Yan Peng
921759056

September 26, 2021

Assignment 2 Documentation

GitHub Repository

Link for assignment 2:

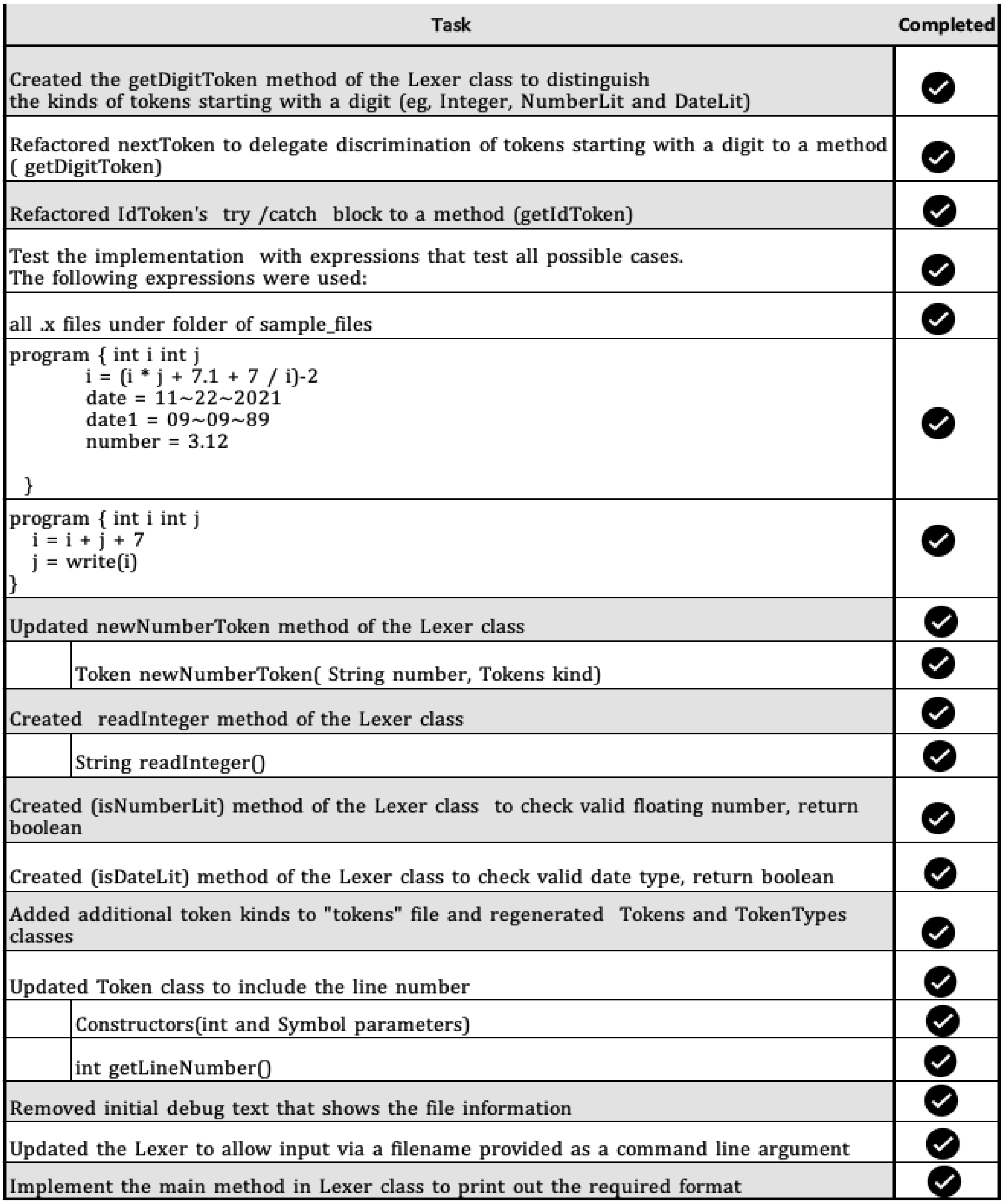
https://github.com/sfsu-csc-413-fall-2021/assignment-2---lexer-yuqiao1205

Project Introduction and Overview

The aim of this project is to understand the function of lexical analysis in the compilation process. The purpose of the lexical analyzer in this assignment is to read the input characters of the source files and convert them into a sequence of tokens, which is the first stage of a compiler.

