UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENERIAS

Seminario de traductores 2

Reporte de actividad final

Nombre del alumno: Sunem Sandoval Gil

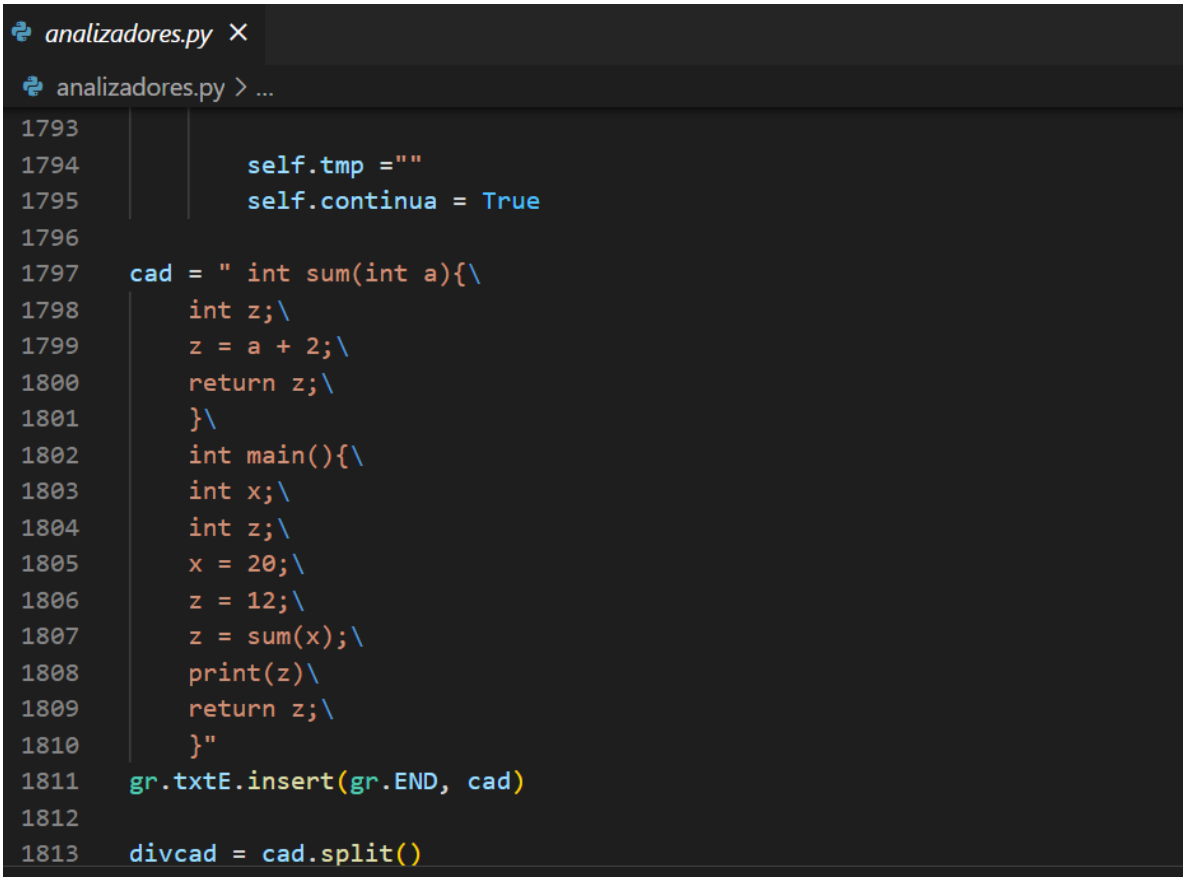
Profesor: Michel Emanuel López Franco

Título de la actividad: Etapa generación de código

Fecha: 28 Noviembre 2023

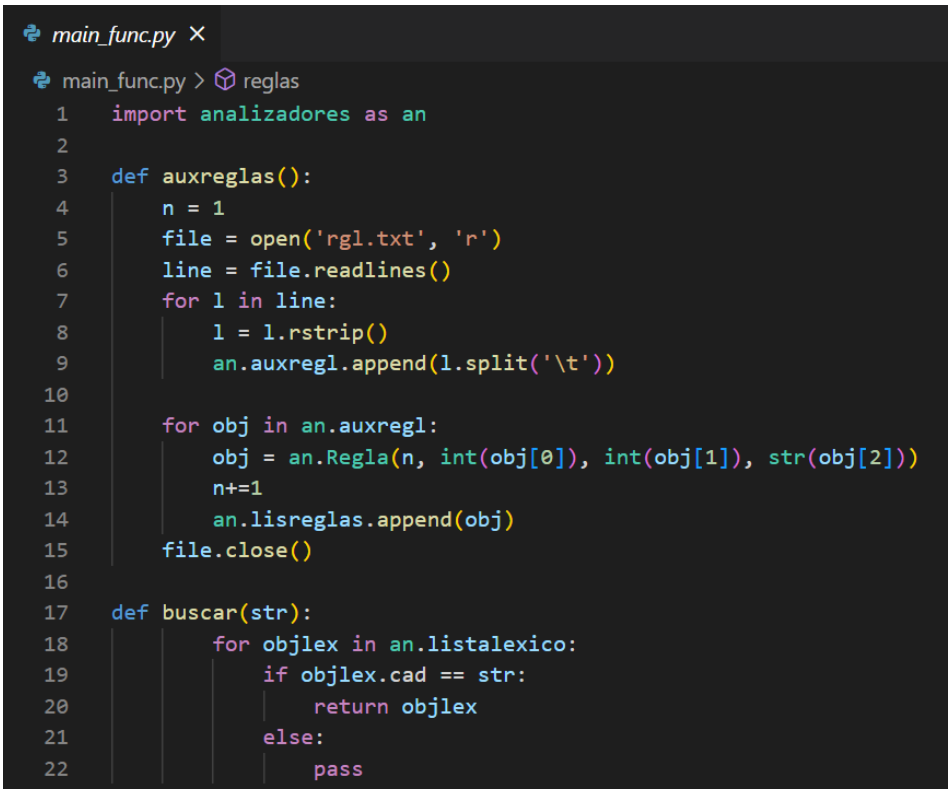
Etapa generación de código

Analizadores.py



Función sum: Se define una función llamada sum que toma un argumento a de tipo entero. Dentro de la función, se declara una variable z de tipo entero, se le asigna el valor de a + 2, y luego se devuelve z.

Main.py



La función auxreglas lee un archivo llamado 'rgl.txt' y procesa sus líneas, cada línea se limpia de espacios al final con rstrip(). Los resultados se almacenan en la lista an.auxregl. Luego, cada elemento de an.auxregl se utiliza para crear un objeto de tipo an.Regla y se añade a la lista an.lisreglas. Se utiliza una variable n para asignar un número único a cada regla.

