

# LABORATORIO GUIADO 2

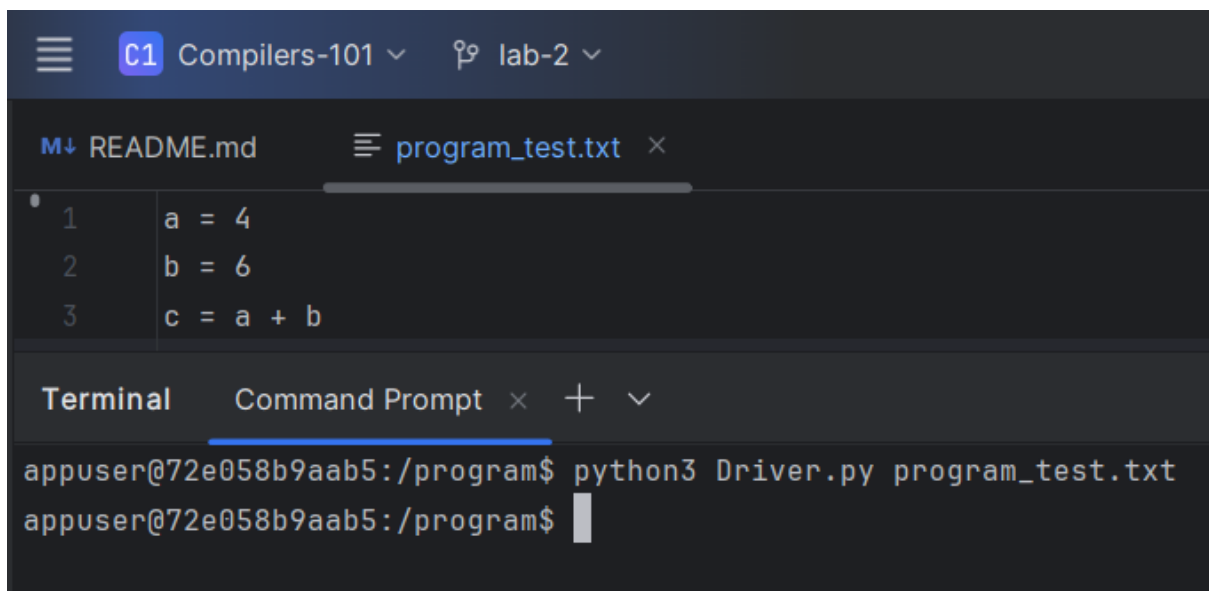
FRANCISCO CASTILLO 21562

## REPOSITORIO DE GITHUB

Para ver el repositorio visite: <https://github.com/FranzCastillo/Compilers-101/tree/lab-2>

