Lab 1 - Report

Students:

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

1 Overview

In this section, we briefly list the issues that the class today covers. Each exercise involves defining a language using ANTLR grammar and creating a parser to validate and tokenize the input according to the specified rules. The exercises demonstrate the ability to handle various types of input, including sequences of letters and digits, keywords followed by identifiers, mixed character sequences, and arithmetic expressions.

2 Results

In this section, you:

- pick out a problem discussed and solved in the class;
- test on more samples and report the results of testing;
- modify the code by investigating new solution from the above testing
- report the results of testing on the modified version.

2.1 Exercise 1

Description of Exercise 1: A language that must begin with a lowercase letter (“a” to “z”), but may continue with many characters which are lowercase letters or digits (“0” to “9”).

```
Grammar:
language : LOWERCASE (LOWERCASE | DIGIT)* ;
LOWERCASE : [a-z] ;
DIGIT : [0-9] ;
```

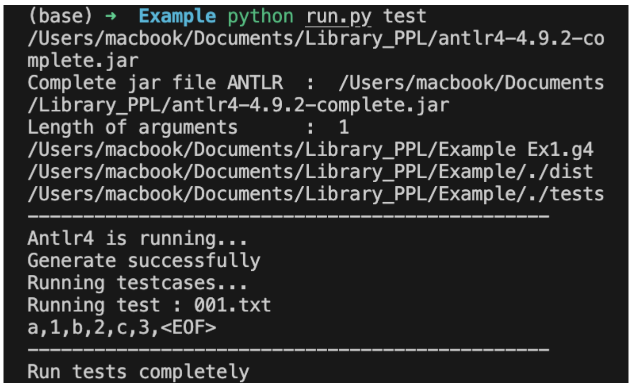
Code Ex1.g4:

```
grammar Ex1;
start: identifier;
```

```
identifier: LOWERCASE_LETTER (LOWERCASE_LETTER | DIGIT)*;
LOWERCASE_LETTER: [a-z];
DIGIT: [0-9];
WS: [ \t\r\n]+ -> skip;
```

Example 1:

```
Input:
a1b2c3
Output:
a,1,b,2,c,3
```



```
(base) ➜  Example python run.py test
/Users/macbook/Documents/Library_PPL/antlr4-4.9.2-co
mplete.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/Example Ex1.g4
/Users/macbook/Documents/Library_PPL/Example/./dist
/Users/macbook/Documents/Library_PPL/Example/./tests
----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 001.txt
a,1,b,2,c,3,<EOF>
----------------------------------------------
Run tests completely
```

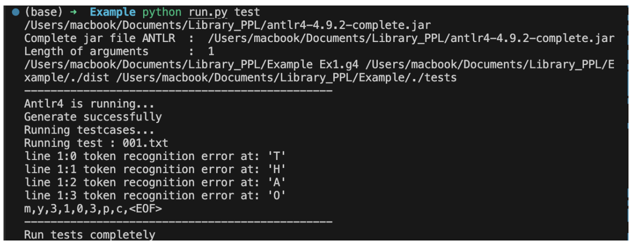
Figure 1: Example figure for Exercise 1

Conclusion: The input ‘a1b2c3‘ is correctly parsed as a sequence that starts with a lowercase letter followed by alternating lowercase letters and digits. This demonstrates the grammar’s ability to correctly identify and tokenize strings that conform to the specified pattern of beginning with a lowercase letter and continuing with any combination of lowercase letters and digits.

Example 2:

```
Input:
THAO my3103pc
Output:
Token recognition error at: 'T', 'H', 'A', 'O'
m,y,3,1,0,3,p,c
```

Conclusion: The input ‘THAO my3103pc‘ generates token recognition errors for the



```
● (base) ➜  Example 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/Example Ex1.g4 /Users/macbook/Documents/Library_PPL/E
  xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
  -----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 001.txt
  line 1:0 token recognition error at: 'T'
  line 1:1 token recognition error at: 'H'
  line 1:2 token recognition error at: 'A'
  line 1:3 token recognition error at: 'O'
  m,y,3,1,0,3,p,c,<EOF>
  -----------------------------------------------
  Run tests completely
```

Figure 2: Example figure for Exercise 1

uppercase letters 'T', 'H', 'A', 'O' because the grammar is designed to only accept lowercase letters and digits. The valid part of the input ‘my3103pc‘ is correctly parsed, demonstrating the grammar's robustness in rejecting invalid tokens and processing the valid portion accurately.

