

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO BÀI TẬP

SIMPLECODE

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 5/2019

SIMPLE CODE



Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên, ĐHQG-HCM
Tháng 5/2019

MỤC LỤC

1	Tổng quan & Milestone	5
	Thông tin cá nhân	5
	Công cụ hỗ trợ	5
2	Nội dung bài tập	6
2.1	ANTLR – ANother Tool for Language Recognition	6
2.2	Cài đặt & sử dụng ANTLR Python Runtime	7
2.3	Xây dựng văn phạm từ đề bài	8
2.4	Cách sử dụng & mô tả	9
2.5	Quá trình phân tích từ vựng	10
2.6	Quá trình phân tích cú pháp	11
2.7	Quá trình phân tích Ngữ nghĩa tĩnh	13
2.8	Một số testcases	16
	Scanner - Testcase 1	16
	Scanner - Testcase 2	16
	Scanner - Testcase 3	16
	Scanner - Testcase 4	17
	Scanner - Testcase 5	17
	Scanner - Testcase 6	17
	Scanner - Testcase 7	17
	Scanner - Testcase 8	18
	Scanner - Testcase 9	18
	Scanner - Testcase 10	19
	Scanner - Testcase 11	19
	Scanner - Testcase 12	20
	Parser - Testcase 1	21
	Parser - Testcase 2	21

Parser - Testcase 3.....	21
Parser - Testcase 4.....	21
Parser - Testcase 5.....	21
Parser - Testcase 6.....	21
Parser - Testcase 7.....	22
Parser - Testcase 8.....	22
Semantic - Testcase 1.....	23
Semantic - Testcase 2.....	23
Semantic - Testcase 3.....	23
Semantic - Testcase 4.....	23
Semantic - Testcase 5.....	24
Semantic - Testcase 6.....	24
Semantic - Testcase 7.....	24
Semantic - Testcase 8.....	25
Semantic - Testcase 9.....	25
Semantic - Testcase 10	25
Semantic - Testcase 11	26
Semantic - Testcase 12	26
Semantic - Testcase 13	26
Semantic - Testcase 14	26
Semantic - Testcase 15	27
Semantic - Testcase 16	27
Semantic - Testcase 17	27
Semantic - Testcase 18	28
Semantic - Testcase 19	28
Semantic - Testcase 20	28

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.
- ANTLR4 & ANTLR4 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 lý 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 lý 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 đệ quy trái (left-recursive), trong một số trường hợp ANTLR vẫn không thể xử lý 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 nhằm 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 escapeWSEscapeAndQuote(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 Quá trình phân tích Ngữ nghĩa tĩnh

ANTLR4 cũng có cung cấp sẵn các class để hỗ trợ trong việc kiểm tra ngữ nghĩa tĩnh của ngôn ngữ lập trình. Để sinh ra các lớp này từ văn phạm, ta cần thêm tham số **-visitor** khi biên dịch văn phạm:

```
java -Xmx500M -cp antlr-4.7.2-complete.jar
org.antlr.v4.Tool -Dlanguage=Python3 -visitor SimpleCode.g4
```

Lúc này sẽ sinh ra thêm file **SimpleCodeVisitor.py**. Trong file này đã thiết kế sẵn class Visitor để có thể duyệt trên cây đã tạo ở bước trước, trong đó có các hàm quan trọng:

- **visitChildren**: Lần lượt duyệt các nút con của nút hiện tại, theo thứ tự từ trái sang phải.
- **getChildCount**: Lấy số lượng nút con tương ứng.
- **visit**: Duyệt trực tiếp nút con.
- **getText**: Lấy ra lexem.

