COMP6421 Assignment 1

Current Version: 0.1
Date: Jan. 30, 2012
Author: Yuan Tao
Instructor: Dr. Joey Paquet

Concordia University

Table of Contents

Table of Figures

1. Lexical Conventions

The requirements for assignment 1 please see:
http://newton.cs.concordia.ca/~paquet/wiki/images/7/7c/Assignment1.COMP442.paquet.2012.4.pdf

1.1. Identifiers

According to the regular expressions of the assignment requirements:
id ::= letter alphanum*
num ::= nonzero digit* fraction | 0 fraction
alphanum ::= letter | digit | _
fraction ::= .digit* nonzero | .0 | ε
letter ::= a..z |A..Z
digit ::= 0..9
nonzero ::= 1..9

The following characters are legal as the first character of an identifier, or any subsequent character:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

The following characters are legal as any character in an identifier except the first:
0 1 2 3 4 5 6 7 8 9 _

For example, the following are valid identifiers:

- if
- iff
- a
- m3
- c_8

The following are not identifiers:

- 8
- _file
- 9min

1.2. Keywords

Keywords are reserved identifiers, according to the requirement, the following are reserved words:
if then else while do class integer real read write return And Or Not

The last three words (And Or Not) are reserved for operators.

1.3. Operators

The following tokens are defined as operators:

- ==
- <>
- <
- >
- <=
- >=
- .
- +
- -
- *
- /
- =
- And
- Or
- Not

1.4. Punctuators

Any of the following characters are considered punctuators:

- ;
- ,
- (
-)
- {
- }
- [
-]

1.5. Comments

The same as C programming language, there are two forms of comments:

- /* any sequence of characters */
- // any sequence of characters, except a new line.

1.6. Literals / constants

Literals consist of integer constants and float constants. For example:

- 0
- 0.0
- .0
- 0.01
- 1
- 2.00345
- 12345.67

2. Lexical Errors

Here just talks about the lexical errors which might happen for the lexical analyzer, and their recovery techniques used in the program.

2.1. Newline characters

Only the following formats of a newline are supports by this program.

- LF (Unix, Linux, Mac OS X, FreeBSD …)
- CR+LF (Windows, DOS …)

These characters are handled as space character.

2.2. Invalid character

Invalid character denotes the characters that not defined by the assignment requirements. Like:
~!@#$%^&":'?

The analyzer of this program will treat them as space characters and skip them, and an error message will be stored in the output file. For example:
abc@gmail.com will be analyzed as the following tokens:
abc
gmail
.
Com

2.3. Maximum length of identifier

The maximum length of an identifier is defined as 255 in the program. Any identifier whose length is longer than this value will be treated as an error identifier. The program will NOT cut the exceeding characters.

2.4. Too big integer

Any integer whose value is between -2147483648 and 2147483647 (included) will be analyzed as a valid integer. Otherwise, an error message will be showed.

2.5. Too big float

Any float number whose value between 1.4E-45 and 3.4028235E38 will be a valid float number. Otherwise, an error message will indicate the error float number.

2.6. End of input file

Because there is no special character when the file reader reaches the end of input file, the lexical analyzer will ignore the last line of the input file.

In order to recovery this error, the analyzer automatically adding a '\n' to nextChar() when it knows the file reaches its end.

2.7. Ambiguities

If a token is composed of two or more tokens, these lexical ambiguities will resolved by the longest match rule. For example:
///
It will be analyzed as two tokens, the first one is //, and the other one is /.

12345HelloWorld
It will be analyzed as two tokens, the first one is 12345, and the other one is HelloWorld.

000
According to the regular expressions defined in the assignment requirement, 0 is a valid number. So 000 will be scanned as three tokens: 0, 0 and 0. The same rule is apply to the tokens like "…" (three dots), <<, ++.

2.8. Others

- Unclose comment/brackets

These errors are not lexical errors. They should be discovered by the syntax analyzer.

3. Data Structure of Token

```
public class Token {
    public String file = null;     // name of the file when analyzing this token
    public int line = 0;           // line number
    public int column = 0;         // the column of the first character
    public String token = null;    // lexeme or value
    public int type;               // type of token

    public boolean error = false;  // true means the token is generated by error
recovery technique
}
```

The possible types of the tokens are defined in another source file:

```
    public static final int TOKEN_TYPE_UNKNOWN = 0;
    public static final int TOKEN_TYPE_KEYWORD = 1;
    public static final int TOKEN_TYPE_ID = 2;
    public static final int TOKEN_TYPE_INT = 3;
    public static final int TOKEN_TYPE_FLOAT = 4;
    public static final int TOKEN_TYPE_OPERATOR = 5;
    public static final int TOKEN_TYPE_PUNCTUATOR = 6;
    public static final int TOKEN_TYPE_COMMENT = 7;
```

4. State Machine

Due to build DFA, all the regular expressions are divided into the following set:

- [a-z][A-Z] ([a-z][A-Z] | [0-9] | _)*
- . | .0 | .[0-9] [1-9]*
- 0 | 0.0 | 0.[0-9] [1-9]*
- [1-9][0-9]* | [1-9][0-9]*.0 | [1-9][0-9]*.[0-9] [1-9]*
- <= | <> | <
- = | ==
- > | >=
- / | // | /*
- | */
- ; | , | + | - | (|) | { | } | [|]

Here are DFAs of some complicated regular expressions.

