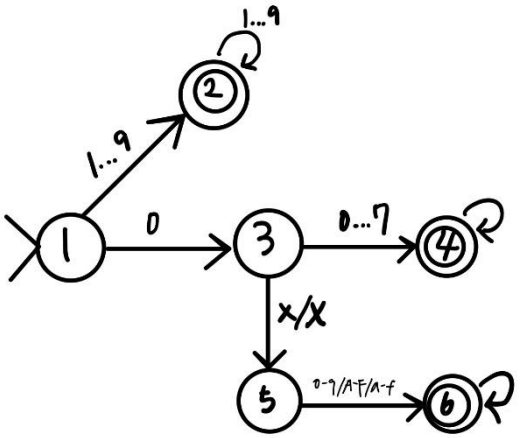
REPORT

미니 컴파일러

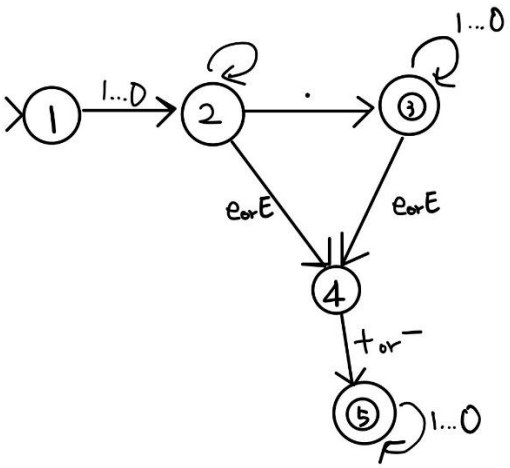
과 목 명 :	오토마타와컴파일러
담당교수 :	이상범
학 과 :	소프트웨어학과
학 번 :	32201669
이 름 :	박세연
제 출 일 :	2023.11.16