Từ đó, trong chương trình chính, ta sẽ thiết kế class **MyVisitor** kế thừa trực tiếp từ class **SimpleCodeVisitor**. Trong class này đã có các hàm **visitXXX**, trong đó **XXX** là các *non-terminal* trong văn phạm đã thiết kế. Vì vậy việc kế thừa đồng nghĩa ta sẽ viết các chương trình con để kiểm tra ngữ nghĩa tĩnh. **Ngoài ra, ta có thể truy xuất trực tiếp terminal-node từ trong nút non-terminal.**

Trong class này, ta khai báo thêm các hash table để lưu lại các thông tin cần thiết:

```
self.table = {}
self.funcTable = {}
self.arrayTable = {}
```

Trong đó:

- **table**: Chứa biến & kiểu dữ liệu tương ứng (kể cả void).
- **funcTable**: Chứa biến kiểu void, và các tham số truyền vào (lưu dưới dạng mảng).
- **arrayTable**: Chứa biến kiểu mảng, và độ dài mảng.

Ví dụ, xét nút **field_decl** (**field declaration**):

```
def visitField_decl(self, ctx: SimpleCodeParser.Field_declContext):
    dataType = ctx.DATA_TYPE().getText() // lấy tên kiểu dữ liệu
    for i in range(ctx.getChildCount()):
        child = ctx.getChild(i)
        if (not isinstance(child, TerminalNodeImpl)) and (child.getText().strip()
!= ' '):
            id = self.visit(child).getText() // Kiểm tra xem đã được khai báo chưa
            if id in self.table: // Nếu đã có
                self.printError(ctx, ERROR_DUPLICATE_VAR_DEFINE, id,
self.table[id]) // In ra bị lặp biến
            else:
```

```
self.table[id] = dataType // Cập nhật biến
```

Xét nếu thấy đang khai báo mảng:

```
def visitArray_decl(self, ctx:SimpleCodeParser.Array_declContext):
    if isinstance(ctx.INTLITERAL(), ErrorNodeImpl): // Không có số phần tử
        self.printError(ctx, ERROR_ARRAY_LENGTH_NOT_DEFINED,
ctx.IDENTIFIER().getText())
    elif ctx.INTLITERAL().getText() == '0': // Mảng có 0 phần tử
        self.printError(ctx, ERROR_ARRAY_ZERO_LENGTH, ctx.IDENTIFIER().getText())
    else:
        # Everything ok
        self.arrayTable[ctx.IDENTIFIER().getText()] =
int(ctx.INTLITERAL().getText()) // Lưu lại số lượng
    return ctx.IDENTIFIER() // Trả về tên biến mảng
```

Để hỗ trợ việc in lỗi, ta khai báo các mẫu lỗi chung:

```
ERROR_DUPLICATE_VAR_DEFINE = 'Variable [{0}] is already declared with type [{1}]\n'
ERROR_VAR_NOT_DEFINED = 'Variable [{0}] has not been defined\n'
ERROR_VAR_EXECUTE_AS_FUNCTION = 'Variable [{0}] is not a function\n'
ERROR_DATA_TYPE_NOT_COMPATIBLE_OPERATOR = 'Type [{0}] is not compatible with operator [{1}]\n'
ERROR_DATA_TYPE_DIFFERENT = "Right hand side should be type [{0}], got type [{1}]\n"
MAIN_METHOD_IS_NOT_AVAILABLE = "There is no 'main' method is declared\n"
ERROR_ARRAY_ZERO_LENGTH = "Array [{0}] is declared with zero-size\n"
ERROR_ARRAY_LENGTH_NOT_DEFINED = "Array [{0}] is declared with wrong size\n"
FUNCTION_NOT_DEFINED = "Function [{0}] is not defined!\n"
FUNCTION_PARAMS_MISMATCHED = "Function [{0}] requires list parameters as {1}, which is actually as {2}\n"
ERROR_VOID_RETURN_DATA_TYPE = "Void [{0}] can't return value, got type [{1}]\n"
ERROR_FUNCTION_RETURN_WRONG_TYPE = "Function [{0}] expected return type as [{1}], got [{2}]\n"
ERROR_FUNCTION_RETURN_EMPTY = "Function [{0}] expected return type as [{1}], got empty\n"
ERROR_ARRAY_INDEX_TYPE = "Array [{0}] index type must be [int], got [{1}]\n"
ERROR_ARRAY_INDEX_UNDEFINED = "Array [{0}] must have index, but got empty\n"
ERROR_RHS_IS_ARRAY = "Right hand side of assignment must not be {0}\n"
ERROR_ARRAY_INDEX_OUT_OF_RANGE = "Array [{0}] index is out of range, must be less than [{1}], actually [{2}]\n"
ERROR_VAR_IS_NOT_ARRAY = "[{0}] is not an array to iterate\n"
ERROR_IF_CONDITION_MUST_BE_BOOLEAN = 'Condition in if-else must be boolean, got [{0}]\n'
ERROR_ITERATOR_LOOP_NOT_INT = 'Iterator in Loop must be int-variable, got [{0}]\n'
```

```
ERROR_MISSING_IDENTIFIER_FOR = 'For-loop is missing iterator\n'  
ERROR_PARAMS_FOR = 'For-loop got {0} param(s), expected 3\n'  
ERROR_INIT_VALUE_FOR = 'For-loop must have initialized value int-type, got [{0}]\n'  
ERROR_FINAL_VALUE_FOR = 'For-loop must have final value int-type, got [{0}]\n'  
ERROR_OPERATOR_MISMATCHED = "Type [{0}] and type [{1}] couldn't operate with [{2}]\n"  
ERROR_NOT_OPERATOR_POSTFIX = '[!] operator requires boolean variable, got [{0}]\n'
```