## ACTIVIDADES CON ANTLR

1. CREE UN PROGRAMA QUE ASIGNE UN VALOR A UNA VARIABLE.

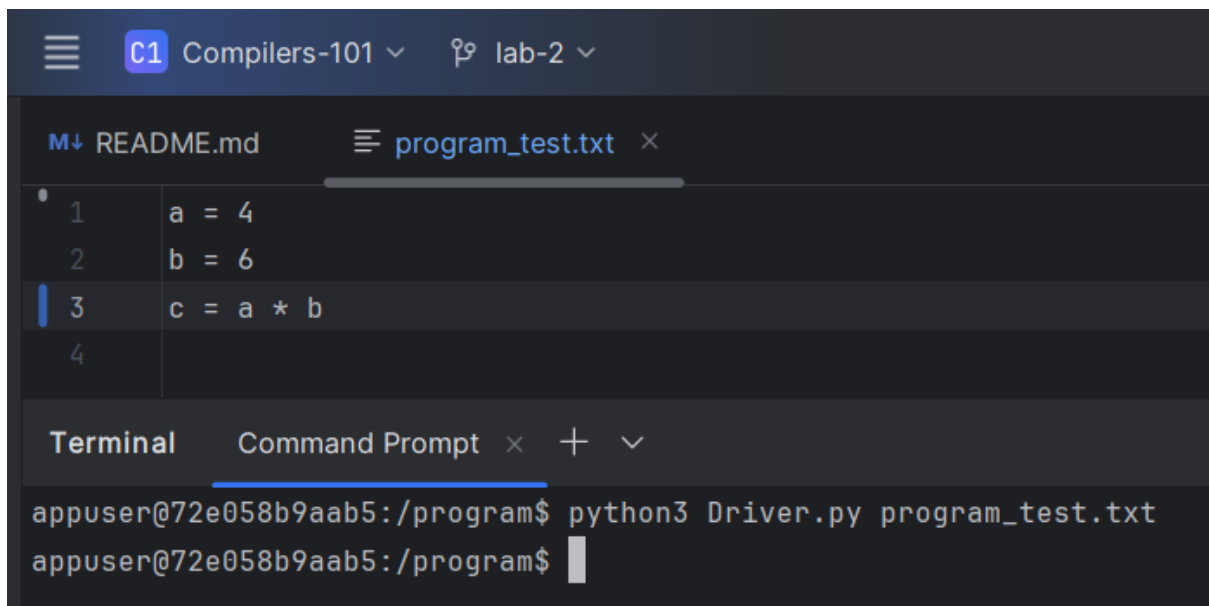


The screenshot shows a code editor with a file named `program_test.txt` open. The code contains three lines of Python code: `a = 4`, `b = 6`, and `c = a + b`. Below the editor, a terminal window is open, showing the command `python3 Driver.py program_test.txt` being executed. The terminal output shows the command being run and the prompt returning.

```
1 a = 4
2 b = 6
3 c = a + b
```

```
Terminal Command Prompt x + v
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
appuser@72e058b9aab5:/program$
```

2. CREE UN PROGRAMA QUE REALICE UNA OPERACIÓN ARITMÉTICA SIMPLE.



The screenshot shows a code editor with a file named `program_test.txt` open. The code contains four lines of Python code: `a = 4`, `b = 6`, `c = a * b`, and an empty line. Below the editor, a terminal window is open, showing the command `python3 Driver.py program_test.txt` being executed. The terminal output shows the command being run and the prompt returning.

```
1 a = 4
2 b = 6
3 c = a * b
4
```

```
Terminal Command Prompt x + v
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
appuser@72e058b9aab5:/program$
```

### 3. EXPERIMENTE CON EXPRESIONES MÁS COMPLEJAS.

The screenshot shows the VS Code editor with the file `program_test.txt` open. The code contains a single line: `temp = 2 * ((10 + 20) / (17 - 2))`. The terminal at the bottom shows the command `python3 Driver.py program_test.txt` being executed successfully.

```
1 temp = 2 * ((10 + 20) / (17 - 2))
2
```

```
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
appuser@72e058b9aab5:/program$
```

### 4. MODIFIQUE EL LENGUAJE PARA INCLUIR LA ASIGNACIÓN DE VARIABLES CON EXPRESIONES ARITMÉTICAS.

The screenshot shows the VS Code editor with the file `program_test.txt` open. The code contains a single line: `temp = 1 + 2 * 3 - 4 / 5`. The terminal at the bottom shows the command `python3 Driver.py program_test.txt` being executed successfully.

```
1 temp = 1 + 2 * 3 - 4 / 5
2
```

```
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
```

### 5. AGREGUE MANEJO DE ERRORES AL COMPILADOR PARA DETECTAR TOKENS INVÁLIDOS EN EL PROGRAMA FUENTE.

The screenshot shows the VS Code editor with the file `MiniLang.g4` open. The code contains the following grammar rules:

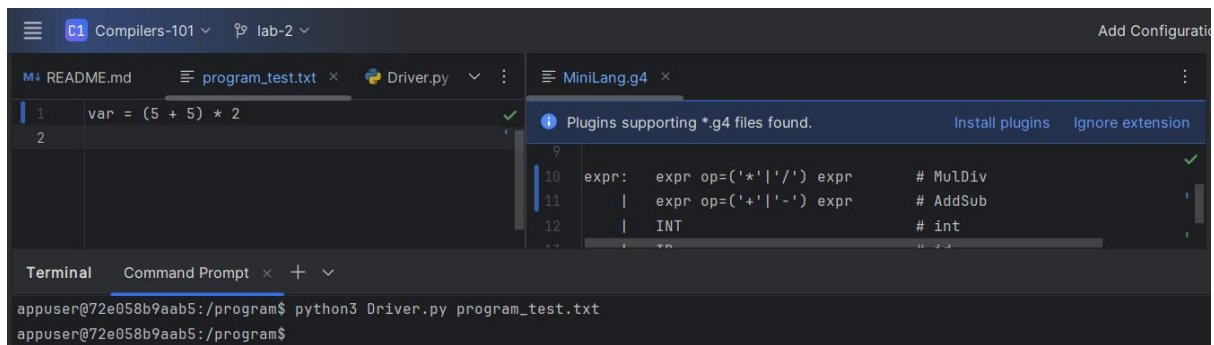
```
21 ID : [a-zA-Z]+ ; // match identifiers
22 INT : [0-9]+ ; // match integers
23 NEWLINE: '\r'? '\n' ; // return newlines to parser (is end-statement sign
24 WS : [ \t]+ -> skip ; // toss out whitespace
25
26 ERROR : . { print("No defined token for: '{0}'".format(self.text)) } ;
```

