6-5-2024

Universidad de Guadalajara
CUCEI

Etapa: Analizador Semántico

Materia: Seminario de solución de problemas de traductores de lenguaje II

Nombre: Nudelstejer Gómez Iván

Código: 218130122

Profesor: Michel Emanuel López Franco

Código del analizador semántico en Python:

Reporte con captura de pantalla de la validación semántica de los siguientes códigos

Ejemplo 1:

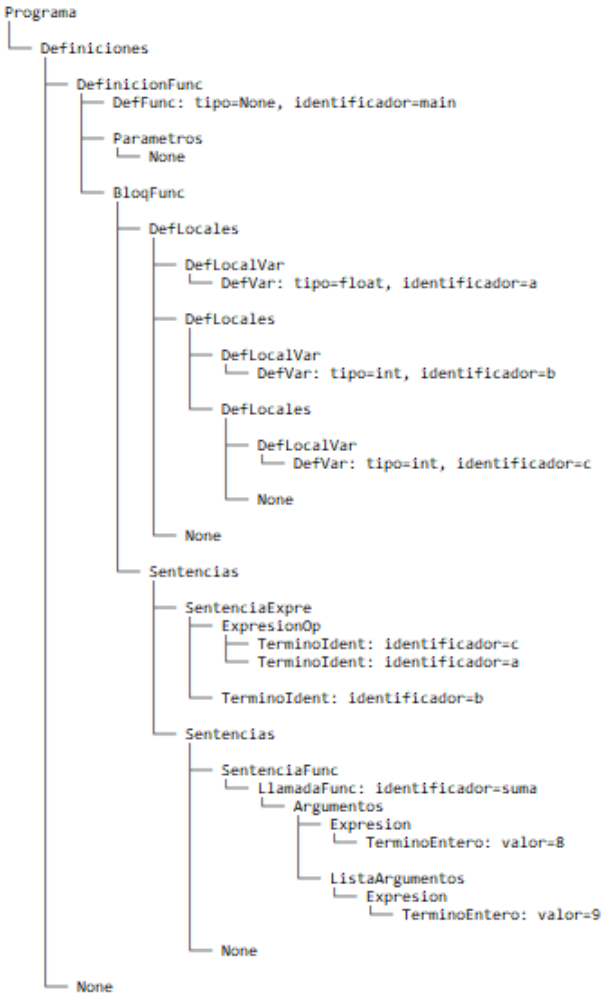
```
class Programa:
    def __init__(self, definiciones=None):
        self.definiciones = definiciones


class Definiciones:
    def __init__(self, definicion=None, definiciones=None):
        self.definicion = definicion
        self.definiciones = definiciones


class DefinicionFunc:
    def __init__(self, identificador, tipo_retorno, parametros, bloque):
        self.identificador = identificador
        self.tipo_retorno = tipo_retorno
        self.parametros = parametros
        self.bloque = bloque


class DefinicionVar:
    def __init__(self, tipo, identificador):
        self.tipo = tipo
        self.identificador = identificador
```

```
class Bloque:
    def __init__(self, sentencias=None):
        self.sentencias = sentencias

class SentenciaAsignacion:
    def __init__(self, variable, expresion):
        self.variable = variable
        self.expresion = expresion

class LlamadaFuncion:
    def __init__(self, identificador, argumentos=None):
        self.identificador = identificador
        self.argumentos = argumentos

class Variable:
    def __init__(self, identificador):
        self.identificador = identificador

class ExpresionBinaria:
    def __init__(self, operador, operando_izquierdo,
operando_derecho):
        self.operador = operador
        self.operando_izquierdo = operando_izquierdo
        self.operando_derecho = operando_derecho
```

```
# Creación del árbol AST para el código de ejemplo
definicion_a = DefinicionVar("float", "a")
definicion_b = DefinicionVar("int", "b")
definicion_c = DefinicionVar("int", "c")

asignacion_1 = SentenciaAsignacion(Variable("c"),
ExpresionBinaria("+", Variable("a"), Variable("b")))
llamada_suma = LlamadaFuncion("suma", [Constante(8),
Constante(9)])
asignacion_2 = SentenciaAsignacion(Variable("c"), llamada_suma)

bloque_principal = Bloque([asignacion_1, asignacion_2])
definicion_main = DefinicionFunc("main", "int", None,
bloque_principal)

programa = Programa(Definiciones(definicion_main))
```

Ejemplo 1.

```
int main(){
float a;
int b;
int c;
c = a+b;
c = suma(8,9);
}
```




Programa
Definiciones
DefinicionVar
DefVar: tipo=float, identificador=a
Definiciones
DefinicionVar
DefVar: tipo=int, identificador=b
Definiciones
DefinicionVar
DefVar: tipo=int, identificador=c
DefinicionFunc
Identificador=main
Parametros
None
BloqFunc
DefLocales
Sentencias
SentenciaAsignacion
Identificador=c
ExpresionBinaria: operador=+, izquierda=a, derecha=b
SentenciaFunc
LlamadaFunc: identificador=suma
Argumentos
ExpresionEntera: valor=8
ExpresionEntera: valor=9
None
None

Ejemplo 2:

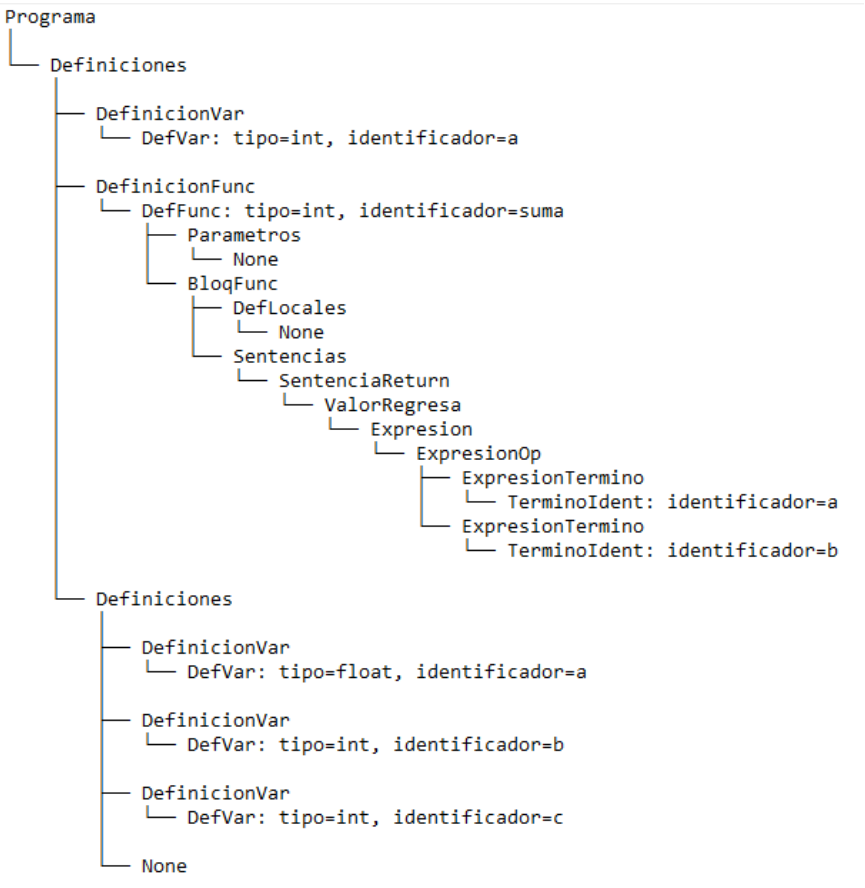
```
class Programa:
    def __init__(self, definiciones=None, main=None):
        self.definiciones = definiciones
        self.main = main

class Definiciones:
    def __init__(self, definicion=None, definiciones=None):
        self.definicion = definicion
        self.definiciones = definiciones

class DefinicionVar:
    def __init__(self, tipo, identificador):
        self.tipo = tipo
        self.identificador = identificador

class DefinicionFunc:
    def __init__(self, tipo_retorno, identificador, parametros=None, bloque=None):
        self.tipo_retorno = tipo_retorno
        self.identificador = identificador
        self.parametros = parametros
        self.bloque = bloque

class Parametro:
    def __init__(self, tipo, identificador):
        self.tipo = tipo
        self.identificador = identificador
```

```
class Bloque:
    def __init__(self, sentencias=None):
        self.sentencias = sentencias

class SentenciaReturn:
    def __init__(self, expresion=None):
        self.expresion = expresion

class SentenciaAsignacion:
    def __init__(self, variable, expresion):
        self.variable = variable
        self.expresion = expresion

class LlamadaFuncion:
    def __init__(self, identificador, argumentos=None):
        self.identificador = identificador
        self.argumentos = argumentos

class Variable:
    def __init__(self, identificador):
        self.identificador = identificador

class ExpresionBinaria:
    def __init__(self, operador, operando_izquierdo, operando_derecho):
        self.operador = operador
        self.operando_izquierdo = operando_izquierdo
```

```
        self.operando_derecho = operando_derecho


# Creación del árbol AST para el código de ejemplo
definicion_a = DefinicionVar("int", "a")


parametro_a = Parametro("int", "a")
parametro_b = Parametro("int", "b")
definicion_suma = DefinicionFunc("int", "suma", [parametro_a, parametro_b],
Bloque([SentenciaReturn(ExpresionBinaria("+", Variable("a"), Variable("b")))]))


definicion_main = DefinicionFunc("int", "main", None, Bloque([
    DefinicionVar("float", "a"),
    DefinicionVar("int", "b"),
    DefinicionVar("int", "c"),
    SentenciaAsignacion(Variable("c"), ExpresionBinaria("+", Variable("a"), Variable("b"))),
    SentenciaAsignacion(Variable("c"), LlamadaFuncion("suma", [ExpresionBinaria("+",
Variable("8.5"), Variable("9.9"))]))
]))


programa = Programa(Definiciones(definicion_a, Definiciones(definicion_suma)),
definicion_main)
```

Ejemplo 2:

```
int a;
int suma(int a, int b){
return a+b;
}

int main(){
float a;
int b;
int c;
c = a+b;
c = suma(8.5,9.9);
}
```



- Hay tres definiciones de variables al comienzo del programa: una para `a` de tipo `int`, y dos para `a` y `b` de tipo `float` e `int` respectivamente dentro de `main()`.

- Hay una definición de función llamada `suma`, que toma dos parámetros `a` y `b` de tipo `int` y devuelve un valor de tipo `int`.

- Dentro de la definición de `suma`, hay una sentencia `return` que devuelve la suma de `a` y `b`.

- Dentro de `main()`, hay algunas asignaciones y una llamada a la función `suma()`.

El árbol refleja correctamente la estructura jerárquica del programa, separando las definiciones de variables, la definición de función y las sentencias dentro de la función `suma` y `main`.