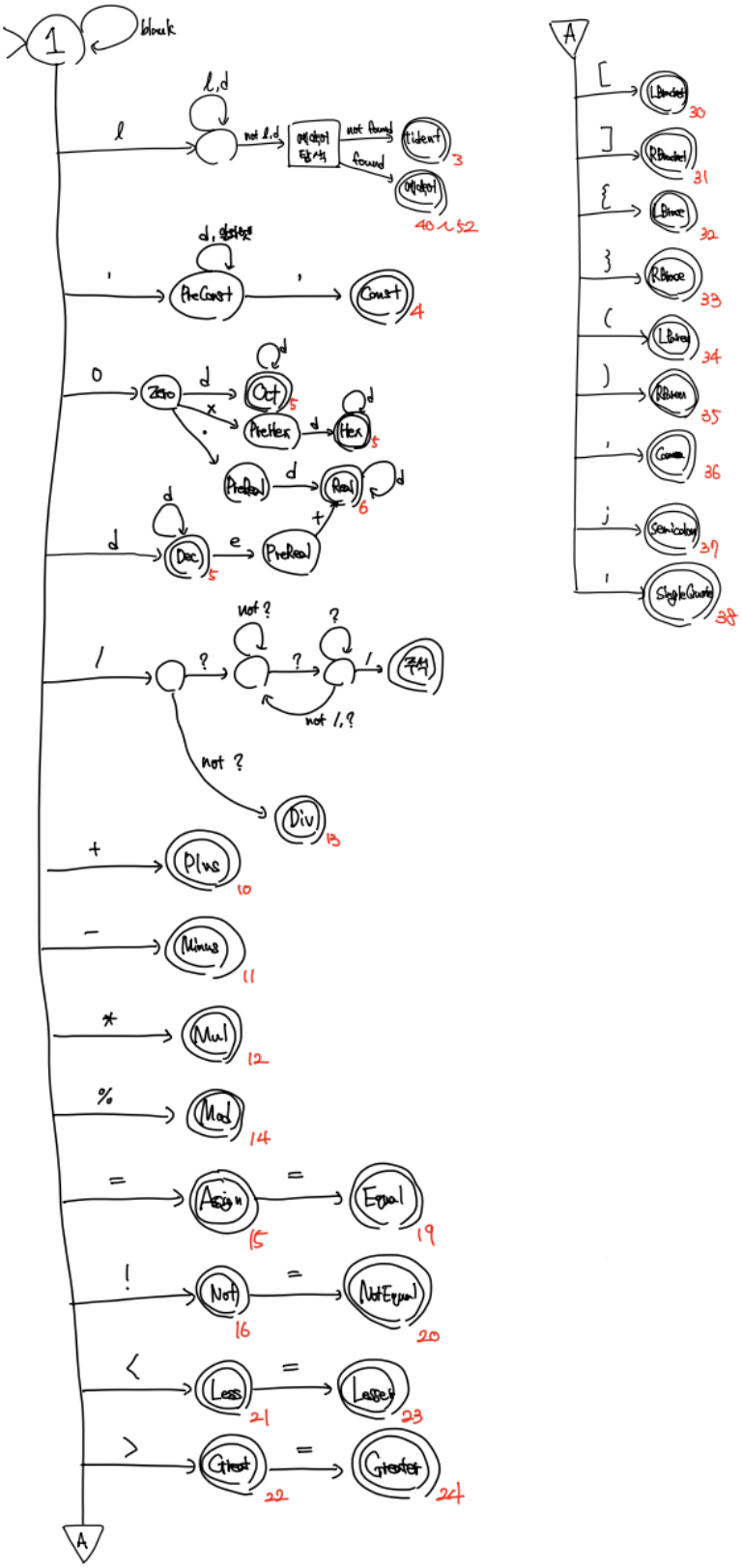
단국대학교 DANKOOK UNIVERSITY

■ 목차

- **DFA**


1
blank
l,d
l
not l,d
Identifier
found
not found
3
40 ~ 52
'
PreConst
Const
4
0
Zero
d
x
.
Oct
PreHex
Hex
PreReal
Real
6
e
PreReal
Dec
5
/
not ?
?
Div
13
+
Plus
10
-
Minus
11
*
Mul
12
%
Mod
14
=
Assign
Equal
15
19
!
Not
NotEqual
16
20
<
Less
LessEq
21
23
>
Great
Greater
22
24
A
[
LBracket
30
]
RBracket
31
{
LBrace
32
}
RBrace
33
(
LParen
34
)
RParen
35
,
Comma
36
;
SemiColon
37
SingleQuote
38

● 프로그램 소스 코드(오픈소스와 차이가 있는 부분 파란 사각형)