The Lexer project required me to extend the Lexer class by adding additional tokens and reserved words to perform complete lexical analysis of files with the suffix ".x", meanwhile, to distinguish lexemes starting with a digit token and decide if it is an Integer, NumberLit (any series of numbers followed by a decimal point) or DateLit (one or two number representing the month, followed by "~".) Also, print out the number of lines and token type and include error reporting if invalid tokens are found.

Scope of Work



Task		Completed
Created the getDigitToken method of the Lexer class to distinguish the kinds of tokens starting with a digit (eg, Integer, NumberLit and DateLit)		✓
Refactored nextToken to delegate discrimination of tokens starting with a digit to a method (getDigitToken)		✓
Refactored IdToken's try /catch block to a method (getIdToken)		✓
Test the implementation with expressions that test all possible cases. The following expressions were used:		✓
all .x files under folder of sample_files		✓

program { int i int j
i = (i * j + 7.1 + 7 / i)-2
date = 11~22~2021
date1 = 09~09~89
number = 3.12
}
		✓

program { int i int j
i = i + j + 7
j = write(i)
}
		✓

Updated newNumberToken method of the Lexer class		✓
	Token newNumberToken(String number, Tokens kind)	✓
Created readInteger method of the Lexer class		✓
	String readInteger()	✓
Created (isNumberLit) method of the Lexer class to check valid floating number, return boolean		✓
Created (isDateLit) method of the Lexer class to check valid date type, return boolean		✓
Added additional token kinds to "tokens" file and regenerated Tokens and TokenTypes classes		✓
Updated Token class to include the line number		✓
	Constructors(int and Symbol parameters)	✓
	int getLineNumber()	✓
Removed initial debug text that shows the file information		✓
Updated the Lexer to allow input via a filename provided as a command line argument		✓
Implement the main method in Lexer class to print out the required format		✓

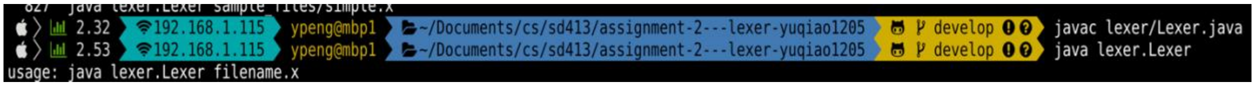
Execution and Development Environment

I used the IntelliJ IDE on my Mac to finish this assignment and tested in both the IDE and shell.