1. 각 토큰 별 DFA

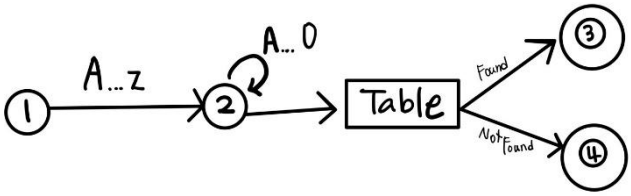
정수 DFA



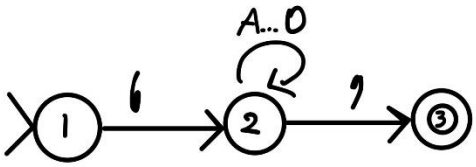
실수 DFA



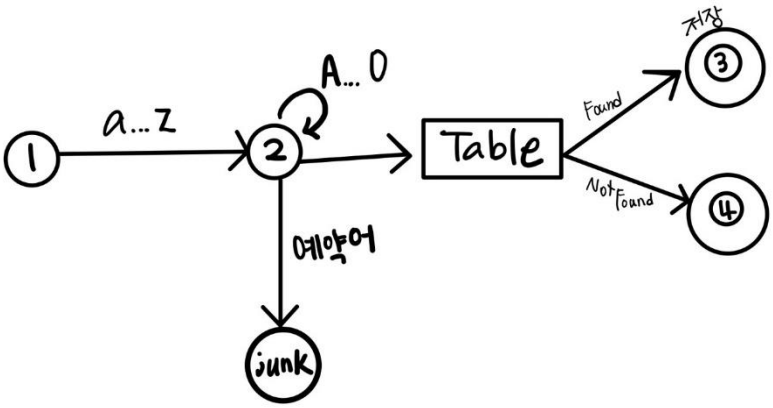
예약어 DFA



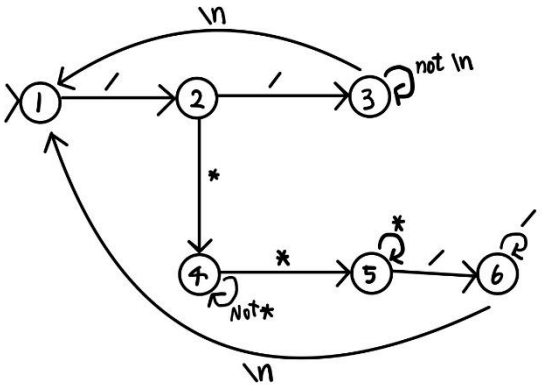
상수 DFA



사용자 정의어 DFA



주석 DFA



2. 개발 환경

- 개발 언어: 자바
- 개발 프레임워크: IntelliJ
- 클래스 및 함수 설명
  - Lexer: 토큰을 추출하는 클래스
    - nextToken(): 소스코드에서 하나의 토큰을 발견한 뒤 리턴. 다시 실행하면 다음 토큰을 읽는다.
    - tokenize(): 소스 코드에 있는 모든 토큰을 읽어올 때까지 반복하며 토큰을 출력한다.
  - FileScanner
    - scanFile(): 파일을 읽어 파일의 내용인 소스 코드를 반환한다.
  - Main
    - main(): 파일을 읽고 소스코드를 얻어 온 다음, 스캐너 객체를 만들어 토큰화 한다.