➢ CScanner.java

```
package CScanner;

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

public class CScanner {
    static public final int ID_LENGTH = 16; // 명칭의 길이 제한
    static public final char EOF = '\255'; // EOF는 파일의 끝을 의미합니다
    static public final String SPECIAL_CHARS = "!=&|<>"; // 두글자로 조합가능한 특수문자들

    private String src; // 소스코드를 String으로 저장해줄 변수
    private Integer cursor; // 소스코드를 읽을 때 커서가 될 변수

    // 어떤 토큰인지 인식하기 위해 만든 상태들
    private enum State {
        Initial, Dec, Oct, Hex, Real, Const, tidOrReserved, Op, Zero, PreHex, PreReal, PreConst, SingleOperator
    }

    // 생성자
    public CScanner(String filePath) {
        src = parseFile(filePath);
        cursor = 0;
    }

    // 소스코드를 String으로 읽어주는 메소드
    private String parseFile(String filePath) {
        String src = "", readedString = ""; // src: 소스코드를 저장해놓을 String 변수, readedString: 소스코드의 한줄을 담아놓을 String 변수
        FileReader fileReader = null; // 소스코드를 읽을 File Reader
        try {
            fileReader = new FileReader(new File(filePath)); // 파일 경로를 이용해 File Reader 생성
        } catch (IOException e) {
            // 파일을 읽을 수 없음
            System.err.print(Error.getErrorMessage(Error.ErrorCode.CannotOpenFile));
            return "";
        }

        BufferedReader reader = new BufferedReader(fileReader); // BufferedReader 객체를 만들어 소스코드 파일을 읽음
        try {
            while ((readedString = reader.readLine()) != null) // 파일로부터 한줄 읽음
                src += readedString + "\n"; // 한줄 맨뒤에 개행문자 추가
            src += EOF;   // 파일의 끝을 의미하는 EOF 문자 추가
            reader.close();
        } catch(IOException e) {
            // 파일을 읽을 수 없음
            System.err.print(Error.getErrorMessage(Error.ErrorCode.CannotOpenFile));
            return "";
        }
        return src;
    }

    // EOF인지 검사하는 메소드
    private boolean isEOF(int idx) {
        return idx >= src.length();
    }
```

```
// 주석 처리 메소드
private boolean exceptComment() {
    char c;
    // 커서로부터 whitespace 문자를 모두 무시
    while(!isEOF(cursor) && Character.isWhitespace(src.charAt(cursor))){
        cursor++;
    }
    if(isEOF(cursor)) {
        return false; // 공백을 무시하고 EOF면 주석 제거 성공하면 false 반환
    }

    if (src.charAt(cursor) == '/') { // '/'가 나올 경우
        if(src.charAt(cursor+1) == '?') { // /? 는 블록 주석
            cursor += 2; // "/?" 다음 문자로 커서 이동
            while(src.charAt(cursor) != '?' && src.charAt(cursor+1) != '/') { // ?/가 나올때 까지 반복
                if(isEOF(cursor+1)) return true; // 찾지 못하면 실패, true 반환
                cursor++;
            }
            cursor += 2; // "?/" 다음 문자로 커서 이동
        }
    }
    return false; // 성공적으로 제거하면 false 반환
}
```

```java
// 문자가 1글자 연산자인지 확인하는 메소드
private boolean isSingleSpecialToken(char c) {
    switch (c) {
        case '(': case ')': case ',': case ';': case '[': case ']': case '{': case '}':
        case '+': case '-': case '*': case '/': case '%': case EOF:
            return true;
        default:
            return false;
    }
}
```

```java
// 문자가 1글자 연산자가 아닌 특수문자인지 확인하는 메소드
private boolean isSpecialChar(char c) {
    for (int i = 0; i < SPECIAL_CHARS.length(); ++i)
        if (SPECIAL_CHARS.charAt(i) == c)
            return true;
    return false;
}
```

```java
private Token.SymType getSymbolType(State s) {
    switch (s) {
        case Dec: // 10, 8, 16, 0은 Digit 반환
        case Oct:
        case Hex:
        case Zero:
            return Token.SymType.Digit;
        case Real: // 실수인 경우 Real 반환
            return Token.SymType.Real;
        case Const: // 상수인 경우 Const 반환
            return Token.SymType.Const;
        case tidOrReserved: // 명칭이나 예약어인 경우 tidOrReserved반환
