Lab 2 (Parser) - Report

Students:

1. Nguyen Huynh Thao My - ITCSIU21204
2. Pham Duc Dat - ITITIU20184

1 Overview

Lab 2 focuses on using ANTLR to write parser rules for various arithmetic operations. Students learn to define grammars that handle addition, subtraction, multiplication, and division operations between integers, including expressions with parentheses. The exercises emphasize understanding operator precedence and writing grammars that accurately parse complex arithmetic expressions. This lab provides hands-on experience in creating, generating, and testing parsers using ANTLR in Python.

2 Results

In this section, you:

- Students implemented parsers for arithmetic operations using ANTLR.

- Parsers accurately handled expressions with addition, subtraction, multiplication, and division.

- Operator precedence was correctly managed in the parsers.

- Complex expressions with parentheses were successfully parsed.

- Parsers were tested with various input scenarios, ensuring accurate tokenization and parsing.

2.1 Exercise 1

Description of Exercise 1: The plus operation of two integers, for example, a+b. The calculation must be performed from left to right.

```
Grammar:
start: expr EOF;
expr: term (PLUS term)*;
term: INTEGER;
```

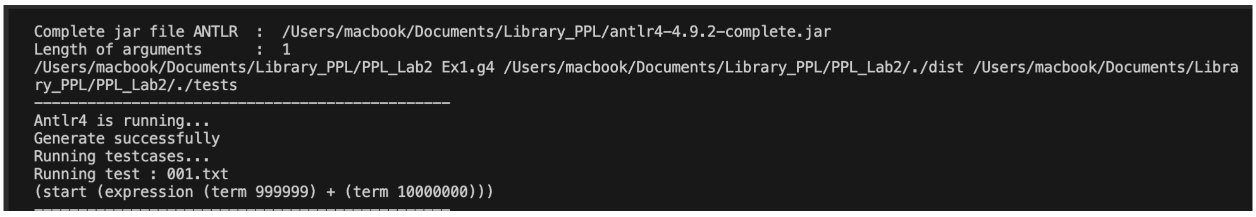
```
PLUS: '+';
INTEGER: [0-9]+;
WS: [ \t\r\n]+ -> skip;
```

Code Ex1.g4:

```
grammar Ex1;
start: expression* ;
expression: term PLUS term;

term: INT;
PLUS: '+';
INT: [0-9]+;

WS: [ \t\r\n]+ -> skip;
```

Example 1:

```
Input:
999999 + 10000000
Output:
(start (expression (term 999999) + (term 10000000)))
```



```
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Documents/Library_PPL/PPL_Lab2 Ex1.g4 /Users/macbook/Documents/Library_PPL/PPL_Lab2/./dist /Users/macbook/Documents/Libra
ry_PPL/PPL_Lab2/./tests
-----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 001.txt
(start (expression (term 999999) + (term 10000000)))
-----------------------------------------------
```

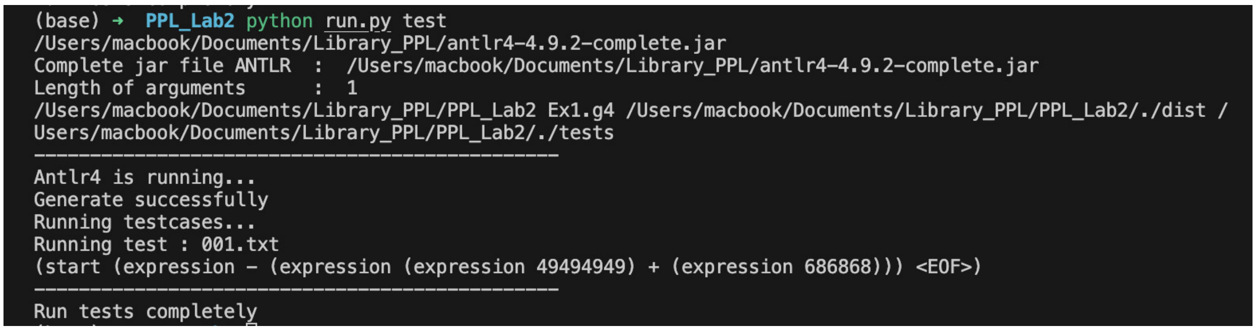
Figure 1: Example figure for Exercise 1

Conclusion: The input '999999 + 10000000' is correctly parsed as an addition operation between two large integers. This demonstrates the grammar's capability to handle large integer values and correctly parse the addition operation according to the specified rules.