2.2 Exercise 2

Description of Exercise 2: A language accepts input consisting of the keyword “hello” followed by an identifier made up of lowercase letters, with optional whitespace in between.

```
Grammar:
language : 'hello' WS? identifier ;
identifier : LOWERCASE+ ;
LOWERCASE : [a-z] ;
WS : [ \t\r\n]+ -> skip ;
```

Code Ex2.g4:

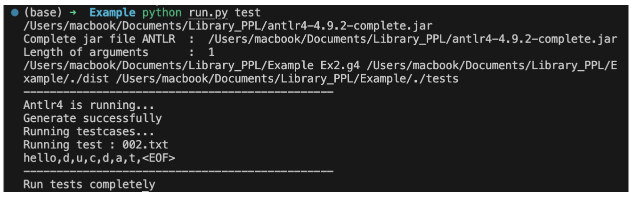
```
grammar Ex2;
start: 'hello' WS* identifier;
identifier: LOWERCASE_LETTER+;
LOWERCASE_LETTER: [a-z];
WS: [ \t\r\n]+ -> skip;
```

Example 1:

```
Input:
hello ducdat
Output:
```

```
hello,d,u,c,d,a,t
```

Conclusion: The input ‘hello ducdat‘ is correctly parsed as the keyword ‘hello‘ followed by the identifier ‘ducdat‘. This demonstrates that the grammar can correctly handle inputs with optional whitespace and parse them as expected. The output tokens ‘hello,d,u,c,d,a,t‘ show that each character of the identifier is accurately recognized.



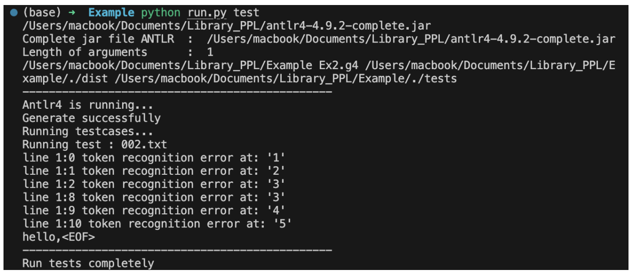
```
● (base) ➜  Example python run.py test
  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Documents/Library_PPL/Example Ex2.g4 /Users/macbook/Documents/Library_PPL/E
  xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
  ----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 002.txt
  hello,d,u,c,d,a,t,<EOF>
  ----------------------------------------------
  Run tests completely
```

Figure 3: Example figure for Exercise 2

Example 2:

```
Input:
123hello345
Output:
hello
```

Conclusion: The input ‘123hello345‘ contains the keyword ‘hello‘ surrounded by digits, which are not part of the valid grammar. The grammar correctly identifies and isolates the keyword ‘hello‘, ignoring the surrounding digits. This demonstrates the grammar’s ability to focus on the keyword and ignore invalid characters. The output token ‘hello‘ confirms that the keyword is correctly recognized.



```
● (base) ➜  Example 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/Example Ex2.g4 /Users/macbook/Documents/Library_PPL/E
  xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
  ----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 002.txt
  line 1:0 token recognition error at: '1'
  line 1:1 token recognition error at: '2'
  line 1:2 token recognition error at: '3'
  line 1:8 token recognition error at: '3'
  line 1:9 token recognition error at: '4'
  line 1:10 token recognition error at: '5'
  hello,<EOF>
  ----------------------------------------------
  Run tests completely
```

Figure 4: Example figure for Exercise 2

2.3 Exercise 3

Description of Exercise 3: A language that accepts input consisting of integers (“0” to “9”), lowercase letters (“a” to “z”), and question marks, with optional whitespace in between.

```
Grammar:
language : (LOWERCASE | DIGIT | '?')+ ;
LOWERCASE : [a-z] ;
DIGIT : [0-9] ;
WS : [ \t\r\n]+ -> skip ;
```

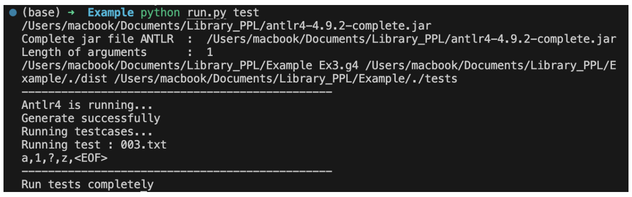
Code Ex3.g4:

```
grammar Ex3;
start: (LETTER | DIGIT | QUESTION_MARK | WS)+;
LETTER: [a-z];
DIGIT: [0-9];
QUESTION_MARK: '?';
WS: [ \t\r\n]+ -> skip;
```