            return Token.SymType.tidOrReserved;
        case Op: // 연산자인 경우 OP반환
        case SingleOperator:
            return Token.SymType.Op;
        case Initial: // 종결상태가 아닌경우에는 실패이므로 NULL을 반환
        case PreHex:
        default:
            return Token.SymType.NULL;
    }
}
```

```java
public Token getToken() {
    Token token = new Token();
    Token.SymType symType = Token.SymType.NULL; // Symbol Type을 NULL로 설정
    String tokenString = "";

    State state = State.Initial;

    // 현재 커서로부터 Comment 제거
    if (exceptComment()) {
        // Comment를 지우는 도중에 ERROR가 발생했을 경우
        System.err.print(Error.getErrorMessage(Error.ErrorCode.InvalidComment));
        return token;
    }
```

```java
while(!isEOF(cursor)) {
    char c = src.charAt(cursor++);

    if(Character.isWhitespace(c)) {
        if(state != State.Initial) break;
        else continue;
    }else if(state == State.Initial && c == '0') {
        state = State.Zero;
    }else if(Character.isDigit(c)) { //숫자를 만났을때 이전의 상태에 따라 현재상태 정의
        if(state == State.Initial) {
            state = State.Dec;
        }else if(state == State.Zero) {
            state = State.Oct;
        }else if(state == State.PreHex) {
            state = State.Hex;
        }else if(state == State.PreReal) {
            state = State.Real;
        }else if(state == State.PreConst) {
            state = State.Const;
        }else if(state == State.Op) { // 연산자가 나온 다음 숫자면 break
            --cursor;
            break;
        }
```

```
            }else if(state == State.Dec && c == '.') { // '숫자.' 을 인식한 경우
                state = State.PreReal;
            }else if(state == State.Zero && c == '.') { // '0.' 을 인식한 경우
                state = State.PreReal;
            }else if(state == State.Zero && c == 'x') { // '0x' 를 인식한 경우
                state = State.PreHex;
            }else if(state == State.Dec && c == 'e') { // '숫자e' 를 인식한 경우
                state = State.PreReal;
            }else if(state == State.PreReal && c == '+') { // 'e+' 를 인식한 경우
                state = State.Real;
            }else if(state == State.Initial && c == '\'') { // '''를 인식한 경우
                state = State.PreConst;
            }else if(state == State.Const && c == '\'') {
                state = State.Const;
            }else if(state == State.PreConst) { // 상수
                state = State.Const;
            }else if(isSingleSpecialToken(c)) { // '(', ')', '{', '}', ',', '[', ']', ';', EOF를 인식한 경우
                if(state == State.Initial) {
                    state = State.SingleOperator;
                    tokenString = String.valueOf(c);
                }else --cursor;
                break;
            }else if(isSpecialChar(c)) { // 문자 두개가 연산자가 될 수 있는 경우
                if(state != State.Initial && state != State.Op) {
                    --cursor;
                    break;
                }
                state = State.Op;
            }else if(Character.isAlphabetic(c)) { //알파벳을 인식한 경우
                if(state != State.Initial && state != State.tidOrReserved) { // ID나 예약어가 아니면 break
                    --cursor;
                    break;
                }
                state = State.tidOrReserved;
            }
            tokenString += String.valueOf(c); // 글자 String으로 추가
        }
        symType = getSymbolType(state);
        if(symType == Token.SymType.tidOrReserved && tokenString.length() > ID_LENGTH) {
            // ID의 길이가 16을 넘으면 에러처리
            System.err.print(Error.getErrorMessage(Error.ErrorCode.AboveIDLimit));
            return token;
        }
        token.setSymbol(tokenString, symType);
        return token;
    }
}
```