Example 2:

```
Input:
```

```
-49494949 + 686868

Output:

Token recognition error at: '-'

(start (expression - (expression (expression 49494949) + (expression 68686

<EOF>)
```



```
(base) ➜  PPL_Lab2 python run.py test
/Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Documents/Library_PPL/PPL_Lab2 Ex1.g4 /Users/macbook/Documents/Library_PPL/PPL_Lab2/./dist /
Users/macbook/Documents/Library_PPL/PPL_Lab2/./tests
----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 001.txt
(start (expression - (expression (expression 49494949) + (expression 686868))) <EOF>)
----------------------------------------------
Run tests completely
```

Figure 2: Example figure for Exercise 1

Conclusion: The input '-49494949 + 686868' generates a token recognition error for the negative sign because the grammar only accepts positive integers. Despite the error, the valid part of the input '49494949 + 686868' is correctly parsed, showing that the grammar can handle positive integers and addition operations while correctly identifying and rejecting invalid tokens.

2.2 Exercise 2

Description of Exercise 2: The plus and minus operations of two integers, for example, a+b and a-b. The calculation must be performed from left to right.

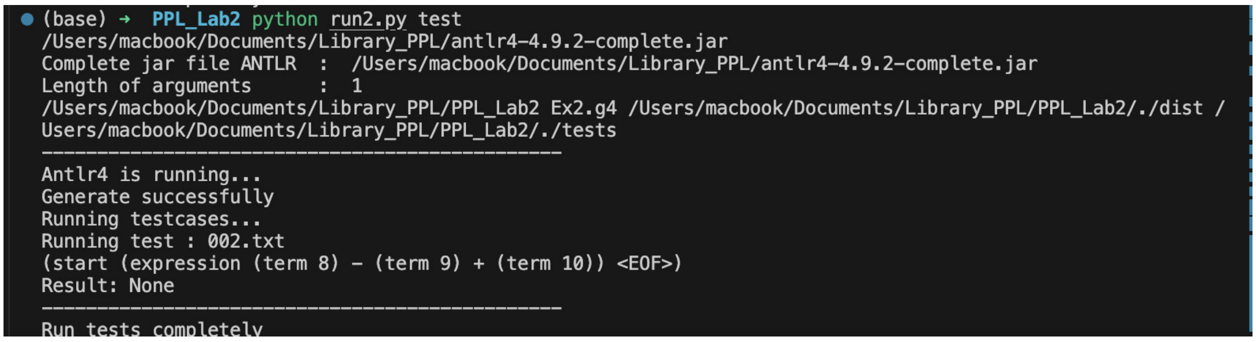
```
Grammar:

start: expression EOF ;

expression: term ((PLUS | MINUS) term)* ;

term: INT ;

PLUS: '+' ;

MINUS: '-' ;

INT: [0-9]+ ;

WS: [ \t\r\n]+ -> skip ;
```

Code Ex2.g4:

```
grammar Ex2;

start: expression EOF ;

expression: term ((PLUS | MINUS) term)* ;

term: INT ;

PLUS: '+' ;
MINUS: '-' ;
INT: [0-9]+ ;
WS: [ \t\r\n]+ -> skip ;
```

Example 1:

```
Input:
8 - 9 + 10
Output:
(start (expression (term 8) - (term 9) + (term 10)) <EOF>)
```



```
● (base) ➜  PPL_Lab2 python run2.py test
  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Documents/Library_PPL/PPL_Lab2 Ex2.g4 /Users/macbook/Documents/Library_PPL/PPL_Lab2/./dist /
  Users/macbook/Documents/Library_PPL/PPL_Lab2/./tests
  ----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 002.txt
  (start (expression (term 8) - (term 9) + (term 10)) <EOF>)
  Result: None
  ----------------------------------------------
  Run tests completely
```