3. 소스 코드

Lexer.java

nextToken()은 두개인데, 주석 처리된 코드는 문자열 단위로 읽어서 토큰을 반환하고, 주석 처리가 되지 않은 코드는 문자 단위로 검사를 하여 토큰을 반환하는 코드이다.

```java
package minic;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Lexer {

    // 예약어 테이블
    private static final Map<String, Integer> RESERVED_WORDS_TOKENS =
Map.ofEntries(
            Map.entry("if", 40), Map.entry("while", 41),
            Map.entry("for", 42), Map.entry("const", 43),
            Map.entry("int", 44), Map.entry("float", 45),
            Map.entry("else", 46), Map.entry("return", 47),
            Map.entry("void", 48), Map.entry("break", 49),
            Map.entry("continue", 50), Map.entry("char", 51),
            Map.entry("then", 52)
    );

    // 연산자 테이블
    private static final Map<String, Integer> OPERATORS = Map.ofEntries(
            Map.entry("+", 10), Map.entry("-", 11),
            Map.entry("*", 12), Map.entry( "/", 13),
            Map.entry("%", 14), Map.entry("=", 15),
            Map.entry("!", 16), Map.entry("&&", 17),
            Map.entry("||", 18), Map.entry("==", 19),
            Map.entry("!=", 20), Map.entry("<", 21),
            Map.entry(">", 22), Map.entry("<=", 23),
            Map.entry(">=", 24)
    );

    // 기호 테이블
    private static final Map<String, Integer> SPECIAL_SYMBOLS = Map.of(
            "[", 30, "]", 31, "{", 32, "}", 33, "(", 34,
            ")", 35, ",", 36, ";", 37, "\"", 38, "'", 39
    );

    // 사용자 정의어 패턴 정의(앞글자 소문자, 이후 숫자 혹은 알파벳)
    private static final Pattern IDENTIFIER_PATTERN = Pattern.compile("[a-
zA-Z][a-zA-Z0-9]*");

//    // 상수 패턴 인식
//    private static final Pattern CONSTANT_PATTERN = Pattern.compile("[A-
Z0-9_\\\\s]+");

    // 소수 패턴 정의
    private static final Pattern DECIMAL_PATTERN = Pattern.compile("^[1-
9][0-9]*");

    // 8진수 패턴 정의
    private static final Pattern OCTAL_PATTERN = Pattern.compile("^0[0-
7]+");

    // 16진수 패턴 정의
```

```
    private static final Pattern HEXADECIMAL_PATTERN =
Pattern.compile("^0[xX][0-9a-fA-F]+");

    // 실수 패턴 정의
    private static final Pattern REAL_PATTERN = Pattern.compile("^[0-
9]*\\.?[0-9]+([eE][-+]?[0-9]+)?");

    private String input;
    private int position;
    private Map<String, String> symbolTable = new HashMap<>(); // 사용자
예약어 테이블
    private int symbolId = 0;

    /*
        소스코드를 받아서 보관
        주석을 제외
     */
    public Lexer(String input) {
        this.input = input
                // 여러 줄 주석 제외
                .replaceAll("(?s)/\\*.*?\\*/", "")
                // (코드 중간에 있기도 한) 한줄 주석 제외
                .replaceAll("(?m)([^\"\\n\\r]+|\"[^\"]*\")//.*", "$1");
        this.position = 0; // 소스 코드를 토큰 단위로 읽은 후 그 다음으로 포지션 유지
    }

//    /*
//        소스 코드 읽어 와서 실행 될 때마다 한글자 한글자 읽고 토큰을 만드는 것
//        매칭이 되면! 리턴
//        position 이라는 어떠한 위치를 lexer 객체가 가지고 있음
//        토큰이 만들어져 return 되면 그 다음 위치 부터 다음 토큰을 찾게 된다.
//        토큰을 확실하게 준다기 보다는, 소스 코드에서 하나의 토큰을 찾아서 반환 하는
과정이라고 볼 수 있다.
//     */
//    private String nextToken() {
//        if (position >= input.length()) { // 이미 토큰을 다 읽으면 null 반환
//            return null;
//        }
//
//        // 공백 건너뛰기
//        while (position < input.length() &&
Character.isWhitespace(input.charAt(position))) {
//            position++;
//        }
//
//        if (position >= input.length()) {
//            return null;
//        }
//
//        // 연산자와 기호 먼저 매치하기 (한 글자)
//        for (String op : OPERATORS.keySet()) {
```

```
//            if (input.startsWith(op, position)) {
//               position += op.length();
//               return op;
//            }
//         }
//
//         for (String sym : SPECIAL_SYMBOLS.keySet()) {
//            if (input.startsWith(sym, position)) {
//               position += sym.length();
//               return sym;
//            }
//         }
//
//         // literal, identifier, number, real number를 위한 코드
//         String remainingInput = input.substring(position);
//         Matcher m;
//
//         // 토큰이 예약어, 사용자 지정어 라면
//         m = IDENTIFIER_PATTERN.matcher(remainingInput);
//         if (m.lookingAt()) {
//            String idOrReserved = m.group();
//            position += idOrReserved.length();
//            if (RESERVED_WORDS_TOKENS.containsKey(idOrReserved)) { //
예약어가 맞는지 확인
//               return idOrReserved; // 예약어 반환
//            } else { // 예약어가 아니라면
//               return idOrReserved; // 사용자 지정어 반환
//            }
//         }
//
////         // 상수 인식
////         m = CONSTANT_PATTERN.matcher(remainingInput);
////         if (m.lookingAt()) {
////            String constant = m.group();
////            position += constant.length();
////            return constant;
////         }
//
//         // real number 라면
//         m = REAL_PATTERN.matcher(remainingInput);
//         if (m.lookingAt()) {
//            String real = m.group();
