

# VÔ TIẾN

Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)  
<https://www.facebook.com/groups/khmt.ktmt.cse.bku>



## Nguyên Lý Ngôn Ngữ Lập Trình (PPL)

---

PPL1 - HK242

### Task 2 - PARSER

---

Thảo luận kiến thức CNTT trường BK  
về KHMT(CScience), KTMT(CEngineering)  
<https://www.facebook.com/groups/khmt.ktmt.cse.bku>

## Mục lục

<b>1</b>	<b>Lý thuyết Lexical structure</b>	<b>2</b>
<b>2</b>	<b>Ví dụ Lexer</b>	<b>5</b>
<b>3</b>	<b>Cách run và test BTL Lexer</b>	<b>7</b>
3.1	Cách hoạt động của lệnh python3/python run.py test LexerSuite .....	7
3.2	Test Case file test/Lexer/LexerSuite.py .....	7
3.3	Chạy test case .....	8
<b>4</b>	<b>Bài tập phần Lexer</b>	<b>9</b>
<b>5</b>	<b>Lexer (Deadline 23:59 16/1)</b>	<b>10</b>
5.1	Kiến thức .....	10
5.2	BTL .....	10



# 1 Lý thuyết Lexical structure

**Tokens** là khối xây dựng cơ bản của chương trình, chuỗi ký tự ngắn nhất có ý nghĩa riêng, Chúng là các thành phần cơ bản hay nguyên tố để xây dựng lên chương trình  
Dễ hiểu nhất trong quá trình sản xuất bánh mì thì các nguyên vật liệu như **bột**, **hương vị**, **nước** ... là các **Tokens** mỗi thứ đều có ý nghĩa riêng của nó.

- **Keywords** là các *Tokens* đã mặc định trước.
- **Operators** là các toán tử cho phép hiện thực trong chương trình
- **Separators** các kí hiệu phân cách của chúng
- **Identifiers** tên của biến, thuộc tính, hàm, class loại này là vô số *Tokens* được tạo ra
- **Literal** là chuỗi trong ngôn ngữ để thể hiện giá trị số nguyên, một chuỗi
- **Lexical errors** gồm 3 loại
  - **Unclosed string** không có kí tự để đóng chuỗi lại khi lập trình thì các phần sau hiện xanh lá trong *vscode*
  - **Illegal escape** in string một kí tự không cho phép trong ngôn ngữ
  - **Error token** không có *token* nào thỏa mãn

## Chú ý

- *Tokens* luôn bắt được chuỗi dài nhất có thể
- *Tokens* chuỗi bằng nhau thì ưu tiên chuỗi đầu tiên gặp được
- Nên viết tất cả các kí tự hoa (nếu không thì bắt buộc kí tự đầu viết hoa ràng buộc của ngôn ngữ rồi)

## Danh sách các kí hiệu và cú pháp thường dùng trong ANTLR

1. 'kí tự'
  - **Ý nghĩa:** Bất kỳ đoạn văn bản nào được đặt trong dấu nháy đơn ('...') sẽ được khớp đúng theo nội dung bên trong, bao gồm cả chữ, số hay kí tự đặc biệt.
  - **Ví dụ:** 'a' khớp đúng với kí tự a. Hoặc 'xyz' khớp đúng với xyz.
2. A B
  - **Ý nghĩa:** Yêu cầu A xuất hiện trước, rồi ngay sau đó là B.
  - **Ví dụ:** 'a' 'b' khớp với chuỗi "ab". Nếu A = 'a' và B = 'b', bạn phải thấy ab liền kề.
3. A | B
  - **Ý nghĩa:** Cho phép chọn một trong hai dạng A hoặc B.
  - **Ví dụ:** 'a' | 'b' khớp với a hoặc b. Tức là chuỗi a hoặc b đều được chấp nhận.
4. 'text'
  - **Ý nghĩa:** Tương tự 'kí tự', nhưng biểu diễn một chuỗi nhiều kí tự liên tiếp (chuỗi tĩnh).
  - **Ví dụ:** 'Hello' khớp đúng với Hello. Chuỗi này không chấp nhận thiếu hay thừa kí tự nào.
5. A?
  - **Ý nghĩa:** A có thể xuất hiện hoặc không (tương đương A | rỗng).
  - **Ví dụ:** 'a'? khớp với '' (rỗng) hoặc 'a'. Tức là có a cũng được, mà không có cũng không sao.
6. A\*
  - **Ý nghĩa:** A có thể lặp lại 0 hoặc nhiều lần ( {ε, A, AA, AAA, ...} ).