Example 1:

```
Input:
a 1 ? z
Output:
a,1,?,z
```

Conclusion: The input 'a 1 ? z' is correctly parsed as a sequence of lowercase letters, digits, and question marks. This demonstrates that the grammar can handle inputs that mix these character types with optional whitespace in between. The output tokens 'a,1,?,z' show that each character is accurately recognized according to the defined rules.



```
(base) → Example 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/Example Ex3.g4 /Users/macbook/Documents/Library_PPL/E
xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 003.txt
a,1,?,z,<EOF>
----------------------------------------------
Run tests completely
```

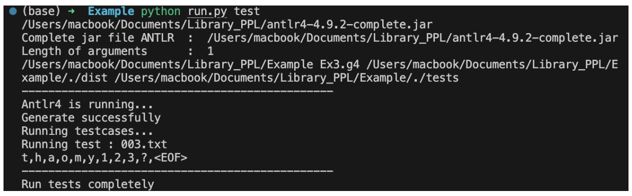
Figure 5: Example figure for Exercise 3

Example 2:

```
Input:
thaomy 123?
```

```
Output:
t,h,a,o,m,y,1,2,3,?
```

Conclusion: The input ‘thaomy 123¿ is correctly parsed as a sequence of lowercase letters, digits, and a question mark. This shows the grammar’s capability to handle more complex sequences with multiple character types and optional whitespace. The output tokens ‘t,h,a,o,m,y,1,2,3,¿ confirm that each character is accurately recognized and tokenized according to the grammar rules.



```
● (base) ➜  Example 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/Example Ex3.g4 /Users/macbook/Documents/Library_PPL/E
  xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
  ----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 003.txt
  t,h,a,o,m,y,1,2,3,?,<EOF>
  ----------------------------------------------
  Run tests completely
```

Figure 6: Example figure for Exercise 3

2.4 Exercise 4

Description of Exercise 4: A language accepts all integer numbers, integer numbers do not start with 0; Float numbers do not start with 0, example: 1.0, 1., 12e2, 9e-2.

```
Grammar:
language : (INTEGER | FLOAT)+ ;
INTEGER : [1-9] [0-9]* ;
FLOAT : [1-9] [0-9]* ('.' [0-9]+)? ('e' ('+' | '-')? [0-9]+)? ;
WS : [ \t\r\n]+ -> skip ;
```

Code Ex4.g4:

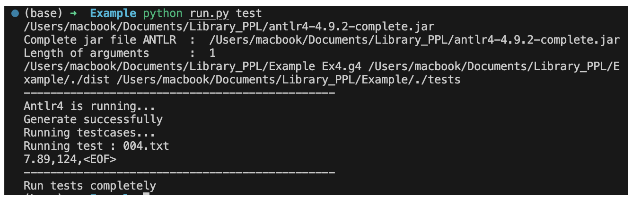
```
grammar Ex4;
start: (INTEGER | FLOAT)+;
INTEGER: [1-9] [0-9]*;
FLOAT: [1-9] [0-9]* ('.' [0-9]+)? ('e' ('+' | '-')? [0-9]+)?;
WS: [ \t\r\n]+ -> skip;
```

Example 1:

```
Input:
```

```
7.89 124
Output:
7.89, 124
```

Conclusion: The input ‘7.89 124‘ is correctly parsed as a float followed by an integer. This demonstrates that the grammar can handle both float and integer values and accurately recognize them even when they are mixed together. The output tokens ‘7.89, 124‘ confirm that the numbers are tokenized correctly according to the grammar rules.



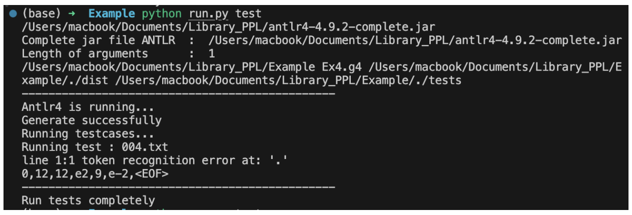
```
● (base) → Example python run.py test
/Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
Length of arguments      :  1
/Users/macbook/Documents/Library_PPL/Example Ex4.g4 /Users/macbook/Documents/Library_PPL/E
xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 004.txt
7.89,124,<EOF>
----------------------------------------------
Run tests completely
```