The terminal at the bottom shows the command `python3 Driver.py program_test.txt` being executed, resulting in the error message: `No defined token for: '>'`.

```
1 temp = 1 + 2 * 3 - 4 / 5
2
3 >
4
```

```
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
No defined token for: '>'
appuser@72e058b9aab5:/program$
```

## 6. CREE UN PROGRAMA QUE UTILICE PARÉNTESIS PARA CAMBIAR LA PRECEDENCIA DE OPERADORES.

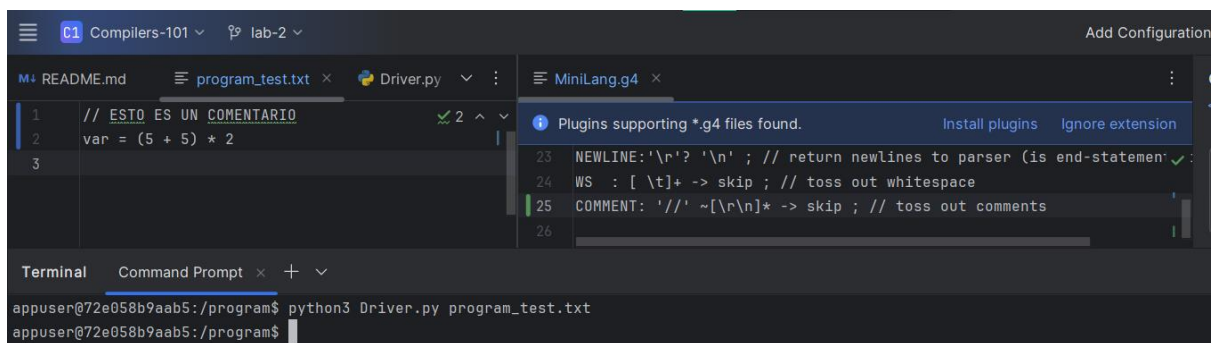


The screenshot shows the VS Code editor with the file `MiniLang.g4` open. The grammar rules for `expr` are defined as follows:

```
9
10 expr: expr op=('*' | '/') expr      # MulDiv
11      | expr op=('+' | '-') expr      # AddSub
12      | INT                          # int
```

The terminal shows the command `python3 Driver.py program_test.txt` being executed successfully.