- **Ví dụ:** 'a'\* có thể khớp '' (chuỗi rỗng), 'a', 'aa', 'aaa', ...
7. A+
- **Ý nghĩa:** A lặp lại 1 hoặc nhiều lần ( {A, AA, AAA, ...} ).
  - **Ví dụ:** 'a'+ khớp với 'a', 'aa', 'aaa', ... (Không được rỗng, phải có ít nhất một a.)
8. [a-z]
- **Ý nghĩa:** Chọn một kí tự trong khoảng a đến z (chữ thường).
  - **Ví dụ:** [a-z] khớp với 'a', 'b', ..., 'z'.
9. [A-C]
- **Ý nghĩa:** Chọn một kí tự trong khoảng A đến C (chữ hoa).
  - **Ví dụ:** [A-C] khớp với A, B, hoặc C.
10. [0-9]
- **Ý nghĩa:** Chọn một chữ số trong khoảng 0 đến 9.
  - **Ví dụ:** [0-9] khớp với 0, 1, ..., 9.
11. [a-zA-Z0-9]
- **Ý nghĩa:** Chọn một kí tự trong khoảng a-z, A-Z hoặc 0-9.
  - **Ví dụ:** [a-zA-Z0-9] khớp với các chữ thường hoặc chữ số, chẳng hạn a, b, ..., 9.
12. [a-zA-Z0-9]
- **Ý nghĩa:** Chọn một kí tự trong các khoảng a-z, A-Z hoặc 0-9.
  - **Ví dụ:** [a-zA-Z0-9] khớp với a, Z, 6, v.v.
13. \n
- **Ý nghĩa:** Kí tự xuống dòng (newline).
  - **Ví dụ:** Trong mã nguồn, \n đại diện cho việc xuống dòng.
14. \f \r \\
- **Ý nghĩa:** Bao gồm \r (quay về đầu dòng), \f (sang trang mới), và \\ (dấu gạch chéo ngược).
  - **Ví dụ:** Có thể gặp chúng khi xử lý chuỗi hoặc văn bản đa dòng.
15. . (đầu chấm)
- **Ý nghĩa:** Khớp với bất kỳ kí tự nào trong ASCII (trừ kí tự xuống dòng trong một số chế độ).
  - **Ví dụ:** '.' có thể khớp với a, 1, '?', v.v.
16. ~ [0-9]
- **Ý nghĩa:** Chọn bất kỳ kí tự nào thuộc ASCII nhưng ngoại trừ 0-9.
  - **Ví dụ:** ~ [0-9] có thể khớp a, '?', '!', ...
17. [a] -> skip
- **Ý nghĩa:** Khi bắt gặp kí tự a, ta bỏ qua (không sinh token).
  - **Ví dụ:** 'abc' trong đó 'a' sẽ bị bỏ qua, còn bc có thể được phân tích tiếp.
18. fragment INT: [0-9]+;
- **Ý nghĩa:** fragment là một đoạn (rule) chỉ dùng lại bên trong các rule lexer khác, không thể khörp độc lập. Ví dụ, INT này mô tả "một hoặc nhiều chữ số".