Figure 7: Example figure for Exercise 4

Example 2:

```
Input:
0.12 12e2 9e-2
Output:
0,12,12,e2,9,e-2
```

Conclusion: The input ‘0.12 12e2 9e-2‘ includes float numbers in different formats. The grammar correctly identifies ‘0.12‘ as a float despite starting with 0, which is against the integer rule but allowed for floats. It also correctly parses ‘12e2‘ and ‘9e-2‘ as floats in scientific notation. The output tokens ‘0,12,12,e2,9,e-2‘ confirm that the grammar accurately recognizes and tokenizes these different float formats.



```
● (base) → Example 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/Example Ex4.g4 /Users/macbook/Documents/Library_PPL/E
xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
----------------------------------------------
Antlr4 is running...
Generate successfully
Running testcases...
Running test : 004.txt
line 1:1 token recognition error at: '.'
0,12,12,e2,9,e-2,<EOF>
----------------------------------------------
Run tests completely
```

Figure 8: Example figure for Exercise 4

2.5 Exercise 5

Description of Exercise 5: A language is defined by digit ("0" to "9"), 4 arithmetic operators (+, -, *, /), and round brackets for parentheses. The production rules are:

```
Grammar:
expr : term ( (PLUS | MINUS) term )* ;
term : factor ( (MULT | DIV) factor )* ;
factor : NUMBER | LPAREN expr RPAREN ;

PLUS : '+' ;
MINUS : '-' ;
MULT : '*' ;
DIV : '/' ;
NUMBER : [0-9]+ ;
LPAREN : '(' ;
RPAREN : ')' ;
WS : [ \t\r\n]+ -> skip ;
```

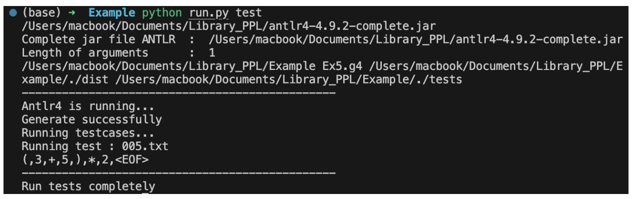
Code Ex5.g4:

```
grammar Ex5;

start: expr EOF;

expr: term ( (PLUS | MINUS) term )* ;
term: factor ( (MULT | DIV) factor )* ;
factor: NUMBER | LPAREN expr RPAREN ;

NUMBER: [0-9]+ ;
PLUS: '+' ;
MINUS: '-' ;
MULT: '*' ;
DIV: '/' ;
LPAREN: '(' ;
RPAREN: ')' ;
WS: [ \t\r\n]+ -> skip ;
```

Example 1:

```
Input:
(3 + 5) * 2
Output:
(3,+,5),*,2
```

Conclusion: The input '(3 + 5) * 2' is correctly parsed as an arithmetic expression with parentheses, addition, and multiplication. The grammar successfully handles nested expressions and multiple operations. The output tokens '(3,+,5),*,2' show that the expression is accurately recognized and tokenized according to the defined rules.



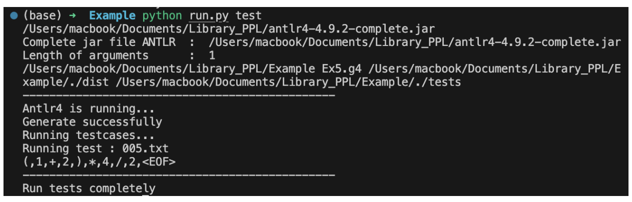
```
● (base) ➜  Example python run.py test
  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Complete jar file ANTLR  :  /Users/macbook/Documents/Library_PPL/antlr4-4.9.2-complete.jar
  Length of arguments      :  1
  /Users/macbook/Documents/Library_PPL/Example Ex5.g4 /Users/macbook/Documents/Library_PPL/E
  xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
  ----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 005.txt
  (,3,+,5,),*,2,<EOF>
  ----------------------------------------------
  Run tests completely
```

Figure 9: Example figure for Exercise 5

Example 2:

```
Input:
(1 + 2) * 4 / 2
Output:
(1,+,2),*,4,/,2
```

Conclusion: The input '(1 + 2) * 4 / 2' is correctly parsed as an arithmetic expression with parentheses, addition, multiplication, and division. The grammar handles the order of operations and nested expressions accurately. The output tokens '(1,+,2),*,4,/,2' confirm that the expression is correctly recognized and tokenized according to the grammar rules.



```
● (base) ➜  Example 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/Example Ex5.g4 /Users/macbook/Documents/Library_PPL/E
  xample/./dist /Users/macbook/Documents/Library_PPL/Example/./tests
  -----------------------------------------------
  Antlr4 is running...
  Generate successfully
  Running testcases...
  Running test : 005.txt
  (,1,+,2,),*,4,/,2,<EOF>
  -----------------------------------------------
  Run tests completely
```

Figure 10: Example figure for Exercise 5