VÕ 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 1 - LEXER

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

TP. HỒ CHÍ MINH, THÁNG 1/2025

Mục lục

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 thõa 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 ($\{\epsilon, A, AA, AAA, \dots\}$).

- **Ví dụ:** `'a'*` có thể khớp `''` (chuỗi rỗng), `'a'`, `'aa'`, `'aaa'`, ...

7. `A+`
   - **Ý nghĩa:** `A` lặp lại 1 hoặc nhiều lần ($\{A, AA, AAA, \dots\}$).
   - **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-z0-9]`
    - **Ý nghĩa:** Chọn một kí tự trong khoảng `a-z` hoặc `0-9`.
    - **Ví dụ:** `[a-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. **. (dấu 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 khớp 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

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 đô 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-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+;
```

Python
COMMENTS: '##'~[\n]* -> skip
Cách hoạt động:
là ký hiệu bắt đầu một comment trong Python (theo định nghĩa này).
~[\n]*: Sau ##, mọi ký tự (ngoại trừ ký tự xuống dòng \n) sẽ được bỏ qua (không tạo thành token).
-> skip: Quy định bỏ qua đoạn này khi xử lý.

C++
COMMENTS: (('//'~[\n]*) | ('/*'.*? '*/')) -> skip;
Cách hoạt động:
(...): Chọn một trong các kiểu comment sau:
'//'~[\n]*: Dùng // để bắt đầu comment dòng. Sau đó, mọi ký tự trên dòng (trừ xuống dòng \n) sẽ được bỏ qua.
'/*'.*?'*/': Dùng /* để bắt đầu và */ để kết thúc comment khối. Bất kỳ nội dung nào giữa hai ký hiệu này sẽ bị bỏ qua.
-> skip: Bỏ qua đoạn comment trong quá trình xử lý token.

6. Identifiers trong PHP
ID: '$'[a-zA-Z_][a-zA-Z0-9_]*;
Cách hoạt động:
ID: Định nghĩa một identifier (tên biến) hợp lệ trong PHP.
$: Bắt buộc identifier trong PHP phải bắt đầu bằng dấu $.
[a-zA-Z_]: Sau $, ký tự đầu tiên phải là một chữ cái (a-z, A-Z) hoặc dấu gạch dưới (_).
[a-zA-Z0-9_]*: Phần còn lại của identifier có thể chứa chữ cái, chữ số (0-9), hoặc dấu gạch dưới (_).
Ví dụ hợp lệ:
$variable
$myVar123
$_hidden
$a
Ví dụ không hợp lệ:
variable // Thiếu $
$123abc // Không được bắt đầu bằng số
$my-var // Dấu gạch ngang không hợp lệ

Identifiers trong C++
ID: [a-zA-Z_][a-zA-Z0-9_]*;
Cách hoạt động:
ID: Định nghĩa một identifier (tên biến, hàm, class,...) hợp lệ trong C++.
[a-zA-Z_]: Tên phải bắt đầu bằng một chữ cái (a-z, A-Z) hoặc dấu gạch dưới (_).
[a-zA-Z0-9_]*: Phần còn lại có thể bao gồm chữ cái, chữ số (0-9), hoặc dấu gạch dưới (_).

DIGIT: Một chữ số bất kỳ từ 0 đến 9.
DIGITS: Một hoặc nhiều chữ số liên tiếp (phần nguyên).
ví dụ hợp lệ:
123
0
456789

fragment OPT_FRAC: ('.' DIGIT*)?;
.: Là dấu chấm thập phân.
DIGIT*: Tùy chọn không có hoặc có nhiều chữ số sau dấu ..
?: Phần thập phân là tùy chọn (có hoặc không).

fragment OPT_EXP: ([Ee] [+-]? DIGITS)?;
[Ee]: Phần mũ bắt đầu bằng e hoặc E.
[+-]?: Tùy chọn dấu + hoặc - sau e/E.
DIGITS: Một hoặc nhiều chữ số (phần mũ là số nguyên).

ví dụ hợp lệ:
123.456 // Có phần thập phân
0. // Dấu chấm không có chữ số sau
123 // Không có phần thập phân

```
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][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
$\backslash b$ backspace
$\backslash f$ form feed
$\backslash r$ carriage return
$\backslash n$ newline
$\backslash t$ horizontal tab
$\backslash'$ single quote
$\backslash\backslash$ backslash
Đối với dấu ngoặc kép (") bên trong chuỗi, dấu gạch chéo ngược phải được viết trước chuỗi đó:$\backslash$".

```
STRING_LIT: '"' STR_CHAR* '"' {self.text = self.text[1:-1] };
fragment STR_CHAR: ~[\n\\"] | ESC_SEQ;
fragment ESC_SEQ: '\\' [bfrnt'\\] | '\'"'; => \b, \f, \r, \n, \t, \', \\, \"
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:])
};
```

Lỗi này xảy ra khi:

Chuỗi mở bằng dấu ", nhưng:
Không có dấu đóng ".
Gặp ký tự xuống dòng (\n) hoặc EOF (kết thúc file) trước khi đóng.