## 7. EXTIENDA EL LENGUAJE PARA SOPORTAR COMENTARIOS DE UNA SOLA LÍNEA.

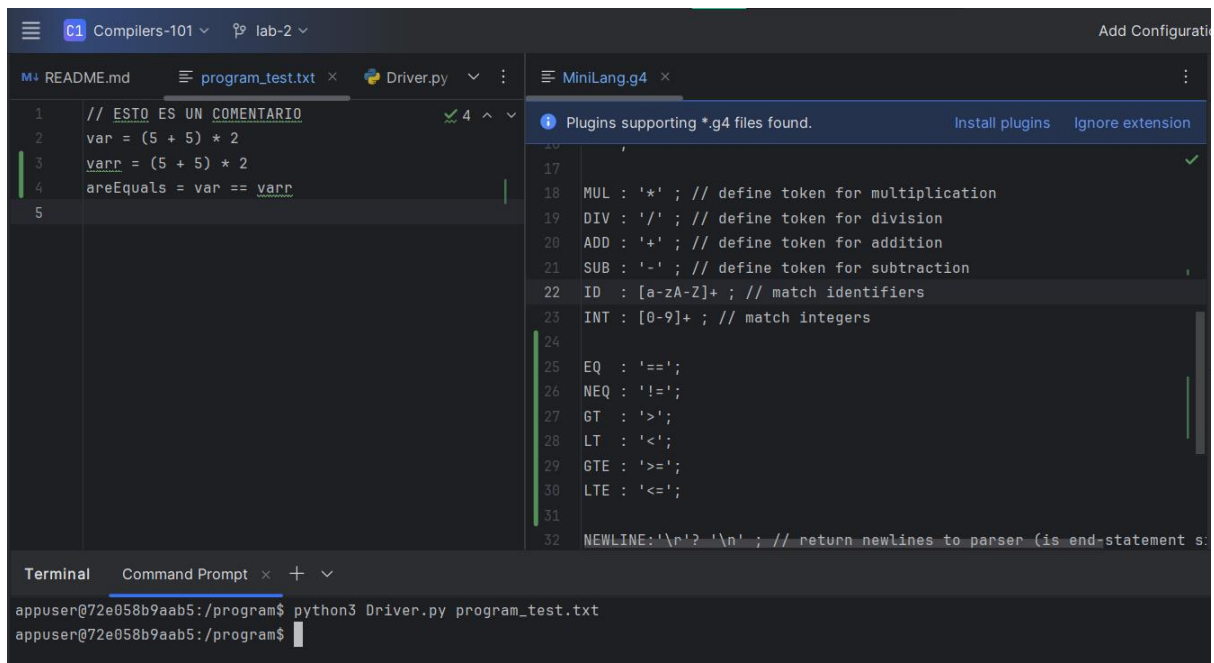


The screenshot shows the VS Code editor with the file `MiniLang.g4` open. The grammar rules are extended to support single-line comments:

```
23 NEWLINE: '\r'? '\n' ; // return newlines to parser (is end-statement)
24 WS : [ \t ]+ -> skip ; // toss out whitespace
25 COMMENT: '//' ~[\r\n]* -> skip ; // toss out comments
```

The terminal shows the command `python3 Driver.py program_test.txt` being executed successfully.