//            position += real.length();
//            return real;
//         }
//
//         // 소수
//         m = DECIMAL_PATTERN.matcher(remainingInput);
//         if (m.lookingAt()) {
//            String decimal = m.group();
//            position += decimal.length();
//            return decimal;
//         }
//
//         // 8진수
```

```
//        m = OCTAL_PATTERN.matcher(remainingInput);
//        if (m.lookingAt()) {
//            String octal = m.group();
//            position += octal.length();
//            return octal;
//        }
//
//        // 16진수
//        m = HEXADECIMAL_PATTERN.matcher(remainingInput);
//        if (m.lookingAt()) {
//            String hexadecimal = m.group();
//            position += hexadecimal.length();
//            return hexadecimal;
//        }
//
//        // 토큰이 발견되지 않으면, 에러
//        throw new IllegalArgumentException("잘못된 코드 발견: " +
input.charAt(position));
//
//    }

    //    /*
//        소스 코드 읽어 와서 실행 될 때마다 한글자 한글자 읽고 토큰을 만드는 것
//        매칭이 되면! 리턴
//        position 이라는 어떠한 위치를 lexer 객체가 가지고 있음
//        토큰이 만들어져 return 되면 그 다음 위치 부터 다음 토큰을 찾게 된다.
//        토큰을 확실하게 준다기 보다는, 소스 코드에서 하나의 토큰을 찾아서 반환 하는
과정이라고 볼 수 있다.
//     */
    private String nextToken() {
        // 입력이 끝났는지 확인
        if (position >= input.length()) {
            return null;
        }

        // 현재 문자 확인
        char currentChar = input.charAt(position++);

        // 공백은 건너뛴다.
        if (Character.isWhitespace(currentChar)) {
            return nextToken();
        }

        // 연산자, 특수 기호 확인
        String token = String.valueOf(currentChar);
        if (OPERATORS.containsKey(token) ||
SPECIAL_SYMBOLS.containsKey(token)) {
            // 두 글자 연산자 확인 (예: ==, &&)
            if (position < input.length()) {
                String twoCharToken = token + input.charAt(position);
                if (OPERATORS.containsKey(twoCharToken) ||
SPECIAL_SYMBOLS.containsKey(twoCharToken)) {
                    position++;
```

```
                return twoCharToken;
            }
        }
        return token;
    }

    // 숫자나 식별자 시작인지 확인
    if (Character.isDigit(currentChar) ||
Character.isLetter(currentChar)) {
        StringBuilder sb = new StringBuilder(token);
        while (position < input.length() &&
(Character.isDigit(input.charAt(position)) ||
Character.isLetter(input.charAt(position)))) {
            sb.append(input.charAt(position++));
        }
        return sb.toString();
    }

    // 알 수 없는 문자
    throw new IllegalArgumentException("잘못된 코드 발견: " + currentChar);
}


/*
    이 메소드에서는 nextToken()에서 반환한 토큰들이 어떤 토큰인지 확인한 후, 토큰을
출력한다.
    null 값을 반환할 때까지 반복해서 출력한다.
 */
public void tokenize() {
    String token;
    while ((token = nextToken()) != null) { // nextToken()에서 한 토큰이
리턴된다. 토큰이 더이상 없는 경우 null 이 반환된다.
        if (RESERVED_WORDS_TOKENS.containsKey(token.toLowerCase())) { //
먼저 예약어 인지 확인한다.
            System.out.println(token + " (예약어, Token: (" +
RESERVED_WORDS_TOKENS.get(token.toLowerCase()) + ", 0)");
        } else if (OPERATORS.containsKey(token)) { // 연산자 인지 확인한다.
            System.out.println(token + " (연산자, Token: (" +
OPERATORS.get(token) + ", 0)");
        } else if (SPECIAL_SYMBOLS.containsKey(token)) { // 기호 인지
확인한다.
            System.out.println(token + " (특수 기호, Token: (" +
SPECIAL_SYMBOLS.get(token) + ", 0)");
        } else if (token.matches("^[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?$"))
{ // 실수인지 확인한다.
            System.out.println(token + " (실수, Token: (6,"+token+")");
        } else if (token.matches("^[1-9][0-9]*")) { // 정수인지 확인한다.
            System.out.println(token + " (정수, Token: (5,"+token+")");
        } else if (token.matches("^0[0-7]+")) { // 8진수인지 확인한다.
```

```
                System.out.println(token + " (8진수, Token: (5,"+token+")");
            } else if (token.matches("^0[xX][0-9a-fA-F]+")) { // 16진수인지
확인한다.
                System.out.println(token + " (16진수, Token: (5," +token+
")");
//            } else if (token.matches("^[A-Z0-9_\\\\s]+")) { // 상수를
처리한다.
//                System.out.println(token + " (Constant, Token: 4)");
            } else if (IDENTIFIER_PATTERN.matcher(token).matches()) {
                // 최종적으로 사용자 정의어인지 확인한다.
                if (!RESERVED_WORDS_TOKENS.containsKey(token.toLowerCase()))
{
                    // 만약 테이블에 없을 경우, 넣는다.
                    if (!symbolTable.containsKey(token)) {
                        symbolTable.put(token, String.format("00,%02d",
symbolId++));
                    }// 있으면 가져 온다.
                    System.out.println(token + " (사용자 정의어, Token: (" +
symbolTable.get(token) + "))");
                }
            } else { // 만약 모든 곳에서도 해당되지 않으면 에러가 발생한다.
                System.err.println("알 수 없는 토큰 발생: " + token);
            }
        }
    }
}
}
```

