ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP 1

SIMPLECODE

SCANNER & PARSER

Môn học	:	Nguyên lý các ngôn ngữ lập trình
GVHD	:	Phạm Trọng Nghĩa
LỚP	:	Cử nhân Tài năng – Khóa 2015
NGƯỜI THỰC HIỆN	:	**1512002 – Lê Dương Tuấn Anh**

Tp. Hồ Chí Minh - Tháng 4/2019

SIMPLE CODE – BÀI TẬP 1

SCANNER & PARSER

Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên, ĐHQG-HCM
Tháng 4/2019

MỤC LỤC

1

Tổng quan & Milestone

Thông tin cá nhân

MSSV	Họ tên	Email
1512002	Lê Dương Tuấn Anh	1512002@student.hcmus.edu.vn leduongtuananh97@gmail.com

Công cụ hỗ trợ

- Python 3
- Java Runtime Environment.
- ANTLR3 & ANTLR3 Runtime for Python3

2

Nội dung bài tập

2.1

ANTLR – ANother Tool for Language Recognition

ANTLR là một trình tạo bộ phân tích cú pháp từ một văn phạm cho trước, được sử dụng bởi rất nhiều lĩnh vực khác nhau, chẳng hạn **công cụ tìm kiếm ở trang Twitter dùng ANTLR để xây dựng luật truy vấn, cho 2 tỉ truy vấn mỗi ngày, hoặc dùng chung với SQL Developer IDE bởi Oracle. NetBeans IDE cũng dùng ANTLR để tạo bộ phân tích cú pháp C, v.v…**

ANTLR có khả năng đọc gần như trong thời gian tuyến tính (linear approximate lookahead), có thể phân tích cú pháp & ngữ nghĩa và phân tích cây cú pháp. ANTLR có thể phân tích cú pháp của văn phạm phi ngữ cảnh, là một dạng **LL-Parser (Top-Down Parser).** Từ phiên bản 4 trở đi, **ANTLR là Adaptive LL-Parser (có khả năng xử lí một số trường hợp đệ quy trái, là một nhược điểm của LL-Parser).**

ANTLR cung cấp rất nhiều thư viện trên nhiều ngôn ngữ khác nhau, trong số đó, nổi tiếng nhất là Java & Python Runtime.

2.2
Cài đặt & sử dụng ANTLR Python Runtime

Sau khi cài đặt Python3 & Java Runtime Environment, ta dùng lệnh pip (được cài đặt kèm Python) để cài đặt antlr như sau:

```
pip install antlr4-python3-runtime
```

Lưu ý: Dùng phiên bản mới nhất là 4.

Tải file **java đã được biên dịch của ANTLR4** tại https://www.antlr.org/download/antlr-4.7.2-complete.jar. Đây được dùng để chuyển đổi từ file văn phạm (*.g4) sang các nền tảng đích (ở đây Python). Sau khi thực thi bằng Java file này, chương trình tạo ra một số thư viện Python. Thư viện này chứa các class hoặc hàm cần thiết để xử lí văn phạm.

Lệnh để tạo trình phân tích cú pháp, đích là Python3 (có sẵn trong file run.bat):

```
java -Xmx500M -cp <antlr_file>.jar org.antlr.v4.Tool -Dlanguage=Python3 <grammar_file>.g4
```

Trong đó:

***antlr_file.jar** là file đã được download ở trên.*

***grammar_file.g4** là file chứa văn phạm.*

Sau khi chạy xong lệnh trên, tại thư mục tạo thêm các file chứa tokens, class và hàm tương ứng để có thể hỗ trợ tạo cây cú pháp, phân tích cú pháp và ngữ nghĩa.

2.3
Xây dựng văn phạm từ đề bài

Văn phạm xây dựng cho ANTLR cũng khá tương đồng với đề bài đã cho. Cách xây dựng luật Lexer của ANTLRc có thể xem tại: https://github.com/antlr/antlr4/blob/master/doc/lexer-rules.md.

Một số lưu ý mà cá nhân đã nhận thấy trong quá trình tạo bộ luật cho ANTLR:

- Có một số luật bị đệ quy trái (left-recursive), trong một số trường hợp ANTLR vẫn không thể xử lí được (mặc dù ANTLR4 đã khắc phục được hầu hết). Trong các trường hợp này cần khử đệ quy trái. Một số luật đệ quy trái được ANTLR mô tả rất rõ tại: https://github.com/antlr/antlr4/blob/master/doc/left-recursion.md.