## 8. AGREGUE OPERADORES DE COMPARACIÓN (==, !=, <, >, <=, >=) AL LENGUAJE.

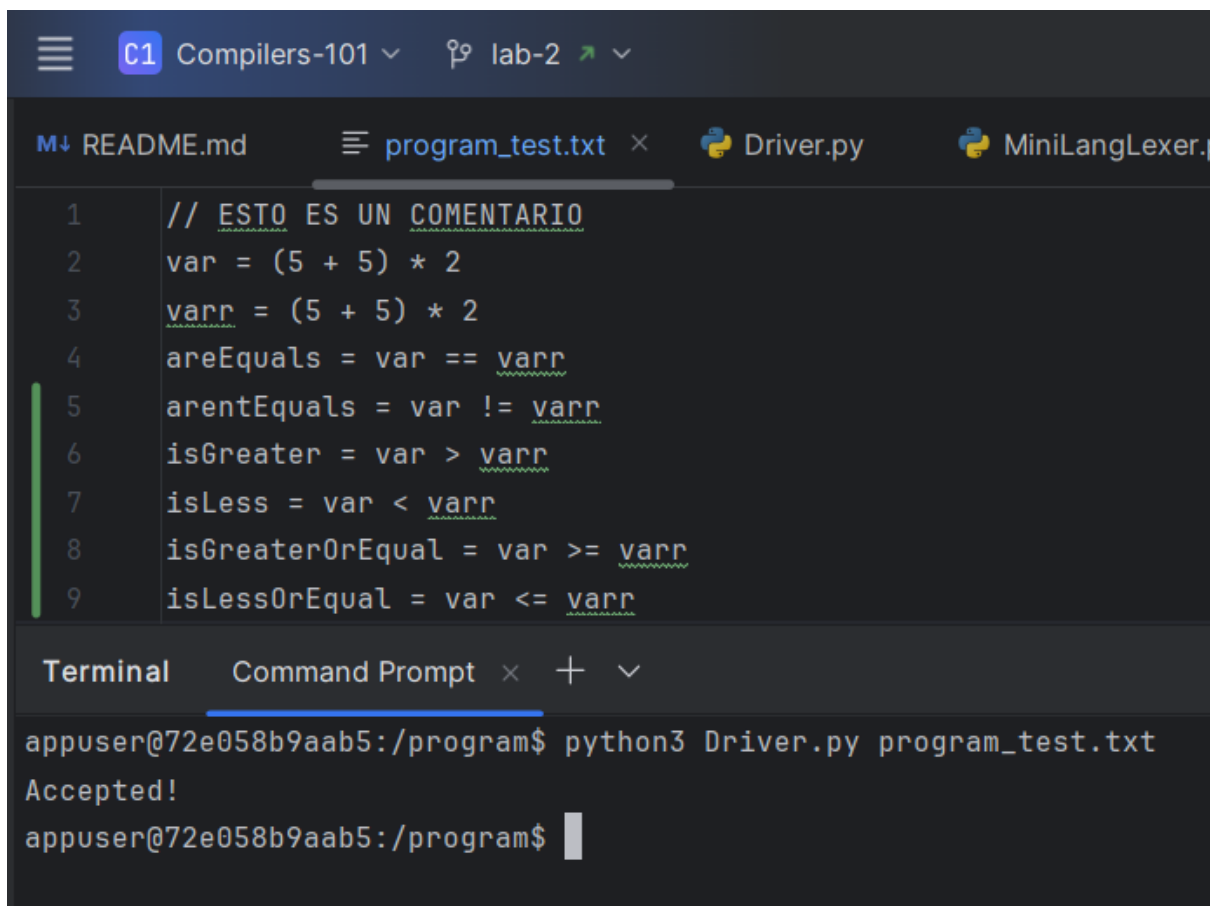


The screenshot shows the VS Code editor with the file `MiniLang.g4` open. The grammar rules are extended to support comparison operators:

```
17
18 MUL : '*' ; // define token for multiplication
19 DIV : '/' ; // define token for division
20 ADD : '+' ; // define token for addition
21 SUB : '-' ; // define token for subtraction
22 ID : [a-zA-Z]+ ; // match identifiers
23 INT : [0-9]+ ; // match integers
24
25 EQ : '==';
26 NEQ : '!=';
27 GT : '>';
28 LT : '<';
29 GTE : '>=';
30 LTE : '<=';
31
32 NEWLINE: '\r'? '\n' ; // return newlines to parser (is end-statement s.
```

The terminal shows the command `python3 Driver.py program_test.txt` being executed successfully.