- **Ví dụ:** fragment INT: [0-9]+; có thể được dùng trong rule khác như NUMBER: INT ('.' INT)?; để mô tả số nguyên hoặc số thực.
19. Biểu thức { self.text = self.text[1:-1]; }
- **Ý nghĩa:** Đoạn code Python trong dấu {...} cho phép tùy biến việc xử lý chuỗi vừa khớp. Ví dụ: self.text[1:-1] sẽ bỏ đi kí tự đầu và cuối trong self.text.
  - **Tham khảo:** [https://www.w3schools.com/python/python\\_strings\\_slicing.asp](https://www.w3schools.com/python/python_strings_slicing.asp)



## 2 Ví dụ Lexer

### 1. Keywords

```
RETURN: 'return';
STRING: 'string';
TRUE: 'true';
WHILE: 'while';
VOID: 'void';
```

### 2. Operators

```
ADD: '+';
SUB: '-';
MUL: '*';
DIV: '/';
MOD: '%';
```

### 3. Separators

```
LSB: '[';
RSB: ']';
LP: '(';
RP: ')';
CM: ',';
DOT: '.';
```

### 4. Comment python

```
COMMENTS: '# #' ~[\n]* -> skip
```

### 5. Comment C++

```
COMMENTS: (( '//' ~[\n]*) | ('/*' .*? '*/')) -> skip;
```

### 6. Identifiers PHP

```
ID: '$' [a-zA-Z_][a-zA-Z0-9_]*;
```

### 7. Identifiers C++

```
ID: [a-zA-Z_][a-zA-Z0-9_]*;
```

### 8. Identifiers JAVA

- Tên có thể chứa các chữ cái, chữ số, dấu gạch dưới và ký hiệu đơ la
- Tên phải bắt đầu bằng một chữ cái
- Tên phải bắt đầu bằng chữ cái viết thường và không được chứa khoảng trắng
- Tên cũng có thể bắt đầu bằng \$ và \_
- Tên có phân biệt chữ hoa chữ thường

```
ID: [a-zA-Z_][a-zA-Z0-9_]*;
```

### 9. Number Literals (BTL HK232 ZCode)

Tất cả các số trong ZCode được coi là số thực trong C/C++, giống như số thực. Số ZCode bao gồm phần nguyên, phần thập phân và phần mũ. Phần nguyên là một dãy gồm một hoặc nhiều chữ số. Phần thập phân là dấu thập phân, theo sau tùy chọn là một số chữ số. Phần số mũ bắt đầu bằng ký tự 'e' hoặc 'E', theo sau là dấu '+' hoặc '-' tùy chọn, sau đó là một hoặc nhiều chữ số. Phần thập phân hoặc phần số mũ có thể được bỏ qua.

```
NUMBER_LIT: DIGITS OPT_FRAC OPT_EXP;
fragment DIGIT: [0-9];
fragment DIGITS: DIGIT+;
```



```
fragment OPT_FRAC: ('.' DIGIT*)?;
fragment OPT_EXP: ([Ee] [+ -]? DIGITS)?;
```

10. Một chuỗi thời gian ngày/tháng/năm với ngày có thể 1 hoặc 2 kí tự số, tháng thì luôn là 2 kí tự số, năm thì có 4 kí tự số

Kết quả:

```
DATE: [0-9] [0-9]? '/' [0-9] [0-9] '/' [0-9] [0-9] [0-9]
```

11. String Literals (BTL HK232 ZCode): Một chuỗi ký tự bao gồm 0 hoặc nhiều ký tự được bao quanh bởi dấu ngoặc kép (""). Sử dụng chuỗi thoát (được liệt kê bên dưới) để biểu thị các ký tự đặc biệt trong một chuỗi. Hãy nhớ rằng dấu ngoặc kép không phải là một phần của chuỗi. Nó là một chuỗi biên dịch. -lỗi thời gian để một dòng mới hoặc ký tự EOF xuất hiện sau phần mở đầu ("") và trước phần khớp đóng (""). Tất cả các chuỗi thoát được hỗ trợ

```
\b backspace
\f form feed
\r carriage return
\n newline
\t horizontal tab
\' single quote
\\ backslash
```