➢ Token.java

```
package CScanner;

public class Token {
    public enum SymType { // 쉽게 분류하기 위한 대분류들
        Op, tidOrReserved, Digit, Real, Const, NULL
    }

    public enum Symbols { // 토큰의 심볼들
        NULL,
        NULL0, NULL1, NULL2, tident, tconst, tint, treal, NULL7, NULL8, NULL9,
        Plus, Minus, Mul, Div, Mod, Assign, Not, And, Or, Equal,
        NotEqual, Less, Great, Lesser, Greater, NULL25, NULL26, NULL27, NULL28, NULL29,
        LBracket, RBracket, LBrace, RBrace, LParen, RParen, Comma, Semicolon, SingleQuote, NULL39,
        If, While, For, Const, Int, Float, Else, Return, Void, Break,
        Continue, Char, Then, EOF
    }

    private Symbols symbol; //토큰이 가진 심볼 변수
    private String value; // 값을 저장하는 변수
    private String tokenString; // 토큰의 String

    // 생성자
    public Token() {
        symbol = Symbols.NULL;
        value = "0";
        tokenString = "NULL";
    }

    // 토큰이 가진 심볼과 value를 올바르게 설정하는 메소드
    public void setSymbol(String token, SymType type) {
        tokenString = token;
        switch(type) {
        case tidOrReserved:
            symbol = GETtidentOrReservedWord(token);
            if(symbol == Symbols.tident) {
                value = token;
            }
            break;
        case Digit:
            symbol = Symbols.tint;
            value = Integer.toString(parseInt(token));
            break;
        case Real:
            symbol = Symbols.treal;
            value = token;
            break;
        case Const:
            symbol = Symbols.tconst;
            value = token;
            break;
        case Op:
            symbol = getOp(token);
            break;
        case NULL:
        default:
            break;
    }
}
```

```java
// 정수의 진법 계산 메소드
private int parseInt(String s) {
    int radix = 10; // default 진법은 10진수
    if (s.startsWith("0x")){ // 16진수일 경우
        radix = 16; // 진법을 16진수로 설정
        s = s.substring(2); // prefix인 0x 제거
    } else if (s.startsWith("0") && s.length() > 1){ // 8진수일 경우
        radix = 8; // 진법을 8진수로 설정
    }
    return Integer.parseInt(s, radix); // 위에서 설정한 진법대로 진법 변환
}
```

```java
// indent나 예약어를 받았을 때 구분해주는 메소드
private Symbols GETtidentOrReservedWord(String token) {
    switch(token) {
    case "If":      return Symbols.If;
    case "While":   return Symbols.While;
    case "For":   return Symbols.For;
    case "Const":   return Symbols.Const;
    case "Int":   return Symbols.Int;
    case "Float":   return Symbols.Float;
    case "Else":   return Symbols.Else;
    case "Return":   return Symbols.Return;
    case "Void":   return Symbols.Void;
    case "Break":   return Symbols.Break;
    case "Continue":   return Symbols.Continue;
    case "Char":   return Symbols.Char;
    case "Then":   return Symbols.Then;

    default:
        return Symbols.tident;
    }
}

// 토큰이 연산자인 경우 구분해주는 메소드
private Symbols getOp(String token) {
    switch (token) {
        case "!":   return Symbols.Not;
        case "!=":  return Symbols.NotEqual;
        case "%":   return Symbols.Mod;
        case "&&":  return Symbols.And;
        case "(":   return Symbols.LParen;
        case ")":   return Symbols.RParen;
        case "*":   return Symbols.Mul;
        case "+":   return Symbols.Plus;
        case ",":   return Symbols.Comma;
        case "-":   return Symbols.Minus;
        case "/":   return Symbols.Div;
        case ";":   return Symbols.Semicolon;
        case "<":   return Symbols.Less;
        case "<=":  return Symbols.Lesser;
        case "=":   return Symbols.Assign;
        case "==":  return Symbols.Equal;
        case ">":   return Symbols.Great;
        case ">=":  return Symbols.Greater;
        case "[":   return Symbols.LBracket;
        case "]":   return Symbols.RBracket;
        case "\255": return Symbols.EOF;
        case "{":   return Symbols.LBrace;
        case "||":  return Symbols.Or;
        case "}":   return Symbols.RBrace;
        case "'":   return Symbols.SingleQuote;
        case "&": // &하나만 있으면 에러
            System.err.print(Error.getErrorMessage(Error.ErrorCode.SingleAmpersand));
            break;
        case "|": // |하나만 있으면 에러
            System.err.print(Error.getErrorMessage(Error.ErrorCode.SingleBar));
            break;
        default: // 그 외 인식을 못하면 에러
            System.err.print(Error.getErrorMessage(Error.ErrorCode.InvalidChar));
            break;
    }
    return Symbols.NULL;
}
```

```
    // 토큰 심볼을 숫자로 리턴해주는 메소드
    public int getSymbolOrdinal() {
        return symbol.ordinal()-1;  // NULL이 -1이기 때문에 -1 해야한다.
    }

    // 토큰이 명칭이나 숫자인 경우에 토큰의 값을 얻는 메소드
    public String getSymbolValue() {
        return value;
    }

    // 출력 편의를 위한 toString 메소드
    public String toString() {
        return tokenString + "\t : (" + getSymbolOrdinal() + ", "+ getSymbolValue() + ")";
    }
}
```