"Hello world // Lỗi: Không có dấu ngoặc kép đóng.
"Hello\n // Lỗi: Gặp xuống dòng trước khi đóng.

Lỗi này xảy ra khi:

Chuỗi chứa một chuỗi thoát không hợp lệ.
Ký tự thoát không hợp lệ:

fragment ESC_ILLEGAL: '\\' ~[bfrnt'"\\];
\\: Bắt đầu bằng gạch chéo ngược.
~[bfrnt'"\\]: Theo sau là một ký tự không hợp lệ, tức không thuộc: b, f, r, n, t, ', ", \\.

"Hello\q" // Lỗi: \q không phải chuỗi thoát hợp lệ.
"Path\z" // Lỗi: \z không hợp lệ.

String Literals trong ZCode

Một chuỗi ký tự (string literal) trong ZCode là:

Một dãy ký tự bất kỳ (bao gồm cả ký tự đặc biệt) được bao quanh bởi dấu ngoặc kép (").
Dấu ngoặc kép chỉ để đánh dấu đầu và cuối chuỗi, không phải là một phần của chuỗi.
Ký tự thoát (escape sequences)
Để biểu diễn ký tự đặc biệt bên trong chuỗi, ta sử dụng chuỗi thoát (escape sequences). Các chuỗi thoát được hỗ trợ trong ZCode gồm:

Escape Sequence	Ký tự đại diện
\b	Backspace
\f	Form feed
\r	Carriage return
\n	Newline (xuống dòng)
\t	Horizontal tab
\'	Single quote (')
\"	Double quote (")
\\	Backslash (\)

Ví dụ hợp lệ:

Chuỗi đơn giản:
"hello"

Chuỗi chứa ký tự đặc biệt:
"hello\nworld" // Xuống dòng
"He said, \"Hi!\"" // Dấu ngoặc kép
"C:\\Program Files\\" // Dấu gạch chéo ngược

Chuỗi trống:
""

Cấu trúc biểu thức trong ZCode
1. Định nghĩa chuỗi hợp lệ
antlr
Copy code
STRING_LIT: "" STR_CHAR* "" {self.text = self.text[1:-1] };
STRING_LIT: Token định nghĩa chuỗi ký tự hợp lệ.
Chuỗi bắt đầu bằng dấu ", chứa các ký tự STR_CHAR (0 hoặc nhiều lần), và kết thúc bằng ".
{self.text = self.text[1:-1]}: Loại bỏ dấu ngoặc kép đầu và cuối để lấy phần nội dung chuỗi thực sự.
Ký tự bên trong chuỗi:
antlr
Copy code
fragment STR_CHAR: ~[\n\"] | ESC_SEQ;
~[\n\"]: Các ký tự bất kỳ không phải:
Xuống dòng (\n),
Gạch chéo ngược (\),
Hoặc dấu ngoặc kép (").
ESC_SEQ: Một chuỗi thoát hợp lệ (đã được định nghĩa bên dưới).
Chuỗi thoát hợp lệ:
antlr
Copy code
fragment ESC_SEQ: '\\' [bfrnt'"\\];
\\: Bắt đầu bằng một gạch chéo ngược.
[bfrnt'"\\]: Theo sau là một trong các ký tự:
b, f, r, n, t: Các ký tự thoát thông thường.
', ", \\: Dấu nháy đơn, nháy kép, hoặc gạch chéo ngược.

2. Lỗi chuỗi không hợp lệ (Errors)
Unclosed String
antlr
Copy code
UNCLOSE_STRING: "" STR_CHAR* ('\r\n' | '\n' | EOF) {
// Phát hiện lỗi và ném ngoại lệ
};
Lỗi này xảy ra khi:

Chuỗi mở bằng dấu ", nhưng:
Không có dấu đóng ".
Gặp ký tự xuống dòng (\n) hoặc EOF (kết thúc file) trước khi đóng.
Ví dụ lỗi:

arduino
Copy code
"Hello world // Lỗi: Không có dấu ngoặc kép đóng.
"Hello\n // Lỗi: Gặp xuống dòng trước khi đóng.
Xử lý lỗi:

Nếu kết thúc bằng ký tự xuống dòng \n, loại bỏ chuỗi xuống dòng và báo lỗi:
python
Copy code
raise UncloseString(self.text[1:-1])
Nếu EOF, báo lỗi với phần chuỗi còn lại:
python
Copy code
raise UncloseString(self.text[1:])
Illegal Escape
antlr
Copy code
ILLEGAL_ESCAPE: "" STR_CHAR* ESC_ILLEGAL {
raise IllegalEscape(self.text[1:])
};
Lỗi này xảy ra khi:

Chuỗi chứa một chuỗi thoát không hợp lệ.
Ký tự thoát không hợp lệ:
antlr
Copy code
fragment ESC_ILLEGAL: '\\' ~[bfrnt'"\\];
\\: Bắt đầu bằng gạch chéo ngược.
~[bfrnt'"\\]: Theo sau là một ký tự không hợp lệ, tức không thuộc:
b, f, r, n, t, ', ", \\.
Ví dụ lỗi:

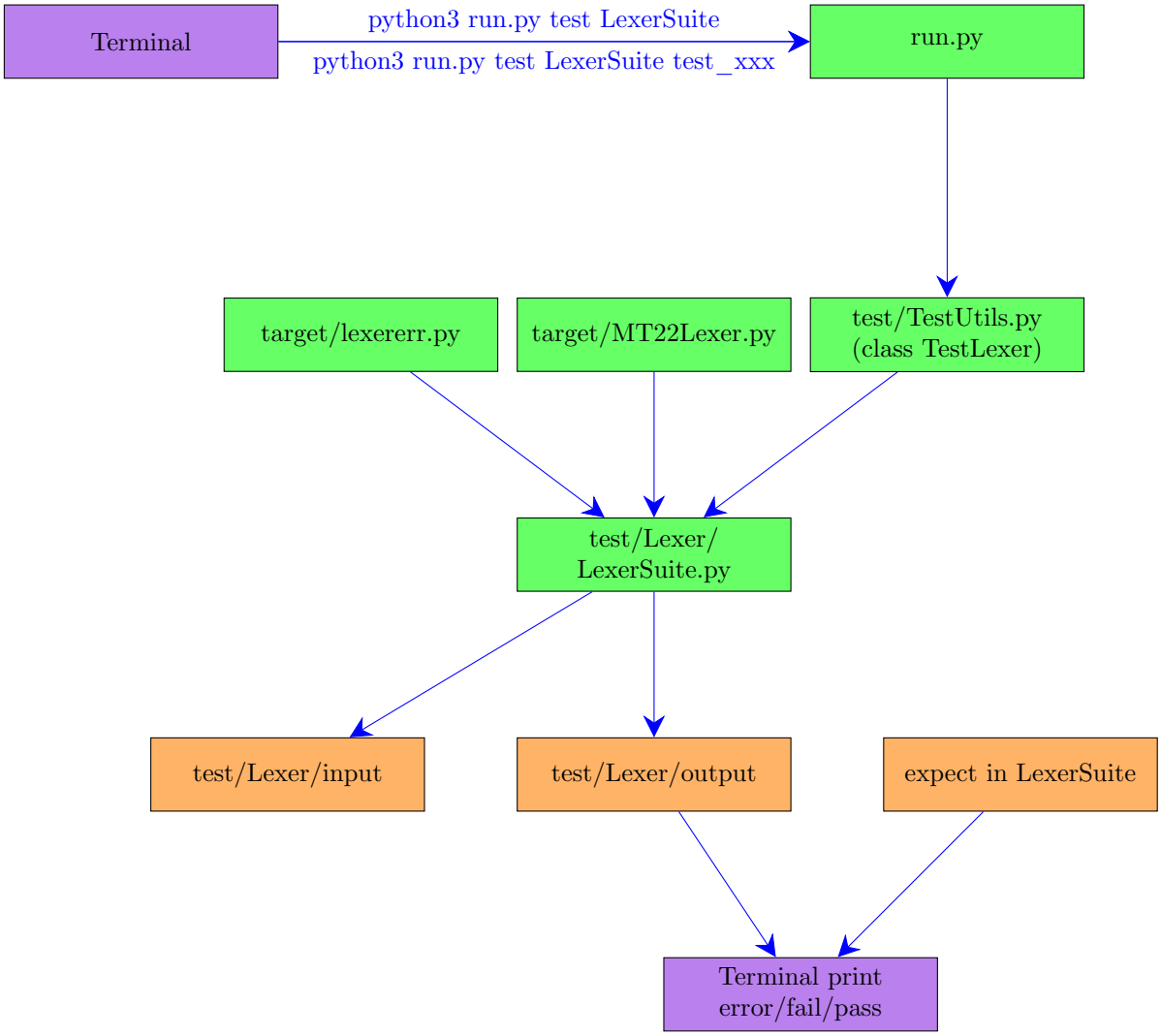
arduino
Copy code
"Hello\q" // Lỗi: \q không phải chuỗi thoát hợp lệ.
"Path\z" // Lỗi: \z không hợp lệ.
Tóm tắt các lỗi:
Unclosed String:

Chuỗi không có dấu đóng ".
Gặp ký tự xuống dòng (\n) hoặc EOF trước khi đóng.
Illegal Escape:

Chuỗi chứa ký tự thoát không hợp lệ (ví dụ: \q, \z, ...).
Tóm tắt quy tắc chính
Chuỗi phải nằm trong cặp dấu ngoặc kép (").
Các ký tự đặc biệt phải sử dụng chuỗi thoát hợp lệ.
Cần thận với chuỗi bị thiếu dấu đóng hoặc chứa chuỗi thoát không hợp lệ.

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 ouput 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 cùa 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 8 ký tự ;chứa ít nhất 1 chữ cái viết hoa, 1 chữ cái viết thường, 1 chữ số, và 1 ký tự đặc biệt.)