla_simbolos:
        print(f"Error semántico: Variable '{p[2]}' ya declarada
previamente.")
    else:
        tabla_simbolos[p[2]] = {'tipo': tipo, 'valor': None}
        print(f"Variable '{p[2]}' declarada como {tipo}")

    p[0] = None

```

- ****Declaración de variables****: Reconoce patrones como "int x;" o "float y;".
- ****Verificación semántica****: Detecta redeclaraciones de variables (error semántico).
- ****Actualización de tabla de símbolos****: Registra la variable con su tipo (sin valor inicial).

```

Python
def p_asignacion(p):
    '''asignacion : ID IGUAL expresion PUNTOCOMA'''
    # Verificar si la variable está declarada
    if p[1] not in tabla_simbolos:
        print(f"Error semántico: Variable '{p[1]}' no declarada.")
        p[0] = None
        return

    # Asignar el valor y verificar compatibilidad de tipos

```

```

valor = p[3]
tipo_var = tabla_simbolos[p[1]]['tipo']

if tipo_var == 'int' and isinstance(valor, float):
    print(f"Advertencia: Asignando valor flotante a variable entera
    '{p[1]}'. Se truncará el valor.")
    valor = int(valor)

tabla_simbolos[p[1]]['valor'] = valor
print(f"Asignación: {p[1]} = {valor}")
p[0] = valor

```

- ****Asignación de valores****: Reconoce patrones como "x = 5;".
- ****Verificación de existencia****: Comprueba que la variable haya sido declarada.
- ****Comprobación de tipos****: Verifica la compatibilidad entre el tipo declarado y el valor asignado.
- ****Conversión implícita****: Trunca valores flotantes asignados a variables enteras.

Python

```

def p_expresion_binaria(p):
    '''expresion : expresion SUMA expresion
                | expresion RESTA expresion
                | expresion MULT expresion
                | expresion DIV expresion'''
    if p[2] == '+':
        p[0] = p[1] + p[3]
    elif p[2] == '-':
        p[0] = p[1] - p[3]
    elif p[2] == '*':
        p[0] = p[1] * p[3]
    elif p[2] == '/':
        # Verificar división por cero
        if p[3] == 0:
            print("Error: División por cero")
            p[0] = None
        else:
            p[0] = p[1] / p[3]

```

- ****Operaciones binarias****: Define las cuatro operaciones aritméticas básicas.
- ****Evaluación de expresiones****: Calcula el resultado de cada operación.
- ****Detección de división por cero****: Verifica este caso especial y emite un error.

Python

```
def p_expression_parenthesis(p):  
    'expression : LPAREN expression RPAREN'  
    p[0] = p[2]
```

- ****Paréntesis****: Permite agrupar expresiones para alterar la precedencia predeterminada.
- ****Preservación del valor****: El valor de la expresión entre paréntesis se preserva.

Python

```
def p_expression_numero(p):  
    'expression : NUMERO'  
    p[0] = p[1]
```

- ****Números como expresiones****: Los literales numéricos son expresiones válidas.
- ****Transferencia de valor****: El valor ya convertido por el lexer se transfiere a la expresión.

Python

```
def p_expression_id(p):  
    'expression : ID'  
    # Verificar si la variable existe en la tabla de símbolos  
    if p[1] not in tabla_simbolos:  
        print(f"Error semántico: Variable '{p[1]}' no declarada.")  
        p[0] = 0 # Valor por defecto para continuar la evaluación  
        return  
  
    # Verificar si la variable tiene valor asignado  
    if tabla_simbolos[p[1]]['valor'] is None:  
        print(f"Error semántico: Variable '{p[1]}' no inicializada.")  
        p[0] = 0 # Valor por defecto para continuar la evaluación  
    else:  
        p[0] = tabla_simbolos[p[1]]['valor']
```

- ****Variables como expresiones****: Los identificadores son expresiones válidas.
- ****Verificación de existencia****: Comprueba que la variable esté declarada.
- ****Verificación de inicialización****: Comprueba que la variable tenga un valor asignado.
- ****Manejo de errores con recuperación****: Proporciona un valor por defecto para continuar la evaluación.