9. CREE UN PROGRAMA QUE UTILICE OPERADORES DE COMPARACIÓN.

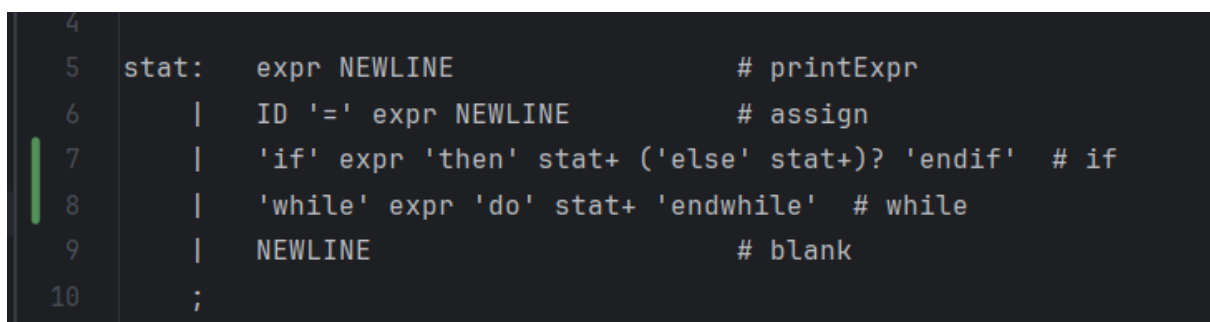


The screenshot shows a code editor with a dark theme. The top bar indicates the project is 'Compilers-101' and the file is 'lab-2'. The editor has several tabs: 'README.md', 'program\_test.txt' (active), 'Driver.py', and 'MiniLangLexer.py'. The code in 'program\_test.txt' is as follows:

```
1 // ESTO ES UN COMENTARIO
2 var = (5 + 5) * 2
3 varr = (5 + 5) * 2
4 areEquals = var == varr
5 arentEquals = var != varr
6 isGreater = var > varr
7 isLess = var < varr
8 isGreaterOrEqual = var >= varr
9 isLessOrEqual = var <= varr
```

Below the code editor is a terminal window. It shows the command 'python3 Driver.py program\_test.txt' being executed, which results in the output 'Accepted!'. The terminal prompt is 'appuser@72e058b9aab5:/program\$'.