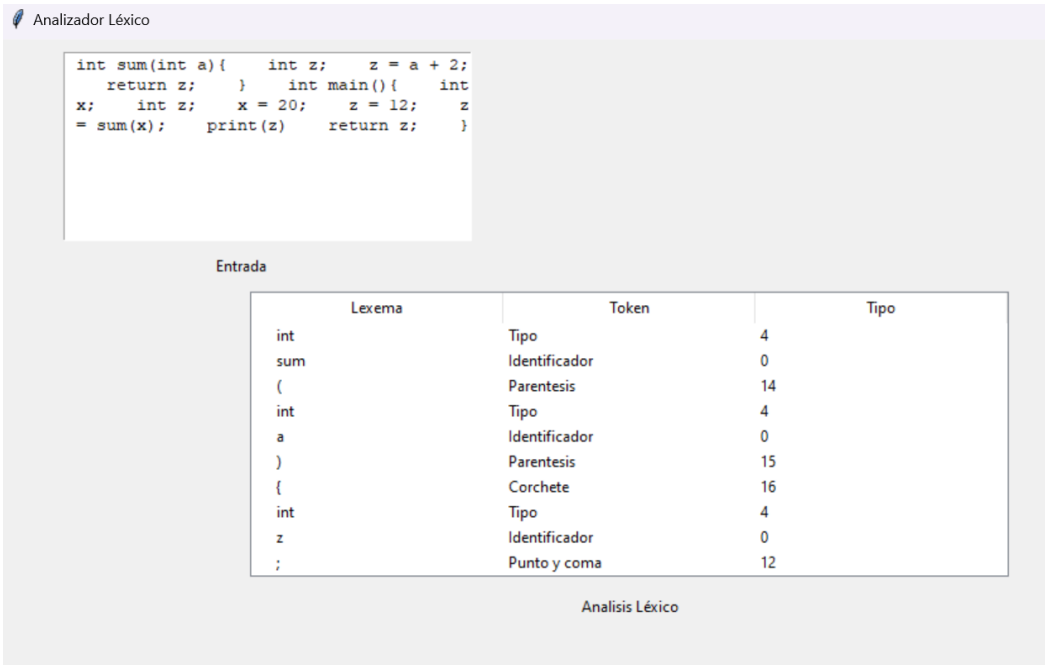
La función buscar busca un objeto en la lista an.listalexico cuyo atributo cad sea igual al parámetro de entrada str. Si encuentra un objeto que cumple la condición, lo devuelve. Si no, no hace nada.

```
24  def reglas():
25      file = open('compilador.lr', 'r')
26      line = file.readlines()
27      for l in line:
28          l = l.rstrip()
29          an.matrizreglas.append(l.split('\t'))
30
31      for i in range (len(an.matrizreglas)):
32          for j in range(len(an.matrizreglas[i])):
33              an.matrizreglas[i][j] = int(an.matrizreglas[i][j])
34      file.close()
35
36  def eliminalistaVar(self):
37          an.an.pila.pop()
38          an.pila.pop()
39          an.pila.pop()
40          self.data = an.pila.pop()
41          an.pila.pop()
42          an.pila.pop()
43          an.listavar.append(an.DefVar('Unknown ', self.data, self.lv))
```

La función reglas realiza una operación similar a auxreglas pero con un archivo llamado 'compilador.lr' y os resultados se almacenan en an.matrizreglas. Luego, se convierten los elementos de la matriz a enteros.

El método manipula pilas (an.an.pila y an.pila) y realiza operaciones de pop en ellas. Luego, crea un objeto an.DefVar y lo agrega a la lista an.listavar.

Compilar



Analizador Semántico

Tipo	Identificador	Ámbito
int	z	sum
int	x	main
int	z	main

Analisis Semántico

Analizador Sintáctico

```
$0
$0int5
$0int5sum8
$0int5sum8(11
$0int5sum8(11int15
$0int5sum8(11int15a18
$0int5sum8(11int15a18ListaParam21
$0int5sum8(11Parametros14
$0int5sum8(11Parametros14)17
$0int5sum8(11Parametros14)17{20
$0int5sum8(11Parametros14)17{20int5
$0int5sum8(11Parametros14)17{20int5z8
$0int5sum8(11Parametros14)17{20int5z8ListaVar9
$0int5sum8(11Parametros14)17{20int5z8ListaVar9;12
$0int5sum8(11Parametros14)17{20DefVar25
$0int5sum8(11Parametros14)17{20DefLocal24
$0int5sum8(11Parametros14)17{20DefLocal24z27
$0int5sum8(11Parametros14)17{20DefLocal24z27=35
$0int5sum8(11Parametros14)17{20DefLocal24z27=35a46
$0int5sum8(11Parametros14)17{20DefLocal24z27=35Termino44
```

```
r30: ValorR
d57
r24: Senten
r18: DefLoc
r15: DefLoc
r16: DefLoc
r16: DefLoc
r16: DefLoc
r16: DefLoc
r16: DefLoc
r16: DefLoc
d33
r14: BloqFu
r9: DefFunc
r5: Definic
r2: Definic
r3: Definic
r3: Definic
r1: program
r0 acept
```