```

Python
def p_error(p):
    if p:
        print(f"Error de sintaxis en '{p.value}', línea {p.lineno}")
    else:
        print("Error de sintaxis al final de la entrada")

# Construir el parser
parser = yacc.yacc()

```

- ****Manejo de errores sintácticos****: Reporta errores de sintaxis con información contextual.
- ****Inicialización del parser****: Crea el analizador sintáctico con las reglas definidas.

4. Funciones Auxiliares y Principal

```

Python
def mostrar_tabla_simbolos():
    print("\n=== TABLA DE SÍMBOLOS ===")
    print("Variable\tTipo\tValor")
    print("-" * 30)
    for var, info in tabla_simbolos.items():
        print(f"{var}\t\t\t{info['tipo']}\t\t{info['valor']}")
    print("=" * 30)

```

- ****Visualización de la tabla de símbolos****: Muestra el estado actual de las variables.
- ****Formato tabular****: Facilita la lectura de la información.

```

Python
def evaluar_ast(expresion):
    """Evalúa una expresión y muestra los resultados"""
    try:
        result = parser.parse(expresion)
        if result is not None:
            print(f"Resultado de evaluación: {result}")
            print("\nValidación semántica completada exitosamente.")
            mostrar_tabla_simbolos()
            return result
    except Exception as e:
        print(f"Error durante el análisis: {e}")
        return None

```

- ****Evaluación de expresiones****: Punto de entrada para evaluar una expresión.

- ****Manejo de excepciones****: Captura errores durante el análisis.
- ****Visualización de resultados****: Muestra el resultado y la tabla de símbolos.

```
Python
def main():
    print("Analizador Semántico - Evaluador de Expresiones")
    print("Ingrese 'salir' para terminar")

    while True:
        try:
            expresion = input(">> ")
            if expresion.lower() == 'salir':
                break
            evaluar_ast(expresion)
        except EOFError:
            break

if __name__ == "__main__":
    main()
```

- ****Interfaz interactiva****: Permite al usuario introducir expresiones una a una.
- ****Bucle principal****: Procesa entradas hasta que el usuario escribe 'salir'.
- ****Manejo de entrada/salida****: Captura errores de entrada.

Flujo de Ejecución para un Ejemplo

Para ilustrar el flujo completo, veamos paso a paso cómo se procesa: ``x = 5 + y``

1. ****Análisis Léxico****:

- Se identifican los tokens: ``ID(x)``, ``IGUAL``, ``NUMERO(5)``, ``SUMA``, ``ID(y)``, ``PUNTOCOMA``

2. ****Análisis Sintáctico****:

- Se reconoce el patrón como una asignación: ``ID IGUAL expresion PUNTOCOMA``
- La expresión ``5 + y`` se identifica como: ``expresion SUMA expresion``
- El primer operando ``5`` se identifica como ``NUMERO``
- El segundo operando ``y`` se identifica como ``ID``

3. ****Análisis Semántico****:

- Se verifica que ``x`` esté declarada en la tabla de símbolos
- Se verifica que ``y`` esté declarada y tenga un valor

- Se evalúa ``5 + y`` obteniendo el resultado
- Se asigna el resultado a ``x`` actualizando la tabla de símbolos
- Se verifica la compatibilidad de tipos entre el resultado y el tipo de ``x``

4. ****Resultado****:

- Se muestra el valor asignado a ``x``
- Se actualiza y muestra la tabla de símbolos

Detección de Errores Semánticos

El analizador detecta los siguientes errores semánticos:

1. ****Variable no declarada****:

Python

```
if p[1] not in tabla_simbolos:
    print(f"Error semántico: Variable '{p[1]}' no
declarada.")
```

2. ****Variable ya declarada****:

Python

```
if p[2] in tabla_simbolos:
    print(f"Error semántico: Variable '{p[2]}' ya declarada
previamente.")
```

3. ****Variable no inicializada****:

Python