10. EXTIENDA EL LENGUAJE PARA SOPORTAR ESTRUCTURAS DE CONTROL COMO 'IF' Y 'WHILE'.



The screenshot shows a grammar rule definition for a control structure. The rule is 'stat' and it is defined as follows:

```
4
5 stat:  expr NEWLINE          # printExpr
6       |  ID '=' expr NEWLINE  # assign
7       |  'if' expr 'then' stat+ ('else' stat+)? 'endif' # if
8       |  'while' expr 'do' stat+ 'endwhile' # while
9       |  NEWLINE              # blank
10      ;
```

## 11. CREE UN PROGRAMA QUE UTILICE UNA ESTRUCTURA 'IF'.

```

C1 Compilers-101 lab-2
M README.md program_test.txt Driver.py MiniLangLexer.py
1 // ESTO ES UN COMENTARIO
2 var = (5 + 5) * 2
3 varr = (5 + 5) * 2
4
5 if var == varr then
6     if var == 20 then
7         newVar = 10
8     else
9         newVar = 5
10    endif
11 endif
12
Terminal Command Prompt
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
Accepted!
appuser@72e058b9aab5:/program$
```

## 12. CREE UN PROGRAMA QUE UTILICE UNA ESTRUCTURA 'WHILE'.

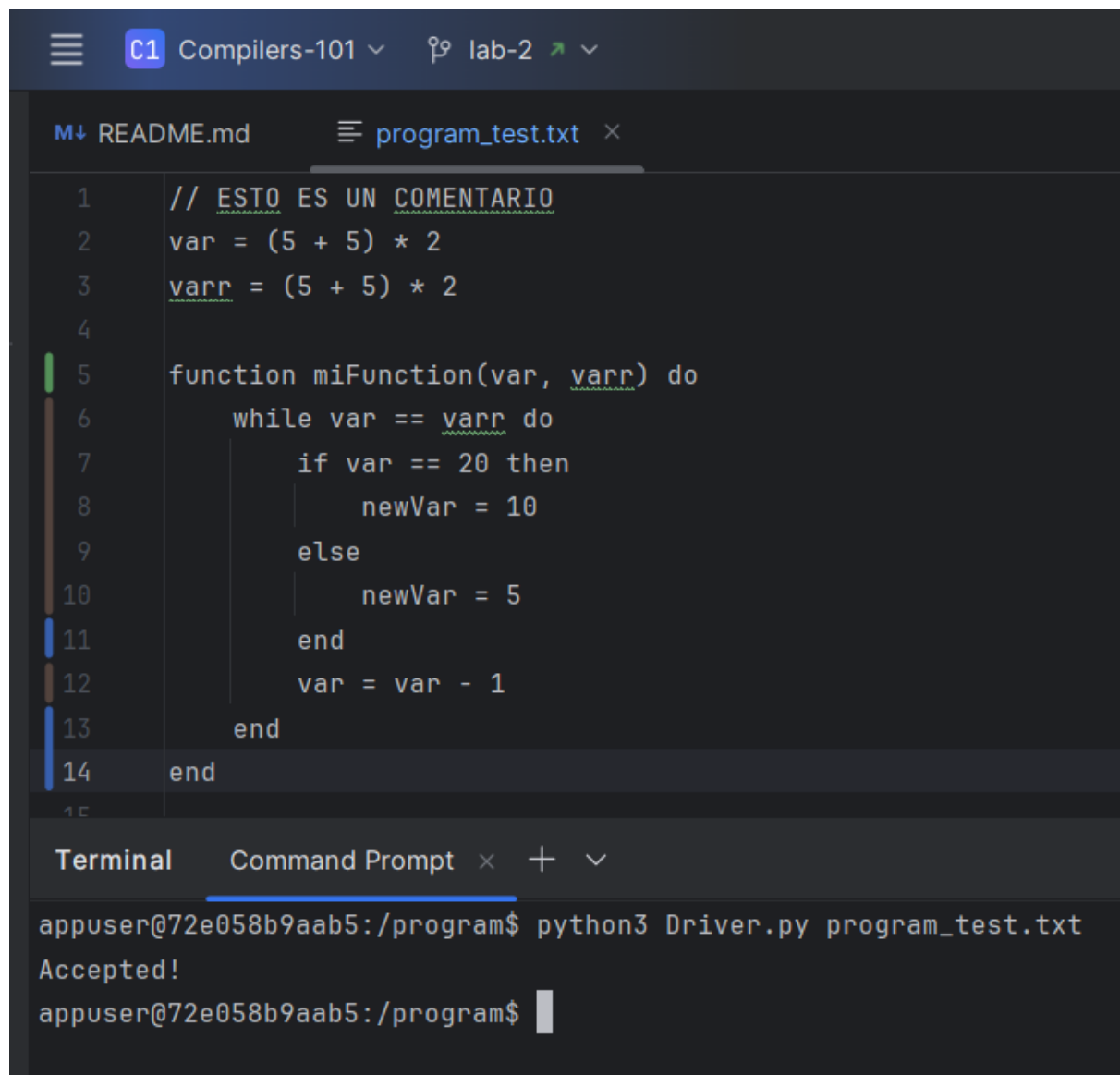
```

C1 Compilers-101 lab-2
M README.md program_test.txt
1 // ESTO ES UN COMENTARIO
2 var = (5 + 5) * 2
3 varr = (5 + 5) * 2
4
5 while var == varr do
6     if var == 20 then
7         newVar = 10
8     else
9         newVar = 5
10    endif
11    var = var - 1
12 endwhile
13
Terminal Command Prompt
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
Accepted!
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
Accepted!
appuser@72e058b9aab5:/program$
```