Analisis Sintáctico

Salida

Árbol ---->

```
└── BloqueFunc
    └── DefLocales
        ├── DefLocal
        │   └── Sentencia
        │       └── Expresion
        │           └── Termino Id: z Contexto: Retorno main
        ├── DefLocal
        │   └── Sentencia
        │       ├── Expresion
        │       │   └── Termino Id: x Contexto: main
        │       └── Expresion
        │           └── Termino
        │               └── LlamadaFunc ID: sum
        │                   └── Argumentos
        ├── DefLocal
        │   └── Sentencia
        │       └── Expresion
```

Ensamblador.asm

Código de ensamblador generado

ensamblador.asm

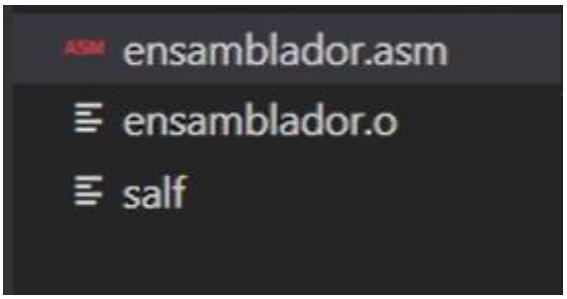
```
section .data
    primr: db  "La impresion es := %lf",10,0
section .bss
    resp: resq 2
section .text

extern printf
global sum
, main
sum:
    PUSH rbp
    MOV rbp, rsp
    SUB rsp, 48
    MOV QWORD [rbp -24], rdi
    MOV rax, QWORD [rbp -24]
    MOV rdi, 2
    ADD rax, rdi

    ADD rsp, 48
    MOV rsp, rbp
    POP rbp
    ret
```

```
24  main:
25      PUSH rbp
26      MOV rbp, rsp
27      SUB rsp, 48
28          MOV WORD [rbp -4] , 20
29      MOV WORD [rbp -8] , 12
30      MOV rax, QWORD [rbp -4]
31      MOV rdi, rax
32
33      call sum
34      MOV QWORD [rbp -8], rax
35
36      PUSH qword[rbp -8]
37      FILD dword[rsp]
38      FSTP qword[rel resp]
39      ADD rsp, 8
40      MOVSD xmm0,qword[rel resp]
41      MOV rdi, primr
42      MOV al, 1
43      call printf WRT ..plt
44      MOV rax, QWORD [rbp -8]
```

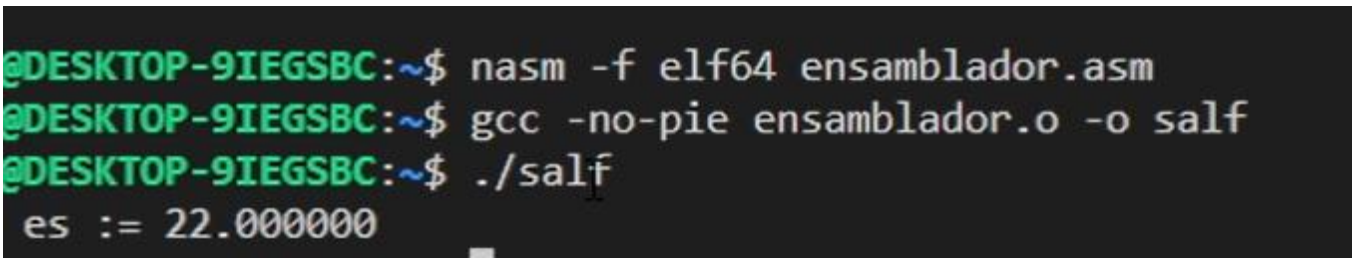
```
46      ADD rsp, 48
47      MOV rsp, rbp
48      MOV rax, 60
49      MOV rdi, 0
50      syscall
51
```

Procedimiento

Hay que copiar el Código generado en el ensamblador y pegarlo en otra ventana con la distribución de Ubuntu para poder utilizar el ensamblador.



- **nas -f elf64 ensamblador.asm:** Nos va a crear el archivo ensamblador.o
- **gcc -no-pie ensamblador.o -o salf:** El segundo lo compila
- **./salf:** Y por último nos metemos a la carpeta en la que esta nuestro Código



Al ser la impresión 22 podemos notar que coincide, entendiendo lo anterior explicado sobre la función suma.