```
if tabla_simbolos[p[1]]['valor'] is None:
    print(f"Error semántico: Variable '{p[1]}' no inicializada.")
```

4. ****División por cero****:

Python

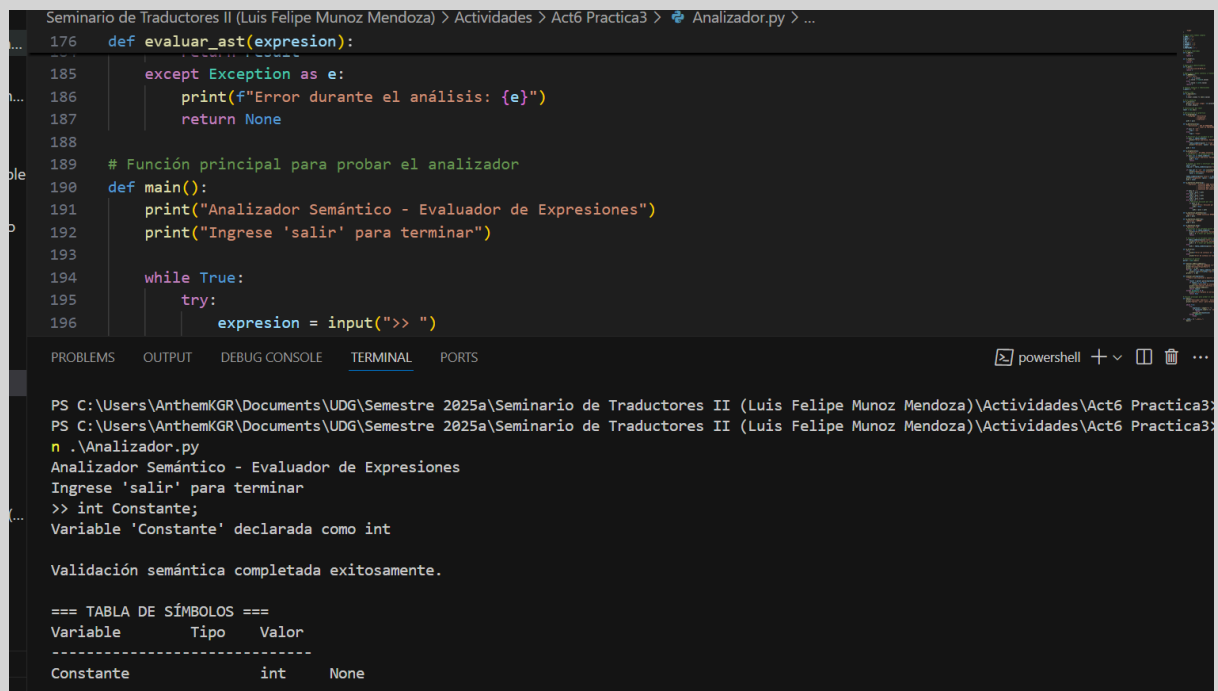
```
if p[3] == 0:
    print("Error: División por cero")
```


5. **Incompatibilidad de tipos**:

Python

```
if tipo_var == 'int' and isinstance(valor, float):  
    print(f"Advertencia: Asignando valor flotante a variable entera  
'{p[1]}'". Se truncará el valor.")
```

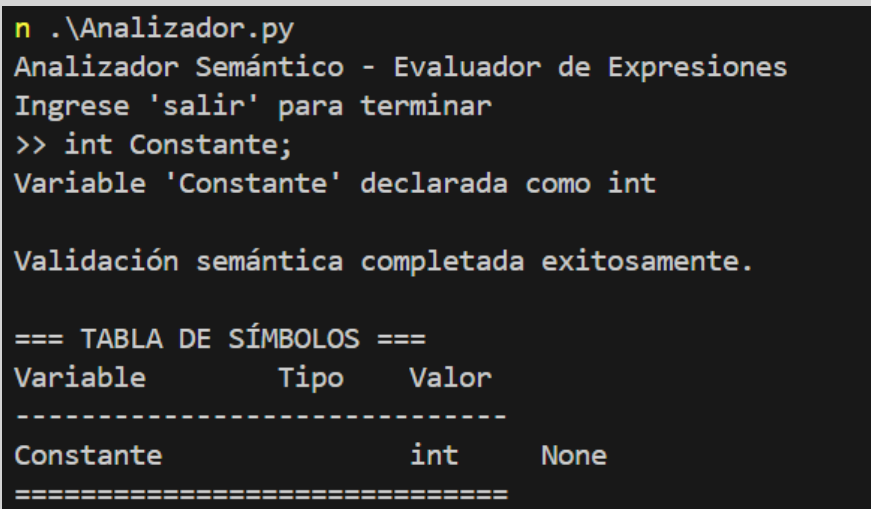
Capturas de pantalla:



The screenshot shows a code editor with a Python script named `Analizador.py`. The code defines a function `evaluar_ast` to handle exceptions and a `main` function to run the analyzer. The terminal output shows the program's execution, including a warning about variable type assignment and a final symbol table.