Đối với dấu ngoặc kép ("") bên trong chuỗi, dấu gạch chéo ngược phải được viết trước chuỗi đó: \".

```
STRING_LIT: """ STR_CHAR* """ {self.text = self.text[1:-1] };
fragment STR_CHAR: ~[\n\\"] | ESC_SEQ;
fragment ESC_SEQ: '\\' [bfrnt'\\'] | '\\''';
fragment ESC_ILLEGAL: '\\' ~[bfrnt'\\'];

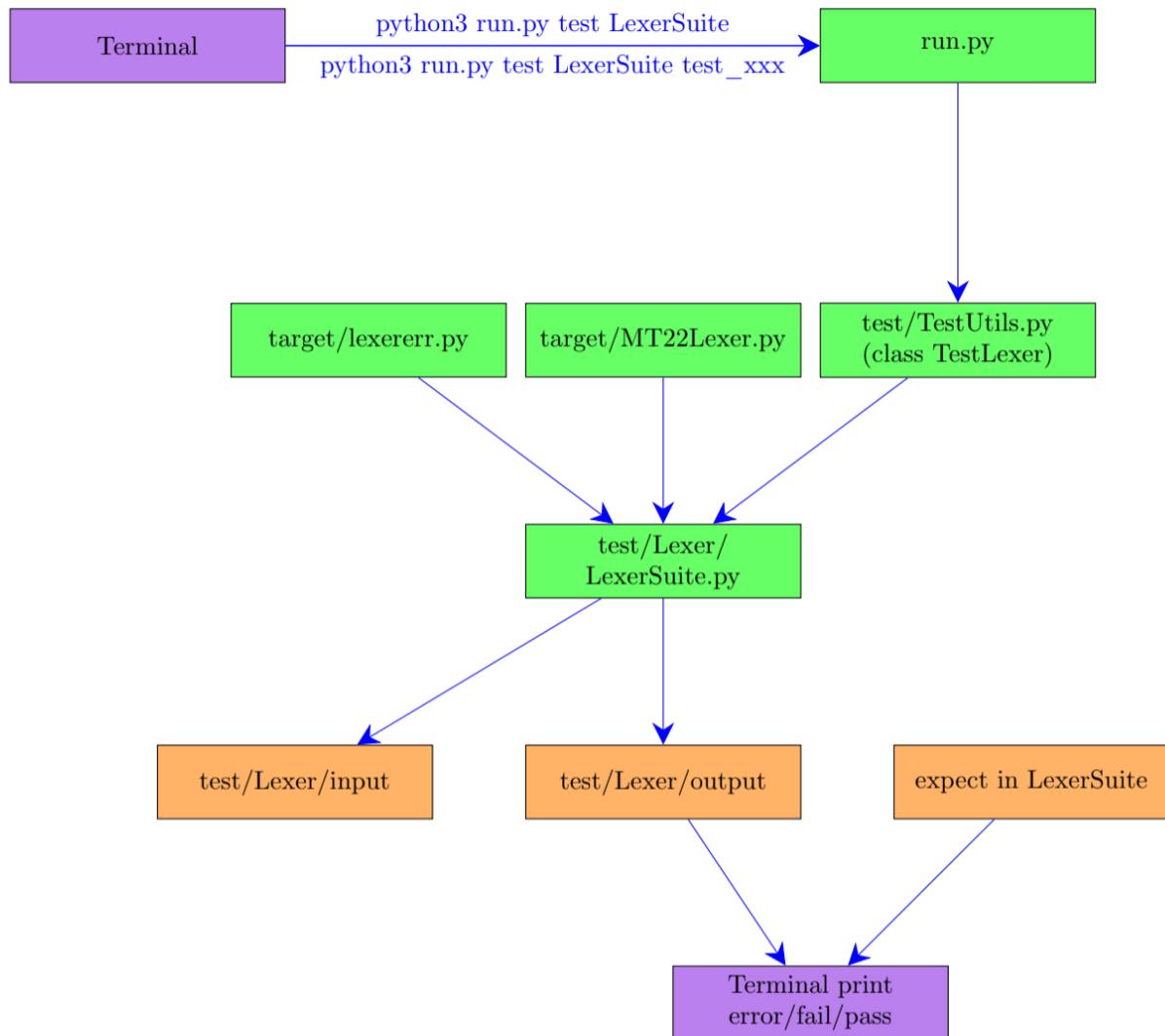
// TODO ERROR
UNCLOSE_STRING: """ STR_CHAR* ('\\r\\n' | '\\n' | EOF) {
    if(len(self.text) >= 2 and self.text[-1] == '\\n' and self.text[-2] == '\\r'):
        raise UncloseString(self.text[1:-2])
    elif (self.text[-1] == '\\n'):
        raise UncloseString(self.text[1:-1])
    else:
        raise UncloseString(self.text[1:])
};

ILLEGAL_ESCAPE: """ STR_CHAR* ESC_ILLEGAL {
    raise IllegalEscape(self.text[1:])
};
```