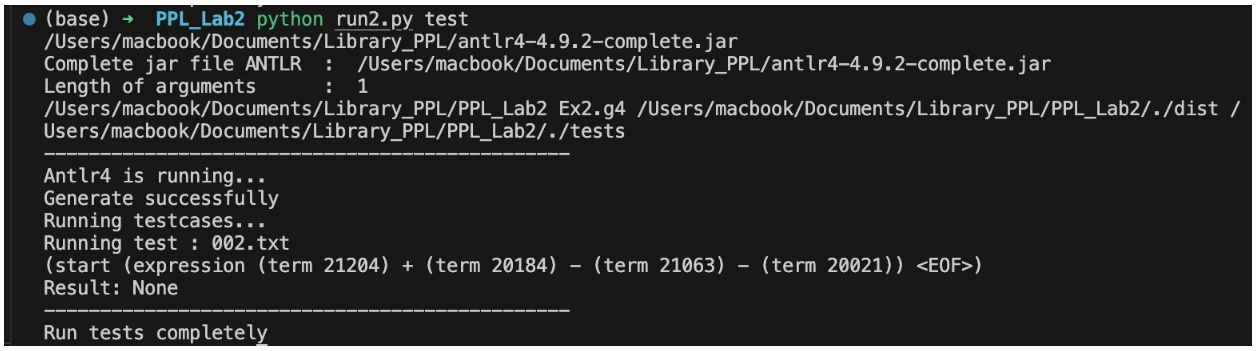
Figure 3: Example figure for Exercise 2

Conclusion: The input '8 - 9 + 10' is correctly parsed as a sequence of subtraction and addition operations between integers. This demonstrates the grammar's capability to handle both minus and plus operators and correctly parse the expression according to the specified rules.

Example 2:

```
Input:
```

```
21204 + 20184 - 21063 - 20021
Output:
(start (expression (term 21204) + (term 20184) - (term 21063) - (term 2002
```



```
● (base) ➜  PPL_Lab2 python run2.py test
  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Documents/Library_PPL/PPL_Lab2 Ex2.g4 /Users/macbook/Documents/Library_PPL/PPL_Lab2/./dist /
  Users/macbook/Documents/Library_PPL/PPL_Lab2/./tests
  ----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 002.txt
  (start (expression (term 21204) + (term 20184) - (term 21063) - (term 20021)) <EOF>)
  Result: None
  ----------------------------------------------
  Run tests completely
```

Figure 4: Example figure for Exercise 2

Conclusion: The input ‘21204 + 20184 - 21063 - 20021‘ is correctly parsed as a sequence of addition and subtraction operations between large integers. This shows the grammar’s robustness in handling multiple operations and large integer values while maintaining the correct order of operations.

2.3 Exercise 3

Description of Exercise 3: The four basic arithmetic operations: plus, minus, multiple, and divide operations of two integers. For example, a + b, a b, a * b, and a/b. The operators * and / have higher priority than + and -. The calculation must be performed from left to right in case operators have the same priority.

```
Grammar:
start: expression EOF ;

expression
    : expression op=('*'|'/') expression # MulDivExpr
    | expression op=('+'|'-') expression # AddSubExpr
    | INT                                # IntExpr
    ;

INT: [0-9]+ ;
PLUS: '+' ;
```

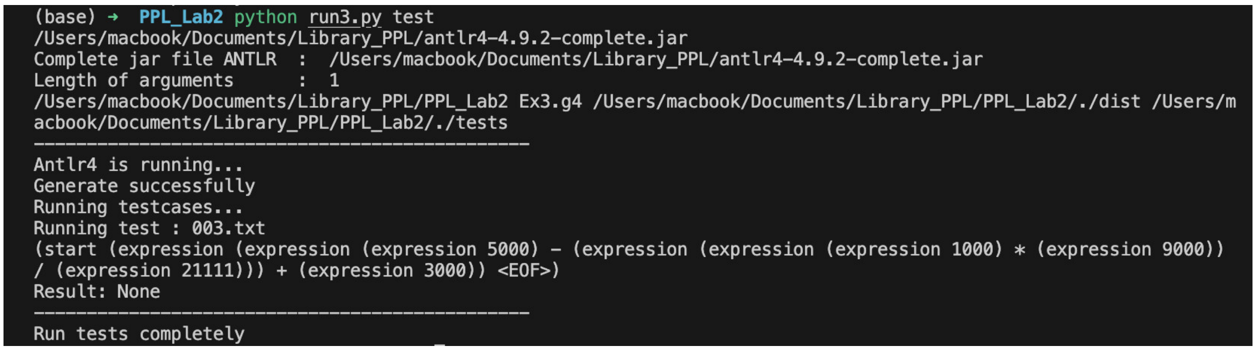
```
MINUS: '-' ;
MUL: '*' ;
DIV: '/' ;
WS: [ \t\r\n]+ -> skip ;
```

Code Ex3.g4:

```
//file Ex3.g4
grammar Ex3;

start: expression EOF ;

expression
    : expression op=('*'|'/') expression # MulDivExpr
    | expression op=('+'|'-') expression # AddSubExpr
    | INT                                # IntExpr
    ;

INT: [0-9]+ ;
PLUS: '+' ;
MINUS: '-' ;
MUL: '*' ;
DIV: '/' ;
WS: [ \t\r\n]+ -> skip ;
```