FileScanner.java

```
package minic;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/*
    파일을 읽어서 파일의 내용인 소스 코드를 반환 한다.
 */
public class FileScanner {
    public static String scanFile(String filename) {
        StringBuilder fileContent = new StringBuilder();
        try (BufferedReader br = new BufferedReader(new
FileReader(filename))) {
            String line;
            while ((line = br.readLine()) != null) {
                fileContent.append(line).append("\n"); // 라인을 읽어서
fileContent에 추가
            }
```

```
        } catch (IOException e) { // 예외 처리
            System.err.println("파일 찾기 실패: " + e.getMessage());
            return null;
        }
        return fileContent.toString();
    }
}
```

Main.java

```
package minic;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // 콘솔로부터 입력을 받기 위한 Scanner 객체 생성
        System.out.println("파일 경로: ");
        String inputText = scanner.nextLine(); // 입력한 파일 경로 읽기


        // 파일 스캔 해서 소스 코드 얻어 오기
        String sourceCode = FileScanner.scanFile(inputText);

        //스캐너 객체 만들기
        Lexer lexer = new Lexer(sourceCode);

        // 소스 코드 토큰화 하기
        lexer.tokenize();
    }
}
```

4. 실행 결과

Sample program #1 실행 결과

```
"C:\Program Files\Java\jdk-21\bin\ja
파일 경로:
sample1.txt
int (예약어, Token: (44, 0)
a (사용자 정의어, Token: (00,00))
, (특수 기호, Token: (36, 0)
b (사용자 정의어, Token: (00,01))
, (특수 기호, Token: (36, 0)
sum (사용자 정의어, Token: (00,02))
; (특수 기호, Token: (37, 0)
float (예약어, Token: (45, 0)
x1 (사용자 정의어, Token: (00,03))
, (특수 기호, Token: (36, 0)
y1 (사용자 정의어, Token: (00,04))
, (특수 기호, Token: (36, 0)
zoom (사용자 정의어, Token: (00,05))
; (특수 기호, Token: (37, 0)
if (예약어, Token: (40, 0)
( (특수 기호, Token: (34, 0)
a (사용자 정의어, Token: (00,00))
> (연산자, Token: (22, 0)
b (사용자 정의어, Token: (00,01))
) (특수 기호, Token: (35, 0)
then (예약어, Token: (52, 0)
sum (사용자 정의어, Token: (00,02))
= (연산자, Token: (15, 0)
a (사용자 정의어, Token: (00,00))
+ (연산자, Token: (10, 0)
b (사용자 정의어, Token: (00,01))
else (예약어, Token: (46, 0)
sum (사용자 정의어, Token: (00,02))
= (연산자, Token: (15, 0)
a (사용자 정의어, Token: (00,00))
+ (연산자, Token: (10, 0)
10 (실수, Token: (6,10)
; (특수 기호, Token: (37, 0)
```

```
while (예약어, Token: (41, 0)
( (특수 기호, Token: (34, 0)
a (사용자 정의어, Token: (00,00))
== (연산자, Token: (19, 0)
b (사용자 정의어, Token: (00,01))
) (특수 기호, Token: (35, 0)
{ (특수 기호, Token: (32, 0)
zoom (사용자 정의어, Token: (00,05))
= (연산자, Token: (15, 0)
( (특수 기호, Token: (34, 0)
sum (사용자 정의어, Token: (00,02))
+ (연산자, Token: (10, 0)
x1 (사용자 정의어, Token: (00,03))
) (특수 기호, Token: (35, 0)
/ (연산자, Token: (13, 0)
10 (실수, Token: (6,10)
; (특수 기호, Token: (37, 0)
asd (사용자 정의어, Token: (00,06))
ch1 (사용자 정의어, Token: (00,07))
= (연산자, Token: (15, 0)
' (특수 기호, Token: (39, 0)
123 (실수, Token: (6,123)
' (특수 기호, Token: (39, 0)
; (특수 기호, Token: (37, 0)
} (특수 기호, Token: (33, 0)

Process finished with exit code 0
```

Sample program #2 실행 결과

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDE
파일 경로:
sample2.txt
int (예약어, Token: (44, 0)
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : 잘못된 코드 발견: &
    at minic.Lexer.nextToken(Lexer.java:168)
    at minic.Lexer.tokenize(Lexer.java:178)
    at minic.Main.main(Main.java:17)

Process finished with exit code 1
```

5. 개발 후기

DFA에 그린 대로 하고 싶었으나 실제로는 전체 부분을 코드로 작성하다 보니 자세하게 고려 되지 않아 아쉽다.

오픈 소스를 참고하였지만 전체적인 코드는 새로 작성했다.

상수를 구현하지 못한 것이 아쉽다.

심볼 테이블과 명령어 테이블을 어떻게 구현해야 좋을지 헷갈려

그리고 String.matches()를 사용했는데, 이 함수를 사용해도 되는 것인지 모르겠다. 이렇게 짜면 코드를 단순화 시킬 수 있는 것 같아 사용했다.