Compilation Result

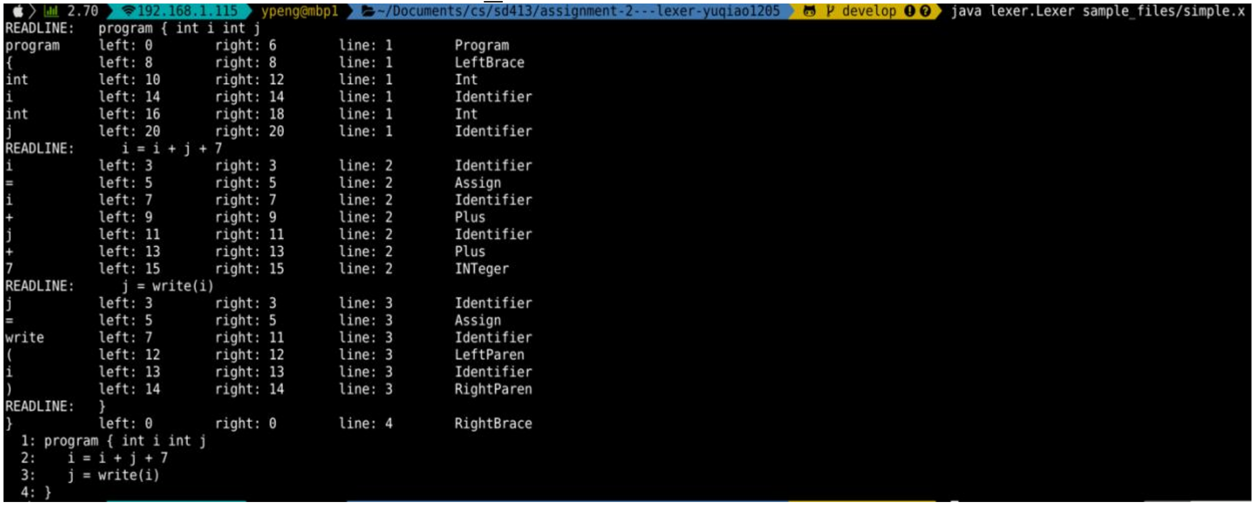
Using the instructions provided in the assignment one specification:

```
> javac lexer/Lexer.java
> java lexer.Lexer
```



```
 827  java lexer.Lexer sample_files/simple.x
 > 2.32 > 192.168.1.115 > ypeng@mbp1 > ~/Documents/cs/sd413/assignment-2---lexer-yuqiao1205 > develop > javac lexer/Lexer.java
 > 2.53 > 192.168.1.115 > ypeng@mbp1 > ~/Documents/cs/sd413/assignment-2---lexer-yuqiao1205 > develop > java lexer.Lexer
usage: java lexer.Lexer filename.x
```

```
> java lexer.Lexer sample_files/simple.x
```



```
 > 2.70 > 192.168.1.115 > ypeng@mbp1 > ~/Documents/cs/sd413/assignment-2---lexer-yuqiao1205 > develop > java lexer.Lexer sample_files/simple.x
READLINE:   program { int i int j
program     left: 0       right: 6      line: 1      Program
{           left: 8       right: 8      line: 1      LeftBrace
int         left: 10      right: 12     line: 1      Int
i           left: 14      right: 14     line: 1      Identifier
int         left: 16      right: 18     line: 1      Int
j           left: 20      right: 20     line: 1      Identifier
READLINE:      i = i + j + 7
i           left: 3       right: 3      line: 2      Identifier
=           left: 5       right: 5      line: 2      Assign
i           left: 7       right: 7      line: 2      Identifier
+           left: 9       right: 9      line: 2      Plus
j           left: 11      right: 11     line: 2      Identifier
+           left: 13      right: 13     line: 2      Plus
7           left: 15      right: 15     line: 2      INTeger
READLINE:      j = write(i)
j           left: 3       right: 3      line: 3      Identifier
=           left: 5       right: 5      line: 3      Assign
write       left: 7       right: 11     line: 3      Identifier
(           left: 12      right: 12     line: 3      LeftParen
i           left: 13      right: 13     line: 3      Identifier
)           left: 14      right: 14     line: 3      RightParen
READLINE:   }
}           left: 0       right: 0      line: 4      RightBrace
  1: program { int i int j
  2:    i = i + j + 7
  3:    j = write(i)
  4: }
```

No error messages or warnings were displayed, and the application ran as expected

Assumptions

I assumed some tokens may be malformed and implemented some error handling for these invalid tokens.

Implementation

Lexer

The Lexer class contains main () method of our application. We first read the input file name and pass it on to the SourceReader for reading. The input file is read character by character and Token is generated when a valid Symbol or Identifier is found for initial analysis. The Lexer reads file name from the command line and outputs usage if no file name is found.

Token

The Token class is updated to include line numbers at which the token was found. This line number is provided by the Lexer while creating the Token object.

TokenSetup

The TokenSetup class is responsible for reading tokens from the tokens file and automatically generating TokenType and Tokens classes.