### 3 Cách run và test BTL Lexer

#### 3.1 Cách hoạt động của lệnh python3/python run.py test LexerSuite



#### 3.2 Test Case file test/Lexer/LexerSuite.py

```
def test_1(self):
    self.assertTrue(TestLexer.test(input="Votien10cham", expect="Votien10cham,<EOF>", num=101))
• input đầu vào của biểu thức chính quy
• expect kết quả chính xác biểu thức chính quy bắt được
• num tên file kết quả output lưu vào trong thư mục test/Lexer/output
• test/Lexer/output/101.txt đầu ra biểu thức chính quy của các bạn chạy được
```

Các bạn có thể viết thêm test vào này dựa trên unittest Python <https://docs.python.org/3/library/unittest.html>



### 3.3 Chạy test case

Câu lệnh chạy với một test duy nhất `python3 run.py test LexerSuite [test_case]` với `test_case` trên của hàm chính xác trong class `LexerSuite` của file `test/Lexer/LexerSuite.py`

```
PS Template_BTL_PPL> python3 run.py test LexerSuite test_2
```

```
Tests run 1
```

```
Errors []
```

```
[]
```

```
Test output
```

```
.
```

---

```
Ran 1 test in 0.001s
```

```
OK
```

---

```
-----  
LexerSuite - test_2 - Assignment- PPL - HK242 - VO TIEN
```

```
Vo Tien : https://www.facebook.com/Shiba.Vo.Tien
```

```
Tests run: 1
```

```
1
```

```
.
```

```
Pass full 10.
```

---

Câu lệnh chạy tất cả các test case `python3 run.py test LexerSuite`

```
PS Template_BTL_PPL> python3 run.py test LexerSuite
```

---

```
-----  
LexerSuite - Assignment- PPL - HK242 - VO TIEN
```

```
Vo Tien : https://www.facebook.com/Shiba.Vo.Tien
```

```
Tests run: 5
```

```
12345
```

```
E.EFF
```

---

```
Pass      : 20.00 %
```

```
Errors    : 1, 3
```

```
Failures : 4, 5
```