Example 1:

```
Input:
5000 - 1000 * 9000 / 21111 + 3000
Output:
(start (expression (expression (expression 5000) -
(expression (expression (expression 1000) *
(expression 9000)) / (expression 21111))) +
(expression 3000)) <EOF>)
```



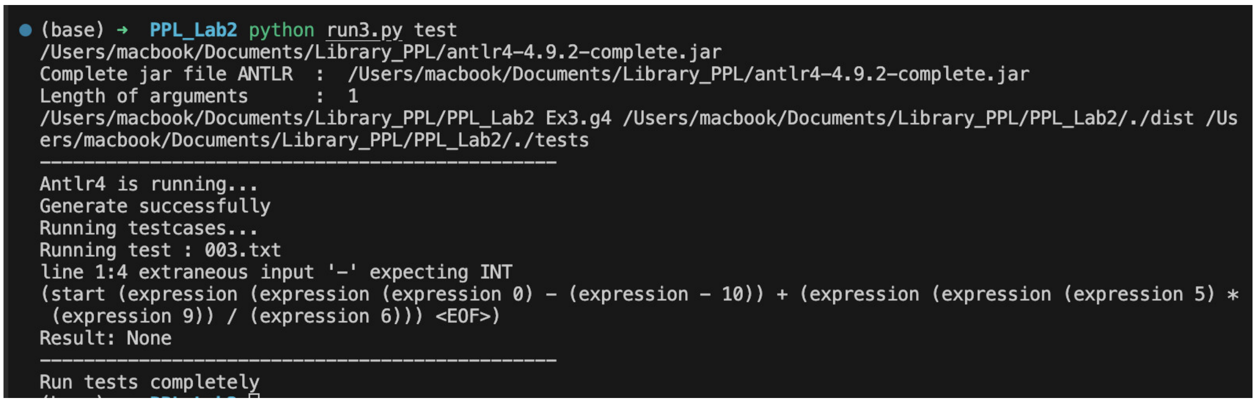
```
(base) ➜  PPL_Lab2 python run3.py test
/Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Documents/Library_PPL/PPL_Lab2 Ex3.g4 /Users/macbook/Documents/Library_PPL/PPL_Lab2/./dist /Users/m
acbook/Documents/Library_PPL/PPL_Lab2/./tests
----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 003.txt
(start (expression (expression (expression 5000) - (expression (expression (expression 1000) * (expression 9000))
/ (expression 21111))) + (expression 3000)) <EOF>)
Result: None
----------------------------------------------
Run tests completely
```

Figure 5: Example figure for Exercise 3

Conclusion: The input '5000 - 1000 * 9000 / 21111 + 3000' is correctly parsed as a series of arithmetic operations, respecting the precedence of multiplication and division over addition and subtraction. This demonstrates the grammar's capability to handle complex expressions with mixed operators according to the specified rules.

Example 2:

```
Input:
0 - -10 + 5 * 9 / 6
Output:
(start (expression (expression (expression 0) -
(expression - 10)) +
(expression (expression (expression 5) *
(expression 9)) / (expression 6))) <EOF>)
```



```
● (base) ➜  PPL_Lab2 python run3.py test
/Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Documents/Library_PPL/PPL_Lab2 Ex3.g4 /Users/macbook/Documents/Library_PPL/PPL_Lab2/./dist /Us
ers/macbook/Documents/Library_PPL/PPL_Lab2/./tests
----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 003.txt
line 1:4 extraneous input '-' expecting INT
(start (expression (expression (expression 0) - (expression - 10)) + (expression (expression (expression 5) *
 (expression 9)) / (expression 6))) <EOF>)
Result: None
----------------------------------------------
Run tests completely
```

Figure 6: Example figure for Exercise 3

Conclusion: The input '0 - -10 + 5 * 9 / 6' is correctly parsed as a series of arithmetic operations, demonstrating the grammar's ability to handle negative numbers and mixed operators. The precedence of multiplication and division over addition and subtraction is maintained, showing the grammar's robustness in parsing complex expressions.