TokenType

TokenType class is a HashMap, mapping from the type Tokens to Symbol.

Tokens

The Tokens class is the enumeration of all the tokens.

SourceReader

SourceReader class is responsible for reading the input file. The file is scanned by a BufferedReader. The position of character and the line number is in the SourceReader.

Symbol

Symbol class is responsible for generating a symbol from the string of the new token and kind of type Tokens. This class also stores the symbol name.

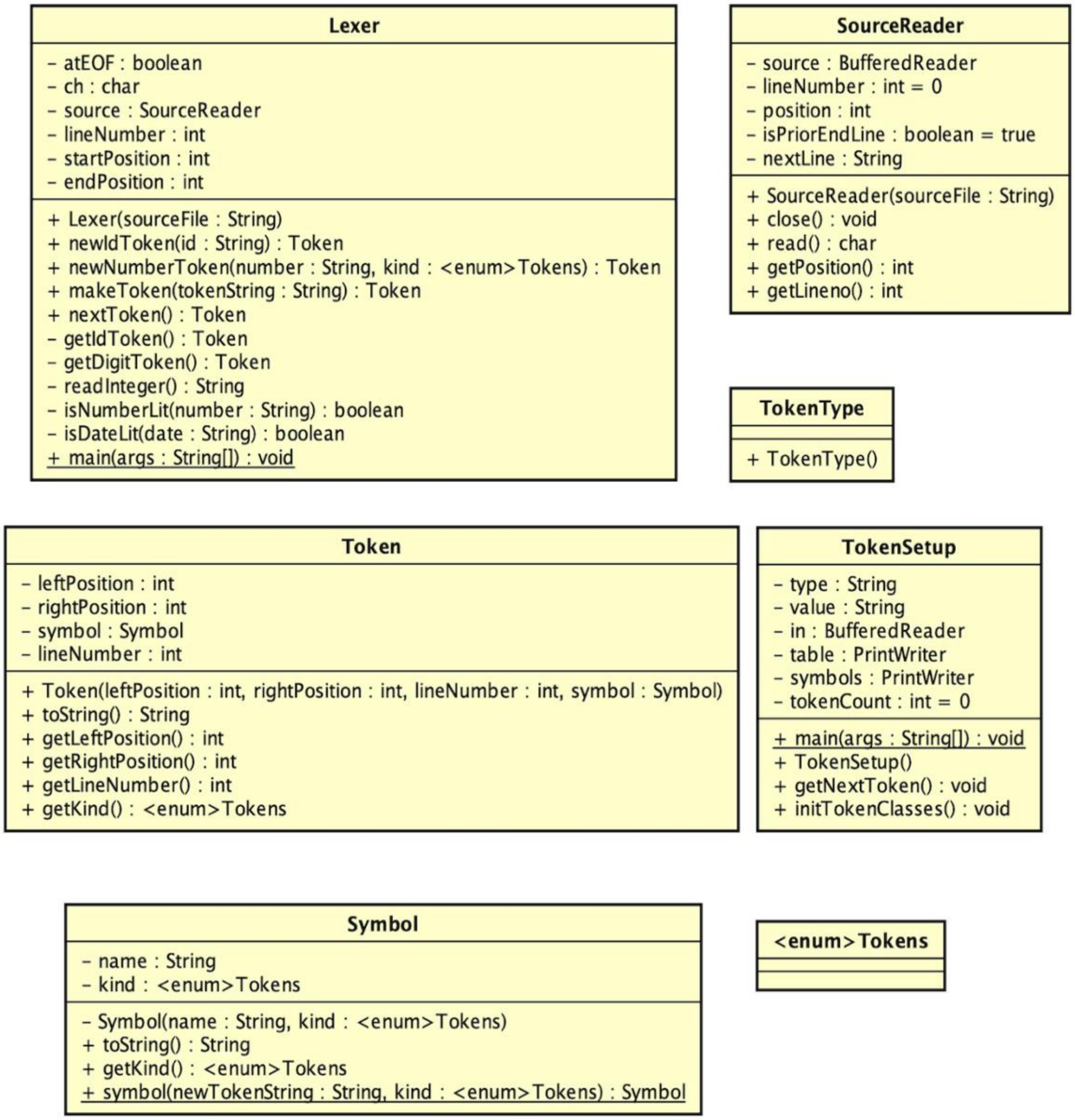
Code Organization

I put processing of all tokens starting with a digit into the method getDigitToken in Lexer class to localize the functionality and avoid repetition. I also noticed that the logic to check if the input is integer was duplicated in a lengthy try/catch block, I refactored this part into a method readInteger() to handle the case of the Integer(also part of NumberLit/Date token kinds). Also, I refactored the idToken try/catch block to a method getIdToken(), to check if the token is id token by checking every character is JavaIdentifierStart.

I also noticed some unnecessary parameters such like (int startPosition, int endPosition) when in defining the function such like newIdToken(), makeToken() as their parameters, so I removed those parameters because they are already available as fields in the Lexer class, accessible to the methods above. Though I understand the original code with these parameters don't have to get the object into a certain state, with certain values in the fields to test, and I can see how the method in original code might have been written in a functional style, but I prefer to have a single/source path for information in this case.

Class Diagram

The following class diagrams show the details of all the classes in this project, including the inheritance hierarchy


Lexer
- atEOF : boolean
- ch : char
- source : SourceReader
- lineNumber : int
- startPosition : int
- endPosition : int
+ Lexer(sourceFile : String)
+ newIdToken(id : String) : Token
+ newNumberToken(number : String, kind : <enum>Tokens) : Token
+ makeToken(tokenString : String) : Token
+ nextToken() : Token
- getIdToken() : Token