Để chạy chương trình:

python run.py <tên file input>

Đánh giá: Đôi lúc còn nhận dạng sai lỗi.

2.8 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"	5 STRINGLITERAL "Then he has an idea: he ties himself into a bow, and loosens up his"
"ends, making them up into nice tassels."	6 STRINGLITERAL "ends, making them up into nice tassels."
"His confidence restored, he walks back into the bar, sits down, and orders"	7 STRINGLITERAL "His confidence restored, he walks back into the bar, sits down, and orders"
"another beer."	8 STRINGLITERAL "another beer."
"The bartender looks at him suspiciously: he looks a bit like the string"	9 STRINGLITERAL "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?\""	10 STRINGLITERAL "that had just walked in. \"Hey,\" he says, \"aren't you a string?\""
"\"Nope,\" says the string. \"I'm a frayed knot.\""	11 STRINGLITERAL "\"Nope,\" says the string. \"I'm a frayed knot.\""

Scanner - Testcase 8

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

Scanner - Testcase 9

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

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

	<pre> 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 </pre>
--	--

Scanner - Testcase 12

Input	Output
<pre> // White-space characters. This should produce several identifiers. foo bar baz quux meep peem whaahboom doom gloom loom weave </pre>	<pre> 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 </pre>

Parser - Testcase 1

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

Parser - Testcase 2

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

Parser - Testcase 3

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

Parser - Testcase 4

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

Parser - Testcase 5

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

Parser - Testcase 6

Input	Output
<pre>class Program { int i[10]; }</pre>	

Parser - Testcase 7

Input	Output
<pre> class Program { int abs(int a) { int b; if (a < 0) { b = -a; } else { b = a; } return b; } int main() { return abs(-2); } } </pre>	

Parser - Testcase 8

Input	Output
<pre> class Program { void main() { int a; int a; // semantically bad, but gramatically ok } } </pre>	

Semantic - Testcase 1

Input	Output
<pre>class Program { void main() { int x; boolean x; // identifier declared twice } }</pre>	<p>Error at line 4, column 4 : Variable [x] is already declared with type [int]</p>

Semantic - Testcase 2

Input	Output
<pre>class Program { void main() { a = 34; // identifier used before being declared } }</pre>	<p>Error at line 3, column 4 : Variable [a] has not been defined</p>

Semantic - Testcase 3

Input	Output
<pre>// No main method class Program { void foo() { } void bar() { } }</pre>	<p>Error at line 0, column 0 : There is no 'main' method is declared</p>

Semantic - Testcase 4

Input	Output
<pre>class Program { int a[0]; // bad array size void main() { } }</pre>	<p>Error at line 2, column 6 : Array [a] is declared with zero-size</p>