- Một số từ như **type, id** ANTLR đã dùng riêng, vì vậy, trong bài tập đã chuyển thành **DATA_TYPE & IDENTIFIER.**

- Trong bộ luật có định nghĩa: **mặc định các khoảng trắng, tab character '\t', kí tự xuống hàng '\r', '\n', sẽ được bỏ qua** (*ANTLR hỗ trợ từ khóa skip cho việc bỏ qua)*. Tuy nhiên trong một số trường hợp ta vẫn cần giữ nguyên khoảng cách cho một số từ khóa đặc biệt, chẳng hạn **class <space> Program; int <space> a.** Các trường hợp này cần khai báo **<space>+** *(tức ít nhất có 1 khoảng trắng trong các trường hợp này)*.

- Trong một số trường hợp đặc biệt, cá nhân vẫn chưa thể giải quyết hết trường hợp lỗi phân tích nhầm các kí tự trắng, dẫn đến sinh ra lỗi. Việc loại trừ lỗi này cũng khá khó khăn. Việc xây dựng bộ dịch lỗi sẽ nói rõ ở phần sau.

Bộ luật có tên là SimpleCode, lưu tại SimpleCode.g4.

2.4 Cách sử dụng & mô tả

Chương trình chạy với cú pháp sau:

`python baitap1.py <type> <input_file>`

Trong đó

type: *0 nếu dùng để phân tích từ vựng và 1 nếu phân tích cú pháp*

input_file: *file văn bản chứa đoạn code được viết bởi ngôn ngữ SimpleCode.*

Trong file python, ta dùng các thư viện chính: SimpleCodeLexer, SimpleCodeListener, SimpleCodeParser với chức năng như sau:

- **SimpleCodeLexer**: Chứa văn phạm mà máy hiểu được từ file SimpleCode.g4. Khi đưa 1 đoạn code nào đó vào thư viện này, ANTLR sẽ cố gắng **ghép các chuỗi con trong đoạn code đã cho với các luật Lexer và tìm Token tương ứng**. **Các chuỗi con được phân tích sẽ ghép với các Lexer Rule, được khai báo bởi các từ bắt đầu bằng chữ in hoa trong file SimpleCode.g4**, chẳng hạn: *CLASS, DIGIT, ALPHA, IDENTIFIER*.

- **SimpleCodeParser**: Các luật phân tích cú pháp được mô tả bởi các từ bắt đầu bằng chữ thường (thường là camelCase). ANTLR sẽ phân tích 1 câu dựa vào token tương ứng và xem nó gắn được với luật nào. Lúc đó ANTLR sẽ có khả năng nhận dạng được câu đó.

- **SimpleCodeListener**: Đây là điểm mới của ANTLR4. Sau khi quá trình phân tích hoàn thành, ta cần đi từ nút gốc (khởi điểm của code, ở đây là luật program) bằng lệnh **`parser.program()`**. ANTLR4 hỗ trợ ta các phương thức để có thể "lắng nghe" ANTLR4 đang phân tích tới nút nào của cây phân tích cú pháp, lỗi nếu có, v.v... Do cuối cùng, ANTLR4 sẽ trả về 1 cây phân tích cú pháp, nên ta không cần in quá trình phân tích tại bước này. Tuy nhiên, với type = 1 (phân tích cú pháp và in lỗi nếu có), ta cần thêm vào một lớp ErrorListener (được cung cấp bởi ANTLR4) để có thể theo dõi các lỗi và xuất nếu có. Có nhiều cách để trình phân tích cú pháp nhận được, tuy nhiên nhóm chọn cách "thủ công nhất" là thêm trực tiếp vào biến private của parser:

```
parser._listeners = [ MyErrorListener() ]
```

2.5 Quá trình phân tích từ vựng

Sau khi tạo được cây phân tích cú pháp, ta cần duyệt trên cây này để có thể xuất ra dòng, lexer & token tương ứng. Để việc xuất từ trên xuống dưới đúng theo thứ tự xuất hiện lệnh, ta cần in theo thứ tự **left-mid-right.** Việc này được mô tả trong hàm **`flattenTree`**:

```
def flattenTree(parent, lexers):
    if (int(sys.argv[1]) == 1):
        return
    for i in range(parent.getChildCount()):
        child = parent.getChild(i)
        if (not isinstance(child, ErrorNodeImpl)):
            if (isinstance(child, TerminalNodeImpl)) and (child.getText().strip() != ''):
                printOutChildNode(child, lexers)
            else:
                flattenTree(child, lexers)
```