## 4 Bài tập phần Lexer

1. Mô tả chuỗi có độ dài là số lẻ
2. Mô tả số thực trong C++.
3. Mô tả chuỗi có nhiều nhất 4 chữ a.
4. Mô tả chuỗi có ít nhất 4 chữ a.
5. Mô tả chuỗi mà a và b không nằm kề nhau.
6. Mô tả chuỗi mà ký tự a phải nằm giữa 2 ký tự b, giữa a và b không có kí tự nào khác.
7. Mô tả chuỗi a và b nằm xen kẽ nhau, ở giữa 2 a là 1 b, giữa 2 b là 1 a, giữa a và b có thể có những kí tự khác.
8. Mô tả chuỗi địa chỉ IPv4(A.B.C.D, với A,B,C,D trong đoạn [0;255].)
9. Mô tả mã màu hexa(VD: #FFFFFF).
10. Mô tả số hệ 8 trong C++.
11. Mô tả số tự nhiên gồm 1 2 3, sao cho 2 chữ số liên tiếp không cách nhau 1 đơn vị
12. Mô tả một chuỗi tương tự thẻ div trong HTML, có thể có cả thẻ đóng - mở, hoặc chỉ có thẻ mở, nội dung giữa 2 thẻ là chuỗi bất kì, bên trong thẻ mở là các cặp attribute="value".
13. Mô tả tên hàm trong C++ theo Camel Case.
14. Mô tả tên hằng số trong C++ theo Snake Case.
15. Mô tả tên hàm trong python.
16. Mô tả địa chỉ email(bắt đầu bằng các ký tự chữ, số, dấu chấm hoặc dấu gạch dưới; kế tiếp là ký tự @; tên miền chỉ chứa chữ cái và có tối đa một dấu chấm.)
17. Mô tả mật khẩu mạnh(dài ít nhất 4 ký tự ;chứa ít nhất 1 chữ cái viết hoa, 1 chữ cái viết thường, 1 chữ số, và 1 ký tự đặc biệt.)



## 5 Lexer (Deadline 23:59 16/1)

### 5.1 Kiến thức

1. Keywords và ID giống nhau làm sao để bắt được Keywords trước, đoạn code sau phần nào đúng

```
//CODE 1  
IF: 'if';  
ID: [a-zA-Z_][a-zA-Z0-9_]*;  
  
//CODE 2  
ID: [a-zA-Z_][a-zA-Z0-9_]*;  
IF: 'if';
```

2. Bắt tên TOKEN nào trước, đoạn code sau phần nào đúng

```
program: BOOL_LIT+ EOF; // CODE 1  
program: (TRUE | FALSE)+ EOF; // CODE 2  
  
TRUE: 'true';  
FALSE: 'false';  
BOOL_LIT: TRUE | FALSE;
```

3. Dưa code Python vào

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

```
// tương đương  
WS: [ \t\f\r\n]+ {self.skip();}
```

Đọc thêm file target/main/MiniGoLexer.py đoạn code python được chèn vào hàm WS\_action

```
def WS_action(self, localctx:RuleContext , actionIndex:int):  
    if actionIndex == 4:  
        self.skip()
```

4. String Literals không kèm ký tự của BTL khác vào xem phần mô tả các ký tự cho phép và không lấy " ở đầu và cuối
5. UNCLOSE\_STRING không được lấy " ở đầu và ký tự xuống hàng ở cuối
6. ILLEGAL\_ESCAPE không được lấy "
7. INT\_LIT yêu cầu chuyển các nhị phân, hex, ... thành kiểu int áp dụng chèn python vào code để xử lí
8. Thay đổi chuỗi trả về dùng self.text trong python
9. Tìm hiểu thêm các hàm python có thể có <https://www.antlr.org/api/Java/org/antlr/v4/runtime/Lexer.html>
10. COMMENT\_Block dùng để quy đồng Parser

### 5.2 BTL

1. file [https://drive.google.com/drive/folders/1g6clytYc7v6D6jkNA\\_Dy8FrHJyj53yjw](https://drive.google.com/drive/folders/1g6clytYc7v6D6jkNA_Dy8FrHJyj53yjw)
2. Keywords 3.3.2 pdf
3. Operators 3.3.3 pdf
4. Separators 3.3.4 pdf
5. Identifiers 3.3.1 pdf
6. Literals 3.3.5 pdf



7. skip 3.1 and 3.2 pdf
8. ERROR pdf BTL1 + lexererr.py
9. Nộp Bài <https://discord.com/channels/1322498638114717706/1328505693409644695>