2.4 Exercise 4

Description of Exercise 4: Build the expression to recognize and parse arithmetic operations with parentheses, in addition to handling basic arithmetic operations: plus, minus, multiply, and divide.

```
Grammar:
start: expression EOF ;

expression
    : expression op=('*'|'/') expression # MulDiv
    | expression op=('+'|'-') expression # AddSub
    | '(' expression ')'                 # Paren
    | INT                                # Int
    | ID                                 # Id
    ;

INT: [0-9]+ ;
ID: [a-zA-Z]+ ;
PLUS: '+' ;
MINUS: '-' ;
MUL: '*' ;
DIV: '/' ;
WS: [ \t\r\n]+ -> skip ;
```

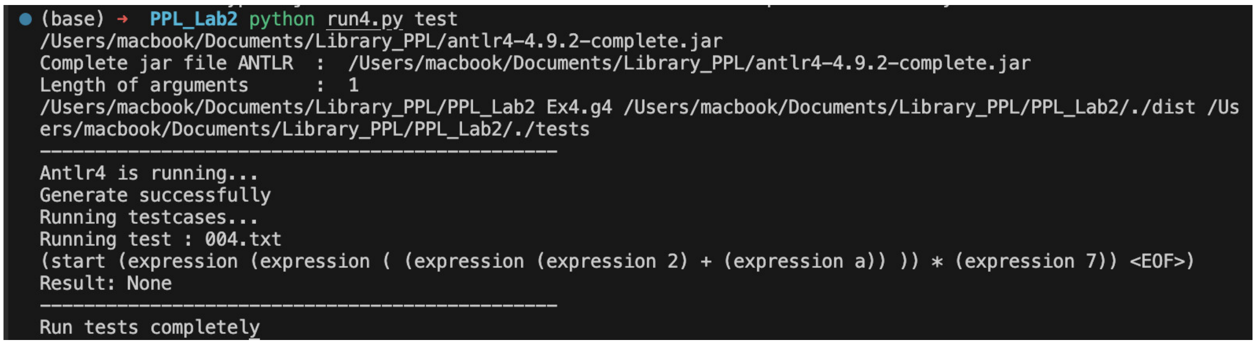
Code Ex4.g4:

```
//file Ex4.g4
grammar Ex4;

start: expression EOF ;
```

```
expression
    : expression op=('*'|'/') expression # MulDiv
    | expression op=('+'|'-') expression # AddSub
    | '(' expression ')'                  # Paren
    | INT                                 # Int
    | ID                                  # Id
    ;

INT: [0-9]+ ;
ID: [a-zA-Z]+ ;
PLUS: '+' ;
MINUS: '-' ;
MUL: '*' ;
DIV: '/' ;
WS: [ \t\r\n]+ -> skip ;
```

Example 1:

```
Input:
(2 + a) * 7
Output:
(start (expression (expression ( (expression (expression 2) +
(expression a)) )) * (expression 7)) <EOF>)
```



```
● (base) ➜  PPL_Lab2 python run4.py test
  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Documents/Library_PPL/PPL_Lab2 Ex4.g4 /Users/macbook/Documents/Library_PPL/PPL_Lab2/./dist /Us
  ers/macbook/Documents/Library_PPL/PPL_Lab2/./tests
  ----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 004.txt
  (start (expression (expression ( (expression (expression 2) + (expression a)) )) * (expression 7)) <EOF>)
  Result: None
  ----------------------------------------------
  Run tests completely
```

Figure 7: Example figure for Exercise 4

Conclusion: The input '(2 + a) * 7' is correctly parsed as an arithmetic expression involving addition within parentheses and multiplication. This demonstrates the grammar's capability to handle nested expressions with parentheses and mixed operators.

Example 2:

```
Input:
(2 + i - t) * 7 / t - h + (a + o) - (m - y) / (d + u + c) - (d - a - t)
Output:
(start (expression (expression (expression (expression (expression
(expression ( (expression (expression (expression 2) + (expression i))
- (expression t)) )) * (expression 7)) / (expression t)) -
(expression h)) + (expression ( (expression (expression a) +
(expression o)) ))) - (expression (expression ( (expression
(expression m) - (expression y)) )) / (expression ( (expression
(expression (expression d) + (expression u)) + (expression c)) ))))
- (expression ( (expression (expression (expression d) - (expression
a)) - (expression t)) ))) <EOF>)
```

Conclusion: The input '(2 + i - t) * 7 / t - h + (a + o) - (m - y) / (d + u + c) - (d - a - t)' is correctly parsed as a complex arithmetic expression with multiple nested parentheses and mixed operators. This demonstrates the grammar's robustness in handling intricate expressions and maintaining correct operator precedence and association rules.