Hàm chỉ in khi đây là nút lá và đây không phải nút lỗi, đồng thời bỏ qua các lexer nếu đó là lexer chuỗi rỗng. Cách thức in ra thông tin của nút theo như đề cho (chỉ in ra token nếu nó là **'IDENTIFIER','INTLITERAL','CHARLITERAL','STRINGLITERAL','BOOLEANLITERAL'**).

```
filterList = ['IDENTIFIER','INTLITERAL','CHARLITERAL','STRINGLITERAL','BOOLEANLITERAL']

def printOutChildNode(child, lexers):
    global filterList, fWrite
    line = child.getSymbol().line
    lexer = child.getText()
    token = lexers.ruleNames[child.getSymbol().type - 1]
    if (token in filterList):
        fWrite.write('{0} {1} {2}\n'.format(line, token, lexer))
    else:
        fWrite.write('{0} {1}\n'.format(line, lexer))
```

2.6 Quá trình phân tích cú pháp

ANTLR4 hỗ trợ khá nhiều loại lỗi khác nhau (https://www.antlr.org/api/Java/org/antlr/v4/runtime/BaseErrorListener.html), đồng thời cho phép người dùng có thể chỉnh sửa cho hợp lí. *Trong bài tập này sẽ chỉ tập trung vào các lỗi cú pháp (syntaxError). Các báo cáo về văn phạm nhập nhằng, không phân tích được với bộ LL-Parser, hoặc ngữ cảnh đặc biệt (ContextSensitivity) sẽ không đề cập đến.*

Các lỗi trong bài tập này có thể xử lý bao gồm:

(Reference: https://www.antlr.org/api/Java/org/antlr/v4/runtime/RecognitionException.html)

- **NoViableAltException**: Lỗi xuất hiện khi parser tìm không ra được phù hợp luật, thường xảy ra khi hoặc tìm thấy được token nhưng không thấy rule, hoặc quá nhiều rule phù hợp để có thể chọn tương ứng.

```
def reportNoViableAlternative(self, recognizer: Parser, e: NoViableAltException, line,
column):
        tokens = recognizer.getTokenStream()
        if tokens is not None:
            if e.startToken.type == Token.EOF:
                input = "<EOF>"
            else:
                input = tokens.getText()
        else:
            input = "<unknown input>"
        if (input.strip() != ''):
            msg = 'No viable alternative at input {0}'.format(
                self.escapeWSAndQuote(input))
            self.printSyntaxError(msg, line, column)
```

- **InputMismatchException**: Lỗi xuất hiện khi ANTLR tìm thấy 1 lexer không tương ứng với 1 token đã định nghĩa sẵn nào. Trong bài tập này, cá nhân cũng đã in ra các token **đáng ra phải xuất hiện theo đúng các rule có sẵn.**

```
def reportInputMismatch(self, recognizer: Parser, e: InputMismatchException, line,
column):
        msg = "Mismatched input {0}. Expected {1}".format(self.getTokenErrorDisplay(
            e.offendingToken), e.getExpectedTokens().toString(recognizer.literalNames,
recognizer.symbolicNames))
        self.printSyntaxError(msg, line, column)
```

- **FailedPredicateException**: Lỗi xuất hiện khi quá trình kiểm tra các luật, ngữ nghĩa của 1 token nào đó bị sai hoặc gây xung đột với một luật khác trong văn phạm, hoặc token có vấn đề trong việc dự đoán ngữ nghĩa của văn phạm hiện tại. Đối với lỗi này, chương trình xuất ra luật bị sai và token/xung đột tương ứng.

```
    def reportFailedPredicate(self, recognizer, e, line, column):
        ruleName = recognizer.ruleNames[recognizer._ctx.getRuleIndex()]
        msg = "Following Rule is error: {0}. Message: {1}".format(
            ruleName, e.message)
        self.printSyntaxError(msg, line, column)
```

Hai hàm sau dùng để format & in lỗi tương ứng ra file (ở đây chỉ quan tâm đến lỗi cú pháp (Syntax Error)

```
    def printSyntaxError(self, msg, line, column):
        fWrite.write(
            '[Syntax Error] Line {0}, column {1}: {2}\n'.format(line, column, msg))

    def escapeWSAndQuote(self, s: str):
        s = s.replace("\n", "\\n")
        s = s.replace("\r", "\\r")
        s = s.replace("\t", "\\t")
        return "'" + s + "'"
```

Đánh giá: Bài tập chưa in được lỗi khi scanner, chỉ in tất cả những lexer & token có thể parse được.