➢ Error.java

```java
package CScanner;

public class Error {
    public enum ErrorCode {
        CannotOpenFile, AboveIDLimit, SingleAmpersand, SingleBar, InvalidChar, InvalidComment
    }

    public static String getErrorMessage(ErrorCode code) {
        String msg;
        msg = "Error occur(code: " + code.ordinal() + ")\n";
        switch (code){
            case CannotOpenFile:
                msg += "cannot open the file. please check the file path.";
                break;
            case AboveIDLimit:
                msg += "an identifier length must be less than 12.";
                break;
            case SingleAmpersand:
                msg += "next character must be &.";
                break;
            case SingleBar:
                msg += "next character must be |.";
                break;
            case InvalidChar:
                msg += "invalid character!!!";
                break;
            case InvalidComment:
                msg += "invalid block comment!!!";
                break;
            default:
                msg += "Unknown Error";
                break;
        }
        return msg;
    }
}
```

➢ Main.java

```java
package CScanner;

public class Main {
    public static void main(String[] args) {
        CScanner sc = new CScanner("D:\\SampleCode01.txt"); // 스캐너 객체
        Token tok = null; // 토큰을 저장하기 위한 변수
        while ((tok = sc.getToken()).getSymbolOrdinal() != -1) // 스캐너가 끝날때까지 토큰을 출력
            System.out.println(tok);
    }
}
```

● 입력 소스 코드

SampleCode01.txt

```
Int a, b, sum;
Float x1, y1, zoom;
If (a>b) Then sum = a+b
Else sum = a+10;
while (a ==b) {
   zoom = (sum + x1)/10;
   ch1 = '123';
}
```