4.1. Identifier

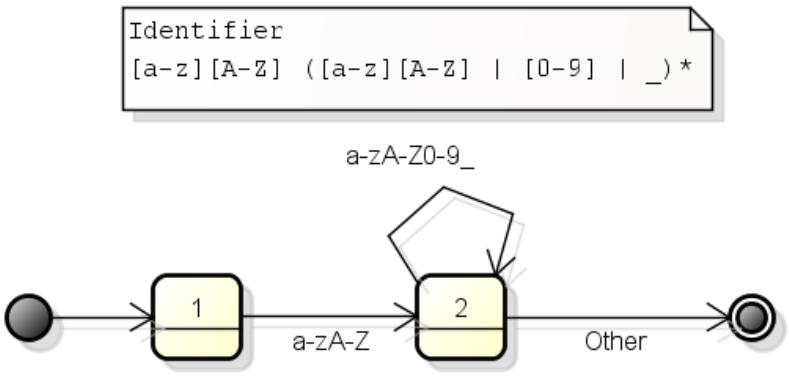

Figure 4-1 Identifier

4.2. Fraction & .

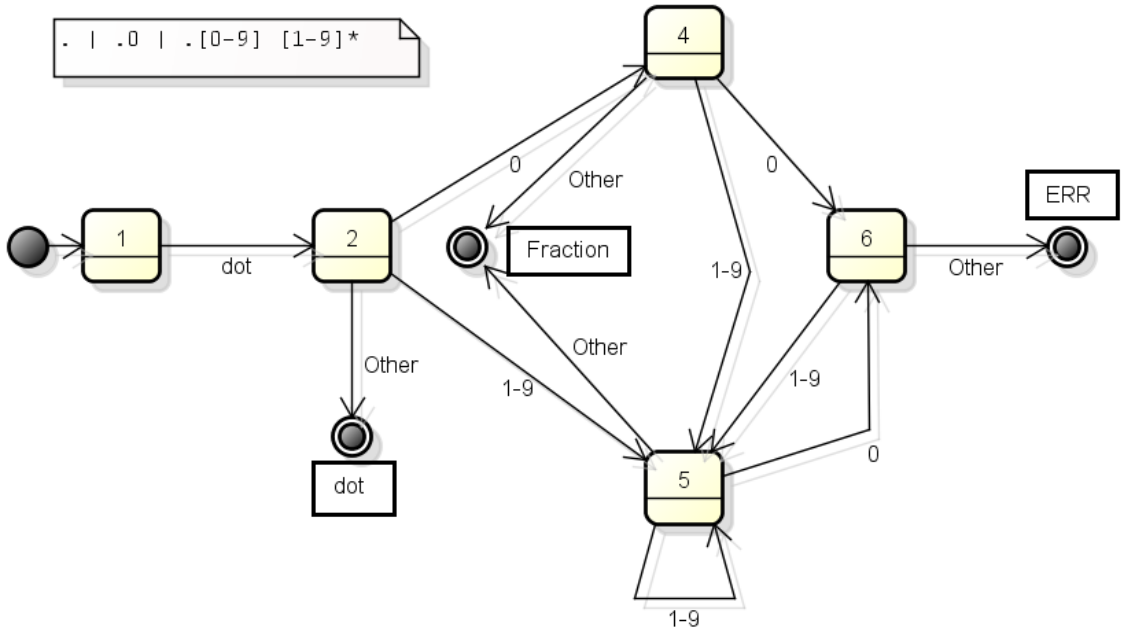

Figure 4-2 Fracture & .

4.3. Fraction & 0

0 | 0.0 | 0.[0-9] [1-9]*

The DFA is nearly same as chapter 4.2, except no dot final state and adding a 0 state.

4.4. Fraction & Integer

[1-9][0-9]* | [1-9][0-9]*.0 | [1-9][0-9]*.[0-9] [1-9]*

Nearly same as chapter 4.3.

5. How to run

1. Install JRE (v5.0 or higher)
2. Extract COMP6421Ass1.zip to $DIR
3. cd $DIR\COMP6421Ass1
4. $ java -jar COMP6421Ass1.jar

Appendix A: Revision History

Version	Date	Author	Remark
V0.1	Jan. 30, 2012	Yuan Tao	Draft