- getDigitToken() : Token
- readInteger() : String
- isNumberLit(number : String) : boolean
- isDateLit(date : String) : boolean
+ main(args : String[]) : void
SourceReader
- source : BufferedReader
- lineNumber : int = 0
- position : int
- isPriorEndLine : boolean = true
- nextLine : String
+ SourceReader(sourceFile : String)
+ close() : void
+ read() : char
+ getPosition() : int
+ getLineno() : int
TokenType
+ TokenType()
Token
- leftPosition : int
- rightPosition : int
- symbol : Symbol
- lineNumber : int
+ Token(leftPosition : int, rightPosition : int, lineNumber : int, symbol : Symbol)
+ toString() : String
+ getLeftPosition() : int
+ getRightPosition() : int
+ getLineNumber() : int
+ getKind() : <enum>Tokens
TokenSetup
- type : String
- value : String
- in : BufferedReader
- table : PrintWriter
- symbols : PrintWriter
- tokenCount : int = 0
+ main(args : String[]) : void
+ TokenSetup()
+ getNextToken() : void
+ initTokenClasses() : void
Symbol
- name : String
- kind : <enum>Tokens
- Symbol(name : String, kind : <enum>Tokens)
+ toString() : String
+ getKind() : <enum>Tokens
+ symbol(newTokenString : String, kind : <enum>Tokens) : Symbol
<enum>Tokens

Results and Conclusion

This project helped me to understand the concept of Lexer, which I had not encountered before. The Lexer’s job is the process of converting a sequence of characters such as the source code of a computer program into a sequence of tokens. I successfully implemented all the required features defined in the project specification. If there is invalid token identified, the Lexer will stop processing the next line and report the error with its position with line numbers.

Challenges

It was challenging for me to implement the different kinds of digit tokens. My first idea was to set the default value to Integer and use it to check integer part of the number case and date case. Later I introduced the helper methods: isNumberLit() to validate the number cases and isDateLit() to validate Date cases.

Future Work

I think in future this module will be used by the Parser and Compiler modules in the project. Some interesting future work could be to extend the lexer with other token types, e.g. ++.