### 13. AGREGUE SOPORTE PARA FUNCIONES DEFINIDAS POR EL USUARIO.

```
5 stat:  expr NEWLINE          # printExpr
6      |  ID '=' expr NEWLINE   # assign
7      |  'if' expr 'then' stat+ ('else' stat+)? 'end' NEWLINE # if
8      |  'while' expr 'do' stat+ 'end' NEWLINE# while
9      |  'function' ID '(' (ID (',' ID)*)? ')' 'do' stat+ 'end' NEWLINE # function
10     |  NEWLINE               # blank
11     ;
```

### 14. CREE UN PROGRAMA QUE DEFINA Y LLAME A UNA FUNCIÓN.

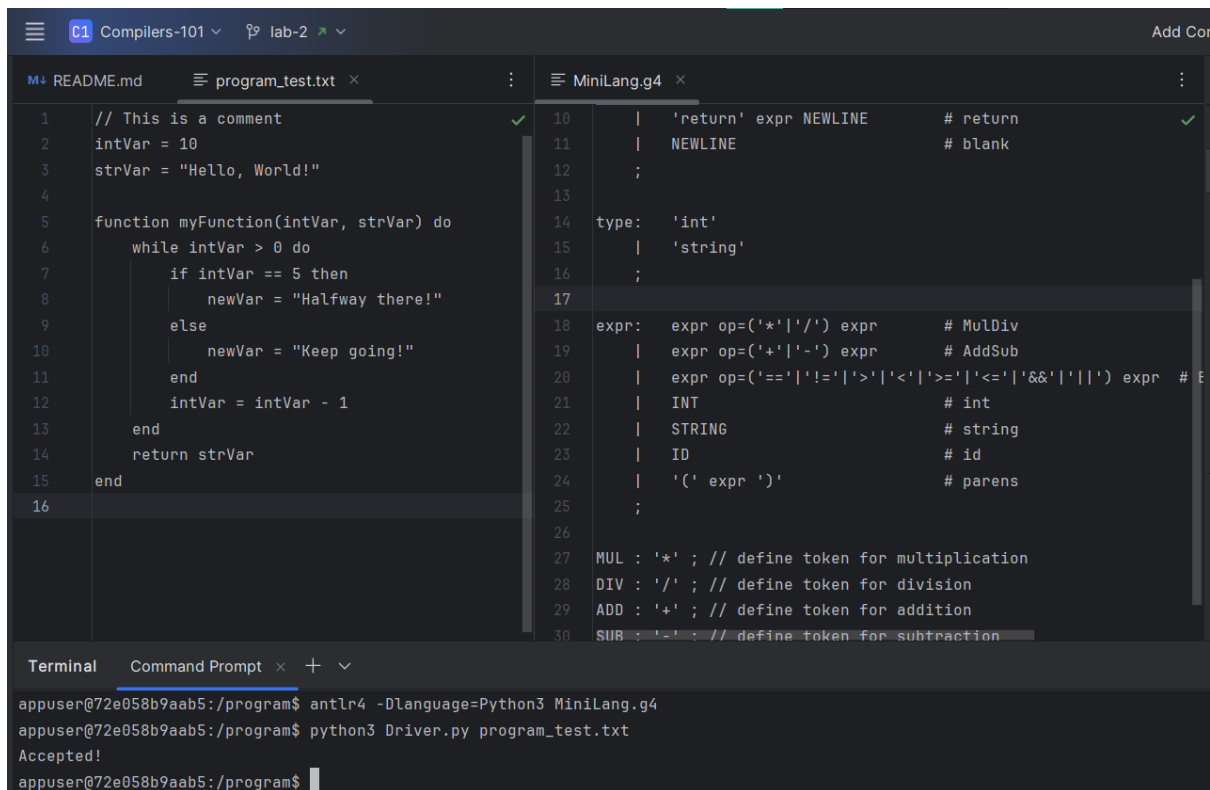


The screenshot shows a code editor with a file named `program_test.txt` open. The code defines a function `miFunction` and then calls it. The function takes two arguments, `var` and `varr`, and enters a `while` loop that continues as long as `var` equals `varr`. Inside the loop, it checks if `var` is 20; if so, it sets `newVar` to 10, otherwise it sets `newVar` to 5. It then decrements `var` by 1. The function ends with `end`. The program also includes a comment `// ESTO ES UN COMENTARIO` and two assignment statements: `var = (5 + 5) * 2` and `varr = (5 + 5) * 2`. Below the code editor, a terminal window shows the command `python3 Driver.py program_test.txt` being executed, with the output `Accepted!`.

```
1 // ESTO ES UN COMENTARIO
2 var = (5 + 5) * 2
3 varr = (5 + 5) * 2
4
5 function miFunction(var, varr) do
6     while var == varr do
7         if var == 20 then
8             newVar = 10
9         else
10            newVar = 5
11        end
12        var = var - 1
13    end
14 end
```

```
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
Accepted!
appuser@72e058b9aab5:/program$
```

15. IMPLEMENTE UN SISTEMA DE TIPOS BÁSICO QUE, ADEMÁS DE INCLUIR ENTEROS, TAMBIÉN INCLUYA CADENAS



```
1 // This is a comment
2 intVar = 10
3 strVar = "Hello, World!"
4
5 function myFunction(intVar, strVar) do
6     while intVar > 0 do
7         if intVar == 5 then
8             newVar = "Halfway there!"
9         else
10             newVar = "Keep going!"
11         end
12         intVar = intVar - 1
13     end
14     return strVar
15 end
16
```

```
10 | 'return' expr NEWLINE # return
11 | NEWLINE # blank
12 ;
13
14 type: 'int'
15 | 'string'
16 ;
17
18 expr: expr op=('*'|'/') expr # MulDiv
19 | expr op=('+'|'-') expr # AddSub
20 | expr op=('=='|'!='|'>'|'<'|'>='|'<='|'&&'|'||') expr # E
21 | INT # int
22 | STRING # string
23 | ID # id
24 | '(' expr ')' # parens
25 ;
26
27 MUL : '*' ; // define token for multiplication
28 DIV : '/' ; // define token for division
29 ADD : '+' ; // define token for addition
30 SUB : '-' ; // define token for subtraction
```

Terminal Command Prompt x + v

```
appuser@72e058b9aab5:/program$ antlr4 -Dlanguage=Python3 MiniLang.g4
appuser@72e058b9aab5:/program$ python3 Driver.py program_test.txt
Accepted!
appuser@72e058b9aab5:/program$
```