Semantic - Testcase 5

Input	Output
<pre> class Program { int foo(int i, boolean b) { return 4; } void main() { int x; x = foo(34, true, true); // argument mismatch } } </pre>	<p>Error at line 8, column 8 : Function [foo] requires list parameters as ['int', 'boolean'], which is actually as ['int', 'boolean', 'boolean']</p>

Semantic - Testcase 6

Input	Output
<pre> class Program { int foo(int i, boolean b) { return 3; } void main() { int x; x = foo(34, 35); // types don't match signature } } </pre>	<p>Error at line 8, column 8 : Function [foo] requires list parameters as ['int', 'boolean'], which is actually as ['int', 'int']</p>

Semantic - Testcase 7

Input	Output
<pre> class Program { void foo(int i, boolean b) { return 3; // should not return value; } void main() { } } </pre>	<p>Error at line 3, column 4 : Void [foo] can't return value, got type [int]</p>

Semantic - Testcase 8

Input	Output
<pre> class Program { int foo(int i, boolean b) { return true; // return value has wrong type } void main() { int x; x = foo(34, true); } } </pre>	<pre> Error at line 3, column 4 : Function [foo] expected return type as [int], got [boolean] </pre>

Semantic - Testcase 9

Input	Output
<pre> class Program { int a[10]; boolean b; void main() { a[b] = 25; // array index has wrong type } } </pre>	<pre> Error at line 6, column 4 : Array [a] index type must be [int], got [boolean] </pre>

Semantic - Testcase 10

Input	Output
<pre> class Program { int a[10]; void main() { a[1] = a; // bad type, rhs should be an int } } </pre>	<pre> Error at line 5, column 4 : Right hand side of assignment must not be array[int] </pre>

Semantic - Testcase 11

Input	Output
<pre>class Program { int a[10]; void main() { if (a[3]) {} // condition should be a boolean } }</pre>	<p>Error at line 5, column 4 : Condition in if-else must be [boolean], got [int]</p>

Semantic - Testcase 12

Input	Output
<pre>class Program { void main() { for x = false, 10 { //initial condition must be an int } } }</pre>	<p>Error at line 3, column 4 : For-loop must have initialized value int-type, got [boolean]</p>

Semantic - Testcase 13

Input	Output
<pre>class Program { int a[10]; int i; void main() { a[3] = i < 35; // rhs should be an int expression } }</pre>	<p>Error at line 6, column 4 : Right hand side should be type [int], got type [boolean]</p>

Semantic - Testcase 14

Input	Output
<pre>class Program { boolean b;</pre>	<p>Error at line 5, column 8 : Type [boolean] and type [boolean] couldn't operate with [>]</p>

```
void main() {
    b = true > false;
    // operands of > must be
ints
}
}
```

Semantic - Testcase 15

Input	Output
<pre>class Program { boolean b; void main() { b = 5 == true; // types of operands of == must be equal } }</pre>	<p>Error at line 5, column 8 : Type [int] and type [boolean] couldn't operate with [==]</p>

Semantic - Testcase 16

Input	Output
<pre>class Program { boolean b; void main() { b = !5; // operand of ! must be boolean } }</pre>	<p>Error at line 5, column 8 : [!] operator requires boolean variable, got [int]</p>

Semantic - Testcase 17

Input	Output
<pre>class Program { boolean b; void main() { b += true; // lhs and rhs of += must be int } }</pre>	<p>Error at line 5, column 4 : Type [boolean] is not compatible with operator [+=]</p>

Semantic - Testcase 18

Input	Output
<pre> class Program { int b; void main() { b = -true; // operand of ! must be boolean } } </pre>	<p>Error at line 5, column 8 : Variable [c] has not been defined</p>

Semantic - Testcase 19

Input	Output
<pre> class Program { int b; int a[10]; void main() { b += a; // lhs and rhs of += must be int } } </pre>	<p>Error at line 5, column 4 : Right hand side of assignment must not be array[int]</p>

Semantic - Testcase 20

Input	Output
<pre> class Program { int b; int a[10]; void main() { a[10] = 1; // lhs and rhs of += must be int } } </pre>	<p>Error at line 5, column 4 : Array [a] index is out of range, must be non-negative and less than [10], actually [10]</p>