```
176 def evaluar_ast(expresion):  
177     except Exception as e:  
185         print(f"Error durante el análisis: {e}")  
186         return None  
187  
188  
189 # Función principal para probar el analizador  
190 def main():  
191     print("Analizador Semántico - Evaluador de Expresiones")  
192     print("Ingrese 'salir' para terminar")  
193  
194     while True:  
195         try:  
196             expresion = input(">> ")  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

```
PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act6 Practica3> n .\Analizador.py  
Analizador Semántico - Evaluador de Expresiones  
Ingrese 'salir' para terminar  
>> int Constante;  
Variable 'Constante' declarada como int  
  
Validación semántica completada exitosamente.  
  
=== TABLA DE SÍMBOLOS ===  
Variable      Tipo      Valor  
-----  
Constante      int      None  
=====
```



```
n .\Analizador.py  
Analizador Semántico - Evaluador de Expresiones  
Ingrese 'salir' para terminar  
>> int Constante;  
Variable 'Constante' declarada como int  
  
Validación semántica completada exitosamente.  
  
=== TABLA DE SÍMBOLOS ===  
Variable      Tipo      Valor  
-----  
Constante      int      None  
=====
```

```

=====
>> int qwerty;
Variable 'qwerty' declarada como int

Validación semántica completada exitosamente.

=== TABLA DE SÍMBOLOS ===
Variable      Tipo      Valor
-----
Constante          int      10
decimal            float    3.1416
z                  int      None
qwerty             int      None
=====

```

Escenario de división por cero:

	PROBLEMS	OUTPUT	DEBUG CONSOLE	<u>TERMINAL</u>	POR
				<pre> ----- cero float None ===== >> cero=10/0; Error: División por cero Asignación: cero = None Validación semántica completada exitosamente. === TABLA DE SÍMBOLOS === Variable Tipo Valor ----- cero float None ===== >> </pre>	

Escenario con espacios:

```

-----
cero          float   None
espacio       int     None
=====
>> espacio = ;
Error de sintaxis en ';', línea 1

Validación semántica completada exitosamente.

=== TABLA DE SÍMBOLOS ===
Variable      Tipo     Valor
-----
cero          float   None
espacio       int     None
=====

```

Conclusión:

Para esta práctica hemos tenido la suerte de encontrar varios repositorios de github que hacían los distintos procesos que se pedía en la asignación, por lo que fue relativamente sencillo juntar las partes.

La mayor parte de los problemas que tuvimos a la hora del desarrollo de este script, fue el entender cómo funcionan las librerías que hacen la mayor parte del trabajo del reconocimiento de valores y el parceo de los tokens.

Así que para el problema de la tabla de signos utilizamos IA para poder completarla y solucionar los problemas que tuvimos del desarrollo.

Pero es un script que funciona de manera satisfactoria, por lo que considero que esta práctica es satisfactoria.

Referencias:

Ortega, C. (2024, May 16). Análisis Semántico: Qué es, cómo funciona y ejemplos. QuestionPro. <https://www.questionpro.com/blog/es/analisis-semantico/>

Rivera, G. L. (2022, November 16). Análisis semántico - Gian Luca Rivera - Medium. Medium. <https://medium.com/@LucaBia/an%C3%A1lisis-sem%C3%A1ntico-i-iii-762cc0d63fb0>

jp-loran/Analizador-Lexico-Sintactico-Semantico: Analizador que reconoce componentes léxicos . (n.d.). GitHub. <https://github.com/jp-loran/Analizador-Lexico-Sintactico-Semantico>

Piloalucard/TraductorLenguajeMaquina: Analizador léxico, sintáctico y semántico, con generación de código y compilador incluido. Realizado en Python. (n.d.). GitHub. <https://github.com/Piloalucard/TraductorLenguajeMaquina>