2.7 Một số testcases

Scanner - Testcase 1

Input	Output
// Mundane characters. 'a' 'b' 'c' 'R' 'i' 'n' 'a' 'r' 'd' '6' '0' '3' '5'	2 CHARLITERAL 'a' 2 CHARLITERAL 'b' 2 CHARLITERAL 'c' 3 CHARLITERAL 'R' 3 CHARLITERAL 'i' 3 CHARLITERAL 'n' 3 CHARLITERAL 'a' 3 CHARLITERAL 'r' 3 CHARLITERAL 'd' 4 CHARLITERAL '6' 4 CHARLITERAL '0' 4 CHARLITERAL '3' 4 CHARLITERAL '5'

Scanner - Testcase 2

Input	Output
// Basic hexadecimal literals. 0x0 0x1 0xe43620 0x11 0xbeef 0xF 0xF00 0xB1ad	2 INTLITERAL 0x0 3 INTLITERAL 0x1 4 INTLITERAL 0xe43620 5 INTLITERAL 0x11 6 INTLITERAL 0xbeef 7 INTLITERAL 0xF 8 INTLITERAL 0xF00 9 INTLITERAL 0xB1ad

Scanner - Testcase 3

Input	Output
// Some valid identifiers of various sorts. abcdefg Rinard martin_rinard six_dot_035 _foo_	2 IDENTIFIER abcdefg 3 IDENTIFIER Rinard 4 IDENTIFIER martin_rinard 5 IDENTIFIER six_dot_035 6 IDENTIFIER _foo_

Scanner - Testcase 4

Input	Output
// Some perfectly normal mundane numbers. 0 1 -1 259 17 43 -620	2 INTLITERAL 0 3 INTLITERAL 1 4 - 4 INTLITERAL 1 5 INTLITERAL 259 6 INTLITERAL 17 7 INTLITERAL 43 7 - 7 INTLITERAL 620

Scanner - Testcase 5

Input	Output
// Some operators. + - * < <= != &&	2 + 2 - 2 * 2 < 2 <= 2 != 2 &&

Scanner - Testcase 6

Input	Output
// ++ is two tokens, so these two lines are equivalent a++ a+ +	2 IDENTIFIER a 2 + 2 + 3 IDENTIFIER a 3 + 3 +

Scanner - Testcase 7

Input	Output
// Simple strings. "A string walks into a bar and orders a beer." "The bartender looks at him and says, \"we don\'t serve strings here.\"" "The string walks out to the street, and sits on the curb, dejected."	2 STRINGLITERAL "A string walks into a bar and orders a beer." 3 STRINGLITERAL "The bartender looks at him and says, \"we don\'t serve strings here.\"" 4 STRINGLITERAL "The string walks out to the street, and sits on the curb, dejected."

"Then he has an idea: he ties himself into a bow, and loosens up his" "ends, making them up into nice tassels." "His confidence restored, he walks back into the bar, sits down, and orders" "another beer." "The bartender looks at him suspiciously: he looks a bit like the string" "that had just walked in. \"Hey,\" he says, \"aren\'t you a string?\"" "\"Nope,\" says the string. \"I\'m a frayed knot.\""	5 STRINGLITERAL "Then he has an idea: he ties himself into a bow, and loosens up his" 6 STRINGLITERAL "ends, making them up into nice tassels." 7 STRINGLITERAL "His confidence restored, he walks back into the bar, sits down, and orders" 8 STRINGLITERAL "another beer." 9 STRINGLITERAL "The bartender looks at him suspiciously: he looks a bit like the string" 10 STRINGLITERAL "that had just walked in. \"Hey,\" he says, \"aren\'t you a string?\"" 11 STRINGLITERAL "\"Nope,\" says the string. \"I\'m a frayed knot.\""

Scanner - Testcase 8

Input	Output

// Decaf keywords
boolean
callout
class
else
false
if
int
return
true
void
for
forpar
break
continue
	2 boolean
	3 callout
	4 class
	5 else
	6 BOOLEANLITERAL false
	7 if
	8 int
	9 return
	10 BOOLEANLITERAL true
	11 void
	12 for
	13 IDENTIFIER forpar
	14 break
	15 continue

