

Laboratorio 2: ANTLR

1. Crear programa que asigne un valor a una variable:

```
(prog (stat a = (expr 5) \r\n) (stat b = (expr 6) \r\n) (stat (expr (expr a) + (expr b)) \r\n) (stat (expr (expr a) * (expr ( (expr (expr b) + (expr 2)) ))) \r\n) (stat c = (exp  
r (expr a) - (expr b)) \r\n) (stat (expr (expr c) / (expr 2)) \r\n)) \r\n)  
Int: 5  
Assigned a = 5
```

2. Cree un programa que realice una operación aritmética simple:

```
Id: a = 5  
Id: b = 6  
AddSub: 5 + 6 = 11  
11
```

3. Experimente con expresiones más complejas

```
(prog (stat (expr (expr (exp  
Int: 4  
Int: 5  
MulDiv: 4 * 5 = 20  
visitParens called  
Int: 10  
Int: 2  
MulDiv: 10 / 2 = 5.0  
AddSub: 20 + 5.0 = 25.0  
25.0
```

4. Modifique el lenguaje para incluir la asignación de variables con expresiones aritméticas:

```
(prog (stat a = (expr (expr 2) + (ex  
Int: 2  
Int: 2  
AddSub: 2 + 2 = 4  
Assigned a = 4  
Int: 5  
Assigned b = 5  
Id: a = 4  
Id: b = 5  
AddSub: 4 + 5 = 9  
Assigned c = 9
```

5. Agregue manejo de errores al compilador para detectar tokens inválidos en el programa fuente:

```
a = 2 + 2
b = 5
c = a + d
```

```
File "/program/MiniLangParser.py", line 405, in accept
    return visitor.visitId(self)
           ^^^^^^^^^^^^^^^^^^^^^^
File "/program/Driver.py", line 61, in visitId
    raise NameError(f"Warning: Variable {id} was not declared") # in case the id was not declared
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
NameError: Warning: Variable d was not declared
```

6. Cree un programa que utilice paréntesis para cambiar la precedencia de operadores:

$$(2 + 3) * 4$$

$$2 + (3 * 4)$$

```
visitParens called
Int: 2
Int: 3
AddSub: 2 + 3 = 5
Int: 4
MulDiv: 5 * 4 = 20
20
Int: 2
visitParens called
Int: 3
Int: 4
MulDiv: 3 * 4 = 12
AddSub: 2 + 12 = 14
14
```

7. Extienda el lenguaje para soportar comentarios de una sola línea:

```
appuser@8d9f77418a71:/program$ python3 Driver.py program_test.txt
line 1:17 missing NEWLINE at '<EOF>'
Int: 2
Int: 3
AddSub: 2 + 3 = 5
5
```

```
1 // 2 + 3 // result 5
```

8. Agregue operadores de comparación (`==`, `!=`, `<`, `>`, `<=`, `>=`) al lenguaje:

```
appuser@8d9f77418a71:/program$ python3 Driver.py program_test.txt
line 2:6 missing NEWLINE at '<EOF>'
Int: 2
Int: 2
Comparison: 2 == 2 = True
True
Int: 3
Int: 4
Comparison: 3 != 4 = True
True
```

```
> program > ≡ prog
  2 == 2
  3 != 4
```

9. Cree un programa que utilice una estructura "if":
10. Cree un programa que utilice una estructura "while":

```
line 3:3 no viable alternative at input 'ifa'
line 3:10 extraneous input 'then' expecting NEWLINE
line 7:6 no viable alternative at input 'whilea'
line 7:13 extraneous input 'do' expecting NEWLINE
line 9:3 no viable alternative at input 'end'
Int: 0
Assigned a = 0
Int: 0
Assigned b = 0
Id: a = 0
Id: b = 0
Comparison: 0 == 0 = True
True
Int: 1
Assigned b = 1
Traceback (most recent call last):
  File "/program/Driver.py", line 134, in <module>
```

If y While

11. Agregue soporte para funciones definidas por el usuario:
12. Cree un programa que defina y llame a una función:
13. Implemente un sistema de tipos básico que, además de incluir enteros, también incluya cadenas.