SampleCode02.txt

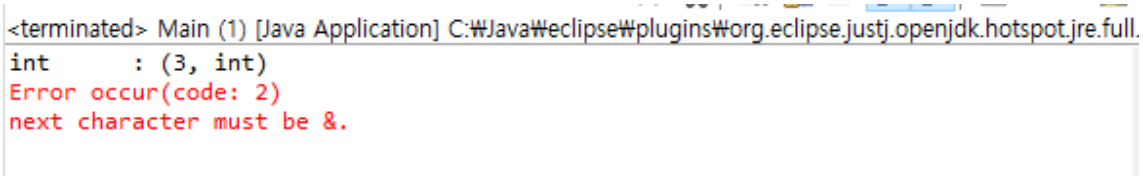
```
int &a1, 2b:
Float x2, y2;
a1 := +100;
X2 = 12.23e+10;
sum = xa123;
```

- **실행 결과**

SampleCode01.txt

```
<terminated> Main (1) [Java Application] C:₩Java₩eclipse₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full
Int       : (44, 0)
a         : (3, a)
,         : (36, 0)
b         : (3, b)
,         : (36, 0)
sum       : (3, sum)
;         : (37, 0)
Float     : (45, 0)
x1        : (3, x1)
,         : (36, 0)
y1        : (3, y1)
,         : (36, 0)
zoom      : (3, zoom)
;         : (37, 0)
If        : (40, 0)
(         : (34, 0)
a         : (3, a)
>         : (22, 0)
b         : (3, b)
)         : (35, 0)
Then      : (52, 0)
sum       : (3, sum)
=         : (15, 0)
a         : (3, a)
+         : (10, 0)
b         : (3, b)
Else      : (46, 0)
sum       : (3, sum)
=         : (15, 0)
a         : (3, a)
+         : (10, 0)
10        : (5, 10)
;         : (37, 0)
while     : (3, while)
(         : (34, 0)
a         : (3, a)
==        : (19, 0)
b         : (3, b)
)         : (35, 0)
{         : (32, 0)
zoom      : (3, zoom)
=         : (15, 0)
(         : (34, 0)
sum       : (3, sum)
+         : (10, 0)
x1        : (3, x1)
)         : (35, 0)
/         : (13, 0)
10        : (5, 10)
;         : (37, 0)
ch1       : (3, ch1)
=         : (15, 0)
'123'     : (4, '123')
;         : (37, 0)
}         : (33, 0)
-         : (53, 0)
```

SampleCode02.txt



● 개발 후기

교수님이 알려주신 오픈소스 코드를 보면서 배움과 성장할 시간을 가질 수 있었습니다. 스캐너에서 식별자나 상수, 예약어, 특수 기호등을 분류해서 토큰의 심볼을 준다는 것이 직접 해보기전까지는 이론적인 느낌으로 다가왔는데, 코드를 짜보며 실습하다보니 제가 사용한 JAVA말고 다른 언어로는 어떻게 구현될것인가.. 아니면 실제로 사용되는 스캐너는 얼마나 더 정교하게 돌아가고 있을지 궁금증이 많이 생겼습니다. 오픈소스를 만드신 분께서 코드를 객체지향적으로 짜셨는데, 특히 대분류와 소분류로 상태를 나누고 상황에 맞춰 분류가 가능하게 한 점이 굉장히 인상깊었습니다. 그리고 Token 객체를 만들어서 각각의 토큰이 완성되는 것을 toString 변수에 추가하여 출력하는 과정을 단순화시키고 후에 컴파일러까지 구현하게 되는 상황을 상정해서 getToken() 메소드만 호출하면 토큰 추출이 가능하게 한 점도 아주 효율적이라고 생각합니다. 새로운 스캐너를 만들어볼 생각도 했었는데, 코드를 분석하면 할수록 아주 잘 만든 코드라 생각해, 기존의 코드를 완벽하게 분석하고 이번 과제 지침에 따르게 만드는 방향으로 코드를 수정, 개선시키게 되었습니다.

읽어주셔서 감사합니다.