Scanner - Testcase 9

Input	Output

// Decaf keywords in uppercase. This should be identifiers.
BOOLEAN
CALLOUT
CLASS
ELSE
	2 IDENTIFIER BOOLEAN
	3 IDENTIFIER CALLOUT
	4 IDENTIFIER CLASS
	5 IDENTIFIER ELSE
	6 IDENTIFIER FALSE
	7 IDENTIFIER IF

FALSE IF INT RETURN TRUE VOID FOR FORPAR BREAK CONTINUE	8 IDENTIFIER INT 9 IDENTIFIER RETURN 10 IDENTIFIER TRUE 11 IDENTIFIER VOID 12 IDENTIFIER FOR 13 IDENTIFIER FORPAR 14 IDENTIFIER BREAK 15 IDENTIFIER CONTINUE

Scanner - Testcase 10

Input	Output
// Decaf keywords stuck together. This should be one big identifier. booleancalloutclasselsefalsei…	2 IDENTIFIER booleancalloutclassels…

Scanner - Testcase 11

Input	Output

// Random tokens
{-123-
a35,id3a,+*;}[||===!=()&&]<><=>==
a[24]='7'; n!=if;
false,-if;true32;
forpar
	2 {
	2 -
	2 INTLITERAL 123
	2 -
	2 IDENTIFIER a35
	2 ,
	2 IDENTIFIER id3a
	2 ,
	2 +
	2 *
	2 ;
	2 }
	2 [
	2 ||
	2 ==
	2 =
	2 !=
	2 (
	2)
	2 &&
	2]
	2 <
	2 >
	2 <=

	2 >= 2 = 3 IDENTIFIER a 3 [3 INTLITERAL 24 3] 3 = 3 CHARLITERAL '7' 3 ; 3 IDENTIFIER n 3 != 3 if 3 ; 4 BOOLEANLITERAL false 4 , 4 - 4 if 4 ; 4 IDENTIFIER true32 4 ; 5 IDENTIFIER forpar

Scanner - Testcase 12

Input	Output
// White-space characters. This should produce several identifiers. foo bar baz quux meep peem whaahboom doom gloom loom weave	2 IDENTIFIER foo 2 IDENTIFIER bar 3 IDENTIFIER baz 3 IDENTIFIER quux 4 IDENTIFIER meep 4 IDENTIFIER peem 5 IDENTIFIER whaah 5 IDENTIFIER boom 6 IDENTIFIER doom 10 IDENTIFIER gloom 10 IDENTIFIER loom 10 IDENTIFIER weave

Parser - Testcase 1

Input	Output

`class Program {`
`  void main() {`
`} // missing closing brace`
	`[Syntax Error] Line 4, column 0: extraneous input '<EOF>' expecting {'}', ' '}`

Parser - Testcase 2

Input	Output

`class Program {`
`  int i[];// missing array size`
`}`
	`[Syntax Error] Line 2, column 8: missing INTLITERAL at ']'`

Parser - Testcase 3

Input	Output

`class Program {`
`  int a[2+3];// bad array decl`
`}`
	`[Syntax Error] Line 2, column 9: Mismatched input '+'. Expected ']'`

Parser - Testcase 4

Input	Output

`class Program {`
`  main() {// no return type`
`  }`
`}`
	`[Syntax Error] Line 2, column 2: Mismatched input 'main'. Expected {'}', DATA_TYPE, ' ', 'void'}`

Parser - Testcase 5

Input	Output

`class Program {`
`  void main() {`
`  callout(5);// first arg must be a string`
`  }`
`}`
	`[Syntax Error] Line 3, column 12: Mismatched input '5'. Expected {' ', STRINGLITERAL}`

Parser - Testcase 6

Input	Output

`class Program {`
`  int i[10];`
`}`	

Parser - Testcase 7

Input	Output

`class Program {`
`  int abs(int a) {`
`  int b;`
`  if (a < 0) {`
`  b = -a;`
`  }`
`  else {`
`  b = a;`
`  }`
 	
`  return b;`
`  }`
 	
`  int main() {`
`  return abs(-2);`
`  }`
`}`	

Parser - Testcase 8

Input	Output

`class Program {`
`  void main() {`
`  int a;`
`  int a;  // semanitcally bad, but gramatically ok`
`  }`
`}`	